BAB II

LANDASAN TEORI

2.1 Interaksi Antar Objek

2.1.1 PHP

Hypertext Preprocessor adalah Bahasa skrip yang dapat ditanamkan atau disisipkan ke dalam HTML. PHP banyak dipakai untuk memprogram situs web dinamis. PHP dapat digunakan untuk membangun sebuah CMS. (Anisya 2013)

Itu PHP masih bernama *Form Interpreted* (FI), yang wujudnya berupa sekumpulan skrip yang digunakan untuk mengelolah data formulir dari web. Selanjutnya Rasmus merilis kode sumber tersebut untuk umum dan menamakannya PHP/FI. Dengan perilisan kode sumber ini menjadi sumber terbuka, maka banyak pemrograman yang tertarik untuk ikut mengembangkan PHP.

2.1.2 MySQL

MySQL adalah suatu perangkat punak *database* relasi *(Relational Database Management System* atau DBMS*)*, seperti halnya ORACLE, POSTGRESQL, MSSQL, dan sebagainya. SQL merupakan singkatan dari *Structure Query Language*, didefinisikan sebagai suatu sintaks perintah-perintah tertentu atau Bahasa program yang digunakan untuk mengelola suatu *database*. Jadi MySQL adalah *software*-nya dan SQL adalah bahasa perintahnya. (Anisya 2013).

Menurut Arief dan Rudianto.M (2011) MySQL termasuk jenis RDBMS (Relational Database Management System). Sehingga istilah seperti tabel, baris, dan kolom tetap digunakan dalam MySQL. Pada mysql sebuah database mengandung satu beberapa tabel, tabel terdiri dari sejumlah baris dan kolom. mysql dikembangkan oleh sebuah perusahaan Swedia bernama mysql *AB*,. dapat dilihat dan gratis, serta *server* mysql dapat dipakai tanpa biaya tapi hanya untuk kebutuhan nonkomersial.

Sementara distribusi *Windows* mysql sendiri dirilis secara *shareware.* Barulah pada Juni 2000 mysql *AB* mengumumkan bahwa sejak versi 3.23.19, mysql adalah *Software* bebas berlisensi GPL. Artinya, "*Source code* mysql dapat dilihat dan gratis, serta *server* mysql dapat dipakai tanpa biaya untuk kebutuhan apa pun. Perangkat lunak ini bermanfaat untuk mengelola data dengan cara yang sangat

fleksibel dan cepat. Berikut adalah sejumlah aktivitas yang terkait dengan data yang didukung oleh perangkat lunak tersebut:

1. Menyimpan data ke dalam *table*
2. Menghapus data dalam *table*
3. Mengubah data dalam *table*
4. Mengambil data yang tersimpa dalam *table*
5. Memungkinan untuk memilih data tertentu yang diambil
6. Memungkinkan untuk melakukan pengaturan hak akses data.

Dan pada penggunaannya dalam mysql ini bisa menggunakan banyak *tools*, salah satunya phpMyAdmin.

2.1.2.1 Perintah – Perintah MySQL

Menurut (Anisya 2013) *Query* dikirimkan ke ***database*** dalam bentuk SQL *Query* beberapa perintah yang umum digunakan adalah sebagai berikut :

a. CREATE : Untuk membuat tabel baru
b. SELECT : Untuk mengambil *record* dari ***database*** yang memenuhi kritreria tertentu. yang memenuhi kritreria tertentu.
c. INSERT : Untuk menambah *record* ke dalam suatu *table*
d. UPDATE : Untuk merubah isi *record* tertentu pada suatu *table*
e. DELETE : Untuk menghapus *record* pada suatu *table*
f. DROP : Untuk menghapus sebuah *table*

2.1.3 Framework

Framework dapat diartikan sebagai koleksi atau kumpulan potongan – potongan program yang disusun atau diorganisasikan sedemikian rupa, sehingga dapat digunakan untuk membantu membuat aplikasi utuh tanpa harus membuat semua kodenya dari awal. Saat ini banyak *framework* PHP, diantaranya : *Zend, Cake PHP, Trax, Symfony, Codeigniter, Laravel* dan sebagainya. Tentu saja setiap *framewok* memiliki kelebihan dan kekurangannya masing – masing. (Octafian, WEB MULTI E-COMMERCE BERBASIS FRAMEWORK CODEIGNITER 2015) Keuntungan yang dapat diperoleh dari penggunaan *framework* adalah:

1. Waktu pembuatan aplikasi *website* jauh lebih singkat.

2. Kode aplikasi *website* menjadi lebih mudah dibaca, karena sedikit dan sifatnya pokok, detailnya adalah kode dari *framework.*
3. *Webiste* menjadi lebih mudah diperbaiki, karena tidak perlu fokus ke semua komponen kode *website,* terutama kode sistem *framework.*
4. Tidak perlu lagi membuat kode penunjang aplikasi *website* seperti koneksi *database,* validasi *form, ,* GUI, dan keamanan.
5. Pikiran pengembang menjadi lebih terfokus ke kode alur permasalahan *website,* apa yang ditampilkan dan layanan apa saja yang diberikan dari aplikasi *website* tersebut.
6. Jika dikerjakan *team work,* maka akan lebih terarah karena sistem *framework,* mengharuskan adanya keteraturan peletakan kode. Seperti bagian pengambilan *database* terpisah dengan bagian pengaturan tampilan untuk penunjang.

2.1.4 Codeigniter

Codeigniter adalah aplikasi *open source* yang berupa *framework* dengan model MVC *(Model, View, Controller)* untuk membangun website dinasmis dengan menggunakan PHP. *Codeigniter* memudahkan developer untuk membuat aplikasi web dengan cepat dan mudah dibandingkan dengan membuatnya dari awal.

Dengan menggunakan *framework* , kita tidak perlu membuat program dari awal, tetapi kita sudah memberikan *library* fungsi-fungsi yang sudah diorganisasi untuk dapat membuat suatu program dengan cepat. Kita hanya perlu memasukan data yang akan diproses dan bagaimana menampilkannya.

Codeigniter merupakan *framework* yang memiliki dokumentasi yang jelas dan lengkap. Yang memudahkan pengembang untuk memelajari dengan lengkap, yang memudahkan pengembang untuk memelajari dengan mudah. Pendekatan dari CI sangatlah mudah, dari membuat sekedar tulisan sampai dengan yang kompleks dapat didekati dengan mudah. Tidak seperti *framework* yang lain, untuk mendapatkan tulisan *Hello World* di *browser* saja, kita harus menggunakan beberapa tahap. CI cukup dengan satu file dan satu prosedur atau *method.*

Framework CI telah dibahas juga dalam forum di IBM dan *Oracle* untuk pengembang aplikasi berbasis web dengan menggunakan PHP. Seperti telah diketahui, IBM dan *Oracle* telah memiliki kerjasama degan Zend, yang mengembangkan PHP, dengan produk yang dikenal dengan *ZenCore for IBM DB3* dan *Informix*, dan perusahaan tersebut untuk memudahkan pengembang aplikasi *Oracle.* Pengembang tidak harus direpotkan atau sulit memasang PHP agar bias mengakses *database – database* tersebut. (Suharyo 2013).

Beberapa keuntungan menggunakan Codeigniter, diantaranya:

1. Gratis

   Codeigniter berlisensi dibawah Apache/ BSD *opensource*, sehingga penggunaannya secara bebas.

2. Berukuran kecil

   Ukuran Codeigniter yang kecil merupakan keunggulan tersendiri. Dibandingkan *framework* lain yang berukuran besar, serta membutuhkan *resource* yang besar pula untuk berjalan. Pada Codeigniter, bisa diatur agar sistem *meload library* yang dibutuhkan saja, sehingga sistem dapat berjalan ringan dan cepat.

3. Menggunakan konsep MVC

   Codeigniter menggunakan konsep MVC (*Model View Controller*) yang memungkinkan pemisahan antara *layer application-logic* dan *presentation.*

4. URL yang sederhana

   Secara *default*, URL yang dihasilkan Codeigniter sangat bersih (*clean*) dan *Search Engine Friendly* (SEF).

5. Memiliki paket *library* yang lengkap

   Codeigniter memiliki *library* yang lengkap untuk mengerjakan operasi-operasi yang umum dibutuhkan oleh sebuah aplikasi berbasis *web*, misalnya mengakses *database*, mengirim *email*, memvalidasi *form*, menangani *session* dan sebagainya.

6. *Extebsible*

   Sistem dapat dikembangkan dengan mudah dengan menggunakan *plugin* dan *helper*, atau dengan menggunakan *hooks*.

7. Tidak memerlukan *template engine*

Meskipun Codeigniter dilengkapi dengan *templateparses* sederhana yang dapat digunakan, tetapi hal ini tidak mengharuskan untuk menggunakannya. Penggunaan *template enggine* dapat mengurangi *performance* dari sistem.

8. Dokumentasi lengkap dan jelas

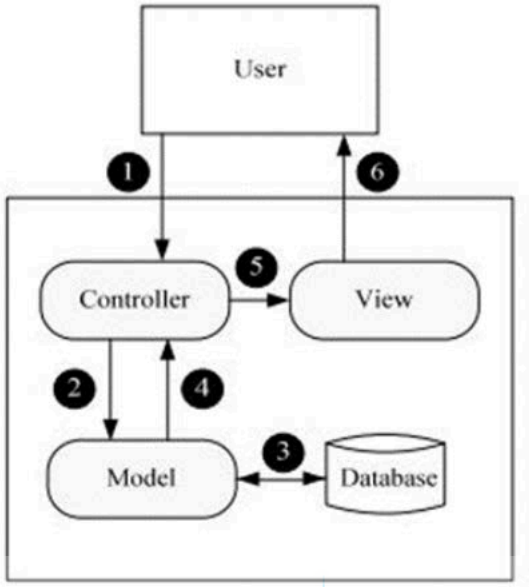
   Dari sekian banyak *framework*, Codeigniter adalah satu-satunya *framework* dengan dokumentasi yang lengkap dan jelas. Tim pengembang Codeigniter berkomitmen bahwa dokumentasi juga sama pentingnya dengan kode program Codeigniter itu sendiri. *Source code* Codeigniter juga dilengkapi komentar didalamnya, sehingga memperjelas fungsi sebuah kode program.

2.1.5 Konsep MVC (Model, View, Controller)

Sebuah *framework* aplikasi web biasanya mengimplementasikan pola desain yang disebut *Model, View* dan *Controller* atau yang biasa disebut sebagai MVC. Modul *model* memuat kelas-kelas yang mewakili tabel pada *database* yang mempunyai *instances* yang digunakan untuk memanipulasi *database. Model* biasanya digunakan sebagai penghubung antara modul *controller* dengan *database* ketika *controller* ingin mengambil dan menggunakan data di *database.*

Modul *controller* adalah kelas-kelas yang dibuat oleh *programmer* untuk menangani logika program dan *user events.* Pada aplikasi yang menggunakan pola MVC *Controller* bertindak sebagai otak dari sistem, menjembatani hubungan antara *model* dan *view. Controller* juga berfungsi menerima *frequest* dari *user* dan kemudian memprosesnya. Modul *view* berfungsi untuk menerima dan menampilkan data yang dikirim oleh *controller. View* dalam aplikasi berbasis *website* biasanya berbentuk kumpulan halaman HTML (Ahmad Leo Yudanto1 2017).

MVC membantu mengurangi kompleksitas dari oembuatan desain dan menambah fleksibilitas dan pemakaian kembali (re-use) kode. Berikut adalah ilustrasi dari konsep MVC:

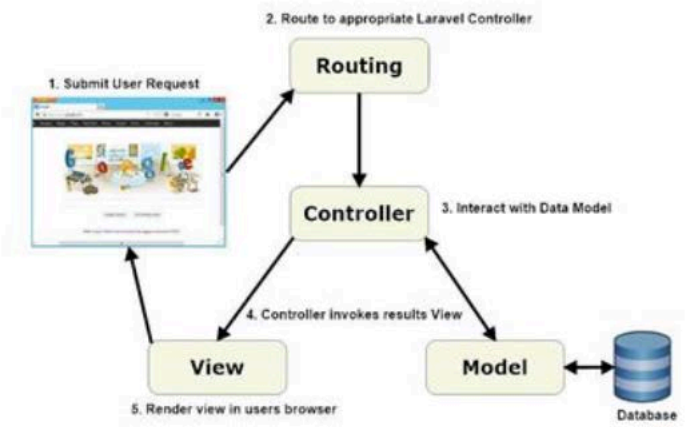


Gambar 1 *Konsep MVC*

Sumber: (Ahmad Leo Yudanto1 2017)

2.1.6 Laravel

Laravel adalah sebuah *framework* web berbasis PHP yang *open source* dan tidak berbayar, diciptakan oleh *Taylor Otwell* dan diperuntukkan untuk pengembangan aplikasi web yang menggunakan pola MVC. Struktur pola MVC pada laravel sedikit berbeda pada struktur pola MVC pada umumnya. Di Laravel terdapat *routing* yang menjembatani antara *request* dari *user* dan *controller*. Jadi *controller* tidak langsung menerima *request* tersebut. (Ahmad Leo Yudanto1 2017). Berikut adalah ilustrasi dari konsep MVC pada laravel yang ditunjukan pada gambar berikut :



Gambar 2 *Konsep MVC Laravel*

Sumber: (Ahmad Leo Yudanto1 2017)

2.1.7 OOP *(Object Oriented Programming)*

OOP *(Object Oriented Programming)* merupakan cara pandang dalam menganalisa sistem dan permasalahan pemrograman. Dalam OOP, setiap bagian dari program adalah *object.* Sebuah *object* mewakili suatu bagian program yang akan diselesaikan. Beberapa konsep OOP dasar, antara lain:

- *Encapsulation (Class* dan *Object).*
- *Inheritance* (Penurunan sifat).
- *Polymorphisme* (Wibowo, Analisa Konsep Object Oriented Programming Pada Bahasa Pemrograman PHP 2015).

2.1.7.1 *Class*

Menurut (Wibowo, Analisa Konsep Object Oriented Programming Pada Bahasa Pemrograman PHP 2015) *Class* merupakan gambaran dari sebuah *object* atau bisa dikatan output dari sebuah *object.* Pada bahasa pemrograman *class* merupakan sekumpulan kode yang dituliskan untuk mendefinisikan *property* dan metod yang ada pada sebuah *object*. Class didefinisikan dengan menampung nilai *property* dan metod, dimana properti adalah sebuah data yang menjelaskan tentang *class* dan metode adalah tingkah laku yang bisa dilakukan oleh *object*.

2.1.7.2 *Object*

Menurut (Wibowo, Analisa Konsep Object Oriented Programming Pada Bahasa Pemrograman PHP 2015) *Object* adalah hasil instansiasi dari *class*, dan mengandung seluruh *resource* yang telah didefinisikan pada *class*. Dikarenakan *class* merupakan cetakan dari *object*, maka *object* hasil instansiasi juga mempunyai *resource* seperti *class*.

2.1.7.3 *Encapsulation*

Menurut (Wibowo, Analisa Konsep Object Oriented Programming Pada Bahasa Pemrograman PHP 2015) *Encapsulation* adalah mekanisme ,membungkus sebuah data pada sebuah *object*. Dalam istilah lain seringkali disebut ,*Information Hiding*. Pada PHP terdapat 3 modifier yang dapat diimplementasikan untuk melakukan ,pembungkusan data yaitu *private*, *protacted* dan *public*. *Modifier* tersebut digunakan untuk mendefinisikan tingkat visibilitas sebuah data(properti) atau fungsi (metode) yang ada di dalam *class*.

2.1.7.4 *Polymorphisme*

Menurut (Wibowo, Analisa Konsep Object Oriented Programming Pada Bahasa Pemrograman PHP 2015) *Polymorphism* membuat objek-objek yang berasal dari *subclass* yang berbeda, diperlakukan sebagai objek-objek dari satu *superclass*. Hal ini terjadi ketika memilih method yang sesuai untuk diimplementasikan ke objek tertentu berdasarkan pada *subclass* yang memiliki method bersangkutan. Kondisi yang harus dipenuhi supaya polimorfisme dapat diimplementasikan adalah:

1. Method yang dipanggil harus melalui variable dari basis class atau *superclass*.
2. Method yang dipanggil harus juga menjadi method dari basis *class*.
3. Signature (*argument/parameter*) method harus sama baik pada *superclass* maupun *subclass*.
4. Method acces attribute pada subclass tidak boleh lebih terbatas dari basic class.

2.1.7.5 *Inheritance*

Menurut (Wibowo, Analisa Konsep Object Oriented Programming Pada Bahasa Pemrograman PHP 2015) Dalam pemrograman berorientasi

objek (OOP), *Inheritance* (pewarisan) adalah cara untuk menggunakan kembali kode objek yang ada, atau untuk mendirikan subtype dari objek yang sudah ada, atau keduanya, tergantung pada dukungan bahasa pemrograman.

2.1.8 Crud *Booster*

CRUDBooster yaitu CRUD *(Create, Read, Update, Delete)* Paket Generator. CRUDBooster menggunakan sistem scafollding untuk memastikan anda hanya fokus pada Controller, bukan model dan tampilan. Semua proses seperti penugasan, validasi, proses database dikontrol oleh CRUDBooster, bukan membuatnya secara manual (https://crudbooster.com/ n.d.).

2.2 Algoritma dan Metode

2.2.1 *Two Factor* Authentication

Two Factor Authentication (TFA) merupakan sebuah metode otentikasi pengguna dimana dua factor (dari tiga) yang bersifat independen akan digunakan dalam membuktikan adanya klaim bahwa sebuah entitas atau identitas itu asli. Dengan menggunakan TFA, maka *password* bukanlah menjadi *single point of attack* dalam hal akses kepada identitas pengguna. Banyak perusahaan besar telah mulai menggunakan TFA sebagai tambahan pengamanan terhadap serangan *brute-force* terhadap *password*, seperti Google, Twitter, dan Facebook (Willy Sudiarto Raharjo 2017).

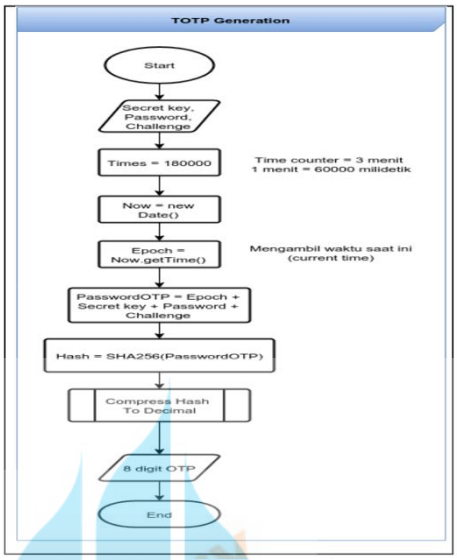
2.2.1 NEXMO

Nexmo exmo adalah salah satu vendor penyedia layanan SMS gateway berbasis cloud yang bias kita gunakan untuk mengirimkan sms ke semua negara yang disupport oleh nexmo. Jadi bukan hanya Indonesia saya yang disupport, Negara lain juga bias. Nexmo juga menyediakan API yang bisa kita gunakan untuk di integrasikan dengan layanan website atau aplikasi yang kita kembangkan (https://www.nexmo.com/ n.d.).

2.2.3 Algoritma TOTP

Time Based One Time Password atau TOTP adalah algoritma yang dikembangkan dari HMAC *(Hashes Message Authentication Code)* yang

menggabungkan antara *secret key* dengan *current timestamp* dan menggunakan fungsi kriptografi *hash* untuk menghasilkan *password* sekali pemakaian (Abukeshipa 2014). Untuk memberikan gambaran cara kerja algoritma TOTP dapat dilihat pada *flowchart* pada gambar berikut :



Gambar 3 Algoritma TOTP

Sumber: (Abukeshipa 2014)

Pada Gambar di atas menjelaskan algoritma TOTP yaitu:

1. Inisialisasi *secret key* diambil dari *serial number* token *virtual* yang sifatnya unik.
2. Inisialisasi *challenge code* berupa bilangan *decimal* 8 digit dengan kombinasi 2 digit dihasilkan secara *random* menggunakan *library* RandomUtils dan 6 digit diambil dari 6 digit terakhir dari nomor rekening.
3. *Times* adalah *counter* atau umur token *password* sebesar 3 menit. Pada Bahasa *scala* satuan terkecil untuk waktu yang dapat dijadikan untuk kalkulasi adalah milidetik sehingga harus di konversi 3 menit = 3 x 60 detik = 180 detik. 180 detik = 180 x 100 milidetik = 180000 milidetik
4. *Epoch* adalah jangka waktu yang dihitung saat *challenge password code* di *generate.*
5. *Password* OTP dihasilkan dari penggabungan *epoch, secret key,* dan *challenge code.*
6. Proses *hashing* di konversi menjadi bilangan *decimal* untuk diambil token *password* sebesar 8 digit.

Rumus Perhitungan

Proses menghasilkan *password* sekali pemakaian menggunakan algoritma *Time – Based One Time Password (TOTP)* menggunakan suatu perhitungan yang ditunjukan pada persamaan (Dzulqaidah 2012).

$$TOTP = HMAC(K, T)$$

Gambar 4 Persamaan TOTP

Sumber: (Dzulqaidah 2012)

Dimana:

TOTP : *Time- Based One Time Password.*

HMAC : *Keyed-hash* merupakan *Authemtication Code* merupakan sebuah fungsi kriptografi yang dikombinasikan dengan *secret key.*

K *(Secret Key)*: Kode rahasia yang diambil dari yang adiambil dari *serial number* token *virtual* dan disimpan pada sisi *server* dan juga *client.*

T : Jumlah *time-steps* antara inisial *counter* T0 dengan *current time* (waktu saat ini).

Untuk menghitung T menggunakan persamaan (Bayu Made Wedagama 2012).

$$T = currentTime\ (now) - T0/X$$

Gambar 5 Persamaan untuk menghitung T

Sumber: (Bayu Made Wedagama 2012)

Dimana :

Unix time (now) : Waktu saat ini *(current time)*

T0 : Waktu awal untuk menghitung *time steps* secara *default* di imisialisasi dengan nilai = 0

X : *time step* dengan inisialisasi = 30 detik.

Proses menghitung algoritma *keyed-hash Message Authentication Code* (HMAC) menggunakan persamaan (Abukeshipa 2014).

$$HMAC = SHA256\ (K + 0x5c5c)... \mid SHA256\ (K + 0x5c5c)... \mid$$

Gambar 6 Persamaan untuk menghitung HMAC

Sumber: (Abukeshipa 2014)

Dimana:

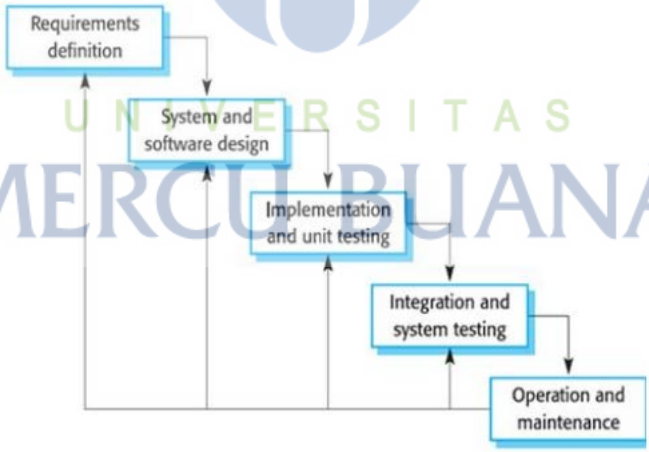
K : *Secret Key*

0x5c5c : nilai *Hexadecimal* dari waktu

2.2.4 Metode Penelitian

Dalam rekayasa perangkat lunak, terdapat suatu pendekatan yang disebut *waterfall model.* Model ini merupakan model yang paling banyak dipakai oleh para pengembang *software.*

Menurut Sommerville (Rosmiati 2015) , tahapan utama dari *waterfall* model langsung mencerminkan aktifitas pengembangan dasar. Terdapat 5 tahapan pada *waterfall* model, yaitu *requirement analysis and definition, system and software design, implementation and unit testing, integration and system testing,* dan *operation and maintenance.*



Gambar 7 Metode *Waterfall* (Sasmito 2017)

Metode *Waterfall* memiliki tahapan-tahapan sebagai besikut :

1. *Requirements and definition*

Layanan sistem, kendala, dan tujuan diterapkan oleh hasil konsultasi dengan pengguna yang kemudian didefinisikan secara rinci dan berfungsi sebagai spesifikasi sistem.

2. *System and Software Design*

Tahapan perancangan sistem mengalokasikan kebutuhan-kebutuhan sistem baik perangkat keras maupun perangkat lunak dengan membentuk arsitektur sistem secara keseluruhan. Perancangan perangkat lunak melibatkan identifikasi dan penggambaran abstraksi sistem dasar perangkat lunak dan hubungannya.

3. *Implementation and unit Testing*

Pada tahap ini, perancangan perangkat lunak direalisasikan sebagai serangkaian program atau unit program. Pengujian melibatkan berifikasi bahwa setiap unit memenuhi spesifikasinya.

4. *Integration and System Testing*

Unit – unit individu program atau program digabung dan diuji sebagai sebuah sistem lengkap untuk memastikan apakah sesuai dengan kebutuhan perangkat lunak atau tidak. Setelah pengujian, perangkat lunak dapat dikirimkan ke *customer*.

5. *Operational and Maintenance*

Biasanya (walaupun tidak selalu), tahapan ini merupakan tahapan yang paling panjang. Sistem dipasang dan digunakan secara nyata. *Maintenance* melibatkan pembetulan kesalahan yang tidak ditemukan pada tahapan-tahapan sebelumnya, meningkatkan implementasi dari unit sistem, dan meningkatkan layanan sistem sebagai kebutuhan baru (Sasmito 2017).

2.2.5 Definisi *Database*

Menurut Connolly dan Begg (2015), Basis data *(database)*, atau sering pula dieja basis data, adalah sekumpulan data yang secara logis dan terkait keterangannya. Basis data dirancang untuk memenuhi kebutuhan informasi dari suatu organisasi.

2.2.5.1 Komponen DBMS

Lima komponen utama dalam lingkungan DBMS, yaitu:

1. Perangkat Keras (*Hardware*)

Pengaplikasian DBMS membutuhkan perangkat keras untuk menjalankannya.

2. Perangkat Lunak (*Software*)

Komponen perangkat lunak DBMS itu sendiri, seperti sistem operasi.

3. Data

Komponen paling penting dalam DBMS yang menghubungkan antara komponen mesin dan komponen manusia.

4. Prosedur

Berupa instruksi dan peraturan design database.

5. Manusia

Komponen terakhir, manusia ikut terlibat dalam sistem.

2.2.5.2 Kelebihan DBMS

Menurut (Connolly dan Begg, 2015), DBMS memiliki beberapa kelebihan:

1. Pengendalian terhadap redudansi data.
2. Konsistensi data.
3. Pembagian data.
4. Meningkatkan integritas data.
5. Meningkatkan keamanan.
6. Meningkatkan aksesibilitas dan respon data.
7. Meningkatkan produktivitas.
8. Meningkatkan pemeliharaan melalui data *independence*.
9. Meningkatkan konkurensi.
10. Meningkatkan kemampuan *backup* dan *recovery*.

2.2.5.3 Kekurangan DBMS

Menurut (Connolly dan Begg, 2015), DBMS juga memiliki kekurangan sebagai berikut:

1. Kompleksitas.
2. Ukuran.
3. Biaya DBMS.
4. Biaya-biaya *hardware* tambahan.
5. Biaya konversi.

6. Performa.
7. Dampak besar ketika mengalami kegagalan.

2.2.6 Definisi UML *(Unified Modelling Language)*

UML singkatan dari *Unified Modelling Language*, UML adalah kosakata umum berbasis objek dan diagram teknik yang cukup efektif untuk memodelkan setiap proyek pengembangan *system* mulai tahap analisis sampai tahap perancangan dan taham implementasi (B. H. Alan Dennis, Systems Analysis & Design 5th Edition 2012).

2.2.6.1 *Use Case* Diagram

Use Case Diagram merupakan suatu diagram yang menangkap kebutuhan bisnis untuk sistem dan untuk menggambarkan interaksi antara sistem dan lingkungannya (B. H. Alan Dennis, Systems Analysis & Design 5th Edition 2012).

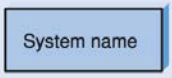
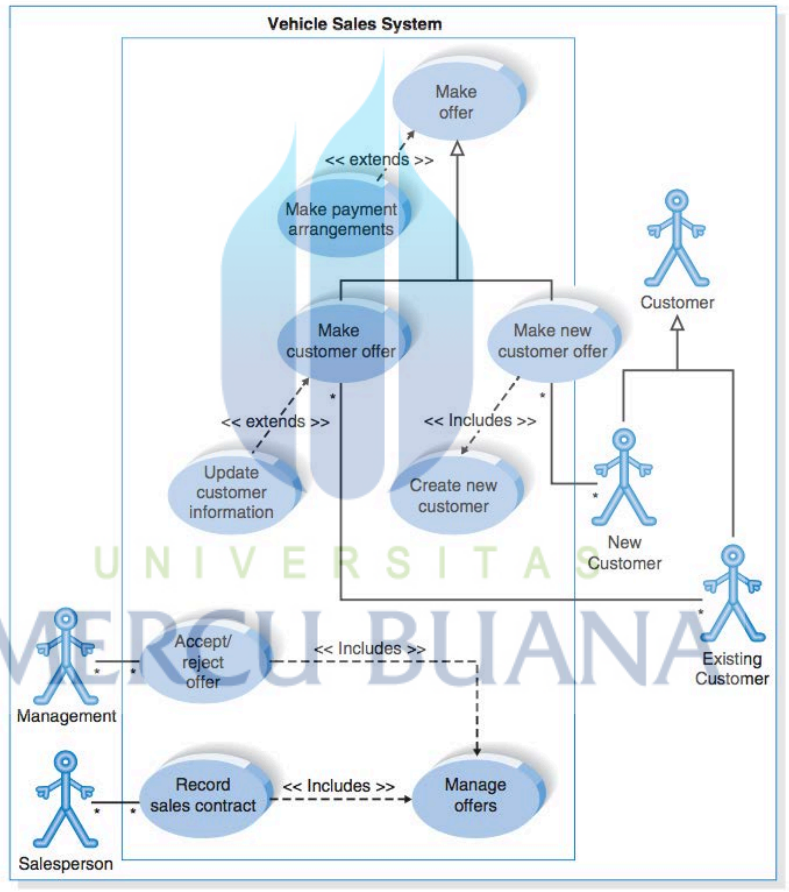
Term and Definition	Symbol
An actor ▪ Is a person or system that derives benefit from and is external to the system. ▪ Is labeled with its role. ▪ Can be associated with other actors by a specialization/superclass association, denoted by an arrow with a hollow arrowhead. ▪ Is placed outside the system boundary.	 Actor role name
A use case ▪ Represents a major piece of system functionality. ▪ Can extend another use case. ▪ Can use another use case. ▪ Is placed inside the system boundary. ▪ Is labeled with a descriptive verb–noun phrase.	 Use case name
A system boundary ▪ Includes the name of the system inside or on top. ▪ Represents the scope of the system.	 System name
An association relationship ▪ Links an actor with the use case(s) with which it interacts.	 * *

Table 1*Use Case Diagram*

Sumber: (B. H. Alan Dennis, Systems Analysis & Design 2012)

Komponen pembentuk diagram *use case* adalah, sebagai berikut:

1. Aktor (*An actor*), menggambarkan pihak-pihak yang berperan dalam sistem.
2. *A Use case*, aktifitas / sarana yang disiapkan oleh bisnis / sistem.
3. *A System boundary*, sebuah kotak yang mewakili sebuah sistem.
4. Hubungan (*An association relationship*), aktor mana saja yang terlibat dalam *use case*, dan bagaimana hubungan *use case* dengan *use case* lain. ada hubungan antar *use case*. Digolongkan menjadi 2: yaitu extend digambarkan dengan keterangan <<*extend*>>, dan include digambarkan dengan keterangan <<*include*>>.



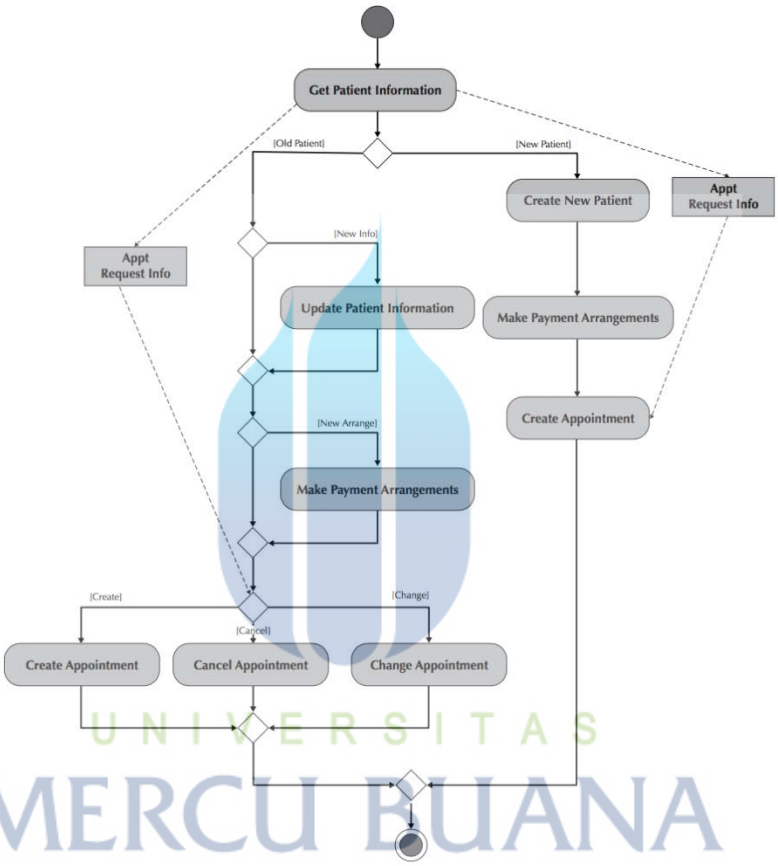
Gambar 8 Perbedaan *include* dan *extend* pada *use case*

Sumber: (B. H. Alan Dennis, Systems Analysis & Design 2012)

2.2.6.2 *Activity* Diagram

Diagram aktivitas atau *Activity Diagram* yang menggambarkan alur

kerja bisnis independen dari kelas, aliran kegiatan dalam *use case* atau desain rinci sebuah metode (B. H. Alan Dennis 2015)



Gambar 9 Contoh *Activity Diagram*

Sumber: (B. H. Alan Dennis 2015)

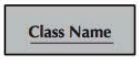
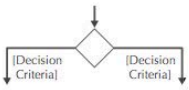
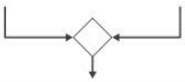
An action: ■ Is a simple, nondecomposable piece of behavior. ■ Is labeled by its name.	
An activity: ■ Is used to represent a set of actions. ■ Is labeled by its name.	
An object node: ■ Is used to represent an object that is connected to a set of object flows. ■ Is labeled by its class name.	
A control flow: ■ Shows the sequence of execution.	
An object flow: ■ Shows the flow of an object from one activity (or action) to another activity (or action).	
An initial node: ■ Portrays the beginning of a set of actions or activities.	
A final-activity node: ■ Is used to stop all control flows and object flows in an activity (or action).	
A final-flow node: ■ Is used to stop a specific control flow or object flow.	
A decision node: ■ Is used to represent a test condition to ensure that the control flow or object flow only goes down one path. ■ Is labeled with the decision criteria to continue down the specific path.	
A merge node: ■ Is used to bring back together different decision paths that were created using a decision node.	
A Fork node: Is used to split behavior into a set of parallel or concurrent flows of activities (or actions).	
A Join node: Is used to bring back together a set of parallel or concurrent flows of activities (or actions).	
A Swimlane: Is used to break up an activity diagram into rows and columns to assign the individual activities (or actions) to the individuals or objects that are responsible for executing the activity (or action). Is labeled with the name of the individual or object responsible.	

Table 2 Komponen *Activity Diagram*

Sumber: (B. H. Alan Dennis 2015)

2.2.6.3 *Sequence* Diagram

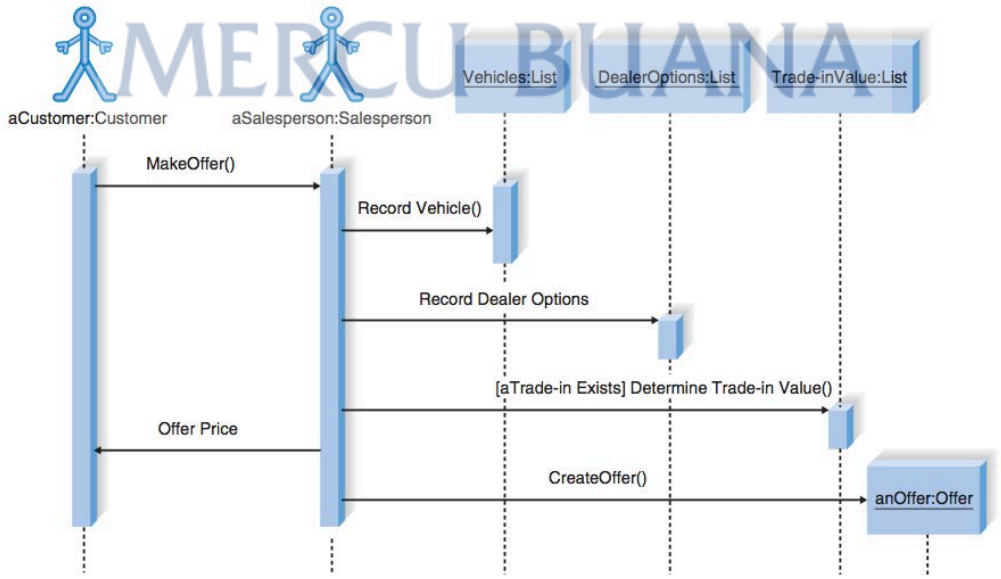
Sequence Diagram merupakan urutan model dinamis yang menggambarkan contoh kelas yang berpartisipasi dalam *use case* dan pesan yang lewat di antara mereka dari waktu ke waktu (B. H. Alan Dennis, System Analysis And Design Fifth Edition 2012).

Berikut merupakan komponen utama dalam *sequence diagram*:

Term and Definition	Symbol
An actor: ▪ Is a person or system that derives benefit from and is external to the system. ▪ Participates in a sequence by sending and/or receiving messages. ▪ Is placed across the top of the diagram.	 anActor
An object: ▪ Participates in a sequence by sending and/or receiving messages. ▪ Is placed across the top of the diagram.	 anObject:aClass
A lifeline: ▪ Denotes the life of an object during a sequence. ▪ Contains an X at the point at which the class no longer interacts.	
A focus of control: ▪ Is a long narrow rectangle placed atop a lifeline. ▪ Denotes when an object is sending or receiving messages.	
A message: ▪ Conveys information from one object to another one.	 aMessage()
Object destruction: ▪ An *X* is placed at the end of an object's lifeline to show that it is going out of existence.	X

Table 3 Komponen *Sequence Diagram*

Sumber: (B. H. Alan Dennis, System Analysis And Design Fifth Edition 2012)

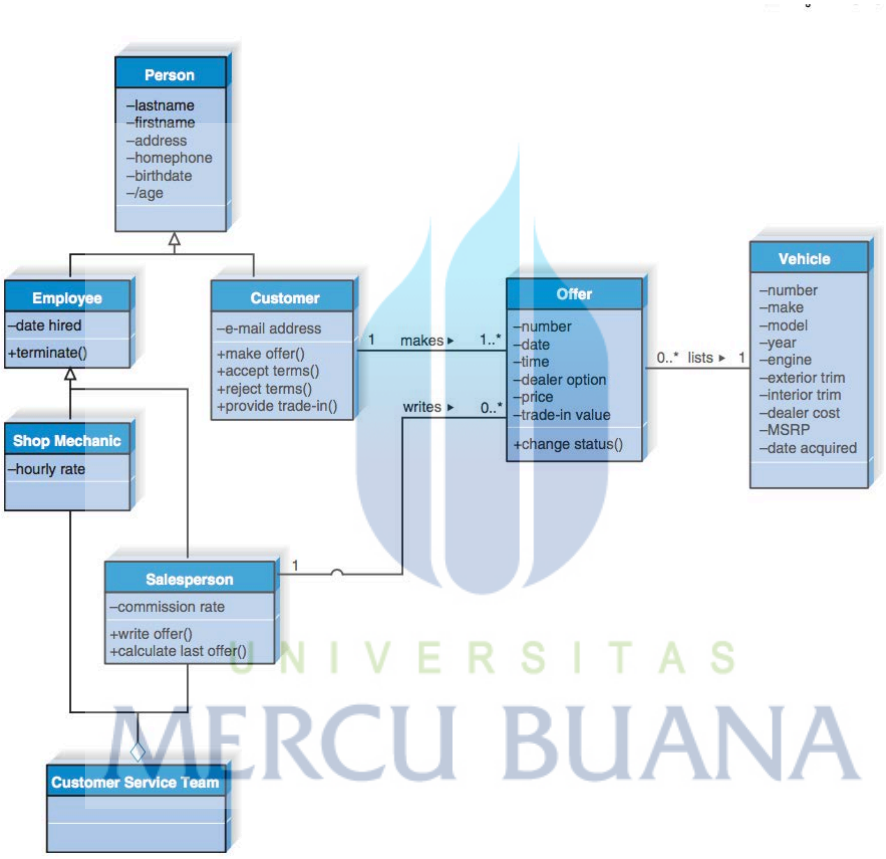


Gambar 10 Komponen *Sequence Diagram*

Sumber: (B. H. Alan Dennis, System Analysis And Design Fifth Edition 2012)

2.2.6.4 *Class* Diagram

Diagram kelas atau *Class Diagram* adalah model statis yang mendukung pandangan statis dari sistem berkembang. Ini menunjukkan kelas dan hubungan antar kelas yang tetap konstan dalam sistem dari waktu ke waktu. Diagram kelas sangat mirip dengan diagram hubungan entitas (ERD) (B. H. Alan Dennis, System Analysis And Design Fifth Edition 2012).



Gambar 11 Contoh *Class Diagram*

Sumber: (B. H. Alan Dennis 2012)

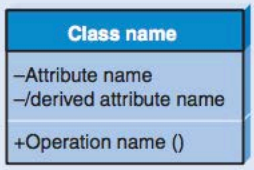
Term and Definition	Symbol
A class ▪ Represents a kind of person, place, or thing about which the system must capture and store information. ▪ Has a name typed in bold and centered in its top compartment. ▪ Has a list of attributes in its middle compartment. ▪ Has a list of operations in its bottom compartment. ▪ Does not explicitly show operations that are available to all classes.	
An attribute ▪ Represents properties that describe the state of an object. ▪ Can be derived from other attributes, shown by placing a slash before the attribute's name.	Attribute name /derived attribute name
A method ▪ Represents the actions or functions that a class can perform. ▪ Can be classified as a constructor, query, or update operation. ▪ Includes parentheses that may contain special parameters or information needed to perform the operation.	Operation name ()
An association ▪ Represents a relationship between multiple classes, or a class and itself. ▪ Is labeled by a verb phrase or a role name, whichever better represents the relationship. ▪ Can exist between one or more classes. ▪ Contains multiplicity symbols, which represent the minimum and maximum times a class instance can be associated with the related class instance.	1..* verb phrase 0..1

Table 4 Komponen *Class Diagram*

Sumber: (B. H. Alan Dennis 2012)

2.2.6.5 *Black Box Testing*

Menurut (Octafian, Web Multi E-commerce Berbasis Framework Codeigniter 2015), proses *black-box testing* adalah pengujian yang berasal dari spesifikasi sistem. Sistem diperlakukan seperti sebuah kotak hitam (*black-box*) yang mana perilaku hanya bisa ditentukan dengan mempelajari inputan dan keluaran yang berhubungan. Nama lainnya adalah pengujian fungsional karena penguji hanya dikonsentrasikan terhadap fungsionalitas dan tidak pengimplementasi dari perangkat lunak.