BAB II

LANDASAN TEORI

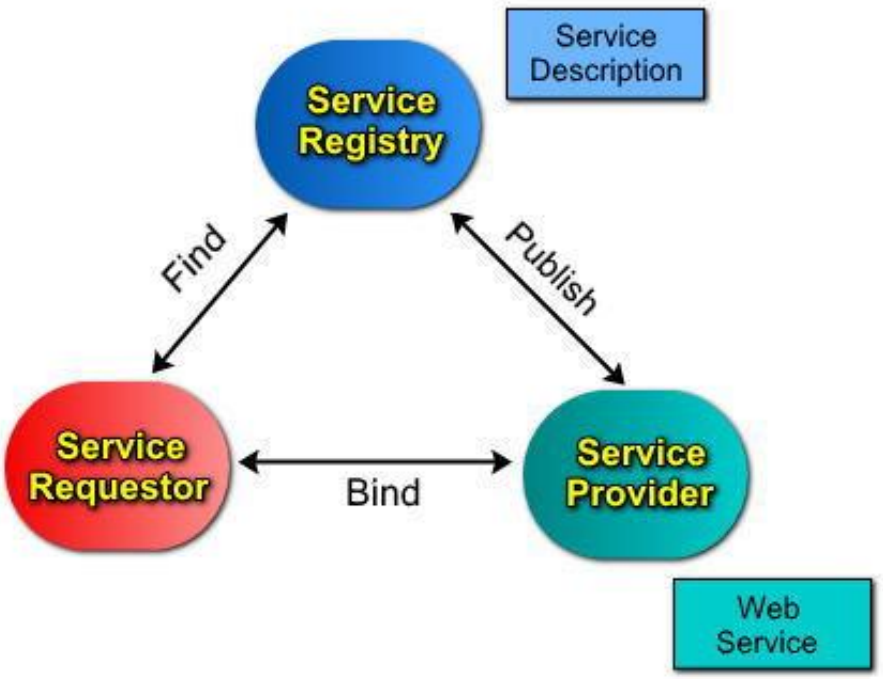
2.1 *Web service*

Web service adalah salah satu bentuk sistem perangkat lunak yang didesain untuk mendukung interaksi mesin-ke-mesin melalui jaringan. *Web service* memiliki *interface* yang dideskripsikan dalam format yang dapat dibaca oleh mesin. Sistem-sistem lainnya berinteraksi dengan *web service* menggunakan pesan SOAP yang umumnya dikirim melalui HTTP dalam bentuk XML [8].

Definisi diatas diberikan oleh *World Wide Web Consortium*(W3C) yang merupakan badan yang menciptakan dan mengembangkan standar *web service.* Tetapi secara umum, *web service* tidak terbatas hanya pada standar SOAP saja. Salah satu pustaka yang mengulas lengkap tentang *web service* menyebutkan definisi yang lebih umum: *web service* adalah aplikasi yang diakses melalui *internet* menggunakan protokol standar *internet* dan menggunakan XML sebagai format pesannya.

2.1.1 Arsitektur *Web service*

Secara umum, arsitektur *web service* dapat dilihat pada gambar II.1.



Gambar II. 1 Arsitektur Web Service [6].

Pada gambar diatas, ada tiga komponen yang membuat *web service* berjalan. Ketiga komponen itu adalah:

1. ***Service provide****r*, merupakan pemilik *Web service* yang berfungsi menyediakan kumpulan operasi dari *Web service*.
2. ***Service requesto****r*, merupakan aplikasi yang bertindak sebagai klien dari *Web service* yang mencari dan memulai interaksi terhadap layanan yang disediakan.
3. ***Service registry***, merupakan tempat dimana *Service provider* mempublikasikan layanannya. Pada arsitektur *Web service*, Service registry bersifat optional. Teknologi *web service* memungkinkan kita dapat menghubungkan berbagai jenis *software* yang memiliki *platform* dan sistem operasi yang berbeda.

2.1.2 Jenis-Jenis *Web service*

Jenis *web service* dapat dibagi menjadi dua, yaitu REST dan SOAP.

1. ***Representational State Transfer* (REST)**

REST adalah salah satu jenis *web service* yang menerapkan konsep perpindahan antar *state*. *State* disini dapat digambarkan seperti jika *browser* meminta suatu halaman *web*, maka *server* akan mengirimkan *state* halaman *web* yang sekarang ke *browser*. Bernavigasi melalui *link-link* yang disediakan sama halnya dengan mengganti *state* dari halaman *web*. Begitu pula REST bekerja, dengan bernavigasi melalui *link-link* HTTP untuk melakukan aktivitas tertentu, seakan-akan terjadi perpindahan *state* satu sama lain. Perintah HTTP yang bisa digunakan adalah fungsi *GET, POST, PUT* atau *DELETE*. Balasan yang dikirimkan adalah dalam bentuk XML sederhana tanpa ada protokol pemaketan data, sehingga informasi yang diterima lebih mudah dibaca dan diparsing disisi *client*.

Dalam pengaplikasiannya, REST lebih banyak digunakan untuk *web service* yang berorientasi pada *resource*. Maksud orientasi pada *resource* adalah orientasi yang menyediakan *resource-resource* sebagai layanannya dan bukan kumpulan-kumpulan dari aktivitas yang mengolah *resource* itu. Alasan mengapa REST tidak digunakan dalam skripsi ini karena orientasi pada *resource*nya itu, sedangkan aplikasi lowongan pekerjaan membutuhkan pemanggilan metode yang bisa dikerjakan terhadap kumpulan *resource* data lowongan. Selain itu, karena standarnya yang kurang sehingga tidak begitu cocok diterapkan dalam aplikasi yang membutuhkan kerjasama antar aplikasi lain, dimana standar yang baik akan

sangat berguna karena berbicara dalam satu bahasa yang sama. Beberapa contoh *web service* yang menggunakan REST adalah: Flickr API(*Application Program Interface*), YouTube API, Amazon API.

2. ***Simple Object Access Protokol*** **(SOAP)**

SOAP merupakan protokol standar yang ringan dan ditujukan untuk pertukaran informasi dalam desentralisasi sistem. Protokol ini melakukan pemanggilan metode dalam bahasa XML (*Extensible Markup Language*). Namun, selain itu juga kita dapat menggunakan format pertukaran data lainnya yaitu JSON (*JavaScript Object Notation*). Dengan demikian, SOAP adalah suatu mekanisme yang sederhana untuk melakukan pertukaran struktur dan tipe informasi dalam lingkungan yang tersebar dan terdistribusi menggunakan XML/JSON. Dokumen SOAP yang digunakan untuk melakukan *request* disebut SOAP *request* sedangkan dokumen SOAP yang diperoleh dari web service disebut dengan SOAP *response*.

Secara garis besar, protocol SOAP dapat dibagi menjadi beberapa bagian berikut :

a. SOAP *Envelope*

Untuk mendefinisikan apa yang terdapat dalam pesan (*message*) yang dikirim, dan kepada siapa pesan tersebut akan digunakan. Elemen ini merupakan root dari pesan dan bersifat wajib.

b. SOAP *Header*

Elemen ini berisi informasi header dan bersifat opsional. Informasi yang ada pada *header* tidak ditampilkan di aplikasi.

c. SOAP *Body*

Elemen ini berisi panggilan dan merespon informasi. Elemen ini wajib.

d. SOAP *Encoding Rules*

Mendefinisikan mekanisme serialisasi yang dapat digunakan untuk pertukaran tipe data yang didefinisikan dari tiap aplikasi.

e. SOAP RPC *Representation*

Menyediakan suatu standar yang dapat digunakan untuk mewakili *remote procedure call* (RPC) dan data hasil kembaliannya.

f. SOAP *Binding*

Mendefinisikan standar yang digunakan utnuk pertukaran SOAP *Envelope* antara *peers* dengan protokol yang umum untuk transportasi.

Sintaks dari protokol SOAP adalah sebagai berikut :

```
<?xml version="1.0"?>
<soap:Envelope
     xmlns:soap="http://www.w3.org/2001/12/soap-
envelope"
     soap:encodingStyle="http://
www.w3.org/2001/12/soap-envelope">
<soap:Header>
  . . .
  . . .
</soap:Header>
<soap:Body>
  . . .
  . . .
     <soap:Fault>
       . . .
       . . .
     </soap:Fault>
</soap:Body>
</soap:Envelope>
```

2.1.3 Web service Definition Language (WSDL)

WSDL merupakan suatu dokumen berbasis XML yang berfungsi untuk menjelaskan fungsi *public* dan metode-metode yang tersedia dalam *web service*. Selain itu, WSDL juga mengatur *end-point* pada pesan-pesan yang berisi dokumen maupun informasi prosedur.

Dokumen WSDL sanagat kompleks, dan memungkinkan untuk mendeskripsikan *end-point* serta pesan yang ada pada protokol jaringan. Salah satu bagian dokumen WSDL yang cukup penting dalam melakukan interaksi dengan SOAP, HTTP, serta MIME adalah *binding*.

SOAP dapat berjalan sendiri tanpa bantuan WSDL, tetapi hal ini akan menyulitkan pengembang di sisi *client* karena tidak tahu harus mengharapkan operasi dan bentuk data apa. Karena itu WSDL diciptakan sebagai pendefinisi sebuah *web service*. Dalam tugas akhir ini, *web service* yang dibangun akan dijabarkan menggunakan WSDL.

Elemen-elemen yang terdapat dalam WSDL adalah sebagai berikut :

1. *Elemen Types*

   Elemen Types menyertakan definisi tipe data yang sesuai untuk pertukaran *message*. Untuk meningkatkan operasi dari dalam dan dukungan netral *platform*, WSDL sering menggunakan XSD sebagai peraturan sistem dan memperlakukannya sebagai tipe sistem pada umumnya.

2. *Elemen Message*

   Message terdiri dari satu atau beberapa bagian, dan masing-masing bagian digabung dengan beberapa tipe sistem menggunakan atribut.

3. *Elemen Port Type*

   Elemen ini menjelaskan tentang operasi dan abstrak *message* yang terlibat di dalamnya. Atribut "*name*" dari elemen ini menyediakan nama unik di antara semua jenis port yang didefinisikan pada penutup dokumen WSDL.

4. *Elemen Binding*

   Binding berfungsi untuk menyatukan format *message* dan detail protokol untuk operasi. Umumnya ada beberapa nomor *binding* untuk menentukan tipe port.

5. *Elemen Port*

   Elemen ini bukanlah elemen *port type*, namun subelemen dalam elemen *service*. Fungsinya untuk mendefinisikan *end-point* individu melalui alamat tunggal khusus untuk *binding*.

6. *Elemen service*

   Kelompok *services* digunakan untuk mengatur hubungan yang terjadi antara port. Dengan demikian, elemen-elemen port akan diletakkan dalam *group services*.

Sintaks dari dokumen WSDL adakah sebagai berikut :

```
<definitions>
<types>
   definition of types……
</types>
<message>
   definition of message…….
</message>
<portType>
   definition of port……...
</portType>
```

```
<binding>
   definition of bind…….
</binding>
</definitions>
```

2.1.4 UDDI (*Uniform Description, Discovery, and Integration*)

Dalam penerapannya, *web service* memerlukan sumber dari semua informasi tentang *SOAP message*, cara membuat respons, mengumumkan *service* yang ada, dan sebagainya. Dengan demikian, aplikasi dapat menggunakannya untuk mencari WSDL. Untuk memenuhi hal tersebut, diperlukan UDDI yang menyediakan *service* untuk mencari informasi yang diperlukan.

UDDI memungkinkan suatu perusahaan untuk mengumumkan detail dari *web service* yang tersedia, biasanya melalui kata kunci atau deskripsi. Dengan adanya UDDI tersebut, kita dapat mengetahui (*discover*) dimana *web service* perusahaan berada.

2.1.5 JSON [3]

JSON (*JavaScript Object Notation*) adalah format pertukaran data yang ringan, mudah dibaca dan ditulis oleh manusia, serta mudah diterjemahkan dan dibuat (*generate*) oleh komputer. Format ini dibuat berdasarkan bagian dari Bahasa Pemprograman JavaScript, Standar ECMA-262 Edisi ke-3 – Desember 1999. JSON merupakan format teks yang tidak bergantung pada bahasa pemprograman apapun karena menggunakan gaya bahasa yang umum digunakan oleh *programmer* keluarga C termasuk C, C++, C#, Java, JavaScript, Perl, Python dll. Oleh karena sifat-sifat tersebut, menjadikan JSON ideal sebagai bahasa pertukaran-data.

Penulis menggunakan metode JSON dalam pengiriman data yang dilakukan, karena JSON memiliki beberapa kelebihan - kelebihan dibandingkan XML, kelebihan – kelebihan tersebut adalah:

1. Format Penulisan

   Untuk merepresentasikan sebuah struktur data yang rumit dan berbentuk hirarkis penulisan JSON relatif lebih terstruktur dan mudah.

2. Ukuran

   Ukuran karakter yang dibutuhkan JSON lebih kecil dibandingkan XML untuk data yang sama. Hal ini tentu berpengaruh pula pada kecepatan pertukaran data, walaupun tidak signifikan untuk data yang kecil, namun cukup berarti jika koneksi yang digunakan relatif lambat untuk mengakses aplikasi web kaya fitur yang memanfaatkan pertukaran data. Di sini JSON lebih unggul dibandingkan XML, kecuali jika data dikompresi terlebih dahulu sebelum dikirimkan, perbedaan JSON dan XML yang telah dikompresi tidaklah signifikan.

3. Browser Parsing

   Proses parsing merupakan proses pengenalan token atau bagian-bagian kecil dalam rangkaian dokumen XML/JSON. Contohnya, terdapat data text dalam format JSON. Data tersebut harus di-parsing terlebih dahulu sebelum dapat diakses dan dimanipulasi. Browser parsing berarti proses parsing yang terjadi pada sisi client/browser.

   Melakukan browser parsing pada JSON lebih sederhana dibandingkan pada XML, JSON menggunakan function JavaScript eval() untuk melakukan

parsing. Sementara dokumen XML di-parsing oleh XMLHttpRequest. Rata-rata survei menobatkan JSON sebagai pemenang jika diadu kecepatan parsingnya.

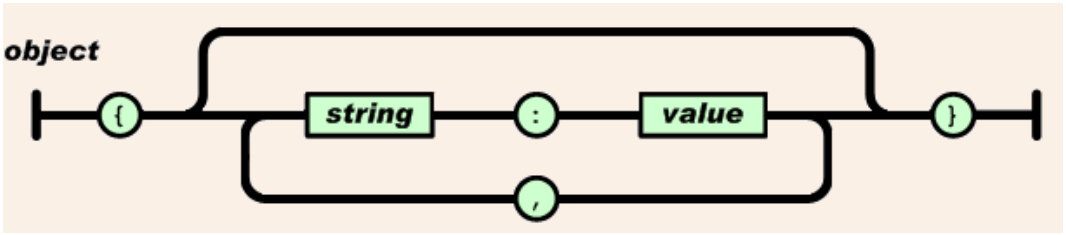
JSON terbuat dari dua struktur:

1. Kumpulan pasangan nama/nilai. Pada beberapa bahasa, hal ini dinyatakansebagai objek (*object*), rekaman (*record*), struktur (*struct*), kamus (*dictionary*), tabel hash (*hash table*), daftar berkunci (*keyed list*), atau associative array.
2. Daftar nilai terurutkan (*an ordered list of values*). Pada kebanyakan bahasa, hal ini dinyatakan sebagai larik (*array*), vektor (*vector*), daftar (*list*), atau urutan (*sequence*).

Struktur-struktur data ini disebut sebagai struktur data universal. Pada dasarnya, semua bahasa pemprograman moderen mendukung struktur data ini dalam bentuk yang sama maupun berlainan. Hal ini pantas disebut demikian karena format data mudah dipertukarkan dengan bahasa-bahasa pemprograman yang juga berdasarkan pada struktur data ini. JSON menggunakan bentuk sebagai berikut:

1. Objek

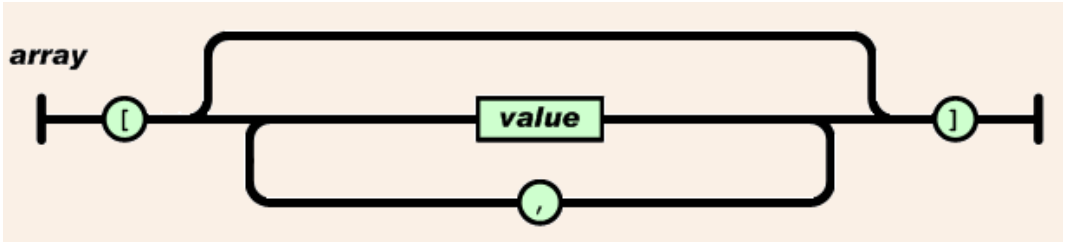
Objek adalah sepasang nama/nilai yang tidak terurutkan. Objek dimulai dengan { (kurung kurawal buka) dan diakhiri dengan } (kurung kurawal tutup). Setiap nama diikuti dengan : (titik dua) dan setiap pasangan nama/nilai dipisahkan oleh , (koma).



Gambar II. 2 Objek JSON

2. Larik

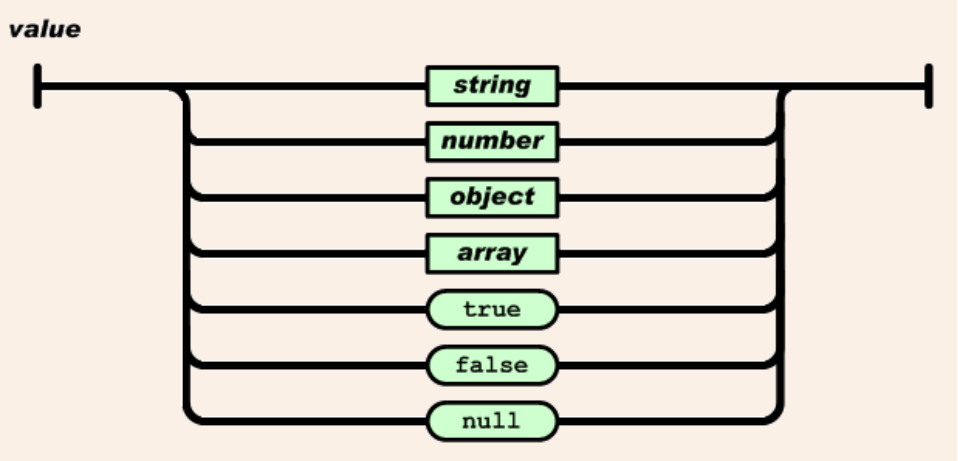
Larik adalah kumpulan nilai yang terurutkan. Larik dimulai dengan [(kurung kotak buka) dan diakhiri dengan] (kurung kotak tutup). Setiap nilai dipisahkan oleh , (koma).



Gambar II. 3 Larik JSON

3. Nilai

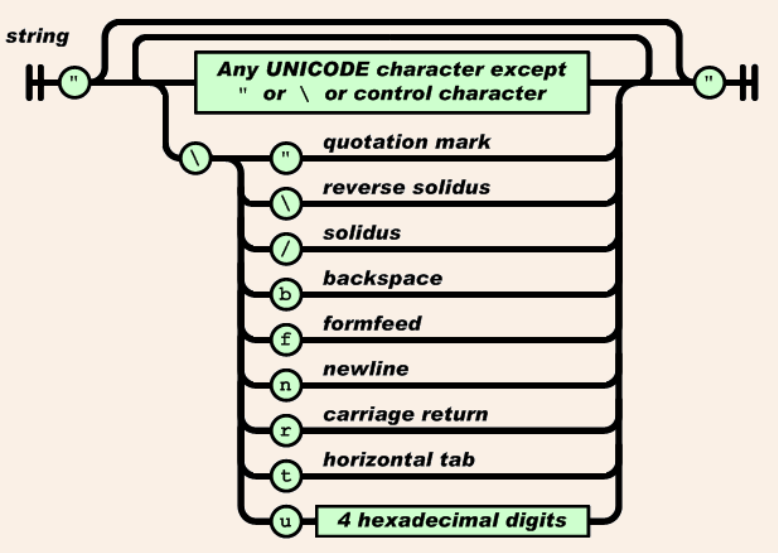
Nilai (*value*) dapat berupa sebuah string dalam tanda kutip ganda, atau angka, atau *true* atau *false* atau *null*, atau sebuah objek atau sebuah larik. Struktur-struktur tersebut dapat disusun bertingkat.



Gambar II. 4 Nilai JSON

4. String

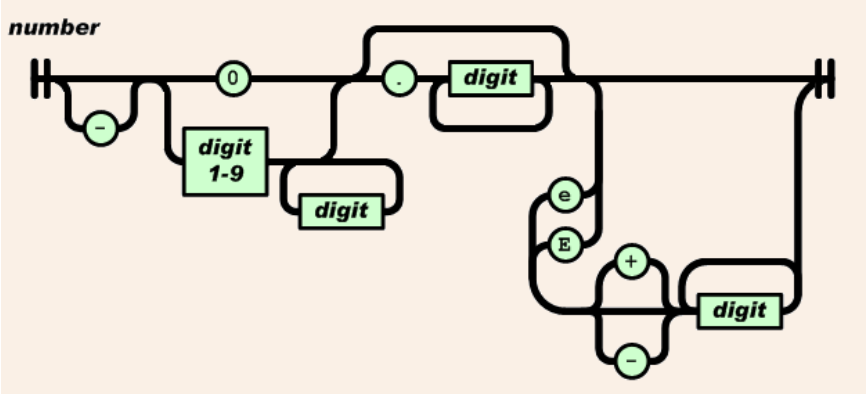
String adalah kumpulan dari nol atau lebih karakter *Unicode*, yang dibungkus dengan tanda kutip ganda. Di dalam string dapat digunakan *backslash escapes* "\" untuk membentuk karakter khusus. Sebuah karakter mewakili karakter tunggal pada string. String sangat mirip dengan string C atau Java.



Gambar II. 5 String JSON

5. Angka

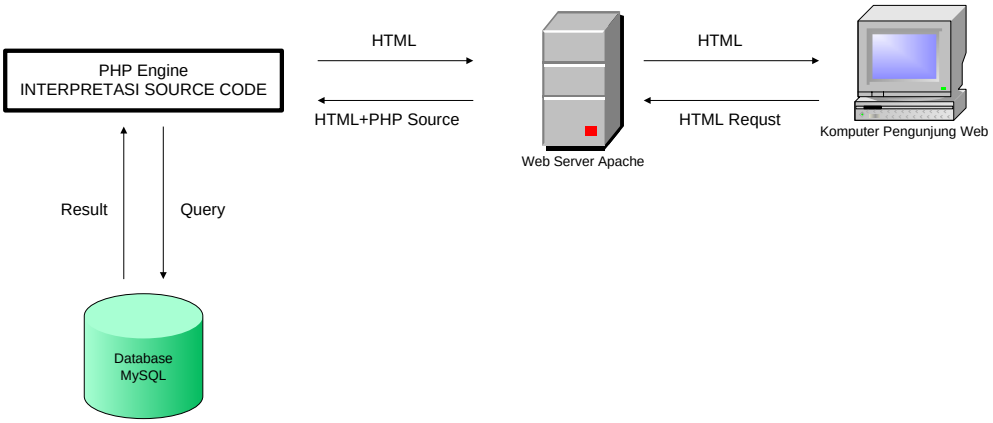
Angka adalah sangat mirip dengan angka di C atau Java, kecuali format oktal dan heksadesimal tidak digunakan.



Gambar II. 6 Angka JSON

2.2 PHP

PHP adalah bahasa skrip dalam *server* (*server-side embedded scripting language*). Artinya, PHP bekerja di dalam HTML dengan tugas membuat isi dokumen sesuai permintaan. Alur kerja PHP itu sendiri dapat kita lihat pada gambar II.7.



Gambar II. 7 Diagram Alur Kerja PHP.

Pada gambar di atas tampak alur kerja *engine* PHP. Pada awalnya, sebuah komputer pengunjung web melakukan HTTP *request* terhadap halaman tertentu. Jika halaman yang di-*request* tersebut adalah halaman PHP, maka *Web Server* Apache akan meneruskan halaman PHP tersebut ke PHP *engine*. PHP *engine* akan melakukan interpretasi terhadap *source* PHP dalam halaman tersebut, dengan bantuan dari *database* jika perlu. Setelah interpretasi selesai, maka PHP akan mengembalikan hasilnya yang berupa HTML murni (tanpa adanya *source* PHP sama sekali) kepada *Web Server Apache*. Setelah itu *Web Server Apache* akan meneruskannya kepada komputer pengunjung web tadi dalam bentuk HTML yang dapat ditampilkan oleh internet browser.

PHP tidak hanya didesain untuk *Web Server Apache* dan *database* MySQL. Bahkan PHP tidak hanya bisa dipakai untuk halaman web. PHP dapat berjalan pada hampir semua *web server* yang mendukung konsep CGI-binary atau ISAPI module, di antaranya *Personal Web Server* (PWS) 3 ke atas, *Internet Information Service* (IIS) 3 ke atas, Apache, OmniHTTPd 2 ke atas, Oreilly Website Pro, Xitami, dan Netscape Enterprise Server.

Koneksi *database* pada PHP amat mudah, cukup dengan menggunakan 2 atau 3 baris perintah saja. Fungsi-fungsi untuk *database* cukup lengkap, dan *database* yang didukung amat banyak, di antaranya MySQL, PostgreSQL, dBase, DB++, FrontBase, ODBC, Sybase-CT, Oracle, Informix, dan Interbase.

PHP memiliki empat kelebihan utama yang menarik minat banyak pengguna. Kelebihan utama PHP tersebut diringkas dalam 4P berikut:

1. *Practicality*. PHP dibuat dengan menitikberatkan pada kepraktisan. Hasilnya, PHP adalah bahasa pemrograman minimalis, dilihat dari segi kebutuhan pengguna dan kebutuhan sintaks.
2. *Power*. PHP memiliki banyak kemampuan, mulai dari kemampuan untuk terhubung dengan basis data, membuat halaman *web* dinamis, membuat dan memanipulasi berkas gambar, *Flash* dan *PDF*, berkomunikasi dengan bermacam protokol seperti *IMAP* dan *POP3*, dan masih banyak lagi.
3. *Possibility*. PHP dapat menyediakan lebih dari satu solusi untuk suatu masalah
4. *Price*. PHP selalu dirilis kepada publik tanpa ada batasan penggunaan, modifikasi, atau redistribusi.

2.3 Android

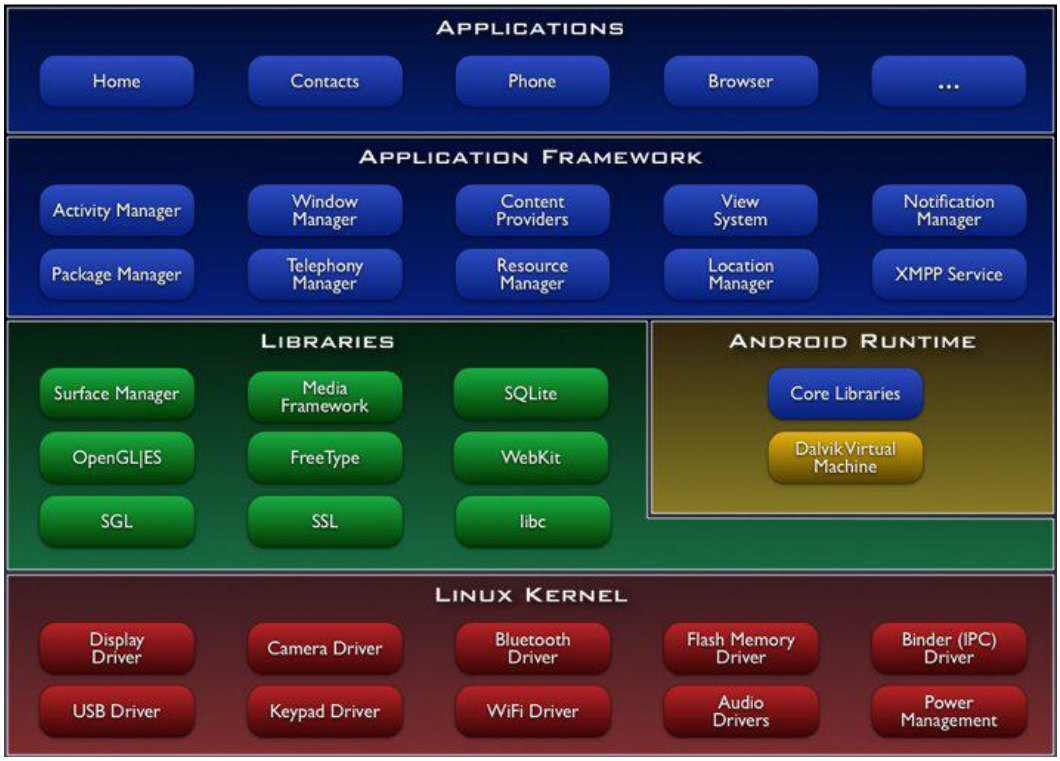
Android merupakan subset perangkat lunak untuk perangkat *mobile* yang meliputi sistem operasi, *middleware* dan aplikasi inti yang di *release* oleh GOOGLE [4]. Aplikasi pada platform ini dikembangkan dalam bahasa pemrograman java. Android sendiri memiliki banyak fitur diantaranya adalah:

- Merupakan sebuah **A*pplication Framework*** sehingga *programmer* dapat menggunakan beberapa fungsi yang telah disediakan.
- **Dalvik *virtual machine***. Tiap aplikasi dalam Android memiliki *instance virtual machine* yang dapat bekerja secara efisien dalam lingkungan memori yang terbatas.
- ***Integrated browser*.** Web browser berbasis WebKit *engine* terdapat pada browser *default* Android atau pun dapat diintegrasikan dengan aplikasi lain.

- ***Optimized graphics***. *Library* grafis 2D yang kaya dan 3D berbasis OpenGL ES 1.0 yang mendukung akselerasi hardware.
- **SQLite**. Basis data relasional yang ringan namun sangat *powerful*.
- ***Media Support***. Mendukung berbagai format audio, video, dan gambar (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF)
- **GSM *Telephony***. Mendukung fungsi komunikasi GSM.
- **Bluetooth, EDGE, 3G** dan **WiFi**. Mendukung komunikasi pada jaringan (tergantung hardware).
- **Kamera**, **GPS**, **kompas** dan ***accelerometer***. Mendukung berbagai fitur yang disediakan oleh *hardware*.
- ***Tools Developer*** yang lengkap. Termasuk *device emulator, tools* untuk *debugging, profiling* memori dan performa, *plugin* untuk Eclipse IDE.

2.3.1 Arsitektur Android

Android terdiri dari beberapa *stack software* yang terdiri dari: *Applications, Application Framework, Libraries, Android Runtime* dan *Kernel Linux.* Arsitektur lengkap platform ini dapat dilihat pada Gambar II.8.



Gambar II. 8 Arsitektur Platform Android [4].

1. ***Aplication***

Application merupakan program yang langsung berhubungan dengan *user*. Baik program yang merupakan bawaan dari Android sendiri maupun program yang dibuat oleh *developer* menggunakan bahasa pemrograman java. Contoh program bawaan dari platform Android sendiri adalah *email client*, program SMS, *calendar*, *maps*, *web browser*, *contact* dan sebagainya.

2. ***Aplication Framework***

Lapisan ini berisi sekumpulan API yang dapat digunakan oleh *programmer* maupun *core application* dari Android. Lapisan ini dirancang untuk memudahkan penggunaan komponen dari Android sendiri. Aplikasi manapun

dalam Android dapat berbagi fungsi sehingga aplikasi lain dapat memanfaatkannya.

3. ***Libraries***

Android mendukung beberapa *library* C/C++ yang digunakan pada berbagai komponen Android. Kemampuan ini dapat diakses oleh *developer* melalui Android *application framework*. Beberapa *library* diantaranya adalah:

- *System* C *library*. Implementasi *library* C standar (libc).
- *Media Libraries*. Mendukung berbagai format multimedia (termasuk MPEG4,H.264, MP3, AAC, AMR, JPG, PNG).
- *Surface Manager*. Mengatur akses ke subsistem *display*.
- LibWebCore. *Engine web browser* modern.
- SGL. *Engine* grafis 2D.
- 3D *Library*. Implementasi OpenGL ES 1.0 yang mendukung akselerasi hardware.
- FreeType. *Rendering* untuk *bitmap* dan *vector font.*
- SQLite. Basis data relasional yang kecil namun sangat ampuh.

4. ***Android Runtime***

Tiap aplikasi pada Android memiliki proses-nya masing-masing. Tiap aplikasi tersebut memiliki *instans* dari Dalvik *virtual machine* (VM). Dalvik *virtual machine* dirancang agar suatu *device* dapat menjalankan beberapa VM secara efisien. Dalvik VM *mengeksekusi* file dengan format Dalvik *Executable format* (.dex) yang dirancang untuk meminimalkan *memory footprint* . Dalvik VM

berbasis *register*, dan dapat menjalankan kelas-kelas yang dikompilasi dengan bahasa pemrograman java dan ditransformasikan menjadi format .dex. Dalvik VM sendiri bergantung pada Kernel Linux untuk fungsi dasarnya, seperti *threading* dan manajemen memori secara *low-level*.

5. **Linux Kernel**

Android menggunakan Kernel Linux versi 2.6 sebagai sistem utama. Fungsi kernel yang digunakan antara lain untuk keamanan, manajemen memori, manajemen proses, manajemen jaringan dan *driver model*. Kernel juga berfungsi sebagai *layer* abstrak antara *hardware* dan lapisan lainnya pada *software stack*.

2.3.2 Siklus Hidup Aplikasi Android

Pada banyak kasus, tiap aplikasi pada Android masing-masing memiliki Linux proses. Proses ini diciptakan untuk aplikasi tersebut pada saat kode program tersebut akan dieksekusi, akan terus berjalan sampai tidak dibutuhkan lagi dan sistem memerlukan memori untuk aplikasi lain. Konsep dasar program Android adalah suatu proses pada aplikasi tidak langsung di atur oleh aplikasi tersebut. Proses tersebut ditentukan oleh sistem melalui kombinasi : informasi aplikasi bagi sistem, seberapa pentingnya aplikasi tersebut bagi *user* dan ketersediaan memori. Contoh kasus *life-cycle bug* adalah *IntentReceiver* yang menciptakan *thread* pada saat menerima *intent* pada *method* onReceiveIntent() dan selesai dari fungsi tersebut. Setelah fungsi tersebut selesai, sistem menganggap bahwa *IntentReceiver* tersebut menjadi tidak aktif sehingga ia memiliki proses yang tidak dibutuhkan (kecuali terdapat komponen aplikasi lain

di dalamnya). Oleh karena itu, sistem dapat menghapus proses tersebut kapanpun apabila dibutuhkan. Hal tersebut akan menghapus semua *thread* yang berjalan pada proses tersebut. Solusi masalah ini adalah memakai *Service* dari IntentReceiver, sehingga sistem mengetahui bahwa ada suatu yang aktif dalam proses tersebut.

Sebagai *programmer,* kita harus mengetahui bagaimana cara yang tepat untuk merancang aplikasi. Kesalahan pada perancangan akan membuat suatu aplikasi akan dihentikan oleh sistem pada saat yang tidak diinginkan. Untuk menentukan proses mana yang akan dihapus pada situasi *low memory*, Android memiliki "*importance hierarchy*" berdasarkan komponen dan *state* yang sedang berjalan. Urutan berdasarkan kepentingan tersebut adalah:

1. ***Foreground process*.** Adalah suatu proses yang dibutuhkan untuk berinteraksi dengan *user* saat ini. Berbagai komponen suatu aplikasi dapat mengakibatkan proses di dalamnya menjadi *foreground proses*. Suatu proses dikatakan *foreground* apabila memenuhi kondisi sebagai berikut:
    - Menjalankan Activity pada *top screen* yang sedang berinteraksi dengan *user* (setelah *method* onResume() dipanggil).
    - Mempunyai IntentReceiver yang sedang berjalan (method IntentReceiver.onReceiveIntent() sedang dieksekusi).
    - Memilik service yang fungsi *callback*-nya (Service.onCreate(), Service.onStart() atau Service.onDestroy()) sedang dieksekusi.

    Hanya akan terdapat beberapa proses seperti itu dalam sistem, proses ini hanya akan dihapus apabila sistem tidak memiliki pilihan lain. Contohnya

apabila memori tinggal sedikit sehingga proses *foreground* tidak bisa lagi berjalan. Hal ini diperlukan untuk menjaga *user interface* tetap interaktif dengan *user*.

2. ***Visible process*.** Adalah proses yang memegang Activity yang dapat dilihat *user* pada layar tetapi tidak *foreground* (pada saat *method* onPause() dipanggil). Hal ini dapat terjadi, sebagai contoh *activity foreground* muncul dengan dialog sehingga *activity* sebelumnya terlihat di belakangnya. Proses ini sangat penting dan tidak akan di hapus kecuali untuk menjaga aplikasi *foreground* tetap berjalan.
3. ***Service process*.** Adalah proses yang memegang Service yang dimulai dengan method StartService(). Meskipun proses ini secara langsung tidak dapat dilihat oleh *user,* biasanya proses ini menjalankan perintah dari *user* (seperti memutar lagu, men-*download* data dari jaringan dan lain-lain). Sistem akan tetap membiarkan proses ini berjalan selama proses *foreground* dan *visible* masih bisa berjalan.
4. ***Background process***. Adalah proses yang memegang Activity yang saat ini tidak dapat dilihat oleh *user* (method onStop() sedang dipanggil). Proses ini tidak secara langsung mempengaruhi *user*. Apabila *activity life cycle* telah dipenuhi dengan benar, sistem dapat menghapus proses ini sewaktu-waktu untuk memberikan memori tambahan. Biasanya banyak ditemukan proses semacam ini, sehingga proses tersebut disimpan dalam LRU (*Least Recently Used*) untuk menjaga agar proses yang baru saja dilihat oleh *user* akan dihapus paling akhir pada kondisi *low memory*.

5. ***Empty Process*.** Adalah proses yang tidak memegang suatu komponen yang sedang aktif. Alasan mengapa membiarkan proses semacam ini adalah untuk melakukan *cache* sehingga proses *startup* menjadi lebih cepat apabila aplikasi tersebut akan dijalankan kembali. Sistem seringkali menghapus proses semacam ini untuk mencapai keseimbangan dari *system resource* antara cached processes dan kernel caches.

2.4 UML

Unified Modeling Language (UML) adalah bahasa spesifikasi standar untuk mendokumentasikan, menspesifikasikan, dan membangun sistem perangkat lunak. Unified Modeling Language (UML) adalah himpunan struktur dan teknik untuk pemodelan desain program berorientasi objek (OOP) serta aplikasinya. UML adalah metodologi untuk mengembangkan sistem OOP dan sekelompok perangkat tool untuk mendukung pengembangan sistem tersebut. UML mulai diperkenalkan oleh Object Management Group, sebuah organisasi yang telah mengembangkan model, teknologi, dan standar OOP sejak tahun 1980-an. Sekarang UML sudah mulai banyak digunakan oleh para praktisi OOP. UML merupakan dasar bagi perangkat (tool) desain berorientasi objek dari IBM.

UML adalah suatu bahasa yang digunakan untuk menentukan, memvisualisasikan, membangun, dan mendokumentasikan suatu sistem informasi. UML dikembangkan sebagai suatu alat untuk analisis dan desain berorientasi objek oleh Grady Booch, Jim Rumbaugh, dan Ivar Jacobson. Namun demikian UML dapat digunakan untuk memahami dan mendokumentasikan setiap sistem informasi. Penggunaan UML dalam industri terus meningkat. Ini merupakan

standar terbuka yang menjadikannya sebagai bahasa pemodelan yang umum dalam industri peranti lunak dan pengembangan sistem.

2.4.1 Sejarah UML

Sampai era tahun 1990 puluhan metodologi pemodelan berorientasi objek telah bermunculan di dunia. Diantaranya adalah: metodologi booch, metodologi coad, metodologi OOSE, metodologi OMT, metodologi shlaer-mellor, metodologi wirfs-brock, dsb. Masa itu terkenal dengan masa perang metodologi (method war) dalam pendesainan berorientasi objek. Masing-masing metodologi membawa notasi sendiri-sendiri, yang mengakibatkan timbul masalah baru apabila kita bekerjasama dengan kelompok/perusahaan lain yang menggunakan metodologi yang berlainan.

Dimulai pada bulan Oktober 1994 Booch, Rumbaugh dan Jacobson, yang merupakan tiga tokoh yang boleh dikata metodologinya banyak digunakan mempelopori usaha untuk penyatuan metodologi pendesainan berorientasi objek. Pada tahun 1995 direlease draft pertama dari UML (versi 0.8) [7]. Sejak tahun 1996 pengembangan tersebut dikoordinasikan oleh Object Management Group (OMG – http://www.omg.org).

2.4.2 Diagram UML

UML menyediakan 10 macam diagram untuk memodelkan aplikasi berorientasi objek, yaitu:

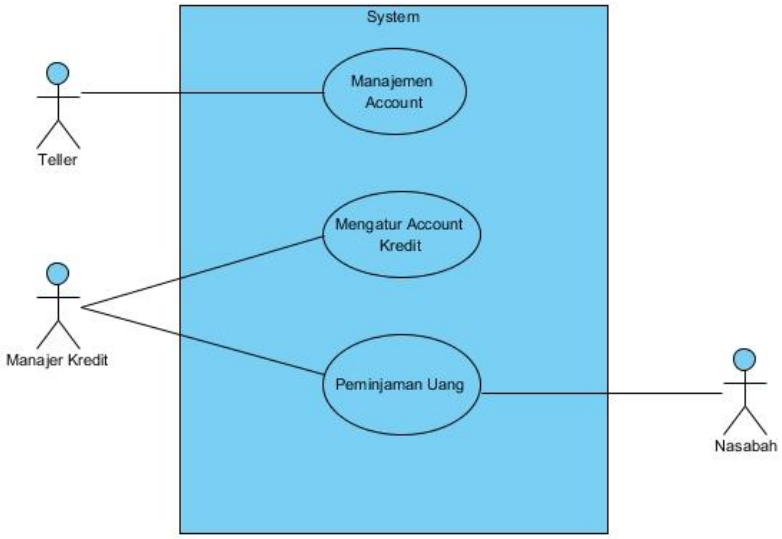
1. Use Case Diagram untuk memodelkan proses bisnis.

2. Conceptual Diagram untuk memodelkan konsep-konsep yang ada di dalam aplikasi.
3. Sequence Diagram untuk memodelkan pengiriman pesan (message) antar objects.
4. Collaboration Diagram untuk memodelkan interaksi antar objects.
5. State Diagram untuk memodelkan perilaku objects di dalam sistem.
6. Activity Diagram untuk memodelkan perilaku Use Cases dan objects di dalam system.
7. Class Diagram untuk memodelkan struktur kelas.
8. Object Diagram untuk memodelkan struktur object.
9. Component Diagram untuk memodelkan komponen object.
10. Deployment Diagram untuk memodelkan distribusi aplikasi.

1. **Use Case Diagram**

Use case diagram menggambarkan fungsionalitas yang diharapkan dari sebuah sistem. Yang ditekankan adalah "apa" yang diperbuat sistem, dan bukan "bagaimana". Sebuah *use case* merepresentasikan sebuah interaksi antara aktor dengan sistem. *Use case* merupakan sebuah pekerjaan tertentu, misalnya login ke sistem, meng-*create* sebuah daftar belanja, dan sebagainya. Seorang/sebuah aktor adalah sebuah entitas manusia atau mesin yang berinteraksi dengan sistem untuk melakukan pekerjaan-pekerjaan tertentu. Sebuah *use case* dapat meng-*include* fungsionalitas *use case* lain sebagai bagian dari proses dalam dirinya. Secara umum diasumsikan bahwa *use case* yang di-*include* akan dipanggil setiap kali *use case* yang meng-*include* dieksekusi secara normal. Sebuah *use case* dapat di-

include oleh lebih dari satu *use case* lain, sehingga duplikasi fungsionalitas dapat dihindari dengan cara menarik keluar fungsionalitas yang *common*. Sebuah *use case* juga dapat meng-*extend use case* lain dengan *behaviour*-nya sendiri. Sementara hubungan generalisasi antar *use case* menunjukkan bahwa *use case* yang satu merupakan spesialisasi dari yang lain. Penggambaran Use case diagram dapat di lihat dalam gambar II.9.



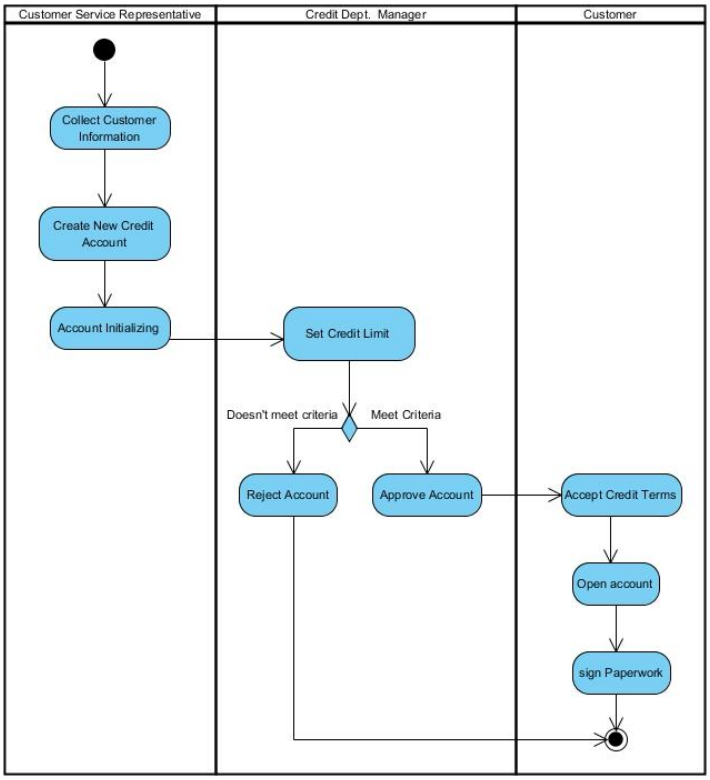
Gambar II. 9 Contoh Use Case Diagram.

2. **Activity Diagram**

Activity diagrams menggambarkan berbagai alir aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alir berawal, *decision* yang mungkin terjadi, dan bagaimana mereka berakhir. *Activity diagram* juga dapat menggambarkan proses paralel yang mungkin terjadi pada beberapa eksekusi. *Activity diagram* merupakan *state diagram* khusus, di mana sebagian besar *state* adalah *action* dan sebagian besar transisi di-*trigger* oleh selesainya *state* sebelumnya (*internal processing*). Oleh karena itu *activity diagram* tidak

menggambarkan behaviour internal sebuah sistem (dan interaksi antar subsistem) secara eksak, tetapi lebih menggambarkan proses-proses dan jalur-jalur aktivitas dari level atas secara umum.

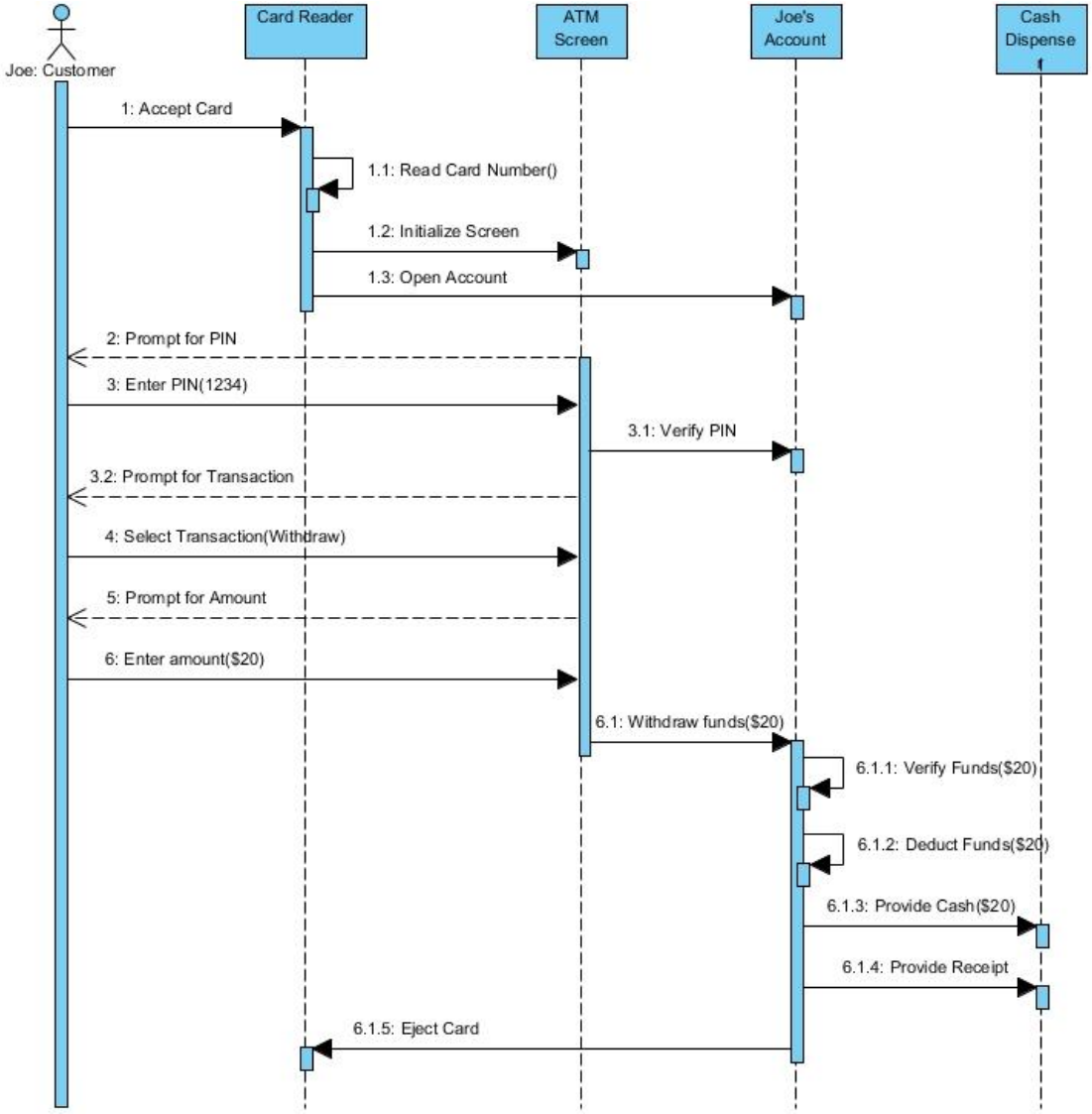
Sebuah aktivitas dapat direalisasikan oleh satu *use case* atau lebih. Aktivitas menggambarkan proses yang berjalan, sementara *use case* menggambarkan bagaimana aktor menggunakan sistem untuk melakukan aktivitas. *Decision* digunakan untuk menggambarkan behaviour pada kondisi tertentu. Untuk mengilustrasikan proses-proses paralel (*fork* dan *join*) digunakan titik sinkronisasi yang dapat berupa titik, garis horizontal atau vertikal. *Activity diagram* dapat dibagi menjadi beberapa *object swimlane* untuk menggambarkan objek mana yang bertanggung jawab untuk aktivitas tertentu. Contoh Activity diagram dapat di lihat pada gambar II.10.



Gambar II. 10 Contoh Activity Diagram.

3. **Sequence Diagram**

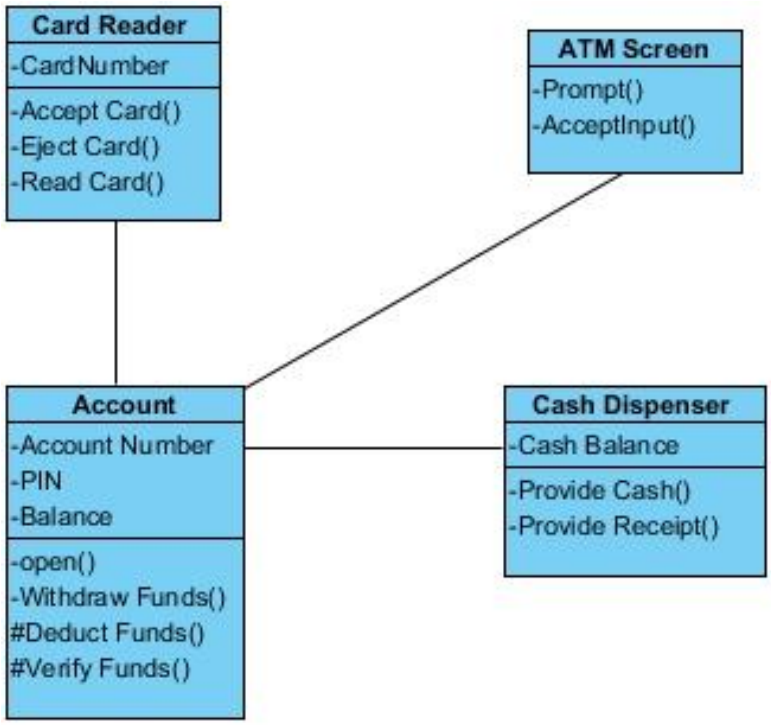
Sequence diagram menjelaskan secara detil urutan proses yang dilakukan dalam sistem untuk mencapai tujuan dari use case: interaksi yang terjadi antar class, operasi apa saja yang terlibat, urutan antar operasi, dan informasi yang diperlukan oleh masing-masing operasi. Contoh sequence diagram dapat di lihat pada gambar II.11.



Gambar II. 11 Contoh Sequence Diagram.

4. **Class Diagram**

Class diagram merupakan diagram yang selalu ada di permodelan sistem berorientasi objek. Class diagram menunjukkan hubungan antar class dalam sistem yang sedang dibangun dan bagaimana mereka saling berkolaborasi untuk mencapai suatu tujuan. Penggambaran class diagram dapat di lihat dalam gambar II.12.



Gambar II. 12 Contoh Class Diagram.

2.5 MySQL

MySQL merupakan sebuah *software* yang berguna sebagai suatu *database server* yang cukup terkenal. *Database server* itu sendiri merupakan suatu *software* yang bertugas untuk melayani permintaan (*request*) *query* dari *client*.

MySQL sebagai suatu *database server* mempunyai beberapa kemampuan, salah satunya harus menyediakan suatu sistem manajemen database yang dapat mengatur bagaimana menyimpan, menambah, mengakses data dan transaksi-transaksi database lainnya. MySQL cepat sekali berkembang, karena MySQL merupakan suatu *software* yang *Open Source*.

MySQL adalah *Relational Database Management System* (RDBMS) yang didistribusikan secara gratis dibawah lisensi GPL (*General Public License*). Dimana setiap orang bebas untuk menggunakan MySQL, namun tidak boleh dijadikan produk turunan yang bersifat komersial. MySQL sebenarnya merupakan turunan salah satu konsep utama dalam *database* sejak lama, yaitu SQL (*Structured Query Language*) [2]. SQL adalah sebuah konsep pengoperasian *database*, terutama untuk pemilihan atau seleksi dan pemasukan data, yang memungkinkan pengoperasian data dikerjakan dengan mudah secara otomatis.

Keandalan suatu sistem database (DBMS) dapat diketahui dari cara kerja optimizer-nya dalam melakukan proses perintah-perintah SQL, yang dibuat oleh *user* maupun program-program aplikasinya. Sebagai database *server*, MySQL dapat dikatakan lebih unggul dibandingkan *database server* lainnya dalam *query* data. Hal ini terbukti untuk *query* yang dilakukan oleh *single user*, kecepatan *query* MySQL bisa sepuluh kali lebih cepat dari PostgreSQL dan lima kali lebih cepat dibandingkan Interbase.

MySQL memiliki beberapa keistimewaan, antara lain :

1. *Portabilitas*. MySQL dapat berjalan stabil pada berbagai sistem operasi seperti *Windows, Linux, FreeBSD, Mac Os X Server, Solaris, Amiga*, dan masih banyak lagi.
2. *Open Source.* MySQL didistribusikan secara *open source*, dibawah lisensi GPL sehingga dapat digunakan secara cuma-cuma.
3. *Multiuse*. MySQL dapat digunakan oleh beberapa *user* dalam waktu yang bersamaan tanpa mengalami masalah atau konflik.
4. *Performance tuning*. MySQL memiliki kecepatan yang menakjubkan dalam menangani *query* sederhana, dengan kata lain dapat memproses lebih banyak SQL per satuan waktu.
5. Jenis Kolom. MySQL memiliki tipe kolom yang sangat kompleks, seperti *signed/unsigned integer, float, double, char, text, date, timestamp*, dan lain-lain.
6. Perintah dan Fungsi. MySQL memiliki operator dan fungsi secara penuh yang mendukung perintah *Select* dan *Where* dalam perintah (*query*).
7. Keamanan. MySQL memiliki beberapa lapisan sekuritas seperti *level subnetmask*, nama host, dan izin akses *user* dengan sistem perizinan yang mendetail serta sandi terenkripsi.
8. Skalabilitas dan Pembatasan. MySQL mampu menangani basis data dalam skala besar, dengan jumlah rekaman (*records)* lebih dari 50 juta dan 60 ribu tabel serta 5 milyar baris. Selain itu batas indeks yang dapat ditampung mencapai 32 indeks pada tiap tabelnya.

9. Konektivitas. MySQL dapat melakukan koneksi dengan klien menggunakan protocol TCP/IP, Unix socket (UNIX), atau Named Pipes.
10. Lokalisasi. MySQL dapat mendeteksi pesan kesalahan pada *client* dengan menggunakan lebih dari dua puluh bahasa. Meskipun demikian, bahasa Indonesia belum termasuk di dalamnya.
11. Antar Muka. MySQL memiliki *interface* (antar muka) terhadap berbagai aplikasi dan bahasa pemrograman dengan menggunakan fungsi API (*Application Programming Interface*).
12. Klien dan Peralatan. MySQL dilengkapi dengan berbagai peralatan (*tool*) yang dapat digunakan untuk administrasi basisdata, dan pada setiap peralatan yang ada disertakan petunjuk *online*.
13. Struktur tabel. MySQL memiliki struktur tabel yang lebih fleksibel dalam menangani *ALTER TABLE*, dibandingkan basisdata lainnya semacam PostgreSQL.