# BAB II

# TINJAUAN PUSTAKA

## 2.1 Resep Masakan

Resep masakan yaitu suatu susunan instruksi yang menunjukan cara membuat suatu masakan, dapat berupa minuman atau makanan (Pusat Bahasa Depdiknas, 2012). Selain menyiapkan bahan-bahan atau komposisi serta kuantitasnya untuk membuat sebuah masakan, di dalam resep masakan juga terdapat keterangan tentang alat-alat yang akan dipergunakan, lama waktu pembuatan, cara mengolah bahan masakan, serta jumlah sajian atas masakan tersebut (Rhodes, 2011).

## 2.2 Katalog *Online*

Menurut KBBI (Kamus Besar Bahasa Indonesia), katalog merupakan carik kartu, daftar, atau buku yang memuat nama benda atau informasi tertentu yang ingin disampaikan, disusun secara berurutan, teratur, dan alfabetis. Maka katalog *online* atau katalog dalam jaringan merupakan kumpulan atau daftar yang memuat informasi tertentu yang datanya disimpan dalam *database* komputer. Salah satu contoh katalog *online* adalah OPAC (*Online Public Access Catalogue*). OPAC menyediakan sarana penulusuran mandiri bagi pengguna karena dapat diakses dimana saja serta kapan saja hanya dengan memiliki aplikasi tertentu atau mengaksesnya melalui *website* menggunakan jaringan LAN atau WAN.

## 2.3 REST API

API (*Application Programming Interface*) adalah kumpulan perintah, fungsi, komponen, dan protokol yang disediakan oleh sistem operasi atau bahasa pemrograman tertentu yang dapat digunakan *programmer* saat membangun perangkat lunak untuk sistem operasi tertentu. Salah satu jenis API adalah REST API (Ekajogja, 2015).

REST (*Representational State Transfer*) merupakan arsitektur metode komunikasi yang sering diterapkan dalam *web service*. Arsitektur REST, dijalankan melalui HTTP, kemudian melibatkan proses pembacaan laman *web* tertentu yang memiliki *file text*, XML atau JSON. *File* inilah yang menguraikan dan memuat data yang hendak ditampilkan. Setelah itu data tersebut akan dikembalikan kepada *user* dari *server*, kemudian *user* dapat melihatnya dalam tampilan dalam *web* atau aplikasi. Biasanya REST API bekerja sama halnya seperti *website*, *user* membuat panggilan atau *request* ke *server*, lalu mendapatkan data dari server melalui HTTP *response* (Feridi, 2016).

Komponen dari HTTP *request* adalah:

1. HTTP *method* yang digunakan misalnya GET, POST, DELETE, PUT.
2. *Uniform Resource Identifier* (URI) untuk mengidentifikasikan lokasi *resource* pada *server*.
3. HTTP *version*, menunjukan versi dari HTTP yang digunakan
4. *Request Header* yang berisi metadata untuk HTTP *request*. Contohnya tipe *client*/*browser*, format yang didukung *client*, pengaturan *cache*.
5. *Request Body* yang merupakan konten dari data.

Sedangkan, komponen dari HTTP *response* adalah:

1. Status/*response code* yang mengindikasikan status *server* terhadap *resource* yang diminta, misalnya: 404 yang berarti *resource* tidak ditemukan atau 200 yang artinya *request* sukses.
2. HTTP *version* menunjukan versi dari HTTP yang digunakan.
3. *Response Header*, berisi metadata untuk HTTP *response*. Contohnya: tipe *server*, panjang konten, tipe content, waktu respon.
4. *Response Body*, konten dari data yang diberikan.

## 2.4 Sistem Operasi *Android*

*Android* merupakan sistem operasi berbasis *Linux*. *Android* dikembangkan oleh Google yang bekerja sama dengan Open Handset Alliance. Sistem operasi *Android* adalah sebuah sistem operasi yang mencakup sistem operasi, *middleware*, dan aplikasi (Safaat, 2011). Android merupakan salah satu sistem operasi pada telepon seluler yang bersifat terbuka (*open source*).

Kelebihan dari sistem operasi Android adalah sebagai berikut (Safaat, 2011):

1. *Complete Platform*

   Sistem operasi *Android* adalah sistem operasi yang menyediakan *tools* yang berguna untuk membangun aplikasi yang kemudian aplikasi tersebut dapat lebih dikembangkan oleh para *developer*.

2. *Open Source Platform*

   Karena bersifat terbuka, membuat *Android platform* mudah dikembangkan oleh para *developer*.

3. *Free Platform*

   *Developer* dengan bebas dapat mengembangkan, serta mendistribusikan dan memperdagangkan sistem operasi *Android* tanpa harus membayar royalti untuk mendapatkan lisensi.

Sistem operasi Android mempunyai bentuk arsitektur jika dilihat secara garis besar adalah sebagai berikut (Subiyantoro, 2013):

1. *Applications and Widgets*

   Puncak dari diagram arsitektur *Android* adalah lapisan aplikasi dan *widget*. Lapisan aplikasi merupakan lapisan yang paling tampak pada pengguna ketika menjalankan program. Lapisan aplikasi *Android* sangat berbeda dibandingkan dengan sistem operasi lainnya. Pada Android, baik aplikasi inti maupun aplikasi pihak ketiga, berjalan di atas lapisan aplikasi menggunakan API (*Application Programming Interface*) yang sama (Subiyantoro, 2013).

2. *Applications Frameworks*

   Kerangka aplikasi menyediakan beberapa kelas yang dapat digunakan untuk mengembangkan aplikasi *Android*. Selain itu, juga menyediakan abstraksi *generic* untuk mengakses perangkat,

serta mengatur tampilan *user interface* dan sumber daya aplikasi (Subiyantoro, 2013).

3. *Libraries*

   *Android* menggunakan beberapa paket pustaka yang terdapat pada C/C++ dengan standar *Berkeley Software Distribution* (BSD) hanya setenngah dari yang aslinya untuk tertanam pada *Kernel Linux* (Subiyantoro, 2013)

4. *Android Runtimes*

   Pada *Android* tertanam paket pustaka inti yang menyediakan sebagian besar fungsi *Android*. Inilah yang membedakan *Android* dibandingkan dengan sistem operasi lain yang juga mengimplementasikan Linux. *Android Runtimes* merupakan mesin virtual yuang membuat aplikasi *Android* menjadi lebih tangguh dengan paket pustaka yang telah ada (Subiyantoro, 2013).

5. *Linux Kernel*

   *Android* dibangun di atas *Kernel* 2.6, tetapi secara keseluruhan *Android* bukanlah Linux karena pada *Android* tidak hanya terdapat paket standar yang dimiliki Linux lainnya (Subiyantoro, 2013).

## 2.5 *Flowchart* Diagram

*Flowchart* merupakan teknik analitis untuk menjelaskan aspek-aspek sistem informasi secara tepat, jelas, dan logis. *Flowchart* menggunakan serangkaian simbol untuk menguraikan prosedur transaksi yang digunakan, sekaligus menguraikan aliran data dalam sebuah sistem.

Krismiaji (2005) membedakan jenis *flowchart* yang biasa digunakan menjadi lima tipe, yaitu:

1. *System Flowchart*

   *System flowchart* merupakan bagan yang menunjukan arus pekerjaan secara keseluruhan dari sebuah sistem, juga menjelaskan urutan dari prosedur-prosedur yang ada di dalam sistem serta menunjukan apa yang dikerjakan di dalam sistem.

2. *Document Flowchart*

   Disebut juga sebagai *flowchart* formulir. *Flowchart* ini menunjukan alur dari laporan dan formulir. Menggunakan simbol yang sama dengan *system flowchart*.

3. *Schematic Flowchart*

   Menggambarkan prosedur sebuah sistem. *Flowchart* ini menggunakan simbol yang sama dengan *flowchart system* dan ditambah dengan gambar-gambar komputer serta peralatan lain yang mendukungnya.

4. *Program Flowchart*

   Tipe *flowchart* ini terdiri dari dua, yaitu *flowchart* logika program untuk menggambarkan langkah di dalam program komputer secara logika, dan

*flowchart* program komputer terinci. *Flowchart* logika program dipersiapkan oleh analis sistem.

5. *Process Flowchart*

*Flowchart* ini menggunakan analis sistem untuk menggambarkan proses dalam suatu prosedur.

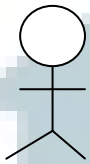
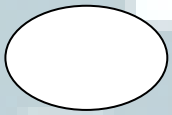
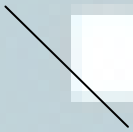
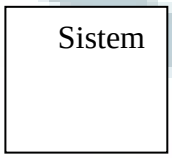
## 2.6 *Unified Modeling Language*

Menurut Nugroho (2010), *Unified Modeling Language* (UML) adalah bentuk bahasa pemodelan yang ditujukan untuk suatu sistem atau perangkat lunak yang berorientasi objek. UML berfungsi sebagai pembantu para pengembang untuk menggambarkan alur dari sebuah sistem yang akan dibangun. Gambaran mengenai alur sistem akan terwakili oleh simbol-simbol yang ada di dalam diagram-diagram.

### *2.6.1 Use Case Diagram*

*Use Case Diagram* adalah unit fungsionalitas koheren yang dapat diekspresikan sebagai transaksi-transaksi yang terjadi antara aktor dan sistem (Nugroho, 2008). Kegunaan dari *use case* adalah menjelaskan suatu bagian perilaku sistem yang bersifat koheren tanpa perlu menyingkap struktur internal sistem atau perangkat lunak yang sedang dikembangkan. Tabel 2.1 menunjukan simbol-simbol pada *use case diagram.*

Tabel 2.1 Daftar Simbol *Use Case*

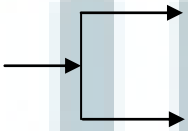
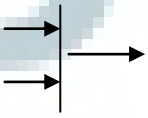
| Simbol | Nama | Keterangan |
|---|---|---|
|  | *Actor* | Orang atau pengguna yang berada di luar sistem. |
|  | *Use Case* | Digunakan untuk fungsionalitas dari sistem. |
|  | *Connectivity* | Menunjukan hubungan atau arah informasi mengalir. |
|   | *System Boundary* | Batas penggambaran sistem dalam sebuah kasus. |

### *2.6.2 Activity Diagram*

Diagram ini bertujuan untuk memodelkan komputasi dan aliran kerja yang terjadi dalam sistem atau perangkat lunak yang sedang dikembangkan (Nugroho, 2008). Diagram ini menggambarkan berbagai alur aktivitas dalam sistem yang sedang dirancang, bagaimana masing-asing alur berawal, *decision* yang mungkin terjadi, dan bagaimana berakhir. *Activity diagram* juga dapat menggambarkan proses paralel

yang mungkin terjadi pada beberapa eksekusi. Tabel 2.2 adalah daftar simbol pada *activity diagram.*

Tabel 2.2 Daftar Simbol *Activity Diagram*

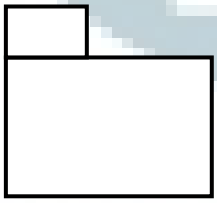
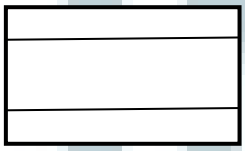
| **Simbol** | **Nama** | **Keterangan** |
|---|---|---|
|  | *Start Point* | Menunjukan dimulainya suatu *workflow* pada sebuah *activity diagram*. |
|  | *End Point* | Menggambarkan akhir dari sebuah *activity diagram*. |
|  | *Activities* | Menggambarkan sebuah pekerjaan atau tugas dalam *workflow*. |
|  | *Fork* | Penunjukan objek mana yang bertanggung jawab atas suatu aktivitas. |
|  | *Joint* | |

### *2.6.3 Class Diagram*

Diagram kelas menunjukan interaksi antar kelas dalam sistem (Sholiq, 2006). Diagram kelas mengandung informasi dan tingkah laku segala sesuatu yang berkaitan dengan informasi tersebut. Berikut adalah kegunaan dari *class diagram* (Hariyanto, 2004), serta simbol-simbolnya pada Tabel 2.3:

1. Mengelompokan beberapa obyek menjadi kelas, artinya mengabstraksikan masalah yang sedang dihadapi.
2. Definisi-definisi *common* (seperti nama kelas dan atribut) cukup disimpan sekali per objek.

Tabel 2.3 Daftar Simbol *Class Diagram*

| **Simbol** | **Nama** | **Keterangan** |
|---|---|---|
|  | *Package* | Merupakan sebuah bungkusan dari satu atau lebih kelas. |
|  | *Class* | Kelas pada struktur sistem. |
|  | *Dependency* | Relasi antar kelas dengan makna kebergantungan antar kelas. |

| | | |
|---|---|---|
|  | *Note* | Menambahkan catatan pada diagram. |

## 2.7 *System Development Life Cycle*

*System Development Life* (SDLC) adalah suatu pendekatan yang memiliki tahap-tahap untuk melakukan analisa dan membangun suatu rancangan sistem dangan siklus yang lebih spesifik tergadap kegiatan pengguna (Kendall & Kendalll, 2006). Terdapat empat langkah kunci yaitu, perencanaan dan seleksi analisis, desain, implementasi, dan operasional (Valacic, George & Hoffer, 2012). SDLC juga merupakan sebuah proses memahami bagaimana sistem informasi dapat mendukung kebutuhan bisnis, merancang sistem, membangun sistem, dan memberikannya kepada pengguna (Dennis, Wixom &Tegarden, 2005).

Ada empat metodologi pengembangan *software* berbasis SDLC, yaitu:

1. *Waterfall*
2. *Prototyping*
3. RAD (*Rapid Application Development*)
4. Iterasi

## 2.8 Metode *Waterfall*

Metode ini menampilkan proses pengembangan secara keseluruhan digambarkan dalam urutan linear (Galin, 2004). Ada tujuh tahapan pada metode ini, yaitu:

1. *Requirements Definition*
2. *Analysis*
3. *Design*
4. *Coding*
5. *System Tests*
6. *Installation and Conversion*
7. *Operation and Maintenance*

Model *waterfall* tidak dapat mengulangi perngulangan tahapan, apa yang menjadi *output* pada tahap A akan menjadi *input* pada tahap B. Di bawah ini (Tabel 2.4) adalah tabel perbandingan antar model yang kemudian menjadi landasan penulis dalam memilih model perancangan sistem untuk pembuatan aplikasi e-katalog resep masakan.

Tabel 2.4 Tabel Perbandingan Model Perancangan Sistem

| **Metode** | **Cara Kerja** |
|---|---|
| *Waterfall* | Tahapan-tahapannya tidak dapat diulang, cocok untk proyek berskala kecil dimana *requirement* dari *user* sudah tetap dan tidak berubah lagi. |

| | |
|---|---|
| *Prototyping* | Adanya pembuatan *prototype* dari sebuah aplikasi sebelum aplikasi tersebut memasuki tahap *design*. |
| RAD (*Rapid Application Development*) | Memerlukan beberapa teknik dan alat-alat khusus agar proses lebih cepat, misalnya melakukan sesi *Join Application Development* atau penggunaan alat-alat *Case Aided Software Engineering* (CASE) *Tools*. |
| Iterasi | Memperbolehkan pengulangan tahap jika terdapat kekurangan atau kesalahan. |

Model *waterfall* memiliki beberapa kelebihan dan kekurangan seperti yang akan dipaparkan di bawah ini.

Kelebihan metode *Waterfall*:

1. Pengembangannya terstruktur/bertahap.
2. Pengerjaan proyek akan terjadwal dengan baik dan mudah dikontrol.
3. Merupakan model pengembangan paling handal dan paling lama digunakan.

4. Dokumen pengembangan sistem sangat terorganisir, karena setiap tahap harus diselesaikan secara lengkap sebelum melangkah ke tahap selanjutnya.

Kekurangan metode *Waterfall*:

1. Diperlukan manajemen yang baik karena proses pengembangan tidak dapat dilakukan secara berulang.
2. Kesalahan kecil pada suatu tahap akan berdampak besar pada tahap selanjutnya.
3. Model ini tidak cocok untuk pemodelan pengembangan sebuah proyek yang memiliki kompleksitas tinggi karena tahap-tahap pada model *waterfall* tidak dapat berulang.
4. Waktu pengembangannya lama karena *input* tahap berikutnya adalah *output* dari tahap sebelumnya. Maka, jika satu tahap waktu penyelesaiannya terlambat, maka waktu keseluruhan pengembangan juga akan terlambat.