PERANCANGAN DAN PEMBUATAN SITUS *SOCIAL NETWORKING* MENGGUNAKAN PRINSIP *REPRESENTATIONAL STATE TRANSFER*

SKRIPSI

KONSENTRASI TEKNIK INFORMATIKA DAN KOMPUTER

Diajukan untuk memenuhi persyaratan
memperoleh gelar Sarjana Teknik

Disusun oleh :
AGI PUTRA KHARISMA
NIM. 0410630003-63

DEPARTEMEN PENDIDIKAN NASIONAL
UNIVERSITAS BRAWIJAYA
FAKULTAS TEKNIK
MALANG
2009

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

repository.ub.ac.id

LEMBAR PERSETUJUAN

PERANCANGAN DAN PEMBUATAN SITUS *SOCIAL NETWORKING* MENGGUNAKAN PRINSIP *REPRESENTATIONAL STATE TRANSFER*

SKRIPSI

KONSENTRASI TEKNIK INFORMATIKA DAN KOMPUTER

Diajukan untuk memenuhi persyaratan
memperoleh gelar Sarjana Teknik

Disusun Oleh:
AGI PUTRA KHARISMA
NIM. 0410630003-63

Telah Diperiksa dan Disetujui
Dosen Pembimbing:

Raden Arief Setyawan, ST., MT.	Herman Tolle, ST.,MT.
NIP. 132 231 713	NIP. 132 283 206

UNIVERSITAS BRAWIJAYA

repository.ub.ac.id

LEMBAR PENGESAHAN

PERANCANGAN DAN PEMBUATAN SITUS *SOCIAL NETWORKING* MENGGUNAKAN PRINSIP *REPRESENTATIONAL STATE TRANSFER*

SKRIPSI

KONSENTRASI TEKNIK INFORMATIKA DAN KOMPUTER

Diajukan untuk memenuhi persyaratan
memperoleh gelar Sarjana Teknik

Disusun Oleh:

AGI PUTRA KHARISMA
NIM. 0410630003-63

Skripsi ini telah diuji dan dinyatakan lulus pada
tanggal 28 Januari 2009

Majelis Penguji:

Ir. Heru Nurwarsito, M.Kom.	Arief Andy Soebroto, ST., M.Kom.
NIP. 131 879 033	NIP. 132 231 567

Suprapto, ST., MT.
NIP. 132 149 320

Mengetahui:
Ketua Jurusan Teknik Elektro

Ir. Heru Nurwarsito, M.Kom.
NIP. 131 879 033

UNIVERSITAS BRAWIJAYA

repository.ub.ac.id

KATA PENGANTAR

Alhamdulillah, puji syukur kehadirat Allah SWT atas segala rahmat dan hidayah-Nya, sehingga skripsi ini dapat kami selesaikan dengan baik, dengan judul "Perancangan dan Pembuatan Situs *Social Networking* Menggunakan Prinsip *Representational State Transfer*".

Skripsi ini disusun untuk memenuhi sebagian persyaratan kelulusa dalam menyelesaikan pendidikan dan memperoleh gelar Sarjana Teknik di Fakultas Teknik Jurusan Teknik Elektro Universitas Brawijaya Malang.

Penulis menyadari bahwa dalam menyelesaikan skripsi ini banyak memperoleh dorongan serta bantuan dari berbagai pihak, maka dalam kesempatan ini penulis mengucapkan terima kasih dan penghargaan kepada:

1. Orang tua ku yang selalu memberikan dorongan dan semangat.
2. Heru Nurwarsito, Ir., M.Kom. dan Rudy Yuwono, ST., M.Sc. selaku Ketua Jurusan dan Sekretaris Jurusan Teknik Elektro Fakultas Teknik Universitas Brawijaya Malang.
3. Arief Andy Soebroto, ST., M.Kom. selaku KKDK Konsetrasi Teknik Informatika dan Komputer yang telah memberikan bimbingan sejak masa-masa perkuliahan dan selama pengerjaan proposal skripsi.
4. Raden Arief Setyawan, ST., MT. dan Herman Tolle, ST., MT. selaku dosen pembimbing yang telah banyak memberikan bimbingan, ide, motivasi, pengarahan, saran dan masukan selama pengerjaan skripsi ini.
5. Seluruh Staf Pengajar dan Karyawan Jurusan Teknik Elektro Universitas Brawijaya Malang.
6. Rekan-rekan Asisten Laboratorium Komputer dan Informatika atas segala bantuannya.
7. Rekan-rekan Paket E dan seluruh angkatan 2004 serta teman-teman lainnya yang tidak mungkin disebutkan satu-persatu.

UNIVERSITAS BRAWIJAYA

Dalam penulisan skripsi ini penulis menyadari bahwa masih banyak kekurangan. Untuk itu, kritikan dan saran dari pembaca sangat penulis harapkan untuk perbaikan skripsi ini dan sebagai masukan. Dan akhirnya penulis berharap semoga skripsi ini dapat memberikan manfaat.

Malang, Januari 2009

Penulis

repository.ub.ac.id

ABSTRAK

Agi Putra Kharisma, Jurusan Teknik Elektro, Fakultas Teknik Universitas Brawijaya, Januari 2008, Perancangan dan Pembuatan Situs *Social Networking* Menggunakan Prinsip *Representational State Transfer*, Dosen Pembimbing: Raden Arief Setyawan, ST., MT. dan Herman Tolle, ST., MT.

Lahirnya web 2.0 membuat perkembangan web semakin pesat, salah satunya adalah munculnya situs *social networking*. Web yang ada saat ini memiliki karakter yang berbeda-beda, baik dari segi teknologi maupun arsitektur. Untuk sistem berskala *world wide web*, perbedaan - perbedaan tersebut berpotensi menimbulkan masalah. Masalah tersebut meliputi skalabilitas, kompleksitas, dan *interoperability.* Untuk mengatasi masalah tersebut, muncul *architectural style* bernama *Representational State Transfer* (REST). Skripsi ini bertujuan untuk memberikan penjelasan dan pengertian tentang prinsip REST melalui contoh kasus aplikasi web *social networking*.

Proses perancangan, pembuatan, dan pengujian situs *social networking* serta perbandingan antara REST dan RPC berdasarkan pada studi literatur yang mengkaji mengenai teori: *Social Networking*, *Hypertext Transfer Protocol* (HTTP), *Uniform Resource Identifier*, REST, Ruby On Rails, Ajax, *Relational Database Model*, dan Rekayasa Perangkat Lunak. Setelah itu dilakukan perancangan menggunakan *object-oriented analysis and design* dan diimplementasikan secara *iterative and incremental* dan *behaviour-driven development*. Pengujian dilakukan menggunakan metode *behaviour driven* secara *black box* untuk mengetahui kesesuaian analisis kebutuhan dengan implementasi aplikasi.

Situs *social networking* berhasil dirancang secara *resource-oriented*, menghasilkan 10 *resources*. Masing – masing *resource* dapat diekspos menggunakan berbagai HTTP *method* yang berbeda sesuai kebutuhan. Situs *social networking* berhasil diimplementasikan dengan mematuhi prinsip REST, yaitu *addressability*, *statelessness*, *connectedness*, dan *uniform interface*. Situs *social networking* dapat ditingkatkan kemampuannya (*scalability*) dengan cara menambah jumlah *web server* menggunakan *cluster* yang berisi kumpulan *web server* yang independen. Implementasi situs *social networking* menggunakan prinsip REST menghasilkan sistem yang lebih sederhana dibanding RPC. Jumlah varian *methods* dalam *interface* yang dihasilkan REST lebih sedikit dibanding RPC. *Methods* yang terdapat di dalam *interface* tersebut adalah GET, POST, PUT, dan DELETE.

Kata kunci: *social networking*, web 2.0, *representational state transfer*, arsitektur, *resource-oriented*

UNIVERSITAS BRAWIJAYA

DAFTAR ISI

KATA PENGANTAR **i**
ABSTRAK **iii**
DAFTAR ISI **iv**
DAFTAR GAMBAR **ix**
DAFTAR TABEL **xii**

BAB I. PENDAHULUAN **1**
1.1 Latar Belakang 1
1.2 Rumusan Masalah 2
1.3 Ruang Lingkup 2
1.4 Tujuan 3
1.5 Manfaat 3
1.6 Sistematika Penulisan 3
BAB II. KAJIAN PUSTAKA **5**
2.1 *Social Networking* 5
2.2 *Hyper Text Transfer Protocol* (HTTP) 5
2.3 *Universal Resource Identifier* (URI) 7
2.4 *Representational State Transfer* (REST) 8
2.4.1 *Resource Oriented Architecture* (ROA) 8
2.5 Ruby On Rails (RoR) 11
2.5.1 *Model View Controller* (MVC) 12
2.5.2 *Object Relational Mapping* (ORM) 13
2.6 Ajax 13
2.7 *Relational Database Model* 15
2.8 Rekyasa Perangkat Lunak 17
2.8.1 Metode *Iterative and Incremental* 20
2.8.2 *Behaviour-Driven Development* (BDD) 21
BAB III. METODE PENELITIAN **23**
3.1 Studi Literatur 23

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

3.2 Definisi Operasional 24
3.3 Perancangan 25
3.4 Implementasi 25
3.5 Pengujian dan Analisis 25
3.6 Pengambilan Kesimpulan dan Saran 26
BAB IV. PERANCANGAN **27**
4.1 Analisis Kebutuhan (*Requirement Analysis*) 27
4.1.1 Identifikasi Aktor 27
4.1.2 Daftar Kebutuhan 28
4.1.3 *Use Case Diagram* 30
4.1.4 *Activity Diagram* 32
4.1.4.1 Daftar 32
4.1.4.2 Melihat Profil Anggota 33
4.1.4.3 *Login* 34
4.1.4.4 *Logout* 34
4.1.4.5 Mengubah Profil 35
4.1.4.6 Mengubah *Avatar* 36
4.1.4.7 Menulis *Posting Blog* 37
4.1.4.8 Mengubah *Posting Blog* 38
4.1.4.9 Menghapus *Posting Blog* 39
4.1.4.10 Menulis Komentar *Posting Blog* 40
4.1.4.11 Menghapus Komentar *Posting Blog* 41
4.1.4.12 Menambah Teman 42
4.1.4.13 Menghapus Teman 43
4.1.4.14 Mencari Anggota 43
4.1.4.15 Menelusuri Anggota 44
4.1.4.16 Membatalkan Akun 44
4.1.4.17 *Login* (Administrator) 45
4.1.4.18 Menghapus Akun (Administrator) 45
4.1.4.5 Logout (Administrator) 46
4.1.5 Perancangan *User Interface* 46
4.2 Perancangan Perangkat Lunak 47

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

4.2.1 Perancangan *Resources* 47
4.2.2 Perancangan Klas (*Class Diagram*) 54
4.2.3 Perancangan Basis Data 56
4.2.3.1 Diagram ER 56
4.2.3.2 *Conceptual Modeling* 57
4.2.3.3 *Logical Modeling* 57
4.2.3.4 *Physical Modeling* 58
4.3 Perancangan *Plan* Sistem 59
BAB V. IMPLEMENTASI **60**
5.1 Spesifikasi Sistem 60
5.1.1 Spesifikasi Perangkat Keras 60
5.1.2 Spesifikasi Perangkat Lunak 60
5.2 Implementasi *Class* Pada *File* Program 61
5.3 Implementasi *Uniform Interface* Pada *Resource* 62
5.4 Implementasi *Object Relational Mapping* (ORM) Pada Basis Data 64
5.4.1 Implementasi Relasi Tabel *Users* 64
5.4.2 Implementasi Relasi Tabel *Profiles* 64
5.4.3 Implementasi Relasi Tabel *Posts* 65
5.4.4 Implementasi Relasi Tabel *Comments* 65
5.4.5 Implementasi Relasi Tabel *Friendships* 66
5.5 Implementasi Algoritma 66
5.5.1 Algoritma *Request Friendship* 66
5.5.2 Algoritma *Accept Friendship* 67
5.5.3 Algoritma *Breakup Friendship* 68
5.6 Implementasi Antarmuka 69
5.6.1 Implementasi Antarmuka Halaman Muka 69
5.6.2 Implementasi Antarmuka Halaman Daftar 70
5.6.3 Implementasi Antarmuka Halaman Profil Anggota 70
5.6.4 Implementasi Antarmuka Halaman *Login* 71
5.6.5 Implementasi Antarmuka Halaman *Hub* 72
5.6.6 Implementasi Antarmuka Halaman Ubah Profil 73
5.6.7 Implementasi Antarmuka Halaman Pertemanan 74

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

5.6.8 Implementasi Antarmuka Halaman Blog 75
5.6.9 Implementasi Antarmuka Halaman *Posting* 76
5.6.10 Implementasi Antarmuka Halaman Membuat *Posting* Baru 77
5.6.11 Implementasi Halaman Ubah Posting 78
5.6.12 Implementasi Halaman Penelusuran Anggota 79
5.6.13 Implementasi Halaman Pencarian Anggota 80
5.7 Kendala Dalam Implementasi 80
BAB VI PENGUJIAN DAN ANALISIS 81
6.1 Pengujian 81
6.1.1 Pengujian Unit 81
6.1.1.1 Pengujian *Class User* 81
6.1.1.2 Pengujian *Class Profile* 82
6.1.1.3 Pengujian *Class Posts* 82
6.1.1.4 Pengujian *Class Comment* 82
6.1.1.5 Pengujian *Class Friendship* 82
6.1.1.6 Pengujian *Class Avatar* 82
6.1.2 Pengujian Fungsional 83
6.1.3 Pengujian Performa dan Skalabilitas Sistem 97
6.2 Analisis 103
6.2.1 Analisis Prinsip *Addressability* 104
6.2.2 Analisis Prinsip *Statelessness* 104
6.2.3 Analisis Prinsip *Connectedness* 104
6.2.4 Analisis Prinsip *Uniform Interface* 104
6.2.4.1 HTTP - GET 105
6.2.4.2 HTTP – POST 106
6.2.4.3 HTTP – PUT 107
6.2.4.3 HTTP – DELETE 108
6.2.5 Analisis Perbandingan Antara REST *Style* Dengan RPC *Style* 108
BAB VII. PENUTUP 111
7.1 Kesimpulan 111

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

7.2 Saran 111

DAFTAR PUSTAKA 113

LAMPIRAN 115

DAFTAR GAMBAR

Gambar 2.1	Mesin Pencari yang *Stateless*	9
Gambar 2.2	Mesin Pencari yang *Stateful*	9
Gambar 2.3	Contoh Daftar Fungsi Dalam Suatu *Service*	10
Gambar 2.4	Implementasi *Uniform Interface* Pada Suatu *Service*	11
Gambar 2.5	Diagram Blok MVC Pada Ruby on Rails	12
Gambar 2.6	*Request/Response* Web Tradisional	14
Gambar 2.7	*Request/Response* Ajax	14
Gambar 2.8	Contoh Relasi *One-to-One*	15
Gambar 2.9	Contoh Relasi *One-to-Many*	16
Gambar 2.10	Contoh Relasi *Many-to-Many*	17
Gambar 2.11	*Iterative and Incremental Development*	20
Gambar 2.12	Contoh *Stories*	22
Gambar 3.1	Siklus Dasar *Test-Driven Development*	25
Gambar 4.1	*Use Case Diagram* – Pengunjung dan Anggota	31
Gambar 4.2	*Use Case Diagram* – Administrator	32
Gambar 4.3	*Activity Diagram* – Daftar	33
Gambar 4.4	*Activity Diagram* – Melihat Profil Anggota	33
Gambar 4.5	*Activity Diagram* – Login	34
Gambar 4.6	*Activity Diagram* – Logout	35
Gambar 4.7	*Activity Diagram* – Mengubah Profil	35
Gambar 4.8	*Activity Diagram* – Mengubah *Avatar*	36
Gambar 4.9	*Activity Diagram* – Menulis *Posting* Blog	37
Gambar 4.10	*Activity Diagram* – Mengubah *Posting* Blog	38
Gambar 4.11	*Activity Diagram* – Menghapus *Posting* Blog	39
Gambar 4.12	*Activity Diagram* – Menulis Komentar *Posting* Blog	40
Gambar 4.13	*Activity Diagram* – Menghapus Komentar *Posting* Blog	41
Gambar 4.14	*Activity Diagram* – Menambah Teman	42
Gambar 4.15	*Activity Diagram* – Menghapus Teman	43
Gambar 4.16	*Activity Diagram* – Mencari Anggota	43
Gambar 4.17	*Activity Diagram* – Menelusuri Anggota	44

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

Gambar 4.18	*Activity Diagram* – Membatalkan Akun	44
Gambar 4.19	*Activity Diagram* – Login (Administrator)	45
Gambar 4.20	*Activity Diagram* – Menghapus Akun (Administrator)	46
Gambar 4.21	*Activity Diagram* – Logout (Administrator)	46
Gambar 4.22	*User Interface*	47
Gambar 4.23	*Class Diagram – Controllers*	54
Gambar 4.24	*Class Diagram – Models*	55
Gambar 4.25	Diagram ER Sistem	56
Gambar 4.26	*Conceptual Diagram* Sistem	57
Gambar 4.27	*Logical Diagram* Sistem	58
Gambar 4.28	*Physical Diagram* Sistem	59
Gambar 4.29	Diagram *Plan* Sistem	59
Gambar 5.1	Implementasi *Class Model – User*	64
Gambar 5.2	Implementasi *Class Model – Profile*	65
Gambar 5.3	Implementasi *Class Model – Post*	65
Gambar 5.4	Implementasi *Class Model – Comment*	65
Gambar 5.5	Implementasi *Class Model – Friendship*	66
Gambar 5.6	Algoritma *Request Friendship*	67
Gambar 5.7	Algoritma *Accept Friendship*	68
Gambar 5.8	Algoritma *Breakup Friendship*	68
Gambar 5.9	Antarmuka Halaman Muka	69
Gambar 5.10	Antarmuka Halaman Daftar	70
Gambar 5.11	Antarmuka Halaman Profil Anggota	71
Gambar 5.12	Antarmuka Halaman *Login*	71
Gambar 5.13	Antarmuka Halaman *Hub*	72
Gambar 5.14	Antarmuka Halaman *Ubah Profil*	73
Gambar 5.15	Antarmuka Halaman Pertemanan	74
Gambar 5.16	Antarmuka Halaman Blog	75
Gambar 5.17	Antarmuka Halaman *Posting*	76
Gambar 5.18	Antarmuka Halaman Membuat *Posting* Baru	77
Gambar 5.19	Antarmuka Halaman Ubah *Posting*	78
Gambar 5.20	Antarmuka Penelusuran Anggota	79

Gambar 5.21 Antarmuka Halaman Pencarian Anggota 80

Gambar 6.1 Hasil Pengujian Performa Sistem Menggunakan 2 Mongrel *Servers* Untuk Menangani 1 *Concurrent Users* 98

Gambar 6.2 Hasil Pengujian Performa Sistem Menggunakan 2 Mongrel *Servers* Untuk Menangani 2 *Concurrent Users* 99

Gambar 6.3 Hasil Pengujian Performa Sistem Menggunakan 2 Mongrel *Servers* Untuk Menangani 5 *Concurrent Users* 99

Gambar 6.4 Hasil Pengujian Performa Sistem Menggunakan 2 Mongrel *Servers* Untuk Menangani 50 *Concurrent Users* 100

Gambar 6.5 Hasil Pengujian Performa Sistem Menggunakan 5 Mongrel *Servers* Untuk Menangani 1 *Concurrent Users* 100

Gambar 6.6 Hasil Pengujian Performa Sistem Menggunakan 5 Mongrel *Servers* Untuk Menangani 2 *Concurrent Users* 101

Gambar 6.7 Hasil Pengujian Performa Sistem Menggunakan 5 Mongrel *Servers* Untuk Menangani 5 *Concurrent Users* 101

Gambar 6.8 Hasil Pengujian Performa Sistem Menggunakan 5 Mongrel *Servers* Untuk Menangani 50 *Concurrent Users* 102

Gambar 6.9 Perbandingan Performa *Cluster* Yang Berisi 2 Mongrel *Servers* dan 5 Mongrel *Servers* Dalam Menangani 1000 *Request* 103

DAFTAR TABEL

Tabel 1.1	Operasi Dasar REST	2
Tabel 2.1	Contoh Pertanyaan Kunci	19
Tabel 4.1	Deskripsi Aktor	28
Tabel 4.2	Daftar Kebutuhan Fungsional dan Non Fungsional	28
Tabel 4.3	*Resource Site*	48
Tabel 4.4	*Resource Users*	48
Tabel 4.5	*Resource Session*	49
Tabel 4.6	*Resource Profile*	49
Tabel 4.7	*Resource Avatar*	50
Tabel 4.8	*Resource Friendships*	50
Tabel 4.9	*Resource Posts*	51
Tabel 4.10	*Resource Comments*	52
Tabel 4.11	*Resource Hub*	53
Tabel 4.12	*Resource Community*	53
Tabel 5.1	Spesifikasi Perangkat Keras Komputer	60
Tabel 5.2	Spesifikasi Perangkat Lunak Komputer	61
Tabel 5.3	Implementasi *Class* Pada Kode Program *.rb	61
Tabel 5.4	Hubungan Antara HTTP *Method* Dengan *Controller method* Pada *Framework* Ruby On Rails	62
Tabel 5.5	Implementasi *Resource* dan HTTP *Method*	62
Tabel 6.1	Pengujian *Class SiteController*	83
Tabel 6.2	Pengujian *Class UsersController*	83
Tabel 6.3	Pengujian *Class ProfilesController*	85
Tabel 6.4	Pengujian *Class SessionsController*	87
Tabel 6.5	Pengujian *Class HubsController*	88
Tabel 6.6	Pengujian *Class PostsController*	88
Tabel 6.7	Pengujian *Class CommentsController*	92
Tabel 6.8	Pengujian *Class FriendshipsController*	93
Tabel 6.9	Pengujian *Class AvatarController*	95
Tabel 6.10	Pengujian *Class CommunityController*	96

Tabel 6.11	Spesifikasi Perangkat Keras Untuk Pengujian	97
Tabel 6.12	Daftar Pengujian Performa dan Skalabilitas	98
Tabel 6.13	Daftar *Resource* Yang Diakses Menggunakan *Method* GET	105
Tabel 6.14	Daftar *Resource* Yang Diekspos Menggunakan *Method* POST	106
Tabel 6.15	Daftar *Resource* Yang Diekspos Menggunakan *Method* PUT	107
Tabel 6.16	Daftar *Resource* Yang Diekspos Menggunakan *Method* DELETE	108
Tabel 6.17	Perbedaan Antara REST *Style* Dengan RPC *Style*	109
Tabel 6.18	Contoh Implementasi Menggunakan REST *Style*	109
Tabel 6.19	Contoh Implementasi Menggunakan RPC *Style*	110

BAB I
PENDAHULUAN

1.1 Latar Belakang

Lahirnya web 2.0[1] berpengaruh terhadap perkembangan web saat ini. Sejak era web 2.0, para pengguna menjadi lebih aktif berpartisipasi dalam internet, khususnya dalam penggunaan media web. Partisipasi tersebut dapat disalurkan melalui *blog*, *social networking*, wiki, dan sebagainya. Hal tersebut tidak lepas dari salah satu ciri web 2.0, yaitu web dengan unsur sosial, sehingga *social networking* memegang peran yang penting dalam web 2.0.

Web yang ada saat ini memiliki karakter yang berbeda-beda, baik dari segi teknologi maupun arsitektur. Untuk sistem berskala *world wide web*, perbedaan - perbedaan tersebut berpotensi menimbulkan masalah. Masalah tersebut meliputi skalabilitas, kompleksitas, dan *interoperability*. Arsitektur RPC (*Remote Procedural Call*) yang cukup populer diimplementasikan pada berbagai *web service* ternyata memiliki masalah pada kompleksitas dan *interoperability* [RIC-07:xvi]. Merujuk dari disertasi Roy Thomas Fielding yang berjudul *"Architectural Styles and the Design of Network-based Software Architectures*" maka lahir suatu model arsitektur (*architectural style*) baru bernama *Representational State Transfer* (REST) [RIC-07:xvi]. REST dirancang secara khusus untuk suatu sistem perangkat lunak terdistribusi yang berbasis jaringan komputer, contohntya internet. Hal ini membuat REST cocok diimplementasikan pada berbagai jenis aplikasi berbasis web pada internet, termasuk *social networking*. Dengan menerapkan prinsip – prinsip REST, situs *social networking* menjadi lebih sederhana, *scalable*, dan memiliki *interoperability* yang baik.

Dengan prinsip REST, situs *social networking* yang dibuat pada skripsi ini dimodelkan menjadi empat operasi dasar saja, yaitu CRUD (*Create, Read, Update, Delete*). Karena hanya terdapat empat jenis operasi, maka arsitektur situs *social networking* yang dihasilkan menjadi sederhana. Masing – masing operasi tersebut memiliki padanan pada operasi basis data maupun HTTP *methods,*

1 Menurut O'Reilly *Network*, Web 2.0 merupakan *design pattern* dalam pembuatan web yang memberikan kesempatan pengguna untuk berpartisipasi aktif, sehingga memungkinkan layanan web menjadi semakin baik seiring dengan pertambahan jumlah pengguna. [ORE-05]

sehingga terdapat suatu keseragaman, seperti ditunjukkan dalam tabel 1.1.

Tabel 1.1 Operasi Dasar REST

Verb	***HTTP Method***	***Database***
Create	POST	INSERT
Read	GET	SELECT
Update	PUT	UPDATE
Delete	DELETE	DELETE

Sumber: [POL-07]

Keseragaman tersebut dapat meningkatkan *interoperability* karena berbagai jenis sistem berinteraksi menggunakan *interface* yang sama dan mudah diprediksi, yaitu REST (dalam hal ini HTTP *method*).

1.2 Rumusan Masalah

Berdasarkan latar belakang, maka dapat ditetapkan rumusan masalah sebagai berikut:

1. Merancang situs *social networking* secara *resource oriented*.
2. Mengimplementasikan prinsip REST pada situs *social networking*.
3. Menguji situs social networking secara *behaviour driven.*
4. Membandingkan REST dengan RPC.

1.3 Ruang Lingkup

Mengacu pada rumusan masalah, maka ruang lingkup dari penulisan skripsi ini meliputi:

1. Parameter yang digunakan untuk membandingkan REST dengan RPC adalah variasi *methods* yang terdapat dalam *interface*.
2. *Web framework* yang digunakan adalah Ruby on Rails 2.
3. *Web server* yang digunakan adalah Mongrel.
4. *Database Management System* (DBMS) yang digunakan adalah MySQL 5.

1.4 Tujuan

Tujuan penulisan skripsi ini dibagi menjadi dua, yaitu:

1. Tujuan Umum

   Memberikan penjelasan dan pengertian tentang prinsip *Representational State Transfer* (REST) melalui contoh kasus aplikasi web *social networking*.

2. Tujuan Khusus

   a. Merancang situs *social networking* secara *resource oriented.*

   b. Mengimplementasikan prinsip REST pada situs *social networking*.

   c. Membandingkan REST dengan RPC.

1.5 Manfaat

Manfaat yang bisa didapat dari skripsi ini adalah:

1. Memberi solusi pembuatan aplikasi berbasis web secara baik dan benar menggunakan prinsip *Representational State Transfer* (REST).
2. Aplikasi web *social networking* yang dibuat dapat digunakan sebagai media komunikasi.

1.6 Sistematika Penulisan

BAB I Pendahuluan

Memuat latar belakang, rumusan masalah, ruang lingkup, tujuan, manfaat, dan sistematika penulisan.

BAB II Kajian Pustaka

Membahas kajian pustaka dan dasar teori yang digunakan untuk menunjang penulisan skripsi ini, meliputi *social networking*, *Hyper Text Transfer Protocol* (HTTP), *Uniform Resource Identifier* (URI), *Representational State Transfer* (REST), Ruby On Rails, Ajax, Rekayasa Perangkat Lunak.

BAB III Metode Penelitian

Membahas metode yang digunakan dalam penulisan yang terdiri dari studi litertur, perancangan perangkat lunak, implementasi perangkat lunak, pengujian dan analisis, serta pengambilan kesimpulan dan saran.

BAB IV Perancangan

Membahas analisis kebutuhan dan perancangan situs *social networking* menggunakan prinsip *Representational State Transfer.*

BAB V Implementasi

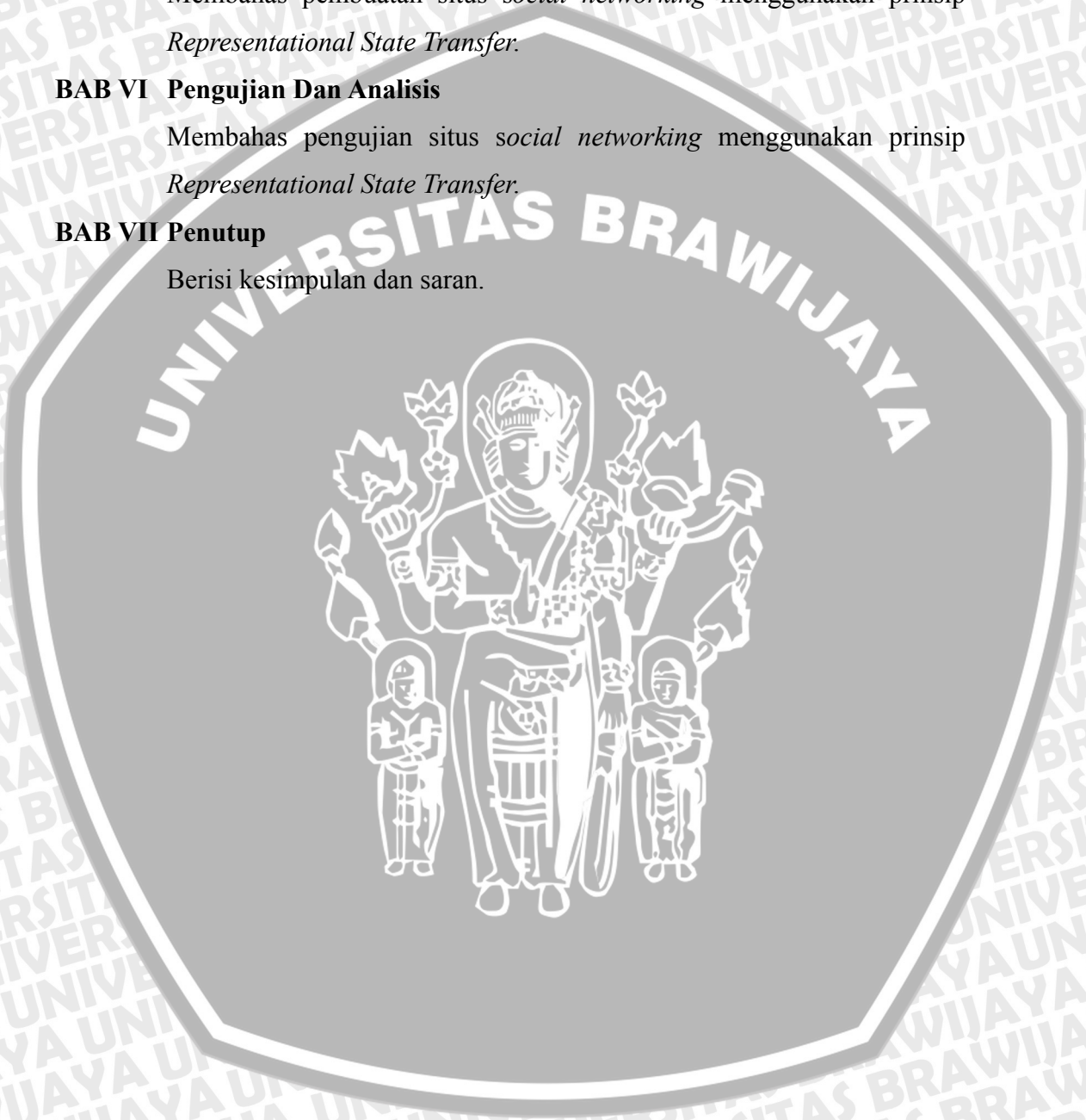
Membahas pembuatan situs s*ocial networking* menggunakan prinsip *Representational State Transfer.*

BAB VI Pengujian Dan Analisis

Membahas pengujian situs s*ocial networking* menggunakan prinsip *Representational State Transfer.*

BAB VII Penutup

Berisi kesimpulan dan saran.



BAB II
KAJIAN PUSTAKA

2.1 *Social Networking*

Social networking atau jejaring sosial merupakan aplikasi berbasis web yang memungkinkan pengguna untuk saling berinteraksi dan berbagi (*share*) satu sama lain. Umumnya situs ini digunakan untuk berbagi pengetahuan, aktivitas, dan kegemaran. Kegiatan tersebut dapat dilakukan melalui berbagai cara, diantaranya melalui *chat*, berkirim pesan (*messaging*), email, video, *blogging*, forum diskusi, dan sebagainya. Web dan internet sangat cocok digunakan sebagai media *social networking* karena mencakup jaringan yang sangat luas.

Pada awalnya situs *social networking* memfokuskan diri pada suatu aktivitas maupun kegemaran tertentu, misalnya Classmates.com yang hanya memfokuskan diri pada jejaring sosial berdasarkan sekolah tertentu. Kini muncul situs – situs *social networking* yang bersifat lebih umum dan fleksibel, seperti Friendster.com, Hi5.com, Facebook.com, Myspace.com, dan sebagainya. Selain itu, platform untuk *social networking* yang saat ini sedang naik daun adalah *blogging service,* seperti Blogspot.com, Wordpress.com, LiveJournal.com, dan sebagainya.

2.2 *Hyper Text Transfer Protocol* (HTTP)

HTTP merupakan protokol utama yang berada pada level aplikasi untuk komunikasi antar komponen web. Protokol ini merupakan satu-satunya protokol yang didesain secara spesifik sebagai media transfer representasi suatu *resource*[2]. Pengembangan protokol ini dikoordinasi oleh *Word Wide Web Consortium* (W3C) dan *Internet Engineering Task Force* (IETF). Informasi mengenai HTTP/1.1 terdapat pada RFC 2616. Saat ini versi HTTP yang umum digunakan adalah HTTP/1.1.

2 *Resource* adalah segala sesuatu yang dianggap cukup penting untuk dijadikan suatu referensi. *Resource* dapat berupa fisik maupun konsep. Masing – masing *resource* minimal memiliki satu URI (*Universal Resource Identifier*). [RIC-07:81]

HTTP memiliki berbagai macam *request method* atau yang lebih dikenal dengan istiah HTTP *method*, antara lain:

- HEAD

  Meminta respon berupa *meta-information* saja, tanpa *response body*.

- GET

  Meminta representasi (*meta-information* dan *response body*) dari suatu *resource* tertentu yang spesifik.

- POST

  Mengirimkan data untuk diproses lebih lanjut ke suatu *resource* tertentu. Data disertakan dalam *body* dari *request*. Hal ini akan menghasilkan suatu *resource* baru maupun pembaharuan (*update*) terhadap *resource* yang lama atau keduanya sekaligus.

- PUT

  Mengirimkan suatu representasi ke *resource* tertentu yang spesifik.

- DELETE

  Menghapus *resource* tertentu yang spesifik.

- TRACE

  Mencetak kembali *request* yang diterima, sehingga *client* dapat melihat apa yang berubah pada *request* selama perjalanan.

- OPTION

  Mengetahui HTTP *methods* yang didukung oleh suatu *server* untuk URI (*Universal Resource Identifier*) tertentu.

- CONNECT

  Mengubah koneksi HTTP *request* menjadi terowongan (*tunnel*) TCP/IP yang transparan. Biasanya digunakan untk memfasilitasi SSL-*encrypted communication* (HTTPS) melalui HTTP *proxy* yang tidak dienkripsi.

HTTP *methods* dikelompokkan menjadi dua, yaitu *safe method* dan *idempotent method*. *Safe method* adalah method yang dinilai aman karena method tersebut hanya digunakan untuk menerima informasi saja dan selayaknya tidak boleh mengubah *state* pada *server*. Yang termasuk dalam *safe methods* adalah HEAD, GET, OPTIONS, dan TRACE. *Idempotent method* memiliki arti bahwa berapapun banyaknya *request* akan mengakibatkan hasil yang sama dengan hanya

satu *request* saja. Secara matematis, dapat diilustrasikan dengan contoh persamaan:

1 x 0 x 0 x 0 x 0 x 0 = 1 x 0 = 0.

Yang termasuk dalam *idempotent methods* adalah PUT, DELETE, HEAD, GET, OPTIONS, dan TRACE.

HTTP merupakan protokol yang *stateless,* dimana *host* tidak perlu menyimpan informasi mengenai *application state* disela – sela tiap *request*. Namun ada kalanya dibutuhkan untuk mempertahankan *application state*, hal ini dapat dilakukan dengan melibatkan pengiriman dan penerimaan *cookies*. Dengan demikian protokol HTTP yang *stateless* dapat disimulasikan menjadi protokol *stateful*. *Cookies* tersebut berupa *string* unik yang menunjukkan suatu *session*. Ada metode lain yang umum digunakan, yaitu *server side session*, *hidden variables*, dan URI berparameter (contoh: /buku/1/?halaman=3).

2.3 *Universal Resource Identifier* (URI)

URI merupakan suatu *string* berisi kumpulan karakter yang digunakan untuk mengidentifikasi suatu *resource*. Informasi mengenai URI terdapat pada RFC 2396. Identifikasi tersebut dapat berupa nama, alamat atau keduanya. URI dapat diklasifikasikan menjadi URN (*Uniform Resource Name*) dan URL (*Uniform Resource Locator*). URI pada web memiliki struktur yang umum, contohnya:

Scheme	***Authority***	***path***	***query***	***fragment***
http ://	xyz.org	/a/b	? id=1	#menu

- *Scheme*

  Bagian ini berisi *string* yang mendefinisikan bagaimana URI ditafsirkan. Untuk protokol HTTP, *scheme* yang digunakan adalah http, untuk FTP, *scheme* yang digunakan adalah ftp, dan sebagainya.

- *Authority*

  Bagian ini memiliki struktur: *userinfo@host:port*. Bagian *userinfo* jarang digunakan dan bagian *port* umumnya tidak diperlukan karena sebagian besar telah menggunakan *port* 80, yang merupakan *default port* untuk HTTP.

- *Path*

  Bagian ini terlihat mirip dengan susunan hirarki *file* pada komputer, namun pada penerapannya, *path* tidak harus menunjukkan suatu hirarki. Contohnya pada web *social bookmarking* del.icio.us, memiliki URI sbb:

  http://del.icio.us/ahmad/musik

  Bagian pertama *path* menunjukkan *user* (ahmad), sedangkan bagian kedua *path* (musik) menunjukkan *tag*. Secara keseluruhan, URI ini mengidentifikasikan semua hal yang dilabeli (*tagged*) oleh Ahmad sebagai musik.

- *Query*

  Bagian ini digunakan untuk data atau parameter yang tidak memiliki hirarki atau berada pada hirarki yang sama.

- *Fragment*

  Bagian ini digunakan untuk mengidentifikasi suatu bagian tertentu dari suatu *resource*.

2.4 *Representational State Transfer* (REST)

REST merupakan *architectural style* yang berisi batasan-batasan terhadap suatu arsitektur. REST pertama kali diungkap dalam desertasi oleh Roy Fielding yang berjudul "*Architectural Styles and the Design of Network Based Software Architectures.*". Secara singkat, REST adalah kumpulan teori tentang bagaimana cara terbaik untuk mengatur dan menyusun sistem *hypermedia*[3] yang terdistribusi, contohnya *World Wide Web*. Dalam REST, web tersusun dari kumpulan *resource* yang membentuk suatu arsitektur bernama *Resource Oriented Architecture*.

2.4.1 *Resource Oriented Architecture* (ROA)

ROA merupakan arsitektur yang digunakan untuk mengubah suatu masalah menjadi aplikasi web yang menggunakan prinsip REST. Dalam ROA terdapat kumpulan *resource* yang terdiri dari nama, lokasi, representasi dari suatu *resource*, dan *link* yang menghubungkan mereka. ROA memiliki ciri-ciri antara

3 *Hypermedia* adalah suatu aplikasi yang menggunakan relasi asosiatif diantara informasi yang terdapat dalam berbagai *media data* (contoh: text, citra, suara, video, dan lain -lain) dengan tujuan untuk memberi kemampuan akses dan manipulasi terhadap data tersebut. [HAL-08]

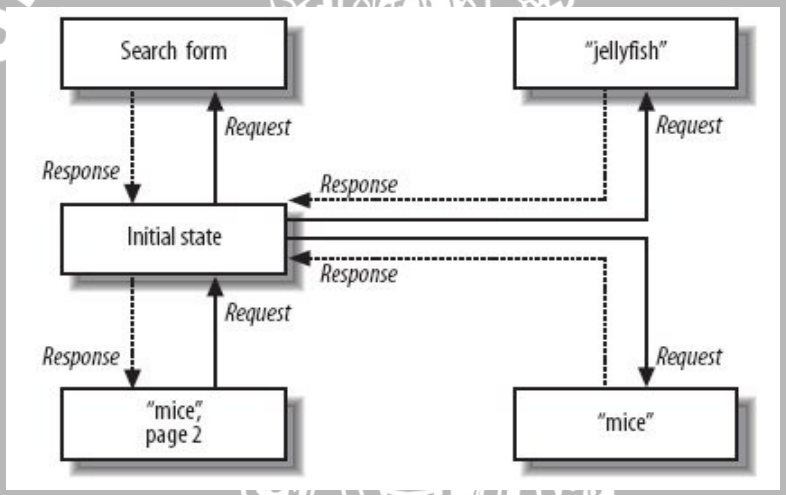
lain:

- *Addressability*

  Setiap *resource* memiliki alamat sendiri, yaitu URI. Dalam prinsip REST, setiap *resource* memiliki representasi yang sama pada masing-masing *client*.

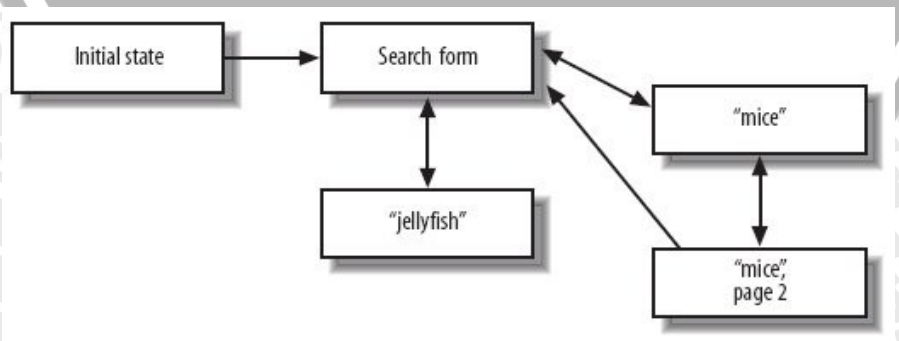
- *Statelessness*

  Statelessness memiliki arti bahwa setiap HTTP *request* terjadi secara terisolasi total, independen, dan tidak saling terhubung satu sama lain. *Server* tidak tahu *state* pada masing-masing *client*, sehingga *client* harus mengirimkan *state* pada saat melakukan *request*. Ada dua macam *state*, yaitu *application state* dan *resource state*. *Aplication state* (yang ideal) berada pada sisi *client*, sedangkan *resource state* berada pada sisi *server*. Berikut ini adalah gambar ilustrasi perbedaan antara *stateless* dan *stateful* dalam bentuk contoh kasus aplikasi mesin pencari:



Gambar 2.1 Mesin Pencari yang *Stateless*

Sumber: [RIC-07:88]



Gambar 2.2 Mesin Pencari yang *Stateful*

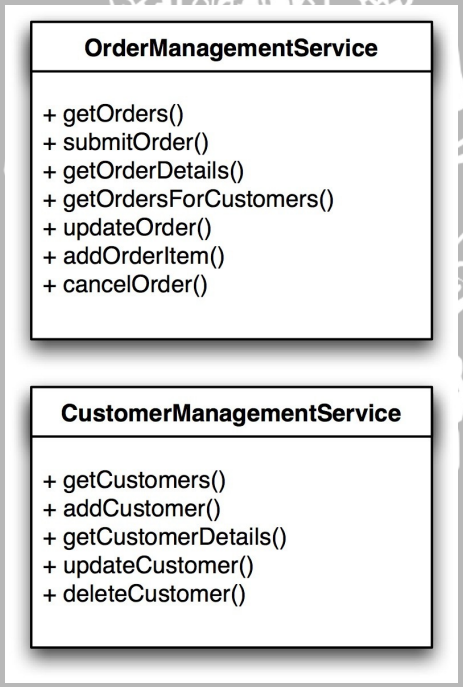
Sumber: [RIC-07:88]

- *Connectedness*

  Server dapat menuntun *client* menuju suatu *application state* tertentu melalui *links* dan *forms* pada suatu representasi *resource*. *Links* dan *forms* itulah yang menghubungkan *resource* yang satu dengan *resource* yang lainnya, membentuk suatu tautan yang saling berhubungan.

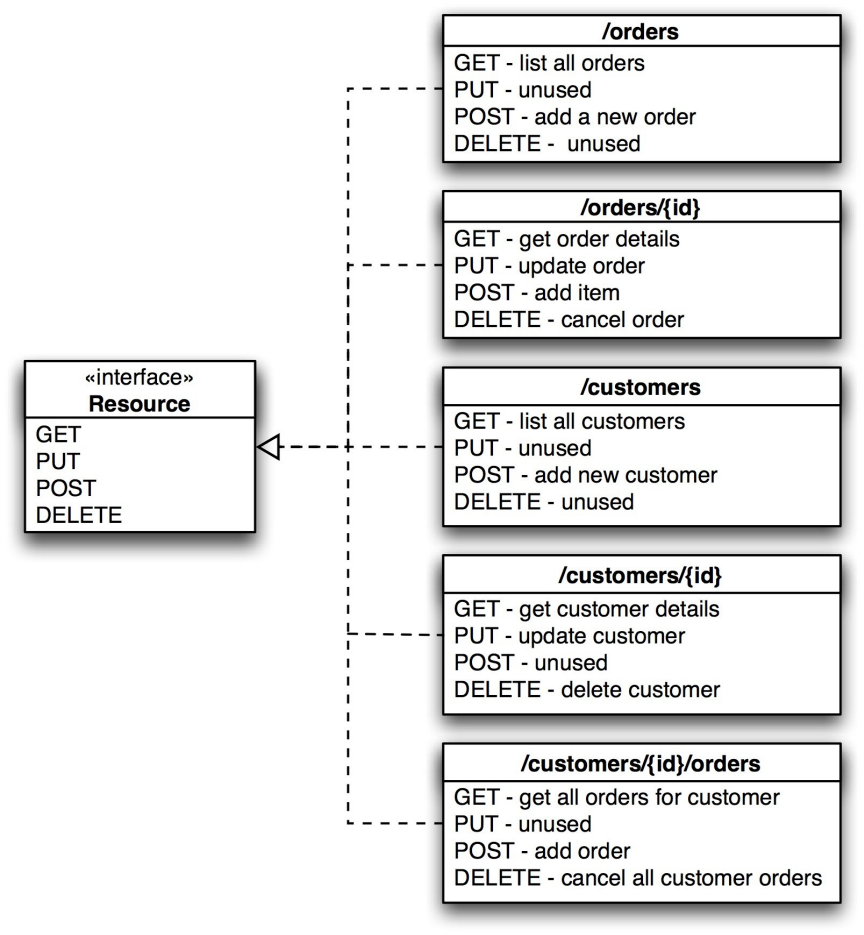
- *Uniform interface*

  Semua interaksi antara *clients* dengan *resources* dilakukan melalui beberapa HTTP *method* yang seragam. Setiap *resource* mendukung beberapa atau semua *method – method* tersebut, dan setiap *method* melakukan hal yang seragam pada setiap *resource* yang mendukungnya. Berikut ini contoh kasus yang digambarkan fungsionalitasnya melalui fungsi-fungsi beserta solusinya menggunakan REST melalui implementasi *uniform interface* dalam bentuk diagram:



Gambar 2.3 Contoh Daftar Fungsi Dalam Suatu *Service*

Sumber: [TIL-07]



Gambar 2.4 Implementasi *Uniform Interface* Pada Suatu *Service*

Sumber: [TIL-07]

Web merupakan salah satu sistem yang mengimplementasikan REST. Dalam hal pengembangan aplikasi web, inti dari REST adalah bagaimana cara mendesain web dengan baik dan benar sehingga web dapat berperilaku sebagaimana mestinya, yaitu:

- Sesuai dengan spesifikasi protokol HTTP.
- Menggunakan arsitektur ROA.

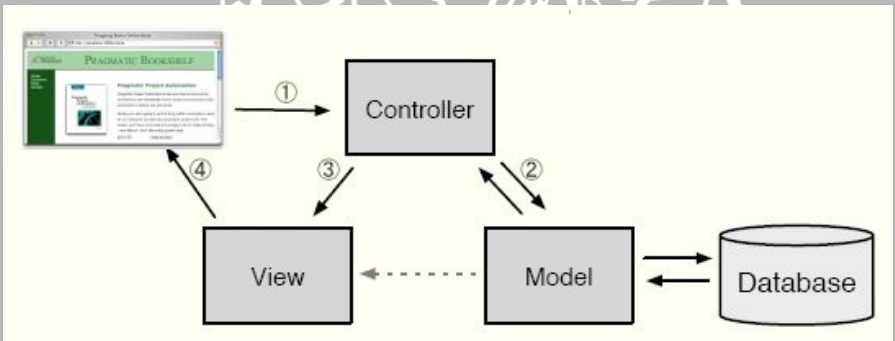
2.5 Ruby On Rails (RoR)

Ruby On Rails (RoR) merupakan *framework* untuk membuat suatu website yang menggunakan basis data pada bagian *back-end*. RoR pertama kali diperkenalkan oleh David Heinemeier Hansson pada tahun 2004. RoR

menggunakan bahasa pemrograman Ruby dan menggunakan arsitektur *Model View Controller*. Bahasa pemrograman Ruby merupakan bahasa pemrograman dinamis (*dynamic programing language*)[4] yang berorientasi objek. RoR memiliki prinsip *Don't Repeat Yourself* (DRY) dan *Convention Over Configuration*. DRY merupakan suatu tindakan yang dilakukan *programmer* untuk meminimalisasi jumlah *code* yang ditulis dengan cara mengurangi pengulangan penulisan *code* yang serupa. *Convention Over Configuration* dimaksudkan untuk meminimalisasi konfigurasi pada framework dengan cara membuat suatu konvensi yang dipatuhi bersama, sehingga *programmer* tidak direpotkan oleh banyaknya file konfigurasi. RoR memanfaatkan kemampuan *metaprogramming*[5] yang terdapat pada bahasa Ruby untuk meningkatkan produktivitas.

2.5.1 *Model View Controller* (MVC)

Arsitektur *Model View Controller* (MVC) yang digunakan oleh Ruby on Rails dapat diilustrasikan seperti gambar 2.5.



Gambar 2.5 Diagram Blok MVC Pada Ruby on Rails

Sumber: [THO-07:23]

Keterangan:

1. *Web browser* mengirim *request*.
2. *Controller* berinteraksi dengan *model*.

4 Bahasa pemrograman dinamis adalah bahasa pemrograman yang programnya dapat mengubah strukturnya pada saat program tersebut dijalankan, seperti suatu fungsi dapat dihilangkan, memungkinkan pembentukan class baru dari suatu object, modul tertentu bisa diperlihatkan. [SRI-05]

5 *Metaprogramming* merupakan teknik pemrograman dimana suatu *code* menulis *code* lainnya atau melakukan introspeksi terhadap dirinya sendiri pada saat *runtime*. [BRA-07:01]

3. *Controller* melakukan pemanggilan (*invoke*) *view.*
4. *View* mengirimkan tampilan halaman selanjutnya ke web *browser.*

Model bertanggungjawab terhadap *state* pada aplikasi, baik *state* yang bersifat sementara maupun permanen yang akan disimpan di tempat lain (biasanya pada basis data). *Model* juga dibuat untuk mengatur *business rules* yang berkenaan dengan suatu data. *View* bertanggungjawab terhadap tampilan (*user interface*) web yang pada umumnya berbasis pada data yang terdapat pada lapisan *model. Controller* berfungsi untuk mengatur jalannya aplikasi dengan cara berinteraksi dengan *model* dan *view.*

2.5.2 *Object Relational Mapping* (ORM)

Lapisan *model* pada Ruby On Rails menggunakan *Object Relational Mapping* (ORM) *design pattern* bernama *Active Record*. ORM berfungsi memetakan tabel pada basis data ke dalam *class*, kolom (*field*) menjadi objek, dan baris (*row*) menjadi atribut dari objek tersebut.

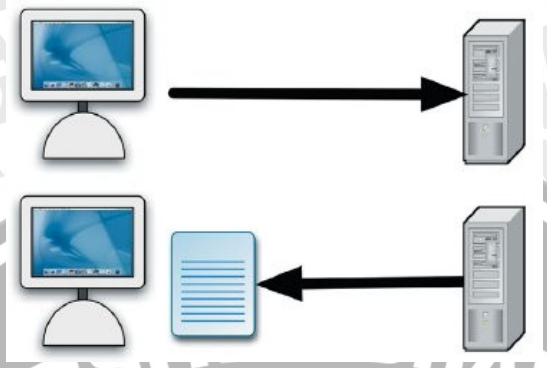
Dengan menggunakan ORM, aplikasi yang dibuat pada RoR dapat berganti *Database Management System* (DBMS) dengan mudah tanpa mengubah kode program. Hal ini dimungkinkan karena programmer tidak harus menggunakan sintaks SQL secara langsung untuk mengakses database. Program mengakses database menggunakan object dari *Active Record*, kemudian *Active Record* menerjemahkannya menjadi sintaks SQL.

2.6 Ajax

Ajax bukan merupakan teknologi, melainkan salah satu metodologi yang digunakan untuk membuat aplikasi web menjadi lebih interaktif [KEI-07:12]. Pada awalnya Ajax merupakan sebuah akronim, yaitu AJAX (*Asynchronous JavaScript and* XML). Namun saat ini data yang digunakan tidak harus menggunakan format XML, tetapi dapat menggunakan format yang lain, seperti JSON (JavaScript *Object Notation*) dan HTML (*Hyper Text Markup Language*).

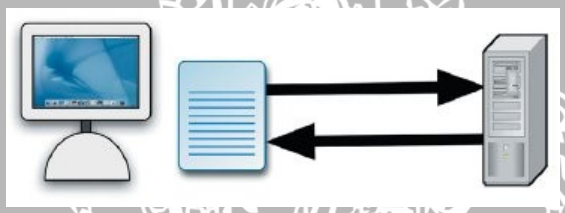
Berbeda dengan web tradisional, dimana *client* mengirim *request* ke *server* dan menerima *response* dari *server* berupa halaman secara keseluruhan, dengan metode Ajax, *response* berupa halaman tersebut dikirim secara diskrit dan

memungkinkan untuk diperbaharui bila ada *request* baru. Perbedaan tersebut ditunjukkan pada gambar 2.6 dan gambar 2.7 berikut:



Gambar 2.6 *Request/Response* Web Tradisional

Sumber: [KEI-07:05]



Gambar 2.7 *Request/Response* Ajax

Sumber: [KEI-07:06]

Teknologi yang umumnya digunakan Ajax adalah:

- HTML (*Hyper Text Markup Language*)

  Merupakan bahasa *markup* untuk memberi struktur *semantic* pada konten web [KEI-07:09].

- CSS (*Cascading Style Sheet*)

  Merupakan teknologi yang digunakan untuk mengatur bagaimana suatu konten web ditampilkan.

- Javascript

  Merupakan bahasa pemrograman yang berjalan pada web *browser* (*client-side programming*) untuk menghasilkan konten dan tampilan yang dinamis.

- DOM (*Document Object Model*)

  Merupakan standar untuk mendeskripsikan struktur suatu dokumen.

- XMLHttpRequest

  Merupakan fitur tambahan dari Javascript yang memungkinkan halaman web berkomunikasi dengan server [KEI-07:08].

- XML (*eXtensible Markup Language*)

  Merupakan bahasa *markup* serbaguna untuk mendeskripsikan segala jenis data [KEI-07:69]. XML merupakan format data.

- JSON (Javascript *Object Notation*)

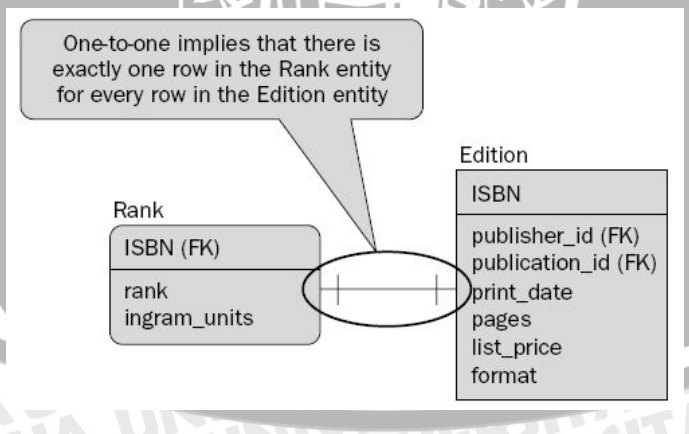
  Merupakan alternatif XML yang lebih ringan. JSON bukan merupakan format data melainkan JSON merupakan Javascript [KEI-07:78].

2.7 *Relational Database Model*

Relational Database Model merupakan salah satu jenis basis data yang paling banyak digunakan. Pada model ini, tabel-tabel dapat dihubungkan satu sama lain sehingga timbul suatu relasi antar tabel. Ada berbagai macam relasi antar tabel, diantaranya:

- *One-to-One*

  Relasi *One-to-One* biasanya digunakan untuk menghilangkan nilai NULL pada baris kolom tabel [POW-06:51]. Satu baris tabel mengacu pada satu baris pada tabel lainnya.

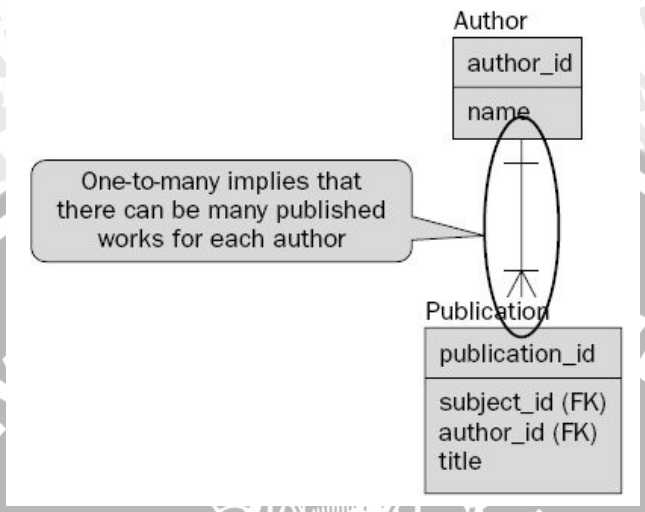


Gambar 2.8 Contoh Relasi *One-to-One*

Sumber: [POW-06:51]

– *One-to-Many*

Relasi *One-to-Many* merupakan relasi yang paling sering dijumpai pada *relational database model* [POW-06:52]. Satu baris tabel mengacu pada satu baris atau lebih pada tabel lainnya.

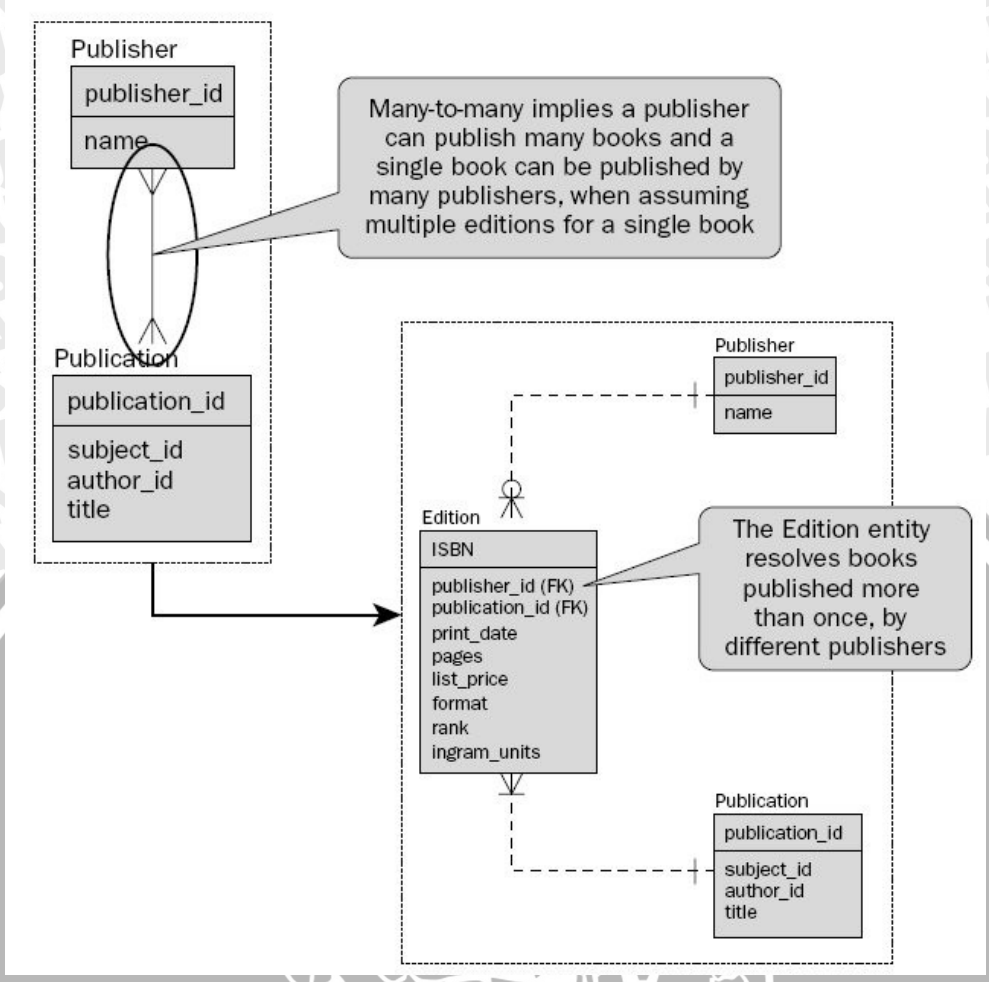


Gambar 2.9 Contoh Relasi *One-to-Many*

Sumber: [POW-06:52]

– *Many-to-Many*

Relasi *many-to-many* berarti bahwa setiap baris tabel memiliki kemungkinan memiliki relasi dengan banyak baris pada tabel lainnya, dan sebaliknya. Biasanya relasi *many-to-many* membutuhkan tabel tambahan sebagai penghubung.



Gambar 2.10 Contoh Relasi *Many-to-Many*

Sumber: [POW-06:54]

Program aplikasi yang digunakan untuk melakukan manajemen terhadap suatu *relational database* disebut *Relational Database Management System* (RDBMS), contohnya MySQL, PosgreSQL, Oracle, Microsoft SQL Server, dsb.

2.8 Rekyasa Perangkat Lunak

Rekayasa perangkat lunak diperlukan sebagai suatu metode panduan dalam proses pengembangan suatu perangkat lunak agar menghasilkan sebuah perangkat lunak yang benar dan berkualitas. Semua perangkat lunak harus dibuat menggunakan suatu metodologi tertentu [ODO-05:100]. Metodologi adalah suatu cara untuk bekerja secara sistematis. Secara umum, proses pengembangan perangkat lunak dapat dibagi menjadi beberapa tahap, yaitu:

- *Requirements*

  Tahap ini terbagi menjadi dua, yaitu *business modelling* dan *system requirements modeling* (atau *functional specification*). *Business modeling* memahami pada konteks apa perangkat lunak yang dibuat beroperasi [ODO-05:102]. Contoh pertanyaan yang muncul pada tahap ini adalah "Bagaimana pengunjung membeli televisi melalui web toko *online*?". S*ystem requirements modeling* menentukan kemampuan apa saja yang harus dimiliki perangkat lunak yang dibuat, kemudian mendokumentasikan semua kemampuan tersebut [ODO-05:102]. Contoh pertanyaan yang muncul pada tahap ini adalah "Bagaimana memperbaharui (*update*) *inventory system* ketika terjadi pembelian televisi?"

- *Analysis*

  Tahap ini berisi proses pemahaman terhadap "benda apa" yang sedang dihadapi [ODO-05:102]. Sebelum membuat suatu perancangan (*design*), dibutuhkan pengetahuan tentang entitas yang relevan beserta properti dan relasinya. Contoh pertanyaan yang muncul pada tahap ini adalah "Produk apa saja yang dijual di web toko *online*? Berasal dari mana produk tersebut? Berapa harganya?"

- *Design*

  Tahap ini bertujuan untuk merancang dan mengambil keputusan bagaimana masalah yang ada dapat dipecahkan. Pada tahap ini, sistem dipecah menjadi dua *subsystem*, yaitu *logical subsystems* (proses) dan *physical subsystems* (komputer dan jaringan komputer) [ODO-05:103]. Contoh keputusan yang diambil pada tahap ini adalah "Sistem dibangun menggunakan MVC *Framework* dan dapat diakses melalui internet (*world wide web*)"

- *Specification*

  Tahap ini bertujuan untuk mendeskripsikan perilaku (*behaviour*) perangkat lunak yang dibuat [ODO-05:103]. Spesifikasi dapat digunakan untuk merancang *test* dalam pengujian perangkat lunak..

- *Implementation*

  Pada tahap ini dilakukan penulisan *code* untuk membangun perangkat lunak yang telah dirancang.

– *Testing*

Ketika perangkat lunak telah selesai, perangkat lunak tersebut harus diuji terhadap *system requirements* (kebutuhan sistem) supaya sesuai dengan tujuan aslinya.

– *Deployment*

Pada tahap ini, perangkat lunak (dan perangkat keras) dikondisikan supaya dapat digunakan oleh pengguna yang sebenarnya (*end-user*) sesuai dengan kebutuhan sistem.

– *Maintenance*

Tahap ini berlangsung saat perangkat lunak telah memasuki tahap *deployment.* Pada umumnya tahap ini digunakan untuk memperbaiki cacat yang ada pada perangkat lunak yang tidak ditemukan ketika proses perancangan. Selain itu, pada tahap ini biasanya terjadi penambahan/pengurangan fitur perangkat lunak akibat adanya perubahan *requirements*.

Masing – masing tahap memiliki jenis pertanyaan yang khas. Tabel 2.1 menunjukkan beberapa contoh pertanyaan kunci yang umum:

Tabel 2.1 Contoh Pertanyaan Kunci

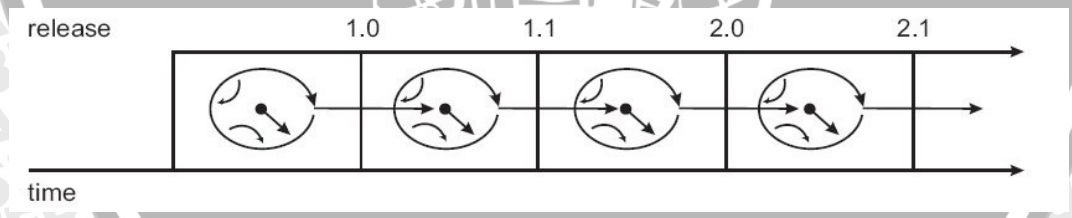
Phase	***Key Questions***
Requirements	*"What is our context?"* *"What are we trying to achieve?"*
Analysis	*"What entities are we dealing with?"* *"How can we be sure we have the right ones?"*
System Design	*"How are we going to solve problem?"* *"What hardware and software will we need in the finished system?"*
Subsystem Design	*"How are we going to implement the solution?"* *"What will the source code and supporting files look like?"*
Specification	*"What rules govern the interface between the system components?"* *"Can we remove ambiguity and ensure correctness?"*

Implementation	*"How can we code the components to meet the specification?"* *"How do we write stylish code?"*
Testing	*"Does the finished system satisfy the requirements?"* *"Can we break the system?"*
Deployment	*"What do the system administrators have to do?"* *"How can we educate the end user?"*
Maintenance	*"Can we find and fix faults?"* *"Can we improve the system?"*

Sumber: [ODO-05:105]

2.8.1 Metode *Iterative and Incremental*

Metode *iterative and incremental* adalah metode pengembangan perangkat lunak baru yang merupakan pengembangan dari metode *waterfall*. Dalam metode ini, proses pengembangan perangkat lunak menggabungkan dua buah metode sekaligus, yaitu *iterative* dan *incremental*. Metode *iterative* memungkikan untuk melakukan iterasi pada suatu kumpulan tahapan tertentu, misal iterasi yang dilakukan pada tahap implementasi dan pengujian. Sedangkan metode *incremental* merupakan metode pengembangan perangkat lunak secara bertahap. Metode *iterative and incremental* ditunjukkan dalam gambar 2.11.



Gambar 2.11 *Iterative and Incremental Development*

Sumber: [ODO-05:114]

2.8.2 *Behaviour-Driven Development* (BDD)

Behaviour-Driven Development (BDD) merupakan evolusi dari *Test-Driven Development* (TDD)[6]. Pada model pengembangan perangkat lunak yang modern, pengujian tidak selalu dilakukan setelah tahap implementasi. Pengujian dapat dilakukan sebelum implementasi karena hal ini dianggap lebih efektif [ODO-05:408]. Beberapa keuntungan yang dapat diperoleh dengan metode TDD adalah:

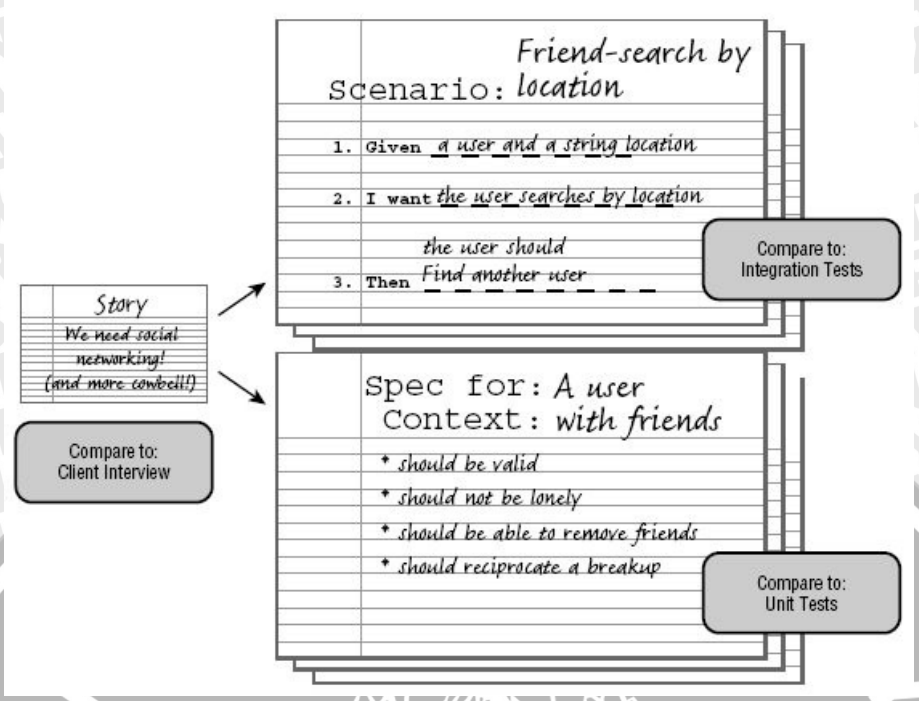
- Meningkatkan kualitas perangkat lunak (semakin banyak pengujian semakin baik).
- Mengurangi biaya pada tahap pengujian.
- Memudahkan programmer untuk melakukan refactoring[7] terhadap kode yang ditulis tanpa mengganggu sistem lain yang telah ditulis.

BDD merupakan suatu pola pikir untuk membuat struktur pengujian dan pengembangan perangkat lunak dalam rangka meningkatkan efektivitas pengembang. Berbeda dengan TDD yang menekankan pada pengujian (*unit test*), BDD lebih menekankan pada spesifikasi. *Test case* yang digunakan sebagian besar mengacu pada kebutuhan fungsional sistem yang dikembangkan. Umumnya dalam pengembangan perangkat lunak secara BDD, permasalahan dimodelkan mencadi kumpulan cerita (*stories*) yang terdiri dari *specs* dan *scenarios* untuk membantu pengembangan perangkat lunak. Contoh *stories* terdapat pada gambar 2.12.

---

6 *Test Driven Development* (TDD) adalah salah satu metode pengembangan perangkat lunak dimana pengujian tidak dilakukan untuk menguji kode yang sudah ditulis, melainkan untuk mendefinisikan suatu ekspektasi terhadap kode yang akan ditulis [BEN-08:276].

7 *Refactoring* adalah proses yang dilakukan untuk meningkatkan kualitas kode dengan cara melakukan penataan ulang [ODO-05:436].



Gambar 2.12 Contoh *Stories*

Sumber: [BEN-08:278]

Pada umumnya metode BDD selalu digabungkan dengan metode *iterative and incremental*, hal ini bertujuan untuk meningkatkan produktivitas dan kualitas perangkat lunak yang dihasilkan. Dengan demikian, proses pengembangan peragkat lunak bisa menjadi lebih singkat. Gabungan ketiga metode ini (*behaviour-driven*, *iterative*, dan *incremental*) termasuk salah satu jenis *agile software development*.

BAB III
METODE PENELITIAN

Metode penelitian menjelaskan langkah-langkah yang dilakukan dalam perancangan, implementasi, dan pengujian dari aplikasi yang akan dibuat. Kesimpulan dan saran disertakan sebagai catatan atas aplikasi dan bagaimana aplikasi dapat dikembangkan lebih lanjut.

3.1 Studi Literatur

Berupa kajian pustaka terhadap sumber-sumber bacaan yang relevan yang dapat menunjang dalam proses pengembangan sistem. Kajian terhadap berbagai literatur ini ditujukan untuk mendapatkan landasan teori yang diperlukan, landasan teori yang dimaksud antara lain, yaitu:

(a) Social Networking

(b) *Hyper Text Transfer Protocol* (HTTP)

(c) *Uniform Resource Identifier* (URI)

(d) *Representational State Transfer* (REST)

- *Resource Orieneted Architecture* (ROA)

(e) Ruby On Rails

- *Model View Controller* (MVC)
- *Object Reletional Mapping* (ORM)

(f) Ajax

(g) Relational Database Model

(h) Rekayasa Perangkat Lunak

- Metode *Iterative and Incremental*
- Pengujian (*Testing*)
- *Behaviour Driven Development* (BDD)

3.2 Definisi Operasional

Definisi operasional yang digunakan pada perancangan dan pembuatan situs social networking menggunakan prinsip representational state transfer adalah:

1. *Representational State Transfer* (REST) adalah kumpulan teori tentang bagaimana cara untuk mengatur dan menyusun sistem *hypermedia* yang terdistribusi, contohnya *World Wide Web*.
2. RPC adalah suatu metode yang memungkinkan dilakukannya pemanggilan suatu fungsi atau prosedur dari jarak jauh.
3. Ruby On Rails (RoR) adalah *framework* dari bahasa pemrograman Ruby untuk membuat suatu aplikasi berbasis web yang menggunakan basis data pada bagian *back-end*.
4. Mongrel adalah salah satu *web server* populer yang digunakan untuk menjalankan aplikasi berbasis web yang dibuat menggunakan *framework* Ruby On Rails. Mongrel memiliki skalabilitas yang baik, sehingga dapat ditingkatkan kemampuannya dengan membuat suatu *cluster*. *Cluster* tersebut dapat diakses melalui suatu *proxy* yang berfungsi sebagai *load balancer*.
5. MySQL adalah salah satu *database management system* (DBMS) jenis relasional yang paling populer untuk aplikasi berbasis web.
6. Skalabilitas adalah kemampuan suatu sistem untuk selalu dapat memenuhi kebutuhan yang ada. Parameter yang digunakan dalam skalalabilitas pada skripsi ini adalah *user concurrency*, yaitu banyaknya pengguna yang dapat dilayani pada saat yang bersamaan. Semakin banyak pengguna yang dapat dilayani, maka web memiliki skalabilitas semakin baik.
7. *Interoperability* adalah kemampuan suatu sistem untuk berinteraksi / bekerjasama dengan sistem yang lain. Parameter yang digunakan dalam *interoperability* pada skripsi ini adalah banyaknya variasi *methods* yang ada dalam *interface*. Semakin sedikit variasi yang ada, maka *interoperability* semakin baik.
8. Membangun web secara baik dan benar adalah membangun web menggunakan prinsip *Representational State Transfer*. Parameter yang digunakan adalah aturan – aturan yang tercantum dalam spesifikasi protokol HTTP, yaitu RFC-

2616.

3.3 Perancangan

Perancangan aplikasi dilakukan untuk mempermudah implementasi, analisis algoritma dan pengujian. Perancangan aplikasi berdasarkan *Object Oriented Analysis* dan *Object Oriented Design* yaitu menggunakan pemodelan UML (*Unified Modeling Language*). Perancangan terdiri dari analisis kebutuhan, perancangan *use case*, *activity*, *resource*, basis data, dan *user interface*.

3.4 Implementasi

Metode pengembangan yang digunakan adalah gabungan dari metode *Iterative and Incremental* dan *Behaviour Driven Development* dengan mengacu pada perancangan situs *social networking* menggunakan prinsip *Representational State Transfer* (REST). Siklus yang digunakan mengacu pada siklus *Test-Driven Development* seperti ditunjukkan pada gambar 3.1.

Gambar 3.1 Siklus Dasar *Test-Driven Development*

Sumber: [FRE-08]

Implementasi perangkat lunak dilakukan dengan menggunakan bahasa pemrograman berorientasi objek, yaitu Ruby dengan bantuan *framework* Ruby On Rails. Implementasi basis data menggunakan MySQL.

3.5 Pengujian dan Analisis

Pengujian aplikasi dilakukan untuk mengetahui kesesuaian analisis kebutuhan yang dibuat dengan implementasi aplikasi. Pengujian yang dilakukan meliputu pengujian unit dan pengujian fungsional. Pengujian dilakukan dengan metode *behaviour driven* menggunakan teknik *black box*. Dalam metode

behaviour driven, teknik *black box* memiliki keunggulan dibanding teknik *white box*, pengujian dengan teknik *black box* akan tetap lolos (*pass*) ketika kode program telah mengalami proses *refactoring* [SAL-08:38], sedangkan teknik *white box* akan menghasilkan pengujian yang gagal (*fail*) [SAL-08:37]. Analisis sistem dilakukan dengan membandingkan sistem aplikasi yang ada dengan teori yang ada sehingga didapatkan suatu kesimpulan.

3.6 Pengambilan Kesimpulan dan Saran

Kesimpulan diambil untuk menjawab rumusan masalah yang telah ditetapkan. Saran ditujukan untuk memberikan pertimbangan atas pengembangan aplikasi selanjutnya.

BAB IV
PERANCANGAN

Bab ini membahas mengenai perancangan perangkat lunak yang meliputi dua tahap. Pada tahap pertama dilakukan proses analisis kebutuhan (*requirement analysis*), sedangkan tahap yang kedua adalah proses perancangan. Pada tahap analisis kebutuhan digunakan pemodelan dengan diagram *use case.* Pada tahap perancangan digunakan diagram aktivitas (*activity diagram*), tabel *resource*, diagram E-R, dan sketsa perancangan *user interface*.

4.1 Analisis Kebutuhan (*Requirement Analysis*)

Tahap ini dilakukan dengan memodelkan kebutuhan sistem ke dalam *use case diagram.* Proses analisis ini mengambil acuan dari hasil pengumpulan, pemahaman dan penetapan kebutuhan-kebutuhan (*requirements*) yang ingin didapatkan oleh pengguna, dan harus mampu disediakan oleh sistem perangkat lunak. Analisis kebutuhan dibagi menjadi tiga tahap, yaitu identifikasi aktor-aktor, penjabaran daftar kebutuhan, dan pemodelan *use case diagram*.

4.1.1 Identifikasi Aktor

Tahap ini dilakukan untuk mengidentifikasi aktor yang berinteraksi dengan sistem. Ada tiga aktor yang terlibat dalam situs *social networking* seperti pada tabel 4.1.

Tabel 4.1 Deskripsi Aktor

Aktor	Deskripsi Aktor
Pengguna	Pengguna merupakan semua aktor pengguna yang menggunakan fasilitas situs social networking namun tidak melakukan *login* pada sistem. Aktor ini memiliki hak akses yang terbatas.
Anggota	Anggota merupakan semua aktor pengguna yang menggunakan fasilitas situs *social networking* dan telah melakukan *login* pada sistem. Aktor ini memiliki semua hak akses pengguna (kecuali melakukan *login* dan pendaftaran) ditambah beberapa hak akses lain khusus anggota.
Administrator	Administrator merupakan aktor dengan hak akses paling tinggi. Administrator memiliki hak menghapus akun anggota lain.

Sumber: Analisis Kebutuhan

4.1.2 Daftar Kebutuhan

Daftar kebutuhan terdiri dari kebutuhan fungsional dan kebutuhan non-fungsional. Daftar kebutuhan fungsional dan non-fungsional sistem yang dikembangkan pada tugas akhir ini ditunjukkan pada Tabel 4.2.

Tabel 4.2 Daftar Kebutuhan Fungsional dan Non Fungsional

ID	Kebutuhan	Aktor	Nama *Use Case*
F01	Sistem harus menyediakan fasilitas pendaftaran bagi pengguna untuk membuat akun baru.	Pengguna	Daftar
F02	Sistem harus menyediakan fasilitas login untuk melakukan *authentication* terhadap pengguna yang ingin menggunakan fasilitas *social networking* secara utuh.	Pengguna	*Login*
F03	Sistem harus menyediakan fasilitas untuk menampilkan profil anggota situs *social networking*.	Pengguna	Melihat Profil Anggota
F03a	Sistem harus menyediakan fasilitas untuk menampilkan *blog* anggota	Pengguna	Melihat *Blog* Anggota
F03b	Sistem harus menyediakan fasilitas untuk menampilkan teman anggota	Pengguna	Melihat Teman Anggota

F04	Sistem harus menyediakan fasiitas *logout* untuk menghapus *session* milik pengguna.	Anggota	*Logout*
F05	Sistem harus menyediakan fasilitas untuk mengubah profil dan menyimpan hasil perubahan ke dalam basis data.	Anggota	Mengubah Profil
F06	Sistem harus menyediakan fasilitas untuk mengubah *avatar* pengguna melalui layanan *upload* maupun hapus gambar berupa *file* JPEG. Sistem harus dapat menyimpan hasil perubahan tersebut ke dalam basis data dan *file system*.	Anggota	Mengubah *Avatar*
F07	Sistem harus menyediakan fasilitas bagi pengguna untuk menulis *posting blog* dan mempublikasikannya. Sistem harus dapat menyimpan hasil tulisan tersebut ke dalam basis data.	Anggota	Menulis *Posting Blog*
F08	Sistem harus menyediakan fasilitas bagi pengguna untuk mengubah *posting blog* yang telah dipublikasikan. Sistem harus dapat menyimpan hasil perubahan tersebut ke dalam basis data.	Anggota	Mengubah *Posting Blog*
F09	Sistem harus menyediakan fasilitas bagi pengguna untuk menghapus *posting blog* yang telah dipublikasikan. Sistem harus dapat menyimpan perubahan tersebut ke dalam basis data.	Anggota	Menghapus *Posting Blog*
F10	Sistem harus menyedikan fasilitas bagi pengguna untuk menulis kementar yang ditujukan pada *posting blog* tertentu. Sistem harus dapat menyimpan komentar tersebut ke dalam basis data.	Anggota	Menulis Komentar *Posting Blog*
F11	Sistem harus menyediakan fasilitas bagi pengguna untuk menghapus komentar *posting blog*. Sistem harus dapat menyimpan perubahan tersebut ke dalam basis data.	Anggota	Menghapus Komentar *Blog*
F12	Sistem harus menyedikan fasilitas bagi pengguna untuk menambah teman. Sistem harus dapat menyimpan perubahan data ke dalam basis data.	Anggota	Menambah Teman

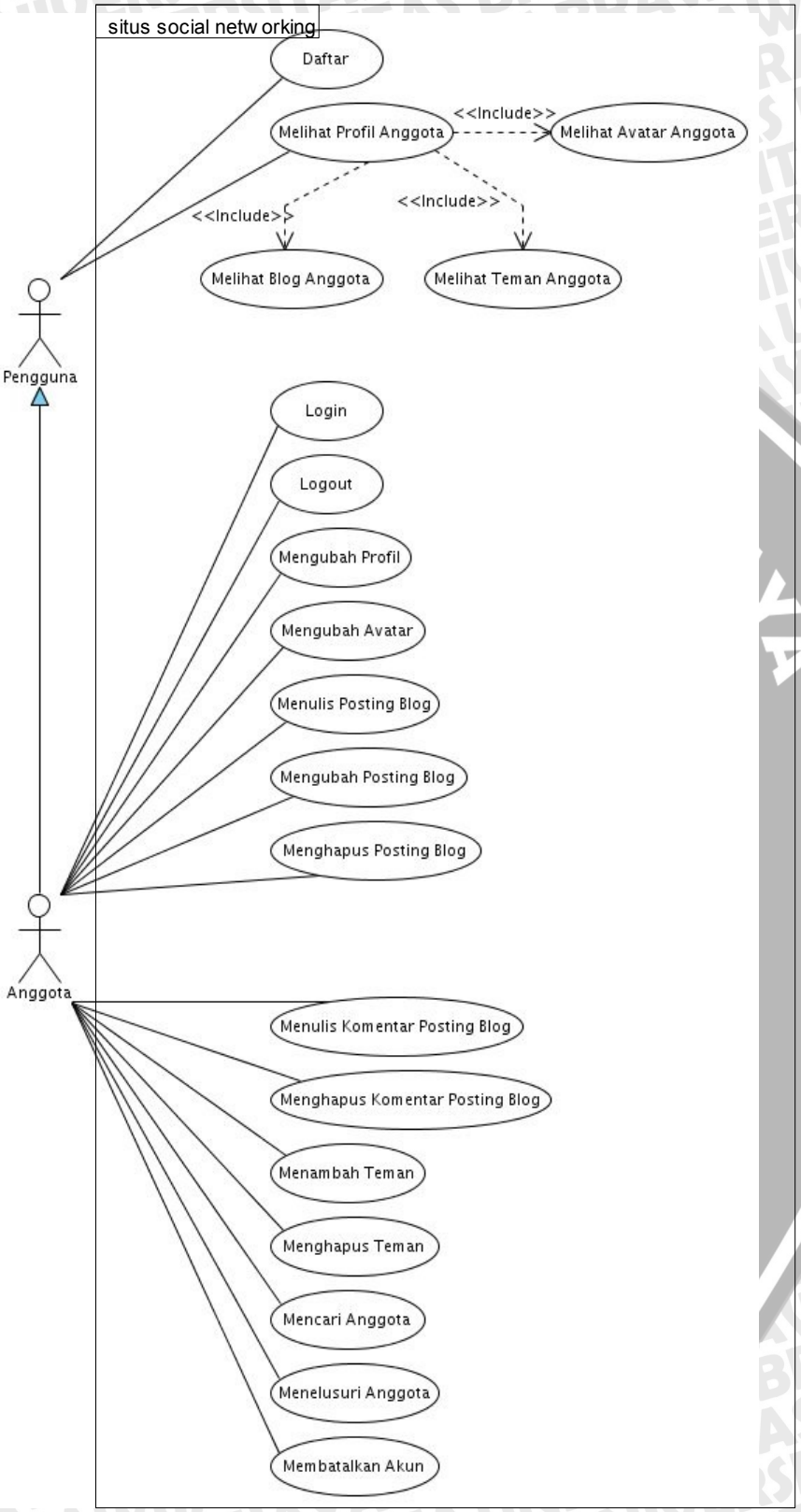
F13	Sistem harus menyediakan fasilitas bagi pengguna untuk menghapus teman. Sistem harus dapat menyimpan perubahan data ke dalam basis data.	Anggota	Menghapus Teman
F14	Sistem harus menyediakan fasilitas bagi pengguna untuk mencari anggota lain menggunakan kata kunci nama, jenis kelamin, dan lokasi anggota.	Anggota	Mencari Anggota
F15	Sistem harus menyediakan fasilitas bagi pengguna untuk menelusuri anggota lain berdasarkan urutan abjad nama.	Anggota	Menelusuri Anggota
F16	Sistem harus menyediakan fasilitas bagi pengguna untuk membatalkan akun yang telah dibuat. Sistem harus mampu menghapus segala sesuatu yang berkaitan dengan anggota tersebut dan menyimpan perubahan ke dalam basis data.	Anggota	Membatalkan Akun
F17	Sistem harus menyediakan fasilitas login untuk melakukan *authentication* terhadap administrator yang akan melakukan pemeliharaan terhadap situs *social networking*.	Administrator	*Login*
F18	Sistem harus menyediakan fasilitas untuk menghapus akun anggota apabila ada anggota yang melanggar peraturan. Sistem harus dapat menyimpan perubahan tersebut ke dalam basis data.	Administrator	Menghapus Akun
F19	Sistem harus menyediakan fasiitas *logout* untuk menghapus *session* milik administrator.	Administrator	*Logout*
N01	Situs *sicoal networking* yang dibuat harus dapat diakses menggunakan *web browser* modern, yaitu Mozilla Firefox 3 dan Opera 9.	-	-

Sumber: Hasil Analisis Kebutuhan

4.1.3 *Use Case Diagram*

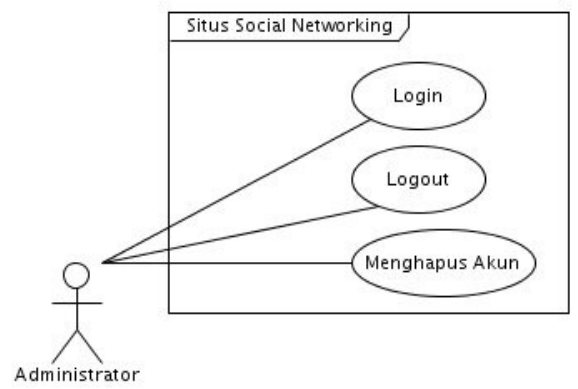
Kebutuhan-kebutuhan fungsional yang diperlukan oleh pengguna dan harus disediakan oleh sistem akan dimodelkan pada diagram *use case*. Dalam situs *social networking* yang dirancang, terdapat tiga aktor dan sembilan belas *use-case*. Gambar 4.1 dan 4.2 merupakan diagram *use-case* yang menggambarkan

fungsionalitas yang disediakan oleh sistem.



Gambar 4.1 *Use Case Diagram* – Pengunjung dan Anggota

Sumber: Hasil Analisis Kebutuhan



Gambar 4.2 *Use Case Diagram* – Administrator

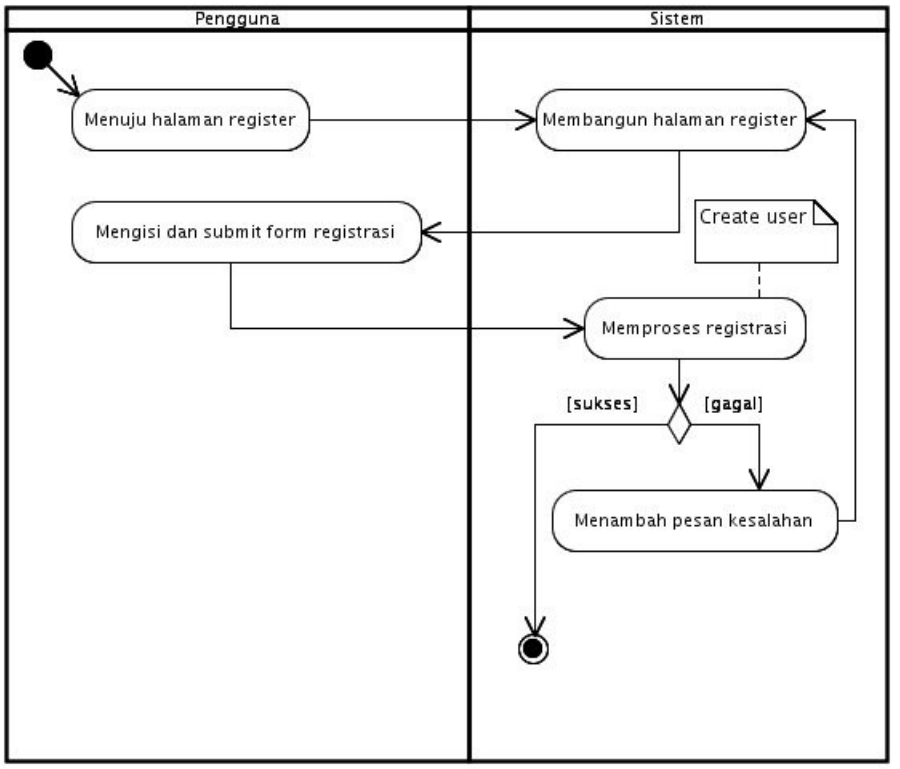
Sumber: Hasil Analisis Kebutuhan

4.1.4 *Activity Diagram*

Secara lebih detail, masing-masing *use case* yang terdapat pada diagram *use case* dijabarkan dalam *activity diagram*. Dalam *activity diagram* juga dijabarkan jalur alternatif pada masing – masing *use case*.

4.1.4.1 Daftar

Pengguna yang belum terdaftar dapat melakukan pendaftaran dengan cara mengisi *form* pendaftaran yang disediakan sistem. *Form* tersebut kemudian dikirim kembali ke *server* untuk diproses lebih lanjut. Proses tersebut meliputi validasi data dan pembuatan sebuah resource baru yang menangani anggota atau *users*. Kegiatan ini berlangsung seperti pada gambar 4.3.

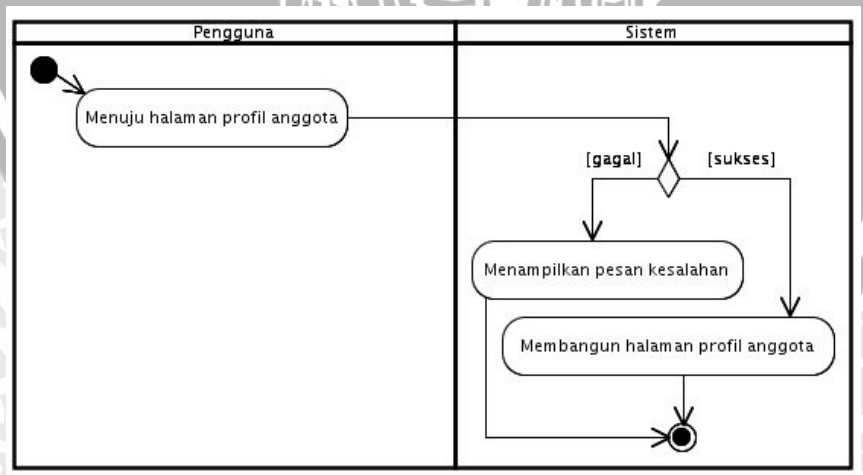


Gambar 4.3 *Activity Diagram* – Daftar

Sumber: Hasil Analisis Kebutuhan

4.1.4.2 Melihat Profil Anggota

Anggota situs *social networking* dapat melihat profil anggota lain dengan cara menuju langsung ke alamat *resource* yang berisi halaman profil anggota yang dituju. Kegiatan ini berlangsung seperti pada gambar 4.4.

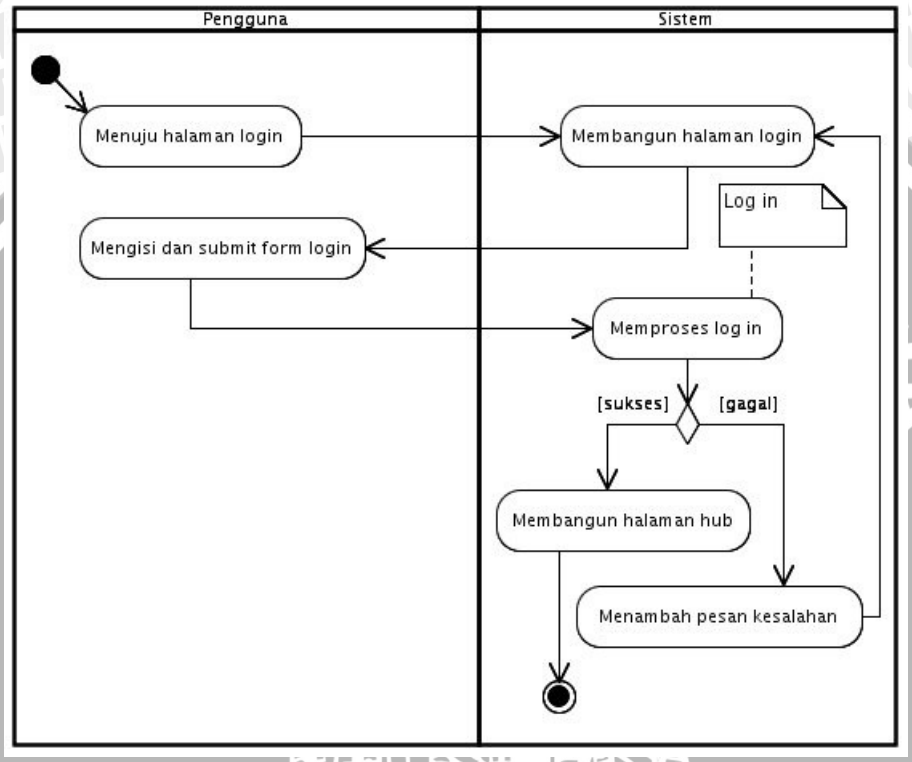


Gambar 4.4 *Activity Diagram* – Melihat Profil Anggota

Sumber: Hasil Analisis Kebutuhan

4.1.4.3 *Login*

Pengguna yang telah terdaftar dapat melakukan *login* dengan cara mengisi *form* pendaftaran yang disediakan sistem. *Form* tersebut kemudian dikirim kembali ke *server* untuk diproses lebih lanjut. Proses tersebut meliputi validasi data dan pembuatan *resource* baru yang menangani *sessions*. Kegiatan ini berlangsung seperti pada gambar 4.5.

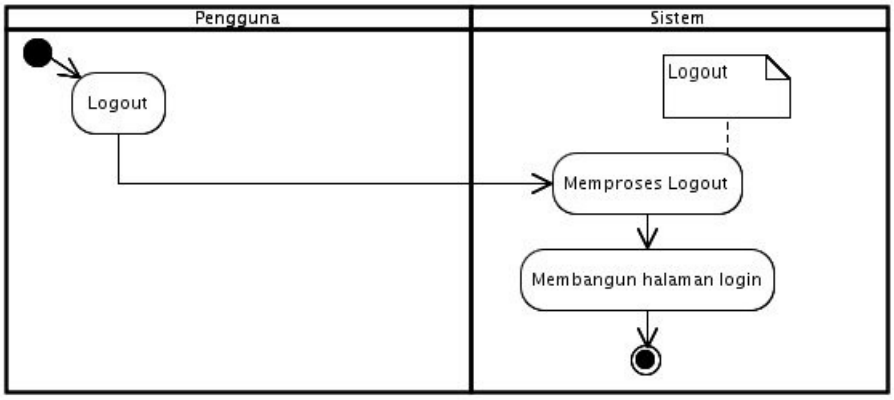


Gambar 4.5 *Activity Diagram – Login*

Sumber: Hasil Analisis Kebutuhan

4.1.4.4 *Logout*

Anggota yang telah melakukan *login* dapat keluar dari sistem atau melakukan *logout* dengan cara melakukan *request* terhadap server untuk menghapus *resource* yang menangani *sessions*. Kegiatan ini berlangsung seperti pada gambar 4.6.

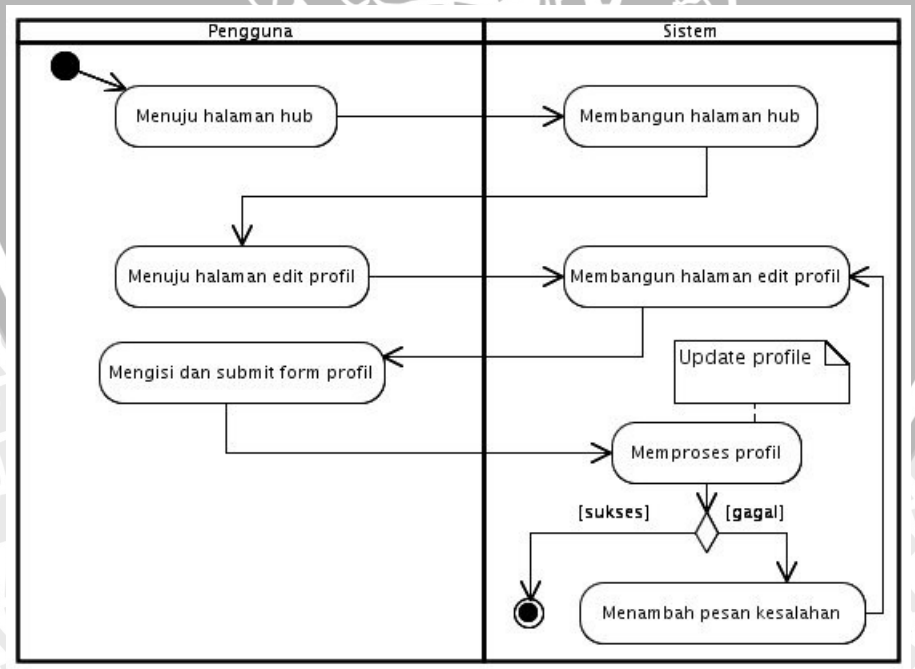


Gambar 4.6 *Activity Diagram – Logout*

Sumber: Hasil Analisis Kebutuhan

4.1.4.5 Mengubah Profil

Anggota dapat mengubah profil sesuai keinginan melalui halaman *hub* yang disediakan sistem. Halaman *hub* merupakan halaman utama yang memungkinkan anggota mengakses fitur - fitur yang disediakan situs *social networking*, termasuk halaman untuk mengubah profil. Anggota mengubah profil melalui *form* yang telah disediakan sistem. Kegiatan ini berlangsung seperti pada gambar 4.7.

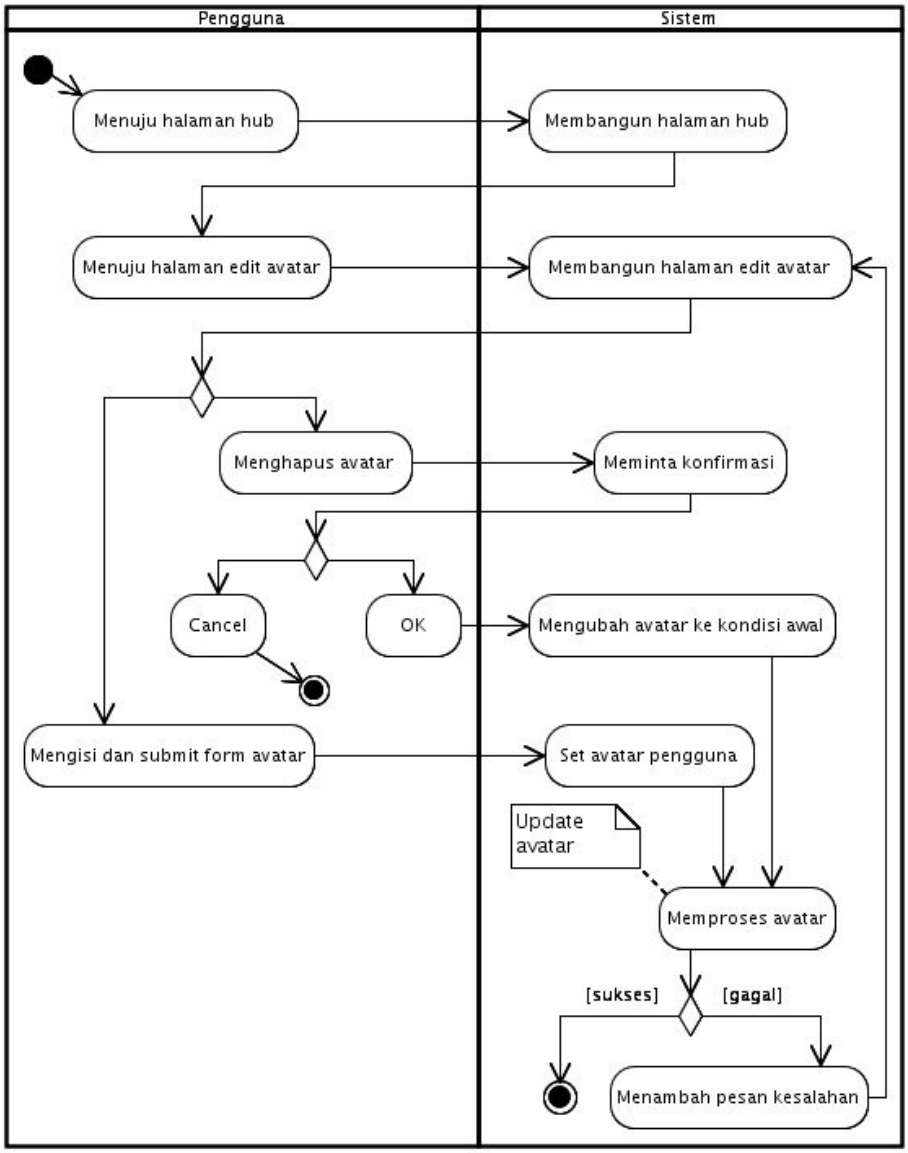


Gambar 4.7 *Activity Diagram* – Mengubah Profil

Sumber: Hasil Analisis Kebutuhan

4.1.4.6 Mengubah *Avatar*

Anggota memiliki *avatar* yang berfungsi sebagai foto identitas. *Avatar* dapat diubah melalui halaman *hub*, kemudian masuk ke halaman yang dibuat untuk mengubah *avatar*. Dalam halaman tersebut, anggota dapat melakukan *upload file* gambar ke *server*. Melalui halaman yang sama, anggota juga dapat menghapus *avatar*. Kegiatan ini berlangsung seperti pada gambar 4.8.

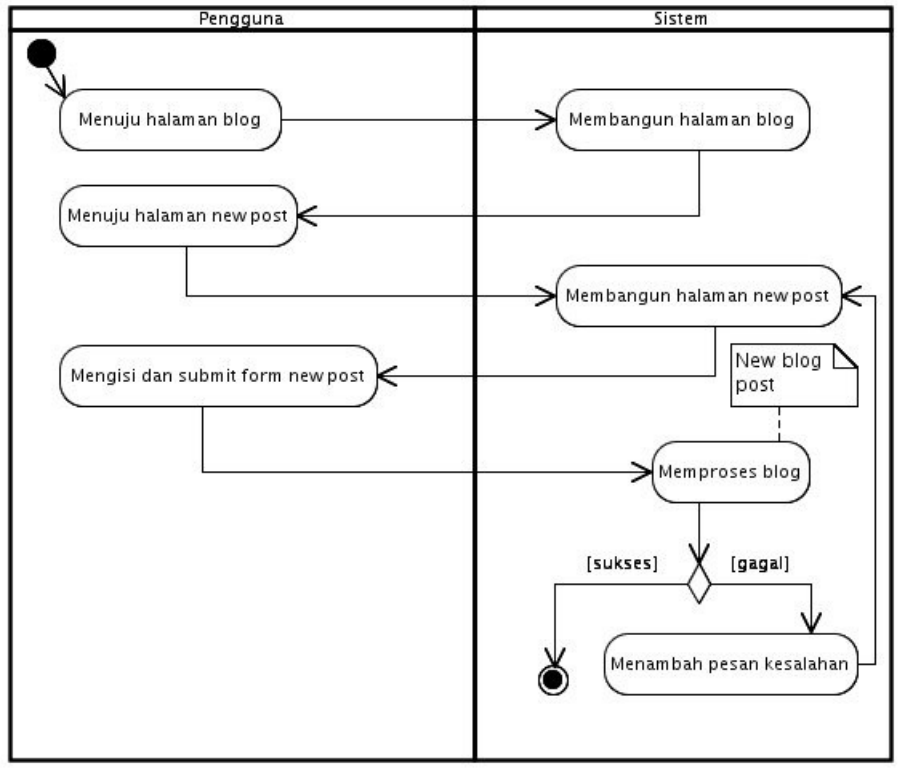


Gambar 4.8 *Activity Diagram* – Mengubah *Avatar*

Sumber: Hasil Analisis Kebutuhan

4.1.4.7 Menulis *Posting Blog*

Anggota memiliki sebuah *blog* yang dapat diisi dengan berbagai *posting*. Untuk menulis *posting*, anggota dapat menuju halaman *blog* dan memilih menu untuk membuat *posting* baru. Kemudian anggota dapat menulis *posting* melalui *form* yang disediakan sistem dan mengirimkannya ke *server* untuk diproses lebih lanjut. Kegiatan ini berlangsung seperti pada gambar 4.9.

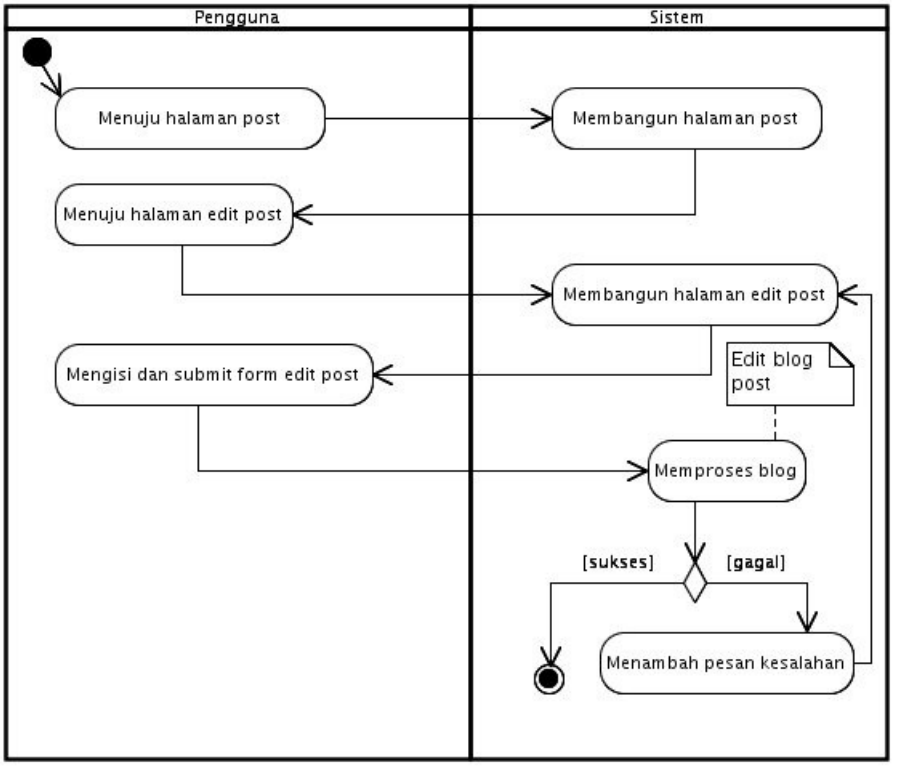


Gambar 4.9 *Activity Diagram* – Menulis *Posting Blog*

Sumber: Hasil Analisis Kebutuhan

4.1.4.8 Mengubah *Posting Blog*

Posting blog yang telah dipublikasikan dapat diubah setiap saat. Anggota dapat mengubah posting blog melalui halaman posting. Pada halaman tersebut terdapat tautan menuju halaman untuk mengubah *posting*. Halaman untuk mengubah *posting* berisi *form* yang berisi isi *posting* yang asli, kemudian pengguna dapat mengubah isi *posting* tersebut dan mengirimkannya ke *server* untuk diproses lebih lanjut. Kegiatan ini berlangsung seperti pada gambar 4.10.

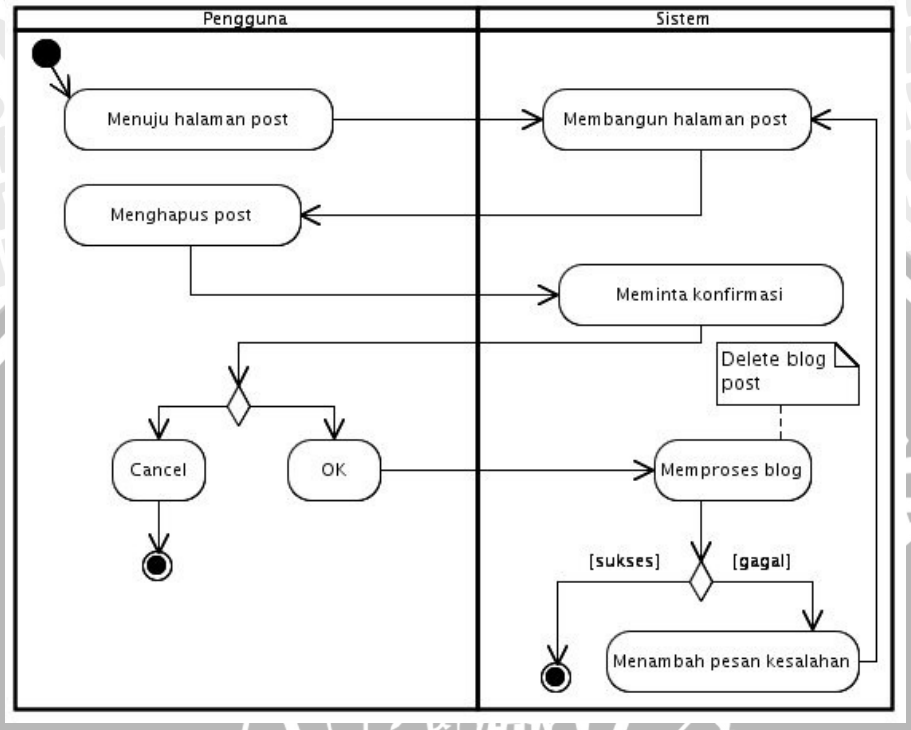


Gambar 4.10 *Activity Diagram* – Mengubah *Posting Blog*

Sumber: Hasil Analisis Kebutuhan

4.1.4.9 Menghapus *Posting Blog*

Anggota dapat menghapus posting *blog* yang telah dipublikasikan melalui tombol penghapusan yang tersedia pada masing - masing halaman *posting* yang dituju. Kegiatan ini berlangsung seperti pada gambar 4.11.

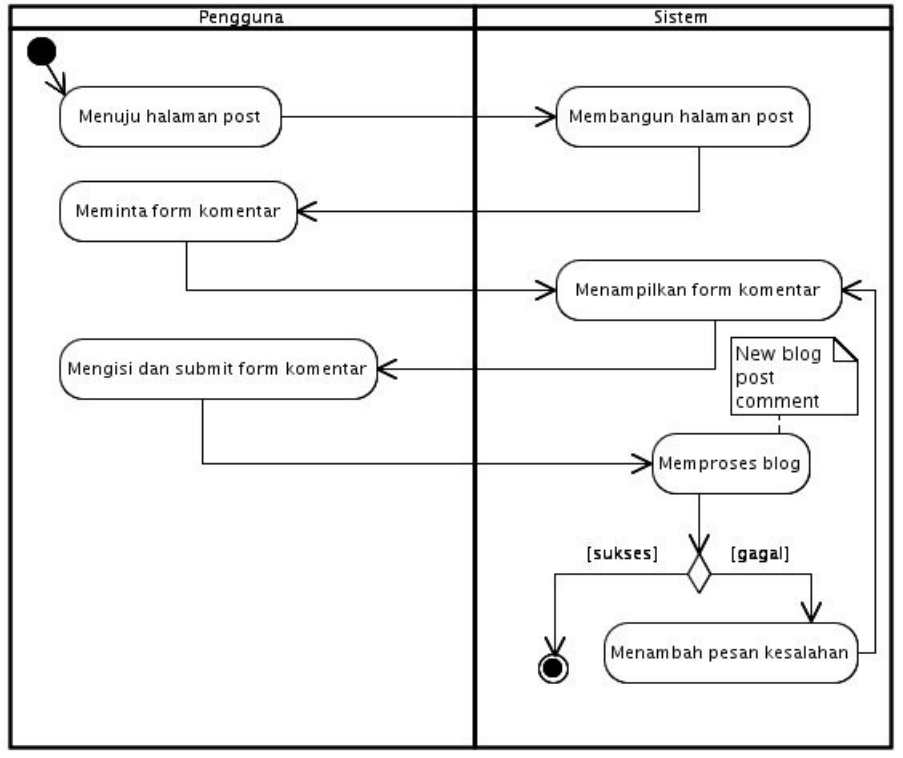


Gambar 4.11 *Activity Diagram* – Menghapus *Posting Blog*

Sumber: Hasil Analisis Kebutuhan

4.1.4.10 Menulis Komentar *Posting Blog*

Anggota dapat menulis komentar untuk masing – masing *posting*. Masing – masing halaman *posting* memiliki *form* untuk menulis komentar. Anggota dapat menulis komentar melalui *form* tersebut. Kegiatan ini berlangsung seperti pada gambar 4.12.

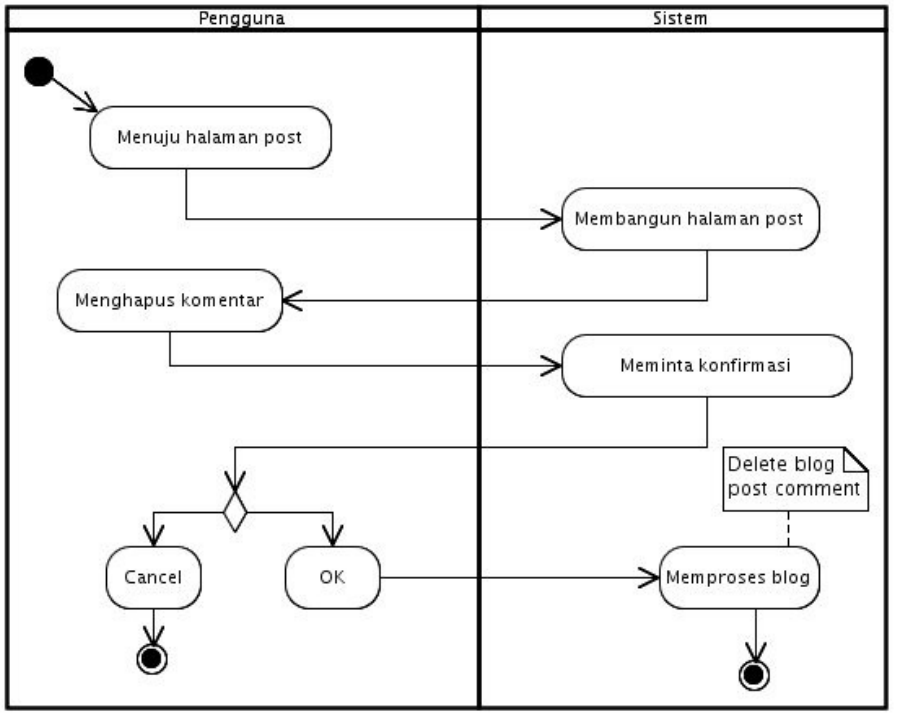


Gambar 4.12 *Activity Diagram* – Menulis Komentar *Posting Blog*

Sumber: Hasil Analisis Kebutuhan

4.1.4.11 Menghapus Komentar *Posting Blog*

Anggota dapat menghapus komentar untuk masing – masing *posting*. Masing – masing halaman *posting* memiliki tombol untuk menghapus komentar. Anggota dapat menghapus komentar melalui tombol tersebut. Kegiatan ini berlangsung seperti pada gambar 4.13.

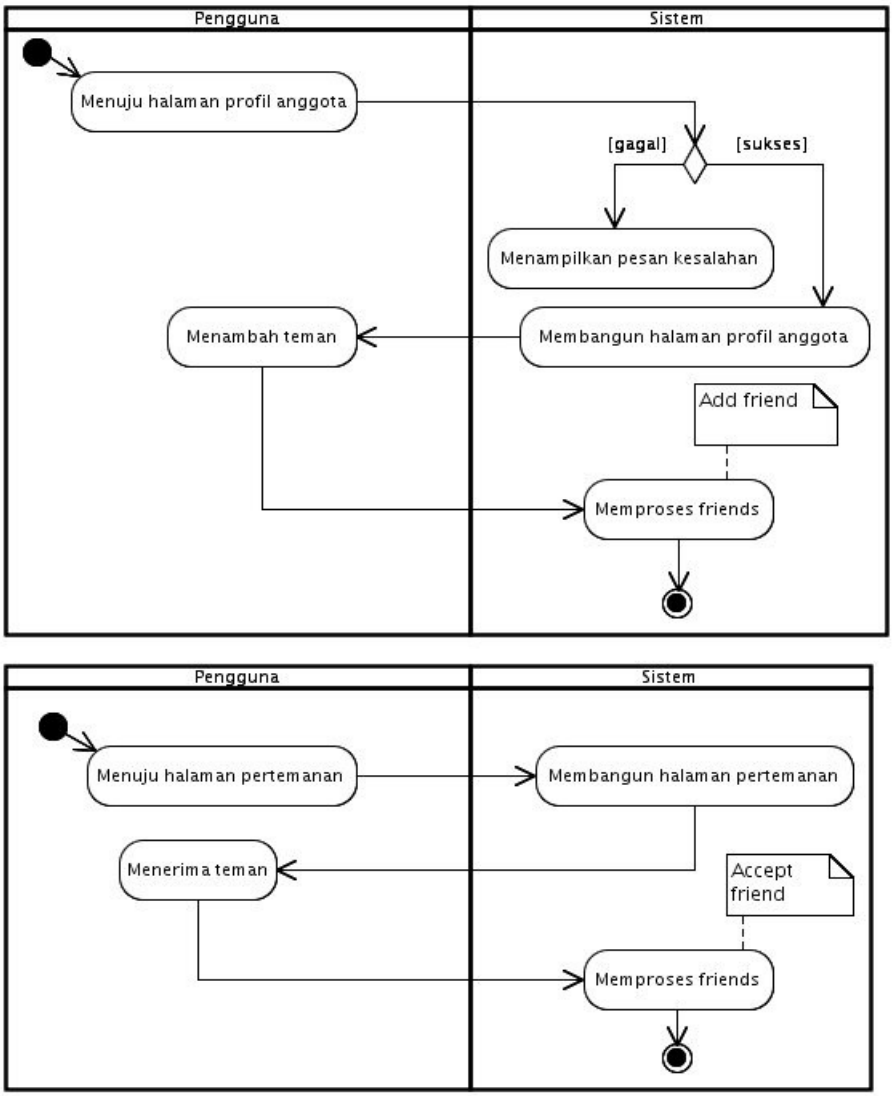


Gambar 4.13 *Activity Diagram* – Menghapus Komentar *Posting Blog*

Sumber: Hasil Analisis Kebutuhan

4.1.4.12 Menambah Teman

Anggota dapat menambah teman dengan cara mengunjungi halaman profil anggota yang dituju, kemudian menambahkan teman melalui tombol yang disediakan sistem. Langkah kedua adalah anggota yang diajak untuk berteman harus menerima ajakan tersebut melalui halaman pertemanan yang disediakan sistem. Kegiatan ini berlangsung seperti pada gambar 4.14.

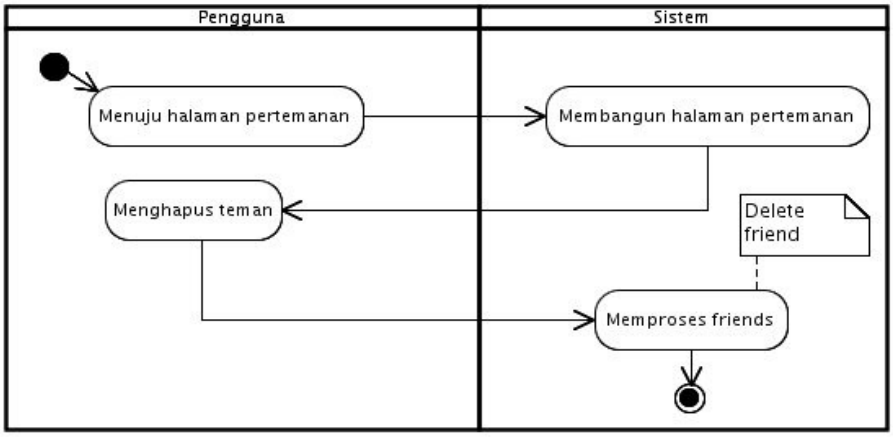


Gambar 4.14 *Activity Diagram* – Menambah Teman

Sumber: Hasil Analisis Kebutuhan

4.1.5.13 Menghapus Teman

Anggota dapat menghapus teman menggunakan tombol yang tersedia di halaman pertemanan. Kegiatan ini berlangsung seperti pada gambar 4.15.

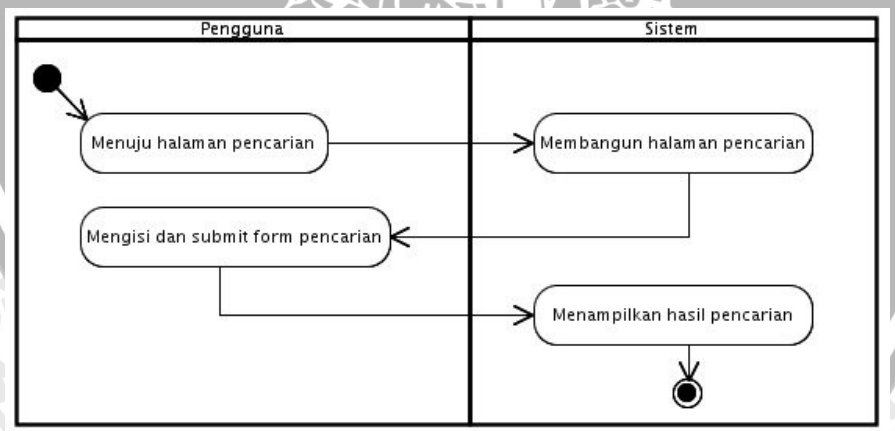


Gambar 4.15 *Activity Diagram* – Menghapus Teman

Sumber: Hasil Analisis Kebutuhan

4.1.4.14 Mencari Anggota

Anggota dapat mencari anggota lain melalui halaman pencarian. Halaman tersebut menyediakan *form* pencarian yang dapat diisi menggunakan kata kunci tertentu yang selanjutnya akan diproses oleh *server*. Kegiatan ini berlangsung seperti pada gambar 4.16.

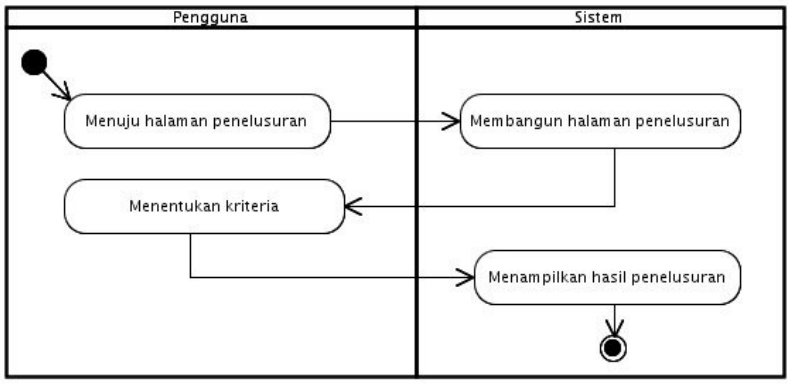


Gambar 4.16 *Activity Diagram* – Mencari Anggota

Sumber: Hasil Analisis Kebutuhan

4.1.4.15 Menelusuri Anggota

Anggota dapat menelusuri anggota lain melalui halaman penelusuran. Halaman tersebut menyediakan tombol – tembol yang merupakan abjad awalan dari nama anggota. Pengguna dapat menggunakan tombol – tombol tersebut untuk melakukan penelusuran. Kegiatan ini berlangsung seperti pada gambar 4.17.

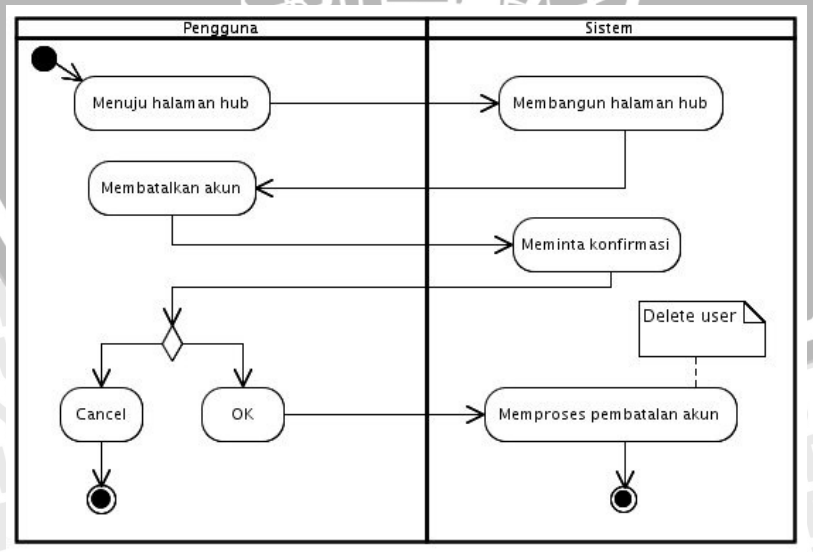


Gambar 4.17 *Activity Diagram* – Menelusuri Anggota

Sumber: Hasil Analisis Kebutuhan

4.1.4.16 Membatalkan Akun

Anggota dapat membatalkan akunnya melalui halaman *hub*. Pada halaman *hub* terdapat tombol untuk membatalkan akun. Pembatalan akun mengakibatkan seluruh data anggota yang bersangkutan terhapus. Kegiatan ini berlangsung seperti pada gambar 4.18.

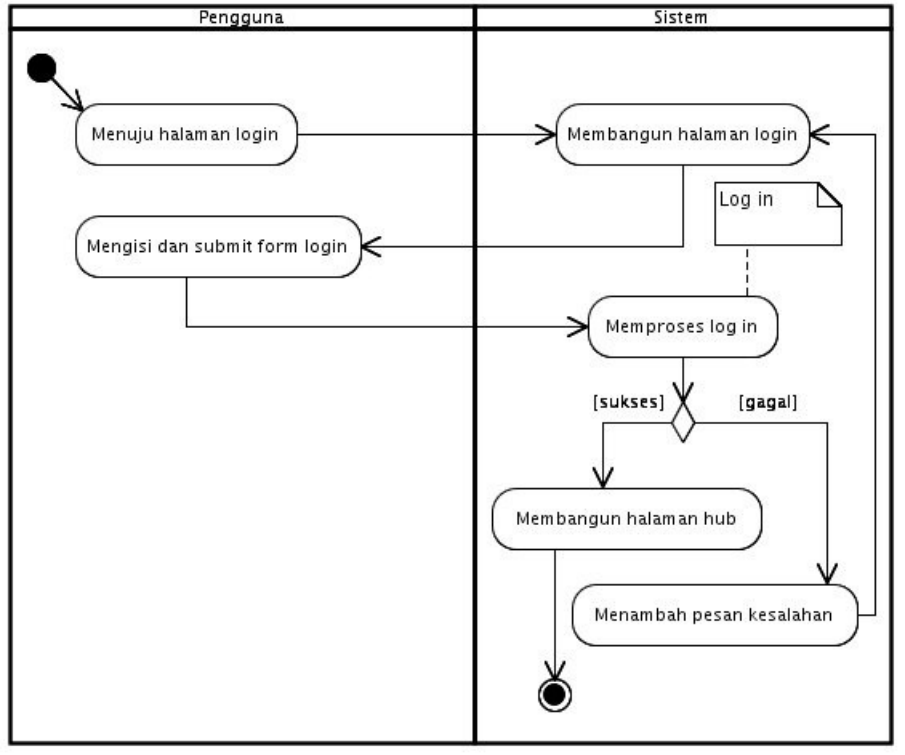


Gambar 4.18 *Activity Diagram* – Membatalkan Akun

Sumber: Hasil Analisis Kebutuhan

4.1.4.17 *Login* (Administrator)

Administrator dapat melakukan login melalui halaman *login* yang sama dengan halaman *login* untuk anggota. Kegiatan ini berlangsung seperti pada gambar 4.19.

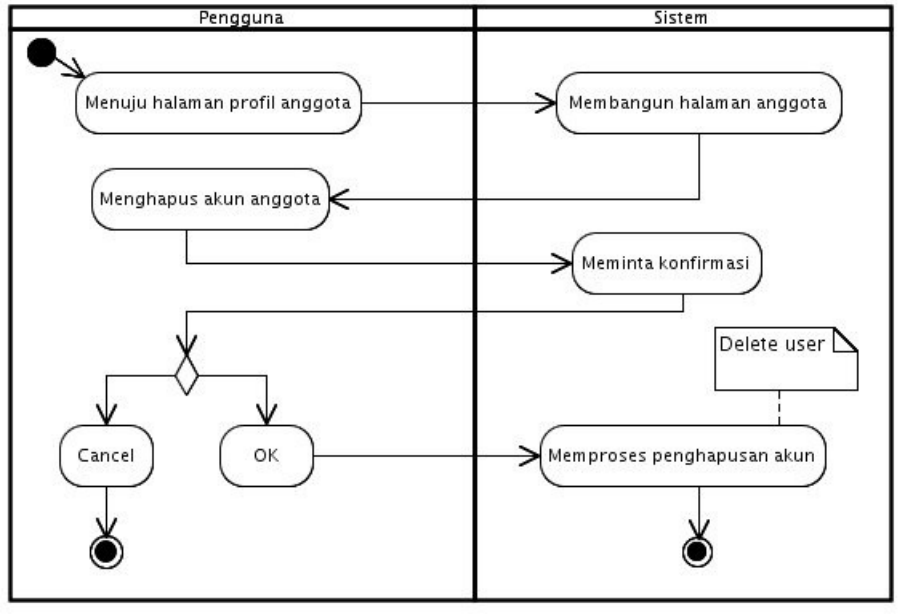


Gambar 4.19 *Activity Diagram – Login* (Administrator)

Sumber: Hasil Analisis Kebutuhan

4.1.4.18 Menghapus Akun (Administrator)

Administrator memiliki wewenang untuk menghapus akun anggota lain. Administrator dapat melakukannya melalui halaman profil anggota yang hendak dihapus. Setiap kali administrator mengunjungi halaman profil anggota, maka terdapat sebuah tombol untuk menghapus akun anggota yang bersangkutan. Kegiatan ini berlangsung seperti pada gambar 4.20.

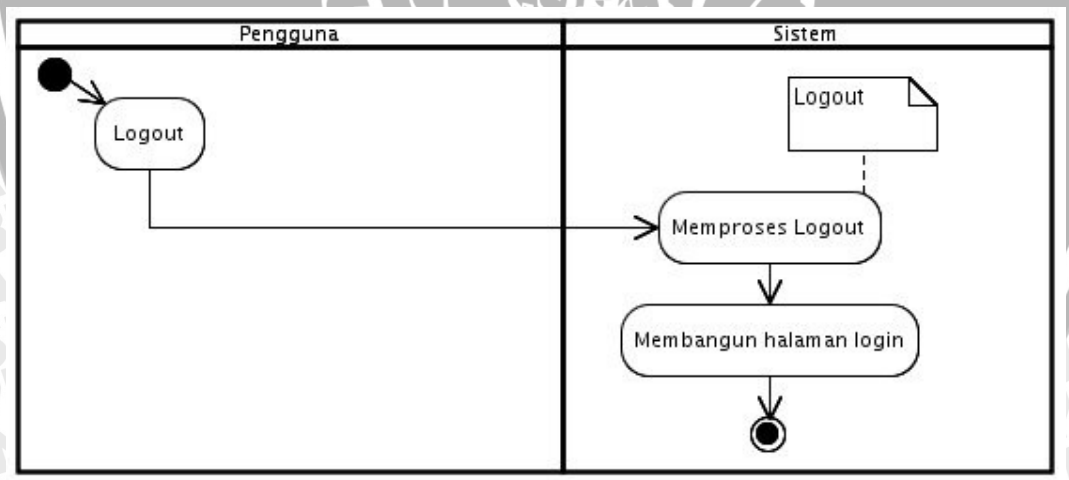


Gambar 4.20 *Activity Diagram* – Menghapus Akun (Administrator)

Sumber: Hasil Analisis Kebutuhan

4.1.4.19 *Logout* (Administrator)

Administrator dapat melakukan logout sebagaimana anggota yang lain. Kegiatan ini berlangsung seperti pada gambar 4.21.



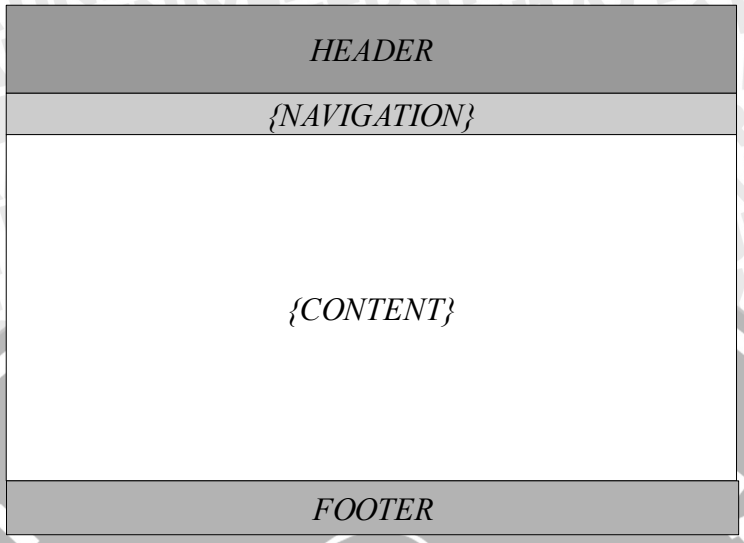
Gambar 4.21 *Activity Diagram – Logout* (Administrator)

Sumber: Hasil Analisis Kebutuhan

4.1.5 Perancangan *User Interface*

Aplikasi situs *social networking* berjalan pada *web browser* dan dibuat menggunakan teknologi standar HTML, CSS, dan Javascript. Secara garis besar,

tampilan situs s*ocial networking* dibagi menjadi empat bagian utama seperti pada gambar 4.22.



Gambar 4.22 *User Interface*

Sumber: Hasil Analisis Kebutuhan

Header berisi nama situs *social networking* dan bersifat statis. *Navigation* berisi navigasi situs dan juga berfungsi menuntun *application state. Content* berisi sajian utama dari situs *social networking*, dimana sebagian besar aktivitas pengguna ada di bagian ini. *Footer* berisi beberapa informasi tentang situs *social networking*.

4.2 Perancangan Perangkat Lunak

Perancangan perangkat lunak terdiri dari perancangan *resource*, percangan klas (*class diagram*), dan perancangan basis data (ER *diagram*).

4.2.1 Perancangan *Resources*

Sesuai dengan prinsip *Representational State Transfer* (REST), maka situs *social networking* dirancang menggunakan *Resource-Oriented Architecture* (ROA). Untuk mengubah suatu *resource* dibutuhkan suatu *modifiers*. *Modifiers* adalah media yang digunakan sebagai antar muka dalam memodifikasi suatu *resource* tertentu. Karena situs *social networking* merupakan suatu aplikasi berbasis web yang berjalan pada *web browser*, maka *modifiers* yang digunakan

berupa halaman web (*web page*) dengan format HTML. *Modifiers* dapat berupa *hyperlink* maupun HTML *form*.

Dari hasil analisis kebutuhan, dapat ditentukan bahwa *resource – resource* yang terdapat pada situs *social networking* adalah sebagai berikut:

1. *Site*

   Resource site digunakan untuk menampilkan dan mengatur halaman depan situs. Daftar *action*, HTTP *method*, dan *path* yang digunakan pada *resource site* dirinci dalam tabel 4.3.

Tabel 4.3 *Resource Site*

Action	**HTTP *Method***	***Path***
Menampilkan halaman muka.	`GET`	/

Sumber: Hasil Perancangan

2. *Users*

   Resource users digunakan untuk menampung dan mengatur keanggotaan situs *social networking*. Daftar *action*, HTTP *method*, dan *path* yang digunakan pada *resource users* dirinci dalam tabel 4.4.

Tabel 4.4 *Resource Users*

Action	**HTTP *Method***	***Path***
Mendaftar	`POST`	/users
Membatalkan akun	`DELETE`	/users/{user_id}
Modifier	**HTTP *Method***	***Path***
Menampilkan *form* pendaftaran	`GET`	/users/new
Menampilkan *link* pembatalan akun	`GET`	/users/{user_id}/hub

Sumber: Hasil Perancangan

3. *Session*

Resource session digunakan untuk proses autentikasi, menampung dan mengatur sesi anggota yang sedang *login*. Daftar *action*, HTTP *method*, dan *path* yang digunakan pada *resource session* dirinci dalam tabel 4.5.

Tabel 4.5 *Resource Session*

Action	**HTTP *Method***	***Path***
Melakukan *login*	`POST`	/session
Melakukan *logout*	`DELETE`	/session
Modifier	**HTTP *Method***	***Path***
Menampilkan *form login*	`GET`	/session/new
Menampilkan *link logout*	`GET`	# semua halaman #

Sumber: Hasil Perancangan

4. *Profile*

Resource profile digunakan untuk menampung dan mengatur profil anggota. Daftar *action*, HTTP *method*, dan *path* yang digunakan pada *resource profile* dirinci dalam tabel 4.6.

Tabel 4.6 *Resource Profile*

Action	**HTTP *Method***	***Path***
Menampilkan profil anggota	`GET`	/users/{user_id}/profile
Mengubah profil	`PUT`	/users/{user_id}/profile
Modifier	**HTTP *Method***	***Path***
Menampilkan *form* untuk mengubah profil	`GET`	/users/{user_id}/profile/edit

Sumber: Hasil Perancangan

5. *Avatar*

Resource avatar digunakan untuk menampung dan mengatur *avatar* anggota. Daftar *action*, HTTP *method*, dan *path* yang digunakan pada *resource avatar* dirinci dalam tabel 4.7.

Tabel 4.7 *Resource Avatar*

Action	**HTTP *Method***	***Path***
Mengubah *avatar*	`PUT`	/users/{user_id}/avatar
Menghapus *avatar*	`DELETE`	/users/{user_id}/avatar
Modifier	**HTTP *Method***	***Path***
Menampilkan *form* untuk mengubah *avatar*	`GET`	/users/{user_id}/avatar/edit
Menmpilkan *link* untuk menghapus *avatar*	`GET`	/users/{user_id}/avatar/edit

Sumber: Hasil Perancangan

6. *Friendships*

Resource friendships digunakan untuk menampung dan mengatur pertemanan antar anggota. Daftar *action*, HTTP *method*, dan *path* yang digunakan pada *resource friendships* dirinci dalam tabel 4.8.

Tabel 4.8 *Resource Friendships*

Action	**HTTP *Method***	***Path***
Menampilkan teman	`GET`	/users/{user_id}/friendships
Menambah teman	`POST`	/users/{friend_id}/friendships
Menghapus teman	`DELETE`	/users/{user_id}/friendships/{friend_id}
Modifier	**HTTP *Method***	***Path***
Menampilkan link untuk mengirim ajakan pertemanan	`GET`	/users/{user_id}/friendships

Menampilkan link untuk menerima ajakan pertemanan	GET	/users/{user_id}/friendships
Menampilkan link untuk membatalkan ajakan pertemanan	GET	/users/{user_id}/friendships
Menampilkan link untuk menolak ajakan pertemanan	GET	/users/{user_id}/friendships
Menampilkan link untuk menghapus pertemanan	GET	/users/{user_id}/friendships

Sumber: Hasil Perancangan

7. *Posts*

Resource posts digunakan untuk menampung dan mengatur *posting* yang terdapat pada *blog* milik anggota. Daftar *action*, HTTP *method*, dan *path* yang digunakan pada *resource posts* dirinci dalam tabel 4.9.

Tabel 4.9 *Resource Posts*

Action	**HTTP *Method***	***Path***
Menampilkan seluruh *posting*	GET	/users/{user_id}/blog/posts
Mempublikasikan *posting* baru	POST	/users/{user_id}/blog/posts
Menampilkan *posting* tertentu	GET	/users/{user_id}/blog/posts/{post_id}
Mengubah *posting* tertentu	PUT	/users/{user_id}/blog/posts/{post_id}
Menghapus *posting* tertentu	DELETE	/users/{user_id}/blog/posts/{post_id}

Modifier	**HTTP *Method***	***Path***
Menampilkan *form* untuk menulis *posting* baru	`GET`	/users/{user_id}/blog/posts/new
Menampilkan *form* untuk mengubah *posting* tertentu	`GET`	users/{user_id}/blog/posts/{post_id}/edit
Menampilkan *link* untuk menghapus *posting* tertentu	`GET`	/users/{user_id}/blog/posts/{post_id}

Sumber: Hasil Perancangan

8. *Comments*

Resource comments digunakan untuk menampung dan mengatur komentar yang terdapat pada *posting* dari *blog* anggota. Daftar *action*, HTTP *method*, dan *path* yang digunakan pada *resource comments* dirinci dalam tabel 4.10.

Tabel 4.10 *Resource Comments*

Action	**HTTP *Method***	***Path***
Menambah komentar terhadap *posting* tertentu	`POST`	/users/{user_id}/blog/posts/{post_id}/comments
Menghapus komentar pada *posting* tertentu	`DELETE`	/users/{user_id}/blog/posts/{post_id}/comments/{comment_id}
Menampilkan *form* untuk menulis komentar terhadap *posting* tertentu	`GET`	/users/{user_id}/blog/posts/{post_id}
Menampilkan *link* untuk menghapus komentar pada *posting* tertentu	`GET`	/users/{user_id}/blog/posts/{post_id}

Sumber: Hasil Perancangan

9. *Hub*

Resource hub digunakan untuk menampilkan halaman utama yang digunakan anggota sebagai navigasi untuk mengakses fitur – fitur situs *social networking*. Daftar *action*, HTTP *method*, dan *path* yang digunakan pada *resource hub* dirinci dalam tabel 4.11.

Tabel 4.11 *Resource Hub*

Action	**HTTP *Method***	***Path***
Menampilkan *hub* anggota	`GET`	/users/{user_id}/hub

Sumber: Hasil Perancangan

10. *Community*

Resource friendships digunakan untuk menampilkan komunitas yang terdapat pada situs *social networking*. *Resource* ini dapat menampilkan daftar seluruh anggota situs *social networking*. Daftar *action*, HTTP *method*, dan *path* yang digunakan pada *resource community* dirinci dalam tabel 4.12.

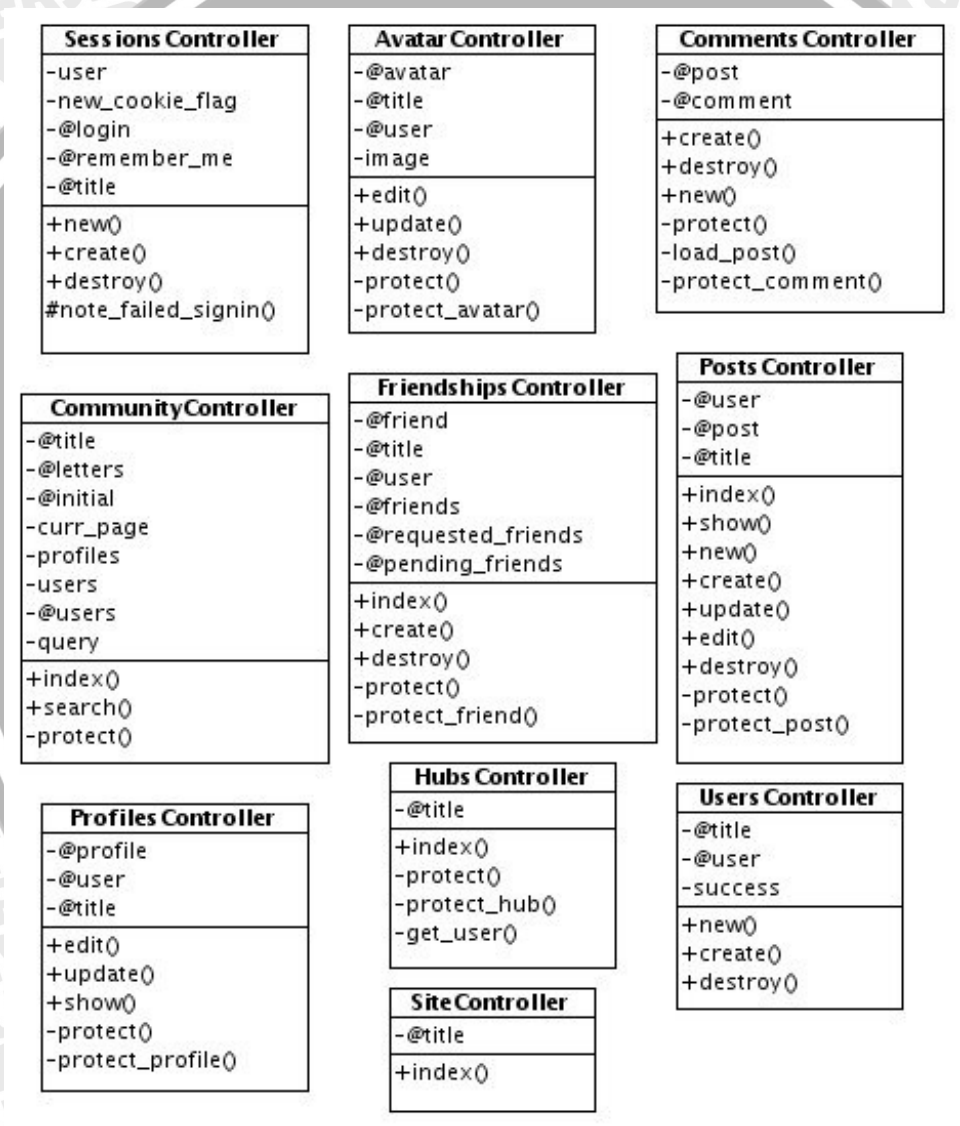
Tabel 4.12 *Resource Community*

Action	**HTTP *Method***	***Path***
Menampilkan halaman muka komunitas	`GET`	/community
Menampilkan daftar anggota berdasarkan abjad tertentu	`GET`	/community/index/{A..Z}
Menampilkan daftar anggota berdasarkan kata kunci tertentu	`GET`	/community/search?q={key_words}

Sumber: Hasil Perancangan

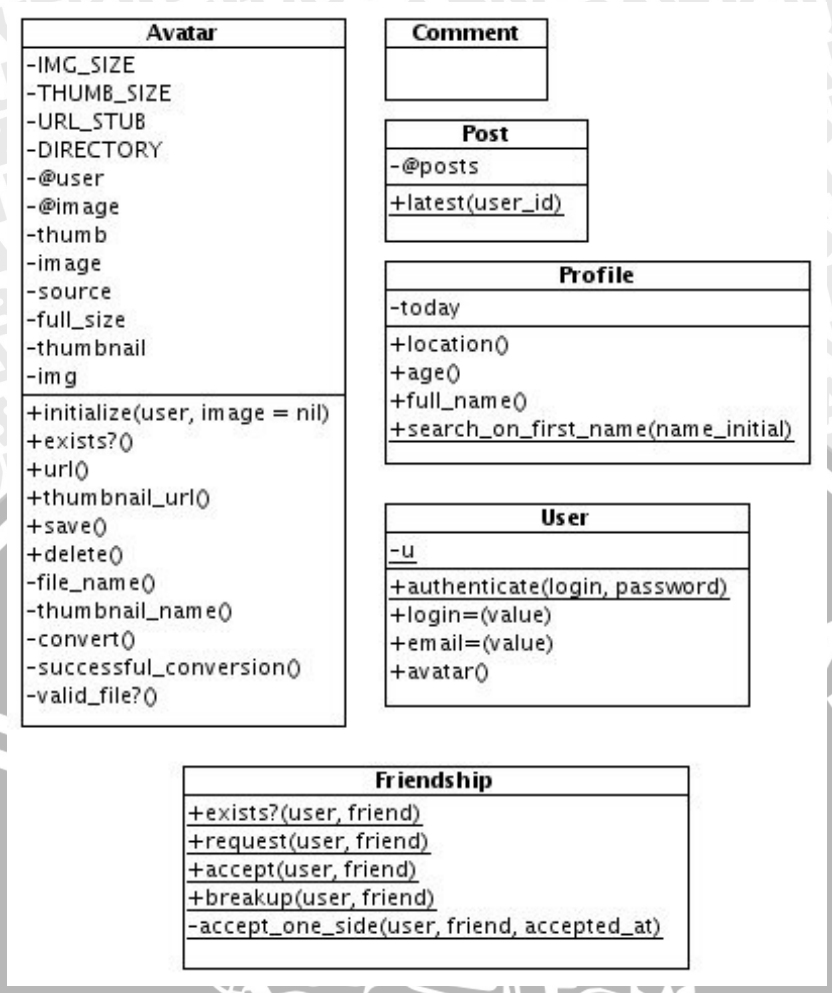
4.2.2 Perancangan Klas *(Class Diagram)*

Class Diagram digunakan untuk menggambarkan *class – class* yang digunakan beserta relasinya. Relasi tersebut dapat berupa pewarisan, asosiasi, dan sebagainya. Dalam pembuatan situs *social networking* pada skripsi ini, terdapat dua kelompok *class* utama, yaitu *Model* dan *Controller*. *Class diagram* untuk *Controller* sesuai dengan gambar 4.23. *Class diagram* untuk *Model* sesuai dengan gambar 4.24.



Gambar 4.23 *Class Diagram – Controllers*

Sumber: Hasil Perancangan



Gambar 4.24 *Class Diagram – Models*

Sumber: Hasil Perancangan

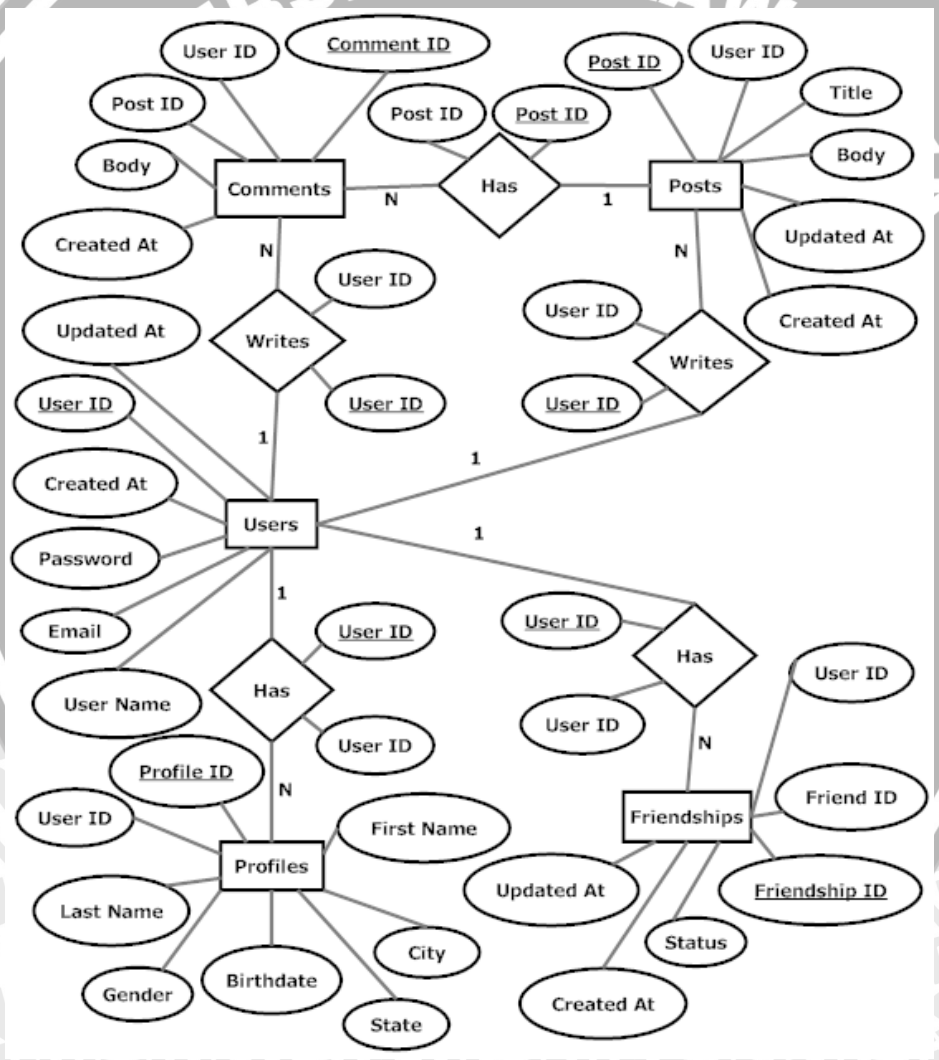
Framework Ruby On Rails memanfaatkan teknik *metaprogramming* untuk meningkatkan produktivitas programmer dengan cara meminimalisasi jumlah penulisan *code* yang berulang. Dengan teknik *metaprogramming*, sebagian *attributes* dan *operations* suatu *class* tidak ditulis secara eksplisit, melainkan ditulis secara dinamis pada saat program berjalan (*runtime*).

4.2.3 Perancangan Basis Data

Basis data yang digunakan adalah basis data tipe relasional. Pertama-tama perancangan dilakukan dengan membuat diagram ER, kemudian dilanjutkan dengan *conceptual modeling*, *logical modeling* dan *physical modeling*.

4.2.3.1 Diagram ER

Diagram ER dibuat untuk merepresentasikan data, yaitu entitas dan relasinya, secara abstrak dan konseptual. Diagram ER sistem dapat digambarkan melalui diagram seperti pada gambar 4.25.

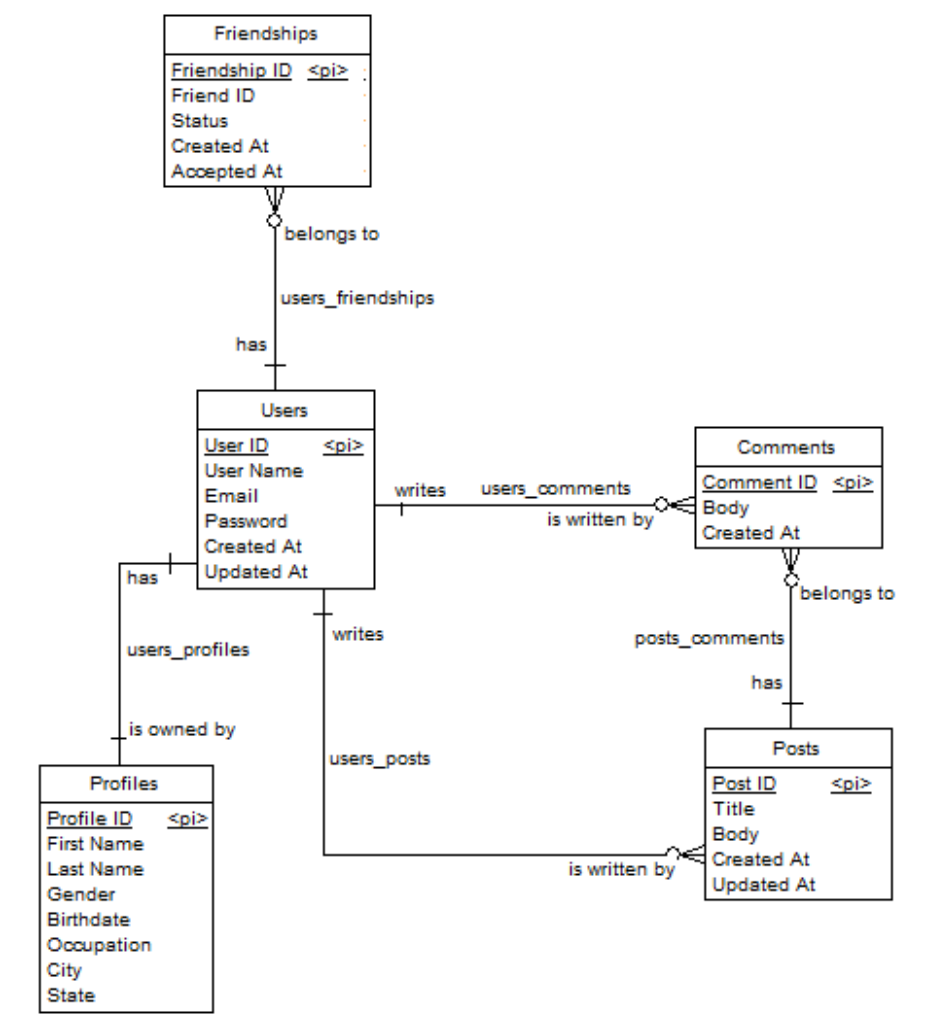


Gambar 4.25 Diagram ER Sistem

Sumber: Hasil Perancangan

4.2.3.2 *Conceptual Modeling*

Conceptual data model dibuat untuk merepresentasikan struktur basis data. Model ini berisi entitas, atribut, dan relasi yang terjadi antar entitas. *Conceptual data model* sistem dapat digambarkan melalui diagram seperti pada gambar 4.26.



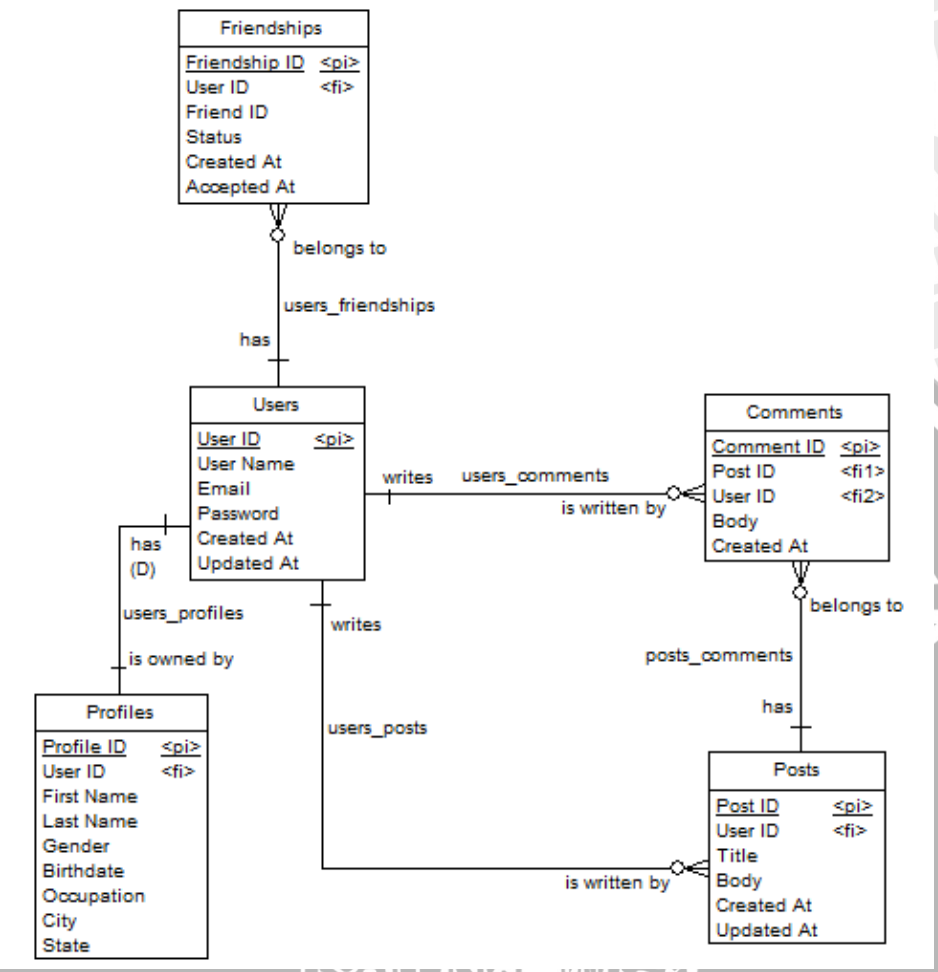
Gambar 4.26 *Conceptual Diagram* Sistem

Sumber: Hasil Perancangan

4.2.3.3 *Logical Modeling*

Logical data model dibuat dengan mengacu pada *conceptual data model*. *Logical data model* dibuat untuk melakukan validasi terhadap relasi yang terjadi antar entitas melalui *foreign identifier* maupun *intermediate entities*. *Logical data*

model sistem dapat digambarkan melalui diagram seperti pada gambar 4.27.

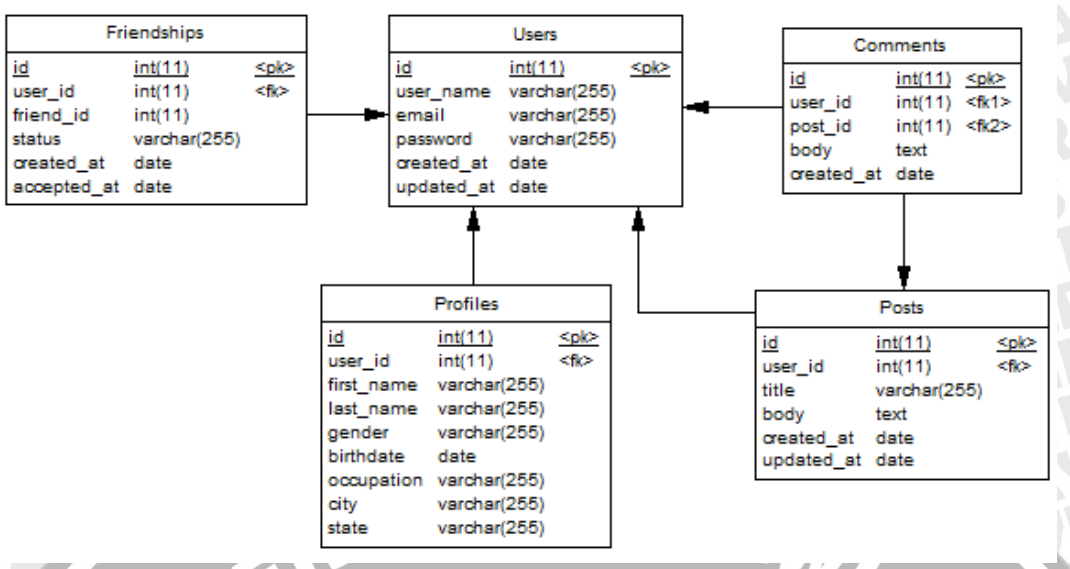


Gambar 4.27 *Logical Diagram* Sistem

Sumber: Hasil Perancangan

4.2.3.4 *Physical Modeling*

Physical data model dibuat dengan mengacu pada *logical data model. Physical data model* dibuat untuk mendefinisikan struktur basis data melalui sudut pandang implementasi. *Physical data model* disesuaikan dengan DBMS (*Database Management System*) yang akan digunakan pada saat implementasi, yaitu MySQL 5. *Physical data model* sistem dapat digambarkan melalui diagram seperti pada gambar 4.28.

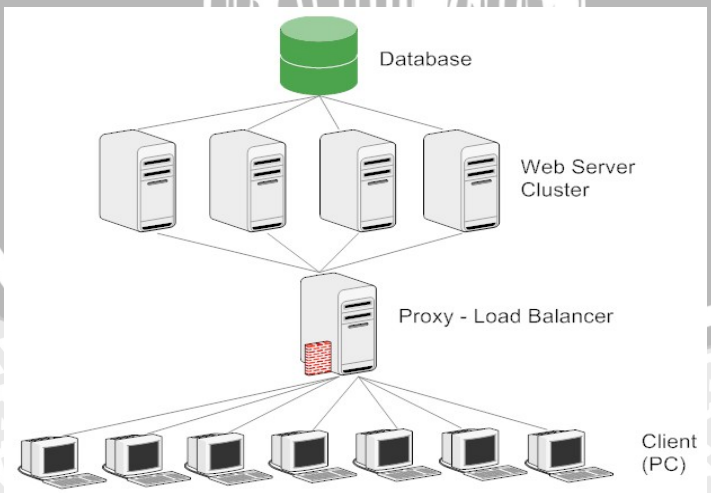


Gambar 4.28 *Physical Diagram* Sistem

Sumber: Hasil Perancangan

4.3 Perancangan *Plan* Sistem

Situs *social networking* merupakan aplikasi berbasis web yang menggunakan arsitektur *client – server.* Sistem dirancang untuk menjadi sebuah sistem dengan *scalability* yang baik, yaitu dengan menggunakan *cluster* pada *web server,* seperti yang ditunjukkan pada gambar 4.29. Bila sistem membutuhkan kinerja ekstra , maka sistem dapat dengan mudah menambahkan *web server* baru ke dalam *cluster.*



Gambar 4.29 Diagram *Plan* Sistem

Sumber: Hasil Perancangan

BAB V
IMPLEMENTASI

Bab ini membahas tentang implementasi perangkat lunak berdasarkan hasil yang telah didapatkan dari analisis kebutuhan dan proses perancangan perangkat lunak sebelumnya.

5.1 Spesifikasi Sistem

Penentuan spesifikasi sistem berdasarkan hasil analisis kebutuhan dan perancangan. Spesifikasi sistem dibagi menjadi dua, yaitu spesifikasi perangkat keras dan spesifikasi perangkat lunak.

5.1.1 Spesifikasi Perangkat Keras

Situs *social networking* merupakan jenis aplikasi *client-server*. Dalam skripsi ini, baik *client* maupun *server* terdapat dalam satu komputer yang sama. Pengembangan situs *social networking* menggunakan sebuah komputer dengan spesifikasi perangkat keras yang dijelaskan pada Tabel 5.1.

Tabel 5.1 Spesifikasi Perangkat Keras Komputer

Nama Komponen	Spesifikasi
Processor:	AMD Turion ML-34 @1.8 GHz
Memory (RAM):	1024 MB DDR-333
Hardisk:	60 GB @ 5400 rpm
Motherboard:	Acer, Inc. ATI R200 *chipset*
Graphics Adapter:	ATI Mobility Radeon X700 128MB VRAM

Sumber: Implementasi

5.1.2 Spesifikasi Perangkat Lunak

Pengembangan situs *social networking* menggunakan perangkat lunak dengan spesifikasi yang dijelaskan pada Tabel 5.2.

Tabel 5.2 Spesifikasi Perangkat Lunak Komputer

Sistem Operasi:	Ubuntu Linux 8.04.1 (Hardy Heron) 64-bit Kernel 2.6.24-19 x86_64 Ext3 *filesystem*
Bahasa Pemrograman:	Ruby (Ruby 1.8.6 x86_64 linux)
Framework:	Ruby On Rails 2.1.0
Web Server:	Mongrel 1.1.5 (Cluster) + Nginx (Load Balancer)
Basis Data:	MySQL 5.0.51a-3ubuntu5.3
IDE :	Netbeans 6.1
Pustaka Tambahan:	- ImageMagick (Linux) - Zentest (Ruby Gem) - Shoulda (Rails Plugin) - RESTful Authentication (Rails Plugin) - WillPaginate (Rails Plugin) - Ferret (Linux & Rails Plugin)
Web Browser:	- Mozilla Firefox 3.0.3 - Opera 9.61

Sumber: Implementasi

5.2 Implementasi *Class* Pada File Program

Masing – masing *class* yang telah dirancang pada proses perancangan direalisasikan dalam sebuah *file* *.rb. Tabel 5.3 menjelaskan mengenai pasangan antara *class* dengan *file* program yang digunakan untuk mengimplentasikannya.

Tabel 5.3 Implementasi *Class* Pada Kode Program *.rb

No	Komponen	Nama *Class*	Nama *File*
1.	Controller	AvatarsController	avatars_controller.rb
2.		BlogController	blog_controller.rb
3.		CommentsController	comments_controller.rb
4.		CommunityController	community_controller.rb
5.		FriendshipsController	friendships_controler.rb
6.		HubsController	hubs_controller.rb
7.		PostsController	posts_controller.rb
8.		ProfilesController	profiles_controller.rb
9.		SessionsController	sessions_controller.rb
10.		SiteController	site_controller.rb

11.	Model	UsersController	users_controller.rb
12.		Avatar	avatar.rb
13.		Comment	comment.rb
14.		Friendship	friendship.rb
15.		Post	post.rb
16.		Profile	profile.rb
17.		User	user.rb

Sumber: Implementasi

5.3 Implementasi *Uniform Interface* Pada *Resource*

Situs *social networking* dirancang dan diimplementasikan berdasarkan prinsip – prinsip *Representational State Transfer* (REST), diantaranya adalah *resource-oriented* dan *uniform interface*. Ruby On Rails memiliki suatu konvensi mengenai prinsip *uniform interface*, dimana masing – masing *resource* memiliki respon yang seragam terhadap HTTP *method* tertentu, seperti pada tabel 5.4.

Tabel 5.4 Hubungan Antara HTTP *Method* Dengan *Controller Method* Pada *Framework* Ruby On Rails

HTTP *Method*	***Controller Method***
`GET`	`index, show, new, edit`
`POST`	`create`
`PUT`	`update`
`DELETE`	`destroy`

Sumber: Implementasi

Konvensi seperti pada tabel di atas mendasari implementasi bagi seluruh *resource* yang ada pada situs *social networking*, seperti pada tabel 5.5.

Tabel 5.5 Implementasi R*esource* dan HTTP *Method*

Path	**HTTP Method**	**Controller**	**Method**
/	`GET`	SiteController	`index`
/users	`POST`	UsersController	`create`

/users/{user_id}	DELETE	UsersController	destroy
/users/new	GET	UsersController	new
/session	POST	SessionsController	create
/session	DELETE	SessionsController	destroy
/session/new	GET	SessionsController	new
/users/{user_id}/profile	GET	ProfilesController	show
/users/{user_id}/profile	PUT	ProfilesController	update
/users/{user_id}/profile/edit	GET	ProfilesController	edit
/users/{user_id}/avatar	PUT	AvatarsController	update
/users/{user_id}/avatar	DELETE	AvatarsController	destroy
/users/{user_id}/avatar/edit	GET	AvatarsController	edit
/users/{user_id}/friendships	GET	FriendshipsController	index
/users/{friend_id}/friendships	POST	FriendshipsController	create
/users/{user_id}/friendships/{friend_id}	DELETE	FriendshipsController	destroy
/users/{user_id}/blog/posts	GET	PostsController	index
/users/{user_id}/blog/posts	POST	PostsController	create
/users/{user_id}/blog/posts/{post_id}	GET	PostsController	show
/users/{user_id}/blog/posts/{post_id}	PUT	PostsController	update
/users/{user_id}/blog/posts/{post_id}	DELETE	PostsController	delete
/users/{user_id}/blog/posts/new	GET	PostsController	new
/users/{user_id}/blog/posts/{post_id}/edit	GET	PostsController	edit
/users/{user_id}/blog/posts/{post_id}/comments	POST	CommentsController	create
/users/{user_id}/blog/posts/{post_id}/comments/{comment_id}	DELETE	CommentsController	destroy
/users/{user_id}/hub	GET	HubsController	index
/community	GET	CommunityController	index
/community/index/{A...Z}	GET	CommunityController	index
/community/search?q={key_words}	GET	CommunityController	search[8]

Sumber: Implementasi

8 Dalam kasus *resource* pencarian (*search*) tidak menggunakan konvensi nama method standar REST pada *framework* Ruby On Rails (*index, new, show, edit*). Walaupun demikian, *resoure* tersebut tetap memenuhi prinsip REST karena menggunakan *method* GET yang bersifat *safe* dan *idempotent*.

5.4 Implementasi *Object Relational Mapping* (ORM) Pada *Basis Data*

Dengan teknik *metaprogramming*, semua nama kolom pada tabel secara dinamis dibaca oleh *ActiveRecord*. Demikian pula semua relasi antar tabel yang ada pada basis data situs *social networking* ditangani sepenuhnya oleh *ActiveRecord* melalui kumpulan *class Model* yang merupakan turunan dari modul *Acitive Record*. Dari hasil analisis kebutuhan dan perancangan, ada lima tabel yang harus diimplementasikan. Masing – masing tabel diwakili oleh satu *class AvtiveRecord*.

5.4.1 Implementasi Relasi Tabel *Users*

Dari hasil analisis kebutuhan dan perancangan, didapatkan bahwa tabel *users* memiliki relasi dengan tabel *profiles* (*one-to-one*), tabel *posts* (*one-to-many*), tabel *comments* (*one-to-many*), dan tabel *friendships* (*many-to-many*). Gambar 5.1 menunjukkan implementasi relasi tabel *users* pada *class User* yang menurun dari *class Active Record*.

```
class User < ActiveRecord::Base

has_one   :profile,     :dependent => :destroy
has_many  :posts,       :dependent => :destroy
has_many  :comments,    :dependent => :destroy
has_many  :friendships, :dependent => :destroy
```

Gambar 5.1 Implementasi *Class Model – User*
Sumber: Implementasi

5.4.2 Implementasi Relasi Tabel *Profiles*

Dari hasil analisis kebutuhan dan perancangan, didapatkan bahwa tabel *profiles* memiliki relasi dengan tabel *users* (*one-to-one*). Gambar 5.2 menunjukkan implementasi relasi tabel *profiles* pada *class Profile* yang menurun dari *class Active Record*.

```
class Profile < ActiveRecord::Base

belongs_to :user
```

Gambar 5.2 Implementasi *Class Model – Profile*

Sumber: Implementasi

5.4.3 Implementasi Relasi Tabel *Posts*

Dari hasil analisis kebutuhan dan perancangan, didapatkan bahwa tabel *posts* memiliki relasi dengan tabel *users* (*one-to-one*) dan tabel *comments* (*one-to-many*). Gambar 5.3 menunjukkan implementasi relasi tabel *posts* pada *class Post* yang menurun dari *class Active Record*.

```
class Post < ActiveRecord::Base

belongs_to  :user
has_many    :comments, :dependent => :destroy
```

Gambar 5.3 Implementasi *Class Model – Post*

Sumber: Implementasi

5.4.4 Implementasi Relasi Tabel *Comments*

Dari hasil analisis kebutuhan dan perancangan, didapatkan bahwa tabel *comments* memiliki relasi dengan tabel *users* (*many-to-one*) dan tabel *posts* (*many-to-one*). Gambar 5.4 menunjukkan implementasi relasi tabel *comments* pada *class Comment* yang menurun dari *class Active Record*.

```
class Comment < ActiveRecord::Base

belongs_to  :user
belongs_to  :post
```

Gambar 5.4 Implementasi *Class Model – Comment*

Sumber: Implementasi

5.4.5 Implementasi Relasi Tabel *Friendships*

Dari hasil analisis kebutuhan dan perancangan, didapatkan bahwa tabel *friendships* memiliki relasi dengan tabel *users* (*many-to-many*). Karena relasi *many-to-many*, maka dibutuhkan bantuan sebuah *class Model* "palsu" dengan nama *friend* yang mengacu pada *class user*. Gambar 5.5 menunjukkan implementasi relasi tabel *friendships* pada *class Friendship* yang menurun dari *class Active Record*.

```
class Friendship < ActiveRecord::Base

belongs_to :user
belongs_to :friend, :class_name => "User",
      :foreign_key => "friend_id"
```

Gambar 5.5 Implementasi *Class Model – Friendship*

Sumber: Implementasi

5.5 Implementasi Algoritma

Inti dari situs *social networking* adalah adanya relasi atau hubungan antara satu anggota dengan anggota yang lain melalui suatu pertemanan (*friendships*). Untuk menjalin hubungan pertemanan tersebut, harus ada kesepakatan dari dua belah pihak, yaitu dari pihak pengajak dan penerima.

5.5.1 Algoritma *Request Friendship*

Proses pertama untuk menjalin pertemanan dengan anggota lain adalah melakukan permintaan (*request*) terhadap anggota tersebut. Proses *request friendship* terdiri dari transaksi basis data. Algoritma ini diterapkan pada *class Friendship*. Algoritma *request friendship* ditunjukkan pada gambar 5.6.

```
implelentasi algoritma "request friendship"

METHOD request PARAMETER user, friend

DESCRIPTION
1 UNLESS user == friend or Friendship.exists?(user, friend)
2  TRANSACTION DO
3    CALL create(user,friend,'pending')
4    CALL create(friend,user,'requested')
5  END
6 END request
```

Gambar 5.6 Algoritma *Request Friendship*

Sumber: Implementasi

Penjelasan algoritma *request friendship* pada gambar 5.6, yaitu:

1. Baris 1 memastikan bahwa anggota yang melakukan permintaan bukanlah orang yang sama dengan teman yang dimaksud. Selain itu baris 1 juga memastikan bahwa kedua anggota tidak memiliki status pertemanan sebelumnya.
2. Baris 2 – 5 menjelaskan transaksi yang terjadi pada basis data. Transaksi memastikan bahwa kedua proses yang berada di dalamnya berhasil semua atau gagal semua. Transaksi tersebut menghasilkan data baru.

5.5.2 Algoritma *Accept Friendship*

Proses kedua untuk menjalin pertemanan dengan anggota lain adalah menerima (*accept*) ajakan pertemanan dari anggota lain. Proses *accept friendship* terdiri dari transaksi basis data. Algoritma ini diterapkan pada *class* Friendship. Algoritma accept friendship ditunjukkan pada gambar 5.7.

```
implelentasi algoritma "accept friendship"

METHOD accept PARAMETER user, friend

DESCRIPTION
1 TRANSACTION DO
2   accepted_at <- Time.now
3   CALL accept_one_side(user, friend, accepted_at)
4   CALL accept_one_side(friend, user, accepted_at)
5 END accept
```

Gambar 5.7 Algoritma *Accept Friendship*

Sumber: Implementasi

Penjelasan algoritma *request friendship* pada gambar 5.7 , yaitu:

1. Baris 1 – 5 menjelaskan transaksi yang terjadi pada basis data. Transaksi memastikan bahwa ketiga proses yang berada di dalamnya berhasil semua atau gagal semua. Dalam transakti tersebut terjadi sejumlah pembaharuan data.

5.5.3 Algoritma *Breakup Friendship*

Ketika ada ajakan pertemanan, seorang anggota dapat membatalkan maupun menolaknya melalui proses *breakup friendship*. Selain itu, proses ini juga berlaku untuk menghapus pertemanan yang telah terjalin. Proses *breakup friendship* terdiri dari transaksi basis data. Algoritma ini diterapkan pada *class Friendship.* Algoritma breakup friendship ditunjukkan pada gambar 5.8.

```
implelentasi algoritma "breakup friendship"

METHOD breakup PARAMETER user, friend

DESCRIPTION
1 TRANSACTION DO
2   CALL destroy(find_by_user_id_and_friend_id(user, friend))
3   CALL destroy(find_by_user_id_and_friend_id(friend, user))
4 END
```

Gambar 5.8 Algoritma *Breakup Friendship*

Sumber: Implementasi

Penjelasan algoritma *request friendship* pada gambar 5.8, yaitu:

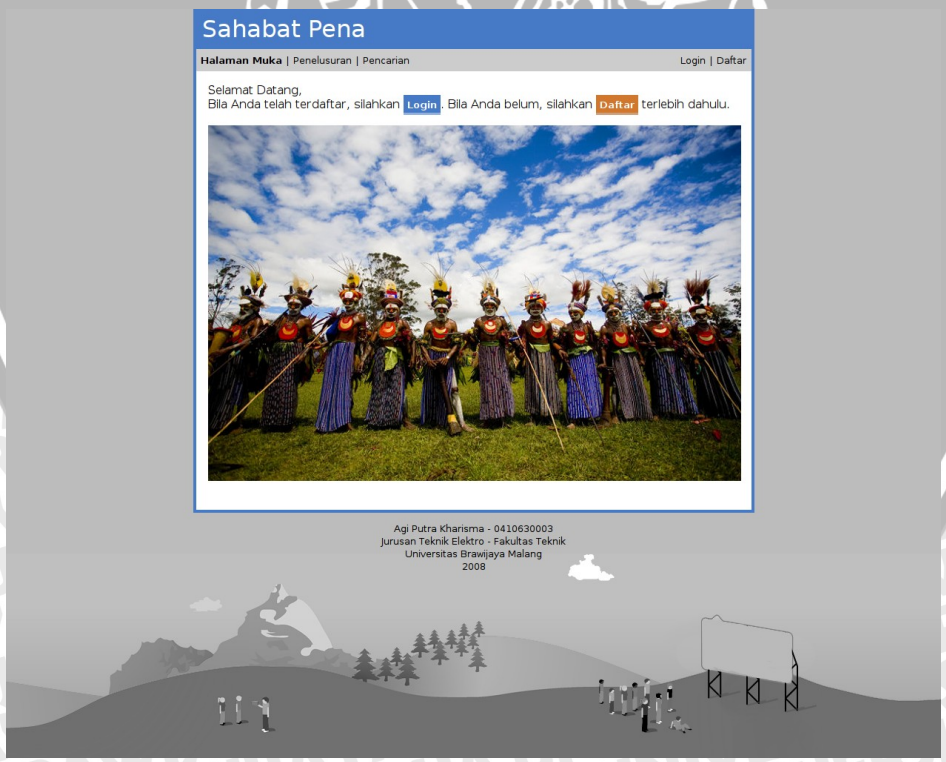
1. Baris 1 – 4 menjelaskan transaksi yang terjadi pada basis data. Transaksi memastikan bahwa ketiga proses yang berada di dalamnya berhasil semua atau gagal semua. Dalam transaksi tersebut terjadi penghapusan sejumlah data.

5.6 Implementasi Antarmuka

Pada aplikasi ini, antarmuka dibuat sesuai dengan kebutuhan fungsional yang harus disediakan oleh sistem. Tujuan utama pengembangan antarmuka adalah untuk memberikan kemudahan bagi pengguna (*user*) dalam menggunakan aplikasi yang telah dibangun.

5.6.1 Implementasi Antarmuka Halaman Muka

Halaman muka merupakan halaman yang ditampilkan ketika pengguna mengakses *path root* dari aplikasi situs *social networking*. Implementasi dari antarmuka halaman muka ditunjukkan pada gambar 5.9.



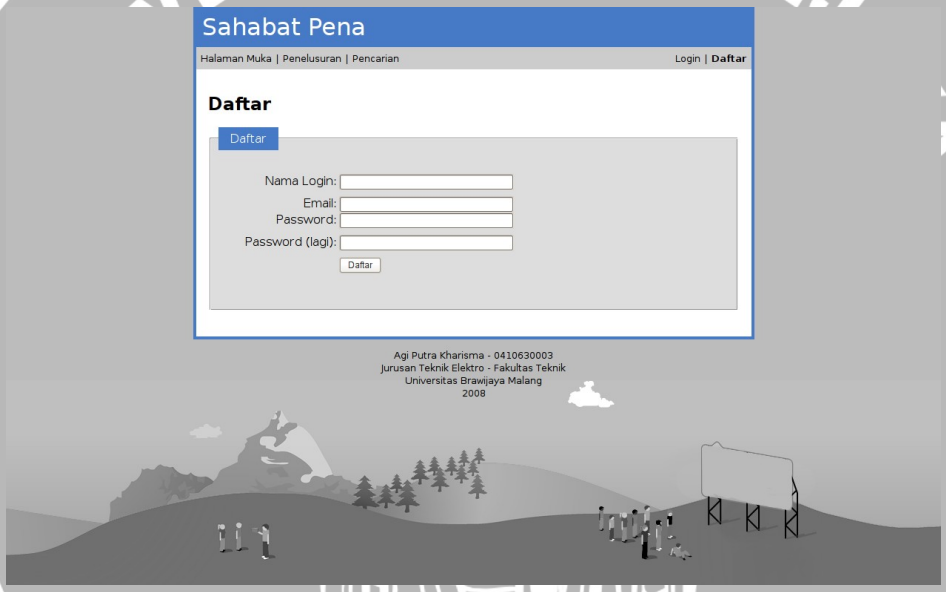
Gambar 5.9 Antarmuka Halaman Muka

Sumber: Implementasi

Sesuai dengan salah satu prinsip REST, yaitu *connectedness,* maka pada halaman ini dan seluruh halaman lainnya, pengguna akan dituntun menuju halaman lainnya melalui *link – link* yang terdapat pada bagian navigasi.

5.6.2 Implementasi Antarmuka Halaman Daftar

Atarmuka halaman daftar merupakan fasilitas yang digunakan oleh pengguna untuk mendaftarkan diri menjadi anggota situs *social networking*. Pengguna diharuskan mengisi beberapa syarat yang dibutuhkan, yaitu nama *login*, *password*, dan alamat *email*. Implementasi dari antarmuka halaman daftar ditunjukkan pada gambar 5.10.



Gambar 5.10 Antarmuka Halaman Daftar
Sumber: Implementasi

5.6.3 Implementasi Antarmuka Halaman Profil Anggota

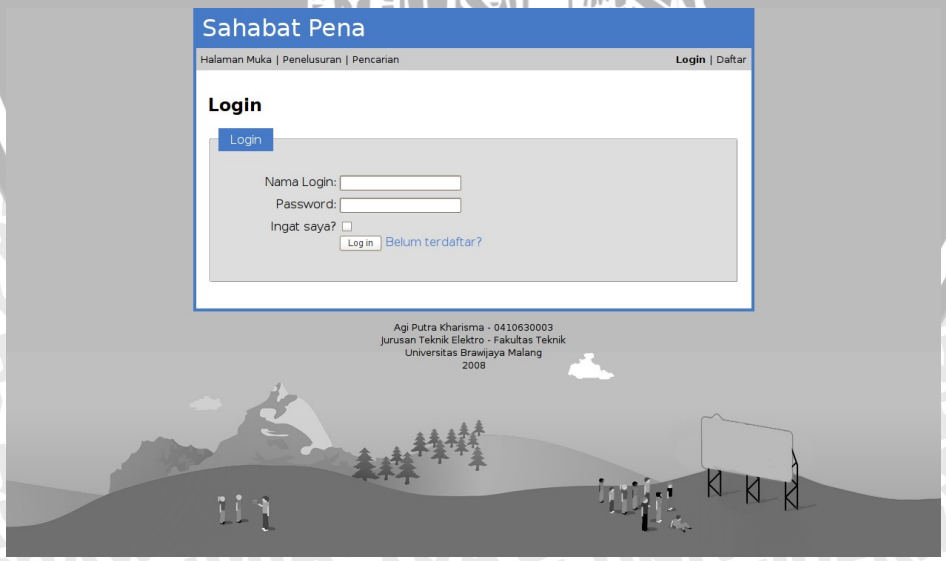
Antarmuka halaman profil anggota memuat profil, *avatar, link blog*, alamat *email*, dan *link* teman. Pengguna yang telah terdaftar dapat mengajak pertemanan melalui halaman profil anggota yang dikehendaki. Implementasi dari antarmuka halaman muka ditunjukkan pada gambar 5.11.

Gambar 5.11 Antarmuka Halaman Profil Anggota

Sumber: Implementasi

5.6.4 Implementasi Antarmuka Halaman *Login*

Antarmuka halaman *login* merupakan fasilitas yang digunakan bagi pengguna untuk melakukan proses autentikasi. Implementasi dari antarmuka halaman *login* ditunjukkan pada gambar 5.12.

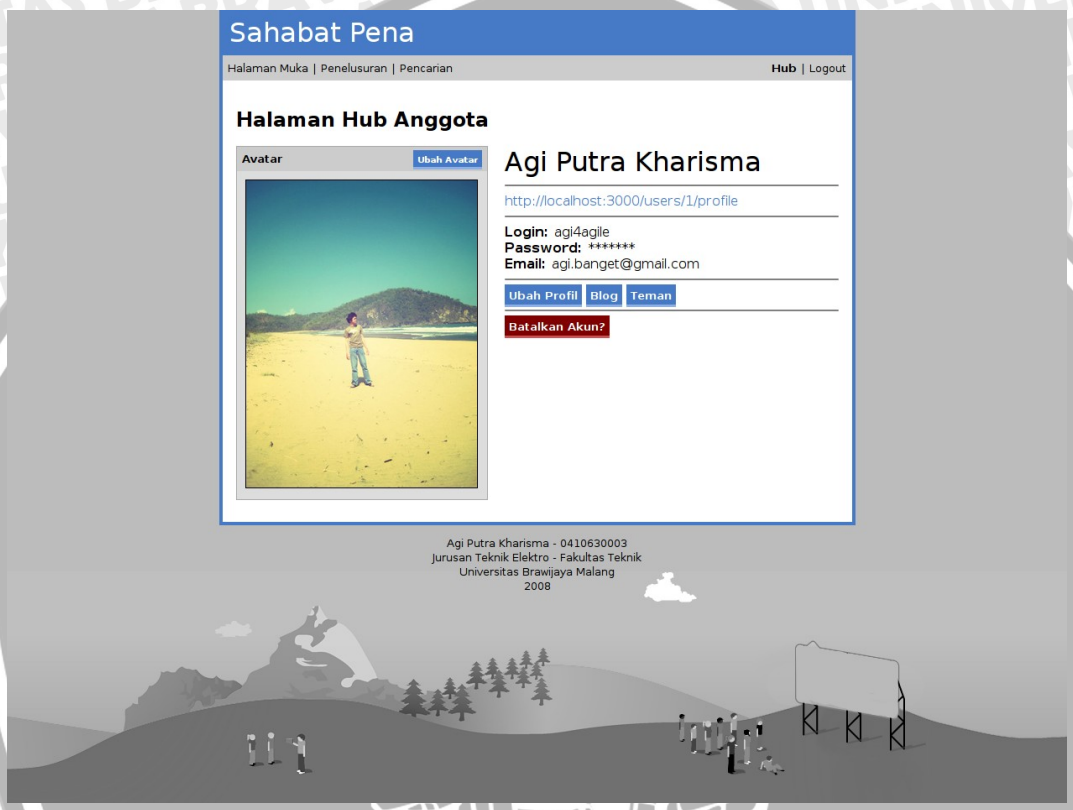


Gambar 5.12 Antarmuka Halaman *Login*

Sumber: Implementasi

5.6.5 Implementasi Antarmuka Halaman *Hub*

Halaman hub merupakan halaman utama bagi pengguna yang telah terdaftar sebagai anggota situs *social networking*. Dari halaman ini, pengguna (anggota) dapat mengelola profil, *avatar*, *blog*, dan membatalkan akun. Implementasi dari antarmuka halaman *hub* ditunjukkan pada gambar 5.13.



Gambar 5.13 Antarmuka Halaman *Hub*

Sumber: Implementasi

5.6.6 Implementasi Antarmuka Halaman Ubah Profil

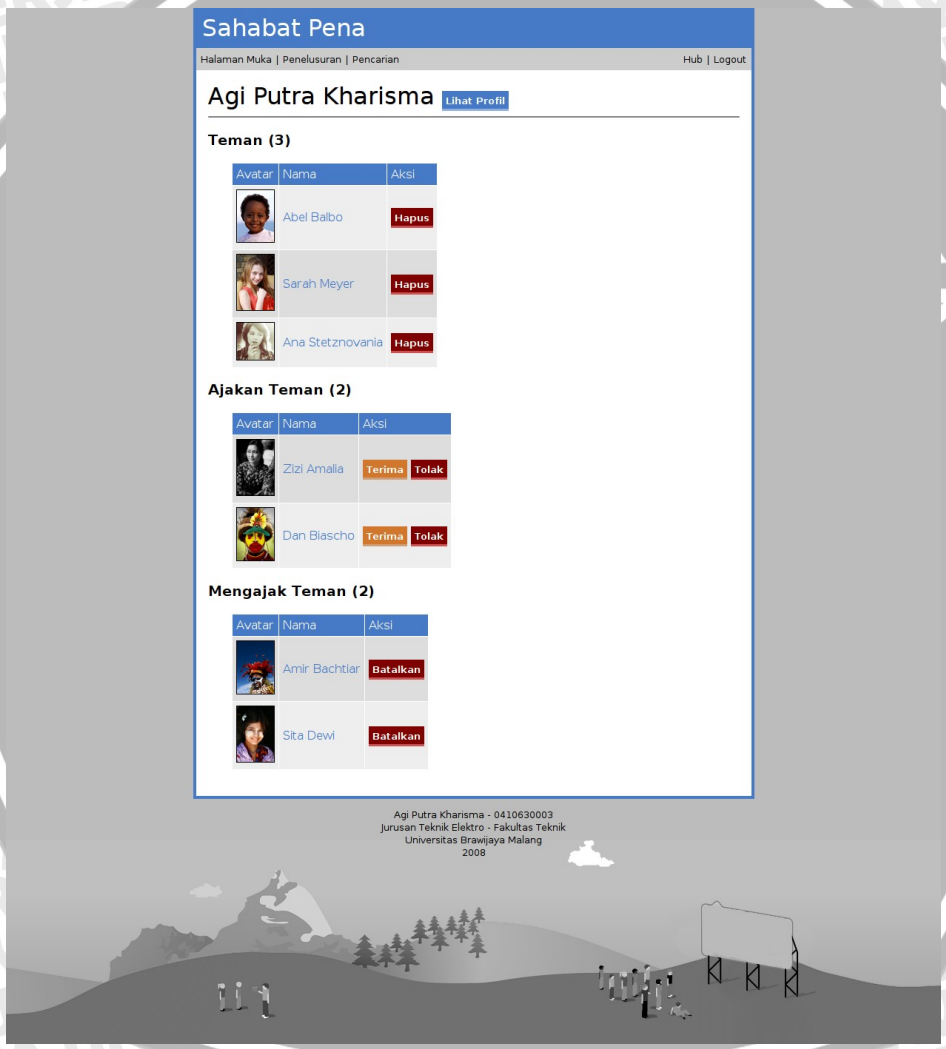
Halaman ubah profil digunakan untuk melakukan perubahan terhadap profil. Perubahan tersebut disimpan dalam basis data. Halaman ini hanya dapat diakses oleh yang pihak yang berwenang saja. Implementasi dari antarmuka halaman ubah profil ditunjukkan pada gambar 5.14.

Gambar 5.14 Antarmuka Halaman Ubah Profil

Sumber: Implementasi

5.6.7 Implementasi Antarmuka Halaman Pertemanan

Halaman pertemanan adalah halaman yang memuat daftar semua teman yang dimiliki oleh seorang anggota. Halaman ini dapat diakses oleh semua pegunjung, namun tidak untuk fitur – fitur tertentu yang dikhususkan bagi pihak yang berwenang saja. Fitur tersebut adalah menghapus teman, menerima ajakan pertemanan, membatalkan ajakan pertemanan, dan menolak ajakan pertemanan. Implementasi dari antarmuka halaman pertemanan ditunjukkan pada gambar 5.15.



Gambar 5.15 Antarmuka Halaman Pertemanan

Sumber: Implementasi

5.6.8 Implementasi Antarmuka Halaman *Blog*

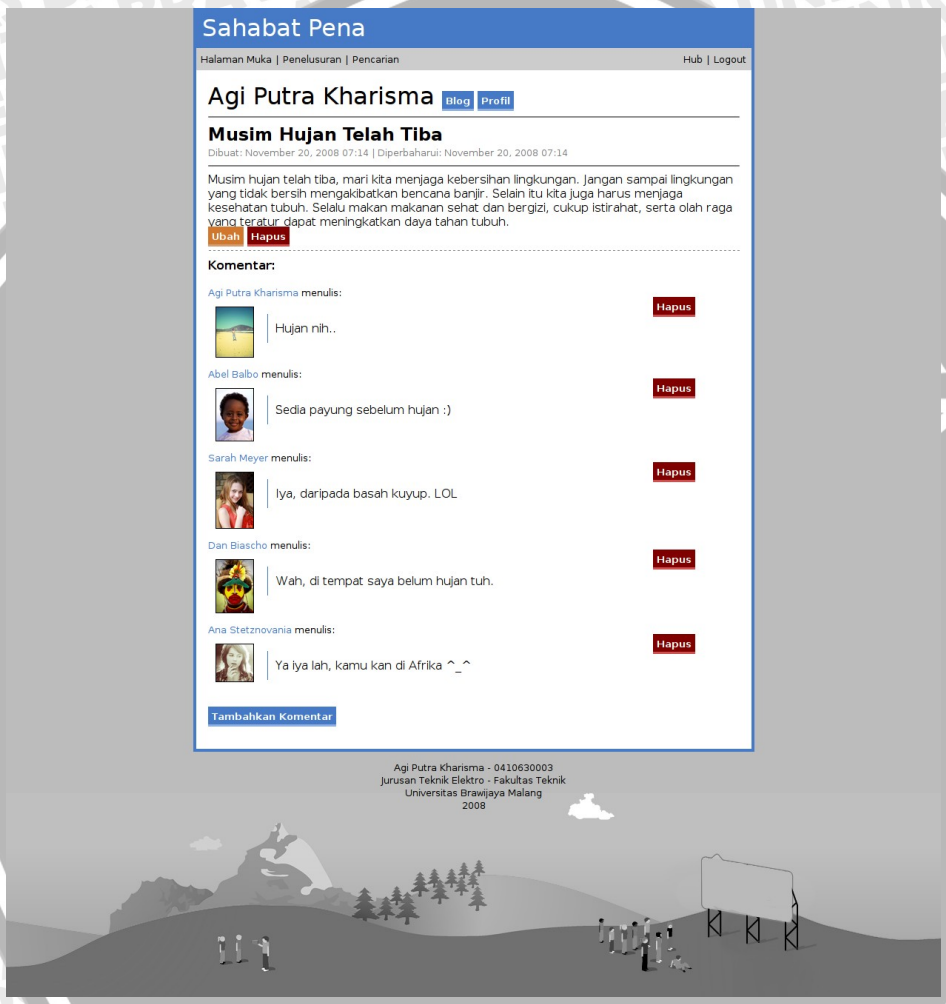
Halaman *blog* berisi daftar kiriman (*posting*) yang dibuat oleh seorang pengguna. Halaman ini dapat diakses oleh semua pegunjung, namun tidak untuk fitur – fitur tertentu yang dikhususkan bagi pihak yang berwenang saja. Fitur tersebut adalah *link* untuk membuat *posting* baru. Implementasi dari antarmuka halaman *blog* ditunjukkan pada gambar 5.16.

Gambar 5.16 Antarmuka Halaman Blog

Sumber: Implementasi

5.6.9 Implementasi Antarmuka Halaman *Posting*

Halaman ini menampilkan sebuah *posting* tertentu beserta komentar yang menyertainya. Pengguna yang berwenang dapat menulis maupun menghapus komentar melalui halaman ini. Implementasi dari antarmuka halaman *posting* ditunjukkan pada gambar 5.17.



Gambar 5.17 Antarmuka Halaman *Posting*

Sumber: Implementasi

5.6.10 Implementasi Antarmuka Halaman Membuat *Posting* Baru

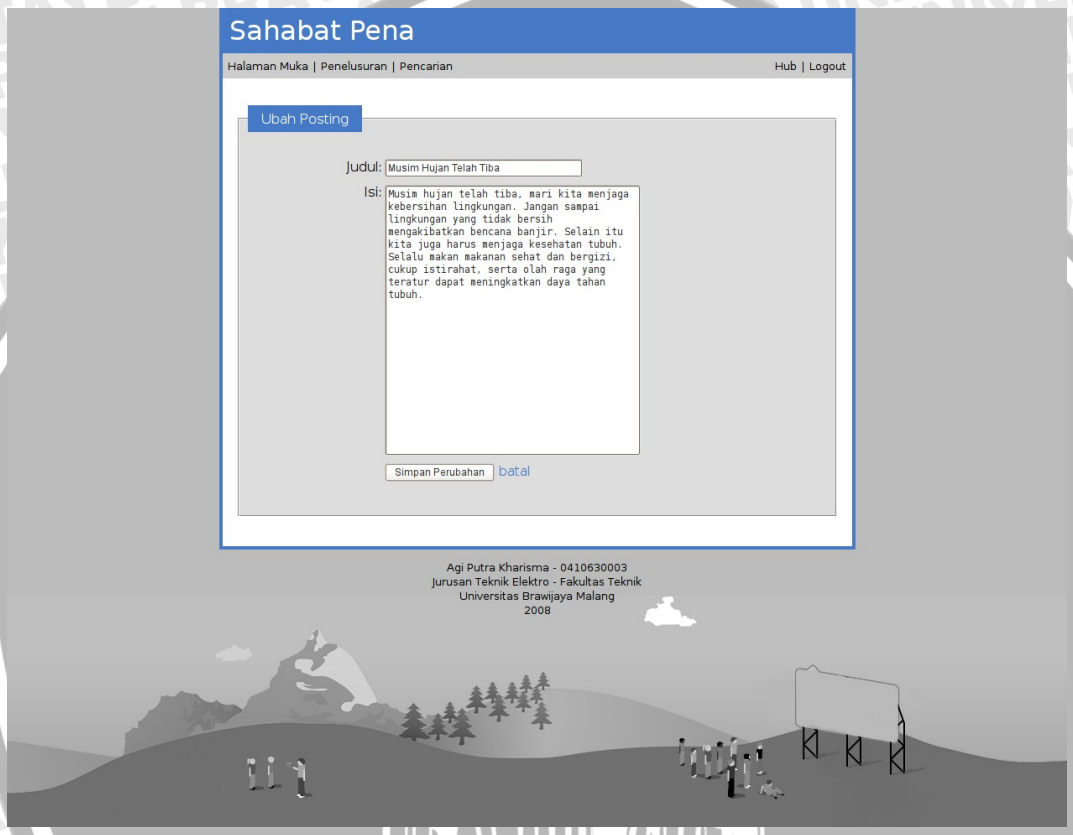
Halaman ini digunakan untuk membuat *posting* baru yang akan ditampilkan dalam bentuk *blog*. Implementasi dari antarmuka halaman membuat *posting* baru ditunjukkan pada gambar 5.18.

Gambar 5.18 Antarmuka Halaman Membuat *Posting* Baru

Sumber: Implementasi

5.6.11 Implementasi Halaman Ubah *Posting*

Halaman ini digunakan untuk mengubah suatu *posting*. Pada dasarnya, *form* yang terdapat pada halaman ini memiliki kolom yang sama dengan *form* yang digunakan untuk membuat *posting* baru. Implementasi dari antarmuka halaman ubah profil ditunjukkan pada gambar 5.19.

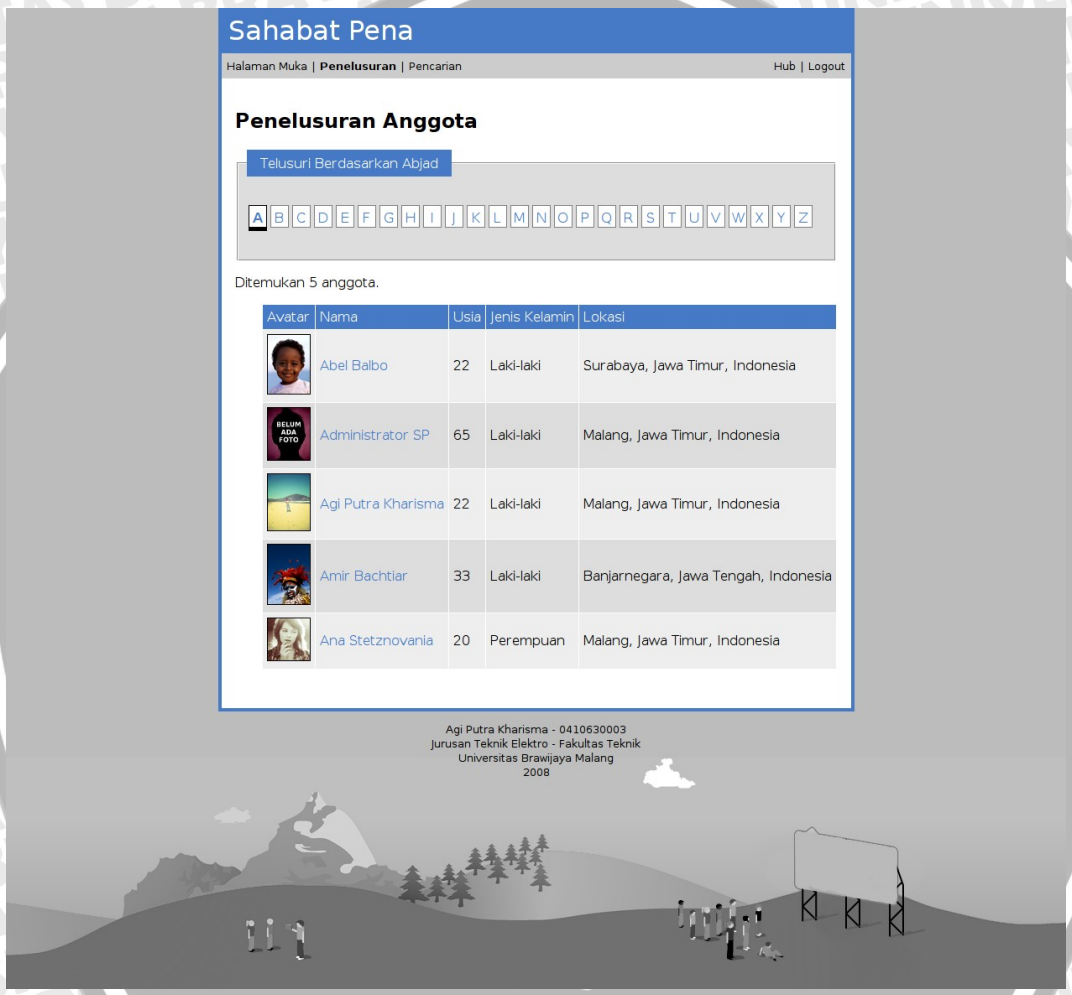


Gambar 5.19 Antarmuka Halaman Ubah *Posting*

Sumber: Implementasi

5.6.12 Implementasi Halaman Penelusuran Anggota

Halaman penelusuran anggota digunakan untuk menelusuri daftar anggota yang ada berdasarkan abjad. Abjad yang digunakan adalah abjad dari nama depan anggota. Halaman ini dikhususkan bagi anggota yang telah *login*. Implementasi dari antarmuka halaman penelusuran anggota ditunjukkan pada gambar 5.20.

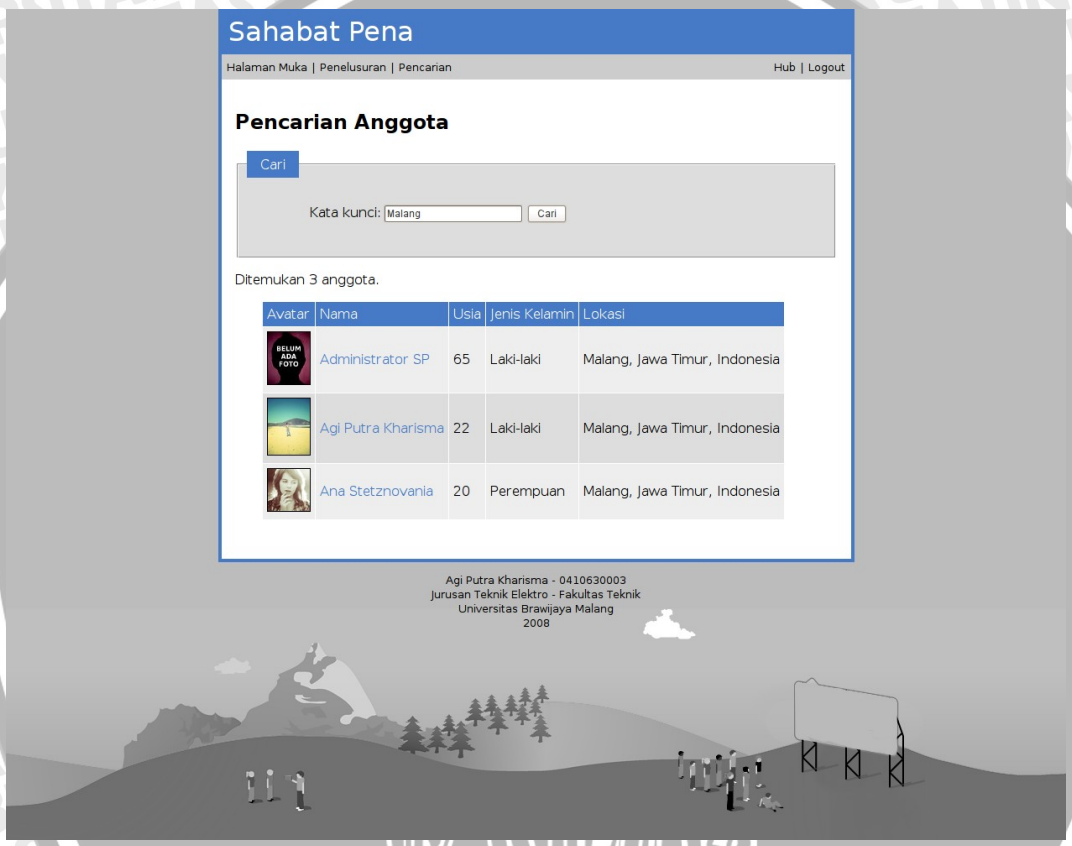


Gambar 5.20 Antarmuka Penelusuran Anggota

Sumber: Implementasi

5.6.13 Implementasi Halaman Pencarian Anggota

Halaman pencarian anggota digunakan untuk melakukan pencarian anggota. Pengguna dapat menggunakan kata kunci berdasarkan nama, jenis kelamin, dan lokasi anggota. Halaman ini dikhususkan bagi anggota yang telah *login*. Implementasi dari antarmuka halaman pencarian anggota ditunjukkan pada gambar 5.21.



Gambar 5.21 Antarmuka Halaman Pencarian Anggota

Sumber: Implementasi

5.7 Kendala Dalam Implementasi

Dalam implementasi terdapat kendala, yaitu *web browser* sebagai *client* dari situs *social networking* hanya mendukung HTTP *method* GET dan POST saja, sehingga sistem harus mensimulasikan *request* yang mengguakan HTTP *method* PUT dan DELETE. Pada proses implementasi, *request* dengan *method* PUT maupun DELETE disimulasikan ke dalam *method* POST dengan bantuan *hidden variable* yang berfungsi sebagai penanda jenis *method* yang digunakan.

BAB VI
PENGUJIAN DAN ANALISIS

Pada bab ini dilakukan proses pengujian dan analisis terhadap situs *social networking* yang dibangun menggunakan prinsip *Representational State Transfer.* Pengujian dilakukan dengan dua teknik, yaitu pengujian unit (*unit test*) dan pengujian fungsional (*functional test*). Analisis dilakukan dengan mengkaji tentang prinsip *Representational State Transfer* dan perbandingannya dengan RPC (*Remote Procedural Call*) *style*.

6.1 Pengujian

Proses pengembangan perangkat lunak dilakukan menggunakan metode *Behaviour Driven Development* dan *Interative and Incremental*, sehingga pada prakteknya perngujian dilakukan secara berkesinambungan dan berulang dengan proses perancangan maupun implementasi. Pengujian dilakukan menggunakan jenis *black-box testing* yang mengacu pada kebutuhan fungsional sistem, sedangkan kasus uji ditentukan berdasarkan perilaku (*behaviour*) sistem. Selain itu juga dilakukan pengujian terhadap performa dan skalabilitas sistem.

6.1.1 Pengujian Unit

Pengujian unit pada *framework* Ruby on Rails pada umumnya dilakukan untuk menguji *class - class Model.* Ada enam *class Model* yang diuji, yaitu *User, Profile, Post, Comment, Friendship, dan Avatar.*

6.1.1.1 Pengujian *Class User*

– Spesifikasi:
- Harus memiliki *method* untuk proses autentikasi
- Harus dapat memiliki satu *Profile*
- Harus dapat memiliki banyak *Posts*
- Harus dapat memiliki banyak *Comments*
- Harus dapat memiliki banyak *Friendships*
- Harus dapat memiliki banyak *friends*
- Harus dapat memiliki banyak *pending_friends*

- Harus dapat memiliki banyak *requested_friends*

6.1.1.2 Pengujian *Class Profile*

Spesifikasi:
- Harus memiliki *method* untuk menampilkan nama lengkap
- Harus memiliki *method* untuk menghitung usia
- Harus mengacu pada satu *User* tertentu

6.1.1.3 Pengujian *Class Posts*

Spesifikasi:
- Harus mengacu pada satu *User* tertentu
- Harus dapat memiliki banyak *Comments*
- Harus menyertakan atribut *title* dan *body*

6.1.1.4 Pengujian *Class Comment*

Spesifikasi:
- Harus mengacu pada satu *Post* tertentu
- Harus mengacu pada satu *User* tertentu
- Harus menyertakan atribut *user_id, post_id, body*

6.1.1.5 Pengujian *Class Friendship*

Spesifikasi:
- Harus memiliki *method* untuk mengetahui ada/tidak pertemanan
- Harus memiliki *method* untuk meminta pertemanan
- Harus memiliki *method* untuk menerima pertemanan
- Harus memiliki *method* untuk menghapus pertemanan
- Harus mengacu pada *User* tertentu
- Harus menyertakan atribut *user_id, friend_id*

6.1.1.6 Pengujian *Class Avatar*

Spesifikasi:
- Harus memiliki *method* untuk menyimpan *avatar* dan *thumbnail*
- Harus memiliki *method* untuk menghapus *avatar* dan *thumbnail*
- Harus memiliki *method* untuk menghasilkan URL *avatar*
- Harus memiliki *method* untuk menghasilkan URL *thumbnail*

6.1.2 Pengujian Fungsional

Pengujian fungsional pada *framework* Ruby On Rails pada umumnya dilakukan untuk menguji *class - class Controller.* Ada sepuluh *class Controller* yang diuji, yaitu *SiteController, UsersController, ProfilesController, SessionsController, HubsController, PostsController, CommentsController, FriendshipsController, AvatarsController,* dan *CommunityController.*

Tabel 6.1 Pengujian *Class SiteController*

1	Skenario:		Mengakses halaman *index* `GET /`
	a	Konteks:	• Halaman *index*
		Hasil:	• Menampilkan halaman dengan judul tertentu
2	Skenario		• Mengakses navigasi dari semua halaman
	a	Konteks:	• Semua halaman • Pengguna sudah *login*
		Hasil:	• Menampilkan *link logout* • Menampilkan *link hub* • Tidak menampilkan *link login* • Tidak menampilkan *link* daftar
	b	Konteks:	• Semua halaman • Pengguna belum *login*
		Hasil:	• Menampilkan *link login* • Menampilkan *link* daftar • Tidak menampilkan *link logout* • Tidak menampilkan *link hub*

Sumber: Pengujian

Tabel 6.2 Pengujian *Class UsersController*

1	Skenario:		Mengakses halaman pendaftaran `GET /users/new`
	a	Konteks:	• Halaman pendaftaran • Pengguna sudah *login*
		Hasil:	• Dialihkan ke halaman *hub* • Menampilkan pesan kesalahan

	b	Konteks:	• Halaman pendaftaran • Pengguna belum *login*
		Hasil:	• Menampilkan halaman pendaftaran dengan judul tertentu • Menampilkan *form* pendaftaran
2	Skenario:		Melakukan pendaftaran `POST` /users
	a	Konteks:	• Pendaftaran • Pengguna sudah *login*
		Hasil:	• Tidak terdaftar sebagai anggota baru • Dialihkan ke halaman *hub* • Menampilkan pesan kesalahan
	b	Konteks:	• Pendaftaran • Pengguna belum *login* • Mengisi *form* secara lengkap dan benar
		Hasil:	• Terdaftar sebagai anggota baru • Login secara otomatis • Dialihkan ke halaman *hub* • Menampilkan pesan
	c	Konteks:	• Pendaftaran • Pengguna belum *login* • Tidak menyertakan nama *login*
		Hasil:	• Tidak terdaftar sebagai anggota baru • Menampilkan pesan kesalahan
	d	Konteks:	• Pendaftaran • Pengguna belum *login* • Tidak menyertakan *password*
		Hasil:	• Tidak terdaftar sebagai anggota baru • Menampilkan pesan kesalahan
	e	Konteks:	• Pendaftaran • Pengguna belum *login* • Tidak menyertakan konfirmasi *password* yang tepat
		Hasil:	• Tidak terdaftar sebagai anggota baru • Menampilkan pesan kesalahan
	f	Konteks:	• Pendaftaran • Pengguna belum *login* • Tidak menyertakan alamat *email*
		Hasil:	• Tidak terdaftar sebagai anggota baru • Menampilkan pesan kesalahan

3	Skenario:		Menghapus akun anggota `DELETE` /users/{user_id}
	a	Konteks:	• Penghapusan • Pengguna belum *login*
		Hasil	• Dialihkan ke halaman *login* • Akun tidak terhapus
	b	Konteks:	• Penghapusan • Pengguna sudah *login* • Menghapus akun anggota lain
		Hasil	• Dialihkan ke halaman *hub* • Menampilkan pesan kesalahan • Akun tidak terhapus
	c	Konteks:	• Penghapusan • Pengguna sudah *login* • Menghapus akun milik sendiri
		Hasil	• Akun terhapus • Pertemanan terhapus • *Avatar* terhapus • Dialihkan ke halaman muka • Menampilkan pesan • *Logout* secara otomatis
	d	Konteks:	• Penghapusan • Pengguna sudah *login* • *Login* sebagai administrator • Menghapus akun anggota lain
		Hasil	• Akun terhapus • Pertemanan terhapus • *Avatar* terhapus • Dialihkan ke halaman muka • Menampilkan pesan • Tetap *login* sebagai administrator

Sumber: Pengujian

Tabel 6.3 Pengujian *Class ProfilesController*

1	Skenario:		Mengakses halaman profil anggota `GET` /users/{user_id}/profile
	a	Konteks:	• Halaman profil anggota
		Hasil:	• Menampilkan halaman profil anggota dengan judul tertentu

	b	Konteks:	• Halaman profil anggota • Pengguna sudah *login* • Halaman profil anggota lain (bukan teman)
		Hasil:	• Menampilkan *link* untuk menjadikan teman
	c	Konteks:	• Halaman profil anggota • Pengguna sudah *login* • Halaman profil anggota lain (teman)
		Hasil:	• Menampilkan status pertemanan
2	Skenario:		Mengakses halaman ubah profil `GET` /users/{user_id}/profile/edit
	a	Konteks:	• Halaman ubah profil • Pengguna belum *login*
		Hasil	• Dialihkan ke halaman *login*
	b	Konteks:	• Halaman ubah profil • Pengguna sudah *login* • Halaman ubah profil milik anggota lain
		Hasil:	• Dialihkan ke halaman hub • Menampilkan pesan kesalahan
	c	Konteks:	• Halaman ubah profil • Pengguna sudah *login* • Halaman ubah profil milik sendiri
		Hasil:	• Menampilkan halaman ubah profil dengan judul tertentu • Menampilkan form untuk mengubah profil
3	Skenario:		Mengubah profil `PUT` /users/{user_id}/profile
	a	Konteks:	• Pengubahan • Pengguna belum *login*
		Hasil:	• Dialihkan ke halaman *login* • Perubahan tidak tersimpan
	b	Konteks:	• Pengubahan • Pengguna sudah *login* • Profil milik anggota lain
		Hasil:	• Dialihkan ke halaman *hub* • Menampilkan pesan kesalahan
	c	Konteks:	• Pengubahan • Pengguna sudah *login* • Profil milik sendiri
		Hasil:	• Menyimpan perubahan • Dialihkan ke halaman *hub* • Menampilkan pesan

Sumber: Pengujian

Tabel 6.4 Pengujian *Class SessionsController*

1	Skenario:		Mengakses halaman *login* `GET` /sessions/new
	a	Konteks:	• Halaman *login*
		Hasil:	• Menampilkan halaman *login* dengan judul tertentu • Menampilkan *form* untuk *login*
2	Skenario		Melakukan *login* `POST` /sessions
	a	Konteks:	• *Login* • Mengisi *form* secara lengkap dan benar
		Hasil:	• Berhasil *login* • Dialihkan ke halaman *hub* • Menampilkan pesan
	a1	Konteks:	• *Login* • Mengisi *form* secara lengkap dan benar • Tanpa menggunakan "*remember me*"
		Hasil:	• *Cookies* tidak memuat *authentication token*
	a2	Konteks:	• *Login* • Mengisi *form* secara lengkap dan benar • Menggunakan "*remember me*"
		Hasil:	• *Cookies* memuat *authentication token*
	b	Konteks:	• *Login* • Mengisi *form* dengat data yang tidak valid
		Hasil:	• Gagal *login* • Tetap berada pada halaman *login* • Menampilkan pesan kesalahan
3	Skenario:		Autentikasi menggunakan *cookies*
	c	Konteks:	• *Cookies* • *Cookies* valid
		Hasil:	• *Login* secara otomatis
	d	Konteks:	• *Cookies* • *Cookies* sudah kadaluwarsa
		Hasil:	• Tidak login secara otomatis
	e	Konteks	• *Login* • *Authentication token* tidak valid
		Hasil:	• Tidak *login* secara otomatis

4	Skenario:		Melakukan *logout* `DELETE` /sessions
	a	Konteks:	• *Logout*
		Hasil:	• Berhasil logout • *Authentication token* pada *cookies* terhapus • Dialihkan ke halaman login

Sumber: Pengujian

Tabel 6.5 Pengujian *Class HubsController*

1	Skenario		Mengakses halaman *hub* `GET` /users/{user_id}/hub
	a	Konteks:	• Halaman *hub* • Pengguna sudah *login* • Halaman *hub* milik sendiri
		Hasil:	• Menampilkan halaman *hub* dengan judul tertentu
	b	Konteks:	• Halaman *hub* • Pengguna sudah *login* • Halaman *hub* anggota lain
		Hasil	• Dialihkan ke halaman *hub*-nya sendiri • Menampilkan pesan kesalahan
	c	Konteks:	• Halaman *hub* • Pengguna belum *login*
		Hasil	• Dialihkan ke halaman *login*

Sumber: Pengujian

Tabel 6.6 Pengujian *Class PostsController*

1	Skenario:		Mengakses halaman *blog* `GET` /users/{user_id}/blog/posts
	a	Konteks:	• Halaman *blog*
		Hasil	• Menampilkan daftar *posting* yang telah ada
	b	Konteks:	• Halaman *blog* • Pengguna sudah *login* • Halaman *blog* milik sendiri
			• Menampilkan *link* untuk membuat *posting* baru

2	Skenario		Mengakses salah satu *posting* `GET` /users/{user_id}/blog/posts/{post_id}
	a	Konteks:	• Halaman salah satu *posting*
		Hasil:	• Menampilkan halaman *posting* dengan judul tertentu • Menampilkan komentar yang berkaitan
	b	Konteks:	• Halaman salah satu *posting* • Pengguna belum *login*
		Hasil:	• Tidak menampilkan *link* untuk menambah komentar
	c	Konteks:	• Halaman salah satu *posting* • Pengguna sudah *login* • *Posting* milik anggota lain
		Hasil:	• Menampilkan *link* (Ajax) untuk menambah komentar • Tidak menampilkan *link* untuk menghapus *posting* • Tidak menampilkan *link* untuk mengubah *posting* • Tidak menampilkan *link* (Ajax) untuk menghapus komentar
	d	Konteks:	• Halaman salah satu *posting* • Pengguna sudah *login* • *Posting* milik sendiri
		Hasil:	• Menampilkan *link* (Ajax) untuk menambah komentar • Menampilkan *link* untuk menghapus *posting* • Menampilkan *link* untuk mengubah *posting* • Menampilkan *link* (Ajax) untuk menghapus komentar
3	Skenario:		Mengakses halaman membuat *posting* baru `GET` /users/{user_id}/blog/posts/new
	a	Konteks:	• Halaman membuat *posting* baru • Pengguna belum *login*
		Hasil:	• Dialihkan ke halaman *login*
	b	Konteks:	• Halaman membuat *posting* baru • Pengguna sudah *login* • Halaman membuat *posting* baru milik sendiri
		Hasil:	• Menampilkan halaman membuat *posting* baru dengan judul tertentu • Menampilkan *form* untuk membuat *posting* baru
	c	Konteks:	• Halaman membuat *posting* baru • Pengguna sudah *login* • Halaman membuat *posting* baru milik anggota lain
		Hasil:	• Dialihkan ke halaman *hub* • Menampilkan pesan kesalahan
4	Skenario:		Mengakses halaman ubah *posting* `GET` /users/{user_id}/blog/posts/{post_id}/edit

	a	Konteks:	• Halaman ubah *posting* • Pengguna belum *login*
		Hasil:	• Dialihkan ke halaman *hub*
	b	Konteks:	• Halaman ubah *posting* • Pengguna sudah *login* • Halaman ubah *posting* milik sendiri
		Hasil:	• Menampilkan halaman ubah *posting* dengan judul tertentu • Menampilkan form untuk mengubah *posting*
	c	Konteks:	• Halaman ubah *posting* • Pengguna sudah *login* • Halaman ubah *posting* milik anggota lain
		Hasil:	• Dialihkan ke halaman *hub* • Menampilkan pesan kesalahan
5	Skenario:		Membuat posting baru `POST` /users/{user_id}/blog/posts
	a	Konteks:	• *Posting* baru • Pengguna belum *login*
		Hasil:	• Tidak menambah *posting* baru • Dialihkan ke halaman *login*
	b	Konteks:	• *Posting* baru • Pengguna sudah *login* • *Posting* pada *blog* milik sendiri • Mengisi *form* secara lengkap dan benar
		Hasil:	• Berhasil dipublikasikan • Dialihkan ke halaman *blog* • Menampilkan pesan
	c	Konteks	• *Posting* baru • Pengguna sudah *login* • *Posting* pada *blog* milik sendiri • Tidak menyertakan judul *posting*
		Hasil	• Tidak menambah *posting* baru • Menampilkan pesan kesalahan
	d	Konteks	• *Posting* baru • Pengguna sudah *login* • *Posting* pada *blog* milik sendiri • Tidak menyertakan isi *posting*
		Hasil	• Tidak menambah *posting* baru • Menampilkan pesan kesalahan

	e	Konteks:	• *Posting* baru • Pengguna sudah *login* • *Posting* pada *blog* milik anggota lain
		Hasil:	• Tidak menambah *posting* baru • Dialihkan ke halaman *hub* • Menampilkan pesan kesalahan
6	Skenario:		Mengubah posting *blog* `PUT` /users/{user_id}/blog/posts/{post_id}
	a	Konteks:	• Ubah *posting* • Pengguna belum *login*
		Hasil:	• Tidak mengubah *posting* • Dialihkan ke halaman *login*
	b	Konteks:	• Ubah *posting* • Pengguna sudah *login* • *Posting* milik anggota lain
		Hasil:	• Tidak mengubah *posting* • Dialihkan ke halaman *hub* • Menampilkan pesan kesalahan
	c	Konteks:	• Ubah *posting* • Pengguna sudah *login* • *Posting* milik sendiri
		Hasil:	• Mengubah *posting* • Dialihkan ke halaman *posting* yang bersangkutan • Menampilkan pesan
7	Skenario:		Menghapus *posting blog* `DELETE` /users/{user_id}/blog/posts/{post_id}
	a	Konteks:	• Hapus *posting* • Pengguna belum *login*
		Hasil:	• *Posting* tidak terhapus • Dialihkan ke halaman *login*
	B	Konteks:	• Hapus *posting* • Pengguna sudah *login* • *Posting* milik anggota lain
		Hasil:	• *Posting* tidak terhapus • Dialihkan ke halaman *hub* • Menampilkan pesan kesalahan

	c	Konteks:	• Hapus *posting* • Pengguna sudah *login* • *Posting* milik sendiri
		Hasil:	• *Posting* terhapus • Dialihkan ke halaman blog • Menampilkan pesan

Sumber: Perancangan

Tabel 6.7 Pengujian *Class CommentsController*

1	Skenario:		Menghapus komentar *posting blog* `xhr`[9] `DELETE` /users/{user_id}/blog/posts/{post_id}/comments/{comment_id}
	a	Konteks:	• Hapus komentar • Pengguna belum *login*
		Hasil:	• Dialihkan ke halaman *login* • Komentar tidak terhapus
	b	Konteks:	• Hapus komentar • Pengguna sudah *login* • *Posting* milik anggota lain
		Hasil:	• Dialihkan ke halaman *hub* • Menampilkan pesan kesalahan • Komentar tidak terhapus
	c	Konteks:	• Hapus komentar • Pengguna sudah *login* • *Posting* milik sendiri
		Hasil:	• Komentar terhapus
2	Skenario:		Menambah komentar *posting blog* `xhr POST` /users/{user_id}/blog/posts/{post_id}/comments
	a	Konteks:	• Tambah komentar • Pengguna belum *login*
		Hasil:	• Dialihkan ke halaman *login* • Komentar tidak bertambah
	b	Konteks:	• Tambah komentar • Pengguna sudah *login*
		Hasil:	• Komentar bertambah

Sumber: Pengujian

9 xhr adalah singkatan dari XmlHttpRequest yang digunakan pada *framework* Ruby On Rails

Tabel 6.8 Pengujian *Class FriendshipsController*

1	Skenario:		Mengakses halaman pertemanan `GET` /users/{user_id}/friendships
	a	Konteks:	• Halaman pertemanan
		Hasil:	• Menampilkan halaman pertemanan dengan judul tertentu • Menampilkan daftar teman
	b	Konteks:	• Halaman pertemanan • Pengguna belum *login*
		Hasil:	• Tidak menampilkan *link* untuk menghapus teman • Tidak menampilkan daftar "Ajakan Teman" • Tidak menampilkan daftar "Mengajak Teman"
	c	Konteks:	• Halaman pertemanan • Pengguna sudah *login* • Pertemanan milik sendiri
		Hasil:	• Menampilkan daftar teman • Menampilkan *link* untuk menghapus teman • Menampilkan daftar "Ajakan Teman" • Menampilkan *link* untuk menolak ajakan teman • Menampilkan *link* untuk menerima ajakan teman • Menampilkan daftar "Mengajak Teman" • Menampilkan *link* untuk membatalkan mengajak teman
	d	Konteks:	• Halaman pertemanan • Pengguna sudah *login* • Pertemanan milik anggota lain
			• Tidak menampilkan *link* untuk menghapus teman • Tidak menampilkan daftar "Ajakan Teman" • Tidak menampilkan daftar "Mengajak Teman"
2	Skenario:		Menambah teman (mengajak & menerima pertemanan) `POST` /users/{user_id}/friendships
	a	Konteks:	• Tambah teman • Pengguna belum *login*
		Hasil:	• Dialihkan ke halaman *login* • Pertemanan tidak bertambah
	b	Konteks:	• Tambah teman • Pengguna sudah *login* • Mengajak pertemanan
		Hasil:	• Berhasil mengajak teman • Menampilkan pesan
	c	Konteks:	• Tambah teman • Pengguna sudah *login* • Menerima ajakan pertemanan

		Hasil:	• Berhasil menerima teman • Menampilkan pesan
3	Skenario:		Menghapus pertemanan `DELETE` /users/{user_id}/friendships/{friend_id}
	a	Konteks:	• Hapus teman • Pengguna belum *login*
		Hasil:	• Dialihkan ke halaman *login* • Tidak menghapus pertemanan
	b	Konteks:	• Hapus teman • Pengguna sudah *login* • Menghapus teman anggota lain
		Hasil:	• Dialihkan ke halaman *hub* • Menampilkan pesan kesalahan • Tidak menghapus pertemanan
	c	Konteks:	• Hapus teman • Pengguna sudah *login* • Menghapus teman sendiri • Membatalkan ajakan pertemanan
		Hasil	• Ajakan dibatalkan • Menampilkan pesan • Dialihkan ke halaman pertemanan
	d	Konteks:	• Hapus teman • Pengguna sudah *login* • Menghapus teman sendiri • Menolak ajakan pertemanan
		Hasil	• Ajakan ditolak • Menampilkan pesan • Dialihkan ke halaman pertemanan
	e	Konteks:	• Hapus teman • Pengguna sudah *login* • Menghapus teman sendiri • Menghapus pertemanan
		Hasil	• Pertemanan dihapus • Menampilkan pesan • Dialihkan ke halaman pertemanan

Sumber: Pengujian

Tabel 6.9 Pengujian *Class AvatarController*

1	Skenario:		Mengakses halaman untuk mengubah *avatar* `GET` /users/{user_id}/avatar/edit
	a	Konteks:	• Halaman ubah *avatar* • Pengguna belum *login*
		Hasil	• Dialihkan ke halaman *login*
	b	Konteks:	• Halaman ubah *avatar* • Pengguna sudah *login* • Halaman ubah *avatar* milik sendiri
		Hasil:	• Menampilkan halaman ubah *avatar* dengan judul tertentu • Menampilkan form ubah *avatar*
	c	Konteks:	• Halaman ubah *avatar* • Pengguna sudah *login* • Halaman ubah *avatar* milik sendiri • Sudah ada *avatar*
		Hasil:	• Menampilkan *link* untuk menghapus *avatar* • Menampilkan *avatar*
	d	Konteks:	• Halaman ubah *avatar* • Pengguna sudah *login* • Halaman ubah *avatar* milik anggota lain
		Hasil:	• Dialihkan ke halaman *hub* • Menampilkan pesan kesalahan
2	Skenario:		Mengubah *avatar* `PUT` /users/{user_id}/avatar
	a	Konteks:	• Ubah *avatar* • Pengguna belum *login*
		Hasil:	• Dialihkan ke halaman *login* • Tidak mengubah *avatar*
	b	Konteks:	• Ubah *avatar* • Pengguna sudah *login* • *Avatar* milik anggota lain
		Hasil:	• Dialihkan ke halaman *hub* • Tidak mengubah *avatar* • Menampilkan pesan kesalahan
	c	Konteks	• Ubah *avatar* • Pengguna sudah *login* • *Avatar* milik sendiri
		Hasil:	• Mengubah *avatar* • Dialihkan ke halaman *hub* • Menampilkan pesan

3	Skenario:		Menghapus *avatar* `DELETE` /users/{user_id}/avatar
	a	Konteks:	• Hapus *avatar* • Pengguna belum *login*
		Hasil	• Dialihkan ke halaman *login* • *Avatar* tidak terhapus
	b	Konteks:	• Hapus *avatar* • Pengguna sudah *login* • *Avatar* milik anggota lain
			• Dialihkan ke halaman *hub* • Menampilkan pesan kesalahan • *Avatar* tidak terhapus
	c	Konteks:	• Hapus *avatar* • Pengguna sudah *login* • *Avatar* milik sendiri
			• *Avatar* terhapus • Dialihkan ke halaman *hub* • Menampilkan pesan kesalahan

Sumber: Pengujian

Tabel 6.10 Pengujian *Class CommunityController*

1	Skenario:		Mengakses halaman penelusuran `GET` /community
	a	Konteks:	• Halaman penelusuran • Pengguna belum *login*
		Hasil:	• Dialihkan ke halaman *login*
	b	Konteks:	• Halaman penelusuran • Pengguna sudah *login*
			• Menampilkan halaman penelusuran dengan judul tertentu • Menampilkan link abjad anggota
2	Skenario:		Mengakses halaman pencarian `GET` /community/search
	a	Konteks:	• Halaman pencarian • Pengguna belum *login*
		Hasil:	• Dialihkan ke halaman *login*
	b	Konteks:	• Halaman pencarian • Pengguna sudah *login*
		Hasil:	• Menampilkan halaman pencarian dengan judul tertentu • Menampilkan form untuk pencarian

3	Skenario:		Melakukan penelusuran `GET` /community/{A...Z}
	a	Konteks:	• Penelusuran • Pengguna sudah *login*
		Hasil:	• Menampilkan hasil penelusuran
4	Skenario:		Melakukan pencarian `GET` /community/search?q={key_words}
	a	Konteks:	• Pencarian • Pengguna sudah *login*
		Hasil:	• Menampilkan hasil pencarian

Sumber: Pengujian

6.1.3 Pengujian Performa dan Skalabilitas Sistem

Salah satu tujuan utama REST adalah menghasilkan sistem dengan skalabilitas yang baik. Skalabilitas pada situs *social networking* dalam skripsi ini ditunjukkan oleh penggunaan *cluster* yang berisi kumpulan *web server* yang independen. Bila beban kerja meningkat, kemampuan sistem dapat ditingkatkan dengan menambahkan jumlah *web server* yang tergabung dalam *cluster*. Bila keadaan ini terjadi, sistem harus tetap bekerja secara normal, tidak boleh ada *request* yang gagal karena server tidak mampu melayani.

Pengujian terhadap performa dan skalabilitas menggunakan perangkat keras dan perangkat lunak yang sama dengan yang digunakan pada implementasi, seperti tercantum pada tabel 6.11.

Tabel 6.11 Spesifikasi Perangkat Keras Untuk Pengujian

Nama Komponen	Spesifikasi	Jumlah
Processor:	AMD Turion ML-34 @1.8 GHz	1
Memory (RAM):	512 MB DDR-333	2
Hardisk:	60 GB @ 5400 rpm	1

Sumber: Implementasi

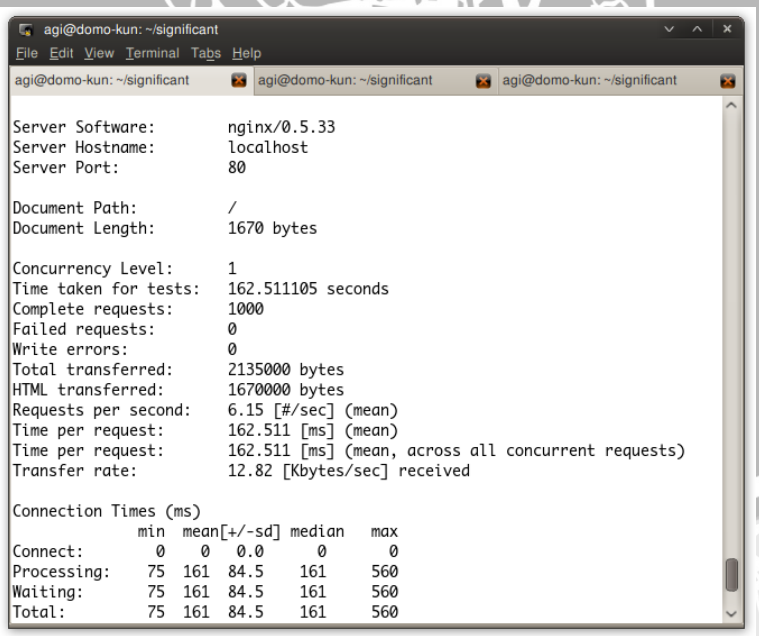
Perangkat lunak yang digunakan untuk melakukan pengujian ini adalah ApacheBench. Pengujian dilakukan sebanyak delapan macam seperti pada tabel 6.12.

Tabel 6.12 Daftar Pengujian Performa dan Skalabilitas

No	Jumlah Mongrel *Server* (x)	Jumlah *Concurrent User* (c)	Jumlah *Request* (r)
1	2	1	1000
2	2	2	1000
3	2	5	1000
4	2	50	1000
5	5	1	1000
6	5	2	1000
7	5	5	1000
8	5	50	1000

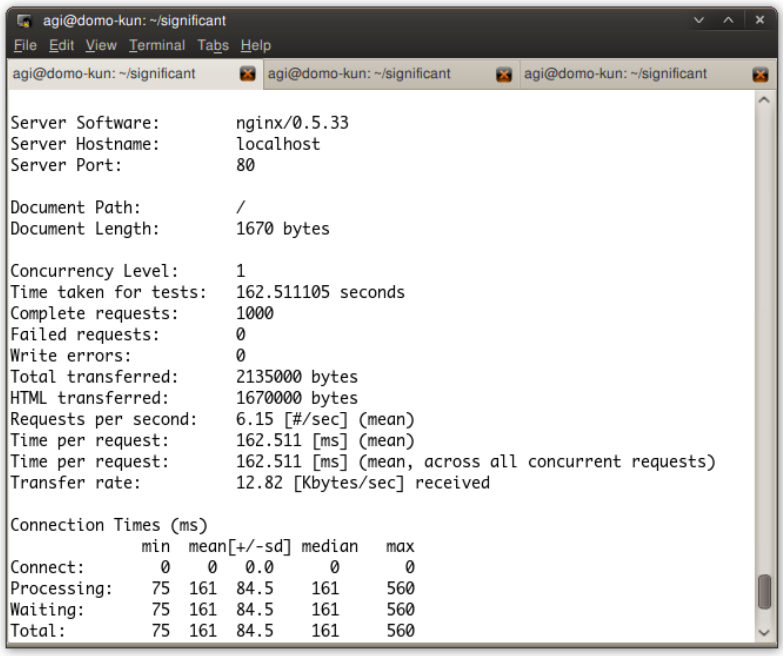
Sumber: Pengujian

Jumlah *concurrent user* (c) ditentukan mulai dari c < x, c = x, dan c > x, dengan x adalah jumlah *server* dalam *cluster*. Sedangkan jumlah *request* (r) yang dilakukan ditentukan oleh angka yang signifikan, yaitu r >> x. Masing - masing jenis pengujian dilakukan 3 kali dan diambil nilai tertinggi. Hasil pengujian performa ditampilkan oleh gambar 6.1, 6.2, 6.3, 6.4, 6.5, 6.6, 6.7, dan 6.8.



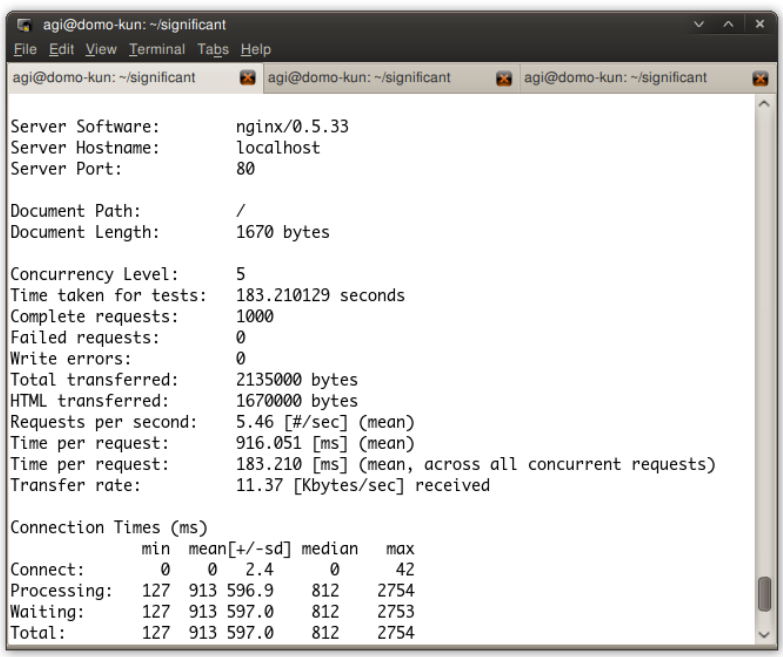
Gambar 6.1 Hasil Pengujian Performa Sistem Menggunakan 2 Mongrel *Servers* Untuk Menangani 1 *Concurrent User* dan 1000 *Requests*

Sumber: Pengujian



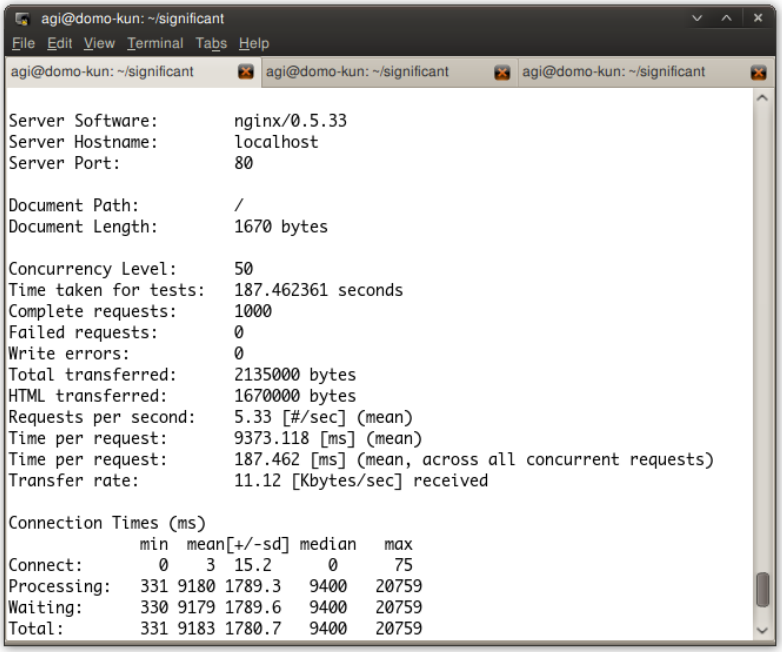
Gambar 6.2 Hasil Pengujian Performa Sistem Menggunakan 2 Mongrel *Servers* Untuk Menangani 2 *Concurrent Users* dan 1000 *Requests*

Sumber: Pengujian



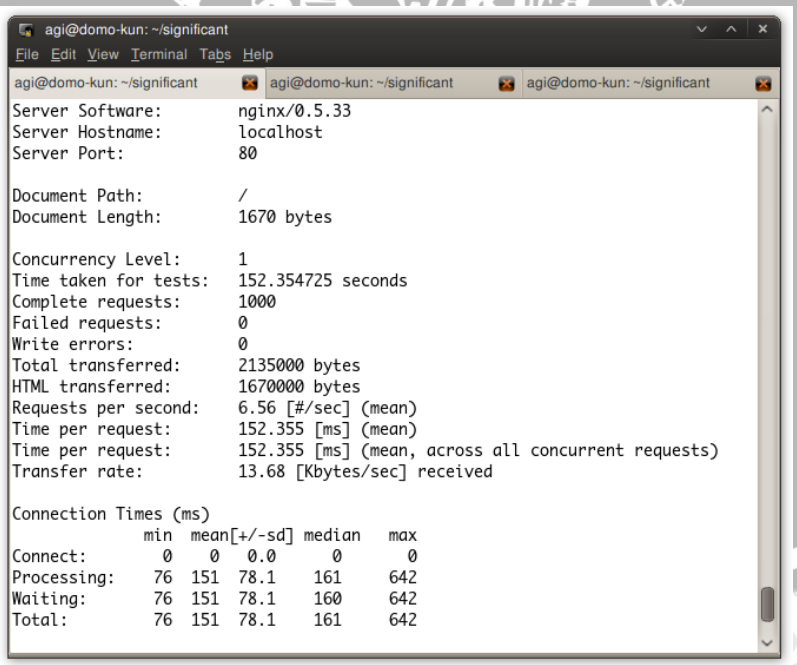
Gambar 6.3 Hasil Pengujian Performa Sistem Menggunakan 2 Mongrel *Servers* Untuk Menangani 5 *Concurrent Users* dan 1000 *Requests*

Sumber: Pengujian



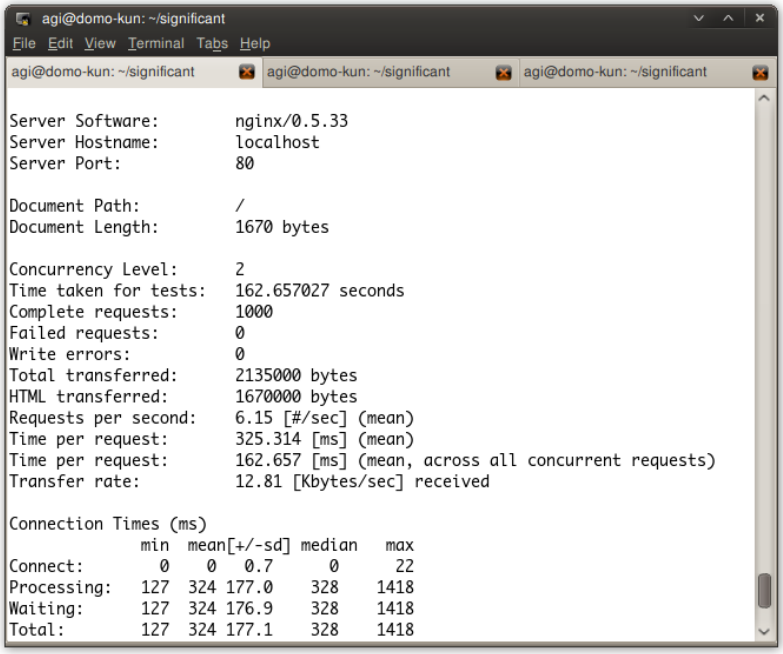
Gambar 6.4 Hasil Pengujian Performa Sistem Menggunakan 2 Mongrel *Servers* Untuk Menangani 50 *Concurrent Users* dan 1000 *Requests*

Sumber: Pengujian



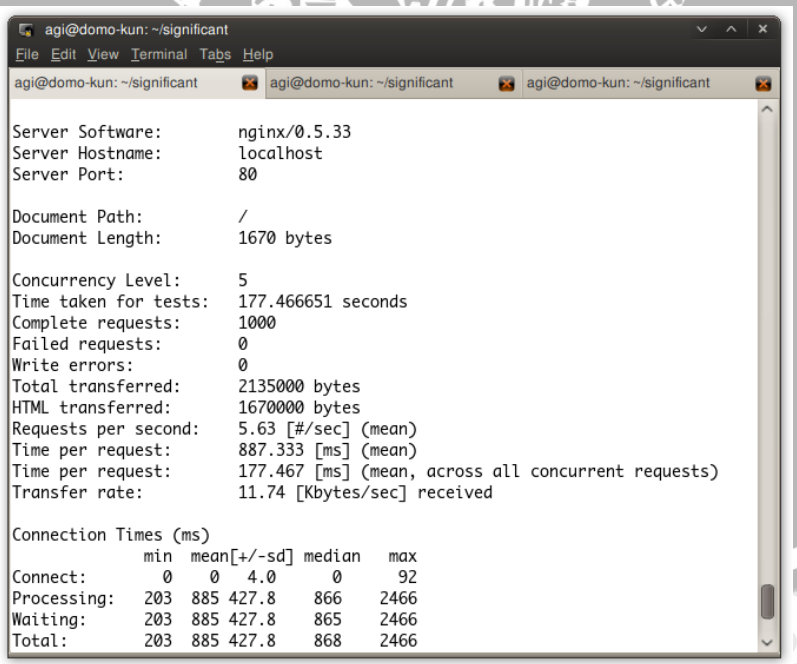
Gambar 6.5 Hasil Pengujian Performa Sistem Menggunakan 5 Mongrel *Servers* Untuk Menangani 1 *Concurrent User* dan 1000 *Requests*

Sumber: Pengujian



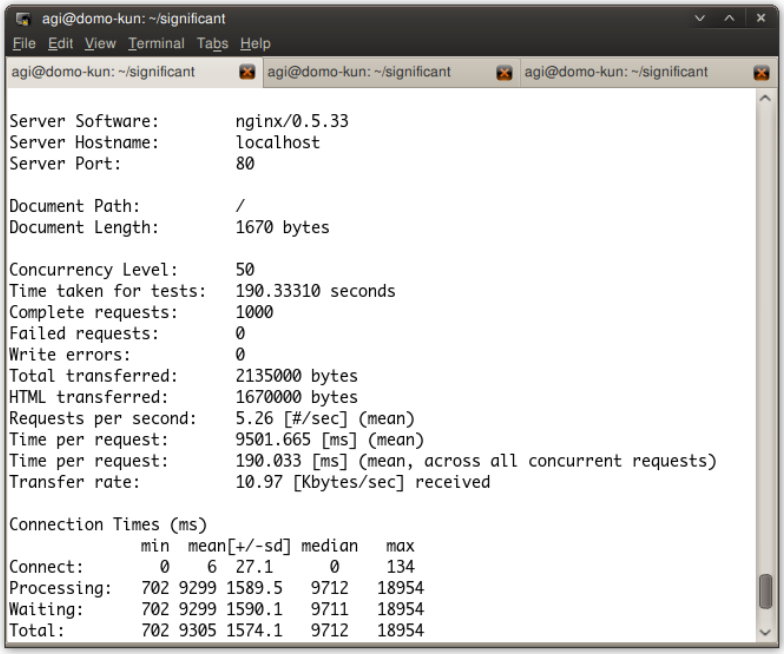
Gambar 6.6 Hasil Pengujian Performa Sistem Menggunakan 5 Mongrel *Servers* Untuk Menangani 2 *Concurrent Users* dan 1000 *Requests*

Sumber: Pengujian



Gambar 6.7 Hasil Pengujian Performa Sistem Menggunakan 5 Mongrel *Servers* Untuk Menangani 5 *Concurrent Users* dan 1000 *Requests*

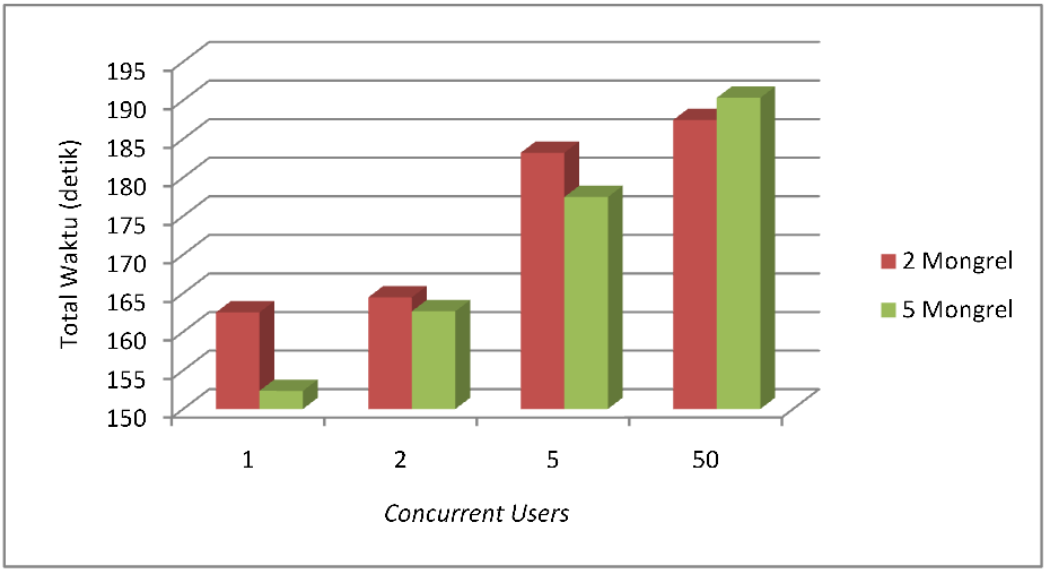
Sumber: Pengujian



Gambar 6.8 Hasil Pengujian Performa Sistem Menggunakan 5 Mongrel *Servers* Untuk Menangani 50 *Concurrent Users* dan 1000 *Requests*

Sumber: Pengujian

Hasil pengujian menggunakan ApacheBench menunjukkan bahwa *server* tidak pernah mengalami kegagalan dalam menangani seluruh *request* dari *client*, baik yang tergabung dalam *cluster* berisi 2 Mongrel *servers* maupun *cluster* yang bersisi 5 Mongrel *servers*. Dari hasil pengujian tersebut di atas, dapat dibuat grafik perbandingan performa antara *cluster* yang terdiri dari 2 Mongrel *servers* dan *cluster* yang terdiri dari 5 Mongrel *servers*, seperti ditunjukkan pada gambar 6.9.



Gambar 6.9 Perbandingan Performa *Cluster* Yang Berisi 2 Mongrel *Servers* Dan 5 Mongrel *Servers* Dalam Menangani 1000 *Requests*

Sumber: Pengujian

Dari grafik di atas dapat disimpulkan bahwa semakin banyak jumlah server yang terdapat dalam cluster mengakibatkan meningkatnya performa sistem, sehingga sistem memiliki skalabilitas yang baik. Namun karena pengujian dilakukan dengan menggunakan infrastruktur yang terbatas, maka perbedaan yang terjadi tidak begitu signifikan. Bahkan untuk menangani 50 *cuncurrent users*, *cluster* dengan 2 buah *server* memiliki performa yang lebih baik. Hal ini diakibatkan perangkat keras yang digunakan untuk pengujian memiliki jumlah prosesor dan *memory* (RAM) yang sangat terbatas. Semakin banyak jumlah *server* dalam *cluster* mengakibatkan performa sistem meningkat namun juga mengakibatkan bertambahnya kebutuhan kemampuan komputasi (prosesor) dan *memory.*

6.2 Analisis

Analisis dilakukan untuk membandingkan sistem yang telah dibuat dengan teori yang ada. Teori tersebut meliputi prinsip – prinsip REST, yaitu *addressability, statelessness, connectedness,* dan *uniform interface* . Selain itu, akan dibahas mengenai perbandingan antara REST *style* dengan RPC *style*.

6.2.1 Analisis Prinsip *Addressability*

Prinsip ini mensyaratkan bahwa setiap *resource* harus memiliki URI yang unik. Dalam situs *social networking* yang dikembangkan skripsi ini, semua *resource* yang diimplementasikan adalah bersifat *addressable*. Semua *resource* memiliki URI seperti pada tabel 4.3, 4.4, 4.5, 4,6, 4.7, 4.8, 4.9, 4.10, 4.11, dan 4.12.

6.2.2 Analisis Prinsip *Statelessness*

Prinsip ini mensyaratkan bahwa setiap *request* yang terjadi antara *client* dan *server* adalah bersifat independen satu sama lain. *Client* harus memelihara *application state* sendiri, sedangkan *server* tidak berurusan dengan pemeliharaan *application state* untuk masing – masing *client*.

Prinsip ini umumnya dilanggar ketika aplikasi membutuhkan sesuatu yang bersifat *stateful*, misalnya proses autentikasi. Dalam aplikasi situs *social networking* pada skripsi ini, *resource* yang mengakibatkan dilanggarnya prinsip *statelessness* ini adalah *resource sessions*. Ketika pengguna melakukan proses autentikasi dengan cara *login* (`POST /sessions`), maka *server* akan terus memelihara suatu *application state* yang berisi informasi tentang pengguna yang telah *login* tersebut, sehingga seolah – olah terjadi hubungan yang *stateful* antara *client* dengan *server*.

6.2.3 Analisis Prinsip *Connectedness*

Web merupakan sistem *hypermedia*, dimana terdapat keterkaitan antara satu *resource* dengan *resource* lainnya. Aplikasi web harus memiliki fasilitas untuk membantu atau menuntun pengguna menuju *resource* atau *application state* berikutnya. Bantuan tersebut berupa *link* maupun HTML *form*. Pada aplikasi situs *social networking* dalam skripsi ini, semua *resource* telah terhubung baik melalui *link* maupun HTML *form* tanpa satupun yang terlewatkan, sehingga situs *social networking* yang dibuat dalam skripsi ini telah memenuhi prinsip *connectedness*.

6.2.4 Analisis Prinsip *Uniform Interface*

Prinsip *uniform interface* mensyaratkan keseragaman pada *interface* yang

digunakan untuk mengakses atau memodifikasi suatu *resource*. *Interface* umum yang digunakan untuk aplikasi berbasis web adalah HTTP *methods*, yaitu GET, POST, PUT, dan DELETE. Dari hasil perancangan dan implementasi, dihasilkan 10 *resources* yang dapat diekspos menggunakan berbagai HTTP *method*. Masing – masing *method* tersebut harus dipergunakan sebagaimana spesifikasi protokol HTTP yang tercantum pada RFC 2616.

6.2.4.1 HTTP - GET

GET adalah salah satu HTTP *method* yang paling banyak digunakan dan memiliki sifat *safe* dan *idempotent*. Daftar *resource* dari situs *social networking* yang diakses menggunakan method GET terdapat pada tabel 6.13

Tabel 6.13 Daftar *Resource* Yang Diakses Menggunakan *Method* GET

Path	HTTP Method	Controller	Method
/	GET	SiteController	index
/users/new	GET	UsersController	new
/session/new	GET	SessionsController	new
/users/{user_id}/profile	GET	ProfilesController	show
/users/{user_id}/profile/edit	GET	ProfilesController	edit
/users/{user_id}/avatar/edit	GET	AvatarsController	edit
/users/{user_id}/friendships	GET	FriendshipsController	index
/users/{user_id}/blog/posts	GET	PostsController	index
/users/{user_id}/blog/posts/{post_id}	GET	PostsController	show
/users/{user_id}/blog/posts/new	GET	PostsController	new
/users/{user_id}/blog/posts/{post_id}/edit	GET	PostsController	edit
/users/{user_id}/hub	GET	HubsController	index
/community	GET	CommunityController	index
/community/index/{A...Z}	GET	CommunityController	index
/community/search?q={key_words}	GET	CommunityController	search

Sumber: Implementasi

Mengacu pada hasil perancangan dan implementasi dapat diketahui bahwa semua operasi GET pada *resource* – *resource* situs *social networking* yang

tercantum dalam tabel 6.11 adalah bersifat *safe method* dan *idempotent*. Dapat dipastikan bahwa setiap operasi GET tidak akan mengubah *resource state* yang terdapat pada *server*. Hal ini memiiki arti bahwa situs *social networking* yang dibuat dalam skripsi ini telah memenuhi spesifikasi protokol HTTP seperti yang tercantum pada RFC 2126 untuk penggunaan *method* GET.

6.2.4.2 HTTP - POST

POST merupakan *method* yang tidak memiliki batasan tertentu, baik syarat *idempotent* maupun syarat sebagai *safe method*. Dalam aplikasi yang dibuat menggunakan prinsip REST, POST digunakan untuk menciptakan (*create*) suatu *resource* baru, atau bila *resource* tersebut telah ada sebelumnya, maka dilakukan pembaharuan (*update*) terhadap *resource* tersebut. Daftar *resource* dari situs *social networking* yang diekspos menggunakan *method* POST terdapat pada tabel 6.14.

Tabel 6.14 Daftar *Resource* Yang Diekspos Menggunakan *Method* POST

Path	HTTP Method	Controller	Method
/users	POST	UsersController	create
/session	POST	SessionsController	create
/users/{friend_id}/friendships	POST	FriendshipsController	create
/users/{user_id}/blog/posts	POST	PostsController	create
/users/{user_id}/blog/posts/{post_id}/comments	POST	CommentsController	create

Sumber: Implementasi

Mengacu pada hasil perancangan dan implementasi dapat diketahui bahwa penggunaan method POST adalah untuk membuat *resource* baru (atau memperbaharui bila *resource* tersebut telah ada sebelumnya), sehingga dapat dipastikan bahwa tidak satupun implementasi dari method POST yang merupakan *safe method* karena setiap operasi POST hampir dapat dipastikan akan menyebabkan perubahan pada *resource state*. Selain itu penggunaan *method* POST pada situs *social networking* dalam skripsi ini umumnya tidak bersifat

idempotent, kecuali pada kasus tertentu, yaitu pada kasus *resource friendships* (POST /users/{user_id}/friendships). Pada kasus ini, apabila pengguna mengajak anggota lain untuk berteman maupun menerima ajakan pertemanan (POST /users/{user_id}/friendships) dan pengguna tersebut melakukan operasi POST lebih dari satu kali, maka operasi yang kedua dan seterusnya akan memberi efek yang sama terhadap *resource state* seperti operasi POST yang pertama sebagaimana sifat *idempotent*. Dengan demikian penggunaan method POST pada situs social networking yang dibuat dalam skripsi ini telah sesuai dengan spesifikasi protokol HTTP yang tercantum pada RFC 2126.

6.2.4.3 HTTP – PUT

Pada perancangan aplikasi web menggunakan prinsip REST, umumnya *method* PUT digunakan untuk melakukan pembaharuan (*update*) terhadap suatu *resource* tertentu. Hal ini didukung oleh sifat dari *method* PUT, yaitu *unsafe* dan *idempotent*. *Unsafe* berarti bahwa *method* PUT diperkenankan untuk mengubah *resource state* yang terletak pada *server*, dan *idempotent* berarti bahwa operasi PUT yang kedua dan seterusnya memiliki efek yang sama dengan operasi PUT yang pertama. Daftar *resource* dari situs *social networking* yang diekspos menggunakan *method* PUT terdapat pada tabel 6.15.

Tabel 6.15 Daftar *Resource* Yang Diekspos Menggunakan *Method* PUT

Path	HTTP Method	Controller	Method
/users/{user_id}/profile	PUT	ProfilesController	update
/users/{user_id}/avatar	PUT	AvatarsController	update
/users/{user_id}/blog/posts/{post_id}	PUT	PostsController	update

Sumber: Implementasi

Mengacu pada hasil perancangan dan implementasi dapat diketahui bahwa seluruh penerapan *method* PUT pada situs *social networking* yang dibuat dalam skripsi ini digunakan untuk melakukan pembaharuan (*update*) terhadap suatu *resource* tertentu yang spesifik. Selain itu, implementasi dari *method* PUT juga telah sesuai dengan spesifikasi HTTP yang tercantum pada RFC 2126, yaitu

merupakan *unsafe method* dan bersifat *idempotent*.

6.2.4.3 HTTP – DELETE

Pada perancangan aplikasi web menggunakan prinsip REST, umumnya *method* DELETE digunakan untuk menghapus suatu *resource* tertentu. *Method* DELETE memiliki sifat *unsafe* dan *idempotent*. Daftar *resource* dari situs *social networking* yang diekspos menggunakan *method* DELETE terdapat pada tabel 6.16.

Tabel 6.16 Daftar *Resource* Yang Diekspos Menggunakan *Method* DELETE

Path	HTTP Method	Controller	Method
/users/{user_id}	DELETE	UsersController	destroy
/session	DELETE	SessionsController	destroy
/users/{user_id}/avatar	DELETE	AvatarsController	destroy
/users/{user_id}/friendships/{friend_id}	DELETE	FriendshipsController	destroy
/users/{user_id}/blog/posts/{post_id}	DELETE	PostsController	delete
/users/{user_id}/blog/posts/{post_id}/comments/{comment_id}	DELETE	CommentsController	destroy

Sumber: Implementasi

Mengacu pada hasil perancangan dan implementasi dapat diketahui bahwa seluruh penerapan *method* DELETE pada situs *social networking* yang dibuat dalam skripsi ini digunakan untuk melakukan penghapusan terhadap suatu *resource* tertentu yang spesifik. Selain itu implementasi dari *method* DELETE juga telah sesuai dengan spesifikasi HTTP yang tercantum pada RFC 2126, yaitu merupakan *unsafe method* dan bersifat *idempotent*.

6.2.5 Analisis Perbandingan Antara REST *Style* Dengan RPC *Style*

Ada beberapa perbedaan mendasar antara REST dengan RPC seperti yang tercantum dalam tabel 6.17.

Tabel 6.17 Perbedaan Antara REST *Style* Dengan RPC *Style*

REST *Style*	RPC *Style*
Berorientasi pada *resource*	Berorientasi pada pemanggilan fungsi atau prosedur
Menerapkan prinsip *uniform interface* untuk mengatur fungsi – fungsi yang diimplementasikan	Tidak memiliki aturan atau *interface* khusus mengenai fungsi – fungsi atau prosedur yang diimplementasikan

Sumber: Hasil Analisis

Perbedaan tersebut menyebabkan REST *style* memiliki beberapa keunggulan dibanding RPC *style*, yaitu:

- REST menghasilkan sistem yang lebih sederhana dan konsisten
- REST menghasilkan sistem yang lebih mudah diprediksi perilakunya
- REST menghasilkan sistem yang memiliki skalabilitas dan *interoperability* baik

Sebagai contoh kasus, berikut ini disertakan perbandingan implementasi dari kebutuhan fungsional F03a (melihat blog anggota), F07 (menulis *posting blog*), F08 (mengubah *posting blog*), F09 (menghapus *posting blog*). Tabel 6.18 merupakan implementasi menggunakan REST style dan tabel 6.19 merupakan implementasi menggunakan RPC style.

Tabel 6.18 Contoh Implementasi Menggunakan REST *Style*

Kebutuhan Fungsional	URI	HTTP *Method*
F03a	/posts	GET
F07	/posts	POST
F08	/posts/{post_id}	PUT
F09	/posts/{post_id}	DELETE

Sumber: Hasil Analisis

Tabel 6.19 Contoh Implementasi Menggunakan RPC *Style*

Kebutuhan Fungsional	URI	HTTP *Method*
F03a	/posts?show=all	GET
F07	/posts?action=create	POST
F08	/posts?action=update&post_id={post_id}	POST
F09	/posts?action=delete&post_id={post_id}	GET

Sumber: Hasil Analisis

Pada implementasi yang menggunakan RPC *style* tidak terdapat batasan – batasan khusus tentang penggunaan HTTP *method*, sehingga berpotensi menimbulkan penyimpangan dalam penggunaan HTTP *method*. Pada tabel 6.19 terdapat penyimpangan terhadap penggunaan *method* GET yang seharusnya merupakan *safe method* menjadi *unsafe method* karena operasi penghapusan (*delete*) mengakibatkan berubahnya *resource state* pada *server*. Hal ini mengakibatkan sistem yang dibuat menggunakan RPC *style* menjadi lebih sulit diprediksi perilakunya. Perilaku sistem yang sulit diprediksi juga dapat berpengaruh buruk terhadap skalabilitas.

Perbedaan lain yang cukup mencolok yaitu adanya *query string* pada URI yang umumnya berisi nama fungsi atau prosedur yang dipanggil pada RPC *style*, sedangkan hal ini tidak dijumpai pada REST *style* sehingga URI yang dihasilkan REST *style* terlihat lebih bersih dan konsisten. Demikian pula tidak ada aturan penamaan fungsi maupun prosedur dalam RPC *style*, sehingga kemungkinan jenis nama yang digunakan sangat banyak sekali dan berpotensi mengganggu *interoperability* sistem. Hal ini berbeda dengan REST *style* yang berorientasi pada *resource*, REST telah memiliki nama *method* yang pasti, terbatas, dan telah disepakati bersama sesuai prinsip *uniform interface*. Penamaan dan perilaku *method* yang konsisten pada REST *style* menyebabkan sistem ini memiliki *interoperability* yang baik.

BAB VII
PENUTUP

7.1 Kesimpulan

Bedasarkan hasil perancangan, implementasi, dan pengujian yang dilakukan, maka diambil kesimpulan sebagai berikut :

1. Situs *social networking* berhasil dirancang secara *resource-oriented*, menghasilkan 10 *resources*. Masing – masing *resource* dapat diekspos menggunakan berbagai HTTP *method* yang berbeda sesuai kebutuhan.
2. Situs *social networking* berhasil diimplementasikan dengan mematuhi prinsip REST, yaitu *addressability*, *statelessness*, *connectedness*, dan *uniform interface*. Situs *social networking* dapat ditingkatkan kemampuannya (*scalability*) dengan cara menambah jumlah *web server* menggunakan *cluster* yang berisi kumpulan *web server* yang independen.
3. Implementasi situs *social networking* menggunakan prinsip REST menghasilkan sistem yang lebih sederhana dibanding RPC. Jumlah varian *methods* dalam *interface* yang dihasilkan REST lebih sedikit dibanding RPC. *Methods* yang terdapat di dalam *interface* tersebut adalah GET, POST, PUT, dan DELETE.

7.2 Saran

Saran yang dapat diberikan untuk pengembangan situs *social networking* menggunakan prinsip *Representational State Transfer*.

1. Fitur – fitur pendukung *social networking* ditambahkan dan dibuat lebih bervariasi.
2. Perangkat lunak untuk *client* dapat dikembangkan supaya mendukung penuh semua HTTP *methods*, atau setidaknya telah mendukung *method* GET, POST, PUT, dan DELETE sehingga tidak perlu melakukan simulasi terhadap method yang belum didukung.

3. Perangkat lunak dapat dikembangkan supaya memiliki kemampuan menghasilkan bentuk representasi *resource* selain HTML, misalnya dalam bentuk XML, JSON, PDF, dan sebagainya.

DAFTAR PUSTAKA

[BEN-08] Bendon, Edward. 2008. ***The Art Of Rails***. Wiley Publishing, Inc. Indianapolis.

[ERI-07] Eric van der Vlist, Alessandro Vernet, Erik Bruchez, Joe Fawcett, dan Danny Ayers. 2007. ***Professional Web 2.0 Programming***. Wiley Publishing, Inc. Indianapolis.

[FRE-08] Steve, Freeman. ***Growing Object-Oriented Software, Guided by Tests***. Akses dari http://www.mockobjects.com/book/tdd-introduction.html. Tanggal akses: 25 November 2008.

[HAL-08] Hall, Lowe N. 2008. ***What Is Hypermedia?***. Akses dari http://users.ecs.soton.ac.uk/lac/LoweNHall/extracts/Hypermedia.html. Tanggal akses: 19 Oktober 2008.

[HAR-07] Hartl, Michael dan Prochazka, Aurelius. 2008. ***RailsSpace***. Addison Wesley. San Francisco.

[KEI-07] Keith, Jeremy.. 2007. ***Bulletproof Ajax***. New Riders. Berkeley.

[MIK-07] Mika, Peter. 2007. ***Social Networks And The Semantic Web***. Springer. New York.

[ODO-05] O'Docherty, Mike. 2005. ***Object-Oriented Analysis & Design***. John Wiley & Sons, Ltd. New York.

[ORE-05] O'Reilly, Tim. 2005. ***What is Web 2.0***. Akses dari http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html. Tanggal akses: 21 Agustus 2008.

[POL-07] Pollack, Greg. 2007. ***Future Of Web Services***. Akses dari http://www.railsenvy.com/2007/12/17/the-future-of-web-services. Tanggal akses: 21 Agustus 2008.

[POW-06] Powell, Gavin. 2006. ***Beginning Database Design***. Wiley Publishing, Inc. Indianapolis.

[ROS-07] Rosenber, Doug dan Stephens, Matt. 2007. ***Use Case Driven Object Modelling With UML***. Apress. Berkeley.

[RIC-07] Richardson, Leonard dan Ruby, Sam. 2007. ***RESTful Web Services.*** O'Reilly. Sebastopol.

[SAL-08] Saleh, Tammer. 2008. ***BDD with Shoulda (MountainWest RubyConf 2008).*** Akses dari http://tammersaleh.com/system/assets/bdd_with_shoulda.pdf. Tanggal akses 4 Januari 2009.

[SOU-07] Souders, Steve. 2007. ***High Performance Websites***. O'Reilly. Sebastopol.

[SRI-05] Sri. 2005. ***Dynamic Programming Language***. Akses dari http://www.total.or.id/info.php?kk=Dynamic%20programming%20language. Tanggal akses: 22 Oktober 2008.

[THO-07] Thomas, Dave dan Hanson, David Heinemeier. 2007. ***Agile Web Development With Rails - Second Edition***. The Pragmatic Bookshelf. Texas.

[TIL-08] Tilkov, Stefan. 2008. ***A Brief Introduction to REST***. Akses dari http://www.infoq.com/articles/rest-introduction. Tanggal akses: 19 Oktober 2008.

LAMPIRAN

RFC 2616 (Halaman 51 – 57)

However, it is possible that a sequence of several requests is non-idempotent, even if all of the methods executed in that sequence are idempotent. (A sequence is idempotent if a single execution of the entire sequence always yields a result that is not changed by a reexecution of all, or part, of that sequence.) For example, a sequence is non-idempotent if its result depends on a value that is later modified in the same sequence.

A sequence that never has side effects is idempotent, by definition (provided that no concurrent operations are being executed on the same set of resources).

9.2 OPTIONS

The OPTIONS method represents a request for information about the communication options available on the request/response chain identified by the Request-URI. This method allows the client to determine the options and/or requirements associated with a resource, or the capabilities of a server, without implying a resource action or initiating a resource retrieval.

Responses to this method are not cacheable.

If the OPTIONS request includes an entity-body (as indicated by the presence of Content-Length or Transfer-Encoding), then the media type MUST be indicated by a Content-Type field. Although this specification does not define any use for such a body, future extensions to HTTP might use the OPTIONS body to make more detailed queries on the server. A server that does not support such an extension MAY discard the request body.

If the Request-URI is an asterisk ("*"), the OPTIONS request is intended to apply to the server in general rather than to a specific resource. Since a server's communication options typically depend on the resource, the "*" request is only useful as a "ping" or "no-op" type of method; it does nothing beyond allowing the client to test the capabilities of the server. For example, this can be used to test a proxy for HTTP/1.1 compliance (or lack thereof).

If the Request-URI is not an asterisk, the OPTIONS request applies only to the options that are available when communicating with that resource.

A 200 response SHOULD include any header fields that indicate optional features implemented by the server and applicable to that resource (e.g., Allow), possibly including extensions not defined by this specification. The response body, if any, SHOULD also include information about the communication options. The format for such a

body is not defined by this specification, but might be defined by future extensions to HTTP. Content negotiation MAY be used to select the appropriate response format. If no response body is included, the response MUST include a Content-Length field with a field-value of "0".

The Max-Forwards request-header field MAY be used to target a specific proxy in the request chain. When a proxy receives an OPTIONS request on an absoluteURI for which request forwarding is permitted, the proxy MUST check for a Max-Forwards field. If the Max-Forwards field-value is zero ("0"), the proxy MUST NOT forward the message; instead, the proxy SHOULD respond with its own communication options. If the Max-Forwards field-value is an integer greater than zero, the proxy MUST decrement the field-value when it forwards the request. If no Max-Forwards field is present in the request, then the forwarded request MUST NOT include a Max-Forwards field.

9.3 GET

The GET method means retrieve whatever information (in the form of an entity) is identified by the Request-URI. If the Request-URI refers to a data-producing process, it is the produced data which shall be returned as the entity in the response and not the source text of the process, unless that text happens to be the output of the process.

The semantics of the GET method change to a "conditional GET" if the request message includes an If-Modified-Since, If-Unmodified-Since, If-Match, If-None-Match, or If-Range header field. A conditional GET method requests that the entity be transferred only under the circumstances described by the conditional header field(s). The conditional GET method is intended to reduce unnecessary network usage by allowing cached entities to be refreshed without requiring multiple requests or transferring data already held by the client.

The semantics of the GET method change to a "partial GET" if the request message includes a Range header field. A partial GET requests that only part of the entity be transferred, as described in section 14.35. The partial GET method is intended to reduce unnecessary network usage by allowing partially-retrieved entities to be completed without transferring data already held by the client.

The response to a GET request is cacheable if and only if it meets the requirements for HTTP caching described in section 13.

See section 15.1.3 for security considerations when used for forms.

repository.ub.ac.id

9.4 HEAD

The HEAD method is identical to GET except that the server MUST NOT return a message-body in the response. The metainformation contained in the HTTP headers in response to a HEAD request SHOULD be identical to the information sent in response to a GET request. This method can be used for obtaining metainformation about the entity implied by the request without transferring the entity-body itself. This method is often used for testing hypertext links for validity, accessibility, and recent modification.

The response to a HEAD request MAY be cacheable in the sense that the information contained in the response MAY be used to update a previously cached entity from that resource. If the new field values indicate that the cached entity differs from the current entity (as would be indicated by a change in Content-Length, Content-MD5, ETag or Last-Modified), then the cache MUST treat the cache entry as stale.

9.5 POST

The POST method is used to request that the origin server accept the entity enclosed in the request as a new subordinate of the resource identified by the Request-URI in the Request-Line. POST is designed to allow a uniform method to cover the following functions:

- Annotation of existing resources;
- Posting a message to a bulletin board, newsgroup, mailing list, or similar group of articles;
- Providing a block of data, such as the result of submitting a form, to a data-handling process;
- Extending a database through an append operation.

The actual function performed by the POST method is determined by the server and is usually dependent on the Request-URI. The posted entity is subordinate to that URI in the same way that a file is subordinate to a directory containing it, a news article is subordinate to a newsgroup to which it is posted, or a record is subordinate to a database.

The action performed by the POST method might not result in a resource that can be identified by a URI. In this case, either 200 (OK) or 204 (No Content) is the appropriate response status, depending on whether or not the response includes an entity that describes the result.

UNIVERSITAS BRAWIJAYA

If a resource has been created on the origin server, the response SHOULD be 201 (Created) and contain an entity which describes the status of the request and refers to the new resource, and a Location header (see section 14.30).

Responses to this method are not cacheable, unless the response includes appropriate Cache-Control or Expires header fields. However, the 303 (See Other) response can be used to direct the user agent to retrieve a cacheable resource.

POST requests MUST obey the message transmission requirements set out in section 8.2.

See section 15.1.3 for security considerations.

9.6 PUT

The PUT method requests that the enclosed entity be stored under the supplied Request-URI. If the Request-URI refers to an already existing resource, the enclosed entity SHOULD be considered as a modified version of the one residing on the origin server. If the Request-URI does not point to an existing resource, and that URI is capable of being defined as a new resource by the requesting user agent, the origin server can create the resource with that URI. If a new resource is created, the origin server MUST inform the user agent via the 201 (Created) response. If an existing resource is modified, either the 200 (OK) or 204 (No Content) response codes SHOULD be sent to indicate successful completion of the request. If the resource could not be created or modified with the Request-URI, an appropriate error response SHOULD be given that reflects the nature of the problem. The recipient of the entity MUST NOT ignore any Content-* (e.g. Content-Range) headers that it does not understand or implement and MUST return a 501 (Not Implemented) response in such cases.

If the request passes through a cache and the Request-URI identifies one or more currently cached entities, those entries SHOULD be treated as stale. Responses to this method are not cacheable.

The fundamental difference between the POST and PUT requests is reflected in the different meaning of the Request-URI. The URI in a POST request identifies the resource that will handle the enclosed entity. That resource might be a data-accepting process, a gateway to some other protocol, or a separate entity that accepts annotations. In contrast, the URI in a PUT request identifies the entity enclosed with the request -- the user agent knows what URI is intended and the server MUST NOT attempt to apply the request to some other resource. If the server desires that the request be applied to a different URI,

it MUST send a 301 (Moved Permanently) response; the user agent MAY then make its own decision regarding whether or not to redirect the request.

A single resource MAY be identified by many different URIs. For example, an article might have a URI for identifying "the current version" which is separate from the URI identifying each particular version. In this case, a PUT request on a general URI might result in several other URIs being defined by the origin server.

HTTP/1.1 does not define how a PUT method affects the state of an origin server.

PUT requests MUST obey the message transmission requirements set out in section 8.2.

Unless otherwise specified for a particular entity-header, the entity-headers in the PUT request SHOULD be applied to the resource created or modified by the PUT.

9.7 DELETE

The DELETE method requests that the origin server delete the resource identified by the Request-URI. This method MAY be overridden by human intervention (or other means) on the origin server. The client cannot be guaranteed that the operation has been carried out, even if the status code returned from the origin server indicates that the action has been completed successfully. However, the server SHOULD NOT indicate success unless, at the time the response is given, it intends to delete the resource or move it to an inaccessible location.

A successful response SHOULD be 200 (OK) if the response includes an entity describing the status, 202 (Accepted) if the action has not yet been enacted, or 204 (No Content) if the action has been enacted but the response does not include an entity.

If the request passes through a cache and the Request-URI identifies one or more currently cached entities, those entries SHOULD be treated as stale. Responses to this method are not cacheable.

9.8 TRACE

The TRACE method is used to invoke a remote, application-layer loop-back of the request message. The final recipient of the request SHOULD reflect the message received back to the client as the entity-body of a 200 (OK) response. The final recipient is either the

origin server or the first proxy or gateway to receive a Max-Forwards value of zero (0) in the request (see section 14.31). A TRACE request MUST NOT include an entity.

TRACE allows the client to see what is being received at the other end of the request chain and use that data for testing or diagnostic information. The value of the Via header field (section 14.45) is of particular interest, since it acts as a trace of the request chain. Use of the Max-Forwards header field allows the client to limit the length of the request chain, which is useful for testing a chain of proxies forwarding messages in an infinite loop.

If the request is valid, the response SHOULD contain the entire request message in the entity-body, with a Content-Type of "message/http". Responses to this method MUST NOT be cached.

9.9 CONNECT

This specification reserves the method name CONNECT for use with a proxy that can dynamically switch to being a tunnel (e.g. SSL tunneling [44]).