1.**Abstraction (Abstraksi)**  
Abstraksi adalah proses menyembunyikan detail implementasi yang kompleks dan hanya menampilkan fitur penting kepada pengguna. Konsep ini memungkinkan pengembang untuk fokus pada apa yang dilakukan oleh suatu objek, bukan bagaimana cara kerjanya.

**Encapsulation (Enkapsulasi)**  
Enkapsulasi adalah konsep membungkus data dan fungsi menjadi satu kesatuan (kelas) serta membatasi akses terhadap data tersebut melalui **modifier akses** seperti public, private, atau protected. Hal ini menjaga integritas data dengan mencegah akses langsung dari luar kelas.

2.**Public**: Dapat diakses dari mana saja (baik di dalam kelas, kelas turunan, maupun luar kelas).

**Private**: Hanya dapat diakses di dalam kelas tempat properti/metode tersebut didefinisikan.

**Protected**: Hanya dapat diakses di dalam kelas itu sendiri dan kelas turunannya.

**Visibility** ini digunakan untuk mengontrol aksesibilitas dan melindungi data dari modifikasi yang tidak diinginkan.



3. **Inheritance (Pewarisan)**  
Pewarisan adalah konsep OOP di mana sebuah kelas baru (kelas turunan) mewarisi properti dan metode dari kelas induk. Ini memungkinkan penggunaan kembali kode dan memudahkan pengelompokan objek dengan sifat yang mirip.

**Fungsi Construct**  
Fungsi \_\_construct() adalah konstruktor yang dipanggil otomatis saat suatu objek dibuat. Pada kelas turunan, konstruktor bisa digunakan untuk:1. Menginisialisasi nilai-nilai awal. 2.Memanggil konstruktor kelas induk untuk mewarisi logika inisialisasi.



**MVC** adalah singkatan dari Model View Controller. MVC adalah teknik pemrograman yang memisahkan bisnis logic (alur pikir), data logic (penyimpanan data) da presentation logic (antarmuka aplikasi) atau secara sederhana adalah memisahkan antara desain, data dan proses.

**Model** Berhubungan dengan data dan interaksi ke text base, database atau webservice. Biasanya di ddlama model berisi akan berisi class dan fungsi untuk mengambil, melakukan update dan menghapus data web.

**View** Berhubungan dengan segala sesuatu yang ditampilkan ke end-user. Bisa berupa halaman web, rss, javascript dan lain-lain. Kita harus menghindari adanya logika atau pemprosesan data di view. View bisa dikatakan sebagai halaman yang dibuat dengan HTML dan bantuan CSS atau Javascript.

**Controller** Controller bertindak sebagai penghubung data dan view. Di Dalam controller inilah terdapat class-class dan fungsi-fungsi yang dapat memproses permintaan dari view ke dalam struktur data di dalam model. Controller juga tidak boleh berisi kode utuk mengakses basis data. Tugasnya adalah menyediakan berbagai variabel yag akan ditampilkan di view, memanggil model untuk melakukan akses ke basis data, menyediakan penanganan kesalahan, mengerjakan proses logika dari aplikasi dan melakukan validasi atau cek terhadap input.

**4.CONTROLLER GETPRODUCTLIST()**

1. **Kesalahan Fungsi getProductList() pada Controller Berdasarkan Konsep MVC**

**Secara Mewing:**

Dalam arsitektur MVC (Model-View-Controller), fungsi getProductList() di controller seharusnya tidak langsung mengakses database untuk mengambil data. Kesalahan dalam pendekatan ini meliputi:

1. **Tanggung Jawab yang Campur Aduk**: Controller seharusnya bertindak sebagai perantara antara model dan view. Jika controller langsung mengakses database, maka tanggung jawabnya bercampur dengan logika bisnis yang seharusnya berada di model.
2. **Kurangnya Abstraksi**: Mengambil data dari database seharusnya dilakukan melalui model yang mengatur interaksi dengan basis data. Ini meningkatkan keterbacaan dan memudahkan pemeliharaan kode.
3. **Keterhubungan yang Kuat**: Ketika controller terikat langsung dengan database, perubahan pada struktur database dapat mempengaruhi controller secara langsung, yang berpotensi menyebabkan kode lebih rentan terhadap kesalahan

**Secara Memusingkan:**

Dalam arsitektur Model-View-Controller (MVC), masing-masing komponen memiliki tanggung jawab yang spesifik:

Model bertanggung jawab untuk mengelola data dan logika bisnis, termasuk interaksi dengan database.

View menampilkan data ke pengguna tanpa berhubungan langsung dengan logika bisnis.

Controller bertindak sebagai penghubung antara Model dan View, mengambil permintaan dari pengguna, meminta data dari Model, dan kemudian mengarahkan data ke View.

Pada skenario yang diberikan, fungsi getProductList() pada Controller mengambil data langsung dari database, memproses data tersebut, lalu mengirimkan data ke View. Ini bertentangan dengan konsep MVC, karena:

Controller seharusnya tidak langsung berinteraksi dengan database. Tugas ini seharusnya dilakukan oleh Model.

Logika pengambilan dan pemrosesan data produk harus berada di dalam Model, karena Model bertanggung jawab atas interaksi database dan logika bisnis.

Kesalahan ini melanggar prinsip separation of concerns dalam MVC, di mana setiap komponen memiliki tugas spesifik dan tidak tumpang tindih. Jika Controller mengambil data langsung dari database, ini akan mengacaukan struktur kode, membuat kode lebih sulit di-maintain, dan mengurangi skalabilitas.

**B. Cara yang Lebih Tepat untuk Menangani Fungsi getProductList() agar Sesuai dengan MVC**

Agar sesuai dengan konsep MVC, sebaiknya kita:

* 1. Pindahkan logika pengambilan data dari database ke Model. Buat fungsi di Model, misalnya ProductModel::getAllProducts(), untuk mengambil data produk dari database.
  2. Controller memanggil fungsi dari Model untuk mendapatkan data produk tanpa langsung berinteraksi dengan database.
  3. Controller meneruskan data dari Model ke View tanpa melakukan logika pengambilan data atau interaksi dengan database.

**CODE MEWING SIGMA**



**CODE RIBET MALAS NULIS NJIR:**

Contoh Implementasi yang Sesuai dengan MVC

1. Model – Menangani logika pengambilan data produk dari database:

<?php

Class ProductModel {

Public function getAllProducts() {

// Koneksi dan logika pengambilan data dari database

// Misalnya menggunakan PDO atau MySQLi

$db = Database::getConnection();

$query = $db->prepare(“SELECT \* FROM products”);

$query->execute();

Return $query->fetchAll(PDO::FETCH\_ASSOC);

}

}

?>

1. Controller – Meminta data dari Model dan mengirimkannya ke View:

<?php

Class ProductController {

Public function getProductList() {

$productModel = new ProductModel(); // Menginisialisasi model

$products = $productModel->getAllProducts(); // Mengambil data dari Model

Require ‘views/productListView.php’; // Mengirim data ke View

}

}

?>

1. View – Menampilkan data produk ke pengguna:

<!—views/productListView.php 🡪

<?php foreach ($products as $product): ?>

<p><?php echo $product[‘name’]; ?> - <?php echo $product[‘price’]; ?></p>

<?php endforeach; ?>

**TAMBAHAN KALO MASUK**

**Function Setter** adalah fungsi yang digunakan untuk memberikan (set) nilai untuk suatu properti. Umumnya function ini digunakan untuk properti yong mempunyai visibilitas protected atau private. Function setter sering digunakan untuk memvolidasi data yong masuk sebelum data tersebut diisikon ke suatu properti.

**Function Getter** adalah suatu function yong digunakan untuk menghasilkon (get) suatu nilai baik dari member public, private, protected atau pun dari hasil perhitungan.



