

LAPORAN LENGKAP
PRAKTIKUM PEMROGRAMAN BERORIENTASI OBJEK



OLEH :

NAMA : WA ODE MURNIWATI
NIM : F1G120054
KELAS : GENAP

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 LENGKAP



OLEH :

NAMA : WA ODE MURNIWATI

NIM : F1G120054

Laporan praktikum Pemrograman Berorientasi Objek (*PBO*) ini disusun Untuk memenuhi tugas akhir menyelesaikan kegiatan praktikum pemrograman berorientasi objek (*PBO*), dan di susun sebagai salah satu syarat lulus mata kuliah pemrograman berorientasi objek (*PBO*).

Kendari, ~~7~~Desember 2021

Menyetujui,

AsistenPraktikum

17-12-2021

WAHID SAFRI JAYANTO
F1G117059

Praktikan


WA ODE MURNIWATI
F1G120054

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 karenaitu, kami mengharap kritik dan saran dari pengguna laporan ini. Sekian terimakasih, *wabillahitaufikwalhidayah, WassalamuAlaikum Warahumatullahi Wabarakatu.*

Kendari, Desember 2021

Penulis

DAFTAR ISI

HALAMAN COVER.....	i
HALAMAN PENGESAHAN.....	ii
KATA PENGANTAR.....	iii
DAFTAR ISI.....	iv
DAFTAR TABEL.....	vi
DAFTAR GAMBAR	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>Resursife Function</i>	16
3.1 Pertemuan Ke Tiga.....	18
3.1.1 <i>ERD</i>	18
3.1.2 <i>DFD</i>	19
3.1.3 <i>Interface</i>	22
3.1.4 <i>Project Crud</i>	23
4.1 Pertemuan Ke Empat.....	27

4.1.1 <i>ERD</i>	27
4.1.2 <i>DFD</i>	28
4.1.3 <i>Interface</i>	32
DAFTAR PUSTAKA	35

DAFTAR TABEL

Tabel 1.1 Tabel alat dan Bahan.....	1
--	---

DAFTAR GAMBAR

Gambar 3.1 Tampilan Awal <i>login</i>	23
Gambar 3.2 Tampilan awal ketika <i>login</i> pada member.....	24
Gambar 3.3 Tampilan berhasil <i>login</i> pada member.....	24
Gambar 3.4 Tampilan halaman data member.....	25
Gambar 3.5 Tampilan ubah data/tambah data.....	26
Gambar 4.1 <i>ERD</i>	27
Gambar 4.2 <i>DFD Level 0</i>	29
Gambar 4.3 <i>DFD level 1</i>	30
Gambar 4.4 Halaman Utama.....	31
Gambar 4.5 Halaman <i>Login</i>	31
Gambar 4.6 Halaman Berhasil <i>login</i>	32
Gambar 4.7 Halaman Pemilik kos.....	33
Gambar 4.8 Halaman Kamar kos.....	34

1.1 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>sublime</i> .
<i>Sublime</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* atau disingkat *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 (*behaviour*) 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. . (Douglas, 1992).

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. (Douglas, 1992)

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. . (Douglas, 1992).

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 (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. 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. (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 :

- a) Nama Constructor sama dengan nama Class.
- b) Tidak ada return type yang diberikan kedalam Constructor Signature.
- c) 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_duniailkom*, 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 *variabel* 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 lainya. 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* 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* (Nurul Huda, 2020).

Cara Pengunannya:

```
<?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 Destructur*:

```
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;
    }
}

class DemoConstructor{
    public static void main(String[] args) {
        User petani = new User("petanikode", "kopi");

        System.out.println("Username: " +
petani.username);

        System.out.println("Password: " +
petani.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 kurang 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 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

3.1.1 ERD

Entity Relationship Model (ERM) merupakan konseptual *representasi* data dan secara abstrak. *Entity Relationship* sendiri adalah salah satu metode pemodelan basis data yang digunakan untuk menghasilkan skema *konstekstual* untuk jenis atau model data semantik sistem.

1. Fungsi ERD

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

- a) Untuk memudahkan kita dalam menganalisis pada suatu basis data atau suatu sistem 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.

3.1.2 DFD (*Data Flow Diagram*)

Data Flow Diagram adalah jenis diagram diagram yang menunjukkan pergerakan informasi dari satu tempat ke tempat lain sebagai bagian dari *prosesor* tertentu pada umumnya. Dalam kasus lain - *DFD* dapat menunjukkan bagaimana berbagai departemen dalam organisasi bekerja sama - itu membuat semuanya menjadi jelas dan *koheren*.

1. Simbol-Simbol *DFD*

a) Proses (*Process*)

Suatu proses adalah kegiatan atau fungsi bisnis di mana manipulasi dan transformasi data terjadi. Suatu proses dapat didekomposisi ke tingkat rincian yang lebih halus, untuk mewakili bagaimana data sedang diproses dalam proses.

b) Penyimpanan Data (*Data Store*)

Penyimpanan data merupakan penyimpanan data persisten yang diperlukan dan / atau diproduksi oleh proses. Berikut adalah beberapa contoh penyimpanan data: formulir keanggotaan, tabel *database*, dll.

c) Entitas Eksternal (*External Entity*)

Entitas eksternal dapat mewakili manusia, sistem atau subsistem. Di sinilah data tertentu berasal atau pergi ke. Ini adalah *eksternal* dari sistem yang kita pelajari, dalam hal proses

bisnis. Untuk alasan ini, orang biasa menggambar *entitas eksternal* di tepi diagram.

d) Aliran data (*Data Flow*)

Aliran data mewakili aliran informasi, dengan arahnya diwakili oleh panah yang menunjukkan di ujung konektor aliran.

2. Jenis-Jenis *DFD* (*Data Flow Diagram*)

a. *Diagram Level 0* (*Diagram Konteks*)

Diagram *level 0* atau bisa juga diagram konteks adalah level diagram paling rendah yang menggambarkan bagaimana sistem 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 sistem. Pada diagram konteks ini juga tidak ada informasi tentang data yang tersimpan dan tampilan diagramnya tergolong sederhana.

b. *Data Flow Diagram 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.

c. Perbedaan *DFD Level 0* dan *DFD Level 1*

Ada perbedaan antara 2 *level DFD* tersebut yang perlu Anda ketahui, berikut ini perbedaannya.

- 1) *DFD level 0* hanya menggambarkan sistem secara basic saja.
- 2) *DFD level 0* hanya menjelaskan aliran data dari input sampai *output*.
- 3) *DFD level 1* menggambarkan aliran data yang lebih *kompleks* pada setiap prosesnya yang kemudian terbentuklah data *store* dan aliran data.
- 4) *DFD level 1* menggambarkan sistem secara sebagian atau seluruhnya secara mendetail.

3. Fungsi *DFD*

- a) *Data Flow Diagram* adalah alat yang sangat berguna untuk komunikasi. Ini membantu untuk memberikan wawasan yang dapat diakses untuk yang belum tahu.
- b) Komponen *visual* sangat penting. Perampingan dan transformasi ke dalam diagram memberikan pemahaman yang jelas tentang apa yang terjadi dengan sistem.
- c) Karena sistem notasi yang mudah diikuti, memungkinkan dicerna bahkan proses yang paling rumit dan memecahnya ke dalam bagan yang dapat dipahami.

3.1.3 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 *milenial* 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 parameter *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* .

3.1.3 Project Crud

1. Tampilan awal *login*

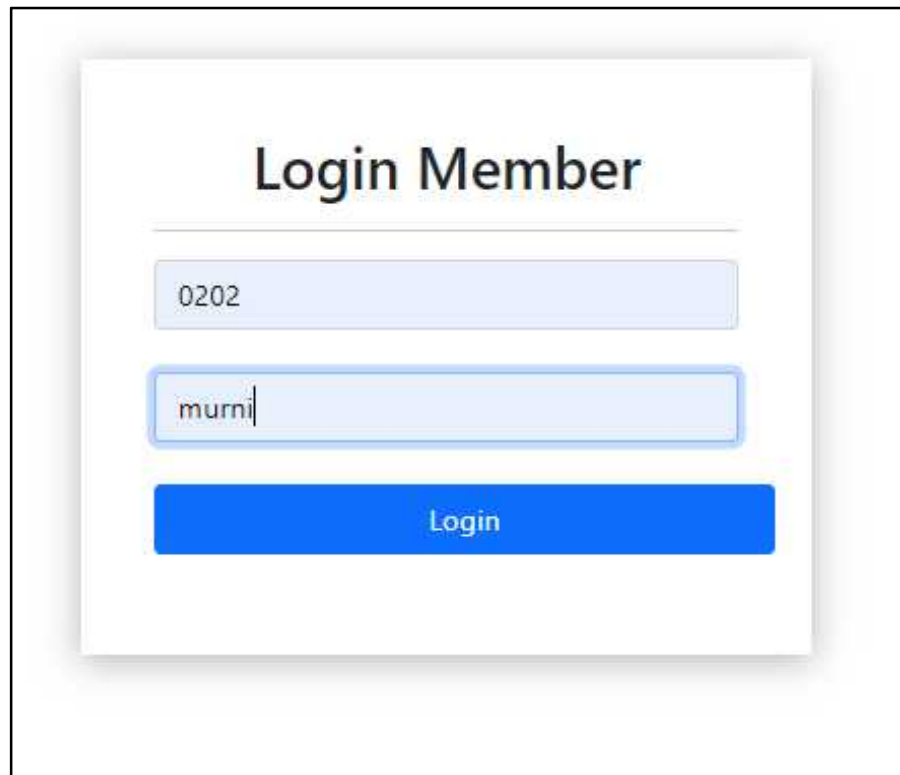


Gambar 3.1 Tampilan awal *login*

Keterangan:

Untuk halaman awal *login* pertama kita perlu mengaktifkan *apache* pada *xampp* kemudian membuka tab baru pada *web browser* lalu mengetikkan `localhost/crud_murni/` Kemudian akan tampil halaman seperti gambar di atas dimana terdapat dua pilihan *login* yaitu *login* sebagai member dan sebagai manager kemudian jika mengklik *member* maka akan muncul halaman seperti diatas.

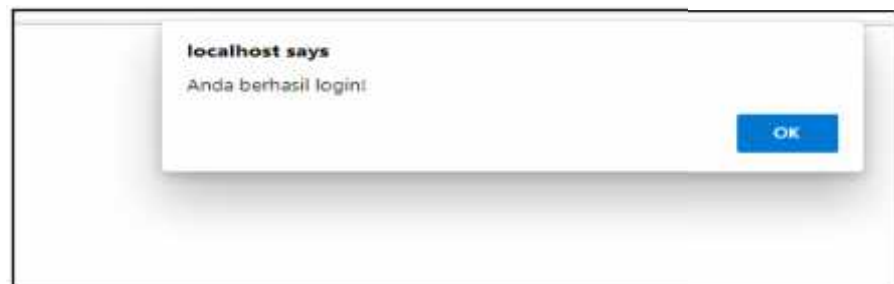
2. Halaman Tampilan awal ketika *login* pada *member*

A screenshot of a web application's login page for members. The page has a white background with a light gray border. At the top, the title "Login Member" is displayed in a large, bold, black font. Below the title, there are two input fields. The first input field contains the text "0202". The second input field contains the text "murni". Below these input fields is a blue button with the text "Login" in white. The entire form is centered on the page.

Gambar 3.2 Tampilan awal *login* pada *member*

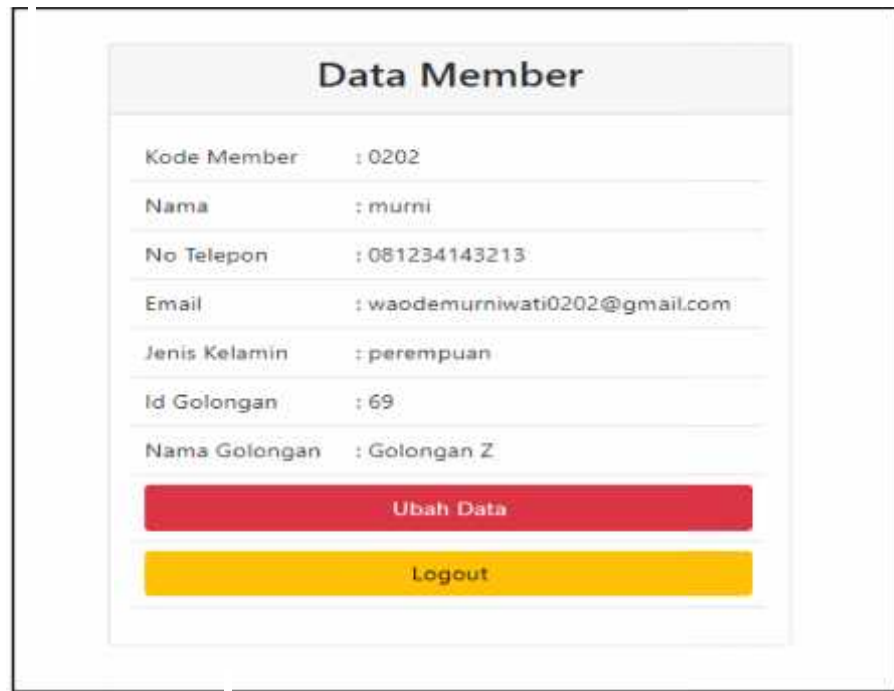
Keterangan:

Kemudian masukkan *kode member* dan nama untuk *login* sesuai dengan yang ada pada *record* database. Jika kode member dan nama yang di masukkan benar, maka akan muncul tulisan anda berhasil *login* seperti gambar berikut.



Gambar 3.3 Tampilan ketika berhasil *login* pada *member*

3. Tampilan halaman data *member*



The screenshot displays a web interface titled "Data Member". It contains a table with the following data:

Data Member	
Kode Member	: 0202
Nama	: murni
No Telepon	: 081234143213
Email	: waodemurniwati0202@gmail.com
Jenis Kelamin	: perempuan
Id Golongan	: 69
Nama Golongan	: Golongan Z

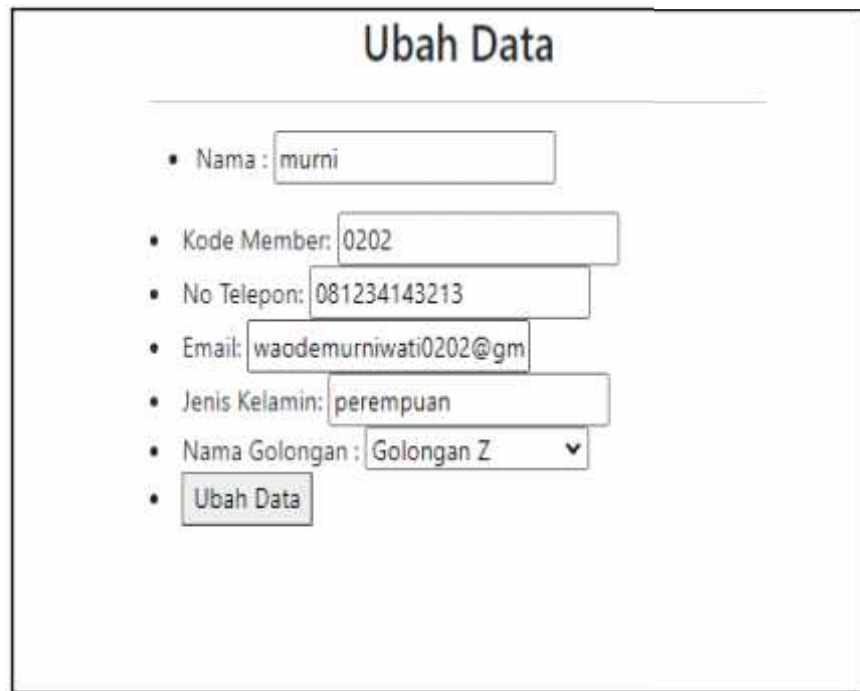
Below the table, there are two buttons: a red button labeled "Ubah Data" and a yellow button labeled "Logout".

Gambar 3.4 Halaman data *member*

Keterangan:

Setelah mengklik ok pada halaman anda berhasil *login* maka data member akan muncul data *member* yang kemudian bisa diubah ataupun dihapus.

4. Halaman Tambah data/Tambah data



The screenshot shows a web form titled "Ubah Data" (Change Data). It contains several input fields for updating member information:

- Nama : murni
- Kode Member: 0202
- No Telepon: 081234143213
- Email: waodemurniwati0202@gm
- Jenis Kelamin: perempuan
- Nama Golongan : Golongan Z (dropdown menu)
- Ubah Data (button)

Gambar 3.5 Halaman ubah data/tambah data

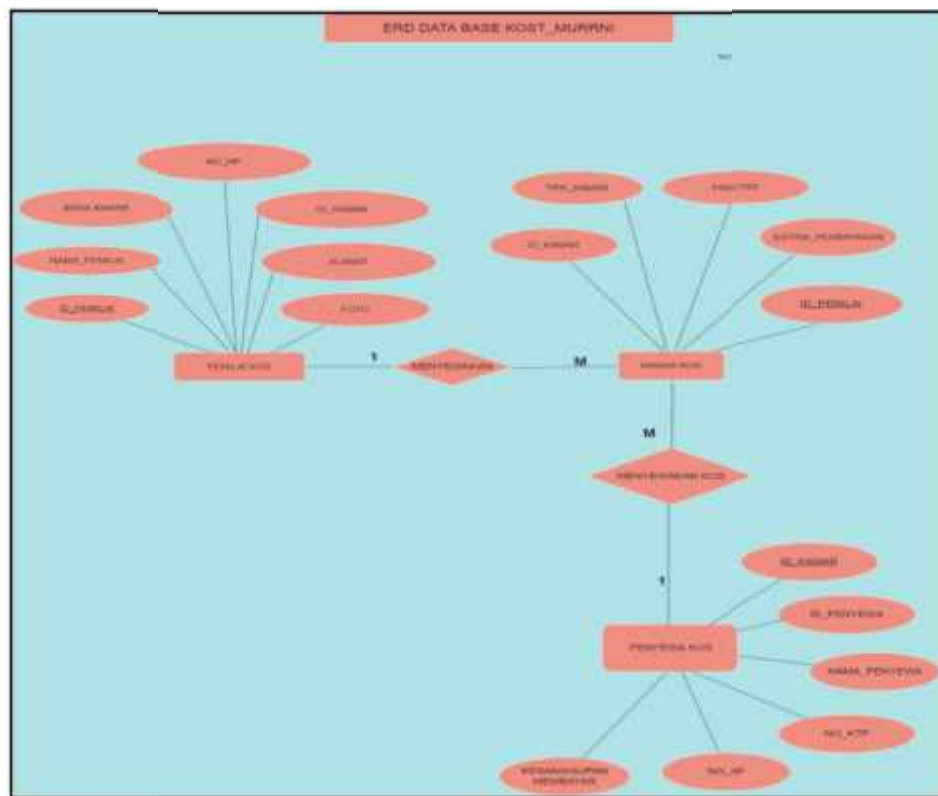
Keterangan:

Setelah masuk pada halaman member apabila kita ingin mengubah data maka kita bisa langsung mengklik ubah data yang berada d bagian bawa pada data *member*, maka setelah itu akan muncul kolom-kolom untuk mengubah data pada *member* tersebut.

4.1 PERTEMUAN KE EMPAT

4.1.2 ERD

Berikut merupakan ERD pada proyek *kost-murni*



Gambar 4.1 Erd kost_Murnni

Keterangan:

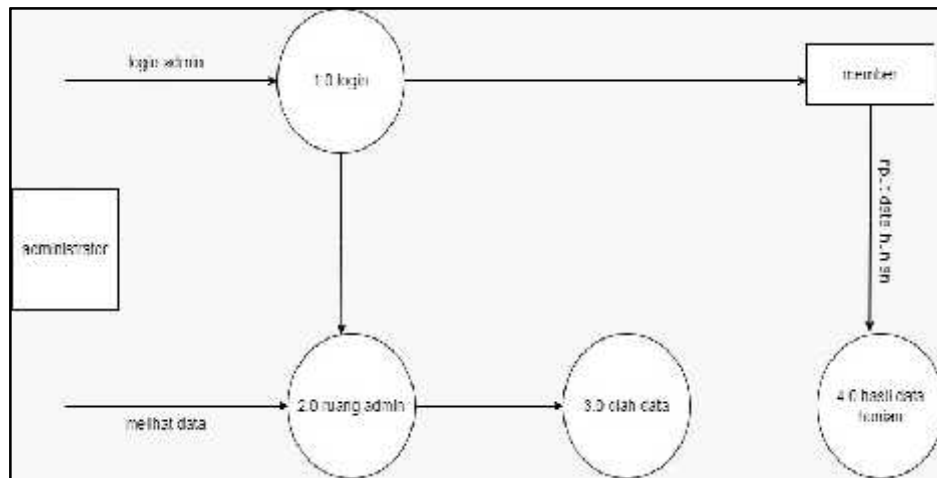
Pada gambar 4.6 terdapat pemilik kos, kamar kos, Penyewa kos yang mana tiap *entitas* terdapat atribut-atribut. Pada *entitas* pemilik terdapat atribut *id-pemilik int(15)*, nama pemilik *varchar (50)*, no-hp *varchar (25)*, *id-kamar int (11)*, alamat *varchar (50)*. Pada *entitas* kamar kos terdapat atribut *id-kamar int (11)*, tipe kamar *varchar (50)*, *id-pemilik int (50)*, sistem pembayaran *varchar (50)*, fasilitas *varchar (50)*, *id-pemilik int (11)*. Relasi pertama yaitu antara pemilik kos berelasi dengan kamar kos

dan kamar kos saling berelasi dengan penyewa kos , Relasi Pemilik kos dengan kamar kos yaitu *many to one* yang artinya dimana satu pemilik kos bisa menyewakan banyak kamar kos. Selanjutnya relasi antara kamar kos dengan penyewa kos memiliki relasi *many to one* yang artinya satu penyewa kamar kos bisa menyewa lebih dari satu kamar kos. Cara merelasikannya yaitu dengan cara menjadikan *entitas primary key* dan salah satunya *foreign key*.

4.1.4 DFD Kost-Murni

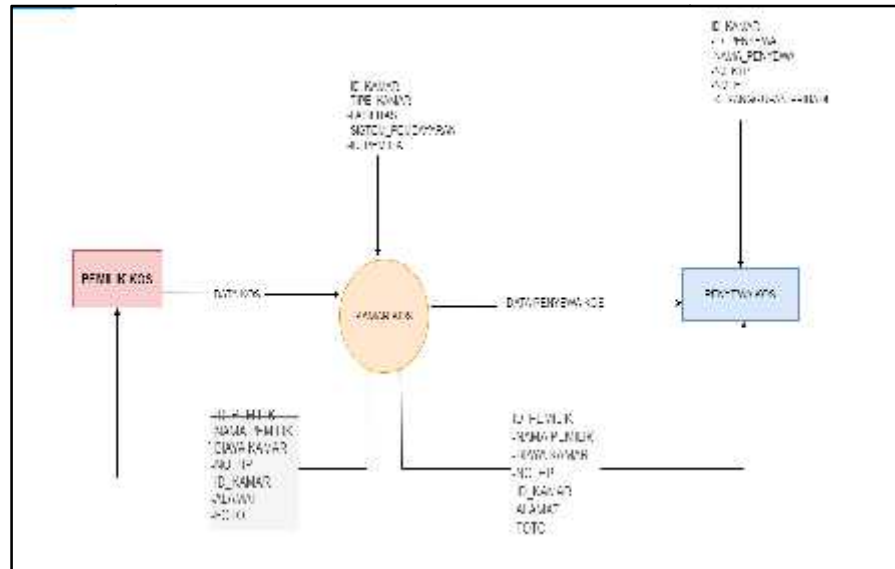
Menurut Saputra (2013:118) *Data flow diagram (DFD)* merupakan salah satu *diagram* yang menggambarkan alir data dalam suatu entitas ke sistem atau ke *entitas*.

DFD adalah suatu diagram yang menggambarkan aliran data dari sebuah proses yang sering disebut dengan sistem informasi. Di dalam data *flow diagram* juga menyediakan informasi mengenai *input* dan *output* dari tiap *entitas* dan proses itu sendiri. Dalam diagram alir data juga tidak mempunyai kontrol terhadap *flow* -nya, sehingga tidak adanya aturan terkait keputusan atau pengulangan. Bentuk penggambaran berupa data *flowchart* dengan skema yang lebih spesifik. Data *flow diagram* berbeda dengan *UML (Unified Modelling Language)*, dimana hal mendasar yang menjadi pembeda antara kedua skema tersebut terletak pada *flow* dan *objective* penyampaian informasi di dalamnya.



Gambar 4.2 DFD level 0

- a. *Diagram Level 0 (Diagram Konteks)* Diagram *level 0* atau bisa juga diagram konteks adalah *level diagram* paling rendah yang menggambarkan bagaimana sistem 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 sistem. Pada diagram konteks ini juga tidak ada informasi tentang data yang tersimpan dan tampilan diagramnya tergolong sederhana.
- b. *Data Flow Diagram 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 dipech menjadi *sub-proses*.



Gambar 4.3 DFD Level 1

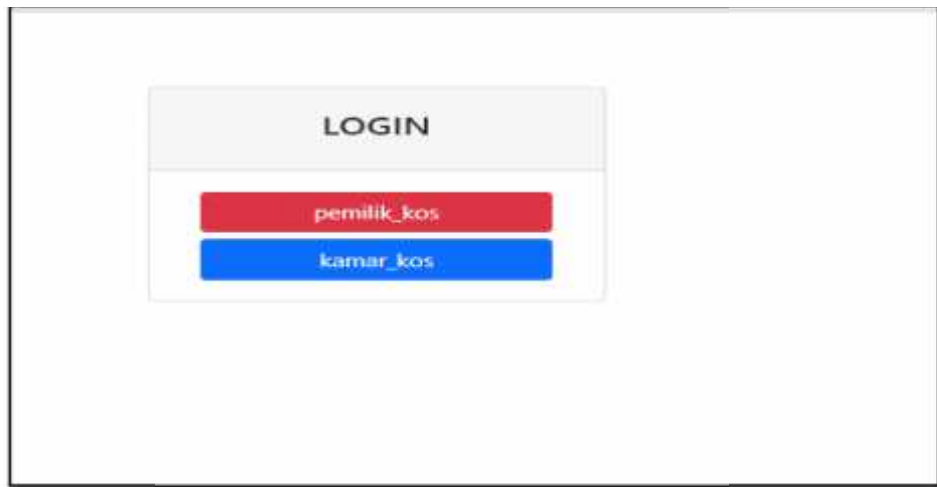
Keterangan:

Pada gambar diatas merupakan bentuk gambaran *DFD* Koss-murni yang menjelaskan terkait sistem aplikasi data kos dengan tiga data store yang dibuat, berdasarkan *Dfd level 0* pemilik kos saling berhubungan dengan kamar kos selanjutnya kamar kos juga berhubungan dengan kamar kos. Aktivitas untuk mengolah *input* menjadi *output* aliran data pada sistem antara proses antara data kos dan data penerima kos, serta menyimpan data pada tabel yang serupa.

4.1.5 Interface

Dalam merancang *interface*, dibagi kedalam beberapa tampilan sebagai berikut:

1. Halaman Utama

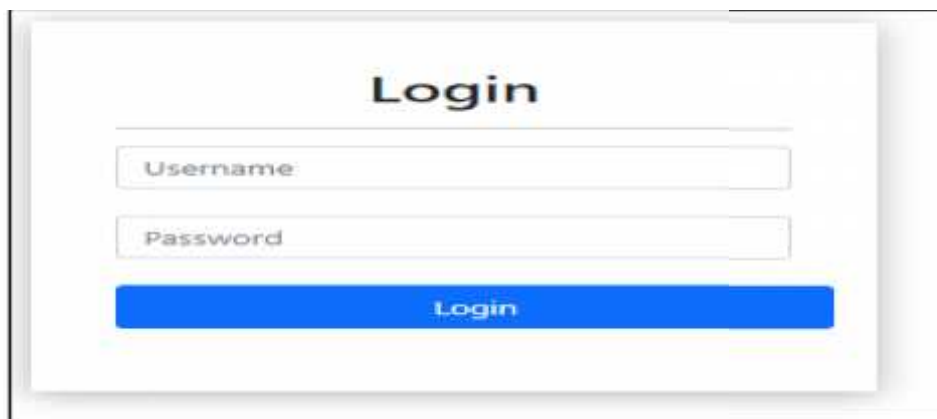


Gambar 4.4 Halaman Utama

Keterangan:

Pada gambar 4.8 halaman Utama berupa halaman untuk melihat pemilik kos atau kamar kos.

2. Halaman Login

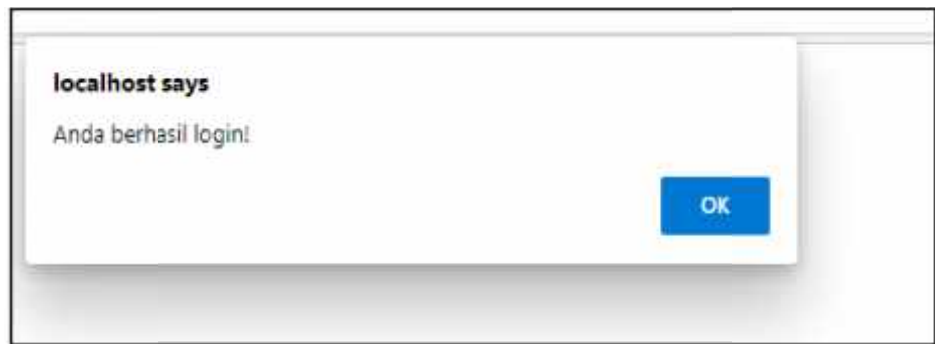


Gambar 4.5 Halaman *login*

Keterangan:

Pada gambar 4.9 diarahkan untuk *login* pada pemilik kos tetapi sebelum itu terlebih dahulu diarahkan untuk mengisi *Username* sama *password*.

3. Halaman berhasil *login*

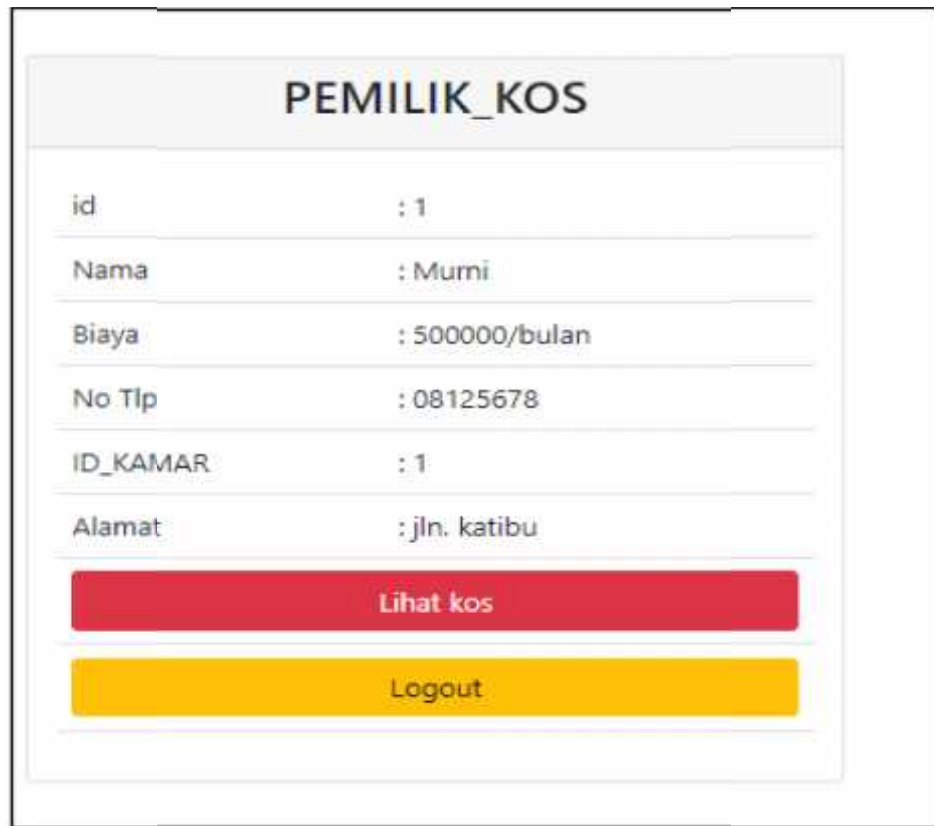


Gambar 4.6 Halaman berhasil *login*

Keterangan:

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

4. Halaman Pemilik Kos



The screenshot displays a web interface for a user named Murni. The page title is 'PEMILIK_KOS'. Below the title, there is a list of user details: id (1), Nama (Murni), Biaya (500000/bulan), No Tlp (08125678), ID_KAMAR (1), and Alamat (jln. katibu). At the bottom of the details section, there are two buttons: a red 'Lihat kos' button and a yellow 'Logout' button.

PEMILIK_KOS	
id	: 1
Nama	: Murni
Biaya	: 500000/bulan
No Tlp	: 08125678
ID_KAMAR	: 1
Alamat	: jln. katibu
Lihat kos	
Logout	

Gambar 4.7 Halaman Pemilik Kos



Keterangan:

Pada gambar 4.11 Halaman ini Akan menampilkan pemilik kos mulai dari *id*, nama, biaya, no tlpn.*id*-kamar dan alamat dan di bawahnya ada tulisan lihat kos dan *logout*.

5. Halaman Kamar Kos

WA ODE MURNIWATI_FIG120054_ILMU KOMPUTER

KAMAR_KOS

ID	NAMA PEMILIK	BIAYA KAMAR	GAMBAR	NO HP	ALAMAT	TIPE KAMAR	FASILITAS	SISTEM PEMBAYARAN	
1	Murni	50000/bulan		081129678	Jln. Kedu	4 x 7	air, listrik, wifi	bulanan	<button>Pesan</button>
2	Ryan	50000/bulan		081129678	Jln. Kedu	5 x 8	air, listrik, wifi	bulanan	<button>Pesan</button>

Keluar

Gambar 4.8 Halaman kamar kos

Keterangan:

Pada gambar 4.12 Halaman ini yang muncul adalah id, nama pemilik, gambar, no-hp, alamat, *tipe* kamar, *fasilitas* dan sistem pembayaran. Di sana di munculkan semua data kamar kos apabila ingin memesan maka tinggal menekan kata pesan yang tulisan warna kuning, sebelum anda memesan bisa melihat gambar kamar kos dengan cara menekan dibagian gambar dibawah ada tulisan lihat gambar maka secara otomatis setelah menekan tulisan itu maka akan muncul gambar kamar kos tersebut. Setelah melihat gambar kamar kost tersebut selanjutnya bisa keluar dari halaman tersebut dengan cara menekan kata keluar dibagian bawah pojok kiri yang tulisan berwarna merah.

DAFTAR PUSTAKA

- T suryana 2021, Materi tentang pengenalan php dan variabel. Jurnal ilmiah dan aplikasi IT materi pertemuan 10. 13(1):22-30
- Douglas, 1992. Pengertian pemrograman berorientasi objek.1992-2004. Jurnal ilmiah aplikasi IT pertemuan 10. 13(1):22-30
- Andre, 2015 Definisi php, object, atribut pada PHP. 2015 books.google.com 1(2):15-19.
- Gunadarma, 2013 Pengertian class, Method, Constructor, Modifier, Object pada java. Jurnal digilid.unical.ac.id 2(14):10_34
- Diki Alfarabi Hadi, 2015. “PHP OOP Part 2 : Pengertian Class, Object, Property dan Method”. Jurnal pilar nusa mandiri.ac.id 1(2):29-45.

