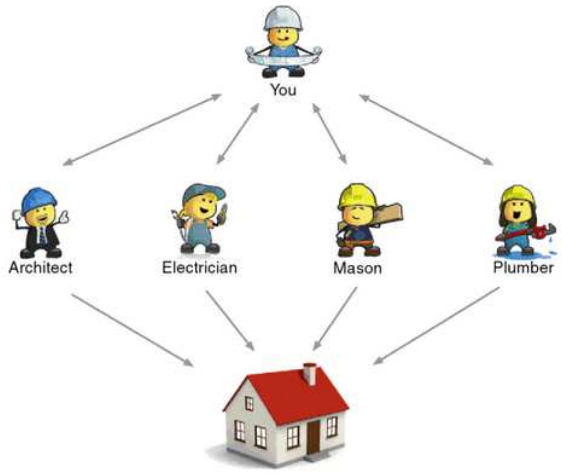
BAB 2

TINJAUAN PUSTAKA

2.1 API (*Application Programming Interface*)

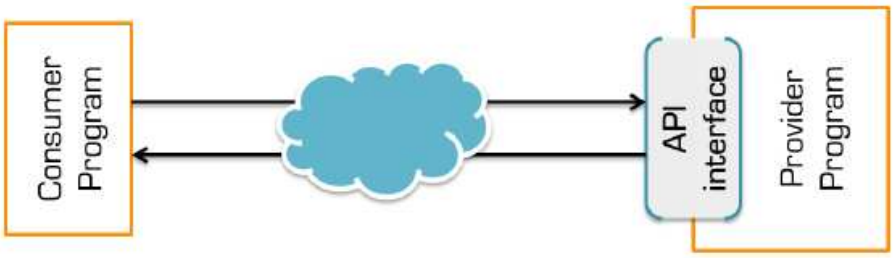
API merupakan *software interface* yang terdiri atas kumpulan instruksi yang disimpan dalam bentuk *library* dan menjelaskan bagaimana agar suatu *software* dapat berinteraksi dengan *software* lain. Penjelasan ini dapat dicontohkan dengan analogi apabila akan dibangun suatu rumah. Dengan menyewa kontraktor yang dapat menangani bagian yang berbeda, pemilik rumah dapat memberikan tugas yang perlu dilakukan oleh kontraktor tanpa harus mengetahui bagaimana cara kontraktor menyelesaikan pekerjaan tersebut. Dari analogi tersebut, rumah merupakan *software* yang akan dibuat, dan kontraktor merupakan API yang mengerjakan bagian tertentu dari *software* tersebut tanpa harus diketahui bagaimana prosedur dalam melakukan pekerjaan tersebut.



Gambar 2.1 Analogi API pada Pembangunan Rumah

(Sumber: *API Design for C*, Reddy, 2011)

Interface pada *software* merupakan suatu *entry points* yang digunakan untuk mengakses seluruh *resources* yang terdapat di dalam *software* tersebut. Dengan adanya API, maka terdapat aturan bagaimana *software* dapat berinteraksi dengan *software* lain untuk mengakses *resources* melalui *interface* yang telah tersedia.



Gambar 2.2 Skema Konektivitas API Antar *Software*

(Sumber: *What is an API?*, 3Scale Networks, 2011)

Secara struktural, API merupakan spesifikasi dari suatu *data structure*, *objects*, *functions*, beserta parameter-parameter yang diperlukan untuk mengakses *resource* dari aplikasi tersebut. Seluruh spesifikasi tersebut membentuk suatu *interface* yang dimiliki oleh aplikasi untuk berkomunikasi dengan aplikasi lain, dan API dapat digunakan dengan berbagai bahasa *programming*, ataupun hanya dengan menggunakan URL (*Uniform Resource Locator*) yang telah disediakan oleh suatu *website*.

API dapat diklasifikasikan menjadi beberapa kategori, hal ini dilihat dari abstraksi apa yang dideskripsikan di dalam sistem. Kategori-kategori ini diantaranya:

Tabel 2.1 Kategori API

Kategori API	Deskripsi	Contoh
Operating System	API yang digunakan untuk fungsi dasar yang dapat dilakukan oleh komputer. Seperti proses I/O, eksekusi program.	API for MS Windows
Programming Languages	API yang digunakan untuk memperluas kapabilitas dalam melakukan eksekusi terhadap suatu bahasa pemrograman.	Java API
Application Services	API yang digunakan untuk mengakses data dan layanan yang disediakan dari suatu aplikasi.	API for mySAP (BAPI/*Business Application Programming Interface*)
Infrastructure Services	Digunakan untuk mengakses infrastruktur dari suatu komputer. Infrastruktur disini adalah komputer beserta *peripheral* seperti *storage*, aplikasi, dan lain-lain.	Amazon EC2 (*Elastic Compute Cloud*) untuk akses untuk *virtual computing* dan Amazon S3 (*Simple Storage Service*) untuk menyimpan data dalam jumlah besar.
Web Services	API yang digunakan untuk mengakses *content* dan layanan yang disediakan oleh suatu *web application*.	*Facebook Graph* APIyang digunakan untuk mengakses informasi yang dapat dibagikan.

2.2 REST (*Representational State Transfer*)

REST (*Representational State Transfer*) merupakan jenis arsitektur yang terdapat pada *web* untuk melayani suatu *service*. REST merepresentasikan cara interaksi antara *server* dan *client* untuk melakukan proses pertukaran informasi melalui media yang sama.

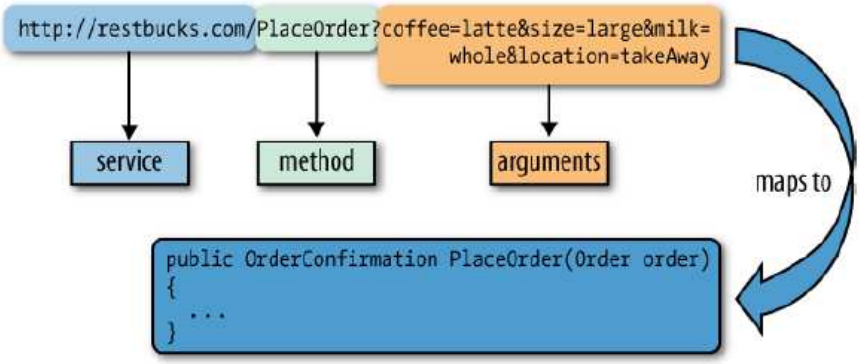
Dalam suatu jaringan. agar suatu *resource* dapat diakses, maka diperlukan identifikasi dan suatu bentuk manipulasi. Dapat digunakan URI (*Uniform Resource Identifier*) yang digunakan untuk mengidentifikasi *resource* yang ada pada suatu jaringan, dan dapat membuat *resource* menjadi *addressable*, yang berarti *resource* dapat diketahui lokasinya dan dapat dimanipulasi dengan menggunakan suatu aplikasi (Webber, Parastatidis, & Robinson, 2010: 5).

REST dapat digunakan sebagai *interface* dari API untuk mengakses suatu *resource*. API yang mengikuti prinsip dari REST *architecture* memberikan kemudahan bagi *developer* untuk tidak perlu mengetahui bagaimana struktur dari API di dalam *server*. Dalam hal ini, *server* akan memberikan informasi bagaimana agar *client* dapat mengakses *service* melalui API yang telah disediakan.

Penggunaan protokol HTTP pada REST *architecture* untuk komunikasi antara *client* dan *server* terletak pada HTTP *method*, yaitu GET, POST, PUT, dan DELETE. *Method* ini dapat digunakan untuk mengakses *resources* yang ada pada *server*, bergantung dari instruksi yang diberikan oleh *server*.

Dengan menggunakan protokol HTTP, URI dapat dijadikan sebagai media yang digunakan untuk mengakses *resource* dari *server*. Hal ini disebut dengan URI *tunneling*.

URI *tunneling* mempergunakan URI untuk mentransfer informasi pada antar sistem yang dalam jaringan dengan melakukan *encode* pada URI itu sendiri. Dengan mengirim HTTP *method* yang telah disebutkan sebelumnya, *server* dapat melakukan eksekusi terhadap suatu program yang menghasilkan atau mengambil suatu *resource* dan mengirimkannya kembali ke *client*. Dalam proses ini terjadi proses *mapping* dari URI menjadi *method call* pada *server* yang dituju (Webber, Parastatidis, & Robinson, 2010: 37).



Gambar 2.3 *Mapping method calls* ke URI

(Sumber: *REST In Practice: Hypermedia and Systems Architecture*, Weber, 2010)

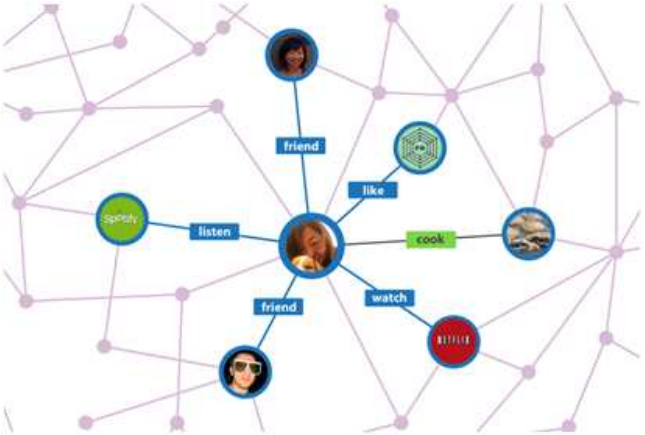
2.3 Facebook Graph API

Facebook memiliki fitur untuk menggabungkan aplikasi dengan fungsi jejaring sosial. Untuk menghasilkan aplikasi yang dapat digunakan di Facebook, Facebook menyediakan *platform* yang dapat digunakan oleh para pengembang. *Platform* yang disediakan adalah kumpulan komponen berupa *software environment*. *Software environment* inidapat digunakan untuk menghasilkan aplikasi yang dapat berjalan pada *website* Facebook, dan untuk mengakses data yang terdapat di dalam Facebook. Selain itu, *Software environment* ini dapat digunakan oleh pihak ketiga, yaitu individu atau kelompok yang mengembangkan aplikasi yang tidak berasal dari pembuat atau pemilik sistem.

Berdasarkan *website* dari Facebook (www.developers.facebook.com), Aplikasi di Facebook terbagi atas dua jenis:

1. *Facebook for Websites: website* yang memanfaatkan API dari Facebook untuk *like* ataupun *share button*.
2. *Canvas application:* aplikasi yang berjalan di dalam Facebook, dan memiliki akses untuk mendapatkan informasi dari objek yang ada pada Facebook.

Salah satu komponen yang disediakan oleh Facebook adalah Graph API. Graph API merupakan komponen utama yang digunakan untuk meminta ataupun memasukkan data ke dalam *social graph* yang ada pada Facebook. *Social graph* merupakan representasi yang menunjukkan informasi yang dimiliki dari suatu entitas beserta relasi dengan entitas lain.



Gambar 2.4 Ilustrasi *Social Graph*

(Sumber: *businessinsider.com*)

Social graph merupakan representasi dari struktur data *graph*. *Graph* terdiri atas *nodes* (titik) dan *links* (tali), dimana *links* berperan untuk menunjukkan hubungan antar *nodes*. Data pada *social graph* yang ada pada Facebook dapat diakses dengan dua cara, yaitu melalui:

1. Graph API: merupakan metode utama yang digunakan untuk melakukan penambahan, penghapusan, ataupun pengambilan data dari *social graph* dengan menggunakan *request* berbasis HTTP ke *server* Facebook.
2. FQL (Facebook Query Language): merupakan metodeyang dapat digunakan untuk mengakses objek *social graph* melalui SQL-*based interface*. Objek dapat diakses dengan menggunakan *query* layaknya pada SQL.

Pada Graph API, objek merepresentasikan *nodes* dan *links* menunjukkan hubungan antar objek (Srivstava & Singh, 2011). Objek yang terdapat di Graph API merupakan kumpulan entitas yang terdapat di dalam Facebook, contohnya seperti individu, foto, video,dan lain-lain. Tiap objek tersebut dapat memiliki suatu *links* yang menyatakan suatu hubungan, yaitu seperti hubungan kekerabatan antar individu, ataupun *tagged photos* yang diberikan pada beberapa individu. Berikut adalah beberapa atribut yang terdapat pada tipe objek "`User`", yaitu pengguna individu pada Facebook:

Tabel 2.2 Atribut pada Tipe Objek *User*

Atribut	Deskripsi
`name`	merepresentasikan nama lengkap yang dimiliki oleh pengguna.
`gender`	menunjukkan jenis kelamin yang dimiliki oleh pengguna.
`education`	menunjukkan semua pendidikan yang ditulis di dalam profil pengguna.
`location`	menunjukkan dimana lokasi tempat tinggal dari pengguna.
`website`	menunjukkan website yang dimiliki oleh pengguna.
`work`	menunjukkan tempat kerja dari pengguna.

Dengan adanya Graph API, maka dapat dilakukan tindakan seperti memasukkan data, mengambil data, ataupun mengubah data yang terdapat di dalam objek pada Graph API (Srivstava & Singh, 2011). Tindakan ini berdampak pada isi pada objek yang ditunjuk, dan relasi yang dimiliki oleh antar objek.

Graph API bekerja dengan memanfaatkan *low-level protocol* HTTP (*Hypertext Transfer Protocol*) untuk melakukan pertukaran data antara *server* pada Facebook dengan aplikasi yang digunakan. Tindakan yang diambil akan di-*mapping* secara langsung ke dalam HTTP *request. Mapping* merupakan proses penyesuaian relasi antar suatu data. Sebagai contohnya, untuk melakukan pengambilan data maka akan digunakan HTTP *method* GET, penulisan atau modifikasi data dengan POST, dan DELETE untuk menghapus data pada *node*. Metode ini mengadopsi dari REST *architecture style*, dimana digunakan URI untuk melakukan *request* terhadap *service* dari suatu *server*. Berikut merupakan contoh HTTP *request* beserta *response* yang diberikan oleh Graph API:

```
GET /774635482?fields=id%2Cname HTTP/1.1
Host: graph.facebook.com
Connection: close

HTTP/1.1 200 OK
Content-Type: text/javascript; charset=UTF-8
Expires: Sat, 01 Jan 2000 00:00:00 GMT
Date: Thu, 11 Oct 2012 17:28:02 GMT
Connection: close
Content-Length: 48

{"id":"774635482","name":"Christopher Blizzard"}
```

Gambar 2.5 HTTP *Request* &*Response* Graph API

2.3.1 *Authentication & Authorization* pada Facebook Graph API

Sebelum aplikasi dapat memanfaatkan Graph API, diperlukan proses otentikasi dan otorisasi. Otorisasi merupakan proses untuk melakukan identifikasi dan perizinan untuk memanipulasi sumber daya tertentu, sedangkan otentikasi merupakan proses untuk melakukan verifikasi atau memastikan bahwa suatu pihak atau aplikasi berhak untuk mempergunakan sumber daya tersebut.

Agar aplikasi yang ada pada Facebook dapat memanfaatkan Graph API, aplikasi harus mendapatkan bukti bahwa ia dapat melakukan pemanggilan ke Graph API, dan aplikasi tersebut telah mendapatkan perizinan dari pengguna untuk mengakses data yang dibutuhkan oleh aplikasi. Untuk memenuhi dua kondisi tersebut, Facebook menyediakan mekanisme Facebook Login, yaitu komponen dari Facebook *platform* yang melakukan otentikasi dan otorisasi pada pengguna aplikasi. Facebook Login memanfaatkan protokol OAuth 2.0 untuk melaksanakan proses tersebut. Berdasarkan RFC (*Request for Comment*) 6749 dari IETF (*Internet Engineering Task Force*) (http://tools.ietf.org/html/rfc6749), OAuth merupakan *open standard* untuk melakukan otorisasi, maksud dari *open standard* sendiri adalah spesifikasi yang dipublikasikan dan dapat digunakan secara bebas tanpa adanya biaya tertentu.

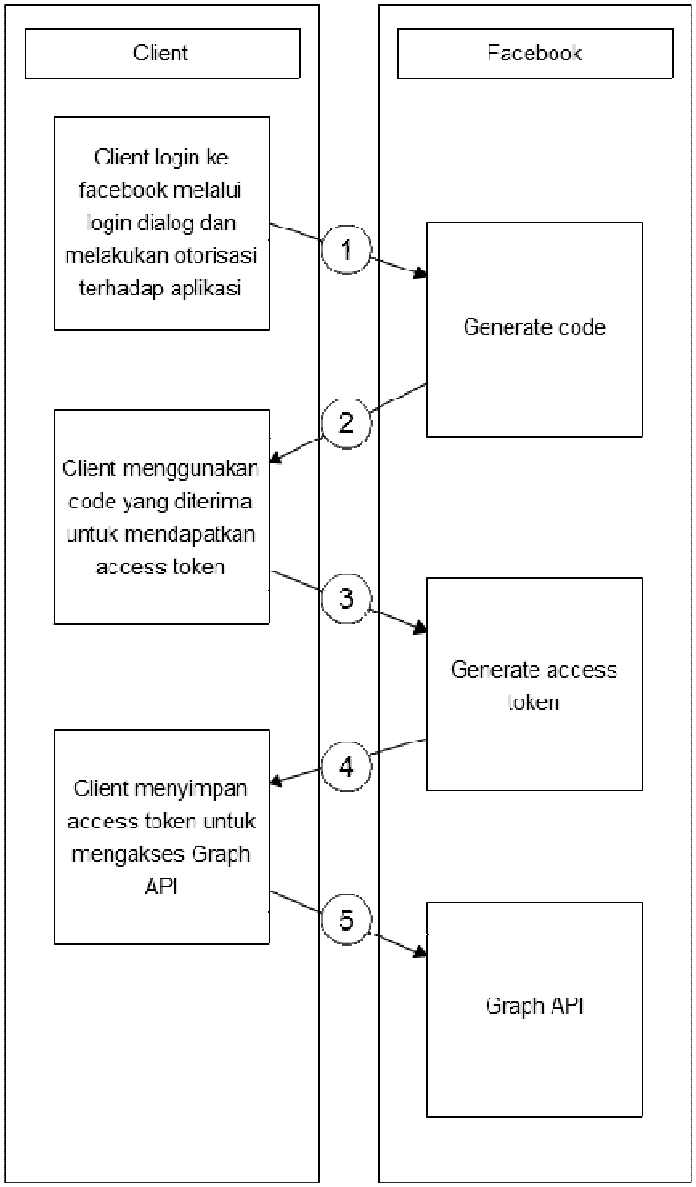
Pada proses otentikasi dan otorisasi, fitur yang digunakan oleh Facebook untuk memastikan aplikasi dapat digunakan, antara lain:

1. *Access token:* merupakan tanda yang digunakan sebagai bukti otentikasi bahwa aplikasi dapat melakukan pemanggilan terhadap API yang

ada pada Facebook. Pemanggilan API dilakukan dengan tujuan mengambil, mengubah, menambahkan, atau menghapus data yang ada pada objek di Graph API. *Access token* yang digunakan berupa *opaque string*, yaitu barisan karakter yang bersifat acak dan tidak dapat dibaca (Hardt, 2012).

2. *Permissions: permissions* merupakan tanda perizinan dari pengguna aplikasi bahwa aplikasi dapat mengakses data yang dimilikinya. Dengan adanya *permission* dari pengguna, aplikasi memiliki otorisasi untuk menggunakan data yang telah diberikan oleh pengguna. Aplikasi dapat mengajukan apa saja jenis *permission* yang diminta dari pengguna. Secara otomatis, informasi umum seperti nama, *profile picture*, umur, *gender*, bahasa, dan negara asal telah menjadi bagian dari *permission.*

Access token merupakan bukti utama yang digunakan untuk mengakses data dari pengguna, untuk memastikan bahwa *access token* dapat diterima oleh aplikasi yang bersangkutan, digunakan *shared secret* berupa *application ID* dan *authorization code* (Srivstava & Singh, 2011).



Gambar 2.6 Proses Otentikasi dan Otorisasi

Proses otentikasi dimulai saat pengguna mengakses aplikasi yang terdapat pada Facebook:

1. Pengguna akan di-*redirect* ke *server* Graph API (`graph.facebook.com`) untuk melakukan otorisasi. *Redirect URL* yang digunakan adalah:

   ```
   https://graph.facebook.com/oauth/authorize?client_id=<app_id>&redirect_uri=<redirect_url>&scope=user_status
   ```

Dilihat dari URL ini, terdapat beberapa *parameter* yang perlu untuk diberikan ke Graph API, diantaranya:

i. `client_id`:

merupakan kunci identifikasi dari aplikasi yang akan digunakan oleh pengguna. Setiap aplikasi memiliki identifikasi yang berbeda-beda.

ii. `redirect_uri`:

menyatakan kemana data yang diberikan oleh Graph API akan dikirim. URI (Uniform Resource Identifier) berupa alamat *server* dari aplikasi yang digunakan.

iii. `scope`:

menentukan apa saja data pengguna yang akan diakses oleh aplikasi. Informasi umum seperti yang telah dijelaskan sebelumnya secara otomatis akan diminta.

Facebook akan mengecek apakah aplikasi tersebut telah memiliki *access token* untuk mengakses data yang dimiliki oleh pengguna, hal ini dilakukan dengan cara mengecek pada parameter `redirect_uri`.

Apabila *access token* tidak dapat ditemukan, Graph API akan melakukan *redirect* ke halaman yang berisi permintaan izin (*permission*) oleh aplikasi agar data pengguna dapat diakses.

2. Facebook akan melakukan *redirect* ke alamat URL yang ada pada redirect_uri, yaitu berupa *web server* yang menyimpan aplikasi. Akan dikirim variabel `code` sementara yang nantinya digunakan untuk mendapatkan *access token*.

3. Aplikasi akan mendapatkan *access token* dengan cara mengakses *URL* berikut:

```
https://graph.facebook.com/oauth/access_token?client_id=<app_id>&redirect_uri=<redirect_url>&client_secret=<app_secret>&code=<code>
```

Di tahap ini, terdapat dua parameter yang dikirim, yaitu:

i. `client_secret`:
   berisi kode rahasia yang dimiliki oleh aplikasi yang akan digunakan.

ii. `code`:
   merupakan *authorization code* yang didapat pada tahap sebelumnya.

4. Pada tahap ini, *access token* telah diterima oleh *client*. *Access token* akan disimpan baik dalam *server* dari aplikasi, ataupun dengan metode-metode lain.
5. Jika *access token* sudah diterima, apabila *client* akan melakukan penulisan ataupun pengambilan data, *access token* akan dikirim agar Graph API dapat diakses.

2.3.2 Proses Graph API *Call*

Menurut dokumentasi yang diberikan oleh Facebook (www.developers.facebook.com), proses pengambilan ataupun penulisan data melewati Graph API memiliki prosedur berikut:

1. Pengambilan dan penulisan data menggunakan arsitektur RESTful, dalam artiannya data yang dibutuhkan dapat diakses melalui parameter yang terdapat pada URL dan telah dilakukan *mapping* secara otomatis.

2. URL yang digunakan memiliki bentuk berikut:

   ```
   http://graph.facebook.com/[user_id]/[object],{fields}
   ```

   Disini `object` merupakan *nodes* yang telah dijelaskan sebelumnya, dan tiap `object` memiliki `fields`, yaitu

kumpulan atribut yang menjelaskan karakteristik dari objek tersebut.

Dalam contoh ini, digunakan objek `statuses,` yaitu objek berupa *status update* yang dihasilkan oleh suatu *user*. Objek `statuses` memiliki atribut berikut:

Tabel 2.3 Atribut pada Objek Statuses

Nama atribut	Deskripsi	Permissions	Returns
`id`	ID dari *status update*	Membutuhkan *access token*	`string`
`from`	*User* yang mengirim status update	Membutuhkan *access token*	objek berisi `id` dan `name`
`message`	Isi dari *status update*	Membutuhkan *access token*	`string`
`place`	Tempat dimana *status update* tersebut berasal	Membutuhkan *access token*	objek berisi `id` dan `name`, serta `latitude`
`updated_time`	Waktu *status update* dikirim	Membutuhkan *access token*	`string`
`type`	Nama jenis objek dari *status update*	Membutuhkan *access token*	`string`

3. Sebagai contoh apabila dibutuhkan *status update* dari *user* pada Facebook dengan ID `“foobar”`dan akan dilihat kapan *status update* dibuat, maka URL yang dihasilkan adalah:

   ```
   http://graph.facebook.com/foobar/statuses,{updated_time}
   ```

4. Aplikasi akan membuka URL tersebut, dan Facebook akan mengecek apakah aplikasi telah memiliki *access token*. Jika ya, maka Facebook akan mengirim informasi *status update* yang telah dibuat oleh *user*, dalam bentuk JSON (JavaScript Object Notation).

2.4 Raspberry Pi Model B

Raspberry Pi merupakan *single-board computer* dengan ukuran kartu kredit yang dikembangkan di UK oleh Raspberry Pi Foundation dengan tujuan untuk mendorong ilmu komputer di berbagai sekolah (Richardson & Wallace, 2012).

Gambar 2.7 Raspberry Pi Model B

(Sumber: www.raspberrypi.org)

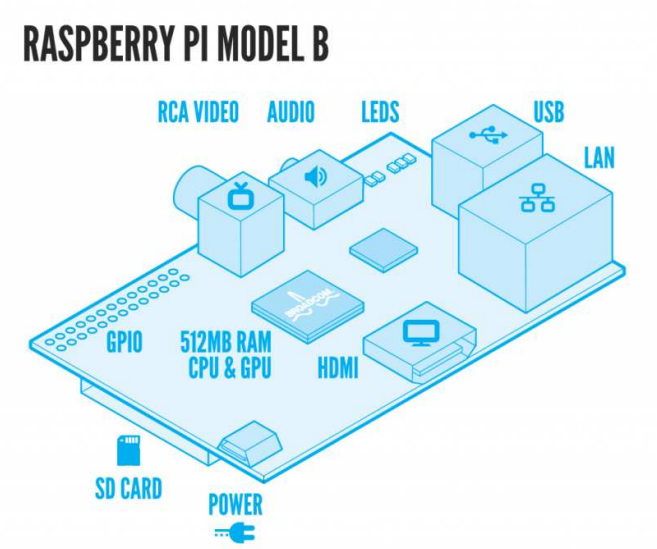
Raspberry Pi memiliki *system on a chip* (SoC) Broadcom BCM2835. SoC merupakan sebuah IC yang mengintegrasi semua komponen dari sebuah komputer seperti CPU, GPU, RAM menjadi satu IC.

SoC Broadcom BCM2835 ini mempunyai prosesor ARM1176JZF-S dengan kecepatan 700 MHz yang dapat ditingkatkan (*overclock)* menjadi 1 GHz, SoC ini juga dilengkapi dengan VideoCore IV GPU dan RAM sebanyak 512 MB untuk model B dan 256 MB untuk model A.

Selain itu, Raspberry Pi ini tidak mempunyai *internal storage* sebagai media penyimpanan.Media penyimpanan yang digunakan adalah SD card yang dipakai

untuk proses *booting* dan penyimpanan data. Raspberry Pi memiliki performa dan konsumsi daya yang cocok untuk digunakan pada berbagai macam pekerjaan tanpa memerlukan banyak daya.

Raspberry Pi memiliki 8P8C (RJ45) *Ethernet port* untuk menghubungkan komputer ini ke jaringan LAN. Selain itu, USB Wi-Fi adapter juga dapat dipasang pada USB port yang ada pada Raspberry Pi ini agar dapat melakukan komunikasi nirkabel.



Gambar 2.8 Raspberry Pi Model B

(Sumber: www.raspberrypi.org)

Selain memperluas kapabilitas dengan antarmuka *port* USB, terdapat antarmuka lain agar *peripheral* dapat berkomunikasi dengan Raspberry Pi. Dengan adanya *port* GPIO serta dukungan protokol I2C, kapabilitas Raspberry Pi dapat dengan mudah diperluas hanya dengan menyambungkan *peripheral* melalui antarmuka yang tersedia pada Raspberry Pi ini.

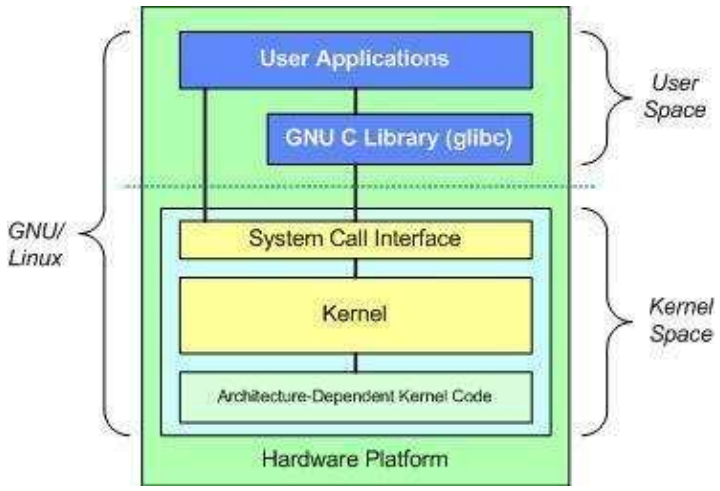
Terdapat macam-macam sistem operasi yang dapat dijalankan pada Raspberry Pi dengan berbagai tujuan penggunaan seperti *media center* dengan menggunakan sistem operasi seperti OpenElec, Raspbmc, XBian. Sistem operasi dapat dengan mudah diganti dengan cara menukar SD card dengan isi sistem operasi yang berbeda.Selain itu, terdapat *bootloader* seperti BerryBoot agar pengguna dapat menyimpan lebih dari satu sistem operasi dalam satu SD card.

2.5 Sistem Operasi Linux

Linux merupakan *operating system* yang mengadaptasi desain kerja dari Unix, yaitu *operating system* yang bersifat *multi-user*. *Operating system* sendiri merupakan *software* yang digunakan agar komputer dapat dioperasikan dan dapat mengakses *hardware* pada komputer tersebut.

Terdapat beberapa keunggulan yang ditawarkan oleh Linux melalui *distribution* yang dihasilkan baik secara *free* ataupun secara *proprietary*.*Distribution* yang dimaksud disini merupakan jenis-jenis *operating system* yang dibangun diatas Linux *kernel* dan tersusun atas *libraries* serta *utilities* yang berasal dari GNU. Tiap *distribution* memiliki kecocokan masing-masing berdasarkan fungsi yang dibutuhkan untuk komputer tersebut. Sebagai contohnya, Fedora dan CentOS yang digunakan sebagai *server*, Ubuntu yang digunakan untuk *general-purpose computing*.

Operating system dari Linux secara garis besar terdiri atas dua *level*, dimana pembagian *level* pada umumnya berdasarkan pada pemisahan *memory* yang digunakan yaitu *user space* dan *kernel space*.*User space* merupakan bagian *memory* digunakan oleh program aplikasi ataupun *libraries* yang melakukan interaksi dengan *kernel* untuk menyelesaikan pekerjaannya.Sedangkan *Kernel space* merupakan *space* yang digunakan hanya untuk tempat berjalannya *software* untuk mengakses *hardware* yang ada pada komputer.



Gambar 2.9 Struktur Sistem Operasi dari Linux

(Sumber: www.ibm.com)

Pemisahan ini bertujuan untuk mencegah adanya intervensi antara data yang dimiliki oleh *user* dengan data yang dimiliki oleh *kernel*. Tanpa adanya pemisahan ini, berisiko terjadi penurunan kinerja dari komputer ataupun menyebabkan sistem menjadi tidak stabil.

2.4.1 Debian

Debian merupakan *operating system* yang mempergunakan berbagai *kernel* untuk mengakses *services* yang terdapat pada *hardware* dan melakukan *handling* terhadap *software* diatasnya. Berdasarkan situs resmi Debian (http://www.debian.org).*Kernel* yang digunakan Debian diantara lain adalah Linux *kernel* (Debian GNU/Linux)dan FreeBSD *kernel* (Debian GNU/kFreeBSD), selain hal tersebut, Debian terdiri atas *software/tools* yang bersifat gratis dan memiliki lisensi GNU General Public License.

Debian kebanyakan digunakan sebagai *operating system* pada *personal computer* dan juga pada mesin *server.*

Secara teknis kelebihan Debian diantaranya adalah:

1. Kemudahan instalasi: hal ini dikarenakan tidak adanya *dependencies, library,* dan perubahan *configuration files.* Seluruh *package* mengikuti *Filesystem Hierarchy Standard* sehingga memiliki *file and directory structure* yang sama.
2. Portabilitas: dapat berjalan di bermacam-macam *hardware platform* dibandingkan dengan *distribution* lain. Terhitung hingga Debian 7.0 (*Wheezy*), arsitektur *hardware* yang telah didukung oleh Debian mencapai 12 arsitektur.
3. Fleksibilitas peningkatan sistem: *upgrading*, dalam hal ini menyangkut *update* pada *package* ataupun pembaharuan versi di Debian dapat dilakukan dengan berbagai cara, yaitu melalui media CD ataupun melalui *network* dengan bantuan *package manager* seperti apt dan dpkg.

Selain hal tersebut, Debian memiliki keunggulan pada *community* yang melakukan pengembangan dan juga melakukan pengecekan untuk mendeteksi *bugs.* Namun kekurangan dari Debian adalah lama tiap *update*

yang dihasilkan. Hal ini disebabkan oleh waktu dan biaya yang dibutuhkan untuk melakukan pengujian Debian pada berbagai macam arsitektur sistem.

Debian mendukung dua jenis arsitektur dari ARM, yaitu armel dan armhf.

1. armel: digunakan pada *lower-end hardware* yang mendukung *instruction set* ARMv4 dan *hardware floating point* melalui *compatibility mode*.
2. armhf: digunakan pada *hardware* yang mendukung ARMv7 *instruction set* dan *hardware floating point* secara langsung tanpa melalui *compatibility mode.*

2.6 Raspbian

Raspbian merupakan *operating system* berbasis pada Debian yang merupakan *platform* dari Linux. *Operating system* yang secara resmi direkomendasikan untuk penggunaan pada Raspberry Pi ini dibuat dengan tujuan untuk mendukung kerja *hardware* dari Raspberry Pi, dan diluncurkan pada bulan Juli tahun 2012 (Richardson & Wallace, 2012).

Berdasarkan situs resmi dari Raspbian (http://www.raspbian.org), Raspbian merupakan hasil *port* tidak resmi dari Debian *wheezy* armhf (ARM Hard Float). *Porting* merupakan proses mengadaptasi suatu program agar dapat berjalan pada *environment* yang berbeda, dalam hal ini, dengan *hardware platform* yang berbeda.

ARM Hard Float merupakan arsitektur ARM yang ditambah dengan *hardware* untuk operasi *floating point* yang disebut dengan VFP (*Vector Floating Point*). Penambahan VFP ini dibuat untuk memanfaatkan *floating point hardware* yang terdapat di dalam prosesor Raspberry Pi, sehingga proses yang memerlukan perhitungan dengan menggunakan *floating point* dapat dikerjakan lebih cepat, dan dapat melakukan pemrosesan terhadap instruksi *advance* yang terdapat pada arsitektur ARMv6. Walaupun begitu, Raspbian pun dapat digunakan untuk sistem non-Raspberry yang mendukung arsitektur diatas dari ARMv6+VFP (contohnya ARMv7-A).

2.7 JSON (JavaScript Object Notation)

JavaScript Object Notation (JSON) merupakan suatu *format* pertukaran data yang dirancang menjadi *format* yang mudah dibaca dan ditulis oleh manusia, serta mudah untuk dihasilkan dan diproses oleh komputer (Nurseitov, Paulson, Reynolds, & Izurieta., 2009).


```
{
   "id": "1636580217",
   "name": "Kevin Dwi Utomo",
   "first_name": "Kevin",
   "middle_name": "Dwi",
   "last_name": "Utomo",
   "link": "https://www.facebook.com/kevin.d.utomo",
   "username": "kevin.d.utomo",
   "gender": "male",
   "timezone": 7,
   "locale": "en_US",
   "verified": true,
   "updated_time": "2013-09-28T04:23:18+0000"
}
```


Gambar 2.10 Contoh JSON Response dari Facebook Graph API

JSON telah menjadi standar *format* pertukaran data internasional resmi sejak Oktober 2013 dengan spesifikasi standar oleh *Ecma International* (ECMA-404). Menurut spesifikasi standar internasional yang diterbitkan oleh *Ecma International*, JSON merupakan *format* teks yang berguna untuk mempermudah pertukaran data berstruktur antara semua bahasa pemrograman.

Format pertukaran data JSON tidak bergantung pada bahasa pemrograman yang dipakai, namun susunan data-nya menggunakan aturan yang mirip dengan bahasa pemrograman C. JSON diperkirakan dapat menguraikan data seratus kali lebih cepat dibandingkan dengan XML di *modern browser*.

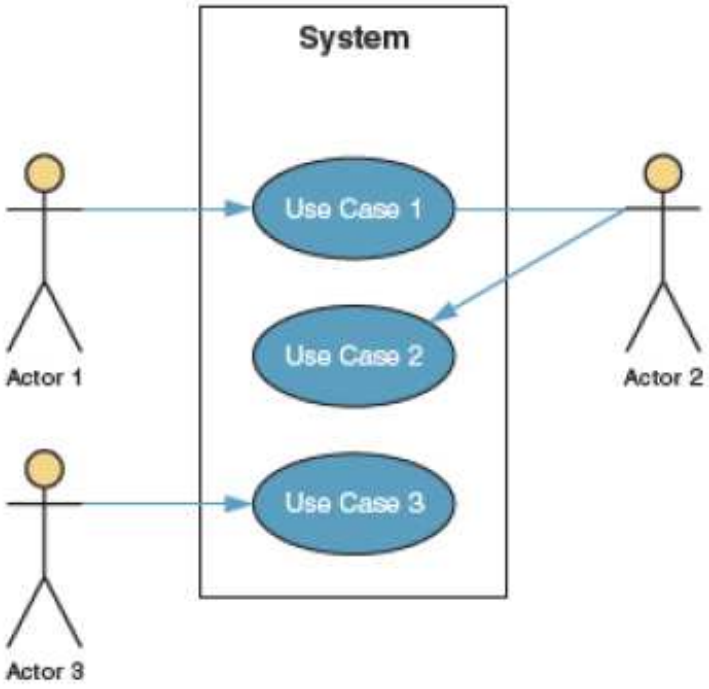
Berdasarkan statistik *ProgrammableWeb*, tiga dari empat API bersifat *RESTful* dan hampir setengah dari API tersebut menggunakan JSON sebagai *format* pertukaran data. Beberapa API yang paling terkenal seperti *Facebook Graph API, Twitter API,* dan *Foursquare API* sekarang hanya mendukung format JSON. Salah satu alasan mengapa JSON mengalahkan XML sebagai standar pertukaran data adalah masalah interoperabilitas yang dialami oleh XML karena struktur skema dari XML berbeda dengan struktur *object oriented* (Markus Lanthaler et. al., 2012)

.

2.8 UML (*Unified Modeling Language*)

UML (*Unified Modeling Language*) merupakan kumpulan model yang digunakan untuk menggambarkan atau menspesifikasi sebuah sistem piranti lunak yang terkait dengan objek (Whitten & Bentley, 2007).Sebuah *management station*,

2.8.1 *Use Case Diagram*



Gambar 2.11 Diagram *Use Case*

(Sumber: *System Analysis and Design Methods*, Whitten & Bentley, 2007)

Use case diagram merupakan diagram yang menggambarkan interaksi antara sistem dengan bagian eksternal dari sistem seperti sistem eksternal dan pengguna. Dapat dikatakan, diagram ini berfungsi untuk mendeskripsikan siapa saja yang berinteraksi dengan sistem, bagaimana cara pengguna berinteraksi dengan sistem. Komponen-komponen yang ada dalam *use case* diantaranya adalah: (Whitten & Bentley, 2007)

1. *Use Cases*

   Model *Use cases* mengidentifikasi dan mendeskripsikan fungsi sistem dengan menggunakan *tool* yang disebut dengan *use cases*. *Use cases* direpresentasikan secara grafis dengan bentuk *ellipse* horizontal dengan nama dari *use case* yang terdapat di atas, di bawah, ataupun di dalam *ellipse* tersebut. Setiap *use case* merepresentasi tujuan dari sistem dan mendeskripsikan langkah-langkah dan interaksi dari pengguna untuk mencapai tujuan tersebut.

2. *Actors*

Gambar 2.12 Simbol *Actor* pada *Use Case*

(Sumber: *System Analysis and Design Methods*, Whitten & Bentley, 2007)

Use cases dipicu oleh *external user* yang disebut dengan *actors.* Suatu *actor* memicu suatu *use case* dengan tujuan untuk menyelesaikan suatu tugas yang menghasilkan sesuatu yang dapat diukur. (Whitten & Bentley, 2007: 247).

1. *Primary business actor*

   Adalah *stakeholder* yang memiliki peran utama dari eksekusi suatu *use case* dengan menerima nilai yang dapat diukur dan di observasi. Aktor ini bisa saja tidak memicu *event*.

2. *Primary system actor*

   Adalah *stakeholder* yang bertatap muka secara langsung dengan sistem untuk memicu *event*.

3. *External server actor*

   Adalah *stakeholder* yang merespon permintaan dari *use case*.

4. *External receiver actor*

   Adalah *stakeholder* yang bukan merupakan *primary actor,* namun menerima nilai yang terukur dari hasil eksekusi *use case*.

Pada berbagai sistem informasi, terdapat beberapa tugas yang dipicu oleh waktu secara otomatis. Hal ini disebut dengan *temporal events*. Dalam hal ini, aktor dari *temporal event* adalah *time* (waktu).

3. *Relationships*

Sebuah *relationship* digambarkan dengan adanya garis di antara dua simbol yang terdapat pada *use-case diagram*. Arti dari *relationship* bisa berbeda-beda bergantung bagaimana garis tersebut digambarkan dan jenis dari simbol yang saling terhubung. Beberapa jenis dari *relationship* diantaranya: (Whitten & Bentley, 2007: 248)

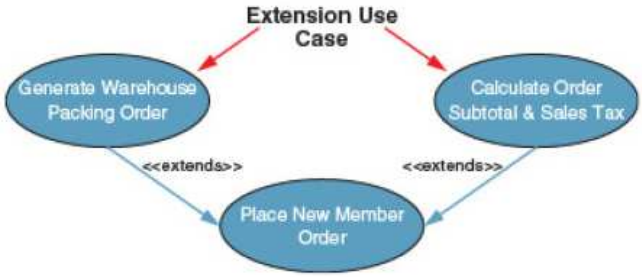
a. *Associations*

Gambar 2.13 *Associations* pada *Use Case*

(Sumber: *System Analysis and Design Methods*, Whitten & Bentley, 2007)

Hubungan antara *use case* dan *actor* muncul ketika adanya interaksi diantaranya. Hubungan ini disebut dengan *associations*. Adanya ujung panah pada garis pada *use case* menunjukkan *actor* memicu adanya *event*, sedangkan tidak adanya ujung panah menunjukkan adanya hubungan antara *external server* atau *receiver actor*.

b. *Extends*

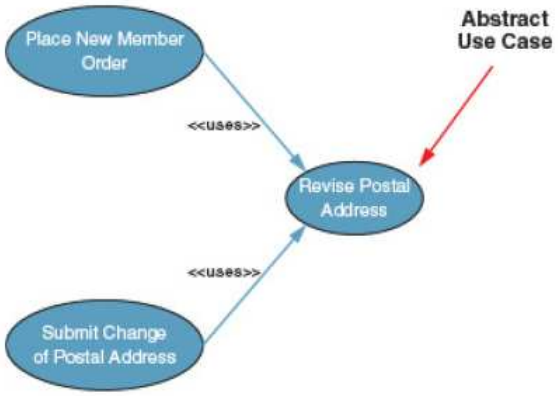


Gambar 2.14 *Extension relationship*

(Sumber: *System Analysis and Design Methods*, Whitten & Bentley, 2007)

Use case dapat berisi fungsi yang kompleks yang terdiri atas beberapa tahapan yang membuat logika *use case* sulit untuk dimengerti. Fungsi ini dapat dipecah menjadi beberapa bagian yang terpisah. Hal ini dinamakan *extension use case* yang merupakan perpanjangan dari *use case* awal. *Use case* ini mengekstraksi fungsi awal menjadi tahapan-tahapan yang lebih mudah dimengerti. Relasi jenis ini ditandai dengan nama <<extends>>.

c. *Uses (*atau *Includes*)

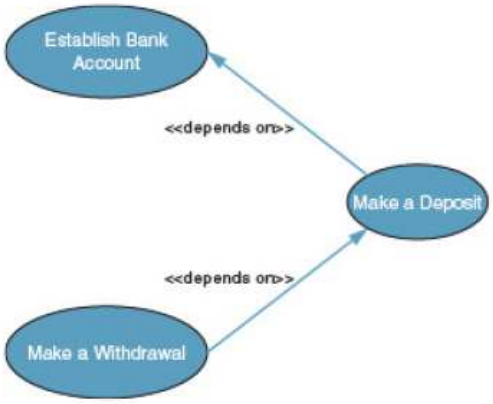


Gambar 2.15 *Uses relationship*

(Sumber: *System Analysis and Design Methods*, Whitten & Bentley, 2007)

Pada umumnya, dapat ditemukan satu atau lebih *use case* yang menjalankan fungsi identik. Dapat dilakukan ekstraksi dari *use case* ini dan menghasil *use case* terpisah menjadi *abstract use case* untuk mengurangi redundansi. Relasi diantara *use case* ini disebut dengan relasi *uses* atau dapat disebut juga relasi *includes*. Relasi ini ditandai dengan nama <<uses>>.

d. Depends On

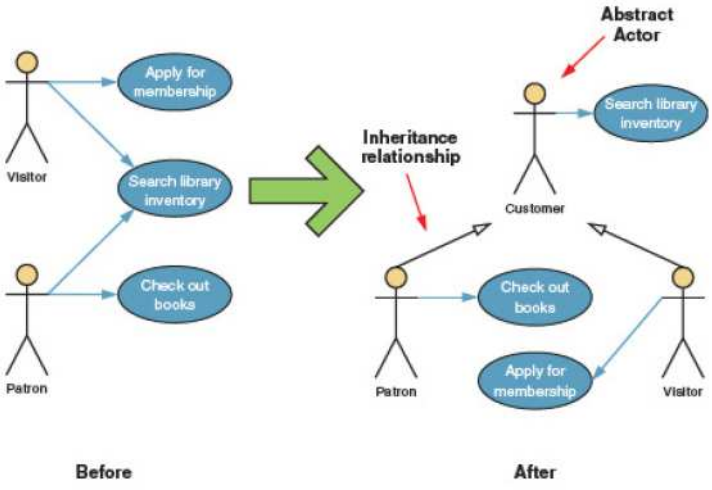


Gambar 2.16 *Depends on Relationship*

(Sumber: *System Analysis and Design Methods*, Whitten & Bentley, 2007)

Relasi *depends on* adalah hubungan antar *use case* yang menunjukkan bahwa suatu *use case* tidak dapat berjalan apabila *use case* lain selesai dijalankan.

e. Inheritance



Gambar 2.17 *Inheritance relationship*

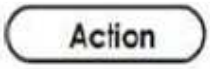
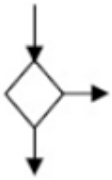
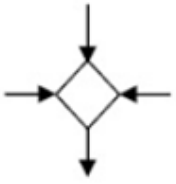
(Sumber: *System Analysis and Design Methods*, Whitten & Bentley, 2007)

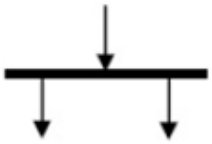
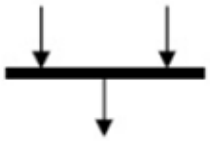
Suatu hubungan antar *actor* yang dibuat untuk menyederhanakan penggambaran suatu *abstract actor* yang menurunkan perannya kepada lebih dari satu *actor*.

2.8.2 *Activity Diagram*

Activity diagram merupakan diagram yang digunakan untuk menggambarkan alur proses bisnis, langkah-langkah pada suatu *use case*, dan logika dari tingkah laku objek (Whitten & Bentley, 2007:390). Notasi yang ada pada diagram ini diantaranya:

Tabel 2.4 Notasi pada *Activity Diagram*

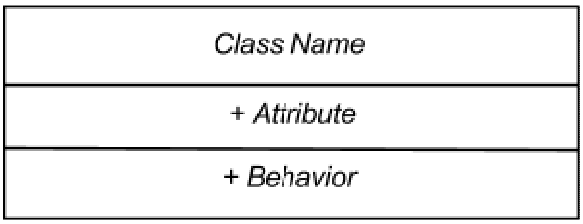
No	Simbol	Gambar	Penjelasan
1.	*Initial node*		Menunjukkan awal dari proses.
2.	*Actions*		Menunjukkan langkah dari suatu aktivitas.
3.	*Flow*		Menunjukkan proses selanjutnya yang akan dilakukan.
4.	*Decision*		Bentuk *diamond* yang memiliki satu *flow* yang masuk dan dua atau lebih *flow* yang keluar. Digunakan untuk menggambarkan suatu kondisi
5.	*Merge*		Bentuk *diamond* yang memiliki satu atau lebih *flow* yang masuk dan satu *flow* yang keluar. Berfungsi untuk menyatukan *flow* yang sebelumnya terpisah.

6.	*Fork*		Menggambarkan proses paralel yang terjadi secara bersamaan.
7.	*Join*		Menggambarkan selesainya proses yang berjalan secara paralel.
8.	*Activity Final*		Menunjukkan akhir dari suatu proses.

2.8.3 *Class Diagram*

Class diagram adalah gambaran berupa struktur objek statis yang terdapat di dalam sistem, yang menunjukkan kelas objek pada sistem dan relasi antar kelas objek. Beberapa konsep yang ada pada *class diagram* diantaranya:

1. *Object class*



Gambar 2.18 *Object Class*

Object class adalah kumpulan objek yang terdiri atas *attribute* dan *behavior*.

a. *Attribute*

Attribute adalah data yang merepresentasikan karakteristik dari suatu objek.

b. *Behavior*

Behavior adalah kumpulan hal yang dapat dilakukan oleh objek dan berhubungan dengan fungsi yang bekerja pada atribut dari objek. *Behavior* dapat disebut juga *method, operation,* atau *service*.

2. *Visibility*

Visibility merupakan istilah yang menunjukkan apakah elemen dalam suatu objek dapat dilihat oleh objek lain.

a. *Public:* menunjukkan bahwa elemen dari *class* dapat dilihat dari luar. Ditandai dengan lambang '+'.
b. *Protected:* menunjukkan bahwa elemen dapat dilihat oleh *class* yang terdefinisi dan *class* turunannya saja. Ditandai dengan lambang '#'.
c. *Private:* menunjukkan bahwa elemen hanya dapat dilihat pada *class* yang terdefinisi saja. Ditandai dengan lambang '-'.

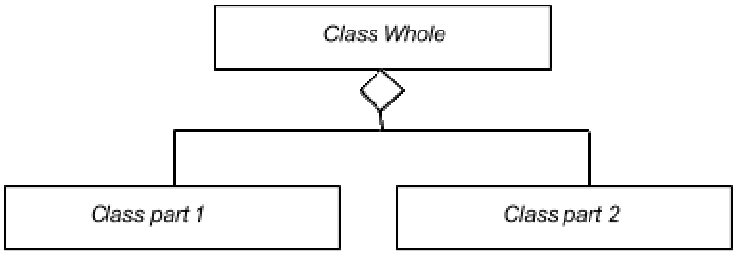
3. *Association*

Association menunjukkan adanya hubungan antar dua *class* atau lebih dalam sebuah *class diagram*.

Gambar 2.19 *Class relationship*

4. *Aggregation*

Aggregation merupakan hubungan antar *class*, dimana suatu *class* merupakan bagian dari *class* lain. Misalkan *class* A merupakan bagian dari *class* B, namun *class* B bukanlah bagian dari *class* A.

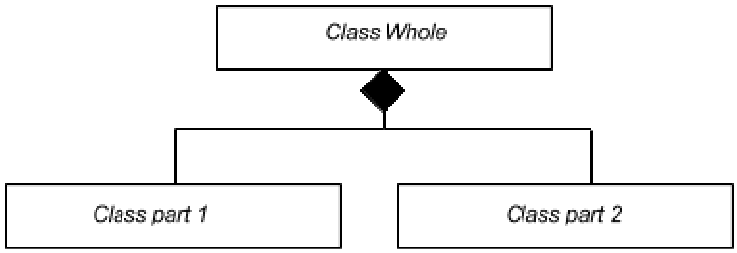


Gambar 2.20 *Aggregation relationship*

5. *Composition*

Composition merupakan hubungan yang mirip dengan *aggregation*, namun *composition* memiliki hubungan yang lebih erat. Sebagai contoh,

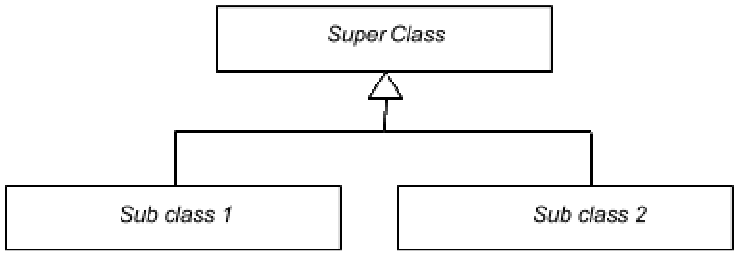
apabila *class A* merupakan bagian dari *class* B. Jika *class* B tidak ada, maka *class* A tidak akan ada.



Gambar 2.21 *Composition relationship*

6. Generalization

Generalization merupakan bentuk *inheritance*, dimana *super class* (induk) merupakan bentuk umum dari *sub class* (turunan). Sehingga dapat dikatakan *sub class* memiliki seluruh atribut dan *method* dari *super class*.



Gambar 2.22 *Generalization relationship*

7. Multiplicity

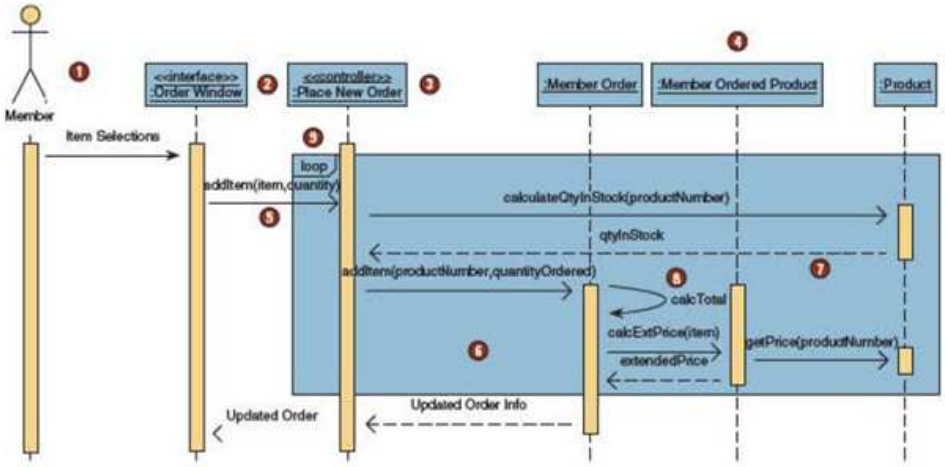
Multiplicity menggambarkan banyaknya hubungan yang dapat terjadi di antara suatu *class*. Terdapat beberapa jenis:

Tabel 2.5 *Multiplicity* pada *Class Diagram*

No.	*Multiplicity*	Deskripsi
1.	1	Hanya satu.
2.	0..1	0 atau 1.
3.	*	0 atau lebih.
4.	1..*	1 atau lebih.
5.	m..n	Bisa memiliki jumlah dari m sampai dengan n.

2.8.4 *Sequence Diagram*

Sequence diagram adalah diagram yang menggambarkan interaksi antara aktor dan sistem dalam waktu yang berurutan pada suatu *use case*. Interaksi digambarkan dalam waktu yang berurutan sesuai dengan skenario *use case*. Notasi pada *sequence diagram* diantaranya: (Whitten & Bentley, 2007: 394).



Gambar 2.23 *Sequence diagram*

(Sumber: *System Analysis and Design Methods*, Whitten & Bentley, 2007)

1. *Actor*

   Merupakan aktor yang memulai *use case* yang ditunjukkan dengan simbol *actor*.

2. *Interface class*

   Merupakan *class* yang digunakan pada sistem untuk berinteraksi dengan *actor*. Di tulis dengan <<*interface*>>.

3. *Controller class*

   Merupakan *class* yang melakukan pengendalian eksekusi, yang terhubung dengan *interface class*. Biasanya tiap *interface class* bisa terdiri dari satu atau lebih *controller class.* Di tulis dengan <<*controller*>>.

4. *Entity class*

   Merupakan *class* yang berkorespondensi dengan objek

dalam dunia nyata yang bertugas untuk eksekusi proses.

5. *Messages*

   Merupakan tanda panah horizontal padat yang menandakan adanya *message input* yang dikirim dari satu *class* ke *class* lain.

6. *Activation bars*

   Merupakan garis berbentuk batang yang menandakan periode waktu ketika suatu *actor* ataupun *class* sedang aktif dalam interaksi.

7. *Return messages*

   Merupakan tanda panah horizontal terputus-putus yang menandakan adanya respon dari *message* yang dikirim.

8. *Self-call*

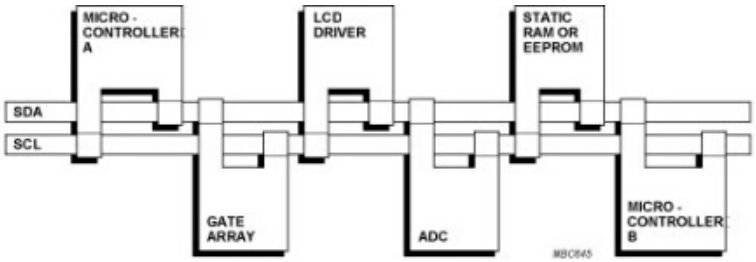
   Ditandakan dengan panah yang yang asal dan tujuan dari *message* memiliki kesamaan. Menandakan objek yang memanggil metodenya sendiri.

9. *Frame*

   Menunjukkan suatu *boundary* dalam *use case.* Dalam hal ini dapat digunakan untuk menunjukkan pengulangan suatu proses, ataupun dibutuhkan *controller* untuk melakukan *looping* pada seluruh *item.*

2.9 I2C

I2C merupakan singkatan dari Inter-Integrated Circuit, standar sistem komunikasi data *serial* antar beberapa komponen atau *peripheral* dengan komputer. I2C dirancang oleh Philips Semiconductors pada tahun 1982. I2C dapat menghubungkan beberapa komponen dengan menggunakan jalur data (*bus*) yang sama (Muzaffar, 2005).



Gambar 2.24 I2C *Bus* yang Dipakai oleh Enam Komponen secara Bersamaan

(Sumber: *Inter-Intergrated-Circuit* (I2C), Muzaffar, 2005)

Menurut Ramandeep Singh et. al. (2013), I2C memiliki *interface* yang sederhana dan mudah untuk digunakan. I2C terdiri dari 2 jalur *bus*, yaitu *serial data line* (SDA) dan *serial clock line* (SCL). Standar komunikasi ini memiliki kelebihan *error detection* saat transmisi data.

Setiap *device* yang terhubung ke *bus* I2C memiliki alamat yang unik agar dapat dikontrol secara terpisah, *software* akan menentukan alamat dari suatu *device*, sehingga *hardware* tidak perlu untuk membaca ulang alamat. *Device* yang terhubung ke *bus* I2C dapat menjadi *master* atau *slave*, *master* adalah *device* yang melakukan inisialisasi transmisi data, dan *slave* adalah *device* yang akan merespon ke *device master*.

I2C memiliki beberapa *field* yang digunakan untuk melakukan transmisi data ke beberapa *device:*

1. *Start* : Transmisi data akan dimulai dengan kondisi *Start.* Ketika kondisi ini dijalankan, maka *bus* akan menjadi *busy state.*
2. *Address* : Alamat 10-bit untuk menentukan ke mana pesan akan dikirimkan.
3. *Acknowledge* : Ketika alamat sudah di transmisikan ke *bus* maka *device* yang memiliki alamat yang dituju harus merespons dengan sebuah ACK.
4. *Stop* : *Field* ini menentukan akhir dari suatu pesan.

   Proses interkoneksi dari I2C memiliki alur berikut:

   1. *Master* akan melakukan kondisi *Start* pada *bus* yang terhubung, kondisi ini akan bertindak sebagai sinyal ke semua *device* yang terhubung untuk mengamati *bus* untuk *data* yang akan masuk.
   2. *Master* akan mengirimkan alamat dari *device* yang akan di akses, bersamaan dengan tipe operasi yang hendak digunakan (*Read* atau *Write*).

Setelah *device* menerima alamat yang dikirim, maka *device* akan melakukan komparasi dengan alamat *device* itu sendiri. Jika alamat tidak cocok, maka *device* akan menunggu sampai *bus* dalam kondisi *Stop*, ketika alamat cocok, maka *device* akan mengirimkan sinyal *Acknowledge.*

3. Setelah *Master* menerima sinyal *acknowledge*, maka transmisi data dapat dilakukan. Ketika transmisi data sudah selesai, *master* akan mengirimkan kondisi *Stop.*

Dalam proses transmisi data, I2C dapat digunakan dalam topologi *single master* atau *multi master*. Pada topologi *single master* tidak ada masalah *collision* karena hanya satu *master device* yang menentukan kapan untuk melakukan operasi *read* atau *write*. Namun, pada topologi *multi master*, dibutuhkan metode transmisi data untuk memastikan tidak adanya *collision.* Sebelum transmisi data dilakukan, dibutuhkan prosedur berikut:

1. *Backing off*:

   Sebelum *master* melakukan transmisi data, *master* harus mengecek *bus* apakah *bus* sedang dipakai, jika *bus* sedang dipakai, maka *master* akan menunggu hingga *bus* sedang dalam kondisi tidak terpakai.

2. *Software Time-outs*:

   Bahkan dengan prosedur *backing off,* dibutuhkan prosedur lain untuk memastikan tidak adanya *collision,* karena masih ada kemungkinan dua atau lebih *master* menggunakan *bus* pada saat yang sama. Maka setiap *device* yang terhubung ke *bus* harus mengamati jalur SDA dan SCL apakah jalur tersebut sedang dipakai dalam waktu yang sangat lama secara terus-menerus. Jika *device* mendapatkan kondisi tersebut, maka *device* tersebut harus melakukan inisialisasi ulang modul I2C dan melepaskan *bus* yang terhubung.

2.10 GPIO (General Purpose Input/Output)

General-purpose input/output (GPIO) merupakan suatu *pin* yang dapat ditemukan di suatu *integrated circuit* (IC) dimana arah *data* pada *pin* tersebut dapat dikontrol oleh *user*. GPIO memiliki kapabilitas sebagai berikut:

Pin GPIO yang ada pada Raspberry Pi dapat diakses untuk mengontrol *hardware* seperti lampu LED, *motor, relay,* dan *peripheral* lainnya sebagai *output*. *Pin* GPIO ini juga dapat dikonfigurasi sebagai *input* sehingga Raspberry Pi dapat

membaca *button state* atau membaca nilai yang diberikan dari berbagai *sensor* seperti temperatur, cahaya, gerakan, dan berbagai *sensor* lainnya (Richardson & Wallace, 2012).

1. *Pin* GPIO dapat dikonfigurasi sebagai *input* atau *output.*
2. *Pin* GPIO dapat diaktifkan dan dinonaktifkan sesuai keingingan.
3. Nilai *input* dapat dibaca (biasanya *high*=1, *low*=0)
4. Nilai *output* dapat dibaca dan ditulis sesuai keinginan
5. Nilai *input* dapat digunakan sebagai *interrupt request* (IRQ)

2.11 Python

Menurut Ljubomir Perkovic (2012), Python merupakan bahasa pemrograman dengan tujuan umum yang dikembangkan secara khusus untuk membuat *source code* mudah dibaca. Python juga memiliki *library* yang lengkap sehingga memungkinkan *programmer* untuk membuat aplikasi yang mutakhir dengan menggunakan *source code* yang tampak sederhana.

Data hiding pada Python hanya merupakan konsep atau konvensi sehingga *client* dapat mengambil atau mengubah atribut di setiap kelas atau *instance*. Atau pada istilah C++, semua atribut pada Python memiliki *modifier* "*public*" dan "*virtual*" sehingga atribut tersebut dapat diakses dari luar kelas. (Mark Lutz, 2013: 944)

Source code aplikasi dalam bahasa pemrograman Python biasanya akan di kompilasi menjadi *format* perantara yang dikenal sebagai *bytecode* yang selanjutnya akan dieksekusi. Kelemahan dalam bahasa pemrograman ini terletak pada kecepatan eksekusi yang tidak secepat bahasa pemrograman yang di kompilasi dan bersifatlebih *low-level* seperti C dan C++. Berikut beberapa kelebihan bahasa pemrograman Python menurut Mark Lutz (2013):

1. **Kualitas *software***

Bagi banyak orang, fokus bahasa pemrograman Python pada kemudahan *source code* untuk dibaca, koherensi *source code*, dan kualitas *software* secara umum membedakan Python dari bahasa pemrograman lain. Bahasa pemrograman Python dirancang agar mudah dibaca, sehingga mendukung penggunaan kembali *source code* (*code reusability)* dan memudahkan *programmer* untuk mengatur *source code*

jika dibutuhkan perubahan. Selain itu, Python juga mendukung *Object-oriented programming* (OOP).

2. **Produktivitas *developer***

Bahasa pemrograman Python lebih meningkatkan produktivitas *developer* dibandingkan dengan bahasa pemrograman lain seperti C, C++, dan Java. *Source code* Python biasanya memiliki besar file sepertiga sampai seperlima dari besar file *source code* dengan bahasa pemrograman C++ atau Java. Hal ini berarti mengurangi besarnya *source code* yang harus ditulis oleh *developer* sehingga proses *debugging* aplikasi akan menjadi lebih mudah. Selain itu, *program* dengan bahasa pemrograman Python dapat dijalankan secara langsung tanpa proses *compiling* dan *linking* yang dibutuhkan oleh bahasa pemrograman lain.

3. **Portabilitas program**

Sebagian besar *program* yang dikembangkan dengan bahasa pemrograman Python berjalan tanpa adanya perubahan pada perangkat yang berbeda-beda. Jika *programmer* ingin menjalankan *program* Python pada perangkat yang menjalankan Linux dan Windows, *programmer* dapat dengan mudah menjalankan *program* tersebut tanpa modifikasi. Selain itu, Python menyediakan opsi untuk mengembangkan *Graphical User Interfaces*(GUI), sistem berbasis web, hingga antarmuka sistem operasi yang dapat digunakan di berbagai *platform*.

4. **Dukungan *library***

Bahasa pemrograman Python dilengkapi dengan kumpulan fungsionalitas yang bersifat portabel, yang dikenal dengan *Python standard library*. *Library* ini mendukung berbagai fungsionalitas dasar sampai kompleks yang portabel. Selain itu, *library* python dapat diperluas lagi dengan menggunakan *library* yang dikembangkan oleh pihak ketiga.

5. **Integrasi komponen**

Python dapat dengan mudah melakukan komunikasi dengan bagian lain dari sebuah aplikasi dengan menggunakan mekanisme integrasi yang bermacam-macam. Integrasi ini menyediakan kapabilitas Python agar dapat dipakai sebagai alat ekstensi. Sebagai contoh, bahasa pemrograman Python dapat memanggil *library* C dan C++, dan sebaliknya.

6. Kenyamanan *programmer*

Dengan kemudahan penggunaan bahasa pemrograman Python dan berbagai alat bantuan yang sudah ada dalam Python, pemrograman dapat dibuat lebih ke kesenangan daripada pekerjaan yang membosankan. Meskipun kelebihan ini mungkin berupa manfaat yang tidak berwujud, pengaruhnya pada produktivitas merupakan pengaruh yang penting.

Contoh aplikasi yang menggunakan bahasa pemrograman Python:

1. *Search engine Google* menggunakan Python pada bagian sistem pencarian *web*-nya.
2. Layanan berbagi *videoYouTube*.
3. Penyedia layanan penyimpanan *Dropbox* menggunakan Python.

2.12 SSH (*Secure Shell*)

Secure Shell (SSH) merupakan suatu protokol yang digunakan untuk menciptakan suatu saluran komunikasi yang terenkripsi antara dua *host.* SSH melindungi data yang melewati jaringan antara dua *host* sehingga orang lain tidak bisa mengakses data yang ditransmisikan. Tatu Ylönen menciptakan protokol awal dan implementasinya pada tahun 1995, dan dengan cepat menyebar dan menggantikan protokol lain yang tidak aman seperti telnet, rsh, dan rlogin (Lucas, 2011).

SSH menggunakan protokol *packet-based* yang dapat bekerja melalui protokol *transport* apapunyang dapat mentransmisikan data biner. Biasanya, TCP/IP akan digunakan sebagai protokol *transport,* namun SSH juga mendukung penggunaan *proxy* dan *firewall* berbasis SOCKS untuk mengirim dan menerima data ke *server*.

2.13 Socket Programming

Socket merupakan suatu *endpoint* pada komunikasi antara komputer. Socket programming memiliki tujuan agar *device* dapat melakukan komunikasi satu sama lain melalui suatu jaringan komputer dengan menggunakan protokol TCP ataupun UDP (Stevens, 2004).

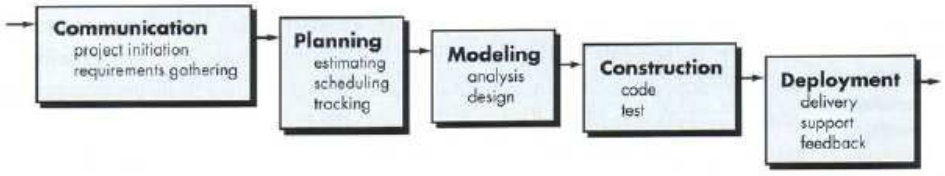
2.14 Wi-Fi

Wi-Fi merupakan singkatan dari *wireless fidelity*. Wi-Fi ini sebenarnya merupakan merek dagang yang diapakai oleh Wi-Fi Alliance dan sebagai nama merek untuk produk yang memiliki standar IEEE 802.11. Karena sebagian besar teknologi *Wireless Local Area Network* (WLAN) mempunyai dasar dari standar Wi-Fi, istilah Wi-Fi umumnya digunakan sebagai sinonim untuk WLAN.

Wi-Fi adalah teknologi yang memungkinkan perangkat elektronik untuk bertukar informasi dan bertukar *data* secara nirkabel. Setiap perangkat elektronik yang mempunyai kapabilitas Wi-Fi dapat dihubungkan melalui suatu *wireless network access point.*

Namun, Wi-Fi juga memiliki beberapa kekurangan. Wi-Fi dikenal kurang aman daripada koneksi kabel (seperti Ethernet) karena penyusup dapat terhubung ke jaringan yang bersifat *private* tanpa membutuhkan akses fisikal. Karena itu, Wi-Fi telah mengadopsi berbagai teknologi enkripsi seperti WEP, WPA, dan WPA2. Pada tahun 2007, dikembangkan fitur *Wi-Fi Protected Setup* (WPS). Namun, fitur ini memiliki kelemahan yang memungkinkan penyerang untuk mendapatkan *password* dari *router* (Haque, Kumar, Pandey& Singh, 2013).

2.15 Model *Waterfall*



Gambar 2.25 Bagan *Waterfall System Development*

(Sumber: *Software Engineering – A Practicioner's Approach*, Pressman, 2005 : 79)

Menurut Pressman (2005), model *Waterfall* cocok untuk pengembangan sistem dengan kebutuhan yang sudah dipahami dengan baik. Model *Waterfall* dikenal juga sebagai *classic life cycle,* suatu pendekatan sistematik dan berurutan dalam pengembangan sistem yang diawali dengan

spesifikasi dari kebutuhan (*requirements)* dan dilanjutkan dengan *planning, modeling, construction,* dan *deployment.*

Model *Waterfall* memiliki tahap-tahap:

1. *Communication*

   Dilakukan analisa kebutuhan sistem dan pengumpulan data dengan melakukan pertemuan dengan *user.*

2. *Planning*

   Dilakukan perencanaan untuk pengembangan sistem dan melakukan evaluasi apa saja pekerjaan teknis yang dilakukan, sumber daya yang dibutuhkan, hasil produk, dan jadwal pengerjaan dari sistem.

3. *Modeling*

   Membuat suatu model yang mempunyai tujuan untuk lebih menjelaskan kebutuhan agar *developer* dan *user* dapat lebih mengerti kebutuhan-kebutuhan yang diperlukan serta disain yang dapat mencapai kebutuhan tersebut. Tahap ini juga bertujuan untuk dapat lebih mengerti bagaimana *end-user* dapat berinteraksi dengan sistem yang dikembangkan.

4. *Construction*

   Mengembangkan kode atau *coding* dengan mengacu pada *requirement* dan *model* yang sudah dibuat pada tahap sebelumnya. Setelah pengembangan kode dilakukan, *testing* akan dilakukan terhadap sistem yang telah dibuat. Tujuan *testing* ini adalah untuk melihat apakah ada kesalahan dalam sistem.

5. *Deployment*

Dilakukan implementasi sistem secara keseluruhan untuk mendapatkan *feedback* dan evaluasi. Sistem yang telah dibuat akan dilakukan pemeliharaan secara berkala.

2.16 Pengujian Hipotesis

Pengujian hipotesis merupakan suatu usaha menguji parameter populasi melalui pengambilan *sample*. Pengujian hipotesis dapat dilakukan untuk satu populasi dan dua populasi. Adapun pengujian hipotesis untuk dua populasi di antara lain:

1. **Berpasangan**

   Pengujian sampel berpasangan adalah percobaan pada sampel dengan subjek yang sama namun mengalami dua perlakuan yang berbeda. Hasil dari percobaan lalu akan dibandingkan untuk menunjukkan suatu hubungan.

   Sebagai contoh adalah dilakukan percobaan pada subjek A. Dilakukan percobaan pertama dimana A tidak diberikan perlakuan apa-apa, sedangkan pada percobaan kedua diberikan perlakuan. Hasil dari percobaan pertama dan kedua lalu akan dibandingkan untuk mengambil suatu hubungan sebab-akibat.

2. **Independen**

   Pengujian sampel independen adalah percobaan pada subjek yang sama namun dengan perlakuan yang masing-masing berbeda dan tidak memiliki hubungan (Anderson, Sweeney, & Williams., 2008).

   Sebagai contoh adalah pada pengujian perbedaan skor prestasi dari penggunaan dua metode belajar yang berbeda. Hasil dari kedua pengujian tersebut lalu akan dibandingkan untuk diketahui apakah terdapat perbedaan yang signifikan.

   Pada jenis pengujian ini, dilakukan dua pengukuran pada subjek yang sama dengan perlakuan yang berbeda, namun tidak terdapat hubungan di antara perlakuan yang berbeda tersebut. Perhitungannya adalah:

1. Untuk σ_1^2 (varians kelompok 1) dan σ_2^2 (varians kelompok 2) diketahui:

Tabel 2.6 Statistik Uji untuk Varians Diketahui

Hipotesis	Statistik Uji	Nilai Kritis
$H_0: \mu_1 - \mu_2 \geq 0$ $H_1: \mu_1 - \mu_2 < 0$	$z = \frac{(\bar{x}_1 - \bar{x}_2) - (\mu_1 - \mu_2)}{\sqrt{\sigma_1^2 / n_1 + \sigma_2^2 / n_2}}$	$z \leq -z_\alpha$
$H_0: \mu_1 - \mu_2 \leq 0$ $H_1: \mu_1 - \mu_2 > 0$		$z \geq z_\alpha$
$H_0: \mu_1 - \mu_2 = 0$ $H_1: \mu_1 - \mu_2 \neq 0$		$z \geq z_{\alpha/2}$ *or* $z \leq -z_{\alpha/2}$

2. Untuk σ_1^2 dan σ_2^2 tidak diketahui:

   a. $\sigma_1^2 = \sigma_2^2$

Tabel 2.7 Statistik Uji untuk Varians Tidak Diketahui dan Dianggap Sama

Hipotesis	Statistik Uji	Nilai Kritis
$H_0: \mu_1 - \mu_2 \geq 0$ $H_1: \mu_1 - \mu_2 < 0$	$t = \frac{(\bar{x}_1 - \bar{x}_2)}{s_p \sqrt{1/n_1 + 1/n_2}}$ $s_p^2 = \frac{(n_1 - 1)s_1^2 + (n_2 - 1)s_2^2}{n_1 + n_2 - 2}$	$t \leq -t_{\alpha;n_1+n_2-2}$
$H_0: \mu_1 - \mu_2 \leq 0$ $H_1: \mu_1 - \mu_2 > 0$		$t \geq t_{\alpha;n_1+n_2-2}$
$H_0: \mu_1 - \mu_2 = 0$ $H_1: \mu_1 - \mu_2 \neq 0$		$t \geq t_{\alpha/2;n_1+n_2-2}$ *or* $t \leq -t_{\alpha/2;n_1+n_2-2}$

   b. $\sigma_1^2 \neq \sigma_2^2$

Tabel 2.8 Statistik Uji untuk Varians Tidak Diketahui dan Berbeda

Hipotesis	Statistik Uji	Nilai Kritis
$H_0 : \mu_1 - \mu_2 \geq 0$ $H_1 : \mu_1 - \mu_2 < 0$	$t = \frac{(\bar{x}_1 - \bar{x}_2)}{\sqrt{s_1^2/n_1 + s_2^2/n_2}}$ $v = \frac{s_1^2/n_1 + s_2^2/n_2}{\frac{(s_1^2/n_1)^2}{n_1 - 1} + \frac{(s_2^2/n_2)^2}{n_2 - 1}}$	$t \leq -t_{\alpha;v}$
$H_0 : \mu_1 - \mu_2 \leq 0$ $H_1 : \mu_1 - \mu_2 > 0$		$t \geq t_{\alpha;v}$
$H_0 : \mu_1 - \mu_2 = 0$ $H_1 : \mu_1 - \mu_2 \neq 0$		$t \geq t_{\alpha/2;v}$ *or* $t \leq -t_{\alpha/2;v}$

Untuk melakukan uji hipotesis terdapat beberapa tahap:

1. Menentukan hipotesis nol (H_0) dan alternatif (H_1): dalam tahap ini dilakukan pemilihan hipotesis yang akan dijadikan tujuan. Hipotesis yang ingin dibuktikan oleh peneliti ditempatkan di H_1, Sedangkan H_0 adalah lawan dari H_1.

2. Menentukan taraf signifikansi (α): Semakin kecil nilai taraf signifikansi, maka semakin besar persentase kebenaran hipotesis.

3. Menentukan statistik uji: pada proses ini dilakukan perhitungan dengan menggunakan rumus yang nantinya akan dibandingkan dengan nilai kritis. Statistik uji ini digunakan untuk menentukan keputusan ditolak atau diterimanya H_0.

4. Menentukan daerah kritis: hasil yang diperoleh dari perhitungan statistik uji lalu dibandingkan dengan nilai kritis untuk diketahui kebenaran hipotesis.

5. Mengambil kesimpulan: tolak atau terima H_0.

2.17 Hasil Penelitian Terkait Sebelumnya

2.17.1 Facebook Linked Data via the Graph API

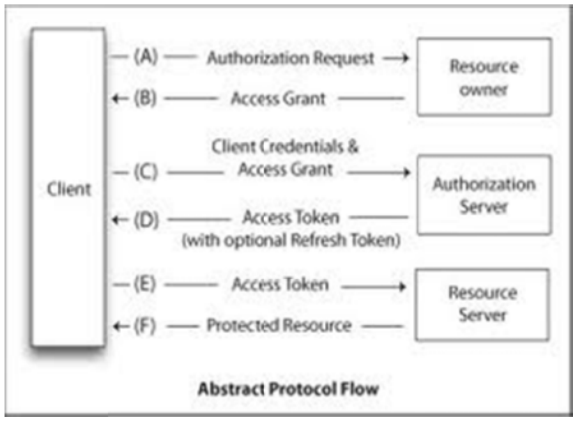
Dalam penelitian ini, penulis menjelaskan translasi dari *format* JSON menjadi *Turtle output*. Facebook Graph API menyajikan representasi sederhana dan konsisten dari *social graph* milik Facebook.

Facebook Graph merepresentasikan objek yang berada dalam *social graph* dan koneksi antara tiap objek. Graph API ini menyediakan data dalam *format* pertukaran data JSON secara eksklusif.

Sebelum aplikasi dapat mengakses informasi dalam *social graph*, protokol OAuth 2.0 harus digunakan terlebih dahulu (Weaver & Tarjan, 2012).

2.17.2 A Survey Paper on Social Sign-On Protocol OAuth 2.0

Penelitian ini menjelaskan tentang protokol OAuth 2.0 yang digunakan pada banyak jejaring sosial, salah satunya adalah Facebook. Protokol ini memperkenalkan *authorization layer* dan memisahkan peran antara *client* dan *resource owner*. Dengan begitu, terdapat keamanan yang lebih baik tanpa harus mengetahui rahasia yang dimiliki oleh masing-masing pihak.



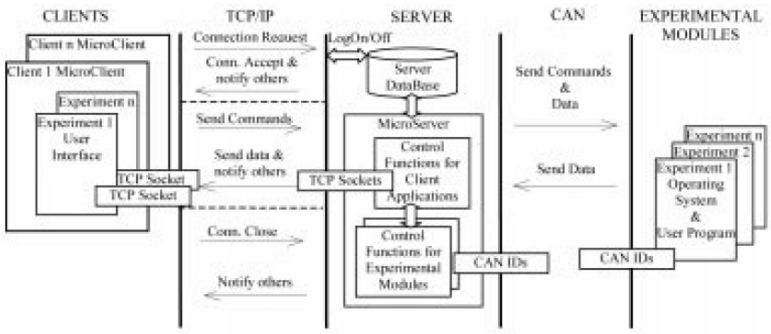
Gambar 2.26 Diagram Alur Protokol OAuth 2.0

(Sumber: *A Survey Paper on Social Sign-On Protocol OAuth 2.0*, Kaur & Aggarwal, 2013)

Dalam protokol ini, diperkenalkan *access token*, yaitu suatu *credentials* yang digunakan untuk mengakses *protected resources* yang dimiliki oleh *resource server* (Kaur & Aggarwal, 2013).

2.17.3 MicroLab: A Web-based Multi-user Remote Microcontroller Laboratory for Engineering Education

Penilitian ini mendiskusikan implementasi dari suatu remote laboratorium mikrokontroler berbasis web dengan kapabilitas *multi-user* yang dirancang untuk edukasi dalam bidang teknik elektro dan elektronik lainnya. MicroLab dikembangkan untuk menyediakan pemrograman dan pemantauan modul mikrokontroler melalui internet untuk mahasiswa. Mahasiswa dapat mengakses MicroLab secara individual dan secara bersamaan.



Gambar 2.27 Diagram *software architecture* dari MicroLab

(Sumber: *MicroLab: A Web-based Multi-user Remote Microcontroller Laboratory for Engineering Education*, Kutlu, 2004)

Fokus dari hasil penelitian ini berfokus pada *software architecture* yang telah dibuat. Terdapat pembagian *layer* dari sistem. Mulai dari *clients* yang memiliki peran sebagai pengguna dari sistem, *server* yang melakukan kontrol terhadap setiap *client* yang terhubung pada sistem, dan *experimental modules* yang merupakan *hardware* yang dapat diakses oleh *client*. Sehingga,

terdapat mediasi antara *client* dan *experimental modules* melalui *server* (Kutlu, 2004).