

**LAPORAN LENGKAP**  
**PRAKTIKUM PEMROGRAMAN BERORIENTASI OBJEK**



**OLEH :**

**NAMA : RAFLI SYAHPUTRA**  
**NIM : F1G120030**  
**KELOMPOK : II (DUA)**

**ASISTEN PENGAMPU :**

**WAHID SAFRI JAYANTO (F1G117059)**

**PROGRAM STUDI ILMU KOMPUTER**  
**JURUSAN MATEMATIKA**  
**FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM**  
**UNIVERSITAS HALU OLEO**  
**KENDARI**  
**2021**

**HALAMAN PENGESAHAN  
LAPORAN PRAKTIKUM**



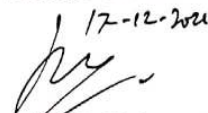
OLEH:

RAFLI SYAHPUTRA (F1G120030)

Laporan praktikum Pemrograman Berorientasi *Object* ini disusun sebagai tugas akhir menyelesaikan praktikum Pemrograman Berorientasi *Object* sebagai salah satu syarat lulus matakuliah Pemrograman Berorientasi *Object*. Menerangkan bahwa yang tertulis dalam laporan lengkap ini adalah benar dan dinyatakan telah memenuhi syarat.

Kendari, 17- Desember 2021

Menyetujui

Asisten Praktikum  
17-12-2021  
  
Wahid Safri Jayanto  
F1G117059

Praktikan  
  
Rafli Syahputra  
F1G120030

## DAFTAR ISI

LEMBAR PENGESAHAN .....	ii
DAFTAR ISI .....	iii
DAFTAR TABEL .....	v
DAFTAR GAMBAR .....	vi
KATA PENGANTAR .....	vii
1.1 Pertemuan pertama .....	1
1.1 Alat dan Bahan .....	1
1.1.2 Pengenalan <i>PBO</i> .....	1
1.1.3 Pengenalan <i>PHP</i> .....	3
2.1 Pertemuan ke dua .....	5
2.1.1 <i>Class</i> .....	5
2.1.2 <i>Method</i> .....	5
2.1.3 <i>Constructor</i> .....	7
2.1.4 <i>Modifier</i> .....	8
2.1.5 <i>Property</i> .....	8
2.1.6 <i>Object</i> .....	9
2.1.7 <i>Atribut</i> .....	9
2.1.8 <i>Composer</i> .....	10
2.1.9 <i>Laravel</i> .....	11
2.1.10 <i>Constructor</i> dan <i>Descructor</i> .....	12
2.1.11 <i>Abstract Class</i> dan <i>Abstract Method</i> .....	14
2.1.12 <i>Interface</i> .....	16
2.1.13 <i>Resursife Function</i> .....	17
3.1 Pertemuan ke tiga .....	19
3.1.1 Proyek Pertama tentang <i>CRUD</i> .....	20
3.1.2 Proyek Ke dua .....	22
4.1 Pertemuan ke empat .....	25
4.1.1 <i>ERD</i> .....	25
4.1.2 <i>DFD</i> .....	28

4.1.3 <i>Interface</i> .....	33
4.1.4 Projek akhir penyewaan kos.....	35
DAFTAR PUSTAKA .....	38

## DAFTAR TABEL

<b>Tabel 1.1</b> Tabel alat dan Bahan .....	1
---	---

## DAFTAR GAMBAR

<b>Gambar 3.1</b> Halaman utama <i>CRUD</i> .....	20
<b>Gambar 3.2</b> Halaman <i>contacts</i> .....	21
<b>Gambar 3.3</b> Halaman utama .....	22
<b>Gambar 3.4</b> Halaman <i>login</i> .....	22
<b>Gambar 3.5</b> Halaman berhasil <i>login</i> .....	23
<b>Gambar 3.6</b> Halaman manager .....	24
<b>Gambar 4.1</b> Contoh <i>ERD</i> .....	26
<b>Gambar 4.2</b> Contoh <i>DFD</i> level 0 .....	29
<b>Gambar 4.3</b> Contoh <i>DFD</i> level 1 .....	30
<b>Gambar 4.4</b> Halaman utama kos.....	35
<b>Gambar 4.5</b> Halaman utama admin .....	36
<b>Gambar 4.6</b> Halaman informasi pemilik .....	37

## KATA PENGANTAR



Puji syukur kami panjat kankehadirat Allah SWT, karena berkat rahmat dan hidayah-nya penyusunan laporan Pemrograman berorientasi objek dapat di selesaikan dengan tepat waktu tanpa ada halangan yang berarti.

Laporan ini disusun berdasarkan kebutuhan mahasiswa. Dengan demikian, Materi yang dibahas dalam laporan ini sudah selesai dengan kebutuhan mahasiswa. Materi yang kami susun dalam laporan ini kami susun dengan sistematis yang baik dan jelas di tulis dengan bahasa yang mudah dimengerti dan dipahami.

Akhir kata, kami menyadari "takadagading yang retak" juga laporan ini tidak lepas dari kekurangan. Oleh karena itu, kami mengharap kritik dan saran dari pengguna laporan ini. Sekian terimakasih, *wabillahaufikwalhidayah, WassalamuAlaikum Warahumatullahi Wabarakatu.*

Kendari, Desember 2021

Penulis

## A.1 Pertemuan pertama

### 1.1.1 Alat dan bahan.

Adapun alat dan bahan yang di gunakan pada praktikum kali ini adalah sebagai berikut:

Alat Dan Bahan	Penjelasan
Leptop	Sebagai tempat untuk menyimpan data, untuk mengerjakan projek dan sebagai tempat untuk mengoding.
<i>Xampp</i>	Sebagai penghubung antara <i>chrome</i> dan <i>Vscode</i> .
<i>Vscode</i>	Sebagai tempat mengoding sebuah program.
<i>Chrome</i>	Sebagai tempat untuk melihat hasil <i>running</i> dari program yang telah di

**Tabel 1.1** Tabel penggunaan alat dan bahan

### 1.1.2 Pengenalan *PBO*

Pemrograman *berorientasi objek Object Oriented Programming (OOP)* adalah paradigma pemrograman yang berorientasikan kepada objek yang merupakan suatu metode dalam pembuatan program, dengan tujuan untuk menyelesaikan kompleksnya berbagai masalah program yang terus meningkat. Objek adalah *entitas* yang memiliki *atribut*, karakter (*bahavour*) dan kadang



kala disertai kondisi (*state*). Pemrograman berorientasi objek ditemukan pada Tahun 1960, dimana berawal dari suatu pembuatan program yang terstruktur (*structured programming*). Metode ini dikembangkan dari bahasa C dan Pascal. Dengan program yang terstruktur inilah untuk pertama kalinya kita mampu menulis program yang begitu sulit dengan lebih mudah.

Ide dasar pada *OOP* adalah mengkombinasikan data dan fungsi untuk mengakses data menjadi sebuah kesatuan unit yang dikenal dengan nama objek. Objek adalah struktur data yang terdiri dari bidang data dan metode bersama dengan interaksi mereka untuk merancang aplikasi dan program komputer. Semua data dan fungsi di dalam paradigma ini dibungkus dalam kelas-kelas atau objek-objek. Setiap objek dapat menerima pesan, memproses data, dan mengirim pesan ke objek lainnya.

Pemrograman berorientasi objek dalam melakukan pemecahan suatu masalah tidak melihat bagaimana cara menyelesaikan suatu masalah tersebut (terstruktur) tetapi objek-objek apa yang dapat melakukan pemecahan masalah tersebut. Sebagai contoh sebuah departemen yang memiliki seorang manager, sekretaris, petugas administrasi data dan lainnya. Jika manager ingin memperoleh data dari bagian administrasi maka manager tersebut tidak harus mengambilnya langsung tetapi dapat menyuruh petugas bagian administrasi untuk mengambilnya. Pada kasus tersebut seorang manager tidak harus mengetahui bagaimana cara mengambil data tersebut tetapi manager bisa

mendapatkan data tersebut melalui objek petugas administrasi. Jadi untuk menyelesaikan suatu masalah dengan kolaborasi antar objek-objek yang ada karena setiap objek memiliki deskripsi tugasnya sendiri.

Pemrograman berorientasi objek bekerja dengan baik ketika dibarengi dengan *Objek-Oriented Analysis And Design Process (OOAD)*. Jika membuat program berorientasi objek tanpa *OOAD*, seperti membangun rumah tanpa terlebih dahulu penganalisis apa saja yang dibutuhkan oleh rumah itu, tanpa perencanaan, tanpa *blue-print*, tanpa menganalisis ruangan apa saja yang diperlukan, beberapa besar rumah yang akan dibangun dan sebagainya. (Douglas, 1992).

#### 1.1.3 Pengenalan *PHP*

Sejarah Bahasa Pemrograman *PHP* Menurut *wikipedia*, Pada awalnya *PHP* merupakan kependekan dari *Personal Home Page (Situs personal)*. *PHP* pertama kali dibuat oleh Rasmus Lerdorf pada tahun 1995. Pada waktu itu *PHP* masih bernama *Form Interpreted (FI)*, yang wujudnya berupa sekumpulan *skrip* yang digunakan untuk mengolah data formulir dari *web*. Selanjutnya Rasmus merilis *kode* sumber tersebut untuk umum dan menamakannya *PHP/FI*. Dengan perilsan *kode* sumber ini menjadi sumber terbuka, maka banyak pemrogram yang tertarik untuk ikut mengembangkan *PHP*. Pada November 1997, dirilis *PHP/FI 2.0*. Pada *rilis* ini, *interpreter PHP* sudah diimplementasikan dalam program C. Dalam *rilis* ini disertakan juga

modul-modul ekstensi yang meningkatkan kemampuan *PHP/FI* secara signifikan. I. (Triwansyah yuliano, 2007).

Pengenalan *PHP* 20 Pada tahun 1997, sebuah perusahaan bernama *Zend* menulis ulang *interpreter PHP* menjadi lebih bersih, lebih baik, dan lebih cepat. Kemudian pada Juni 1998, perusahaan tersebut merilis *interpreter* baru untuk *PHP* dan meresmikan rilis tersebut sebagai *PHP 3.0* dan singkatan *PHP* diubah menjadi akronim berulang *PHP: Hypertext Preprocessing*. Pada pertengahan tahun 1999, *Zend* merilis *interpreter PHP* baru dan rilis tersebut dikenal dengan *PHP 4.0*. *PHP 4.0* adalah versi *PHP* yang paling banyak dipakai pada awal abad ke-21. Versi ini banyak dipakai disebabkan kemampuannya untuk membangun aplikasi *web* kompleks tetapi tetap memiliki kecepatan dan stabilitas yang tinggi. Pada Juni 2004, *Zend* merilis *PHP 5.0*. Dalam versi ini, inti dari *interpreter PHP* mengalami perubahan besar. Versi ini juga memasukkan model pemrograman berorientasi objek ke dalam *PHP* untuk menjawab perkembangan bahasa pemrograman ke arah paradigma berorientasi objek. *Server web* bawaan ditambahkan pada versi 5.4 untuk mempermudah pengembang menjalankan kode *PHP* tanpa meng-install *software* server. Pada saat buku ini ditulis, *PHP* telah mencapai versi 7.2 dengan penambahan ekstensi dan perbaikan performa yang menjanjikan. (Triwansyah yuliano, 2007).

## 2.1 Pertemuan ke dua

### 2.1.1 *Class*

*Class* merupakan suatu *blueprint* atau cetakan untuk menciptakan suatu *instant* dari *object*. *Class* juga merupakan grup suatu *object* dengan kemiripan *attributes/properties*, *behaviour* dan relasi ke *object* lain. (Gunadarman, 2013)

Contoh *syntax*:

```
<?php  
  
//Cara penulisan class OOP PHP - www.malasngoding.com  
class nama_class{  
  
    //isi dari class ini  
  
}  
  
?>
```

### 2.1.2 *Method*

*Method* merupakan suatu operasi berupa fungsi-fungsi yang dapat dikerjakan oleh suatu *object*. *Method* didefinisikan pada *class* akan tetapi dipanggil melalui *object*. Metode menentukan perilaku objek, yakni apa yang terjadi ketika objek itu dibuat serta berbagai operasi yang dapat dilakukan objek sepanjang hidupnya. Ada 4 (Empat) bagian dasar yang dimiliki metode antara lain:

1. Nama *metode*
2. Tipe Objek atau tipe *primitive* yang dikembalikan metode.
3. Daftar parameter.
4. Badan atau isi metode.

Tiga bagian pertama mengindikasikan informasi penting tentang metode itu sendiri. Dengan kata lain, nama metode tersebut=metode lain dalam program. Dalam java kita dapat memiliki metode-metode berbeda yang memiliki nama sama tetapi berbeda tipe kembalian atau daftar argumennya, sehingga bagian-bagian definisi metode ini menjadi penting. Ini disebut *overloading* metode(proses yang berlebihan pada suatu metode). Untuk menjalankan program yang memiliki sifat *polymorphism* tersebut, diperlukan suatu kemampuan *overloading*, yaitu suatu kemampuan untuk menentukan fungsi yang mana yang harus digunakan atau dijalankan jika terdapat nama fungsi yang sama. *Polimorfisme* bisa diartikan seperti kemampuan suatu *variable* untuk mengubah perangkat sesuai dengan objek hasil *instansiasi* yang digunakan. *Polimorfisme* membiarkan lebih dari 1 objek dari *sub class* *sub class* dan diperlakukan sebagai objek dari super class tunggal. (Gunadarman, 2013)

#### Contoh Syntax

```
<?php
//Cara penulisan class dan property OOP PHP -
www.malasngoding.com
class mobil{
    // property oop
    var $warna;
    var $merek;
    var $ukuran;
    //method oop
    function maju(){
        //isi method
    }
    function berhenti(){
        //isi mehod
    }
}
?>
```

### 2.1.3 *Constructor*

*Constructor* adalah *Constructor* merupakan suatu method yang akan memberikan nilai awal pada saat suatu objek dibuat. Pada saat program dijalankan. (Gunadarman, 2013) , *constructor* akan bekerja dengan *constructor*, hal mendasar yang perlu diperhatikan, yaitu :

- 1) Nama *Constructor* sama dengan nama *Class*.
- 2) Tidak ada *return type* yang diberikan kedalam *Constructor Signature*.
- 3) Tidak ada return statement, didalam tubuh *constructor*.

Contoh Program:

```
class Kotak {
    double panjang;
    double lebar;
    double tinggi;
    //Mendefenisikan constructor dengan parameter
    kotak(double p, double l, double t) {
        panjang = p;
        lebar = l;
        tinggi = t;
    }
    double hitungVolume() {
        return (panjang * lebar * tinggi)
    }
}

class DemoConstructor2 {
    public static void main(String[] args) {
        kotak k1, k2;
        k1 = new kotak(4, 3, 2)
        k2 = new kotak (6, 5, 4)
        system.out.println("volume k1 = " + k1.hitungVolume())
    }
    system.out.println("volume k2 = " + k2.hitungVolume())
}
```

#### 2.1.4 *Modifier*

*Modifier* adalah kata, *phrase* , atau *clause* yang berfungsi sebagai *adjective* atau *adverb* yang menerangkan kata atau kelompok kata lain. Sebagai *adjective* dan *adverb* ketika berfungsi sebagai *adjective* ( dapat berupa *simple adjective*, *adjective phrase*, *clause participle*, *infinitive* ), *modifier* menerangkan *noun*, sedangkan ketika berfungsi sebagai *adverb* ( dapat berupa *simple adverb* , *adverb phrase*, *clause*, *preposition phrase*, *infinitive*), kata ini menerangkan *verb*, *adjective* atau *adverb* lain. . (Gunadarman, 2013)

Contoh Program:

```
Public class bank balance
{
    public String owner
    public int balance

    public bank_balance(String name, int dollars )
    {
        owner = name;

        if(dollars > = 0)
            balance = dollars;
        else
            dollars =0;
    }
}
```

#### 2.1.5 *Property*

*Property* (atau disebut juga dengan *atribut*) adalah data yang terdapat dalam sebuah *class*. Melanjutkan analogi tentang laptop, *property* dari laptop bisa berupa *merk*, *warna*, *jenis processor*, *ukuran layar*, dan lain-lain. (Andre 2015).

Contoh Syntax:

```
<?php
class laptop {
    var $pemilik;
    var $merk;
    var $ukuran_layar;
    // lanjutan isi dari class laptop...
}
?>
```

#### 2.1.6 Object

*Object* atau Objek adalah hasil cetak dari *class*, atau hasil ‘konkrit’ dari *class*. Jika menggunakan *analogi class* laptop, maka objek dari *class* laptop bisa berupa: *laptop\_andi*, *laptop\_anto*, *laptop\_duniaikom*, dan lain-lain. Objek dari *class* laptop akan memiliki seluruh ciri-ciri laptop, yaitu *property* dan *method*-nya. Proses ‘mencetak’ objek dari *class* ini disebut dengan ‘*instansiasi*’ (atau *instantiation* dalam bahasa inggris). Pada *PHP*, proses *instansiasi* dilakukan dengan menggunakan *keyword* ‘*new*’. Hasil cetakan *class* akan disimpan dalam *variable* untuk selanjutnya digunakan dalam proses program. (Andre 2015).

Contoh Syntax:

```
<?php
class laptop {
    //... isi dari class laptop
}
```

#### 2.1.7 Atribut

*Atribut* merupakan nilai data yang terdapat pada suatu *object* di dalam *class*. *Attribute* mempunyai karakteristik yang membedakan



*object* yang satu dengan *object* yang lainnya. Contoh : pada *Class* Buah terdapat *attribute*: warna, berat. Misalkan pada *object* mangga: warna berisi kuning dan berat 0.5 kg dan pada *object* apel : warna merah dan berat 0.6 kg (Andre 2015).

#### 2.1.8 *Composer*

*Composer* merupakan *tool* yang di dalamnya terdapat *dependencies* dan, kumpulan *library*. Seluruh *dependencies* disimpan menggunakan format *file composer.json* sehingga dapat ditempatkan di dalam folder utama *website*. Inilah mengapa *composer* terkadang dikenal dengan *dependencies management*. *Composer* adalah *tools dependency manager* pada *PHP*, *Dependency* (ketergantungan) sendiri diartikan ketika *project PHP* yang kamu kerjakan masih membutuhkan atau memerlukan *library* dari luar. *Composer* berfungsi sebagai penghubung antara *project PHP* kamu dengan *library* dari luar. *Composer* adalah *package-manager* (di level aplikasi) untuk bahasa pemrograman *PHP*. Menawarkan standarisasi cara pengelolaan *libraries* dan *software dependencies* dalam proyek *PHP*. Dengan *Composer* kita tidak perlu repot-repot lagi *mendownload source code* pustaka yang kita butuhkan secara manual, lalu memasangnya di aplikasi kita, lalu *mengupdate*-nya secara manual jika ada versi baru. Itu semua tidak perlu lagi karena *Composer* bisa menangani semua proses tersebut dengan mudah. (Nurul Huda, 2020).

### Cara Penggunannya:

```
<?php
// misalkan ini adalah file index.php

require_once __DIR__ . '/vendor/autoload.php';
$fb = new \Facebook\Facebook([
    'app_id' => '{app-id}',
    'app_secret' => '{app-secret}',
    'default_graph_version' => 'v2.10',
    //'default_access_token' => '{access-token}', //
    optional
]);
```

#### 2.1.9 *Laravel*

*Laravel* adalah satu-satunya *framework* yang membantu Anda untuk memaksimalkan penggunaan *PHP* di dalam proses pengembangan *website*. *PHP* menjadi bahasa pemrograman yang sangat dinamis, tapi semenjak adanya *Laravel*, dia menjadi lebih *powerful*, cepat, aman, dan simpel. Setiap *rilis versi* terbaru, *Laravel* selalu memunculkan teknologi baru di antara *framework PHP* lainnya. *Laravel* diluncurkan sejak tahun 2011 dan mengalami pertumbuhan yang cukup *eksponensial*. Di tahun 2015, *Laravel* adalah *framework* yang paling banyak mendapatkan bintang di *Github*. Sekarang *framework* ini menjadi salah satu yang populer di dunia, tidak terkecuali di Indonesia. *Laravel* fokus di bagian *end-user*, yang berarti fokus pada kejelasan dan kesederhanaan, baik penulisan maupun tampilan, serta menghasilkan *fungsionalitas* aplikasi *web* yang bekerja sebagaimana mestinya. Hal ini membuat *developer* maupun perusahaan menggunakan *framework* ini untuk membangun apa pun, mulai dari *proyek* kecil hingga skala

perusahaan kelas atas. *Laravel* mengubah pengembangan website menjadi lebih *elegan, ekspresif*, dan menyenangkan, sesuai dengan jargonnya “*The PHP Framework For Web Artisans*”. Selain itu, *Laravel* juga mempermudah proses pengembangan website dengan bantuan beberapa fitur unggulan, seperti *Template Engine, Routing*, dan *Modularity*. (Yasin k, 2019).

#### 2.1.10 *Constructor* dan *Destructor*

*Constructor* adalah sebuah *method* khusus yang dieksekusi ketika sebuah *class* diinstansiasi. *Constructor* digunakan untuk mempersiapkan *object* ketika *keyword new* dipanggil. Dalam *constructor* kita dapat melakukan apapun yang kita dapat lakukan pada *method* biasa namun tidak bisa mengembalikan *return value*. Muncul pertanyaan, kenapa *constructor* tidak dapat mengembalikan *return value*? Ya jelas lah tidak bisa mengembalikan *return value*, kan *keyword new* itu sudah mengembalikan berupa *object* dari *class* yang diinstansiasi. Masa kemudian *constructor* mengembalikan lagi nilai yang sesuai? Misalnya, kita punya *class A* maka ketika menginisiasi *class A* tersebut dengan *keyword new* kedalam variable \$a maka saat itu sebenarnya telah mengembalikan nilai berupa *object A* ke dalam *variable \$a* tersebut. Bagaimana jadinya jika didalam *constructor* kita dapat mengembalikan nilai dan kemudian membuat *constructor* dengan mengembalikan nilai integer 1 misalnya. Maka yang terjadi ketika X. *Constructor* dan *Destructor* 79 kita melakukan *instansiasi class A* dan

fungsi *constructor* dipanggil, alih-alih kita mendapatkan *object* A yang ada kita justru mendapatkan integer 1 . . (Ahmad Muhardian 2019).

*Destructor* adalah sebuah *method* khusus yang dieksekusi ketika sebuah *object* dihapus dari *memory*. Secara mudah, *destructor* adalah kebalikan dari *constructor*. Sama seperti pada *constructor*, *PHP* juga akan membuat *destructor* tanpa parameter dan tanpa *logic* jika kita tidak mendefinisikan *destructor* secara *eksplisit*. Berbeda dengan *constructor* yang dapat memiliki parameter, *destructor* tidak dapat memiliki parameter dan hanya dapat berisi *logic*. (Ahmad Muhardian 2019).

Contoh *syntax Denstructor*:

```
class User {
public:
    User( String *username ); // <-- ini constructor
    ~User(); // <-- ini destructor.
private:
    String username;
    String password;
};
```

Contoh *syntax Constructor*:

```
package konstruktor;

public class User {
    public String username;
    public String password;

    public User(String username, String password){
        this.username = username;
        this.password = password;
    }
}
```

#### 2.1.11 *Abstract Class* dan *Abstract Method*

*Abstract class* adalah sebuah *class* dalam *OOP* yang tidak dapat diinstansiasi atau dibuat *object*-nya. *Abstract class* biasanya berisi fitur-fitur dari sebuah *class* yang belum implementasikan. Seperti pada pembahasan sebelumnya tentang *class Connection* dimana kita harus membuat implementasi dari *class* tersebut dengan *meng-extends*-nya menjadi *MySQLConnection* dan *PostgreSQLConnection* . Karena *abstract class* harus diimplementasikan melalui proses pewarisan, maka dalam *abstract class* berlaku aturan-aturan yang ada pada konsep pewarisan yang telah kita bahas sebelumnya. Didalam sebuah *abstract class* kita dapat membuat *property* dan *method* yang nantinya dapat digunakan oleh *child class*. Tentu saja *property* dan *method* yang dapat digunakan oleh *child class* adalah *property* dan *method* yang memiliki *visibilitas protected* dan *public*. (Andre, 2018).

### *Syntax Abstract Class:*

```
<?php
abstract class komputer {
    // isi dari class komputer
}
?>
```

Sama seperti *abstract class*, *abstract method* adalah sebuah *method* yang harus diimplementasikan oleh *child class*. *Abstract method* hanya ada pada *abstract class* dan *interface* (akan dibahas secara terpisah). Bila biasanya setiap *method* yang kita buat pasti mempunyai kurawal {}, pada *abstract method* hal tersebut tidak dapat ditemui karena *abstract method* adalah sebuah *method* yang tidak memiliki *body* atau badan *method*. Pada *child class*, *abstract method* harus didefinisikan ulang dan kita tidak dapat menggunakan *keyword parent* untuk memanggil *abstract method* pada *parent class*. Bila kita melakukan hal tersebut maka akan terjadi *error*. (Andre, 2018).

### *Contoh Syntax Abstract Method:*

```
<?php
abstract class komputer {
    abstract public function lihat_spec();
}
?>
```

Kegunaan *Abstract Class* dan *Abstract Method* Yaitu Secara mudah *abstract class* dan *abstract method* berguna untuk memastikan *child*

*class* memiliki fitur-fitur yang telah ditentukan sebelumnya. *Abstract class* akan sangat berguna pada saat kita membahas tentang *type hinting* atau parameter hinting. Dengan *abstract class* dan *abstract method* kita bisa lebih percaya diri ketika memanggil sebuah *method* karena dapat dipastikan *method* tersebut dimiliki *child class*. (Andre, 2018).

#### 2.1.12 Interface

Dalam pemrograman berbasis objek, *interface* adalah sebuah *class* yang semua *method*-nya adalah *abstract method*. Karena semua *method*-nya adalah *abstract method* maka *interface* pun harus diimplementasikan oleh *child class* seperti halnya pada *abstract class*. Hanya saja bila kita sebelumnya menggunakan *keyword extends* untuk mengimplementasikan sebuah *abstract class*, maka pada *interface* kita menggunakan *keyword implements* untuk mengimplementasikan sebuah *interface*.

Di era *milennial* seperti sekarang ini penggunaan *interface* sangat masif. Banyak *framework* dan *library* yang kalau kita mau membaca *source code*-nya maka akan mudah sekali bagi kita untuk menemukan *interface*. Penggunaan *interface* tidak lain karena fitur yang dimiliki *interface* itu sendiri yaitu sebagai *hirarki* tertinggi pada *parameter casting* (akan dibahas pada bab tersendiri) dimana setiap *object* yang mengimplementasikan sebuah *interface* akan *valid* jika dimasukkan kedalam *method* yang menggunakan *interface* tersebut sebagai *type hinting* atau *parameter casting*. Seperti pada *framework Laravel*,

dimana *interface* akan sangat mudah ditemukan pada folder *Contracts* seperti nampak pada *Github* repository *Laravel* berikut. Pada paradigma pemrograman modern, ada istilah "*interface as contract*" yang maksudnya adalah *interface* digunakan pada parameter *casting* sebagai pengikat bahwa *object* yang akan XV. *Interface* 116 dimasukkan kedalam *method* pasti memiliki fitur-fitur atau *methodmethod* yang didefinisikan pada *interface* tersebut. Sehingga dengan menggunakan *interface* tersebut sebagai paramter *casting* pada *method* maka didalam *method* tersebut kita bisa dengan percaya diri untuk menggunakan *method-method* yang ada pada *interface* tanpa takut terjadi *error undefined method* .

#### 2.1.13 Recursive Function

*Recursive function* adalah sebuah *function* yang memanggil dirinya sendiri dalam badan *function*-nya. *Recursive function* biasanya dipakai untuk menyelesaikan permasalahan yang mempunyai pola dasar yang berulang seperti perhitungan *faktorial*. Keuntungan menggunakan *recursive function* adalah mempersingkat *code* yang kita tulis. Namun yang perlu diperhatikan adalah bahwa kita harus benar-benar paham bagaimana *function* tersebut bekerja. Jika kita tidak paham bagaimana *nested call* yang terjadi didalam *recursive function* bisa saja bukan solusi singkat yang didapat tapi justru permasalahan yang justru kita sama sekali tidak mengetahui bagaimana cara mengatasinya. *Recursive function* sangat perlu dipelajari dan dipahami oleh programmer karena



dalam banyak kasus *recursive function* terbukti mampu menyelesaikan permasalahan yang *kompleks* dan dinamis.

### 3.1 Pertemuan ke tiga

#### A. DEFINISI *CRUD*

*CRUD* adalah singkatan yang berasal dari *Create*, *Read*, *Update*, dan *Delete*, dimana keempat istilah tersebut merupakan fungsi utama yang nantinya diimplementasikan ke dalam basis data. Empat poin tersebut mengindikasikan bahwa fungsi utama melekat pada penggunaan *database* relasional beserta aplikasi yang mengelolanya, seperti *Oracle*, *MySQL*, *SQL Server*, dan lain – lain. Jika dihubungkan dengan tampilan antarmuka (*interface*), maka peran *CRUD* sebagai fasilitator berkaitan dengan tampilan pencarian dan perubahan informasi dalam bentuk formulir, tabel, atau laporan. Nantinya, akan ditampilkan dalam *browser* atau aplikasi pada perangkat komputer *user*. Istilah ini pertama kali diperkenalkan oleh James Martin pada tahun 1983 dalam bukunya yang berjudul “*Managing the Database Environment*”.

#### B. FUNGSI DARI *CRUD*

Terdapat empat *poin* penting dari *akronim* fungsi *CRUD* untuk mengembangkan perangkat lunak, baik berbasis *web* maupun *mobile*.

##### a) *CREATE*

Fungsi *CRUD* yang pertama adalah *create*. Fungsi ini memungkinkanmu membuat *record* baru dalam *database*. Dalam aplikasi SQL, fungsi *create* sering disebut juga sebagai *insert*.

b) *READ*

Fungsi *read* hampir mirip dengan fungsi *search*. Fungsi ini memungkinkan kamu untuk mencari dan mengambil data tertentu dalam tabel dan membaca nilainya.

c) *UPDATE*

Untuk memodifikasi *record* yang telah tersimpan di *database*, fungsi *CRUD* yang bisa kamu gunakan adalah fungsi *update*.

d) *DELETE*

Ketika ada *record* atau data yang tidak lagi dibutuhkan dalam *database*, fungsi *CRUD* yang digunakan adalah fungsi *delete*. Fungsi ini dapat digunakan untuk menghapus data tersebut.

### 3.1.1 Projek pertama Tentang *CRUD*

Disini saya akan menjelaskan bagaimana projek saya yang tentang *crud* ,disini juga saya akan menampilkan gambar beserta keterangannya.

#### 1. Halaman Utama *CRUD*

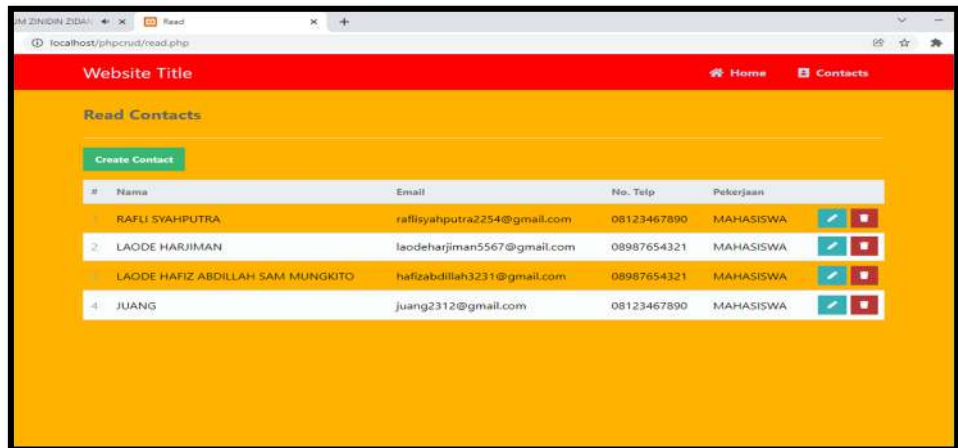


**Gambar 3.1** halaman utama *CRUD*

Keterangan:

Pada halaman utama dalam sebuah *CRUD* disana terdapat *Home*, dan *Contacts*.

## 2. Halaman *Contacts*



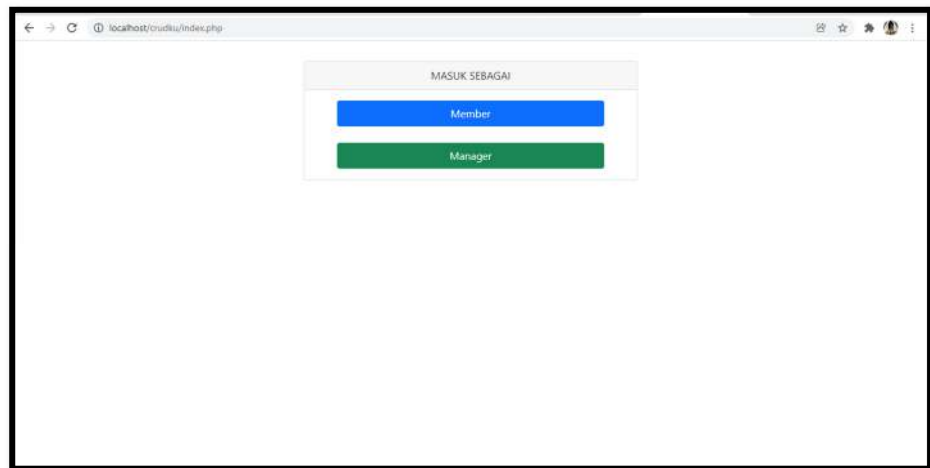
**Gambar 3.2** Halaman *Contacts*

Keterangan:

Pada Halaman *Contacts* ini terdapat sekumpulan data dan disana juga terdapat tombol tambah data (*create*) untuk menambah data, ubah (*update*) Untuk memodifikasi *record* yang telah tersimpan di *database* dan hapus (*delete*) untuk menghapus data yang ada.

### 3.1.2 Projek Yang Ke Dua

#### 1. Halaman Utama

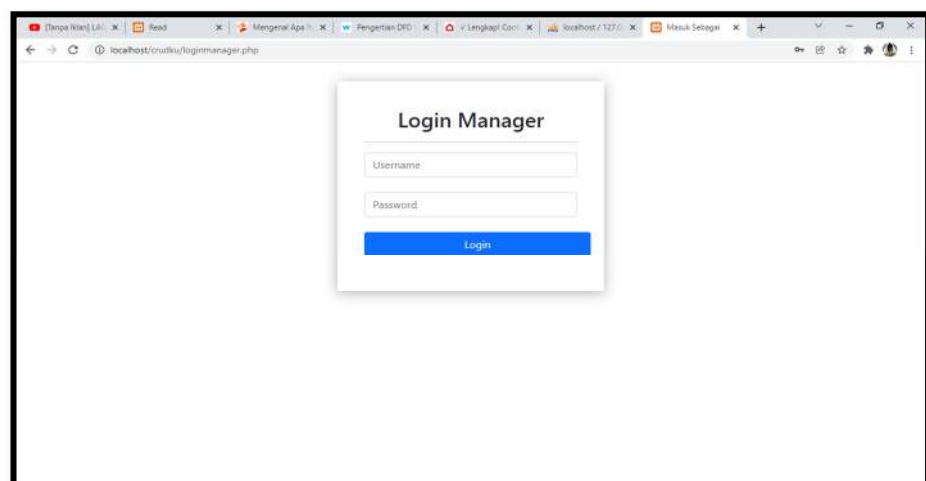


**Gambar 3.3** Halaman Utama

Keterangan:

Pada halaman Utama berupa halaman untuk masuk sebagai member atau manager

#### 2. Halaman Login

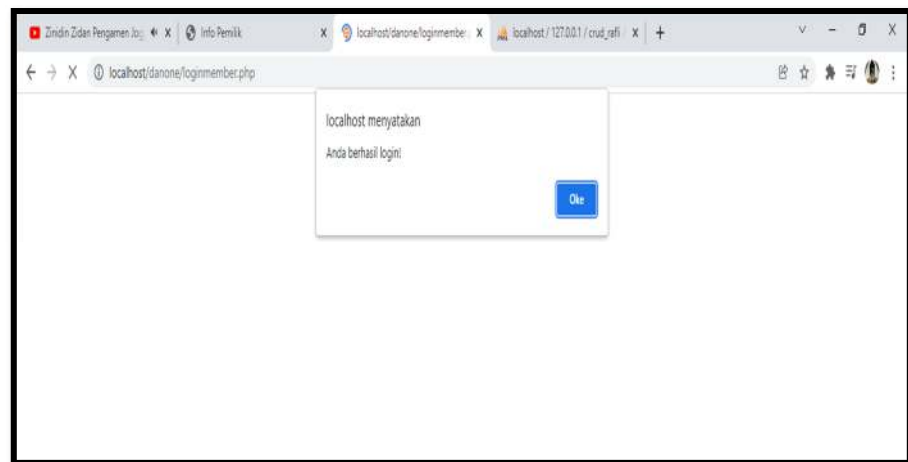


**Gambar 3.4** Halaman *login*.

Keterangan:

Pada Halaman ini Kita di arahkan untuk *login* pada manager tetapi sebelum itu terlebih dahulu di arahkan untuk mengisi *Username* sama *password*.

### 3. Halaman berhasil *login*

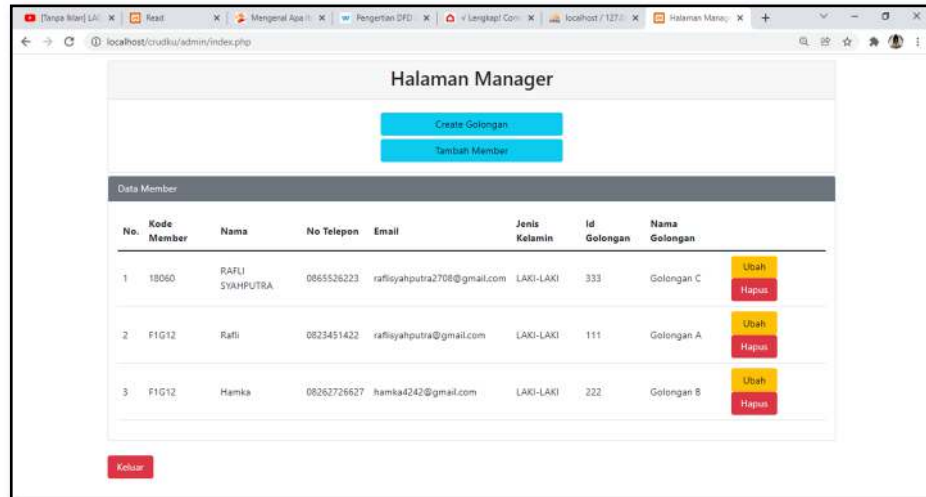


**Gambar 3.5** Halaman berhasil *login*

Keterangan:

Selanjutnya apabila kita sudah mengisi *username* dan *password* maka selanjutnya tekan *login* maka setelah itu akan muncul tulisan anda berhasil *login*.

#### 4. Halaman manager



**Gambar 3.6** Halaman manager

Keterangan:

Pada Halaman ini Akan menampilkan data member mulai dari kode member, nama, no telepon, email, jenis kelamin, id golongan dan nama golongan. dan terdapat tombol untuk hapus data, ubah data yang ada serta tombol keluar untuk *logout*.

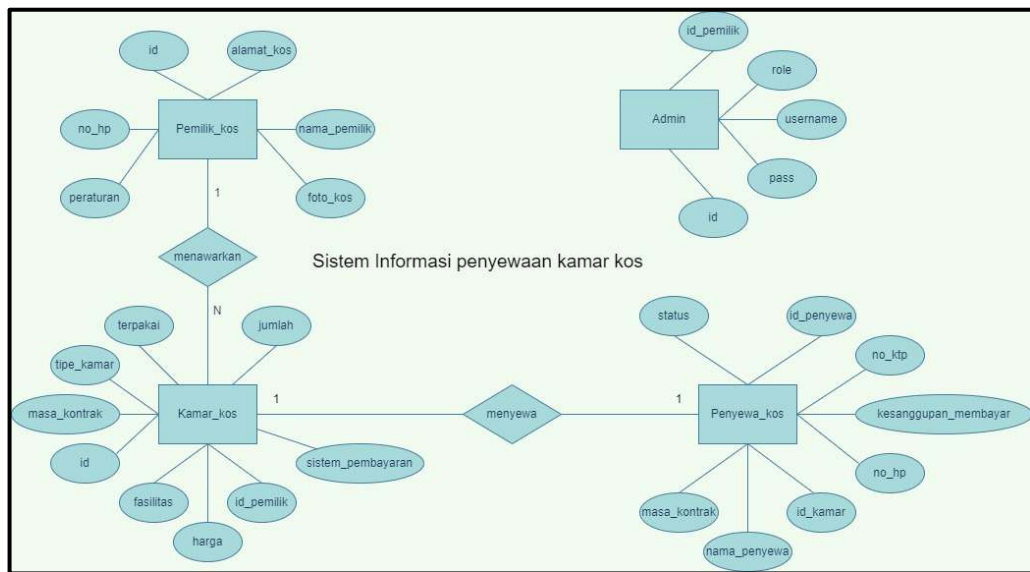
#### 4.1 Pertemuan empat

Pengolahan data pembayaran masih menggunakan sistem *konvensional* atau manual yaitu melakukan pencatatan kedalam buku catatan pembayaran. Dengan pengolahan sistem manual, kendala yang dihadapi adalah pengecekan data penyewa yang telah membayar maupun yang belum, pengecekan data kamar yang kosong, dan pencarian data penyewa. Dari permasalahan tersebut, maka dibangun Sistem Informasi berbasis *web* agar dapat digunakan untuk membantu pemilik kos mengolah berbagai administrasi dan keuangan kos. Sistem ini dapat membantu calon penyewa memonitoring kamar kos yang telah terisi, rusak, maupun yang belum terisi, dapat membantu penyewa kos dalam pembayaran kos, dan dapat membantu pemilik kos dalam membuat laporan keuangan.

##### 4.1.1 ERD

*Entity Relationship Diagram (ERD)* adalah menyediakan cara untuk mendeskripsikan perancangan basis data pada peringkat logika. *ERD* merupakan suatu model untuk menjelaskan hubungan antar data dalam basis data berdasarkan objek-objek dasar data yang mempunyai hubungan antar relasi. *ERD* untuk memodelkan struktur data dan hubungan antar data, untuk menggambarannya digunakan beberapa notasi dan simbol. *ERD* adalah suatu model jaringan yang menggunakan susunan data yang disimpan dalam *system* secara abstrak. *ERD* berbeda dengan *Data Flow Diagram (DFD)* yang merupakan suatu model jaringan fungsi yang akan dilaksanakan oleh *system*, sedangkan *ERD* merupakan model jaringan data yang menekankan pada struktur-struktur dan relationship data.





**Gambar 4.1** Contoh ERD

Keterangan :

Berdasarkan Gambar 4.1, dapat dilihat bahwa Pemilik kos memiliki hubungan *One To Many* dengan Kamar kos yang artinya satu pemilik kos dapat menawarkan banyak kamar kos, begitupun sebaliknya, banyak kamar kos dapat ditawarkan oleh satu pemilik kos. Dan pada Kamar kos memiliki hubungan *One To One* dengan Penyewa kos yang dimana dapat diartikan bahwa satu kamar kos dapat disewa oleh satu penyewa dan juga sebaliknya, satu penyewa dapat menyewa satu kamar kos.

#### 1. Fungsi ERD

Fungsi penggambaran *Entity Relationship Diagram (ERD)* yang ada saat ini yaitu sebagai berikut :

- Untuk memudahkan kita dalam menganalisis pada suatu basis data atau suatu system dengan cara yang cepat dan murah.

- b) Dapat dilakukan pengujian pada model yang telah dibuat dan dapat mengabaikan proses yang sudah dibuat hanya dengan menggambar *Entity Relationship Diagram (ERD)*.
- c) Menjelaskan hubungan-hubungan antar data-data dalam basis data berdasarkan objek –objek dasar data yang memiliki hubungan yang dihubungkan oleh suatu relasi.
- d) Untuk mendokumentasikan data-data yang ada dengan cara *mengidentifikasi* setiap *entitas* dari data-data dan hubungannya pada suatu *Entity Relationship Diagram (ERD)* itu sendiri.

#### 4.1.2 DFD

*Data Flow Diagram (DFD)* adalah suatu diagram yang menggunakan notasi-notasi untuk menggambarkan arus dari data *system*, yang penggunaannya sangat membantu untuk memahami *system* secara logika, tersruktur dan jelas.

*DFD* merupakan alat bantu dalam menggambarkan atau menjelaskan *system* yang sedang berjalan logis. Dalam sumber lain dikatakan bahwa *DFD* ini merupakan salahsatu alat pembuatan model yang sering digunakan, khususnya bila fungsi-fungsistem merupakan bagian yang lebih penting dan kompleks dari pada data yang dimanipulasi oleh *system*. Dengan kata lain, *DFD* adalah alat pembuatan model yang memberikan penekanan hanya pada fungsi *system*. *DFD* ini merupakan alat perancangan *system* yang berorientasi pada alur data dengan konsep dekomposisi dapat digunakan untuk penggambaran *system* maupun rancangan *system* yang mudah dikomunikasikan oleh *system*

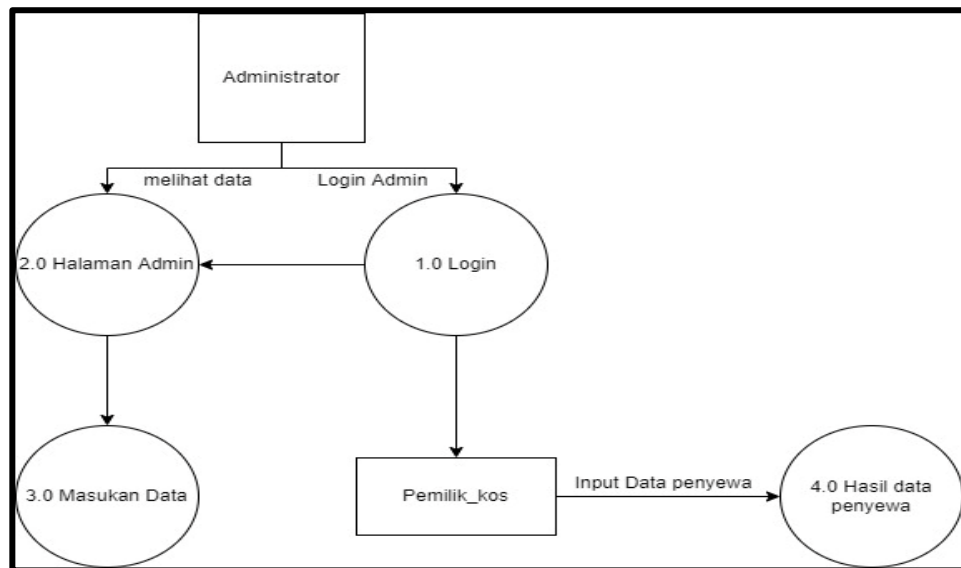
kepada pemakai maupun pembuat program. Suatu yang lazim bahwa ketika menggambarkan sebuah sistem kontekstual *data flow diagram* yang akan pertama kali muncul adalah interaksi antara *system* dan entitas luar. *DFD* didesain untuk menunjukkan sebuah *system* yang terbagi-bagi menjadi suatu bagian *sub-system* yang lebih kecil dan untuk menggarisbawahi arus data antara kedua hal yang tersebut diatas. Diagram ini lalu “dikembangkan” untuk melihat lebih rinci sehingga dapat terlihat model-model yang terdapat di dalamnya. Merupakan alat yang digunakan untuk menggambarkan suatu *system* yang telah ada atau *system* baru yang akan dikembangkan secara logika tanpa mempertimbangkan lingkungan fisik dimana data tersebut mengalir ataupun lingkungan fisik dimana data tersebut akan disimpan.

#### A. *DFD* LEVEL 0

Diagram level 0 atau bisa juga diagram konteks adalah level diagram paling rendah yang menggambarkan bagaimana *System* berinteraksi dengan *external entitas*. Pada diagram konteks akan diberikan nomor untuk setiap proses yang berjalan, umumnya mulai dari angka 0 untuk start awal.

Semua entitas yang ada pada diagram konteks termasuk juga aliran datanya akan langsung diarahkan kepada *System*. Pada diagram konteks ini juga tidak ada informasi tentang data yang tersimpan dan tampilan diagramnya tergolong sederhana.

### Contoh *DFD* level 0



**Gambar 4.2** Contoh *DFD* level 0

Keterangan :

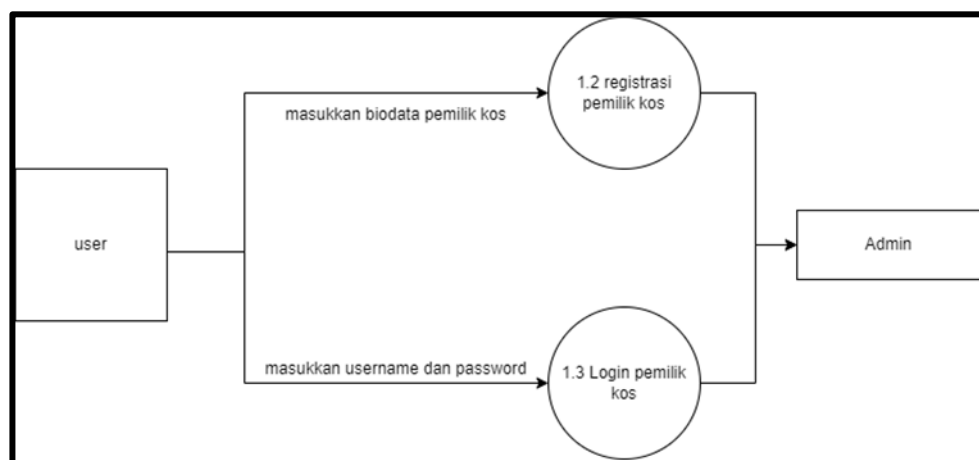
Pada gambar 4.2 di atas *System* menguraikan tahap-tahap dari sebuah kos, yaitu :

1. Tahapan atau proses melakukan akses untuk login sebagai admin atau pemilik kos
2. Tahapan atau proses mengakses ke halaman admin
3. Tahapan atau proses memasukkan data penyewa
4. Tahapan atau proses menginput hasil data penyewa

## B. DFD LEVEL 1

DFD level 1 adalah tahapan lebih lanjut tentang DFD level 0, dimana semua proses yang ada pada DFD level 0 akan dirinci dengan lengkap sehingga lebih lengkap dan detail. Proses-proses utama yang ada akan dipecah menjadi sub-proses.

Contoh DFD level 1



**Gambar 4.3** Contoh DFD level 1

Keterangan :

1. Pada tahap ini pemilik kos akan memasukan biodata diri seperti nama, alamat, no handphone, alamat email dan gambar kos
2. Admin akan menginputkan daftar kos seperti alamat kos, fasilitas kos, gambar kos dan lainnya dengan harga yang sudah ditentukan
3. Kemudian sistem akan menyimpan data dari pemilik kos untuk memudahkan penyewa saat melakukan penyewaan
4. Selanjutnya admin akan mengakses semua data tersebut ke sistem informasi

### C. Fungsi *Data Flow Diagram*

Secara *fundamental*, terdapat tiga fungsi dari pembuatan diagram alir data untuk kebutuhan *software development*. Berikut ini merupakan penjelasan dari masing – masing fungsi di bawah ini.

#### 1. Menyampaikan Rancangan Sistem

Dengan pembuatan DFD, maka proses penyampaian informasi menjadi lebih mudah dengan tampilan visual yang simple dan dapat dimengerti oleh tiap stakeholder. Dimana, data yang disajikan mampu menggambarkan alur data secara terstruktur dengan pendekatan yang lebih efisien.

#### 2. Menggambarkan Suatu Sistem

Fungsi yang kedua, DFD dapat membantu proses penggambaran sistem sebagai jaringan fungsional. Maksudnya adalah, di dalam jaringan terdapat berbagai komponen yang saling terhubung menggunakan alur data.

#### 3. Perancangan Model

Fungsi yang terakhir, diagram ini juga dapat membuat rancangan model baru dengan menekankan pada fungsi sistem tertentu. Hal tersebut dapat dimanfaatkan untuk melihat bagian yang lebih detail dari diagram alir data tersebut.

#### 4.1.3 *Interface*

Antarmuka (*Interface*) merupakan mekanisme komunikasi antara pengguna (*user*) dengan sistem. Antarmuka (*Interface*) dapat menerima informasi dari pengguna (*user*) dan memberikan informasi kepada pengguna

(*user*) untuk membantu mengarahkan alur penelusuran masalah sampai ditemukan suatu solusi *Interface*, berfungsi untuk menginput pengetahuan baru ke dalam basis pengetahuan sistem pakar (*ES*), menampilkan penjelasan sistem dan memberikan panduan pemakaian sistem secara menyeluruh / *step by step* sehingga pengguna mengerti apa yang akan dilakukan terhadap suatu sistem. Yang terpenting adalah kemudahan dalam memakai / menjalankan sistem, interaktif, komunikatif, sedangkan kesulitan dalam mengembangkan / membangun suatu program jangan terlalu diperlihatkan. *Interface* yang ada untuk berbagai sistem, dan menyediakan cara : *Input*, memungkinkan pengguna untuk memanipulasi sistem. *Output*, memungkinkan sistem untuk menunjukkan efek manipulasi pengguna.

#### A. Tujuan *Interface*

Tujuan sebuah *interface* adalah mengkomunikasikan fitur-fitur sistem yang tersedia agar user mengerti dan dapat menggunakan sistem tersebut. Dalam hal ini penggunaan bahasa amat efektif untuk membantu pengertian, karena bahasa merupakan alat tertua (barangkali kedua tertua setelah *gesture*) yang dipakai orang untuk berkomunikasi sehari-harinya. Praktis, semua pengguna komputer dan Internet (kecuali mungkin anak kecil yang memakai komputer untuk belajar membaca) dapat mengerti tulisan. Meski pada umumnya panduan *interface* menyarankan agar ikon tidak diberi tulisan supaya tetap mandiri dari bahasa, namun elemen interface lain seperti teks pada tombol, *caption window*, atau teks-teks singkat di sebelah kotak *input* dan tombol pilihan semua menggunakan bahasa. Tanpa bahasa pun kadang ikon

bisa tidak jelas maknanya, sebab tidak semua lambang ikon bisa bersifat *universal*. Meskipun penting, namun sayangnya kadang penggunaan bahasa, seperti pemilihan istilah, sering sekali dianggap kurang begitu penting. Terlebih dari itu dalam dunia desain situs *Web* yang serba grafis, bahasa sering menjadi sesuatu yang nomor dua ketimbang elemen-elemen *interface* lainnya.

*Interface* ada dua jenis, yaitu :

- 1) *Graphical Interface* : Menggunakan unsur-unsur multimedia (seperti gambar, suara, video) untuk berinteraksi dengan pengguna.
- 2) *Text-Based* : Menggunakan *syntax*/rumus yang sudah ditentukan untuk memberikan perintah.

## B. PERBANDINGAN *INTERFACE*

Ada 5 tipe utama interaksi untuk *interaction*:

- 1) *Direct manipulation* – pengoperasian secara langsung : interaksi langsung dengan objek pada layar. Misalnya *delete file* dengan memasukkannya ke *trash*. Contoh: Video games. Kelebihan : Waktu pembelajaran sangat singkat, *feedback* langsung diberikan pada tiap aksi sehingga kesalahan terdeteksi dan diperbaiki dengan cepat. Kekurangan : *Interface* tipe ini rumit dan memerlukan banyak fasilitas pada sistem komputer, cocok untuk penggambaran secara visual untuk satu operasi atau objek.
- 2) *Menu selection* – pilihan berbentuk menu : Memilih perintah dari daftar yang disediakan. Misalnya saat *click* kanan dan memilih aksi yang dikehendaki. Kelebihan : tidak perlu ingat nama perintah.



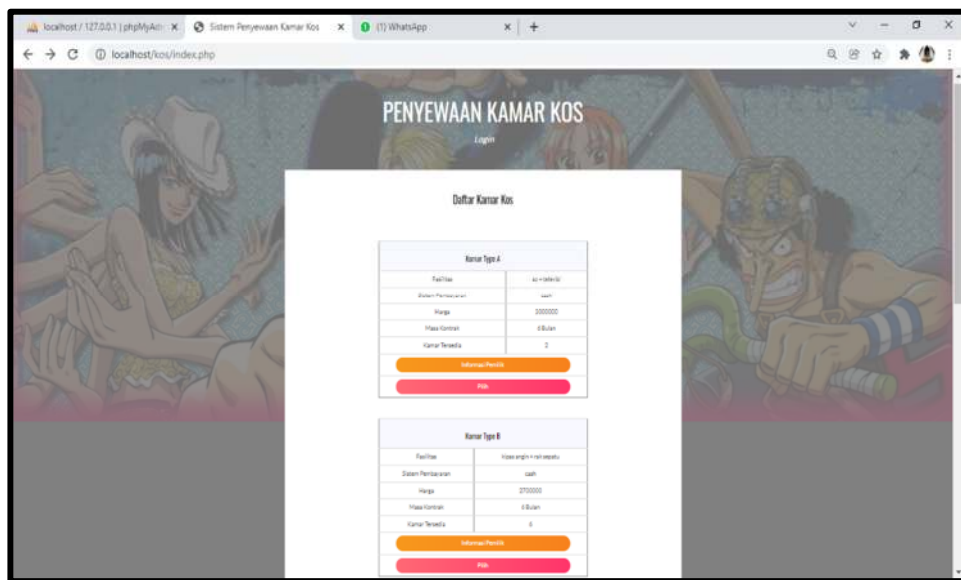
Pengetikan minimal. Kesalahan rendah. Kekurangan : Tidak ada logika *AND* atau *OR*. Perlu ada struktur menu jika banyak pilihan. Menu dianggap lambat oleh *expert* dibanding *command language*.

- 3) *Form fill-in* – pengisian form : Mengisi area-area pada form. Contoh : *Stock control*. Kelebihan : Masukan data yang sederhana. Mudah dipelajari Kekurangan : Memerlukan banyak tempat di layar. Harus menyesuaikan dengan *form manual* dan kebiasaan.
- 4) *Command language* – perintah tertulis : Menuliskan perintah yang sudah ditentukan pada program. Contoh: *operating system*. Kelebihan : Perintah diketikan langsung pada *system*. Misal *UNIX, DOS command*. Bisa diterapkan pada terminal yang murah. Kombinasi perintah bisa dilakukan. Misal *copy file* dan *rename* nama *file*. Kekurangan : Perintah harus dipelajari dan diingat cara penggunaannya, tidak cocok untuk biasa. Kesalahan pakai perintah sering terjadi. Perlu ada sistem pemulihan kesalahan. Kemampuan mengetik perlu.
- 5) *Natural language* – perintah dengan bahasa alami : Menggunakan bahasa alami untuk mendapatkan hasil. Contoh: *search engine* di Internet. Kelebihan: Perintah dalam bentuk bahasa alami, dengan kosa kata yang terbatas (singkat), misalnya kata kunci yang kita tentukan untuk dicari oleh *search engine*. Ada kebebasan menggunakan kata-kata. Kekurangan: Tidak semua sistem cocok gunakan ini. Jika digunakan maka akan memerlukan banyak pengetikan.

#### 4.1.4 Projek akhir Penyewaan Kos

Disini saya akan menjelaskan bagaimana proyek saya yang berjudul Penyewaan kamar kos, disini juga saya akan menampilkan gambar beserta keterangan dan *ERD*nya.

##### 1. Halaman Utama

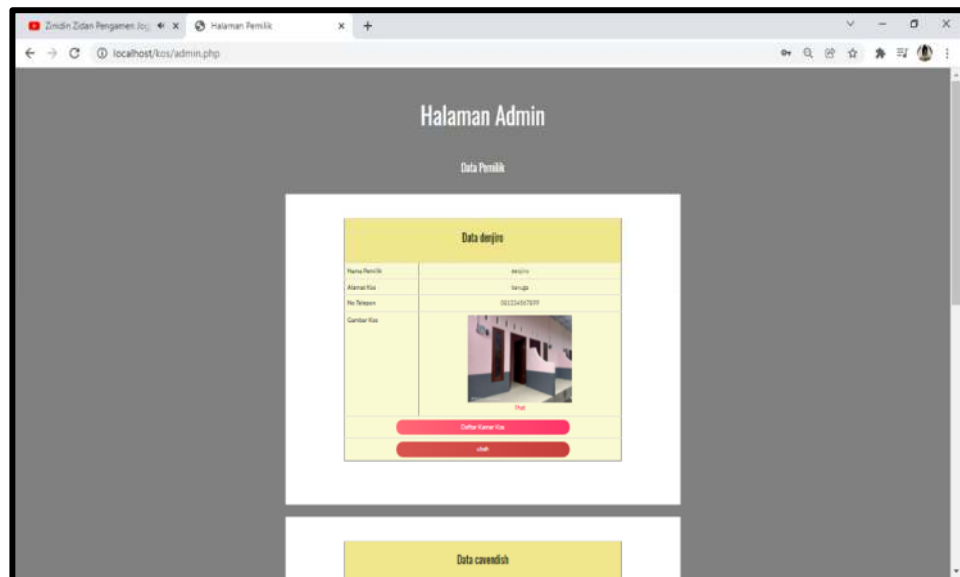


**Gambar 4.4** Halaman utama kos

Keterangan :

Pada halaman utama dalam sebuah *web* kos disana terdapat *Login*, daftar kamar kos, *type* kamar, informasi pemilik, dan pilih. Disana juga di jelaskan kamar yang tersedia, selain itu disana juga di perlihatkan fasilitas kamar apabila ada yang berminat untuk menempati kos tersebut sehingga mempermudah penyewa mengenali kos tersebut maka tinggal lihat fasilitas yang telah tettera pada daftar kamar kos tersebut.

## 2. Halaman Admin

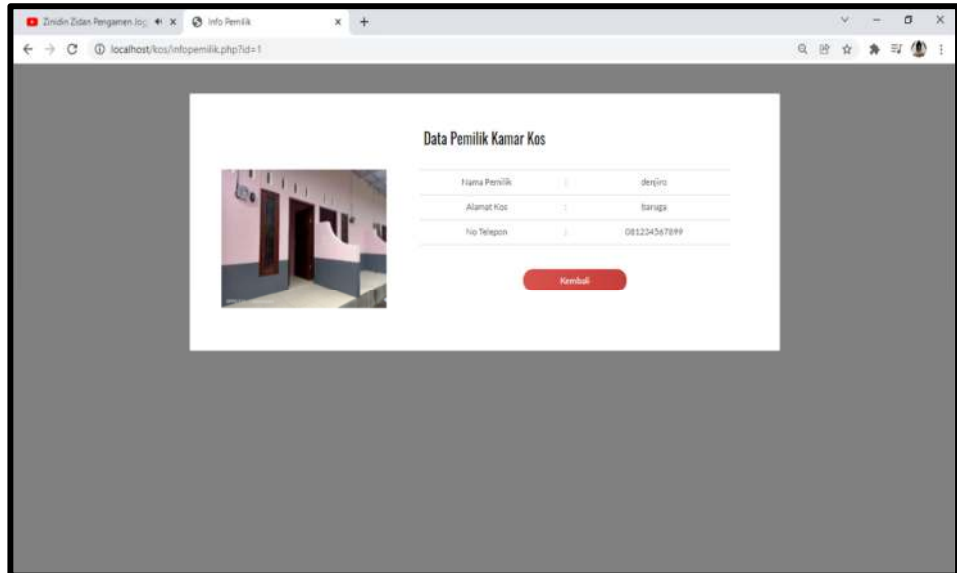


**Gambar 4.5** Halaman admin

Keterangan :

Pada halaman Admin ini hamper sama dengan halaman utama kos sebelumnya, disini terdapat nama, nomor telepon, gambar kos, dan alamat kos tersebut. Disini kita bisa menambah data pada sebuah kos dengan cara mengisi form, setelah semua di isi maka klik simpan data dan data yang di isi tadi akan muncul dengan sendirinya dalam daftar kamar kos dan juga pada halaman ini kita bisa mengubah data pemilik kos serta bisa menghapus data pemilik kos.

### 3. Halaman informasi pemilik



**Gambar 4.6** Halaman informasi pemilik

Keterangan :

Pada halaman informasi pemilik diatas, dapat di jelaskan apabila kita ingin menghubungi atau mengetahui alamat kos maka kita langsung saja mengunjungi kolom informasi pemilik, disini akan diperlihatkan nama pemilik, alamat kos, dan no telepon pemilik beserta gambar kos.

## DAFTAR PUSTAKA

<https://smkn1panjalu.sch.id/pengertian-pemrograman-berorientasi-objek-pbo/>

<https://waesalqorny.blogspot.com/2014/11/definisi-class-object-method-atribut.html>

<https://www.niagahoster.co.id/blog/laravel-adalah/>

<https://wikishare27.wordpress.com/pengertian-class-method-constructor-modifier-object-pada-java/>

<https://www.duniailkom.com/tutorial-belajar-oop-php-pengertian-class-object-property-dan-method/>

<https://www.petanikode.com/java-oop-constructor/>