

**RANCANG BANGUN METAPROGRAM SEBAGAI CODE
GENERATOR PADA PTIIK-APPS DENGAN MENGGUNAKAN
BAHASA PEMROGRAMAN PHP**

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Agie Ghazy Falah
NIM: 115060800111091



INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2016

PERSETUJUAN

RANCANG BANGUN METAPROGRAM SEBAGAI CODE GENERATOR PADA PTIIK-
APPS DENGAN MENGGUNAKAN BAHASA PEMROGRAMAN PHP

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh:

Nama Agie Ghazy Falah

NIM: 115060800111091

Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I

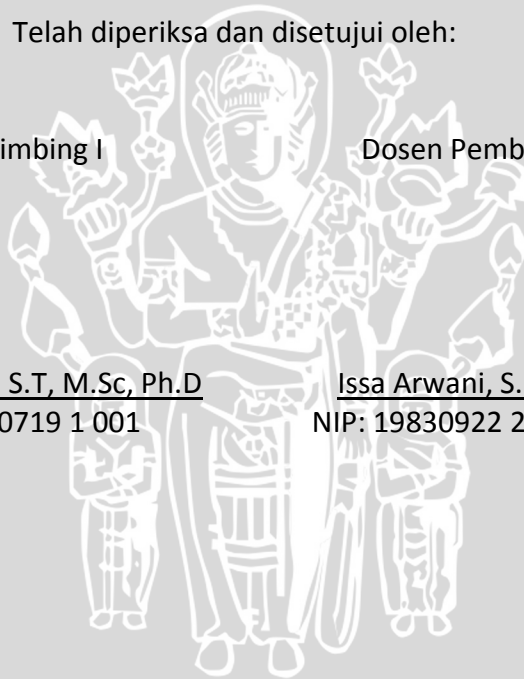
Dosen Pembimbing II

Ismiarta Aknuranda, S.T, M.Sc, Ph.D

NIK: 201006 740719 1 001

Issa Arwani, S.Kom, M.Sc

NIP: 19830922 201212 1 003



PERNYATAAN ORISINALITAS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah skripsi ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis disitasi dalam naskah ini dan disebutkan dalam daftar pustaka.

Apabila ternyata didalam naskah skripsi ini dapat dibuktikan terdapat unsur-unsur plagiasi, saya bersedia skripsi ini digugurkan dan gelar akademik yang telah saya peroleh (sarjana) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 1 Januari 2016



Agie Ghazy Falah

NIM: 115060800111091

KATA PENGANTAR

Puji syukur penulis panjatkan kehadist Allah SWT karena berkat Rahmat dan Karunia-Nya penulis dapat menyelesaikan penyusunan skripsi ini. Shalawat beserta salam semoga senantiasa terlimpah kepada Nabi Muhammad SAW, kepada keluarganya, para sahabatnya, hingga kepada umatnya hingga akhir zaman, aamiin.

Penulisan skripsi ini diajukan untuk memenuhi syarat untuk memperoleh gelar Sarjana pada Fakultas Ilmu Komputer jurusan Informatika Universitas Brawijaya. Judul yang penulis ajukan adalah 'Rancang Bangun Metaprogram sebagai *Code Generator* pada PTIIK-Apps dengan Menggunakan Bahasa Pemrograman PHP'.

Dalam penyusunan dan penulisan skripsi ini tidak terlepas dari bantuan, bimbingan serta dukungan dari berbagai pihak. Oleh karena itu dalam kesempatan ini penulis menyampaikan terima kasih kepada yang terhormat.

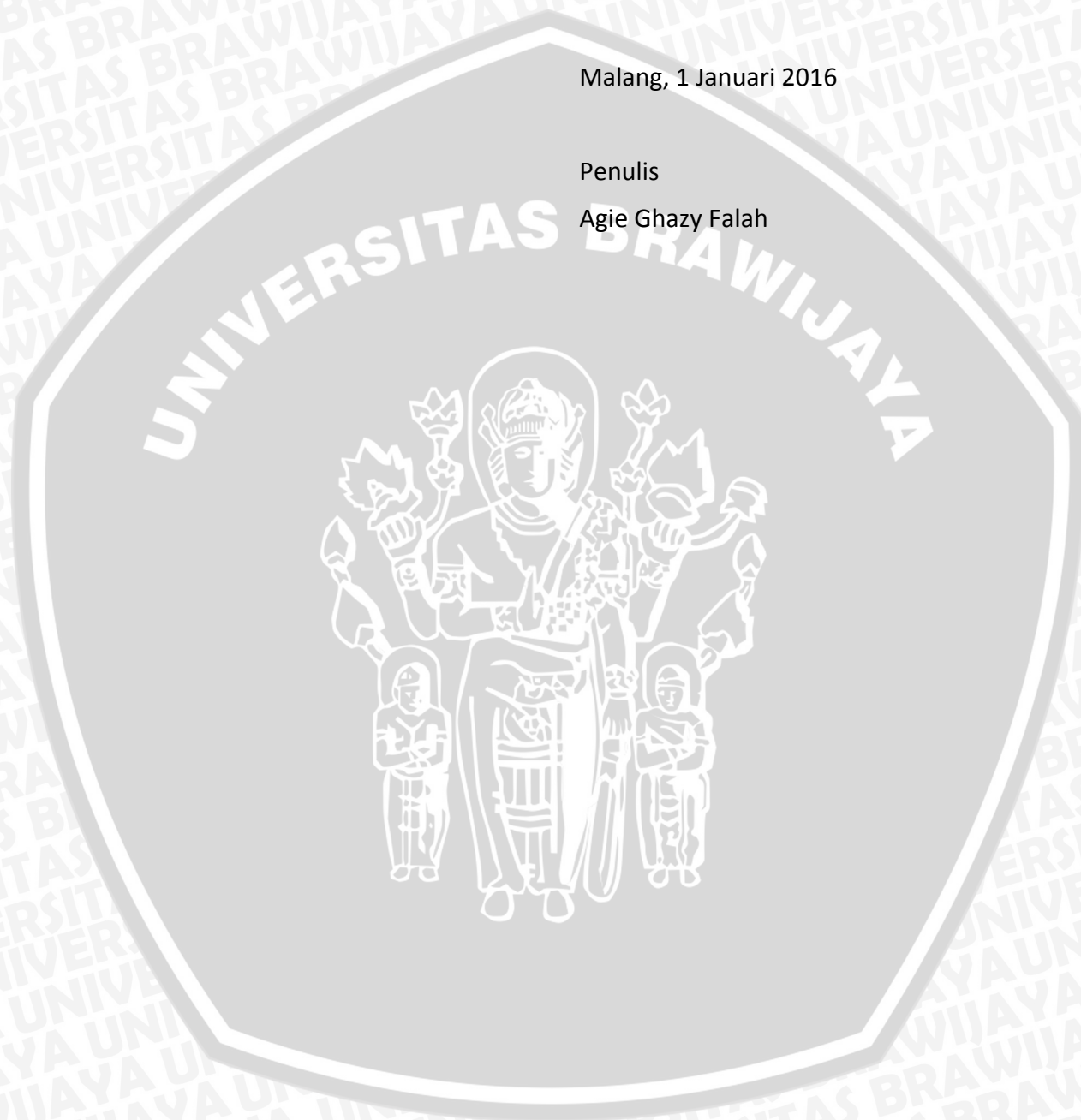
1. Bapak Ismiarta Aknuranda, S.T, M.Sc, Ph.D selaku dosen pembimbing I yang selalu bijaksana memberikan bimbingan, nasehat serta waktunya selama penelitian dan penulisan skripsi ini.
2. Bapak Issa Arwani, S.Kom, M.Sc selaku dosen pembimbing II yang telah mencurahkan perhatian dan bimbingan yang sangat berarti bagi penulis.
3. Staff dosen Fakultas Ilmu Komputer Universitas Brawijaya yang telah membekali penulis berbagai ilmu selama mengikuti perkuliahan sampai akhir penulisan skripsi.
4. Staff karyawan Fakultas Ilmu Komputer Universitas Brawijaya yang telah membantu penulis selama mengikuti perkuliahan dan penulisan skripsi ini.
5. Bapak Kasyful Amron, S.T, M.Sc yang telah membantu penulis menyediakan tempat untuk melakukan penelitian di lingkungan unit kerja Badan Pengembangan Teknologi Informasi dan Komunikasi (BPTIK) Fakultas Ilmu Komputer Universitas Brawijaya.
6. Mbak Thusti Dwi Purnami, A.Md yang telah membimbing penulis untuk melakukan penelitian di lingkungan unit kerja BPTIK.
7. Ayah dan Ibu atas jasa – jasanya, kesabaran, do'a yang tidak pernah lelah dalam mendidik dan memberikan cinta yang tulus ikhlas kepada penulis sejak kecil.
8. Teman – teman semua atas kebersamaan dan bantuan yang berarti bagi penulis.
9. Semua pihak yang telah banyak membantu penulis dalam menyelesaikan skripsi ini.

Semoga Allah SWT memberikan balasan yang berlipat ganda kepada semuanya. Demi perbaikan selanjutnya, saran dan kritik yang membangun akan penulis terima dengan senang hati. Akhirnya, hanya kepada Allah SWT penulis serahkan segalanya mudah – mudahan dapat bermanfaat khususnya bagi penulis umumnya bagi kita semua.

Malang, 1 Januari 2016

Penulis

Agie Ghazy Falah



ABSTRAK

PTIIK-Apps adalah sebuah aplikasi berbasis web yang digunakan untuk mengelola sistem informasi Fakultas Ilmu Komputer Universitas Brawijaya. Proses pengembangan PTIIK-Apps dilakukan oleh staff pengembang pada unit kerja BPTIK (Badan Pengembangan Teknologi Informasi dan Ilmu Komputer). Kendala pada proses pengembangan adalah implementasi kode program yang berulang – ulang karena komponen yang dikembangkan memiliki beberapa karakteristik kesamaan. Maka dari itu sebuah *code generator/program generator* akan dikembangkan untuk mempercepat proses implementasi kode program karena dapat menghasilkan kode program secara otomatis.

Proses pembuatan *code generator* membutuhkan sebuah desain untuk memodelkan komponen kode program yang akan dihasilkan secara otomatis sebagai masukan (*application description*). Basis pengetahuan sistem (*domain knowledge*) juga dibutuhkan oleh *code generator* agar sistem dapat mengubah *application description* menjadi kode program PTIIK-Apps.

Rekayasa domain (*domain engineering*) dilakukan untuk menganalisis domain PTIIK-Apps dan menghasilkan beberapa model yang dibutuhkan untuk membuat desain model *application description* dan *domain knowledge*. Rekayasa domain terdiri dari tiga tahap, yaitu 1) analisis domain 2) pemodelan domain dan 3) implementasi domain. Tahap analisis domain dilakukan dengan metode *Feature Oriented Domain Analysis* (FODA). Metode tersebut digunakan untuk menganalisis domain PTIIK-Apps dengan membandingkan kesamaan dan perbedaan fitur – fitur yang ada pada domain.

Tahap implementasi domain dilakukan dengan bahasa pemrograman PHP dan menggunakan teknik *templating*. Teknik tersebut digunakan untuk menerjemahkan masukan model *application description* kedalam basis pengetahuan domain (*domain knowledge*). Komponen kode program yang telah dihasilkan secara otomatis dapat diunduh oleh pengguna sistem melalui *browser*.

Penelitian ini menghasilkan sebuah *code generator* yang bisa menghasilkan komponen kode program PTIIK-Apps secara otomatis. Pengujian aplikasi *code generator* dilakukan dengan dua aspek, yaitu 1) untuk menguji bahwa *code generator* dapat berjalan sesuai dengan harapan dan 2) untuk menguji bahwa komponen kode program yang dihasilkan dapat berjalan dengan baik pada domain PTIIK-Apps.

Kata kunci: rekayasa domain, FODA, penggunaan ulang perangkat lunak, *code generator*, PHP

ABSTRACT

PTIIK-Apps is a information system that used to managed Faculty of Computer Science University of Brawijaya. PTIIK-Apps development process is done by developer unit in that faculty which called BPTIK. The obstacle of development process is repeated code implementation of component which have some similarity aspects. Therefore a code generator/program generator will be developed to make code implementation process faster because that code generator have an ability to generate code program automatically.

Development process of code generator require a design for modeling code component that will be generated as input. Domain knowledge also required by code generator so system can render the input of application description to code components of PTIIK-Apps.

Domain engineering is used to analyze the domain of PTIIK-Apps and will derive some required model to design application description and domain knowledge. Domain engineering consist three major step, that is 1) domain analysis, domain modeling and 3) domain implementation. Domain analysis is done by Feature Oriented Domain Analysis (FODA) method. That method is used for analyze the PTIIK-Apps domain by compare the similarity and variability of features.

The code generator system on this research is implemented by PHP language and used templating technique. Templating technique used for render input of application description model to domain knowledge of system. Code components which generated automatically by system can be downloaded by user form his browser.

This research produce a code generator that can generated code components of PTIIK-Apps automatically. That system is tested by two major aspect, that is 1) to test that code generator run its functionality correctly and 2) to test the generated code components is valid and run correctly in its domain.

Keywords: domain engineering, FODA, software reuse, *code generator*, PHP

DAFTAR ISI

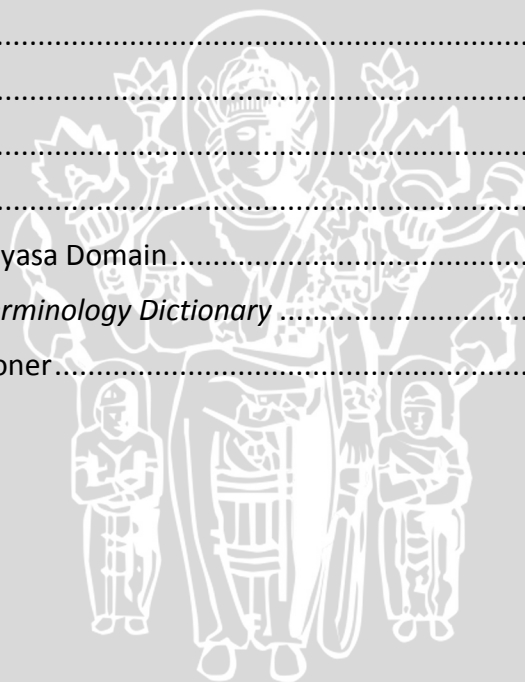
RANCANG BANGUN METAPROGRAM SEBAGAI <i>CODE GENERATOR</i> PADA PTIIK-APPS DENGAN MENGGUNAKAN BAHASA PEMROGRAMAN PHP	i
PERSETUJUAN	ii
PERNYATAAN ORISINALITAS	iii
KATA PENGANTAR	iv
ABSTRAK	vi
ABSTRACT	vii
DAFTAR ISI	viii
DAFTAR TABEL	xiii
DAFTAR GAMBAR	xvi
DAFTAR LAMPIRAN	xx
BAB 1 PENDAHULUAN	1
1.1 Latar belakang	1
1.2 Rumusan masalah	2
1.3 Tujuan	3
1.4 Manfaat	3
1.5 Batasan masalah	3
1.6 Sistematika pembahasan	3
BAB 2 LANDASAN KEPUSTAKAAN	5
2.1 Kajian pustaka	5
2.2 Bahasa pemrograman PHP	5
2.3 PTIIK-Apps	6
2.3.1 <i>Reusable assets</i> pada PTIIK-Apps	6
2.4 Definisi <i>metaprogramming</i>	7
2.4.1 <i>Code generator</i>	8
2.4.2 Metaprogramming pada PHP	10
2.4.3 Fungsi <code>__autoload()</code>	10
2.4.4 Fungsi <code>__get()</code> dan <code>__set()</code>	11
2.4.5 Fungsi <i>callback</i>	12
2.4.6 Fungsi <code>ob_start()</code>	12

2.4.7 Fungsi extract()	13
2.5 Template programming dengan PHP	14
2.6 Database MySql	15
2.6.1 Tabel <i>information_schema.schemata</i>	16
2.6.2 Tabel <i>information_schema.columns</i>	16
2.7 Software Reuse Menurut Ian Sommerville	17
2.7.1 Dasar Penggunaan Ulang Perangkat Lunak	18
2.7.2 Generator-Based Reuse	20
2.8 Domain engineering	21
2.8.1 Domain Analysis	22
2.8.2 Domain Design	22
2.8.3 Domain Implementation	23
2.9 Feature oriented domain analysis (FODA)	23
2.9.1 Context Analysis	24
2.9.2 Domain Modeling	25
2.9.2.1 Feature Analysis	26
2.9.2.2 Entity-Relationship Modeling	27
2.9.2.3 Functional Analysis	29
2.9.3 Architecture Model	31
2.10 Pengujian perangkat lunak	33
2.10.1 Pemilihan Unit Test Case	34
2.11 Development Testing	35
2.11.1 Unit Testing	35
2.11.2 Component Testing	36
2.11.3 System Testing	37
2.12 Release Testing	39
2.12.1 Requirement-based Testing	39
2.12.2 Scenario Testing	40
2.12.3 Performance Testing	41
2.13 User Testing	41
BAB 3 METODOLOGI	44
3.1 Studi literatur	44

3.2 Pengumpulan data	44
3.2.1 Observasi	44
3.2.2 Kuisisioner	45
3.2.3 Riset pustaka	45
3.3 Analisis persyaratan	45
3.4 Perancangan sistem	45
3.4.1 <i>Domain engineering</i>	45
3.4.2 Perancangan sistem	46
3.4.2.1 Perancangan application description	46
3.4.2.2 Perancangan domain knowledge	46
3.4.2.3 Perancangan code generator	46
3.5 Implementasi	47
3.6 Pengujian dan analisis	47
3.7 Pengambilan kesimpulan dan saran	47
BAB 4 ANALISIS PERSYARATAN	48
4.1 Gambaran umum aplikasi	48
4.2 Analisis persyaratan	49
4.2.1 Identifikasi aktor	49
4.2.2 Identifikasi persyaratan fungsional	49
4.2.2.1 Use case diagram	50
4.2.2.2 Use case specification	50
4.2.3 Identifikasi persyaratan non-fungsional	58
BAB 5 PERANCANGAN	59
5.1 <i>Domain Engineering</i>	59
5.2 Analisis Domain	59
5.2.1 Analisis konteks	59
5.2.2 Pemodelan domain (<i>domain modelling</i>)	63
5.2.2.1 Analisis Fitur	63
5.2.2.2 Pemodelan entity-relationship	95
5.2.2.3 Analisis Fungsional	97
5.2.3 Pemodelan Arsitektur	120
5.3 Desain Domain	123

5.3.1 Perancangan halaman tatap muka	124
5.3.2 Perancangan <i>application description</i>	127
5.3.3 Perancangan <i>application domain knowledge</i>	128
5.3.3.1 Perancangan aktivitas application domain knowledge	129
5.3.3.2 Perancangan urutan domain knowledge	141
5.3.4 Perancangan <i>code generator</i>	152
5.3.4.1 Perancangan aktivitas code generator	152
5.3.4.2 Perancangan urutan code generator	164
5.3.4.3 Perancangan Class Diagram	178
BAB 6 IMPLEMENTASI	180
6.1 Hasil implementasi <i>domain knowledge</i>	180
6.1.1 Implementasi <i>controller knowledge</i>	180
6.1.2 Implementasi <i>model knowledge</i>	181
6.1.3 Implementasi Kode Program <i>view add knowledge</i>	182
6.1.4 Implementasi Kode Program <i>view list knowledge</i>	184
6.1.5 Implementasi Kode Program <i>view filter knowledge</i>	187
6.1.6 Implementasi Kode Program <i>view edit knowledge</i>	190
6.1.7 Implementasi Kode Program <i>view detail knowledge</i>	193
6.2 Hasil implementasi <i>program generator</i>	195
6.2.1 Implementasi kode program <i>login dan logout</i>	195
6.2.2 Implementasi kode program kelola sambungan server database	195
6.2.3 Implementasi kode program pengisian model masukan	198
6.2.4 Implementasi kode program proses data model komponen menjadi kode program	202
6.3 Hasil Implementasi Halaman Tatap Muka	204
6.3.1 Implementasi halaman <i>login</i>	204
6.3.2 Kelola sambungan koneksi	204
6.3.3 Implementasi halaman pengisian model masukan	206
6.3.4 Implementasi halaman unduh kode program	210
BAB 7 PENGUJIAN DAN ANALISIS	211
7.1 <i>Unit Testing</i>	211
7.1.1 <i>Unit testing controller knowledge</i>	212

7.1.2 Unit testing model knowledge	228
7.1.3 Unit testing view add knowledge.....	237
7.2 Release Testing	243
7.2.1 Pengujian fitur verifikasi akun	244
7.2.2 Pengujian fitur sambungan <i>database</i>	245
7.2.3 Pengujian fitur kelola komponen.....	246
7.2.4 Pengujian fitur tambah komponen.....	247
7.2.5 Pengujian fitur parametisasi komponen	248
7.2.6 Pengujian fitur hapus komponen.....	252
7.2.7 Pengujian fitur pembangkitan kode program	253
7.3 Analisis Pengujian	254
BAB 8 Penutup.....	256
8.1 Kesimpulan	256
8.2 Saran.....	257
DAFTAR PUSTAKA	259
Lampiran A Form Rekayasa Domain.....	260
Lampiran B <i>Domain Terminology Dictionary</i>	269
Lampiran C Hasil Kuisisioner.....	271



DAFTAR TABEL

Tabel 2.1 Kode program <i>autoload</i>	10
Tabel 2.2 Kode program fungsi <i>__set()</i> dan <i>__get()</i>	11
Tabel 2.3 Kode program untuk membaca data <i>inaccessible</i>	11
Tabel 2.4 Kode program fungsi <i>callback</i>	12
Tabel 2.5 Kode program <i>ob_start()</i>	12
Tabel 2.6 Tabel kode program fungsi <i>extract</i>	13
Tabel 2.7 Kode program <i>sample.php</i>	14
Tabel 2.8 Kode program <i>template.tpl</i>	15
Tabel 2.9 Perintah untuk Melihat Metadata Skema Basis Data	16
Tabel 2.10 Perintah untuk Melihat Metadata Kolom Basis Data.....	16
Tabel 2.11 Teknik yang Mendukung Penggunaan Ulang Perangkat Lunak	18
Tabel 4.1 Identifikasi Aktor	49
Tabel 4.2 <i>Glossary of Use Case Specifications</i>	57
Tabel 4.3 Spesifikasi Persyaratan Non-fungsional.....	58
Tabel 5.1 <i>Glossary</i> Modul KKNP	75
Tabel 5.2 Matriks Fungsionalitas Modul KKNP.....	76
Tabel 5.3 <i>Glossary</i> Modul Kerjasama	91
Tabel 5.4 Matriks Fungsionalitas Modul Kerjasama.....	92
Tabel 5.5 Gabungan Matriks Fungsionalitas Modul KKNP dan Modul Kerjasama	94
Tabel 6.1 Implementasi Kode Program <i>controller knowledge</i>	180
Tabel 6.2 Implementasi Kode Program <i>model knowledge</i>	181
Tabel 6.3 Implementasi Kode Program <i>view add knowledge</i>	182
Tabel 6.4 Implementasi Kode Program <i>view list knowledge</i>	185
Table 4.5 Implementasi Kode Program <i>view filter knowledge</i>	187
Tabel 6.6 Implementasi Kode Program <i>view edit knowledge</i>	190
Tabel 6.7 Implementasi Kode Program <i>view detail knowledge</i>	193
Tabel 6.8 Implementasi Kode Program <i>Login</i> dan <i>Logout</i>	195
Tabel 6.9 Implementasi Kode Program Sambungan ke Server Database	195

Table 6.10 Implementasi Kode Program Pemutusan Sambungan dari Server Database.....	196
Tabel 6.11 Implementasi Kode Program Muat Ulang Sambungan Server Database	197
Tabel 6.12 Implementasi Kode Program Pengambilan Model Tabel dari Server Database.....	197
Tabel 6.13 Implementasi Kode Program Kelola Model Komponen	198
Tabel 6.14 Implementasi Kode Program Kelola Model Fitur	198
Tabel 6.15 Implementasi Kode Program <i>Table Manager</i>	199
Tabel 6.16 Implementasi Kode Program <i>Column Manager</i>	200
Tabel 6.17 Implementasi Kode Program <i>Input-Option Manager</i>	201
Tabel 6.18 Implementasi Kode Program Proses Data Model Menjadi Kode Program	202
Tabel 7.1 Kode Program <i>Controller Knowledge</i>	212
Tabel 7.2 Tabel <i>Test Case Controller Knowledge</i>	218
Tabel 7.3 Tabel <i>Test Case</i> Fungsi Tambah Data pada <i>Controller Knowledge</i>	221
Tabel 7.4 Tabel <i>Test Case</i> Fungsi Ubah Data pada <i>Controller Knowledge</i>	226
Tabel 7.5 Kode Program Model Knowledge.....	229
Tabel 7.6 Table <i>Test Case Model Knowledge</i>	234
Tabel 7.7 Kode Program <i>View Add Knowledge</i>	237
Tabel 7.8 Table <i>Test Case View Add Knowledge</i>	241
Tabel 7.8 Pengujian <i>Black Box</i> Fitur Verifikasi Akun.....	244
Tabel 7.9 Pengujian <i>Black Box</i> Sambungan <i>Database</i>	245
Tabel 7.10 Pengujian <i>Black Box</i> Kelola Komponen	246
Tabel 7.11 Pengujian <i>Black Box</i> Tambah Komponen	247
Tabel 7.12 Pengujian <i>Black Box</i> Parameterisasi <i>Module Name</i>	248
Tabel 7.13 Pengujian <i>Black Box</i> Parameterisasi <i>Component Model</i>	249
Tabel 7.14 Pengujian <i>Black Box</i> Parameterisasi <i>Feature Model</i>	249
Tabel 7.15 Pengujian <i>Black Box</i> Parameterisasi <i>Table Model</i>	251
Tabel 7.16 Pengujian <i>Black Box</i> Parameterisasi <i>Column Model</i>	251
Tabel 7.17 Pengujian <i>Black Box</i> Parameterisasi <i>Input-Option Model</i>	252
Tabel 7.18 Pengujian <i>Black Box</i> Hapus Komponen	253

Tabel 7.19 Pengujian *Black Box* Pembangkitan Kode Program dan Unduh
Komponen 253



DAFTAR GAMBAR

Gambar 2.1 Contoh Template-based Code Generation	9
Gambar 2.2 Popularitas bahasa pemrograman berdasarkan jumlah buku yang tersedia di Amazon dan website	10
Gambar 2.3 Arsitektur <i>Generator-based Reuse</i>	21
Gambar 2.4 Contoh <i>Structure Diagram</i>	25
Gambar 2.5 Contoh <i>Context Diagram: Data-flow</i> Utama pada <i>Domain Analysis</i>	25
Gambar 2.6 <i>Feature Diagram</i> Mobil	27
Gambar 2.7 Entity-relationship model pada FODA	29
Gambar 2.8 Parameterisasi pada <i>Functional Analysis</i>	31
Gambar 2.9 Lapisan Arsitektur	33
Gambar 2.10 Model Proses Pengujian Perangkat Lunak	34
Gambar 2.11 Component Interface	36
Gambar 2.12 Sequence Diagram	38
Gambar 2.13 Proses Acceptance Testing	42
Gambar 3.1 Bagan Metodologi Penelitian	44
Gambar 4.1 Perancangan Umum Sistem	48
Gambar 4.2 <i>Use Case Diagram</i> Sistem <i>Program Generator PTIIK-Apps</i>	50
Gambar 5.1 <i>Structure Diagram</i> PTIIK-Apps	60
Gambar 5.2 <i>Data Flow Diagram</i> Modul KKNP	62
Gambar 5.3 <i>Data Flow Diagram</i> Modul Kerjasama	62
Gambar 5.4 Entity Relationship Diagram (ERD) Modul KKNP	64
Gambar 5.5 Diagram <i>Use Case</i> Modul KKNP	64
Gambar 5.6 Model Fitur Modul KKNP	77
Gambar 5.7 <i>Entity Relationship Diagram</i> Modul Kerjasama	77
Gambar 5.8 Diagram <i>Use Case</i> Modul Kerjasama	78
Gambar 5.9 Modul Fitur Modul Kerjasama	93
Gambar 5.10 Model Fitur Umum Modul PTIIK-Apps	95
Gambar 5.11 <i>Entity Relational Model</i> Modul PTIIK-Apps	96
Gambar 5.12 Diagram Aktivitas Kelola Data KKNP	98
Gambar 5.13 Diagram Aktivitas Kelola Komponen Nilai KKNP	99

Gambar 5.14 Model Fungsional Fitur Lihat Data V.1	100
Gambar 5.15 Diagram Aktivitas Kelola Instansi Kerjasama	101
Gambar 5.16 Model Fungsional Fitur Lihat Data V.2	102
Gambar 5.17 Diagram Aktivitas Tambah Komponen Nilai KKNP	104
Gambar 5.18 Diagram Aktivitas Tambah Instansi Kerjasama	105
Gambar 5.19 Model Fungsional Fitur Tambah Data V.1	107
Gambar 5.20 Diagram Aktivitas Fitur Tambah Data KKNP (bagian 1)	108
Gambar 5.21 Diagram Aktivitas Fitur Tambah Data KKNP (bagian 2)	110
Gambar 5.22 Model Fungsional Fitur Tambah Data V.2 (bagian 1)	111
Gambar 5.23 Model Fungsional Fitur Tambah Data V.2 (bagian 2)	112
Gambar 5.24 Model Fungsional Fitur Perbarui Data	113
Gambar 5.25 Model Fungsional Fitur Detail Data	114
Gambar 5.26 Model Fungsional Fitur Hapus Data	115
Gambar 5.27.1 Diagram Keadaan Modul KKNP (bagian 1)	115
Gambar 5.27.2 Diagram Keadaan Modul KKNP (bagian 2)	116
Gambar 5.27.3 Diagram Keadaan Modul Kerjasama (bagian 1)	117
Gambar 5.27.4 Diagram Keadaan Modul Kerjasama (bagian 2)	118
Gambar 5.27.5 Diagram Keadaan Modul Kerjasama (bagian 3)	118
Gambar 5.27.6 Diagram Keadaan Modul Kerjasama (bagian 4)	119
Gambar 5.28 Model Arsitektur pada Lapisan <i>Domain Architecture Layer</i>	120
Gambar 5.29 Model Arsitektur pada Lapisan <i>Domain Utilities Layer</i>	122
Gambar 5.30 Desain <i>Program Generator</i>	124
Gambar 5.30.1 Rancangan Halaman <i>Login</i>	125
Gambar 5.30.2 Rancangan Halaman Parameterisasi Komponen	125
Gambar 5.30.3 Rancangan Form Sambungan ke <i>Server Basis Data</i>	126
Gambar 5.30.4 Rancangan Halaman Hasil Pembangkitan Komponen MVC	127
Gambar 5.31 Gambar <i>Application Description Model</i>	128
Gambar 5.32 <i>Activity Diagram Controller Knowledge</i>	130
Gambar 5.33 <i>Activity Diagram</i> Proses Simpan Data	132
Gambar 5.34 <i>Activity Diagram</i> Proses Ubah Data	134
Gambar 5.35 <i>Activity Diagram Model Knowledge</i>	136
Gambar 5.36 <i>Activity Diagram View List Knowledge</i>	137

Gambar 5.37 Activity Diagram View Detail Knowledge	137
Gambar 5.38 Activity Diagram View Filter Knowledge.....	138
Gambar 5.39 Activity Diagram View Save Knowledge	139
Gambar 5.40 Activity Diagram View Edit Knowledge.....	140
Gambar 5.41 Sequence Diagram Controller Knowledge (bagian 1)	142
Gambar 5.42 Sequence Diagram Controller Knowledge (bagian 2)	143
Gambar 5.43 Sequence Diagram Model Knowledge (bagian 1).....	145
Gambar 5.44 Sequence Diagram Model Knowledge (bagian 2).....	146
Gambar 5.45 Sequence Diagram Model Knowledge (bagian 3).....	147
Gambar 5.46 Sequence Diagram View List Knowledge	148
Gambar 5.47 Sequence Diagram View Detail Knowledge	149
Gambar 5.48 Sequence Diagram View Filter Knowledge	150
Gambar 5.49 Sequence Diagram View Save Knowledge	151
Gambar 5.50 Sequence Diagram View Edit Knowledge	152
Gambar 5.51 Activity Diagram Login	153
Gambar 5.52 Activity Diagram Logout	153
Gambar 5.53 Activity Diagram Kelola Model Komponen.....	154
Gambar 5.54 Activity Diagram Menyambungkan ke Server Database	155
Gambar 5.55 Activity Diagram Pemutusan Sambungan Server Database	156
Gambar 5.56 Activity Diagram Sambungan Ulang Server Database.....	156
Gambar 5.57 Activity Diagram Mengambil Model Tabel dari Database.....	157
Gambar 5.58 Activity Diagram Hapus Komponen	158
Gambar 5.59 Activity Diagram Proses Tambah Komponen	159
Gambar 5.60 Activity Diagram Proses Tambah Fitur	160
Gambar 5.61 Activity Diagram Table Manager	161
Gambar 5.62 Activity Diagram Column Manager	162
Gambar 5.63 Activity Diagram Input-Option Manager	163
Gambar 5.64 Activity Diagram Proses Data Model Komponen.....	164
Gambar 5.65 Sequence Diagram Proses Login	165
Gambar 5.66 Sequence Diagram Proses Logout	166
Gambar 5.67 Sequence Diagram Proses Kelola Model Komponen	167
Gambar 5.68 Sequence Diagram Penyambungan Server Database	168

Gambar 5.69 <i>Sequence Diagram</i> Proses Pemutusan Server Database	169
Gambar 5.70 <i>Sequence Diagram</i> Pengambilan Model Tabel dari Database	170
Gambar 5.71 <i>Sequence Diagram</i> Kelola Model Komponen	171
Gambar 5.72 <i>Sequence Diagram</i> Proses Kelola Fitur Model	172
Gambar 5.73 <i>Sequence Diagram Table Manager</i>	173
Gambar 5.74 <i>Sequence Diagram Column Manager</i>	174
Gambar 5.75 <i>Sequence Diagram Input-Option Manager</i>	175
Gambar 5.76 <i>Sequence Diagram</i> Proses Pengolahan Data Model Komponen ..	176
Gambar 5.77 <i>Sequence Diagram</i> Proses Pembangkitan Kode Program	177
Gambar 5.78 <i>Class Diagram code Generator</i>	179
Gambar 6.1 Implementasi Tatap Muka Halaman <i>Login</i>	204
Gambar 6.2 Implementasi Tatap Muka Sambungan ke Server Database	205
Gambar 6.3 Implementasi Tatap Muka Kelola Sambungan Server Database dan Kelola Model Tabel	206
Gambar 6.4 Implementasi Tatap Muka Halaman Parameterisasi Komponen ...	206
Gambar 6.5 Implementasi Tatap Muka untuk Mengelola Model Komponen dan Model Fitur	207
Gambar 6.6 Implementasi Tatap Muka <i>Table Manager</i>	208
Gambar 6.7 Implementasi Tatap Muka <i>Column Manager</i>	209
Gambar 6.8 Implementasi Tatap Muka <i>Input-Option Manager</i>	209
Gambar 6.9 Implementasi Tatap Muka Halaman Unduh Kode Program	210
Gambar 7.1 Diagram Alir <i>Controller Knowledge</i>	217
Gambar 7.2 Diagram Alir <i>Controller Knowledge</i> (Proses Tambah Data)	220
Gambar 7.3 Diagram Alir <i>Controller Knowledge</i> (Proses Ubah Data)	224
Gambar 7.4 Diagram Alir <i>Model Knowledge</i>	232
Gambar 7.5 Diagram Alir <i>View Add Knowledge</i>	239

DAFTAR LAMPIRAN

DAFTAR PUSTAKA	259
Lampiran A Form Rekayasa Domain	260
A.1 Form Definisi Model Fitur	260
A.2 Form Deskripsi <i>Entity</i>	262
A.3 Form Tipe Relasi	264
A.4 Form Deskripsi Isu	267
Lampiran B <i>Domain Terminology Dictionary</i>	269
Lampiran C Hasil Kuisisioner	271
C.1 Kuisisioner Proses Pengembangan Aplikasi PTIIK-Apps	271
C.2 Kuisisioner Penelitian Pembangunan <i>Code Generator</i>	271



BAB 1 PENDAHULUAN

Bab ini menjelaskan hal – hal yang akan dikerjakan dan mengapa hal tersebut dikerjakan. Bab ini berisi 1) latar belakang, 2) rumusan masalah, 3) tujuan, 4) manfaat, 5) batasan masalah dan 6) sistematika pembahasan.

1.1 Latar belakang

PTIIK-Apps adalah sebuah sistem informasi yang berguna untuk mengelola proses bisnis yang ada di Fakultas Ilmu Komputer (FILKOM) Universitas Brawijaya (UB) Malang. Sistem tersebut dikembangkan menggunakan bahasa pemrograman PHP dan Javascript. PTIIK-Apps dikembangkan oleh Badan Pengembangan Teknologi Informasi dan Ilmu Komputer (BPTIK).

Pada proses pengembangan PTIIK-Apps ditemukan sebuah kendala. Implementasi kode program memakan waktu yang lama, karena terdapat penulisan kode program dengan struktur yang sama secara berulang – ulang. Struktur kode program tersebut ditemukan untuk proses *create, read, update* dan *delete* (CRUD) pada tabel di basis data (*database*).

Dampak dari masalah tersebut menyebabkan para pengembang aplikasi (*developer*) menjadi lelah dan bosan, sehingga kesalahan manusia (*human error*) bisa terjadi pada proses implementasi kode program. Dalam penelitian oleh Safavi and Shaikh (2011, p.271) dijelaskan bahwa proses implementasi *code unit & testing* dapat memakan 40% usaha dari keseluruhan proses pembuatan perangkat lunak (*software*).

Untuk saat ini, permasalahan tersebut dapat diselesaikan dengan *reusable assets*. *Reusable assets* yang dimaksud adalah suatu komponen MVC yang memiliki fitur yang hampir sama dengan fitur yang akan dikembangkan. Jadi para pengembang tidak perlu membuat komponen MVC baru dari awal, cukup memodifikasi *reusable assets* sesuai dengan persyaratan. Namun cara tersebut belum cukup efektif dalam menyelesaikan masalah, karena pengembang harus melakukan pengecekan ulang terhadap *reusable assets* yang telah dimodifikasi.

Solusi yang lebih efektif dalam menyelesaikan masalah penulisan kode yang sama secara berulang – ulang adalah dengan menggunakan *code generator*, yaitu suatu aplikasi yang digunakan untuk menghasilkan kode program secara otomatis. *Code generator* dapat membantu para pengembang untuk menghasilkan kode program secara otomatis, sehingga pengembang dapat mengerjakan proses implementasi kode program lebih cepat. Namun cara tersebut belum pernah dicoba dalam menyelesaikan permasalahan PTIIK-Apps, sehingga diperlukan sebuah penelitian untuk menguji cara tersebut.

Code generator akan dibangun dengan bahasa PHP dan ditanam pada *server* FILKOM, sehingga dapat diakses oleh pengembang melalui *web browser*. Arsitektur *server-side programming* digunakan agar *code generator* lebih mudah untuk diperbarui dan didistribusikan untuk persyaratan yang akan datang.

Code generator adalah salah satu klasifikasi dari *metaprogram* (Gabrysiak, Marr and Menge: 1) yang dapat digunakan untuk menyelesaikan permasalahan pada PTIIK-Apps. *Metaprogram* adalah suatu program yang berfungsi untuk menulis kode program yang baru. Definisi lain dari *metaprogram* adalah program komputer yang berfungsi untuk memanipulasi program lain (Hazard and Bock, 2013: 6-7). *Code generator* terdiri dari rancangan *application description* dan *domain knowledge* sebagai sebuah persyaratan dalam membuat *code generator* (Sommerville 8th, 2007, p. 425).

Pada proses perancangan *code generator* dibutuhkan rekayasa domain (*domain engineering*) untuk menganalisis domain pada PTIIK-Apps dan menghasilkan model domain sebagai abstraksi domain PTIIK-Apps. *Domain model* membantu proses perancangan *application description* dan *domain knowledge*. Metode *Feature Oriented Domain Analysis* (FODA) digunakan sebagai panduan dalam melakukan proses rekayasa domain (Cox, 2010) (Harsu, 2015, p.11).

Sebelum ini, terdapat penelitian yang terkait dengan pembuatan *code generator* berjudul "*Automatically Propagating Changes from Reference Implementations to Code Generation Templates*" oleh Marcos Antonio Possatto, Daniel Lucredio pada tahun 2015. Penelitian tersebut menjelaskan pengertian dan konsep dasar dalam membuat *code generator*. Kajian pustaka tersebut dapat digunakan untuk membantu peneliti dalam mengembangkan *code generator* pada domain PTIIK-Apps.

Untuk menyelesaikan permasalahan yang telah dibahas, maka penelitian ini akan membahas tentang proses pengembangan *code generator* yang sesuai dengan domain PTIIK-Apps. Pembahasan tersebut meliputi proses analisis persyaratan, analisis persyaratan, perancangan, implementasi, analisis dan pengujian.

1.2 Rumusan masalah

Rumusan masalah umum pada penelitian ini adalah:

Dapatkah dibangun sebuah prototipe sistem *code generator* yang mendemonstrasikan kemampuan untuk menghasilkan kode program PTIIK-Apps dengan domain yang sesuai?

Rumusan masalah khusus yang diturunkan dari rumusan masalah umum tersebut adalah:

1. Bagaimanakah persyaratan sistem *code generator*?
2. Bagaimanakah hasil rancangan *code generator*?
3. Bagaimanakah hasil implementasi *code generator* yang digunakan untuk menyelesaikan permasalahan?
4. Bagaimanakah hasil pengujian *code generator*?

1.3 Tujuan

Tujuan umum yang ingin dicapai dalam penulisan skripsi ini adalah:

Membuat prototipe sistem *code generator* yang mendemonstrasikan kemampuan untuk menghasilkan kode program PTIIK-Apps.

Tujuan khusus yang ingin dicapai dalam penulisan skripsi ini adalah:

1. Mengidentifikasi dan menganalisis persyaratan untuk membuat sistem *code generator*.
2. Membuat rancangan sistem *code generator*.
3. Melakukan implementasi sistem *code generator* untuk menyelesaikan permasalahan.
4. Melakukan pengujian sistem *code generator* untuk mengetahui kelayakan sistem dalam menyelesaikan permasalahan.

1.4 Manfaat

Manfaat yang diperoleh dari penulisan skripsi ini adalah:

Code generator diharapkan dapat membantu pengembang PTIIK-Apps dalam membuat komponen PTIIK-Apps dengan waktu yang lebih singkat.

1.5 Batasan masalah

Batasan masalah dalam penulisan skripsi ini adalah:

1. Objek yang dijadikan penelitian pada skripsi ini adalah PTIIK-Apps.
2. Bahasa pemrograman yang digunakan untuk membuat *code generator* adalah PHP, Javascript dan HTML.
3. *Database* yang digunakan dalam *code generator* adalah MySQL.
4. Komponen yang dihasilkan oleh *code generator* adalah komponen *model*, *view* dan *controller* dari *framework* Panada.
5. Modul PTIIK-Apps yang akan dianalisis pada proses *domain engineering* adalah modul knnp dan modul kerjasama.

1.6 Sistematika pembahasan

Bagian ini berisi struktur skripsi ini mulai Bab Pendahuluan sampai Bab Penutup dan deskripsi singkat dari masing-masing bab. Diharapkan bagian ini dapat membantu pembaca dalam memahami sistematika pembahasan isi dalam skripsi ini. Sistematika dari laporan penelitian ini adalah sebagai berikut:

1. BAB I PENDAHULUAN

Berisi tentang latar belakang, rumusan masalah, batasan masalah, tujuan penelitian, manfaat penelitian dan sistematika penulisan.

2. BAB II TINJAUAN PUSTAKA

Berisi kajian pustaka serta teori – teori yang berhubungan dengan topik penelitian.

3. BAB III METODOLOGI PENELITIAN

Menjelaskan tentang tahapan – tahapan yang dilakukan dalam penelitian ini.

4. BAB IV ANALISIS PERSYARATAN

Menjelaskan tentang proses analisis persyaratan untuk membangun sistem yang akan dikembangkan.

5. BAB V PERANCANGAN

Menjelaskan tentang proses perancangan sistem *code generator* yang akan dibuat dalam penelitian ini.

6. BAB V IMPLEMENTASI

Menjelaskan tentang implementasi sistem dan pembahasan dari hasil perancangan dan analisis persyaratan.

7. BAB VI PENGUJIAN DAN ANALISIS

Menjelaskan tentang hasil pengujian dan analisis terhadap sistem yang telah direalisasikan.

8. BAB VII PENUTUP

Merupakan bab terakhir dari laporan penelitian yang berisi kesimpulan yang diperoleh dari pembuatan dan pengujian sistem, serta saran – saran untuk pengembangan lebih lanjut.

BAB 2 LANDASAN KEPUSTAKAAN

Bab ini berisi kajian pustaka dan dasar teori yang digunakan untuk menunjang penyusunan skripsi. Kajian pustaka berisi tentang pembahasan penelitian sebelumnya yang berkaitan dengan skripsi ini. Sedangkan dasar teori berisi tentang pembahasan teori yang digunakan untuk menyusun penelitian.

2.1 Kajian pustaka

Kajian pustaka pada skripsi ini adalah berdasarkan penelitian yang berjudul *"Automatically Propagating Changes from Reference Implementations to Code Generation Templates"*. Penelitian tersebut menjelaskan pengertian dan konsep dasar dalam membuat *code generator*. Teknik dan cara pembuatan *code generator* dijelaskan dengan menggunakan cara *template-based*, yaitu suatu cara untuk menghasilkan kode program secara otomatis berdasarkan sebuah *template*. Penelitian tersebut menjelaskan bahwa dalam pengembangan *code generator* terdapat dua tingkatan (level), yaitu tingkatan untuk mengembangkan/menganalisis *code generator* dan tingkatan untuk mengembangkan/menganalisis kode program yang dihasilkan oleh *code generator* tersebut.

Pada penelitian yang berjudul *"Meta Programming and Reflection in PHP"* digunakan sebagai kajian pustaka kedua. Penelitian Gregor Gabrysiak dan rekan – rekannya menjelaskan bahwa *metaprogramming* dapat dilakukan dengan menggunakan bahasa pemrograman PHP. PHP memiliki beberapa fitur yang dapat digunakan sebagai penerapan *metaprogramming*.

Buku yang berjudul *"Feature-Oriented Domain Analysis (FODA) Feasibility Studi"* digunakan sebagai panduan dalam melakukan analisis domain yang merupakan tahap awal dalam melakukan rekayasa domain. Buku karya Ian Sommerville tahun 2011 yang berjudul *"Software Engineering"* edisi 9 digunakan penulis sebagai acuan dalam melakukan proses rekayasa perangkat lunak.

2.2 Bahasa pemrograman PHP

Bahasa pemrograman PHP (*Personal Home Page*) adalah bahasa pemrograman *scripting*. PHP dilengkapi dengan berbagai macam fitur yang sangat membantu (Lockhart, 2015, p.1).

PHP adalah bahasa yang *interpreted server-side scripting*, yaitu setiap bahasa PHP yang ditulis diunggah ke *server* dan diterjemahkan oleh *interpreter* di sisi *web server* (biasanya menggunakan *Apache*). Pada awalnya Rasmus Lerdorf (pembuat PHP) memulai membuat PHP dengan sekumpulan CGI (*Common Gateway Interface*) *script* untuk mengetahui pengunjung resume online miliknya. Lerdorf memberi nama kumpulan CGI *script* tersebut *"personal home page tools"*, pada saat itu PHP bukanlah suatu *scripting* language. Kemudian PHP dikembangkan oleh Andi Gutmans dan Zeev Suraski. Mereka mengembangkan PHP menjadi suatu

bahasa dengan sintaks yang konsisten dan mendukung konsep dasar *object-oriented* (PHP 3) yang liris pada tahun 1998 (Lockhart, 2015, p.2).

Sekarang bahasa pemrograman PHP berevolusi dengan cepat dan dikembangkan oleh tim dari berbagai negara. Berdasarkan literatur "*Meta Programming and Reflection in PHP*" telah membuktikan bahwa bahasa PHP dapat digunakan untuk implementasi *metaprogramming*.

2.3 PTIIK-Apps

PTIIK-Apps adalah sebuah sistem informasi yang berguna untuk mengelola proses bisnis yang ada di Fakultas Ilmu Komputer (FILKOM) Universitas Brawijaya (UB) Malang. PTIIK-Apps adalah aplikasi berbasis web, yang bisa diakses pada alamat <http://ptiik.ub.ac.id>. PTIIK-Apps memiliki banyak fitur yang dibagi dalam modul – modul. Fitur yang bisa diakses tanpa harus melakukan *login* adalah fitur – fitur yang menyediakan informasi umum berupa informasi berita, biodata dosen dan karyawan, profile fakultas dan sebagainya. Fitur – fitur yang membutuhkan pengecekan hak akses adalah fitur yang digunakan oleh para karyawan, dosen dan mahasiswa FILKOM untuk mengelola data fakultas seperti: modul peminjaman ruangan, modul berita dan lain – lain.

PTIIK-Apps dikembangkan menggunakan bahasa pemrograman PHP dan *Javascript*. *Framework* PHP yang digunakan adalah *Panada Framework* dimana dokumentasinya dapat dilihat pada alamat berikut ini: <http://panadaframework.com>. PTIIK-Apps menggunakan *framework Bootstrap* untuk mengembangkan sisi *interface* (tatap muka), dimana dokumentasinya dapat dilihat pada halaman berikut ini: <http://getbootstrap.com>.

PTIIK-Apps dikembangkan oleh Badan Pengembangan Teknologi Informasi dan Ilmu Komputer (BPTIK). PTIIK-Apps dikembangkan dengan konsep *Model View Controller* (MVC) yang dikemas dalam sebuah bentuk hirarkikal. Hirarkikal MVC adalah suatu struktur MVC yang didalamnya terdapat struktur MVC lagi. Setiap MVC *child* (cabang) mewakili setiap modul PTIIK-Apps.

Proses pengembangan PTIIK-Apps masih belum selesai sepenuhnya, karena masih banyak proses bisnis fakultas yang belum bisa dikelola oleh PTIIK-Apps. Karena proses pengembangannya masih belum selesai, maka penambahan modul baru masih memungkinkan. Proses yang dilakukan untuk mengembangkan modul baru adalah dengan menambahkan komponen MVC dengan kelas dan fungsi baru.

2.3.1 *Reusable assets* pada PTIIK-Apps

Reusable assets adalah sebuah sekumpulan komponen (*assets*) perangkat lunak yang dapat digunakan ulang untuk pengembangan perangkat lunak baru (*software reuse*). *Reusable assets* terdiri dari beberapa macam komponen, seperti: 1) sekumpulan kode program untuk menjalankan fungsional tertentu, 2) sebuah fitur, 2) modul sistem ,3) arsitektur sistem, 4) domain sistem (Spoelstra & Wouter, 2010, p.8-9).

Reusable assets membantu para pengembang untuk membuat komponen perangkat lunak. Pengembang tidak perlu membuat perangkat lunak dari awal, cukup mengambil sumber dari *reusable assets* dan dimodifikasi sesuai dengan persyaratan (Spoelstra & Wouter, 2010, p.1).

Proses penggunaan ulang dengan bantuan *reusable assets* bukan merupakan solusi yang terbaik. Karena pengembang harus menguji setiap komponen yang dibuat dari *reusable assets*.

2.4 Definisi *metaprogramming*

Metaprogramming berasal dari kata *meta* dan *programming*. *Programming* adalah sebuah proses penulisan kode program untuk membuat sistem komputer. Secara lebih spesifik, *programming* adalah suatu proses untuk menerjemahkan tugas yang diberikan kedalam bentuk/spesifikasi yang bisa diterjemahkan/dieksekusi oleh komputer. Kata *meta* sebagai prefix memiliki makna bahwa *metaprogramming* adalah proses *programming* dalam tingkatan/level yang lebih tinggi. *Metaprogramming* adalah paradigma pemrograman tingkat tinggi yang bertujuan untuk membangun kode program secara otomatis (Stuikys & Damasevicius, 2013, p.3).

Metaprogramming berdasarkan proses implemetasinya dibagi menjadi beberapa taksonomi, yaitu:

1. *Transformation*: proses untuk merubah sebuah bentuk program menjadi bentuk yang lainnya. (*manipulation, modification and adaption*)
2. *Generation*: proses pembuatan sistem target secara otomatis berdasarkan sebuah spesifikasi. (*program generation, code generation*)
3. *Reflection*: kemampuan sebuah program untuk mengamati dan memodifikasi struktur dan perilakunya sendiri pada saat sistem dijalankan. (*intropection, intersection*)
4. *Generalization*: proses transformasi sebuah domain yang spesifik kedalam komponen yang umum sehingga dapat digunakan untuk tujuan yang lebih luas dan lebih bisa digunakan ulang dari bentuk asalnya (*parametization*) (Stuikys & Damasevicius, 2013, p.24).

Metaprogramming adalah sebuah program komputer yang dapat menulis kode program komputer baru. Definisi tersebut sekilas hampir sama seperti *compiler*. Namun tugas *compiler* hanya menghasilkan program lain kedalam bentuk *executable code* (bentuk kode program yang bisa langsung dijalankan oleh komputer) dari sebuah *source code*. Teknik *metaprogramming* sangat berbeda dengan *compiler*, karena *metaprogram* selalu digunakan untuk berurusan dengan tipe masukan yang berbeda – beda (Hazard & Bock, 2013, p.6).

Teknik *metaprogramming* berbeda dengan *compiler*, karena *metaprogramming* membutuhkan sebuah parameter sebagai acuan, misalnya untuk menghasilkan komponen MVC secara otomatis dibutuhkan parameter

berupa skema tabel *database*. Parameter tersebut digunakan untuk membuat kode program agar bisa beradaptasi dengan domainnya (Hazard and Bock, 2013, p.6).

Metaprogramming dapat digunakan untuk menghasilkan program secara otomatis, sehingga tingkat produksi pengembangan perangkat lunak akan meningkat. Alasan kenapa *metaprogramming* dibutuhkan adalah untuk menangani permasalahan perangkat lunak dengan spesifikasi: a) kompleksitas sistem yang tinggi dan terus menerus berkembang, b) persyaratan penambahan komponen perangkat lunak secara terus menerus pada suatu sistem (Stuikys & Damasevicius, 2013, p.8).

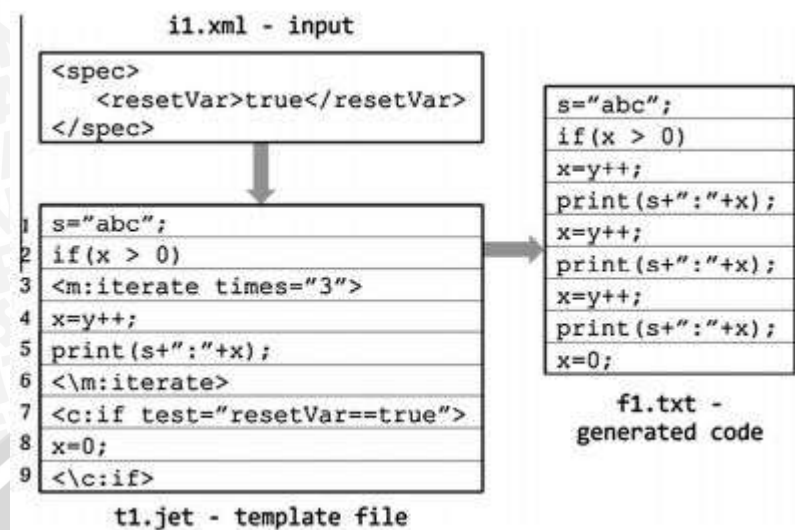
Dalam membangun *metaprogram* dibutuhkan analisis domain. Analisis domain dibutuhkan agar kode program yang dihasilkan dapat beradaptasi dengan baik dengan lingkungannya. (Stuikys and Damasevicius, 2013: 11). Analisis domain dapat dilakukan dengan metode *Feature Oriented Domain Analysis* (FODA) (Stuikys & Damasevicius, 2013, p.152).

2.4.1 Code generator

Code generator adalah aplikasi yang bisa digunakan dalam menghasilkan kode program secara otomatis berdasarkan masukan sebagai acuan, masukan tersebut berupa 1) model/rancangan sistem, 2) beberapa parameter konfigurasi (Possatto & Lucrédio, 2015, p.1).

Teknik *template-based* merupakan sebuah cara untuk menghasilkan kode secara otomatis. Proses pengembangan *code generator* dengan *template-based* memiliki banyak tantangan. Jika dibandingkan dengan kode program biasa, *template* memiliki kompleksitas yang lebih rumit, karena *template* terdiri dari perpaduan 1) kode program yang akan dihasilkan dan 1) kode yang digunakan untuk menghasilkan kode program baru. *Template* harus bisa mengolah masukan yang beragam dan harus bisa mengimplementasikan semua fungsionalitas yang dibutuhkan (Possatto & Lucrédio, 2015, p.1).

Template adalah bagian dari perangkat lunak yang didesain untuk menghasilkan output textual (berupa kode program). *Template* menggunakan masukan sebagai acuan, masukan tersebut bisa berupa model dari UML diagram. Model (yang digunakan sebagai acuan) merupakan spesifikasi dari suatu konsep domain yang akan dibaca dan digunakan selama proses pembuatan kode program. Jika model yang digunakan sebagai acuan diambil dari *class diagram* maka, model yang dikirimkan kepada *template* dapat berupa informasi konseptual seperti: nama *class*, *attribute* dan *method*. *Template* berisi bagian kode program yang belum lengkap misalkan: 1) definisi struktur *class*, 2) beberapa label (*tags*) dan 3) komentar (*comments*) yang mengimplementasikan logika dalam melakukan pengolahan keluaran (*output*). Selama proses pengolahan kode, *tags* dan *comments* dieksekusi dan diganti berdasarkan informasi yang dibaca dari masukan (Possatto & Lucrédio, 2015, p.2).



Gambar 2.1 Contoh Template-based Code Generation

Sumber: (Possatto & Lucrédio, 2015, p.3)

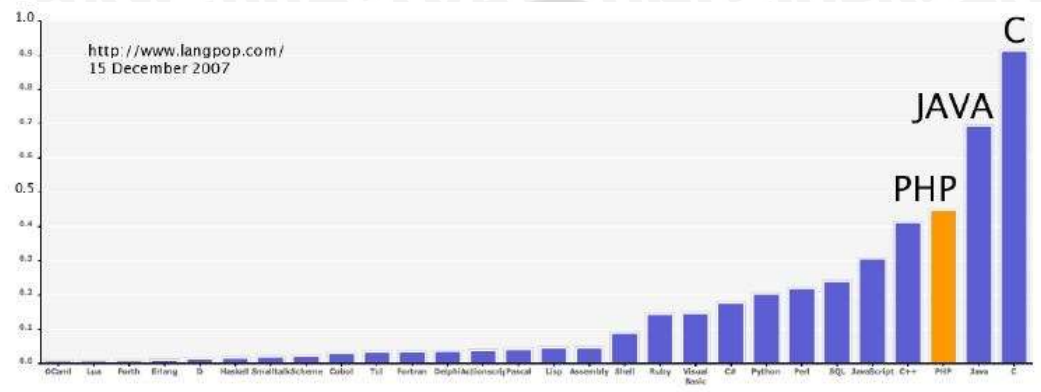
Gambar 2.1 merupakan ilustrasi *template-based code generation*. Pada contoh tersebut masukan berupa file i1.xml, dan *template* yang digunakan untuk mengolah kode adalah file t1.jet. *Template* tersebut akan menghasilkan kode program seperti f1.txt. File t1.jet memiliki label XML yang digunakan untuk mengimplementasikan logika pengolahan kode (*code generation*). Sebagai contoh, label <m:iterate> menjalankan proses perulangan dan label <c:if> menjalankan proses pemeriksaan kondisi. File *template* t1.jet juga memiliki beberapa baris kode normal yang digunakan sebagai baris kode yang akan dihasilkan (Possatto & Lucrédio, 2015, p.2).

Berdasarkan contoh Gambar 2.1, ditemukan kendala berupa kompleksitas ekstra, karena kode yang akan dihasilkan bercampur dengan kode untuk menjalankan proses pengolahan. Hal ini dapat dilihat dari pada file *template* t1.jet terdapat dua perintah kondisi, yaitu pada baris 2 dan 7. Baris 2 merupakan perintah kondisi yang tidak akan dieksekusi selama proses pengolahan kode, karena perintah tersebut merupakan kode yang akan dihasilkan pada keluaran f1.txt. Sedangkan perintah pada baris ke 7 didesain agar bisa dieksekusi dan menjalankan logika proses pengolahan kode (Possatto & Lucrédio, 2015, p.2-3).

Pengembangan *code generator* dilakukan dengan dua tingkatan (*level*), yaitu tingkatan kode program yang dihasilkan dan tingkatan *code generator*. Jika dipandang dari aspek pengujian, maka dalam pengembangan *code generator* harus dilakukan dua tingkatan pengujian yaitu: 1) untuk memastikan bahwa *code generator* dapat bekerja sesuai dengan harapan, 2) untuk menguji bahwa kode yang dihasilkan dapat dieksekusi dengan benar pada lingkungannya (Possatto & Lucrédio, 2015, p.2-3).

2.4.2 Metaprogramming pada PHP

PHP merupakan salah satu dari bahasa pemrograman yang populer untuk aplikasi web. Seperti bahasa pemrograman lainnya, PHP menyediakan sekumpulan fungsi metaprogramming (Gabrysiak, Marr and Menge: 1). Berikut grafik yang menunjukkan tingkat popularitas bahasa pemrograman.



Gambar 2.2 Popularitas bahasa pemrograman berdasarkan jumlah buku yang tersedia di Amazon dan website

Sumber: (Gabrysiak, Marr & Menge, p.3)

Sejak versi 4 yang rilis pada tahun 2000, fungsi *object model* telah ditambahkan pada PHP untuk mengubah fokus PHP yang awalnya hanya memiliki paradigma *procedural programming* saja (Gabrysiak, Marr & Menge, p.3).

Magic method pada PHP digunakan untuk menyesuaikan tindakan saat *object cloning* atau saat operasi properti objek. (Gabrysiak, Marr & Menge, p.4) Berikut ini penjelasan *magic method* dan beberapa *method* lain yang mendukung *metaprogramming* pada PHP.

2.4.3 Fungsi `__autoload()`

Fungsi *autoload* berguna untuk melakukan pemanggilan kelas secara otomatis tanpa harus didefinisikan satu per satu (PHP). Berikut contoh implementasi kode *autoload*:

Tabel 2.1 Kode program *autoload*

1	<code><?php</code>
2	<code>function __autoload(\$class_name) {</code>
3	<code>include 'class/' . \$class_name . '.php';</code>
4	<code>}</code>
5	<code>\$obj = new MyClass1();</code>
6	<code>?></code>

Kode pada tabel 2.1 menjelaskan tentang penggunaan fungsi *autoload*. Ketika objek baru dari kelas `MyClass1()` dibuat. Maka fungsi *autoload* secara otomatis

akan dijalankan, dimana isi dari fungsi tersebut berupa perintah `include`. Perintah `include` digunakan untuk menyisipkan suatu file pada file yang bersangkutan (PHP).

2.4.4 Fungsi `__get()` dan `__set()`

Fungsi `__get()` digunakan untuk membaca data *inaccessible* dari suatu objek. Fungsi `__set()` digunakan untuk menulis data *inaccessible* dari suatu objek. Data *inaccessible* adalah suatu atribut objek yang tidak dideklarasikan pada objek tersebut, data *inaccessible* dibuat dengan perintah `__set()`. Berikut contoh penggunaannya:

Tabel 2.2 Kode program fungsi `__set()` dan `__get()`

1	<?php
2	class Manusia {
3	private \$vars = array();
4	public function __get(\$name) {
5	return \$this->vars[\$name];
6	}
7	public function __set(\$name, \$value) {
8	\$this->vars[\$name] = \$value;
9	}
10	}
11	?>

Implementasi kode pada tabel 2.2 membuat suatu kelas dengan nama Manusia. Kelas tersebut memiliki atribut `$vars` dengan tipe data *array* dan memiliki dua *method*. Atribut *array* digunakan untuk menampung data *inaccessible*. *Method* `__set()` dan `__get()` digunakan untuk mengelola data *inaccessible* pada kelas tersebut (PHP). Berikut ini contoh cara mengelola data *inaccessible* pada kelas Manusia.

Tabel 2.3 Kode program untuk membaca data *inaccessible*

1	<?php
2	\$view = new Manusia();
3	\$view->nama = "Agie Ghazy";
4	echo \$view->nama;
5	?>

Kode program pada tabel 2.3 baris ke-2 berfungsi untuk membuat objek dari kelas Manusia. Baris ke-3 adalah perintah untuk menjalankan fungsi `__set()`, data *inaccessible* akan dibuat pada objek `$view` dengan *key* = nama dan *value* =

“Agie Ghazy”. Baris ke-4 menjelaskan penggunaan fungsi `__get()`, dimana membaca data *inaccessible* yang memiliki *key* = nama. Kode program baris ke-4 akan mengembalikan nilai “Agie Ghazy”.

2.4.5 Fungsi *callback*

Fungsi *callback* adalah suatu fungsi yang menggunakan fungsi *lain* sebagai parameter yang dikirim (PHP). Fungsi *callback* digunakan untuk melakukan eksekusi kode program yang sedang diproses oleh *code generator*. Fungsi *callback* digunakan untuk menghasilkan kode program pada saat *runtime* (Gabrysiak, Marr and Menge:6). Berikut contoh penggunaan fungsi *callback*:

Tabel 2.4 Kode program fungsi *callback*

1	<?php
2	function my_callback_function() {
3	echo 'hello world!';
4	}
5	call_user_func('my_callback_function');
6	?>

Kode program pada tabel 2.4 menjelaskan tentang penggunaan fungsi *callback*. Baris kode ke-2 mendefinisikan fungsi biasa, pada baris ke-5 adalah suatu contoh pemanggilan fungsi *callback*, dimana parameter yang digunakan adalah nama fungsi pada baris ke-2.

2.4.6 Fungsi *ob_start()*

Fungsi *ob_start()* digunakan untuk mengaktifkan *output buffering* pada PHP. Ketika *output buffering* aktif maka tidak ada *output* yang dikirimkan oleh *script*, bahkan *output* tersebut tersimpan di dalam *internal buffer*. (PHP)

Konten di dalam *internal buffer* dapat disalin kedalam variabel *string* dengan menggunakan perintah *ob_get_content()*. Perintah *ob_end_flush()* dapat digunakan untuk mengeluarkan konten yang tersimpan pada *internal buffer*. Perintah *ob_clean()* digunakan untuk mengosongkan isi pada *internal buffer*. Berikut ini contoh penggunaan fungsi *ob_start()* (PHP).

Tabel 2.5 Kode program *ob_start()*

1	<?php
2	function callback(\$buffer){
3	// replace all the apples with oranges
4	return (str_replace("apples", "oranges",
5	\$buffer));
	}

6	<code>ob_start("callback");</code>
7	<code>?></code>
8	<code><html></code>
9	<code><body></code>
10	<code><p></code>
11	<code>comparing apples to oranges.</code>
12	<code></p></code>
13	<code></body></code>
14	<code></html></code>
15	<code><?php</code>
16	<code>ob_end_flush();</code>
17	<code>ob_clean();</code>
18	<code>?></code>
19	

Kode pada tabel 2.5 menjelaskan penggunaan fungsi `ob_start()` dan `callback`. Baris kode HTML pada baris ke-9 sampai baris ke-15 menampilkan konten *"comparing apples to oranges"*. Namun pada baris ke-7 perintah `ob_start()` dijalankan, sehingga konten tidak akan tampil pada halaman *browser* karena konten tersimpan di *internal buffer*. Baris ke-17 berfungsi untuk menampilkan konten yang ada pada *internal buffer*, sehingga muncul pada tampilan *browser*. Ketika kode baris ke-19 dijalankan maka tampilan *browser* akan kosong, karena konten yang ada di *internal buffer* dibersihkan dengan perintah `ob_clean()`.

Pada baris ke-7, saat penggunaan perintah `ob_start()` juga dilakukan pemanggilan fungsi `callback`. Fungsi `ob_start()` menggunakan nama fungsi baris ke-2 sebagai parameter. Fungsi `callback` menjalankan perintah untuk mengganti kata *apples* menjadi *oranges*. Sehingga kata *apples* pada konten *internal buffer* akan berubah menjadi *oranges*.

2.4.7 Fungsi `extract()`

Fungsi `extract()` digunakan untuk membuat suatu variabel berdasarkan struktur sebuah *array* [PHP]. Berikut ini contoh penggunaan fungsi `extract()`:

Tabel 2.6 Tabel kode program fungsi `extract`

1	<code><?php</code>
2	<code>\$tes = array(</code>
3	<code> "name" => "agie",</code>
4	<code> "gender" => "male"</code>
5	<code>);</code>

6	
7	<code>extract(\$tes);</code>
8	<code>echo \$name . ' ' . \$gender;</code>
9	<code>?></code>

Tabel 2.6 menjelaskan penggunaan perintah `extract()`, baris kode ke-7 berfungsi untuk merubah array `$tes` pada baris ke-2 menjadi variabel `$name` dan `$gender`. Jika perintah `echo` dijalankan pada kedua variabel tersebut (pada baris ke-8) maka akan menghasilkan nilai "agie male".

2.5 Template programming dengan PHP

Template programming adalah suatu metode atau cara untuk memisahkan kode program yang dibuat oleh *programmer* (PHP) dari kode program yang dibuat oleh *designer* (HTML) (Gheorghe, Hayderh and Maia 2006: 5). Alasan mengapa *template programming* dibutuhkan adalah karena semakin berkembangnya aplikasi web. Suatu project web yang besar seperti *E-commerce* dan sistem ERP (*EnterpriseResource Planning*) akan menjadi rumit. Maka dari itu domain kerja *programmer* dan *designer* harus dipisahkan tetapi harus saling terintegrasi (Gheorghe, Hayderh & Maia 2006, p.6).

Template programming menggunakan struktur logika pada suatu *template*. *Template* yang digunakan merupakan lapisan yang digunakan untuk menampilkan data dan terpisah dengan lapisan pengolah data. Misalkan ada sebuah fitur menampilkan data daftar produk, maka *programmer* hanya perlu melemparkan data produk ke dalam *template* dengan bentuk *array*. *Designer* hanya perlu mengolah *array* yang berisi data produk pada *tempate* sesuai dengan keinginannya (Gheorghe, Hayderh & Maia 2006, p.10). Berikut ini contoh penggunaan *template programming* dengan *template engine* Smarty:

Tabel 2.7 Kode program sample.php

Sumber: (Gheorghe, Hayderh & Maia, 2006, p.53)

1	<code><?php</code>
2	<code>include("libs/smarty.class.php");</code>
3	<code>\$smarty = new smarty();</code>
4	<code>\$sample_name = "Jamil Ahmed";</code>
5	<code>\$smarty->assign("name", \$sample_name);</code>
6	<code>\$smarty->assign("greetings", "Good Morning");</code>
7	<code>\$smarty->display("sample.tpl");</code>
8	<code>?></code>

Kode program pada tabel 2.7 merupakan implementasi *template programming*. Pada baris ke-2 adalah perintah untuk mengimpor plugin Smarty,

baris ke-3 merupakan perintah untuk membuat objek `$smarty` (kelas tersebut menyediakan fitur *template programming*). Pada baris ke-6 dan 7 merupakan proses inisialisasi variabel dengan `keynama` dan `greetings`. Selanjutnya dari data yang ada pada kelas `$smarty`, akan ditampilkan pada `template sample.tpl`. Pada tabel 2.8 akan dijelaskan penggunaan *template* untuk mengolah data yang dikirimkan.

Tabel 2.8 Kode program `template.tpl`

Sumber: (Gheorghe, Hayderh & Maia 2006, p.52)

1	<html>
2	<body>
3	Hello {\$name}, {\$greetings}.
4	</body>
5	</html>

Kode program pada tabel 2.8 adalah *template* yang digunakan pada *template programming*. *Template* tersebut berada pada file yang berbeda dengan kode program pada tabel 2.7. Tanpa ada inisialisasi variabel, *template* pada tabel 2.8 akan menampilkan data `$name` dan `$greetings` sesuai dengan nilai yang diterima. Pengiriman data tersebut dilakukan pada perintah tabel 2.7 baris ke-7.

Berdasarkan konsep tersebut, peneliti berusaha untuk memodifikasi *template programming* untuk menghasilkan kode program baru. Peneliti membuat kelas yang mendukung fungsi *template programming* tanpa bantuan *template engine* Smarty. *Template* yang telah dijelaskan sebelumnya digunakan oleh peneliti untuk merepresentasikan komponen MVC yang akan dihasilkan. Peneliti menggunakan struktur database sebagai parameter. *Template* yang dibuat akan beradaptasi sesuai dengan parameter yang diterima.

Hasil dari pengolahan *template* tersebut tidak langsung dikeluarkan sebagai *output*, namun ditampung dalam *internal buffer* dan diolah kembali agar menjadi suatu bentuk kode program baru. Setelah pengolah tersebut selesai, maka hasilnya akan disimpan ke dalam sebuah variabel dengan tipe *string* dan ditampilkan kepada pengguna sebagai sebuah komponen MVC yang dibutuhkan.

2.6 Database MySQL

MySQL adalah sebuah database management system, yaitu sebuah aplikasi yang berfungsi untuk mengelola sebuah basis data. Untuk membuat koneksi dari aplikasi ke basis data dapat menggunakan perintah `mysql_connect($hostname, $username, $password)`. Parameter `$hostname` merupakan alamat *host* dimana *database* berada, parameter `$username` dan `$password` berisi *username* dan *password* dari pengguna yang diberi hak akses terhadap database. Jika koneksi berhasil dibuat, selanjutnya dapat menjalankan perintah untuk memilih nama database mana yang ingin

dipilih dengan memasukkan perintah `mysql_select_db($dbname)` (Oracle, 2015, p.3).

Dalam penelitian ini, kajian basis data MySQL yang digunakan adalah informasi data dari sekumpulan tabel pada *information schema*. Tabel tersebut memungkinkan pengguna untuk mengakses metadata dari basis data yang bersangkutan. Metadata yang dimaksud adalah data dari objek data yang ada pada basis data, misalnya: 1) nama basis data, 2) nama tabel, 3) tipe data dari suatu kolom, 4) hak akses dan sebagainya. Tabel pada *information schema* merupakan tabel yang tidak dapat diubah isi datanya (*read-only*) (Oracle Corporation, 2016).

Tabel pada *information schema* yang dibutuhkan dalam penelitian ini adalah 1) tabel *information_schema.schemata* untuk mendapatkan metadata skema basis data dan 2) tabel *information_schema.columns* untuk mendapatkan metadata kolom tabel pada suatu basis data (Oracle Corporation, 2016).

2.6.1 Tabel *information_schema.schemata*

Tabel *information_schema.schemata* memberikan informasi tentang metadata dari skema basis data. Perintah yang bisa digunakan untuk mengambil informasi skema basis data dijelaskan pada Tabel 2.9.

Tabel 2.9 Perintah untuk Melihat Metadata Skema Basis Data

1	SELECT SCHEMATA.SCHEMA_NAME
2	FROM INFORMATION_SCHEMA.SCHEMATA

2.6.2 Tabel *information_schema.columns*

Tabel *information_schema.columns* memberikan informasi metadata dari kolom suatu tabel basis data. Informasi yang bisa didapat adalah: 1) nama skema basis data, 2) nama tabel, 3) nama kolom, 4) urutan posisi, 5) penilaian otomatis kolom jika kosong, 6) setelah bahwa kolom boleh berisi data kosong (*nullable*), 7) tipe data, 8) karakter maksimum dan 9) setelah keunikan kolom (*column key*).

Tabel *information_schema.key_column_usage* digunakan untuk mendapatkan metadata hubungan antar kolom (*foreign key*). Informasi yang bisa didapatkan dari tabel tersebut adalah: 1) nama skema, 2) nama batasan (*constraint*), 3) nama tabel, 4) nama kolom, 5) urutan posisi, 6) nama skema yang terhubung, 7) nama tabel yang terhubung, dan 8) nama kolom yang terhubung.

Berikut ini perintah yang bisa digunakan untuk mengambil informasi metadata kolom pada tabel basis data.

Tabel 2.10 Perintah untuk Melihat Metadata Kolom Basis Data

1	SELECT *
2	FROM INFORMATION_SCHEMA.COLUMNS
3	LEFT JOIN INFORMATION_SCHEMA.KEY_COLUMN_USAGE
4	On KEY_COLUMN_USAGE.CONSTRAINT_SCHEMA

5	= COLUMNS.TABLE_SCHEMA
6	AND KEY_COLUMN_USAGE.TABLE_NAME
7	= COLUMNS.TABLE_NAME
8	AND COLUMNS.COLUMN_NAME
9	= KEY_COLUMN_USAGE.COLUMN_NAME

2.7 Software Reuse Menurut Ian Sommerville

Rekayasa penggunaan ulang perangkat lunak (*reuse-based software engineering*) adalah strategi rekayasa perangkat lunak dimana proses pengembangan diarahkan untuk menggunakan kembali perangkat lunak yang sudah ada. *Reuse-based software engineering* digunakan untuk memenuhi persyaratan: tingkat produksi *software* dan biaya perawatan yang lebih rendah, pembuatan sistem yang lebih cepat dan meningkatkan kualitas *software*.

Unit perangkat lunak yang digunakan ulang, pada umumnya dibagi kedalam beberapa ukuran. Contohnya:

1. *Application system reuse*: keseluruhan sistem aplikasi mungkin akan digunakan ulang tanpa ada perubahan atau dengan mengkonfigurasi aplikasi untuk disesuaikan dengan pelanggan yang berbeda.
2. *Component reuse*: penggunaan ulang pada komponen perangkat lunak, komponen perangkat lunak meliputi subsystem sampai objek tunggal dalam sebuah sistem.
3. *Object and function reuse*: penggunaan ulang pada fungsi tunggal dalam sebuah komponen atau objek perangkat lunak. Misalnya fungsi matematis suatu objek (Sommerville, 2011, p.426 – 427).

Penggunaan ulang perangkat lunak memiliki beberapa manfaat, yaitu:

1. Meningkatkan tingkat kepercayaan: perangkat lunak yang dihasilkan dari proses penggunaan ulang, yang mana telah diuji akan memiliki ciri – ciri yang mirip dengan perangkat lunak induk. Sehingga bisa lebih diandalkan dari pada perangkat lunak baru.
2. Mengurangi resiko proses: biaya dalam pembuatan perangkat lunak yang sebelumnya sudah diketahui. Sehingga ketika perangkat lunak tersebut akan digunakan ulang, maka akan mengurangi tingkat kesalahan dalam hal estimasi biaya proyek.
3. Lebih efektif: sebuah aplikasi dapat dikembangkan dengan penggunaan ulang perangkat lunak yang sudah ada, sehingga tidak perlu melakukan pekerjaan yang sama secara berulang – ulang.
4. Standarisasi aplikasi: beberapa standar dapat diterapkan kedalam beberapa aplikasi melalui proses penggunaan ulang.
5. Mempercepat proses pengembangan: penggunaan ulang dapat mempercepat produksi sistem karena waktu proses pengembangan dan pengujian dapat dikurangi (Sommerville, 2011, p.427).

Dalam penerapan pengembangan penggunaan ulang, terdapat beberapa hambatan, yaitu:

1. Meningkatkan maintenance costs: jika kode program aplikasi yang dihasilkan dari proses penggunaan ulang tidak tersedia, maka biaya pemeliharaan sistem akan lebih tinggi. Karena elemen yang digunakan ulang akan menjadi semakin tidak sesuai jika terjadi perubahan sistem.
2. Kurang alat pendukung: beberapa alat (*tools*) tidak mendukung proses pengembangan dengan penggunaan ulang.
3. Asumsi umum dari pengembang: beberapa pengembang perangkat lunak memilih untuk menulis ulang kode program karena: 1) mereka percaya mereka dapat mempertajam kode program tersebut, 2) kode program yang mereka tulis sendiri akan lebih terpercaya, 3) menulis kode program ulang lebih menantang daripada menggunakan ulang perangkat lunak orang lain.
4. Membuat komponen yang bisa digunakan ulang: untuk membuat komponen yang bisa digunakan ulang dan untuk memastikan bahwa setiap pengembang dapat menggunakan komponen tersebut membutuhkan biaya dan waktu yang besar.
5. Menemukan, mempelajari dan mengadaptasi komponen untuk digunakan ulang: komponen perangkat lunak harus dipelajari dan diadaptasi agar bisa bekerja dengan lingkungan baru. Sehingga pengembang memerlukan waktu yang lama untuk melakukan hal tersebut (Sommerville, 2011, p.428).

2.7.1 Dasar Penggunaan Ulang Perangkat Lunak

Sekarang banyak teknik yang telah dikembangkan untuk mendukung penggunaan ulang perangkat lunak. Teknik – teknik tersebut mengemukakan fakta bahwa sistem didalam domain aplikasi yang sama adalah mirip dan memiliki potensi untuk digunakan ulang. Kemungkinan tersebut berada didalam beberapa tingkatan mulai dari fungsi yang sederhana sampai keseluruhan aplikasi. Tabel 2.11 menjabarkan sekumpulan cara yang bisa digunakan untuk menerapkan penggunaan ulang perangkat lunak (Sommerville, 2011, p.428).

Tabel 2.11 Teknik yang Mendukung Penggunaan Ulang Perangkat Lunak

Teknik	Deskripsi
<i>Architectural patterns</i>	Arsitektur perangkat lunak standar yang mendukung tipe umum sistem aplikasi digunakan sebagai arsitektur dasar
<i>Design patterns</i>	Abstraksi umum dari sebuah aplikasi, yang menunjukkan objek konkret dan abstrak beserta interaksinya
<i>Component-based development</i>	Sistem dikembangkan dengan melakukan integrasi komponen (sekumpulan objek) yang sesuai dengan model komponen standar

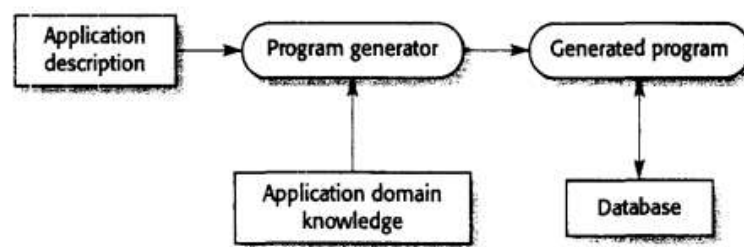
<i>Application framework</i>	Sekumpulan kelas abstrak dan konkret diadaptasikan dan diperluas untuk membuat sistem aplikasi
<i>Legacy system wrapping</i>	Sistem yang lama (<i>legacy</i>) dibungkus dengan mendefinisikan sekumpulan <i>interface</i> dan menyediakan akses terhadap sistem yang lama melalui <i>interface</i> tersebut
<i>Service-oriented systems</i>	Sistem dikembangkan dengan menghubungkan layanan bersama
<i>Software product lines</i>	Sebuah tipe aplikasi yang disamakan dengan arsitektur umum sehingga dapat beradaptasi terhadap pelanggan yang berbeda
<i>COTS product reuser</i>	Sistem dikembangkan dengan melakukan konfigurasi dan integrasi terhadap aplikasi yang sudah ada
<i>ERP system</i>	Sistem berskala besar untuk sebuah organisasi yang merangkum fungsi proses bisnis dan aturan – aturan yang telah dikonfigurasi
<i>Configurable vertical application</i>	Desain sebuah sistem yang bisa dikonfigurasi untuk mencukupi persyaratan dari sistem spesifik yang dibutuhkan pengguna
<i>Program libraries</i>	Sekumpulan kelas dan fungsi yang dapat digunakan untuk digunakan ulang
<i>Model-driven engineering</i>	Perangkat lunak digambarkan sebagai model domain dan model independen, kode program dihasilkan dari model tersebut secara otomatis
<i>Program generator</i>	Sebuah sistem yang dilengkapi dengan sekumpulan pengetahuan, sehingga dapat digunakan untuk menghasilkan sebuah sistem pada suatu domain dari sebuah model sistem tertentu
<i>Aspect-oriented software development</i>	Sekumpulan komponen bersama disusun menjadi sebuah aplikasi pada tempat yang berbeda ketika program tersebut di-jalankan

Berdasarkan sekumpulan teknik untuk penggunaan ulang, muncul sebuah pertanyaan “teknik mana yang paling tepat untuk digunakan dalam situasi tertentu?”. Hal tersebut tergantung pada: daftar persyaratan dari sistem yang akan dikembangkan, teknologi yang dimiliki dan *reusable assets* yang tersedia, dan tingkat keahlian yang dimiliki tim pengembang. Faktor – faktor yang harus dipertimbangkan dalam merencanakan penggunaan ulang perangkat lunak adalah sebagai berikut:

1. Jadwal pengembangan perangkat lunak: jika perangkat lunak harus dikembangkan dengan cepat, maka disarankan untuk menggunakan keseluruhan sistem daripada komponen individual. Kesesuaian sistem dengan persyaratan mungkin kurang sempurna, karena pendekatan ini meminimalisasikan proses pengembangan yang dibutuhkan.
2. Jangka waktu perangkat lunak yang diharapkan: jika ingin mengembangkan sistem dengan jangka waktu yang panjang, maka tim harus fokus pada tingkat perawatan sistem (*maintainability*). Sistem harus didesain agar bisa ditingkatkan dengan mudah.
3. Tingkat keahlian yang dimiliki tim pengembang: kemampuan/keahlian tim menjadi faktor penting dalam pengembangan penggunaan ulang, diharapkan tim memiliki potensi yang cukup untuk mengembangkan aplikasi yang sesuai dengan harapan.
4. Tingkat kekritisan perangkat lunak dan persyaratan non-fungsional yang diharapkan: untuk sistem kritis (misal dibutuhkan tingkat performa yang tinggi), diperlukan akses yang mendalam terhadap perangkat lunak yang bersangkutan. Sehingga tim pengembang bisa memodifikasi persyaratan non-fungsional dari sistem yang akan digunakan ulang.
5. Domain aplikasi: pada beberapa domain aplikasi, seperti sistem manufaktur dan sistem informasi medis, ada beberapa produk umum yang mungkin bisa digunakan ulang dengan mengkonfigurasi sistem tersebut didalam situasi lokal.
6. *Platform* dimana sistem akan berjalan: beberapa sistem aplikasi mungkin membutuhkan *platform* tertentu dan hanya bisa digunakan ulang kedalam *platform* yang sama (Sommerville, 2011, p.428 – 430).

2.7.2 Generator-Based Reuse

Konsep penggunaan ulang perangkat lunak (*software reuse*) melalui sebuah pola yang dideskripsikan dengan model abstrak, selanjutnya proses implementasi dilakukan oleh *software developer* (pengembang perangkat lunak). Pendekatan yang dapat digunakan adalah *generator-based reuse*, pendekatan tersebut membutuhkan 1) *application description*, dan 2) *application domain knowledge* sebagai masukan. Hasil keluaran dari pendekatan tersebut berupa sistem perangkat lunak yang dihasilkan secara otomatis. Gambar 2.3 menunjukkan arsitektur umum dari pendekatan *generator-based reuse* (Sommerville, 2007, p.423).



Gambar 2.3 Arsitektur Generator-based Reuse

Sumber: (Sommerville, 2007, p.423)

Generator-based reuse memiliki keunggulan dalam menghasilkan sistem secara otomatis dalam domain yang telah ditentukan, misalnya sebuah sistem perangkat lunak skala memiliki arsitektur umum dan beberapa fungsi yang dapat disamakan. Sebagai contoh pada sistem pemrosesan data, pada umumnya memiliki 1) model sebagai unit masukan, proses dan keluaran, 2) operasi verifikasi data dan 3) pembuatan laporan. Oleh karena itu dengan bantuan pembuat aplikasi otomatis (*application generator*), sebuah komponen umum untuk melakukan operasi pada sistem pemrosesan data dapat dibuat dan digabungkan dengan sistem yang sudah ada. *Application generator* hanya membutuhkan masukan berupa: 1) item data yang akan diproses, 2) memilih data item yang akan diverifikasi, dan 3) format laporan yang akan dibuat (Sommerville, 2007, p.424).

Application domain knowledge digunakan sebagai basis pengetahuan oleh *application generator* sehingga dapat mengolah masukan yang diberikan oleh pengguna dan menghasilkan keluaran yang sesuai dengan domainnya.

Application description digunakan dalam *generator-based reuse* sebagai 1) cara untuk mendeskripsikan model abstraksi aplikasi yang akan dihasilkan, 2) komponen mana yang akan digunakan, dan 3) acuan dalam melakukan parameterisasi dan kombinasi antar model.

Generator-based reuse telah sukses untuk diterapkan khususnya pada aplikasi bisnis. *Application generator* dapat didesain untuk menghasilkan aplikasi secara lengkap atau dapat didesain untuk menghasilkan aplikasi secara sebagian dan memungkinkan pembuat aplikasi (*programmer*) untuk melengkapi detailnya (Sommerville, 2007, p.424).

2.8 Domain engineering

Domain adalah 1) bagian pengetahuan yang spesial, 2) area sebuah keahlian, 3) kumpulan dari fungsi – fungsi yang saling berhubungan (Harsu: 11). *Domain engineering* adalah sebuah pendekatan untuk melakukan penggunaan ulang perangkat lunak pada domain aplikasi tertentu dengan melakukan analisis domain dan memodelkan basis pengetahuan domain dalam bentuk aset perangkat lunak yang bisa digunakan kembali (*reusable asset*) (Cox & Lisa, 2010, p.15).

Proses *domain engineering* dibagi menjadi tiga bagian, yaitu: 1) *domain analysis*, 2) *domain design*, 3) *domain implementation*. Berikut ini penjelasan dari setiap proses *domain engineering*:

2.8.1 Domain Analysis

Domain analysis adalah suatu proses untuk mengidentifikasi, mengumpulkan, mengelompokkan dan menggambarkan informasi pada suatu domain berdasarkan studi pada sistem yang ada dan riwayat pengembangan sistem, basis pengetahuan disusun dari ahli domain, teori pokok dan teknologi pada domain. *Domain analysis* menghasilkan *domain knowledge* yang dapat digunakan untuk mengidentifikasi fitur – fitur sistem pada domain tersebut untuk mendukung penggunaan ulang komponen perangkat lunak. Hasil penggunaan ulang komponen perangkat lunak dapat digunakan untuk meningkatkan proses pengembangan dengan waktu yang lebih singkat dan biaya yang lebih sedikit (Cox & Lisa, 2010, p.16) (Harsu, 2015, p.9).

Ada beberapa pendekatan untuk melakukan *domain analysis*, yaitu: 1) berdasarkan proses *product line*, 2) berdasarkan rekayasa domain dan analisis desain berorientasi objek, 3) berdasarkan analisis tujuan perangkat lunak dan memodelkan domain. Pendekatan berbasis tujuan (*goals*) perangkat lunak dipilih oleh peneliti untuk melakukan analisis (Cox & Lisa, 2010, p.17).

Analisis domain berdasarkan tujuan perangkat lunak terdiri dari beberapa metode, yaitu *Feature-Oriented Domain Analysis* (FODA), *Feature Oriented Reuse Method* (FORM) dan *Feature Reuse-Driven Software Engineering Business* (FeatuRSEB). Metode FORM dan FeatuRSEB adalah perluasan dari metode FODA (Cox & Lisa, 2010, p.17).

2.8.2 Domain Design

Domain design adalah sebuah kegiatan untuk membuat desain arsitektur inti untuk sekumpulan aplikasi yang sejenis. Arsitektur inti harus memungkinkan untuk adanya variabilitas antar aplikasi. Berdasarkan *feature model* dan kesamaan yang telah dianalisis, arsitektur harus memungkinkan untuk memilih komponen mana yang akan dipakai pada arsitektur inti dan komponen mana yang dipakai sebagai variasi pada aplikasi individu (Harsu, 2015, p.13).

Domain design menghasilkan rencana produksi yang berguna untuk menjelaskan bagaimana sebuah aplikasi konkret dapat dihasilkan dari arsitektur inti. Rencana produksi terdiri dari deskripsi perilaku sistem terhadap pengguna, pedoman untuk merakit proses perakitan komponen dan pedoman untuk mengelola permintaan perubahan dari pelanggan. Proses perakitan tersebut dapat didesain secara manual, semi-otomatis atau otomatis (Harsu, 2015, p.13).

Domain design dapat dilibatkan untuk menilai arsitektur inti, misalnya menganalisis arsitektur terhadap persyaratan kualitas untuk menemukan resiko yang berpotensi terjadi pada arsitektur tersebut. Ada berbagai macam pendapat tentang waktu yang tepat untuk menilai sebuah arsitektur inti. Sebuah arsitektur

lebih baik dievaluasi sejak tahap awal proses pembuatan, karena semakin cepat permasalahan ditemukan maka semakin mudah pula permasalahan tersebut dapat diselesaikan. Arsitektur inti harus diperiksa untuk mengetahui seberapa baik arsitektur tersebut memenuhi persyaratan dari komponen yang digunakan ulang (Harsu, 2015, p.13).

2.8.3 Domain Implementation

Domain Implementation meliputi proses implementasi dari arsitektur inti, komponen dan alat yang telah didesain pada tahap sebelumnya. Sebagai contoh, tahap tersebut meliputi: 1) penulisan dokumentasi, 2) implementasi bahasa pemrograman spesifik pada aplikasi individu (*domain-specific language*), 3) implementasi aplikasi untuk menghasilkan kode atau aplikasi secara otomatis (*generator*) (Harsu, 2015, p.13).

Tujuan dari rekayasa domain (*domain engineering*) adalah untuk menghasilkan aset perangkat lunak yang bisa digunakan kembali yang mana akan diimplementasikan pada tahap *domain implementation*. Hasil dari keseluruhan proses rekayasa domain adalah: 1) komponen, 2) model fitur (*feature model*), 3) analisis dan desain model, 4) arsitektur inti, 5) pola perancangan, 6) bingkai kerja (*framework*), 7) *domain-specific language*, 8) rencana produksi dan 9) *generator* (Harsu, 2015, p.13).

Sebagai catatan, bahwa rekayasa domain adalah proses secara terus menerus (*continuous process*). Basis pengetahuan domain harus selalu dirawat dan diperbarui setiap waktu berdasarkan: 1) pengalaman baru, 2) perluasan cakupan, dan 3) tren baru. Sebagai tambahan, rekayasa domain harus bisa beradaptasi terhadap umpan balik dari proses rekayasa aplikasi (*application engineering*), yaitu proses untuk menghasilkan komponen perangkat lunak sebagai solusi untuk menyelesaikan masalah yang telah didefinisikan pada proses rekayasa domain. (Cox & Lisa, 2010, p.12). Proses penyusunan model domain tidak akan pernah selesai, model tersebut harus selalu ditingkatkan agar menjadi semakin akurat. Domain model terkadang mengandung beberapa macam perbedaan dan pandangan yang tidak konsisten dari beberapa ahli (Harsu, 2015, p.14).

2.9 Feature oriented domain analysis (FODA)

FODA adalah sebuah metode dalam melakukan *domain analysis*. Fokus utama dalam penyusunan FODA adalah identifikasi fitur – fitur yang menonjol dari perangkat lunak pada sebuah domain. Tujuan dari FODA adalah membuat sebuah produk agar bisa digunakan ulang (*reusable*). Fitur – fitur yang diidentifikasi adalah aspek yang terlihat oleh *end-user*. FODA mengidentifikasi faktor – faktor yang menyebabkan perbedaan antara aplikasi – aplikasi pada *domain* yang sama (Cox & Lisa, 2010, p.18).

Analisis FODA pada kesamaan dan perbedaan suatu aplikasi akan menghasilkan suatu istilah yang disebut dengan *mandatory*, *optional* dan *alternatie* pada suatu domain. *Domain analysis* pada FODA dibagi menjadi tiga

tahap, yaitu: 1) *context analysis*, 2) *domain modeling*, 3) *architecture modeling* (Cox & Lisa, 2010, p.18).

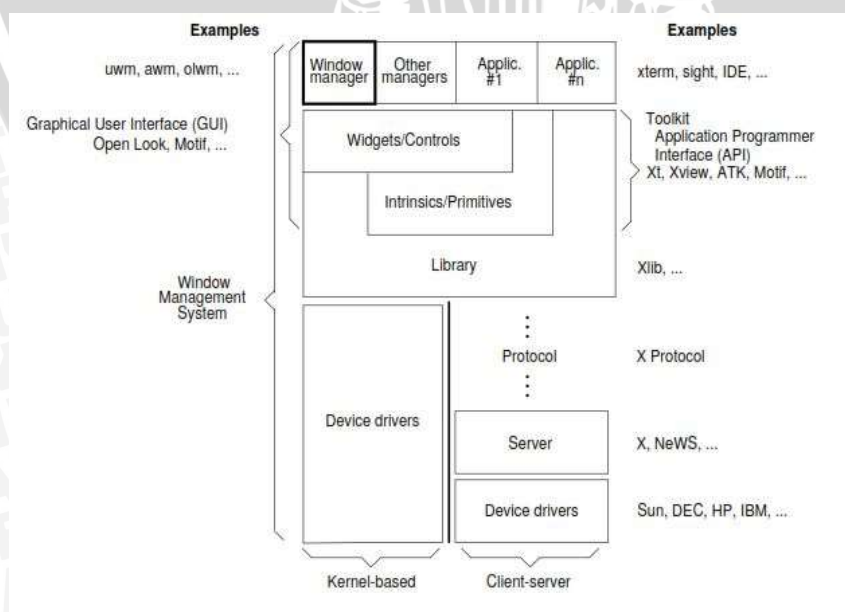
2.9.1 Context Analysis

Context analysis adalah suatu proses untuk mendefinisikan cakupan suatu domain dengan bantuan *domain knowledge expert* (Cox & Lisa, 2010, p.18).

Tujuan dari *context analysis* adalah untuk mendefinisikan cakupan suatu *domain* agar dapat menghasilkan sebuah produk domain yang bisa dimanfaatkan. Pada tahap *context analysis*, hubungan antara beberapa calon *domain* beserta elemen eksternalnya dianalisis, misalnya: perbedaan aplikasi yang menggunakan domain dan data yang sama, perbedaan sistem operasi yang digunakan dan lain - lain (Kang, et al., 1990, p.31).

Hasil dari pengolahan *context analysis* didokumentasikan ke dalam *context model*. *Context model* mendefinisikan batasan domain yang mana akan digunakan sebagai acuan dalam melakukan proses pemodelan domain (*domain modelling*) (Kang, et al., 1990, p.31).

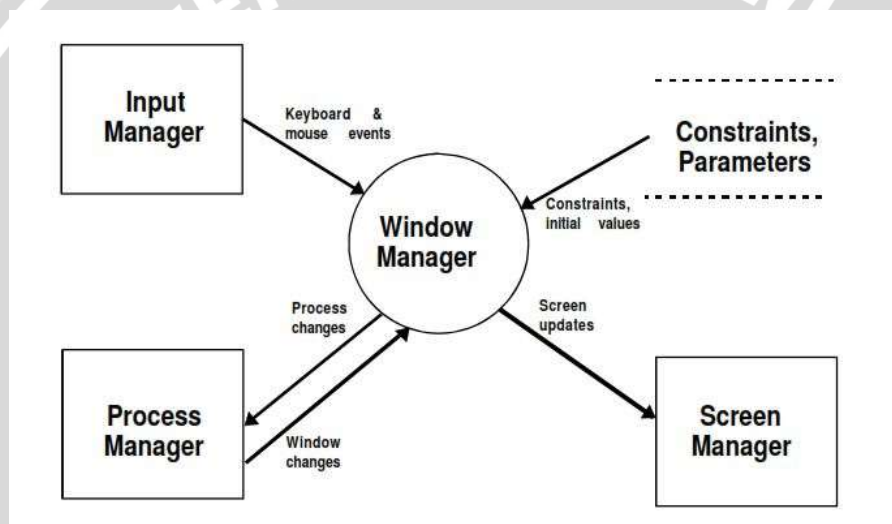
Context model terdiri satu atau lebih *structure diagram* dan *data-flow diagram*. Sebuah *structure diagram* adalah suatu diagram blok informal dimana domain yang menjadi target analisis ditempatkan dalam urutan posisi tertinggi, terendah atau pada level yang sama. Tingkatan domain yang lebih tinggi adalah objek yang merupakan bagian dari domain target. Jika tidak ada tingkatan domain yang lebih tinggi, maka tipe aplikasi yang ada pada domain harus diidentifikasi. Tingkatan domain yang lebih rendah (misalnya sub-domain) adalah objek yang berada didalam cakupan target domain. Jika ada domain lain yang berinteraksi dengan domain target (misalnya *peer-level domain*) juga harus dilibat kedalam diagram. Lebih dari satu diagram stuktur dapat digunakan jika dibutuhkan. Gambar 2.4 menunjukkan contoh diagram stuktur (Kang, et al., 1990, p.31).



Gambar 2.4 Contoh Structure Diagram

Sumber: (Kang, et al., 1990, p.60)

Data-flow diagram menunjukkan aliran data antara suatu domain target dan juga aliran data pada *entity* yang dipakai oleh sebuah domain untuk berkomunikasi. Gambar 2.5 menunjukkan contoh *data-flow diagram* pada *domain analysis*. Perbedaan penggunaan *data-flow diagram* pada proses *domain analysis* adalah kesamaan aliran data pada cakupan *domain* harus ditunjukkan dalam diagram. Hal tersebut dapat digambarkan dalam sekumpulan diagram, dimana setiap diagram menunjukkan konteks yang berbeda, atau bisa digambarkan dengan satu diagram yang dapat menjelaskan perbedaan antara satu konteks dengan lainnya. Jika ada perbedaan fitur antar *domain* maka harus dideskripsikan. Deskripsi tekstual tentang fungsionalitas dan usabilitas suatu *domain* dan tujuan dari *domain analysis* harus dimasukkan ke dalam *context model*.



Gambar 2.5 Contoh Context Diagram: Data-flow Utama pada Domain Analysis

Sumber: (Kang, et al., 1990, p.61)

Kondisi yang harus dipenuhi dalam membangun *reusable software* adalah pemahaman pada beberapa konteks dimana *reusable software* akan dioperasikan. Pemahaman tentang: 1) tingkat perbedaan kontekstual, 2) kapan dan bagaimana perubahan konteks terjadi, dapat digunakan untuk membantu pengembang aplikasi untuk menentukan apakah *reusable software* dapat dikembangkan. Jika memungkinkan, pemahaman tentang *context model* dapat membantu: 1) dalam menentukan hal apa saja yang bisa dijadikan sebagai parameter, 2) bagaimana mendesain struktur perangkat lunak agar bisa beradaptasi terhadap konteks yang berbeda – beda sesuai persyaratan (Kang, et al., 1990, p.32).

2.9.2 Domain Modeling

Perbedaan dan kesamaan permasalahan aplikasi pada domain dianalisis pada tahap pemodelan domain (*domain modelling*), sejumlah model yang menggambarkan aspek yang berbeda dihasilkan pada tahap tersebut. Pemodelan

domain terdiri dari tiga aktivitas utama, yaitu: 1) *feature analysis*, 2) *entity—relational modeling*, dan 3) *function analysis* (Cox & Lisa, 2010, p.18) (Kang, et al., 1990, p.35).

2.9.2.1 Feature Analysis

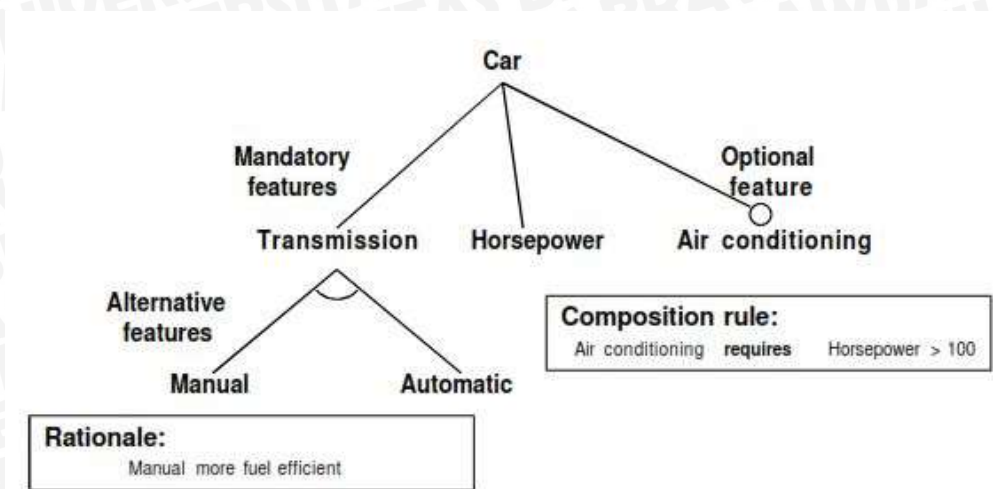
Tujuan dari *feature analysis* adalah untuk menangkap atau memodelkan pemahaman pelanggan (*customer*) tentang kemampuan aplikasi dalam suatu domain. Pada umumnya *data-flow diagram* digunakan untuk menggambarkan persyaratan *customer*. Pada *data-flow diagram* terdapat penjelasan fitur aplikasi untuk memuaskan persyaratan pelanggan, namun informasi tersebut sering tidak menarik perhatian *customer*. *Customer* perlu tahu detail lengkap kemampuan aplikasi yang diinginkan mereka. Detail kemampuan tersebut adalah:

1. Layanan yang disediakan oleh aplikasi
2. *Performance* aplikasi
3. Perangkat keras yang dibutuhkan oleh aplikasi
4. Biaya (Kang, et al., 1990, p.35)

Feature analysis berfokus pada analisis fungsionalitas aplikasi dari perspektif *customer*, fungsionalitas tersebut meliputi: 1) layanan yang disediakan oleh aplikasi, 2) lingkungan (*environment*) dimana aplikasi tersebut berjalan (Kang, et al., 1990, p.35).

Sejak awal proses *domain analysis*, fokus utama adalah analisis kesamaan dari sekumpulan aplikasi. Maka *feature model* harus memodelkan: 1) fitur umum dari sekumpulan aplikasi, 2) perbedaan pada setiap aplikasi dalam suatu *domain*. Analisis fitur pada *feature model* digunakan untuk menyamakan (*generalize*) dan membuat parameter pada model lainnya (Kang, et al., 1990, p.35).

Fitur adalah atribut sistem yang secara langsung berpengaruh pada pengguna. Pengguna harus membuat keputusan mengenai ketersediaan fitur didalam sistem. Gambar 2.6 adalah contoh model untuk mendeskripsikan kumpulan fitur mobil. Ketika pelanggan ingin membeli mobil, sebuah keputusan harus dipilih terkait 1) transisi mobil mana yang dipilih, 2) fitur tambahan apa yang diinginkan, 3) apakah mungkin membeli sebuah mobil dengan dua tipe transisi: otomatis dan manual.



Gambar 2.6 Feature Diagram Mobil

Sumber: (Kang, et al., 1990, p.36)

Feature model menggambarkan fitur standar dari sekumpulan sistem pada suatu *domain* dan hubungan diantara mereka. Struktur hubungan tersebut berisi representasi sebuah pengelompokan logikal dari sekumpulan fitur. Fitur tambahan dan alternatif dari setiap pengelompokan harus ditunjukkan pada *feature model*. Sebagai contoh yang ditunjukkan pada gambar 2.6, fitur transisi otomatis dan manual adalah fitur alternatif (tanda anak panah), fitur *air-conditioning* adalah fitur tambahan (tanda lingkaran). Setiap fitur harus diberi penamaan yang unik (Kang, et al., 1990, p.36).

Fitur alternative dapat dianalogikan sebagai spesialisasi dari suatu kategori umum. Sebagai contoh fitur transisi manual dan otomatis adalah spesialisasi dari fitur “transisi” umum. Fitur alternative menunjukkan bahwa hanya satu tipe spesialisasi saja yang mungkin diterapkan. Atribut fitur umum akan mewarisi fitur spesialisasi yang dipilih (Kang, et al., 1990, p.36).

Feature model digunakan sebagai media komunikasi antara *customer* dengan *developer*. *Feature model* menunjukkan kepada *customer*: 1) fitur standar sebuah aplikasi, 2) fitur lain yang bisa mereka pilih, 3) dan kapan *customer* bisa memilih fitur lainnya. *Feature model* menunjukkan kepada *developer*: 1) hal apa saja yang butuh diparameterisasi pada model lainnya dan arsitektur *software*, 2) dan bagaimana proses parameterisasi harus dilakukan (Kang, et al., 1990, p.37).

2.9.2.2 Entity-Relationship Modeling

Entity-relationship model berfungsi untuk memodelkan dan mendefinisikan *domain knowledge* yang mana memegang peranan penting dalam mengimplementasi aplikasi terhadap *domain* tersebut. *Domain knowledge* merupakan deskripsi kontekstual yang akan hilang setelah tahap pengembangan, atau tertanam dalam perangkat lunak dan sulit untuk ditelusuri. Bagi *developer* yang ingin memodifikasi atau menggunakan ulang *software* harus memperoleh informasi tentang domain tersebut (*domain knowledge*) untuk memahami

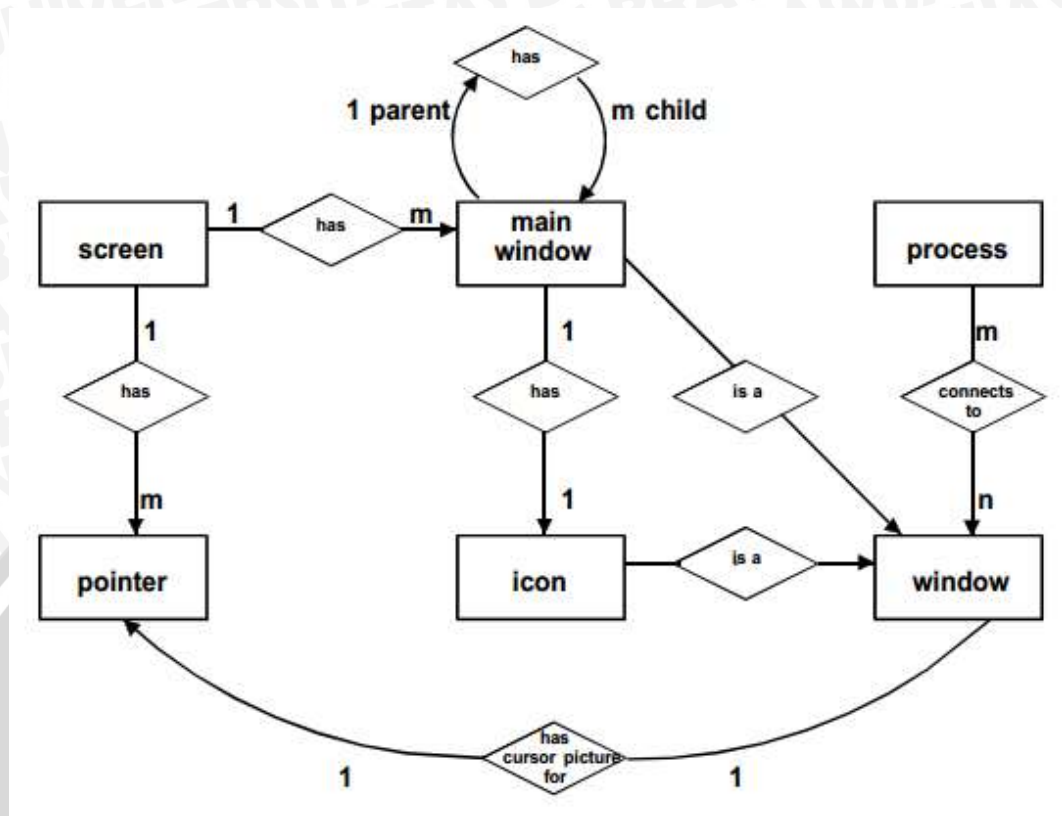
permasalahan yang ada pada aplikasi tersebut. Oleh karena itu tujuan dari *entity-relationship modelling* adalah merepresentasikan *domain knowledge* secara eksplisit berupa *domain entities* dan relasinya, dan memungkinkan untuk penurunan objek dan definisi data selama tahap *functional analysis* dan *architecture modelling* (Kang, et al., 1990, p.40).

Teknik pemodelan *entity-relationship* pada metode FODA diadopsi dengan konsep *generalization* dan *aggregation*. Oleh karena itu, untuk pembentuk dasar pada model adalah kelas *entity*, hubungan *is-a* dan *consists-of*. **Kelas *entity*** menggambarkan abstraksi objek pada domain, hubungan *is-a* dan *consists-of* menggambarkan hubungan *generalization* dan *aggregation*. Hubungan *aggregation* menentukan stuktur komposisi antar *entity*, sedangkan hubungan *generalization* menentukan persamaan dan perbedaan antar *entity* (Kang, et al., 1990, p.40).

Satu atau lebih atribut mungkin didefinisikan untuk sebuah *entity*. Atribut juga didefinisikan secara terpisah dari *entity* dan mungkin digunakan pada definisi *entity* (Kang, et al., 1990, p.40).

Kegunaan utama dari model *entity-relationship* adalah untuk mendukung analisis dan memahami permasalahan domain dan untuk menghasilkan stuktur objek domain yang digunakan pada proses pengembangan aplikasi. *entity* adalah unit informasi domain yang harus diproses dan/atau dirawat oleh sistem.

Hubungan *consists-of* dan *is-a* pada umumnya digunakan untuk mengidentifikasi komposisi, persamaan dan perbedaan objek domain, yang mana akan memungkinkan pengembangan produk domain secara umum dan dapat diparameterisasi. Sebagai contoh, struktur pewarisan (*inheritance*) berasal dari hubungan *is-a* pada pengembangan berbasis objek. Kumpulan hubungan *consists-of* pada sejumlah *entity* menunjukkan bahwa *entity* tersebut saling terpadu (*integrated*). Informasi tersebut dapat digunakan untuk mengidentifikasi konsistensi, aturan integritas, dan menghasilkan definisi penyimpanan data pada model fungsional (*functional model*) (Kang, et al., 1990, p.41). Gambar 2.7 menunjukkan contoh model *entity-relationship* pada metode FODA.



Gambar 2.7 Entity-relationship model pada FODA

Sumber: (Kang, et al., 1990, p.62)

2.9.2.3 Functional Analysis

Functional analysis digunakan untuk mengidentifikasi kesamaan dan perbedaan sekumpulan aplikasi dalam suatu *domain*. Tahap ini mengabstraksikan dan menstrukturkan fungsi umum kedalam sebuah model dari sekumpulan fungsi spesifik aplikasi sehingga bisa dihasilkan fungsi yang bisa beradaptasi dengan tepat (Kang, et al., 1990, p.42).

Feature model dan *entity-relationship model* digunakan sebagai pedoman dalam membuat *functional model*. *Mandatory feature* dan *entity* adalah dasar dalam mendefinisikan model abstrak fungsional. Fitur alternatif dan tambahan tertanam ke dalam model selama proses penyempurnaan. Faktor lain yang menyebabkan perbedaan antar aplikasi akan didefinisikan sebagai persoalan (*issue*) dan keputusan (*decision*), yang mana akan digunakan sebagai penyempurnaan untuk proses parameterisasi (Kang, et al., 1990, p.42).

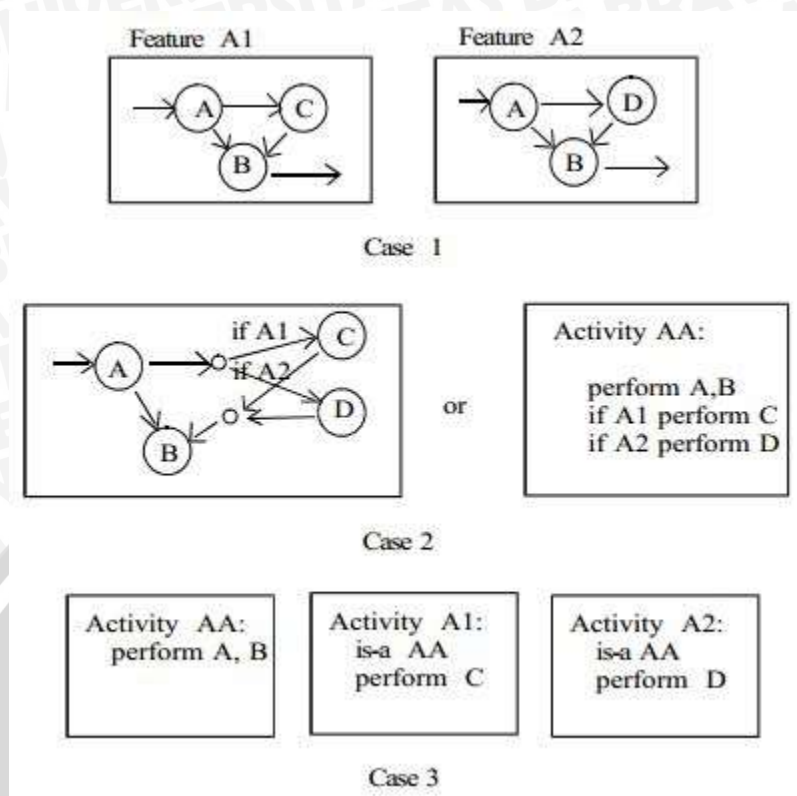
Spesifikasi *functional model* dapat diklasifikasikan menjadi dua kategori utama, yaitu: 1) spesifikasi fungsi, 2) dan spesifikasi tingkah laku (*behavior*). Spesifikasi fungsi mendeskripsikan aspek struktural dari sebuah aplikasi dalam hal: masukan, keluaran, aktivitas, data internal, struktur logika, dan hubungan *data-flow* diantara komponen tersebut. Spesifikasi tingkah laku mendeskripsikan bagaimana sebuah aplikasi merespon dalam hal: *event* (kejadian), masukan, *states*

(keadaan), *condition* (percabangan), dan *state transision* (transisi keadaan). Bagan aktivitas (*activitycharts*) menggambarkan aspek fungsional dan bagan keadaan (*statecharts*) menggambarkan aspek perilaku sistem (Kang, et al., 1990, p.42).

Sebuah model abstrak fungsional dari domain aplikasi didefinisikan pada tingkatan atas. Selama proses penyempurnaan model, memungkinkan adanya kasus dimana sebuah *entity* model dapat disempurnakan lebih dari satu kali. Misalnya, pada sebuah kasus dimana ada lebih dari satu teknologi domain yang dipilih, proses pemilihan teknologi tertentu dapat mengakibatkan spesifikasi fungsional yang berbeda dari yang lainnya (cakupan aplikasi lain dalam satu domain). Seorang analis harus memahami rincian dari teknologi yang dipilih dan membuat keputusan terbaik untuk domain aplikasi. Keputusan yang diambil pada setiap permasalahan dideskripsikan sebagai *issue* dan *decision* (Kang, et al., 1990, p.42).

Salah satu hal yang membedakan metode FODA dengan metode analisis domain yang lain adalah proses parameterisasi pada penggunaan fitur dan *issue/decision*. Sebagai sebuah model abstrak, fitur alternatif dan opsional tertanam kedalam model. Ada beberapa bermasalah yang muncul selama proses analisis dan resolusi *issue* dan proses penggabungannya dalam sebuah model. Secara umum, ada tiga cara untuk menggabungkan fitur dan *issue/decision* (ditunjukkan pada gambar 2.8), yaitu:

1. **Case 1:** dengan membuat komponen secara terpisah untuk setiap alternatif.
2. **Case 2:** membuat satu komponen, dengan parametisasi untuk mengadaptasi setiap alternatif.
3. **Case 3:** dengan mendefinisikan sebuah komponen umum dan membuat setiap alternatif sebagai turunan dari komponen umum. ((Kang, et al., 1990, p.43)



Gambar 2.8 Parameterisasi pada *Functional Analysis*

Sumber: (Kang, et al., 1990, p.43)

Kegunaan utama dari model fungsional (*functional model*) adalah 1) memahami premasalahan domain, 2) menggunakan ulang model pada proses analisis persyaratan (*requirement analysis*). Model fungsional menggambarkan fungsionalitas aplikasi dari tingkat abstrak kedalam tingkatan yang lebih detail. Struktur dekomposisi dan alasan-alasan yang berhubungan dengan dekomposisi akan membantu analis memahami permasalahan domain. Analis juga dapat menggunakan ulang model pada suatu tingkatan yang merupakan pilihan paling tepat untuk aplikasi individu yang akan dikembangkan (Kang, et al., 1990, p.44).

2.9.3 Architecture Model

Tujuan dari *architecture modelling* adalah untuk memberikan solusi perangkat lunak terhadap permasalahan yang didefinisikan pada tahap *domain modelling*. Sebuah *architecture model* akan dibuat pada tahap ini, model tersebut memberikan gambaran detail desain dan konstruksi komponen (Kang, et al., 1990, p.47).

Tujuan utama dari metode FODA adalah untuk membuat produk domain yang bisa digunakan ulang. Pada proses pengembangan model arsitektur (*architecture model*), proses *architecture layering* dilakukan agar proses penggunaan ulang dapat dilakukan pada lapisan yang tepat untuk aplikasi tertentu dan dampak dari perubahan teknis dan perubahan persyaratan model dapat ditentukan (Kang, et al., 1990, p.47).

Model arsitektur pada FODA merupakan desain tingkat tinggi dari sebuah aplikasi pada suatu domain. Oleh karena itu metode FODA fokus pada identifikasi proses yang berjalan secara bersamaan, penempatan fitur, fungsi dan objek data yang telah didefinisikan pada model domain kedalam sebuah proses dan modul (Kang, et al., 1990, p.47).

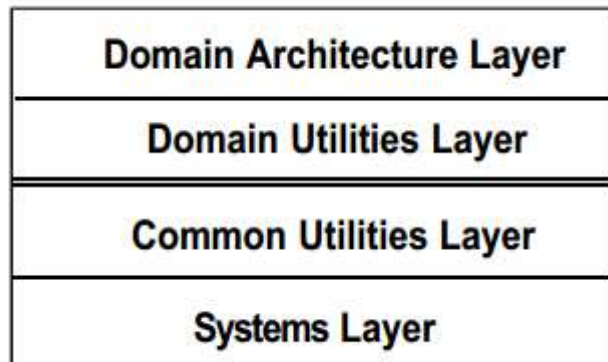
Sebuah model arsitektur harus mampu mengatasi masalah yang didefinisikan pada domain sehingga model tersebut dapat beradaptasi terhadap perubahan teknologi di masa depan. Adaptasi yang harus dicapai dari model arsitektur tersebut adalah:

1. arsitektur didefinisikan pada beberapa tingkatan yang berbeda sehingga proses penggunaan ulang dapat dilakukan pada tingkatan yang tepat untuk aplikasi individu yang akan dibuat.
2. pemaketan fungsi dan objek domain dilakukan secara terpisah dengan proses pemaketan pada teknik implementasi sehingga:
 - Pengambilan keputusan implementasi bisa dipisahkan dari pemaketan fungsionalitas domain
 - Meningkatkan tingkat penggunaan ulang modul
 - Dampak ke seluruh sistem yang timbul akibat perubahan teknik implementasi dapat ditangani (Kang, et al., 1990, p.47)

Banyak keputusan yang dibuat selama proses desain perangkat lunak. Sebuah aplikasi didekomposisi kedalam sekumpulan program yang dapat di-*compile* secara terpisah dan dieksekusi secara paralel. Setiap proses harus didesain sebagai sebuah modul hirarki dengan alokasi fungsi dan objek data yang telah didefinisikan pada model alir data (*data-flow model*). Modul umum berbasis domain yang bisa digunakan diseluruh aplikasi harus diidentifikasi untuk meningkatkan penggunaan ulang. Keputusan implementasi harus dibuat pada berbagai macam teknik implementasi seperti: mekanisme komunikasi dan sinkronisasi, metode proses penjadwalan, sistem manajemen basis data dan bahasa pemrograman yang pilih (Kang, et al., 1990, p.47-48).

Arsitektur sistem berlapis didefinisikan oleh Gambar 2.9. Arsitektur sistem berlapis tersebut disusun berdasarkan tipe keputusan desain yang dibahas pada paragraf sebelumnya dan berdasarkan urutan umum dalam membuat keputusan desain tersebut. Lapisan paling atas adalah **domain architecture layer**, yang mewakili sebuah model yang menunjukkan proses domain secara bersamaan dan hubungan diantara proses tersebut, model tersebut disebut sebagai *process interaction model* pada FODA. Lapisan **domain utilities layer** menunjukkan proses pemaketan fungsi dan objek data kedalam modul dan hubungan antar modul, model tersebut disebut sebagai *module structure chart* yang digambarkan dengan bagan struktur (*structure chart*). Lapisan **common utilities layer** mengandung modul yang bisa digunakan di domain yang berbeda, yaitu modul yang digunakan untuk komunikasi dan sinkronisasi antar proses (contoh:

implementasi antrian pesan dan implementasi penanganan kejadian/*event handler*). Lapisan **system layer** berisi utilitas yang disediakan oleh sistem operasi dan bahasa pemrograman (Kang, et al., 1990, p.48).



Gambar 2.9 Lapisan Arsitektur
(Kang, et al., 1990, p.48)

Metodologi FODA fokus kepada dua lapisan arsitektur teratas, yaitu pada proses pengembangan sebuah arsitektur berbasis domain. Hal tersebut merupakan desain tingkat tinggi dimana proses pemaketan fungsi dan objek menjadi modul perangkat lunak merupakan tujuan utama (Kang, et al., 1990, p.48).

2.10 Pengujian perangkat lunak

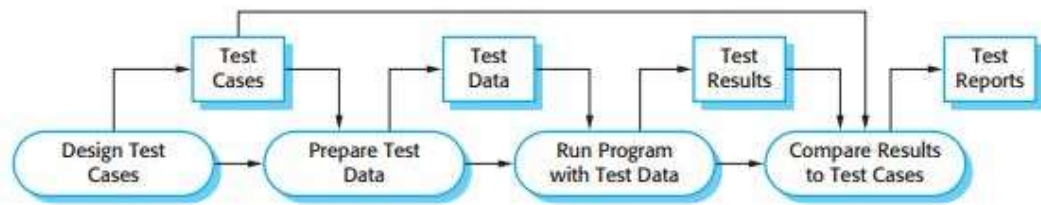
Pengujian perangkat lunak bertujuan untuk menunjukkan bahwa program berjalan sesuai dengan yang dimaksudkan dan untuk menemukan kekurangan dari perangkat lunak tersebut sebelum digunakan. Ketika menguji perangkat lunak, bisa dilakukan dengan mengeksekusi program dengan data tiruan. Kemudian memeriksa hasilnya untuk menemukan 1) *error*, 2) kegagalan, 3) dan informasi terkait atribut non-fungsional program (Sommerville, 2011, p.206).

Proses pengujian perangkat lunak memiliki dua tujuan yang berbeda, yaitu:

1. Untuk mendemonstrasikan kepada *developer* dan *customer* bahwa perangkat lunak telah memenuhi persyaratan. Pengujian harus dilakukan untuk setiap fitur dan kombinasi antar fitur, dan sistem harus berjalan dengan benar sesuai dengan *test case* yang diujikan. (*validation testing*)
2. Untuk menemukan situasi dimana perilaku *software* salah, tidak sesuai yang diinginkan atau tidak sesuai dengan spesifikasi. *Defect testing* bertujuan untuk menghilangkan perilaku sistem yang tidak menyimpang seperti: *system crashes*, interaksi yang tidak diinginkan dengan sistem lain, kesalahan perhitungan dan data yang korup (Sommerville, 2011, p.206).

Gambar 2.10 menjelaskan tentang model proses pengujian. *Test case* berisi spesifikasi masukan yang akan diuji, hasil keluar yang sesuai dengan harapan dan pernyataan tentang apa yang sedang diuji. *Test data* adalah masukan yang telah

dirancang untuk digunakan dalam pengujian. *Test data* terkadang bisa dihasilkan secara otomatis. *Test result* adalah hasil dari proses pengujian, yang akan dibandingkan dengan *test case* untuk mengetahui kesesuaian perangkat lunak (Sommerville, 2011, p.209).



Gambar 2.10 Model Proses Pengujian Perangkat Lunak

Sumber: (Sommerville, 2011, p.210)

Biasanya, sistem perangkat lunak komersial akan menjalani tiga tahap pengujian, yaitu:

1. *Development testing*: dimana sistem diuji selama proses pengembangan untuk menemukan *bugs* dan *defect* (kecacatan sistem). *System designer* dan *programmer* mungkin dapat dilibatkan pada proses ini.
2. *Release testing*: dimana tim penguji yang terpisah akan menguji versi lengkap dari sistem sebelum sistem digunakan oleh pengguna. Tujuannya adalah untuk memastikan bahwa sistem memenuhi persyaratan *stakeholder*.
3. *User testing*: dimana pengguna sistem menguji sistem tersebut pada lingkungan mereka. *Acceptance testing* merupakan salah satu tipe dari *user testing* dimana pengguna menguji sistem untuk memutuskan apakah sistem dapat diterima oleh pengguna atau perlu dikembangkan lebih lanjut (Sommerville, 2011, p.210).

2.10.1 Pemilihan Unit *Test Case*

Pengujian perangkat lunak adalah proses yang mahal dan memakan banyak waktu. Maka dari itu *test case* harus didesain dengan efektif dengan kriteria berikut ini:

1. *Test case* harus bisa menunjukkan bahwa, bila digunakan seperti yang diharapkan, komponen yang diuji akan melakukan apa yang seharusnya.
2. Jika ada *defect* pada suatu komponen, maka hal tersebut diungkapkan oleh *test case* (Sommerville, 2011, p.213).

Oleh karena itu ketika mendesain dua tipe *test case*. Tipe pertama adalah desain *test case* yang menunjukkan bahwa sistem akan bekerja dengan benar pada pemakaian yang normal. Tipe kedua adalah *degain test case* yang menguji perilaku sistem ketika menerima masukan yang tidak normal untuk memastikan masukan tersebut akan diproses dengan tepat dan tidak menyebabkan komponen mengalami malfungsi (Sommerville, 2011, p.213).

2.11 Development Testing

Development testing pada umum dilakukan oleh tim pengembang sistem. Orang yang melakukan pengujian ini biasanya adalah *programmer* dari sistem tersebut. Selama proses pengembangan, pengujian dapat dilakukan kedalam tiga level, yaitu:

1. *Unit testing*: merupakan proses pengujian untuk unit program atau kelas objek. Pengujian ini fokus dalam menguji fungsionalitas suatu objek atau *method*.
2. *Componen testing*: dimana beberapa unit program diintegrasikan untuk membuat komponen gabungan. Pengujian ini fokus dalam menguji *component interface* suatu sistem.
3. *System testing*: dimana beberapa atau seluruh komponen sistem diintegrasikan dan sistem diuji secara keseluruhan. *System testing* fokus dalam menguji integrasi antar komponen.

*Development testing*intinya merupakan proses *defect testing*, tujuannya adalah untuk menemukan *bugs* pada perangkat lunak yang harus dibenahi (Sommerville, 2011, p.210-211).

2.11.1 Unit Testing

Unit testing merupakan proses untuk menguji komponen program seperti *method* (fungsi individual) atau *object*. *Method* adalah tipe komponen yang paling sederhana. Pengujian dilakukan untuk menjalankan komponen tersebut dengan parameter masukan yang berbeda – beda.

Ketika proses pengujian kelas objek, desain pengujian harus mencakup semua fitur objek. Hal yang harus dilakukan adalah:

1. Menguji semua operasi yang berkaitan dengan kelas tersebut.
2. Menguji atribut kelas dengan melakukan operasi *set* dan melihat hasilnya
3. Menguji objek kedalam kondisi – kondisi yang memungkinkan. Semua kejadian (*event*) yang menyebabkan perubahan kondisi objek harus disimulasikan (Sommerville, 2011, p.211).

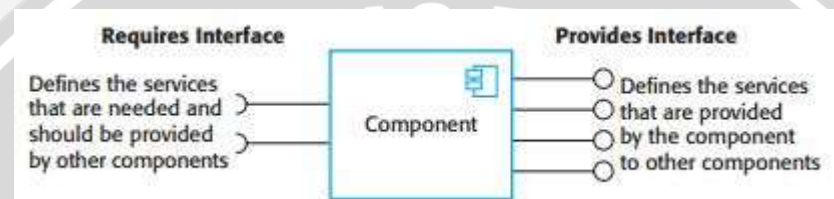
Generalization atau *inheritance* membuat pengujian suatu kelas objek menjadi rumit. Karena pengujian juga harus dilakukan terhadap operasi yang mewarisi kelas tersebut dalam semua konteks dimana kelas tersebut digunakan (Sommerville, 2011, p.212).

Terkadang objek yang sedang diuji memiliki ketergantungan dengan objek lain yang 1) tidak dikembangkan sendiri oleh tim, 2) atau akan memperlamban proses pengujian jika digunakan. Misalnya jika menguji suatu objek yang membutuhkan pemanggilan *database*, hal tersebut mungkin akan membuat proses eksekusi objek menjadi lebih lambat karena harus memanggil objek *database*. *Mock object* dapat digunakan untuk mengatasi masalah tersebut, *mock object* adalah objek

yang memiliki *interface* yang sama dengan eksternal objek yang digunakan. Namun *mock object* berisi data yang dimanipulasi agar bisa diakses dengan cepat oleh objek lain (Sommerville, 2011, p.213).

2.11.2 Component Testing

Komponen perangkat lunak merupakan komponen gabungan yang dibuat dari beberapa objek yang saling berinteraksi (Sommerville 9th, 2011: 216). Seperti yang ditunjukkan oleh gambar 2.10, sebuah komponen memiliki dua relasi *interface*. *Interface* tersebut menggambarkan layanan yang disediakan oleh komponen dan layanan yang dibutuhkan oleh komponen untuk beroperasi dengan benar (Sommerville 9th, 2011: 457).



Gambar 2.11 Component Interface

Sumber: (Sommerville, 2011, p.457)

Pengujian komponen harus fokus untuk memastikan bahwa *component interface* berperilaku sesuai dengan ketentuan. Apda pengujian komponen dapat diasumsikan bahwa setiap unit komponen telah diuji dengan benar pada *unit testing* (Sommerville, 2011, p.216).

Gambar 2.11 menunjukkan sebuah model pengujian *component interface*. Diasumsikan bahwa komponen A, B dan C telah terintegrasi untuk membangun sebuah komponen yang lebih besar atau sub-sistem. *Test case* tidak diterapkan untuk menguji setiap unit komponen, melainkan untuk menguji *interface* pada komponen gabungan tersebut. Jika terjadi *error* pada pengujian, mungkin tidak bisa dideteksi dengan menguji setiap unit komponen, karena *error* tersebut terjadi karena interaksi antar komponen (Sommerville, 2011, p.216).

Ada beberapa tipe *error* yang mungkin terjadi pada suatu *interface*, yaitu:

1. *Interface misuse*: sebuah kejadian dimana komponen yang sedang dipanggil memanggil komponen lain dan menyebabkan *error* pada penggunaan *interface* tersebut. Tipe *error* seperti ini dikarenakan kesalahan dalam mengirim *parameter*.
2. *Interface misunderstanding*: kejadian dimana komponen yang dipanggil salah paham terhadap spesifikasi yang diminta oleh komponen pemanggil. Sehingga komponen yang dipanggil tidak berjalan sesuai harapan.
3. *Timing errors*: kesalahan ini terjadi pada sistem *real-time* yang menggunakan *memory* secara bersama atau dapat terjadi pada *message-passing interface*. Produsen data dan konsumen data mungkin beroperasi dengan kecepatan yang berbeda. Sehingga komsumen akan mendapatkan informasi yang tidak

akurat karena produsen data belum memperbarui informasi dalam waktu tersebut (Sommerville, 2011, p.217-218).

Pedoman umum dalam melakukan *component testing* adalah sebagai berikut ini:

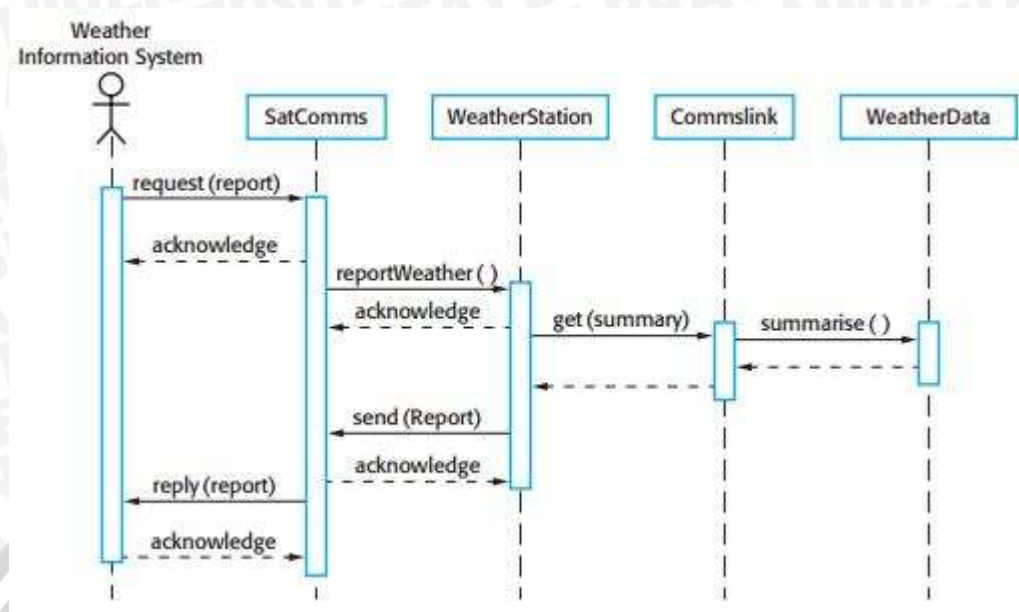
1. Memeriksa kode program yang akan diuji dan mencatat semua pemanggilan komponen eksternal. Mendesain kasus uji dengan menggunakan nilai parameter yang ekstrim untuk dikirimkan ke eksternal komponen. Nilai tersebut digunakan untuk menguji konsistensi suatu *interface*.
2. Menguji suatu *interface* dengan nilai *null* pada parameter yang dikirimkan.
3. Ketika komponen dipanggil melalui *interface*, desain kasus uji dengan membuat suatu komponen tersebut gagal dipanggil. Sehingga jika terjadi *interface misunderstanding* dapat terdeteksi.
4. Menggunakan *stress testing* pada sistem pengiriman pesan. Pengujian tersebut dilakukan dengan membuat pesan yang dikirim dalam jumlah yang banyak daripada jumlah yang mungkin terjadi dalam praktik. Pengujian tersebut efektif dalam menguji adanya *timing problem* atau tidak.
5. Ketika beberapa komponen berinteraksi melalui sebuah *memory* secara bersamaan, desain kasus uji untuk menguji urutan komponen yang diaktifkan dalam urutan yang berbeda – beda. Pengujian tersebut dapat digunakan untuk menguji asumsi *programmer* tentang urutan diaman data yang akan dipakai bersama – sama akan diproduksi dan dikonsumsi (Sommerville, 2011, p.218).

2.11.3 System Testing

System testing dilakukan selama proses pengembangan aplikasi, untuk menguji bahwa semua komponen yang telah dibuat dapat saling berintegrasi dengan baik. *System testing* memeriksa apakah komponen – komponen tersebut cocok, dapat berinteraksi dengan benar dan dapat mengirimkan data dengan benar pada waktu yang tepat melalui *interface* masing – masing (Sommerville, 2011, p.219).

System testing fokus pada pengujian interaksi antar komponen yang akan membentuk sebuah sistem. Seluruh sistem utuh akan diuji untuk memastikan bahwa mereka bekerja seperti yang diharapkan. Pengujian dapat digunakan untuk mendeteksi *bugs* yang terjadi ketika suatu komponen digunakan oleh komponen lain didalam sistem (Sommerville, 2011, p.219).

Karena *system testing* fokus terhadap interaksi antar komponen, maka pengujian berbasis *use case* merupakan pendekatan yang cocok untuk menguji sistem. Biasanya setiap *use case* diimplementasikan oleh beberapa komponen. Dengan menguji *use case*, maka mekanisme interaksi antar komponen dapat terjadi. Jika *sequence diagram* telah dibuat, maka akan mempermudah pengujian karena dapat melihat komponen atau kelas yang terlibat dalam interaksi tersebut (Sommerville, 2011, p.220).



Gambar 2.12 Sequence Diagram

Sumber: (Sommerville, 2011, p.220)

Untuk mengilustrasikan *system testing*, dapat dibantu dengan gambar 2.12 yang menunjukkan sebuah operasi *sequence diagram*. Diagram tersebut menunjukkan urutan proses sistem stasiun cuaca dalam merespon permintaan data. Diagram tersebut dapat digunakan untuk membantu penyusunan *test case*. *Sequence diagram* dapat digunakan untuk mendesain *test case* tentang masukan yang dibutuhkan dan hasil keluaran yang diinginkan (Sommerville, 2011, p.220).

Pada kebanyakan sistem, sangat sulit untuk menentukan jumlah pengujian sistem yang harus dilakukan dan kapan harus berhenti. Pengujian secara lengkap, dimana setiap urutan eksekusi program yang mungkin harus diuji, adalah hal yang mustahil. Oleh karena itu pengujian sistem dilakukan berdasarkan sebagian *test case* yang mungkin dapat dilakukan. Pada umumnya, perusahaan perangkat lunak harus memiliki ketentuan dalam memilih bagian *test case* yang dapat diujikan. Ketentuan tersebut dapat dibuat berdasarkan ketentuan umum pengujian, seperti berikut ini:

1. Semua baris program harus dieksekusi minimal satu kali.
2. Semua fungsionalitas sistem yang diakses melalui daftar opsi harus diujikan.
3. Kombinasi antar fungsi (misal: *text formatting*) yang diakses melalui opsi yang sama harus diujikan.
4. Ketika sistem memungkinkan adanya masukan dari pengguna, maka semua fungsi harus diuji dengan menggunakan masukan yang benar dan masukan yang salah (Sommerville, 2011, p.221).

2.12 Release Testing

Release testing adalah proses pengujian versi rilis tertentu dari suatu sistem yang akan digunakan diluar lingkungan pengembangan. Pada umumnya sistem akan dirilis untuk memenuhi persyaratan pengguna atau pelanggan. Pada sebuah proyek yang kompleks, versi rilis sistem dapat dilakukan oleh tim pengembang lain yang berhubungan dengan sistem tersebut (Sommerville, 2011, p.224).

Ada dua hal yang membedakan antara *release testing* dengan *system testing*, yaitu:

1. Tim terpisah yang tidak dilibatkan dalam proses pengembangan sistem harus bertanggungjawab terhadap *release testing*.
2. *System testing* dilakukan oleh tim pengembang fokus dalam mencari *bugs* pada sistem (*defect testing*). *Release testing* bertujuan untuk memeriksa bahwa sistem memenuhi persyaratan untuk digunakan pada lingkungan luar (*validation testing*) (Sommerville, 2011, p.224).

Tujuan utama dari *release testing* adalah untuk memastikan kepada pengguna bahwa sistem layak untuk digunakan. *Release testing* harus menunjukkan bahwa sistem dapat bekerja dengan baik sesuai dengan fungsi yang ditentukan, tingkat *performace*, kehandalan dan tidak akan gagal selama dalam penggunaan yang normal (Sommerville, 2011, p.224).

Pada umumnya *release testing* menggunakan *black-box testing*, dimana pengujian dilakukan berdasarkan spesifikasi sistem. Sistem diperlakukan secara *black-box* dimana perilaku sistem dapat ditentukan dari pengamatan masukan dan keluaran yang saling berhubungan. Nama lain dari pengujian ini adalah *functional testing* karena penguji hanya memperhatikan fungsionalitas sistem (Sommerville, 2011, p.224).

2.12.1 Requirement-based Testing

Requirement-based testing adalah sebuah sistematisa pendekatan penyusunan *test case* berdasarkan setiap *requirement* sistem. Pendekatan berdasarkan *requirement* merupakan pengujian validasi untuk mendemonstrasikan bahwa sistem dapat meng-implementasikan persyaratannya dengan tepat (Sommerville, 2011, p.225).

Sebagai contoh jika ada daftar persyaratan sistem alergi obat dengan spesifikasi berikut:

Jika pasien diketahui alergi terhadap obat tertentu, maka sistem akan menampilkan pesan peringatan kepada pengguna sistem.

Jika pengguna memilih untuk mengabaikan peringatan tersebut, maka harus melampirkan alasan kenapa pesan tersebut diabaikan.

Untuk memastikan bahwa persyaratan tersebut dapat dipenuhi oleh sistem, maka *test case* yang bisa dibuat adalah sebagai berikut:

1. Memproses masukkan pasien yang tidak memiliki alergi obat. Memastikan bahwa sistem tidak akan memberikan pesan peringatan.
2. Memproses masukan pasien yang memiliki alergi obat. Memastikan bahwa sistem akan memebrika pesan peringatan.
3. Memproses masukan pasien yang memiliki alergi lebih dari satu obat, kemudian tulis resep obat tersebut secara terpisah. Memeriksa bahwa sistem memberikan peringatan untuk setiap alergi obat dengan tepat.
4. Menulis dua atau lebih resep obat yang menyebabkan pasien alergi. Memeriksa apakah sistem akan memberikan peringatan untuk kedua obat tersebut.
5. Memproses masukan pasien yang memiliki alergi obat, kemudian diabaikan. Periksa apakah sistem membutuhkan pengguna untuk memberikan informasi kenapa alergi obat tersebut diabaikan (Sommerville, 2011, p.225).

Dari contoh dapat disimpulkan bahwa dari sebuah *requirement* dapat dilakukan lebih dari satu *test case*. Pada umumnya lebih dari satu *test case* digunakan untuk menguji sebuah *requirement* untuk memastikan bahwa semua kemungkinan telah diuji. Sebuah catatan *traceability* harus dibuat untuk mempermudah penelusuran *test case* terhadap *requirement* (Sommerville, 2011, p.225).

2.12.2 Scenario Testing

Scenario testing adalah satu pendekatan *release testing* untuk merancang suatu *test case* berdasarkan skenario khusus dari penggunaan sistem. Skenario adalah sebuah cerita yang mendeskripsikan salah satu cara untuk menggunakan sistem. Skenario harus realistis dan pengguna sistem yang nyata harus bisa berhubungan dengan skenario tersebut (Sommerville, 2011, p.225).

Skenario yang dibuat harus merupakan cerita naratif yang mendeskripsikan harus masuk akal dan cukup lengkap. Sebagai contoh skenario yang mungkin untuk mendeskripsikan sistem konsultasi medis *online*:

1. Melakukan proses *authentication* dengan melakukan login kedalam sistem.
2. Mengunduh dan mengunggah catatan pasien ke dalam laptop.
3. Melakukan konsultasi *online*.
4. Melakukan enkripsi dan deskripsi catatan pasien pada perangkat bergerak.
5. Mencatat perubahan dan modifikasi data.
6. Merubah database obat berdasarkan modifikasi tersebut (Sommerville, 2011, p.226).

Skenario pada paragraf sebelumnya akan digunakan oleh penguji. Penguji akan mengoperasikan sistem dengan menjalankan langkah – langkah sesuai dengan skenario. Selama mengikuti skenario, penguji juga akan memberikan masukan

yang tidak valid terhadap sistem, hal tersebut digunakan untuk mengecek respon sistem jika memproses masukan yang tidak valid (Sommerville, 2011, p.226).

Ketika menggunakan pendekatan berbasis skenario, maka pada umumnya beberapa *requirement* dapat diuji kedalam satu skenario. Hal tersebut digunakan untuk memeriksa bahwa kombinasi antar *requirement* bekerja dengan baik (Sommerville, 2011, p.227).

2.12.3 Performance Testing

Performance testing bertujuan untuk mendemonstrasikan bahwa sistem dapat mencukupi persyaratan yang telah ditentukan dan untuk menemukan *error* dan *defect*. Pada *performance testing* dilakukan proses penekanan sistem dengan mengirim permintaan data melebihi batasan yang telah didesain. Pengujian tersebut dikenal dengan nama *stress testing* (Sommerville, 2011, p.227).

Sebagai contoh, jika sebuah sistem didesain untuk memproses 300 transaksi per detik. Maka *stress testing* dimulai dengan menguji sistem untuk memproses transaksi kurang dari 300 per detik, kemudian ditingkatkan menjadi 300 per detik, dan terus ditingkatkan hingga sistem mengalami kegagalan dalam beroperasi (Sommerville, 2011, p.227).

Ada dua tipe yang merupakan kegunaan *stress testing*, yaitu:

1. Untuk menguji perilaku sistem ketika terjadi *failure*. *Stress testing* akan menguji sistem dengan membebani jumlah permintaan data melebihi batas yang ditentukan sampai sistem mengalami kegagalan operasi (*failure*). Pada kondisi tersebut harus dipastikan bahwa ketika terjadi *failure* tidak akan merusak kinerja sistem, seperti korupsi data (*data corruption*) dan malfungsi sistem dalam melayani pengguna.
2. Pada *stress testing* mungkin akan ditemukan *defect* pada sistem yang baru muncul ketika dilakukan *stress testing*. Namun bisa diaktakan bahwa *defect* ini tidak akan terjadi selama dalam penggunaan sistem secara normal (Sommerville, 2011, p.227).

2.13 User Testing

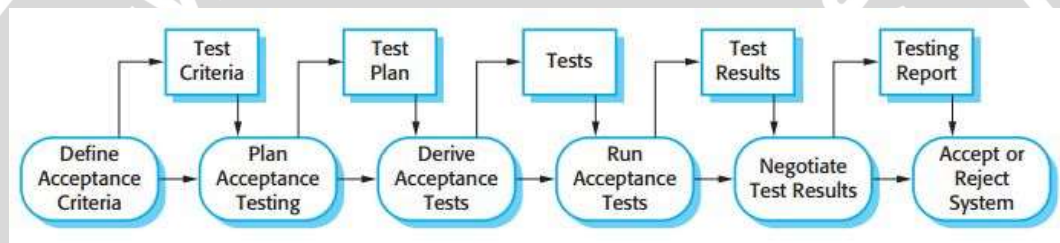
User testing merupakan suatu tahap dimana pengguna atau pelanggan menggunakan sistem dan memberi saran terhadap sistem tersebut. *User testing* penting untuk dilakukan meskipun sistem telah diuji pada tahap *development* dan *release*. Alasan pentingnya dilakukan *user testing* karena pengaruh lingkungan kerja pengguna berpengaruh besar terhadap keandalan, kinerja, daya guna dan ketahanan sistem (Sommerville, 2011, p.228).

Secara praktek, ada tiga macam tipe *user testing* yaitu:

1. *Alpha testing*, dimana pengujian dilakukan oleh pengguna yang berkolaborasi dengan tim pengembang aplikasi pada lingkungan pengembangan. *Alpha testing* digunakan pada pengembangan perangkat lunak secara *custom*

dengan *agile method* (melibatkan pengguna terlibat kedalam proses pengembangan).

2. *Beta testing*, dimana sebuah versi rilis *software* dapat memungkinkan pengguna untuk ber-eksperimen dan mengajukan masalah yang mereka temukan kepada tim pengembang aplikasi. *Beta testing* digunakan untuk ketika sistem yang belum diselesaikan secara menyeluruh dirilis kepada grup pengguna tertentu untuk proses evaluasi.
3. *Acceptance testing*, dimana pelanggan menguji sistem untuk memutuskan apakah sistem tersebut layak diterima oleh pelanggan dan digunakan pada lingkungan pelanggan. *Acceptance testing* merupakan bagian yang melekat pada proses pengembangan aplikasi. Pengujian ini dilakukan setelah *release testing*. *Acceptance testing* melibatkan pelanggan secara langsung dalam menguji sistem untuk menentukan apakah sistem layak diterima oleh pelanggan atau tidak (Sommerville, 2011, p.228-229).



Gambar 2.13 Proses Acceptance Testing

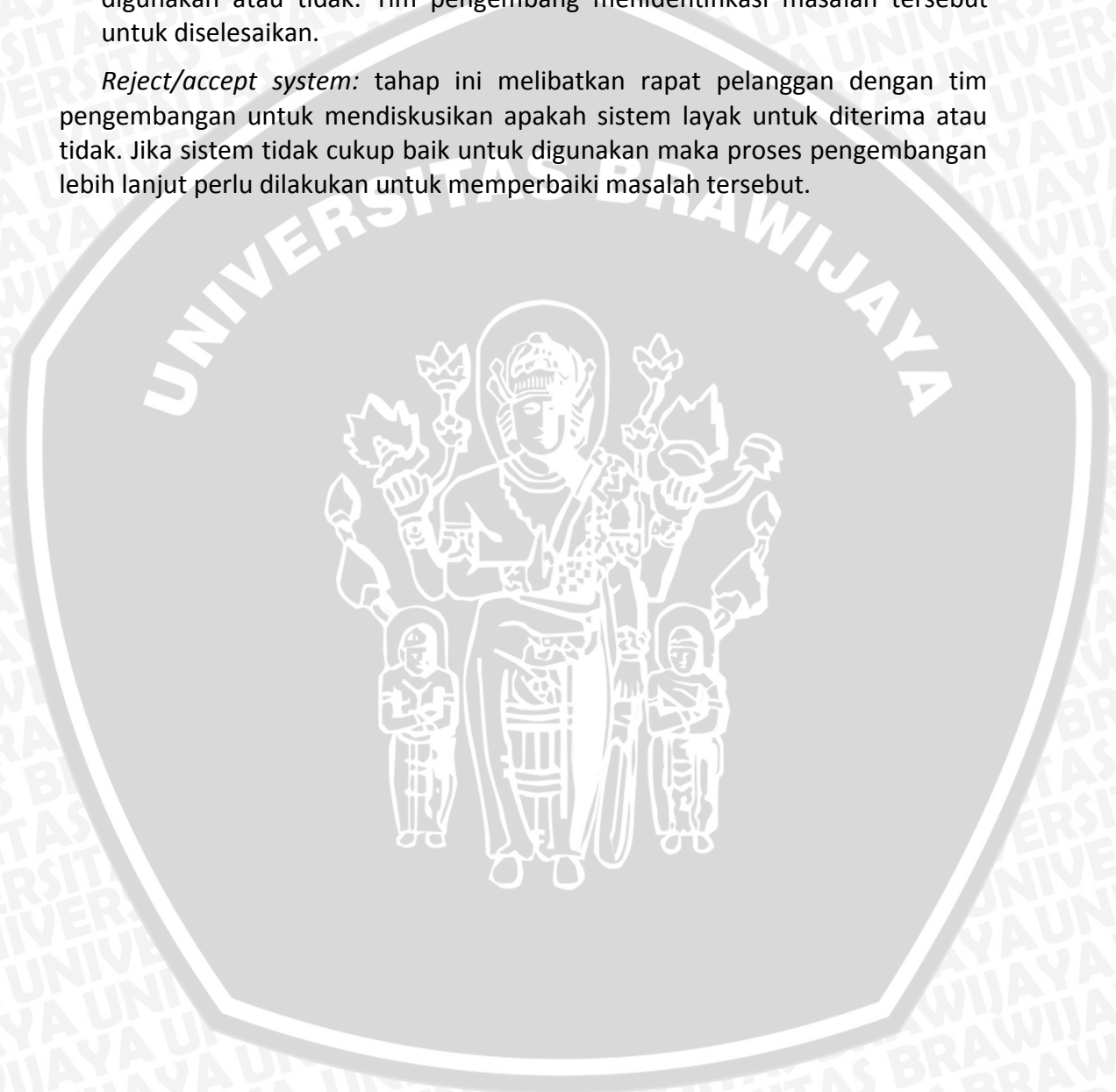
Ada enam tahapan pada proses *acceptance testing*, yang digambarkan pada gambar 2.13 yaitu:

1. *Define acceptance criteria*: tahap ini merupakan tahap awal sebelum kontrak sistem disetujui. *Acceptance criteria* merupakan bagian dari kontrak sistem yang disetujui antara pengembang aplikasi dan pelanggan. Pada prakteknya *acceptance criteria* sulit untuk didefinisikan diawal dan mungkin terjadi perubahan permintaan secara signifikan selama proses pengembangan.
2. *Plan acceptance testing*: terdiri dari proses penentuan sumber daya, waktu dan anggaran untuk *acceptance testing* dan membuat jadwal pengujian. *Acceptance plan* harus mendiskusikan cakupan *requirement* yang diujikan dan urutan fitur sistem yang akan diuji. Pada tahap ini didefinisikan resiko yang mungkin terjadi pada proses pengujian, seperti *system crash* dan kinerja yang tidak tidak mencukupi dan mendiskusikan cara yang digunakan untuk mengurangi resiko yang mungkin terjadi.
3. *Derive acceptance tests*: pada tahap ini dilakukan desain pengujian untuk menentukan apakah sistem layak diterima atau tidak. Desain pengujian harus mencakup keseluruhan sistem dengan lengkap baik fitur fungsional maupun fitur non-fungsional.

4. *Run acceptance test*: desain pengujian yang telah disetujui akan dieksekusi terhadap sistem. Tahap ini harus dilakukan pada lingkungan nyata dimana sistem tersebut akan digunakan.

5. *Negotiate test result*: ada tahap ini dilakukan negosiasi terhadap hasil pengujian. Jika ditemukan masalah pada sistem, maka pelanggan dan tim pengembang harus mendiskusikan apakah sistem sudah cukup baik untuk digunakan atau tidak. Tim pengembang mengidentifikasi masalah tersebut untuk diselesaikan.

Reject/accept system: tahap ini melibatkan rapat pelanggan dengan tim pengembangan untuk mendiskusikan apakah sistem layak untuk diterima atau tidak. Jika sistem tidak cukup baik untuk digunakan maka proses pengembangan lebih lanjut perlu dilakukan untuk memperbaiki masalah tersebut.



BAB 3 METODOLOGI

Pada bab ini berisi tentang metode atau langkah – langkah yang digunakan dalam penelitian untuk membuat sebuah *code generator*. Langkah – langkah tersebut terdiri dari: 1) studi literatur, 2) pengumpulan data, 3) analisis persyaratan, 4) perancangan sistem, 5) implementasi, 6) pengujian dan analisis, 7) pengambilan kesimpulan dan saran. Gambar 3.1 menggambarkan bagan metodologi penelitian.



Gambar 3.1 Bagan Metodologi Penelitian

3.1 Studi literatur

Studi literatur yang dibutuhkan untuk mempelajari dasar teori terkait dengan penelitian yang dilakukan. Teori – teori tersebut diperoleh dari berbagai sumber seperti buku, jurnal, e-book, penelitian sebelumnya, internet dan kuisioner. Teori yang dipelajari terkait dengan penelitian ini adalah konsep metaprogramming, template programming dan penggunaan database MySQL untuk melihat detail tabel.

3.2 Pengumpulan data

Dalam melakukan penelitian, terdapat beberapa teknik dalam mengumpulkan data, yaitu:

3.2.1 Observasi

Penelitian lapangan dilakukan dengan menganalisis komponen penyusun PTIIK-Apps yang ada di server pengembangan BPTIK. Data yang didapat dari proses observasi adalah data komponen MVC penyusun PTIIK-Apps.

Data tersebut digunakan untuk membuat pemetaan pada diagram FODA. Hasil dari pemetaan dapat digunakan untuk membuat perancangan metaprogram yang akan dibuat (Stuikys & Damasevicius, 2013, p.152).

3.2.2 Kuisisioner

Kuisisioner dilakukan untuk mengetahui kondisi yang ada di lapangan. Data hasil kuisisioner digunakan untuk membantu proses analisis persyaratan dan perancangan aplikasi.

3.2.3 Riset pustaka

Riset pustaka dilakukan untuk dengan cara mengumpulkan data dari berbagai referensi. Data tersebut digunakan untuk menyelesaikan permasalahan dalam penelitian untuk membuat aplikasi code generator, sehingga dapat membantu dalam membuat code generator untuk menyelesaikan masalah yang ada.

3.3 Analisis persyaratan

Analisis persyaratan bertujuan untuk menganalisis dan mendapatkan semua persyaratan yang diperlukan dalam implementasi *program generator* dan siapa saja yang terlibat di dalamnya. *Use case* diagram digunakan untuk mendeskripsikan perilaku sistem, yaitu bagaimana pengguna berinteraksi dengan pengguna.

Tahap analisis persyaratan terdiri dari tiga langkah, yaitu: 1) melakukan penjabaran gambaran umum aplikasi, 2) melakukan proses identifikasi aktor yang terlibat dalam aplikasi, 3) membuat daftar persyaratan pengguna dengan pemodelan diagram *use case*.

3.4 Perancangan sistem

Perancangan perangkat lunak dilakukan setelah semua persyaratan perangkat lunak selesai dilakukan. Perancangan yang dilakukan dengan menggunakan notasi – notasi berdasarkan pada UML (Unified Modeling Language) (Sommerville, 2011). Perancangan sistem dibagi menjadi beberapa tahap, yaitu:

3.4.1 Domain engineering

Domain engineering digunakan untuk mendapatkan *domain model* dari PTIIK-Apps. *Domain model* diperlukan untuk implementasi *program generator* sehingga sesuai dengan domain PTIIK-Apps.

Domain engineering dilakukan dengan proses analisis domain, desain domain dan implementasi domain. Analisis domain dilakukan untuk mengetahui kesamaan dan perbedaan fitur pada modul – modul PTIIK-Apps untuk menghasilkan komponen PTIIK-Apps yang bisa digunakan ulang. Desain domain dilakukan untuk mendefinisikan referensi arsitektur untuk mengembangkan model domain. Implementasi domain adalah proses untuk merealisasikan desain domain menjadi sebuah aplikasi yang merupakan solusi domain.

3.4.2 Perancangan sistem

Code generator adalah hasil akhir dari penelitian skripsi ini. *Program generator* dapat digunakan pengembang untuk menghasilkan kode program PTIIK-Apps dengan otomatis.

Proses perancangan sistem *code generator* membutuhkan beberapa tahap, yaitu: 1) perancangan *application description*, 2) perancangan *domain knowledge*, 3) perancangan *code generator*.

3.4.2.1 Perancangan *application description*

Application description digunakan untuk mendefinisikan masukan yang dibutuhkan oleh *program generator* untuk menghasilkan kode program. *Application description* berupa rancangan *class diagram* dan dijelaskan dengan definisi atribut kelas dan kardinalitas antar kelas.

3.4.2.2 Perancangan *domain knowledge*

Domain knowledge adalah sebuah basis pengetahuan pada *program generator*. *Domain knowledge* digunakan *program generator* untuk menerjemahkan masukan (*application description*) menjadi kode program yang sesuai dengan domain yang telah didefinisikan.

Perancangan *domain knowledge* dilakukan dengan menjabarkan *domain model* kedalam skema logis untuk mengubah masukan menjadi keluaran yang valid. Proses perancangan *domain knowledge* dimulai dengan membuat *activity diagram* berdasarkan model domain. Perancangan *activity diagram* digunakan untuk mengetahui alur basis pengetahuan dalam menjalankan tugasnya.

Perancangan *sequence diagram* digunakan untuk mengetahui interaksi antar komponen pada *domain knowledge*. Perancangan *sequence diagram* dilakukan berdasarkan *activity diagram* yang telah dibuat. Hasil dari rancangan *sequence diagram* dapat digunakan untuk membantu proses pembuatan rancangan *class diagram*.

Perancangan *class diagram* pada *domain knowledge* disatukan dengan rancangan *class diagram* pada *program generator*, karena dimungkinkan adanya sebuah komponen identik yang dibutuhkan *domain knowledge* dan *program generator*. Jadi, pada proses perancangan *class diagram* tidak akan terjadi redundansi komponen (komponen yang sama dengan komponen lain).

3.4.2.3 Perancangan *code generator*

Code generator adalah sebuah sistem yang memanfaatkan *application description* dan *domain knowledge* untuk menghasilkan kode program yang sesuai dengan domain yang telah didefinisikan. Proses perancangan *program generator* dilakukan dengan beberapa tahap, yaitu: 1) perancangan *activity diagram*, 2) perancangan *sequence diagram*, 3) perancangan *class diagram*.

Activity diagram digunakan untuk menjabarkan alur proses *code generator* dalam menjalankan fungsinya. *Sequence diagram* digunakan untuk menganalisis hubungan antar komponen penyusun *code generator*. Hasil akhir dari perancangan *code generator* adalah *rancangan class diagram*. Rancangan *class diagram* digunakan untuk membantu proses implementasi kode program.

3.5 Implementasi

Implementasi aplikasi *code generator* dilakukan berdasarkan rancangan yang telah dibuat. Implementasi aplikasi *code generator* ini menggunakan bahasa pemrograman PHP berorientasi objek. Pembuatan aplikasi *code generator* membutuhkan database dengan satu tabel saja, yaitu tabel `tbl_user` untuk menyimpan data pengguna yang bisa mengakses sistem.

3.6 Pengujian dan analisis

Pengujian digunakan untuk memastikan bahwa aplikasi yang dikembangkan:1) dapat digunakan dengan baik untuk menyelesaikan masalah, 2) telah mampu bekerja sesuai dengan spesifikasi dari persyaratan yang telah dibuat. Pengujian yang digunakan dalam penelitian ini adalah:

1. Development Testing

Development testing dilakukan untuk memastikan bahwa komponen sistem dapat bekerja secara baik pada penggunaan yang normal. Pengujian ini dilakukan dengan melibatkan tim pengembang aplikasi.

Development testing dilakukan secara *white box*, pengujian dilakukan dengan menguji kode program penyusun sistem. Metode *basis path* digunakan untuk menelusuri setiap alur kode program untuk memastikan bahwa setiap cabang dilewati dan dapat berjalan sesuai harapan.

2. Release Testing

Release testing pada penelitian dilakukan dengan berbasis requirement. Test case dibuat berdasarkan daftar persyaratan sistem dan daftar persyaratan pada *domain model*. Pengujian dilakukan dengan metode *black box*, dengan menguji fungsionalitas sistem tanpa meninjau kode program penyusun sistem tersebut. Tujuan dari pengujian ini adalah untuk melakukan verifikasi bahwa sistem yang dikembangkan telah memenuhi persyaratan yang telah didefinisikan.

3.7 Pengambilan kesimpulan dan saran

Pengambilan kesimpulan dilakukan setelah proses pengujian sistem, sehingga dapat diketahui efektifitas kinerja sistem. Tahap terakhir yaitu pengambilan saran yang dapat membantu dalam pengembangan aplikasi *code generator* selanjutnya.

BAB 4 ANALISIS PERSYARATAN

Bab ini membahas proses analisis persyaratan untuk membuat *program generator*. Pada bab ini akan menjelaskan gambaran umum aplikasi, kemudian akan membahas tentang proses analisis persyaratan yaitu: identifikasi aktor, identifikasi persyaratan fungsional dan non-fungsional.

4.1 Gambaran umum aplikasi

Sistem yang akan dikembangkan berjalan pada lingkungan *web browser*. Sistem dikembangkan dengan bahasa pemrograman PHP, HTML, CSS dan Javascript. *Framework* PHP yang digunakan untuk membangun sistem adalah *CodeIgniter*. *Web server* yang digunakan untuk menjalankan sistem adalah *Apache*, *web server* digunakan sebagai *server* yang menyediakan fungsi HTTP sebagai protokol transfer untuk sarana komunikasi antara sistem dengan klien.

Pengguna sistem yang dituju adalah staff pengembang yang bekerja di BPTIK. Dengan bantuan sistem ini diharapkan mempermudah mereka dalam melakukan implementasi kode program.



Gambar 4.1 Perancangan Umum Sistem

Gambar 4.1 menunjukkan skema tentang gambaran sistem yang akan dibuat. Berikut akan dijelaskan proses – proses yang terjadi pada sistem:

1. Pengembang mengakses alamat sistem, yang akan diproses oleh *web server*.
2. *Web server* memetakan alamat yang diakses oleh pengembang ke dalam sistem.

3. Sistem akan menjalankan perintah berdasarkan alamat yang dikirim oleh *web server*. Namun, sebelumnya sistem akan memeriksa apakah pengembang yang bersangkutan memiliki hak akses terhadap sistem.
4. Jika belum memiliki hak akses, sistem akan meminta pengembang untuk melakukan *login* dan memeriksa data akun pengembang ke sistem basis data.
5. Sistem memberikan layanan kepada pengembang untuk mengakses *server* basis data diluar sistem untuk mengambil *metadata* yang diperlukan.
6. Sistem membaca struktur data dari *server* basis data, kemudian mengambil *metadata* yang akan digunakan sebagai parameter masukan untuk proses pengisian model masukan.
7. Pengembang mengisi model masukan yang digunakan sebagai abstraksi dari komponen kode program yang akan dibangkitkan oleh sistem.
8. Sistem memproses data model masukan pengembang menjadi komponen kode program.

4.2 Analisis persyaratan

Proses analisis persyaratan diawali dengan 1) identifikasi aktor, 2) identifikasi persyaratan fungsional, 3) dan identifikasi persyaratan non-fungsional.

4.2.1 Identifikasi aktor

Pada tahap ini dilakukan proses identifikasi aktor – aktor yang akan berinteraksi dengan sistem. Tabel 4.1 menunjukkan aktor yang terlibat beserta penjelasannya.

Tabel 4.1 Identifikasi Aktor

Aktor	Deskripsi
Pengembang (<i>Developer</i>)	Pembuat aplikasi PTIIK-Apps yang menggunakan sistem <i>program generator</i> untuk menghasilkan komponen MVC secara otomatis.

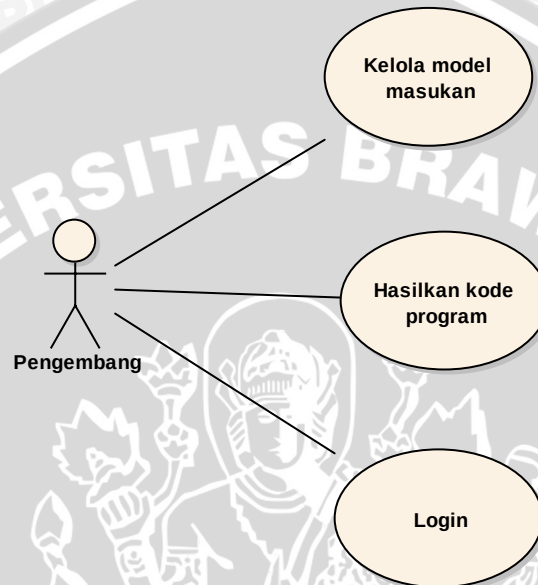
4.2.2 Identifikasi persyaratan fungsional

Daftar persyaratan fungsional merupakan daftar layanan yang harus disediakan oleh sistem untuk memenuhi persyaratan pengguna. Spesifikasi persyaratan fungsional ditunjukkan dengan *use case model* yang terdiri dari *use case diagram* dan *use case specification*.

4.2.2.1 Use case diagram

Use case diagram digunakan untuk menggambarkan skenario sederhana untuk mendeskripsikan tentang sistem yang diharapkan oleh pengguna. *Use case diagram* menunjukkan sekumpulan aktor, *use case* dan hubungan antara aktor dengan *use case*.

Use case diagram sistem *program generator* PTIIK-Apps ditunjukkan pada Gambar 4.2 berikut ini:



Gambar 4.2 Use Case Diagram Sistem Program Generator PTIIK-Apps

4.2.2.2 Use case specification

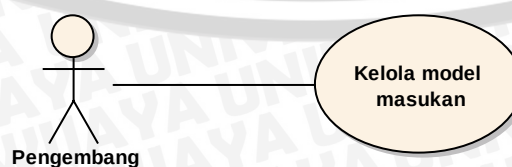
Use case specification adalah deskripsi alur kejadian dari sebuah *use case* yang menjelaskan hubungan antara aktor dengan sistem. Berikut ini *use case specification* sistem *program generator* PTIIK-Apps:

Use case specification kelola model masukan

1. Brief Description

Use case ini menjelaskan bagaimana **pengembang** menggunakan sistem *code generator* untuk mengelola model masukan.

2. Use-Case Diagram



3. Preconditions

- Pengembang telah melakukan *login* ke dalam sistem *program generator*
- Dibutuhkan jaringan internet untuk mengakses halaman kelola model masukan

4. Basic Flow of Events

4.1 Mengakses Fungsi Kelola Model Masukan

Use case ini diawali ketika aktor **pengembang** mengakses fungsi kelola model masukan.

Sistem akan menampilkan daftar model komponen kepada **pengembang** beserta dengan opsi – opsi untuk melakukan operasi model komponen.

4.2 Melakukan Operasi Model Masukan

Pengembang melakukan operasi terhadap model komponen sesuai dengan keinginannya.

Ketika **pengembang** menjalankan operasi tambah model masukan, maka *use case* akan menjalankan **sub flow 6.1**.

Ketika **pengembang** menjalankan operasi perbarui model masukan, maka *use case* akan menjalankan **sub flow 6.2**.

Ketika **pengembang** menjalankan operasi hapus model masukan, maka *use case* akan menjalankan **sub flow 6.3**.

Pengembang memuat *metadata* dari *server* basis data dengan menjalankan **sub flow 6.4**.

4.3 Muat Ulang Tampilan Model Masukan

Sistem akan memuat ulang tampilan model masukan sesuai dengan operasi yang dilakukan oleh pengguna.

Pengembang dapat melakukan operasi model masukan lagi dengan menjalankan **flow 4.2**.

Jika model masukan sudah sesuai dengan persyaratan, maka **pengembang** dapat menjalankan **flow 4.4**.

4.4 Use Case Berakhir

Use case ini berakhir ketika **pengembang** menjalankan *use case* **Hasilkan Kode Program**.

5. Alternative Flows

5.1 Validasi Model Masukan

5.1.1 Model Masukan Valid

Pada tahap **operasi model masukan** pada **sub flow 6.1**, **sub flow 6.2** dan **sub flow 6.3**, jika model masukan valid maka:

1. **Sistem** akan mengaktifkan tombol proses data sehingga **pengembang** bisa menjalankan *use case* **Hasilkan Kode Program**.

2. Melanjutkan *Basic Flow* proses **Muat Ulang Tampilan Model Masukan**.

5.1.2 Model Masukan Tidak Valid

Pada tahap **Operasi Model Masukan** pada *sub flow* 6.1, *sub flow* 6.2 dan *sub flow* 6.3, jika model masukan valid maka:

1. **Sistem** akan menon-aktifkan tombol proses data sehingga **pengembang** tidak bisa menjalankan *use case* **Hasilkan Kode Program**.

2. Melanjutkan *Basic Flow* proses **Muat Ulang Tampilan Model Masukan**.

5.2 Penanganan Sambungan Server Basis Data

5.2.1 Sambungan Gagal

Pada tahap **Penyambungan Server Basis Data** pada *sub flow* 6.4, ketika sambungan gagal dibuat maka:

1. **Sistem** akan memberikan pesan kepada **pengembang**.

2. Melanjutkan *Sub Flow* 6.4.

6. Sub Flows

6.1 Tambah Model Komponen

1. Tambah Model Masukan

Sistem menambahkan jenis model masukan sesuai dengan perintah **pengembang**.

Ketika **pengembang** menambahkan model tabel, maka harus dipastikan bahwa **pengembang** telah menjalankan *sub flow* 6.4.

2. Validasi Model Masukan

Sistem melakukan validasi model masukan dengan menjalankan *alternative flow* A.1.

3. Sub Flow Berakhir

Sistem menjalankan flow 4.3 pada *basic flow*.

6.2 Perbarui Model Masukan

1. Mengisi Data Model Masukan

Pengembang mengisi data pada model masukan yang diinginkan.

2. Memperbarui Model Masukan

Sistem memperbarui model masukan.

3. Validasi Model Masukan

Sistem melakukan validasi model masukan dengan menjalankan **alternative flow A.1**.

4. Sub Flow Berakhir

Sistem menjalankan flow 4.3 pada *basic flow*.

6.3 Hapus Model Masukan

1. Menampilkan Pesan Konfirmasi

Sistem menampilkan pesan konfirmasi hapus model masukan.

Jika **pengembang** membatalkan operasi hapus model masukan, maka *sub flow* berakhir.

2. Konfirmasi Hapus Model Masukan

Pengembang melakukan konfirmasi hapus model masukan.

Sistem akan menghapus data model masukan tersebut.

3. Validasi Model Masukan

Sistem melakukan validasi model masukan dengan menjalankan **alternative flow A.1**.

4. Sub Flow Berakhir

Sistem menjalankan flow 4.3 pada *basic flow*.

6.4 Penyambungan Server Basis Data

1. Mengisi Form Sambungan Basis Data

Pengembang mengisi form sambungan basis data yang berisi masukan alamat *server*, *username* dan *password server* basisdata.

Pengembang mengirim form sambungan basis data.

2. Melakukan Sambungan Basis Data

Sistem melakukan sambungan ke server basis data yang diinginkan oleh **pengembang**.

Jika sambungan gagal, maka akan menjalankan **alternative flow A.2**.

3. Memuat Metadata Basis Data

Sistem akan memuat *metadata* basis data.

4. Memilih Basis Data

Pengembang memilih *metadata* basis data yang diinginkan untuk memuat *metadata* model tabel.

5. Memuat Metadata Model Tabel

Sistem memuat *metadata* model tabel dari *metadata* basis data yang dipilih oleh **pengembang**.

6. Sub Flow Berakhir

Sistem menjalankan flow 4.3 pada *basic flow*.

7. Key Scenario

7.1 Success Scenario

Kelola Model Masukan – *Basic Flow*

Kelola Model Masukan – *Basic Flow, Alternative Flow*: Model Masukan Valid

7.2 Failure Scenario

Karena model masukan tidak valid – *Basic Flow, Alternative Flow*: Model Masukan Tidak Valid

8. Postconditions

- **Sistem** dapat mengelola model masukan sesuai dengan persyaratan **pengembang**.
- **Pengembang** dapat menjalankan *use case* **Hasilkan Kode Program** jika model masukan telah dinyatakan valid oleh **sistem**.

9. Extension Points

Tidak ada

10. Special Requirements

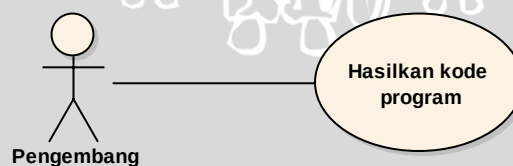
Tidak ada

Use case specification hasilkan kode program

1. Brief Description

Use case ini menjelaskan bagaimana **pengembang** menggunakan sistem *code generator* untuk menghasilkan kode program.

2. Use-Case Diagram



3. Preconditions

- **Pengembang** telah melakukan *login* ke dalam sistem *program generator*
- Dibutuhkan jaringan internet untuk menjalankan *use case*
- Model masukan harus dinyatakan valid oleh **sistem**

4. Basic Flow of Events

4.1 Menjalankan Fitur Proses Data

Use case ini diawali ketika aktor **pengembang** menekan tombol proses data model masukan.

4.2 Memproses Model Komponen

Sistem akan model masukan menjadi komponen kode program.

4.3 Menampilkan Hasil Kode Program

Sistem menampilkan hasil kode program yang telah dibangkitkan dari model masukan yang dikirim oleh **pengembang**.

4.3 Mengunduh Komponen Kode Program

Pengembang mengunduh kode program yang telah ditampilkan oleh **sistem**.

4.4 Use Case Berakhir

Use case berakhir.

5. Alternative Flows

Tidak ada

6. Sub Flows

Tidak ada

7. Key Scenarios

7.1 Success Scenario

Hasilkan Kode Program – *Basic Flow*

7.2 Failure Scenario

Tidak ada

8. Postconditions

- **Sistem** mampu mengolah model masukan menjadi kode program
- **Sistem** mampu menampilkan kode program yang telah dibangkitkan kepada **pengembang**
- **Pengembang** dapat mengunduh kode program yang telah dibangkitkan oleh **sistem**

9. Extension Points

Tidak ada

10. Special Requirement

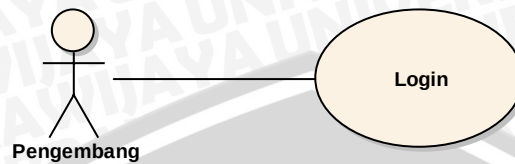
Tidak ada

Use case specification login

1. Brief Description

Use case ini menjelaskan bagaimana **pengembang** melakukan login kedalam siste sistem *code generator*.

2. Use-Case Diagram



3. Preconditions

- Dibutuhkan jaringan internet untuk menjalankan *use case*

4. Basic Flow of Events

4.1 Mengakses Fungsi Login

Use case ini diawali ketika aktor **pengembang** mengakses fungsi *login*.

4.2 Mengisi Form Login

Pengembang mengisi form *login* yang berisi masukan *username* dan *password* akun pengembang.

4.3 Validasi Pengembang

Sistem akan melakukan validasi terhadap data akun yang dikirim.

Jika akun dinyatakan valid, maka **sistem** akan mengarahkan **pengembang** untuk menjalankan *use case* **Kelola Model Masukan**.

Jika akun dinyatakan tidak valid maka *use case* akan menjalankan **alternative flow 5.1**.

4.3 Mengunduh Komponen Kode Program

Pengembang mengunduh kode program yang telah ditampilkan oleh **sistem**.

4.4 Use Case Berakhir

Use case berakhir.

5. Alternative Flows

5.1 Validasi Akun Pengembang

5.1.1 Akun Tidak Valid

Pada tahap **Validasi pengembang** dari **Basic Flow 4.3**, jika proses validasi gagal maka:

1. **Sistem** akan memuat ulang halaman *login*.
2. Pengembang dapat mengulangi *use case* pada **Basic Flow 4.2**.

6. Sub Flows

Tidak ada

7. Key Scenarios

7.1 Success Scenario

Login – Basic Flow

7.2 Failure Scenario

Login – Basic Flow, Alternative Flow: Akun Tidak Valid

8. Postconditions

- **Sistem** mampu melakukan validasi terhadap akun **pengembang**

9. Extension Points

Tidak ada

10. Special Requirement

Tidak ada

Glossary Use Case Specification

Tabel 4.2 adalah *glossary* dari *use case specification* program generator yang telah didefinisikan. Tabel tersebut memberika informasi yang lebih detail tentang istilah yang digunakan pada *use case specification*.

Tabel 4.2 Glossary of Use Case Specifications

- **Form sambungan server basis data:** adalah form yang digunakan pengembang sebagai masukan dalam melakukan proses sambungan ke server basis data. Form tersebut terdiri dari: 1) alamat server, 2) username dan 3) password.
- **Form login:** adalah form yang digunakan pengguna untuk melakukan proses login. Form tersebut berisi masukan username dan password akun pengembang.
- **Model masukan:** adalah masukan dari pengembang yang memiliki struktur/model berdasarkan desain dari *application description*.
- **Model masukan yang valid:** model masukan yang dinyatakan valid adalah sebagai berikut:
 - 1) model masukan tidak identik dengan dengan model komponen yang lain
 - 2) model masukan tidak memiliki nilai kosong pada atribut yang didefinisikan sebagai atribut *not null*
 - 3) model masukan tidak mengandung karakter simbol dan huruf

4.2.3 Identifikasi persyaratan non-fungsional

Persyaratan non fungsional pada sistem harus mencakup aspek: 1) *compability*, 2) *security* dan 3) sesuai dengan domain PTIIK-Apps. Persyaratan non-fungsional dideskripsikan pada Tabel 4.3.

Tabel 4.3 Spesifikasi Persyaratan Non-fungsional

Parameter	Deskripsi Persyaratan
<i>Compability</i>	- Aplikasi harus bisa memberikan layanannya dengan baik pada beberapa browser, yaitu: Google Chrome, Mozilla Firefox dan Safari. - Aplikasi <i>code generator</i> harus memiliki kemampuan untuk menghasilkan kode program yang bisa berjalan pada domain PTIIK-Apps.
<i>Security</i>	- Aplikasi harus memberikan tingkat keamanan terhadap lannya. Hanya pengguna yang memiliki akses ke dalam sistem yang bisa menjalankan aplikasi tersebut. (<i>authetication</i>) - Hak akses pengguna yang bisa masuk ke dalam sistem juga selalu diperiksa untuk memastikan bahwa pengguna tersebut memiliki hak akses terhadap fitur/layanan yang diminta. (<i>authorization</i>)

BAB 5 PERANCANGAN

Perancangan sistem meliputi proses *domain engineering*, perancangan *application description*, perancangan *domain knowledge* dan perancangan sistem *program generator*. Proses *domain engineering* digunakan untuk menghasilkan *domain model* yang digunakan untuk membantu proses perancangan selanjutnya. *Application description* dan *domain knowledge* merupakan komponen persyaratan untuk membuat *program generator*.

5.1 Domain Engineering

Rekayasa domain (*domain engineering*) dilakukan dengan melakukan analisis domain, desain domain dan implementasi domain. Istilah – istilah yang dipakai dalam penjabaran rekayasan domain dijelaskan dengan ***domain terminology dictionary*** pada bab Lampiran B. Adapun penjelasan pada setiap proses *domain engineering* adalah sebagai berikut:

Analisis domain dilakukan untuk mengetahui kesamaan dan perbedaan fitur pada modul – modul PTIIK-Apps yang akan menghasilkan model domain. Metode yang digunakan untuk melakukan analisis domain adalah *Feature Oriented Domain Analysis* (FODA). Tujuan utama FODA adalah menghasilkan produk domain yang bisa digunakan ulang.

Desain domain dilakukan untuk mendefinisikan referensi arsitektur atau bingkai kerja (*framework*) untuk pengembangan penggunaan ulang komponen yang telah didefinisikan pada proses analisis domain.

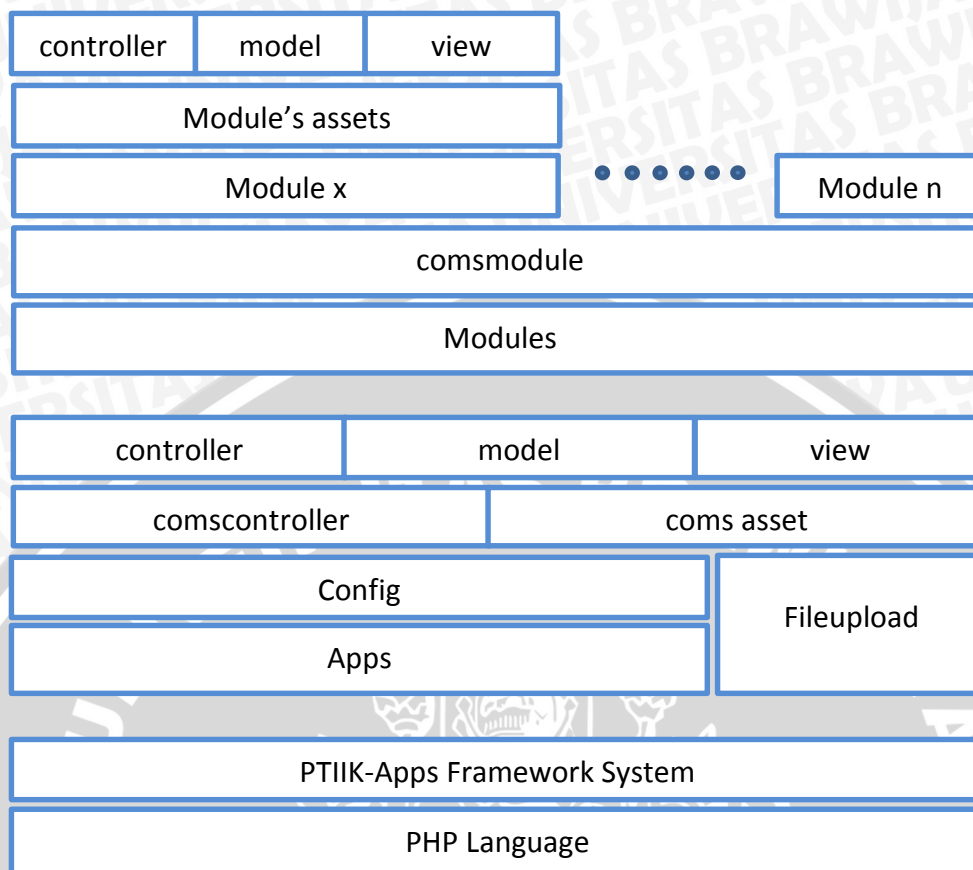
Implementasi domain adalah proses untuk merealisasikan desain domain menjadi sebuah aplikasi yang merupakan solusi domain.

5.2 Analisis Domain

Metode FODA digunakan untuk menganalisis domain dengan berorientasi terhadap fitur sistem. Metode FODA terdiri dari tiga tahap, yaitu: analisis konteks (*context analysis*), pemodelan domain (*domain modelling*) dan pemodelan arsitektur (*architecture modelling*).

5.2.1 Analisis konteks

Analisis konteks digunakan untuk mengidentifikasi cakupan domain pada aplikasi PTIIK-Apps yang akan digunakan ulang. Analisis konteks membantu peneliti untuk mengetahui bagaimana *reusable assets* dapat dibangun pada domain PTIIK-Apps. Domain PTIIK-Apps akan dimodelkan kedalam *structure diagram* yang ditunjukkan pada Gambar 5.1.



Gambar 5.1 Structure Diagram PTIIK-Apps

Sumber: (Perancangan)

Gambar 5.1 menunjukkan bahwa PTIIK-Apps memiliki komponen: 1) *panada framework system*, 2) *apps*, 3) *config*, 4) *comscontroller*, 5) *coms assets*, 6) *modules*, 7) *comsmodule*, 8) *module's assets*, 9) *controller*, 10) *model* dan 11) *view*. Berdasarkan analisis pada Gambar 5.1 domain PTIIK-Apps dapat disimpulkan bahwa:

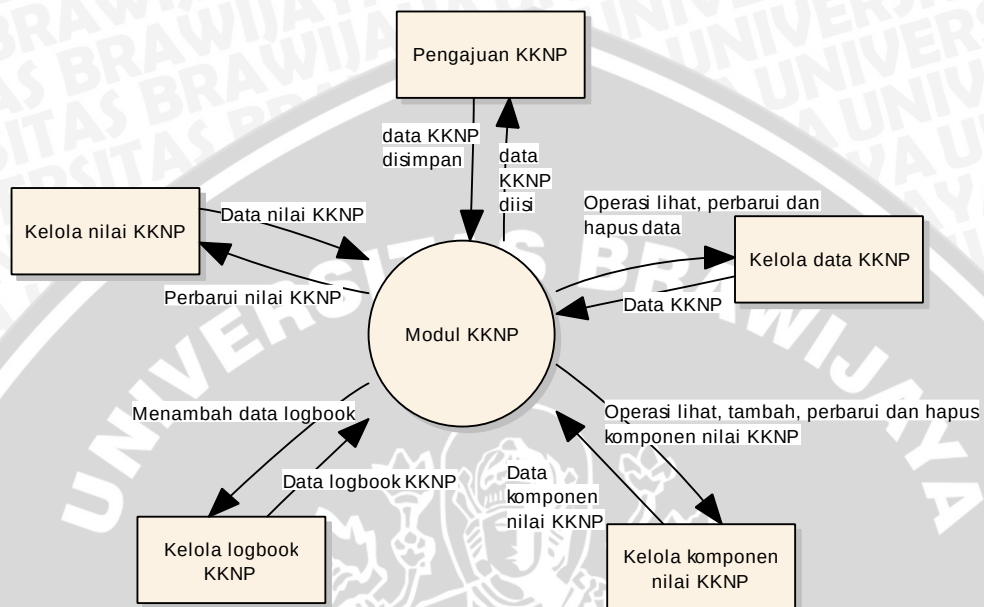
1. Domain PTIIK-Apps dibuat dengan bahasa pemrograman PHP.
2. Semua komponen PTIIK-Apps dijalankan dengan *library* yang ada pada **PTIIK framework system**.
3. Komponen **apps** merupakan komponen dimana aplikasi PTIIK-Apps berada.
4. Komponen **apps** terdiri dari **comscontroller**, **coms assets**, **modules** dan beberapa komponen **controller**, **model** dan **view**.
5. Komponen **controller** berinteraksi dengan beberapa unit **model** dan **view**.
6. Komponen **modules** terdiri dari **comsmodule** dan beberapa komponen **modul**.
7. Komponen **modul** terdiri dari **modul's assets** dan beberapa komponen **controller**, **model** dan **view**.

Modules merupakan komponen yang sering mengalami penambahan fitur. Modul – modul PTIIK-Apps dibuat dengan mengembangkan komponen **modul**. Komponen **modul** adalah komponen penyusun PTIIK-Apps yang paling banyak, selain itu komponen tersebut memiliki beberapa kesamaan struktur antara satu komponen dengan komponen yang lain. Sehingga pengembangan *reusable assets* sangat cocok jika disusun dari komponen **modul**.

Berikut ini penjelasan setiap komponen yang telah dijabarkan pada diagram stuktur pada Gambar 5.1:

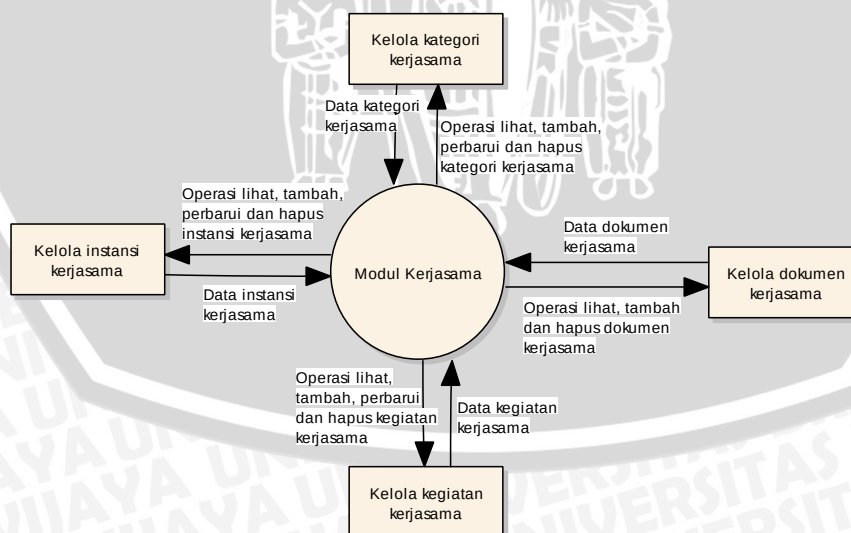
1. **PHP Language**: merupakan bahasa pemrograman penyusun aplikasi PTIIK-Apps.
2. **PTIIK Framework System**: komponen yang menyediakan arsitektur framework PTIIK-Apps dengan pola hirarkikal MVC (*Model View Controller*).
3. **Apps**: merupakan komponen dimana aplikasi PTIIK-Apps berada, komponen tersebut memakai fungsi yang disediakan oleh **PTIIK Framework System**.
4. **Config**: adalah unit komponen yang menyediakan konfigurasi komponen **Apps**.
5. **Fileupload**: merupakan lokasi tujuan ketika aplikasi PTIIK-Apps melakukan proses unggah *file*.
6. **Comscontroller**: adalah komponen yang menyediakan fungsi umum yang bisa diakses oleh MVC pada komponen **Apps**, baik MVC pusat maupun MVC cabang yang berada disetiap komponen **Modul**.
7. **Coms asset**: merupakan komponen yang menyediakan kumpulan aset seperti *file cascading style sheet (CSS)* dan *javascript* yang akan diakses oleh komponen **Controller** dan **View** pusat maupun **Controller** dan **View** cabang.
8. **Controller**: adalah komponen yang bertugas untuk mengolah alur proses yang akan membutuhkan komponen **Model** sebagai penyedia operasi data dan komponen **View** pengelola tatap muka.
9. **Model**: adalah komponen untuk mengatur operasi data, misalkan mengambil data dari basis data, menambah, mengubah dan menghapus data.
10. **View**: adalah komponen untuk mengolah tampilan tatap muka, sehingga aplikasi bisa melakukan interaksi dengan pengguna.
11. **Modules**: merupakan komponen yang menjadi wadah dari sekumpulan komponen **Modul**.
12. **Comsmodule**: merupakan fungsi umum yang bisa digunakan oleh komponen **Controller** dan **View** cabang yang berada pada komponen **Modul**.
13. **Modul**: merupakan komponen MVC cabang, yang menggambarkan setiap modul yang ada pada PTIIK-Apps.
14. **Modul's assets**: merupakan sekumpulan aset file *css* dan *javascript* yang diakses oleh komponen MVC cabang pada **Modul** yang bersangkutan.

Diagram alir data (*data-flow diagram*) digunakan untuk menggambarkan konteks domain PTIIK-Apps yang akan dianalisis. Penjelasan tersebut digambarkan menjadi dua diagram, yaitu: diagram untuk menjelaskan aliran data modul KKNP dan diagram untuk menjelaskan aliran data modul kerjasama. Gambar 5.2 menunjukkan diagram alir data modul KKNP.



Gambar 5.2 Data Flow Diagram Modul KKNP

Gambar 5.3 menjelaskan diagram alir data modul kerjasama. Analisis konteks dilakukan dengan membandingkan konteks pada Gambar 5.2 dan Gambar 5.3 untuk mencari persamaan dan perbedaan antar modul pada PTIIK-Apps.



Gambar 5.3 Data Flow Diagram Modul Kerjasama

Analisis yang dilakukan pada diagram alir pada Gambar 5.2 dan Gambar 5.3 dapat disimpulkan bahwa modul KKNP dan modul kerjasama:

- Jika dilihat dari sudut pandang pengguna sistem (administrator), kedua modul tersebut tidak memiliki kesamaan fitur yang bisa digunakan untuk membangun *reusable assets* domain PTIIK-Apps.
- Namun jika dilihat dari sudut pandang pengembang, kedua modul tersebut memiliki kesamaan dari segi operasi data yang dilakukan, misalkan: melihat data, menambah data, memperbarui dan menghapus data.
- Analisis fitur dapat dilakukan dengan membandingkan fitur pada kedua modul tersebut dari sudut pandang pengembang.

Hasil analisis konteks menunjukkan bahwa, modul pada PTIIK-Apps harus dianalisis lebih mendalam dari sudut pandang pengembang, sehingga dapat dihasilkan komponen *reusable assets* yang bisa dipakai untuk mewakili modul PTIIK-Apps dalam hal penggunaan ulang.

5.2.2 Pemodelan domain (*domain modelling*)

Proses pemodelan domain dilakukan dengan menganalisis domain yang telah dicakup pada proses analisis konteks. Analisis tersebut dilakukan untuk mengetahui persamaan dan perbedaan permasalahan yang ditunjukkan oleh aplikasi, kemudian akan menghasilkan sejumlah model yang menggambarkan aspek yang berbeda dari permasalahan tersebut. Pemodelan domain PTIIK-Apps dilakukan dengan tiga aktivitas utama, yaitu: 1) analisis fitur, 2) pemodelan *entity-relational*, 3) analisis fungsional.

5.2.2.1 Analisis Fitur

Analisis fitur pada domain PTIIK-Apps dilakukan pada beberapa modul PTIIK-Apps. Analisis fitur dilakukan untuk merancangan model fitur. Pada penelitian ini modul yang dianalisis adalah modul KKNP dan modul kerjasama.

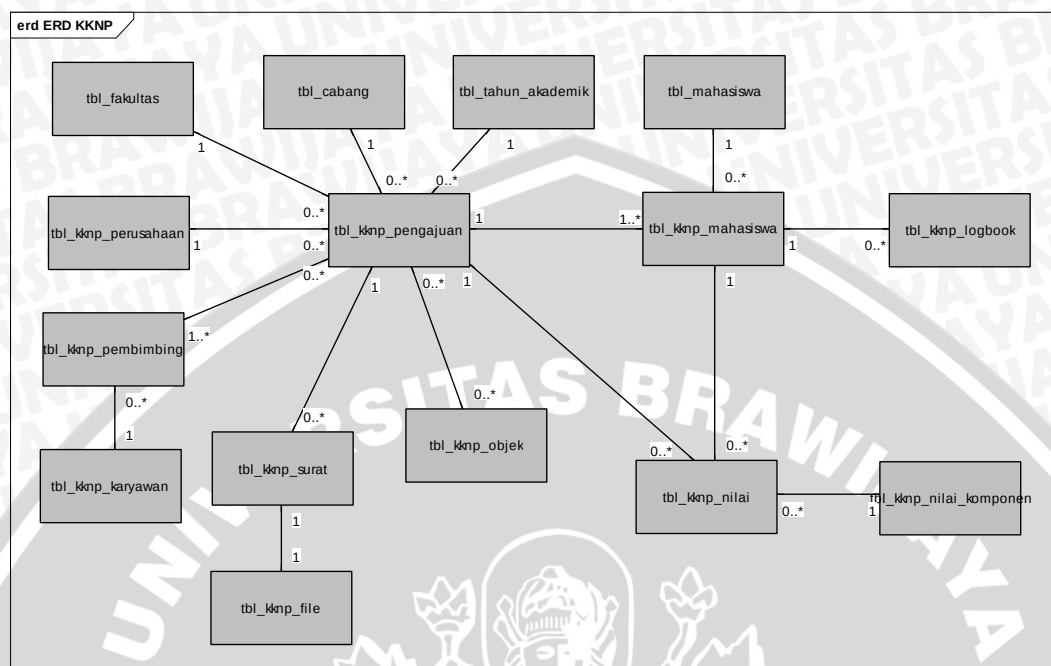
Analisis domain dilakukan dengan menjabarkan fungsionalitas modul sistem kedalam *use case model*. Kemudian kesamaan dan perbedaan antar fitur dikelompokkan berdasarkan perilaku fitur dalam mengolah data dari tabel basis data. *Use case specification* dijabarkan untuk menjelaskan setiap komponen *use case* pada *use case diagram*, pada pendefinisian *use case specification* ditambahkan analisis fitur untuk menjelaskan pola pemikiran ahli domain dalam menganalisis fitur tersebut.

Analisis modul KKNP

Modul Kuliah Kerja Nyata dan Praktek (KKNP) adalah modul yang digunakan untuk mengelola data KKNP mahasiswa PTIIK. Modul tersebut mengelola data KKNP mahasiswa, pembimbing yang terlibat dalam program KKNP, nilai hasil KKNP dan catatan harian (*logbook*) KKNP.

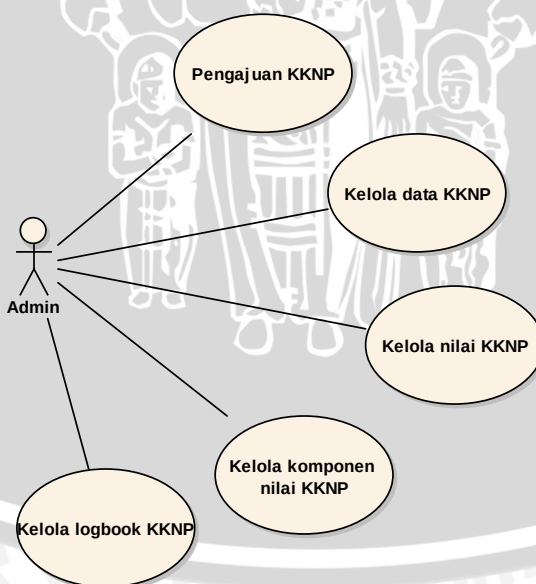
Gambar 5.4 adalah *entity relationship diagram* (ERD) yang menggambarkan hubungan antar *entity* pada modul KKNP. Hubungan tersebut ditunjukkan dengan tanda garis antar *entity*, pada tanda penghubung tersebut terdapat sebuah simbol

multiplicity yang menunjukkan tipe dari hubungan tersebut, seperti: *one-to-many*, *one-to-one* atau *many-to-many*.



Gambar 5.4 Entity Relationship Diagram (ERD) Modul KKNP

Fungsionalitas modul KKNP digambarkan ke dalam diagram *use case* pada Gambar 5.5.



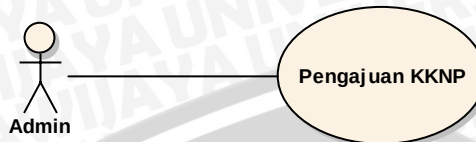
Gambar 5.5 Diagram Use Case Modul KKNP

1. Use Case Specification Pengajuan KKNP

1. Brief Description

Use case ini menjelaskan bagaimana **admin** melakukan pengajuan KKNP mahasiswa.

2. Use-Case Diagram



3. Preconditions

- **Admin** telah melakukan *login* ke dalam PTIIK-Apps
- Dibutuhkan jaringan internet untuk menjalankan *use case*

4. Basic Flow of Events

4.1 Mengakses Fungsi Pengajuan KKNP

Use case ini diawali ketika aktor **pengembang** mengakses fungsi pengajuan KKNP.

Sistem mengambil data dari tabel `tbl_cabang`, `tbl_fakultas`, `tbl_tahun_akademik`, `tbl_kknp_perusahaan`, `tbl_kknp_karyawan`, `tbl_kknp_mahasiswa` dan `tbl_kknp_objek` yang digunakan untuk mengisi masukan **admin** pada form pengajuan KKNP.

Sistem menampilkan form pengajuan KKNP.

4.2 Mengisi Form Pengajuan KKNP

Admin mengisi data KKNP pada form pengajuan KKNP.

4.3 Menyimpan data KKNP baru

Admin menekan tombol simpan pengajuan KKNP.

Sistem akan melakukan pengecekan masukan form pengajuan KKNP, jika form tidak terisi dengan lengkap maka akan menjalankan **Alternative Flow 5.1**.

Sistem menyimpan data KKNP yang telah dimasukkan oleh **admin** dengan ketentuan sebagai berikut:

- Beberapa atribut data KKNP disimpan pada `tbl_pengajuan_kknp`.
- Atribut data pembimbing KKNP disimpan pada tabel `tbl_kknp_pembimbing`
- Atribut data mahasiswa disimpan pada tabel `tbl_kknp_mahasiswa`.
- Atribut data surat KKNP disimpan pada tabel `tbl_kknp_surat`.

4.4 Use Case Berakhir

Use case berakhir.

5. Alternative Flows

5.1 Validasi Form Pengajuan KKNP

5.1.1 Form Pengajuan KKNP Tidak Valid

Pada **Basic Flow 4.3**, jika form pengajuan KKNP tidak valid maka:

1. **Sistem** akan memberikan informasi bagian form yang tidak valid
2. Menjalankan proses **Basic Flow 4.2**.

6. Sub Flows

Tidak ada

7. Key Scenarios

7.1 Success Scenario

Pengajuan KKNP – *Basic Flow*

7.2 Failure Scenario

Pengajuan KKNP – *Basic Flow, Alternative Flow: Form Pengajuan KKNP Tidak Valid*

8. Postconditions

- Data pengajuan KKNP disimpan berdasarkan masukan yang diisi oleh admin

9. Extension Points

Tidak ada

10. Special Requirement

Tidak ada

11. Analisis fitur

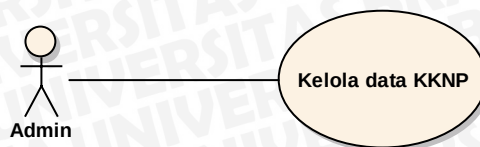
Use case ini digolongkan sebagai fitur **tambah data** pada tabel utama dan tambah data pada tabel cabang. Tabel utama yang dimaksud adalah tbl_pengajuan, sedangkan tabel cabang yang dimaksud adalah 1) tbl_kknp_mahasiswa, 2) tbl_kknp_pembimbing, 3) tbl_objek, 4) tbl_kknp_surat.

2. Use Case Specification Kelola Data KKNP

1. Brief Description

Use case ini menjelaskan bagaimana **admin** mengelola data KKNP.

2. Use-Case Diagram



3. Preconditions

- **Admin** telah melakukan *login* ke dalam PTIIK-Apps
- Dibutuhkan jaringan internet untuk menjalankan *use case*

4. Basic Flow of Events

4.1 Mengakses Fungsi Kelola KKNP

Use case ini diawali ketika aktor **admin** mengakses fungsi kelola KKNP.

Sistem akan menampilkan daftar KKNP dari `tbl_kkn_pengajuan` kepada **admin**.

Sistem menampilkan tombol opsi agar **admin** bisa melihat detail dan memperbarui data KKNP.

Admin bisa mengelola data nilai dan *logbook* KKNP dengan menjalankan *use case* **Kelola Nilai KKNP** dan **Kelola Logbook KKNP**.

4.2 Memilih Data KKNP

Admin memilih data KKNP yang ingin diperbarui.

4.3 Menampilkan Form Perbarui Data KKNP

Sistem mengambil data dari tabel `tbl_kkn_pengajuan` dan tabel yang berhubungan dengan `tbl_kkn_pengajuan` untuk melakukan pengisian data pada form perbarui data KKNP.

Sistem menampilkan form perbarui data KKNP.

4.3 Mengisi Form Perbarui Data KKNP

Admin memperbarui data pada form perbarui data KKNP.

4.4 Simpan Hasil Perubahan

Admin menekan tombol simpan hasil perubahan data KKNP.

Sistem akan melakukan validasi terhadap form data KKNP yang telah diperbarui oleh **admin**, jika form KKNP tidak valid maka akan menjalankan **Alternative Flow 5.1**.

Sistem menyimpan hasil perubahan data KKNP.

4.4 Use Case Berakhir

Use case berakhir.

5. Alternative Flows

5.1 Validasi Form KKNP

5.1.1 Form Pengajuan KKNP Tidak Valid

Pada **Basic Flow 4.4**, jika form pengajuan KKNP tidak valid maka:

1. **Sistem** akan memberikan informasi bagian form yang tidak valid
2. Menjalankan proses **Basic Flow 4.3**.

6. Sub Flows

Tidak ada

7. Key Scenarios

7.1 Success Scenario

Kelola Data KKNP – *Basic Flow*

7.2 Failure Scenario

Kelola Data KKNP – *Basic Flow, Alternative Flow: Form KKNP Tidak Valid*

8. Postconditions

- Data KKNP dapat diperbarui sesuai dengan perubahan yang dilakukan oleh admin

9. Extension Points

- Kelola Nilai KKNP
- Kelola Logbook KKNP

10. Special Requirement

Tidak ada

11. Analisis Fitur

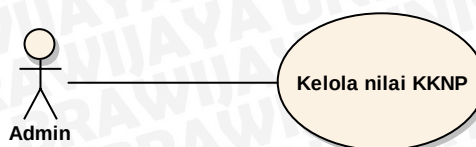
Use case ini digolongkan sebagai fitur **lihat data** dari tabel tbl_kknpp_pengajuan dengan tampilan tabel sederhana. Fitur ini menyediakan opsi untuk melakukan operasi perbarui data pada tabel utama dan operasi tambah, perbarui atau hapus data pada tabel cabang.

3. Use Case Specification Kelola Nilai KKNP

1. Brief Description

Use case ini menjelaskan bagaimana **admin** mengelola data nilai KKNP.

2. Use-Case Diagram



3. Preconditions

- **Admin** telah melakukan *login* ke dalam PTIIK-Apps
- Dibutuhkan jaringan internet untuk menjalankan *use case*
- **Admin** telah menjalankan *use case* **Kelola Data KKNP**

4. Basic Flow of Events

4.1 Mengakses Fungsi Kelola Nilai KKNP

Use case ini diawali ketika aktor **admin** mengakses fungsi kelola nilai KKNP.

Sistem menampilkan detail data KKNP dan form untuk mengelola data nilai KKNP berdasarkan data KKNP yang dipilih oleh **admin**.

4.2 Memanipulasi Data Nilai KKNP

Admin melakukan manipulasi data nilai KKNP.

4.3 Simpan Hasil Perubahan

Admin menekan tombol simpan hasil perubahan data nilai KKNP.

Sistem akan melakukan validasi terhadap form data nilai KKNP yang telah diperbarui oleh **admin**, jika form data nilai KKNP tidak valid maka akan menjalankan **Alternative Flow 5.1**.

Sistem menyimpan hasil perubahan data nilai KKNP.

4.4 Use Case Berakhir

Menjalankan *use case* **Kelola Data KKNP**.

5. Alternative Flows

5.1 Validasi Form Nilai KKNP

5.1.1 Form Nilai KKNP Tidak Valid

Pada **Basic Flow 4.3**, jika form data nilai KKNP tidak valid maka:

1. **Sistem** akan memberikan informasi bagian form yang tidak valid
2. Menjalankan proses **Basic Flow 4.2**.

6. Sub Flows

Tidak ada

7. Key Scenarios

7.1 Success Scenario

Kelola Nilai KKNP – *Basic Flow*

7.2 Failure Scenario

Kelola Nilai KKNP – *Basic Flow, Alternative Flow: Form Nilai KKNP Tidak Valid*

8. Postconditions

- Data nilai KKNP dapat diperbarui sesuai dengan keinginan **admin**

9. Extension Points

Tidak ada

10. Special Requirement

Tidak ada

11. Analisis Fitur

Use case ini digolongkan sebagai fitur **perbarui data**, karena pada fitur tersebut terjadi proses perbarui (tambah, ubah dan hapus) data nilai KKNP pada tabel cabang (tbl_kkn_p_nila) berdasarkan tabel utama (tbl_kkn_p_pengajuan).

4. Use Case Specification Kelola Logbook KKNP

1. Brief Description

Use case ini menjelaskan bagaimana **admin** mengelola data *logbook* KKNP.

2. Use-Case Diagram



3. Preconditions

- **Admin** telah melakukan *login* ke dalam PTIIK-Apps
- Dibutuhkan jaringan internet untuk menjalankan *use case*
- **Admin** telah menjalankan *use case* **Kelola Data KKNP**

4. Basic Flow of Events

4.1 Mengakses Fungsi Kelola Logbook KKNP

Use case ini diawali ketika aktor **admin** mengakses fungsi kelola *logbook* KKNP.

Sistem menampilkan detail data KKNP dan daftar data *logbook* KKNP berdasarkan data KKNP yang dipilih oleh **admin**.

4.2 Menambah Data Logbook KKNP

Admin dapat menekan tombol tambah *logbook* KKNP.

Sistem menampilkan form tambah *logbook* KKNP.

4.3 Mengisi Form Tambah Logbook KKNP

Admin mengisi data pada form tambah *logbook* KKNP.

4.4 Simpan Hasil Perubahan

Admin menekan tombol simpan hasil perubahan data *logbook* KKNP.

Sistem akan melakukan validasi terhadap form data *logbook* KKNP yang telah diperbarui oleh **admin**, jika form data *logbook* KKNP tidak valid maka akan menjalankan **Alternative Flow 5.1**.

Sistem menyimpan hasil perubahan data *logbook* KKNP.

4.5 Use Case Berakhir

Menjalankan *use case* **Kelola Data KKNP**.

5. Alternative Flows

5.1 Validasi Form KKNP

5.1.1 Form Data *Logbook* KKNP Tidak Valid

Pada **Basic Flow 4.3**, jika form data *logbook* KKNP tidak valid maka:

1. **Sistem** akan memberikan informasi bagian form yang tidak valid
2. Menjalankan proses **Basic Flow 4.2**.

6. Sub Flows

Tidak ada

7. Key Scenarios

7.1 Success Scenario

Kelola *Logbook* KKNP – *Basic Flow*

7.2 Failure Scenario

Kelola *Logbook* KKNP – *Basic Flow, Alternative Flow: Form Data Logbook KKNP Tidak Valid*

8. Postconditions

- Data *logbook* KKNP dapat dikelola sesuai dengan keinginan **admin**

9. Extension Points

Tidak ada

10. Special Requirement

Tidak ada

11. Analisis Fitur

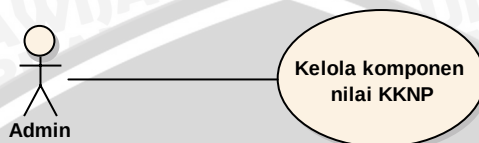
Use case ini digolongkan sebagai fitur **perbarui data**, karena pada fitur tersebut terjadi proses perbarui (tambah dan ubah) data *logbook* KKNP tabel cabang (tbl_kknp_logbook) berdasarkan tabel utama (tbl_kknp_pengajuan).

5. Use Cases Specification Kelola Komponen Nilai KKNP

1. Brief Description

Use case ini menjelaskan bagaimana **admin** mengelola data komponen nilai KKNP.

2. Use-Case Diagram



3. Preconditions

- **Admin** telah melakukan *login* ke dalam PTIIK-Apps
- Dibutuhkan jaringan internet untuk menjalankan *use case*

4. Basic Flow of Events

4.1 Mengakses Fungsi Kelola Komponen Nilai KKNP

Use case ini diawali ketika aktor **admin** mengakses fungsi kelola komponen nilai KKNP.

Sistem menampilkan data komponen nilai KKNP beserta opsi tambah, ubah dan hapus data.

4.2 Melakukan Operasi Data Komponen Nilai KKNP

Admin melakukan operasi terhadap data komponen nilai KKNP, operasi yang bisa dilakukan admin adalah tambah, ubah dan hapus data.

- Jika **admin** melakukan operasi tambah, maka akan menjalankan **Sub Flow 6.1**.
- Jika **admin** melakukan operasi ubah, maka akan menjalankan **Sub Flow 6.2**.
- Jika **admin** melakukan operasi hapus, maka akan menjalankan **Sub Flow 6.3**.

4.3 Memperbarui Data Komponen Nilai KKNP

Sistem memperbarui tampilan data komponen nilai KKNP sesuai dengan operasi yang dilakukan admin.

4.4 Use Case Berakhir

Use case berakhir.

5. Alternative Flows

5.1 Validasi Form Komponen Nilai KKNP

5.1.1 Form Komponen Nilai KKNP Tidak Valid

Pada **Basic Flow 4.2**, ketika admin melakukan operasi terhadap data komponen nilai KKNP. Jika form komponen nilai KKNP tidak valid maka:

1. **Sistem** akan memberikan informasi bagian form yang tidak valid
2. Melanjutkan proses pada saat pemanggilan **alternative flow** ini.

6. Sub Flows

6.1 Tambah Komponen Nilai KKNP

1. Menjalankan Fungsi Tambah Komponen Nilai KKNP

Admin menekan opsi tambah data komponen nilai KKNP.

2. Menampilkan Form Tambah Komponen Nilai KKNP

Sistem menampilkan form tambah komponen nilai KKNP.

3. Mengisi Form Tambah Komponen Nilai KKNP

Admin mengisi form tambah komponen nilai KKNP.

4. Menyimpan Data Komponen Nilai KKNP

Admin menekan tombol simpan data komponen nilai KKNP.

Sistem melakukan validasi terhadap form tambah komponen nilai KKNP, jika form tidak valid maka akan menjalankan **Alternative Flow 5.1**.

Sistem menyimpan data komponen nilai KKNP.

5. Sub Flow Berakhir

Menjalankan **Basic Flow 4.1**.

6.2 Perbarui Komponen Nilai KKNP

1. Menjalankan Fungsi Perbarui Komponen Nilai KKNP

Admin memilih data komponen nilai KKNP yang ingin diperbarui.

2. Menampilkan Form Perbarui Komponen Nilai KKNP

Sistem menampilkan form perbarui komponen nilai KKNP.

3. Mengisi Form Perbarui Komponen Nilai KKNP

Admin melakukan manipulasi dengan memperbarui form komponen nilai KKNP.

4. Menyimpan Data Komponen Nilai KKNP

Admin menekan tombol simpan hasil pembaruan data komponen nilai KKNP.

Sistem melakukan validasi terhadap form komponen nilai KKNP, jika form tidak valid maka akan menjalankan **Alternative Flow 5.1**.

Sistem menyimpan data komponen nilai KKNP.

5. *Sub Flow* Berakhir

Menjalankan **Basic Flow 4.1**.

6.3 Hapus Komponen Nilai KKNP

1. Menjalankan Fungsi Hapus Komponen Nilai KKNP

Admin memilih data komponen nilai KKNP yang ingin diperbarui.

2. Menampilkan Konfirmasi Hapus Data

Sistem menampilkan konfirmasi hapus data komponen nilai KKNP.

3. Konfirmasi Hapus Data

Jika **admin** tidak menyetujui konfirmasi hapus data, maka **Sub Flow** ini berakhir.

Admin melakukan konfirmasi hapus data komponen nilai KKNP.

4. Menghapus Data Komponen Nilai KKNP

Sistem menghapus data komponen nilai KKNP.

5. *Sub Flow* Berakhir

Menjalankan **Basic Flow 4.1**.

7. Key Scenarios

7.1 Success Scenario

Kelola Komponen Nilai KKNP – *Basic Flow*

7.2 Failure Scenario

Kelola Komponen Nilai KKNP – *Basic Flow, Alternative Flow*: Form Komponen Nilai KKNP Tidak Valid

8. Postconditions

- Data komponen nilai KKNP dapat dikelola sesuai dengan keinginan **admin**

9. Extension Points

Tidak ada

10. Special Requirement

Tidak ada

11. Analisis Fitur

Use case ini digolongkan sebagai fitur **lihat data** dari tabel `tbl_kknp_nilai_komponen` dengan tampilan tabel sederhana.

Use case ini juga menyediakan opsi untuk melakukan operasi data tambah, perbarui dan hapus data komponen nilai.

Sub Flow 6.1 digolongkan sebagai fitur **tambah data**.

Sub Flow 6.2 digolongkan sebagai fitur **perbarui data**.

Sub Flow 6.3 digolongkan sebagai fitur **hapus data**.

Glossary

Glossary modul KKNP menjelaskan istilah – istilah yang dipakai pada penjelasan *use specification* modul KKNP. *Glossary* modul KKNP dijelaskan pada Tabel 5.1.

Tabel 5.1 Glossary Modul KKNP

- **Form pengajuan KKNP:** form yang digunakan untuk proses pengajuan KKNP, form tersebut berisi masukan: 1) universitas cabang mahasiswa, 2) fakultas mahasiswa, 3) semester pengajuan, 4) nama dan alamat perusahaan, 5) tanggal pelaksanaan KKNP, 6) daftar mahasiswa, 7) pembimbing lapangan dan dosen pembimbing KKNP, 8) tanggal pengumpulan laporan, 9) status pengerjaan, 10) objek KKNP, 11) catatan, 12) tugas dosen, 13) surat pengantar perusahaan, 14) surat balasan, 15) surat pembatalan.
- **Data KKNP:** data KKNP berisi informasi – informasi yang meliputi: 1) universitas cabang mahasiswa, 2) fakultas mahasiswa, 3) semester pengajuan, 4) nama dan alamat perusahaan, 5) tanggal pelaksanaan KKNP, 6) daftar mahasiswa, 7) pembimbing lapangan dan dosen pembimbing KKNP, 8) tanggal pengumpulan laporan, 9) status pengerjaan, 10) objek KKNP, 11) catatan, 12) tugas dosen, 13) surat pengantar perusahaan, 14) surat balasan, 15) surat pembatalan.
- **Data nilai KKNP:** data nilai KKNP berisi informasi: 1) komponen nilai KKNP, 2) nilai KKNP, 3) nama mahasiswa, 4) pembimbing KKNP memberikan nilai.
- **Data logbook KKNP:** data *logbook* KKNP berisi informasi: 1) pembimbing KKNP, 2) nama kegiatan *logbook*, 3) tanggal kegiatan, 4) catatan.
- **Data komponen nilai KKNP:** adalah data yang digunakan untuk proses kelola komponen nilai KKNP data yang berisi informasi nama komponen nilai KKNP.
- **Form tambah komponen nilai KKNP:** form masukan untuk proses kelola komponen nilai KKNP yang meliputi: 1) keterangan komponen, 2) bobot nilai, 3) nomor urut, 4) data cabang komponen, 5) parameter aktivasi komponen nilai KKNP.

Fitur – fitur modul KKNP telah dijabarkan dengan menggambarkan *use case diagram* dan *use scenario*. Fitur modul KKNP akan dianalisis berdasarkan komponen penyusun modul tersebut. Satu komponen modul didefinisikan sebagai satu unit komponen *controller*, satu atau lebih komponen *model* dan *view*.

Analisis fitur modul KKNP didefinisikan berdasarkan kesamaan dan perbedaan fitur dari sudut pandang aktor yang telah didefinisikan pada sub-bab 4.2.2 identifikasi aktor, yaitu pengembang. Kesamaan dan perbedaan fitur didefinisikan menjadi beberapa klasifikasi fitur, yaitu:

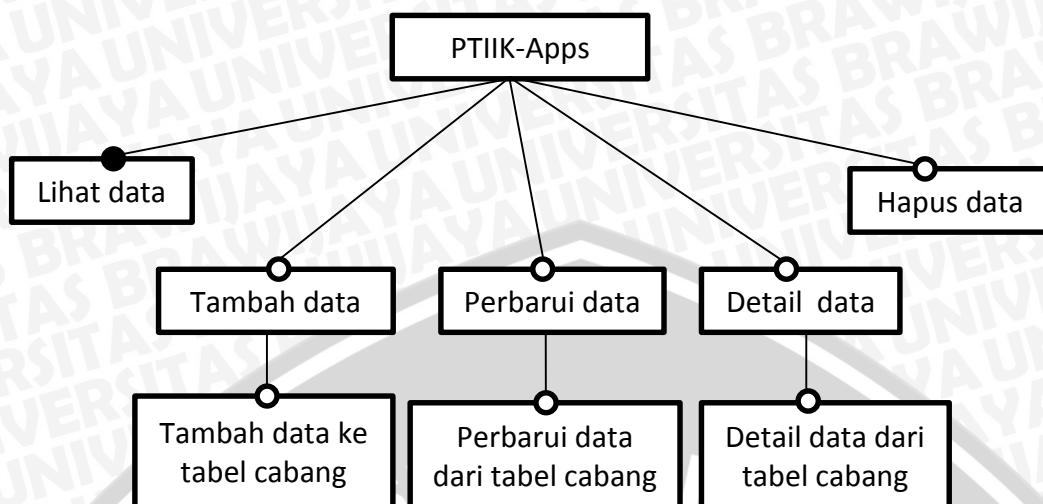
- Lihat data: untuk menampilkan daftar data dari tabel basis data dengan beberapa atribut data, beserta opsi untuk mengelola data tersebut.
- Detail data: untuk menampilkan detail data dari tabel basis data dengan atribut data yang lebih lengkap.
- Detail data dari tabel cabang: untuk menampilkan detail data dari tabel cabang yang berhubungan dengan tabel yang bersangkutan
- Tambah data: untuk menambah data ke tabel basis data.
- Tambah data ke tabel cabang: untuk menambah data ke tabel basis data yang berhubungan dengan tabel utama.
- Perbarui data: untuk memperbarui data dari tabel basis data.
- Perbarui data ke tabel cabang: untuk memperbarui data dari tabel basis data yang berhubungan dengan tabel utama.
- Hapus data: untuk menghapus data dari tabel basis data.

Analisis fungsionalitas modul KKNP dijabarkan kedalam tabel matriks fungsi pada Tabel 5.2.

Tabel 5.2 Matriks Fungsionalitas Modul KKNP

Fitur / Komponen	Pengajuan KKNP	Kelola data KKNP	Kelola nilai KKNP	Kelola <i>logbook</i> KKNP	Kelola komponen nilai KKNP
Lihat data	-	v	v	v	v
Detail data	-	v	v	v	-
Detail data dari tabel cabang	-	v	v	v	-
Tambah data	v	-	-	v	v
Tambah data ke tabel cabang	v	-	v	-	-
Perbarui data	-	v	-	-	v
Perbarui data dari tabel cabang	-	v	v	-	-
Hapus data	-	v	-	-	v

Model fitur modul KKNP dibentuk dari proses analisis matriks fungsionalitas modul KKNP pada Tabel 5.2. Model fitur modul KKNP dipaparkan pada Gambar 5.6.



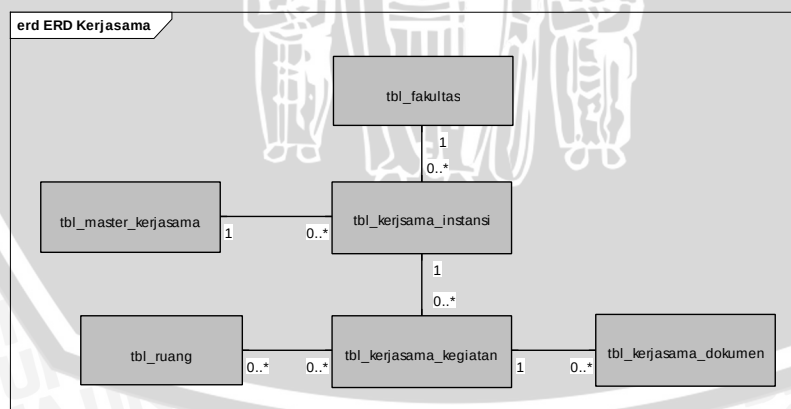
Gambar 5.6 Model Fitur Modul KKNP

Model fitur modul KKNP menjelaskan bahwa setiap komponen pada modul tersebut pasti memiliki fitur untuk melihat data. Fitur tambah, perbarui, detail dan hapus data merupakan fitur opsional. Fitur tambah, detail dan perbarui data memiliki fitur opsional dimana operasi tersebut juga bisa dilakukan kepada *entity*/tabel yang berhubungan dengan tabel yang bersangkutan.

Analisis modul kerjasama

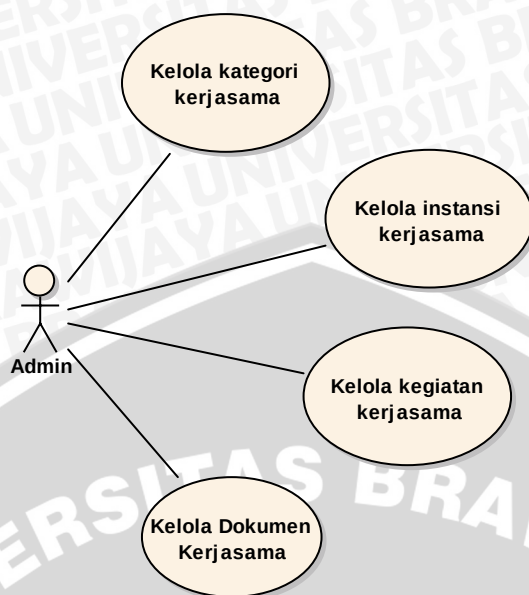
Modul kerjasama adalah modul yang digunakan untuk mengelola data kerjasama antara fakultas dengan instansi lain. Modul tersebut mengelola data kategori kerjasama, instansi kerjasama, kegiatan dan dokumen kerjasama.

Gambar 5.7 adalah *entity relationship diagram* yang menggambarkan hubungan antar *entity* pada modul kerjasama.



Gambar 5.7 Entity Relationship Diagram Modul Kerjasama

Fungsionalitas modul kerjasama digambarkan ke dalam diagram *use case* pada Gambar 5.8.



Gambar 5.8 Diagram *Use Case* Modul Kerjasama

***Use Case Specification* Modul Kerjasama**

Use case specification pada modul kerjasama ditunjukkan pada paparan berikut ini. *Use case specification* digunakan untuk menjabarkan detail dari *use case* dan sebagai tambahan juga dijelaskan perilaku *use case* terhadap tabel basis data.

1. *Use Case Specification* Kelola Kategori Kerjasama

1. Brief Description

Use case ini menjelaskan bagaimana **admin** mengelola data kategori kerjasama.

2. *Use-Case Diagram*



3. *Preconditions*

- **Admin** telah melakukan *login* ke dalam PTIIK-Apps
- Dibutuhkan jaringan internet untuk menjalankan *use case*

4. *Basic Flow of Events*

4.1 Mengakses Fungsi Kelola Kategori Kerjasama

Use case ini diawali ketika aktor **admin** mengakses fungsi kelola kategori kerjasama.

Sistem menampilkan data kategori kerjasama beserta opsi tambah, ubah dan hapus data.

4.2 Melakukan Operasi Data Kategori Kerjasama

Admin melakukan operasi terhadap data kategori kerjasama, operasi yang bisa dilakukan admin adalah tambah, ubah dan hapus data.

- Jika **admin** melakukan operasi tambah, maka akan menjalankan **Sub Flow 6.1**.
- Jika **admin** melakukan operasi ubah, maka akan menjalankan **Sub Flow 6.2**.
- Jika **admin** melakukan operasi hapus, maka akan menjalankan **Sub Flow 6.3**.

4.3 Memperbarui Data Kategori Kerjasama

Sistem memperbarui tampilan data kategori kerjasama sesuai dengan operasi yang dilakukan admin.

4.4 Use Case Berakhir

Use case berakhir.

5. Alternative Flows

5.1 Validasi Form Kategori Kerjasama

5.1.1 Form Kategori Kerjasama Tidak Valid

Pada **Basic Flow 4.2**, ketika admin melakukan operasi terhadap data kategori kerjasama. Jika form kategori kerjasama tidak valid maka:

1. **Sistem** akan memberikan informasi bagian form yang tidak valid
2. Melanjutkan proses pada saat pemanggilan **alternative flow** ini

6. Sub Flows

6.1 Tambah Kategori Kerjasama

1. Menjalankan Fungsi Tambah Kategori Kerjasama

Admin menekan opsi tambah kategori kerjasama.

2. Menampilkan Form Tambah Kategori Kerjasama

Sistem menampilkan form tambah kategori kerjasama.

3. Mengisi Form Tambah Kategori Kerjasama

Admin mengisi form tambah kategori kerjasama.

4. Menyimpan Data Kategori Kerjasama

Admin menekan tombol simpan data kategori kerjasama.

Sistem melakukan validasi terhadap form tambah kategori kerjasama, jika form tidak valid maka akan menjalankan **Alternative Flow 5.1**.

Sistem menyimpan data kategori kerjasama.

5. Sub Flow Berakhir

Menjalankan **Basic Flow 4.1**.

6.2 Perbarui Kategori Kerjasama

1. Menjalankan Fungsi Perbarui Kategori Kerjasama

Admin memilih data kategori kerjasama yang ingin diperbarui.

2. Menampilkan Form Perbarui Kategori Kerjasama

Sistem menampilkan form perbarui kategori kerjasama.

3. Mengisi Form Perbarui Kategori Kerjasama

Admin melakukan manipulasi dengan memperbarui form kategori kerjasama.

4. Menyimpan Data Kategori Kerjasama

Admin menekan tombol simpan hasil pembaruan data kategori kerjasama.

Sistem melakukan validasi terhadap form kategori kerjasama, jika form tidak valid maka akan menjalankan **Alternative Flow 5.1**.

Sistem menyimpan data kategori kerjasama.

5. Sub Flow Berakhir

Menjalankan **Basic Flow 4.1**.

6.3 Hapus Kategori Kerjasama

1. Menjalankan Fungsi Hapus Kategori Kerjasama

Admin memilih data kategori kerjasama yang ingin dihapus.

2. Menampilkan Konfirmasi Hapus Data

Sistem menampilkan konfirmasi hapus data kategori kerjasama.

3. Konfirmasi Hapus Data

Jika **admin** tidak menyetujui konfirmasi hapus data, maka **Sub Flow** ini berakhir.

Admin melakukan konfirmasi hapus data kategori kerjasama.

4. Menghapus Data Kategori Kerjasama

Sistem menghapus data kategori kerjasama.

5. Sub Flow Berakhir

Menjalankan **Basic Flow 4.1**.

7. Key Scenarios

7.1 Success Scenario

Kelola Kategori Kerjasama – *Basic Flow*

7.2 Failure Scenario

Kelola Kategori Kerjasama – *Basic Flow, Alternative Flow*: Form Kategori Kerjasama Tidak Valid

8. Postconditions

- Data kategori kerjasama dapat dikelola sesuai dengan keinginan **admin**

9. Extension Points

Tidak ada

10. Special Requirement

Tidak ada

11. Analisis Fitur

Use case ini digolongkan sebagai fitur **lihat data** dengan tampilan tabel sederhana. Fitur ini menyediakan opsi untuk melakukan proses tambah, perbarui dan hapus data dari tbl_master_kerjasama.

Sub Flow 6.1 digolongkan sebagai fitur **tambah data**.

Sub Flow 6.2 digolongkan sebagai fitur **perbarui data**.

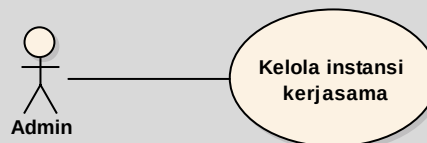
Sub Flow 6.3 digolongkan sebagai fitur **hapus data**.

2. Use Case Specification Kelola Instansi Kerjasama

1. Brief Description

Use case ini menjelaskan bagaimana **admin** mengelola data instansi kerjasama.

2. Use-Case Diagram



3. Preconditions

- Admin** telah melakukan *login* ke dalam PTIIK-Apps
- Dibutuhkan jaringan internet untuk menjalankan *use case*

4. Basic Flow of Events

4.1 Mengakses Fungsi Kelola Instansi Kerjasama

Use case ini diawali ketika aktor **admin** mengakses fungsi kelola instansi kerjasama.

Sistem menampilkan data instansi kerjasama beserta opsi tambah, ubah dan hapus data.

4.2 Melakukan Operasi Data Instansi Kerjasama

Admin melakukan operasi terhadap data instansi kerjasama, operasi yang bisa dilakukan admin adalah tambah, ubah dan hapus data.

- Jika **admin** melakukan operasi tambah, maka akan menjalankan **Sub Flow 6.1**.
- Jika **admin** melakukan operasi ubah, maka akan menjalankan **Sub Flow 6.2**.
- Jika **admin** melakukan operasi hapus, maka akan menjalankan **Sub Flow 6.3**.
- Jika **admin** melakukan operasi sortir data, maka akan menjalankan **Sub Flow 6.4**.

4.3 Memperbarui Data Instansi Kerjasama

Sistem memperbarui tampilan data instansi kerjasama sesuai dengan operasi yang dilakukan admin.

4.4 Use Case Berakhir

Use case berakhir.

5. Alternative Flows

5.1 Validasi Form Instansi Kerjasama

5.1.1 Form Instansi Kerjasama Tidak Valid

Pada **Basic Flow 4.2**, ketika admin melakukan operasi terhadap data instansi kerjasama. Jika form instansi kerjasama tidak valid maka:

1. **Sistem** akan memberikan informasi bagian form yang tidak valid
2. Melanjutkan proses pada saat pemanggilan **alternative flow** ini

6. Sub Flows

6.1 Tambah Instansi Kerjasama

1. Menjalankan Fungsi Tambah Instansi Kerjasama

Admin menekan opsi tambah instansi kerjasama.

2. Menampilkan Form Tambah Instansi Kerjasama

Sistem menampilkan form tambah instansi kerjasama.

3. Mengisi Form Tambah Instansi Kerjasama

Admin mengisi form tambah instansi kerjasama.

4. Menyimpan Data Instansi Kerjasama

Admin menekan tombol simpan data instansi kerjasama.

Sistem melakukan validasi terhadap form tambah instansi kerjasama, jika form tidak valid maka akan menjalankan **Alternative Flow 5.1**.

Sistem menyimpan data instansi kerjasama.

5. Sub Flow Berakhir

Menjalankan **Basic Flow 4.1**.

6.2 Perbarui Instansi Kerjasama

1. Menjalankan Fungsi Perbarui Instansi Kerjasama

Admin memilih data instansi kerjasama yang ingin diperbarui.

2. Menampilkan Form Perbarui Instansi Kerjasama

Sistem menampilkan form perbarui instansi kerjasama.

3. Mengisi Form Perbarui Instansi Kerjasama

Admin melakukan manipulasi dengan memperbarui form instansi kerjasama.

4. Menyimpan Data Instansi Kerjasama

Admin menekan tombol simpan hasil pembaruan data instansi kerjasama.

Sistem melakukan validasi terhadap form instansi kerjasama, jika form tidak valid maka akan menjalankan **Alternative Flow 5.1**.

Sistem menyimpan data instansi kerjasama.

5. Sub Flow Berakhir

Menjalankan **Basic Flow 4.1**.

6.3 Hapus Instansi Kerjasama

1. Menjalankan Fungsi Hapus Instansi Kerjasama

Admin memilih data instansi kerjasama yang ingin dihapus.

2. Menampilkan Konfirmasi Hapus Data

Sistem menampilkan konfirmasi hapus data instansi kerjasama.

3. Konfirmasi Hapus Data

Jika **admin** tidak menyetujui konfirmasi hapus data, maka **Sub Flow** ini berakhir.

Admin melakukan konfirmasi hapus data instansi kerjasama.

4. Menghapus Data Instansi Kerjasama

Sistem menghapus data instansi kerjasama.

5. Sub Flow Berakhir

Menjalankan **Basic Flow 4.1**.

6.4 Sortir Data Instansi Kerjasama

1. Mengisi Form Sortir Data Instansi Kerjasama

Admin mengisi parameter sortir pada form sortir data instansi kerjasama.

2. Menyortir Data Instansi Kerjasama

Admin menekan tombol sortir data instansi kerjasama.

Sistem menyortir data instansi kerjasama berdasarkan parameter yang dikirimkan oleh **Admin**.

3. Sub Flow Berakhir

Menjalankan **Basic Flow 4.1**.

7. Key Scenarios

7.1 Success Scenario

Kelola Instansi Kerjasama – *Basic Flow*

7.2 Failure Scenario

Kelola Instansi Kerjasama – *Basic Flow, Alternative Flow*: Form Instansi Kerjasama Tidak Valid

8. Postconditions

- Data instansi kerjasama dapat dikelola sesuai dengan keinginan **admin**

9. Extension Points

Tidak ada

10. Special Requirement

- Data kategori kerjasama tidak boleh kosong, karena pada saat proses tambah dan perbarui instansi kerjasama dibutuhkan referensi data kategori kerjasama.

11. Analisis Fitur

Use case ini digolongkan sebagai fitur **lihat data** dengan tampilan tabel **sortir data**, sehingga pengguna sistem bisa melakukan sortir data dari tbl_kerjasama_instansi berdasarkan tbl_master_kerjasama dan tbl_fakultas. Fitur ini menyediakan opsi untuk melakukan proses tambah, perbarui dan hapus data dari tbl_kerjasama_instansi.

Sub Flow 6.1 digolongkan sebagai fitur **tambah data**.

Sub Flow 6.2 digolongkan sebagai fitur **perbarui data**.

Sub Flow 6.3 digolongkan sebagai fitur **hapus data**.

3. Use Case Specification Kelola Kegiatan Kerjasama

1. Brief Description

Use case ini menjelaskan bagaimana **admin** mengelola data kegiatan kerjasama.

2. Use-Case Diagram



3. Preconditions

- **Admin** telah melakukan *login* ke dalam PTIIK-Apps
- Dibutuhkan jaringan internet untuk menjalankan *use case*

4. Basic Flow of Events

4.1 Mengakses Fungsi Kelola Kegiatan Kerjasama

Use case ini diawali ketika aktor **admin** mengakses fungsi kelola kegiatan kerjasama.

Sistem menampilkan data kegiatan kerjasama beserta opsi tambah, ubah dan hapus data.

4.2 Melakukan Operasi Data Kegiatan Kerjasama

Admin melakukan operasi terhadap data kegiatan kerjasama, operasi yang bisa dilakukan admin adalah tambah, ubah dan hapus data.

- Jika **admin** melakukan operasi tambah, maka akan menjalankan **Sub Flow 6.1**.
- Jika **admin** melakukan operasi ubah, maka akan menjalankan **Sub Flow 6.2**.
- Jika **admin** melakukan operasi hapus, maka akan menjalankan **Sub Flow 6.3**.
- Jika **admin** melakukan operasi sortir data, maka akan menjalankan **Sub Flow 6.4**.
- Pada tahap ini **Admin** dapat menjalankan *use case* **Kelola Dokumen Kerjasama**.

4.3 Memperbarui Data Kegiatan Kerjasama

Sistem memperbarui tampilan data kegiatan kerjasama sesuai dengan operasi yang dilakukan admin.

4.4 Use Case Berakhir

Use case berakhir.

5. Alternative Flows

5.1 Validasi Form Kegiatan Kerjasama

5.1.1 Form Kegiatan Kerjasama Tidak Valid

Pada **Basic Flow 4.2**, ketika admin melakukan operasi terhadap data kegiatan kerjasama. Jika form kegiatan kerjasama tidak valid maka:

1. **Sistem** akan memberikan informasi bagian form yang tidak valid
2. Melanjutkan proses pada saat pemanggilan **alternative flow** ini6. **Sub Flows**

6.1 Tambah Kegiatan Kerjasama

1. Menjalankan Fungsi Tambah Kegiatan Kerjasama

Admin menekan opsi tambah kegiatan kerjasama.

2. Menampilkan Form Tambah Kegiatan Kerjasama

Sistem menampilkan form tambah kegiatan kerjasama.

3. Mengisi Form Tambah Kegiatan Kerjasama

Admin mengisi form tambah kegiatan kerjasama.

4. Menyimpan Data Kegiatan Kerjasama

Admin menekan tombol simpan data kegiatan kerjasama.

Sistem melakukan validasi terhadap form tambah kegiatan kerjasama, jika form tidak valid maka akan menjalankan **Alternative Flow 5.1**.

Sistem menyimpan data kegiatan kerjasama.

5. Sub Flow Berakhir

Menjalankan **Basic Flow 4.1**.

6.2 Perbarui Kegiatan Kerjasama

1. Menjalankan Fungsi Perbarui Kegiatan Kerjasama

Admin memilih data kegiatan kerjasama yang ingin diperbarui.

2. Menampilkan Form Perbarui Kegiatan Kerjasama

Sistem menampilkan form perbarui kegiatan kerjasama.

3. Mengisi Form Perbarui Kegiatan Kerjasama

Admin melakukan manipulasi dengan memperbarui form kegiatan kerjasama.

4. Menyimpan Data Kegiatan Kerjasama

Admin menekan tombol simpan hasil pembaruan data kegiatan kerjasama.

Sistem melakukan validasi terhadap form kegiatan kerjasama, jika form tidak valid maka akan menjalankan **Alternative Flow 5.1**.

Sistem menyimpan data kegiatan kerjasama.

5. Sub Flow Berakhir

Menjalankan **Basic Flow 4.1**.

6.3 Hapus Kegiatan Kerjasama

1. Menjalankan Fungsi Hapus Kegiatan Kerjasama

Admin memilih data kegiatan kerjasama yang ingin dihapus.

2. Menampilkan Konfirmasi Hapus Data

Sistem menampilkan konfirmasi hapus data kegiatan kerjasama.

3. Konfirmasi Hapus Data

Jika **admin** tidak menyetujui konfirmasi hapus data, maka **Sub Flow** ini berakhir.

Admin melakukan konfirmasi hapus data kegiatan kerjasama.

4. Menghapus Data Kegiatan Kerjasama

Sistem menghapus data kegiatan kerjasama.

5. Sub Flow Berakhir

Menjalankan **Basic Flow 4.1**.

6.4 Sortir Data Kegiatan Kerjasama

1. Mengisi Form Sortir Data Kegiatan Kerjasama

Admin mengisi parameter sortir pada form sortir data kegiatan kerjasama.

2. Menyortir Data Kegiatan Kerjasama

Admin menekan tombol sortir data kegiatan kerjasama.

Sistem menyortir data kegiatan kerjasama berdasarkan parameter yang dikirimkan oleh **Admin**.

3. Sub Flow Berakhir

Menjalankan **Basic Flow 4.1**.

7. Key Scenarios

7.1 Success Scenario

Kelola Kegiatan Kerjasama – Basic Flow

7.2 Failure Scenario

Kelola Kegiatan Kerjasama – Basic Flow, Alternative Flow: Form Kegiatan Kerjasama Tidak Valid

8. Postconditions

- Data kegiatan kerjasama dapat dikelola sesuai dengan keinginan **admin**

9. Extension Points

- Kelola Dokumen Kerjasama

10. Special Requirement

- Data instansi kerjasama tidak boleh kosong, karena ada saat proses tambah dan perbarui kegiatan kerjasama dibutuhkan referensi data instansi kerjasama.

11. Analisis Fitur

Use case ini digolongkan sebagai fitur **lihat data** dengan tampilan **tabel sortir data**, sehingga pengguna sistem bisa melakukan sortir data dari tbl_kerjasama_kegiatan berdasarkan tbl_kerjasama_instansi, tbl_fakultas dan tbl_master_kerjasama.

Fitur ini menyediakan opsi untuk melakukan proses tambah, perbarui dan hapus data dari tbl_kerjasama_instansi.

Sub Flow 6.1 digolongkan sebagai fitur **tambah data**.

Sub Flow 6.2 digolongkan sebagai fitur **perbarui data**.

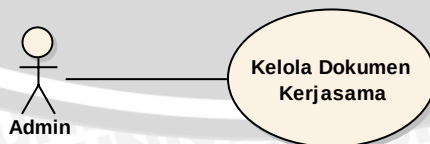
Sub Flow 6.3 digolongkan sebagai fitur **hapus data**.

4. Use Case Specification Kelola Dokumen Kerjasama

1. Brief Description

Use case ini menjelaskan bagaimana **admin** mengelola data dokumen kerjasama.

2. Use-Case Diagram



3. Preconditions

- Admin** telah melakukan *login* ke dalam PTIIK-Apps
- Dibutuhkan jaringan internet untuk menjalankan *use case*

4. Basic Flow of Events

4.1 Mengakses Fungsi Kelola Dokumen Kerjasama

Use case ini diawali ketika aktor **admin** mengakses fungsi kelola dokumen kerjasama.

Sistem menampilkan data dokumen kerjasama beserta opsi tambah dan hapus data.

4.2 Melakukan Operasi Data Dokumen Kerjasama

Admin melakukan operasi terhadap data dokumen kerjasama, operasi yang bisa dilakukan admin adalah tambah dan hapus data.

- Jika **admin** melakukan operasi tambah, maka akan menjalankan **Sub Flow 6.1**.
- Jika **admin** melakukan operasi hapus, maka akan menjalankan **Sub Flow 6.2**.
- Jika **admin** melakukan operasi hapus, maka akan menjalankan **Sub Flow 6.3**.

4.3 Memperbarui Data Dokumen Kerjasama

Sistem memperbarui tampilan data dokumen kerjasama sesuai dengan operasi yang dilakukan admin.

4.4 Use Case Berakhir

Use case berakhir.

5. Alternative Flows

5.1 Validasi Form Dokumen Kerjasama

5.1.1 Form Dokumen Kerjasama Tidak Valid

Pada **Basic Flow 4.2**, ketika admin melakukan operasi terhadap data dokumen kerjasama. Jika form dokumen kerjasama tidak valid maka:

1. **Sistem** akan memberikan informasi bagian form yang tidak valid
2. Melanjutkan proses pada saat pemanggilan **alternative flow** ini

6. Sub Flows

6.1 Tambah Dokumen Kerjasama

1. Menjalankan Fungsi Tambah Dokumen Kerjasama

Admin menekan opsi tambah dokumen kerjasama.

2. Menampilkan Form Tambah Dokumen Kerjasama

Sistem menampilkan form tambah dokumen kerjasama.

3. Mengisi Form Tambah Dokumen Kerjasama

Admin mengisi form tambah dokumen kerjasama.

4. Menyimpan Data Dokumen Kerjasama

Admin menekan tombol simpan data dokumen kerjasama.

Sistem melakukan validasi terhadap form tambah dokumen kerjasama, jika form tidak valid maka akan menjalankan **Alternative Flow 5.1**.

Sistem menyimpan data dokumen kerjasama.

5. Sub Flow Berakhir

Menjalankan **Basic Flow 4.1**.

6.2 Hapus Dokumen Kerjasama

1. Menjalankan Fungsi Hapus Dokumen Kerjasama

Admin memilih data dokumen kerjasama yang ingin diperbarui.

2. Menampilkan Konfirmasi Hapus Data

Sistem menampilkan konfirmasi hapus data dokumen kerjasama.

3. Konfirmasi Hapus Data

Jika **admin** tidak menyetujui konfirmasi hapus data, maka **Sub Flow** ini berakhir.

Admin melakukan konfirmasi hapus data dokumen kerjasama.

4. Menghapus Data Dokumen Kerjasama

Sistem menghapus data dokumen kerjasama.

5. Sub Flow Berakhir

Menjalankan **Basic Flow 4.1**.

7. Key Scenarios

7.1 Success Scenario

Kelola Dokumen Kerjasama – *Basic Flow*

7.2 Failure Scenario

Kelola Dokumen Kerjasama – *Basic Flow, Alternative Flow: Form Dokumen Kerjasama Tidak Valid*

8. Postconditions

- Data dokumen kerjasama dapat dikelola sesuai dengan keinginan **admin**

9. Extension Points

Tidak ada

10. Special Requirement

- Data kegiatan kerjasama tidak boleh kosong, karena ada saat proses tambah dokumen kerjasama dibutuhkan referensi data kegiatan kerjasama.

11. Analisis Fitur

Use case ini digolongkan sebagai fitur **lihat data** dengan tampilan tabel sortir data, sehingga pengguna sistem bisa melakukan sortir data dari tbl_kerjasama_instansi berdasarkan tbl_kerjasama_instansi, tbl_kerjasama_kegiatan, tbl_master_kerjasama dan tbl_fakultas.

Fitur ini menyediakan opsi untuk melakukan proses tambah dan hapus data dari tbl_kerjasama_instansi.

Sub Flow 6.1 digolongkan sebagai fitur **tambah data**.

Sub Flow 6.2 digolongkan sebagai fitur **hapus data**.

Glossary

Glossary modul kerjasam menjelaskan istilah – istilah yang dipakai pada penjelasan *use specification* modul KKNP. *Glossary* modul KKNP dijelaskan pada Tabel 5.3.

Tabel 5.3 Glossary Modul Kerjasama

- **Data kategori kerjasama:** data kategori kerjasama berisi informasi nama kategori.
- **Form tambah kategori kerjasama:** form untuk mengelola data kategori kerjasama yang berisi masukan nama kategori kerjasama.
- **Data instansi kerjasama:** data yang digunakan pada proses kelola instansi kerjasama yang berisi informasi: 1) nama fakultas yang terlibat kerjasama, 2) kategori kerjasama, 3) nama instansi, 4) nomor penanggungjawab kerjasama (*contact person*), 5) telepon instansi, 6) alamat instansi, 7) keterangan instansi, 8) logo instansi.
- **Form tambah instansi kerjasama:** form yang digunakan untuk mengelola data instansi kerjasama yang berisi masukan: 1) nama fakultas yang terlibat kerjasama, 2) kategori kerjasama, 3) nama instansi, 4) nomor penanggungjawab kerjasama (*contact person*), 5) telepon instansi, 6) alamat instansi, 7) keterangan instansi, 8) logo instansi.
- **Data kegiatan kerjasama:** data yang digunakan pada proses kelola kegiatan kerjasama yang berisi informasi: 1) nama instansi kerjasama yang menyelenggarakan kegiatan, 2) judul kegiatan, 3) tanggal pelaksanaan kegiatan, 4) ruang yang dipakai selama kegiatan, 5) lokasi kegiatan, 6) keterangan kegiatan.
- **Form tambah kegiatan kerjasama:** berisi form untuk mengelola data kegiatan kerjasama yang meliputi: 1) nama instansi kerjasama yang

menyelenggarakan kegiatan, 2) judul kegiatan, 3) tanggal pelaksanaan kegiatan, 4) ruang yang dipakai selama kegiatan, 5) lokasi kegiatan, 6) keterangan kegiatan.

- **Data dokumen kerjasama:** data dokumen kerjasama yang berisi informasi: 1) Detail data kegiatan kerjasama, 2) judul dokumen, 3) keterangan dokumen, 4) file dokumen
- **Form tambah dokumen kerjasama:** form untuk mengelola dokumen kerjasama yang berisi masukan: 1) judul dokumen, 2) keterangan dokumen, 3) file dokumen

Fitur – fitur modul kerjasama telah dijabarkan dengan menggambarkan *use case diagram* dan *use scenario*. Fitur modul kerjasama akan dianalisis berdasarkan komponen penyusun modul tersebut. Satu komponen modul didefinisikan sebagai satu unit komponen *controller*, satu atau lebih komponen *model* dan *view*.

Analisis fitur modul kerjasama didefinisikan berdasarkan kesamaan dan perbedaan fitur dari sudut pandang pengembang. Kesamaan dan fitur didefinisikan menjadi beberapa klasifikasi fitur, yaitu:

- Lihat data: untuk menampilkan daftar data dari tabel basis data dengan beberapa atribut data, beserta opsi untuk mengelola data tersebut.
- Sortir data: sama seperti fitur lihat data, namun terdapat form untuk menyortir data yang ditampilkan sesuai dengan masukan pengguna.
- Detail data: untuk menampilkan detail data dari tabel basis data dengan atribut data yang lebih lengkap dan detail data dari tabel cabang yang berhubungan dengan tabel yang bersangkutan.
- Tambah data: untuk menambah data ke tabel basis data.
- Perbarui data: untuk memperbarui data dari tabel basis data.
- Hapus data: untuk menghapus data dari tabel basis data.

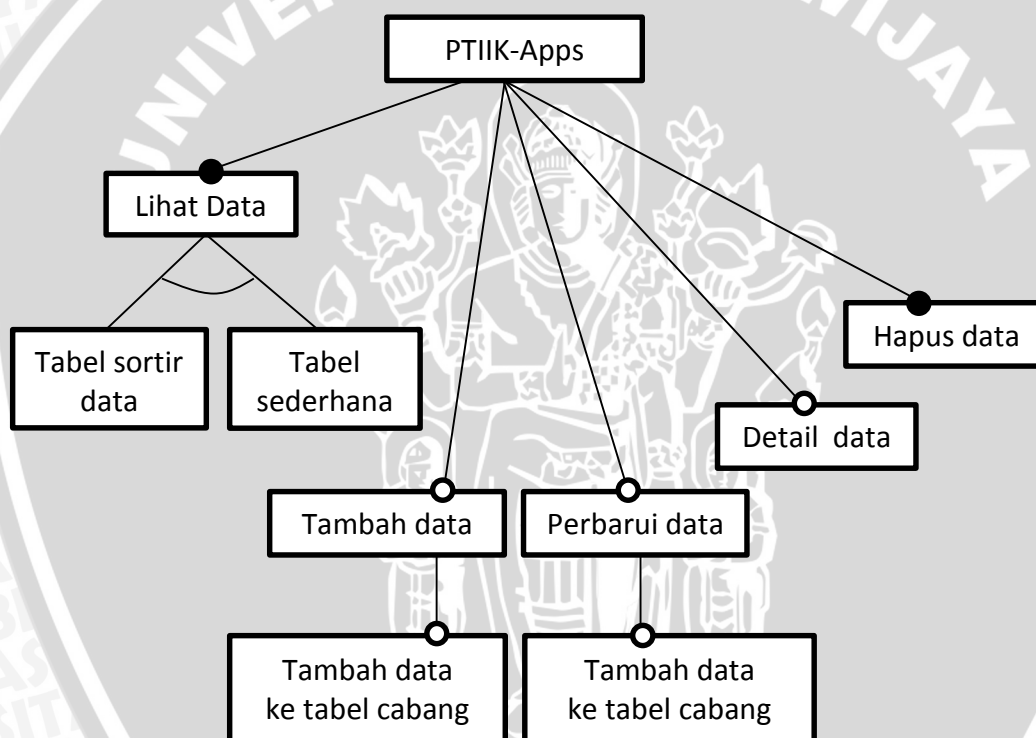
Analisis fungsionalitas modul KKNP dijabarkan kedalam tabel matriks fungsi pada Tabel 5.4. Kolom fitur pada matriks Tabel 5.4 digabungkan dengan kolom fitur dari matriks Tabel 5.2 untuk mengetahui persamaan dan perbedaan klasifikasi fitur antar modul kknp dengan modul kerjasama.

Tabel 5.4 Matriks Fungsionalitas Modul Kerjasama

Fitur / Komponen	Kelola kategori kerjasama	Kelola instansi kerjasama	Kelola kegiatan kerjasama	Kelola dokumen kerjasama
Lihat data	v	-	-	v
Sortir data	-	v	v	-
Detail data	-	-	v	-

Detail data dari tabel cabang	-	-	-	-
Tambah data	v	v	v	-
Tambah data ke tabel cabang	-	-	-	v
Perbarui data	v	v	v	-
Perbarui data dari tabel cabang	-	-	-	v
Hapus data	v	v	v	v

Model fitur modul kerjasama dibentuk dari proses analisis matriks fungsionalitas modul kerjasama pada Tabel 5.4. Model fitur modul kerjasama dipaparkan pada Gambar 5.9.



Gambar 5.9 Modul Fitur Modul Kerjasama

Model fitur modul KKNP menjelaskan bahwa setiap komponen pada modul tersebut pasti memiliki fitur untuk melihat data, dan hapus data. Fitur lihat data memiliki fitur alternatif, yang mana fitur tersebut bisa ditampilkan dengan tabel sortir data atau dengan tabel sederhana data. Fitur tambah dan perbarui data memiliki fitur opsional untuk melakukan operasi pada tabel cabang yang berhubungan.

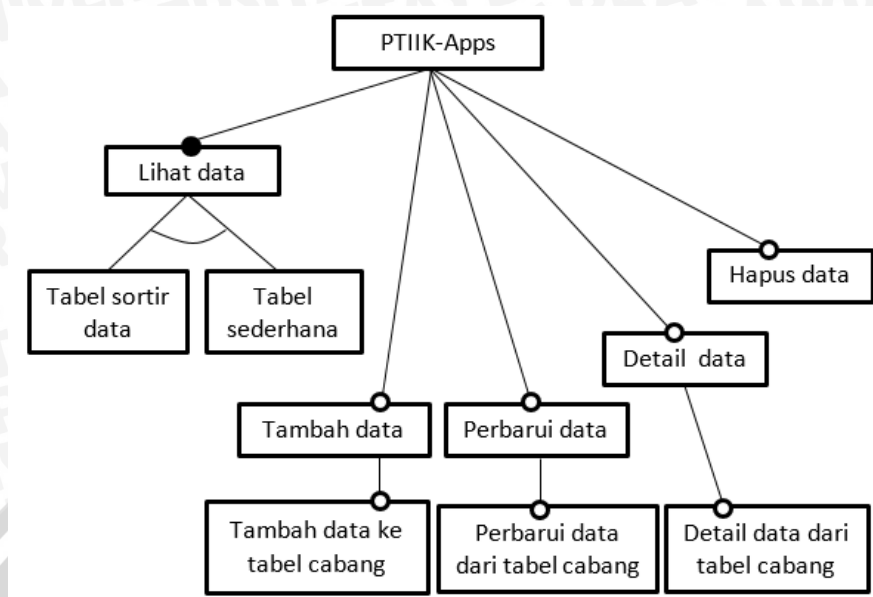
Analisis fitur dilanjutkan dengan membandingkan model fitur pada modul KKNP pada Gambar 5.6 dan modul fitur kerjasama pada Gambar 5.9. Analisis pada kedua model fitur dilakukan untuk mendefinisikan fitur yang bersifat wajib,

alternatif maupun opsional sehingga model fitur umum bisa didefinisikan. Analisis dilakukan dengan menggabungkan matriks fungsionalitas pada modul KKNP dan modul kerjasama yang digambarkan oleh Tabel 5.5.

**Tabel 5.5 Gabungan Matriks Fungsionalitas
Modul KKNP dan Modul Kerjasama**

Fitur / Komponen	Pengaju an KKNP	Kelola data KKNP	Kelola nilai KKNP	Kelola <i>log- book</i> KKNP	Kelola kom- ponen nilai KKNP	Kelola kate- gori kerja- sama	Kelola ins- tansi kerja- sama	Kelola kegi- atan kerja- sama	Kelola doku- men kerja- sama
Lihat data	-	v	v	v	v	v	-	-	v
Sortir data	-	-	-	-	-	-	v	v	-
Detail data	-	v	v	v	-	-	-	v	-
Detail data dari tabel cabang	-	v	v	v	-	-	-	-	-
Tambah data	v	v	-	v	v	v	v	v	-
Tambah data ke tabel cabang	v	v	v	-	-	-	-	-	v
Perbarui data	-	v	-	-	v	v	v	v	-
Perbarui data dari tabel cabang	-	v	v	-	-	-	-	-	v
Hapus data	-	v	-	-	v	v	v	v	v

Dari analisis yang dilakukan pada matriks fungsionalitas gabungan pada Tabel 5.5 menghasilkan model fitur umum modul PTIIK-Apps. Model fitur tersebut menjelaskan komposisi fitur yang ada pada domain. Pada setiap komponen modul PTIIK-Apps pasti memiliki fitur lihat data, yang mana fitur tersebut memiliki alternatif untuk menampilkan tabel sortir data atau tabel sederhana. Fitur tambah, perbarui, detail dan hapus data merupakan fitur opsional dari modul PTIIK-Apps. Fitur tambah, perbarui dan detail data juga memiliki fitur opsional dimana fitur tersebut juga melakukan operasi terhadap tabel cabang yang berhubungan. Model fitur umum modul PTIIK-Apps ditunjukkan pada Gambar 5.10. Penjelasan detail setiap fitur pada Gambar 5.10 dan isu pengambilan keputusan berdasarkan fitur alternatif dijelaskan pada bab Lampiran A.



Gambar 5.10 Model Fitur Umum Modul PTIIK-Apps

5.2.2.2 Pemodelan *entity-relationshipship*

Model ER (*Entity-relationshipship model*) digunakan untuk menggambarkan dan mendefinisikan komponen penyusun domain dan interaksi antar komponen tersebut. Tujuan utama dari model tersebut adalah untuk membantu proses penurunan objek dan definisi data pada proses analisis fungsional dan pemodelan arsitektur domain. Model ER pada domain yang telah didefinisikan ditunjukkan oleh Gambar 5.11. Penjelasan lengkap terkait *entity* dan hubungan antar *entity* didefinisikan pada bab Lampiran A.

Model ER pada Gambar 5.11 didapat dari proses analisis hubungan antar komponen pada domain yang didefinisikan pada proses analisis konteks. Modul KKNP dan modul kerjasama memiliki detail struktur komponen penyusun yang hampir sama. Kedua modul tersebut memiliki kesamaan interaksi unit *controller*, *model* dan *view* dan kesamaan hubungan antara komponen penyusun sistem dengan komponen sistem *framework* PTIIK-Apps.



Komponen *framework* pada PTIIK-Apps memiliki komponen *framework controller*, *framework model*, *config* dan sekumpulan komponen *modul*. *Framework controller* berisi sekumpulan fungsi – fungsi yang digunakan oleh komponen *controller* aplikasi dalam mengelola alur suatu proses. *Framework model* berisi sekumpulan fungsi – fungsi yang digunakan oleh *model* aplikasi dalam mengelola proses pengambilan dan operasi data. Komponen *config* menyediakan variabel global yang bisa diakses oleh komponen – komponen *controller*, *model* dan *view*. Komponen *modul* berfungsi untuk mengelompokkan sekumpulan komponen *controller*, *model* dan *view* menjadi sebuah modul.

96

merupakan spesialisasi dari komponen *framework controller*. Komponen *model* merupakan spesialisasi dari komponen *framework model*.

Komponen penyusun modul terdiri dari *controller*, *model* dan *view*. Komponen *controller* merupakan unit untuk memproses alur data, yang mana membutuhkan komponen *model* untuk menjalankan proses operasi data dan membutuhkan komponen *view* untuk menampilkan proses interaksi tatap muka. Komponen *view* memiliki komponen *javascript* untuk menjalankan proses pada bagian tatap muka tanpa harus memuat ulang komponen *view*.

Cmscontroller merupakan spesialisasi dari komponen *framework controller*, yang berfungsi untuk menjalankan fungsi – fungsi komponen *framework controller* sekaligus mengandung fungsi – fungsi umum yang digunakan oleh *controller*. *Cmsmodule* merupakan spesialisasi dari komponen *cmscontroller* yang mengandung fungsi – fungsi umum modul yang juga bisa digunakan oleh komponen *controller*.

Komponen *model* merupakan spesialisasi dari komponen *framework model*, yang berfungsi untuk menjalankan fungsi – fungsi komponen *model* dan mengandung fungsi – fungsi operasi data yang dibutuhkan oleh komponen *controller*. Komponen *config* menyediakan nilai variabel umum yang bisa digunakan oleh komponen *cmscontroller* dan komponen turunannya.

5.2.2.3 Analisis Fungsional

Analisis fungsional digunakan untuk mengidentifikasi persamaan dan perbedaan fungsional aplikasi pada domain. Analisis fungsional dilakukan dengan menganalisis kesamaan dan perbedaan: 1) aktivitas sistem dalam mengelola masukan dan menghasilkan keluaran 2) perilaku sistem dalam menjalankan fungsinya.

Analisis aktivitas domain dilakukan dengan membandingkan setiap fungsi pada fitur yang telah didefinisikan pada model fitur untuk menghasilkan model fungsional. Pada model fungsional juga dijelaskan proses parameterisasi yang didesain untuk menangani persamaan dan perbedaan fungsi antar fitur yang dibandingkan.

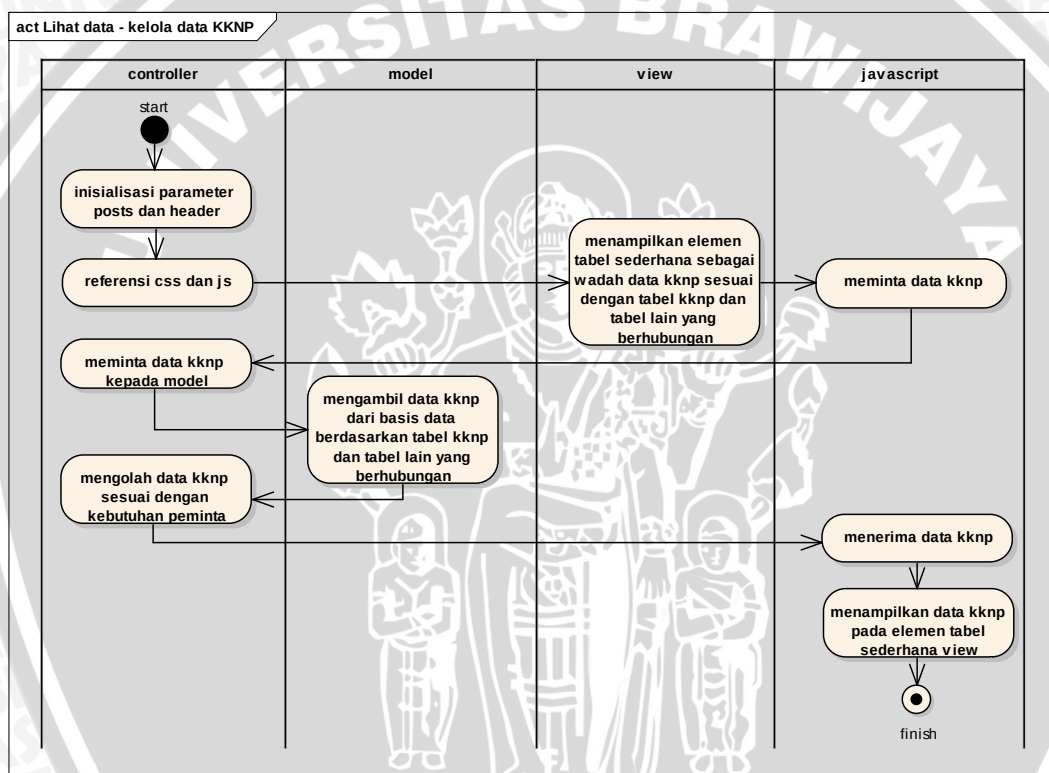
Analisis dilakukan dengan membuat model fitur pada fitur wajib sebagai model fungsional umum. Fitur – fitur opsional dianalisis untuk menghasilkan spesialisasi dari model fungsional umum. Model fungsional digambarkan dengan diagram aktivitas untuk menjelaskan persamaan dan perbedaan pada sudut pandang fungsionalitas. Model fungsional juga digambarkan dengan diagram kondisi untuk menggambarkan persamaan dan perbedaan perilaku sistem.

Diagram aktivitas digambarkan dengan melibatkan komponen yang telah didefinisikan pada model ER sebagai segmen dalam menjelaskan aktivitas sistem. Analisis fitur dilakukan pertama kali pada fitur wajib, yaitu lihat data, kemudian dikembangkan dengan analisis fitur opsional pada fitur tambah, perbarui, detail dan hapus data.

Analisis fitur lihat data pada *use case* kelola data KKNP

Diagram aktivitas dari *use case* kelola data KKNP ditunjukkan oleh Gambar 5.12. Pada diagram tersebut dapat dipahami bahwa proses kelola data KKNP melibatkan komponen *controller*, *model*, *view* dan *javascript*. Proses kelola data KKNP pertama kali dilakukan oleh *controller*, kemudian merujuk ke komponen *view* untuk menampilkan data KKNP. Komponen *view* memanggil komponen *javascript* untuk mengambil data KKNP dari basis data dan menampilkan data KKNP pada elemen yang telah disediakan oleh komponen *view*.

Diagram aktivitas Gambar 5.12 merupakan desain model fungsional umum yang pertama, selanjutnya akan dilakukan proses penyempurnaan model dengan melakukan analisis fitur lihat data pada *use case* lain.



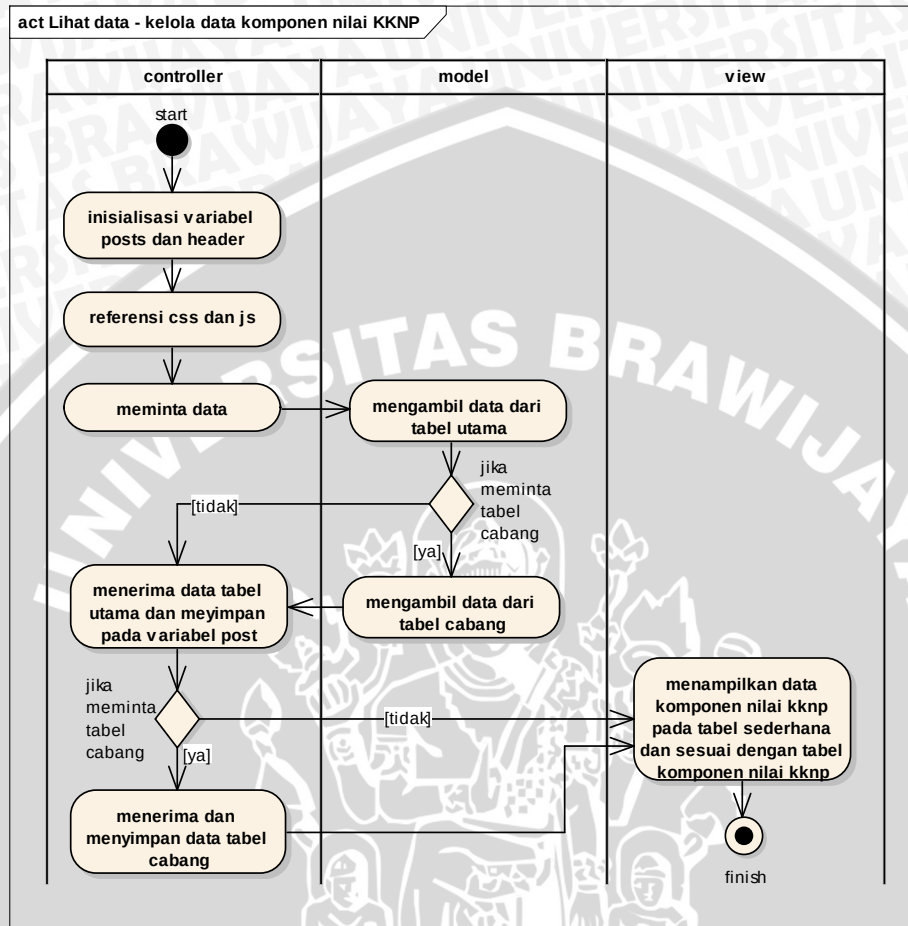
Gambar 5.12 Diagram Aktivitas Kelola Data KKNP

Analisis fitur lihat data pada *use case* kelola komponen nilai KKNP

Diagram aktivitas pada Gambar 5.13 menunjukkan aktivitas sistem pada fitur lihat data. Jika dibandingkan dengan Gambar 5.12 terdapat beberapa persamaan dan perbedaan, maka dari itu diagram tersebut harus sempurnakan untuk menghasilkan model fungsional umum. Perbedaan antar aktivitas pada Gambar 5.12 dan Gambar 5.13 yaitu:

- Pada proses pengambilan dan penampilan data, ada yang melibatkan tabel utama dan ada juga yang melibatkan tabel cabang yang berhubungan.

- Komponen penyusun *use case* kelola data KKNP melibatkan *javascript*, sedangkan komponen penyusun *use case* kelola data komponen nilai KKNP tidak melibatkan *javascript*.



Gambar 5.13 Diagram Aktivitas Kelola Komponen Nilai KKNP

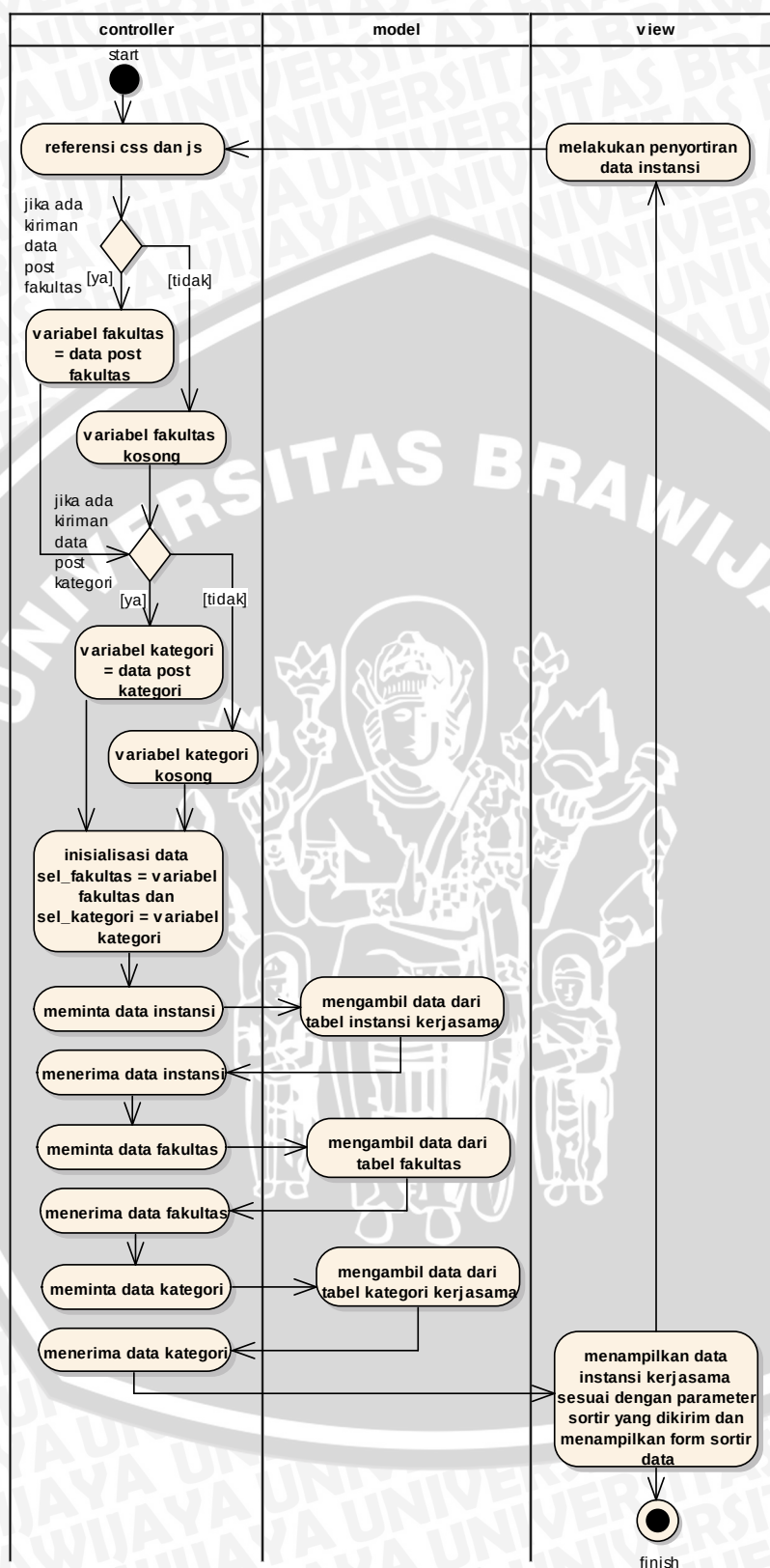
Gambar 5.14 menunjukkan model fungsional umum versi satu pada fitur lihat data. Model tersebut adalah hasil penggabungan antar diagram aktivitas pada Gambar 5.12 dan 5.13. Hasil penggabungan tersebut dilakukan berdasarkan keputusan *domain engineer* dengan alasan sebagai berikut:

- Penggunaan komponen *javascript* dihilangkan untuk membuat komponen penyusun sistem yang lebih sederhana dan ringan, karena peran *javascript* untuk sarana mengambil dan menampilkan data sudah bisa ditangani oleh komponen *model* dan *view*.
- Pada proses pengambilan data dibuat sebuah proses kondisi, apakah sistem akan mengambil data dari tabel utama atau juga dari tabel cabang yang berhubungan (tabel cabang).

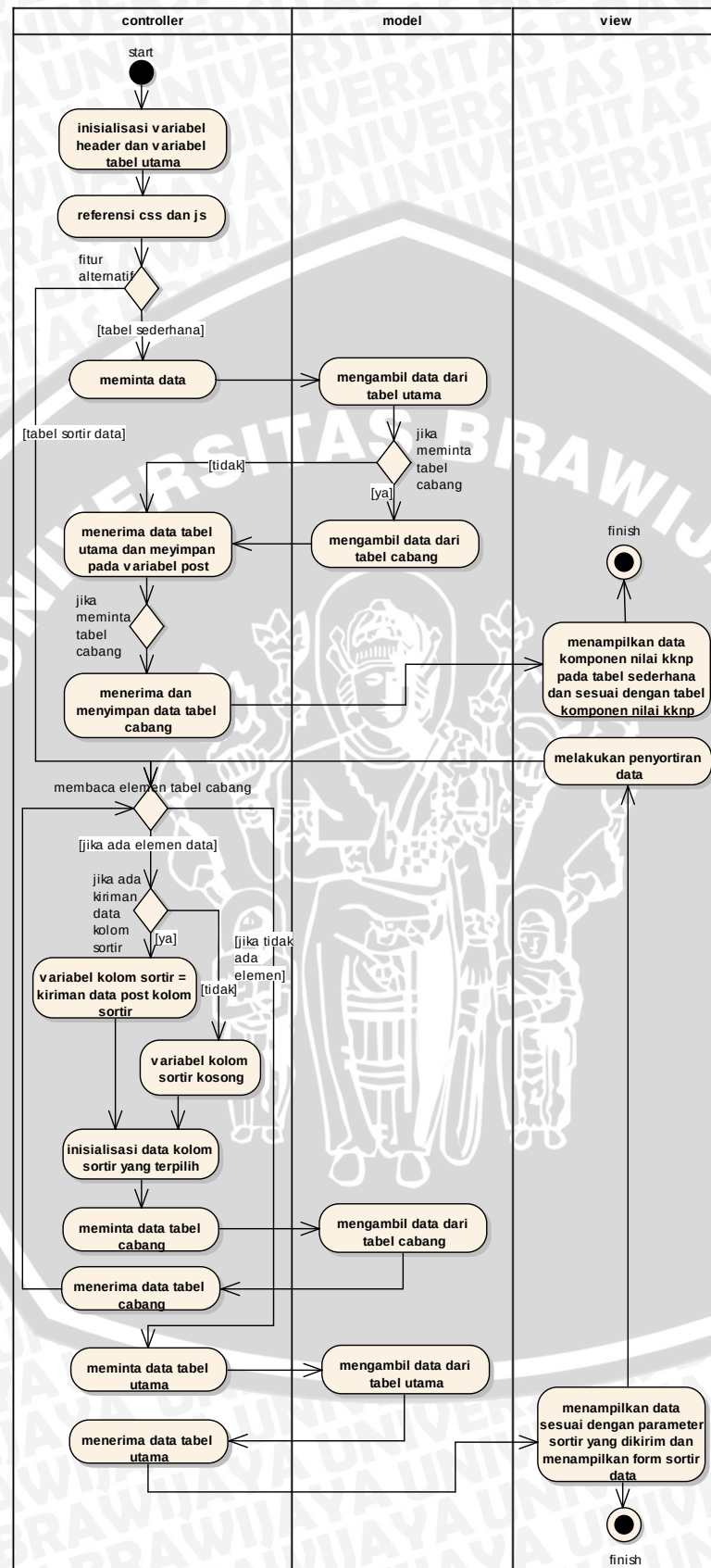


Diagram aktivitas kelola instansi kerjasama ditunjukkan oleh Gambar 5.15, aktivitas tersebut memiliki banyak perbedaan dengan model fungsional V.1 pada Gambar 5.14. Perbedaan tersebut disebabkan karena model fungsional V.1 menerapkan fitur alternatif fungsi tabel sederhana sedangkan diagram aktivitas Gambar 5.15 merupakan fitur alternatif dengan fungsi sortir data. Perbedaan pada diagram aktivitas *use case* kelola instansi kerjasama adalah sebagai berikut:

- Berdasarkan perbedaan tersebut, maka model fungsional umum V.1 harus disempurnakan lagi agar bisa menangani fitur alternatif sortir data.



Gambar 5.15 Diagram Aktivitas Kelola Instansi Kerjasama



Gambar 5.16 menunjukkan model fungsional umum dari fitur wajib lihat data, model tersebut juga mencakup fitur alternatif untuk menampilkan data dengan tabel sederhana atau dengan tabel sortir data. Pada awal proses akan dilakukan identifikasi apakah fitur lihat data menggunakan fitur alternatif tabel sederhana atau tabel sortir data, selanjutnya proses spesialisasi fitur dilakukan berdasarkan hasil identifikasi tersebut.

Proses sortir data dimodifikasi sehingga bisa menangani berbagai macam fungsi sortir data yang dibentuk dari beberapa tabel cabang. Tabel utama dan tabel cabang akan dimasukkan kedalam sebuah model masukan agar bisa diproses secara dinamis oleh sistem, sehingga menghasilkan fitur sortir data yang sesuai dengan persyaratan.

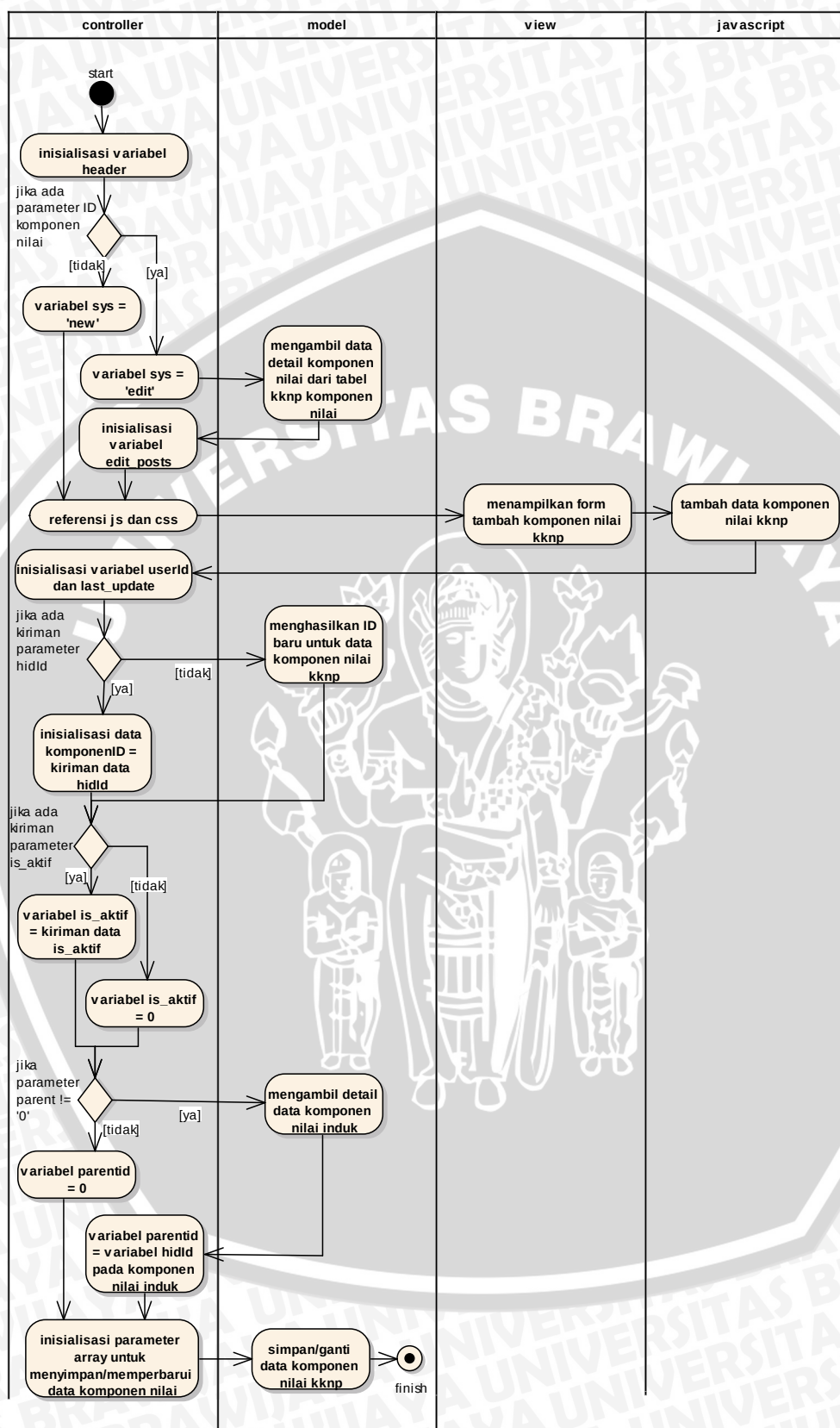
Analisis fungsional fitur lihat data pada *use case* kelola kategori kerjasama menunjukkan bahwa stuktur proses tersebut identik dengan diagram aktivitas pada Gambar 5.13. Stuktur fungsional fitur pada *use case* kelola kegiatan kerjasama identik dengan diagram aktivitas pada Gambar 5.15. Sehingga bisa disimpulkan bahwa model fungsional fitur lihat data V.2 adalah model fungsional umum final pada domain modul PTIIK-Apps.

Fitur wajib pada domain telah dimodelkan menjadi model fungsional umum pada Gambar 5.16. Fitur – fitur opsional juga dianalisis untuk melakukan spesialisasi model fungsional umum.

Analisis fitur tambah data pada *use case* tambah komponen nilai KKNP

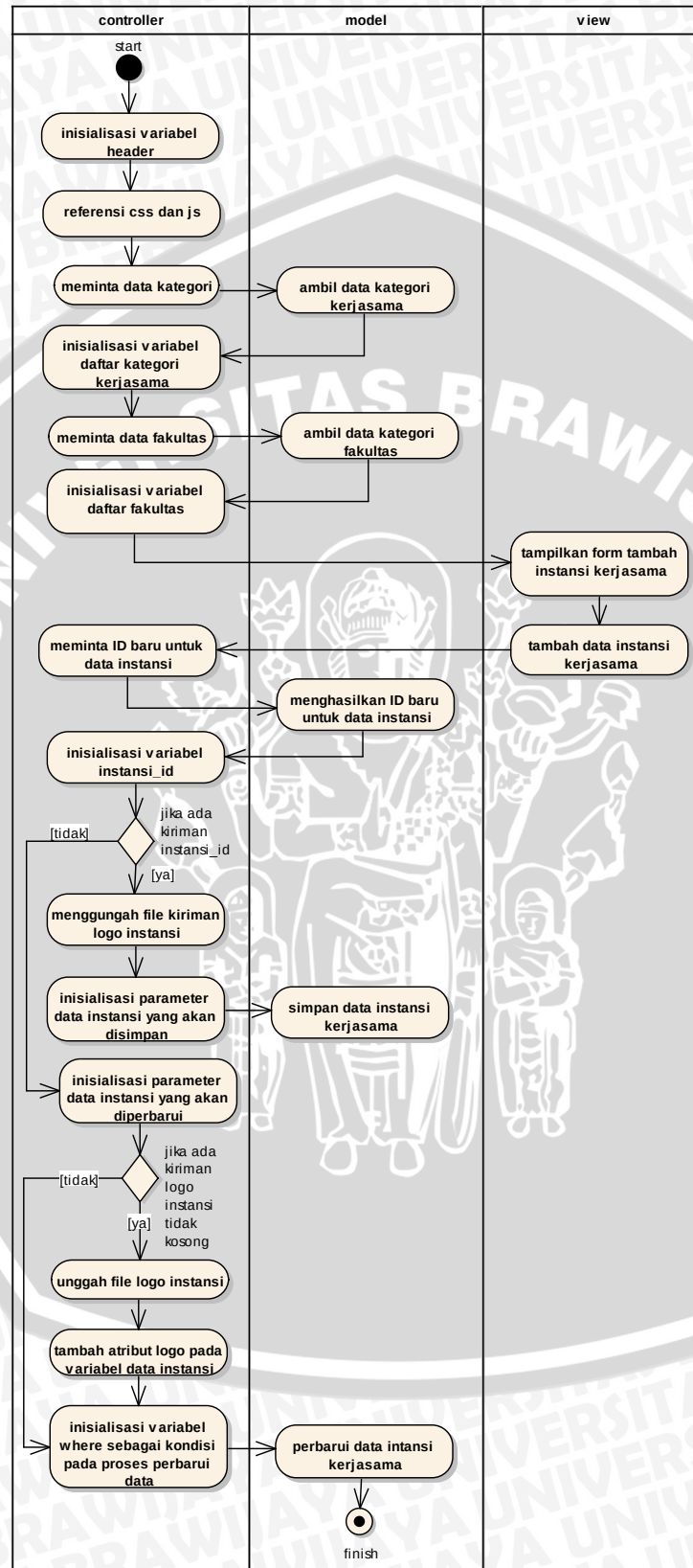
Gambar 5.17 menunjukkan diagram aktivitas fitur tambah data komponen nilai KKNP. Berdasarkan keputusan *domain engineer*, proses yang harus direvisi untuk membentuk model fungsional dari fitur tambah data adalah sebagai berikut:

- Pada komponen *controller*, terdapat proses identifikasi apakah fitur yang akan dijalankan adalah tambah atau perbarui data. Proses tersebut bisa digantikan dengan membagi antara proses tambah dan perbarui data kedalam fungsi yang berbeda.
- Diagram aktivitas tersebut melibatkan komponen *javascript* untuk melakukan proses simpan data, komponen *javascript* lebih efektif digunakan untuk mengelola proses tambah tabel cabang.
- Pada komponen *controller* terdapat proses untuk memeriksa apakah ada nilai variabel *is_aktif*, untuk tipe masukan *checkbox*. Proses tersebut bisa diganti dengan menggunakan tipe data *radio button* atau *select-option* dengan pilihan *is_aktif* sama dengan 1 dan 0, sehingga tidak perlu proses seleksi kondisi pada komponen *controller*.



Gambar 5.17 Diagram Aktivitas Tambah Komponen Nilai KKNP

Analisis fitur tambah data pada *use case* tambah instansi kerjasama



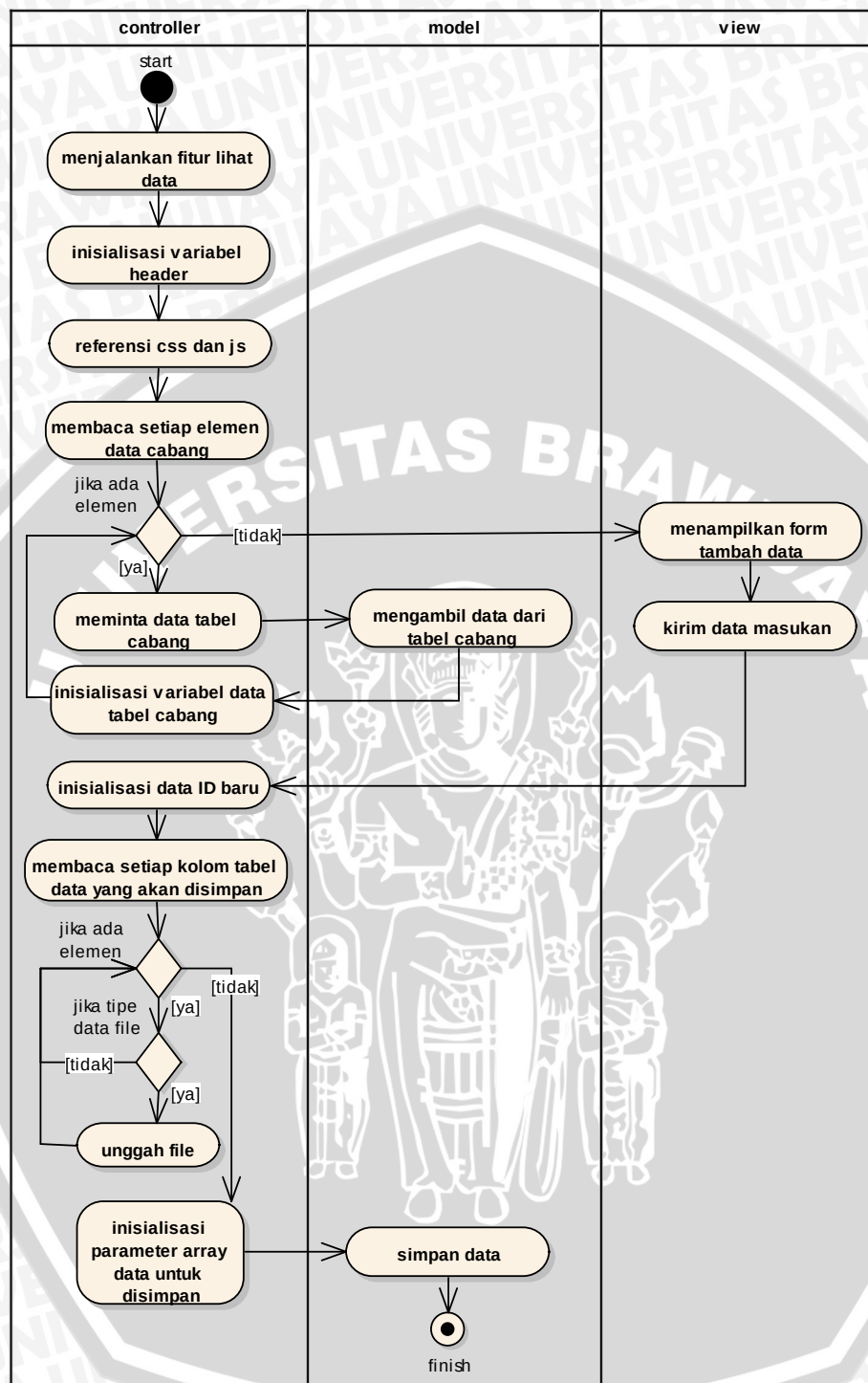
Gambar 5.18 Diagram Aktivitas Tambah Instansi Kerjasama

Gambar 5.18 menunjukkan diagram aktivitas fitur tambah data instansi kerjasama. Berdasarkan keputusan *domain engineer*, proses yang harus direvisi untuk membentuk model fungsional dari fitur tambah data adalah sebagai berikut:

- Pada komponen *controller*, terdapat proses identifikasi apakah fitur yang akan dijalankan adalah tambah atau perbarui data. Proses tersebut bisa digantikan dengan membagi antara proses tambah dan perbarui data kedalam fungsi yang berbeda.

Gambar 5.19 menunjukkan model fungsional fitur tambah data V.1, model tersebut disusun berdasarkan analisis pada diagram aktivitas pada Gambar 5.17 dan Gambar 5.18. Perubahan dan pengambilan keputusan dalam menyusun model fungsional fitur tambah data V.1 adalah sebagai berikut:

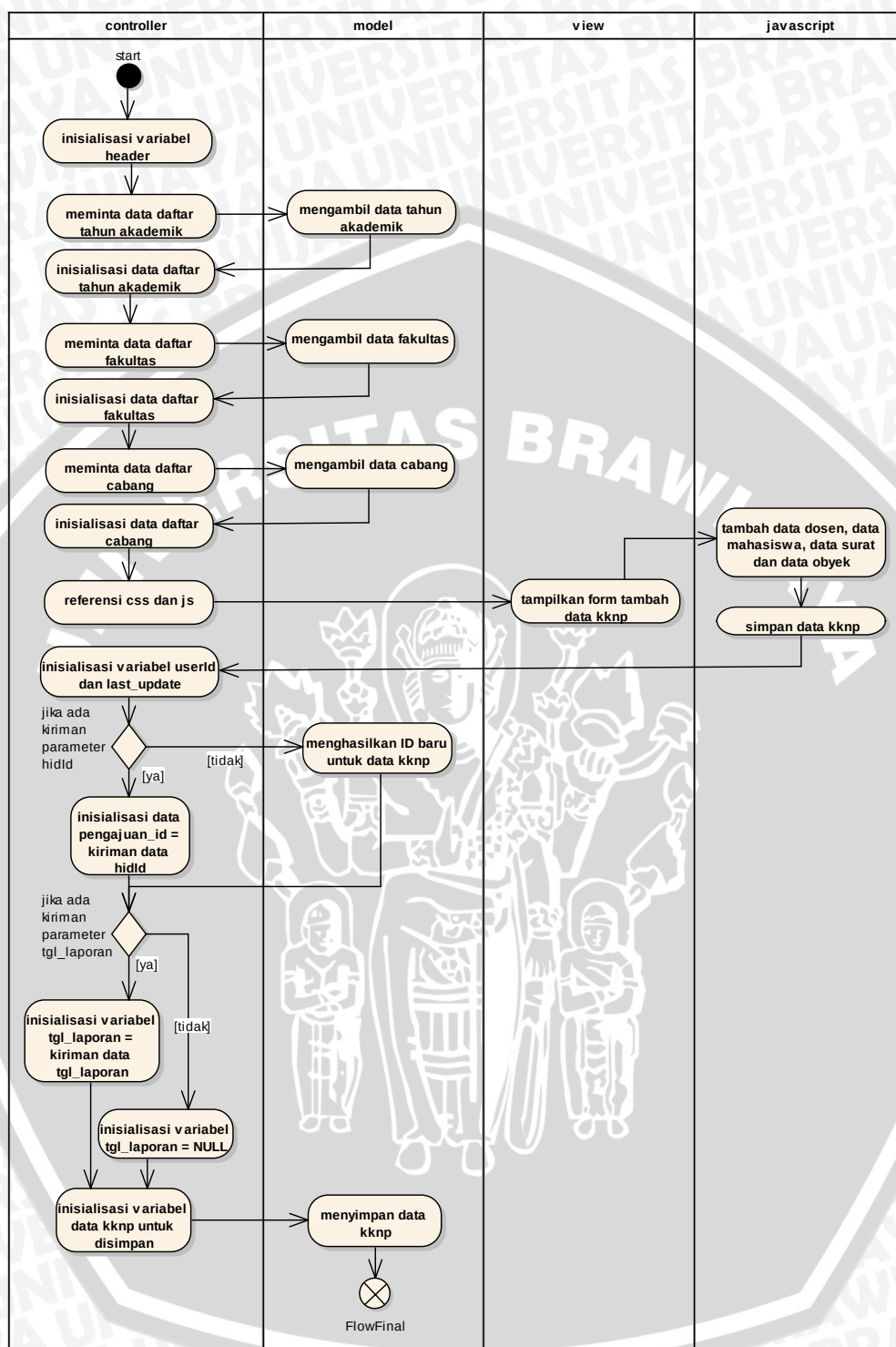
- Pada komponen *controller*, model fungsional disusun khusus untuk menjalankan fungsi tambah data.
- Pada saat menampilkan data form tambah data, komponen *controller* akan membaca setiap tabel cabang untuk mengambil detail data dari model.
- Komponen *javascript* tidak digunakan pada proses tambah data pada satu tabel saja.
- Proses untuk menghasilkan ID baru melibatkan *controller* dengan memanggil fungsi warisan pada komponen *comscontroller*.
- Komponen *controller* akan memeriksa setiap kolom masukan, jika ada tipe data *file* maka sistem akan mengunggah *file* tersebut.



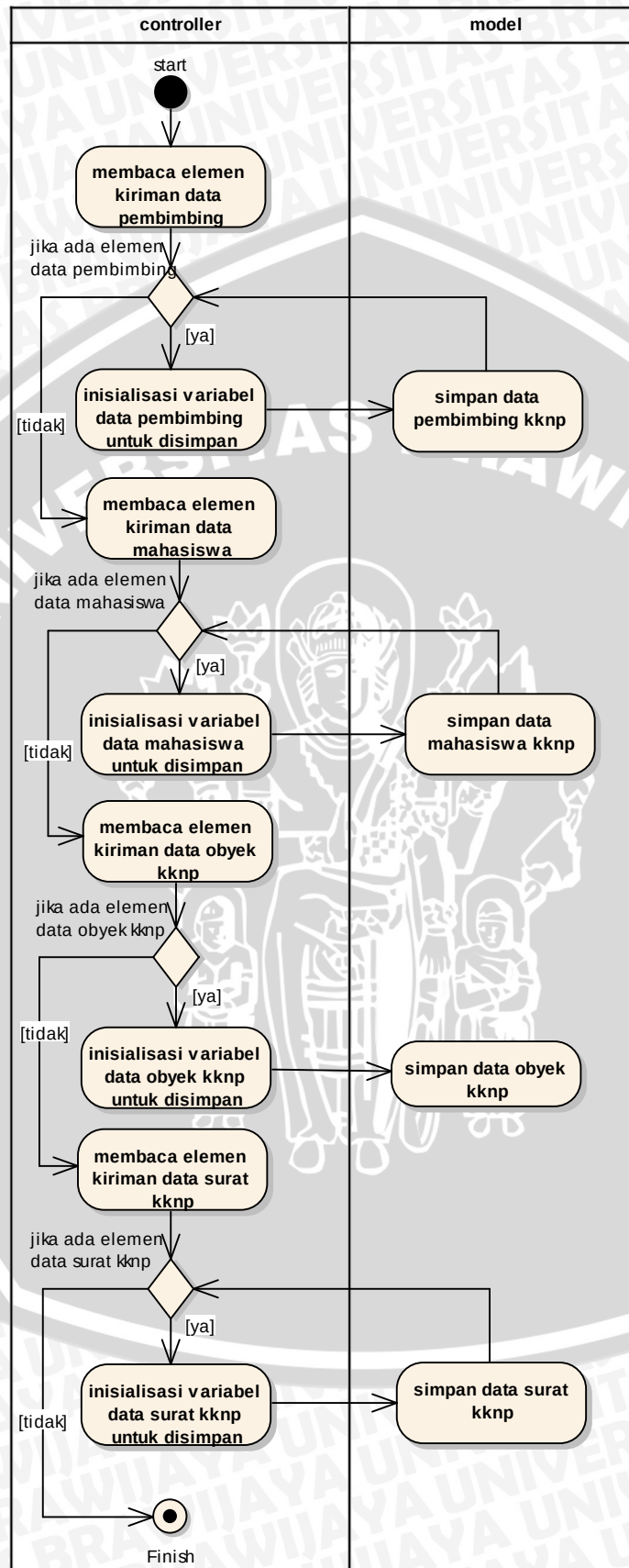
Gambar 5.19 Model Fungsional Fitur Tambah Data V.1

Analisis fitur tambah data pada *use case* tambah data kknp

Analisis fitur tambah data kknp digambarkan dengan diagram aktivitas pada Gambar 5.20 dan Gambar 5.21.



Gambar 5.20 Diagram Aktivitas Fitur Tambah Data KKNP (bagian 1)

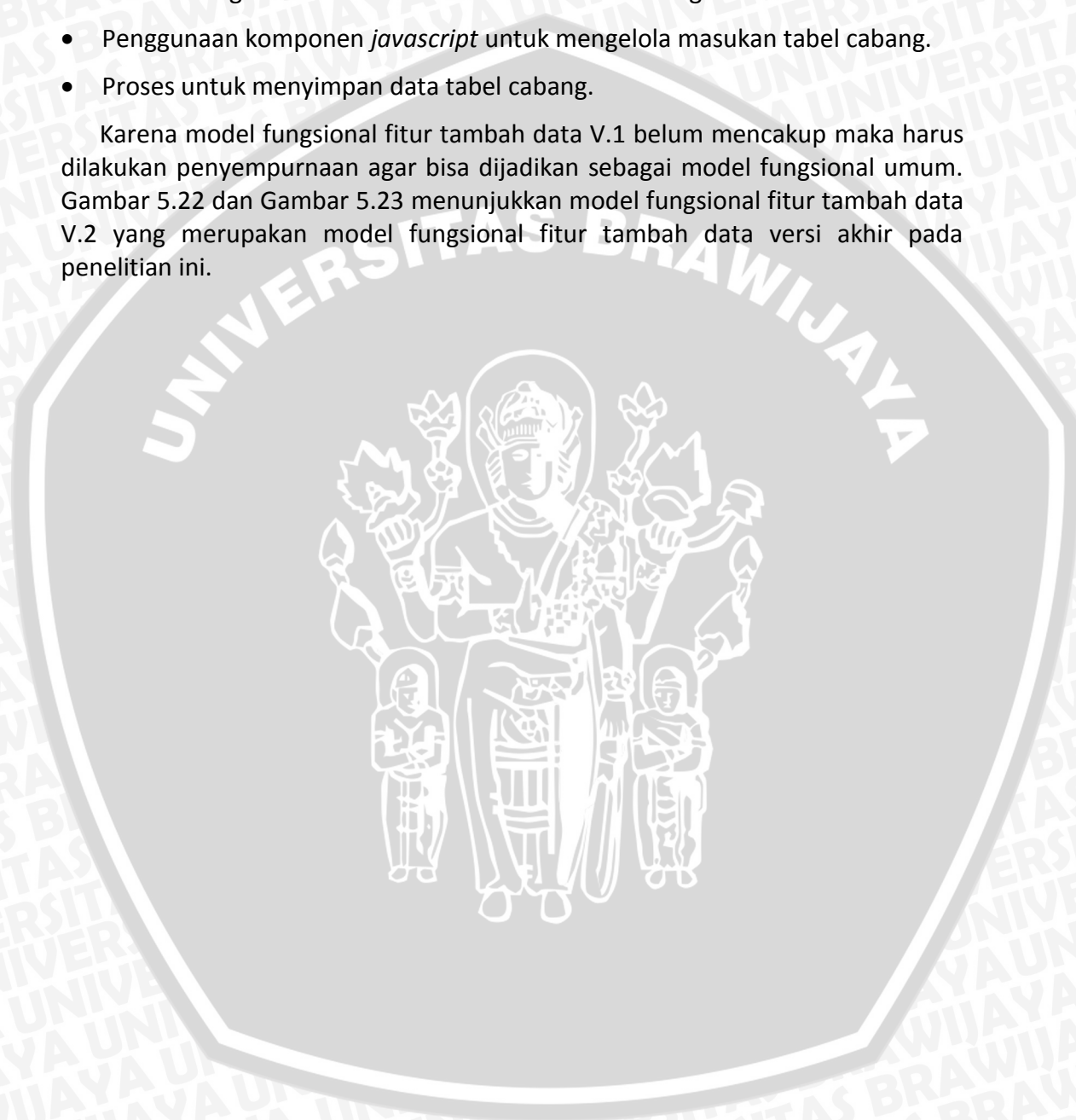


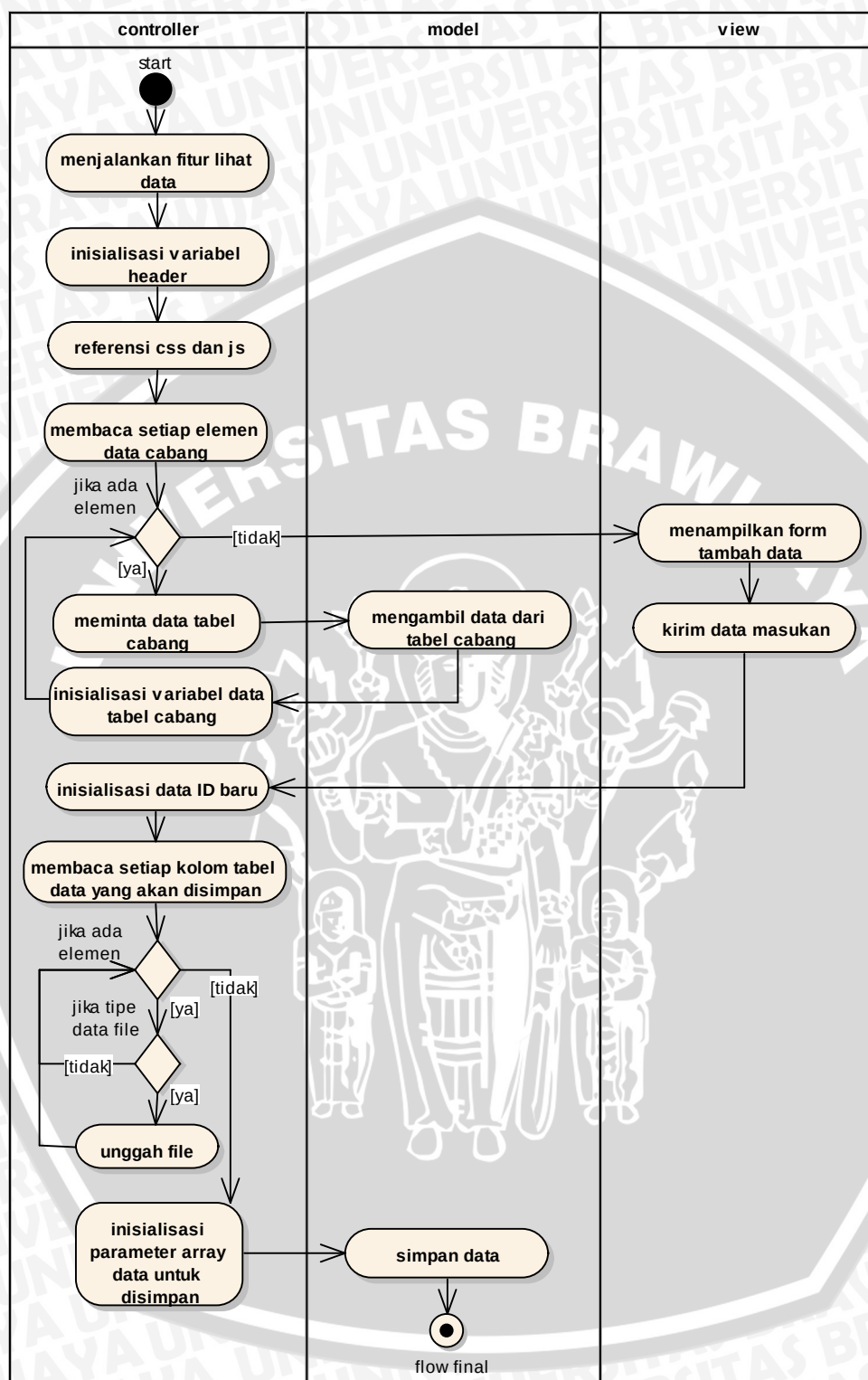
Gambar 5.21 Diagram Aktivitas Fitur Tambah Data KKNP (bagian 2)

Analisis dilakukan dengan membandingkan model fungsional fitur tambah data V.1 pada Gambar 5.19 dengan diagram aktivitas *use case* tambah data KKNP pada Gambar 5.20 dan 5.21. Aktivitas tambah data kknp yang belum bisa dicakup oleh model fungsional fitur tambah data V.1 adalah sebagai berikut:

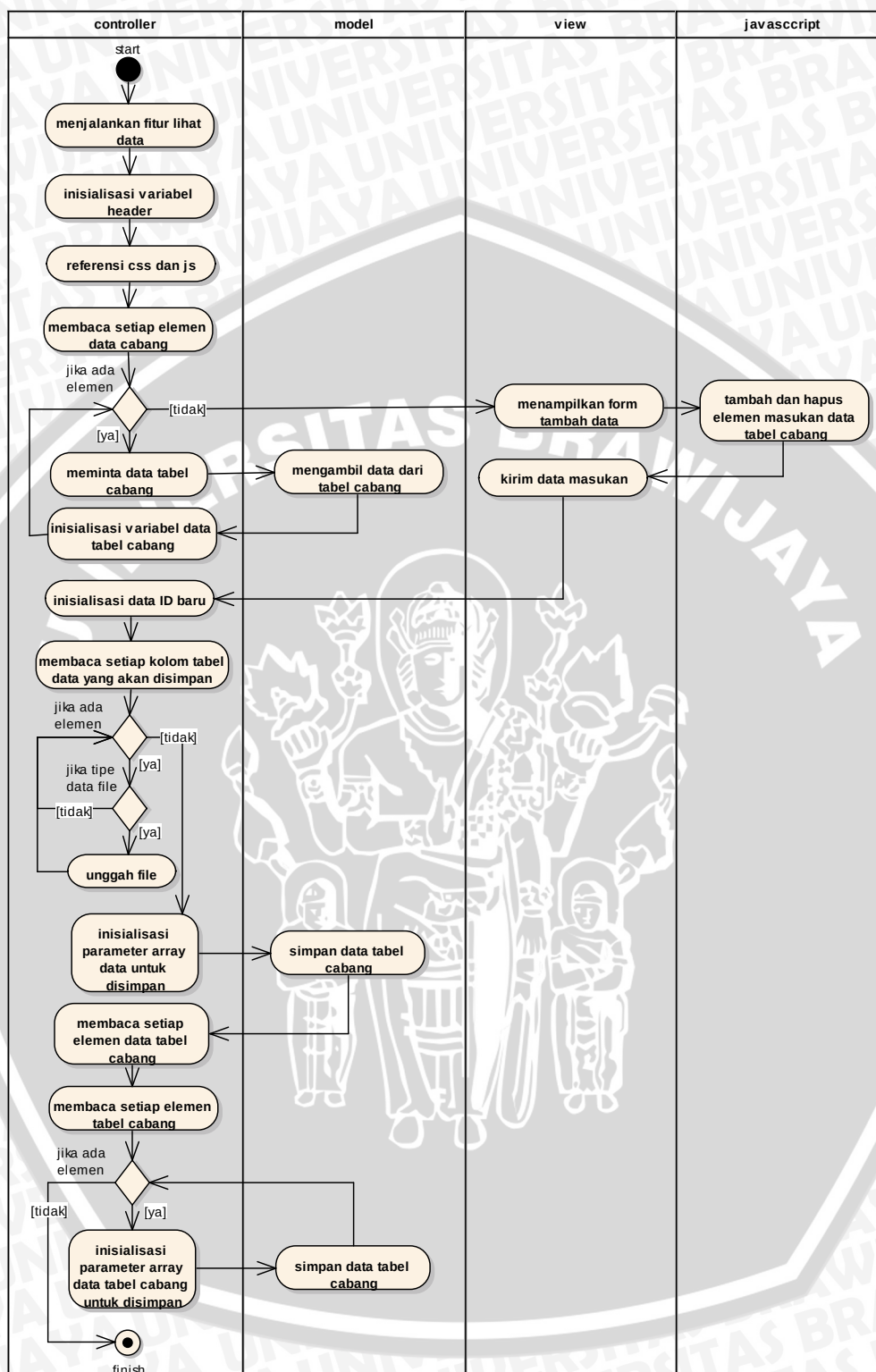
- Penggunaan komponen *javascript* untuk mengelola masukan tabel cabang.
- Proses untuk menyimpan data tabel cabang.

Karena model fungsional fitur tambah data V.1 belum mencakup maka harus dilakukan penyempurnaan agar bisa dijadikan sebagai model fungsional umum. Gambar 5.22 dan Gambar 5.23 menunjukkan model fungsional fitur tambah data V.2 yang merupakan model fungsional fitur tambah data versi akhir pada penelitian ini.

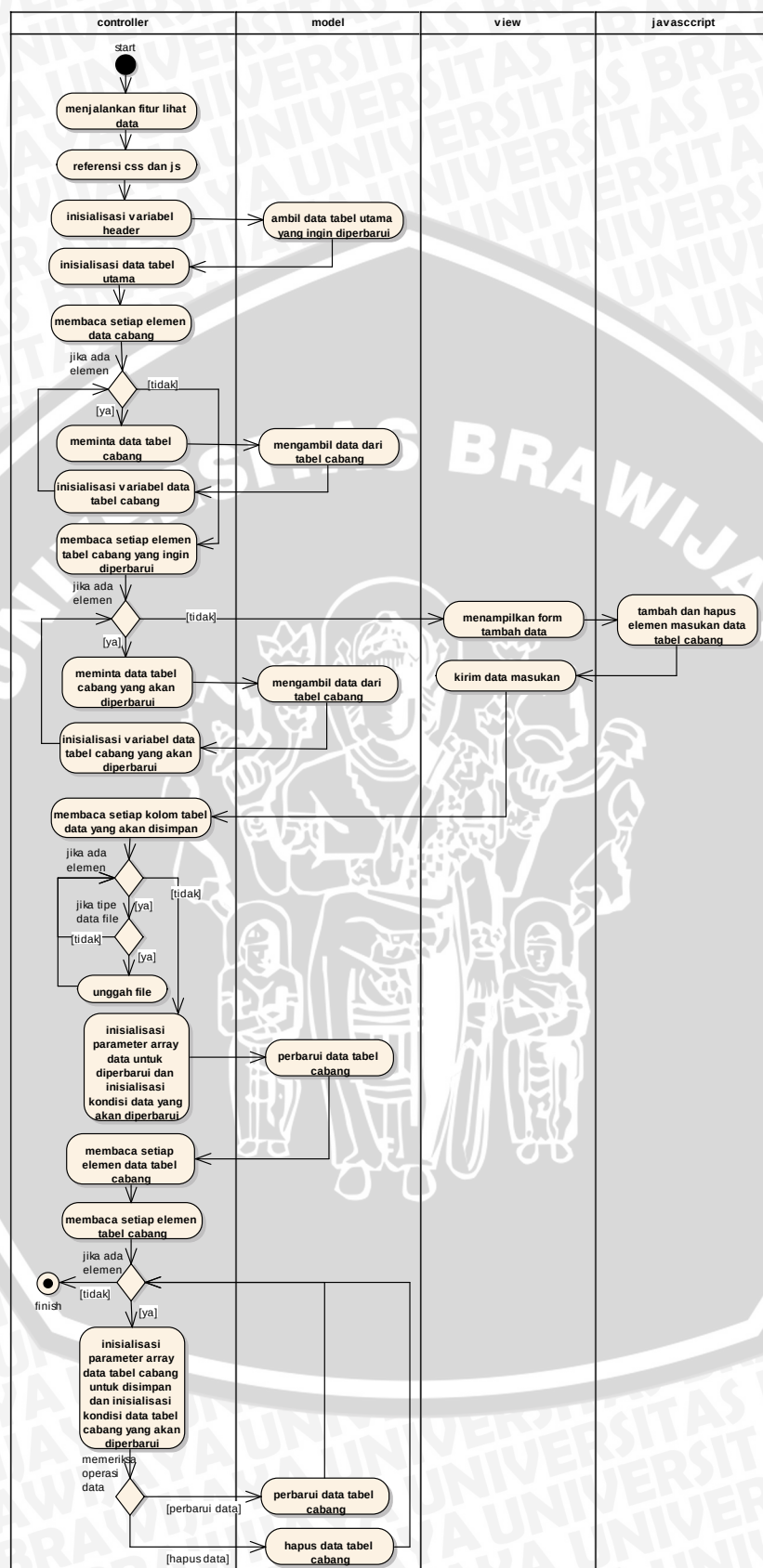




Gambar 5.22 Model Fungsional Fitur Tambah Data V.2 (bagian 1)



Gambar 5.23 Model Fungsional Fitur Tambah Data V.2 (bagian 2)

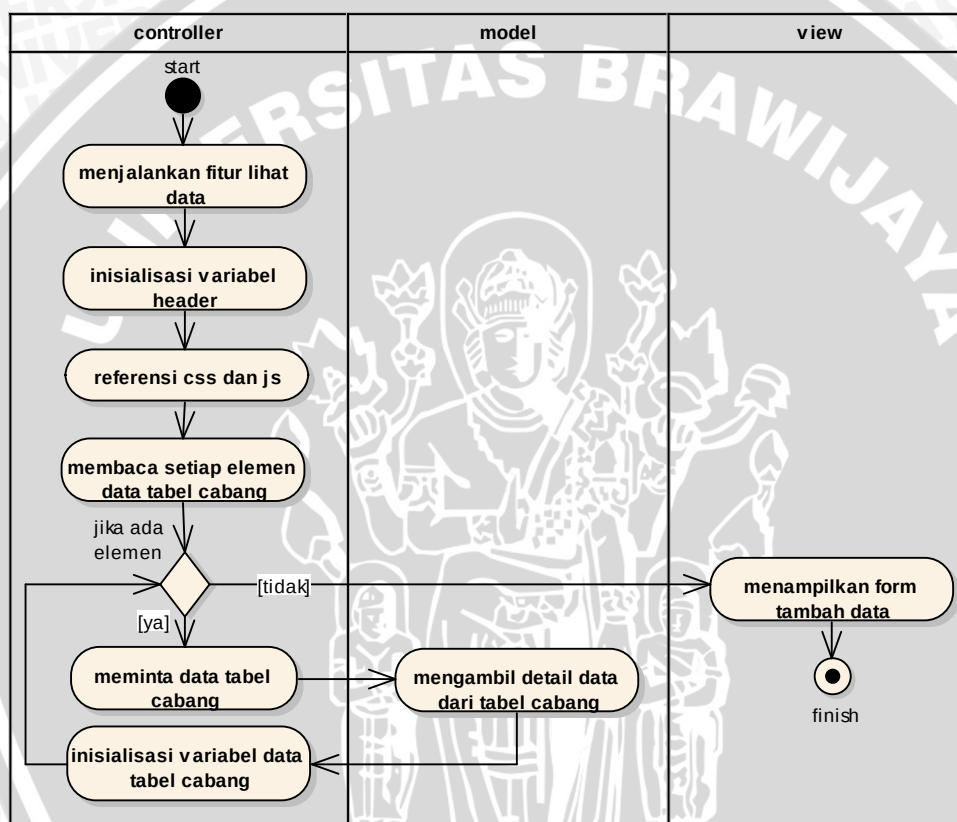


Gambar 5.24 Model Fungsional Fitur Perbarui Data

Gambar 5.24 menunjukkan model fungsional fitur perbarui data, pada model tersebut ditunjukkan alur umum domain dalam menjalankan fitur perbarui data. **Proses yang dilakukan untuk membuat model fungsional pada Gambar 5.24 sama seperti proses yang telah dilakukan untuk menyusun model fungsional sebelumnya.**

Berdasarkan keputusan *domain engineer*, terdapat modifikasi pada model fungsional fitur perbarui data, yaitu:

- Dilakukan pemeriksaan pada operasi perbarui data tabel cabang, sehingga bisa dilakukan operasi tambah, perbarui atau hapus data pada tabel cabang.



Gambar 5.25 Model Fungsional Fitur Detail Data

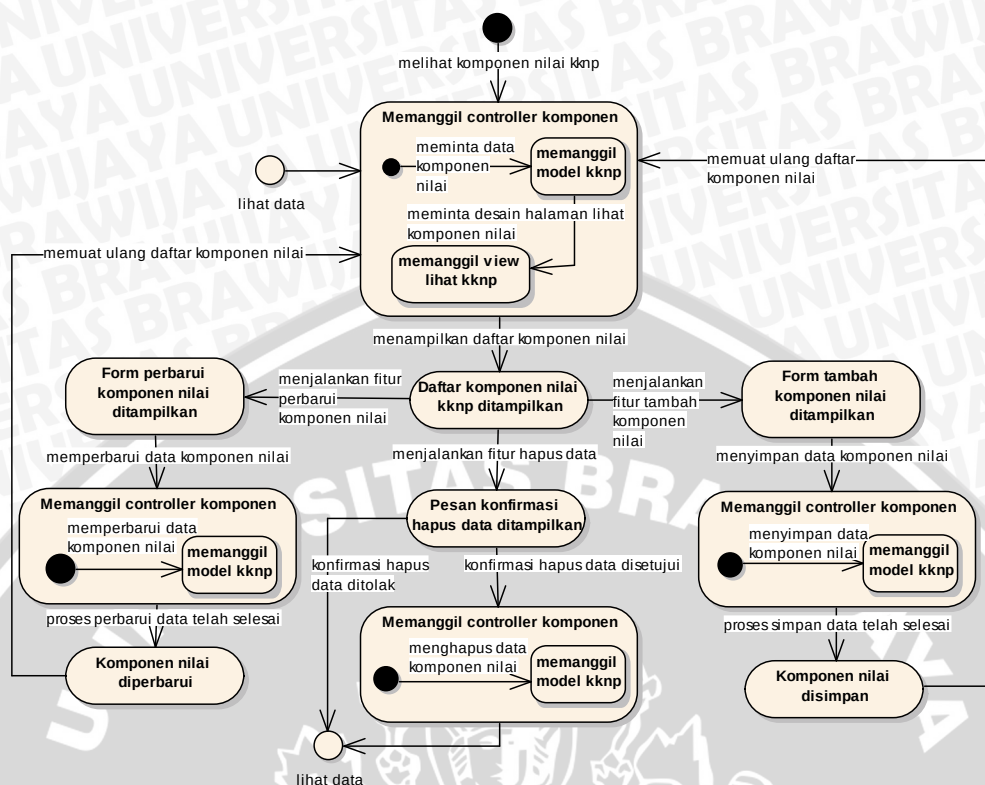
Model fungsional fitur detail data ditunjukkan oleh Gambar 5.25. Model tersebut telah mencakup fitur opsional untuk melihat detail data dari tabel cabang.



Analisis perilaku domain

Analisis perilaku domain dilakukan untuk mengetahui persamaan dan perbedaan perilaku pada masing – masing modul PTIIK-Apps yang dianalisis. Analisis perilaku juga membantu peneliti untuk mengambil keputusan dalam melakukan desain model domain. Gambar 5.27.1 dan Gambar 5.27.2 menunjukkan diagram keadaan pada modul KKNP.





Gambar 5.27.2 Diagram Keadaan Modul KKNP (bagian 2)

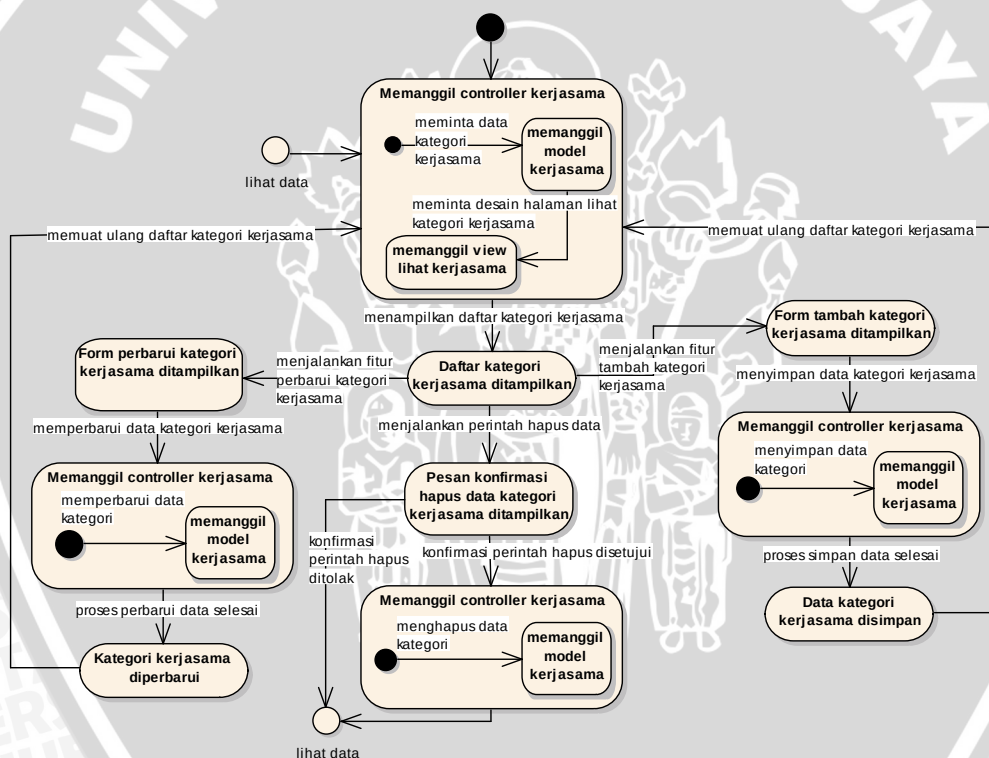
Berdasarkan diagram keadaan (*state diagram*) pada Gambar 5.27.1 dan Gambar 5.27.2 dapat dijelaskan bahwa:

- Fitur lihat data adalah fitur yang diakses pertama kali sebelum menjalankan fitur – fitur lainnya (lihat data, perbarui data dan hapus data).
- Fitur lihat data membutuhkan pemanggilan komponen *controller* sebanyak satu kali yaitu untuk mendapatkan 1) daftar data yang ingin ditampilkan dan 2) desain halaman lihat data yang ingin ditampilkan.
- Fitur pengajuan kknp dikategorikan sebagai fitur tambah data, proses tersebut membutuhkan pemanggilan komponen *controller* sebanyak dua kali, yaitu: 1) untuk mendapatkan data tabel cabang yang berhubungan dengan `tbl_kkn_pengajuan` dan desain halaman lihat data, 2) untuk melakukan proses simpan data ke sistem basis data.
- Fitur perbarui kknp diategori sebagai fitur perbarui data, proses tersebut juga membutuhkan dua kali pemanggilan komponen *controller*, yaitu: 1) untuk mendapatkan detail data yang akan diperbarui, data cabang yang berhubungan dan desai halaman perbarui data, 2) untuk melakukan proses perbarui data ke sistem basis data.
- Fitur tambah dan perbarui data pada Gambar 5.27.2, kelola komponen nilai KKNP cukup membutuhkan satu kali pemanggilan komponen *controller*, karena form tambah dan perbarui data dijadikan satu dengan halaman lihat

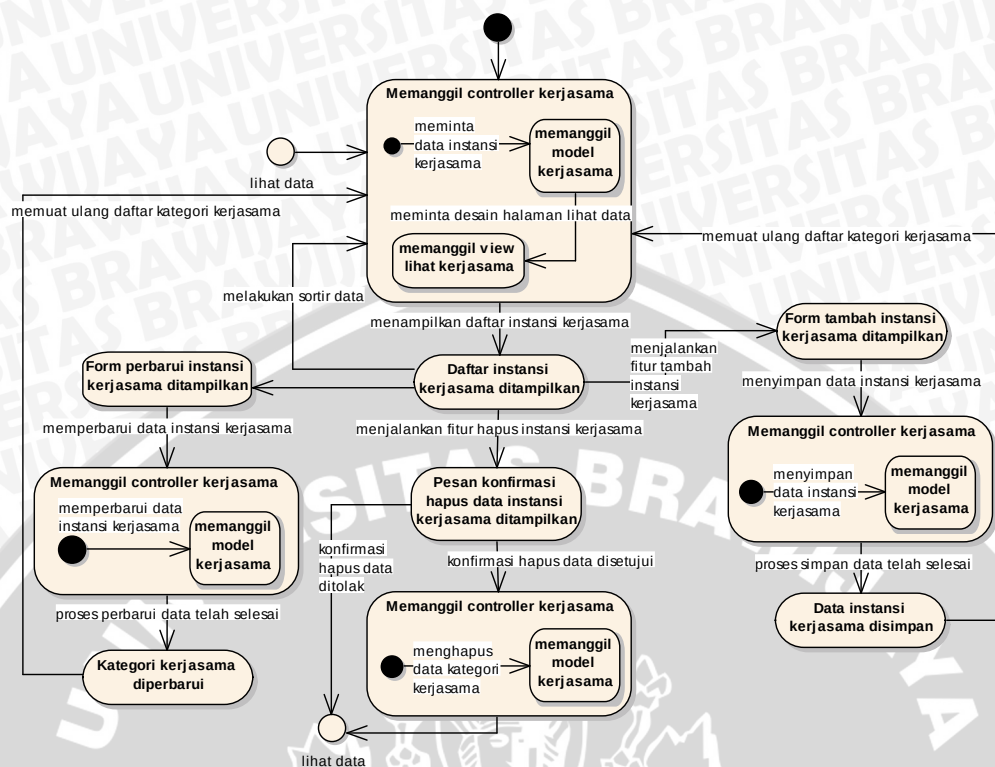
data dengan alasan bahwa form tersebut memiliki masukan yang sederhana dan tidak melibatkan hubungan dengan tabel lain. Berdasarkan perbedaan ini, maka peneliti memutuskan bahwa pada fitur tambah dan perbarui data tetap membutuhkan dua kali pemanggilan komponen *controller*. Fitur tambah dan perbarui data akan lebih leluasan untuk dikembangkan jika melakukan pemanggilan komponen *controller* sebanyak dua kali, karena pengembang bisa memodifikasi kode program pada bagian pemanggilan form tambah/perbarui data dan pada bagian operasi data.

- Pada fitur hapus data, sistem akan memberikan pesan konfirmasi untuk menjalankan fitur tersebut. Jika konfirmasi hapus data disetujui maka sistem akan memanggil komponen *controller* untuk menghapus data pada sistem basis data dan akan memuat halaman lihat data kembali.

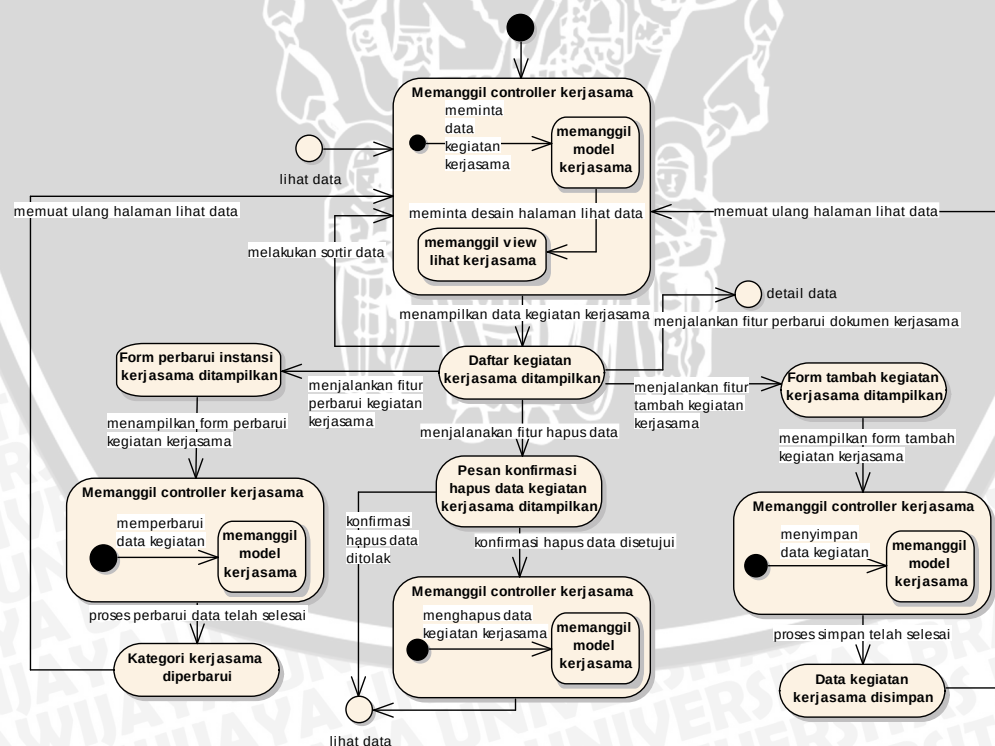
Gambar 5.27.3, Gambar 5.27.4, Gambar 5.27.5 dan Gambar 5.27.6 menunjukkan diagram keadaan pada modul kerjasama.



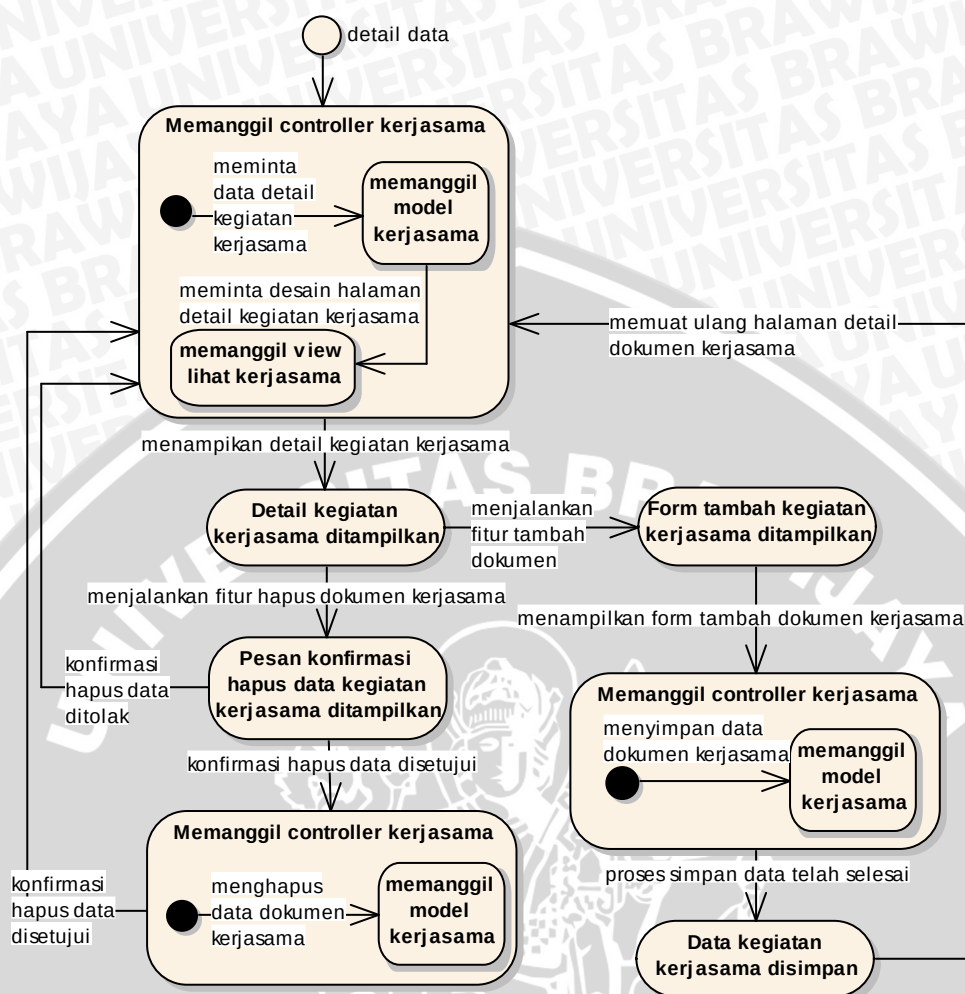
Gambar 5.27.3 Diagram Keadaan Modul Kerjasama (bagian 1)



Gambar 5.27.4 Diagram Keadaan Modul Kerjasama (bagian 2)



Gambar 5.27.5 Diagram Keadaan Modul Kerjasama (bagian 3)



Gambar 5.27.6 Diagram Keadaan Modul Kerjasama (bagian 4)

Gambar 5.27.3 adalah perilaku dari fitur kelola kategori kerjasama pada modul kerjasama, perilaku fitur tersebut identik dengan perilaku kelola komponen nilai pada modul KKNP.

Gambar 5.27.4 menjelaskan tentang perilaku fitur kelola instansi kerjasama, yang secara garis besar memiliki kesamaan perilaku dengan fitur kelola komponen nilai pada modul KKNP. **Perbedaan** perilaku sistem terdapat pada perilaku fitur lihat data, pada fitur lihat data instansi kerjasama memungkinkan sebuah interaksi untuk melakukan sortir data dengan mengirimkan parameter sortir dan memuat ulang halaman lihat data.

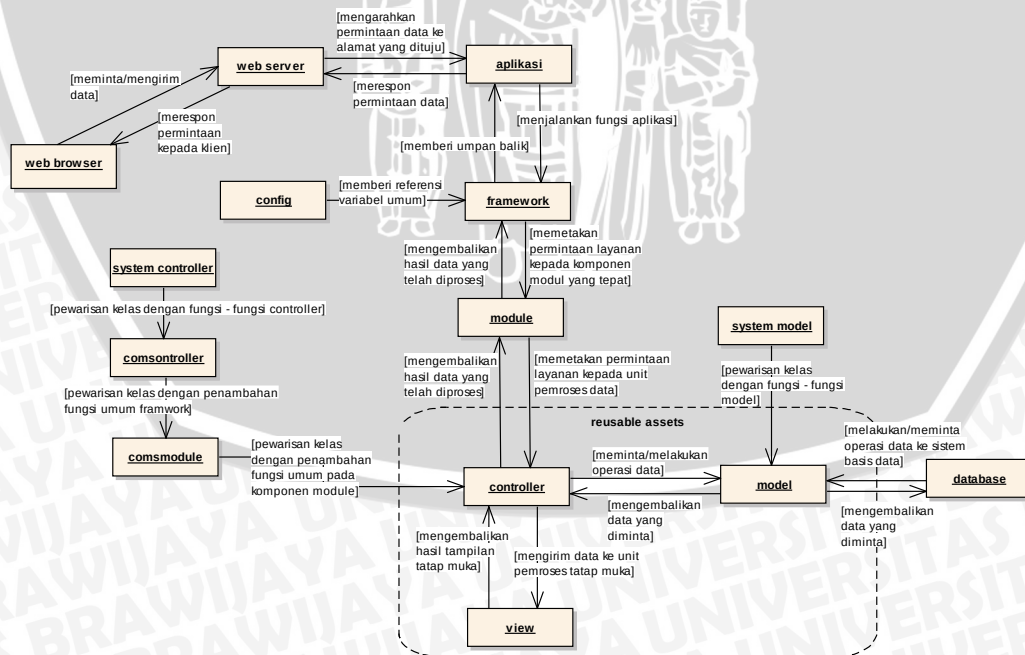
Gambar 5.27.5 menunjukkan perilaku fitur kelola data kegiatan kerjasama yang memiliki kesamaan fitur dengan perilaku yang ditunjukkan oleh Gambar 5.27.4. **Perbedaan** perilaku terletak pada adanya perilaku fitur detail data untuk melakukan proses kelola dokumen kerjasama berdasarkan data kegiatan kerjasama pada Gambar 5.27.6. Fitur kelola dokumen kerjasama juga memiliki kesamaan perilaku dengan fitur kelola komponen nilai modul KKNP.

5.2.3 Pemodelan Arsitektur

Pemodelan arsitektur sistem disusun dengan membuat beberapa model arsitektur. Lapisan tersebut terdiri dari *domain architecture layer* yang menjelaskan hubungan dan interaksi antar proses yang berjalan pada domain, lapisan dibawahnya adalah *domain utilities layer* yang menjelaskan proses pemaketan dan hubungan fungsi dan objek data menjadi sebuah modul.

Gambar 5.28 adalah model arsitektur yang menggambarkan *domain architecture layer*. Pada model tersebut ditunjukkan interaksi proses pada domain beserta komponen yang bertanggungjawab terhadap antar proses tersebut. *Web browser* adalah komponen yang digunakan pengguna untuk berinteraksi dengan sistem. *Web server* merupakan penyedia lingkungan yang tepat sehingga aplikasi berbasis web bisa berjalan. *Framework* adalah sebuah kerangka sistem yang mengatur alur jalannya sebuah sistem. *Framework* akan menunjuk komponen *controller* sebagai unit pemroses data yang telah mewarisi sifat dari objek *system controller*, *comscontroller* dan *comsmodule*. Komponen *config* menyediakan variabel umum sehingga bisa diakses oleh komponen sistem. *Controller* menggunakan komponen *model* sebagai unit pemroses operasi data, komponen *model* mewarisi sifat dari objek *system model* sehingga bisa berinteraksi dengan sistem pengelola basis data.

Pada Gambar 5.28, komponen berada pada lingkup *reusable assets* merupakan definisi komponen PTIIK-Apps yang bisa digunakan ulang pada domain yang ditunjuk. Gambar 5.28 menggambarkan arsitektur sistem PTIIK-Apps secara keseluruhan, sehingga membantu memposisikan *reusable assets* yang telah didefinisikan agar dapat berinteraksi dengan sistem.



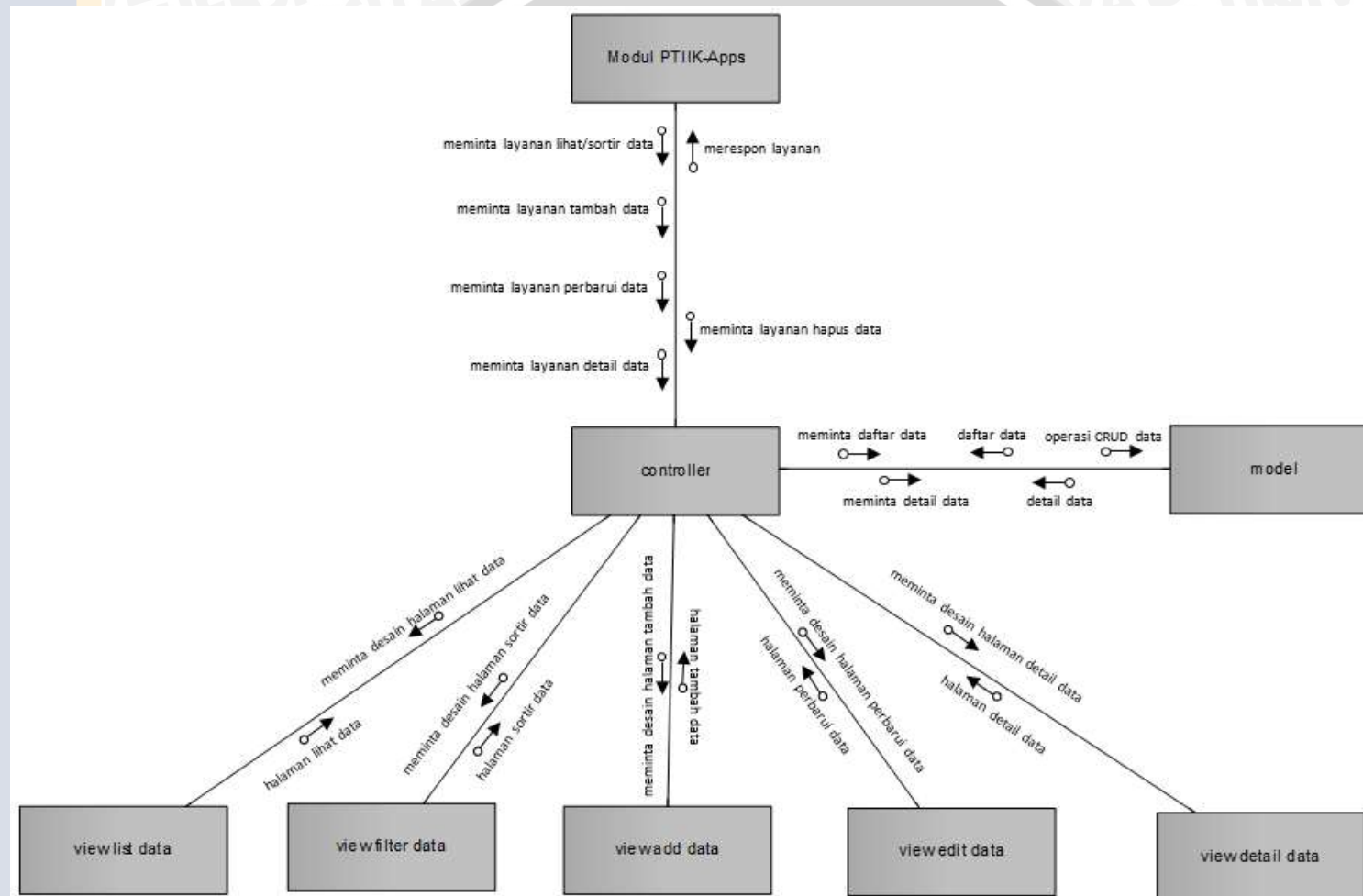
Gambar 5.28 Model Arsitektur pada Lapisan *Domain Architecture Layer*

Arsitektur sistem pada lapisan *domain utilities layer* ditunjukkan oleh Gambar 5.29. Model tersebut menjelaskan pemaketan fungsi dan objek *reusable assets* yang telah didefinisikan pada Gambar 5.28. Model pada Gambar 5.29 menjelaskan bagaimana fitur yang telah didefinisikan pada model fitur Gambar 5.10 dikelola oleh komponen *reusable assets*.

Module merupakan komponen yang ditugaskan oleh komponen *framework* aplikasi untuk menjalankan layanan. Selanjutnya komopnen *module* akan menunjukkan unit pemroses data (*controller*) **pada lingkungnya** untuk menjalankan alur layanan data, seperti layanan: lihat, sortir, tambah, perbarui, detail dan hapus data.

Dalam satu komponen, satu komponen *controller* bisa berkomunikasi dengan 1) satu komponen *model*, 2) dan lima macam komponen *view* sesuai dengan fitur yang telah didefinisikan pada fitur model.





Gambar 5.29 Model Arsitektur pada Lapisan *Domain Utilities Layer*

5.3 Desain Domain

Analisis domain telah menghasilkan beberapa model domain yang bisa digunakan untuk melakukan rekayasa domain. Rekayasa domain yang dilakukan pada penelitian ini adalah dengan membuat sebuah *program/code generator*. *Program generator* yang akan dibuat dapat menghasilkan komponen MVC secara otomatis berdasarkan masukan dari pengguna aplikasi. Komponen yang dihasilkan dapat menjalankan fitur – fitur seperti yang telah didefinisikan pada model fitur.

Masukan yang dibutuhkan oleh *program generator* berupa sebuah abstraksi komponen MVC yang ingin dibangun. Abstraksi tersebut meliputi masukan berupa: 1) model modul, 2) model komponen, 3) model fitur, 4) model tabel, 5) model kolom dan 6) model pilihan masukan. Aplikasi *program generator* akan menerjemahkan masukan tersebut sesuai dengan desain basis pengetahuan yang dimiliki aplikasi, sehingga akan dihasilkan susunan komponen MVC yang sesuai dengan abstraksi yang telah dibuat.

Model modul digunakan untuk melakukan abstraksi nama modul pada komponen *controller* dan *model* yang akan dibangun. Model komponen digunakan untuk melakukan abstraksi nama komponen *controller* dan *model* yang akan dibangun. Model fitur digunakan untuk melakukan abstraksi: 1) fungsi – fungsi yang akan dibangun pada komponen *controller* dan *model* 2) dan spesifikasi komponen *view* yang akan dibangun.

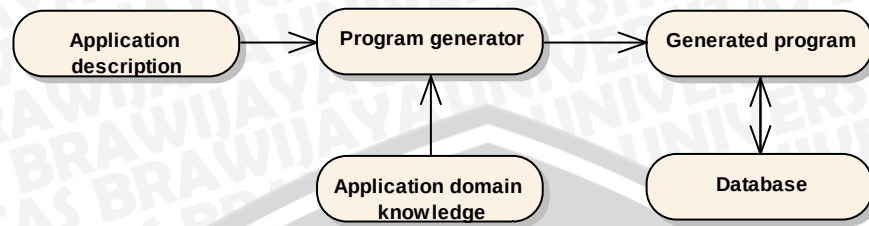
Masukan model tabel digunakan sebagai abstraksi tabel pada sistem basis data agar komponen MVC yang akan dibangun dapat berinteraksi dengan valid pada tabel tersebut. Agar pengguna aplikasi (pengembang) dapat mengisi masukan model tabel, maka pengembang harus melakukan koneksi terhadap server basis data yang dikehendaki. Aplikasi akan mengambil informasi metadata tabel dari server basis data tersebut. Aplikasi akan mengolah metadata tersebut menjadi model tabel agar bisa digunakan oleh pengembang untuk melakukan abstraksi komponen.

Model tabel juga mengabstraksikan relasi antar tabel basis data (*foreign key*), sehingga komponen MVC yang dihasilkan dapat melakukan operasi data terhadap susunan tabel basis data yang memiliki relasi dengan tabel lainnya. Abstraksi *foreign key* tersebut dilakukan ketika proses pengambilan metadata tabel basis data yang dilakukan oleh aplikasi.

Model kolom memberikan abstraksi bahwa kolom dari tabel basis data mana saja yang akan dipakai pada proses operasi data. Informasi abstraksi kolom tersebut juga meliputi 1) tipe data kolom yang dipakai untuk proses operasi data dan 2) pilihan masukan yang dipakai untuk menampilkan beberapa pilihan pada masukan yang memiliki tipe data *checkbox*, *radio* dan *select-option*.

Komponen MVC yang dihasilkan dapat melakukan operasi data terhadap susunan tabel basis data yang memiliki relasi dengan tabel lainnya. Relasi tersebut diabstraksikan pada model tabel yang menunjukkan bahwa model tabel tersebut

Gambar 5.30 menjelaskan tentang arsitektur dari *program generator* yang akan dibuat sebagai solusi domain. Desain tersebut menjelaskan komponen – komponen penyusun sistem dan fungsi masing – masing komponen.



Gambar 5.30 Desain Program Generator

Komponen penyusun *program generator* terdiri dari *application description*, *program generator* dan *application domain knowledge*. Komponen *application description* digunakan sebagai parameter konfigurasi, sehingga sistem bisa menghasilkan variabilitas aplikasi yang sesuai dengan keinginan pengembang. *Application domain knowledge* digunakan sebagai basis pengetahuan sistem yang bertugas menerjemahkan masukan *application description* menjadi kode program. *Program generator* adalah komponen yang berfungsi untuk mengelola masukan sehingga bisa dimodelkan sesuai dengan arsitektur *application description* yang telah didefinisikan. *Program generator* akan mengirim masukan pengembang kepada *application domain knowledge* untuk diterjemahkan menjadi kode program (*generated program*) yang bisa berinteraksi dengan komponen sistem basis data (*database*).

5.3.1 Perancangan halaman tatap muka

Aplikasi *program generator* yang akan dibuat memiliki tiga buah halaman tatap muka. Halaman tersebut terdiri dari: 1) halaman *login*, 2) halaman parameterisasi model komponen, 3) halaman untuk menampilkan hasil komponen MVC yang telah dibangun dan memungkinkan pengguna untuk mengunduh komponen MVC tersebut.

Halaman pertama adalah halaman *login* yang digunakan oleh pengembang untuk melakukan proses autentikasi. Halaman tersebut berisi form login, jika pengembang berhasil melakukan *login*, maka sistem akan mengarahkan pengembang menuju ke halaman parameterisasi komponen. Gambar 5.30.1 menunjukkan rancangan halaman login.

Gambar 5.30.1 Rancangan Halaman Login

Rancangan halaman login pada Gambar 5.30.1 berisi form login yang berisi masukan *username* dan *password*. Tombol login digunakan untuk melakukan proses *login*.

Rancangan halaman parameterisasi komponen didesain menjadi tiga segmen, yaitu: 1) segmen pertama untuk melakukan parameterisasi masukan model tabel, 2) segmen kedua digunakan untuk melakukan parameterisasi masukan model komponen dan model fitur, 3) segmen ketiga digunakan untuk melakukan parameterisasi model kolom dan model pilihan masukan. Gambar 5.30.2 menunjukkan rancangan tatap muka pada halaman parameterisasi komponen.

Gambar 5.30.2 Rancangan Halaman Parameterisasi Komponen

Berikut ini penjelasan ilustrasi halaman tatap muka pada gambar 5.30.2:

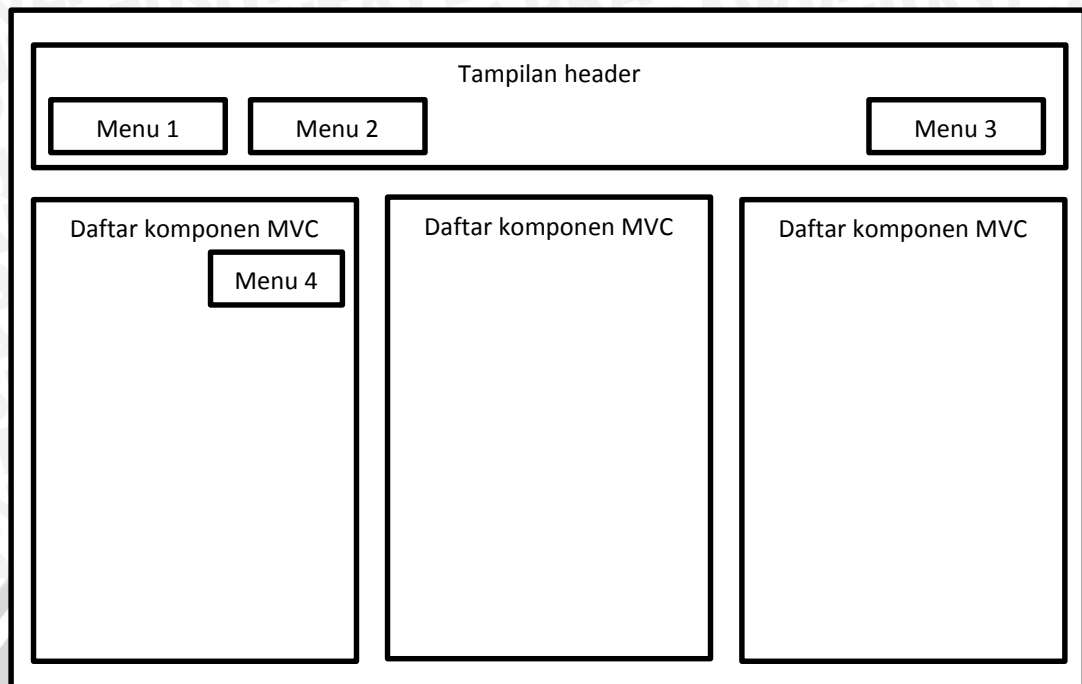
- Tampilan header berisi beberapa opsi untuk mengelola sambungan ke server basis data dan opsi untuk melakukan *logout*.

- Menu 1 adalah opsi yang digunakan untuk memuat ulang halaman parameterisasi model komponen
- Menu 2 adalah opsi yang digunakan untuk menampilkan form untuk melakukan sambungan ke *server* basis data yang ditunjukkan pada Gambar 5.30.3. Jika koneksi berhasil, maka sistem akan memuat metadata skema basis data pada bagian 'Dropdown skema basis data'. Jika pengguna memiliki salah satu dari skema tersebut, maka sistem akan menampilkan detail model tabel basis data pada bagian 'Daftar model tabel yang dimuat skema basis data dipilih'
- Menu 3 adalah opsi yang digunakan untuk memuat ulang sambungan ke *server* basis data
- Menu 4 adalah opsi yang digunakan untuk memutuskan sambungan *server* basis data
- Menu 5 adalah opsi yang digunakan untuk melakukan proses *logout*
- Menu 6 adalah opsi yang digunakan untuk memproses masukan model komponen menjadi komponen MVC, kemudian sistem akan mengarahkan ke halaman untuk menampilkan hasil pembangkitan komponen.
- Menu 7 adalah opsi yang digunakan untuk menambahkan model komponen baru

The diagram illustrates the layout of the database connection form. It features a header section with five menu items (Menu 1 to Menu 5). The main body is organized into three vertical segments. Segment 1 contains a text input field for the database server address. Segment 2 contains two text input fields for the database username and password. Segment 3 displays a list of database schemas selected from a dropdown menu. At the bottom of the form, there are two buttons labeled 'Tombol 1' and 'Tombol 2'.

Gambar 5.30.3 Rancangan Form Sambungan ke *Server* Basis Data

Gambar 5.30.3 menjelaskan tentang rancangan untuk melakukan sambungan ke *server* basis data. Form sambungan tersebut terdiri dari masukan alamat *server* basis data, masukan *username* *server* basis data dan masukan *password* *server* basis data.



Gambar 5.30.4 Rancangan Halaman Hasil Pembangkitan Komponen MVC

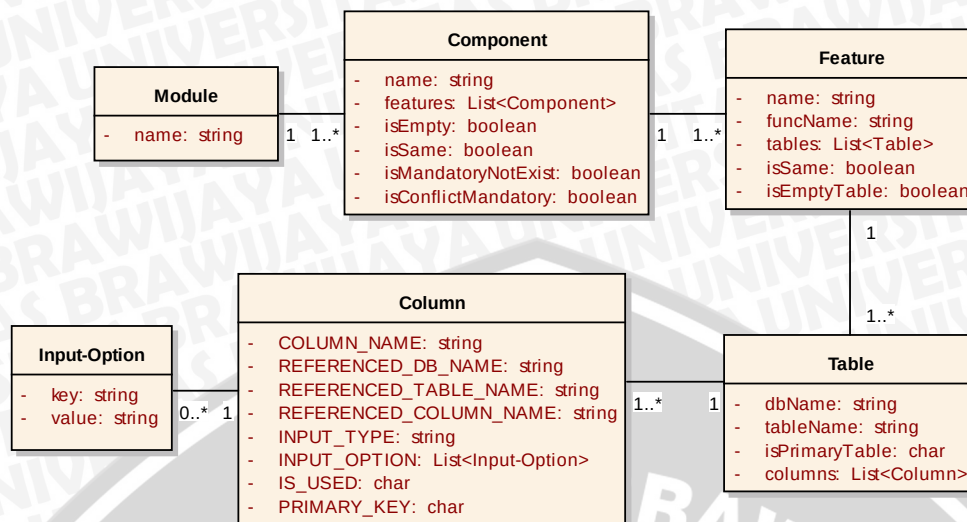
Rancangan halaman hasil pembangkitan komponen MVC ditunjukkan pada Gambar 5.30.4. Berdasarkan ilustrasi pada rancangan tersebut, beberapa komponen MVC ditampilkan di bawah komponen *header* halaman dan disertai opsi untuk mengunduh komponen MVC tersebut. Pada tampilan *header* terdapat beberapa opsi. Berikut ini penjelasan opsi – opsi pada Gambar 5.30.4:

- Menu 1 digunakan untuk mengarahkan pengembang menuju ke halaman parameterisasi komponen
- Menu 2 digunakan untuk mengunduh semua komponen MVC yang telah dibangkitkan
- Menu 3 digunakan untuk melakukan proses *logout*
- Menu 4 digunakan untuk mengunduh komponen MVC pada bagian yang bersangkutan

5.3.2 Perancangan *application description*

Application description adalah sebuah model yang digunakan untuk mendeskripsikan struktur komponen sistem secara abstrak yang akan diproses oleh sistem *program generator*, sehingga bisa dihasilkan komponen sistem konkret sesuai dengan abstraksi model tersebut. Rancangan *application description model* dibuat berdasarkan hasil analisis domain.

Application description model dideskripsikan dengan *class diagram* pada Gambar 5.31 berikut ini:



Gambar 5.31 Gambar Application Description Model

Berikut ini penjelasan tiap kelas pada Gambar 5.31 berserta fungsi dalam melakukan abstraksi kode program yang akan dibangkitkan:

1. Kelas *Module*: berfungsi untuk abstraksi komponen modules, kelas *module* akan dijadikan sebagai parameter dalam membangkitkan nama modul pada komponen *controller* dan *model*
2. Kelas *Component*: berfungsi untuk abstraksi komponen *controlle* dan *model* yang akan dibangkitkan, satu kelas *component* akan dibangkitkan menjadi satu komponen *controller* dan *model*.
3. Kelas *Feature*: berfungsi untuk abstraksi fungsi – fungsi yang akan dibangkitkan pada komponen *controller* dan *model*, yaitu fungsi: *list* data, *filter* data, hapus data, perbarui data, detail data dan tambah data. Satu komponen *feature* akan dipetakan untuk menghasilkan komponen view sesuai dengan jenis *feature*.
4. Kelas *Table*: digunakan untuk abstraksi tabel basis data yang akan dioperasikan oleh komponen *controller*, *model* dan *view* (MVC) yang telah dibangkitkan. Satu model *feature* memiliki beberapa model *table* dan salah satu dari model *table* tersebut harus memiliki atribut *primaryTable* = 1
5. Kelas *Column*: berfungsi untuk abstraksi kolom yang ada pada tabel basis data. Abstraksi kolom dimodelkan dengan atribut nama kolom, hubungan dengan kolom lain, kolom *primary key* dan atribut apakah kolo tersebut dipakai dalam proses pembangkitan atau tidak
6. Kelas *Input-Option*: digunakan untuk abstraksi dari pilihan masukan suatu kolom jika tipe masukannya *dropdown*, *checkbox* atau *radio button*

5.3.3 Perancangan *application domain knowledge*

Application domain knowledge berisi basis pengetahuan tentang domain PTIIK-Apps. Domain knowledge digunakan untuk memproses masukan yang diterima menjadi kode program yang bisa dieksekusi pada domain PTIIK-Apps. **Model fitur**

yang telah didefinisikan digunakan sebagai panduan dalam merancang fitur – fitur yang harus dipenuhi oleh *application domain knowledge*. **Model fungsional** yang telah didefinisikan digunakan sebagai acuan untuk membuat diagram aktivitas *application domain knowledge*. **ER-model** yang telah didefinisikan digunakan sebagai acuan dalam membuat komponen *application domain knowledge*.

Application domain knowledge pada penelitian ini berupa file dengan ekstensi .php yang dibedakan menjadi tiga kategori, yaitu:

1. *controller knowledge*: basis pengetahuan untuk menghasilkan kode program komponen *controller*.
2. *model knowledge*: basis pengetahuan untuk menghasilkan kode program *model controller*.
3. *view knowledge*: basis pengetahuan untuk menghasilkan kode program komponen *view*.

View knowledge dibagi menjadi lima macam kategori, yaitu:

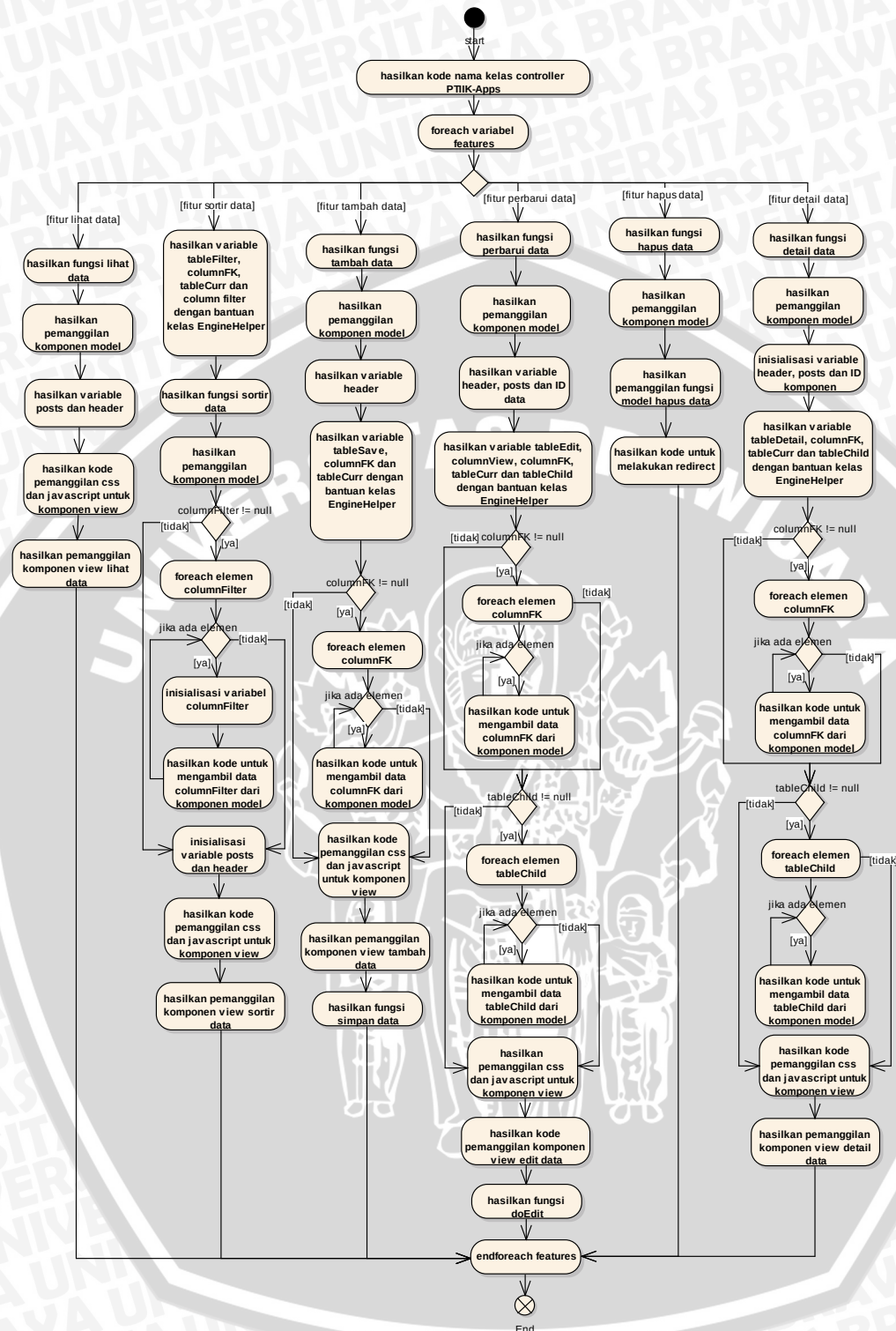
1. *view list knowledge*: basis pengetahuan untuk komponen tatap muka untuk menampilkan daftar data suatu *entity*.
2. *view filter knowledge*: basis pengetahuan untuk komponen tatap muka untuk menampilkan daftar data suatu *entity* disertai fitur sortir (*filter*).
3. *view detail knowledge*: basis pengetahuan untuk komponen tatap muka untuk menampilkan detail data suatu *entity*.
4. *view edit knowledge*: basis pengetahuan untuk komponen tatap muka untuk mengubah data suatu *entity*.
5. *view save knowledge*: basis pengetahuan untuk komponen tatap muka untuk menambahkan data suatu *entity*.

Perancangan *application domain knowledge* diawali dengan penjabaran *activity diagram* untuk menjabarkan alur kerja *application domain knowledge*. Penjabaran *sequence diagram* dapat dilakukan dengan menganalisis rancangan *activity diagram* yang sudah dibuat untuk mengetahui komponen – komponen kelas yang akan dibutuhkan untuk proses implementasi.

5.3.3.1 Perancangan aktivitas *application domain knowledge*

1. *Activity diagram controller knowledge*

Gambar 5.32 menunjukkan *activity diagram* pada *controller knowledge*. *Controller knowledge* membaca masukan pada variabel *feature* kemudian memilah proses pembangkitan kode program berdasarkan enam macam fitur yang dikenali (yang telah didefinisikan pada *domain model*).



Gambar 5.32 Activity Diagram Controller Knowledge

Gambar 5.32 menjelaskan alur kerja *controller knowledge* dalam menghasilkan kode program komponen *controller*. Alur utama dilakukan dengan membaca setiap variabel *feature*, yaitu variabel yang berisi sekumpulan abstraksi fitur. Variabel *feature* akan diseleksi berdasarkan jenis fitur dan akan dihasilkan kode

program yang sesuai. Berikut ini penjelasan istilah dan penamaan variabel pada Gambar 5.3:

- Setiap fungsi komponen controller akan melakukan inisialisasi komponen *model*
- Variabel *feature* adalah variabel yang berisi sekumpulan model fitur
- Operasi *foreach* adalah operasi yang digunakan untuk membaca setiap elemen variabel *array* (suatu variabel yang memiliki beberapa atribut)
- Proses *redirect* adalah proses untuk mengarahkan proses dari suatu fungsi ke fungsi lain
- Variabel *header* digunakan sebagai identitas jenis fitur ketika memanggil komponen *view*
- Variabel *posts* berisi data tabel utama yang akan ditampilkan pada komponen *view*
- Variabel *tableFilter* berisi model tabel yang berada pada model fitur sortir data
- Variabel *tableSave* berisi model tabel yang berada pada model fitur tambah data
- Variabel *tableEdit* berisi model tabel yang berada pada model fitur perbarui data
- Variabel *tabelDetail* berisi model tabel yang berada pada model fitur detail data
- Variabel ID data berisi kunci utama (*primary key*) dari sebuah data yang akan dioperasikan
- Variabel *columnFK* berisi sekumpulan model kolom yang memiliki hubungan dengan kolom pada tabel cabang
- Variabel *tableCurr* merupakan model tabel didefinisikan sebagai tabel utama, yaitu tabel yang menjadi acuan dalam melakukan operasi data
- Variabel *tableChild* merupakan variabel *array* yang berisi sekumpulan model tabel cabang, yaitu tabel yang berhubungan dengan tabel utama
- Atribut model kolom *IS_USED* digunakan sebagai indikasi apakah model kolom tersebut akan diproses atau tidak

Gambar 5.33 menjelaskan tentang proses generasi kode program pada fitur simpan data, yang merupakan detail lanjutan dari Gambar 5.32 pada proses hasilkan **fungsi simpan data**. Pembangkitan kode program simpan data terdiri dari dua tahap utama, yaitu 1) pembangkitan kode program berdasarkan tabel utama 2) dan pembangkitan kode program berdasarkan tabel cabang. Berikut ini penjelasan istilah pada Gambar 5.33:

- Variabel data array tabel utama/cabang yang akan disimpan: adalah sekumpulan atribut yang mewakili kolom tabel basis data yang akan dikirim ke

Variabel `primaryColumn`: adalah variabel yang digunakan untuk menyimpan nama kolom kunci tabel utama

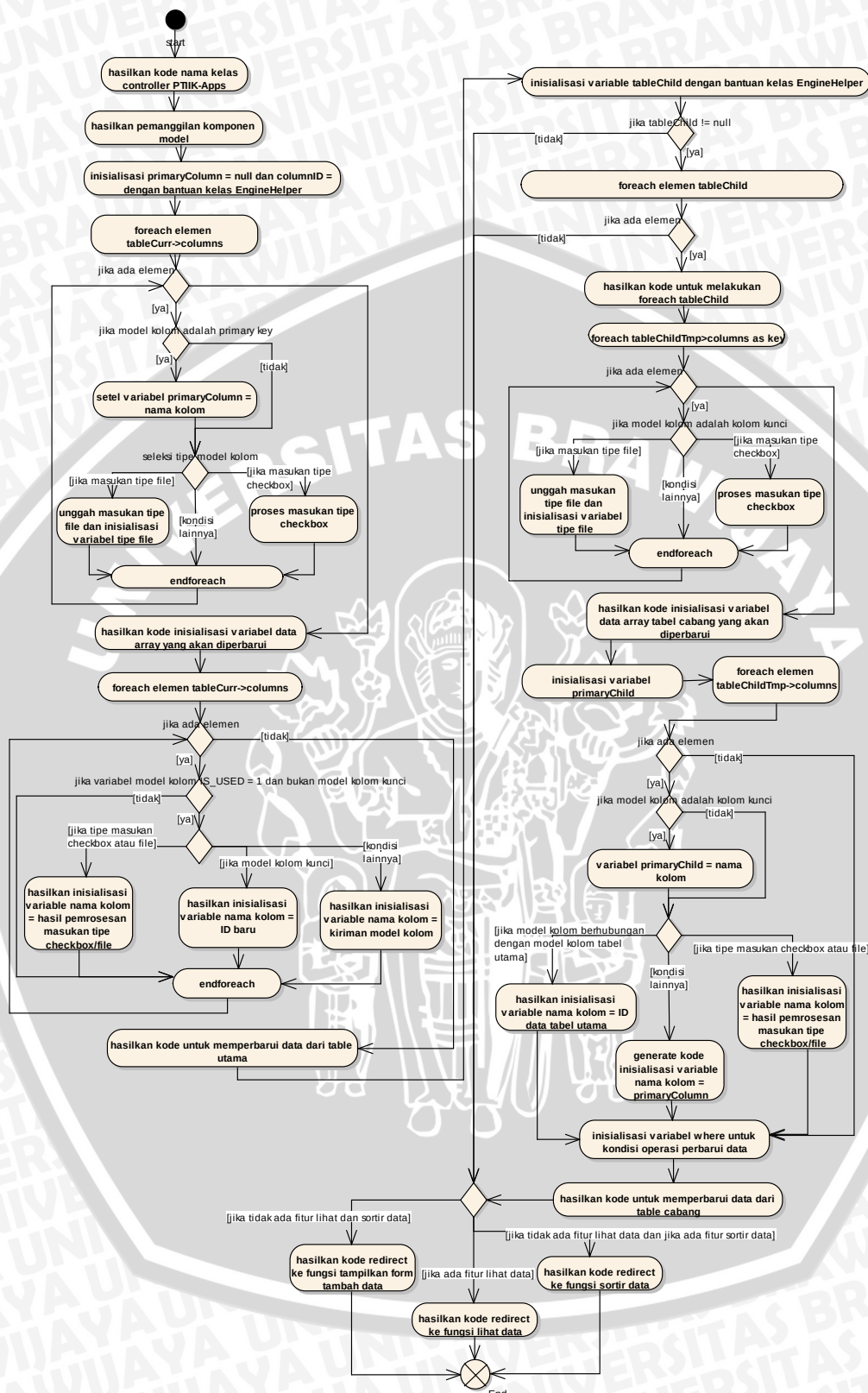
Variabel `primaryColumnChild`: adalah variabel yang digunakan untuk menyimpan nama kolom kunci pada setiap tabel cabang



Gambar 5.34 menjelaskan tentang proses generasi kode program pada fitur ubah data (detail lanjutan dari Gambar 5.32). Pembangkitan kode program ubah data terdiri dari dua tahap utama, yaitu 1) pembangkitan kode program berdasarkan tabel utama 2) dan pembangkitan kode program berdasarkan tabel cabang. Berikut ini penjelasan istilah yang dipakai pada Gambar 5.34:

- Variabel *primaryChild* adalah variabel yang menyimpan kolom kunci pada setiap tabel cabang
- Variabel *array where* digunakan sebagai kondisi dalam melakukan proses perbarui data



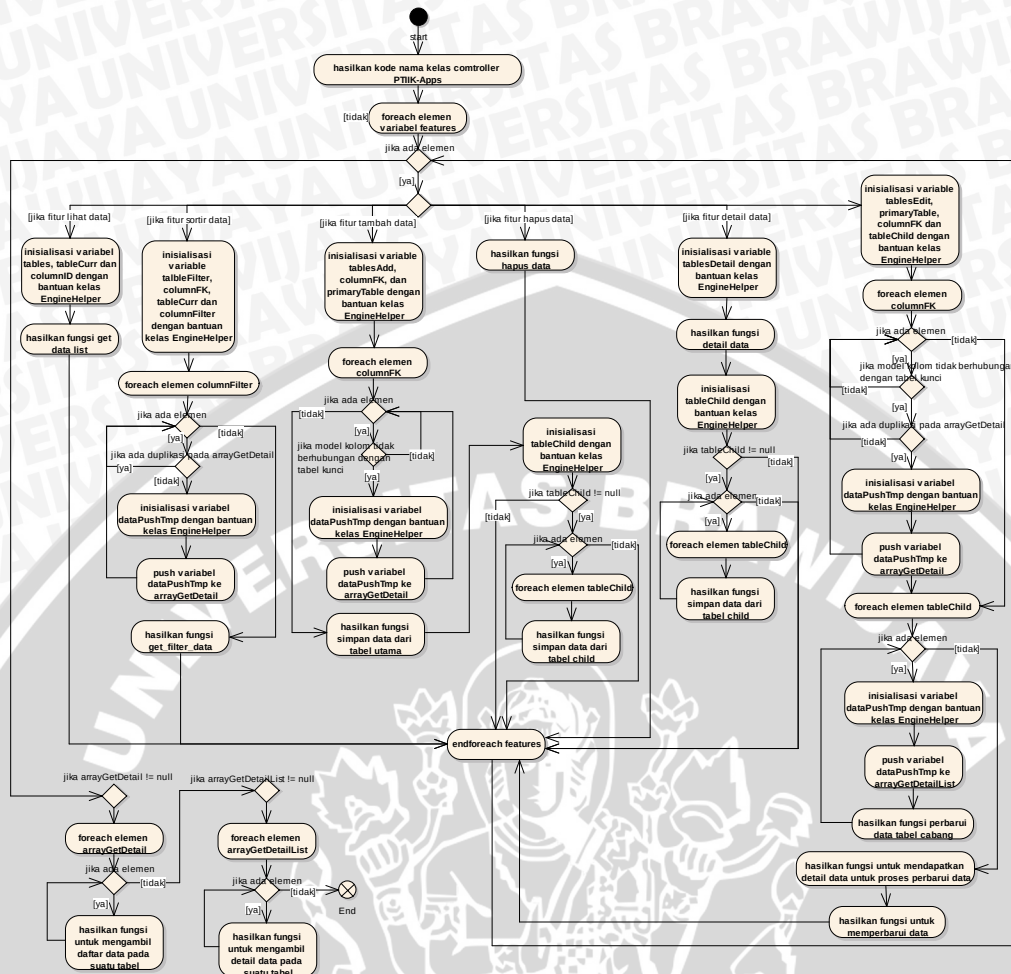


Gambar 5.34 Activity Diagram Proses Ubah Data

2. Activity diagram model knowledge

Gambar 5.35 menunjukkan *activity diagram* pada *model knowledge*. *Model knowledge* membaca masukan pada variabel *feature* kemudian memilah proses pembangkitan kode program berdasarkan enam macam fitur yang dikenali (yang telah didefinisikan pada model domain). Di akhir proses pembacaan variabel *feature*, *model knowledge* melakukan pembangkitan kode program untuk fungsi – fungsi umum pada komponen *model*, yaitu 1) mendapatkan detail data dari suatu tabel dan 2) mendapatkan daftar data dari suatu tabel. Penjelasan istilah dan nama variabel yang digunakan pada Gambar 5.35 sebagian telah dijelaskan pada sub bab sebelumnya, berikut istilah dan nama variabel lain yang belum dijelaskan pada Gambar 5.35:

- Variabel *tables* berisi sekumpulan model tabel pada fitur lihat data
- Variabel *tableCurr* berisi model tabel utama pada sebuah model fitur
- Variabel *columnID* berisi model kolom yang merupakan model kolom kunci (*primary key*)
- Variabel *dataPushTmp* berisi model tabel yang berhubungan dengan tabel yang sedang diproses, variabel tersebut akan ditambahkan ke indeks variabel *arrayGetDetail* atau *arrayGetDetailList*
- Variabel *arrayGetDetail* berisi sekumpulan model tabel yang akan diproses untuk mengambil daftar data pada tabel tersebut
- Variabel *arrayGetDetailList* berisi sekumpulan model tabel yang akan diproses untuk mengambil detail data pada tabel tersebut
- Fungsi *get_filter_data* adalah fungsi yang digunakan untuk mengambil daftar data dari suatu tabel berdasarkan parameter sortir yang dikirim



Gambar 5.35 Activity Diagram Model Knowledge

3. Activity diagram view list knowledge

Gambar 5.36 menunjukkan *activity diagram* pada *view list knowledge*. *View list knowledge* memproses data masukan dari variabel *columnView* untuk menampilkan daftar model kolom pada suatu tabel. Berikut ini penjelasan istilah dan nama variabel yang belum dijelaskan sebelumnya pada Gambar 3.36:

- Tampilan *header* adalah tampilan tatap muka bagian atas, tampilan tersebut juga berisi fungsi – fungsi untuk inialisasi halaman tatap muka
- Tampilan *foot* adalah tampilan tatap muka bagian bawah, tampilan tersebut berisi deklarasi fungsi – fungsi umum *javascript* yang digunakan pada PTIHK-Apps

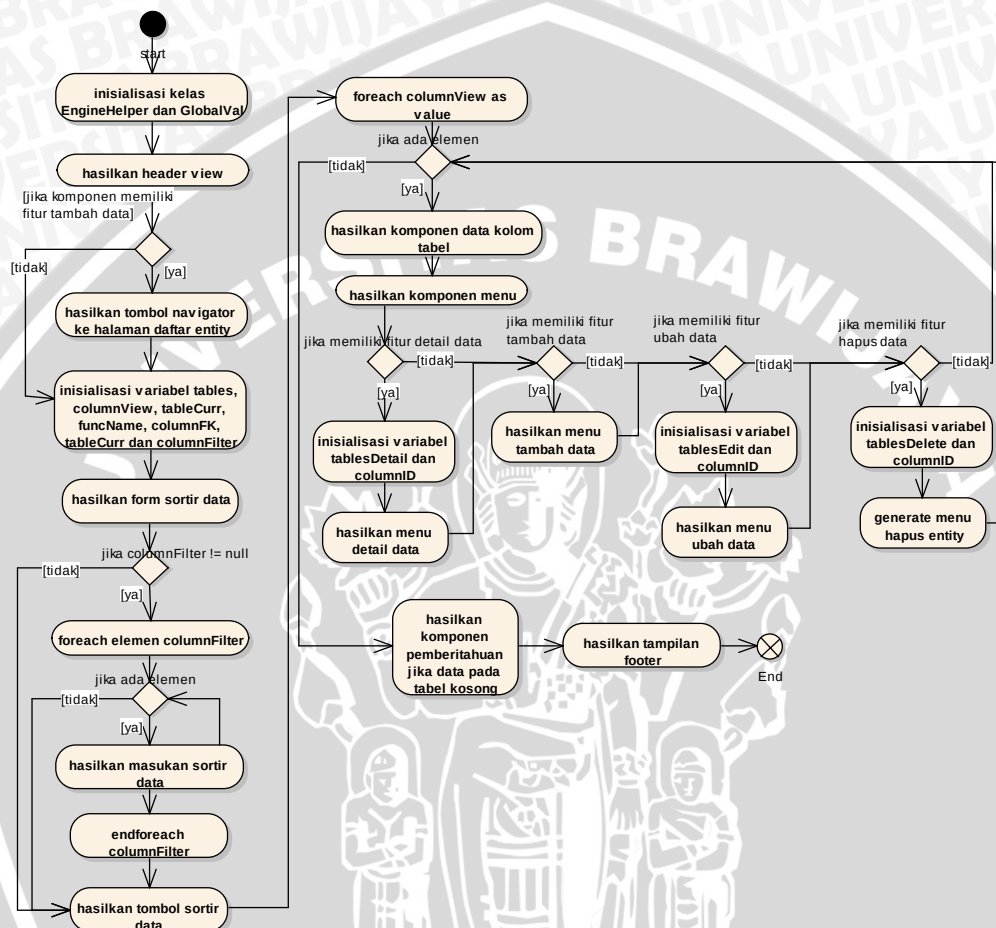


Gambar 5.37 menunjukkan *activity diagram* pada *view detail knowledge*. *View detail knowledge* memproses masukan untuk menampilkan detail data pada tabel utama dan tabel cabang.



5. Activity diagram view filter knowledge

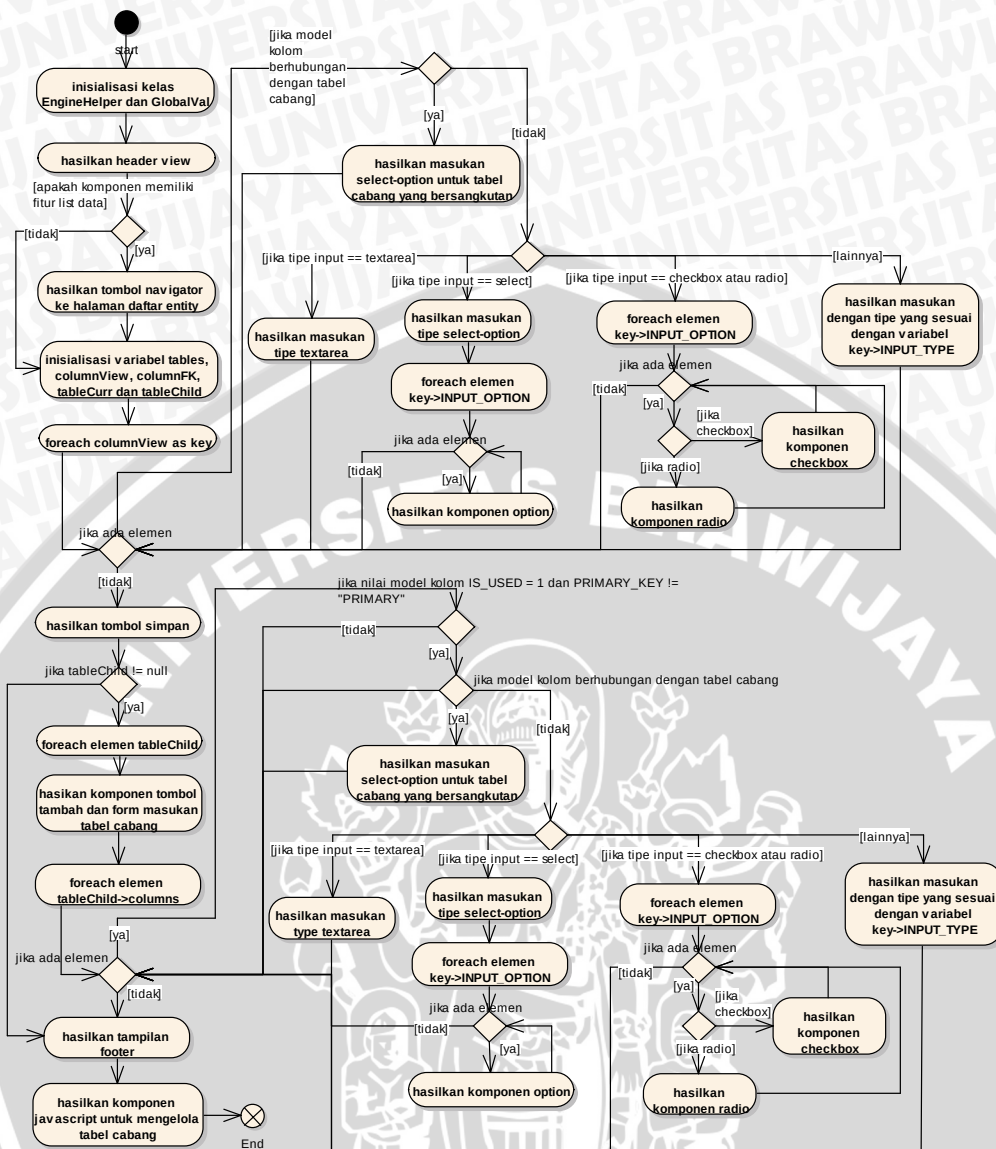
Gambar 5.38 menunjukkan *activity diagram* pada *view filter knowledge*. *View filter knowledge* memproses data sama seperti *view list knowledge*, namun keluaran yang dihasilkan berbeda. Sebelum memproses data variabel *columnView*, *view filter knowledge* membangkitkan form sortir data yang diproses dari variabel *columnFilter*.



Gambar 5.38 Activity Diagram View Filter Knowledge

6. Activity diagram view save knowledge

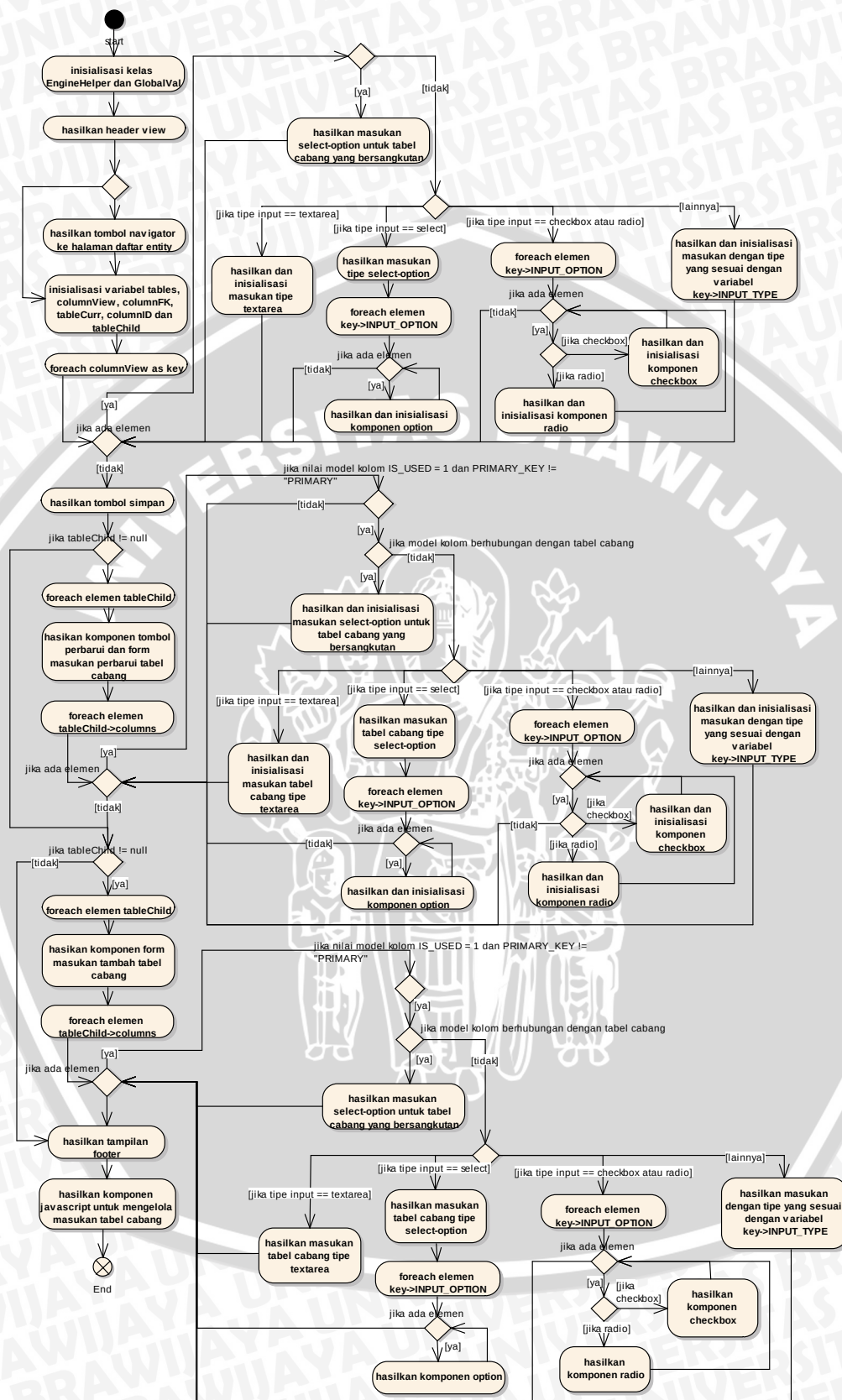
Gambar 5.39 menunjukkan *activity diagram* pada *view save knowledge*. *View save knowledge* memproses masukan menjadi kode program penyusun tatap muka fitur simpan data. *View save knowledge* mengolah variabel *columnView* untuk menghasilkan form masukan ke tabel utama, dan mengolah variabel *tableChild* untuk menghasilkan form masukan ke tabel cabang.



Gambar 5.39 Activity Diagram View Save Knowledge

7. Activity diagram view edit knowledge

Gambar 5.40 menunjukkan *activity diagram* pada *view edit knowledge*. *Activity diagram edit knowledge* memiliki alur proses yang hampir sama dengan *view save knowledge*. Perbedaannya ketika proses menampilkan form perbarui data disertai inisialisasi dari nilai data tabel yang aktual. Pada proses menampilkan data tabel cabang juga disertai proses inisialisasi nilai tabel cabang yang aktual.



5.3.3.2 Perancangan urutan *domain knowledge*

Sequence diagram digunakan untuk menjabarkan urutan proses yang dilakukan oleh komponen kode program yang akan dibuat, sehingga membantu proses pemetaan *class diagram*.

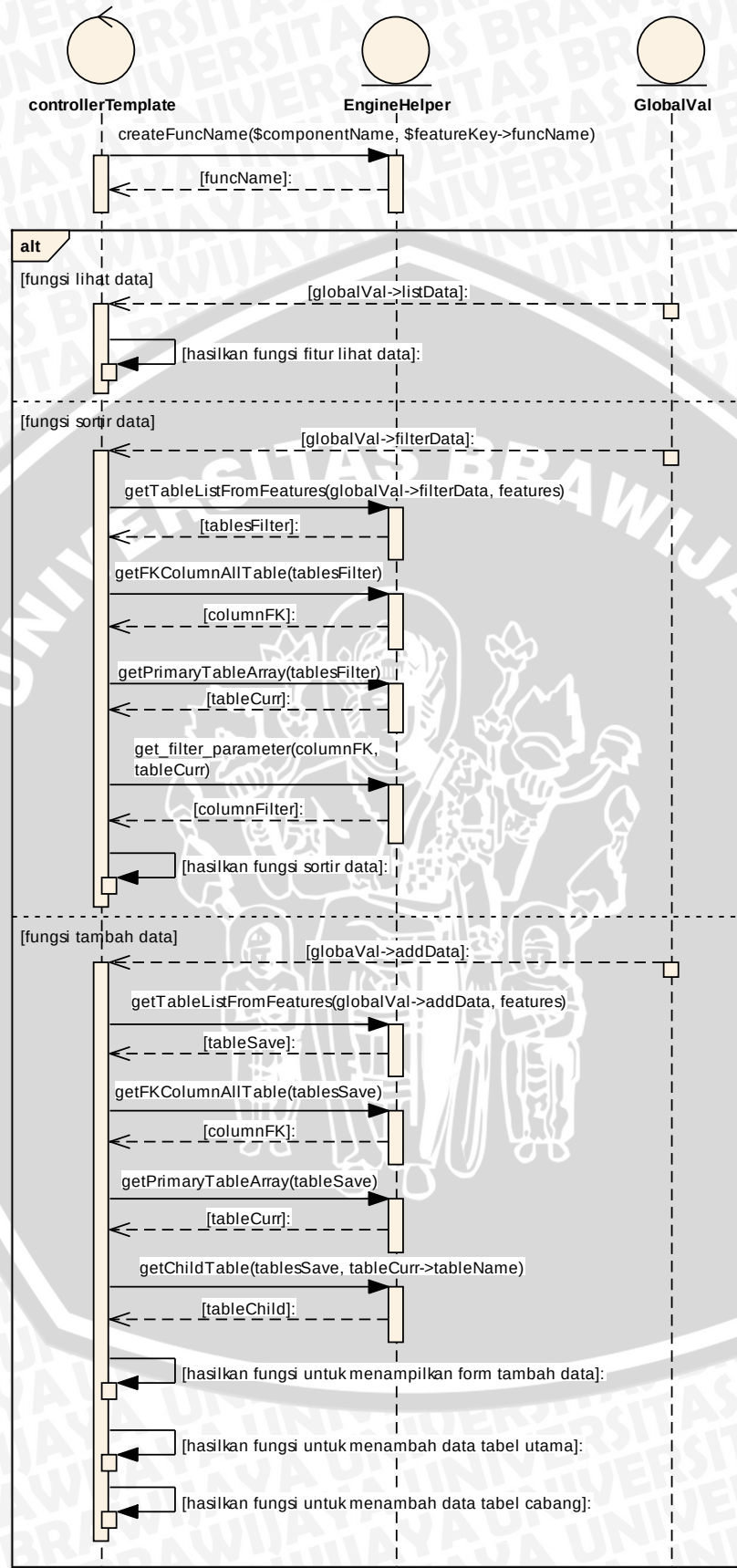
1. *Sequence diagram controller knowledge*

Sequence diagram pada *controller knowledge* melibatkan peran tiga komponen dalam prosesnya, yaitu:

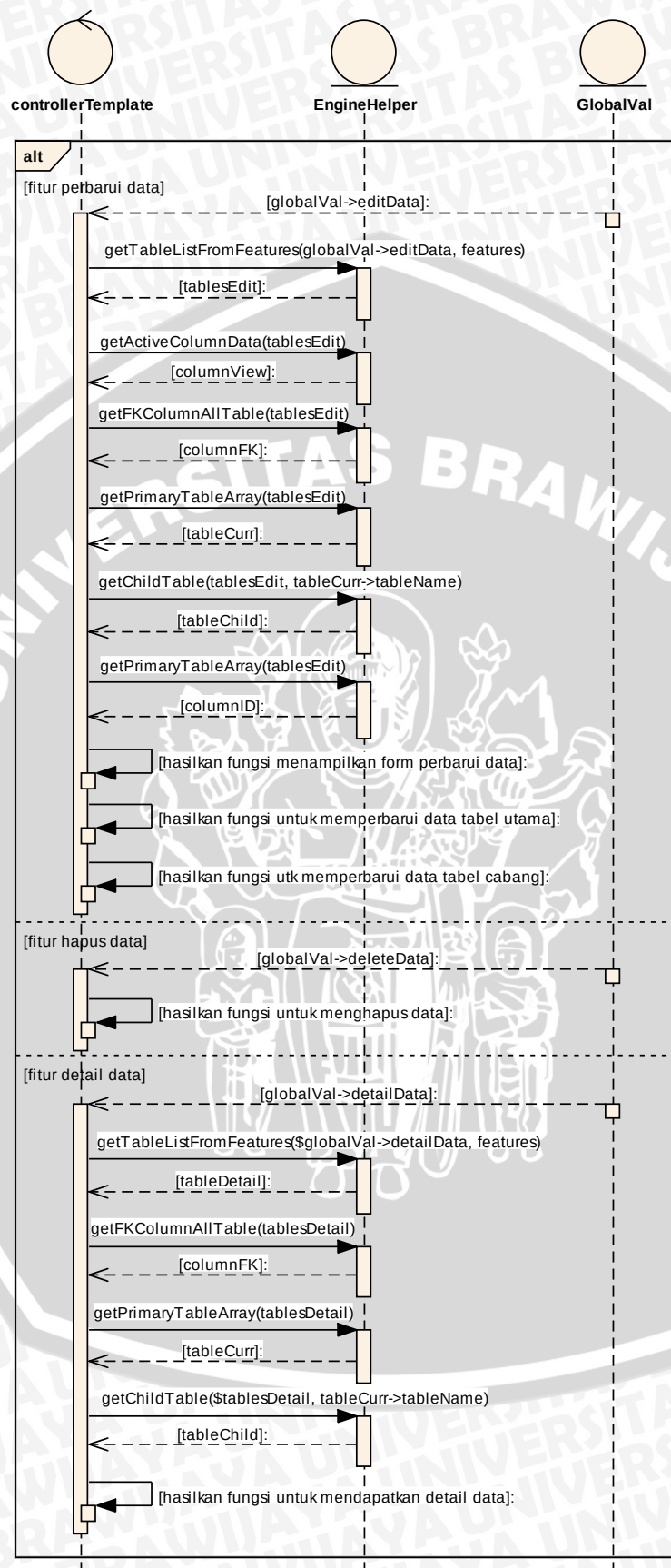
1. *controllerTemplate*: digunakan untuk menjalankan fungsi – fungsi *controller knowledge*.
2. *EngineHelper*: digunakan untuk membantu *controllerTemplate* menjalankan fungsi – fungsi umum (yang sering digunakan oleh *domain knowledge*).
3. *GlobalVal*: digunakan untuk memanggil nilai dari variabel – variabel global di dalam sistem, sehingga sistem mendapatkan perspektif yang sama dalam mengenali nilai variabel global.

Sequence diagram controller knowledge dibagi menjadi dua bagian, yaitu pada Gambar 5.41 dan Gambar 5.42.





Gambar 5.41 Sequence Diagram Controller Knowledge (bagian 1)

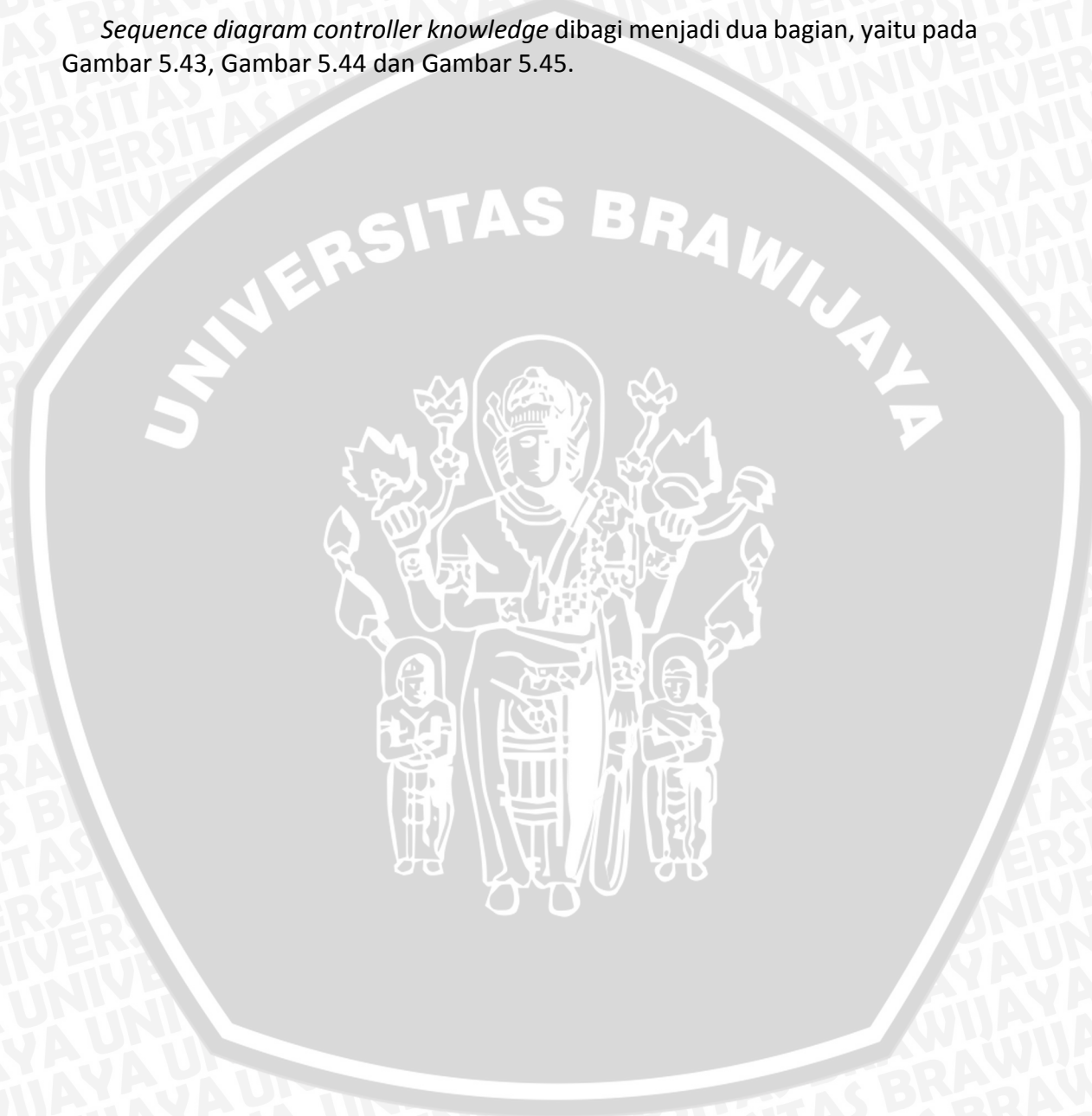


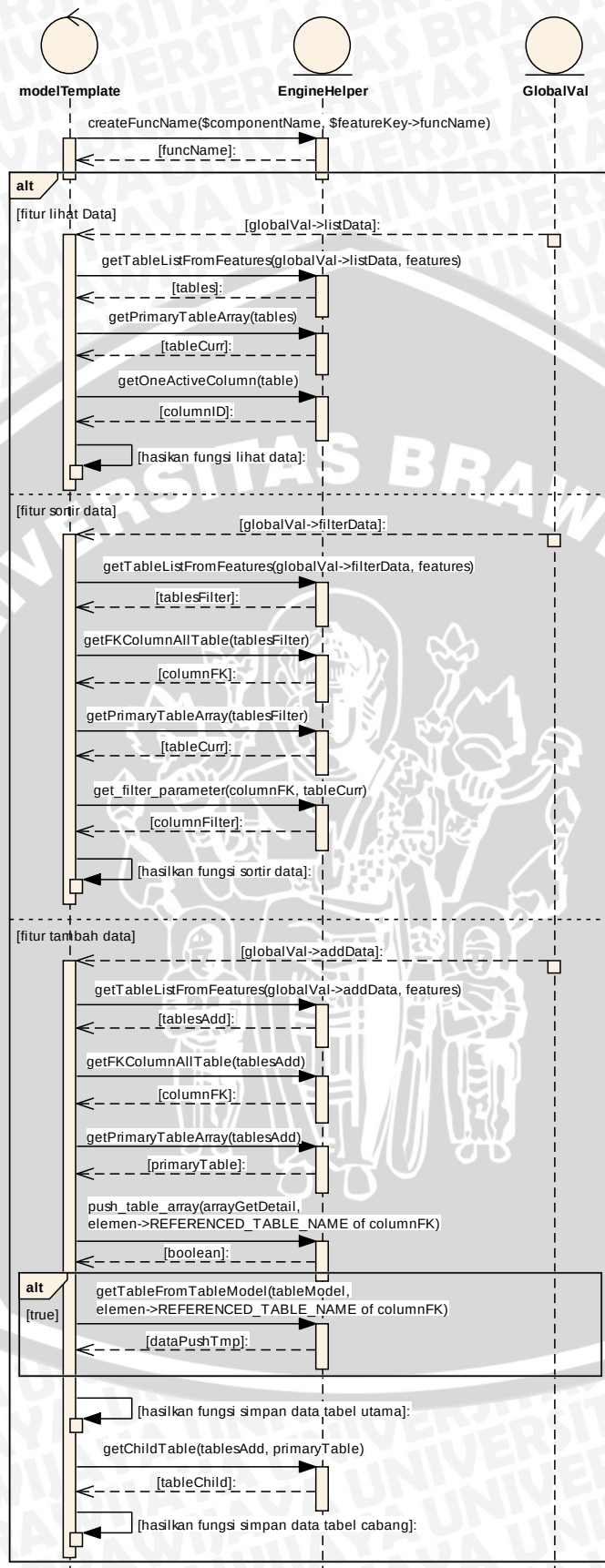
Gambar 5.42 Sequence Diagram Controller Knowledge (bagian 2)

2. Sequence diagram model knowledge

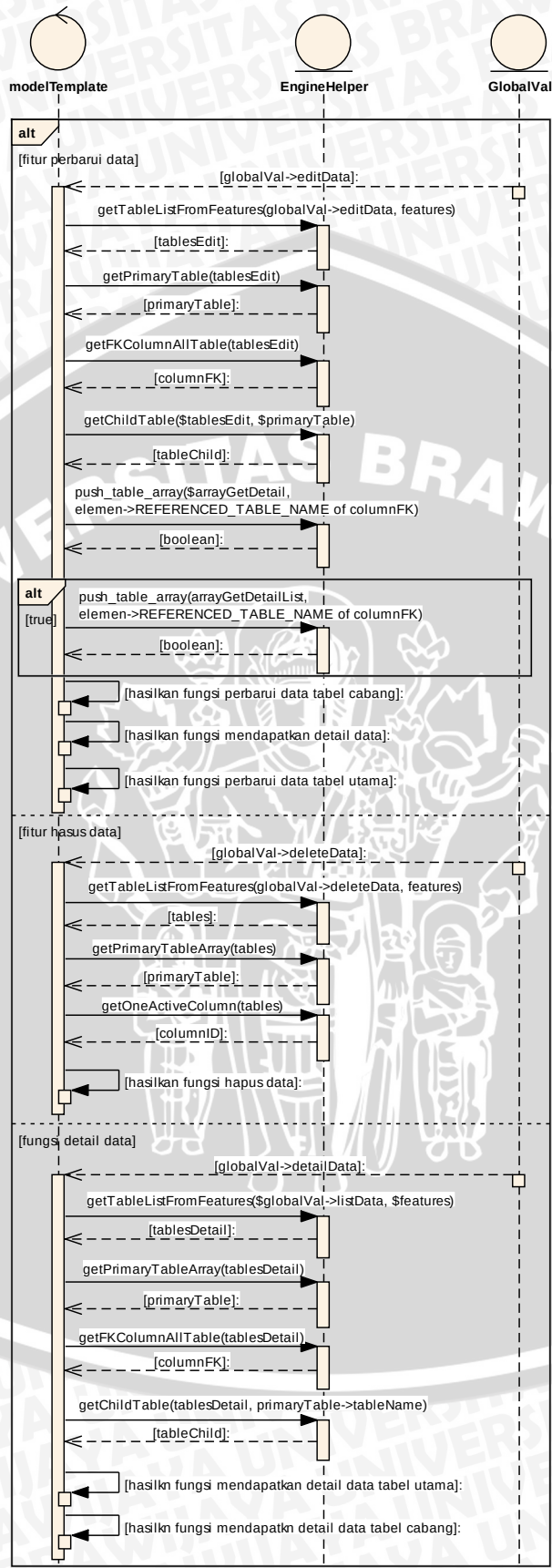
Sequence diagram pada *model knowledge* melibatkan peran tiga komponen dalam prosesnya, yaitu: 1) *modelTemplate*, 2) *EngineHelper* dan 3) *GlobalVal*. Komponen *modelTemplate* digunakan untuk menjalankan fungsi – fungsi dari *model knowledge*. Variabel ***tableModel*** adalah variabel yang berisi sekumpulan model tabel yang diambil dari basis data yang dipilih oleh pengembang.

Sequence diagram controller knowledge dibagi menjadi dua bagian, yaitu pada Gambar 5.43, Gambar 5.44 dan Gambar 5.45.

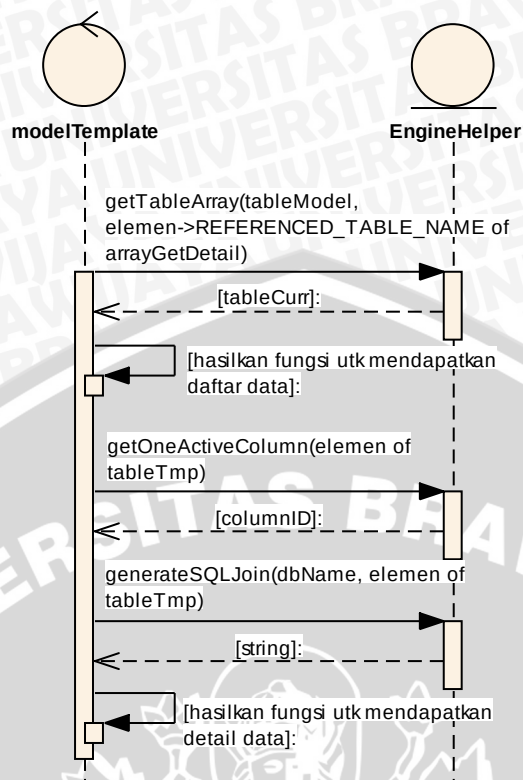




Gambar 5.43 Sequence Diagram Model Knowledge (bagian 1)



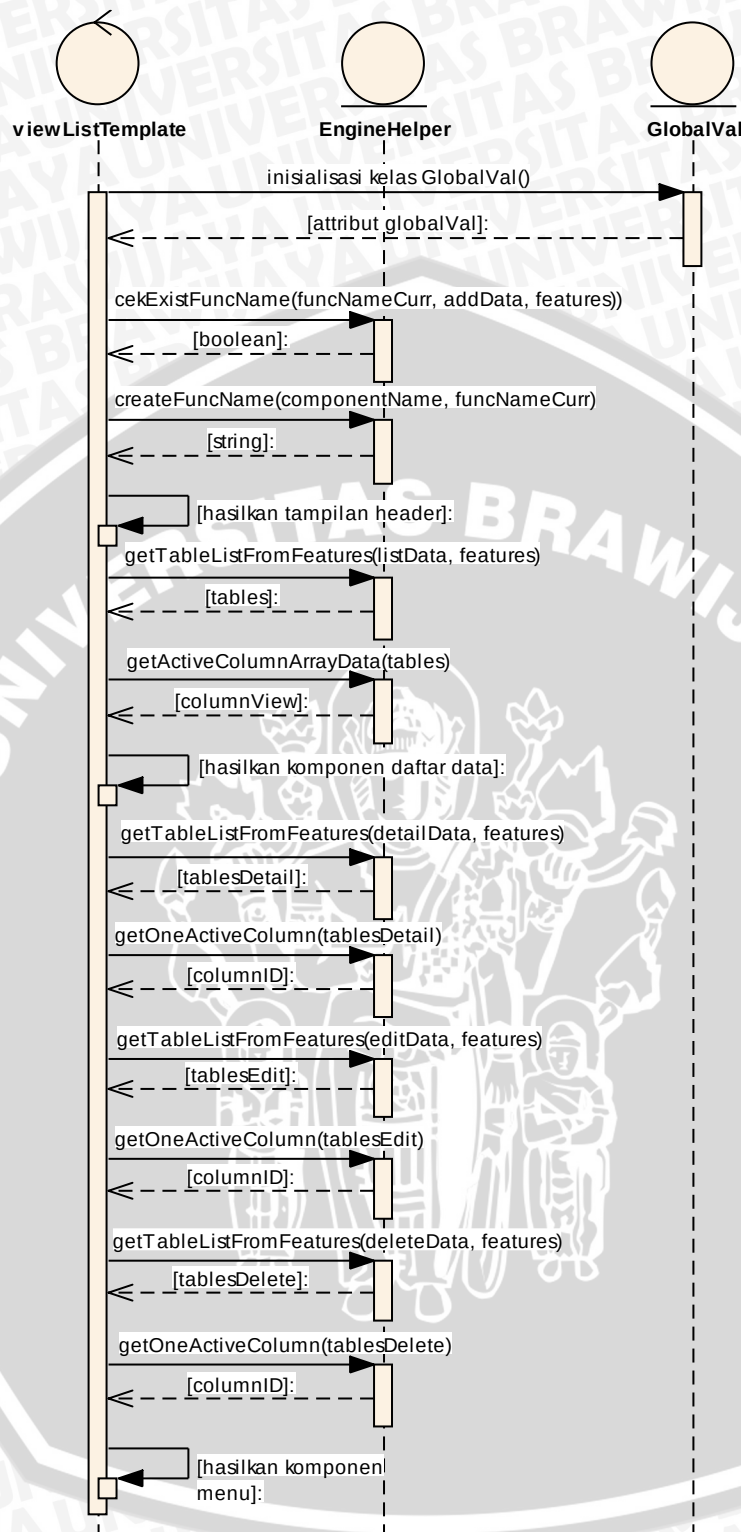
Gambar 5.44 Sequence Diagram Model Knowledge (bagian 2)



Gambar 5.45 Sequence Diagram Model Knowledge (bagian 3)

3. Sequence diagram view list knowledge

Gambar 5.46 merupakan *sequence diagram* pada *view list knowledge* melibatkan peran tiga komponen dalam prosesnya, yaitu: 1) *viewListTemplate*, 2) *EngineHelper* dan 3) *GlobalVal*. Komponen *viewListTemplate* digunakan untuk menjalankan fungsi – fungsi dari *view list knowledge*.

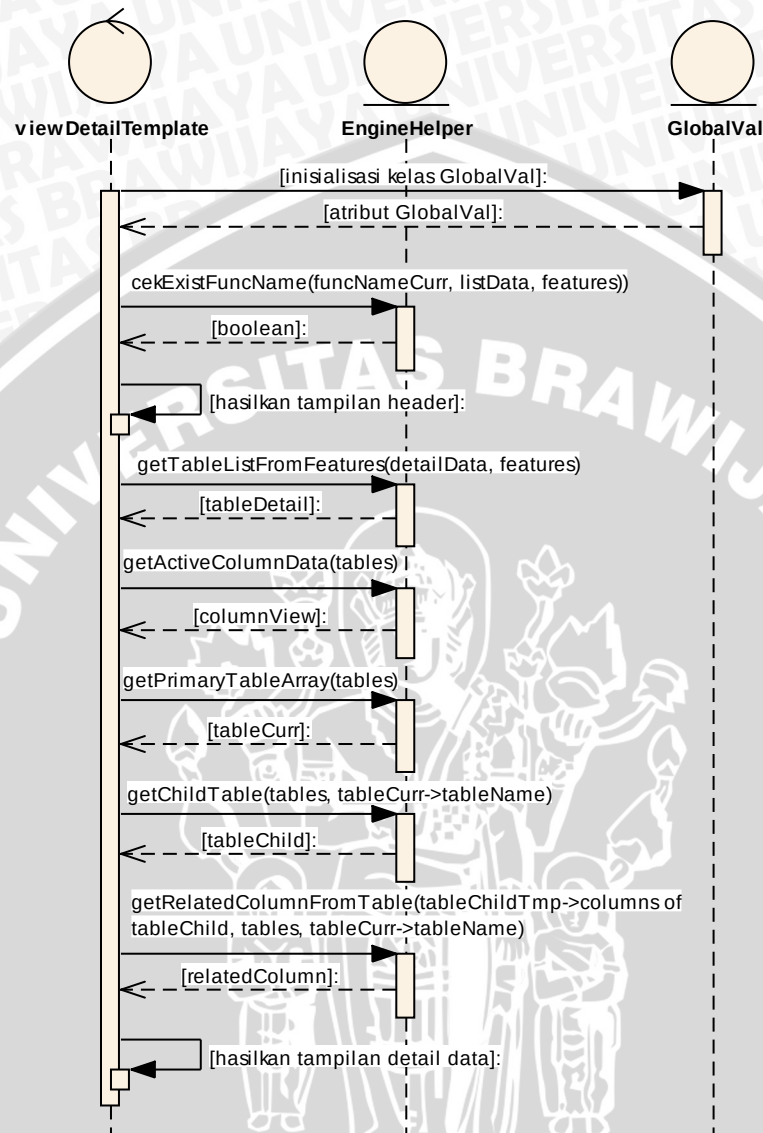


Gambar 5.46 Sequence Diagram View List Knowledge

4. Sequence diagram view detail knowledge

Gambar 5.47 menunjukkan *sequence diagram view detail knowledge*, yang melibatkan tiga komponen, yaitu: 1) *viewDetailTemplate*, 2) *EngineHelper* dan 3) *GlobalVal*. Komponen *viewDetailTemplate* digunakan untuk menjalankan fungsi –

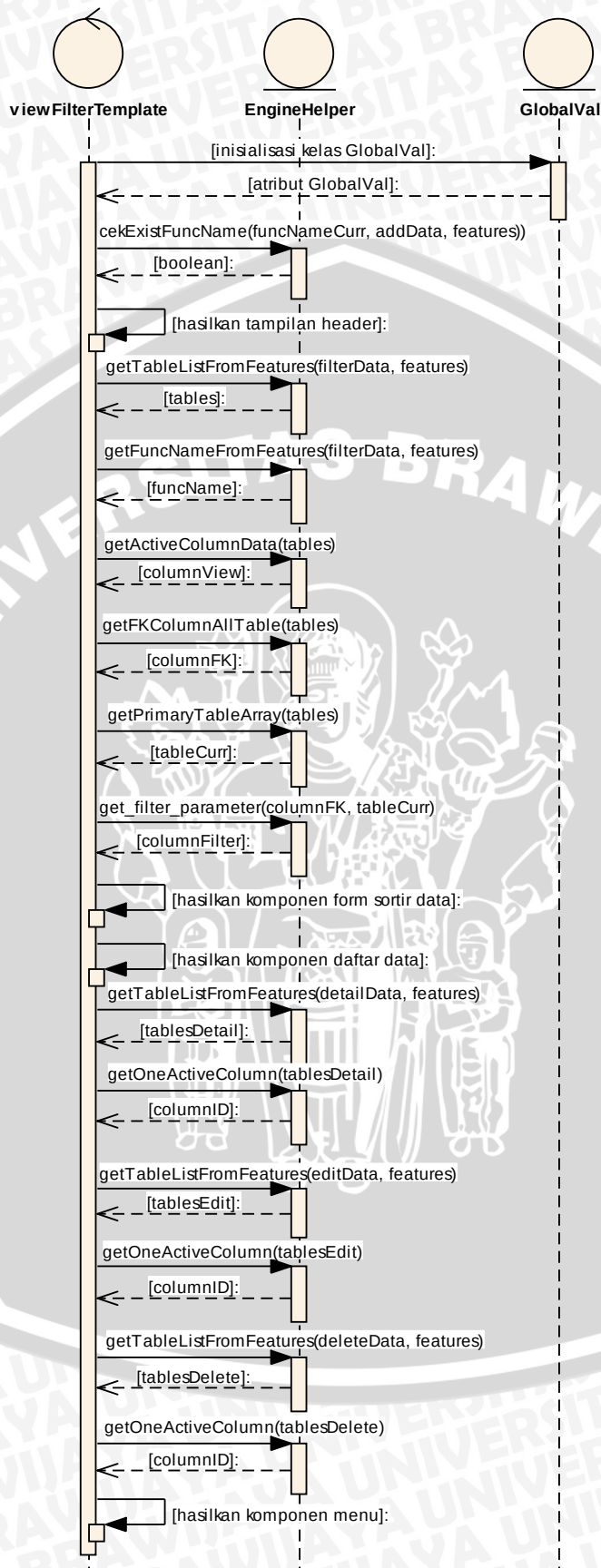
fungsi dari *view detail knowledge*. Fungsi ***getRelatedColumnFromTable*** akan mengembalikan variabel ***relatedColumn*** yang berisi sekumpulan model kolom yang memiliki hubungan dengan tabel cabang kecuali dengan tabel utama.



Gambar 5.47 Sequence Diagram View Detail Knowledge

5. Sequence diagram view filter knowledge

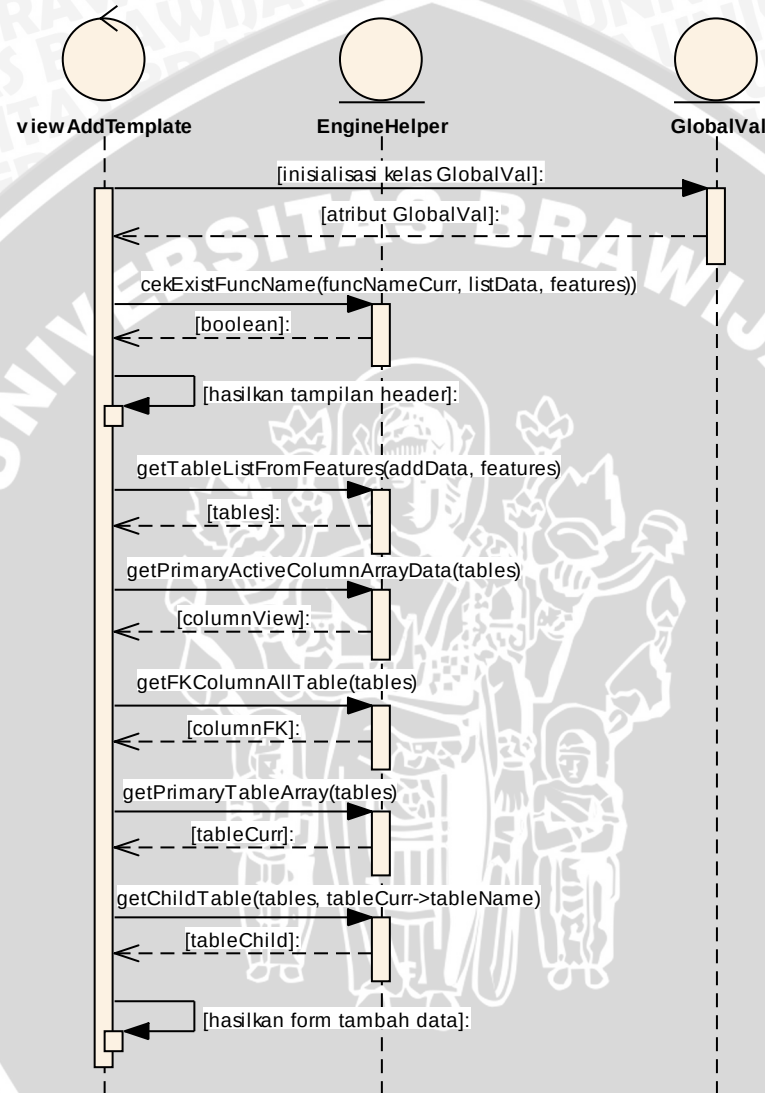
Sequence diagram view filter knowledge melibatkan tiga komponen, yaitu: 1) *viewFilterTemplate*, 2) *EngineHelper* dan 3) *GlobalVal*. Komponen *viewFilterTemplate* digunakan untuk menjalankan fungsi – fungsi dari *view filter knowledge*. Gambar 5.48 menunjukkan *sequence diagram view filter knowledge*.



Gambar 5.48 Sequence Diagram View Filter Knowledge

6. Sequence diagram view save knowledge

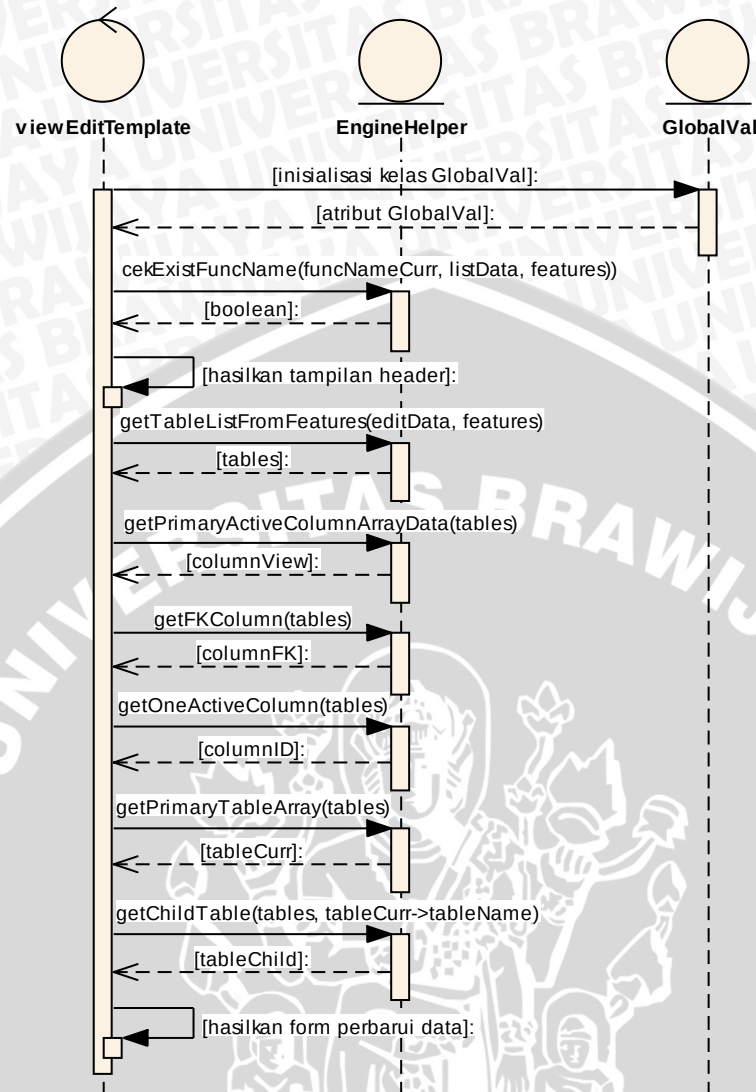
Sequence diagram view save knowledge melibatkan tiga komponen, yaitu: 1) *viewAddTemplate*, 2) *EngineHelper* dan 3) *GlobalVal*. Komponen *viewAddTemplate* digunakan untuk menjalankan fungsi – fungsi dari *view save knowledge*. Gambar 5.49 menunjukkan *sequence diagram view save knowledge*.



Gambar 5.49 Sequence Diagram View Save Knowledge

7. Sequence diagram view edit knowledge

Sequence diagram view save knowledge melibatkan tiga komponen, yaitu: 1) *viewEditTemplate*, 2) *EngineHelper* dan 3) *GlobalVal*. Komponen *viewSaveTemplate* digunakan untuk menjalankan fungsi – fungsi dari *view save knowledge*. Gambar 5.50 menunjukkan *sequence diagram view save knowledge*.



Gambar 5.50 Sequence Diagram View Edit Knowledge

5.3.4 Perancangan *code generator*

Perancangan *code generator* dimulai dari pemetaan daftar persyaratan fungsional ke dalam *activity diagram* agar bisa dianalisis alur kerja dan interaksi antar aktornya. Kemudian *sequence diagram* dibuat untuk menganalisis interaksi antar komponen. Hasil akhir dari perancangan *code generator* adalah *class diagram* yang bisa digunakan sebagai acuan pada proses implementasi kode program.

5.3.4.1 Perancangan aktivitas *code generator*

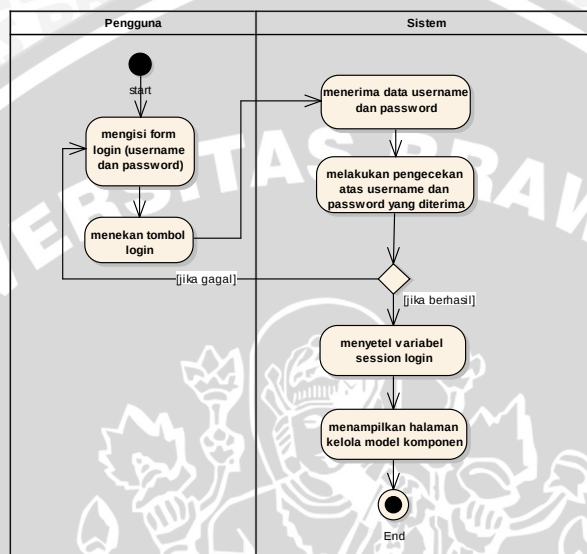
Perancangan aktivitas dilakukan berdasarkan daftar persyaratan fungsional sistem, yaitu:

1. Proses *authentication* (login dan logout)
2. Proses pengelolaan model komponen
3. Proses pengambilan model tabel dari database mySQL diluar sistem

4. Proses parameterisasi model komponen
5. Proses pengolahan model komponen menjadi kode program PTIIK-Apps

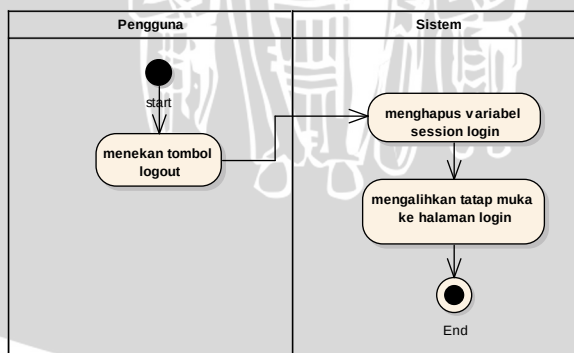
1. Activity diagram login dan logout

Gambar 5.51 menunjukkan *activity diagram* untuk proses *login*. Proses tersebut akan menampilkan halaman kelola model komponen jika pengguna berhasil melakukan *login*. Sistem akan menyetel variabel *session* jika proses *login* berhasil.



Gambar 5.51 Activity Diagram Login

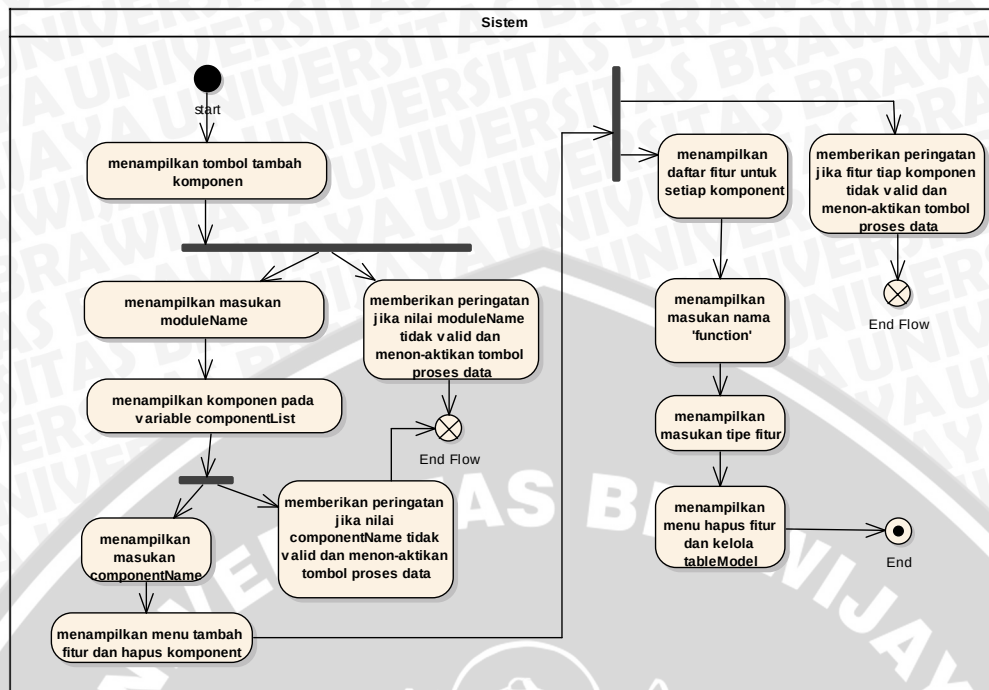
Gambar 5.52 menunjukkan *activity diagram* dari proses *logout*. Pada proses *logout*, sistem akan menghapus variabel *session* pengguna dan mengalihkan tatap muka ke halaman *login*.



Gambar 5.52 Activity Diagram Logout

2. Activity diagram proses kelola model komponen

Proses kelola model komponen ditunjukkan oleh Gambar 5.53. Alur dari proses tersebut adalah menampilkan form masukan komponen model dan opsi – opsi yang digunakan untuk melakukan parameterisasi komponen.



Gambar 5.53 Activity Diagram Kelola Model Komponen

Pada Gambar 5.53 ada beberapa istilah yang akan dijelaskan pada paparan ini. Masukan *moduleName* adalah masukan yang digunakan sebagai parameter nama modul yang akan dihasilkan, *moduleName* yang valid adalah 1) tidak berisi masukan kosong dan 2) tidak berisi masukan angka, simbol dan spasi.

Masukan *componentName* digunakan sebagai parameter nama komponen *controller*, *model* dan *view* yang akan dihasilkan. Nilai *componentName* yang valid adalah 1) tidak berisi masukan kosong, 2) tidak berisi masukan angka, simbol dan spasi dan 3) harus berbeda dengan nilai *componentName* elemen lain.

Masukan fitur pada model komponen digunakan untuk identifikasi model fitur yang akan dibangkitkan menjadi fungsi – fungsi penyusun komponen *controller*, *model* dan *view*. Model fitur yang valid adalah 1) tidak sama dengan model fitur yang lain pada model komponen yang sama, 2) jika model fitur sama dengan yang lain maka masukan nama fungsi (*funcName*) harus berbeda, 3) model fitur tidak boleh memiliki model tabel kosong, 4) pada setiap model komponen harus memiliki fitur lihat data atau sortir data.

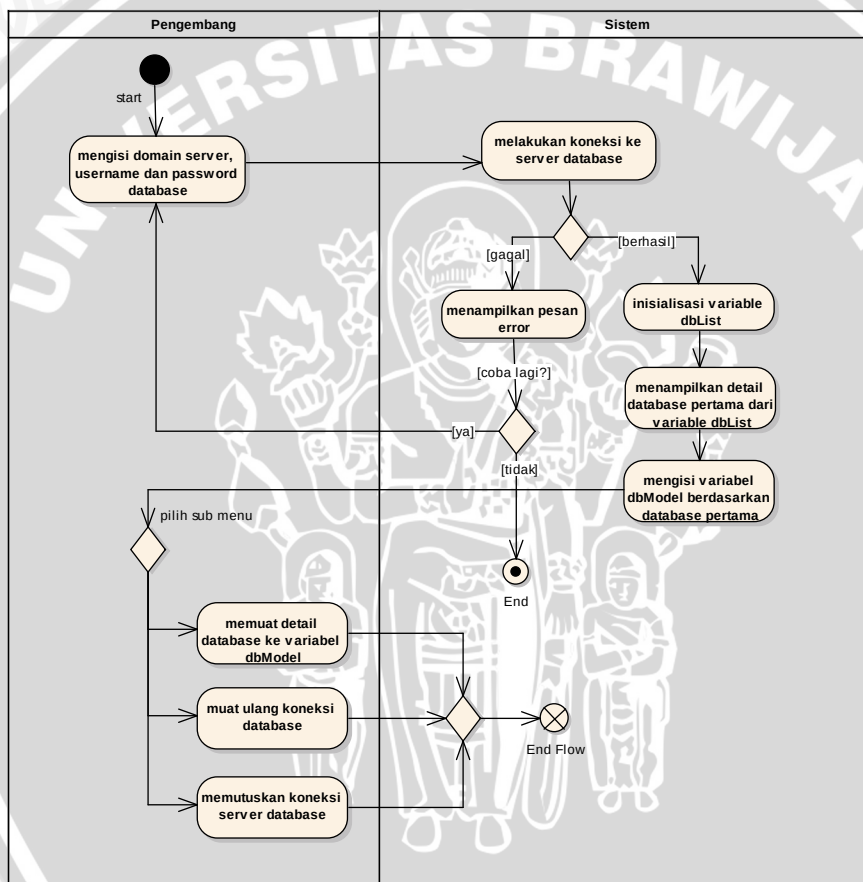
3. Activity diagram proses kelola koneksi database

Proses kelola koneksi database digunakan untuk mengambil model tabel dari database yang disetel oleh pengguna. Proses kelola koneksi database terdiri dari:

1. Menyambungkan ke server database: ditunjukkan oleh Gambar 5.54
2. Memutuskan sambungan ke server database: ditunjukkan oleh Gambar 5.55
3. Menyambung ulang server database: ditunjukkan oleh Gambar 5.56

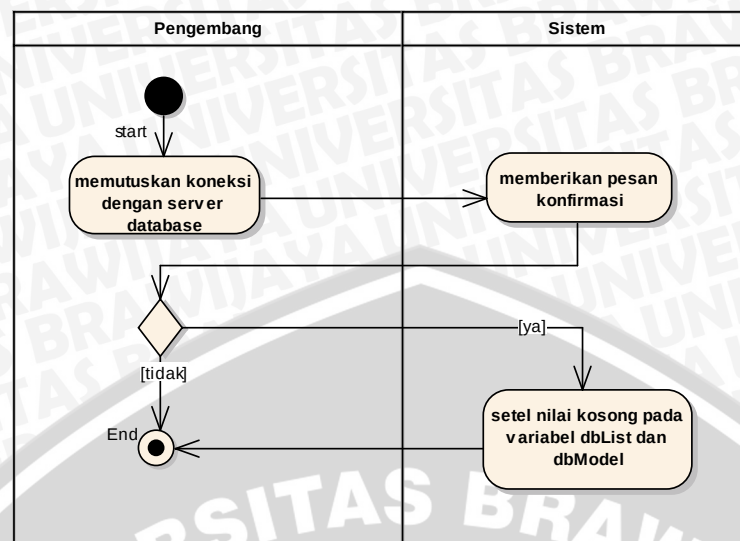
4. Memilih unit database untuk mengambil model tabel: ditunjukkan oleh Gambar 5.33

Gambar 5.54 menjabarkan proses menyambungkan ke server database. Proses tersebut membutuhkan pengguna untuk mengisi masukan 1) domain server, 2) username dan 3) password. Setelah berhasil disambungkan maka sistem akan mengambil data daftar nama – nama database yang ada di server. Secara otomatis sistem akan memilih unit database paling awal untuk diambil model tabelnya (proses memilih unit database pada poin ke-4), selanjutnya sistem akan memberikan tiga pilihan kepada pengguna, yaitu: 1) memilih unit database, 2) memuat ulang sambungan ke server database dan 3) memutuskan sambungan ke server.



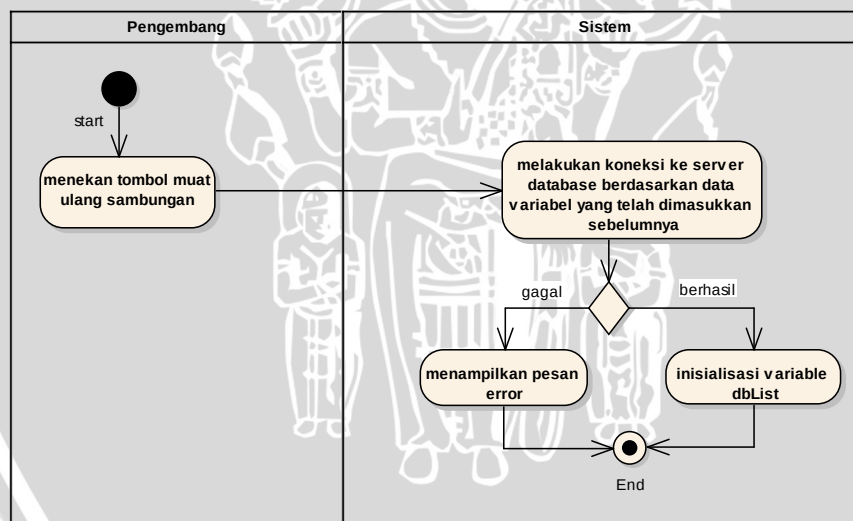
Gambar 5.54 Activity Diagram Menyambungkan ke Server Database

Gambar 5.55 menunjukkan proses pemutusan sambungan dari server database. Sistem akan meminta persetujuan pengguna untuk memutuskan koneksi database, jika pengguna menyetujui maka sistem akan mengosongkan data dari variabel *dbList* dan *dbModel*. Variabel *dbList* adalah variabel yang berisi sekumpulan daftar nama basis data dari server yang disambungkan. Variabel *dbModel* berisi sekumpulan model tabel dari basis data yang dipilih dari salah satu elemen *dbList*.



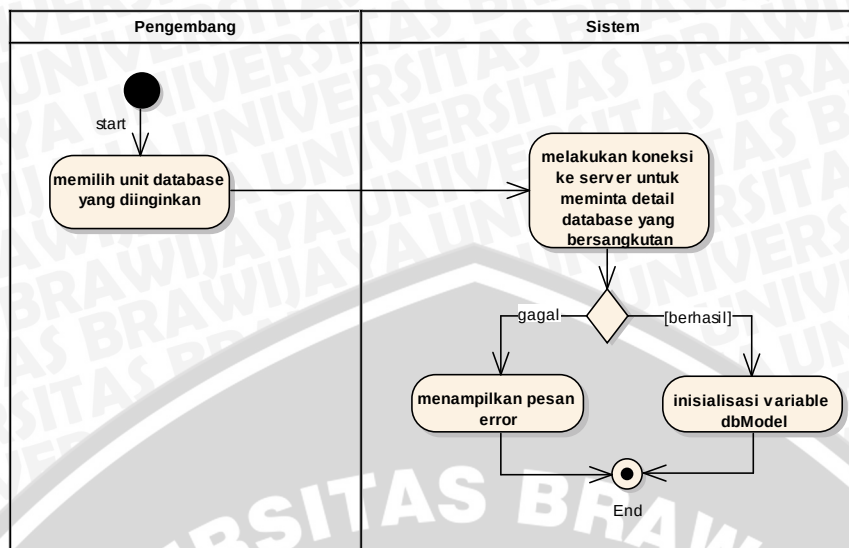
Gambar 5.55 Activity Diagram Pemutusan Sambungan Server Database

Gambar 5.56 menunjukkan proses muat ulang sambungan ke server database. Proses tersebut sama seperti proses penyambungan server database (pada Gambar 5.54) namun tanpa meminta pengguna untuk mengisi form penyambungan ke server. Sistem melakukan sambungan ulang ke server database berdasarkan form masukan yang dimasukkan pengguna sebelumnya.



Gambar 5.56 Activity Diagram Sambungan Ulang Server Database

Gambar 5.57 menunjukkan proses pengambilan model tabel dari database yang dipilih oleh pengguna. Pada proses sambungan ke server database (Gambar 5.54) sistem akan menampilkan daftar database dari server tersebut ke dalam form dropdown, dimana pengguna bisa memilih database mana yang ingin dimuat tabel modelnya ke dalam sistem.



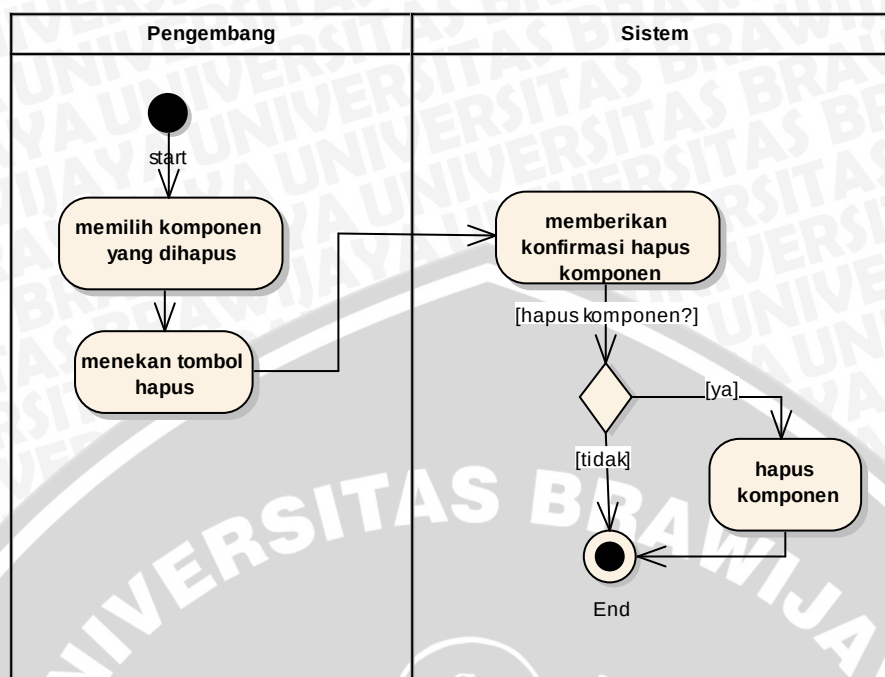
Gambar 5.57 Activity Diagram Mengambil Model Tabel dari Database

4. Activity diagram proses parameterisasi model komponen

Proses parameterisasi komponen digunakan untuk mengelola model komponen agar sesuai dengan keinginan pengguna. Proses tersebut terdiri dari beberapa opsi, yaitu:

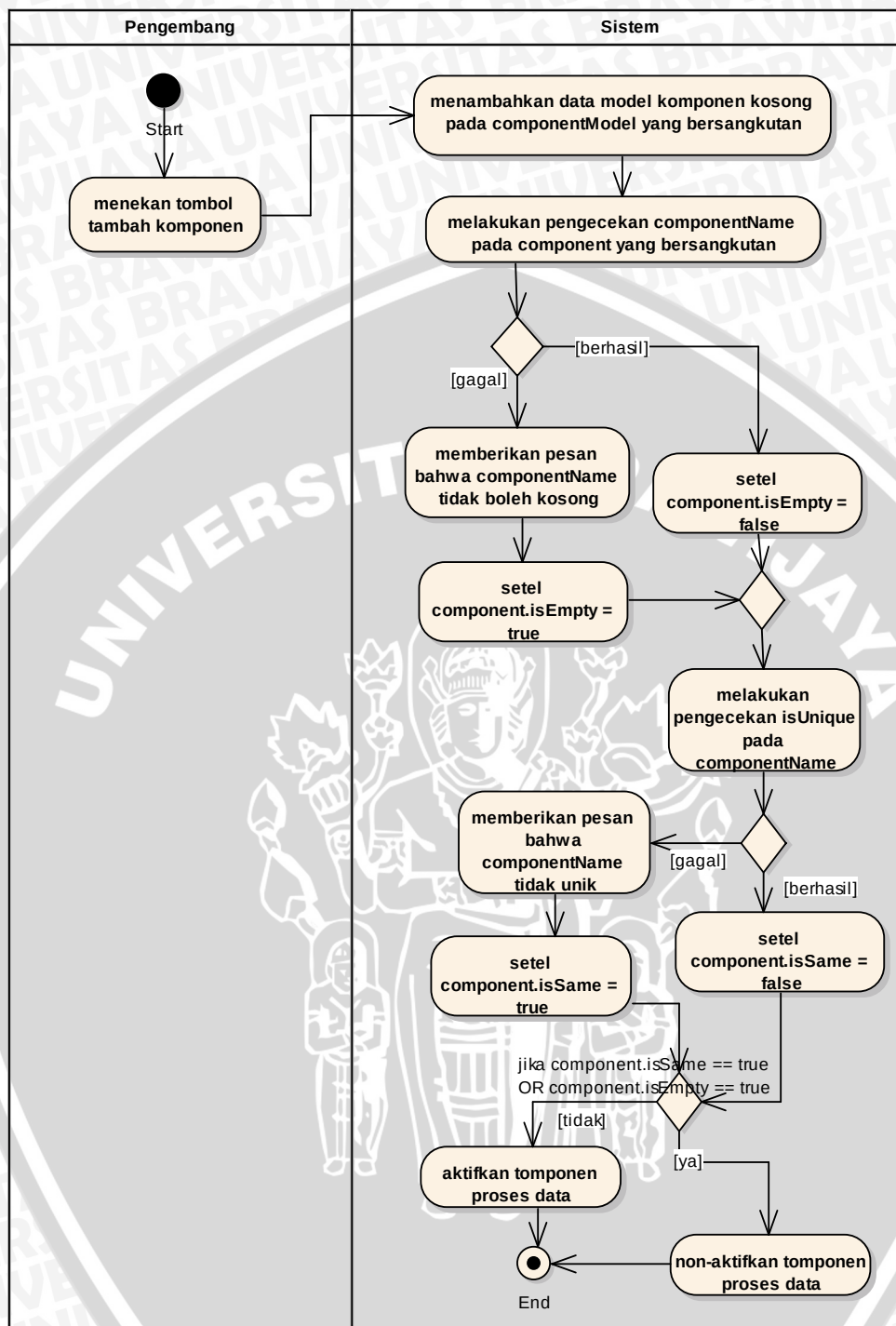
1. Tambah dan hapus model komponen
2. Tambah dan hapus model fitur
3. *TableManager* digunakan untuk mengelola model tabel didalam model fitur
4. *ColumnManager* digunakan untuk mengelola model kolom yang ada di dalam model tabel
5. *InputManager* digunakan untuk mengelola pilihan masukan yang ada di dalam model kolom.

Gambar 5.58 menunjukkan proses hapus model komponen. Sistem akan memberikan pesan konfirmasi dari pengguna ketika proses hapus komponen. Sistem akan menghapus unit komponen yang dipilih beserta data yang ada di dalamnya.



Gambar 5.58 Activity Diagram Hapus Komponen

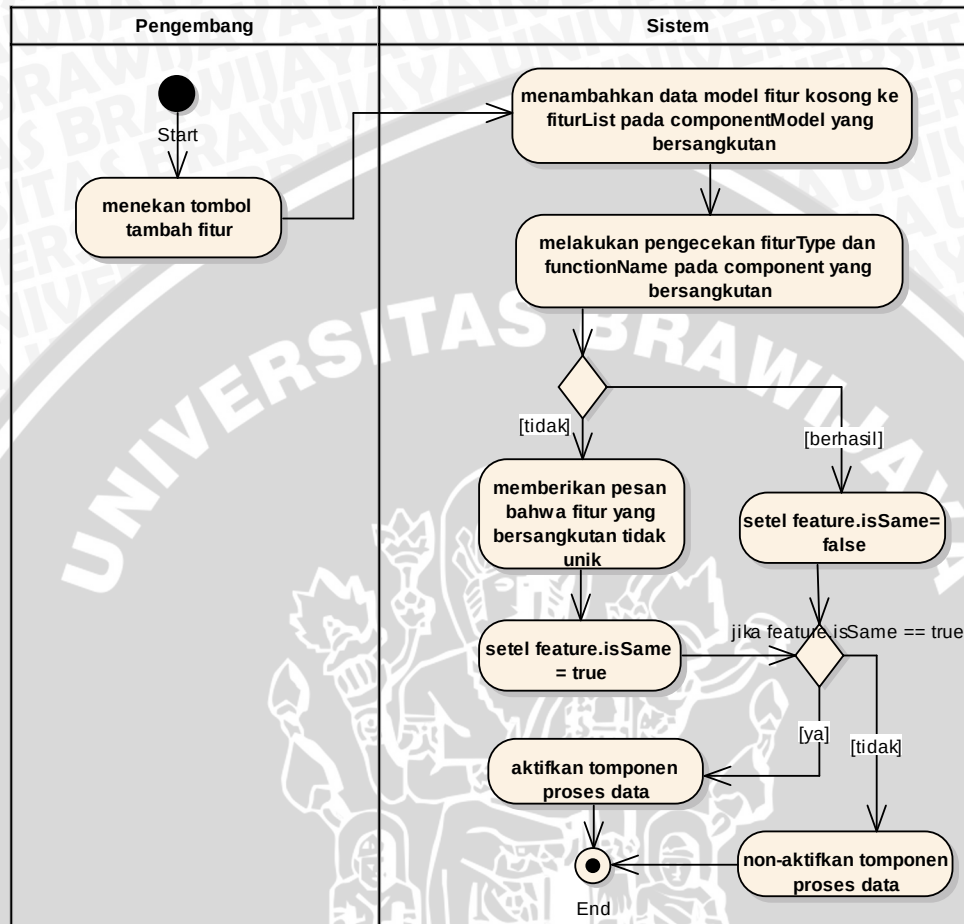
Gambar 5.59 menunjukkan proses tambah model komponen. Sistem akan menambahkan variabel model komponen baru, secara otomatis sistem akan mengecek apakah model komponen tersebut tidak kosong dan tidak identik dengan model komponen yang lain. Jika pengecekan gagal, maka sistem akan membuat tombol proses data menjadi tidak aktif. Variabel *componentModel* adalah variabel yang berisi sekumpulan model komponen.



Gambar 5.59 Activity Diagram Proses Tambah Komponen

Proses tambah fitur ditunjukkan oleh Gambar 5.60, data fitur baru yang ditambahkan akan dimasukkan ke dalam model komponen. Sistem akan melakukan pengecekan pada nama fungsi (*funcName*) dan jenis fitur (*featureType*) untuk memastikan bahwa tidak ada model fitur yang identik. Nama fungsi digunakan untuk membedakan pembangkitan kode program pada jenis fitur yang sama dalam satu komponen model. Sekumpulan model fitur akan disimpan pada variabel *featureList* pada model komponen.

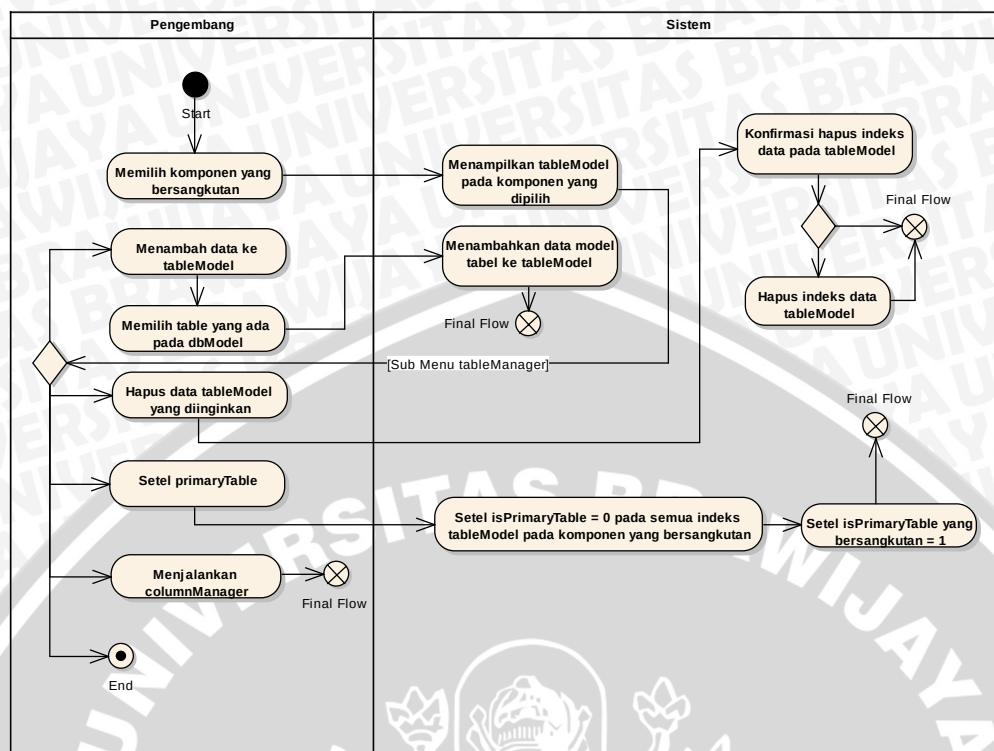
Proses hapus fitur model memiliki alur yang sama dengan proses hapus komponen. Sistem akan meminta persetujuan pengguna untuk menghapus fitur yang dipilih. Sistem akan menghapus fitur yang disetujui oleh pengguna beserta data yang berada di dalam fitur model tersebut.



Gambar 5.60 Activity Diagram Proses Tambah Fitur

Proses *TableManager* digunakan untuk mengelola model tabel yang ada di dalam fitur model. Gambar 5.61 menunjukkan proses *TableManager*. Sistem memberikan pilihan opsi kepada pengguna untuk mengelola model tabel, opsi tersebut adalah:

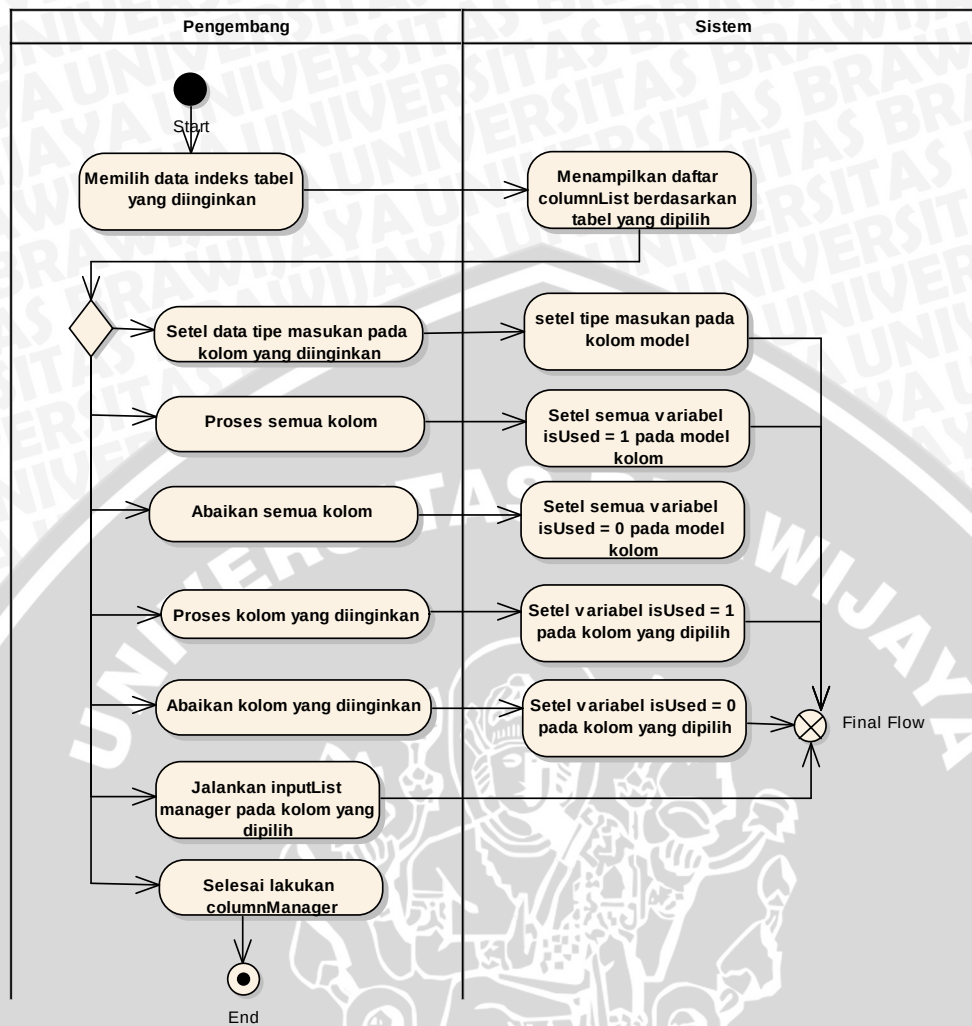
1. Menambah model tabel dari variabel *dbModel*
2. Menghapus model tabel yang diinginkan (sistem akan memberikan pesan konfirmasi untuk menghapus tabel)
3. Menyetel model tabel utama (sistem akan menyetel model tabel utama ketika model tabel kosong dan pengguna pertama kali menambahkan model tabel)
4. Melakukan opsi *ColumnManager* pada model tabel yang dipilih



Gambar 5.61 Activity Diagram Table Manager

Proses *ColumnManager* digunakan untuk mengelola model kolom pada model tabel yang dikehendaki oleh pengguna. Gambar 5.62 menunjukkan proses *ColumnManager*. Sistem memungkinkan pengguna melakukan beberapa pilihan opsi pada *ColumnManager*, yaitu:

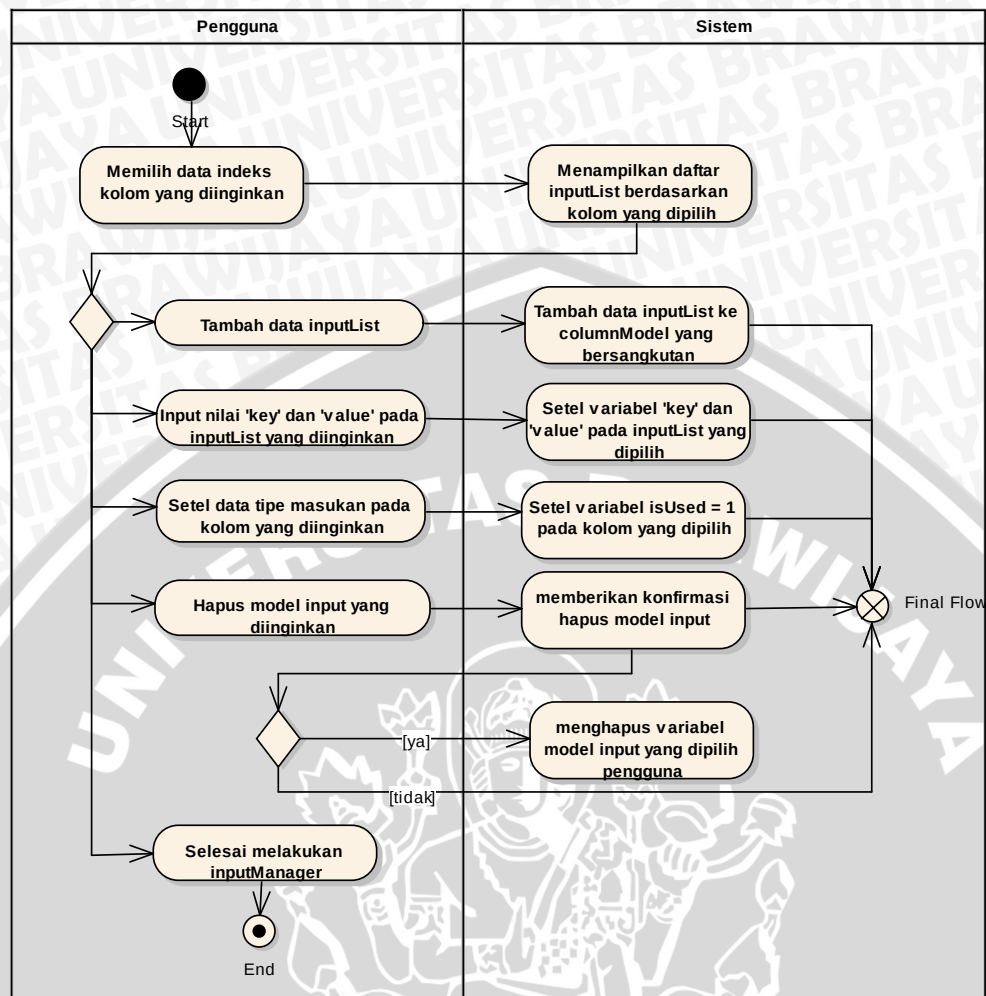
1. Menyetel variabel *isUsed* pada model kolom (model kolom dengan *isUsed* == 1 yang akan diproses oleh sistem)
2. Memilih tipe masukan pada model kolom
3. Setel variabel *isUsed* menjadi 1 atau 0 pada semua model kolom pada satu tabel model
4. Menjalankan *InputManager*



Gambar 5.62 Activity Diagram Column Manager

Gambar 5.63 menunjukkan proses *InputManager*, proses tersebut digunakan untuk mengelola model pilihan masukan pada model kolom yang diinginkan. Sistem memberikan pilihan opsi kepada pengguna untuk melakukan *InputManager*, yaitu:

1. Tambah model pilihan masukan
2. Hapus model pilihan masukan

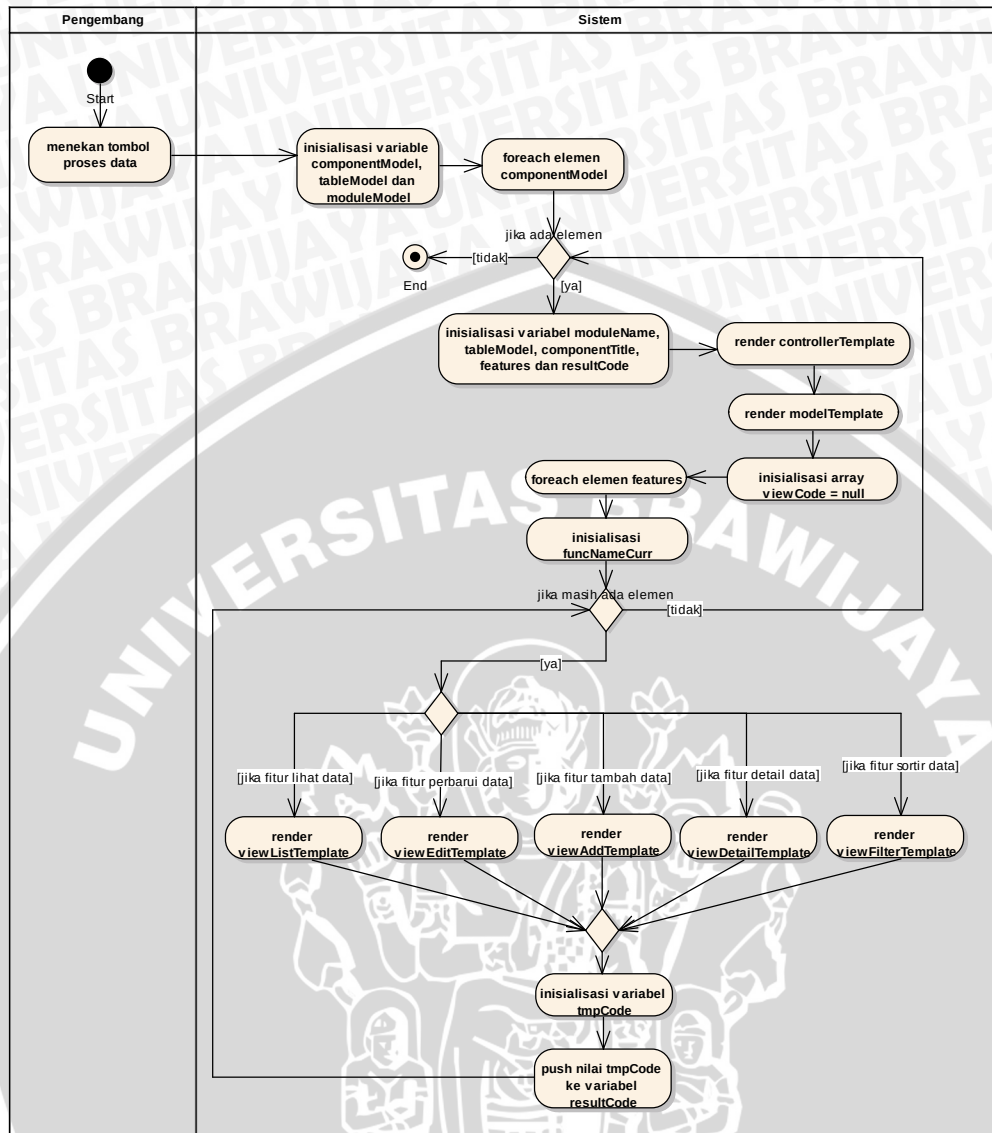


Gambar 5.63 Activity Diagram Input-Option Manager

5. Activity diagram proses data model komponen menjadi kode program

Gambar 5.64 menunjukkan proses pengolahan data model komponen menjadi kode program PTIIK-Apps. Data model komponen dikirim dalam variabel array *componentModel*, dimana satu unit model komponen akan diubah menjadi: satu komponen *controller*, satu komponen *model* dan beberapa komponen *view* (berdasarkan model fitur yang ada pada model komponen yang bersangkutan).

Hasil dari proses pengolahan data model komponen akan disimpan ke dalam variabel *resultCode*. Variabel *resultCode* digunakan untuk menampilkan konten komponen pada halaman tatap muka agar bisa diunduh oleh pengembang. Istilah *render* yang dipakai pada Gambar 5.64 adalah sebuah proses untuk menerjemahkan masukan berupa model menjadi komponen kode program. Istilah *push* adalah sebuah operasi untuk menambahkan sebuah variabel kedalam elemen variabel *array*.



Gambar 5.64 Activity Diagram Proses Data Model Komponen

5.3.4.2 Perancangan urutan code generator

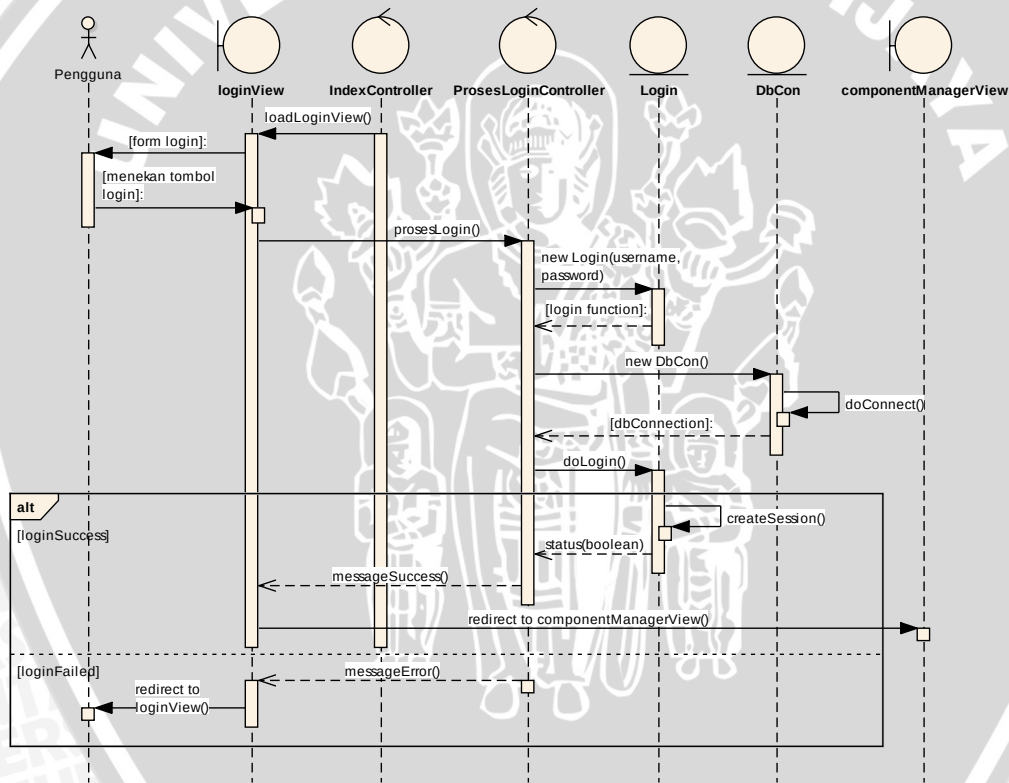
Perancangan *sequence* dilakukan untuk menganalisis urutan proses yang dilakukan oleh komponen kode program yang akan dibuat, sehingga membantu proses pemetaan *class diagram* interaksi antar komponen pada *program generator*. Perancangan *sequence diagram* dilakukan berdasarkan rancangan *activity diagram* yang telah dibuat. *Sequence diagram* yang dibuat terdiri dari:

1. Proses *authentication* (*login* dan *logout*)
2. Proses pengelolaan model komponen
3. Proses pengambilan model tabel dari database *mysql* diluar sistem
4. Proses parameterisasi model komponen
5. Proses pengolahan model komponen menjadi kode program PTIIK-Apps

1. Sequence diagram login dan logout

Gambar 5.65 menunjukkan *sequence diagram* untuk proses *login*. Proses tersebut melibatkan beberapa komponen, yaitu:

1. *loginView*: digunakan sebagai interaksi tatap muka dengan pengguna
2. *IndexController*: digunakan untuk mengontrol halaman tatap muka *loginView*
3. *ProsesLoginController*: digunakan untuk mengontrol alur kerja sistem pada proses *login*
4. *Login*: kelas yang digunakan untuk menjalankan fungsi – fungsi login
5. *DbCon*: kelas yang digunakan untuk menyambungkan aplikasi dengan database aplikasi
6. *componentManagerView*: digunakan sebagai tatap muka untuk mengelola model komponen ketika pengguna berhasil melakukan *login*

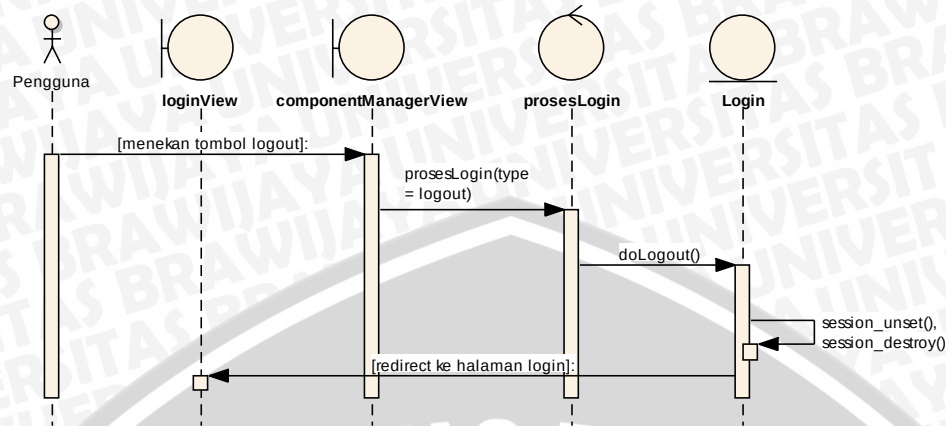


Gambar 5.65 Sequence Diagram Proses Login

Sequence diagram proses *logout* ditunjukkan pada Gambar 5.66. proses tersebut melibatkan beberapa komponen, yaitu:

1. *loginView*: ketika proses *logout* selesai, maka sistem akan mengalihkan halaman tatap muka ke *loginView*
2. *componentManagerView*: digunakan sebagai tatap muka yang menampilkan tombol *logout*
3. *prosesLogin*: digunakan untuk mengontrol alur kerja sistem pada proses *logout*

4. *Login*: kelas yang digunakan untuk menjalankan fungsi *logout*



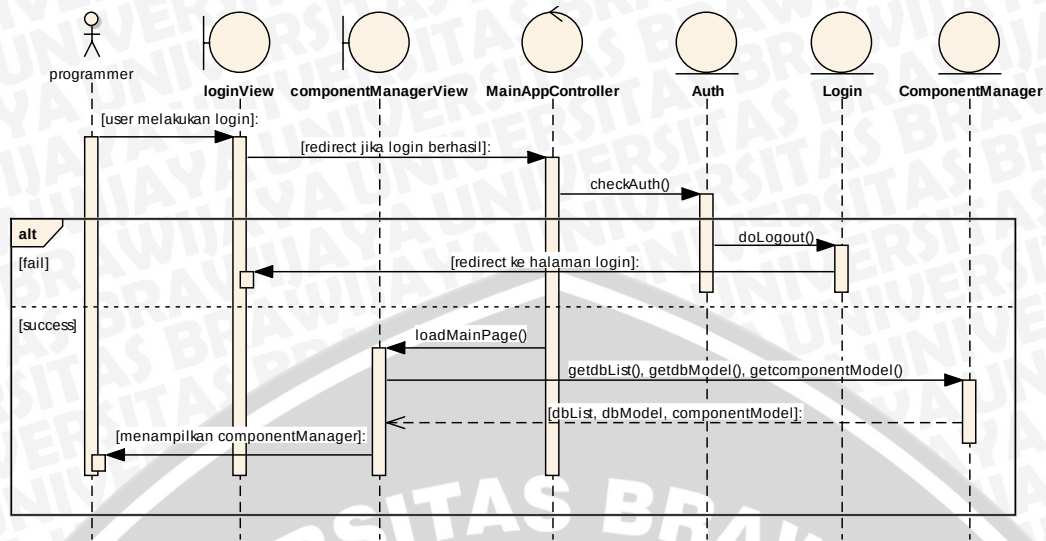
Gambar 5.66 Sequence Diagram Proses Logout

2. Sequence diagram proses kelola model komponen

Proses kelola model komponen memungkinkan pengguna untuk melakukan proses parameterisasi model komponen dan melakukan pengambilan model tabel dari server database yang diinginkan. Gambar 5.67 menunjukkan *sequence diagram* proses kelola model komponen.

Proses tersebut melibatkan beberapa komponen, yaitu:

1. *loginView*: sistem akan mengalihkan tampilan tatap muka ke *loginView* ketika pengguna tidak memiliki akses pada halaman *componentManagerView*
2. *componentManagerView*: komponen yang menampilkan tampilan untuk melakukan parameterisasi model komponen dan pengambilan model tabel dari server database
3. *MainAppController*: komponen yang digunakan untuk memanggil halaman tampilan *componentManagerView*
4. *Auth*: kelas yang digunakan untuk mengecek hak akses pengguna atas opsi/halaman yang diakses
5. *Login*: kelas yang digunakan untuk melakukan fungsi *logout* ketika pengguna tidak memiliki akses terhadap opsi yang diakses
6. *ComponentManager*: komponen yang digunakan untuk mengelola variabel model komponen



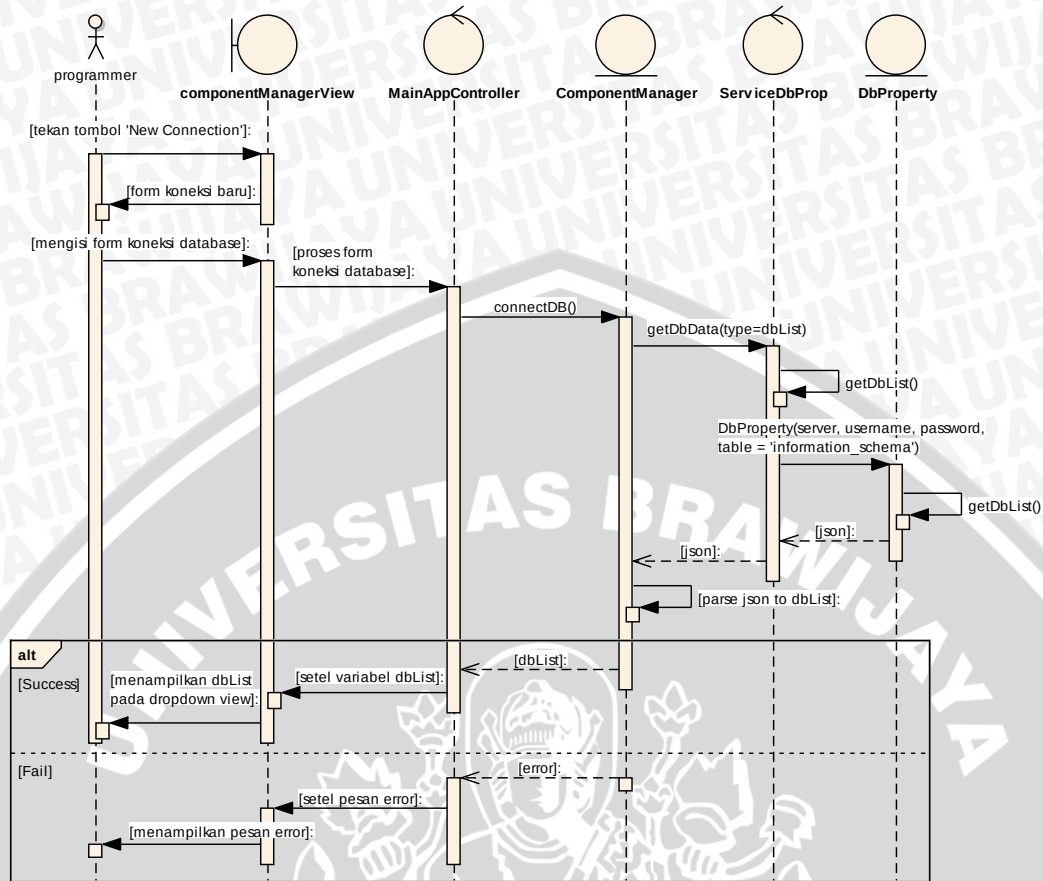
Gambar 5.67 *Sequence Diagram* Proses Kelola Model Komponen

3. *Sequence diagram* proses kelola koneksi database

Gambar 5.68 menunjukkan *sequence diagram* dari proses penyambungan ke server database. Proses penyambungan server database melibatkan beberapa komponen, yaitu:

1. *componentManagerView*: komponen yang digunakan untuk menampilkan halaman tatap muka untuk melakukan sambungan ke server database
2. *MainAppController*: digunakan untuk mengontrol dan menampilkan *componentManagerView*
3. *ComponentManager*: komponen yang digunakan untuk mengelola variabel *dbList* dan *dbModel*
4. *ServiceDbProp*: digunakan untuk mengontrol proses pengambilan data model tabel dari server database
5. *DbProp*: kelas yang digunakan untuk mengambil data model tabel dari server database

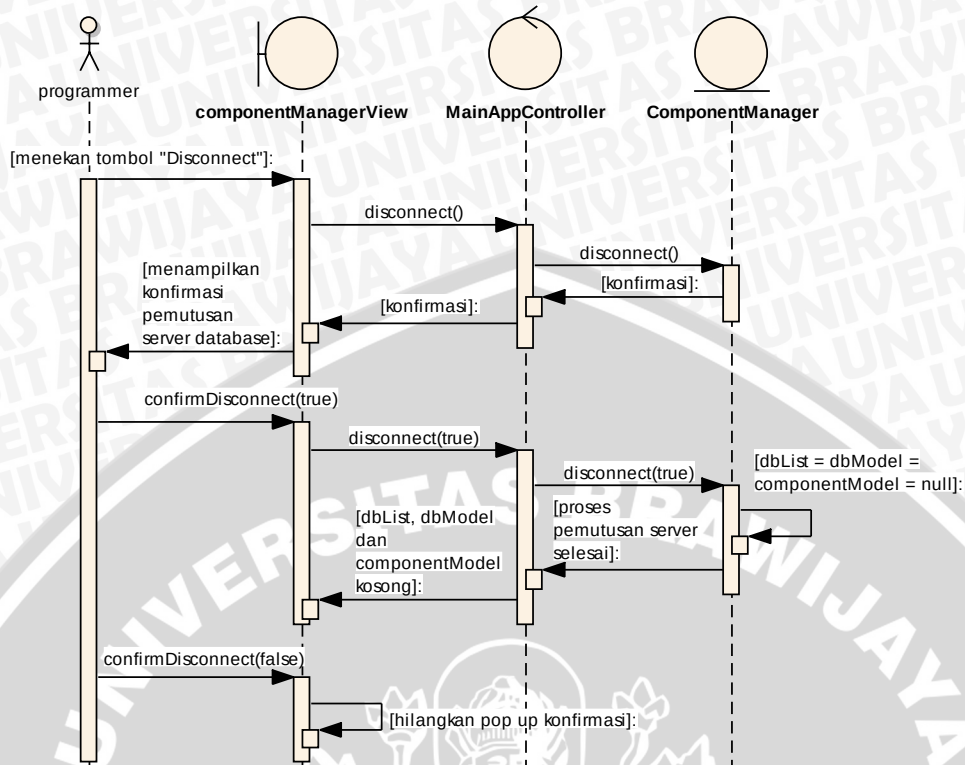
Proses pemuatan ulang sambungan server database melibatkan komponen yang sama dengan proses penyambungan server database. Perbedaannya pada proses pemuatan ulang sambungan, tidak meminta pengguna mengisi masukan untuk menyetel sambungan ke server database.



Gambar 5.68 Sequence Diagram Penyambungan Server Database

Gambar 5.69 menunjukkan *sequence diagram* dari proses pemutusan server database. Proses tersebut melibatkan interaksi beberapa komponen, yaitu:

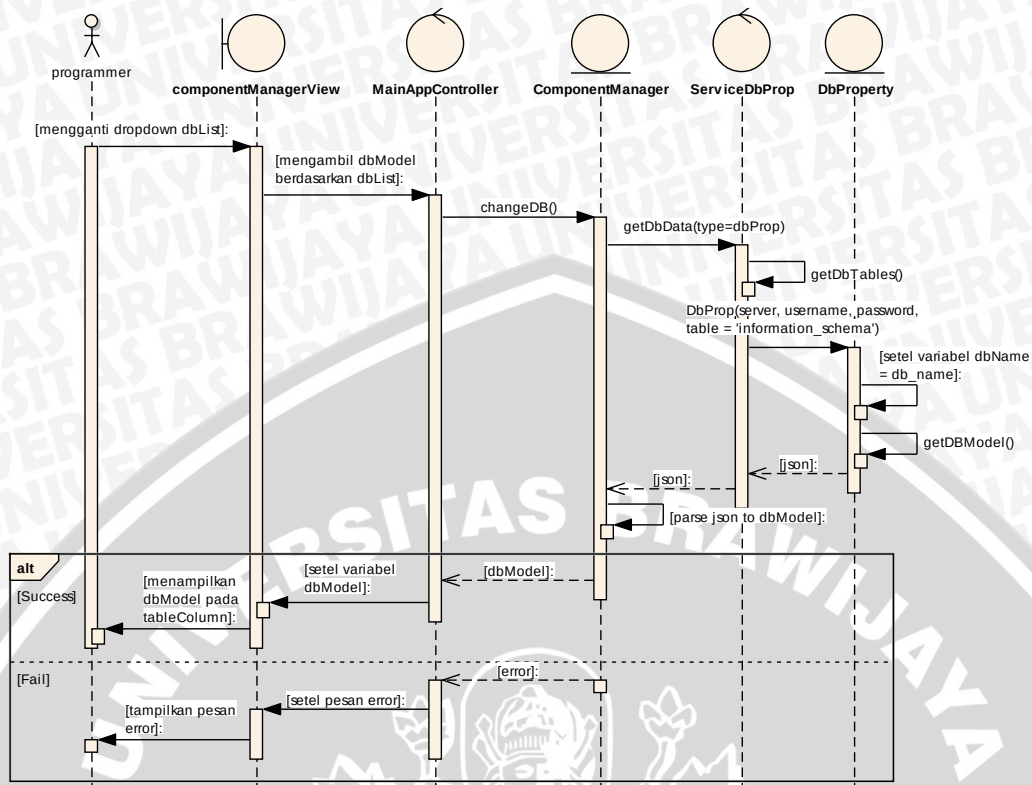
1. *componentManagerView*: komponen yang digunakan untuk menampilkan tampilan tatap muka tombol logout
2. *MainAppController*: digunakan untuk menampilkan *componentManagerView*
3. *ComponentManager*: digunakan untuk mengelola proses pemutusan sambungan server database



Gambar 5.69 Sequence Diagram Proses Pemutusan Server Database

Gambar 5.70 menunjukkan *sequence diagram* proses pengambilan model tabel dari server database yang dipilih. Pengguna memilih unit database yang dipilih, kemudian sistem menampilkan model tabel ke dalam variabel *dbModel*. Proses tersebut melibatkan interaksi antar komponen sebagai berikut:

1. *componentMangerView*: digunakan untuk menampilkan halaman tatap muka halaman pemilihan unit database
2. *MainAppController*: digunakan untuk menampilkan *componentManagerView*
3. *ComponentManager*: digunakan untuk mengontrol proses pengambilan model tabel
4. *ServiceDbProp*: komponen yang menyediakan layanan untuk mengontrol proses pengambilan data model tabel dari server database
5. *DbProperty*: kelas digunakan untuk menjalankan fungsi untuk mengambil data model tabel

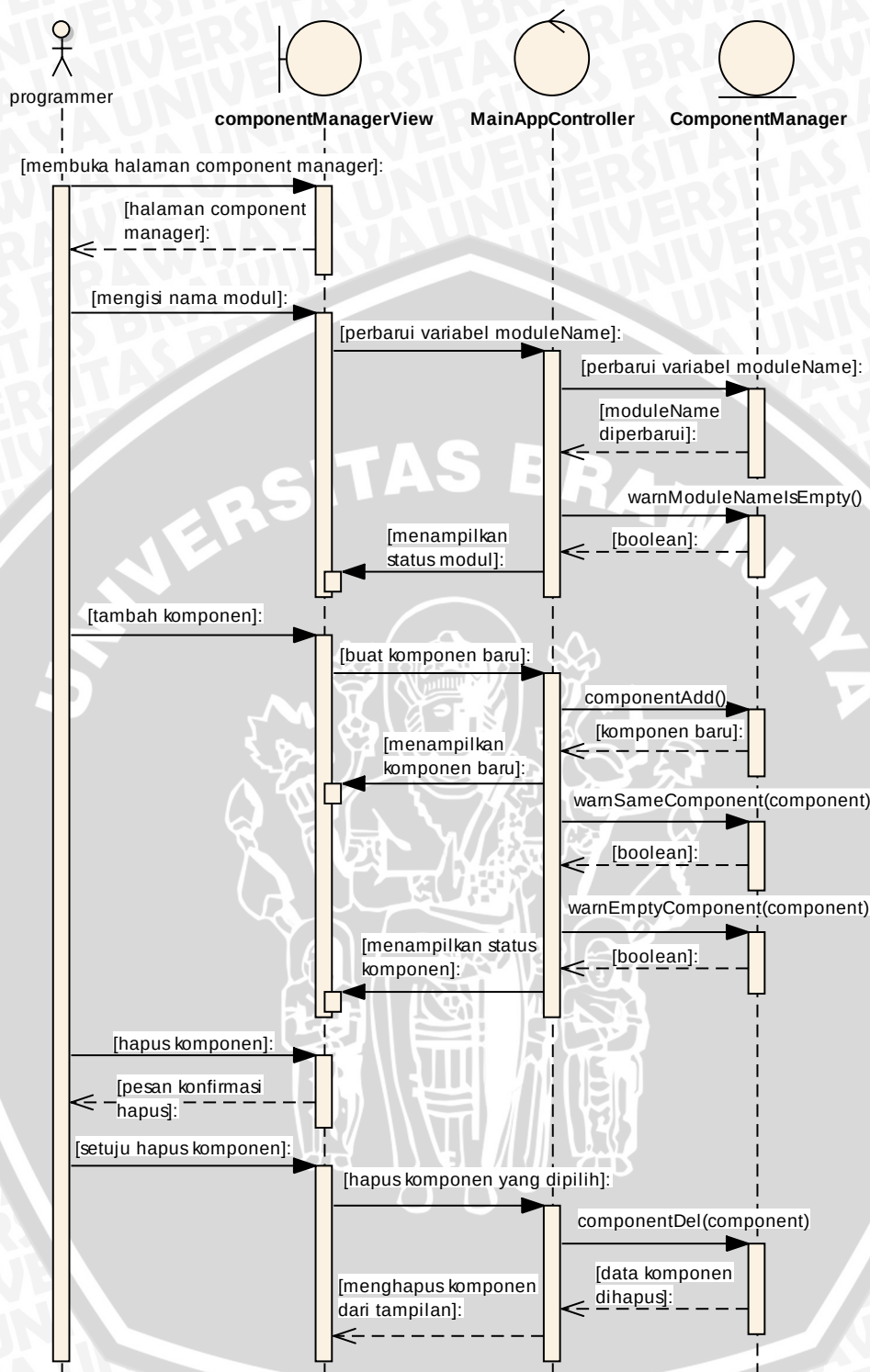


Gambar 5.70 Sequence Diagram Pengambilan Model Tabel dari Database

4. Sequence diagram proses parameterisasi komponen

Sequence diagram kelola model komponen ditunjukkan oleh Tabel 5.71. Proses tersebut melibatkan beberapa komponen, yaitu:

1. *componentManagerView*: digunakan untuk menampilkan halaman tatap muka halaman kelola model komponen
2. *MainAppController*: digunakan untuk menampilkan *componentManagerView*
3. *ComponentManager*: digunakan untuk menjalankan fungsi kelola model komponen

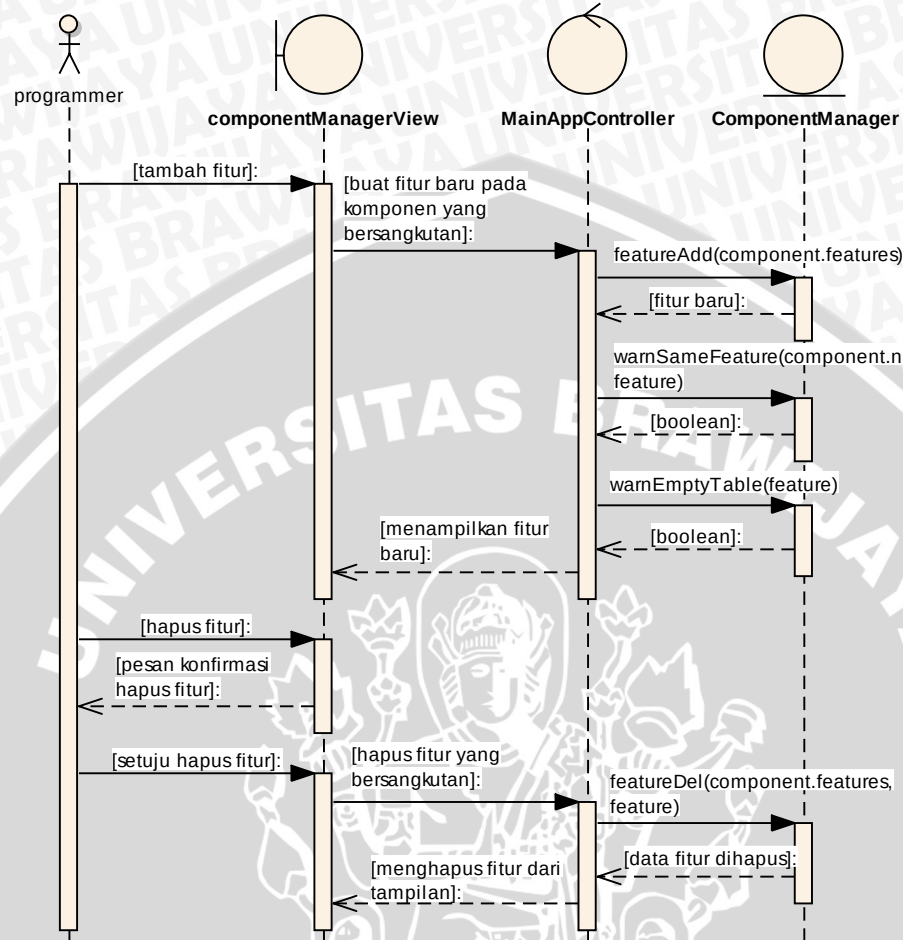


Gambar 5.71 Sequence Diagram Kelola Model Komponen

Gambar 5.72 menunjukkan *sequence diagram* kelola model fitur. Proses tersebut melibatkan komponen sebagai berikut:

1. *componentManagerView*: digunakan untuk menampilkan halaman tatap muka halaman kelola model fitur
2. *MainAppController*: digunakan untuk menampilkan *componentManagerView*

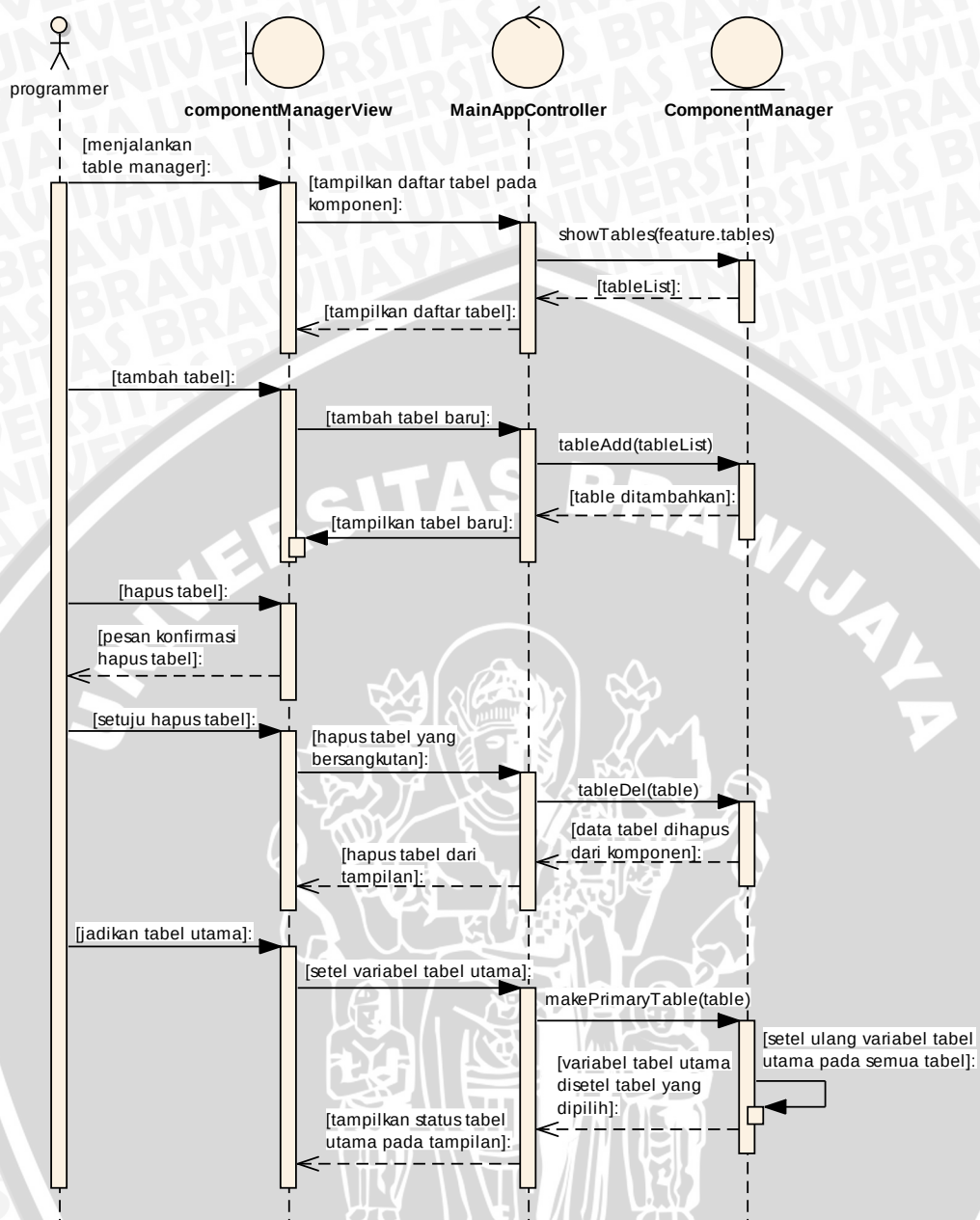
3. *ComponentManager*: digunakan untuk menjalankan fungsi kelola model komponen



Gambar 5.72 Sequence Diagram Proses Kelola Fitur Model

Gambar 5.73 menunjukkan *sequence diagram* dari proses *tableManager*. Proses tersebut melibatkan komponen sebagai berikut:

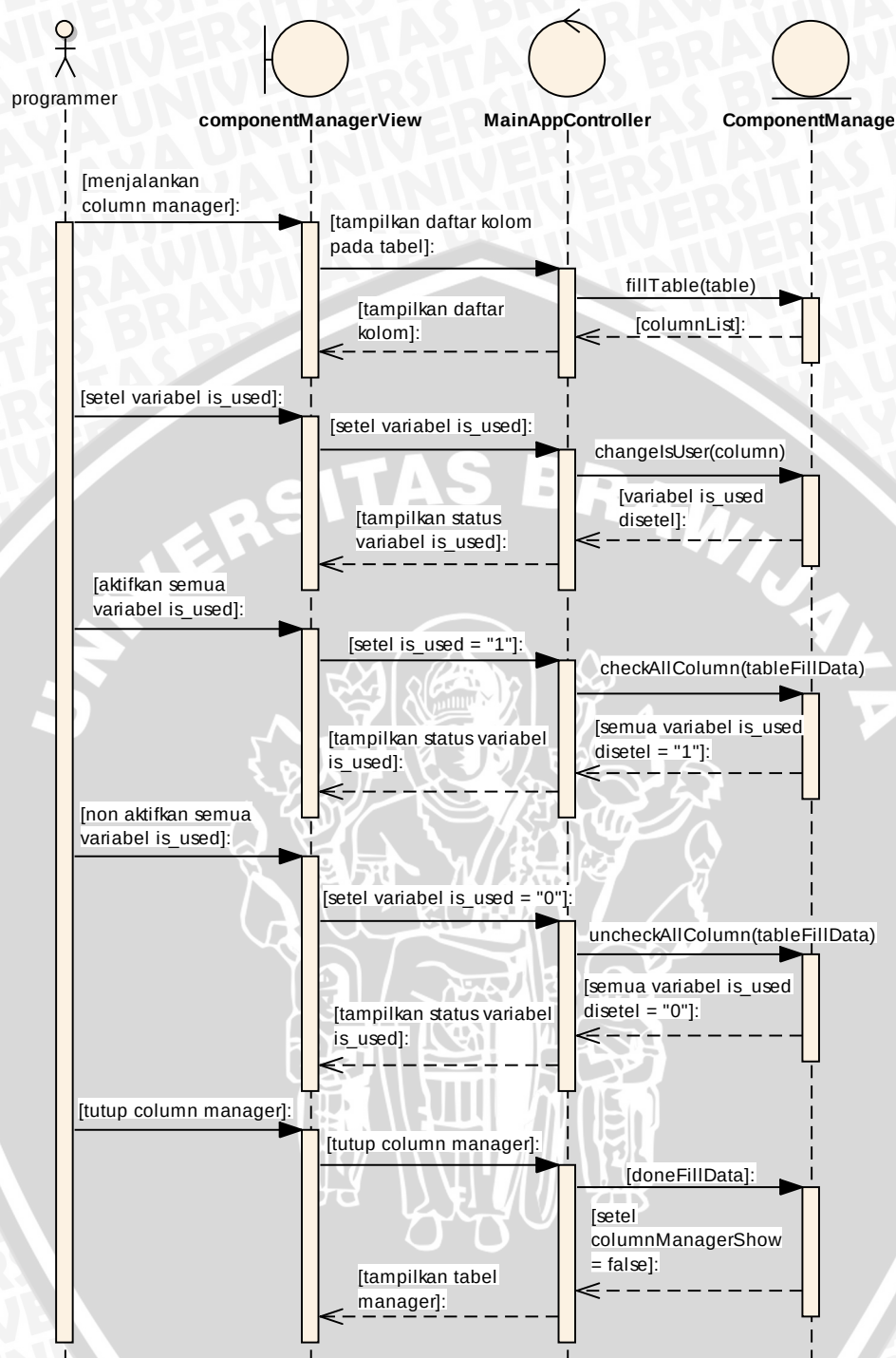
1. *componentManagerView*: digunakan untuk menampilkan halaman tatap muka *tableManager*
2. *MainAppController*: digunakan untuk menampilkan *componentManagerView*
3. *ComponentManager*: digunakan untuk menjalankan fungsi kelola model komponen



Gambar 5.73 Sequence Diagram Table Manager

Gambar 5.74 menunjukkan *sequence diagram* dari proses *columnManager*. Proses tersebut melibatkan komponen sebagai berikut:

1. *componentManagerView*: digunakan untuk menampilkan halaman tatap muka *columnManager*
2. *MainAppController*: digunakan untuk menampilkan *componentManagerView*
3. *ComponentManager*: digunakan untuk menjalankan fungsi kelola model komponen

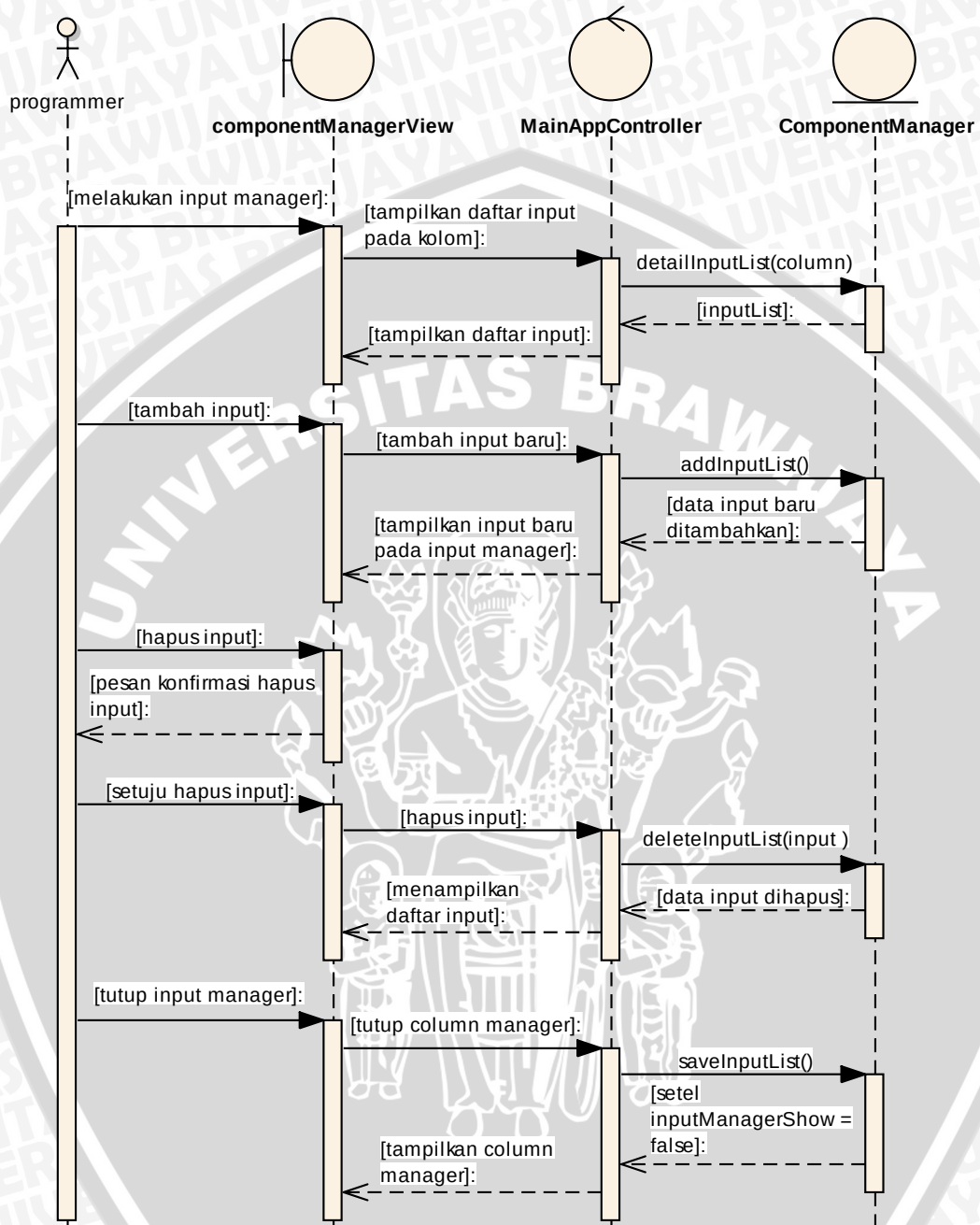


Gambar 5.74 Sequence Diagram Column Manager

Gambar 5.75 menunjukkan *sequence diagram* dari proses *inputManager*. Proses tersebut melibatkan komponen sebagai berikut:

1. *componentManagerView*: digunakan untuk menampilkan halaman tatap muka *inputManager*
2. *MainAppController*: digunakan untuk menampilkan *componentManagerView*

3. *ComponentManager*: digunakan untuk menjalankan fungsi kelola model komponen



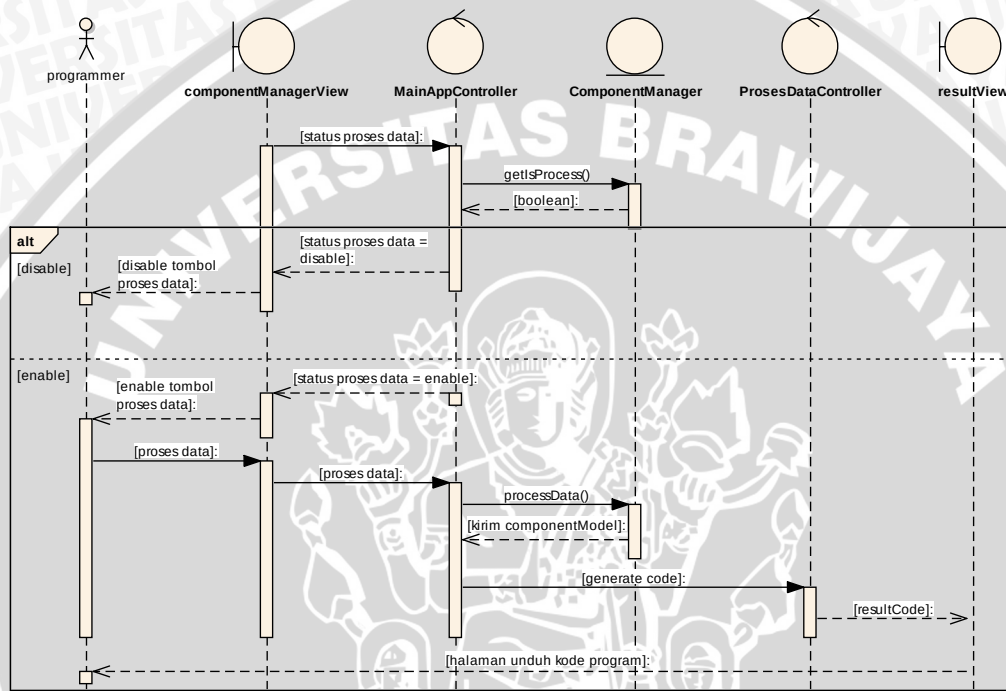
Gambar 5.75 Sequence Diagram Input-Option Manager

5. Sequence diagram proses data model komponen menjadi kode program

Gambar 5.76 menunjukkan *sequence diagram* pada proses pengolahan komponen model. Proses tersebut melibatkan interaksi antar komponen sebagai berikut:

1. *componentManagerView*: digunakan untuk menampilkan tombol proses data

2. *MainAppController*: digunakan untuk menampilkan *componentManagerView* dan mengirim data model komponen kepada *ProsesDataController*
3. *ComponentManager*: menyediakan fungsi untuk mengirim model komponen kepada *ProsesDataController*
4. *ProsesDataController*: digunakan untuk mengolah data model komponen menjadi kode program PTIIK-Apps
5. *resultView*: digunakan untuk menampilkan halaman tatap muka unduh kode program yang telah dibangkitkan oleh sistem

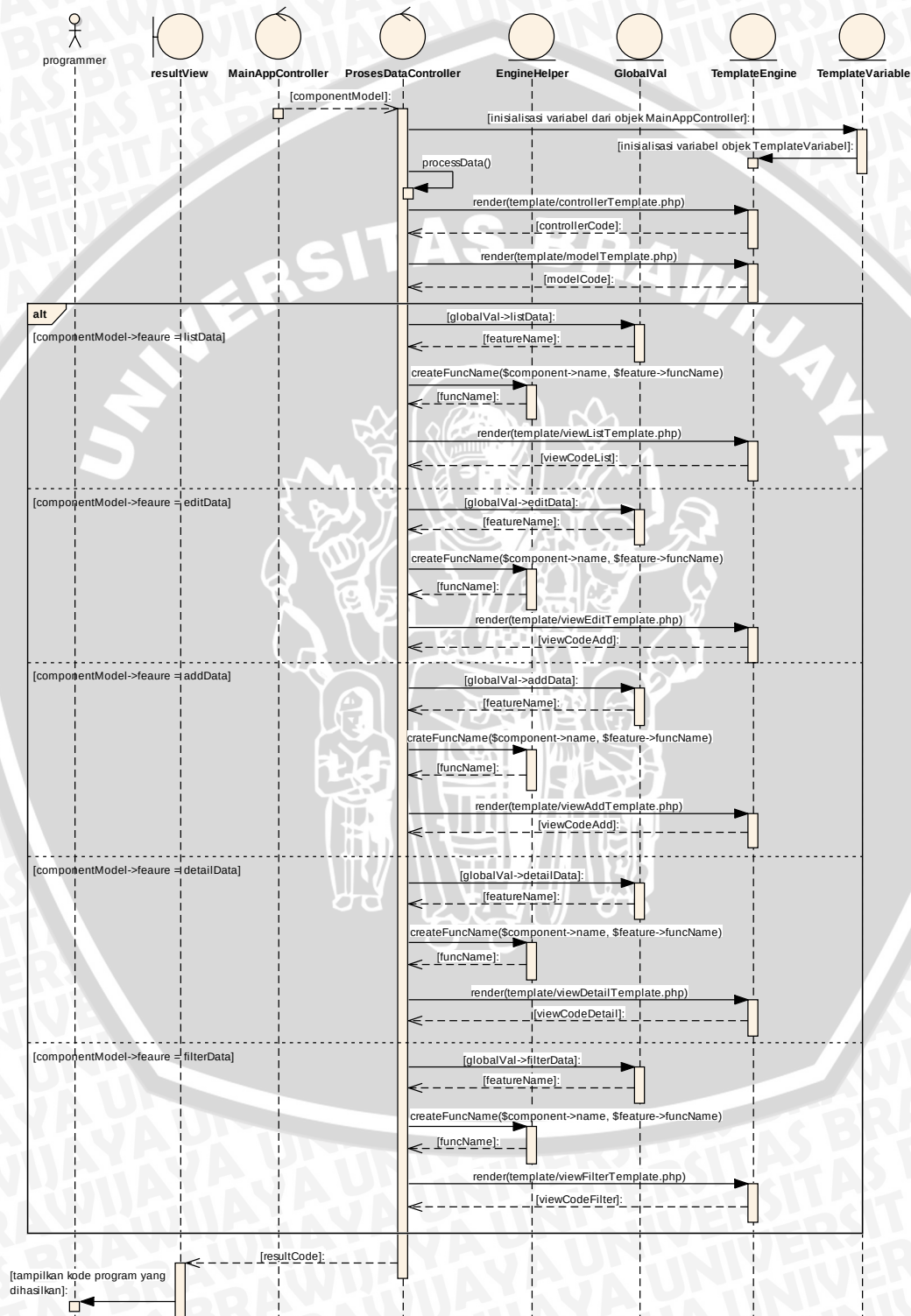


Gambar 5.76 Sequence Diagram Proses Pengolahan Data Model Komponen

Gambar 5.77 menunjukkan *sequence diagram* pada proses pembangkitan kode program PTIIK-Apps. Proses tersebut melibatkan interaksi antar komponen sebagai berikut:

1. *resultView*: digunakan untuk menampilkan halaman tatap muka unduh kode program yang telah dibangkitkan oleh sistem
2. *ProsesDataController*: digunakan untuk mengolah data model komponen menjadi kode program PTIIK-Apps
3. *MainAppController*: digunakan untuk mengirim data model komponen kepada *ProsesDataController*
4. *EngineHelper*: digunakan untuk membantu menjalankan fungsi – fungsi yang sering digunakan oleh *ProsesDataController*
5. *GlobalVal*: digunakan untuk memanggil nilai dari variabel – variabel global di dalam sistem, sehingga sistem mendapatkan perspektif yang sama dalam mengenali nilai variabel global

6. *TemplateEngine*: kelas yang digunakan untuk mengubah variabel model komponen menjadi kode program berdasarkan cetakan (*template*) yang diambil dari *domain knowledge*
7. *TemplateVariable*: kelas yang digunakan untuk memudahkan pengiriman parameter dari komponen *ProsesDataController* ke *model knowledge*

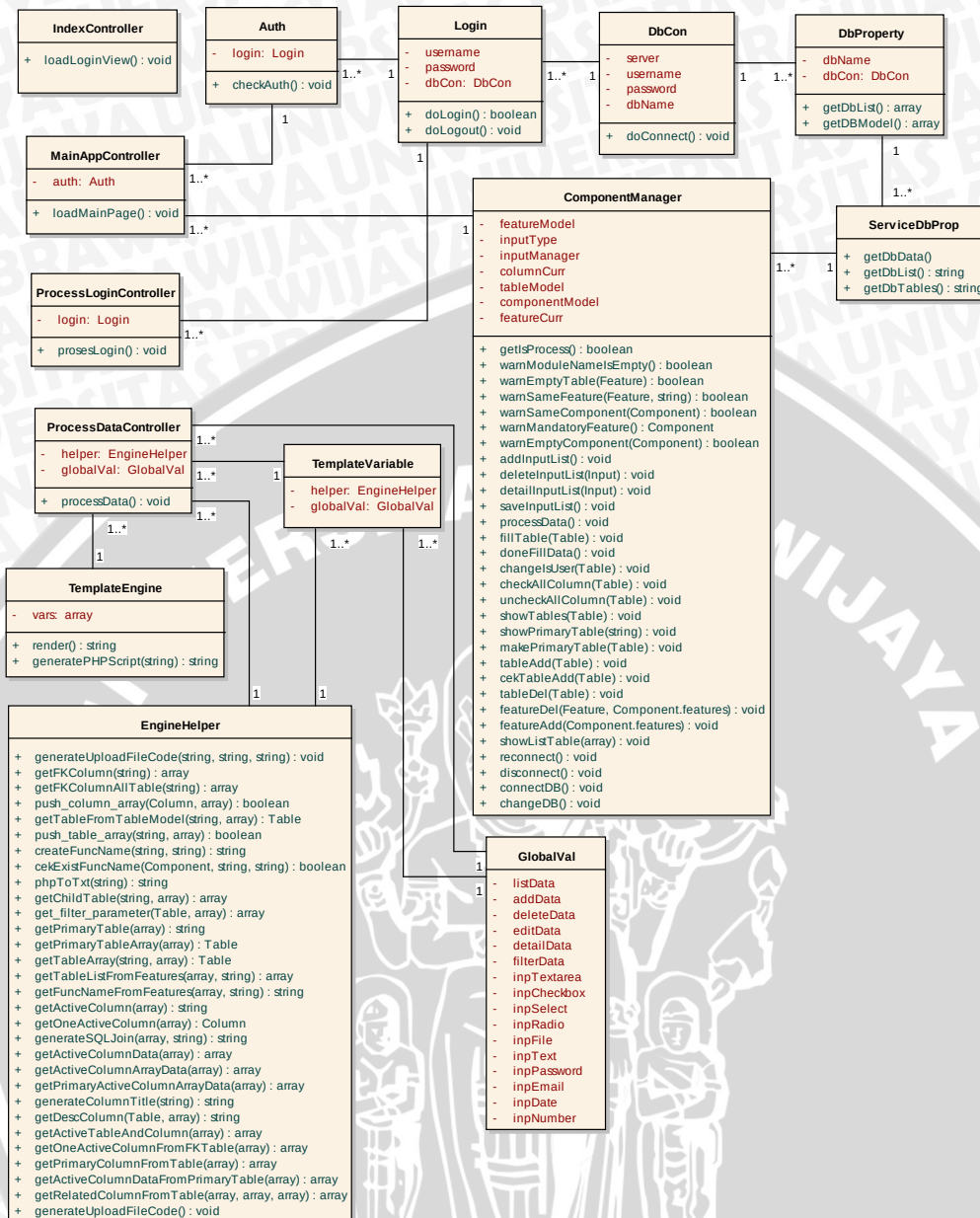


Gambar 5.77 Sequence Diagram Proses Pembangunan Kode Program

5.3.4.3 Perancangan Class Diagram

Perancangan *class diagram* dilakukan untuk mengetahui komponen – komponen penyusun sistem dan interaksi antar komponen. *Class diagram* digunakan untuk menjalankan implemantasi kode program untuk membangun sistem yang telah dirancang. Gambar 5.78 menunjukkan *class diagram* dari sistem *code generator* PTIIK-Apps. *Class diagram program generator* terdiri dari komponen:

1. *IndexController*: digunakan untuk memanggil komponen *loginView*
2. *MainAppController*: digunakan untuk memanggil komponen *componentManagerView*
3. *ProcessLoginController*: digunakan untuk mengontrol proses *login* dan *logout*
4. *Auth*: digunakan untuk mengecek hak akses pengguna
5. *Login*: digunakan untuk menjalankan proes *login* dan *logout*
6. *DbCon*: digunakan untuk melakukan koneksi dari sistem ke database
7. *DbProperty*: digunakan untuk mengambil data model tabel dari server database yang dimasukkan pengguna
8. *ServiceDbProp*: untuk mengontrol proses pengambilan mode tabel
9. *ComponentManager*: digunakan untuk menjalankan fungsi parameterisasi komponen
10. *GlobalVal*: digunakan untuk menyimpan nilai variabel – variabel global
11. *TemplateEngine*: digunakan untuk menguba dari model komponen menjadi kode program
12. *EngineHelper*: digunakan untuk memanggil fungsi – fungsi yang sering dipanggil oleh *domain knowledge* dan *ProcessDataController*
13. *TemplateVariable*: digunakan untuk melakukan parameterisasi variabel pada saat pembangunan kode program



Gambar 5.78 Class Diagram code Generator

BAB 6 IMPLEMENTASI

Pada bab ini dijelaskan tentang tahap – tahap implementasi dari hasil desain domain yang dilakukan pada sub-bab 5.3. Tahap implementasi meliputi: 1) implementasi *domain knowledge*, 2) implementasi *program generator* dan 3) implementasi halaman tatap muka.

6.1 Hasil implementasi *domain knowledge*

Implementasi *domain knowledge* terdiri dari tujuh komponen, yaitu: 1) *controller knowledge*, 2) *model knowledge*, 3) *view add knowledge*, 4) *view filter knowledge*, 5) *view list knowledge*, 6) *view edit knowledge*, 7) *view detail knowledge*.

6.1.1 Implementasi *controller knowledge*

Implementasi *controllerTemplate* berupa file dengan ekstensi .php. Komponen tersebut membaca data variabel `$this->var->features` untuk diseleksi jenis fiturnya. Kemudian kode program akan dibangkitkan berdasarkan jenis fitur tersebut. Implementasi kode *controller knowledge* ditunjukkan oleh Tabel 6.1.

Tabel 6.1 Implementasi Kode Program *controller knowledge*

1	[[
2	class <?php echo \$this->var->moduleName . \'_' . \$this-
3	>var->componentName ?> extends comsmodule {
4	
5	..generate fungsi <i>construct</i> komponen controller
6	
7	<?php foreach (\$this->var->features as \$featureKey):
8	?>
9	<?php \$funcName = \$this->var->helper-
10	>createFuncName(\$this->var->componentName,
11	\$featureKey->funcName); ?>
12	<?php if(\$featureKey->name == \$this->var->globalVal-
13	>listData): ?>
14	..generate fungsi controller list data
15	<?php endif; ?>
16	<?php elseif(\$featureKey->name == \$this->var-
17	>globalVal->filterData):
18	?>
19	..generate fungsi controller filter data
20	<?php endif;?>
21	<?php elseif(\$featureKey->name == \$this->var-
22	>globalVal->addData): ?>
23	..generate fungsi controller add data
24	<?php endif;?>
25	<?php elseif(\$featureKey->name == \$this->var-
26	>globalVal->editData): ?>
27	..generate fungsi controller edit data
28	<?php endif; ?>
29	

30	<?php elseif(\$featureKey->name == \$this->var-
31	>globalVal->deleteData): ?>
32	..generate fungsi controller hapus data
33	<?php endif; ?>
34	<?php if(\$featureKey->name == \$this->var->globalVal-
35	>detailData): ?>
36	..generate fungsi controller detail data
37	<?php endif; ?>
38	<?php endforeach; ?>
39	}
	/]]

6.1.2 Implementasi *model knowledge*

Implementasi *modelTemplate* berupa file dengan ekstensi .php. Komponen tersebut membaca data variabel `$this->var->features` untuk diseleksi jenis fiturnya. Kemudian kode program akan dibangkitkan berdasarkan jenis fitur tersebut. Implementasi kode *model knowledge* ditunjukkan oleh Tabel 6.2.

Tabel 6.2 Implementasi Kode Program *model knowledge*

1	[[
2	class <?php echo 'model_m' . \$this->var->componentName
3	?> extends model {
4	
5	private \$coms;
6	public function __construct() {
7	parent::__construct();
8	}
9	
10	<?php foreach (\$this->var->features as \$featureKey):
11	<?php \$funcName = \$this->var->helper-
12	>createFuncName(\$this->var->componentName,
13	\$featureKey->funcName); ?>
14	
15	<?php if(\$featureKey->name == \$this->var->globalVal-
16	>listData): ?>
17	..generate fungsi model list data
18	<?php endif; ?>
19	<?php if(\$featureKey->name == \$this->var->globalVal-
20	>filterData):
21	?>
22	..generate fungsi model filter data
23	<?php endif; ?>
24	<?php if(\$featureKey->name == \$this->var->globalVal-
25	>addData):
26	?>
27	..generate fungsi model add data
28	<?php endif; ?>
29	<?php
30	if(\$featureKey->name == \$this->var->globalVal-
31	>editData):
32	?>

```

33     ..generate fungsi model edit data
34 <?php endif; ?>
35 <?php if($featureKey->name == $this->var->globalVal-
36 >deleteData): ?>
37     ..generate fungsi model delete data
38 <?php endif; ?>
39 <?php if($featureKey->name == $this->var->globalVal-
40 >detailData): ?>
41     ..generate fungsi model detail data
42 <?php endif; ?>
42 <?php endforeach; //end of main foreach functionList
43 ?>
44
45 <?php
46     if($arrayGetDetail){
47         foreach ($arrayGetDetail as $stableTmp) {
48             ?>
49             ..generate fungsi get list data
50 <?php
51                 }
52             }
53     if($arrayGetDetailList){
54         foreach ($arrayGetDetailList as $stableTmp)
55     {
56         $arrayTmp = array($stableTmp);
57         $columnID    = $this->var->helper-
58 >getOneActiveColumn($arrayTmp);
59         ?>
60         ..generate fungsi get detail data
61 <?php
62             }
63         }
64         ?>
65     }
66 /]]

```

6.1.3 Implementasi Kode Program *view add knowledge*

Implementasi *view add knowledge* berupa file dengan ekstensi .php. Implementasi kode *view add knowledge* ditunjukkan oleh Tabel 6.3.

Tabel 6.3 Implementasi Kode Program *view add knowledge*

```

1  [[ $this->head(); /]]
2
3  ..generate header view
4
5  <?php                                     if($this->var->helper-
6  >cekExistFuncName($funcNameCurr, $listData, $this-
7  >var->features)): ?>
8      ..generate opsi go to list data
9  <?php endif; ?>
10
11 <div class="row" id="index-form">

```

```

12     <div class="col-md-12">
13     <?php
14         $tables      =      helper->getTableListFromFeatures
15     ($addData, $this->var->features);
16         $columnView      =      $this->var->helper-
17 >getPrimaryActiveColumnArrayData($tables);
18         $columnFK      =      $this->var->helper->getFKColumn
19     ($tables);
20         $tableCurr      =      $this->var->helper-
21 >getPrimaryTableArray($tables);
22         $tableChild      =      $this->var->helper-
23 >getChildTable($tables, $tableCurr->tableName);
24     ?>
25
26     <form ..generate form attribute >
27     <?php foreach($columnView as $key): ?>
28     <div class="form-group">
29     <label class="col-sm-2 control-label">
30         <?php echo ($key->COLUMN_NAME?)>
31     </label>
32
33     <div class="col-sm-10">
34     <?php
35     if(in_array($key, $columnFK)):
36         $columnDesc      =      $this->var->helper-
37 >getDescColumn($this->var->tableModel, $key-
38 >REFERENCED_TABLE_NAME);
39     ?>
40     ..generate input select option for columnFK
41     <?php else: ?>
42     <?php if($key->INPUT_TYPE == $inpTextarea): ?>
42     ..generate input textarea
43     <?php elseif($key->INPUT_TYPE == $inpSelect ):
44     ?>
45     ..generate input select option
46     <?php elseif($key->INPUT_TYPE == $inpCheckbox ||
47 $key->INPUT_TYPE == $inpRadio ):
48     ..generate input radio/checkbox
49     <?php endif; endif; ?>
50     <?php endforeach; ?>
51
52     <?php if($tableChild): ?>
53     <?php foreach($tableChild as $table): ?>
54     ..generate form add child entity
55     <?php endforeach; ?>
56     <?php endif ?>
57
58     <?php if($tableChild): ?>
59
60     <span style="display: none">
61     <?php foreach($tableChild as $table): ?>
62     <?php
63         foreach ($table->columns as $key):

```

```

64         if($key->IS_USED == 1 && $key->PRIMARY_KEY !=
65         "PRIMARY"):
66             ?>
67
68         <?php
69             if(in_array($key, $columnFK)):
70                 $columnDesc = $this->var->helper-
71                 >getDescColumn($this->var->tableModel, $key-
72                 >REFERENCED_TABLE_NAME);
73             ?>
74             ..generate input select option for folumnFK
75         <?php else: ?>
76         <?php if($key->INPUT_TYPE == $inpTextarea): ?>
77             ..generate input textarea
78         <?php elseif($key->INPUT_TYPE == $inpSelect ): ?>
79             ..generate input select option
80         <?php elseif($key->INPUT_TYPE == $inpCheckbox || $key-
81         >INPUT_TYPE == $inpRadio ):
82             ..generate input radio/checkbox
83         <?php endforeach; else: ?>
84             ..generate input tipe
85         <?php endif; endif; ?>
86         <?php endforeach; ?>
87         <?php endif; ?> //endif tableChild != null
88
89         [[ $this->foot(); //]]
90
91         <?php if($tableChild): ?>
92         <script type="text/javascript">
93             var count = {
94                 <?php foreach($tableChild as $table): ?>
95                 "<?php echo $table->tableName ?>": 0
96                 <?php endforeach; ?>
97             }
98             function add(key){
99                 var form = $("#form-
100 "+key).html().replace(/repVal/g, count[key]);
101                 $("#wrap-"+key).append(form);
102                 count[key]++;
103             }
104             function del(key, id){
105                 $("#input-"+key+"-"+id).remove();
106             }
107         </script>
108         <?php endif; ?>

```

6.1.4 Implementasi Kode Program *view list knowledge*

Implementasi *view list knowledge* berupa file dengan ekstensi .php. Implementasi kode *view list knowledge* ditunjukkan oleh Tabel 6.4.

Tabel 6.4 Implementasi Kode Program *view list knowledge*

1	[[\$this->head(); /]]
2	
3	..generate header view
4	
5	<?php if(\$this->var->helper-
6	>cekExistFuncName(\$funcNameCurr, \$listData, \$this-
7	>var->features)): ?>
8	..generate opsi go to list data
9	<?php endif; ?>
10	
11	<div class="row" id="index-form">
12	<div class="col-md-12">
13	<?php
14	\$tables = helper->getTableListFromFeatures
15	(\$addData, \$this->var->features);
16	\$columnView = \$this->var->helper-
17	>getPrimaryActiveColumnArrayData(\$tables);
18	\$columnFK = \$this->var->helper->getFKColumn
19	(\$tables);
20	\$tableCurr = \$this->var->helper-
21	>getPrimaryTableArray(\$tables);
22	\$tableChild = \$this->var->helper-
23	>getChildTable(\$tables, \$tableCurr->tableName);
24	?>
25	
26	<form ..generate form attribute >
27	<?php foreach(\$columnView as \$key): ?>
28	<div class="form-group">
29	<label class="col-sm-2 control-label">
30	<?php echo (\$key->COLUMN_NAME?)
31	</label>
32	
33	<div class="col-sm-10">
34	<?php
35	if(in_array(\$key, \$columnFK)):
36	\$columnDesc = \$this->var->helper-
37	>getDescColumn(\$this->var->tableModel, \$key-
38	>REFERENCED_TABLE_NAME);
39	?>
40	..generate input select option for columnFK
41	<?php else: ?>
42	<?php if(\$key->INPUT_TYPE == \$inpTextarea): ?>
42	..generate input textarea
43	<?php elseif(\$key->INPUT_TYPE == \$inpSelect):
44	?>
45	..generate input select option
46	<?php elseif(\$key->INPUT_TYPE == \$inpCheckbox
47	\$key->INPUT_TYPE == \$inpRadio):
48	..generate input radio/checkbox
49	<?php endif; endif; ?>
50	<?php endforeach; ?>
51	

```

52     <?php if($tableChild): ?>
53     <?php foreach($tableChild as $table): ?>
54         ..generate form add child entity
55     <?php endforeach; ?>
56     <?php endif ?>
57
58     <?php if($tableChild): ?>
59
60     <span style="display: none">
61     <?php foreach($tableChild as $table): ?>
62     <?php
63         foreach ($table->columns as $key):
64             if($key->IS_USED == 1 && $key->PRIMARY_KEY !=
65 "PRIMARY"):
66                 ?>
67
68     <?php
69         if(in_array($key, $columnFK)):
70             $columnDesc = $this->var->helper-
71 >getDescColumn($this->var->tableModel, $key-
72 >REFERENCED_TABLE_NAME);
73             ?>
74             ..generate input select option for folumnFK
75     <?php else: ?>
76     <?php if($key->INPUT_TYPE == $inpTextarea): ?>
77         ..generate input textarea
78     <?php elseif($key->INPUT_TYPE == $inpSelect ): ?>
79         ..generate input select option
80     <?php elseif($key->INPUT_TYPE == $inpCheckbox || $key-
81 >INPUT_TYPE == $inpRadio ):
82         ..generate input radio/checkbox
83     <?php endif; endif; ?>
84     <?php endforeach; ?>
85     <?php endif; ?> //endif tableChild != null
86
87     [[ $this->foot(); /]]
88
89     <?php if($tableChild): ?>
90     <script type="text/javascript">
91         var count = {
92             <?php foreach($tableChild as $table): ?>
93                 "<?php echo $table->tableName ?>": 0
94             <?php endforeach; ?>
95         }
96         function add(key){
97             var form = $("#form-
98 "+key).html().replace(/repVal/g, count[key]);
99             $("#wrap-"+key).append(form);
100             count[key]++;
101         }
102         function del(key, id){
103             $("#input-"+key+"-"+id).remove();
104         }

```

105	</script>
106	<?php endif; ?>

6.1.5 Implementas Kode Program *view filter knowledge*

Implementasi *view filter knowledge* berupa file dengan ekstensi .php. Implementasi kode *view filter knowledge* ditunjukkan oleh Tabel 6.5.

Table 6.5 Implementasi Kode Program *view filter knowledge*

1	[[\$this->head(); /]]
2	
3	..generate header
4	
5	<?php if(\$this->var->helper->cekExistFuncName(\$this->
6	>var->funcNameCurr, \$addData, \$this->var->features)):
7	?>
8	..generate opsi add data
9	<?php endif; ?>
10	<?php
11	\$tables = \$this->var->helper->
12	>getTableListFromFeatures(\$filterData, \$this->var->
13	>features);
14	\$funcName = \$this->var->helper->
15	>getFuncNameFromFeatures(\$filterData, \$this->var->
16	>features);
17	\$columnView = \$this->var->helper->
18	>getActiveColumnData(\$tables);
19	\$columnFK = \$this->var->helper->
20	>getFKColumnAllTable(\$tables);
21	\$tableCurr = \$this->var->helper->
22	>getPrimaryTableArray(\$tables);
23	\$columnFilter = \$this->var->helper->
24	>get_filter_parameter(\$columnFK, \$tableCurr);
25	?>
26	
27	<form>
28	<?php
29	if(\$columnFilter){
30	foreach (\$columnFilter as \$key) {
31	\$columnDesc = \$this->var->helper->
32	>getDescColumn(\$this->var->componentName, \$key->
33	>REFERENCED_TABLE_NAME);
34	?>
35	
36	<div class="form-group">
37	<label class="control-label">
38	..generate label header
39	</label>
40	
41	<div class="">
42	..generate input select option
42	</div>
43	</div>

```

44 <?php
45     }
46 }
47 ?>
48 ..generate button filter
49 </form>
50 </div>
51
52 <div class="col-md-8">
53     [[ if( isset($posts) ): /]]
54
55     <div class="block-box">
56         <table class='table table-hover' id='example'>
57             <thead>
58                 <tr>
59                     <th>No</th>
60                     <?php
61
62                     foreach($columnView as $value)
63                         echo
64                             '13%">' .ucfirst($value) . '</th>';
65                 ?>
66
67                 <th>&nbsp;</th>
68             </tr>
69         </thead>
70         <tbody>
71             [[ $no =1; foreach($posts as $key){ /]]
72             <tr>
73                 <td>[[ echo $no; $no++; /]]</td>
74                 <?php foreach($columnView as $value){ ?>
75                 <td>[[ echo $key-><?php echo $value ?> /]]</td>
76                 <?php } ?>
77                 <td>
78
79                 <ul class='nav nav-pills'>
80                     <li class='dropdown pull-right'>
81                         <a class='dropdown-toggle btn btn-table' data-
82 toggle='dropdown' href='#'>Action <b
83 class='caret'></b>
84                     </a>
85
86                     <ul id='menu1' class='dropdown-opsi' role='opsi'
87 aria-labelledby='drop4'>
88                         <?php foreach($this->var->features as $key){
89                         if($funcNameCurr == $key->funcName){ ?>
90
91                         <?php
92                         if($key->name == $detailData):
93                             $tablesDetail = $this->var->helper-
94 >getTableListFromFeatures($detailData, $this->var-
95 >features);
96

```

```

97         $columnID          =          $this->var->helper-
98 >getOneActiveColumn($tablesDetail);
99     ?>
100         ..generate opsi detail
101     <?php endif; //detail data button ?>
102     <?php if($key->name == $addData): ?>
103         ..generate opsi add
104     <?php endif; //add data button ?>
105     <?php
106         if($key->name == $editData):
107             $tablesEdit          =          $this->var->helper-
108 >getTableListFromFeatures($editData,          $this->var-
109 >features);
110             $columnID          =          $this->var->helper-
111 >getOneActiveColumn($tablesEdit);
112             ?>
113             ..generate opsi edit
114         <?php endif; ?>
115         <?php
116             if($key->name == $deleteData):
117                 $tablesDelete          =          $this->var->helper-
118 >getTableListFromFeatures($deleteData,          $this->var-
119 >features);
120                 $columnID          =          $this->var->helper-
121 >getOneActiveColumn($tablesDelete);
122                 ?>
123                 ..generate opsi delete
124             <?php endif; //delete data button ?>
125         <?php }} ?>
126
127     </ul>
128 </li>
129 </ul>
130 </td>
131 </tr>
132 [[ } /]]
133
134 </tbody>
135 </table>
136 </div>
137
138     [[ else: /]]
139     <div class="well" align="center">No Data
140 Available</div>
141
142     [[ endif; /]]
143 </div>
144 </div>
145
146     [[ $this->foot(); /]]
147 <?php
148     }

```

149	} ?>
-----	---------

6.1.6 Implementas Kode Program *view edit knowledge*

Implementasi *view edit knowledge* berupa file dengan ekstensi .php. Implementasi kode *view edit knowledge* ditunjukkan oleh Tabel 6.6.

Tabel 6.6 Implementasi Kode Program *view edit knowledge*

1	[[\$this->head(); /]]
2	
3	..generate header view
4	
5	<?php
6	if(\$this->var->helper-
7	>cekExistFuncName(\$funcNameCurr, \$listData, \$this-
8	>var->features)): ?>
9	..generate opsi list data
10	<?php endif; ?>
11	<div class="row" id="index-form">
12	<div class="col-md-12">
13	<?php
14	\$tables = \$this->var->helper-
15	>getTableListFromFeatures(\$editData, \$this->var-
16	>features);
17	\$columnView = \$this->var->helper-
18	>getPrimaryActiveColumnArrayData(\$tables);
19	\$columnFK = \$this->var->helper-
20	>getFKColumn(\$tables);
21	\$columnID = \$this->var->helper-
22	>getOneActiveColumn(\$tables);
23	\$tableCurr = \$this->var->helper-
24	>getPrimaryTableArray(\$tables);
25	
26	\$tableChild = \$this->var->helper-
27	>getChildTable(\$tables, \$tableCurr->tableName); //get
28	table child
29	?>
30	
31	<form>
32	<?php foreach(\$columnView as \$key): ?>
33	<div class="form-group">
34	<label class="col-sm-2 control-
35	label"><?php echo \$this->var->helper-
36	>generateColumnTitle(\$key->COLUMN_NAME) ?></label>
37	<div class="col-sm-10">
38	<?php
39	if(in_array(\$key, \$columnFK)):
40	\$columnDesc = \$this->var->helper-
41	>getDescColumn(\$this->var->tableModel, \$key-
42	>REFERENCED_TABLE_NAME);
42	?>
43	

```

44      ..generate input select option untuk columnFK
45      <?php else: ?>
46      <?php if($key->INPUT_TYPE == $inpTextarea): ?>
47      ..generate input textarea
48      <?php elseif($key->INPUT_TYPE == $inpSelect ):
49      ?>
50      ..generate input select option
51      <?php elseif($key->INPUT_TYPE == $inpCheckbox ||
52      $key->INPUT_TYPE == $inpRadio ):
53      foreach($key->INPUT_OPTION as $input):
54      ?>
55      ..generate input checkbox/radio
56      <?php endforeach; else: ?>
57      ..generate input tipe
58      <?php endif; endif; ?>
59      <?php endforeach; ?>
60
61      <?php if($tableChild): ?>
62      <?php foreach($tableChild as $table): ?>
63      ..generate header child edit form
64      <button>Add<?php      echo      $this->var->helper-
65      >generateColumnName($table->tableName) ?>
66      </button>
67
68      <span id="wrap-<?php echo $table->tableName ?>">
69      [[ $count['<?php echo $table->tableName ?>'] =
70      0; foreach($detail_<?php echo $table->tableName ?> as
71      $key): /]]
72
73      <span id="input-<?php echo $table->tableName . '-'
74      ?>[[ $count['<?php echo $table->tableName ?>'] /]] ">
75      <input      type="hidden"
76      class="form-control      typeahead"      name="primary_key"
77      value="" />
78
79      <?php foreach ($table->columns as $key):
80      if($key->IS_USED == 1 && $key->REFERENCED_TABLE_NAME
81      != 'order_list' && $key->PRIMARY_KEY != "PRIMARY"): ?>
82      ..generate input select option untuk columnFK
83      <?php else: ?>
84      <?php if($key->INPUT_TYPE == $inpTextarea): ?>
85      ..generate input textarea
86      <?php elseif($key->INPUT_TYPE == $inpSelect ):
87      ?>
88      ..generate input select option
89      <?php elseif($key->INPUT_TYPE == $inpCheckbox ||
90      $key->INPUT_TYPE == $inpRadio ):
91      foreach($key->INPUT_OPTION as $input):
92      ?>
93      ..generate input checkbox/radio
94      <?php endforeach; else: ?>
95      ..generate input tipe
96      <?php endif; endif; ?>

```

```

97      <?php endif; endforeach; ?>
98
99      ..generate button delete child data
100     [[ endforeach; /]]
101     </span>
102     <?php endforeach; ?>
103     <?php endif ?>
104
105
106     <?php if($tableChild): ?>
107
108     ..generate wrapper add child data
109     <?php foreach ($table->columns as $key): if($key-
110     >IS_USED == 1 && $key->PRIMARY_KEY != "PRIMARY"): ?>
111
112         <?php
113         if(in_array($key, $columnFK)):
114             $columnDesc = $this->var->helper-
115             >getDescColumn($this->var->tableModel, $key-
116             >REFERENCED_TABLE_NAME);
117             ?>
118             ..generate in
119             <?php else: ?>
120             <?php if($key->INPUT_TYPE == $inpTextarea): ?>
121             ..generate input textarea
122             <?php elseif($key->INPUT_TYPE == $this->var-
123             >globalVal->inpSelect ): ?>
124             ..generate input select option
125             <?php elseif($key->INPUT_TYPE == $inpCheckbox ||
126             $key->INPUT_TYPE == $inpRadio ):
127                 foreach($key->INPUT_OPTION as $input):
128                     ?>
129                     ..generate input checkbox/radio
130                     <?php endforeach; else: ?>
131                     ..generate input
132                     <?php endif; endif; ?>
133                     <?php endif; endforeach; ?>
134
135             ..generate button delete child data
136             <?php endforeach; ?>
137     <?php endif; ?>
138
139     [[ $this->foot(); /]]
140
141     <script type="text/javascript">
142         var count = {
143             <?php $i=0; foreach($tableChild as $table):
144             if($i > 0) echo ", "; $i++; ?><?php echo $table-
145             >tableName ?>": [[ echo $count['<?php echo $table-
145             >tableName ?>'] /]]
146             <?php endforeach; ?>
147         }
148         function add(key){

```

```

149         var         form         =         $("#form-
150 "+key).html().replace(/repVal/g, count[key]);
151         $("#wrap-"+key).append(form);
152         count[key]++;
153     }
154     function del(key, id){
155         $("#[name=init-"+key+"-"+id+"]").val('DELETE');
156         $("#input-"+key+"-"+id).hide();
157     }
158 </script>
159 <?php
160     }
161 }
162 ?>
163

```

6.1.7 Implementas Kode Program *view detail knowledge*

Implementasi *view detail knowledge* berupa file dengan ekstensi .php. Implementasi kode *view detail knowledge* ditunjukkan oleh Tabel 6.7.

Tabel 6.7 Implementasi Kode Program *view detail knowledge*

```

1  [[ $this->head(); /]]
2
3  ..generate header view
4  <?php                                     if($this->var->helper-
5  >cekExistFuncName($funcNameCurr, $listData, $this-
6  >var->features)): ?>
7      ..generate opsi goto list data
8  <?php endif; ?>
9
10 <?php
11     $tables         =         $this->var->helper-
12 >getTableListFromFeatures($detailData, $this->var-
13 >features);
14     $columnView     =         $this->var->helper-
15 >getActiveColumnData($tables);
16     $tableCurr      =         $this->var->helper-
17 >getPrimaryTableArray($tables);
18     $tableChild     =         $this->var->helper-
19 >getChildTable($tables, $tableCurr->tableName); //get
20 table child
21 ?>
22
23 <table>
24     <?php foreach($columnView as $key): ?>
25         <tr>
26             <td>
27                 <?php         echo         $this->var->helper-
28 >generateColumnTitle($key) ?>
29             </td>
30             <td>[[ $posts['<?php echo $key ?>'] /]]</td>

```

```

31         </tr>
32     <?php endforeach; ?>
33 </table>
34
35 <?php
36 if($tableChild): //data for detail child
37     foreach ($tableChild as $tableChildTmp):
38         ?>
39         <h3 class="page-header">
40             <?php echo $this->var->helper-
41                 >generateColumnTitle($tableChildTmp->tableName)
42             ?> Detail
43         </h3>
44         <table class="table table-condensed table-bordered">
45             <tr>
46                 <?php
47                 foreach ($tableChildTmp->columns as $key) {
48                     if($key->IS_USED == '1') {
49                         ?>
50
51                     <th>
52                         <?php echo $this->var->helper-
53                             >generateColumnTitle($key->COLUMN_NAME) ?>
54                     </th>;
55                     <?php } } ?>
56             </tr>
57
58             [[
59                 foreach($detail_<?php echo $tableChildTmp-
60                     >tableName ?> as $key):
61             /]]
62             <tr>
63                 <?php
64                 foreach ($tableChildTmp->columns as $key) {
65                     ?>
66                     <td>
67                         [[ echo $key-><?php echo $key->COLUMN_NAME ?> /]]
68                     </td>
69                     <?php } ?>
70             </tr>
71             [[ endforeach; /]]
72             </table>
73         <?php
74             endforeach;
75             endif;
76         ?>
77
78     </div>
79 </div>
80
81 [[ $this->foot(); /]]
82

```

--	--

6.2 Hasil implementasi *program generator*

Implementasi *program generator* berupa beberapa file dengan ekstensi .php. berikut ini paparan kode program penyusun *program generator*.

6.2.1 Implementasi kode program *login* dan *logout*

Implementasi kode program fungsi *login* dan *logout* ditunjukkan pada Tabel 6.8.

Tabel 6.8 Implementasi Kode Program *Login* dan *Logout*

1	function doLogin(){
2	\$sql = "SELECT * FROM tbl_user WHERE username =
3	'\$this->username' AND password = '\$this->password'";
4	\$x = mysql_query(\$sql);
5	
6	\$data = mysql_fetch_array(\$x);
7	
8	if(\$data == null) return false;
9	session_start();
10	\$_SESSION['id_user'] = \$data['id_user'];
11	\$_SESSION['username'] = \$data['username'];
12	
13	return true;
14	}
15	
16	function doLogout(){
17	session_start();
18	
19	session_unset();
20	session_destroy();
21	}

Sumber: [Implementasi]

6.2.2 Implementasi kode program kelola sambungan server database

Implementasi kode program kelola sambungan ke server database terdiri dari beberapa poin, yaitu: 1) sambungan ke server database, 2) pemutusan sambungan dari server database, 3) muat ulang sambungan dan 4) mengambil model tabel dari unit database yang dipilih.

Implementasi kode program sambungan ke server database ditunjukkan oleh Tabel 6.9.

Tabel 6.9 Implementasi Kode Program Sambungan ke Server Database

1	\$scope.dbCurr = 'No db selected';
2	\$scope.connectDB = function(){
3	\$scope.finishDbList = false;
4	\$scope.loadDbList = true;
5	\$scope.dbList = [];
6	

```

7
8     $.ajax({
9         type: "POST",
10        dataType: "html",
11        data:
12            $.param({
13                server: $scope.server,
14                username: $scope.username,
15                password: $scope.password }),
16        url: "serviceDbProp.php?type=dbList",
17        success: function(msg) {
18            try{
19                var json = JSON.parse(msg);
20                $scope.dbList = json;
21                $scope.dbCurr = $scope.dbList[0];
22                $scope.changeDB($scope.dbList[0]);
23                $('#modal-connection').modal('hide');
24            } catch(e) {
25                alert('Connection failed');
26            }
27        },
28        complete: function(){
29            $scope.loadDbList = false;
30            $scope.finishDbList = true;
31
32            $scope.btnLoadReconnect = false;
33            $scope.btnReconnect = true;
34
35            $scope.$apply();
36        }
37    });
38 }

```

Implementasi kode program pemutusan sambungan dari server database ditunjukkan oleh Tabel 6.10.

Table 6.10 Implementasi Kode Program Pemutusan Sambungan dari Server Database

```

1    $scope.disconnect = function(){
2        if(confirm("Are you wanna disconnect DB model?
3            (component model will be deleted)")){
4            $scope.dbList = [];
5            $scope.dbModel = [];
6            $scope.componentModel = [];
7            $scope.dbCurr = 'No db selected';
8        }
9    }

```

Implementasi kode program muat ulang sambungan ke server database ditunjukkan oleh Tabel 6.11.

Tabel 6.11 Implementasi Kode Program Muat Ulang Sambungan Server Database

1	\$scope.reconnect = function(){
2	\$scope.btnLoadReconnect = true;
3	\$scope.btnReconnect = false;
4	
5	\$scope.connectDB();
6	}

Implementasi kode program pengambilan model tabel dari server database ditunjukkan oleh Tabel 6.12.

Tabel 6.12 Implementasi Kode Program Pengambilan Model Tabel dari Server Database

1	\$scope.changeDB = function(dbName){
2	\$scope.btnChangeDb = false;
3	\$scope.btnLoadChangeDb = true;
4	
5	\$.ajax({
6	type: "POST",
7	dataType: "html",
8	data:
9	\$.param({
10	server: \$scope.server,
11	username: \$scope.username,
12	password: \$scope.password,
13	db_name: dbName }),
14	url: "serviceDbProp.php?type=dbProp",
15	success: function(msg){
16	// console.log(msg);
17	try{
18	var data = JSON.parse(msg);
19	\$scope.dbModel = data;
20	} catch(e) {
21	alert('Connection failed');
22	}
23	},
24	complete: function(){
25	\$scope.btnChangeDb = true;
26	\$scope.btnLoadChangeDb = false;
27	\$scope.\$apply();
28	
29	\$scope.btnChangeDb = true;
30	\$scope.btnLoadChangeDb = false;
31	})
32	});
33	}

6.2.3 Implementasi kode program pengisian model masukan

Implementasi kode program pengisian model masukan terdiri dari beberapa poin, yaitu: 1) kelola model komponen, 2) kelola model fitur, 3) *Table Manager*, 4) *Column Manager*, dan 5) *Input-Option Manager*.

Implementasi kode program kelola model komponen ditunjukkan oleh Tabel 6.13.

Tabel 6.13 Implementasi Kode Program Kelola Model Komponen

1	//hapus model komponen
2	\$scope.componentDel = function(item){
3	if(confirm('Delete this component?')){
4	var index =
5	\$scope.componentModel.indexOf(item);
6	\$scope.componentModel.splice(index, 1);
7	}
8	}
9	
10	//tambah model komponen
11	\$scope.componentAdd = function(){
12	var temp = {
13	"name": "",
14	"features": [
15	{
16	"name": "",
17	"funcName": "",
18	"tables": [
19	]
20	}
21	]
22	};
23	
24	\$scope.componentModel.push(temp);
25	}

Implementasi kode program kelola model fitur ditunjukkan oleh Tabel 6.14.

Tabel 6.14 Implementasi Kode Program Kelola Model Fitur

1	//hapus model fitur
2	\$scope.featureDel = function(data, item){
3	if(confirm('Delete this feature?')){
4	var index = data.indexOf(item);
5	data.splice(index, 1);
6	
7	\$scope.featureCurr = null;
8	\$scope.tableManagerShow = true;
9	\$scope.columnManagerShow = false;
10	\$scope.inputManagerShow = false;
11	}
12	}
13	
14	//tambah model fitur

```

15 $scope.featureAdd = function(data){
16     var temp = {
17         "name": "",
18         "funcName": "",
19         "tables": []
20     };
21     data.push(temp);
22 }

```

Implementasi kode program *Table Manager* ditunjukkan oleh Tabel 6.15.

Tabel 6.15 Implementasi Kode Program *Table Manager*

```

1 //table parameterization manager
2 //go to table manager
3 $scope.showTables = function(data){
4     if(data.length == 0)
5         $scope.noteTableEmpty = true;
6     else $scope.noteTableEmpty = false;
7     $scope.featureCurr = data;
8     $scope.columnManagerShow = false;
9     $scope.tableManagerShow = true;
10 }
11 //untuk mencetak status primary tabel
12 $scope.showPrimaryTable = function(data){
13     if(data == "0") return "fa fa-star grey";
14     else return "fa fa-star gold";
15 }
16 //setel menjadi primary table
17 $scope.makePrimaryTable = function(data){
18     if($scope.featureCurr != null){
19         $.each($scope.featureCurr, function(k, v){
20             v.isPrimaryTable = "0"
21         });
22     }
23     data.isPrimaryTable = "1";
24 }
25
26 //tableModel manager
27 //tambah model tabel
28 $scope.tableAdd = function(data){
29     var dataJSON = JSON.stringify(data);
30     if($scope.featureCurr){
31         if($scope.featureCurr.length == 0) {
32             data.isPrimaryTable = "1";
33             var dataJSON = JSON.stringify(data);
34         }
35         if($scope.cekTableAdd(dataJSON)){
36             $scope.featureCurr.push(JSON.parse(dataJSON));
37         }
38         else alert('Can\'t add same table');
39         $scope.noteTableEmpty = false;
40     }
41 }
42 }

```

```

43         else alert('Select component\'s feature first');
44     }
45     //periksa bahwa tidak ada model tabel yang identik
46     $scope.cekTableAdd = function(data){
47         var isAdd = 1;
48         data = JSON.parse(data);
49         $.each($scope.featureCurr, function(k, v){
50             if(data.$$hashKey == v.$$hashKey) {
51                 isAdd = 0;
52             }
53         });
54     };
55     return isAdd;
56 }
57 //hapus model tabel
58 $scope.tableDel = function(item){
59     if(confirm('Delete this table from current
60 feature?')){
61         var index =
62 $scope.featureCurr.indexOf(item);
63         $scope.featureCurr.splice(index, 1);
64     }
65 }

```

Implementasi kode program *Input-Option Manager* ditunjukkan oleh Tabel 6.16.

Tabel 6.16 Implementasi Kode Program *Column Manager*

```

1 //goto input-option manager
2 $scope.fillTable = function(table){ //column manager
3     $scope.tableFillData = table;
4     $scope.columnManagerShow = true;
5     $scope.tableManagerShow = false;
6 }
7
8 //close input-option manager
9 $scope.doneFillData = function(){ //column manager
10 finish
11     $scope.columnManagerShow = false;
12     $scope.tableManagerShow = true;
13 }
14
15 //change variabel is used in column model
16 $scope.changeIsUser = function(data){ //change isUsed
17     if(data.IS_USED == '1') data.IS_USED = '0';
18     else data.IS_USED = '1';
19 }
20
21 //check is used to all column model
22 $scope.checkAllColumn = function(tableFillData){
23     if(tableFillData.columns){
24         $.each(tableFillData.columns, function(k, v){
25             v.IS_USED = 1;

```

26	});
27	}
28	}
29	//unchecked is used to all column model
30	\$scope.uncheckAllColumn = function(tableFillData){
31	if(tableFillData.columns){
32	\$.each(tableFillData.columns, function(k, v){
33	v.IS_USED = 0;
34	});
35	}
36	}

Implementasi kode program *Column Manager* ditunjukkan oleh Tabel 6.17.

Tabel 6.17 Implementasi Kode Program *Input-Option Manager*

1	//tambah model pilihan masukan
2	\$scope.addInputList = function(){
3	\$scope.inputManager.push({
4	key: "", value: ""
5	});
6	}
7	
8	//hapus model pilihan masukan
9	\$scope.deleteInputList = function(data){
10	if(confirm('Delete this input option?')){
11	var index = \$scope.inputManager.indexOf(data);
12	\$scope.inputManager.splice(index, 1);
13	}
14	}
15	
16	//goto input-option manager
17	\$scope.detailInputList = function(data){
18	\$scope.columnManagerShow = false;
19	\$scope.inputManagerShow = true;
20	\$scope.columnInputCurr = data.COLUMN_NAME;
21	\$scope.columnCurr = data;
22	var dataJSON
23	= JSON.stringify(data.INPUT_OPTION);
24	
25	if(dataJSON == '') {
26	\$scope.inputManager = [];
27	} else{
28	\$scope.inputManager
29	= JSON.parse(dataJSON);
30	}
31	}
32	
33	//tutup model pilihan masukan
34	\$scope.saveInputList = function(){
35	\$scope.columnManagerShow = true;
36	\$scope.inputManagerShow = false;
37	var dataJSON
38	= JSON.stringify(\$scope.inputManager);

39	
40	\$scope.columnCurr.INPUT_OPTION
41	= JSON.parse(dataJSON);
42	}

6.2.4 Implementasi kode program proses data model komponen menjadi kode program

Implementasi kode program proses data model komponen menjadi kode program ditunjukkan oleh Tabel 6.18.

Tabel 6.18 Implementasi Kode Program Proses Data Model Menjadi Kode Program

1	\$resultCode = array();
2	foreach(\$componentModel as \$key => \$component){
3	// echo \$component->name;
4	
5	\$engine = new TemplateEngine();
6	\$templateVar = new TemplateVariable();
7	\$templateVar->moduleName = \$moduleName;
8	\$templateVar->tableModel = \$tableModel;
9	\$templateVar->componentName = \$component->name;
10	\$templateVar->componentTitle
11	= ucfirst(\$component->name);
12	\$templateVar->features = \$component->features;
13	\$templateVar->dbName = \$dbName;
14	\$engine->var = \$templateVar;
15	
16	\$controllerCode
17	= \$engine->render('template/controllerTemplate.php');
18	\$file = 'result/controller/' . \$component-
19	>name . ".txt";
20	file_put_contents(\$file, \$controllerCode);
21	
22	\$modelCode = \$engine-
23	>render('template/modelTemplate.php');
24	\$file = 'result/model/' . \$component->name . ".txt";
25	file_put_contents(\$file, \$modelCode);
26	
27	\$viewCode = array();
28	foreach(\$component->features as \$feature){
29	\$templateVar->funcNameCurr = \$feature->funcName;
30	\$engine->var = \$templateVar;
31	if(\$feature->name == \$this->globalVal->listData) {
32	\$viewCodeList = \$engine-
33	>render('template/viewListTemplate.php');
34	array_push(\$viewCode, \$this->helper-
35	>createFuncName(\$component->name, \$feature->funcName)
36	. '_list.php');
37	\$file = 'result/view/' . \$this->helper-
38	>createFuncName(\$component->name, \$feature->funcName)
39	. '_list.txt';
40	file_put_contents(\$file, \$viewCodeList);

```

41     }
42     elseif($feature->name == $this->globalVal->editData)
43     {
44         $viewCodeEdit = $engine-
45 >render('template/viewEditTemplate.php');
46         array_push($viewCode, 'edit_'. $this->helper-
47 >createFuncName($component->name, $feature->funcName)
48 . '.php');
49         $file = 'result/view/' . 'edit_'. $this->helper-
50 >createFuncName($component->name, $feature->funcName)
51 . '.txt';
52         file_put_contents($file, $viewCodeEdit);
53     }
54     elseif($feature->name == $this->globalVal->addData) {
55         $viewCodeAdd = $engine-
56 >render('template/viewAddTemplate.php');
57         array_push($viewCode, 'add_'. $this->helper-
58 >createFuncName($component->name, $feature->funcName)
59 . '.php');
60         $file = 'result/view/' . 'add_'. $this->helper-
61 >createFuncName($component->name, $feature->funcName)
62 . '.txt';
63         file_put_contents($file, $viewCodeAdd);
64     }
65     elseif($feature->name == $this->globalVal-
66 >detailData) {
67         $viewCodeDetail = $engine-
68 >render('template/viewDetailTemplate.php');
69         array_push($viewCode, 'detail_'. $this->helper-
70 >createFuncName($component->name, $feature->funcName)
71 . '.php');
72         $file = 'result/view/' . 'detail_'. $this->helper-
73 >createFuncName($component->name, $feature->funcName)
74 . '.txt';
75         file_put_contents($file, $viewCodeDetail);
76     }
77     elseif($feature->name == $this->globalVal-
78 >filterData) {
79         $viewCodeFilter = $engine-
80 >render('template/viewFilterTemplate.php');
81
82         array_push($viewCode, $this->helper-
83 >createFuncName($component->name, $feature->funcName)
84 . '_filter_list.php');
85         $file = 'result/view/' . $this->helper-
86 >createFuncName($component->name, $feature->funcName)
87 . '_filter_list.txt';
88         file_put_contents($file, $viewCodeFilter);
89     }
90 }
91
92 $tmpCode = array(
93     "componentName" => $component->name,

```

```

94     "controllerName"
95         => $component->name.".php",
96     "modelName"
97         => 'm' . $component->name.".php",
98     "views" => $viewCode
99 );
100 array_push($resultCode, $tmpCode);
101 }
102
103

```

6.3 Hasil Implementasi Halaman Tatap Muka

Hasil implementasi halaman tatap muka *program generator* terdiri dari tiga halaman, yaitu: 1) halaman *login*, 2) halaman parameterisasi komponen dan 3) halaman unduh kode program.

6.3.1 Implementasi halaman *login*

Halaman *login* berisi form masukan yang harus diisi pengguna untuk melakukan proses *login*. Form tersebut terdiri dari masukan *username* dan *password*. Setelah mengisi masukan pada form *login*, pengguna harus menekan tombol "Login" untuk melakukan validasi akun. Implementasi halaman *login* ditunjukkan oleh Gambar 6.1.

Berikut ini keterangan fungsi tampilan Gambar 6.1:

1. Form Login
2. Masukan *username*
3. Masukan *password*
4. Tombol *login*



Gambar 6.1 Implementasi Tatap Muka Halaman *Login*

6.3.2 Kelola sambungan koneksi

Implementasi tatap muka untuk memulai sambungan ke server database ditunjukkan oleh Gambar 6.2. Pada tampilan ini pengguna dapat melakukan

sambungan ke server database. Pengguna harus mengisi masukan: 1) domain server database, 2) akses username server database, 3) dan akses password server database.

Berikut ini keterangan fungsi tampilan pada Gambar 6.2:

1. form sambungan ke server database
2. masukan domain server database
3. masukan username server database
4. masukan password server database
5. tombol untuk menyambungkan ke server database
6. tombol untuk menutup form sambungan ke server database (membatalkan proses sambungan)

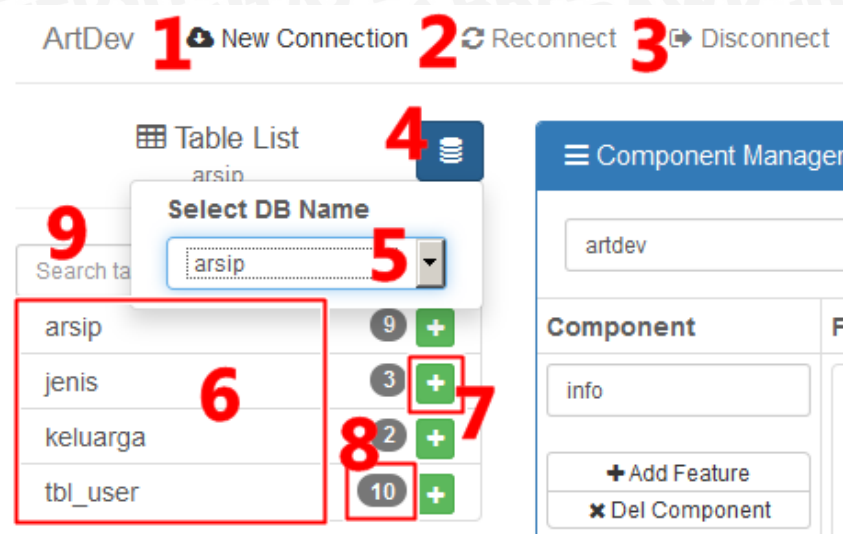


Gambar 6.2 Implementasi Tampilan Sambungan ke Server Database

Implementasi tampilan kelola sambungan koneksi database ditunjukkan oleh Gambar 6.3. Pada tampilan ini, pengguna dapat menyetel sambungan server database dan memuat model tabel dari unit database server ke dalam *program generator*.

Berikut ini keterangan fungsi tampilan pada Gambar 6.3:

1. tombol untuk membuat sambungan baru ke server database
2. tombol muat ulang sambungan
3. tombol pemutusan sambungan
4. tombol untuk menampilkan form pilihan daftar unit database
5. form pilihan daftar unit database
6. daftar model tabel yang diambil dari server database
7. tombol menambahkan model tabel pada proses *Table Manager*
8. keterangan jumlah kolom pada model tabel yang bersangkutan
9. form untuk menyortir model tabel



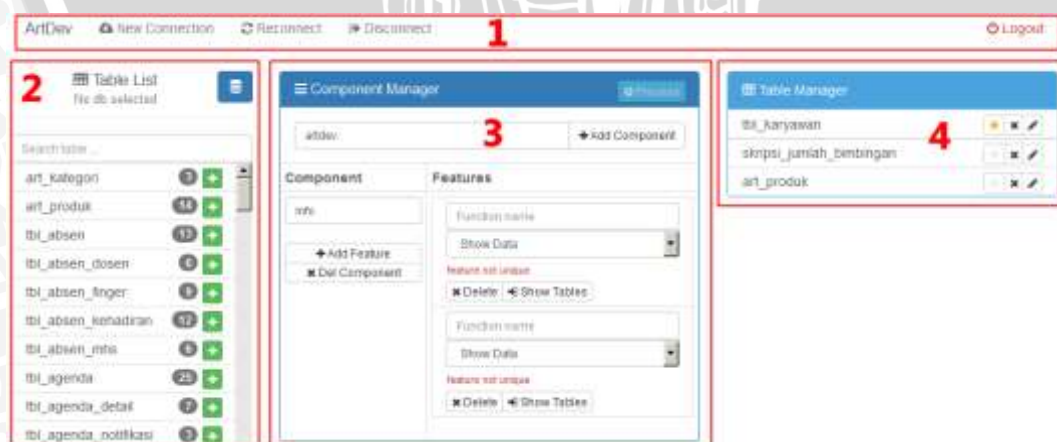
Gambar 6.3 Implementasi Tatap Muka Kelola Sambungan Server Database dan Kelola Model Tabel

6.3.3 Implementasi halaman pengisian model masukan

Halaman pengisian model masukan dibagi menjadi empat bagian, dimana setiap bagian memiliki fungsi yang berbeda – beda. Gambar 6.4 menjelaskan fungsi dari masing – masing bagian pada halaman parameterisasi komponen.

Berikut ini keterangan fungsi tampilan pada Gambar 6.4:

1. Menu *navbar* yang berisi opsi – opsi untuk mengelola sambungan ke *server database* dan tombol *logout*.
2. Bagian untuk mengelola model tabel yang diambil dari *server database*
3. Bagian untuk mengelola model komponen dan model fitur
4. Bagian untuk menampilkan fungsi *Table Manager*, *Column Manager* dan *Input-Option Manager*



Gambar 6.4 Implementasi Tatap Muka Halaman Parameterisasi Komponen

Gambar 6.5 menunjukkan tampilan tatap muka yang memungkinkan pengguna untuk mengelola model komponen dan model fitur. Berikut ini keterangan fungsi dari masing – masing tampilan pada Gambar 6.5:

1. tombol untuk memproses model komponen menjadi kode program
2. tombol untuk memasukkan nama modul yang akan dibangkitkan
3. tombol untuk menambahkan model komponen baru
4. masukan untuk nama model komponen
5. tombol untuk menambahkan model fitur baru pada model komponen yang bersangkutan
6. tombol untuk menghapus model komponen
7. masukan nama fungsi untuk model fitur yang bersangkutan
8. masukan untuk jenis model fitur
9. tombol untuk menghapus model fitur yang bersangkutan
10. tombol untuk melakukan *table manager* pada model fitur yang bersangkutan

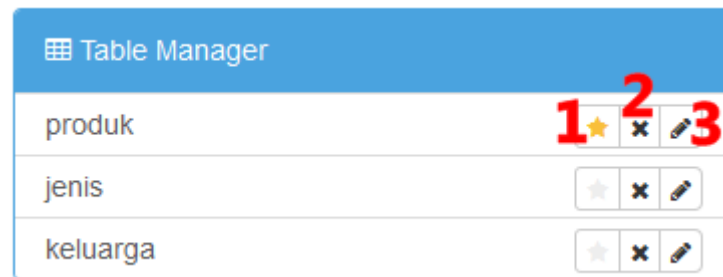
The screenshot shows a web application titled 'Component Manager'. At the top right is a 'Process' button (1). Below it is a 'Modul Name' input field (2) with a '+ Add Component' button (3) to its right. A red error message 'module name is empty' is displayed below the input field. The main area is divided into two columns: 'Component' and 'Features'. In the 'Component' column, there is a 'Component name' input field (4) with a red error message 'component name is empty' below it. Below this are two buttons: '+ Add Feature' (5) and 'x Del Component' (6). In the 'Features' column, there is a 'Function name' input field (7) with a dropdown menu (8) below it. A red error message 'feature not unique' is displayed below the dropdown. Below the dropdown are two buttons: 'x Delete' (9) and 'Show Tables' (10). At the bottom of the 'Features' column, there is another 'Function name' input field, a dropdown menu, a red error message 'feature not unique', and a red error message 'table model is empty'. Below these are two buttons: 'x Delete' and 'Show Tables'.

Gambar 6.5 Implementasi Tatap Muka untuk Mengelola Model Komponen dan Model Fitur

Implementasi tatap muka *Table Manager* ditunjukkan pada Gambar 6.6. Berikut ini keterangan fungsi dari masing – masing tampilan pada Gambar 6.6:

1. tombol untuk menjadikan model tabel yang bersangkutan menjadi model tabel utama (warna kuning menunjukkan nilai model tabel utama = 1, warna abu – abu menunjukkan nilai model tabel utama = 0)
2. tombol untuk menghapus model tabel yang bersangkutan

3. tombol untuk menjalankan *Column Manager* dari model tabel yang bersangkutan



Gambar 6.6 Implementasi Tatap Muka *Table Manager*

Gambar 6.7 menunjukkan tampilan tatap muka proses *Column Manager*. Berikut ini keterangan fungsi dari masing – masing tampilan pada Gambar 6.7:

1. tombol untuk menyetel nilai IS_USED semua model kolom menjadi 1
2. tombol untuk menyetel nilai IS_USED semua model kolom menjadi 0
3. masukan untuk memilih jenis model kolom
4. tombol untuk melakukan *Input-Option Manager* pada model kolom yang bersangkutan
5. sebuah tombol yang menandakan bahwa model kolom yang bersangkutan **tidak dipilih**, jika ditekan akan membuat model kolom **aktif** dan akan diproses oleh sistem
6. sebuah tombol yang menandakan bahwa model kolom yang bersangkutan **dipilih**, jika ditekan akan membuat model kolom **tidak aktif** dan akan diproses oleh sistem
7. tombol untuk menutup tampilan *Column Manager*
8. tampilan yang memberikan informasi bahwa model kolom yang bersangkutan adalah *primary key*
9. tampilan yang memberikan informasi tentang hubungan antara model kolom yang bersangkutan dengan model kolom lainnya

Column Manager : art_produk

1 ☒ check all ☐ unchecked all 2

3 4 5

8 produk_id (PK) text

9 FK (kategori_id in art_kategori) kategori_id text

6 judul text

keterangan text

harga num

gambar file

stok num

7 Save

Gambar 6.7 Implementasi Tatap Muka Column Manager

Gambar 6.8 menunjukkan tampilan tatap muka proses *Input-Option Manager*. Berikut ini keterangan fungsi dari masing – masing tampilan pada Gambar 6.8:

1. tombol untuk menambahkan model pilihan masukan baru
2. masukan untuk kolom *key*
3. masukan untuk kolom *value*
4. tombol untuk menghapus model pilihan masukan yang bersangkutan
5. tombol untuk menutup tampilan *Input-Option Manager*

Input Manager : gender

1 Add Input Option

2 L 3 Laki - laki 4 x

P Perempuan x

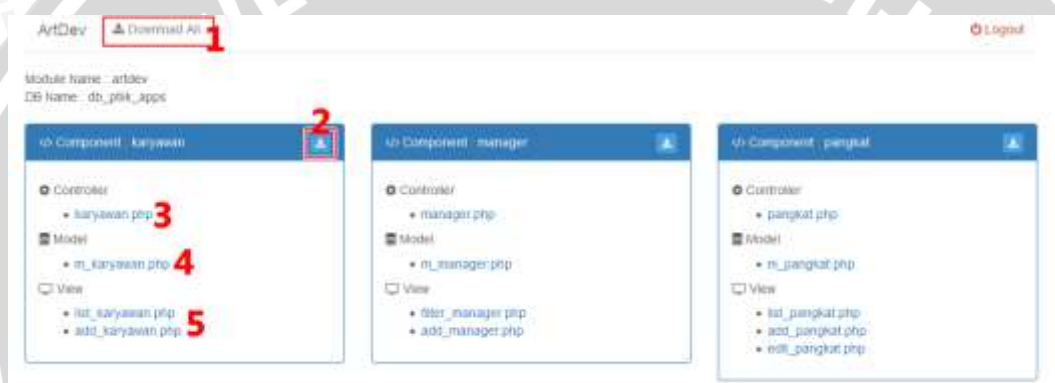
5 Save

Gambar 6.8 Implementasi Tatap Muka Input-Option Manager

6.3.4 Implementasi halaman unduh kode program

Implementasi halaman unduh kode program ditunjukkan oleh Gambar 6.9. Pada halaman tersebut pengguna dapat mengunduh file kode program secara menyeluruh, unduh per modul atau unduh tiap komponen. Berikut ini keterangan fungsi dari masing – masing tampilan pada Gambar 6.9:

1. tombol yang digunakan untuk mengunduh file semua kode program
2. tombol untuk mengunduh kode program pada komponen yang bersangkutan
3. file kode program *controller*, jika diklik maka file yang bersangkutan akan diunduh
4. file kode program *model*, jika diklik maka file yang bersangkutan akan diunduh
5. file kode program *view*, jika diklik maka file yang bersangkutan akan diunduh



Gambar 6.9 Implementasi Tatap Muka Halaman Unduh Kode Program

BAB 7 PENGUJIAN DAN ANALISIS

Bab ini membahas pengujian sistem dari *program generator* yang dibuat. Pengujian dilakukan dengan metode *black box* dan *white box*. Pengujian dilakukan untuk memastikan bahwa sistem *program generator* dapat berjalan sesuai dengan harapan dan dapat digunakan untuk memenuhi spesifikasi persyaratan.

Pengujian dilakukan dengan dua tahap, yaitu 1) *unit testing* dan 2) *release testing*. *Unit testing* dilakukan pada waktu proses pengembangan aplikasi, *unit testing* dilakukan dengan menguji bahwa semua jalur/cabang (*branch*) pada kode program dieksekusi minimal satu kali. *Release testing* dilakukan setelah proses pengembangan aplikasi, pengujian tersebut dilakukan untuk memastikan bahwa sistem dapat berjalan dengan benar sesuai dengan harapan.

7.1 Unit Testing

Unit testing dilakukan pada saat proses pengembangan sistem. Pengujian tersebut dilakukan pada setiap unit (kelas) kode program dalam sistem. Pada sub bab ini *unit testing* dilakukan pada unit kode program penyusun *domain knowledge*, yaitu: 1) kode program *controllerKnowledge*, 2) *modelKnowledge*, 3) *viewAddKnowledge*, 4) *viewDetailKnowledge*, 5) *viewEditKnowledge*, 6) *viewFilterKnowledge* dan 7) *viewListKnowledge*.

Unit testing dilakukan dengan menganalisis setiap cabang dari kode program yang bersangkutan (*basis path method*). Pengujian tersebut digunakan untuk memastikan bahwa setiap cabang kode program dieksekusi minimal satu kali dan setiap cabang dapat menjalankan fiturnya dengan baik.

Metode *basis path* menggunakan rumus untuk melakukan validasi dari diagram alir yang dibuat. Adapun rumus tersebut adalah sebagai berikut:

$$V(G) = R$$

$$V(G) = P + 1$$

$$V(G) = E - N + 2$$

R: jumlah *region* dalam suatu diagram alir, *region* ditandai dengan tanda bulat pada diagram alir yang dibuat

P: jumlah *primary node*, yaitu *node* (titik) yang memiliki lebih *edge*, *primary node* ditandai dengan *prefix* (awalan) 'P:' pada nama *node*.

E: jumlah *edge*, yaitu garis penghubung antar titik

N: jumlah *node* (titik)

Suatu diagram alir *basis path* dinyatakan valid jika nilai dari rumus $V(G)$ bernilai sama. Nilai dari $V(G)$ merupakan jumlah dari *independent path* dari diagram alir tersebut. *Independent path* adalah sebuah path yang memiliki minimal satu *node* yang tidak dimiliki *path* lain (unik). *Node* unik tersebut ditandai dengan warna

kuning pada laporan ini. *Independent path* dapat dipakai untuk membuat sebuah *test case* (kasus uji) pada *unit testing*.

7.1.1 Unit testing controller knowledge

Unit testing pada *controler knowledge* dibagi menjadi tiga bagian, yaitu: 1) bagian alur utama, 2) bagian alur proses simpan data 3) dan bagian alur proses ubah data.

Tabel 7.1 menjabarkan kode program unit *controller knowledge* beserta nomor baris kode. Tabel 7.1 digunakan untuk memetakan nomor baris kode program pada Tabel 7.1 kedalam diagram alir Gambar 7.1.

Tabel 7.1 Kode Program Controller Knowledge

1	[[
2	class <?php echo \$this->var->moduleName ... ?> { ...
3	..generate construct class
4	
5	<?php
6	foreach (\$this->var->features as \$featureKey):
7	\$funcName = \$this-> ...
8	if(\$featureKey->name == listData...): ?>
9	..generate list data function
10	
11	<?php //end if show dat
12	elseif(\$featureKey->name == ...filterData):
13	\$tablesFilter = ...
14	\$columnFK = ...
15	\$tableCurr = ...
16	\$columnFilter = ...
17	?>
18	..generate filter data function
19	<?php
20	if(\$columnFilter){
21	foreach(\$columnFilter as \$key) {
22	?>
23	..generate init variabel for filter data
24	<?php
25	}
26	}
27	?>
28	..generate filter data function
29	
30	<?php //end if filter data
31	elseif(\$featureKey->name == addData):
32	\$tablesSave = ...
33	\$columnFK = ...
34	\$tableCurr = ...
35	?>
36	..generate addData function
37	
38	..generate saveData function
39	<?php

```

40     if($tableCurr):
41         $primaryColumn = "";
42         foreach($tableCurr->columns as $key){
43             if($key->PRIMARY_KEY == 'PRIMARY')
44                 $primaryColumn = $key->COLUMN_NAME;
45
46
47             if($key->INPUT_TYPE == inputFile){
48                 ..init inputFile
49             } elseif($key->INPUT_TYPE == inpCheckbox) {
50                 ..init inpCheckbox
51             }
52             }?>
53     $data = array(
54     <?php
55     $no=1;
56     foreach ($tableCurr->columns as $key) {
57         if($key->IS_USED == "1") {
58             if(INPUT_TYPE == inpCheckbox || inputFile){
59                 ..generate init variable
60             } elseif (PRIMARY_KEY == 'PRIMARY') {
61                 ..generate variable for primary
62             } else {
63                 ..generate input
64             }
65         }
66     } //end of foreach
67     endif;
68     ?>
69
70     ..saveData function
71     <?php //tablechild manager
72     $tableChild = ... //get table child
73     if($tableChild){ //if table child exist
74         foreach ($tableChild as $tableChildTmp) {
75             ..generate read table child function
76             foreach($tableChildTmp->columns as $key){
77                 if($key->INPUT_TYPE == inputFile){
78                     ..init inputFile
79                 } elseif(INPUT_TYPE == inpCheckbox) {
80                     ..init inpCheckbox
81                 }
82
83                 $data = array( //init variabel child
84                 $no=1;
85                 foreach ($tableChildTmp->columns as $key) {
86                     if($key->IS_USED == "1") {
87                         if(INPUT_TYPE == inpCheckbox || inputFile){
88                             ..generate input checkbox/file
89                         } elseif(REFERENCED_TABLE == tableName){
90                             ..generate FK variabel
91                         } else { //if general column type
92                             ..geneate general input

```

```

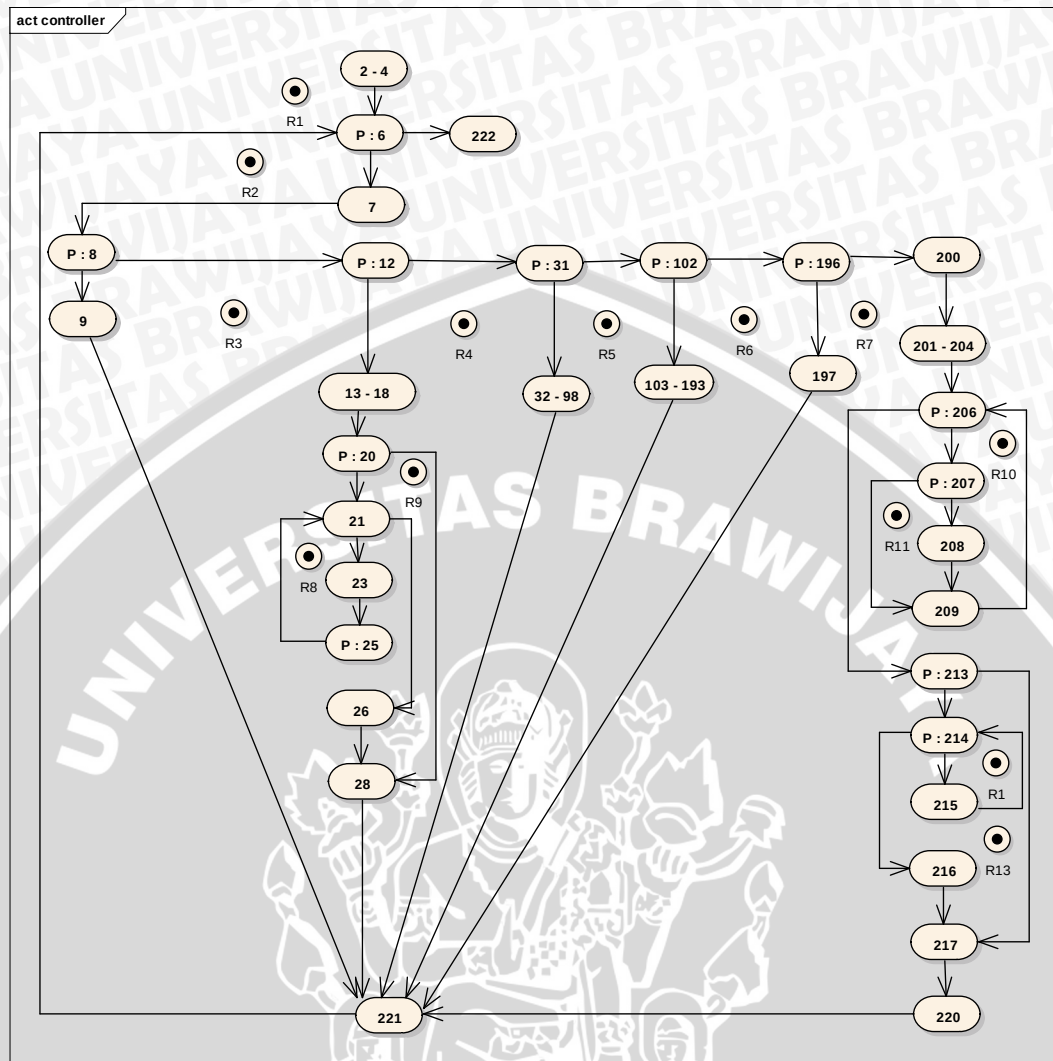
93         }
94     } //end if
95 } //end foreach tableChild
96 ..save tableChild function
97 } //end foreach tableChild
98 } //end if tableChild
99 ?>
100
101 <?php //end if addData
102 elseif($featureKey->name == editData):
103     $tablesEdit = ...
104     $columnView = ...
105     $columnFK = ...
106     $tableCurr = ...
107     $tableChild = ...
108 ?>
109 ..generate edit data function
110 <?php
111     foreach($columnFK as $key):
112         if(REFERENCED_TABLE != tableName):
113             ..generate data for dropdown view
114         endif;
115     endforeach;
116
117     if($tableChild): //data for detail child
118         foreach ($tableChild as $tableChildTmp):
119             ..generate data for tableChild
120         endforeach;
121     endif;
122 ?>
123 ..generate edit data view
124 ..generate doEdit data function
125
126 <?php
127 $columnID = ...
128 $primaryColumn = "";
129 foreach($tableCurr->columns as $key){
130     if($key->PRIMARY_KEY == 'PRIMARY')
131         $primaryColumn = $key->COLUMN_NAME;
132
133     if(INPUT_TYPE == inpCheckbox || inpFile){
134         ..generate init variable
135     } elseif (PRIMARY_KEY == 'PRIMARY') {
136         ..generate variable for primary
137     } else {
138         ..generate input
139     }
140 }
141 }
142 ..generate editData function
143 $data = array(
144 <?php
145 $no=1;

```

```

146     foreach ($tableCurr->columns as $key) {
147         if(IS_USED == "1" && PRIMARY_KEY != 'PRIMARY') {
148             if(INPUT_TYPE == inpCheckbox || inpFile){
149                 ..generate variabel inpCheckbox/File
150             }else {
151                 ..generate general variabel
152             };
153         }
154     }
155     ..generate edit where function
156     ..generate editData function
157
158     <?php //tablechild manager
159     if($tableChild){
160         ..set primary column
161         foreach ($tableChild as $tableChildTmp) {
162             ..read all child table
163             foreach($tableChildTmp->columns as $key){
164                 if($key->INPUT_TYPE == inpFile){
165                     ..set variabel for inpFile
166                 } elseif($key->INPUT_TYPE == inpCheckbox)
167                     ..set variabel for inpCheckbox
168             }
169         }
170     }
171 }
172 ?>
173
174 $data = array(
175     <?php //init variable data child untuk disimpan
176     $no=1;
177     $primaryChild = ""; //primary key for child
178     foreach ($tableChildTmp->columns as $key) {
179         if($key->PRIMARY_KEY == "PRIMARY")
180             $primaryChild = $key->COLUMN_NAME;
181
182         if(IS_USED == "1" && PK != 'PRIMARY') {
183             if(INPUT_TYPE == inpCheckbox || inpFile){
184                 ..generate variabel checkbox/file
185             } elseif(REFERENCED == tableName){
186                 ..generate variabel FK
187             } else { //if general column type
188                 ..generate general variabel
189             }
190         }
191     }
192     ..generate editData function
193     ..generate redirect function
194
195     <?php //end if editData
196     elseif($featureKey->name == deleteData): ?>
197         ..generate deleteData function
198 
```

```
199 <?php //end if delete data
200 elseif($featureKey->name == detailData):
201     $tablesDetail = ...
202     $columnFK = ...
203     $tableCurr = ...
204     $tableChild = ...
205
206     foreach($columnFK as $key):
207         if(REFERENCED_TABLE_NAME != tableName):
208             ..set variabel for column FK
209
210         endif;
211     endforeach;
212
213     if($tableChild): //data for detail child
214         foreach ($tableChild as $tableChildTmp):
215             ..set variabel for tableChild
216         endforeach;
217     endif;
218 ?>
219
220     ..generate detailData view
221 <?php endif; //end if detail data ?>
222 <?php endforeach; //end of main foreach ?>
```



Gambar 7.1 Diagram Alir *Controller Knowledge*

Berdasarkan diagram alir pada Gambar 7.1, maka fungsi untuk melakukan validasi *unit testing* adalah sebagai berikut:

$$V(G) = P + 1 = 12 + 1 = 13$$

$$V(G) = R = 13$$

$$V(G) = E - N + 2 = 44 - 33 + 2 = 13$$

Berikut ini *independent path* yang dibentuk dari *unit testing* pada *controllerKnowledge*:

1. 2 - 4, 6, 222
2. 2 - 4, 6, 7, 8, 9, 221, 6, 222
3. 2 - 4, 6, 7, 8, 12, 13 - 18, 20, 28, 221, 6, 222
4. 2 - 4, 6, 7, 8, 12, 13 - 18, 20, 21, 26, 28, 221, 6, 222
5. 2 - 4, 6, 7, 8, 12, 13 - 18, 20, 21, 23, 25, 26, 28, 221, 6, 222

6. 2 – 4, 6, 7, 8, 12, 31, 32 – 98, 221, 6, 222
7. 2 – 4, 6, 7, 8, 12, 31, 102, 103 – 193, 221, 6, 222
8. 2 – 4, 6, 7, 8, 12, 31, 102, 196, 197, 221, 6, 222
9. 2 – 4, 6, 7, 8, 12, 31, 102, 196, 200, 201 – 204, 206, 213, 217, 220, 221, 6, 222
10. 2 – 4, 6, 7, 8, 12, 31, 102, 196, 200, 201 – 204, 206, 207, 209, 206, 213, 217, 220, 221, 6, 222
11. 2 – 4, 6, 7, 8, 12, 31, 102, 196, 200, 201 – 204, 206, 207, 208, 209, 206, 213, 217, 220, 221, 6, 222
12. 2 – 4, 6, 7, 8, 12, 31, 102, 196, 200, 201 – 204, 206, 213, 214, 216, 217, 220, 221, 6, 222
13. 2 – 4, 6, 7, 8, 12, 31, 102, 196, 200, 201 – 204, 206, 213, 214, 215, 214, 216, 217, 220, 221, 6, 222

Independent path yang dibentuk dari *controllerKnowledge* akan dipetakan kedalam kasus uji (*test case*) yang ditunjukkan oleh Tabel 7.2.

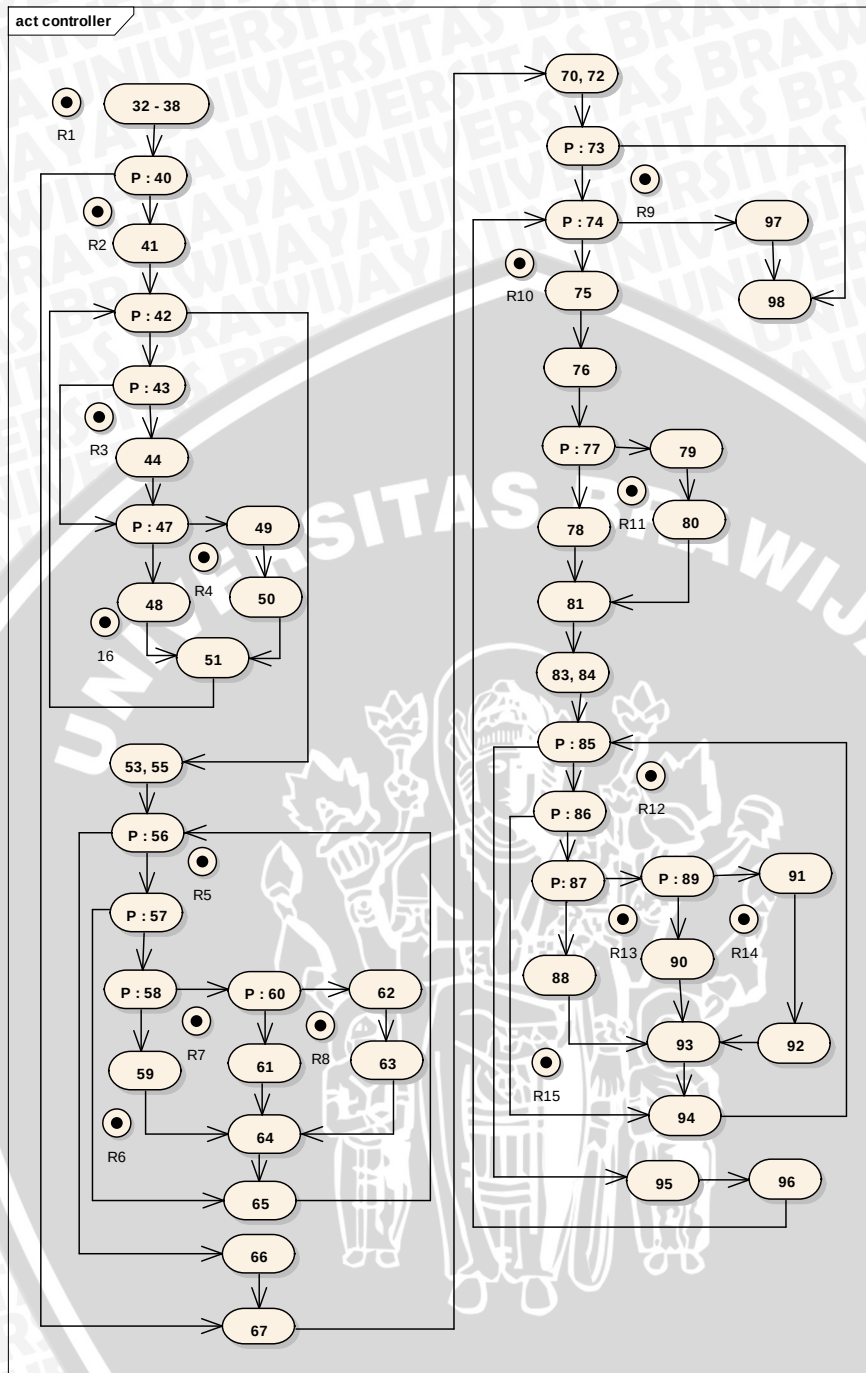
Tabel 7.2 Tabel Test Case Controller Knowledge

No	Pengujian	Validasi	
		Valid	Tidak
1	Menguji jika tidak data fitur kosong, maka komponen tidak membangkitkan kode fitur tersebut	v	
2	Komponen akan membangkitkan fungsi melihat daftar data tabel jika ada fitur <i>listData</i>	v	
3	Komponen akan membangkitkan fungsi sortir data jika ada fitur <i>filterData</i>	v	
4	Komponen tidak akan membangkitkan data kolom sortir jika isi variabel <i>columnFilter</i> kosong	v	
5	Komponen akan membangkitkan data kolom sortir sesuai dengan isi variabel <i>columnFilter</i>	v	
6	Komponen akan membangkitkan fungsi tambah data jika ada fitur <i>addData</i>	v	
7	Komponen akan membangkitkan fungsi ubah data jika ada fitur <i>editData</i>	v	
8	Komponen akan membangkitkan fungsi hapus data jika ada fitur <i>deleteData</i>	v	
9	Komponen akan membangkitkan fungsi detail data jika ada fitur <i>detailData</i>	v	

10	Komponen akan membaca variabel <i>columnFK</i> , jika nama tabel tidak cocok maka komponen tidak akan membangkitkan variabel tabel yang berhubungan	V	
11	Komponen akan membaca variabel <i>columnFK</i> , jika nama tabel cocok maka komponen akan membangkitkan variabel tabel yang berhubungan	V	
12	Jika variabel <i>tableChild</i> ada, maka komponen akan membaca variabel <i>tableChild</i> . Jika variabel <i>tableChild</i> kosong maka komponen tidak akan membangkitkan variabel dari tabel yang berhubungan	V	
13	Jika variabel <i>tableChild</i> tidak kosong, maka komponen akan membangkitkan variabel dari tabel yang berhubungan	V	

Diagram alir pada *controllerKnowledge* bagian tambah data ditunjukkan oleh Gambar 7.2. Diagram alir tersebut disusun berdasarkan kode program *controllerKnowledge* pada Tabel 7.1.





Gambar 7.2 Diagram Alir Controller Knowledge (Proses Tambah Data)

Berdasarkan diagram alir pada Gambar 7.2, maka fungsi untuk melakukan validasi *unit testing* adalah sebagai berikut:

$$V(G) = P + 1 = 15 + 1 = 16$$

$$V(G) = R = 16$$

$$V(G) = E - N + 2 = 63 - 49 + 2 = 16$$

Independent path yang dibentuk dari diagram alir pada Gambar 7.2 adalah sebagai berikut:

1. 32 – 38, 40, 67, 70, 72, 73, 98
2. 32 – 38, 40, 41, 42, 53, 55, 56, 66, 67, 70, 72, 73, 98
3. 32 – 38, 40, 41, 42, 43, 47, 48, 51, 42, 53, 55, 56, 66, 67, 70, 72, 73, 98
4. 32 – 38, 40, 41, 42, 43, 44, 47, 48, 51, 42, 53, 55, 56, 66, 67, 70, 72, 73, 98
5. 32 – 38, 40, 41, 42, 43, 44, 47, 49, 50, 51, 42, 53, 55, 56, 66, 67, 70, 72, 73, 98
6. 32 – 38, 40, 41, 42, 53, 55, 56, 57, 65, 56, 66, 67, 70, 72, 73, 98
7. 32 – 38, 40, 41, 42, 53, 55, 56, 57, 58, 59, 64, 65, 56, 66, 67, 70, 72, 73, 98
8. 32 – 38, 40, 41, 42, 53, 55, 56, 57, 58, 60, 61, 65, 56, 66, 67, 70, 72, 73, 98
9. 32 – 38, 40, 41, 42, 53, 55, 56, 57, 58, 60, 62, 63, 64, 65, 56, 66, 67, 70, 72, 73, 98
10. 32 – 38, 40, 67, 70, 72, 73, 74, 97, 98
11. 32 – 38, 40, 67, 70, 72, 73, 74, 75, 76, 77, 78, 81, 83, 84, 85, 95, 96, 74, 97, 98
12. 32 – 38, 40, 67, 70, 72, 73, 74, 75, 76, 77, 79, 80, 81, 83, 84, 85, 95, 96, 74, 97, 98
13. 32 – 38, 40, 67, 70, 72, 73, 74, 75, 76, 77, 78, 81, 83, 84, 85, 86, 94, 85, 95, 96, 74, 97, 98
14. 32 – 38, 40, 67, 70, 72, 73, 74, 75, 76, 77, 78, 81, 83, 84, 85, 86, 87, 88, 93, 94, 85, 95, 96, 74, 97, 98
15. 32 – 38, 40, 67, 70, 72, 73, 74, 75, 76, 77, 78, 81, 83, 84, 85, 86, 87, 89, 90, 93, 94, 85, 95, 96, 74, 97, 98
16. 32 – 38, 40, 67, 70, 72, 73, 74, 75, 76, 77, 78, 81, 83, 84, 85, 86, 87, 89, 91, 92, 94, 85, 95, 96, 74, 97, 98

Indepent path yang dibentuk dari diagram alir pada Gambar 7.2 dipetakan menjadi *test case* untuk menguji validasi tiap alur/cabang. Tabel 7.3 menunjukkan *test case* tersebut.

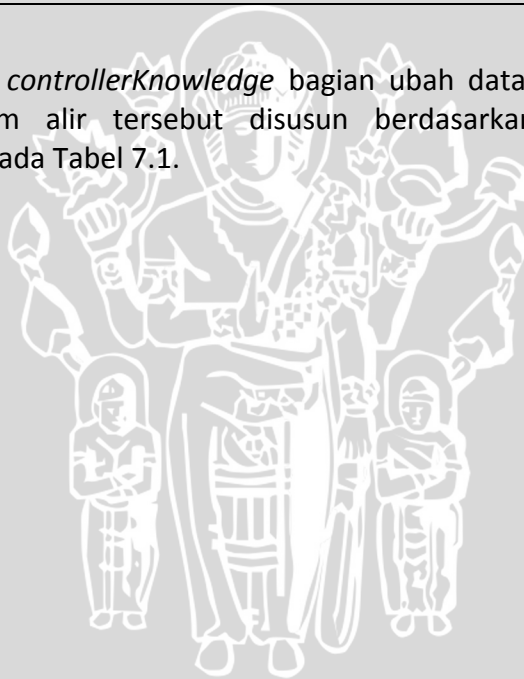
Tabel 7.3 Tabel Test Case Fungsi Tambah Data pada Controller Knowledge

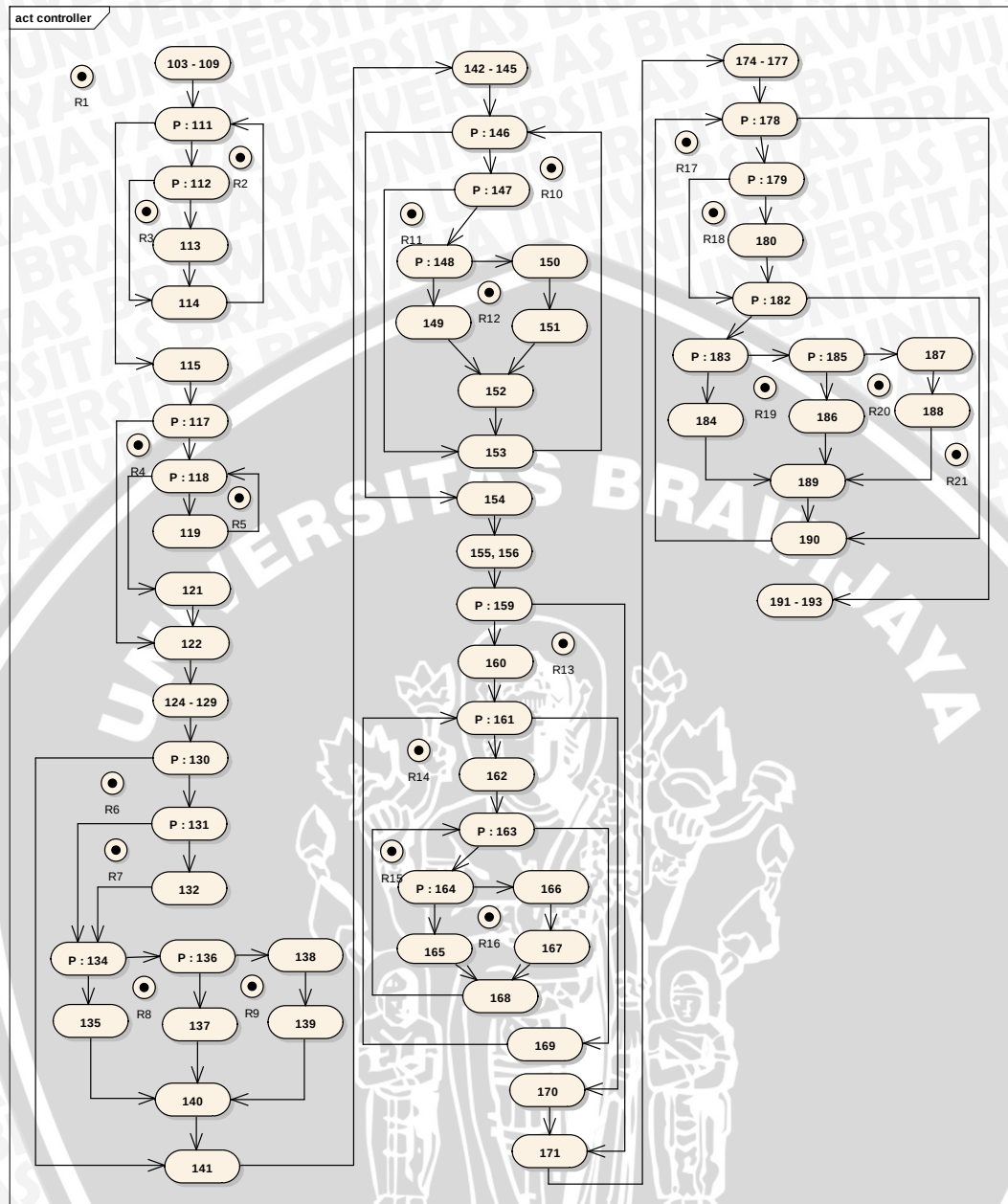
No	Pengujian	Validasi	
		Valid	Tidak
1	Komponen akan membangkitkan fungsi tambah data jika data tabel utama tidak kosong	V	
2	Komponen tidak akan membangkitkan data variabel jika tidak variabel <i>tableCurr->columns</i> kosong	V	

3	Komponen membaca variabel <i>tableCurr->column</i> . Komponen akan membangkitkan data variabel jika ada jenis masukan <i>file</i>	V	
4	Komponen membaca variabel <i>tableCurr->column</i> . Komponen akan membangkitkan variabel <i>primaryColumn</i> jika kolom yang bersangkutan adalah kolom utama	V	
5	Komponen membaca variabel <i>tableCurr->column</i> . Komponen akan membangkitkan data variabel jika ada jenis masukan <i>checkbox</i>	V	
6	Komponen membaca variabel <i>tableCurr->column</i> . Jika variabel kolom <i>IS_USED != 1</i> , maka komponen tidak akan membangkitkan variabel pada kolom tersebut	V	
7	Komponen membaca variabel <i>tableCurr->column</i> . Komponen akan membangkitkan variabel untuk jenis masukan <i>file</i> dan <i>checkbox</i> , jika variabel <i>INPUT_TYPE</i> sama dengan <i>inpFile</i> atau <i>inpCheckbox</i>	V	
8	Komponen membaca variabel <i>tableCurr->column</i> . Komponen akan mengkitkan variabel untuk kolom utama pada tabel yang bersangkutan, jika variabel <i>PRIMARY_KEY</i> sama dengan <i>PRIMARY</i>	V	
9	Komponen membaca variabel <i>tableCurr->column</i> . Komponen akan membangkitkan masukan sesuai dengan variabel <i>COLUMN_NAME</i> , jika tidak masuk kedalam kriteria nomor 7 dan 8	V	
10	Komponen tidak akan membangkitkan kode simpan tabel cabang jika variabel <i>tableChild</i> kosong	V	
11	Komponen akan membaca variabel <i>tableChild</i> . Komponen akan membangkitkan kode program untuk membaca variabel <i>tableChild</i> . Komponen akan membangkitkan kode program inisialisasi variabel tipe <i>file</i> jika nilai variabel <i>INPUT_FILE</i> sama dengan <i>inpFile</i>	V	
12	Komponen akan membaca variabel <i>tableChild</i> . Komponen akan membangkitkan kode program untuk membaca variabel <i>tableChild</i> . Komponen akan membangkitkan kode program inisialisasi variabel tipe <i>checkbox</i> jika nilai variabel <i>INPUT_FILE</i> sama dengan <i>inpCheckbox</i>	V	
13	Komponen akan membaca variabel <i>tableChildTmp->columns</i> . Jika variabel <i>IS_USED != 1</i> maka komponen tidak akan membangkitkan kode program inisialisasi pada kolom tersebut	V	

14	Komponen akan membaca variabel <i>tableChildTmp->columns</i> . Jika variabel <i>IS_USED</i> sama dengan 1 dan variabel <i>INPUT_TYPE</i> sama dengan <i>inpCheckbox</i> atau <i>inpFile</i> , maka komponen akan membangkitkan kode program inisialisasi masukan <i>checkbox</i> atau <i>file</i>	V	
15	Komponen akan membaca variabel <i>tableChildTmp->columns</i> . Jika variabel <i>IS_USED</i> sama dengan 1 dan variabel <i>REFERENCED_TABLE_NAME</i> sama dengan variabel <i>tableCurr->tableName</i> , maka komponen akan membangkitkan kode program inisialisasi kolom utama	V	
16	Komponen akan membaca variabel <i>tableChildTmp->columns</i> . Jika variabel <i>IS_USED</i> sama dengan 1, maka komponen akan membangkitkan kode program inisialisasi masukan pada kolom tersebut berdasarkan variabel <i>COLUMN_NAME</i>	V	

Diagram alir pada *controllerKnowledge* bagian ubah data ditunjukkan oleh Gambar 7.3. Diagram alir tersebut disusun berdasarkan kode program *controllerKnowledge* pada Tabel 7.1.





Gambar 7.3 Diagram Alir Controller Knowledge (Proses Ubah Data)

Berdasarkan diagram alir pada Gambar 7.3, maka fungsi untuk melakukan validasi *unit testing* adalah sebagai berikut:

$$V(G) = P + 1 = 20 + 1 = 21$$

$$V(G) = R = 21$$

$$V(G) = E - N + 2 = 80 - 61 + 2 = 21$$

Independent path yang dibentuk dari diagram alir pada Gambar 7.3 adalah sebagai berikut:

1. 103 – 109, 111, 115, 117, 122, 124 – 129, 130, 141, 142 – 145, 146, 154, 155, 156, 159, 171, 174 – 177, 178, 191 – 193

2. 103 – 109, 111, 112, 114, 111 115, 117, 122, 124 – 129, 130, 141, 142 – 145, 146, 154, 155, 156, 159, 171, 174 – 177, 178, 191 – 193
3. 103 – 109, 111, 112, 113, 114, 111 115, 117, 122, 124 – 129, 130, 141, 142 – 145, 146, 154, 155, 156, 159, 171, 174 – 177, 178, 191 – 193
4. 103 – 109, 111, 115, 117, 118, 121, 122, 124 – 129, 130, 141, 142 – 145, 146, 154, 155, 156, 159, 171, 174 – 177, 178, 191 – 193
5. 103 – 109, 111, 115, 117, 118, 119, 118, 121, 122, 124 – 129, 130, 141, 142 – 145, 146, 154, 155, 156, 159, 171, 174 – 177, 178, 191 – 193
6. 103 – 109, 111, 115, 117, 122, 124 – 129, 130, 131, 134, 135, 140, 141, 142 – 145, 146, 154, 155, 156, 159, 171, 174 – 177, 178, 191 – 193
7. 103 – 109, 111, 115, 117, 122, 124 – 129, 130, 131, 132, 134, 135, 140, 141, 142 – 145, 146, 154, 155, 156, 159, 171, 174 – 177, 178, 191 – 193
8. 103 – 109, 111, 115, 117, 122, 124 – 129, 130, 131, 132, 134, 136, 137, 140, 141, 142 – 145, 146, 154, 155, 156, 159, 171, 174 – 177, 178, 191 – 193
9. 103 – 109, 111, 115, 117, 122, 124 – 129, 130, 131, 132, 134, 136, 138, 139, 140, 141, 142 – 145, 146, 154, 155, 156, 159, 171, 174 – 177, 178, 191 – 193
10. 103 – 109, 111, 115, 117, 122, 124 – 129, 130, 141, 142 – 145, 146, 147, 153, 146, 154, 155, 156, 159, 171, 174 – 177, 178, 191 – 193
11. 103 – 109, 111, 115, 117, 122, 124 – 129, 130, 141, 142 – 145, 146, 147, 148, 149, 152, 153, 146, 154, 155, 156, 159, 171, 174 – 177, 178, 191 – 193
12. 103 – 109, 111, 115, 117, 122, 124 – 129, 130, 141, 142 – 145, 146, 147, 148, 149, 150, 151, 152, 153, 146, 154, 155, 156, 159, 171, 174 – 177, 178, 191 – 193
13. 103 – 109, 111, 115, 117, 122, 124 – 129, 130, 141, 142 – 145, 146, 154, 155, 156, 159, 160, 161, 170, 171, 174 – 177, 178, 191 – 193
14. 103 – 109, 111, 115, 117, 122, 124 – 129, 130, 141, 142 – 145, 146, 154, 155, 156, 159, 160, 161, 162, 163, 169, 161, 170, 171, 174 – 177, 178, 191 – 193
15. 103 – 109, 111, 115, 117, 122, 124 – 129, 130, 141, 142 – 145, 146, 154, 155, 156, 159, 160, 161, 162, 163, 164, 165, 168, 163, 169, 161, 170, 171, 174 – 177, 178, 191 – 193
16. 103 – 109, 111, 115, 117, 122, 124 – 129, 130, 141, 142 – 145, 146, 154, 155, 156, 159, 160, 161, 162, 163, 164, 166, 167, 168, 163, 169, 161, 170, 171, 174 – 177, 178, 191 – 193
17. 103 – 109, 111, 115, 117, 122, 124 – 129, 130, 141, 142 – 145, 146, 154, 155, 156, 159, 171, 174 – 177, 178, 179, 182, 190, 178, 191 – 193
18. 103 – 109, 111, 115, 117, 122, 124 – 129, 130, 141, 142 – 145, 146, 154, 155, 156, 159, 171, 174 – 177, 178, 179, 180, 182, 190, 178, 191 – 193

19. 103 – 109, 111, 115, 117, 122, 124 – 129, 130, 141, 142 – 145, 146, 154, 155, 156, 159, 171, 174 – 177, 178, 179, 182, 183, 184, 189, 190, 178, 191 – 193
20. 103 – 109, 111, 115, 117, 122, 124 – 129, 130, 141, 142 – 145, 146, 154, 155, 156, 159, 171, 174 – 177, 178, 179, 182, 183, 185, 186, 189, 190, 178, 191 – 193
21. 103 – 109, 111, 115, 117, 122, 124 – 129, 130, 141, 142 – 145, 146, 154, 155, 156, 159, 171, 174 – 177, 178, 179, 182, 183, 185, 187, 188, 189, 190, 178, 191 – 193

Indepent path yang dibentuk dari diagram alir pada Gambar 7.3 dipetakan menjadi *test case* untuk menguji validasi tiap alur/cabang. Tabel 7.4 menunjukkan *test case* tersebut.

Tabel 7.4 Tabel Test Case Fungsi Ubah Data pada Controller Knowledge

No	Pengujian	Validasi	
		Valid	Tidak
1	Komponen akan membangkitkan fungsi ubah pada tabel utama	V	
2	Komponen membaca variabel <i>columnFK</i> , jika nama tabel pada kolom tersebut sama dengan nama tabel utama maka komponen tidak membangkitkan inisialisasi kode program kolom cabang	V	
3	Komponen membaca variabel <i>columnFK</i> , jika nama tabel pada kolom tersebut tidak sama dengan nama tabel utama maka komponen tidak membangkitkan inisialisasi kode program kolom cabang	V	
4	Jika ada variabel <i>tableChild</i> namun nilainya kosong, maka komponen tidak akan membangkitkan inisialisasi kode untuk tabel cabang	V	
5	Jika variabel <i>tableChild</i> tidak kosong, maka komponen akan membangkitkan inisialisasi kode untuk tabel cabang	V	
6	Jika variabel <i>tableCurr->columns</i> tidak kosong, maka komponen akan membaca variabel tersebut. Jika nilai variabel <i>INPUT_TYPE</i> sama dengan <i>inpCheckbox</i> atau <i>inpFile</i> maka komponen akan membangkitkan inisialisasi variabel tipe <i>file</i> dan <i>checkbox</i>	V	
7	Jika variabel <i>tableCurr->columns</i> tidak kosong, maka komponen akan membaca variabel tersebut. Jika variabel <i>PRIMARY_KEY</i> sama dengan <i>PRIMARY</i> maka komponen akan melakukan inisialisasi variabel <i>primaryColumn</i>	V	

8	Jika variabel <i>tableCurr->columns</i> tidak kosong, maka komponen akan membaca variabel tersebut. Jika nilai variabel <i>PRIMARY_KEY</i> sama dengan <i>PRIMARY</i> , maka komponen akan membangkitkan inisialisasi variabel kolom utama	V	
9	Jika variabel <i>tableCurr->columns</i> tidak kosong, maka komponen akan membaca variabel tersebut. Komponen akan membangkitkan inisialisasi variabel sesuai dengan variabel <i>COLUMN_NAME</i>	V	
10	Jika variabel <i>tableCurr->columns</i> tidak kosong, maka komponen akan membaca variabel <i>tableCurr->columns</i> . Jika variabel <i>IS_USED</i> pada kolom tersebut != 1 atau kolom tersebut adalah kolom utama, maka komponen tidak akan membangkitkan kode inisialisasi atribut array	V	
11	Jika variabel <i>tableCurr->columns</i> tidak kosong, maka komponen akan membaca variabel <i>tableCurr->columns</i> . Jika variabel <i>IS_USED</i> pada kolom tersebut samadengan 1 dan kolom tersebut bukan kolom utama dan variabel <i>INPUT_TYPE</i> sama dengan <i>inpCheckbox</i> atau <i>inpFile</i> maka komponen akan membangkitkan kode inisialisasi atribut array untuk masuk tipe <i>checkbox</i> atau <i>file</i>	V	
12	Jika variabel <i>tableCurr->columns</i> tidak kosong, maka komponen akan membaca variabel <i>tableCurr->columns</i> . Jika variabel <i>IS_USED</i> pada kolom tersebut samadengan 1 dan kolom tersebut bukan kolom utama maka komponen akan membangkitkan kode inisialisasi atribut array	V	
13	Jika variabel <i>tableChild</i> ada maka komponen akan menyetel variabel kolom utama dan membaca variabel <i>tableChild</i> . Jika variabel <i>tableChild</i> kosong maka komponen tidak membangkitkan kode program untuk mengelola tabel cabang	V	
14	Jika variabel <i>tableChild</i> tidak kosong, maka komponen akan membangkitkan kode untuk membaca elemen dari <i>tableChild</i> dan melakukan perulangan terhadap elemen variabel <i>tableChild</i> kedalam sebuah variabel <i>tableChildTmp</i> . Jika nilai <i>tableChildTmp</i> kosong maka komponen tidak akan membangkitkan variabel untuk tipe masukan <i>file</i> dan <i>checkbox</i>	V	
15	Jika nilai <i>tableChildTmp</i> tidak kosong maka komponen akan membaca elemen <i>tableChildTmp</i> . Jika variabel <i>INPUT_TYPE</i> sama dengan <i>inpFile</i> maka komponen akan membangkitkan variabel untuk tipe masukan <i>file</i>	V	

16	Jika nilai <i>tableChildTmp</i> tidak kosong maka komponen akan membaca elemen <i>tableChildTmp</i> . Jika variabel <i>INPUT_TYPE</i> sama dengan <i>inpCheckbox</i> maka komponen akan membangkitkan variabel untuk tipe masukan <i>checkbox</i>	V	
17	Jika variabel <i>tableChildTmp->columns</i> tidak kosong maka komponen akan membaca elemen dari variabel tersebut. Jika variabel <i>IS_USED</i> != 1 atau kolom merupakan kolom utama maka komponen tidak akan membangkitkan inisialisasi atribut array	V	
18	Jika variabel <i>tableChildTmp->columns</i> tidak kosong maka komponen akan membaca elemen dari variabel tersebut. Jika variabel <i>PRIMARY_KEY</i> sama dengan <i>PRIMARY</i> maka komponen akan menyetel variabel <i>primaryChild</i>	V	
19	Jika variabel <i>tableChildTmp->columns</i> tidak kosong maka komponen akan membaca elemen dari variabel tersebut. Jika variabel <i>IS_USED</i> == 1 dan kolom bukan merupakan kolom utama dan variabel <i>INPUT_TYPE</i> sama dengan <i>inpCheckbox</i> atau <i>inpFile</i> maka komponen akan membangkitkan inisialisasi atribut array untuk masukan tipe <i>checkbox</i> atau <i>file</i>	V	
20	Jika variabel <i>tableChildTmp->columns</i> tidak kosong maka komponen akan membaca elemen dari variabel tersebut. Jika variabel <i>IS_USED</i> == 1 dan kolom bukan merupakan kolom utama dan variabel <i>REFERENCED_TABLE_NAME</i> sama dengan variabel <i>tableCurr->tableName</i> maka komponen akan membangkitkan atribut array sama dengan variabel <i>primaryColumn</i>	V	
21	Jika variabel <i>tableChildTmp->columns</i> tidak kosong maka komponen akan membaca elemen dari variabel tersebut. Jika variabel <i>IS_USED</i> == 1 dan kolom bukan merupakan kolom utama maka komponen akan membangkitkan inisialisasi atribut array berdasarkan variabel <i>COLUMN_NAME</i>	V	

7.1.2 Unit testing model knowledge

Unit testing pada *model knowledge* diawali dengan menjabarkan kode program *model knowledge* kedalam Tabel 7.5. Tabel 7.5 digunakan untuk memetakan nomor baris kode program pada Tabel 7.5 kedalam diagram alir Gambar 7.4.

Tabel 7.5 Kode Program Model Knowledge

```

1  [[
2  class <?php echo 'model_m'.componentName ?> ... model {
3
4  ..generate construct class
5
6  <?php
7  foreach ($this->var->features as $featureKey):
8  $funcName = ...
9  if($featureKey->name == listData):
10     $tables = ...
11     ?>
12     ..generate getDataList function
13
14     //end if list data
15     elseif($featureKey->name == filterData):
16         $tablesFilter = ...
17         $columnFK = ...
18         $tableCurr = ...
19         $columnFilter = ...
20
21         foreach ($columnFilter as $key) :
22             if(push_table_array(...)){
23                 $dataPushTmp = ...
24                 array_push(arrayGetDetail, ...);
25             }
26         endforeach;
27     ?>
28     ..generate getFilter function
29     <?php //generate param variable
30     $no=0;
31     foreach($columnFilter as $key) {
32         if($no>0)
33             echo ",";
34         $no++;
35         ..generate param variable
36     }
37
38     ..generate sql function
39     <?php
40     if($columnFilter):
41         foreach ($columnFilter as $key) {
42             ..generate column filter
43         }
44     endif;
45     ?>
46     ..generate return data function
47
48     <?php //end filter data
49     elseif($featureKey->name == addData):
50         $tablesAdd = ...
51         $columnFK = ...
52         $primaryTable = ...

```

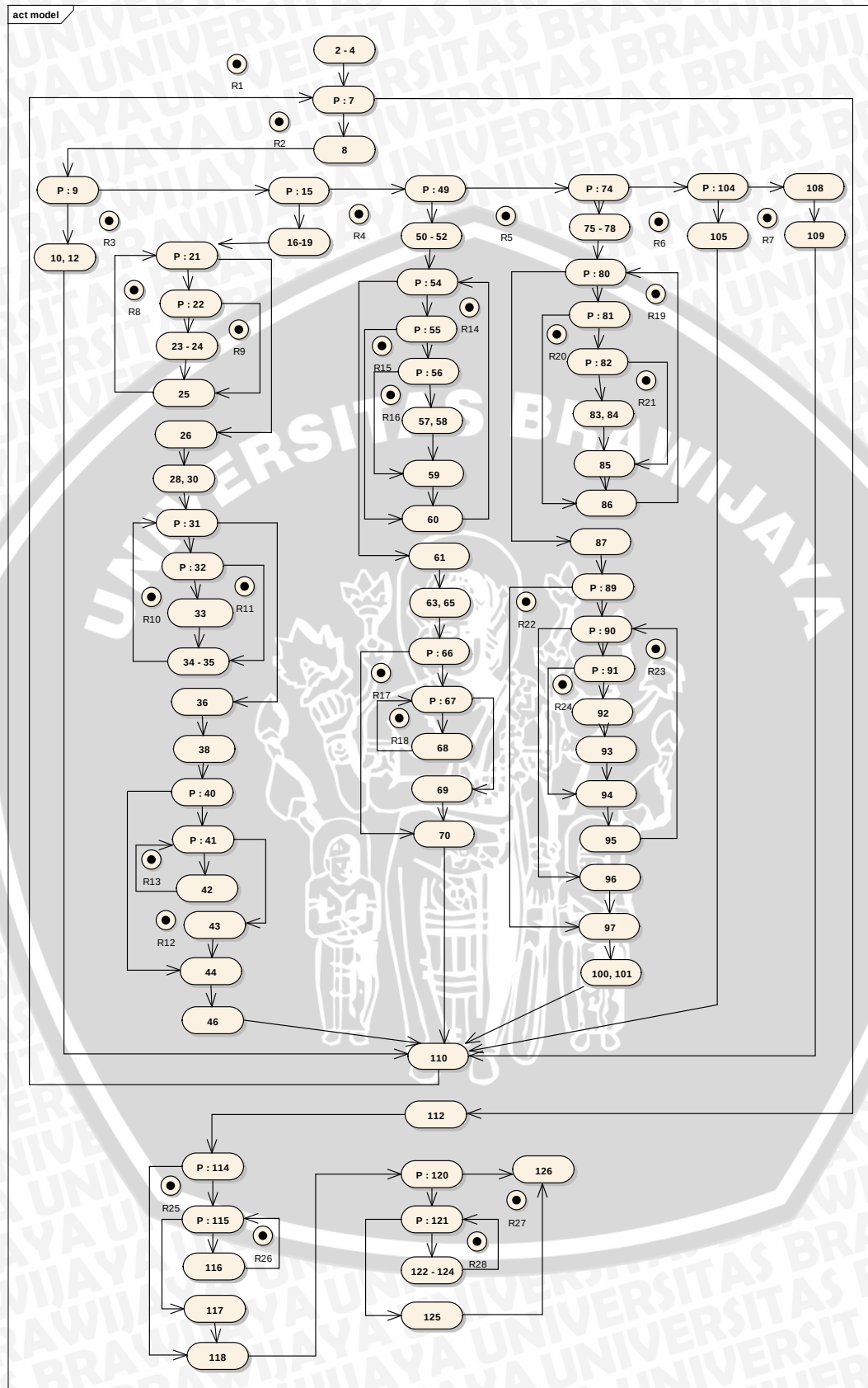
```

53
54         foreach($columnFK as $key):
55             if(REFERENCED_TABLENAME != primaryTable):
56                 if(push_table_array(...)){
57                     $dataPushTmp = ...
58                     array_push(...);
59                 }
60             endif;
61         endforeach;
62     ?>
63     ..generate addData function
64     <?php
65     $stableChild = ...
66     if($stableChild){ //jika ada table child
67         foreach ($stableChild as $stableChildTmp) {
68             ..generate addData child function
69         }
70     } //end of save tableChild
71     ?>
72
73     <?php //end of addData
74     if($featureKey->name == editData):
75         $tablesEdit = ...
76         $primaryTable = ...
77         $columnFK = ...
78         $stableChild = ...
79
80         foreach($columnFK as $key):
81             if(REFERENCED_TABLENAME != primaryTable):
82                 if(push_table_array(...)){
83                     $dataPushTmp = ...
84                     array_push(...);
85                 }
86             endif;
87         endforeach;
88
89         if($stableChild): //push data detail entity
90             foreach ($stableChild as $stableChildTmp):
91                 if(push_table_array(...)){
92                     $dataPushTmp = $stableChildTmp;
93                     array_push($arrayGetDetailList, ...);
94                 }
95             }
96             ..generate edit child data function
97         endforeach;
98     endif;
99     ?>
100     ..generate edit data function
101     ..generate get detail data for edit function
102
103     <?php //end if edit data
104     elseif($featureKey->name == deleteData): ?>
105         ..generate deleteData function

```

```
106
107     <?php //end of delete data
108     elseif($featureKey->name == detailData): ?>
109         ..generate getDetailData function
110     <?php endif; //end if detail data ?>
111
112     <?php
113     endforeach;
114     if($arrayGetDetail){ //create function for dorpdwn
115         foreach ($arrayGetDetail as $tableTmp) {
116             ..generate getDetailData function
117         }
118     }
119
120     if($arrayGetDetailList){
121         foreach ($arrayGetDetailList as $tableTmp) {
122             $arrayTmp = array($tableTmp);
123             $columnID = ...
124             ..generate getListData function
125         }
126     }
127     ?>
```





Gambar 7.4 Diagram Alir Model Knowledge

Berdasarkan diagram alir pada Gambar 7.4, maka fungsi untuk melakukan validasi *unit testing* adalah sebagai berikut:

$$V(G) = P + 1 = 27 + 1 = 28$$

$$V(G) = 28$$

$$V(G) = E - N + 2 = 101 - 75 + 2 = 28$$

Independent path yang dibentuk dari diagram alir pada Gambar 7.4 adalah sebagai berikut:

1. 2 – 4, 7, 112, 114, 118, 120, 126
2. 2 – 4, 7, 8, 9, 10, 12, 110, 7, 112, 114, 118, 120, 126
3. 2 – 4, 7, 8, 9, 15, 16, 19, 21, 26, 28, 30, 31, 36, 38, 40, 44, 46, 110, 7, 112, 114, 118, 120, 126
4. 2 – 4, 7, 8, 9, 15, 16, 19, 21, 22, 25, 21, 26, 28, 30, 31, 36, 38, 40, 44, 46, 110, 7, 112, 114, 118, 120, 126
5. 2 – 4, 7, 8, 9, 15, 16, 19, 21, 22, 23 – 24, 25, 21, 26, 28, 30, 31, 36, 38, 40, 44, 46, 110, 7, 112, 114, 118, 120, 126
6. 2 – 4, 7, 8, 9, 15, 16, 19, 21, 26, 28, 30, 31, 32, 34 – 35, 31, 36, 38, 40, 44, 46, 110, 7, 112, 114, 118, 120, 126
7. 2 – 4, 7, 8, 9, 15, 16, 19, 21, 26, 28, 30, 31, 32, 33, 34 – 35, 31, 36, 38, 40, 44, 46, 110, 7, 112, 114, 118, 120, 126
8. 2 – 4, 7, 8, 9, 15, 16, 19, 21, 22, 25, 21, 26, 28, 30, 31, 36, 38, 40, 41, 43, 44, 46, 110, 7, 112, 114, 118, 120, 126
9. 2 – 4, 7, 8, 9, 15, 16, 19, 21, 22, 25, 21, 26, 28, 30, 31, 36, 38, 40, 41, 42, 41, 43, 44, 46, 110, 7, 112, 114, 118, 120, 126
10. 2 – 4, 7, 8, 9, 15, 49, 50 – 52, 54, 61, 63, 65, 66, 70, 110, 7, 112, 114, 118, 120, 126
11. 2 – 4, 7, 8, 9, 15, 49, 50 – 52, 54, 55, 60, 54, 61, 63, 65, 66, 70, 110, 7, 112, 114, 118, 120, 126
12. 2 – 4, 7, 8, 9, 15, 49, 50 – 52, 54, 55, 56, 59, 60, 54, 61, 63, 65, 66, 70, 110, 7, 112, 114, 118, 120, 126
13. 2 – 4, 7, 8, 9, 15, 49, 50 – 52, 54, 55, 56, 57, 58, 59, 60, 54, 61, 63, 65, 66, 70, 110, 7, 112, 114, 118, 120, 126
14. 2 – 4, 7, 8, 9, 15, 49, 50 – 52, 54, 61, 63, 65, 66, 67, 69, 70, 110, 7, 112, 114, 118, 120, 126
15. 2 – 4, 7, 8, 9, 15, 49, 50 – 52, 54, 61, 63, 65, 66, 67, 68, 67, 69, 70, 110, 7, 112, 114, 118, 120, 126
16. 2 – 4, 7, 8, 9, 15, 49, 74, 78, 80, 87, 89, 97, 100, 101, 110, 7, 112, 114, 118, 120, 126

17. 2 – 4, 7, 8, 9, 15, 49, 74, 78, 80, 81, 86, 80, 87, 89, 97, 100, 101, 110, 7, 112, 114, 118, 120, 126
18. 2 – 4, 7, 8, 9, 15, 49, 74, 78, 80, 81, 82, 85, 86, 80, 87, 89, 97, 100, 101, 110, 7, 112, 114, 118, 120, 126
19. 2 – 4, 7, 8, 9, 15, 49, 74, 78, 80, 81, 82, 83, 84, 85, 86, 80, 87, 89, 97, 100, 101, 110, 7, 112, 114, 118, 120, 126
20. 2 – 4, 7, 8, 9, 15, 49, 74, 78, 80, 87, 89, 90, 96, 97, 100, 101, 110, 7, 112, 114, 118, 120, 126
21. 2 – 4, 7, 8, 9, 15, 49, 74, 78, 80, 87, 89, 90, 91, 94, 95, 90, 96, 97, 100, 101, 110, 7, 112, 114, 118, 120, 126
22. 2 – 4, 7, 8, 9, 15, 49, 74, 78, 80, 87, 89, 90, 91, 92, 93, 94, 95, 90, 96, 97, 100, 101, 110, 7, 112, 114, 118, 120, 126
23. 2 – 4, 7, 8, 9, 15, 49, 74, 104, 105, 110, 7, 112, 114, 118, 120, 126
24. 2 – 4, 7, 8, 9, 15, 49, 74, 104, 108, 109, 110, 7, 112, 114, 118, 120, 126
25. 2 – 4, 7, 112, 114, 115, 117, 118, 120, 126
26. 2 – 4, 7, 112, 114, 115, 116, 115, 117, 118, 120, 126
27. 2 – 4, 7, 112, 114, 118, 120, 121, 125, 126
28. 2 – 4, 7, 112, 114, 118, 120, 121, 122 – 124, 121, 125, 126

Indepent path yang dibentuk dari diagram alir pada Gambar 7.4 dipetakan menjadi *test case* untuk menguji validasi tiap alur/cabang. Tabel 7.6 menunjukkan *test case* tersebut.

Tabel 7.6 Table Test Case Model Knowledge

No	Pengujian	Validasi	
		Valid	Tidak
1	Komponen tidak akan membangkitkan fungsi kode program jika variabel <i>features</i> kosong	V	
2	Komponen akan membangkitkan fungsi lihat daftar data, jika ada fitur <i>listData</i>	V	
3	Komponen akan membangkitkan fungsi sortir data, jika ada fitur <i>filterData</i>	V	
4	Jika elemen variabel <i>columnFilter</i> tidak kosong, maka komponen akan membaca isinya. Jika variabel <i>REFERENCED_TABLE_NAME</i> ada pada array <i>arrayGetDetail</i> maka komponen tidak akan melakukan <i>push data</i> ke array tersebut	V	
5	Jika elemen variabel <i>columnFilter</i> tidak kosong, maka komponen akan membaca isinya. Jika variabel	V	

	<i>REFERENCED_TABLE_NAME</i> tidak ada pada array <i>arrayGetDetail</i> maka komponen akan melakukan <i>push data</i> ke array tersebut		
6	Jika elemen variabel <i>columnFilter</i> tidak kosong, maka sistem akan membaca isinya. Komponen akan membangkitkan kode variabel untuk kolom tersebut	V	
7	Jika elemen variabel <i>columnFilter</i> tidak kosong, maka sistem akan membaca isinya. Jika variabel <i>no</i> lebih dari 0, maka sistem akan mencetak karakter koma	V	
8	Jika ada variabel <i>columnFilter</i> , maka sistem akan membaca setiap elemen dari variabel tersebut. Jika elemen kosong, maka sistem tidak akan membangkitkan kode program untuk kolom tersebut	V	
9	Jika ada variabel <i>columnFilter</i> , maka sistem akan membaca setiap elemen dari variabel tersebut. Jika elemen tidak kosong, maka sistem akan membangkitkan kode program untuk kolom tersebut	V	
10	Komponen akan membangkitkan fungsi tambah data jika ada fitur <i>addData</i>	V	
11	Komponen akan membaca elemen variabel <i>columnFK</i> , jika variabel <i>REFERENCED_TABLE_NAME</i> sama dengan variabel <i>primaryTable</i> maka komponen tidak akan melakukan <i>push data</i> ke array <i>arrayGetDetail</i>	V	
12	Komponen akan membaca elemen variabel <i>columnFK</i> , jika variabel <i>REFERENCED_TABLE_NAME</i> != variabel <i>primaryTable</i> namun nilai variabel <i>REFERENCED_TABLE_NAME</i> sudah ada pada <i>arrayGetDetail</i> , maka komponen tidak akan melakukan <i>push data</i> ke array <i>arrayGetDetail</i>	V	
13	Komponen akan membaca elemen variabel <i>columnFK</i> , jika variabel <i>REFERENCED_TABLE_NAME</i> != variabel <i>primaryTable</i> dan nilai variabel <i>REFERENCED_TABLE_NAME</i> belum ada pada <i>arrayGetDetail</i> , maka komponen akan melakukan <i>push data</i> ke array <i>arrayGetDetail</i>	V	
14	Komponen akan membaca elemen variabel <i>tableChild</i> , jika elemen tersebut kosong maka komponen tidak akan membangkitkan kode program tambah data pada tabel cabang	V	
15	Komponen akan membaca elemen variabel <i>tableChild</i> , jika elemen tersebut tidak kosong maka komponen akan	V	

	membangkitkan kode program tambah data pada tabel cabang		
16	Komponen akan membangkitkan fungsi ubah data jika ada fitur <i>editData</i>	V	
17	Komponen akan membaca elemen variabel <i>columnFK</i> , jika variabel <i>REFERENCED_TABLE_NAME</i> sama dengan variabel <i>primaryTable</i> maka komponen tidak akan melakukan <i>push data</i> ke array <i>arrayGetDetail</i>	V	
18	Komponen akan membaca elemen variabel <i>columnFK</i> , jika variabel <i>REFERENCED_TABLE_NAME</i> sama dengan variabel <i>primaryTable</i> namun nilai variabel <i>REFERENCED_TABLE_NAME</i> sudah ada pada <i>arrayGetDetail</i> , maka komponen tidak akan melakukan <i>push data</i> ke array <i>arrayGetDetail</i>	V	
19	Komponen akan membaca elemen variabel <i>columnFK</i> , jika variabel <i>REFERENCED_TABLE_NAME</i> sama dengan variabel <i>primaryTable</i> dan nilai variabel <i>REFERENCED_TABLE_NAME</i> belum ada pada <i>arrayGetDetail</i> , maka komponen akan melakukan <i>push data</i> ke array <i>arrayGetDetail</i>	V	
20	Komponen akan membaca elemen variabel <i>columnFK</i> , jika variabel <i>REFERENCED_TABLE_NAME</i> sama dengan variabel <i>primaryTable</i> maka komponen tidak akan melakukan <i>push data</i> ke array <i>arrayGetDetailList</i>	V	
21	Komponen akan membaca elemen variabel <i>columnFK</i> , jika variabel <i>REFERENCED_TABLE_NAME</i> sama dengan variabel <i>primaryTable</i> namun nilai variabel <i>REFERENCED_TABLE_NAME</i> sudah ada pada <i>arrayGetDetailList</i> , maka komponen tidak akan melakukan <i>push data</i> ke array <i>arrayGetDetailList</i>	V	
22	Komponen akan membaca elemen variabel <i>columnFK</i> , jika variabel <i>REFERENCED_TABLE_NAME</i> sama dengan variabel <i>primaryTable</i> dan nilai variabel <i>REFERENCED_TABLE_NAME</i> belum ada pada <i>arrayGetDetailList</i> , maka komponen akan melakukan <i>push data</i> ke array <i>arrayGetDetailList</i>	V	
23	Komponen akan membangkitkan fungsi hapus data jika ada fitur <i>deleteData</i>	V	
24	Komponen akan membangkitkan fungsi detail data jika ada fitur <i>detailData</i>	V	

25	Komponen akan membaca elemen pada variabel <i>arrayGetDetail</i> , jika elemen kosong maka komponen tidak akan membangkitkan kode program fungsi <i>getDetailData</i>	V	
26	Komponen akan membaca elemen pada variabel <i>arrayGetDetail</i> , jika elemen tidak kosong maka komponen akan membangkitkan kode program fungsi <i>getDetailData</i>	V	
27	Komponen akan membaca elemen pada variabel <i>arrayGetDetailList</i> , jika elemen kosong maka komponen tidak akan membangkitkan kode program fungsi <i>getListData</i>	V	
28	Komponen akan membaca elemen pada variabel <i>arrayGetDetailList</i> , jika elemen tidak kosong maka komponen akan membangkitkan kode program fungsi <i>getListData</i>	V	

7.1.3 Unit testing view add knowledge

Unit testing pada *view add knowledge* diawali dengan menjabarkan kode program *view add knowledge* kedalam Tabel 7.7. Tabel 7.7 digunakan untuk memetakan nomor baris kode program pada Tabel 7.7 kedalam diagram alir Gambar 7.5.

Tabel 7.7 Kode Program View Add Knowledge

1	<code>[[\$this->head(); /]]</code>
2	
3	<code>..generate view header</code>
4	<code><?php</code>
5	<code>if(cekExistFuncName(...)): ?></code>
6	<code>..generate opsi goto list data view</code>
7	<code>endif;</code>
8	<code>?></code>
9	
10	<code><?php</code>
11	<code>\$tables = ...</code>
12	<code>\$columnView = ...</code>
13	<code>\$columnFK = ...</code>
14	<code>\$tableCurr = ...</code>
15	<code>\$tableChild = ...</code>
16	<code>?></code>
17	
18	<code>..generate form element</code>
19	<code><?php</code>
20	<code>foreach(\$columnView as \$key):</code>
21	<code>..generate column label</code>
22	<code>if(in_array(\$key, \$columnFK)):</code>
23	<code>\$columnDesc = ...</code>
24	<code>..generate select-option input for FK</code>

```

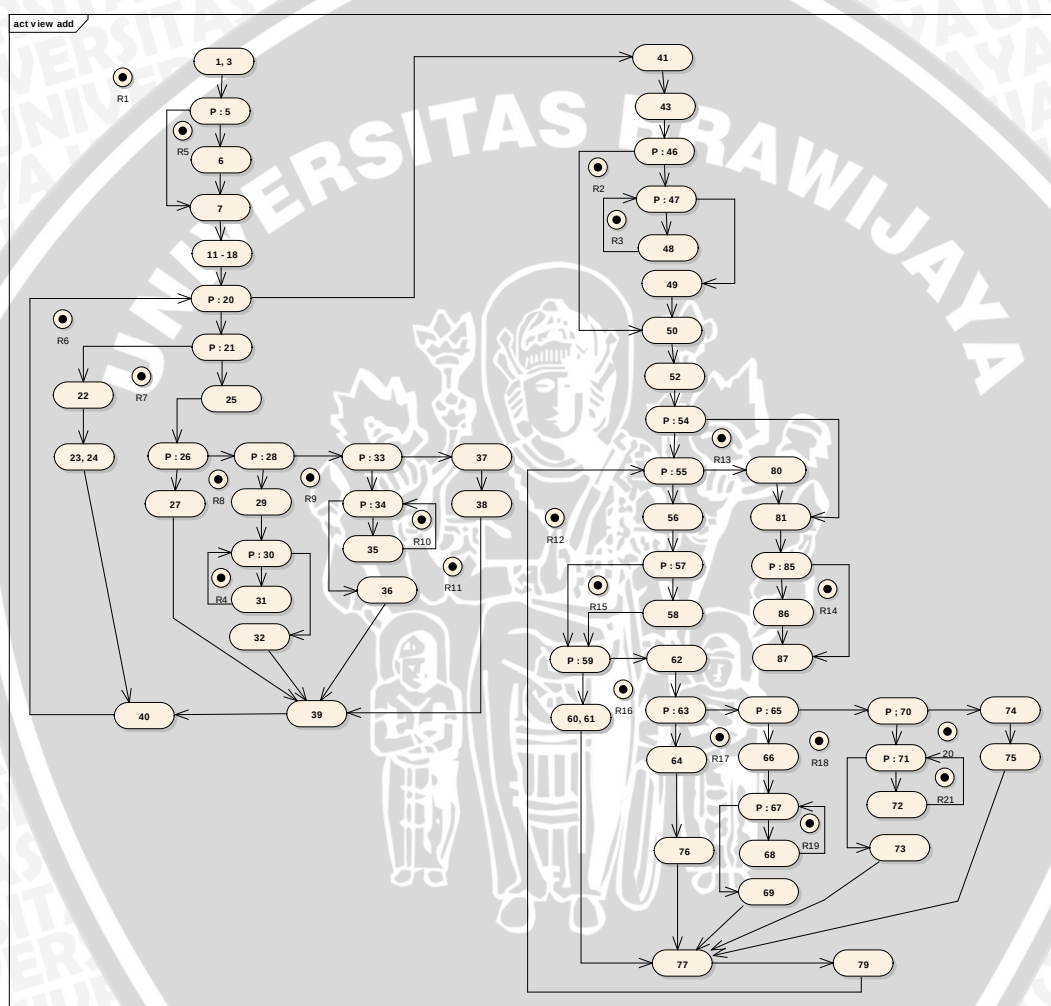
25         else:
26             if($key->INPUT_TYPE == $inpTextarea):
27                 ..generate input textarea
28             elseif($key->INPUT_TYPE == $inpSelect ):
29                 ..generate input select-option
30                 foreach(INPUT_OPTION as $input):
31                     ..generate option element
32                 endforeach;
33             elseif(... = $inpCheckbox || $inpRadio ):
34                 foreach(INPUT_OPTION as $input):
35                     ..generate input checkbox/radio
36                 endforeach;
37             else:
38                 ..generate general input
39             endif;
40         endforeach; ?>
41     ..generate button save
42
43 <?php
44 if($tableChild):
45     foreach($tableChild as $table): ?>
46         ..generate wrap child data
47     endforeach;
48 endif
49
50 ..generate end of div element
51
52 if($tableChild):
53     ..generate element id for add child data wrap
54     foreach ($table->columns as $key):
55         if(IS_USED == 1 && PRIMARYKEY != "PRIMARY"):
56             ..generate column title
57         if(in_array($key, $columnFK)):
58             $columnDesc = ...
59             ..generate input select-option for FK
60         else:
61             if($key->INPUT_TYPE == $inpTextarea):
62                 ..generate input textarea
63             elseif($key->INPUT_TYPE == $inpSelect ):
64                 ..generate input select-option
65                 foreach(INPUT_OPTION as $input):
66                     ..generate option element
67                 endforeach;
68             elseif(... = $inpCheckbox || $inpRadio ):
69                 foreach(INPUT_OPTION as $input):
70                     ..generate input checkbox/radio
71                 endforeach;
72             else:
73                 ..generate general input
74             endif;
75         endforeach;
76     endif;
77

```

```

78
79     ..generate button delete data
80     endforeach;
81 endif; ?>
82 [[ $this->foot(); /]]
83
84 <?php
85 if($stableChild):
86     ..generate javascript for DOM child data
87 endif; ?>

```



Gambar 7.5 Diagram Alir View Add Knowledge

Berdasarkan diagram alir pada Gambar 7.5, maka fungsi untuk melakukan validasi *unit testing* adalah sebagai berikut:

$$V(G) = P + 1 = 20 + 1 = 21$$

$$V(G) = R = 21$$

$$V(G) = E - N + 2 = 82 - 63 + 2 = 21$$

Independent path yang dibentuk dari diagram alir pada Gambar 7.5 adalah sebagai berikut:

1. 1, 3, 5, 7, 11 – 18, 20, 41, 43, 46, 50, 52, 54, 81, 85, 87
2. 1, 3, 5, 6, 7, 11 – 18, 20, 41, 43, 46, 50, 52, 54, 81, 85, 87
3. 1, 3, 5, 6, 7, 11 – 18, 20, 21, 22, 23, 24, 40, 20, 41, 43, 46, 50, 52, 54, 81, 85, 87
4. 1, 3, 5, 6, 7, 11 – 18, 20, 21, 25, 26, 27, 39, 40, 20, 41, 43, 46, 50, 52, 54, 81, 85, 87
5. 1, 3, 5, 6, 7, 11 – 18, 20, 21, 25, 26, 28, 29, 30, 32, 39, 40, 20, 41, 43, 46, 50, 52, 54, 81, 85, 87
6. 1, 3, 5, 6, 7, 11 – 18, 20, 21, 25, 26, 28, 29, 30, 31, 30, 32, 39, 40, 20, 41, 43, 46, 50, 52, 54, 81, 85, 87
7. 1, 3, 5, 6, 7, 11 – 18, 20, 21, 25, 26, 28, 33, 34, 36, 39, 40, 20, 41, 43, 46, 50, 52, 54, 81, 85, 87
8. 1, 3, 5, 6, 7, 11 – 18, 20, 21, 25, 26, 28, 33, 34, 35, 34, 36, 39, 40, 20, 41, 43, 46, 50, 52, 54, 81, 85, 87
9. 1, 3, 5, 6, 7, 11 – 18, 20, 21, 25, 26, 28, 33, 37, 38, 39, 40, 20, 41, 43, 46, 50, 52, 54, 81, 85, 87
10. 1, 3, 5, 7, 11 – 18, 20, 41, 43, 46, 47, 49, 50, 52, 54, 81, 85, 87
11. 1, 3, 5, 7, 11 – 18, 20, 41, 43, 46, 47, 48, 47, 49, 50, 52, 54, 81, 85, 87
12. 1, 3, 5, 7, 11 – 18, 20, 41, 43, 46, 50, 52, 54, 81, 85, 86, 87
13. 1, 3, 5, 7, 11 – 18, 20, 41, 43, 46, 50, 52, 54, 55, 80, 81, 85, 86, 87
14. 1, 3, 5, 7, 11 – 18, 20, 41, 43, 46, 50, 52, 54, 55, 56, 57, 59, 60, 61, 77, 79, 55, 80, 81, 85, 86, 87
15. 1, 3, 5, 7, 11 – 18, 20, 41, 43, 46, 50, 52, 54, 55, 56, 57, 58, 59, 60, 61, 77, 79, 55, 80, 81, 85, 86, 87
16. 1, 3, 5, 7, 11 – 18, 20, 41, 43, 46, 50, 52, 54, 55, 56, 57, 58, 59, 62, 63, 64, 76, 77, 79, 55, 80, 81, 85, 86, 87
17. 1, 3, 5, 7, 11 – 18, 20, 41, 43, 46, 50, 52, 54, 55, 56, 57, 58, 59, 62, 63, 65, 66, 67, 69, 77, 79, 55, 80, 81, 85, 86, 87
18. 1, 3, 5, 7, 11 – 18, 20, 41, 43, 46, 50, 52, 54, 55, 56, 57, 58, 59, 62, 63, 65, 66, 67, 68, 67, 69, 77, 79, 55, 80, 81, 85, 86, 87
19. 1, 3, 5, 7, 11 – 18, 20, 41, 43, 46, 50, 52, 54, 55, 56, 57, 58, 59, 62, 63, 65, 70, 71, 73, 77, 79, 55, 80, 81, 85, 86, 87
20. 1, 3, 5, 7, 11 – 18, 20, 41, 43, 46, 50, 52, 54, 55, 56, 57, 58, 59, 62, 63, 65, 70, 71, 72, 71, 73, 77, 79, 55, 80, 81, 85, 86, 87

21. 1, 3, 5, 7, 11 – 18, 20, 41, 43, 46, 50, 52, 54, 55, 56, 57, 58, 59, 62, 63, 65, 70, 74, 75, 77, 79, 55, 80, 81, 85, 86, 87

Indepent path yang dibentuk dari diagram alir pada Gambar 7.5 dipetakan menjadi *test case* untuk menguji validasi tiap alur/cabang. Tabel 7.8 menunjukkan *test case* tersebut.

Tabel 7.8 Table Test Case View Add Knowledge

No	Pengujian	Validasi	
		Valid	Tidak
1	Komponen akan membangkitkan fungsi tambah data, jika variabel <i>columnView</i> kosong maka sistem tidak akan membangkitkan kode program kolom tersebut	V	
2	Jika ada fitur <i>listData</i> maka komponen akan membangkitkan opsi ke halaman <i>listData</i>	V	
3	Komponen akan membaca elemen variabel <i>columnView</i> , jika kolom yang bersangkutan ada pada <i>array columnFK</i> maka komponen akan membangkitkan kode program untuk menghasilkan masukan <i>dropdown</i> untuk kolom yang bersangkutan	V	
4	Jika variabel <i>INPUT_TYPE</i> pada <i>columnView</i> sama dengan <i>inpTextArea</i> maka komponen akan menghasilkan kode program masukan tipe <i>textarea</i>	V	
5	Jika variabel <i>INPUT_TYPE</i> pada <i>columnView</i> sama dengan <i>inpSelect</i> maka komponen akan menghasilkan kode program masukan tipe <i>select-option</i> . Komponen akan membaca elemen <i>INPUT_OPTION</i> , jika elemen tersebut kosong maka komponen tidak akan menghasilkan elemen <i>option</i> pada masukan tersebut	V	
6	Jika variabel <i>INPUT_TYPE</i> pada <i>columnView</i> sama dengan <i>inpSelect</i> maka komponen akan menghasilkan kode program masukan tipe <i>select-option</i> . Komponen akan membaca elemen <i>INPUT_OPTION</i> , jika elemen tersebut tidak kosong maka komponen akan menghasilkan elemen <i>option</i> pada masukan tersebut	V	
7	Jika variabel <i>INPUT_TYPE</i> pada <i>columnView</i> sama dengan <i>inpRadio</i> atau <i>inpCheckbox</i> maka komponen akan membaca elemen <i>INPUT_OPTION</i> , jika elemen tersebut kosong maka komponen tidak akan menghasilkan elemen masukan <i>radioButton</i> atau <i>checkbox</i>	V	
8	Jika variabel <i>INPUT_TYPE</i> pada <i>columnView</i> sama dengan <i>inpRadio</i> atau <i>inpCheckbox</i> maka komponen akan	V	

	membaca elemen <i>INPUT_OPTION</i> , jika elemen tersebut kosong maka komponen tidak akan menghasilkan elemen masukan <i>radioButton</i> atau <i>checkbox</i>		
9	Komponen akan menghasilkan tipe masukan yang sesuai dengan variabel <i>INPUT_TYPE</i> pada <i>columnView</i> jika tipe masukan bukan <i>inpTextarea</i> , <i>inpSelect</i> , <i>inpCheckbox</i> dan <i>inpRadio</i>	V	
10	Jika variabel <i>tableChild</i> tidak kosong, maka komponen akan membaca elemen dari variabel <i>tableChild</i> tersebut. Jika elemen <i>tableChild</i> kosong maka komponen tidak akan menghasilkan kode HTML untuk <i>tableChild</i>	V	
11	Jika variabel <i>tableChild</i> tidak kosong, maka komponen akan membaca elemen dari variabel <i>tableChild</i> tersebut. Jika elemen <i>tableChild</i> tidak kosong maka komponen akan menghasilkan kode HTML untuk <i>tableChild</i>	V	
12	Jika elemen <i>tableChild</i> tidak kosong, maka komponen akan menghasilkan kode program <i>javascript</i> untuk mengelola HTML masukan tabel cabang	V	
13	Jika elemen variabel <i>table->columns</i> kosong maka komponen tidak akan menghasilkan kode program tambah data table cabang	V	
14	Komponen akan membaca elemen variabel <i>tableChild->columns</i> , jika kolom yang bersangkutan ada pada <i>array columnFK</i> maka komponen akan membangkitkan kode program untuk menghasilkan masukan <i>dropdown</i> untuk kolom yang bersangkutan	V	
15	Jika variabel <i>IS_USED</i> pada <i>tableChild</i> sama dengan 1 dan variabel <i>PRIMARY_KEY</i> pada <i>tableChild</i> sama dengan <i>PRIMARY</i> maka komponen akan menghasilkan kode HTML untuk judul kolom	V	
16	Jika variabel <i>INPUT_TYPE</i> pada <i>tableChild</i> sama dengan <i>inpTextArea</i> maka komponen akan menghasilkan kode program masukan tipe <i>textarea</i>	V	
17	Jika variabel <i>INPUT_TYPE</i> pada <i>tableChild</i> sama dengan <i>inpSelect</i> maka komponen akan menghasilkan kode program masukan tipe <i>select-option</i> . Komponen akan membaca elemen <i>INPUT_OPTION</i> , jika elemen tersebut kosong maka komponen tidak akan menghasilkan elemen <i>option</i> pada masukan tersebut	V	
18	Jika variabel <i>INPUT_TYPE</i> pada <i>tableChild</i> sama dengan <i>inpSelect</i> maka komponen akan menghasilkan kode	V	

	program masukan tipe <i>select-option</i> . Komponen akan membaca elemen <i>INPUT_OPTION</i> , jika elemen tersebut tidak kosong maka komponen akan menghasilkan elemen <i>option</i> pada masukan tersebut		
19	Jika variabel <i>INPUT_TYPE</i> pada <i>tableChild</i> sama dengan <i>inpRadio</i> atau <i>inpCheckbox</i> maka komponen akan membaca elemen <i>INPUT_OPTION</i> , jika elemen tersebut kosong maka komponen tidak akan menghasilkan elemen masukan <i>radioButton</i> atau <i>checkbox</i>	V	
20	Jika variabel <i>INPUT_TYPE</i> pada <i>tableChild</i> sama dengan <i>inpRadio</i> atau <i>inpCheckbox</i> maka komponen akan membaca elemen <i>INPUT_OPTION</i> , jika elemen tersebut kosong maka komponen tidak akan menghasilkan elemen masukan <i>radioButton</i> atau <i>checkbox</i>	V	
21	Komponen akan menghasilkan tipe masukan yang sesuai dengan variabel <i>INPUT_TYPE</i> pada <i>tableChild</i> jika tipe masukan bukan <i>inpTextarea</i> , <i>inpSelect</i> , <i>inpCheckbox</i> dan <i>inpRadio</i>	V	

7.2 Release Testing

Release testing dilakukan setelah proses pengembangan sistem selesai dilakukan. *Release testing* digunakan untuk memastikan bahwa sistem yang dibangun memenuhi persyaratan yang telah didefinisikan. *Test case* (kasus uji) pada pengujian ini dibentuk berdasarkan daftar persyaratan fungsional dan persyaratan non-fungsional berupa aspek: 1) *security* dengan menguji proses *authetication* dan *authorization* 2) *compability* dengan menguji kinerja sistem pada beberapa *web browser*.

Beberapa *test case* juga dirancang berdasarkan jenis masukan, yaitu: 1) masukan kosong, 2) masukan yang mengandung spasi, 3) masukan yang mengandung angka, dan 4) masukan yang mengandung simbol.

Release testing pada penelitian ini dibagi menjadi beberapa bagian, yaitu:

1. Pengujian fitur verifikasi akun
2. Pengujian fitur sambungan *database*
3. Pengujian fitur kelola komponen
4. Pengujian fitur tambah komponen
5. Pengujian fitur parameterisasi komponen
 - a. Parameterisasi *module name*
 - b. Parameterisasi *component model*
 - c. Parameterisasi *feature model*

- d. Parameterisasi *table model*
- e. Parameterisasi *column model*
- f. Parameterisasi *input-option model*
6. Pengujian fitur hapus komponen
7. Pengujian fitur pembangkitan kode program

7.2.1 Pengujian fitur verifikasi akun

Pengujian fitur verifikasi akun dijabarkan pada Tabel 7.8. Fitur yang diuji adalah fitur *login*, ubah *password*, *logout* dan cek hak akses.

Tabel 7.8 Pengujian *Black Box* Fitur Verifikasi Akun

No	Kondisi	Pengujian	Skema yang Diinginkan	Validasi
1	Masukan <i>username</i> dan <i>password</i>	 Fitur <i>login</i>	Pengguna bisa masuk ke halaman parameterisasi komponen	V
2	Masukan <i>username</i> kosong		Sistem akan memberikan peringatan untuk mengisi masukan <i>username</i>	
3	Masukan <i>password</i> kosong		Sistem akan memberikan peringatan untuk mengisi masukan <i>password</i>	V
4	Masukan <i>username</i> dan <i>password</i> menggunakan spasi dan simbol		Sistem akan memproses data tersebut dengan benar	V
5	<i>Username</i> dan <i>password</i> salah		Sistem akan mengarahkan pengguna ke halaman login lagi	V
6	Jika <i>password</i> lama tidak valid	Fitur ubah <i>password</i>	Sistem akan memberikan pesan bahwa <i>password</i> lama tidak valid	V
7	Jika <i>password</i> baru tidak sama dengan <i>password</i> yang dikonfirmasi		Sistem akan memberi pesan bahwa <i>password</i> baru tidak cocok	V

8	Jika <i>password</i> baru mengandung karakter spasi dan simbol		Sistem akan memproses data tersebut	V
9	-	Fitur <i>logout</i>	Sistem akan mengarahkan pengguna kembali ke halaman <i>login</i>	V
10	Jika memiliki hak akses	Cek hak akses	Sistem akan memperbolehkan pengguna untuk mengakses opsi yang bersangkutan	V
11	Jika tidak memiliki hak akses		Sistem akan melakukan fungsi <i>logout</i>	V

7.2.2 Pengujian fitur sambungan *database*

Pengujian fitur sambungan *database* dijabarkan pada Tabel 7.9. Skema pengujian dijabarkan pada kolom **Kondisi** dan fitur yang diuji dijabarkan pada kolom **Pengujian**.

Tabel 7.9 Pengujian *Black Box* Sambungan *Database*

No	Kondisi	Pengujian	Skema yang Diinginkan	Validasi
1	Semua masukan diisi dan proses sukses	 Koneksi ke server database	Sistem mengambil daftar database pada server database dan menampilkan kepada pengguna. Sistem akan memungkinkan pengguna untuk menjalankan fitur: 1) koneksi baru, 2) muat ulang koneksi dan 3) pemutusan koneksi	V
2	Semua masukan diisi dan proses gagal		Sistem akan memberikan pesan kepada pengguna bahwa koneksi gagal dibuat	V
3	Masukan diisi dengan karakter spasi dan simbol		Sistem akan memproses masukan tersebut	V

4	Jika masukan tidak diisi dengan lengkap		Sistem akan memberikan pesan bahwa koneksi gagal dibuat	V
4	Jika daftar tabel ada	Lihat daftar tabel database	Sistem akan menampilkan daftar tabel dari database yang dipilih pengguna.	V
5	Jika daftar tabel kosong		Sistem akan menampilkan daftar kosong pada halaman daftar tabel	V

7.2.3 Pengujian fitur kelola komponen

Pengujian fitur kelola komponen dijabarkan pada Tabel 7.10. Skema pengujian dijabarkan pada kolom **Kondisi** dan fitur yang diuji dijabarkan pada kolom **Pengujian**.

Tabel 7.10 Pengujian *Black Box* Kelola Komponen

No	Kondisi	Pengujian	Skema yang Diinginkan	Validasi
1	Jika <i>component model</i> tidak kosong	Kelola <i>component model</i>	Sistem akan menampilkan <i>component model</i> dengan opsi: 1) hapus <i>component model</i> dan 2) tambah <i>feature model</i>	V
2	Jika <i>component model</i> kosong		Sistem tidak akan menampilkan daftar <i>component model</i>	V
3	Jika <i>feature model</i> tidak kosong	Kelola <i>feature model</i>	Sistem akan menampilkan daftar <i>feature model</i> pada <i>component model</i> yang bersangkutan dengan opsi: 1) hapus <i>feature model</i> dan 2) melakukan <i>Table Manager</i>	V
4	Jika <i>feature model</i> kosong		Sistem akan menampilkan daftar <i>feature model</i>	
5	Jika <i>table model</i>	Kelola <i>table model</i>	Sistem akan menampilkan daftar <i>table model</i> dengan opsi: 1) jadikan tabel utama, 2) hapus <i>table</i>	V

	tidak kosong		<i>model</i> dan 3) melakukan <i>Column Manager</i>	
6	Jika <i>table model</i> kosong		Sistem tidak akan menampilkan daftar <i>table model</i>	V
7	Jika <i>column model</i> tidak kosong	Kelola <i>column model</i>	Sistem akan menampilkan <i>column model</i> dengan opsi: 1) aktifkan semua <i>column model</i> , 2) non-aktifkan semua <i>column model</i> , 3) ubah status aktif <i>column model</i> dan 4) melakukan <i>Input-Option Manager</i>	V
8	Jika <i>column model</i> kosong		Sistem tidak akan menampilkan daftar <i>column model</i>	V
9	Jika <i>input option model</i> tidak kosong	Kelola <i>input option model</i>	Sistem akan menampilkan <i>input option model</i> dengan opsi: 1) tambah <i>input option model</i> dan 2) hapus <i>input option model</i>	V
10	Jika <i>input option model</i> kosong		Sistem tidak akan menampilkan daftar <i>input option model</i> . Sistem akan menampilkan opsi tambah <i>input option model</i>	V

7.2.4 Pengujian fitur tambah komponen

Pengujian fitur tambah komponen dijabarkan pada Tabel 7.11. Skema pengujian dijabarkan pada kolom **Kondisi** dan fitur yang diuji dijabarkan pada kolom **Pengujian**.

Tabel 7.11 Pengujian *Black Box* Tambah Komponen

No	Kondisi	Pengujian	Skema yang Diinginkan	Validasi
1	-	Tambah <i>component model</i>	Sistem akan menambahkan <i>component model</i>	V
2	-	Tambah <i>feature model</i>	Sistem akan menambahkan <i>feature model</i> pada	V

			<i>component model</i> yang bersangkutan	
3	-	Tambah <i>table model</i>	Sistem akan menambahkan <i>table model</i> pada <i>feature model</i> yang bersangkutan	V
4	-	Tambah <i>input option model</i>	Sistem akan menambahkan <i>input option model</i> pada <i>column model</i> yang bersangkutan	V

7.2.5 Pengujian fitur parametrisasi komponen

Pengujian fitur parametrisasi komponen *module name* dijabarkan pada Tabel 7.12. Pengujian dilakukan dengan mencoba beberapa jenis masukan.

Tabel 7.12 Pengujian *Black Box* Parameterisasi *Module Name*

No	Kondisi	Pengujian	Skema yang Diinginkan	Validasi
1	Module name diisi kosong	Parameterisasi module name	Sistem akan memberikan peringatan dan menon-aktifkan tombol proses data	V
2	Module name diisi karakter spasi, angka atau simbol		Sistem akan memberikan peringatan bahwa masukan tidak boleh mengandung spasi dan simbol dan menghapus karakter tersebut	V
3	Module name diisi dengan masukan karakter uppercase		Sistem akan mengubah masukan menjadi lowercase	V
4	Module name diisi dengan karakter lowercase dan tidak mengandung spasi dan simbol		Sistem akan menerima masukan tersebut	V

Pengujian fitur parametrisasi *component model* dijabarkan pada Tabel 7.13. Pengujian dilakukan dengan mencoba beberapa jenis masukan.

Tabel 7.13 Pengujian *Black Box* Parameterisasi *Component Model*

No	Kondisi	Pengujian	Skema yang Diinginkan	Validasi
1	Component name diisi kosong	Parameterisasi component model	Sistem akan memberikan peringatan dan menon-aktifkan tombol proses data	V
2	Component name diisi karakter spasi atau simbol		Sistem akan memberikan peringatan bahwa masukan tidak boleh mengandung spasi dan simbol dan menghapus karakter tersebut	V
3	Component name diisi dengan masukan karakter uppercase		Sistem akan mengubah masukan menjadi lowercase	V
4	Component name diisi dengan karakter lowercase dan tidak mengandung spasi dan simbol		Sistem akan menerima masukan tersebut	V
5	Jika masukan component name sama dengan component name lain		Sistem akan memberikan pesan bahwa component name tidak boleh identik dan menon-aktifkan tombol proses data	V

Pengujian fitur parameterisasi *feature model* dijabarkan pada Tabel 7.14. Pengujian dilakukan dengan mencoba beberapa jenis masukan.

Tabel 7.14 Pengujian *Black Box* Parameterisasi *Feature Model*

No	Kondisi	Pengujian	Skema yang Diinginkan	Validasi
1	Function name diisi kosong	Parameterisasi feature model	Sistem akan menerima masukan kosong	V

2	Function name diisi karakter spasi atau simbol	Sistem akan memberikan peringatan bahwa masukan tidak boleh mengandung spasi dan simbol dan menghapus karakter tersebut	V
3	Function name diisi dengan masukan karakter uppercase	Sistem akan mengubah masukan menjadi lowercase	V
4	Function name diisi dengan karakter lowercase dan tidak mengandung spasi dan simbol	Sistem akan menerima masukan tersebut	V
5	Jika masukan Function name dan feature type sama dengan feature model lain dalam component model yang sama	Sistem akan memberikan pesan bahwa feature name atau feature type tidak boleh identik dan menon-aktifkan tombol proses data	V
6	Jika feature type dalam satu component model sama dengan feature type yang lain, namun function name berbeda	Sistem akan menerima masukan tersebut	
7	Mengubah tipe fitur dari feature model	Sistem akan mengubah data jenis fitur dari feature model yang bersangkutan	V
8	Jika table model pada feature model kosong	Sistem akan memberikan peringatan bahwa table model tidak boleh kosong. Sistem akan menon-	V

		aktifkan tombol proses data	
--	--	-----------------------------	--

Pengujian fitur parametisasi *table model* dijabarkan pada Tabel 7.15. Pengujian dilakukan dengan mencoba beberapa jenis masukan.

Tabel 7.15 Pengujian *Black Box* Parameterisasi *Table Model*

No	Kondisi	Pengujian	Skema yang Diinginkan	Validasi
1	Tidak memilih salah satu table model sebagai tabel utama	Parameterisasi table model	Sistem secara otomatis akan menjadikan table model pertama yang dipilih pengguna menjadi tabel utama	V
2	Memilih salah satu table model menjadi tabel utama		Sistem akan menghapus status tabel utama dari semua column model pada column manager, kemudian sistem akan menyetel status column model yang dipilih pengguna menjadi tabel utama	V

Pengujian fitur parametisasi *column model* dijabarkan pada Tabel 7.16. Pengujian dilakukan dengan mencoba beberapa jenis masukan.

Tabel 7.16 Pengujian *Black Box* Parameterisasi *Column Model*

No	Kondisi	Pengujian	Skema yang Diinginkan	Validasi
1	Mengubah tipe masukan pada column model	Parameterisasi column model	Sistem akan mengubah tipe masukan column model sesuai dengan pilihan pengguna	V
2	Mengubah status column model		Jika status column model yang bersangkutan aktif,	V

			maka sistem akan menon-aktifkan column model tersebut, begitu sebaliknya	
3	Mengaktifkan semua column model pada column manager		Sistem akan mengaktifkan semua column model pada column manager	V
4	Menon-aktifkan semua column model pada column manager		Sistem akan menon-aktifkan semua column model pada column manager	V

Pengujian fitur parametisasi *input-option model* dijabarkan pada Tabel 7.17. Pengujian dilakukan dengan mencoba beberapa jenis masukan.

Tabel 7.17 Pengujian *Black Box* Parameterisasi *Input-Option Model*

No	Kondisi	Pengujian	Skema yang Diinginkan	Validasi
1	Mengisi masukan key dan value dengan karakter spasi dan simbol	Parameterisasi input option manager	Sistem akan menerima masukan tersebut	V
2	Mengisi masukan key atau value dengan masukan kosong		Sistem akan menon-aktifkan tombol 'Save'	V
3	Mengisi masukan key dan value dengan karakter uppercase atau lowercase		Sistem akan menerima masukan tersebut	V

7.2.6 Pengujian fitur hapus komponen

Pengujian fitur hapus komponen dijabarkan pada Tabel 7.18. Pengujian dilakukan untuk memastikan bahwa fitur hapus komponen dapat berjalan sesuai harapan.

Tabel 7.18 Pengujian *Black Box* Hapus Komponen

No	Kondisi	Pengujian	Skema yang Diinginkan	Validasi
1	-	Hapus component model	Sistem akan memberikan pesan konfirmasi, jika pengguna meyetujui maka sistem akan menghapus data component model	V
2	-	Hapus feature model	Sistem akan memberikan pesan konfirmasi, jika pengguna meyetujui maka sistem akan menghapus data feature model	V
3	-	Hapus table model	Sistem akan memberikan pesan konfirmasi, jika pengguna meyetujui maka sistem akan menghapus data table model	V
4	-	Hapus input option model	Sistem akan memberikan pesan konfirmasi, jika pengguna meyetujui maka sistem akan menghapus data input option model	V

7.2.7 Pengujian fitur pembangkitan kode program

Pengujian fitur pembangkitan kode program dan unduh komponen dijabarkan pada Tabel 7.19. Pengujian dilakukan untuk memastikan bahwa fitur tersebut dapat berjalan sesuai harapan.

Tabel 7.19 Pengujian *Black Box* Pembangkitan Kode Program dan Unduh Komponen

No	Kondisi	Pengujian	Skema yang Diinginkan	Validasi
1	-	Hasilkan komponen	Sistem akan menghasilkan kode program yang sesuai dengan component model	v
2	Melakukan unduh semua komponen	Unduh komponen	Sistem akan mengunduh semua kode program yang telah dihasilkan	v

3	Mengunduh kode program per komponen		Sistem akan mengunduh kode program pada komponen yang bersangkutan	v
4	Mendunduh satu unit kode program		Sistem akan mengunduh satu unit kode program yang dipilih oleh pengguna	v

7.3 Analisis Pengujian

Berdasarkan *unit testing* yang telah dilakukan, diketahui bahwa komponen *controler knowledge* memiliki nilai *cyclomatic complexity* 50. Jumlah tersebut menunjukkan bahwa komponen tersebut memiliki tingkat kompleksitas yang tinggi. Pengujian komponen *controller knowledge* membutuhkan sumber daya waktu dan tenaga yang besar, karena penguji harus memeriksa kasus uji sebanyak 50.

Komponen *model knowledge* memiliki nilai *cyclomatic complexity* senilai 28. Berdasarkan data tersebut, untuk melakukan *unit testing* komponen *model knowledge* membutuhkan 56% sumber daya pada saat *unit testing* komponen *controller knowledge*.

Komponen *view add knowledge* memiliki nilai *cyclomatic complexity* senilai 21. Berdasarkan data tersebut, untuk melakukan *unit testing* komponen *view add knowledge* membutuhkan 42% sumber daya pada saat *unit testing* komponen *controller knowledge*.

Unit testing dari ketiga komponen *domain knowledge* menunjukkan bahwa rata – rata komponen *domain knowledge* memiliki nilai *cyclomatic complexity* senilai 33. Rata – rata komponen *domain knowledge* memiliki struktur percabangan yang banyak, maka dari itu proses *unit testing* harus dilakukan dengan teliti.

Dari 99 kasus uji yang dilakukan pada *unit testing*, menunjukkan tingkat validasi 100%. Hal tersebut menunjukkan bahwa komponen *domain knowledge* dapat menerjemahkan kode program dengan valid.

Release testing dilakukan pada fitur – fitur fungsional *code generator*, dari 64 kasus uji memiliki tingkat validasi 100%. Hal tersebut bisa disimpulkan bahwa *code generator* dapat menjalankan fungsionalitas sistem sesuai dengan harapan.

Fitur *login* memiliki 11 jumlah kasus uji, fitur kelola model komponen memiliki 45 kasus uji dan fitur hasilkan kode program memiliki 4 kasus uji. Fitur kelola model komponen memiliki jumlah kasus uji yang paling banyak, sehingga fitur model kelola komponen bisa disimpulkan adalah fitur yang rumit. Fitur kelola model komponen harus didesain dengan tingkat pengalaman pengguna (*use experience*)

yang baik agar fitur tersebut dapat digunakan dengan mudah dan tidak membuat pengguna bingung.



BAB 8 PENUTUP

Pada bab ini dipaparkan penutup hasil penelitian berupa kesimpulan dan saran. Sub-bab kesimpulan menjelaskan apakah hasil penelitian yang telah dilakukan bisa menjawab pertanyaan yang diajukan pada sub-bab rumusan masalah. Sub-bab saran berisi pernyataan-pernyataan yang ringkas dan jelas tentang masalah-masalah atau hal-hal yang dapat dilakukan untuk mengembangkan penelitian ini lebih lanjut.

8.1 Kesimpulan

Kesimpulan yang bisa diambil dari penelitian ini adalah sebagai berikut:

- 1) Prototipe *code generator* dapat dibuat untuk mendemonstrasikan kemampuan menghasilkan kode program PTIIK-Apps.
- 2) Identifikasi persyaratan dalam membuat *code generator* dilakukan dengan: identifikasi aktor, analisis persyaratan fungsional dan non-fungsional. Syarat yang harus dipenuhi adalah:
 - ✓ Aktor yang terlibat dalam sistem adalah **pengembang**
 - ✓ Persyaratan fungsional yang utama untuk sistem ini meliputi: 1) kelola model masukan, 2) hasilkan kode program dan 3) *login*
 - ✓ Persyaratan non-fungsional sistem meliputi aspek: 1) *compability* dan 2) *security*
- 3) Rancangan *code generator* terdiri dari *application description* dan *domain knowledge*, berikut ini penjelasan hasil rancangan dari:
 - ✓ *Application description* didefinisikan kedalam sebuah rancangan diagram kelas yang terdiri dari: 1) model modul, 2) model komponen, 3) model fitur, 4) model tabel, 5) model kolom dan 6) model pilihan masukan.
 - ✓ *Domain knowledge* didefinisikan kedalam rancangan diagram aktifitas dan diagram urutan yang menjelaskan bagaimana sebuah unit basis pengetahuan sistem dapat membangkitkan kode program PTIIK-Apps.
 - ✓ *Code generator* definisikan kedalam rancangan diagram aktifitas, diagram urutan dan diagram kelas yang menjelaskan bagaimana sebuah *code generator* mengolah model masukan *application description* dan mengirim ke unit *domain knowledge* untuk diterjemahkan menjadi kode program.
- 4) Implementasi *code generator* dilakukan dengan mengembangkan aplikasi berbasis web dengan bahasa pemrograman PHP. Aplikasi *code generator* memposisikan *application description* sebagai model masukan dan menggunakan *domain knowledge* sebagai basis pengetahuan untuk menerjemahkan model masukan menjadi kode program PTIIK-Apps. Sistem *code generator* berjalan dengan beberapa tahapan proses, yaitu:

- 1) proses *login*
- 2) proses pengisian model masukan
- 3) proses penerjemahan model masukan menjadi kode program
- 4) proses menampilkan hasil kode program agar bisa diunduh oleh pengembang
 - ✓ Implementasi *application description* diterapkan pada proses halaman pengisian model masukan. Pada halaman tersebut, model masukan ditampilkan berdasarkan struktur model *application description* yang telah didefinisikan. Pengisian model tabel dilakukan dengan mengambil metadata tabel dari *server* basis data yang sesuai dengan persyaratan pengembang.
 - ✓ Implementasi *domain knowledge* berupa komponen *template* yang berisi model perintah kode program PTIIK-Apps dan alur logis untuk menampilkan data dari parameter yang diterima. Komponen *template domain knowledge* terdiri dari:
 - 1) *controller knowledge*
 - 2) *model knowledge*
 - 3) *view add knowledge*
 - 4) *view edit knowledge*
 - 5) *view list knowledge*
 - 6) *view filter knowledge*
 - 7) *view detail knowledge*
- 5) Berdasarkan hasil pengujian, dapat disimpulkan bahwa *code generator* yang dibuat dapat menghasilkan komponen kode program PTIIK-Apps dengan valid. Komponen kode program yang dihasilkan oleh *code generator* dapat menjalankan fungsi lihat, tambah, perbarui dan hapus data terhadap sebuah tabel basis data dan tabel lain yang berhubungan.

8.2 Saran

Saran yang peneliti paparkan agar penelitian ini bisa dikembangkan ke tahap yang lebih lanjut adalah:

- Karena waktu dan sumber daya manusia pada penelitian ini terbatas maka analisis modul tidak bisa seluruhnya dilakukan. Harapannya cakupan modul yang dianalisis bisa diperluas lagi sehingga bisa dihasilkan model domain yang lebih lengkap dalam mendeskripsikan domain PTIIK-Apps.
- Penelitian ini telah menghasilkan sebuah *code generator* yang bisa menghasilkan kode program secara otomatis. Namun belum diketahui seberapa efisien *code generator* dapat membantu para pengembang, sehingga

perlu dilakukan penelitian untuk mengetahui dan membuktikan argumen tersebut.



DAFTAR PUSTAKA

- Cox, L.J., 2010. *Featured-Oriented Domain Analysis for Semantic Web Service*. Alberta: Athabasca University.
- Spoelstra, W., 2010. *Reusing Software Assets in Agile Development - A Management Tool*. Hengelo: Information Systems and Change Management (ISCM).
- Possatto, M.A. & Lucrédio, D., 2015. *Automatically Propagating from Reference Implementations to Code Generation Templates*. Washington: Computing Department, Federal University of São Carlos.
- Harsu, M. 2015. *A Survey on Domain Engineering*. Tampere: Tampere University.
- Bayer, J., Eisenbarth, M., Lehrer, T., Puhlmann, F., Richter, E., Schnieders, A. & Weiland, J., 2004. *Domain Engineering Technique and Process Modeling*. Lisbon: PESOA Project.
- Safavi, S.A. & Shaikh, M.U., 2011. *Effort Estimation Model for Each Phase of Software Development Life Cycle*. Islamabad: COMSATS Institute of Information Technology.
- Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E. & Peterson, A.S., 1990. *Feature-Oriented Domain Analysis (FODA) Feasibility Study*. Pittsburgh: Carnegie Mellon University.
- Stuikys, V. & Damasevicius, R., 2013. *Meta-Programming and Model-Driven Meta-Program Development*. New York: Springer.
- Hazard, K. & Bock, J., 2013. *Metaprogramming in .NET*. New York: Manning Publications Co.
- Gabrysiak, G., Marr, S. & Menge, F., n.d, *Meta Programming and Reflection in PHP*. Potsdam: Hasso-Plattner Institute.
- Lockhart, J., 2015. *Modern PHP*. California: O'Reilly Media.
- Sommerville, I., 2007. *Software Engineering*. 8th ed. London: Addison-Wesley.
- Sommerville, I., 2011. *Software Engineering*. 9th ed. London: Addison-Wesley.
- Gheorghe, L., Hayder, H. & Maia, J.P., 2006. *Smarty, PHP template Programming and Applications*. Birmingham: PACKT Publishing.
- Oracle, 2015. *MySQL Tutorial*. California: Oracle Corporation.
- PHP Group, 2016. *PHP: Hypertext Preprocessor*. [online] Tersedia di: <<http://php.net>> [Diakses 1 Januari 2016]
- Oracle Corporation, 2016. *MySQL:: Developer Zone*. [online] Tersedia di: <<https://dev.mysql.com>> [Diakses 1 Januari 2016]

LAMPIRAN A FORM REKAYASA DOMAIN

A.1 Form Definisi Model Fitur

Pada form berikut ini, dijelaskan definisi setiap fitur yang digunakan untuk menyusun sebuah model fitur pada Gambar 5.10.

Nama: Lihat Data

Deskripsi: Fitur yang digunakan untuk menampilkan daftar data dari sebuah tabel beserta opsi untuk melakukan operasi data (opsi tambah, perbarui, hapus dan/atau melihat detail data).

Hubungan *consists-of*: Tabel Sortir Data atau Tabel Sederhana [opsional]

Sumber: Analisis fitur sistem PTIIK-Apps pada modul KKNP dan modul kerjasama.

Tipe: compile-time

Nama: Tabel Sortir

Deskripsi: Fitur yang digunakan untuk menampilkan data pada fitur Lihat Data dengan dilengkapi form untuk melakukan sortir data berdasarkan kolom pada tabel cabang yang berhubungan.

Hubungan *consists-of*: -

Sumber: Analisis fitur sistem PTIIK-Apps pada modul KKNP dan modul kerjasama.

Tipe: compile-time

Fitur Mandatory: Lihat Data

Nama: Tabel Sederhana

Deskripsi: Fitur yang digunakan untuk menampilkan data pada fitur Lihat Data yang disajikan dengan form sederhana (tidak ada form sortir data).

Hubungan *consists-of*: -

Sumber: Analisis fitur sistem PTIIK-Apps pada modul KKNP dan modul kerjasama.

Tipe: compile-time

Fitur Mandatory: Lihat Data

Nama: Tambah Data

Deskripsi: Fitur yang digunakan untuk menambahkan data kedalam sebuah tabel.

Hubungan *consists-of*: Tambah Data ke Tabel Cabang [alternatif]

Sumber: Analisis fitur sistem PTIIK-Apps pada modul KKNP dan modul kerjasama.

Tipe: compile-time

Nama: Tambah Data ke Tabel Cabang

Deskripsi: Fitur yang digunakan untuk menambahkan data kedalam semua tabel cabang yang berhubungan dengan tabel utama yang didefinisikan.

Hubungan *consists-of*: -

Sumber: Analisis fitur sistem PTIIK-Apps pada modul KKNP dan modul kerjasama.

Tipe: compile-time

Nama: Perbarui Data

Deskripsi: Fitur yang digunakan untuk mengubah nilai data pada sebuah tabel.

Hubungan *consists-of*: Perbarui Data dari Tabel Cabang [alternatif]

Sumber: Analisis fitur sistem PTIIK-Apps pada modul KKNP dan modul kerjasama.

Tipe: compile-time

Nama: Perbarui Data dari Tabel Cabang

Deskripsi: Fitur yang digunakan untuk memperbarui (menambah, mengubah atau menghapus) data dari semua tabel cabang yang berhubungan dengan tabel utama yang didefinisikan.

Hubungan *consists-of*: -

Sumber: Analisis fitur sistem PTIIK-Apps pada modul KKNP dan modul kerjasama.

Tipe: compile-time

Nama: Detail Data

Deskripsi: Fitur yang digunakan untuk menampilkan detail data dari sebuah tabel.

Hubungan *consists-of*: Detail Data dari Tabel Cabang [alternatif]

Sumber: Analisis fitur sistem PTIIK-Apps pada modul KKNP dan modul kerjasama.

Tipe: compile-time

Nama: Detail Data dari Tabel Cabang

Deskripsi: Fitur yang digunakan untuk menampilkan detail data pada semua tabel cabang yang berhubungan dengan tabel utama yang didefinisikan.

Hubungan *consists-of*: -

Sumber: Analisis fitur sistem PTIIK-Apps pada modul KKNP dan modul kerjasama.

Tipe: compile-time

Nama: Hapus Data

Deskripsi: Fitur yang digunakan untuk menghapus data dari sebuah tabel.

Hubungan *consists-of*: -

Sumber: Analisis fitur sistem PTIIK-Apps pada modul KKNP dan modul kerjasama.

Tipe: compile-time

A.2 Form Deskripsi *Entity*

Form berikut ini menjelaskan tentang detail setiap *entity* yang digambarkan pada Gambar 5.11.

Entity: *PTIIK Framework System*

Deskripsi: Komponen yang digunakan sebagai penyedia arsitektur *framework* sistem.

Sumber: Analisis fitur sistem PTIIK-Apps pada modul KKNP dan modul kerjasama

Entity: *Config*

Deskripsi: Komponen yang menyediakan variabel – variabel global yang bisa digunakan oleh komponen *controller*, *model* dan *view* (MVC).

Sumber: Analisis fitur sistem PTIIK-Apps pada modul KKNP dan modul kerjasama

Entity: *Modul*

Deskripsi: Komponen yang berfungsi untuk mengelompokkan komponen MVC kedalam sebuah modul.

Sumber: Analisis fitur sistem PTIIK-Apps pada modul KKNP dan modul kerjasama

Entity: *Framework Controller*

Deskripsi: Komponen yang berisi sekumpulan fungsi – fungsi yang digunakan oleh komponen *controller* aplikasi dalam mengelola alur suatu proses. Misalnya: 1) fungsi untuk mengarahkan ke fungsi *controller* lain, 2) fungsi untuk mendefinisikan kelas komponen *model*, 3) fungsi untuk mendapatkan desain halaman komponen *view*.

Sumber: Analisis fitur sistem PTIIK-Apps pada modul KKNP dan modul kerjasama

Entity: *Comscontroller*

Deskripsi: Komponen yang berisi fungsi – fungsi global yang bisa diakses oleh komponen MVC pusat dan komponen MVC tiap modul.

Sumber: Analisis fitur sistem PTIIK-Apps pada modul KKNP dan modul kerjasama

Entity: *Comsmodule*

Deskripsi: Komponen yang berisi fungsi – fungsi global yang bisa diakses oleh komponen MVC tiap modul.

Sumber: Analisis fitur sistem PTIIK-Apps pada modul KKNP dan modul kerjasama

Entity: *Controller*

Deskripsi: Komponen yang berisi fungsi – fungsi untuk mengatur alur proses dalam menjalankan fitur aplikasi.

Sumber: Analisis fitur sistem PTIIK-Apps pada modul KKNP dan modul kerjasama

Entity: *Framework Model*

Deskripsi: Komponen yang berisi sekumpulan fungsi – fungsi yang digunakan oleh komponen *model* dalam menjalankan fungsi operasi (tambah, perbarui dan hapus) data dan fungsi untuk mendapatkan data.

Sumber: Analisis fitur sistem PTIIK-Apps pada modul KKNP dan modul kerjasama

Entity: *Model*

Deskripsi: Komponen yang berisi fungsi – fungsi untuk mengelola alur proses dalam mendapatkan dan mengelola operasi data (tambah, perbarui dan hapus) pada tabel basis data. Misalhnya: 1) fungsi untuk menyimpan data, 2) fungsi untuk memperbarui data, 3) fungsi untuk menghapus data, 4) fungsi untuk mengeksekusi bahasa SQL (*structured query language*).

Sumber: Analisis fitur sistem PTIIK-Apps pada modul KKNP dan modul kerjasama

Entity: *View*

Deskripsi: Komponen yang mengatur alur data dan komponen HTML untuk mengelola tampilan tatap muka.

Sumber: Analisis fitur sistem PTIIK-Apps pada modul KKNP dan modul kerjasama

Entity: *Javascript*

Deskripsi: Komponen yang digunakan untuk mengelola komponen tatap muka tanpa memuat ulang komponen *view*.

Sumber: Analisis fitur sistem PTIIK-Apps pada modul KKNP dan modul kerjasama

A.3 Form Tipe Relasi

Form berikut ini menjelaskan tentang detail relasi antar *entity* yang telah dijelaskan pada sub-bab lampiran A.2.

Tipe Relasi: Relasi consists-of dari komponen *PTIIK Framework System* dengan komponen *Framework Controller*, *Framework Model*, *config* dan *Modul*

Deskripsi: Komponen *PTIIK Framework Sytem* memiliki relasi agregat dengan komponen *Framework Controller*, sehingga komponen *PTIIK Framework System* memiliki kelengkapan fungsi – fungsi dari komponen *Framework Controller*.

Relasi agregat komponen *PTIIK Framework System* dengan komponen *config* berfungsi agar variabel umum pada komponen *config* bisa digunakan oleh komponen – komponen yang memiliki hubungan agregat dengan komponen *PTIIK Framework System*.

Relasi agregat antara *PTIIK Framework System* dengan *Framework Model* agar komponen *PTIIK Framework System* memiliki kelengkapan fungsi – fungsi dari komponen *Framework Model*.

Komponen *PTIIK Framework System* memiliki relasi agregat dengan komponen *Modul*, sehingga komponen *PTIIK Framework System* memiliki mekanisme untuk mengelompokkan komponen MVC dalam bentuk sekumpulan komponen *Modul*.

Parts: (Memiliki *controller* {*PTIIK Framework System* | *Framework Controller* })

(Memiliki *model* {*PTIIK Framework System* | *Framework Model* })

(Memiliki *config* {*PTIIK Framework System* | *Config* })

(Memiliki *modul* {*PTIIK Framework System* | *Modul* })

Konektifitas: (Memiliki *controller* {ONE | ONE})

(Memiliki *model* {ONE | ONE})

(Memiliki *config* {ONE | ONE})

(Memiliki *modul* {ONE | MANY})

Keterangan: Seluruh komponen *Framework Controller* adalah bagian dari *PTIIK Framework System*.

Seluruh komponen *Framework Model* adalah bagian dari *PTIIK Framework System*.

Seluruh komponen *config* adalah bagian dari *PTIIK Framework System*.

Seluruh komponen *Modul* adalah bagian dari *PTIIK Framework System*.

Komponen *PTIIK Framework System* terdiri dari beberapa komponen *Modul*.

Tipa Relasi: Relasi is-a dari komponen *Framework Controller* dengan komponen *comscontroller*

Deskripsi: Komponen *comscontroller* merupakan spesialisasi dari komponen *Framework Controller*, dengan memiliki fungsi – fungsi umum yang bisa diakses oleh komponen MVC pusat dengan MVC cabang pada tiap modul.

Parts: (Spesialisasi *Framework Controller* {*Framework Controller* | *comscontroller*})

Konektifitas: (Spesialisasi *Framework Controller* {ONE | ONE})

Keterangan: Komponen *comscontroller* memiliki kesamaan dengan komponen *Framework Controller* dan memiliki spesialisasi berupa penambahan beberapa fungsi umum yang bisa digunakan oleh komponen MVC pusat dengan MVC cabang pada tiap modul.

Tipa Relasi: Relasi is-a dari komponen *comscontroller* dengan komponen *comsmodule*

Deskripsi: Komponen *comsmodule* merupakan spesialisasi dari komponen *comscontroller*, dengan memiliki fungsi – fungsi umum yang bisa diakses oleh komponen MVC cabang pada tiap modul.

Parts: (Spesialisasi *comscontroller* {*comscontroller* | *comsmodule*})

Konektifitas: (Spesialisasi *comscontroller* {ONE | ONE})

Keterangan: Komponen *comsmodule* memiliki kesamaan dengan komponen *comscontroller* dan memiliki spesialisasi berupa penambahan beberapa fungsi umum yang bisa digunakan oleh komponen MVC cabang pada tiap modul.

Tipa Relasi: Relasi is-a dari komponen *controller* dengan komponen *comsmodule*

Deskripsi: Komponen *controller* merupakan spesialisasi dari komponen *comsmodule*. Komponen *controller* memiliki fungsi – fungsi spesialisasi dalam mengelola alur data untuk menjalankan fitur aplikasi.

Parts: (Spesialisasi *comsmodule* {*comsmodule* | *controller*})

Konektifitas: (Spesialisasi *comsmodule* {ONE | MANY})

Keterangan: Komponen *controller* memiliki kesamaan dengan komponen *comsmodule* dan memiliki spesialisasi berupa penambahan beberapa fungsi – fungsi khusus dalam mengelola alur data untuk menjalankan fitur aplikasi.

Type Relasi: Relasi is-a dari komponen *controller* dengan komponen *comsmodule*

Deskripsi: Komponen *controller* merupakan spesialisasi dari komponen *comsmodule*. Komponen *controller* memiliki fungsi – fungsi spesialisasi dalam mengelola alur data untuk menjalankan fitur aplikasi.

Parts: (Spesialisasi *comsmodule* {*comsmodule* | *controller*})

Konektifitas: (Spesialisasi *comsmodule* {ONE | MANY})

Keterangan: Komponen *controller* memiliki kesamaan dengan komponen *comsmodule* dan memiliki spesialisasi berupa penambahan beberapa fungsi – fungsi khusus dalam mengelola alur data untuk menjalankan fitur aplikasi.

Type Relasi: Relasi is-a dari komponen *Framework Model* dengan komponen *Model*

Deskripsi: Komponen *model* merupakan spesialisasi dari komponen *Framework Model*. Komponen *model* memiliki fungsi – fungsi spesialisasi dalam mengelola alur proses operasi data dan pengambilan data pada sistem basis data.

Parts: (Spesialisasi *framework model* {*framework model* | *model*})

Konektifitas: (Spesialisasi *framework model* {ONE | MANY})

Keterangan: Komponen *model* memiliki kesamaan dengan komponen *framework model* dan memiliki spesialisasi berupa penambahan beberapa fungsi – fungsi khusus dalam mengelola alur proses operasi data dan pengambilan data pada basis data.

Type Relasi: Relasi consist-of dari komponen *Modul* dengan komponen *Controller*

Deskripsi: Komponen *Modul* terdiri dari beberapa komponen *Controller* sehingga komponen MVC bisa dikelompokkan sesuai dengan persyaratan.

Parts: (Memiliki *Controller* {*Modul* | *Controller*})

Konektifitas: (Memiliki *Controller* {ONE | MANY})

Keterangan: Komponen *Modul* memiliki beberapa komponen *Controller* yang telah dikelompokkan.

Tipe Relasi: Relasi consist-of dari komponen *Controller* dengan komponen *Model* dan *View*

Deskripsi: Komponen *Controller* memiliki relasi dengan beberapa komponen *Model*, sehingga komponen *Controller* dapat melakukan fungsi – fungsi operasi data yang dimiliki oleh komponen *Model*.

Komponen *Controller* juga memiliki relasi dengan komponen *View*, sehingga komponen *Controller* memiliki kemampuan untuk menampilkan komponen HTML sebagai tampilan tatap muka.

Parts: (Memiliki *Model* {*Controller* | *Model*})

(Memiliki *View* {*Controller* | *View*})

Konektifitas: (Memiliki *Model* {MANY | MANY})

(Memiliki *View* {MANY | MANY})

Keterangan: Komponen *Controller* memiliki beberapa komponen *Model*.

Sebuah komponen *Model* bisa dimiliki oleh beberapa komponen *Controller*.

Komponen *Controller* memiliki beberapa komponen *View*.

Sebuah komponen *View* bisa dimiliki oleh beberapa komponen *Controller*.

Tipe Relasi: Relasi consist-of dari komponen *View* dengan komponen *Javascript*

Deskripsi: Komponen *View* memiliki beberapa komponen *Javascript* untuk mengelola proses interaksi tatap muka tanpa memuat ulang komponen *View*.

Parts: (Memiliki *Javascript* {*View* | *Javascript*})

Konektifitas: (Memiliki *Javascript* {MANY | MANY})

Keterangan: Komponen *View* memiliki beberapa komponen *Javascript*.

Sebuah komponen *Javascript* bisa dimiliki oleh beberapa komponen *View*.

A.4 Form Deskripsi Isu

Form berikut ini menjelaskan tentang detail isu yang muncul ketika proses pengambilan keputusan pada model fitur.

Isu: bentuk tampilan tatap muka yang digunakan untuk menampilkan data

Deskripsi: sebuah keputusan yang harus diambil pada fitur lihat data, dimana pengembang harus memilih bahwa data yang ditampilkan ditunjukkan dengan model tabel sederhana atau dengan model tabel dengan fitur sortir data.

Muncul pada: komponen fitur Lihat Data

Keputusan: Tabel Sederhana

Deskripsi: tabel sederhana cocok digunakan untuk menampilkan data yang memiliki karakteristik: 1) tidak memiliki baris yang banyak dalam sebuah tabel, 2) tidak memiliki relasi dengan tabel lain.

Alasan: tabel sederhana menampilkan daftar data dan opsi operasi data dengan tampilan sederhana (tanpa form sortir data), tampilan tersebut efektif digunakan untuk menampilkan data dengan jumlah baris yang sedikit.

Kendala/Persyaratan: ketika pengembang memilih menampilkan data dengan form sederhana, maka data tersebut tidak ditampilkan dengan form sortir data.

Berlaku untuk: komponen fitur Tabel Sederhana

Keputusan: Tabel Sortir Data

Deskripsi: tabel sortir data cocok digunakan untuk menampilkan data yang memiliki karakteristik: 1) memiliki baris yang banyak dalam sebuah tabel, 2) dan memiliki relasi terhadap satu atau lebih dengan tabel lain.

Alasan: tabel sortir data menampilkan daftar data dan opsi operasi data dilengkapi dengan form sortir data, sehingga pengguna dapat melakukan sortir data dengan parameter yang dikehendaki.

Kendala/Persyaratan: ketika pengembang memilih menampilkan data dengan form sortir data, maka data tersebut tidak ditampilkan dengan form sederhana.

Berlaku untuk: komponen fitur Tabel Sortir Data

LAMPIRAN B DOMAIN TERMINOLOGY DICTIONARY

Application Description: sebuah model yang digunakan untuk mendeskripsikan struktur komponen sistem secara abstrak yang akan diproses oleh sistem *program generator*, sehingga bisa dihasilkan komponen sistem konkret sesuai dengan abstraksi model tersebut.

Application Domain Knowledge: sebuah arsitektur yang berisi basis pengetahuan tentang domain PTIIK-Apps, *application domain knowledge* digunakan untuk memproses masukan yang diterima menjadi kode program yang bisa dieksekusi pada domain PTIIK-Apps.

Template: adalah lapisan/bagian komponen yang digunakan untuk menampilkan data dan terpisah dengan lapisan komponen pengolah data.

Reusable Assets: adalah sebuah sekumpulan komponen (*assets*) perangkat lunak yang dapat digunakan ulang untuk pengembangan perangkat lunak baru.

Prototipe: adalah versi awal dari sistem perangkat lunak yang digunakan untuk: 1) mendemonstrasikan sebuah konsep, 2) mencoba suatu pilihan desain dan 3) untuk menemukan permasalahan lebih lanjut beserta solusinya.

Metadata: adalah sekumpulan data yang memodelkan data itu sendiri.

Model Masukan: adalah sebuah desain model yang digunakan sebagai struktur dalam memodelkan masukan pengembang.

Tabel Utama: adalah sebuah tabel target dimana operasi (tambah, ubah dan hapus) data dilakukan.

Tabel Cabang: adalah satu atau beberapa tabel yang memiliki hubungan dengan tabel utama.

Component Manager: adalah sekumpulan proses yang digunakan untuk mengelola model komponen pada saat pendefinisian *application description*.

Table Manager: adalah sekumpulan proses yang digunakan untuk mengelola model tabel pada saat pendefinisian *application description*.

Column Manager: adalah sekumpulan proses yang digunakan untuk mengelola model kolom pada saat pendefinisian *application description*.

Input-Option Manager: adalah sekumpulan proses yang digunakan untuk mengelola model pilihan masukan pada saat pendefinisian *application description*.

Entity: adalah sebuah model yang digunakan sebagai abstraksi obyek atau komponen pada domain.

PTIIK Framework System: adalah arsitektur framework berbasis hirarkikal MVC (*model view controller*) yang digunakan untuk membuat aplikasi berbasis web PTIIK-Apps.

MVC: adalah sekumpulan komponen *model*, *view* dan *controller*.

Controller: adalah sebuah unit yang digunakan untuk memproses alur suatu data pada saat menjalankan fitur aplikasi.

Model: adalah sebuah unit yang digunakan untuk mengelola operasi data ke sistem pengelola basis data dan untuk mendapatkan data yang dibutuhkan oleh unit *controller*.

View: adalah unit yang digunakan *controller* untuk menghasilkan tampilan tatap muka.

Domain: adalah area/cakupan sebuah keahlian yang terdiri dari fungsi – fungsi yang saling berhubungan, pada penelitian ini domain yang dimaksud adalah cakupan arsitektur perangkat lunak PTIIK-Apps yang dijadikan objek penelitian.

CRUD: adalah proses operasi data *create* (menambah data), *read* (membaca data), *update* (memperbarui data) dan *delete* (menghapus data).

Checkbox: adalah tipe masukan ganda dimana pengguna dapat memilih satu atau lebih dari pilihan tersebut.

Radio: adalah tipe masukan ganda dimana pengguna hanya dapat memilih salah satu dari masukan tersebut.

Select-option: adalah tipe masukan ganda dimana pengguna hanya dapat memilih salah satu dari masukan tersebut. Masukan tersebut ditampilkan dalam sebuah form masukan, ketika pengguna melakukan klik maka form tersebut akan menampilkan beberapa opsi pilihan.

Dropdown: adalah tipe tombol yang menampilkan daftar opsi ketika pengguna melakukan klik pada tombol tersebut.

LAMPIRAN C HASIL KUISIONER

C.1 Kuisisioner Proses Pengembangan Aplikasi PTIIK-Apps

Kuisisioner ini digunakan untuk mengetahui kondisi PTIIK-Apps saat ini. Kuisisioner dibagikan ke enam orang pengembang PTIIK-Apps, hasil dari kuisisioner ini adalah sebagai berikut:

1. Apakah proses pengembangan aplikasi PTIIK-Apps sudah selesai 100%?

100% narasumber menjawab belum selesai, karena:

- masih ada beberapa proses bisnis fakultas yang belum bisa dikelola oleh PTIIK-Apps
- masih perlu penambahan modul lain yang perlu diimplementasikan

2. Apakah penambahan modul baru pada PTIIK-Apps masih memungkinkan?

100% narasumber menjawab ya, karena:

- persyaratan pengguna aplikasi bisa bertambah atau mungkin berubah sesuai dengan perkembangan
- masih banyak persyaratan pengguna yang belum terpenuhi

3. Bagaimana proses pengembangan aplikasi PTIIK-Apps jika ada penambahan modul baru?

- membuat kode program modul baru, jika terdapat fungsi yang sudah ada maka harus memodifikasinya
- menambahkan komponen modul baru pada PTIIK-Apps dan membuat kelas dan fungsi baru

4. Apakah PTIIK-Apps dibangun dengan bahasa pemrograman PHP dan Javascript?

100% narasumber menjawab ya.

5. Apakah PTIIK-Apps menggunakan *design pattern Model View Controller (MVC)*?

100% narasumber menjawab ya.

C.2 Kuisisioner Penelitian Pembangunan *Code Generator*

Kuisisioner ini digunakan untuk mengambil data yang dibutuhkan untuk proses pengembangan *code generator*. Kuisisioner dibagikan ke lima orang pengembang PTIIK-Apps, hasil dari kuisisioner ini adalah sebagai berikut:

1. Apakah dalam proses pengembangan aplikasi PTIIK-Apps ditemukan kendala?

100% narasumber menjawab ya, terdapat kendala pada proses pengembangan PTIIK-Apps.

2. Jika ditemukan kendala, apakah salah satunya adalah implementasi kode program yang memakan waktu yang lama?

100% narasumber menjawab ya, yaitu implementasi kode program pada:

3. Apakah dalam proses implementasi kode program terdapat kendala berupa penulisan baris kode dengan struktur yang sama secara berulang – ulang?

- proses tambah, perbarui dan hapus pada struktur tabel basis data yang sederhana
- fungsi yang sama, namun pada modul yang berbeda

4. Bagaimana dampak yang ditimbulkan jika “penulisan baris kode dengan struktur yang sama secara berulang – ulang” tetap dilakukan?

- memakan ruang memori yang lebih besar
- membuat pengembang aplikasi menjadi jenuh dan lelah
- pemborosan waktu kerja
- jika terjadi perubahan kode, maka harus dilakukan perubahan pada semua komponen

5. Apakah kendala tersebut sudah terselesaikan? Dengan cara yang seperti apa?

- menggunakan aset modul yang sudah ada, kemudian melakukan modifikasi sesuai dengan persyaratan
- tiga dari lima narasumber menjawab belum menemukan kendala

6. Apakah cara yang dipakai sudah efektif dan menyelesaikan masalah?

tiga dari lima narasumber menjawab tidak, karena:

- perlu ada metode yang memudahkan dalam menghasilkan kode program tanpa harus ada penulisan fungsi yang berulang – ulang
- meskipun memanfaatkan aset modul yang sudah ada, namun masih perlu melakukan modifikasi dan membutuhkan waktu yang cukup lama

satu narasumber menjawab bahwa solusi yang dipakai sudah cukup efektif

7. Apakah ada solusi yang lain yang memungkinkan untuk menyelesaikan masalah tersebut namun belum pernah dicoba?

- membutuhkan sebuah alat yang bisa menghasilkan komponen MVC yang bisa melakukan operasi tambah, lihat, perbarui dan hapus data secara otomatis dengan mengirimkan parameter yang dibutuhkan
- memerlukan pengembangan *metaprogram* untuk mendukung pengembangan PTIIK-Apps untuk masa depan