BAB 2
TINJAUAN PUSTAKA

2.1 Tinjauan Tempat Penelitian

2.1.1 Sejarah Perkembangan SMK Bina Wirausaha Talaga

SMK Bina Wirausaha berdiri pada tahun 1 Juli 2010 di bawah naungan Yayasan Pembina Pendidikan Profesional Indonesia (YP3I). Pada awal berdirinya sekolah SMK Bina Wirausaha Talaga belum memiliki gedung sekolah sendiri sehingga menyewa bangunan TPA untuk melangsungkan pembelajaran, pada tahun 2013 SMK Bina Wirausaha Talaga memiliki gedung sekolah sendiri yang berdomisili di Jalan raya Cikijing – Talaga.

2.1.2 Visi dan Misi SMK Bina Wirausaha Talaga

Visi dari SMK Bina Wirausaha Talaga yaitu menjadi lembaga pendidikan profesional yang akan melahirkan sumber daya manusia Handal, mandiri dan berakhlak mulia.

Misi dari SMK Bina Wirausaha Talaga adalah:

1. Menyelenggarakan pendidikan berkualitas yang dapat menghasilkan lulusan yang siap berwirausaha dan bekerja dengan biaya terjangkau oleh seluruh lapisan masyarakat.
2. Mencetak generasi muda lulusan SMK yang siap untuk mandiri usaha, profesional, Handal, dan berakhlak mulia.

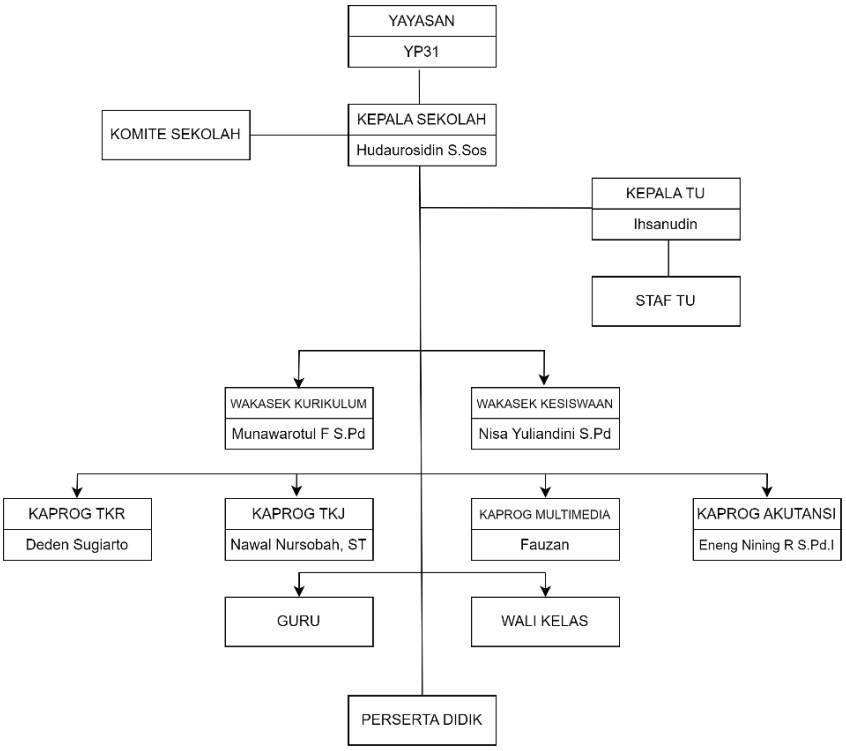
2.1.3 Tempat dan Kedudukan Sekolah

SMK Bina Wirausaha Talaga berdomisili di Jl. Raya Cikijing – Talaga, Blok Jajaway Desa Jatipamor, SMK Bina Wirausaha Talaga memiliki beberapa program keahlian sebagai berikut:

1. Jurusan Teknik Komputer Jaringan (TKJ)
2. Jurusan Teknik Kendaraan Ringan (TKR)
3. Jurusan Multimedia (MM)
4. Jurusan Akuntansi (AK)

2.1.4 Struktur Organisasi SMK Bina Wirausaha Talaga

Berikut ini adalah struktur organisasi di SMK Bina Wirausaha Talaga terdapat pada gambar 2.1



Gambar 2.1 Stuktur Organisasi Sekolah

2.1.5 Logo SMK Bina Wirausaha Talaga

Logo SMK Bina Wirausaha Talaga pada gambar 2.2

Gambar 2.2 Logo sekolah

2.2 Landasan Teori

Landasan teori adalah dasar atau asumsi yang menjadi dasar bagi suatu teori atau konsep. Landasan teori meliputi prinsip, pandangan, atau hipotesis yang menjadi dasar bagi suatu perumusan teori. Landasan teori bertujuan untuk memberikan dasar yang kuat bagi suatu teori dan membantu dalam memahami dan menjelaskan suatu fenomena, serta membantu dalam penyusunan penelitian.

2.2.1 Software Quality Assurance

Software Quality Assurance (SQA) adalah proses yang dirancang untuk memastikan bahwa perangkat lunak atau software yang dibuat memenuhi standar kualitas tertentu dan memenuhi persyaratan bisnis dan pengguna. Tujuan utama dari SQA adalah untuk memastikan bahwa perangkat lunak yang dikembangkan memiliki kinerja yang handal, dapat diandalkan, aman, dan memenuhi kebutuhan pengguna dan pelanggan. SQA juga berfokus pada pengembangan dan penerapan standar dan praktik terbaik yang dapat meningkatkan kualitas perangkat lunak dan memastikan keselarasan antara perangkat lunak dan kebutuhan bisnis dan pengguna [5].

2.2.2 Maintainability

Maintainability adalah kemampuan suatu perangkat lunak atau sistem untuk dipelihara atau diperbaiki dengan mudah dan efisien setelah diimplementasikan. Dalam konteks pengembangan perangkat lunak, maintainability merupakan salah satu faktor kualitas yang sangat penting karena setelah perangkat lunak diterapkan, sering kali dibutuhkan perawatan dan perbaikan untuk menjaga kinerja perangkat lunak agar tetap optimal [3]. Dalam *McCall*, *maintainability* memiliki beberapa sub faktor yang dapat dilihat pada tabel 2.1.

Tabel 2.1 Sub-faktor *maintainability*

Faktor	Sub-Faktor	keterangan
Maintainability	*Simplicity*	Kemudahan dalam memahami perangkat lunak
	Modularity	Ukuran modul pada perangkat lunak
	Self-Descriptiveness	Kode sumber yang mudah untuk dipahami
	Coding and Documentation Guidelines	Ketersediaan petunjuk penulisan kode dan dokumentasi
	Compliance (Cosistency)	Penggunaan desain dan teknik implementasi yang seragam
	Document Accessibility	Kemudahan pengaksesan dokumen perangkat lunak

2.2.3 Maintainability Index

Maintainability Index (MI) adalah metrik yang digunakan untuk mengukur seberapa mudah suatu perangkat lunak dapat dipelihara atau diperbaiki setelah diimplementasikan [10]. MI biasanya dihitung berdasarkan beberapa faktor, seperti kompleksitas kode, struktur program, dokumentasi, dan ukuran perangkat lunak. Maintainability Index (MI) merupakan kalkulasi formula yang didasarkan pada empat faktor utama, yaitu Lines of Code (LOC), Cyclomatic Complexity, dan Halstead Volume (HV), dan persentase jumlah komentar per modul (COM) [5]. keempat faktor ini dianggap sebagai indikator penting dalam mengevaluasi maintainability suatu perangkat lunak. Berikut formula *Maintainability Index*

$$MI = 171 - 5{,}1 * \ln(HV) - 0{,}28 * CC - 16{,}2 * \ln(LOC) + 50 * \sin\left(\sqrt{2{,}4 * COM}\right)$$

Keterangan:

HV (Halstead Metrics Volumes) = ukuran yang menghitung kompleksitas kode dari segi penggunaan variabel, operator, dan ekspresi dalam program.

CC (Cyclomatic Complexity) = ukuran yang menghitung kompleksitas struktur kontrol alur program dalam program.

LOC (Lines of Code) = ukuran sederhana yang menghitung jumlah baris kode dalam program.

COM (Percent line of comment) = ukuran yang menghitung persentase baris kode yang berisi komentar dalam program.

Akan tetapi, terdapat banyak varian dalam melakukan perhitungan *Maintainability Index*. Seperti yang digunakan oleh tools PHPmetric, berikut formula *Maintainability Indexnya.*

$$MI = \frac{171 - 5{,}2 * \log(HV) - 0{,}23 * CC - 16{,}2 * LN\ (LOC)}{171} * 100$$

$$Dimana\ 0\ \leq\ MI \leq 100$$

Pada penelitian ini formula Maintainability Index yang akan digunakan adalah formula yang digunakan pada tools PHPMetric. Secara umum nilai klasifikasi maintainability index diukur dari 0 sampai 100 [5]. Nilai tersebut terbagi menjadi tiga kategori, nilai maintainability index dapat dilihat pada tabel 2.2

Tabel 2.2 Klasifikasi Maintainability Index [5]

Maintainability Index	Klasifikasi
MI > 85	Dapat dipelihara dengan baik
65 < MI ≤ 85	Cukup untuk dapat dipelihara
MI ≤ 65	Sulit untuk dapat dipelihara

2.2.3.1 Halstead Metic

Halstead Metric adalah suatu metrik perangkat lunak yang digunakan untuk mengukur kompleksitas kode program dengan menggunakan penghitungan *matematis*. Metrik ini digunakan untuk mengukur berbagai aspek dari kode program, termasuk ukuran program, kompleksitas program, dan jumlah kesalahan yang mungkin terjadi dalam program, Semakin kompleks kode sumber, semakin sulit untuk memahami, menguji, dan memelihara kode sumber tersebut [6]. Oleh karena itu, penggunaan *Halstead's Metrics* dan metrik perangkat lunak lainnya dapat membantu developer dalam memperkirakan kompleksitas program dan mengambil tindakan yang tepat untuk mengurangi kesalahan dalam kode sumber. Pengukuran halstead didasarkan pada variabel-variabel berikut:

n1 = jumlah operator unik atau berbeda.

n2 = jumlah operan unik atau berbeda.

N1 = jumlah total kemunculan operator.

N2 = jumlah total kemunculan operan.

Halstead's Metrics mempunyai enam jenis komponen yang digunakan untuk mengukur kompleksitas program, yaitu:

1. Legth of program

   Persamaan Legth of program dirumuskan sebagai berikut

   $N = N1 + N2$

   Keterangan:

   N1 = Jumlah Operator

   N2 = Jumlah Operand

2. Vocabulary of the program

   Persamaan Vocabulary of the program dirumuskan sebagai berikut

   $n = n1 + n2$

   Keterangan:

   n = Vocabulary of program

   n1 = Jumlah operator unik

   n2 = Jumlah operand unik

3. Volume of the program

   Persamaan Volume of the program dirumuskan sebagai berikut

   $V = (N1 + N2)log2(n1 + n2)$

   Keterangan:

   V = Volume of the program

   N = hasil pengukuran length of the program

   n = hasil pengukuran vocabulary of the program

4. Difilculty

   Persamaan Difilculty dirumuskan sebagai berikut

   $D\frac{n1}{2} * \frac{N2}{n2}$

   Keterangan:

   D = Difficulty

   N2 = Total semua operan yang ada

   n1 = Jumlah total operator unik

   n2 = Jumlah total operan unik

5. Effort

   Persamaan effort dirumuskan sebagai berikut

   $E = D \times V$

   Keterangan:

   E = Effort

   D = nilai difficulty

   V = Volume of the program

6. Number of bugs expected in the program

7. Persamaan Number of bugs expected in the program

   Dapat dirumuskan

   $\mathrm{B}\frac{v}{3000}$

   Keterangan:

   B = number of bugs expected in the program

   V = nilai kalkulasi Volume of the program

2.2.3.2 Cylomatic Complexity

Cyclomatic Complexity menghitung jumlah jalur independen dalam kode sumber yang ditentukan oleh percabangan (if statement, switch case, loop, dsb.) dan titik masuk (entry point) ke kode tersebut. Semakin banyak jalur independen yang ada dalam kode sumber, semakin kompleks kode tersebut [7]. Cyclomatic Complexity sering kali digunakan untuk menentukan seberapa mudah kode tersebut dapat diuji, dipelihara, dan dimodifikasi**.**

Cyclomatic Complexity dapat dihitung dengan menggunakan rumus:

(g) = $e - n + 2$ = dengan menghitung nodes dan edge

(g) = $p + 1$ = dengan menghitung node percabangan atau predicate node.

Keterangan:

e = Jumlah edge

n = Jumlah node

p = Jumlah predicate node

2.2.4 Phpmetric

PHPMetrics adalah alat analisis kode sumber PHP open-source yang digunakan untuk memeriksa kompleksitas dan kualitas kode sumber PHP [8] [10]. Alat ini dapat membantu memahami kompleksitas kode PHP, mengidentifikasi area yang memerlukan perbaikan, dan meningkatkan efisiensi dan keandalan kode. Dengan analisis metrik yang diberikan, dapat memahami area mana yang perlu diperbaiki untuk meningkatkan efisiensi dan keandalan kode. Pengukuran phpmetrics memiliki pengukuran yang cukup lengkap beberapa diantaranya adalah sebagai berikut:

1. Lack of Cohesion Method

   Menghitung jumlah dari method - method berbeda dalam suatu kelas yang menggunakan variabel dalam kelas tersebut.

2. Cyclomatic Complexity

   Menghitung kompleksitas suatu program dengan mengukur banyaknya alur kontrol dalam suatu modul.

3. Perhitungan Operator Dan Operan

Phpmetrics dapat menghitung keseluruhan operator dan operan pada tiap filenya sehingga dapat memudahkan untuk pengukuran halsted metric kedepannya.

4. Perhitungan Volume of program

   Phpmetrics dapat melakukan perhitungan volume of program pada halstead metric.

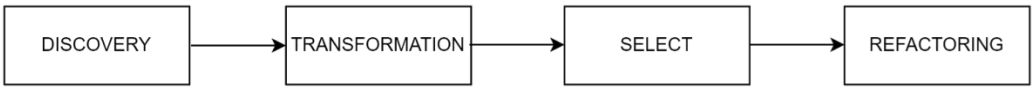
2.2.5 Code Readability Metric

Code Readability Metric adalah ukuran atau metrik yang digunakan untuk mengukur seberapa mudah kode sumber dapat dibaca, dimengerti, dan dipelihara oleh programmer lain. Metrik ini biasanya digunakan untuk mengevaluasi kualitas kode sumber dan membantu meningkatkan kualitas kode sumber.

Dengan menggunakan Code Readability Metric, programmer dapat mengevaluasi kualitas kode sumber dan mengidentifikasi area yang perlu diperbaiki untuk meningkatkan readability dan kemudahan pemeliharaannya [9]. Misalnya, jika Cyclomatic Complexity dari sebuah fungsi terlalu tinggi, programmer dapat mempertimbangkan untuk membagi fungsi menjadi beberapa bagian atau menggunakan teknik refactoring lainnya untuk mengurangi kompleksitas kode sumber. Hal ini akan membuat kode sumber lebih mudah dipahami dan dipelihara.

2.2.6 Refactoring

Refactoring adalah proses mengubah kode program yang sudah ada dengan tujuan meningkatkan kualitas, membaca dengan mudah, dan mempermudah pemeliharaan kode tersebut tanpa mengubah fungsionalitasnya [11]. *Refactoring* juga dapat membantu mengurangi redundansi dan meningkatkan efisiensi kode. Adapun tahapan proses yang dilakukan dalam *refactoring* secara umum dapat dilihat pada gambar 2.3



Gambar 2.3 Tahapan refactoring secara umum

1. *Discovery*

Discovery merupakan proses untuk menemukan dan memahami kode yang perlu diperbaiki atau diubah. Ini melibatkan identifikasi area kode yang kompleks, tidak efisien, sulit dipahami, atau rentan terhadap kesalahan.

2. *Transformation*

   Tahapan "*Transformation*" dalam refactoring adalah langkah-langkah konkret yang diambil untuk mengubah atau memperbaiki kode yang telah diidentifikasi pada tahap *discovery*. Transformasi ini dilakukan dengan tujuan meningkatkan struktur, kebersihan, dan kualitas keseluruhan dari kode yang ada.

3. *Select*

   Tahapan ini dilakukan pemilihan dari bentuk solusi yang rerbentuk agar solusi yang akan diterapkan merupakan solusi yang terbaik yang dapat diberikan.

4. *Refactoring*

   Tahapan *refactoring* akan dilakukan penerapan solusi yang dipilih.

2.2.7 Clean Code

Clean code atau kode bersih adalah kode program yang ditulis dengan cara yang mudah dipahami dan dipelihara oleh pengembang lain. Kode bersih memiliki struktur yang terorganisir, mudah dibaca, dan efisien, sehingga dapat meminimalkan kesalahan dan mempercepat pengembangan [11]. Prinsip dasar dari clean code adalah meningkatkan kejelasan, keterbacaan, dan pemeliharaan kode.

2.2.7.1 Meaningful Names

Meaningful names atau nama yang bermakna adalah salah satu hal yang sangat penting dalam clean code. Nama yang jelas dan bermakna dapat membantu pengembang lain memahami kode dengan mudah dan cepat. Nama yang buruk atau tidak bermakna dapat membuat kode sulit dipahami dan memperlambat pengembangan. Contoh dari konsep ini dapat dilihat pada gambar 2.4

```
Buruk
public function lhtGuru(){
        $act = 'guru';
        $ttl = 'Admin - Guru';
        $dg = Guru::all();
Baik
public function lihatGuru(){
        $action = 'guru';
        $tttle = 'Admin - Guru';
        $dataGuru = Guru::all();
```

Gambar 2.4 Contoh penerapan konsep meaningfull names

Dalam konsep celan code pada meaningful names terdapat beberapa sub-sub prinsip yang bertujuan untuk membuat code lebih dipahami, prinsip-prinsip tersebut antara lain adalah:

a. *Use intention-revealing names*

   Pastikan penamaan yang digunakan memiliki makna yang mudah dimengerti, mengungkapkan niat atau tujuan dari elemen yang dinamai. Agar membantu pengembang memahami maksud dan tujuan dari kode lebih cepat [23].

b. *Avoid Disinformation*

   Prinsip ini menekankan jangan menggunakan penamaan yang bisa menyebabkan salah arti atau membingungkan, pemilihan penamaan harus mencerminkan secara akurat penamaan elemen yang dinamai [23].

c. *Make Meaningful Distinctions*

   Gunakan penamaan yang memiliki makna berbeda, jika ada beberapa penamaan yang mirip pastikan pemilihan nama tersebut berbeda agar tidak menimbulkan kebingungan [23].

d. *Use pronounceable names*

   Gunakan pemilihan nama yang dapat diucapkan, pastikan penamaan variabel atau method menggunakan kata yang mudah untuk diucapkan [23].

e. *Use Searchable Names*

   Pastikan pemilihan penamaan mudah untuk dicari, hindari juga penamaan yang menggunakan integer atau angka atau menggunakan sama *single letter* seperti H, S dan sebagainya [23].

f. *Avoid Mental Mapping*

Hindari memaksa pembaca kode untuk mengartikan atau menerjemahkan makna nama. Nama-nama seharusnya cukup jelas sehingga tidak memerlukan usaha berlebihan untuk memahaminya [23]

g. *Class Names*

Pemilihan nama kelas sebaiknya mencerminkan tujuan dan tanggung jawab dari kelas tersebut, gunakan kata kerja [23].

h. *Method Names*

Nama-nama method sebaiknya menggambarkan dengan jelas apa yang akan dilakukan oleh metode tersebut [23].

i. *Avoid encoding*

Hindari pengkodean dalam penamaan, seperti singkatan yang tidak jelas atau kode yang tidak bermakna. Nama-nama seharusnya informatif dan dapat dimengerti oleh orang lain [23].

Masih ada beberapa sub prinsip pada konsep clen code meaningful names seperti dont be cute, pick one word per concept, dont pun, use splution domain names, use problem domain names, add meaning context, dan dont add gtatuitious sontext.

2.2.7.2 Clean Funtions

Sebuah konsep dalam clean code yang mengacu pada cara menulis fungsi atau method yang bersih dan mudah dipahami. Fungsi yang bersih dan mudah dipahami dapat membantu dalam pengembangan dan pemeliharaan kode yang lebih mudah, serta memudahkan pengembang lain dalam memahami dan menggunakan kode yang ditulis. Adapun contoh dari penerapan konsep celan function dapat dilihat pada gambar 2.5.

```
Buruk
procedure pencarian_kode(var bData : integer; var BK : Buku);
var
kode:string;
posisidata:integer;
begin
clrscr;
writeln('Pencarian berdasarkan Kode Buku');
writeln('------------------------------');
write('Kode buku yang dicari : ');readln(kode);
posisidata:=sKode(bData,BK,kode);
if posisidata<>0 then
begin
writeln('Data ditemukan di posisi ',posisidata);
writeln('Kode : ',BK[posisidata].kodeBK);
end
else
writeln('Data tidak ditemukan');
writeln('Tekan Enter Untuk Melanjutkan.');readln;
end;

Baik
procedure cariBukuDariKode(var bData: integer; var BK: Buku);
var
kode: string;
posisidata: integer;
begin
clrscr;
writeln('Pencarian berdasarkan Kode Buku');
writeln('------------------------------');
write('Kode buku yang dicari : ');
readln(kode);
```

Gambar 2.5 Contoh penerapan konsep clean funtions

Terdapat beberapa prinsip-prinsip pada celan functions, prinsip-prinsip tersebut adalah:

a. *Small*

   Fungsi sebaiknya dibuat dengan ukuran yang kecil dan fokus pada tugas-tugas yang spesifik. Fungsi yang baik maksimal 20 baris [22].

b. *Do one thing*

   Satu fungsi hanya melakukan satu tugas, hindari 1 fungsi yang melakukan banyak tugas [23].

c. *Functions arguments*

   Batasi jumlah argumen yang diterima oleh fungsi. Terlalu banyak argumen dapat membingungkan dan mempersulit penggunaan fungsi [23].

d. *Have No Side Effects*

   Fungsi sebaiknya tidak memiliki efek samping yang tidak diharapkan [23].

e. *Command Query Separation*

   Prinsip ini menyatakan bahwa sebuah fungsi seharusnya entah memberikan komando (mengubah sesuatu) atau memberikan kueri (mengembalikan informasi). Fungsi yang mencampur keduanya dapat menyebabkan kebingungan [23].

f. *Don't Reapeat Yourself*

Hindari pengulangan kode atau duplikasi kode sumber baik pada class yang sama ataupun berbeda [23].

Masih ada beberapa sub prinsip seperti Prefer Exceptions to Returning Error Codes, Extract Try/Catch Blocks, Error Handling Is One Thing, The Error.java Dependency Magnet dan Structured Programming.

2.2.7.3 Clean Comments

Clean Comments adalah komentar yang ditulis secara jelas dan informatif dalam kode program. Komentar yang bersih dan terstruktur dapat membantu pengembang lain memahami kode yang ditulis dengan mudah, dan dapat mempercepat proses pemeliharaan dan pengembangan kode. Contoh penerapan konsep clean comment dapat dilihat pada gambar 2.6

```
Buruk
function hashIt(data) {
   // ini hash
   let hash = 0;

   // panjang dari sebuah string
   const lenght = data.leght;

  //Lopp setiap karakter dalam sebuah data
  for (let i = 0; i <length; i++) {
     // mendapatkan karakter di kode
     const char= data.charCodeAt(i);
     // membuat hash
     hash = (hash << 5) - hash) + char;
     // mengonversi ke 32-bit integer
     hash &= hash;
   }
}

Baik
function hashIt(data) {
   let hash = 0;
   const lenght = data.leght;


  for (let i = 0; i <length; i++) {
     const char= data.charCodeAt(i);
     hash = (hash << 5) - hash) + char;
     // mengonversi ke 32-bit integer
     hash &= hash;
   }
}
```

Gambar 2.6 Contoh penerapan konsep clean commnent

2.2.7.4 Clean Code Formatting

Konsep dalam clean code yang mengacu pada cara format dan tata letak kode

yang bersih, terstruktur, dan mudah dibaca. Penataan dan tata letak kode yang baik dapat membantu pengembang lain memahami kode dengan lebih mudah dan mempercepat proses pengembangan dan pemeliharaan kode. Adapun contoh penerapan konsep clean code formatting dapat dilihat pada gambar 2.7

```
Buruk
public function index(){
        $title = 'Dashboard Siswa';
        $action = 'siswa';
        $mapelajaran = Course::where('siswa_id', auth()->user()->id)->get();
        return view('siswa.dash',[
            'ttl' => $title,
            'act' => $action,
            'mapel' => $mapel,
        ]);
    }
    public function crs($id){
        $ttl = 'Course - ';
        $course = Course::where('pelajaran_id', $id)->first();
        $mtr = Materi::where('pelajaran_id', $id)->get();
        return view('siswa.course',[
            'ttl' => $title,
            'crs' => $cours,
            'mtr' => $materi,
        ]);
    }
Baik
public function index(){
        $title = 'Dashboard Siswa';
        $action = 'siswa';
        $mapel = Course::where('siswa_id', auth()->user()->id)->get();
        return view('siswa.dash',[
            'title' => $title,
            'action' => $action,
            'mapel' => $mapel,
        ]);
    }
    public function crs($id){
        $title = 'Course - ';
        $cours = Course::where('pelajaran_id', $id)->first();
        $materi = Materi::where('pelajaran_id', $id)->get();
        return view('siswa.course',[
            'title' => $title,
            'cours' => $cours,
            'materi' => $materi,
        ]);
    }
```

Gambar 2.7 Contoh penerapan konsep Clean Code Formatting

2.2.7.5 Clean Error Handling

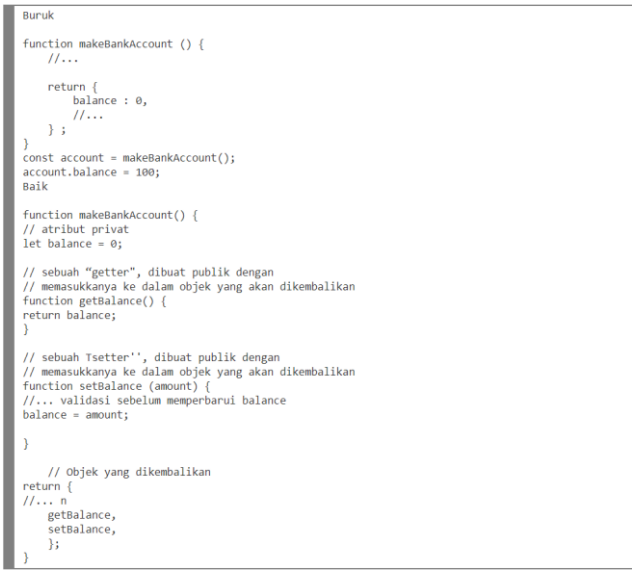
Clean Error Handling merujuk pada praktik-praktik yang digunakan untuk mengelola dan menangani kesalahan (errors) pada program dengan cara yang jelas, efektif, dan mudah dipahami. Tujuannya adalah untuk meminimalkan risiko terjadinya kesalahan dan memperbaiki kesalahan dengan cepat dan akurat saat terjadi. Contoh penerapan konsep clean error handling dapat dilihat pada gambar 2.8

```
Buruk
function authenticateAdmin(Request $request){
        $credentials = $request->validate([
            'email' => 'required',
            'password' => 'required',
        ]);

        if(Auth::attempt($credentials)){
            $request->session()->regenerate();
            return redirect()->intended('/dashadmin/kelas');
        }
        return back()->with('LoginError','Login Gagal');
    }

Baik
function authenticateAdmin(Request $request){
    $credentials = $request->validate([
        'email' => 'required',
        'password' => 'required',
    ]);

    if(Auth::attempt($credentials)){
        $request->session()->regenerate();
        return redirect()->intended('/dashadmin/kelas');
    } else {
        return back()->withErrors([
            'LoginError' => 'Email atau password salah. Silakan coba lagi.',
        ]);
    }
}
```

Gambar 2.8 Contoh penerapan konsep Clean Error Handling

2.2.7.6 Clean Object dan Data Stuctures

Paradigma pemrograman yang berfokus pada penggunaan struktur data dan objek yang mudah dipahami dan mudah dimodifikasi. Clean ODS menekankan penggunaan struktur data dan objek yang sederhana, tidak rumit, dan terorganisir dengan baik untuk mengurangi kompleksitas dan meningkatkan kejelasan program. Adapun contoh dari penerapan konsep cleans object dan data stuctures dapat dilihat pada gambar 2.9



```
Buruk

function makeBankAccount () {
    //...

    return {
        balance : 0,
        //...
    } ;
}
const account = makeBankAccount();
account.balance = 100;
Baik

function makeBankAccount() {
// atribut privat
let balance = 0;

// sebuah “getter”, dibuat publik dengan
// memasukkanya ke dalam objek yang akan dikembalikan
function getBalance() {
return balance;
}

// sebuah Tsetter'', dibuat publik dengan
// memasukkanya ke dalam objek yang akan dikembalikan
function setBalance (amount) {
//... validasi sebelum memperbarui balance
balance = amount;

}

    // Objek yang dikembalikan
return {
//... n
    getBalance,
    setBalance,
    };
}
```

Gambar 2.9 Contoh penerapan Clean objec dan Data Stuctures

Ada beberapa prinsip clean object and data structures, prinsip tersebut antara lain:

a. *Data Abstraction*

   Abstraksi data mengacu pada konsep dalam pemrograman di mana Anda menyembunyikan detail internal suatu objek dan hanya mengekspos fungsionalitas yang penting atau relevan [23].

b. *Data/Object Anty Symentrry*

   Objek menyembunyikan datanya di balik abstraksi dan menampilkan fungsi yang beroperasi pada data tersebut [23].

c. *The Law of Dementer*

   Prinsip ini menyarankan bahwa sebuah objek seharusnya hanya berinteraksi dengan objek-objek yang berdekatan dengannya. [23].

d. *Data Transfer Objects*

   Tujuan utama DTOs adalah meminimalkan jumlah panggilan ke basis data atau layanan jarak jauh dengan mengumpulkan beberapa data ke dalam objek DTO yang satu dan mengirimnya dalam satu panggilan [23].

2.2.7.7 Clean Classes

Pada konsep ini terdapat petunjuk dalam membuat modul yang lebih bersih dan terorganisir. Contoh penerapan konsep clean classes dapat dilihat pada gambar 2.10

```
Buruk
class UserStettings {
    constuctor (user){
        this.user = user;
    }
    changeSetting (setting) {
        if (this.verifyCredentials()){
            //...
        }
    }
    verifyCredentials() {
        //...
    }
}

Baik
class UserAuth {
    constuctor (user){
        this.user = user;
    }
    changeSetting (setting) {
        if (this.verifyCredentials()){
            //...
        }
    }
    verifyCredentials() {
        //...
    }
}
class UserSettings {
    constuctor (user){
        this.user = user;
        this.auth = new UserAuth (user);
   }
   changeSettings (settings) {
    if (this.auth.verifyCredentials()) {
       //...
     }
   }
}
```

Gambar 2.10 Contoh penerapan Clean Class

Ada beberapa prinsip clean class, prinsip tersebut antara lain:

a. *Class Organization*

   Prinsip ini menekankan pentingnya memisahkan tanggung jawab dan fungsi-fungsi yang berbeda ke dalam kelas-kelas terpisah, sehingga setiap kelas memiliki fokus yang jelas dan terorganisir dengan baik [23].

b. *Class Should Be Small*

   Prinsip ini mengajukan bahwa kelas seharusnya dibuat dengan ukuran yang kecil dan fokus pada satu tanggung jawab utama [23].

2.2.8 Design Pattern

Pola desain (design patterns) adalah solusi umum untuk masalah yang sering muncul dalam pengembangan perangkat lunak. Design patten merupakan pedoman atau panduan yang membantu pengembang perangkat lunak merancang struktur kode yang lebih baik, lebih terstruktur, dan lebih mudah dipahami. Pola desain membantu menghindari masalah umum dalam pengembangan perangkat lunak seperti kompleksitas yang berlebihan, kesalahan yang sulit diidentifikasi, dan kesulitan dalam pemeliharaan kode.

Berdasarkan buku *Design Patterns Explained Simply* karangan Alexander Shvets. Ditemukan 3 kategori design pattern. Yaitu, *Creational patterns, Stuctural patterns*, dan *Behavioral patterns*. Kategori design pattern *Creatinal Pattern* dapat dilihat pada tabel 2.3

Tabel 2.3 Kategori design patterns

Kategori Design Pattern	Design Pattern	Penjelasan
Creatinal Pattern	*Singletone*	*Singleton* adalah sebuah pola desain (design pattern) yang digunakan dalam pengembangan perangkat lunak untuk memastikan bahwa sebuah kelas hanya memiliki satu instance (objek) tunggal yang dapat diakses dari seluruh bagian aplikasi.
	Factory Method	Pola desain ini fokus pada pembuatan objek-objek, dengan tujuan untuk mengabstraksi proses pembuatan objek sehingga aplikasi dapat lebih fleksibel dalam menghasilkan objek-objek yang sesuai dengan kebutuhan.
	Prototype	Tujuan utama dari pola Prototype adalah untuk menciptakan objek baru

Kategori Design Pattern	Design Pattern	Penjelasan
		dengan menggandakan (kloning) objek yang sudah ada, yang sering disebut sebagai "prototipe." Ini sangat berguna ketika pembuatan objek yang baru memerlukan biaya yang tinggi atau kompleksitas yang tinggi.
	Abstract Factory	Abstract Factory adalah salah satu pola desain kreasional (creational design pattern) yang digunakan dalam pengembangan perangkat lunak. Pola desain ini bertujuan untuk menciptakan keluarga objek yang terkait atau berhubungan, tanpa harus mengungkapkan detail implementasi dari objek-objek tersebut kepada klien
	Builder	Pola desain Builder digunakan untuk mengkonstruksi objek yang kompleks dengan banyak atribut atau parameter konfigurasi yang beragam. Tujuannya adalah untuk membuat proses pembuatan objek menjadi lebih terstruktur, fleksibel, dan mudah dipahami.
Behavioral Pattern	*Chain Of Responbility*	Pola desain ini bertujuan untuk menghubungkan sejumlah objek pemrosesan menjadi rantai (chain), di mana setiap objek dalam rantai memiliki kemampuan untuk

Kategori Design Pattern	Design Pattern	Penjelasan
		memproses permintaan (request), atau meneruskannya ke objek berikutnya dalam rantai.
	Command	Pola desain ini bertujuan untuk mengubah permintaan atau tindakan menjadi objek terkonkritkan, sehingga memungkinkan pemisahan antara pengirim permintaan (client) dan objek yang menjalankan permintaan tersebut.
	Interpreter	Pola desain ini bertujuan untuk menginterpretasikan atau mengevaluasi ekspresi atau pernyataan yang terdiri dari simbol-simbol atau token-token. Dengan menggunakan pola Interpreter, dapat membangun suatu bahasa atau tata bahasa (grammar) dan memungkinkan aplikasi untuk menafsirkan atau menjalankan perintah atau instruksi yang dinyatakan dalam bahasa tersebut.
	Iterator	Pola desain Iterator digunakan untuk mengakses elemen-elemen koleksi objek tanpa harus mengungkapkan struktur internal koleksi tersebut. Ini memungkinkan klien (pengguna) untuk mengakses elemen-elemen koleksi secara sekuensial tanpa harus

Kategori Design Pattern	Design Pattern	Penjelasan
		tahu tentang implementasi koleksi atau cara mengaksesnya.
	Mediator	Pola desain ini digunakan untuk mengurangi ketergantungan yang kuat antara berbagai komponen atau objek dalam suatu sistem dengan mengintroduksi objek mediator yang bertindak sebagai perantara (hub) antara komponen-komponen tersebut. Mediator membantu mengatur komunikasi dan koordinasi antara komponen-komponen tanpa perlu mereka tahu tentang satu sama lain.
	Memento	Pola desain ini digunakan untuk menyimpan dan mengembalikan keadaan internal suatu objek tanpa mengungkapkan detail implementasinya. Memento memungkinkan Anda untuk menangani penarikan kembali dan pencatatan keadaan objek dengan cara yang terstruktur.
	Observer	Pola desain ini memungkinkan objek-objek yang bergantung pada suatu objek untuk secara otomatis menerima pembaruan ketika keadaan subjek berubah. Dengan kata lain, Observer digunakan untuk

Kategori Design Pattern	Design Pattern	Penjelasan
		mengimplementasikan mekanisme notifikasi atau pemberitahuan antara objek-objek dalam sistem.
	State	Pola desain ini memungkinkan objek untuk mengubah perilaku atau keadaannya ketika kondisi internalnya berubah. Dengan menggunakan State, objek dapat terlihat seolah-olah berubah kelas ketika keadaannya berubah, tetapi tanpa harus mengubah kodenya secara ekstensif.
	Strategi	Pola desain ini memungkinkan Anda untuk mendefinisikan sejumlah algoritma atau strategi yang berbeda dan memungkinkan klien untuk memilih strategi yang sesuai pada waktu eksekusi. Strategy membantu memisahkan implementasi algoritma dari klien yang menggunakannya.
	Template Method	ola desain ini memungkinkan Anda untuk mendefinisikan kerangka dari sebuah algoritma dalam sebuah metode, tetapi memungkinkan subkelas untuk mengganti beberapa langkah dari algoritma tersebut tanpa mengubah strukturnya. Dengan kata lain, Template Method memisahkan bagian yang tidak berubah dari

Kategori Design Pattern	Design Pattern	Penjelasan
		algoritma dari bagian yang berubah.
	Visitor	Pola desain ini digunakan untuk memisahkan algoritma dari objek-objek yang dioperasikan oleh algoritma tersebut. Dengan menggunakan Visitor, dapat menambahkan operasi baru ke objek-objek tanpa harus mengubah kelas-kelas objek tersebut.
Stuctural patterns	*Adapter*	Pola desain ini memungkinkan untuk menghubungkan dua antarmuka yang berbeda agar dapat bekerja bersama, meskipun mereka memiliki antarmuka yang tidak kompatibel.
	Bridge	Pola desain ini memisahkan antarmuka dari implementasi sehingga keduanya dapat berubah secara independen. Bridge membantu menghindari pembentukan kombinasi eksplosif dari kelas-kelas dengan beberapa varian dari implementasi dan antarmuka.
	Composite	Pola desain ini memungkinkan untuk menggabungkan objek-objek menjadi struktur pohon untuk mewakili bagian-bagian dari keseluruhan hierarki. Composite memungkinkan klien untuk memperlakukan objek tunggal dan komposisi objek dengan cara yang

Kategori Design Pattern	Design Pattern	Penjelasan
		sama.
	Decorator	Pola desain ini memungkinkan untuk menambahkan perilaku atau fungsi tambahan pada objek tanpa mengubah struktur dasar objek tersebut. Decorator memungkinkan komposisi objek dengan cara yang fleksibel dan dinamis.
	Flyweight	Pola desain ini bertujuan untuk mengurangi penggunaan memori atau penyimpanan dengan berbagi sebanyak mungkin informasi yang sama antara banyak objek. Ini sangat berguna ketika memiliki banyak objek yang sebagian besar memiliki data yang sama atau serupa.
	Proxy	Pola desain ini memungkinkan Anda untuk membuat objek pengganti (proxy) yang bertindak sebagai perantara atau representasi objek asli. Proxy digunakan untuk mengontrol akses ke objek asli atau untuk menambahkan fungsi tambahan saat mengakses objek tersebut.

2.2.9 Laravel

Laravel adalah sebuah kerangka kerja (*framework*) aplikasi web *opensource* yang ditulis dengan bahasa pemrograman PHP. Laravel dirancang dengan tujuan untuk memudahkan pengembangan aplikasi web dengan memperhatikan efisiensi

dan kualitas kode [12]. *Laravel* menawarkan banyak fitur yang dapat memudahkan pengembang dalam membuat aplikasi web, seperti sistem *routing* yang mudah dipahami, sistem template *engine*, ORM (*Object-Relational Mapping*) dengan *Eloquent*, sistem migrasi database, dan masih banyak lagi.

Selain itu, *Laravel* sangat mendukung pengembangan aplikasi berbasis API (*Application Programming Interface*) dan real-time. *Laravel* juga terus mengalami perkembangan dan peningkatan dari versi ke versi sehingga mampu memberikan pengalaman pengembangan yang lebih baik.

2.2.10 Analisis dan Desain Berorientasi Objek

Unified Modeling Language (UML) adalah bahasa modeling standar yang digunakan untuk menggambarkan, spesifikasi, dan dokumentasi sistem software dan aplikasi. UML menyediakan set diagram dan notasi visual yang membantu dalam menggambarkan dan memahami sifat dan perilaku sistem, seperti objek, class, interaksi, aktivitas, dan banyak lagi. UML banyak digunakan dalam proses pengembangan perangkat lunak dan dapat membantu dalam mempermudah komunikasi antara tim pengembang dan stakeholder [13]. Notasi UML terutama diturunkan dari 3 notasi yang telah ada sebelumnya yaitu: *Grady Booch OOD* (*Object Oriented Design*), Jim Rumbaugh OMT (*Object Modeling Technique*) dan Ivar Jacobson OOSE (*Object Oriented Software Engineering*).

2.2.9.1 *Diagram UML*

Unified Modeling Language (UML) adalah bahasa pemodelan grafis untuk menggambarkan, mendeskripsikan, mengkonstruksikan, dan mendokumentasikan artefak-artefak dari sebuah sistem Piranti lunak [9]. Didalam spesifikasi UML terdapat 13 jenis diagram yang terbagi ke dalam 3 jenis diagram [16].