BAB II

KAJIAN TEORI

Kumpulan resep masakan pada umumnya memuat berbagai macam makanan yang siap untuk dimasak. Makanan merupakan elemen penting yang membentuk suatu resep karena makanan adalah produk hasil akhir yang ingin dicapai melalui panduan yang terdapat dalam resep masakan. Resep masakan kemudian dijadikan konten utama dalam pengembangan yang dilakukan oleh penulis dengan menggunakan Android sebagai wadah bagi aplikasi atau perangkat lunak tersebut. Pengembangan resep masakan ini dilakukan berdasarkan tahapan *System Development Life Cycle* (SDLC) yang berlaku secara umum dalam dunia *software developing*.

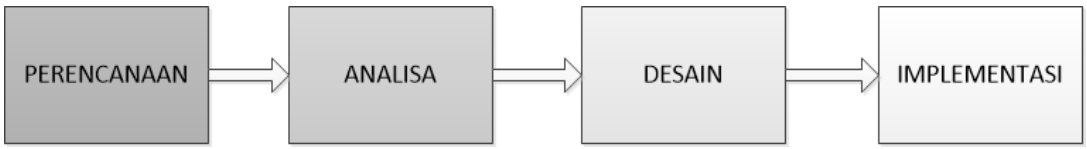
2.1 Pengembangan Aplikasi Perangkat Lunak

Pengembangan perangkat lunak pada umumnya sama dengan membangun sebuah gedung. Proses pertama dimulai dengan sebuah ide atau gagasan, lalu dilanjutkan dengan merubah ide atau gagasan tersebut menjadi sebuah desain sehingga dapat memberi gambaran lebih jelas tentang ide tersebut. Proses selanjutnya bersifat opsional, yakni melakuan revisi atau perubahan terhadap desain apabila dirasa tidak sesuai dengan yang diinginkan. Setelah itu, dilanjutkan dengan melakukan pembuatan cetak biru terhadap desain tersebut sehingga memperoleh gambaran yang lebih mendetail lagi dari masing-masing bagian gedung tersebut. Proses yang terakhir adalah membangun gedung tersebut berdasarkan cetak biru yang ada [9].

Dalam proses pengembangan aplikasi perangkat lunak, terdapat istilah SDLC atau *System Development Life Cycle*. SDLC adalah sebuah metode atau proses untuk memberikan pengertian bagaimana sebuah sistem informasi dapat mendukung kebutuhan bisnis dengan melakukan desain sebuah sistem, membangun sistem tersebut,

dan mengimplementasikannya ke pengguna [7].

SDLC memiliki 4 tahapan utama yakni perencanaan, analisa, desain, dan implementasi. Dalam sebuah proyek pengembangan aplikasi perangkat lunak pastilah memiliki keempat tahapan tersebut, namun cara untuk melaksanakan tahapan-tahapan tersebut dapat berbeda-beda antara satu proyek dengan proyek lainnya. Setiap tahapan tersebut memiliki langkah-langkah tersendiri dalam menjalankan tahapan-tahapan tersebut sesuai dengan tujuan akhir dari masing-masing tahapan tersebut.



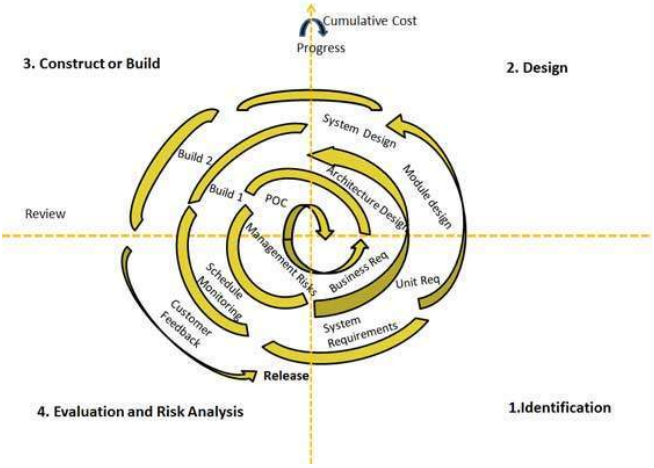
Gambar 2.1: Empat Tahapan Utama SDLC

Jenis-jenis pelaksanaan dari tahapan-tahapan SDLC terdiri dari [15]:

1. *Linear Process Flow*, yang mengeksekusi 4 tahapan tersebut secara berurutan
2. *Iterative Process Flow*, yang mengeksekusi secara berulang satu atau beberapa tahapan sebelum melanjutkan ke tahapan selanjutnya
3. *Evolutionary Process Flow*, mengeksekusi tahapan tersebut secara melingkar
4. *Parallel Process Flow*, mengeksekusi satuu atau beberapa tahapan secara bersamaan dengan tahapan lain.

Dalam metode SDLC terdapat beberapa model pengembangan yang dapat dijadikan acuan dasar kerangka kerja dalam proses pengembangan sebuah aplikasi perangkat lunak, yakni *Waterfall Model*, *Iterative Model*, *Spiral Model*, *V-Model*, *Big Bang Model*, *Agile Model*, *Rapid Application Development (RAD) Model*, dan *Prototyping Model*. Dalam pengembangan aplikasi resep masakan ini, penulis memilih menggunakan *Spiral Model*.

Model spiral memadukan sistematika dari pembangunan antara model iteratif (*Iterative Model*) dengan aspek pengendalian dari model *waterfall* (*Waterfall Model*). Model spiral termasuk dalam jenis *Evolutionary Process Flow*, yang mengeksekusi tahapan pengerjaan secara melingkar [3]. Tahapan-tahapan yang akan dilakukan dalam penelitian ini mengikuti model pengembangan spiral, yaitu: *Identification* (identifikasi kebutuhan), *Design* (Desain Sistem), *Construct and Build* (Pembuatan Sistem), dan *Evaluation* (*delivery feedback and release*)



Gambar 2.2: Model Spiral

Penulis memilih menggunakan model spiral sebagai model dari SDLC dalam proses pengembanga aplikasi kumpulan resep masakan terkoneksi *food channel* YouTube dengan alasan apabila dalam proses pengembangan terdapat kesalahan, baik dalam proses identifikasi, desain, atau pembuata sistem, maka penulis dapat kembali ke tahap dimana ditemukan kesalahan untuk kemudian memperbaiki kesalahan tersebut sehingga proses pegembangan dapat dilaksanakan secara semestinya dan penulis juga tidak melanggar ketentuan yang telah ditetapkan oleh metode pengembangan SDLC karena fleksibilitas dari model spiral itu sendiri.

2.2 *Unified Modeling Language* (UML)

Unified Modeling Language atau UML adalah sebuah Bahasa pemodelan terstandarisasi dan digunakan untuk berbagai kebutuhan dalam memodelkan pembuatan *software*yang berorientasi pada obyek (*object-oriented*). Standar tersebut dbuat dan dikelola oleh Object Management Group (OMG) pada tahun 1997 dan telah menjadi standar di bidang industri pembuatan *software*. UML meliputi sebuah set dari teknik notasi grafis untuk membuat model dari sebuah sistem perangkat lunak berorientasi obyek secara visual. UML juga merupakan peralatan untuk menspesifikasikan dan memvisualisasikan sebuah sistem pada perangkat lunak. Hal tersebut mencakup tipe diagram yang telah terstandarisasi untuk mendeskripsikan serta memetakan sebuah aplikasi komputer atau desain serta struktur dari sebuah sistem basis data. UML memiliki fungsi untuk mengelola sistem yang besar dan kompleks. Memiliki struktur aplikasi yang jelas dapat membantu *developer* dalam mengenalkan proyek kepada orang-orang baru maupun awam [10].

Jenis UML yang digunakan oleh peneliti pada pengembangan aplikasi ini adalah *Use Case Diagram*, *Class Diagram*, dan *Activity Diagram*.

2.2.1 *Use Case Diagram*

Fungsionalitas dari sebuah sistem basis data maupun sebuah perangkat lunak komputer dapat diilustrasikan dengan diagram *use case*. Tugas utama dari sebuah *use case* adalah memvisualisasikan kebutuhan fungsionalitas dari sebuah sistem, termasuk hubungan antara aktor (orang yang akan berinteraksi dengan sistem) dengan proses-proses yang esensial dalam sebuah sistem untuk mencapai suatu tujuan. Hal tersebut dapat dikategorikan sebagai kumpulan langkah-langkah yang mendefinisikan interaksi antara aktor dan sistem [5].

Use Case Diagram terdiri dari beberapa komponen, yakni:

1. Aktor

   Menggambarkan seorang aktor atau pengguna sistem yang akan berinteraksi dengan sistem

Gambar 2.3: Aktor dalam *Use Case*

2. Oval

   Bentuk Oval menggambarkan sistem yang dapat berinteraksi dengan aktor. Bentuk Oval tersebut biasanya disertai dengan nama aktivitas sistem di dalam-nya yang terletak ditengah-tengah bentuk oval tersebut

Gambar 2.4: Sistem dengan Bentuk Oval

3. Garis Penghubung Aktor dan Sistem

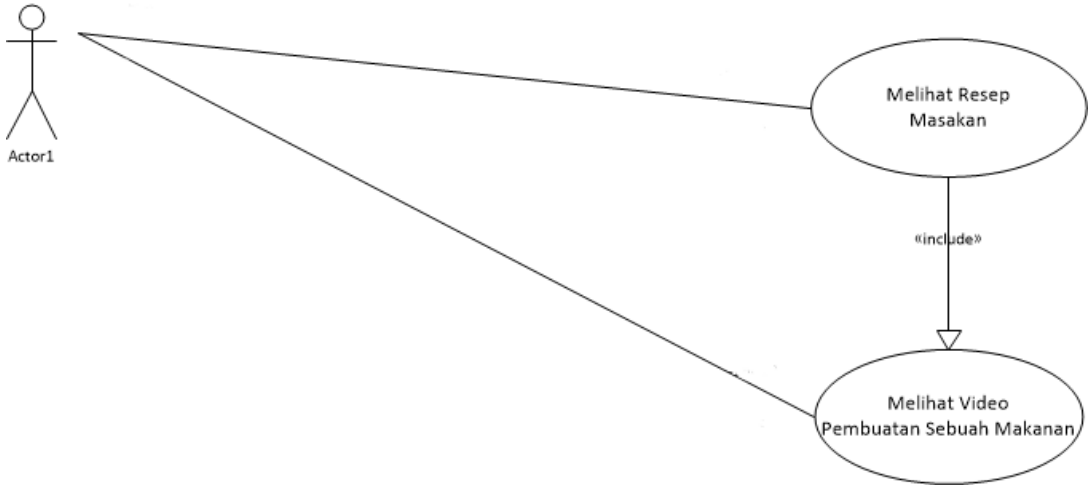
   Untuk menggambarkan interaksi antara Aktor dan Sistem, diperlukan sebuah garis yang menghubungkan antara aktor dan sistem. Sebuah garis digunakan untuk menghubungkan keduanya

Gambar 2.5: Garis Penghubungan antara Aktor dan Sistem

Gambar 2.5 memperlihatkan garis yang menyambungkan aktor dengan interaksi sistem menyatakan bahwa Aktor dapat Melihat Resep Masakan

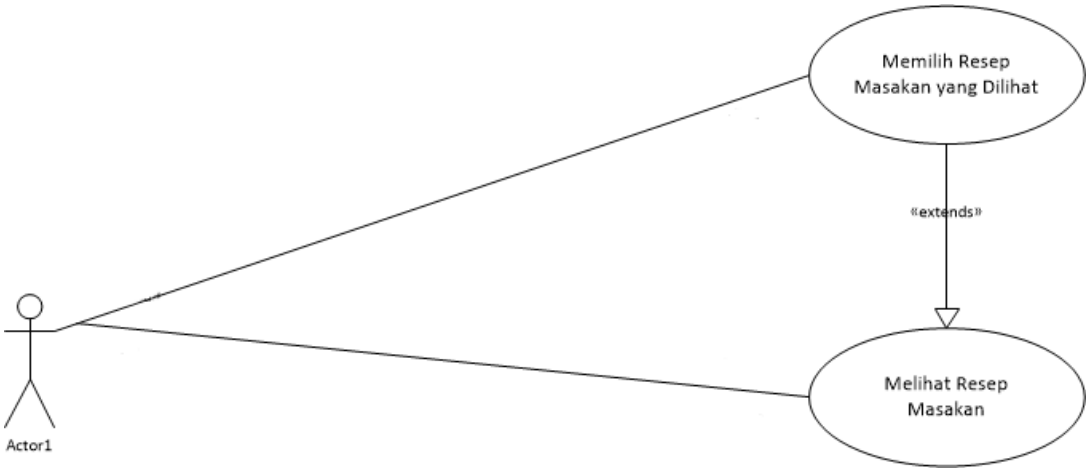
4. Garis Penghubung Antar Sistem

   Sistem yang terdapat pada *use case* dapat berinteraksi satu sama lain. Terdapat dua jenis hubungan antar sistem, yakni *include* dan *extend*. Sesuai dengan namanya, *include* menyatakan bahwa sebuah interaksi sistem yang terhubung adalah bagian dari interaksi sistem lainnya



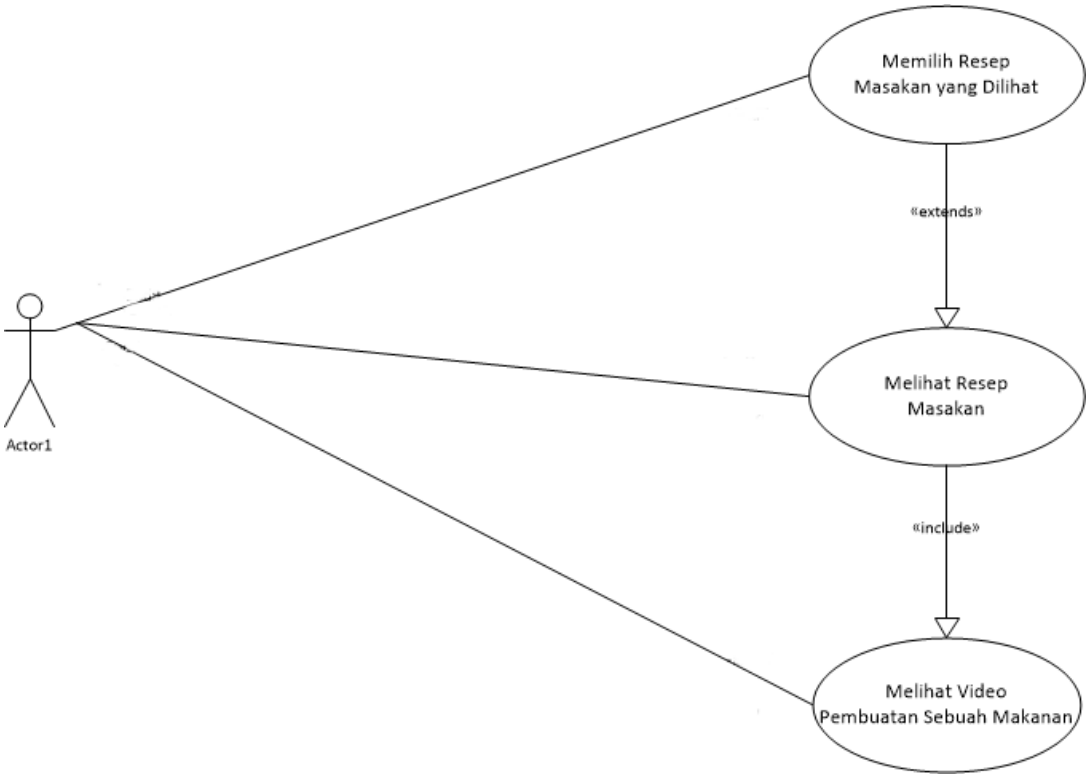
Gambar 2.6: Contoh Hubungan *Include*

Gambar 2.6 menjelaskan bahwa Melihat Video Pembuatan Sebuah Makanan merupakan bagian dari Melihat Resep Masakan. Sedangkan penghubung *extend* menyatakan bahwa interaksi sebuah sistem merupakan perluasan dari interaksi sistem lainnya



Gambar 2.7: Contoh Hubungan *Extend*

Gambar 2.7 menjelaskan bahwa Melihat Resep Masakan merupakan perluasan dari Memilih Resep Masakan yang Dilihat. Sebagai contoh, seorang Aktor dapat berinteraksi dengan sebuah sistem berbentuk aplikasi resep masakan. Interaksi yang dimungkinkan oleh sistem adalah Memilih Resep Masakan, Melihat Resep Masakan, dan Melihat Video Pembuatan Sebuah Resep Masakan. *Use Case* dari interaksi tersebut terdapat pada Gambar 2.8. Gambar tersebut menjelaskan bahwa aktor dapat memilih resep masakan dan kemudian juga melihat resep masakan yang telah dipilih. Melihat resep masakan yang dipilih merupakan ekstensi dari interaksi antara akor denga sistem yaki memilih resep masakan yang dilihat. Kemudian pengguna juga dapat melihat video pembuatan sebuah makanan ketika melihat resep masakan. Hal tersebut dimungkinkan karena interaksi melihat resep masakan memiliki hubungan *include* dengan innteraksi melihat video pembuatan sebuah makanan.



Gambar 2.8: Contoh *Use Case* dalam Penggunaan Aplikasi Resep Masakan

Jadi, dapat disimpulkan bahwa Use Case Diagram adalah sebuah diagram yang menggambarkan interaksi antara Aktor (pengguna) dengan sistem yang digunakan oleh Aktor.

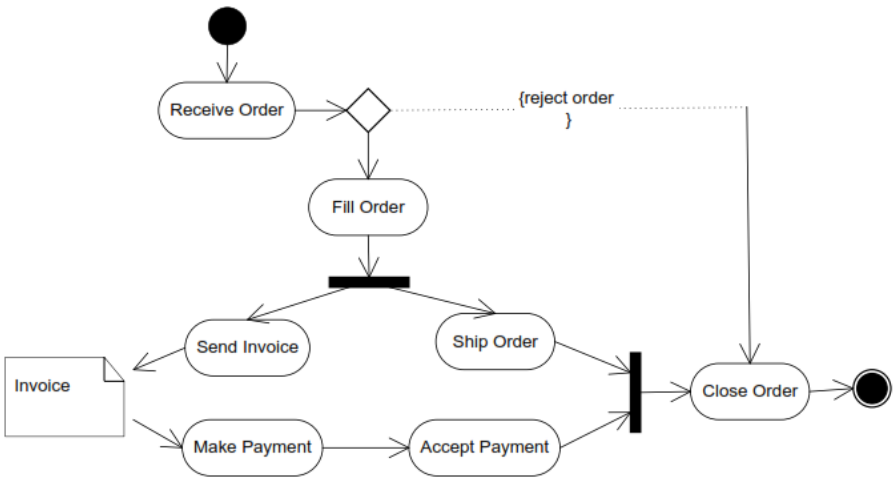
2.2.2 *Class Diagram*

Struktur statis dari sebuah aplikasi komputer atau pangkalan basis data dapat digambarkan melalui sebuah *class diagram*. *Class diagram* juga menunjukkan perbedaan entitas (pengguna, alat, maupun data) yang berelasi satu sama lain. Diagram tersebut biasa digunakan untuk menampilkan class-class secara logikal, dan untuk mengimplementasikan class yang telah dibuat. *Class diagram* merupakan unsur terpenting dalam pemodelan aplikasi berbasis obyek (*object-oriented*) dan merupakan sebuah tipe struktur diagram statis yang mendeskripsikan struktur dari sebuah sis-

tem dengan menampilkan *class* dari sistem, atribut, operasi (method), dan relasi antar *class* [10].

2.2.3 *Activity Diagram*

Kontrol alur serta prosedur antara dua atau lebih obyek *class* dalam memproses sebuah aktivitas dapat ditunjukkan melalui sebuah *activity diagram*. Diagram tersebut dapat digunakan untuk memodelkan proses bisnis tingkat atas dalam sebuah level unit bisnis, atau untuk memodelkan aksi-aksi dalam class internal yang bersifat *low level* (Bell, 2003). *Activity Diagram* adalah representasi grafis dari sebuah alur pengerjaan yang berupa tahap-tahap dari berbagai aktivitas serta aksi dengan dukungan pilihan, iterasi, dan persetujuan. Pada UML, *activity diagram* dapat digunakan untuk mendeskripsikan bisnis serta langkah-langkah operasional serta alur pengerjaan dari berbagai komponen yang berada dalam sebuah sistem. Sebuah *activity diagram* menjelaskan keseluruhan alur dari sebuah sistem [10]. Contoh dibawah ini menjelaskan aktivitas pemrosesan pesanan dalam sebuah sistem Point of Sales (POS).



Gambar 2.9: Contoh *Activity Diagram* dalam sistem Point Of Sales (POS)

2.3 *Structured Query Language* (SQL)

SQL merupakan singkatan dari *Structure Query Language*, didefinisikan sebagai suatu sintaks perintah-perintah tertentu atau bahasa program yang digunakan untuk mengelola suatu *database* [1]. 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. Query dikirimkan ke database dalam bentuk SQL Query Beberapa perintah yang umum digunakan adalah sebagai berikut:

1. CREATE : Untuk membuat tabel baru

```
CREATE TABLE <NAMA TABLE>
(<NAMA KOLOM> <TIPE>,
<NAMA KOLOM> <TIPE>,
. . . . . .
PRIMARY KEY (<NAMA KOLOM>)
FOREIGN KEY (<NAMA KOLOM>)
REFERENCES <NAMA_TABLE>
<NAMA KOLOM>)
```

2. SELECT : Untuk mengambil *record* dari database yang memenuhi kriteria tertentu

```
SELECT <NAMA KOLOM>,
 <NAMA KOLOM>,
 . . .
FROM <NAMA  TABEL>
WHERE <KONDISI>
```

3. INSERT : Untuk menambah *record* ke dalam suatu tabel

```
INSERT INTO <NAMA TABEL>
            (<NAMA KOLOM>,
             <NAMA KOLOM> )
VALUES (<NILAI KOLOM>, <NILAI
KOLOM>, . . .)
```

4. UPDATE : Untuk merubah isi *record* tertentu pada suatu tabel

```
UPDATE <NAMA TABEL>
SET (<NAMA KOLOM> = <NILAI
KOLOM>,<NAMA KOLOM> = <NILAI
KOLOM>,    . . . )
WHERE <KONDISI>
```

5. DELETE : Untuk menghapus *record* pada suatu tabel

```
DELETE FROM <NAMA TABEL>
WHERE <KONDISI>
```

6. DROP : Untuk menghapus sebuah tabel

```
DROP <NAMA TABEL>
```

7. JOIN : Untuk menggabungkan dua atau lebih tabel

```
JOIN <NAMA TABEL>
    ON <NAMA TABEL>.<ID> = <NAMA TABEL KEDUA>.<ID>
```

2.4 *Entity Relationship Diagram* (ERD)

Entity Relationship Diagram (ERD) adalah sekumpulan cara atau peralatan untuk mendeskripsikan data-data atau objek-objek yang dibuat berdasarkan dan berasal dari dunia nyata yang disebut entitas (*entity*) serta relasi (*relationship*) antar entitas-entitas tersebut dengan menggunakan beberapa notasi [8]. Komponen-komponen pembentuk ERD dapat dilihat pada Tabel 2.1.

Notasi	Komponen	Keterangan
	Entitas	Individu yang mewakili suatu objek dan dapat dibedakan dengan objek yang lain.
	Atribut	Properti yang dimiliki oleh suatu entitas, dimana dapat mendeskripsikan karakteristik dari entitas tersebut.
	Relasi	Menunjukkan hubungan diantara sejumlah entitas yang berbeda.
	Relasi 1 : 1	Relasi yang menunjukkan bahwa setiap entitas pada himpunan entitas pertama berhubungan dengan paling banyak satu entitas pada himpunan entitas kedua
	Relasi 1 : N	Relasi yang menunjukkan bahwa hubungan antara entitas pertama dengan entitas kedua adalah satu banding banyak atau sebaliknya. Setiap entitas dapat berelasi dengan banyak entitas pada himpunan entias yang lain
	Relasi N : N	Hubungan ini menunjukkan bahwa setiap entitas pada himpunan entitas yang pertama dapat berhubungan dengan banyak entitas pada himpunan entitas yang kedua, demikian juga sebaliknya

Tabel 2.1: Komponen-komponen Pembentuk ERD

Sesuai dengan namanya, sebuah ERD memodelkan data sebagai entitas serta relasi (*relationship*). Entitas adalah data yang ingin disimpan. Chen (1976) dalam presentasinya mendeskripsikan entitas sebagai "sesuatu yang dapat diidentifikasi secara jelas." Jadi, entitas dapat berupa seseorang, sebuah tempat, obyek, kejadian, atau konsep yang ingin disimpan sebagai data.

Relasi (*relationship*) adalah koneksi antara entitas-entitas yang ada. Relasi sendiri memiliki kardinalitas, yaitu ukuran kasar dari jumlah entitas (satu atau lebih) yang akan terkait dengan entitas lain (atau entitas). Berdasarkan kardinalitasnya, terdapat tiga jenis relasi yang dapat ditemukan dalam entitas-entitas yang ada yaitu relasi *one-to-one* (1:1), *one-to-many* (1:M), dan *many-to-many* (M:N). Dalam relasi *one-to-one*, sebuah entitas dapat berasosiasi dengan satu entitas lainnya dan sebalik-

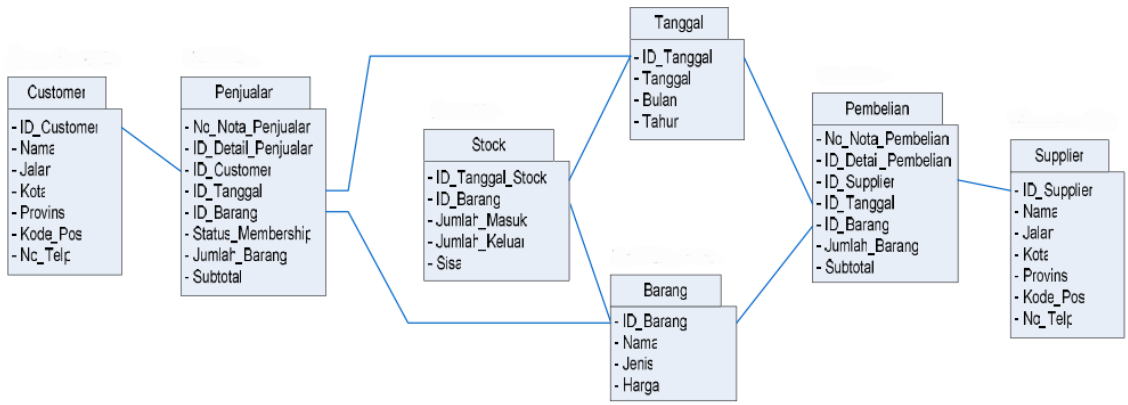
nya. Contohnya, seorang siswa hanya memiliki satu sepeda. Dengan analogi antara siswa dengan sepeda tersebut, relasi *one-to-many* menggambarkan bahwa seorang siswa dapat memiliki banyak sepeda. Sedangkan relasi *many-to-many* pada analogi siswa dengan sepeda adalah banyak siswa yang dapat memiliki banyak sepeda dan sebaliknya [4].

Atribut adalah kategori dari sebuah data yang mendeskripsikan sebuah entitas atau relasi. Contoh dari atribut adalah sebuah entitas bernama MOBIL memiliki atribut tipe, warna, id_kendaraan, dan lain sebagainya. Basis data sering digunakan untuk menyimpan data yang akan digunakan pada tahapan-tahapan selanjutnya. Sebuah atribut yang dapat digunakan untuk mencari suatu entitas tertentu disebut kunci atau *key*. Atribut yang bersifat kunci (*key*) dapat muncul secara natural apabila sebuah basis data dimodelkan dengan menggunakan ERD. Jika sebuah atribut dapat diperhitungkan menjadi sebuah ciri khas dari suatu entitas karena keunikannya, maka atribut tersebut disebut dengan *candidate key* atau kandidat kunci. *Candidate key* akan berubah menjadi *primary key* apabila telah dipilih sebagai atribut khas suatu entitas yang bersifat paling unik daripada atribut lain yang terdapat pada suatu entitas.

Studi Kasus Minimarket

Sebuah minimarket yang menjual berbagai jenis barang kebutuhan sehari-hari memiliki sebuah sistem informasi untuk mengelola penjualan secara langsung (point of sales), pengadaan barang, dan *stock control*. Proses bisnis dalam penjualan barangnya dimulai pada saat *customer* memilih barang yang akan dibeli. Setelah *customer* memutuskan untuk membeli barang tersebut, maka kasir akan meminta informasi tentang identitas *customer* untuk dicatat jika *customer* yang bersangkutan terdaftar sebagai member. Namun jika *customer* tersebut bukanlah member minimarket, maka data-data *customer* akan diabaikan. Kemudian kasir akan membuatkan nota penjual-

an barang. Setelah barang diterima oleh *customer*, *customer* akan melakukan pembayaran. Proses berakhir ketika kasir memberikan bukti pembayaran kepada *customer*. Sistem informasi yang tersedia tidak melayani proses pengembalian barang dan pemesanan barang. Proses bisnis untuk pembelian barang dari supplier dimulai ketika pihak minimarket menghubungi supplier dan memesan barang. *Supplier* kemudian akan membuatkan nota pembelian. Barang yang sudah dipesan lalu akan diantarkan ke minimarket. Jika barang sudah diterima, maka proses yang terjadi adalah pembayaran dari pihak minimarket ke pihak *supplier*. Setelah semua proses pembayaran selesai, *supplier* akan memberikan bukti pembayaran dan proses selesai. Seperti halnya pada proses penjualan, proses pembelian tidak menangani pengembalian barang kepada *supplier*. Untuk proses *stock control*, dilakukan proses pencatatan terhadap barang yang di- supply, barang yang dibeli oleh *customer* dan sisa barang yang ada di gudang per harinya. Hal ini dimaksudkan agar setiap keluar masuknya barang yang ada dapat terawasi dan menjaga barang selalu tersedia di gudang. Maka ERD yang dapat dibuat dari kasus diatas adalah sebagai berikut:

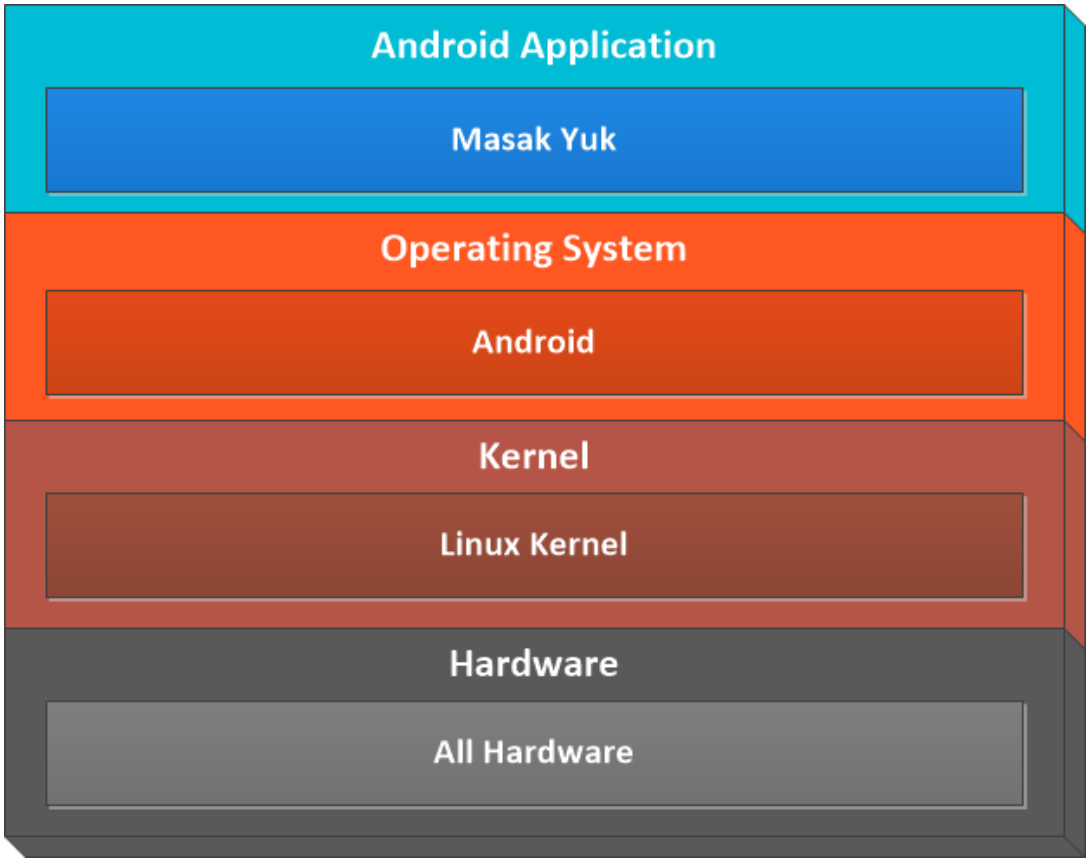


Gambar 2.10: ERD Studi Kasus Minimarket

2.5 Pengertian Android

Android adalah sebuah sistem operasi untuk perangkat *mobile* berbasis linux yang mencakup sistem operasi, *middleware* dan aplikasi. Android menyediakan *platform* terbuka bagi para pengembang untuk menciptakan aplikasi mereka [11]. Dikutip dari situs resmi Open Handset Alliance, yang merupakan pengembang sistem operasi Android pertama di dunia, Android dikembangkan untuk memungkinkan para pengembang aplikasi membuat suatu aplikasi *mobile* yang dapat memanfaatkan seluruh kelebihan yang ditawarkan dari suatu perangkat *mobile*. Android dibuat secara terbuka dan semua orang dapat mengembangkan dan menggunakan semua fungsionalitasnya. Contohnya, sebuah aplikasi yang dapat digunakan untuk melakukan berbagai macam fungsionalitas sebuah ponsel seperti membuat panggilan, mengirimkan pesan singkat, atau mengambil gambar dengan kamera, memungkinkan para pengembang aplikasi untuk membuat banyak sekali fitur lainnya yang dapat digunakan oleh para penggunanya. Android dibuat dengan menggunakan Kernel Linux yang bersifat terbuka (*open source*). Selain itu, Android juga merupakan hasil dari modifikasi mesin virtual (*virtual machine*) yang didesain untuk mengoptimalkan memori serta perangkat keras (*hardware*) pada sebuah ponsel. Android sendiri adalah open source yang berarti dapat terus dikembangkan menjadi sebuah teknologi baru sesuai dengan kebutuhan manusia pada saat ini. *Platform* ini akan terus berkembang seiring dengan pergerakan para pengembang aplikasi yang bekerja bersama-sama untuk membangun aplikasi *mobile* yang inovatif.

Gambar 2.11 menunjukkan hirarki aplikasi yang dibuat oleh peneliti terhadap *platform* Android.



Gambar 2.11: *Stack Diagram* Hirarki Aplikasi Android dengan Komponen Lainnya

Selain kemampuannya untuk mendukung dan memfasilitasi berbagai fungsi familiar seperti membuat panggilan, berkirim pesan elektronik, dan mencari sebuah restoran, perangkat Android dapat menjadi sebuah perangkat dengan fungsi tanpa batas dan dapat dikembangkan kapan saja bergantung pada pemikiran para pengembangnya [16]. Saat ini, Android tidak hanya terdapat pada sebuah ponsel tetapi juga tertanam dalam sebuah tablet, televisi, kacamata pintar, kulkas, mobil dan lain sebagainya. Jadi dapat disimpulkan bahwa Android merupakan sistem operasi untuk sebuah perangkat cerdas yang dapat memiliki banyak fitur dan fungsi sesuai dengan keinginan para pengembangnya untuk menjawab masalah-masalah yang terdapat dalam kehidupan manusia.

2.6 Android Studio

Android Studio merupakan sebuah *Integrated Developing Environment* (IDE) untuk *platform* Android. Android Studio ini diumumkan pada tanggal 16 Mei 2013 pada konferensi Google I/O oleh *Product Manager* Google, Ellie Powers. Android Studio bersifat *free* dibawah Apache *License* 2.0. Versi awal Android Studio yaitu 0.1 dirilis pada bulan Mei 2013. Kemudian dibuat versi *beta* 0.8 yang dirilis pada bulan Juni 2014. Versi yang paling stabil dirilis pada bulan Desember 2014, dimulai dari versi 1.0. Berbasiskan JetBrainns' IntelliJ IDEA, Studio di desain khusus untuk Android *Development*. Android Studio sudah bisa diunduh untuk Windows, Mac OS X, dan Linux

Berikut ini adalah beberapa keunggulan Android Studio:

1. Android *Memory* (HPROF) *Viewer*

   Android Studio kini mengizinkan penggunanya untuk menangkap dan menganalisa *snapshot* memori dalam format Android HPROF.

2. *Allocation Tracker*

   Untuk mempermudah dalam menganalisa penggunaan alokasi memori aplikasi yang dibuat oleh penggunanya, *Allocation Tracker* kini menambahkan fitur visual. Dengan adanya fitur ini, alokasi memori yang digunakan oleh aplikasi yang dibuat oleh penggunanya dapat terlihat dalam diagram berbentuk lingkaran.

3. APK *Test*

   Kini plugin baru ('com.android.test') telah ditambahkan untuk mempermudah penggunanya dalam mencoba atau melakukan *testing* aplikasi Android yang sedang dikembangkan. Untuk menggunakan fitur ini, penggunanya harus memiliki Gradle Plugin versi 1.3.

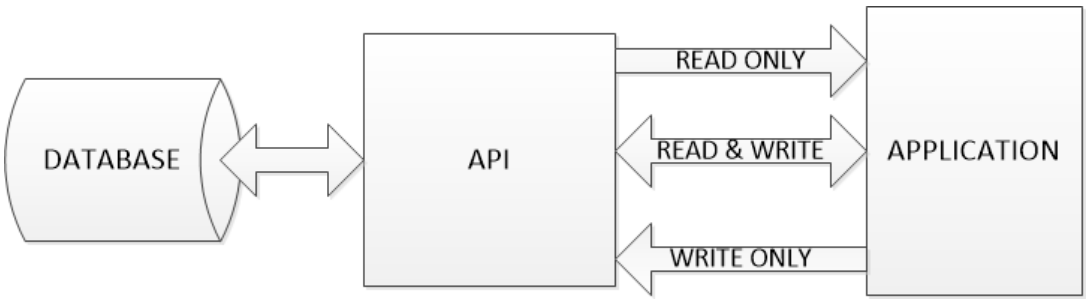
4. *Application Permission Annotation*

   Android Studio (sejak versi 1.3) mendukung *inline code annotation* yang akan membantu penggunanya mengatur *app permission* yang dirilis pada Android M (Android 6.0/Marshmallow).

5. SDK *Auto Update* dan SDK *Manager* Android Studio mengatur pembaharuan SDK dan plugin-plugin lainnya secara otomatis.

2.7 *Application Programming Interface* (API)

Application Programming Interfaces atau biasa disingkat API, yang didalamnya termasuk kumpulan *library*, *framework*, peralatan penunjang (*toolkits*), dan peralatan pengembangan *software* lainnya, digunakan secara virtual dalam semua bahasa pemrograman [12]. Jika sebuah perangkat lunak mengandung API *internal* (yang berasal dari dalam perangkat lunak tersebut) serta API publik (seperti Java Platform SDK, Windows .NET Framework, jQuery untuk JavaScript, dan Web services seperti Google Map), maka setiap baris dari kode yang ditulis oleh *programmer* memanggil dan menggunakan API. API menyediakan sebuah mekanisme dalam penggunaan kembali sebuah kode yang telah dibuat sebelumnya sehingga *programmer* dapat memanfaatkannya kembali untuk keperluan yang berbeda. Hal ini sangat efektif apabila dibandingkan dengan membuat setiap program dan kode dari awal. Lebih lanjut lagi, penggunaan API sangat dibutuhkan karena akses yang bersifat *low-level* menuju kepada sebuah sumber dari suatu sistem (seperti grafik atau gambar, *networking*, dan *file* sistem) yang tersedia hanya melalui API yang terproteksi.



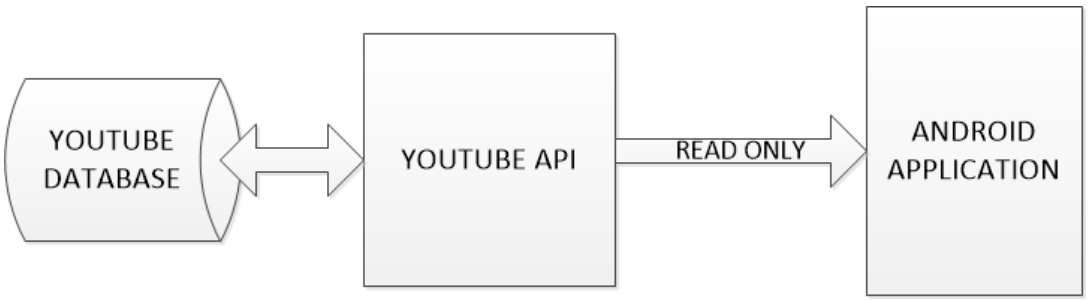
Gambar 2.12: Cara Kerja API

API menjembatani sebuah basis data dengan aplikasi yang mengakses basis data tersebut. Terdapat tiga jenis metode yang dihasilkan oleh API, yaitu metode *read only* (hanya membaca), *write only* (menulis atau mengubah saja), dan *read and write* (membaca dan menulis atau mengubah data). Ketiga jenis metode tersebut dibuat sesuai dengan kebutuhan aplikasi yang akan dijembatani oleh API.

2.8 Integrasi YouTube dengan Aplikasi Android

YouTube dapat diintegrasikan ke dalam sebuah aplikasi Android dengan meng-implementasikan API (*Application Programming Interface*) yang disediakan oleh YouTube dan Google secara gratis (*open source*) yang berarti semua orang dapat menggunakannya tanpa terkecuali. API tersebut dinamakan YouTube Android Player API.

Dilansir dari situs Google Developer, API tersebut memungkinkan penggunanya untuk memasukkan fungsi pemutaran video ke dalam aplikasi Android. API tersebut mendefinisikan metode untuk menampilkan dan memainkan video-video YouTube (dan juga *playlist*) dan untuk memodifikasi serta mengatur pemutaran video dalam aplikasi Android.



Gambar 2.13: Cara Kerja YouTube API

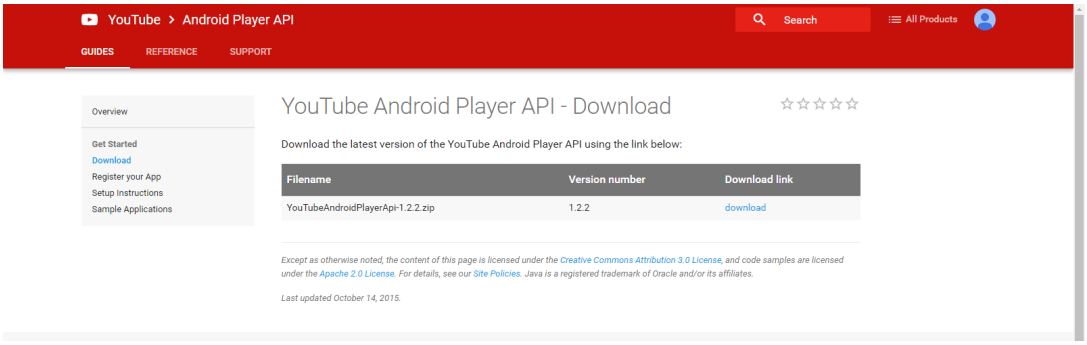
YouTube Android Player API termasuk dalam jenis API yang menyediakan *read only method*, yaitu API yang menyediakan metode untuk membaca data yang terdapat dalam sebuah basis data, dimana basis data yang digunakan pada aplikasi ini adalah basis data YouTube. Video dialirkan kedalam aplikasi Android menggunakan berbagai macam metode yang terdapat dalam API YouTube tersebut.

Dengan menggunakan API tersebut, pengguna dapat menampilkan berbagai video ke dalam sebuah pemutar yang terintegrasi pada *User Interface* (UI) penggunanya. Kemudian penggunanya juga dapat mengatur pemutaran video secara terprogram. Sebagai contoh, pengguna dapat memainkan, melakukan jeda, atau mempercepat video ke titik tertentu pada video yang sedang diputar

Pengguna juga dapat memasukkan *event listener* untuk mendapatkan *callbacks* dari beberapa event, seperti pemuatan video pada pemutar video atau perubahan tahap pada pemutar video. Dan yang terakhir, API tersebut memiliki *helper functionality* (fungsionalitas pembantu) untuk mendukung perubahan orientasi seperti transisi menjadi pemutaran dalam layar penuh (*fullscreen playback*).

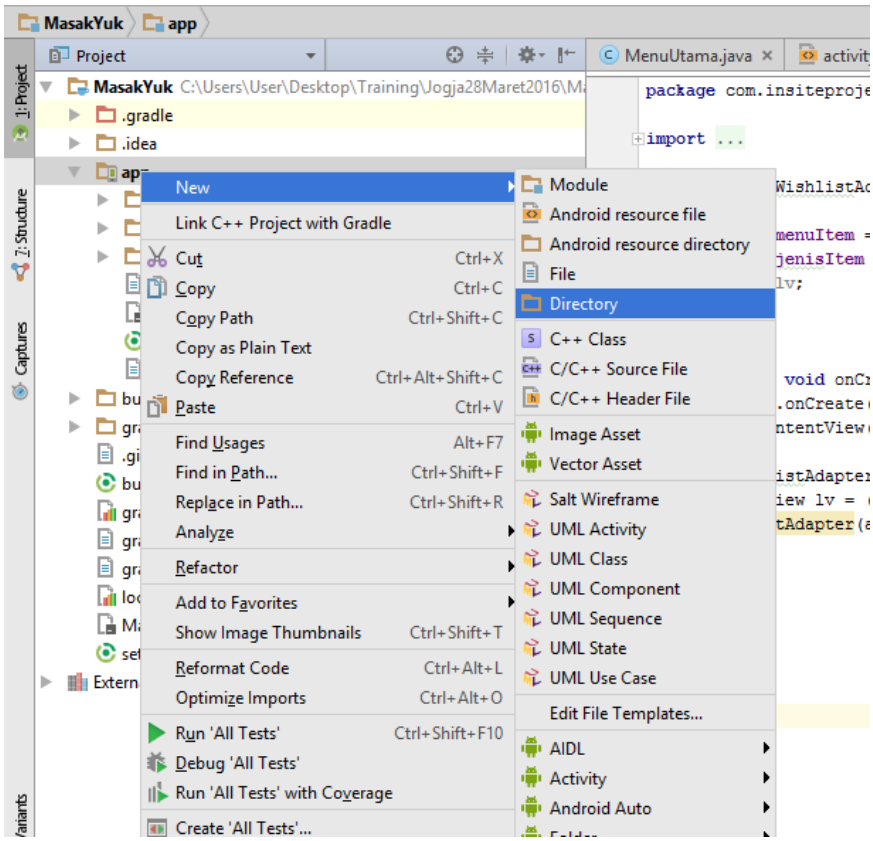
Langkah-langkah untuk menginisialisasi integrasi YouTube dengan aplikasi Android:

1. Unduh terlebih dahulu file YouTube Android Player API pada link `https://developers.google.com/youtube/android/player/downloads/`



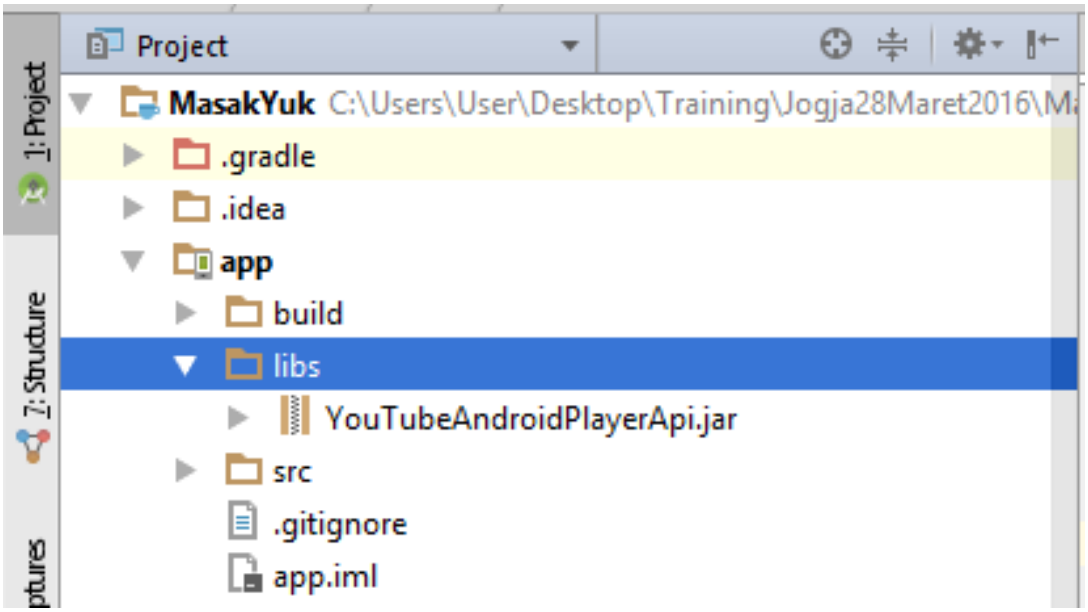
Gambar 2.14: Laman Unduh YouTube Player Android API

2. Kemudian pada Android Studio, ubah tampilan *tree* dari Android menjadi Project dan buat direktori baru bernama *libs* pada direktori *app*



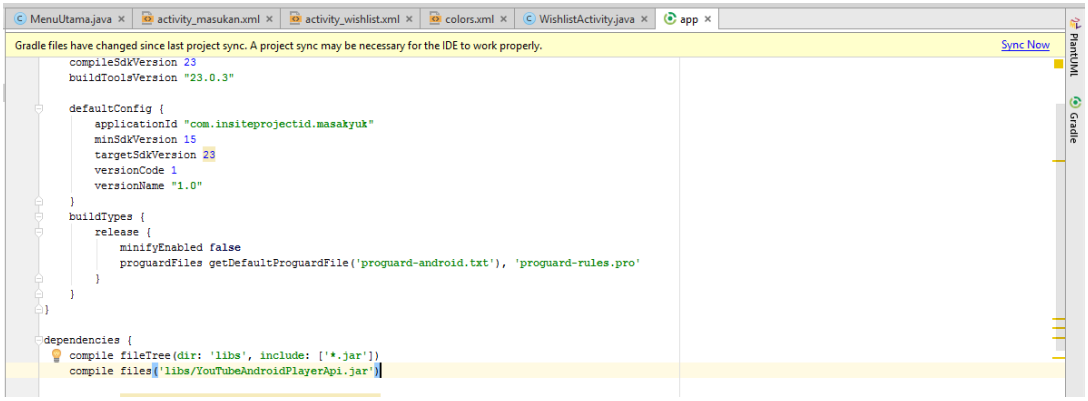
Gambar 2.15: Tampilan *tree* Project

3. Ekstrak berkas yang telah diunduh pada langkah pertama dan salin berkas YouTubeAndroidPlayerApi.jar pada direktori *libs*



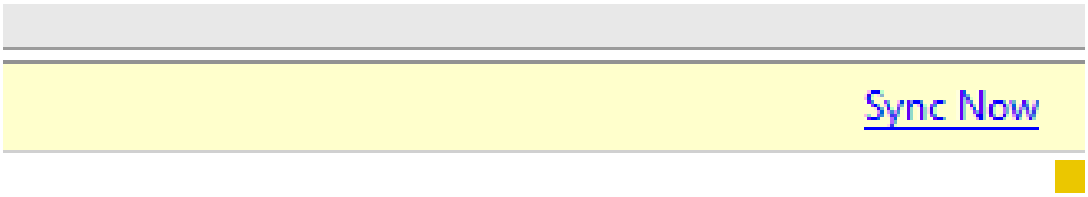
Gambar 2.16: Menyalin Berkas YouTube API

4. Tambahkan kode compile files('libs/YouTubeAndroidPlayerApi.jar') pada berkas build.gradle yang terdapat di dalam direktori *app*



Gambar 2.17: Menambah Baris Kode

5. Akan muncul sebuah pemberitahuan untuk melakukan sinkronisasi pustaka yang ada dengan pustaka YouTube API. Klik *Sync Now* pada pemberitahuan tersebut untuk menginisialisasikan Android dengan YouTube



Gambar 2.18: *Sync Now*

6. Buat sebuah Activity dan tambahkan baris kode yang sama dengan kode yang tertera di bawah ini

```
public class ResepActivity extends YouTubeBaseActivity implements YouTubePlayer.OnInitializedListener {
```

Gambar 2.19: Menambah Baris Kode pada Activity

7. Inisialisasikan YouTubePlayerView pada Activity tersebut

```
private YouTubePlayerView youTubeView;
```

Gambar 2.20: Inisialisasi YouTubePlayerView

8. Inisialisasikan juga YouTubePlayerView yang sudah disiapkan pada Activity tersebut ke dalam *method* onCreate.

```
youTubeView = (YouTubePlayerView) findViewById(R.id.youtube_view);
youTubeView.initialize(Config.YOUTUBE_API_KEY, this);
```

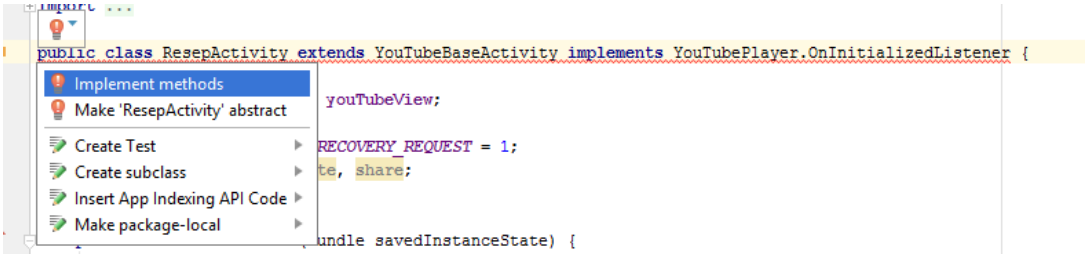
Gambar 2.21: Inisialisasi YouTubePlayerView ke dalam *method* onCreate

9. Pada *layout*, tambahkan baris kode di bawah ini untuk menginisialisasi tampilan YouTube

```
<com.google.android.youtube.player.YouTubePlayerView
    android:id="@+id/youtube_view"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"/>
```

Gambar 2.22: Inisialisasi Tampilan YouTube

10. Implementasikan *method* yang berasal dari API YouTube dengan melakukan klik pada lampu merah yang terdapat pada inisialisasi *class* Activity seperti yang terdapat di bawah ini



Gambar 2.23: Implementasi *Method* YouTube

11. Akan muncul dua *method* penting dalam mengintegrasikan YouTube ke dalam Android, yaitu onInitializationSuccess dan onInitializationFailure seperti di bawah ini

```
@Override
public void onInitializationSuccess(YouTubePlayer.Provider provider, YouTubePlayer youTubePlayer, boolean b) {

}

@Override
public void onInitializationFailure(YouTubePlayer.Provider provider, YouTubeInitializationResult errorReason) {

}
```

Gambar 2.24: Inisialisasi Tampilan YouTube

onInitializationSuccess adalah method yang berfungsi untuk menjalankan fitur-fitur YouTube di dalam aplikasi Android apabila inisialisasi sukses. Sedangkan