BAB II
TEORI PENUNJANG

2.1 Teori Tentang Binatang

Binatang adalah kelompok organisme yang diklasifikasikan dalam *Kingdom Animalia* (kerajaan hewan) atau metazoan. Binatang sering disebut juga fauna atau satwa.

Pada umumnya binatang atau hewan dapat kita bagi menjadi beberapa jenis berdasarkan makanannya, yaitu: 1. Herbivora adalah jenis hewan yang memakan makanan yang berasal dari tumbuh-tumbuhan seperti daun, batang, biji dan juga umbi-umbian. Contoh binatang herbivora pemakan rumput dan dedaunan misalnya sapi, kuda zebra, badak, gajah, rusa, dan kudanil. 2. Karnivora adalah jenis binatang yang makananya berasal dari tubuh hewan lainnya seperti daging dan darah. Binatang karnivora pada umumnya memiliki gigi seri dan gigi geraham. Gigi seri digunakan untuk memotong makanan sebelum dikunyah, sedangkan gigi geraham dengan permukaan yamg luas digunakan untuk mengunyah makanan. Ada beberapa binatang yang tergolong kedalam karnivora misalnya singa, buaya, beruang, dan macan tutul. 3. Omnivora adalah binatang yang memakan keduanya baik itu tumbuhan maupun daging hewan lainnya, contohnya ayam, tikus, bebek, ikan, babi hutan, dan rubah.

Dari berbagai macam binatang yang ada maka diambil 10 binatang diantanya adalah badak, buaya, beruang, gajah, kuda nil, rusa, rubah, sapi, singa, zebra. Dari binatang yang dipilih sudah mewakili binatang herbivora, karnivora, dan omnivora dan binatang tersebut merupakan binatang yang mudah dijumpai oleh anak-anak. setelah menentukan binatang, maka dibuatlah aplikasi pembelajaran hewan menggunakan teknologi *Augmented Reality* berbasis android.

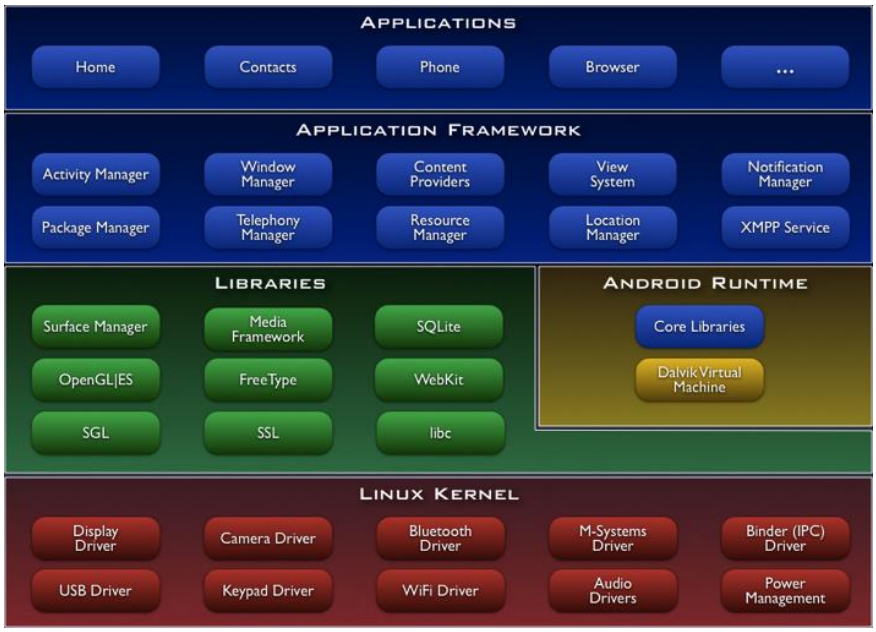
2.2 Android

Android adalah *sistem operasi* (OS) berbasis linux yang diperuntukan khusus untuk *mobile device* seperti *smartphone* atau PC *table*, persis seperti symbian yang dipergunakan oleh Nokia dan BlackBerry OS, jelasnya seperti

microsoft windows yang sangat dikenal baik oleh para pengguna komputer dan laptop, jika kita analogikan, *Andoid* adalah windows nya sedangkan *smartphone* atau *hand phone* atau tablet adalah unit komputernya.[3]

Android di ambil dari nama perusahaan penemunya yaitu *Android.inc* yang kemudian di akuisisi oleh Google pada pertengahan tahun 2005 dan mengubah nama penyedia aplikasi Android dari Android market menjadi *Google play*. Dengan sistem distribusi open sources yang di gunakan memungkinkan para pengembang untuk menciptakan beragam apikasi menarik yang dapat dinikmati oleh para penggunanya, seperti game, chatting dan lain-lain, hal ini pulalah yang membuat *smartphone* berbasis Android ini lebih murah dibanding *gadget* sejenis.[2]

Android memiliki 5 *layer*, setiap *layer* tersebut terdiri dari beberapa program yang mendukung fungsi-fungsi spesifik dari sistem operasi. Secara garis besar arsitektur Android dapat dijelaskan dan digambarkan sebagai berikut :



Gambar 2.1 Arsitektur android

1. *Applications*

Applications Adalah *layer* yang berhubungan dengan aplikasi saja, yaitu pada saat melakukan instalasi dan menjalankan aplikasi tersebut. Di *layer* terdapat aplikasi inti termasuk klien *email*, program SMS (*Short Message Sevice*), kalender, peta, *browser*, kontak, dan lain-lain.

2. *Applications Frameworks*

Adalah *layer* di mana para pembuat aplikasi melakukan pengembangan/pembuatan aplikasi yang akan dijalankan di sistem operasi Android, karena pada *layer* inilah aplikasi dapat dirancang dan dibuat, seperti *content-providers* yang berupa SMS dan panggilan telepon.

3. *Libraries*

Adalah *layer* di mana fitur-fitur Android berada, biasanya para pembuat plikasi mengakses *libraries* untuk menjalankan aplikasinya. Berjalan di atas kernel, *layer* ini meliputi berbagai *library* C/C++ inti seperti Libc dan SSL, serta :

a. *Libraries* media untuk pemutaran media audio dan video.
b. *Libraries* untuk manajemen tampilan.
c. *Libraries Graphics* mencakup SGL dan OpenGL untuk grafis dua Dimensi (2D) dan tiga Dimensi (3D).
d. *Libraries* SQLite untuk mendukung *database.*
e. *Libraries* SSL dan WebKit terintegrasi dengan *web browser* dan *security*.
f. *Libraries* LiveWebcore mencakup modern *web browser* dan *engine embeded web view*.
g. *Libraries* tiga Dimensi (3D) yang mencakup implementasi OpenGL ES 1.0 API's.

4. *Android Run Time*

Layer yang membuat aplikasi Android dapat dijalankan di mana dalam prosesnya menggunakan Implementasi Linux. *Dalvik Virtual Machine*(DVM) merupakan mesin yang membentuk dasar kerangka aplikasi Android. Di dalam Android *Run Time* dibagi menjadi dua bagian, yaitu :

1. *Core Libraries*

   Aplikasi Android dibangun dalam bahasa java, sementara *Dalviik* sebagai virtual mesinya bukan *virtual machine* java, sehingga diperlukan sebuah *libraries* yang berfungsi untuk menterjemahkan bahasa java/c yang ditangani oleh *CoreLibraries*.

2. *Dolvik Virtual Machine*

   Virtual mesin berbasis register yang dioptimalkan untuk menjalankan fungsi-fungsi secra efisien, dimana merupakan pengembangan yang

mampu membuat linux kernel untuk melakukan *threading* dan manajemen tingkat rendah.

5. Linux *Kernel*

Adalah *layer* di mana inti dari sistem operasi Android itu berada. *layer* ini berisi *fle-file system* yang mengatur sistem *processing, memory, resource, drivers,* dan sistem-sistem operasi Android lainnya.[13]

2.3 Augmented Reality

Menurut Suryawinata (2010), *Augmented Reality* (AR) adalah kombinasi antara dunia maya (*virtual*) dan dunia nyata (*real*) yang dibuat oleh komputer. Obyek *virtual* dapat berupa teks, animasi, model tiga Dimensi (3D) atau video yang digabungkan dengan lingkungan sebenarnya sehingga pengguna merasakan obyek *virtual* berada di lingkungannya. *Augmented Reality* (AR) adalah cara baru dan menyenangkan dimana manusia berinteraksi dengan komputer, karena dapat membawa obyek virtual ke lingkunagan pengguna, memberikan pengalaman visualisasi yang alami dan menyenangkan.[7]

Benda-benda maya menampilkan informasi yang tidak dapat diterima oleh pengguna dengan inderanya sendiri. Hal ini membuat *Augmented Reality* sesuai sebagai alat untuk membantu persepsi dan interaksi penggunanya dengan dunia nyata. Informasi yang ditampilkan oleh benda maya membantu pengguna melaksanakan kegiatan-kegiatan dalam dunia nyata.

Selain menambahkan benda maya dalam lingkungan nyata, *Augmented Reality* juga berpotensi menghilangkan benda-benda yang sudah ada. Menambah sebuah lapisan gambar maya dimungkinkan untuk menghilangkan atau menyembunyikan lingkungan nyata dari pandangan pengguna. Misalnya, untuk menyembunyikan sebuah meja dalam lingkungan nyata, perlu digambarkan lapisan representasi tembok dan lantai kosong yang diletakkan di atas gambar meja nyata, sehingga menutupi meja nyata dari pandangan pengguna.

Augmented Reality dapat diaplikasikan untuk semua indera, termasuk pendengaran, sentuhan, dan penciuman. Selain digunakan dalam bidang-bidang seperti kesehatan, militer, industri manufaktur, *Augmented Reality* juga telah

diaplikasikan dalam perangkat-perangkat yang digunakan orang banyak, seperti pada telepon genggam.[9]

Dalam membuat aplikasi *Augmented Reality* ini, penulis menggunakan game engine *Unity 3D engine* dan *Vuforia.*

2.3.1 Unity 3D Engine

Unity 3D Engine merupakan suatu *software game engine* yang terus berkembang saat ini. Penggunaan *engine versi free* dibatasi dengan beberapa fitur yang dikurangi atau bonus modul atau *prefab* tertentu yang ditiadakan dan hanya tersedia untuk pengguna berbayar. *Unity Engine* dapat mengolah beberapa data seperti objek tiga dimensi, suara, tekstur, dan lain sebagainya. Keunggulan dari *Unity 3D Engine* ini dapat menangani grafik dua dimensi dan tiga dimensi. Namun *Unity 3D Engine* ini lebih konsentrasi pada pembuatan grafik tiga dimensi. Dari beberapa *game engine* yang sama-sama menangani grafik tiga dimensi, *Unity 3D Engine* dapat menangani lebih banyak.

Beberapa diantaranya yaitu *Windows, MacOS X, iOS, PS3, wii, Xbox 360,* dan Android yang lebih banyak daripada *game engine* lain seperti *Source Engine, Game Maker, Unigine, id Tech 3 Engine, id Tech 4 Engine, Blender Game Engine, NeoEngine, Unity, Quake Engine, C4 Engine* atau *game engine* lain. *Unity 3D Engine* memiliki kerangka kerja (*framework*) lengkap untuk pengembangan profesional. Sistem inti *engine* ini menggunakan beberapa pilihan bahasa pemrograman, diantaranya C#, javascript maupun *boo*. [10]

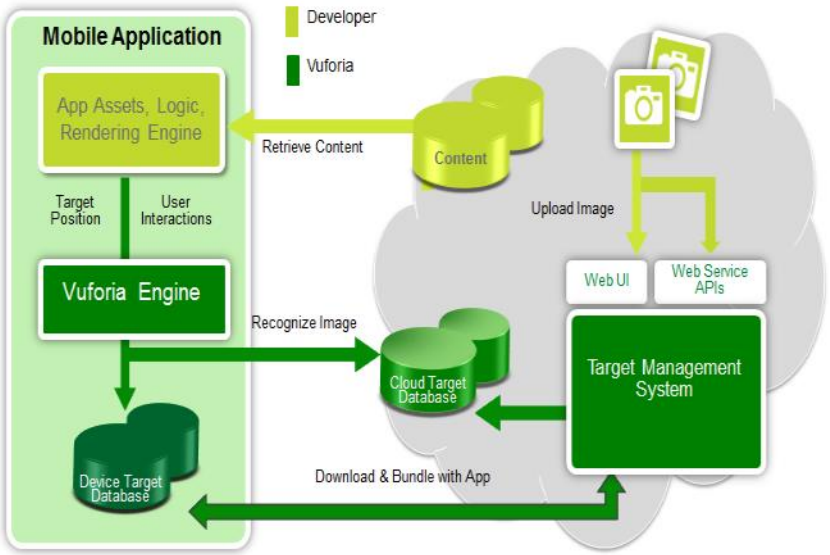
Unity 3D editor menyediakan beberapa alat untuk mempermudah pengembangan yaitu *Unity Tree* dan *terrain creator* untuk mempermudah pembuatan *vegetasi* dan *terrain* serta *MonoDevelop* untuk proses pemrograman.

2.3.2 Vuforia Augmented Reality

Dengan *support* untuk iOS, Android, dan *Unity3D, platform Vuforia* mendukung para pengembang untuk membuat aplikasi yang dapat digunakan di hampir seluruh jenis *smartphone* dan *tablet*.

Pengembang juga diberikan kebebasan untuk mendesain dan membuat aplikasi yang mempunyai kemampuan antara lain :

1. Teknologi *computer vision* tingkat tinggi yang mengijinkan *developer* untuk membuat efek khusus pada *mobile device*.
2. Terus-menerus mengenali *multiple image*.
3. *Tracking* dan *Detection* tingkat lanjut.
4. Dan solusi pengaturan database gambar yang fleksibel.

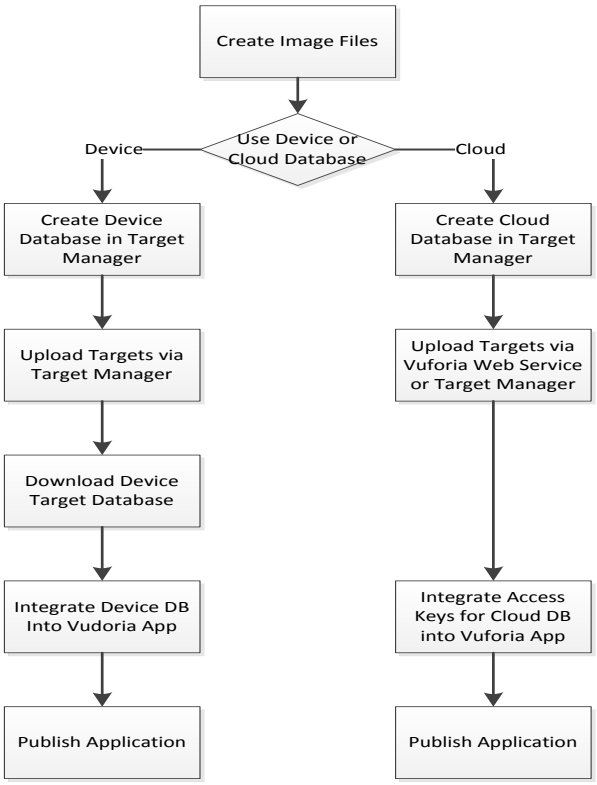


Gambar 2.2 Vuforia structure

Target pada *vuforia* merupakan obyek pada dunia nyata yang dapat dideteksi oleh kamera, untuk menampilkan obyek virtual. Beberapa jenis target pada vuforia adalah :

1. *Image targets*, contoh : foto, papan permainan, halaman majalah, sampul buku, kemasan produk, poster, kartu ucapan. Jenis target ini menampilkan gambar sederhana dari *Augmented*.
2. *Frame markers*, tipe *frame* gambar 2D dengan *pattern* khusus yang dapat digunakan sebagai potongan permainan di permainan pada papan.
3. *Multi-target*, contohnya kemasan produk atau produk yang berbentuk kotak ataupun persegi. Jenis ini dapat menampilkan gambar sederhana *Augmented* tiga Dimensi (3D).
4. *Virtual buttons*, yang dapat membuat tombol sebagai daerah kotak sebagai sasaran gambar.

Pada Vuforia, ada 2 jenis *workflow* dengan dasar database yang dapat dipilih oleh *developer*, yaitu *Cloud Database* dan *Device Database.* Setelah memilih metode yang dipilih, *developer* dapat mengikuti panduan di bawah ini untuk membuat aplikasi pada *Vuforia*.[5]



Gambar 2.3 Developing guide

2.4 Pemrograman Berorientasi Objek

Pemrograman berorientasi objek (*object-oriented programming* disingkat OOP) merupakan paradigma pemrograman yang berorientasikan kepada objek, dimana semua data dan fungsi dibungkus dalam *class-class* atau *object-object*. Setiap *object* dapat menerima pesan, memproses data, mengirim, menyimpan dan memanipulasi data. Beberapa *object* berinteraksi dengan saling memberikan informasi satu terhadap lainya. Masing-masing *object* harus berisikan informasi mengenai dirinya sendiri dan dapat dihubungkan dengan *object* yang lain. Konsep dasar dari Pemrograman berorientasi objek antara lain[14]:

1. Kelas (*Class*)

   Merupakan model yang berisi kumpulan *attribute* dan *method* dalam suatu unit untuk suatu tujuan tertentu. Sebagai contoh *class* manusia memiliki *attribute* berat, tinggi, usia kemudian memiliki *method* makan, minum, tidur. *Method* dalam sebuah *class* dapat merubah *attribute* yang dimiliki oleh *class* tersebut. Sebuah *class* merupakan dasar dari modularitas dan struktur dalam pemrograma berorientasi objek.

2. Atribut (*Attribute*)

   Attribute dari sebuah *class* adalah variabel global yang dimiliki sebuah *class*, *attribute* dapat memiliki hak akses *private*, *public* maupun *protected*.

3. Metode (*Method*)

   Method adalah fungsi atau prosedur yang dibuat oleh seorang *programmer* di dalam suatu *class.* Dengan kata lain, *method* pada sebuah *class* hampir sama dengan fungsi atau prosedur pada pemrograman prosedural. Pada sebuah *method* di dalam sebuah *classi* juga memiliki izin akses seperti halnya *attribute*.

4. Objek (*Object*)

   Merupakan perwujudan dari *class*, setiap *object* akan mempunyai *atrribute* dan *method* yang dimiliki oleh *class*-nya, contoh: andi, budi, yani merupakan *object* dari *class* manusia. Setiap *object* dapat berinteraksi dengan *object* lainnya meskipun berasal dari *class* yang berbeda.

5. Abstraksi (*Abstraction*)

   Adalah suatu cara untuk melihat suatu *object* dalam bentuk yang lebih sederhana. Sebagai contoh kita tidak melihat sebuah mobil sebagai ribuan komponen elektronik, sistem mekanik dengan empat bauh ban, jok, stir kemudi dan sebagainya. Dengan abstraksi, suatu sistem yang kompleks dapat dipandang sebagai kumpulan subsistem-subsistem yang lebih sederhana, seperti halnya mobil merupakan suatu sistem yang terdiri atas berbagai subsistem, seperti subsistem kemudi, subsistem pengereman dan sebagainya.

6. Enkapsulasi (*Encapsulation*)

   Merupakan suatu mekanisme untuk menyembunyikan atau memproteksi suatu proses dari kemungkinan interferensi atau penyalahgunaan dari luar

sistem dan sekaligus menyederhanakan penggunaan sistem tersebut. Memastikan pengguna sebuah *object* tidak dapat mengganti keadaan dalam dari sebuah *object* dengan cara yang tidak layak. Hanya metode dalam *object* tersebut yang diberi ijin untuk mengakses keadaannya. Setiap *object* mengakses *interface* yang menyebutkan bagaimana *object* lainnya dapat berinteraksi dengannya. *Object* lainnya tidak akan mengetahui dan tergantung kepada representasi dalam *object* tersebut.

7. Polimorfisme (*Polymorphism*)

   Merupakan konsep yang memungkinkan digunakannya suatu *interface* yang sama untuk memerintah suatu *object* agar melakukan suatu tindakan yang mungkin secara prinsip sama tetapi secara proses berbeda. Contoh, bila seekor burung menerima pesan "gerak cepat", dia akan menggerakan sayapnya dan terbang. Bila seekor singa menerima pesan yang sama, dia akan menggerakan kakinya dan berlari. Keduanya menjawab sebuah pesan yang sama, namun yang sesuai dengan kemampuan hewan tersebut.

8. Pewarisan (*Inheritance*)

   Merupakan konsep mewariskan *attribute* dan *method* yang dimiliki oleh sebuah *class* kepada *class* turunannya. Dengan konsep ini, *class* yang dibuat cukup mendefinisikan *attribute* dan *method* yang spesifik di dalamnya, sedangkan *attribute* dan *method* yang lebih umum akan didapatkan dari *class* yang menjadi induknya. Mengatur polimorfisme dan enkapsulasi dengan mengijinkan *object* didefinisikan dan diciptakan dengan jenis khusus dari *object* yang sudah ada. *Object-object* ini dapat membagi (dan memperluas) perilaku mereka tanpa harus mengimplementasikan ulang perilaku tersebut.

2.5 Unified Modeling Language

Unified Modeling Language (UML) adalah bahasa spesifikasi standar untuk mendokumentasikan, menspesifikasikan, dan membangun sistem perangkat lunak. UML merupakan himpunan struktur dan teknik untuk pemodelan desain program berorientasi objek (OOP) serta aplikasinya. UML adalah metodologi untuk mengembangkan sistem OOP dan sekelompok perangkat (*tool*) untuk mendukung pengembangan sistem tersebut. UML mulai diperkenalkan oleh Object

Management Group, sebuah organisasi yang telah mengembangkan model, teknologi, dan standar OOP sejak tahun 1980-an. Sekarang UML sudah mulai banyak digunakan oleh para praktisi OOP. UML merupakan dasar bagi perangkat (*tool*) desain berorientasi objek dari IBM.

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

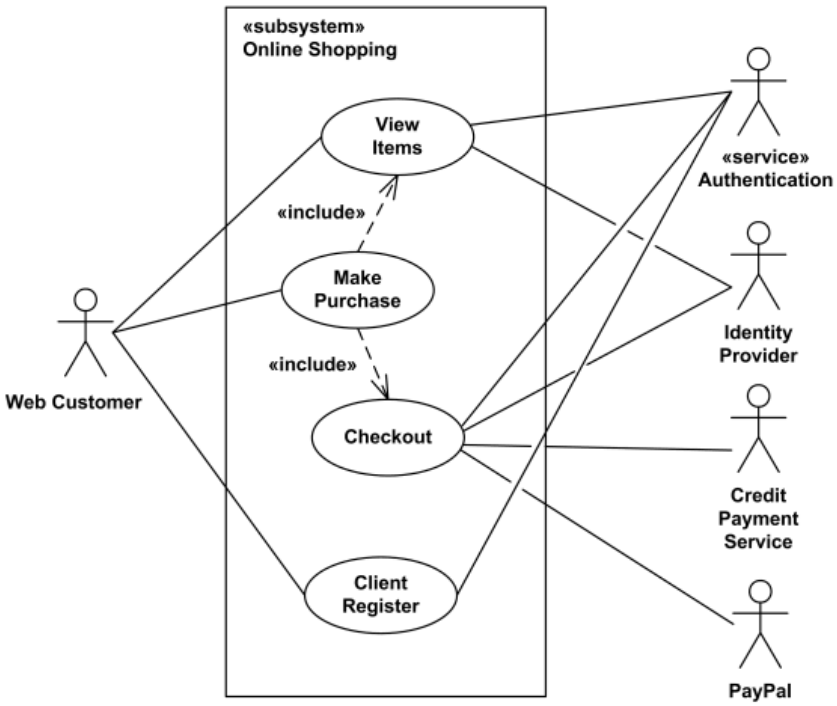
2.5.1 Diagram Unified Modeling Language

Unified Modeling Language (UML) menyediakan 10 macam diagram untuk memodelkan aplikasi berorientasi objek, yaitu[8]:

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

2.5.2 *Use Case* Diagram

Use case diagram menggambarkan fungsionalitas yang diharapkan dari sebuah sistem. Yang ditekankan adalah "apa" yang diperbuat sistem dan bukan "bagaimana". Sebuah *use case* merepresentasikan sebuah interaksi anatara aktor dengan sistem. *Use case* merupakan sebuah pekerjaan tertentu, misalnya *login* ke sistem, meng-*create* sebuah daftar belanja dan sebagainya. Seorang/sebuah aktor adalah sebuah entitas manusia atau mesin yang berinteraksi dengan sistem untuk melakukan pekerjaan-pekerjaan tertentu. Sebuah *use case* dapat meng-*include* fungsionalitas *use case* lain sebagai bagian dari proses dalam dirinya. Secara umum diasumsikan bahwa *use case* yang di-*include* akan dipanggil setiap kali *use case* yang meng-*include* dieksekusi secara normal. Sebuah *use case* dapat di-*include* oleh lebih dari satu *use case* lain, sehingga duplikasi fungsionalitas dapat dihindari dengan cara menarik keluar fungsionalitas yang *common*. Sebuah *use case* juga dapat meng-*extend* *use case* lain dengan *behaviour*-nya *sendiri*. Sementara hubungan generalisasi antar *use case menunjukkan bahwa* use case yang satu merupakan spesialisasi dari yang lain. Penggambaran *use case diagram* dapat dilihat pada gambar 2.4 berikut.

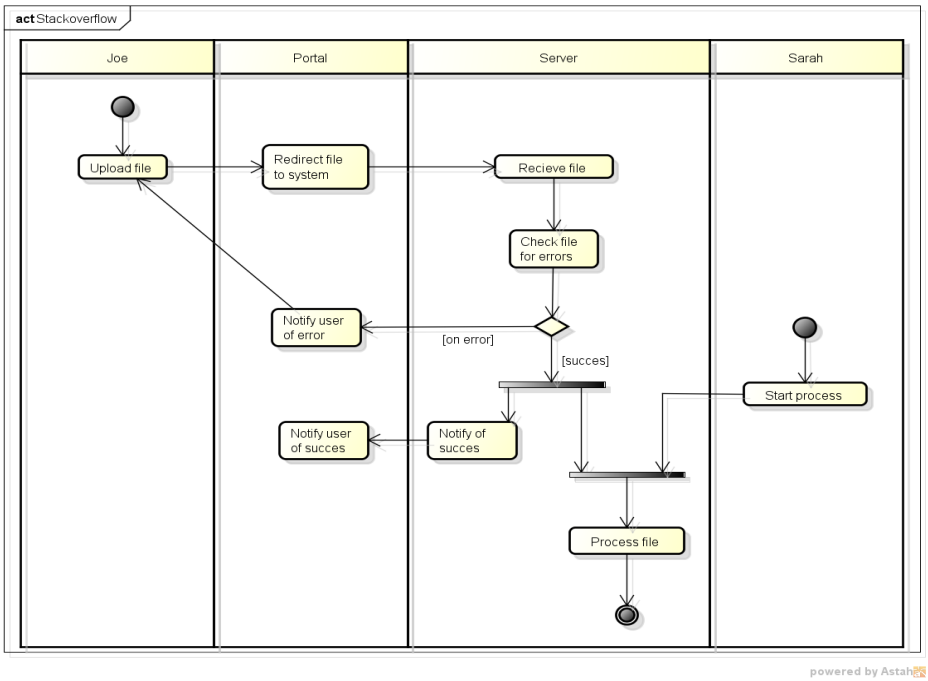


Gambar 2.4. *Contoh diagram use case*

2.5.3 *Activity* Diagram

Activity diagram menggambarkan berbagai alir aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alir berawal, *decision* yang mungkin terjadi dan bagaimana mereka berakhir. *Activity diagram* juga dapat menggambarkan proses paralel yang mungkin terjadi pada beberapa eksekusi. *Activity diagrami* merupakan *state diagram* khusus, dimana sebagian besar *state* adalah *action* dan sebagian besar transisi di-*trigger* oleh selesainya *state* sebelumnya *(internal processing)*. Oleh karena itu *activity diagram* tidak menggambarkan *behaviour* internal sebuah sistem (dan interaksi antar subsistem) secara eksak, tetapi lebih menggambarkan proses-proses dan jalur-jalur aktivitas dari level atas secara umum.

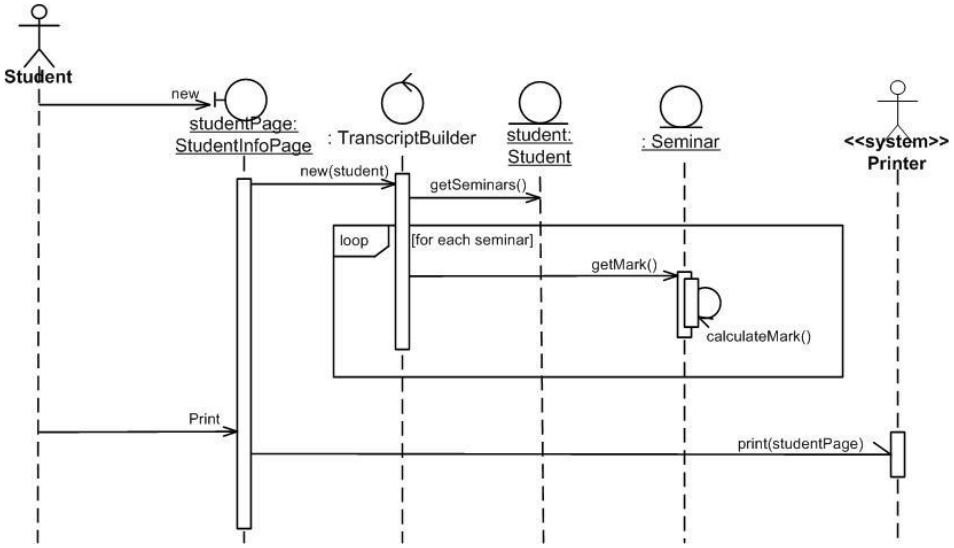
Sebuah aktivitas dapat direalisasikan oleh satu *use case* atau lebih. Aktivitas menggambarkan proses yang berjalan, sementara *use case* menggambarkan bagaimana aktor menggunakan sistem untuk melakukan aktivitas. *Decision* digunakan untuk menggambarkan *behaviour* pada kondisi tertentu. Untuk mengilustrasikan proses-proses paralel (*fork* dan *join*) digunakan titik sinkronisasi yang dapat berupa titik, garis horizontal atau vertikal. *Activity diagram* dapat dibagi menjadi beberapa *object swimline* untuk menggambarkan objek mana yang bertanggung jawab untuk aktivitas tertentu. Contoh *Activity* diagram dapat dilihat pada gambar 2.5 berikut.



Gambar 2.5. *Contoh diagram activity*

2.5.4 *Sequence* Diagram

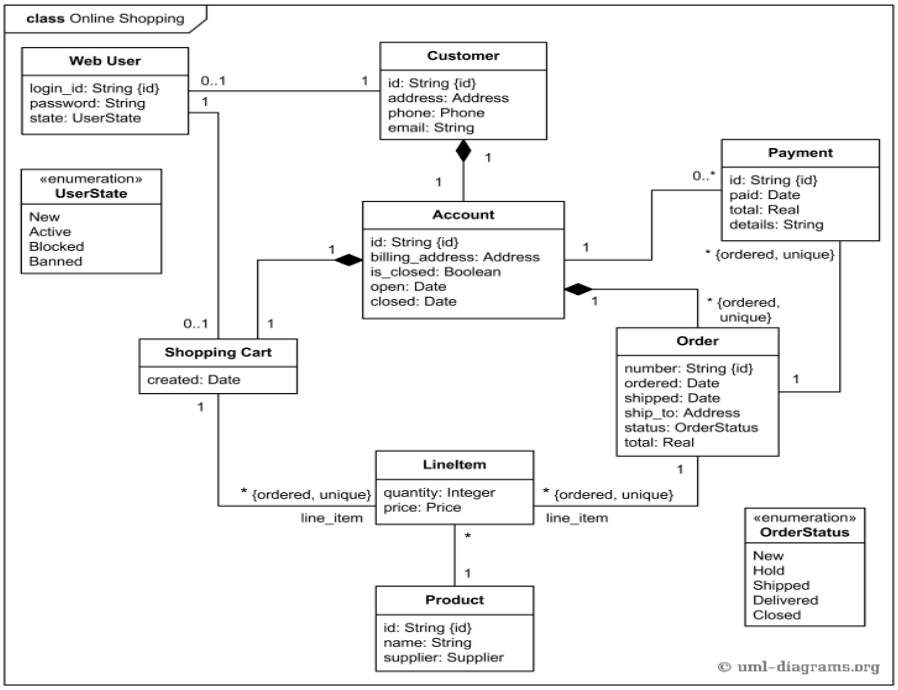
Sequence diagram menjelaskan secara detil urutan proses yang dilakukan dalam sistem untuk mencapai tujuan dari *use case*: interaksi yang terjadi antar *class*, operasi apa saja yang terlibat, urutan antar operasi dan informasi yang diperlukan oleh masing-masing operasi. Contoh *sequence* diagram dapat dilihat pada gambar 2.6 berikut.



Gambar 2.6. *Contoh diagram squence*

2.5.5 *Class* Diagram

Class diagram merupakan diagram yang selalu ada di permodelan sistem berorientasi objek. *Class* diagram menunjukan hubungan antar *class* dalam sistem yang sedang dibangun dan bagaimana mereka saling berkolaborasi untuk mencapai suatu tujuan. Penggambaran *class diagram* dapat dilihat dalam gambar 2.7 berikut.



Gambar 2.7 *Contoh class diagram*