BAB 2 LANDASAN KEPUSTAKAAN

Pada bab kajian pustaka ini membahas mengenai kajian pustaka serta dasar teori pada penelitian. Kajian pustaka menjelaskan kenapa penulis menggunakan RESTful *web service* pada sistem yang akan dikembangkan pada penelitian, serta menjelaskan dasar teori pendukung untuk mengimplementasikan penelitian.

2.1 *Representational State Transfer* (REST)

REST merupakan sebuah standar arsitektur komunikasi yang berbasis web dan sering digunakan dalam pengembangan *service* (layanan) berbasis web yang dikenalkan oleh Roy Fielding pada tahun 2000. Arsitektur REST ini menggunakan *client-server* dimana *client* akan mengirimkan *request* kepada *server* lalu *server* akan meproses *request* tersebut dan memberikan *response* (Feridi,2016)

Dalam arsitektur REST, REST *server* menyediakan sumber daya (*resources*) dapat berupa data atau apapun. dan REST *client* mengakses *resource* tersebut. Setiap *resource* diidentifikasi oleh *Universal Resource Identifier*s (URI). *Resource* tersebut direpresentasikan dalam bentuk format teks, Javascript *Object Notation* (JSON) atau *Extensible Markup Language* (XML) (Richardson, Ruby,2016). Tetapi biasanya format *response* menggunakan JSON atau XML. Tetapi pemakaian JSON lebih populer, karena performa JSON lebih cepat daripada XML. REST tidak memerlukan parsing XML serta tidak memerlukan *Header* untuk pertukaran datanya. Hal tersebut pada akhirnya akan mereduksi penggunaan *bandwidth*. Perbandingan ukuran pesan dan waktu antara SOAP dan REST pada *paper* yang berjudul *Performance Evaluation of RESTful Web Services for Mobile Devices* dalam *International Arab Journal of e-Technology* dapat dilihat pada gambar 2.1. berikut :

Number of array elements	Message Size (byte)				Time (Milliseconds)			
	SOAP/HTTP		REST (HTTP)		SOAP/HTTP		REST (HTTP)	
	String Concatenation	Float Numbers Addition	String Concatenation	Float Numbers Addition	String Concatenation	Float Numbers Addition	String Concatenation	Float Numbers Addition
2	351	357	39	32	781	781	359	359
3	371	383	48	36	828	781	344	407
4	395	409	63	35	828	922	359	375
5	418	435	76	39	969	1016	360	359
6	443	461	93	43	875	953	359	359
7	465	487	104	47	875	875	469	360
8	493	513	127	51	984	875	437	344

Gambar 2.1 Perbandingan ukuran pesan dan waktu antara SOAP dan REST untuk penggabungan string dan penjumlahan float.

Sumber: (Hamad, Saad, Abed, 2009)

Meskipun *resource* pada arsitektur REST dapat berupa apapun, tetapi *client* tidak dapat melakukan operasi secara sembarangan terhadap *resource*. Pada arsitektur REST, *client* dan *server* hanya diperbolehkan berinteraksi dengan saling mengirimkan pesan sesuai protokol yang telah didefinisikan. Arsitektur REST biasanya menggunakan protokol *Hypertext Transfer Protocol* (HTTP) untuk berkirim pesan antara *client* dan *server,* pada saat pertukaran pesan ini *client* dan

server akan mengirimkan pesan HTTP yang berbeda (Richardson, Amundsen,2013). Terdapat 8 macam jenis pesan yang berbeda tetapi biasanya hanya 4 macam yang pasti digunakan yang dapat dilihat pada tabel 2.1. Dasar teori mengenai *Representational State Transfer* (REST) digunakan sebagai arsitektur dari *web service* untuk acuan implementasi sistem informasi Tiga Pilar yang dibuat.

Tabel 2.1 HTTP *method* pada REST yang sering digunakan

Jenis	**Deskripsi**
GET	Untuk mendapatkan representasi dari *resource*.
POST	Membuat *resource* baru berdasarkan representasi yang diberikan.
PUT	Memperbarui/mengganti *resource* yang sudah ada.
DELETE	Menghapus *resource*.

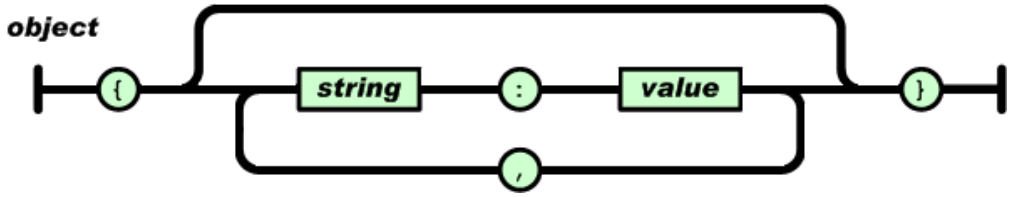
Sumber: (Richardson, Amundsen, 2013)

2.2 *Javascript Object Notation* (JSON)

JSON adalah format pertukaran data yang ringan (*lightweight data-interchange format*), mudah untuk dan ditulis oleh manusia, serta mudah untuk diterjemahkan dan dibuat oleh komputer. JSON tidak tergantung pada format penulisan bahasa pemrograman apapun, oleh karena hal tersebut, menjadikan JSON cocok sebagai format pertukaran data (json.org). JSON terdiri dari dua struktur :

- Kumpulan dari nama dan nilai yang berpasangan, pada beberapa bahasa hal ini dianggap sebagai objek, *record*, *struct*, *dictionary*, *hash table*, *keyed list*, atau *associative array*.
- Direpresentasikan dengan nilai yang berurutan. Pada banyak bahasa hal ini disebut sebagai *vector, array, sequence,* atau *list.*

Struktur data tersebut disebut sebagai *universal data structure*, kebanyakan bahasa pemrograman saat ini dapat menggunakannya dalam satu bentuk atau bentuk lainnya (json.org). JSON menggunakan format sebagai berikut : sebuah *object* adalah sekumpulan nama / nilai yang berpasangan yang tidak terurutkan. Sebuah *object* diawali dengan tanda "{" dan diakhiri dengan tanda "}". Setiap nama diikuti dengan tanda ":" serta pasangan nama/nilai dipisahkan oleh tanda "," seperti yang ditunjukkan oleh gambar 2.2. (json.org).



Gambar 2.2 Format penulisan *object* JSON

Sumber: (json.org)

Tabel 2.2 Penulisan *array* gabungan pada JSON

Penulisan *array* gabungan pada JSON	
1	[3, "three"]

Tabel 2.3 Penulisan *array* gabungan pada XML

Penulisan *array* gabungan pada XML	
1	<value>
2	<array>
3	<data>
4	<value><i4>3</i4></value>
5	<value><string>three</string></value>
6	</data>
7	</array>
8	</value>

Tabel 2.2 dan tabel 2.3 merupakan contoh penulisan *array* gabungan pada JSON dan XML, karena dalam penulisan format JSON sangat sederhana, hal ini berdampak pada ukuran *file* yang dihasilkan menjadi lebih ramping daripada format XML, dan menyebabkan kecepatan *loading* pengiriman data yang lebih cepat daripada format XML. *Javascript Object Notation* (JSON) pada penelitian ini digunakan sebagai format *request* dan *response* data dari dan ke *web service*.

2.3 Java *Server Page* (JSP)

JSP merupakan sebuah *template* untuk sebuah halaman web yang menggunakan kode bahasa Java untuk membuat sebuah dokumen HTML dinamis. JSP dijalankan pada komponen sisi *server* yang disebut sebagai JSP *container*, yang akan menerjemahkannya kedalam Java *Servlets*. Karena alasan tersebut *servlets* dan JSP sangat berkaitan. Karena JSP diterjemahkan kedalam *servlets*, maka JSP memiliki semua kelebihan *servlets*. (Hanna, 2001).

Beberapa diantaranya :

- Memiliki kelebihan performa dan *scalability* daripada skrip *Common Gateway Interface* (CGI) karena berada di memori dan *multithread*.
- Tidak memerlukan pengaturan khusus di sisi *client.*
- Mendukung penggunaan HTTP *sessions*, yang dimungkinkan untuk membuat program aplikasi.
- Memiliki akses penuh ke teknologi *Java* -jaringan, *threads,* dan konektivitas ke basis data- tanpa limitasi dari sisi *client*. (Hanna, 2001).

Tetapi sebagai tambahan, JSP juga memiliki kelebihannya sendiri yaitu :

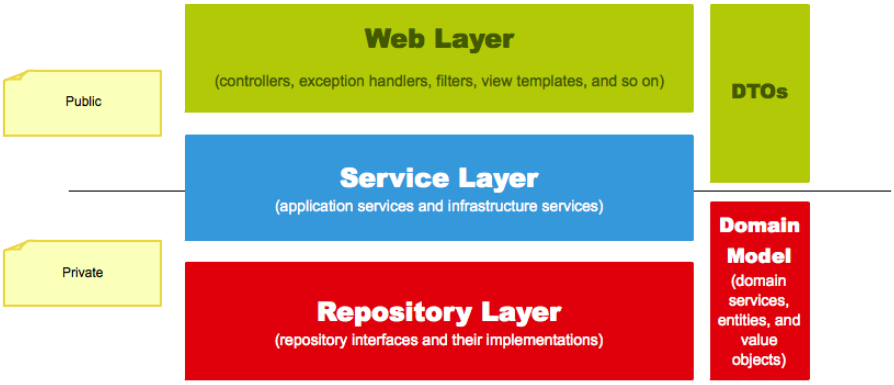
- Dapat secara otomatis di kompilasi ulang ketika dibutuhkan
- Karena berada pada area dokumen *web server*, mengalamatkan halaman JSP lebih sederhana daripada *servlets*.
- Karena halaman JSP seperti HTML, maka JSP memiliki kompatibilitas yang lebih tinggi pada alat pengembangan Web. (Hanna, 2001).

Java *Server Page* (JSP) digunakan sebagai halaman antarmuka (*view*) yang dihubungkan dengan penggunaan Spring Boot *Framework*.

2.4 Spring Boot *Framework*

Spring Boot merupakan sebuah *framework* Java yang digunakan untuk membuat aplikasi berbasis web dan aplikasi *enterprise*. Spring Boot merupakan *framework* baru yang dibuat untuk mempermudah *bootstrapping* dan pengembangan pada aplikasi Spring (Woods, 2014). Kelebihan Spring Boot jika dibandingkan dengan *framework* lain, pada Spring Boot memberikan berbagai macam fitur yang mengutamakan kebutuhan bisnis modern. Pada Spring Boot memberikan fleksibilitas untuk melakukan konfigurasi *beans* dengan berbagai cara seperti XML, *Annotations*, dan JavaConfig. Selain itu pada Spring Boot mudah digunakan karena kemampuan manajemen transaksi basis data, serta menyederhanakan integrasi dengan *framework* Java lain seperti JPA/Hibernate ORM, Struts, JSF, dan lainnya (Katamreddy, 2016).

Spring Boot *Framework* disini digunakan sebagai bagian *backend* yang digunakan untuk membuat *web service.* Pada Spring Boot *Framework* sendiri *layer-layer* nya jelas dan dapat digunakan untuk membuat sistem *micro-service* yang *reusable* dan *scalable*. Diagram arsitektur pada Spring Boot *framework* ditunjukkan oleh gambar 2.3.



Gambar 2.3 Diagram arsitektur Spring Boot *framework*

Sumber: (Kainulainen, 2014)

2.5 AngularJs

AngularJs merupakan sebuah teknologi pada sisi *client* yang ditulis menggunakan JavaScript. AngularJs berjalan dengan teknologi web yang telah lama ada (HTML, CSS, dan JavaScript) untuk membuat pengembangan web *apps* lebih mudah dan cepat daripada sebelumnya. (Lerner, 2013).

Beberapa kelebihan AngularJs untuk pengembangan web adalah :

- *Data Binding* : mensinkronisasi secara otomatis data antara *view* (HTML) dan *model* (variabel pada *JavaScript*).
- *Routing Support* : digunakan untuk membuat *Single Page Application* (SPA).
- *Templating* : membuat cetakan *view* dengan menggunakan HTML.

- *Directives* : memperluas HTML dengan elemen baru yang sebelumnya secara *default* tidak ada pada HTML.
- *Embeddable, testable,* dan *Injectable* : dapat dimasukkan pada aplikasi lain, teknolgi lain, dan dapat diuji. (Panda, 2014).

AngularJs digunakan untuk memanggil *web service* yang sudah diimplementasikan. Dalam angular terdapat 3 bagian utama, yaitu *app, controller* dan *service*. Bagian *app* ini berfungsi untuk menentukan apa saja *controller* dan *service* yang akan digunakan oleh angular. Bagian *controller* ini berfungsi untuk melakukan logika bisnis yang diperlukan untuk memproses data. Sedangkan pada bagian *service* ini merupakan bagian untuk mengirimkan atau menerima data dari RESTful *api service*.

2.6 PostgreSQL

PostgreSQL merupakan sebuah RDBMS *opensource* yang sangat bagus, yang dapat berjalan pada hampir semua sistem operasi. PostgreSQL mendukung ACID sepenuhnya yaitu *Atomicity, Consistency, Isolation, dan Durability*. Pada *PostgreSQL* juga mendukung hampir semua tipe data, serta memiliki *native programming interfaces* untuk hampir semua bahasa pemrograman (PostgreSQL.org).

Fitur-fitur modern yang dibawakan oleh PostgreSQL:

- *Complex Queries*
- *Foreign Keys*
- *Triggers*
- *Views*
- *Transactional Integrity*
- *Multiversion Concurrency Control*

Selain itu, PostgreSQL telah mendukung teknologi lama dengan menambahkan fitur-fitur baru pada:

- *Data Types*
- *Functions*
- *Operators*
- *Aggregate Functions*
- *Index Methods*
- *Procedural Languages*

PostgreSql memiliki beberapa kelebihan, yaitu:

- PostgreSQL memiliki arsitektur multiproses (*forking*) yang berarti memiliki stabilitas yang lebih tinggi, sebab satu proses anak yang mati tidak akan menyebabkan seluruh *daemon* mati.
- Dalam kondisi *load* tinggi (jumlah koneksi simultan besar), kecepatan PostgreSQL lebih cepat daripada MySQL untuk *query* dengan klausa *JOIN* yang kompleks, hal ini dikarenakan PostgreSQL mendukung *locking* di level yang lebih rendah, yaitu *row*.

PostgreSQL memiliki fitur berorientasi objek seperti pewarisan tabel dan tipe data, atau tipe data *array* yang kadang praktis untuk menyimpan banyak item data di dalam satu record.Dengan adanya kemampuan berorientasi objek ini maka di PostgreSQL, kita dapat mendefinisikan sebuah tabel yang mewarisi definisi tabel lain. PostgreSQL pada penelitian ini digunakan sebagai basis data tempat penyimpanan data pada sistem.

2.7 JSON Web Token (JWT)

JWT adalah standar yang mendefinisikan cara yang kompak dan mandiri untuk mentransmisikan informasi antar pihak sebagai objek JSON dengan aman. Informasi yang dikirimkan bisa diverifikasi dan dipercaya karena sudah ditandatangani secara digital. JWT biasanya digunakan untuk melakukan autentikasi ke sistem, dan pertukaran informasi (jwt.io). JWT terdiri dari 3 bagian yang dipisahkan oleh dot (.) yang direpresentasikan xxxx.yyyy.zzzz dimana bagian tersebut adalah :

- *Header*

  Pada *header* biasaya terdiri dari 2 bagian : jenis *token* dimana jenisnya adalah *JWT*, dan algoritma *hashing* yang digunakan misalkan SHA256, RSA dan lain sebagainya.
- *Payload*

  Pada bagian kedua yaitu *payload* yang didalamnya terdapat *claims. Claims* adalah penjelasan mengenai entitas (yang biasanya user) dan informasi tambahan mengenai user tersebut.
- *Signature*

  Untuk membuat *signature* harus mengambil *header* yang dikodekan, *payload* yang dikodekan, dan algoritma yang digunakan. *Signature* digunakan untuk melakukan pengecekan terhadap pengirim *JWT* siapa dia, dan memastikan pesan tidak berubah saat dikirmkan. JWT disini digunakan sebagai token untuk mengecek apakah *user* tersebut memiliki hak akses untuk mengakses *web service* yang dibuat.

2.8 *Unified Modelling Language* (UML)

UML adalah sebuah bahasa pemodelan untuk spesifikasi, visualisasi, merancang dan mendokumentasikan artifak dari perangkat lunak. UML menangkap keputusan dan pemahaman sistem yang harus dibangun. Ini digunakan untuk memahami, merancang, menjelajah, mengkonfigurasi, memelihara, dan mengendalikan informasi tentang sistem. Hal ini dimaksudkan untuk digunakan dalam semua metode pengembangan, siklus hidup, domain aplikasi dan media. (Rumbaugh, 2005).

Tujuan UML adalah membuat desain visual dari sebuah sistem. UML bukan bahasa pemrograman tetapi sebuah alat untuk membuat kode dari berbagai jenis bahasa pemrograman menggunakan diagram UML. UML memberikan cara untuk

menggambarkan cetak biru dari arsitektur sistem dalam sebuah diagram, meliputi elemen seperti :

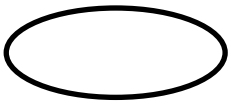
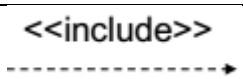
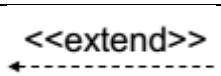
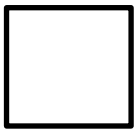
- Aktivitas
- Komponen individual dari sistem serta bagaimana interaksi dengan komponen lain
- Bagaimana sistem berjalan
- Bagaimana sebuah entitas berinteraksi dengan entitas lainnya
- *User interface*

Karena dalam pengembangan sistem penulis menggunakan pendekatan *object-oriented* maka diagram UML yang digunakan penulis adalah diagram UML *Behavioral* yang meliputi diagram *use case*, *entity relationship diagram*, *sequence* diagram, *class diagram*, serta *physical data model diagram*.

2.8.1 *Use case Diagram*

Use case diagram digunakan untuk menggambarkan fungsionalitas yang diinginkan dari sistem. *Use case* diagram menekankan pada "apa" yang diperbuat oleh sistem bukan "bagaimana". *Use case* menggambarkan interaksi antara aktor dan sistem. Simbol-simbol yang ada pada diagram *use case* dapat dilihat pada tabel 2.4.

Tabel 2.4 Simbol-simbol pada *use case diagram*

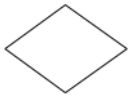
No	Gambar	Nama	Keterangan
1		Aktor	Menggambarkan sebuah tugas/peran dalam sistem
2		*Use case*	Menggambarkan apa yang dapat dilakukan pada sistem.
4		Relasi asosiasi	Menggambarkan hubungan antara objek-objek
5		Relasi *include*	Menggambarkan hubungan bahwa sebuah *use case* membutuhkan *use case* lainnya
6		Relasi *Extend*	Menggambarkan bahwa *use case* tertentu memperluas fungsionalitas dari suatu case
7		Relasi Generalisasi	Menggambarkan hubungan bahwa objek anak (*child*) berbagi perilaku dengan objek yang diatasnya (*parent*)
8		*System*	Menspesifikasikan paket yang menampilkan sebuah sistem secara terbatas.

9		*Note*	Elemen fisik saat sebuah aplikasi dijalankan dan menjelaskan detail dari sebuah obyek

2.8.2 *Entity Relationship Diagram* (ERD)

Entity Relationship Diagram (ERD) digunakan sebagai perancangan dari basis data yang akan diimplementasikan. Pada ERD hanya menjelaskan atribut yang dimiliki sebuah entitas, hubungan antar satu entitas dengan entitas lain, serta kardinalitas dari entitas, serta belum adanya *constraint-constraint* dari basis data. Simbol-simbol pada ERD dapat dilihat pada tabel 2.5.

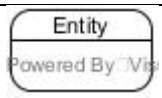
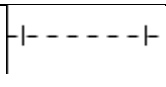
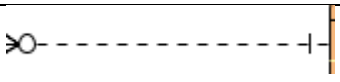
Tabel 2.5 Simbol-simbol pada *entity relationship diagram*

No	Gambar	Nama	Deskripsi
1		Entitas	Merupakan representasi dari tabel yang ada pada basis data
2		Atribut	Merupakan representasi dari kolom yang ada pada tabel dalam basis data
3		Relasi	Menggambarkan hubungan antara satu entitas dengan entitas lain

2.8.3 *Physical Data Model Diagram*

Physical data model diagram merepresentasikan desain cetak biru dari basis data relasional. Dimana diagram ini merepresentasikan bagaimana struktur data dan relasi antar data pada basis data relasional. Merupakan hal yang penting untuk mengetahui batasan dari *DBMS,* yang artinya kita harus mengetahui tipe data yang digunakan untuk kolom dan atribut apa saja yang dibutuhkan pada sebuah entitas. Simbol-simbol pada diagram *physical data model* dapat dilihat pada tabel 2.6.

Tabel 2.6 Simbol-simbol pada *physical data model diagram*

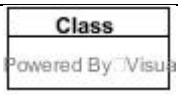
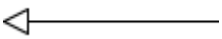
No	Gambar	Nama	Deskripsi
1		Entitas	Menggambarkan tabel dalam sebuah basis data
2		Relasi satu ke satu	Menggambarkan hubungan antar entitas, yang artinya hubungan antar entitas tepat satu
3		Relasi satu ke banyak	Menggambarkan hubungan antar entitas, yang artinya hubungan satu entitas dengan satu atau lebih entitas

4		Relasi banyak ke banyak	Menggambarkan hubungan antar entitas, yang artinya hubungan satu atau lebih entitas dengan satu atau lebih entitas.

2.8.4 *Class Diagram*

Class diagram digunakan untuk memodelkan dari kelas-kelas yang akan digunakan pada sistem yang akan dibuat. *Class diagram* menjelaskan megenai struktur dari sistem, atribut yang digunakan, operasi yang digunakan, serta hubungan antar kelas. Simbol-simbol pada *class diagram* dapat dilihat pada tabel 2.7.

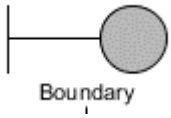
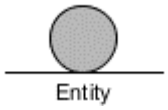
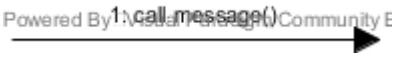
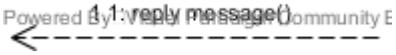
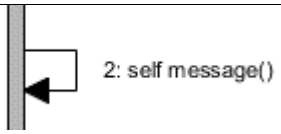
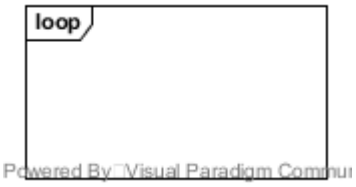
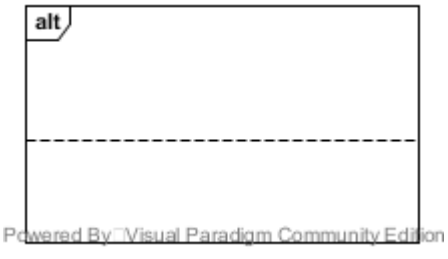
Tabel 2.7 Simbol-simbol pada *class diagram*

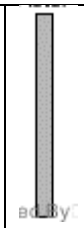
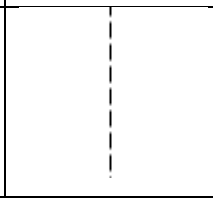
No	Gambar	Nama	Deskripsi
1		*Class*	Kelas yang digunakan pada sistem.
2		*Generalization*	Merepresentasikan bahwa sebuah kelas merupakan turunan dari kelas lain.
3		*Realization*	Menggambarkan sebuah kelas merupakan implementasi dari kelas lain (*interface*).
4		*Association*	Menggambarkan hubungan antar kelas, bahwa sebuah kelas menggunakan kelas lain.
5		*Aggregation*	Menggambarkan hubungan yang lebih khusus dari asosiasi, yaitu kelas induk "memiliki" kelas anak, tetapi jika kelas induk hilang kelas anak dapat tetap ada.
6		*Composition*	Menggambarkan hubungan yang lebih khusus dari asosiasi, yaitu kelas induk "memiliki" kelas anak, apabila kelas induk hilang maka kelas anak juga hilang, karena tidak dapat berdiri sendiri.

2.8.5 *Sequence Diagram*

Sequence diagram adalah diagram yang menunjukan bagaimana objek-objek saling berinteraksi dalam urutan-urutan tertentu dan dibatasi oleh waktu. *Sequence* diagram digunakan dalam memberikan gambaran detail mengenai setiap *use case* diagram yang telah dibuat. Setiap obyek yang terlibat dalam sebuah *use case* direpresentasikan dengan garis putus-putus vertikal yang menandakan garis hidup dari sebuah obyek, kemudian pesan yang dikirim oleh obyek direpresentasikan dengan garis panah horisontal, penggambaran pada *sequence* diagram berurutan secara waktu dari atas ke bawah. Simbol-simbol pada diagram *sequence* dapat dilihat pada tabel 2.8.

Tabel 2.8 Simbol-simbol pada *sequence diagram*

No	Gambar	Nama	Deskripsi
1		Aktor	Mewakili aktor dari sistem
2		Boundary	Mewakili semua antarmuka dari sistem
3		Controller	Mewakili semua objek didalam sistem dan memiliki tugas sebagai pengatur proses logik dari sistem
4		Entity	Mewakili semua objek didalam sistem yang merepresentasikan konsep bisnis dan penyimpanan data
5		Call Message	Mewakili pesan yang dikirimkan dari objek ke objek lainnya (method/function)
6		Reply Message	Mewakili balasan pesan (return) setelah call message dipanggil
7		Self message	Mewakili pesan yang dikirimkan ke dirinya sendiri
8		Loop	Mewakili perulangan yang dilakukan didalam sebuah sistem
9		Alternate	Mewakili jalur alternatif dalam sebuah sistem

10		Activation	Mewakili objek yang dipanggil pada saat tertentu
11		Lifeline	Merupakan siklus hidup dari objek.

2.9 Pengujian Perangkat Lunak

Pengujian perangkat lunak (*software testing*) merupakan suatu cara yang dilakukan untuk mendapatkan informasi mengenai kualitas dari produk yang sedang diuji. Pengujian perangkat lunak dapat memberikan gambaran mengenai perangkat lunak secara obyektif dan independen, dan memiliki manfaat dalam operasional bisnis untuk memahami tingkat risiko saat implementasi. (Sommerville, 2011).

2.9.1 *Unit Test*

Unit test adalah sebuah proses untuk menguji komponen-komponen dari program, seperti *method*, atau *object class*. Sebuah *method* adalah komponen yang paling sederhana dari program. Pengujian ini harus memanggil semua jalur yang ada pada *method* dengan beberapa masukan yang berbeda.

2.9.2 *Usability Test*

Usability Test digunakan untuk menguji kemudahan sistem yang dibuat. Nilai *usability* bisa didapatkan dengan *Sytem Usability Scale* (SUS). SUS memiliki kelebihan yaitu cara ini lebih murah karena tidak memerlukan banyak sumber daya dan lebih cepat karena telah tersedia *template* kuisioner sehingga lebih cepat dari pada membuat kuisioner baru (Thomas, 2015). Gambar 2.4 menunjukkan contoh *template* dari SUS.

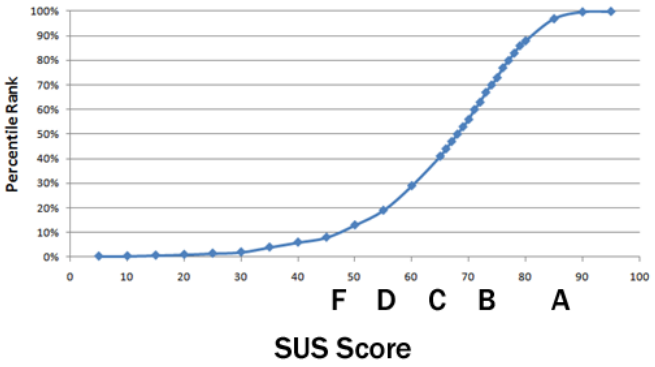
No	Template question
1	I think that I would like to use this system frequently.
2	I found the system unnecessarily complex.
3	I thought the system was easy to use.
4	I think that I would need the support of a technical person to be able to use this system.
5	I found the various functions in this system were well integrated.
6	I thought there was too much inconsistency in this system.
7	I would imagine that most people would learn to use this system very quickly.
8	I found the system very cumbersome to use.
9	I felt very confident using the system.
10	I needed to learn a lot of things before I could get going with this system.

Gambar 2.4 Template kuisioner SUS

Sumber: (Usabilitygeek, 2015)

SUS dihitung dengan memberikan skala satu sampai lima pada setiap pertanyaan. Untuk mendapatkan nilai akhir maka nilai pada pertanyaan dengan nomor ganjil akan dikurangi satu, dan untuk nomor genap maka akan diberi nilai lima kemudian dikurangi dengan nilai dari tes. Kemudian dijumlahkan antara nomor genap dan ganjil selanjutnya dikalikan dengan angka 2,5. Hasil sempurna dari SUS adalah 100 (Thomas, 2015).

Nilai rata-rata dari SUS adalah 68. Jika nilai yang didapatkan dibawah 68 maka dapat dikatakan ada masalah pada aplikasi yang dibuat sehingga membuat *usability* tidak optimal. Contoh *percentil rank* pada SUS dapat dilihat pada gambar 2.5.



Gambar 2.5 *Percentil Rank* SUS

Sumber: (Sauro, 2011)

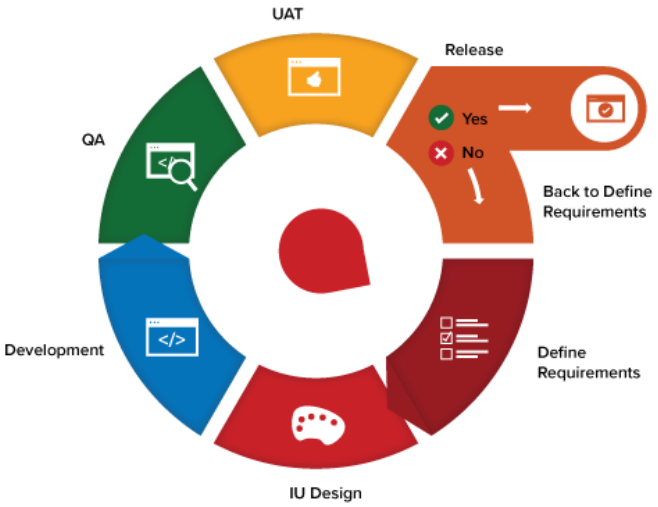
Usability juga memiliki beberapa karakteristik parameter sesuai dengan ISO/IEC 9126-4 yaitu *effectiveness*. *Effectiveness* disini berkaitan dengan akurasi dan penyelesaian dimana *user* dapat mencapai tujuan yang diinginkan. Salah satu cara pengukuran *efectiveness* adalah dengan menghitung *completion rate*. Pengukuran *completion rate* dihitung dengan memberikan nilai 1 pada partisipan ketika berhasil menyelesaikan tugas yang diberikan dan nilai 0 ketika partisipan tidak dapat melakukannya. Kemudian hasilnya dihitung menggunakan rumus pada persamaan 2.1. Menurut penelitian Jeff Sauro bahwa sistem dikatakan memiliki *usability* bagus jika nilai *effectiveness* lebih dari sama dengan 78%. (Mifsud, 2015).

$$effectiveness = \frac{number\ of\ tasks\ completed\ successfully}{total\ number\ of\ task\ undertaken}\ x\ 100\% \qquad (2.1)$$

2.10 *Software Development Life Cycle* (SDLC)

SDLC merupakan sebuah cara yang digunakan dalam perancangan, pengembangan, dan pengujian perangkat lunak. Tujuan dari *SDLC* adalah untuk menghasilkan perangkat lunak dengan kualitas terbaik yang memenuhi kebutuhan *user*, dan selesai dalam waktu yang ditentukan dengan biaya yang sesuai (tutorialspoint.com).

2.10.1 *Agile Method*



Gambar 2.6 Bagan aliran proses iterasi metode Agile pada pengembangan perangkat lunak

Sumber: (smartsheet.com)

Agile adalah sebuah filosofi. Cara berpikir tentang pengembangan perangkat lunak. Deksripsi resmi dari cara berpikir ini adalah *agile manifesto*, yang terdiri dari 4 nilai dan 12 prinsip. Metode *agile* adalah sebuah proses yang mendukung dari filosofi *agile*. Metode *agile* menggunakan kontrol versi, konfigurasi standar kode, dan memberikan presentasi / demo secara berkala terhadap perangkat yang dikembangkan kepada *stakeholder*. (Shore, 2007). Adapun bagan aliran proses iterasi metode *agile* pada pengembangan perangkat lunak dapat dilihat pada gambar 2.7 yang secara ringkasnya ada definisi kebutuhan, desain *user interface*, pengembangan/implementasi, proses pengujian, *user acceptance testing*, dan *release*. Metode *agile* disini digunakan sebagai SDLC dalam pengembangan perangkat lunak yang dibuat karena sifatnya yang lebih fleksibel jika ada perubahan dari *user*.