BAB 2
TINJAUAN PUSTAKA

2.1 Landasan Teori

Pada tahapan ini akan menjelaskan tentang konsep – konsep yang berkaitan dengan pembangunan *class library* untuk domain *file manager* pada aplikasi *cloud storage* berbasis web.

2.1.1 *Cloud Storage*

Cloud Storage merupakan lapisan terbawah dari sistem *cloud computing* yang mendukung layanan dari lapisan lain yang berada diatasnya. Selain itu, *cloud storage* adalah suatu cara yang efektif untuk menyimpan dan mengelola data[9]. Pada dasarnya *cloud storage* merupakan pengembangan dari teknologi *cloud computing*, yang mana pada konsepnya *cloud storage* adalah sebuah teknologi yang memanfaatkan sebuah *server* sebagai media penyimpanan. Berbeda dengan media penyimpanan konvensional seperti *flashdisk* dan *hardisk*, teknologi *cloud storage* tidak membutuhkan perangkat tambahan. Melainkan perangkat komputer atau gadget yang terhubung dengan internet untuk dapat mengakses file digital pada suatu server.

2.1.2 *File Manager*

File manager merupakan salah satu domain aplikasi yang digunakan untuk mengelola *file* dan *folder* supaya pengguna (*user*) dapat mengakses dan menggunakan data dengan mudah, cepat, dan fleksibel[10]. Selain itu *file* manager berfungsi untuk membuat *folder*, menghapus *folder*, menyalin *file*, memindahkan *file*, dan menampilkan data.

2.1.3 *Class Library*

Didalam pemrograman berorientasi objek akan ditemui sebuah metode *reuseable code* atau bisa disebut dengan *class library*. *Class library* adalah

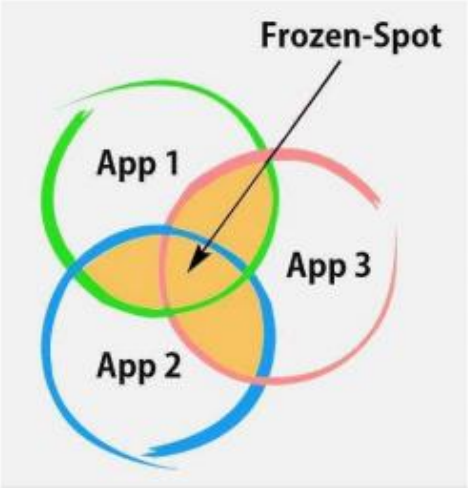
kumpulan kelas dan *method* yang telah ditulis sebelumnya dan bertujuan untuk dapat digunakan kembali oleh *programmer* didalam mengembangkan sebuah perangkat lunak[6]. Dengan kata lain *class library* adalah sebuah *source code* yang terdiri fungsi – fungsi yang dapat digabung dan di implementasikan pada perangkat lunak baru untuk menyederhanakan kerja *programmer*.

2.1.4 Analisis Domain

Analisis domain adalah sebuah proses pengelompokan perangkat lunak berlandaskan pada kesamaan fungsionalitas pada perangkat lunak yang ada didalam satu domain. Dengan analisis yang mendetail terhadap perangkat lunak dengan domain yang sama, dapat ditemukan persamaan dan perbedaan pada setiap perangkat lunak[11]. Hal ini dilakukan untuk mendefinisikan kebutuhan dasar dalam pembangunan *class library*, setidaknya analisis ini dilakukan pada tiga perangkat lunak yang berada didalam domain yang sama.

2.1.5 Analisis *Frozenspot*

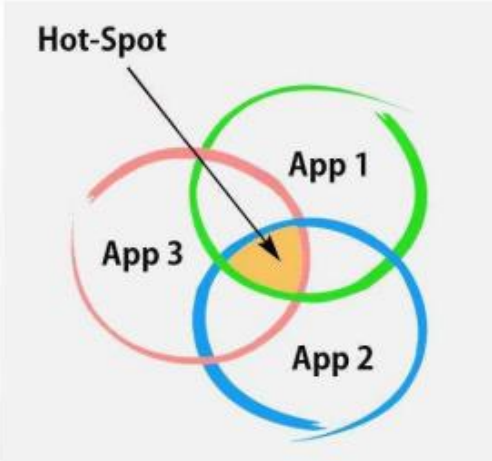
Frozen spot merupakan fungsionalitas yang identik ada pada suatu perangkat lunak domain kasus tertentu [12]. *Frozenspot* adalah suatu analisis yang dilakukan pada fungsional dasar maupun keseluruhan fungsional perangkat lunak yang berada pada domain kasus yang sama. Selain itu *frozenspot* dapat dikatakan sebagai rujukan dasar yang kemudian dijadikan sebagai suatu *hotspot*. Berikut ini adalah ilustrasi *sample* analisis *frozenspot* yang dapat dilihat pada gambar 2.3



Gambar 2.1 *Frozenspot*

2.1.6 Analisis *Hotspot*

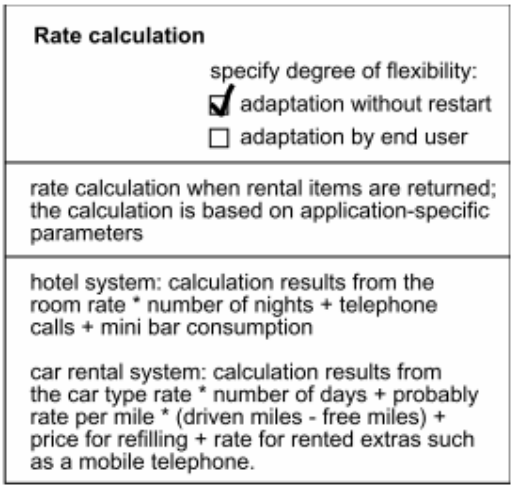
Hotspot merupakan fungsionalitas yang dapat diubah sesuai dengan kebutuhan perangkat lunak yang akan dikembangkan[12]. Pada dasarnya *hotspot* adalah fungsionalitas dasar yang terdapat pada masing – masing perangkat lunak pada domain yang sama. *Hotspot* didapatkan dari hasil analisis frozenspot pada masing – masing aplikasi didomain yang sama. Selain itu hotspot bisa diperoleh melalui pengujian *whitebox* dan *blackbox*. *Whitebox* merupakan metode pendefinisian *hotspot* melalui kode sumber yang telah ada. *Blackbox* adalah pendefinisian *hotspot* berdasarkan analisis *frozenspot* perangkat lunak yang berada didalam domain yang sama[13]. Berikut ini adalah contoh ilustrasi analisis *Hotspot* yang dapat dilihat pada gambar 2.4



Gambar 2.2 *Hotspot*

2.1.7 Kartu *Hotspot*

Kartu *hotspot* berguna untuk mendeskripsikan fungsionalitas dari setiap *hotspot* yang telah didefinikan sebelumnya[14]. Kartu *hotspot* terdiri dari bagian nama *hotspot*, derajat fleksibilitas, deskripsi fungsionalitas, serta perilaku *hotspot* pada dua situasi yang berbeda. Nama dari kartu *hotspot* harus menggambarkan fungsionalitasnya. Adaptasi fleksibilitas terdiri dari dua pilihan yaitu adaptasi tanpa *restart* dan adaptasi dari *end user*. Adaptasi tanpa *restart* adalah adaptasi dari *hotspot* dapat langsung digunakan, sedangkan adaptasi oleh *end user hotspot* harus diadaptasi oleh *end user* sebelum digunakan.



Gambar 2.3 Ilustrasi Kartu *Hotspot*

2.1.8 Perangkat Lunak

Perangkat lunak atau lebih dikenal dengan nama *software* adalah program komputer yang berguna untuk menjembatani terjadinya interaksi antara pengguna (*user*) dengan perangkat keras (*hardware*)[13]. Selain itu *software* juga dapat dikatakan sebagai penerjemah perintah – perintah yang dijalankan oleh pengguna (*user*) komputer untuk diproses dan dijalankan oleh perangkat keras (*hardware*).

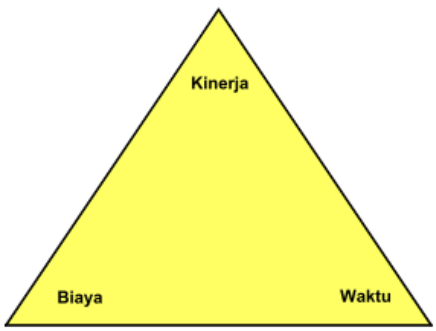
Selain itu perangkat lunak dapat diartikan sebagai seluruh perintah yang digunakan untuk memproses dan memperoleh informasi. Perangkat lunak dapat berupa program atau prosedur. Program adalah sekumpulan atau baris perintah kode yang dimengerti oleh komputer sedangkan prosedur adalah perintah yang dibutuhkan oleh pengguna dalam memproses informasi[15].

2.1.9 Rekayasa Perangkat Lunak

Rekayasa perangkat lunak atau dikenal dengan istilah *software engineering* adalah suatu bidang ilmu yang membahas semua aspek produksi perangkat lunak, tahap awal yaitu analisa kebutuhan, menentukan spesifikasi dari kebutuhan, desain, pengkodean, pengujian sampai pemeliharaan sistem setelah digunakan[15].

2.1.9.1 Tujuan Rekayasa Perangkat Lunak

Adapun tujuan dari rekayasa perangkat lunak adalah sebagai berikut:



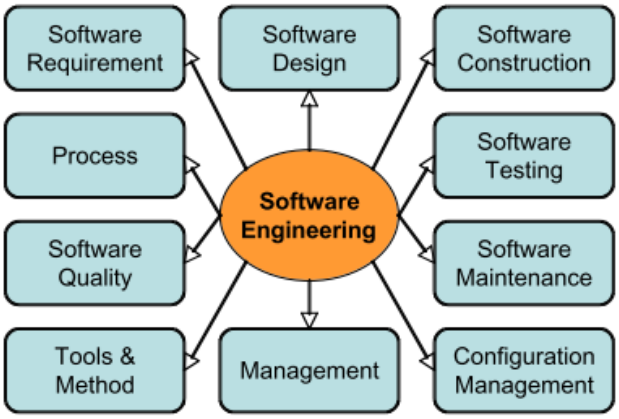
Gambar 2.4 Tujuan RPL

Dari gambar 2.1 dapat dilihat bahwa rekayasa perangkat lunak bertujuan untuk menghasilkan keluaran dengan kinerja tinggi, biaya rendah dan waktu penyelesaian yang tepat. Adapun tujuan utama dari rekayasa perangkat lunak adalah :

1. Menghasilkan biaya produksi perangkat lunak yang murah
2. Menghasilkan perangkat lunak dengan kinerjanya tinggi, andal dan tepat waktu
3. Menghasilkan perangkat lunak yang bisa bekerja pada berbagai *platform*
4. Menghasilkan perangkat lunak dengan biaya perawatannya rendah.

2.1.9.2 Ruang Lingkup Rekayasa Perangkat Lunak

Sebelumnya telah dijabarkan tentang tujuan dari rekayasa perangkat lunak, berikut ini adalah ruang lingkup RPL dapat digambarkan sebagai berikut :



Gambar 2.5 Ruang Lingkup RPL

1. *Sofware Requirements* berhubungan dengan spesifikasi kebutuhan dan persyaratan perangkat lunak.
2. *Software Design* berhubungan dengan proses penentuan arsitektur, antarmuka, komponen, dan karakteristik lain dari perangkat lunak.
3. *Software Construction* berhubungan dengan detail pengembangan perangkat lunak, algoritma, pengujian dan pencarian kesalahan.

4. *Software Testing* meliputi pengujian terhadap seluruh perilaku perangkat lunak.
5. *Software Maintenance* meliputi pada cara perawatan ketika perangkat lunak dioperasikan
6. *Software Engineering Management* berhubungan pada pengelolaan, pengukuran RPL, dan termasuk perencanaan terhadap perangkat lunak.
7. *Software engineering tools and methods* mencakup kajian teoritis tentang alat bantu dan metode RPL.
8. *Software engineering process* berhubungan pada definisi, implementasi, pengukuran, pengelolaan, perubahan dan perbaikan proses RPL.
9. *Software quality* berhubungan pada kualitas dan siklus dari perangkat lunak.

2.1.10 Bahasa Pemrograman Berorientasi Objek

Object Oriented Programming (OOP) adalah sebuah pendekatan di dalam pengembangan perangkat lunak, dimana struktur dari perangkat lunak dianalisis dari objek yang saling berinteraksi[16]. Di dalam interaksi dibutuhkan pertukaran informasi atau pesan bolak balik antara satu objek dengan objek yang lainnya. Setelah pesan diterima, objek dapat melakukan *action* atau *method*. *Object Oriented programming* memiliki enam konsep dasar, yaitu:

2.1.10.1 *Object*

Didalam bahasa pemrograman berorientasi objek. Objek adalah sebuah instansiasi dari sebuah kelas (*class*), yang terdiri dari sekumpulan variable dan *method – method* terkait. Contoh: sebuah *class* bisa terdiri dari *object* Pisang, Mangga, Apel dan lain – lain[17].

2.1.10.2 *Abstraction*

Abstraction merupakan kemampuan untuk menyaring sejumlah informasi dengan kata lain tidak perlu untuk mengetahui seluruh sifat-sifat asing suatu objek tetapi hanya menggunakan apa yang diperlukan saja[16]. Kemampuan abstaksi

diperlukan karena sangat sulit untuk memproses sejumlah informasi tanpa adanya kemampuan untuk menyaring sifat objek.

2.1.10.3 *Encapsulation*

Encapsulation adalah sebuah gagasan yang berguna untuk memastikan bahwa setiap bagian dari aplikasi berdiri sendiri dan tidak menggangu komponen – komponen lainnya, kecuali telah sesuai dengan cara yang ditentukan[16]. Saat mendefinisikan properti didalam kelas harus menentukan sifat dari properti atau *method* apakah bersifat *public, protected*, atau *private*.

1. *Public*

   Sebuah *property* atau *method* yang dinyatakan sebagai *public*, semua kode program yang berada dari luar *class* dapat mengaksesnya, termasuk *class* turunan.

2. *Protected*

   Sebuah *property* atau *method* yang dinyatakan sebagai *protected*, seluruh kode program yang berada dari luar kelas tidak dapat mengaksesnya, kecuali kelas teresebut adalah turunan.

3. *Private*

   Sebuah *property* atau *method* yang dinyatakan sebagai *private* ketika seluruh kode program dari luar *class* tidak dapat mengaksesnya, termasuk pada *class* turunannya.

2.1.10.4 *Inheritence*

Inheritance atau pewarisan digunakan untuk mengklasifikasikan objek menuruk karakteristik dan fungsi[17]. Perawarisan sendiri merupakan salah satu tujuan dasar dari OOP yaitu usabilitas dan keandalan. Dengan menggunakan *inheritance* membuat pekerjaan dengan menggunakan objek menjadi lebih mudah[16].

2.1.10.5 *Polymorphysm*

Polymorphysm adalah kemampuan dua buah objek berbeda untuk menerima pesan permintaan yang sama dengan cara yang berbeda pada tiap objek. Pada

implementasinya dapat menggunakan objek dengan nama yang sama. Artinya dapat menggunakan nama yang sama dalam konteks yang berbeda[16]. Contoh : kata "*polymorphysm*" berasal dari dua kata Yunani yang berarti "banyak bentuk". Kepala manusia adalah bentuk yang sangat berbeda dari kepala kuda, tetapi fungsinya pada dasarnya sama (makan, bernapas, melihat, dan sebagainya), jadi dapat menggunakan kata yang sama tanpa kebingungan. OOP menerapkan ini pada pemrograman dengan memungkinkan dalam pemberian nama yang sama pada metode dan properti dengan melakukan peran yang serupa didalam kelas yang berbeda.

2.1.10.6 *Agregation*

Agregation adalah ketika sebuah objek yang terdiri dari gabungan objek yang saling bekerjasama. Dengan menggunakan agregasi memungkinkan untuk mengimplementasikan proses bisnis sesusai dengan model[16].

2.1.11 Bahasa Pemrograman PHP

PHP *Preprocessor Hypertext* adalah sebuah bahasa pemrograman yang dapat digunakan untuk disisipkan kedalam HTML. Biasanya bahasa pemrograman PHP digunakan untuk memprogram situs *website* supaya menjadi dinamis. Bahasa pemrograman php termasuk kedalam sistem *server side* programming, yang mana program php akan dieksekusi atau dijalankan pada sisi server[16]. Bahasa PHP sangat cocok untuk digunakan didalam mengembangkan sebuah perangkat lunak berbasis web yang cukup kompleks, handal, dan cepat. Selain itu bahasa ini memiliki kelebihan yaitu mudah untuk digunakan, sederhana, mudah untuk dimengerti dan dipelajari.

2.1.12 *Programming Style* pada PHP

Setiap bahasa pemrograman tentunya memiliki aturan – aturan atau gaya penulisan tersendiri sesuai begitu juga dengan bahasa pemrograman PHP. *Programming style* bertujuan untuk menciptakan kode yang konsisten sehingga

mudah untuk dipahami. Berikut ini adalah programming style pada bahasa pemrograman PHP[18].

2.1.12.1 Gambaran Umum

1. *Tag* yang digunakan didalam *file* harus diawali oleh *tag* <?php atau <?=
2. Format pengkodean karakter menggunakan format UTF-8
3. Harus mendeklarasikan kelas, *symbol*, fungsi, konstanta dan lain – lain
4. *StudlyCaps* adalah suatu bentuk didalam pendeklarasian sebuah kelas
5. *camelCase* adalah suatu bentuk didalam mendeklarasikan sebuah metode (*method*)

2.1.12.2 Files

1. *Tag* PHP

   Didalam file PHP, kode program harus ditulis didalam tag <?php ?> atau tag <?= ?> untuk *short-echo*.

2. *Character Encoding*

   Kode PHP harus ditulis menggunakan format UTF-8

3. *Side Effects*

Sebuah file harus mendeklarasikan sebuah simbol *classes*, *functions*, *constants*, dan lain – lain. Maksudnya sebuah *class* harus berada pada *file* yang berbeda, agar tidak dieksekusi secara langsung ketika kelas tersebut tidak dibutuhkan. Jika suatu kelas yang dibutuhkan berada pada *file* yang berbeda dapat memanggilnya dengan menggunakan metode *include*() atau *require*(). Berikut ini adalah contoh deklarasi dari *file* dengan *side effects* yang harus dihindari.

Tabel 2.1 Dengan Side Effects

Dengan Side Effects
<?php // *side effect*: mengubah pengaturan **ini** ini_set('error_reporting', E_ALL);

```
// side effect: membuka sebuah file
include "file.php";

// side effect: menghasilkan keluaran
echo "<html>\n";

// deklarasi
function foo()
{
   // badan fungsi
        }
```

Dan berikut ini adalah file yang berisi deklarasi tanpa efek samping yang disarankan.

Tabel 2.2 Tanpa *Side Effects*

Tanpa side effects

<?php
// deklarasi
function foo()
{
// badan fungsi
}
 	
// pernyataan bersyarat adalah bukan *side effect*
if (! function_exists('bar')) {
function bar()
{
// badan fungsi

```
        }
            }
```

2.1.12.3 *Namespace* dan *Class Names*

1. *Namespaces* dan *class* harus berada didalam satu file tunggal dan *namespace* harus berada pada baris paling atas didalam sebuah file PHP.
2. *Class Name* harus dideklarasikan dalam bentuk *StudlyCaps*.
3. Untuk menggunakan namespace minimal versi PHP berada pada versi 5.3 atau diatasnya. Berikut ini contoh penggunaan *namespace* dan *class*

Tabel 2.3 Penulisan *Namespace* dan *Class*

Contoh Penulisan *Namespace* dan *Class*

<?php
// PHP 5.3 atau diatasnya:
namespace Vendor\Model;
class Foo
{
// class body
}

2.1.12.4 *Constant, Properties,* dan *Methods*

1. *Constant*

Variabel *constant* harus dideklarasikan menggunakan huruf besar, dan pemisahnya harus menggunakan tanda underscore **(_)**

Tabel 2.4 *Constant*

Constant
<?php namespace Vendor\Model;

```
class Foo
{
   const VERSION = '1.0';
   const DATE_APPROVED = '2012-06-01';
        }
```

2. *Properties*

Didalam panduan penamaan pada *properties* variabel dapat ditulis dalam bentuk $*StudlyCaps*, $*camelCase*, atau pun dalam bentuk *$under_score.*

3. *Method*

Untuk *method* harus dideklarasikan dalam bentuk $*camelCase*, contoh:

Tabel 2.5 Deklarasi Nama *Method*

Deklarasi Nama *Method*

```
<?php
    namespace Vendor\Package;

    class ClassName
    {
        public function fooBarBaz($arg1, &$arg2, $arg3 = [])
        {
            // method body
        }
            }
```

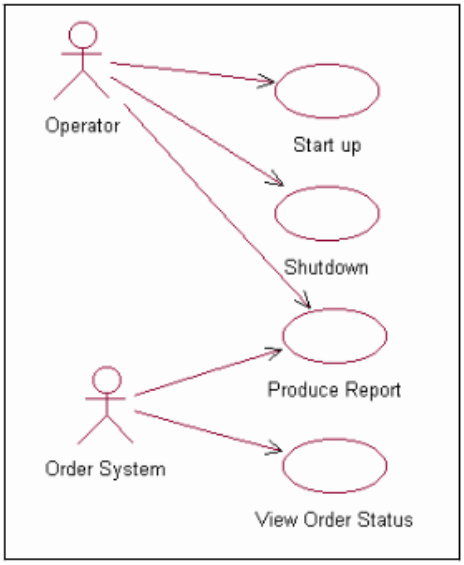
2.1.13 *Unified Modeling Language* (UML)

Alat pemodelan merupakan alat yang digunakan untuk melakukan pemodelan perangkat lunak, pemodelan yang dilakukan berupa memodelkan konsep, perancangan, maupun implementasi. Alat pemodelan dapat digunakan untuk membangun suatu model yang berisi bentuk objek dan konsep dari suatu gambaran yang akan disampaikan. *Unified Modelling Language* (UML) adalah

bahasa pemodelan grafis untuk memodelkan suatu sistem yang menggunakan konsep berorientasi objek[19].

2.1.13.1 *Use Case Diagram*

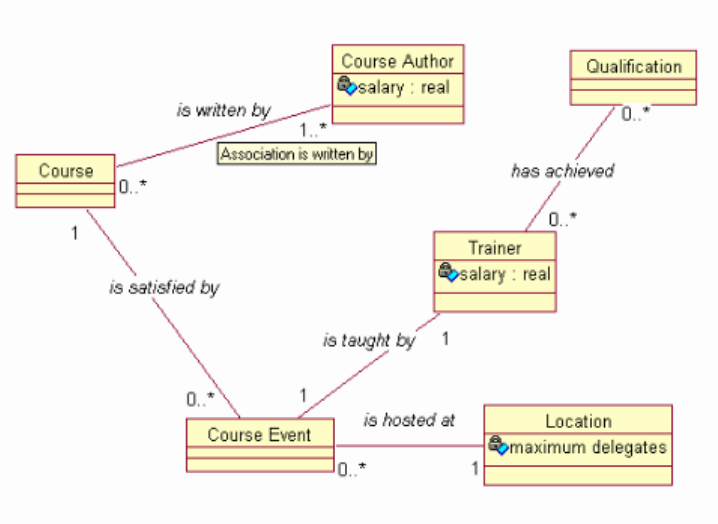
Use Case Diagram merupakan deskripsi dari fungsi-fungsi utama yang terdapat dalam sistem dari sudut pandang pengguna[19]. Diagram pemodelan yang memberikan gambaran apa yang harus dilakukan oleh sistem berdasarkan fungsionalitas yang harus ada pada sistem yang dibangun. Diagram ini sangat befungsi dari menganalisis hingga pengembangan sistem serta membantu dalam memahami kebutuhan yang diperlukan.



Gambar 2.6 Ilustrasi *Use Case Diagram*

2.1.13.2 *Class Diagram*

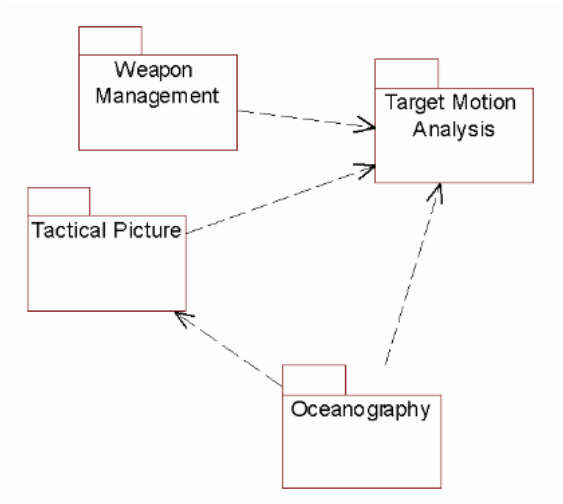
Class diagram merupakan salah satu pemodelan pada UML yang menggambarkan struktur kelas – kelas yang akan dibuat untuk membangun sistem[19]. *Class diagram* mendeskripsikan tipe-tipe objek yang digunakan dalam sistem dan keterhubungan antar kelas. Setiap objek pada kelas memiliki hak akses terdapat tiga jenis hak akses, yaitu *protected*, *public*, dan *private*.



Gambar 2.7 Ilustrasi *Class Diagram*

2.1.13.3 *Package Diagram*

Dalam UML *package*, kelas – kelas yang ada kemudian dimodelkan kedalam bentuk *package*, kegunaan *package* yaitu untuk menghindari kesamaan nama kelas. Dalam *class library* sendiri, *package* memudahkan pengembang aplikasi dalam memahami penstrukturan kelas[19]. *Package diagram* digunakan untuk mengetahui ketergantungan antar bagian dari suatu perangkat lunak, mencari masalah pada perangkat lunak, dan menentukan urutan kompilasi.



Gambar 2.8 Ilustrasi *Package Diagram*

2.1.14 Pengujian *Class Library*

Pengujian yang dapat dilakukan untuk menguji sebuah *class library* adalah dengan melakukan *unit testing*, *integration testing*, dan *acceptance testing*.

2.1.14.1 *Unit Testing*

Unit Testing adalah sebuah metode verifikasi pada sebuah perangkat lunak yang dilakukan oleh *programmer* untuk menguji suatu unit / area spesifik dari suatu fungsionalitas kode yang telah dibangun pada tahapan sebelumnya[13]. Unit – unit ini bisa berupa fungsi atau sekumpulan fungsi, kelas atau sekumpulan kelas didalam suatu file pemrograman terstruktur. *Unit Testing* juga dapat dilakukan setelah *programmer* menuliskan kode fungsi atau *method* yang terdapat didalam suatu kelas, *unit testing* juga dapat dilakukan setelah melakukan penambahan fungsionalitas atau *method* baru pada sebuah kelas. Salah satu yang dapat digunakan untuk melakukan *unit testing* adalah PHPUnit[20] untuk bahasa pemrograman PHP. Berikut ini adalah salah satu contoh cara penggunaan PHPUnit. Misalkan kelas yang akan diuji adalah kelas dengan nama *Test* dengan *source code* seperti berikut dapat dilihat pada Tabel 2.6

Tabel 2.6 Kelas *Test*

Kelas Test

<?php
namespace APP;
 	
class Test
{
public function testNIM($NIM)
{
return $NIM;
}
public function testNama($nama)

```
 {
   return $nama;
 }
}
?>
```

Untuk melakukan pengujian pada pada kelas *Test* maka dibuat sebuah kelas pengujian dengan nama *TestForClassTest* yang merupakan kelas turunan dari tools pengujian PHPUnit. Pengujian ini bertujuan untuk membandingkan nilai keluaran dari *method – method* yang terdapat didalam kelas *Test*, jika nilai keluaran sama dengan nilai ekspektasi maka pengujian berhasil. Jika nilai keluaran dengan nilai ekspektasi tidak sama maka pengujian belum berhasil. Berikut ini adalah *source code* dari class *TestForClassTest* untuk menguji kelas *Test* yang telah dibuat sebelumnya dapat dilihat pada Tabel 2.7

Tabel 2.7 Kelas *TestForClassTest*

Kelas *TestForClassTest*

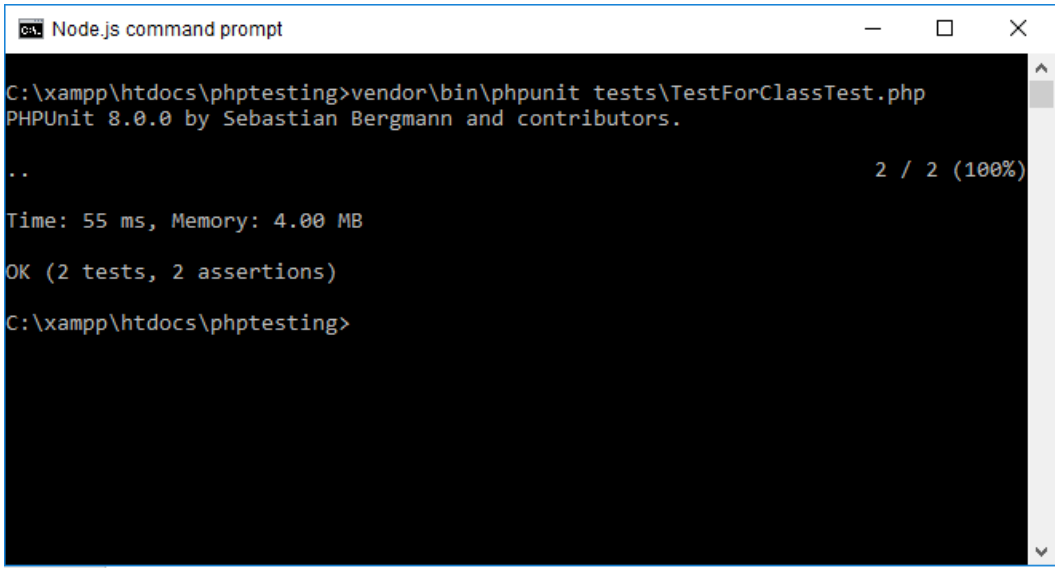
<?php
 	
use App\Test;
 	
class TestForClassTest extends PHPUnit\Framework\TestCase
{
public function testForTestNama()
{
// Instansiasi Kelas Test
$test = new Test();
// Passing Data Nama
$resultNama = $test->testNama("Willy Zondri");
// Nama Keluaran Yang Inginkan

```
    $expectedNama = "Willy Zondri";
    // Method untuk menguji perbandingan data keluaran
    // dengan data ekspektasi
    $this->assertEquals($expectedNama, $resultNama);
  }

  public function testForTestNIM()
  {
    // Instansiasi Kelas Test
    $test = new Test();
    // Passing Data NIM
    $resultNIM = $test->testNIM("10115565");
    // NIM Keluaran Yang Inginkan
    $expectedNIM = "10115565";

    // Method untuk menguji perbandingan data keluaran
    // dengan data ekspektasi
    $this->assertEquals($expectedNIM, $resultNIM);
  }
}
?>
```

Langkah selanjutnya adalah melakukan pengetesan dengan menggunakan tools PHPUnit dengan melakukan eksekusi terhadap kelas *TestForClassTest* untuk membuktikan bahwa kelas *Test* yang dibuat telah berjalan dengan benar. Berikut ini adalah hasil dari pengujian kelas Test dengan menggunakan PHPUnit, dapat dilihat pada Gambar 2.9



Gambar 2.9 Contoh Hasil Pengujian dengan PHPUnit

2.1.14.2 *Integration Testing*

Integration testing adalah sebuah tahapan pengujian yang dilakukan setelah tahapan pengujian unit. Didalam *integration testing* dilakukan penggabungan dua unit atau lebih pada perangkat lunak[21]. Pada tahapan *integration testing* input berupa modul - modul yang telah diuji pada tahap *unit testing*, diproses kedalam sub *integration testing* berupa *user interface testing*, *interaction testing*, dll.

2.1.14.3 *Acceptance Testing*

Pengujian *Acceptance* adalah sebuah tahapan pengujian yang dilakukan oleh pembuat kode program dengan pengguna untuk memvalidasi kode program yang telah dibangun, apakah kode program yang dibuat dapat bekerja dengan baik. *Acceptence testing* bertujuan untuk mengetahui kepuasan pengguna terhadap perangkat lunak yang dibangun[13].