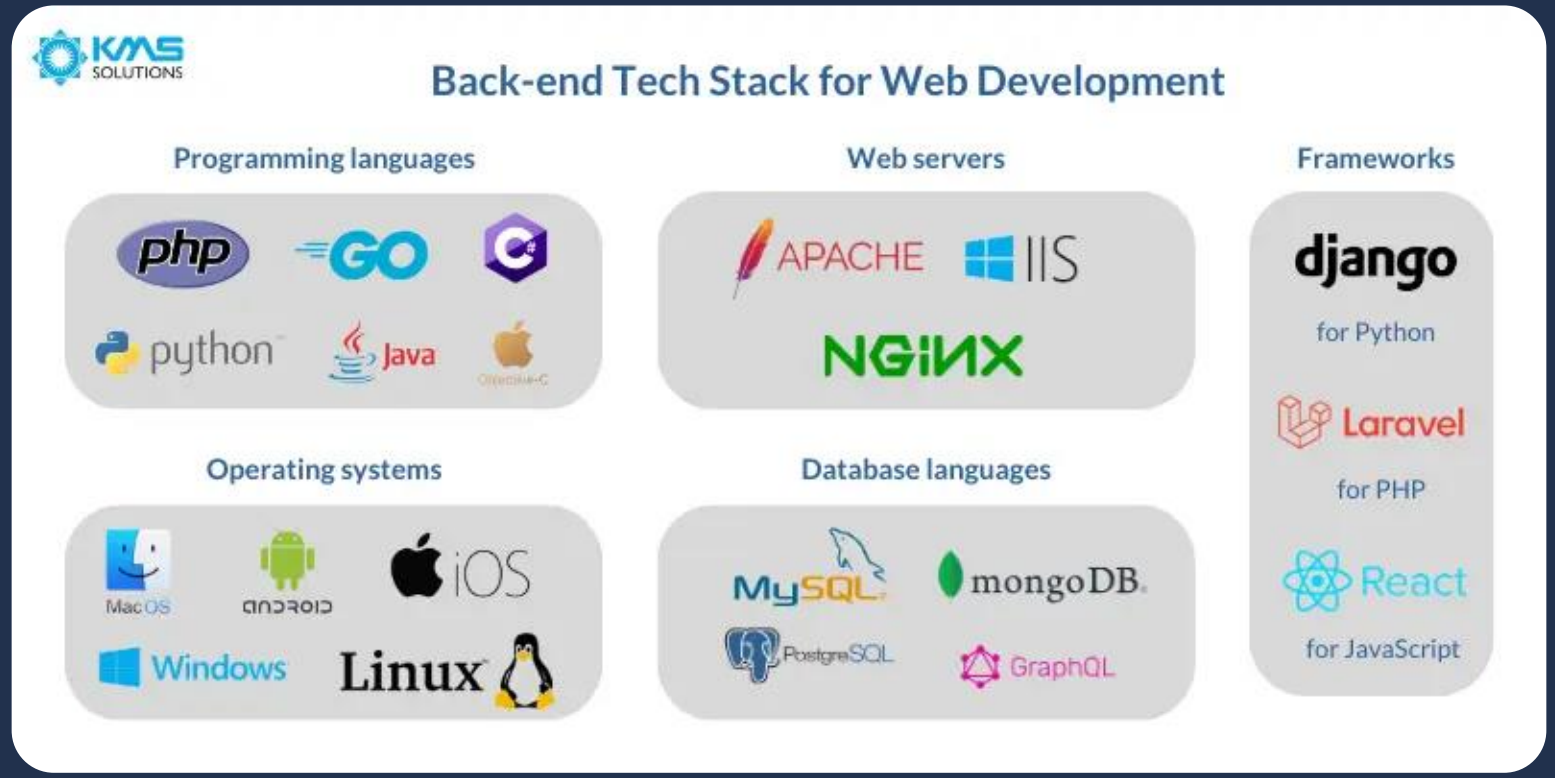
CSE485 – Công nghệ Web

dungkt@tlu.edu.vn



Bài 5. Mở rộng ứng dụng

NỘI DUNG

1. Tiêm phụ thuộc
2. Mô hình MVC

1. Tiêm phụ thuộc (Dependency Injection)

- **Dependency Injection (DI)** là một kỹ thuật trong lập trình phần mềm, nơi một đối tượng nhận các đối tượng mà nó phụ thuộc (dependencies) từ một nguồn bên ngoài thay vì tạo chúng trực tiếp. Điều này tăng cường sự linh hoạt, giảm sự phụ thuộc giữa các thành phần mã nguồn và làm cho việc kiểm thử (testing) dễ dàng hơn.
  - Hãy tưởng tượng bạn đang xây dựng một chiếc xe hơi. Thay vì chế tạo động cơ bên trong chiếc xe, bạn chọn một động cơ từ một nhà cung cấp và lắp nó vào xe. Ở đây, chiếc xe phụ thuộc vào động cơ, nhưng thay vì tạo ra nó (động cơ) một cách cứng nhắc bên trong chiếc xe, bạn "tiêm" nó từ bên ngoài. Điều này giúp bạn có thể dễ dàng thay đổi loại động cơ mà không cần thay đổi cấu trúc của chiếc xe.

Trong ví dụ này, EmailClient tạo ra EmailService trực tiếp bên trong constructor của nó. Điều này làm cho EmailClient cứng nhắc và khó kiểm thử vì nó luôn phụ thuộc vào EmailService cụ thể.

```
class EmailService {
    // ...
}

class EmailClient {
    private $service;

    public function __construct() {
        // Tạo EmailService một cách cứng nhắc
        $this->service = new EmailService();
    }

    // ...
}
```

KHÔNG SỬ DỤNG DI

1. Tiêm phụ thuộc (Dependency Injection)

```php
interface EmailServiceInterface {
    public function sendEmail($to, $subject, $message);
}
```

```php
class GoogleService implements EmailServiceInterface {
    public function sendEmail($to, $subject, $message) {
        // Gửi email sử dụng Google's API
    }
}

class MicrosoftService implements EmailServiceInterface {
    public function sendEmail($to, $subject, $message) {
        // Gửi email sử dụng Microsoft's API
    }
}
```

```php
class EmailClient {
    private $service;

    public function __construct(EmailServiceInterface $service) {
        $this->service = $service;
    }

    public function sendEmail($to, $subject, $message) {
        $this->service->sendEmail($to, $subject, $message);
    }
}
```

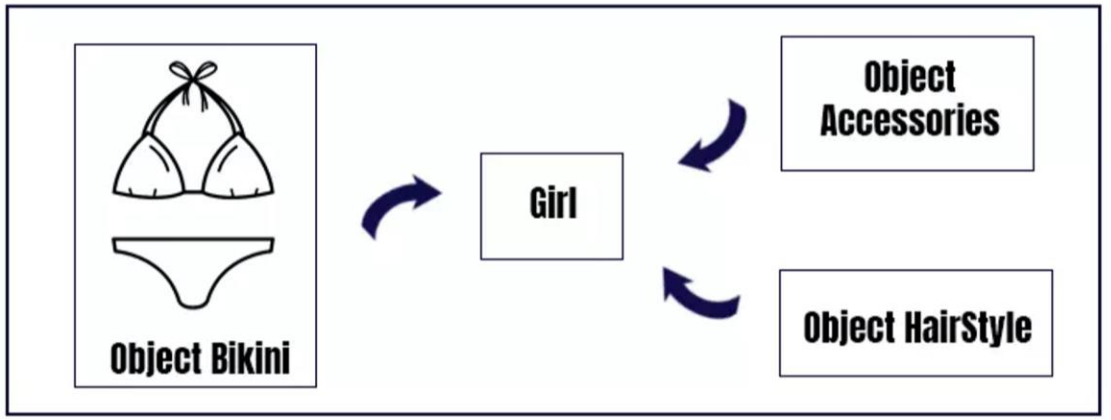
```php
// Sử dụng GoogleService
$googleService = new GoogleService();
$clientWithGoogle = new EmailClient($googleService);
$clientWithGoogle->sendEmail("example@example.com", "Subject",
"Message");

// Sử dụng MicrosoftService
$microsoftService = new MicrosoftService();
$clientWithMicrosoft = new EmailClient($microsoftService);
$clientWithMicrosoft->sendEmail("example@example.com", "Subject",
"Message");
```

LỢI ÍCH CỦA CÁCH TIẾP CẬN NÀY

- **Linh hoạt**: EmailClient không cần biết dịch vụ cụ thể nào được sử dụng. Nó chỉ biết rằng nó có thể gửi email. Điều này làm cho nó trở nên linh hoạt hơn và dễ dàng tích hợp với các dịch vụ khác.
- **Dễ dàng mở rộng**: Bạn có thể dễ dàng thêm các dịch vụ email mới mà không cần sửa đổi EmailClient.
- **Dễ dàng kiểm thử**: Bạn có thể sử dụng mock objects hoặc stubs để kiểm thử EmailClient mà không cần phụ thuộc vào các dịch vụ bên ngoài thực tế.

1. Tiêm phụ thuộc (Dependency Injection)

- Chúng ta định nghĩa trước toàn bộ các dependency có trong Project, mô tả nó và tống nó vào 1 cái kho và giao cho một thằng tên là framework quản lý. Bất kỳ các Class nào khi khởi tạo, nó cần dependency gì, thì cái framework này sẽ tự tìm trong kho rồi inject vào đối tượng thay chúng ta. sẽ tiện hơn phải không?



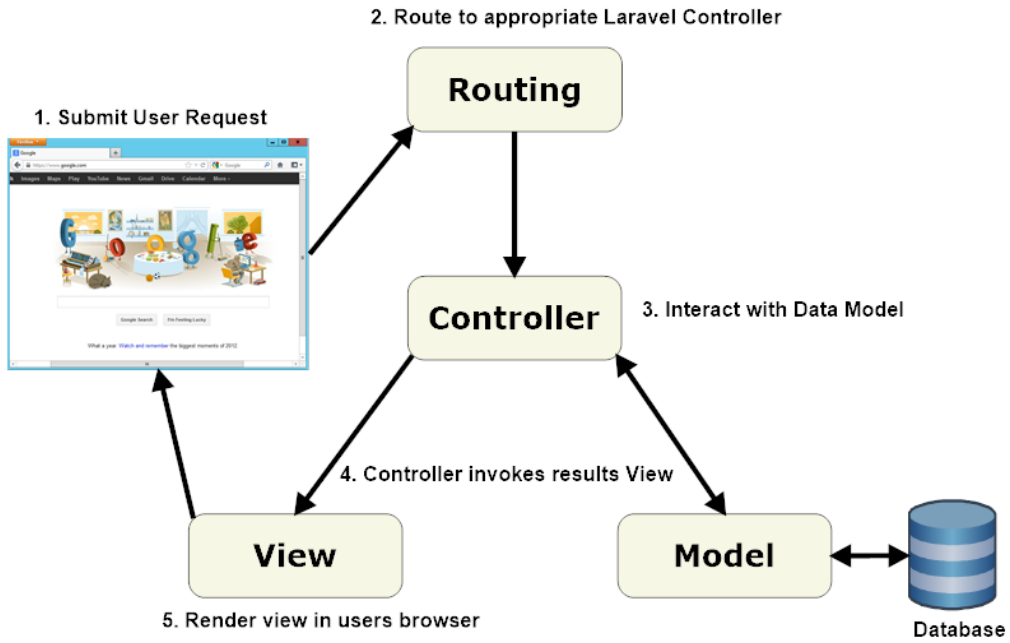
Framwork Container

get(Girl.class)

Nguồn tham khảo DI: https://loda.me/courses/spring-boot

2. Mô hình MVC

- Mô hình Model-View-Controller (MVC) là một mô hình kiến trúc phần mềm phân tách một ứng dụng thành ba thành phần logic chính:
  - Model: Chứa dữ liệu và logic liên quan của nó.
  - View: Chịu trách nhiệm hiển thị dữ liệu cho người dùng.
  - Controller: Nhận yêu cầu từ người dùng và xử lý chúng, sau đó chuyển dữ liệu cho Model hoặc View để xử lý.
- Lợi ích của mô hình MVC:
  - Tăng tính bảo trì và mở rộng
  - Tăng khả năng kiểm tra
  - Tăng khả năng tái sử dụng



2. Mô hình MVC

- CSDL và Mô hình MVC mẫu:

```
CREATE TABLE articles (
    id INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(255) NOT NULL,
    content TEXT NOT NULL,
    category_id INT,
    author_id INT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (category_id) REFERENCES categories(id),
    FOREIGN KEY (author_id) REFERENCES members(id)
);
```

```
CREATE TABLE categories (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255) NOT NULL
);
```

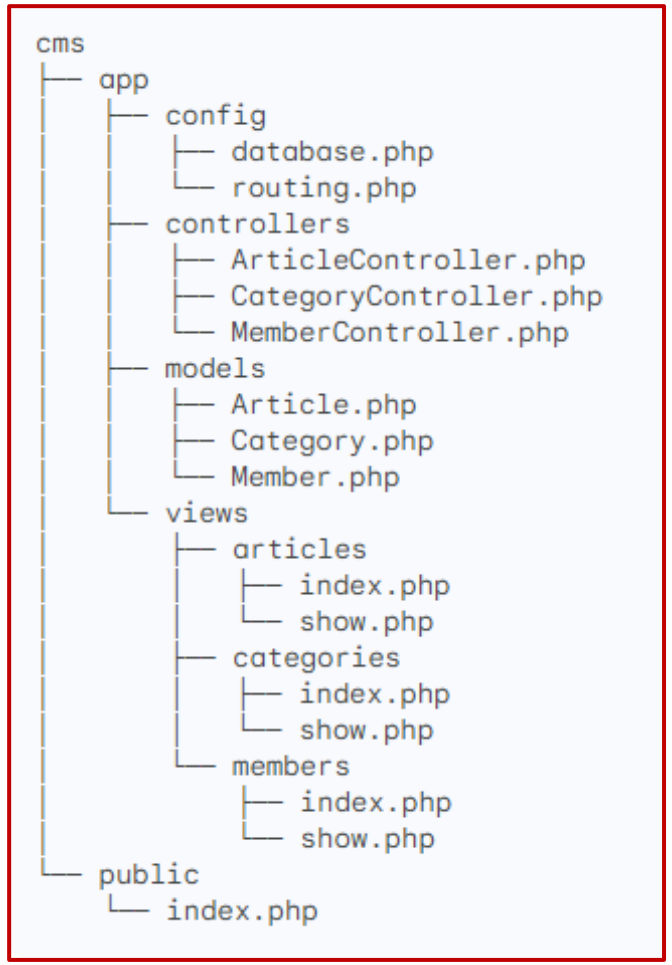
```
CREATE TABLE members (
    id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(255) NOT NULL,
    password VARCHAR(255) NOT NULL,
    email VARCHAR(255) NOT NULL
);
```

```
/your_project
    /models
        - ArticleModel.php
        - CategoryModel.php
        - MemberModel.php
        - Database.php
    /views
        /articles
            - index.php
            - create.php
            - edit.php
            - show.php
        /categories
            - index.php
            ...
        /members
            - index.php
            ...
        - home.php
    /controllers
        - ArticlesController.php
        - CategoryController.php
        - MemberController.php
    - index.php
    - .htaccess
```

2. Mô hình MVC

- CSDL và Mô hình MVC mẫu:

```
CREATE TABLE articles (
    id INT NOT NULL AUTO_INCREMENT,
    title VARCHAR(255) NOT NULL,
    content TEXT NOT NULL,
    category_id INT NOT NULL,
    PRIMARY KEY (id)
);

CREATE TABLE categories (
    id INT NOT NULL AUTO_INCREMENT,
    name VARCHAR(255) NOT NULL,
    PRIMARY KEY (id)
);

ALTER TABLE articles
ADD CONSTRAINT fk_articles_categories
FOREIGN KEY (category_id) REFERENCES categories (id);

CREATE TABLE members (
    id INT NOT NULL AUTO_INCREMENT,
    username VARCHAR(255) NOT NULL,
    password VARCHAR(255) NOT NULL,
    PRIMARY KEY (id)
);
```



2. Mô hình MVC

- Mô hình MVC mẫu:

  - Trong ví dụ này, index.php đóng vai trò là routing, phụ trách việc điều hướng yêu cầu đến controller phù hợp. Điều này đảm bảo rằng tất cả logic xử lý yêu cầu được tập trung vào một nơi.
  - .htaccess: Được sử dụng để cấu hình sử dụng index.php làm điểm định tuyến cho tất cả các yêu cầu:
  - Mỗi thành phần trong MVC (Model, View, và Controller) có nhiệm vụ cụ thể: Model xử lý dữ liệu, View hiển thị dữ liệu, và Controller làm cầu nối giữa Model và View.

```
cms
├── app
│   ├── config
│   │   ├── database.php
│   │   └── routing.php
│   ├── controllers
│   │   ├── ArticleController.php
│   │   ├── CategoryController.php
│   │   └── MemberController.php
│   ├── models
│   │   ├── Article.php
│   │   ├── Category.php
│   │   └── Member.php
│   └── views
│       ├── articles
│       │   ├── index.php
│       │   └── show.php
│       ├── categories
│       │   ├── index.php
│       │   └── show.php
│       └── members
│           ├── index.php
│           └── show.php
└── public
    └── index.php
```

Trong mỗi thư mục con của views, chẳng hạn views/articles, bạn có thể có các file view cụ thể cho các chức năng khác nhau của Article:

- index.php: Hiển thị danh sách các bài viết.
- create.php: Hiển thị form tạo bài viết mới.
- edit.php: Hiển thị form chỉnh sửa bài viết.
- show.php: Hiển thị chi tiết một bài viết.

2. Mô hình MVC

- Mô hình MVC mẫu:

Ví dụ về **.htaccess**

```
RewriteEngine On
RewriteBase /your_project/public/
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule ^ index.php [QSA,L]
```

```
cms
├── app
│   ├── config
│   │   ├── database.php
│   │   └── routing.php
│   ├── controllers
│   │   ├── ArticleController.php
│   │   ├── CategoryController.php
│   │   └── MemberController.php
│   ├── models
│   │   ├── Article.php
│   │   ├── Category.php
│   │   └── Member.php
│   └── views
│       ├── articles
│       │   ├── index.php
│       │   └── show.php
│       ├── categories
│       │   ├── index.php
│       │   └── show.php
│       └── members
│           ├── index.php
│           └── show.php
└── public
    └── index.php
```

2. Mô hình MVC

- Mô hình MVC mẫu: index.php

```php
require '../app/config/database.php';

// Lấy thông tin về controller và action từ URL
$controller = isset($_GET['controller']) ? ucfirst($_GET['controller'])
: 'Home';
$action = isset($_GET['action']) ? $_GET['action'] : 'index';

// Tạo đường dẫn đến file controller dựa trên thông tin từ URL
$controllerPath = "../app/controllers/{$controller}Controller.php";

// Kiểm tra và include controller
if (file_exists($controllerPath)) {
    require $controllerPath;
    $controllerClass = $controller . 'Controller';
    $controllerObject = new $controllerClass();

    // Gọi action
    if (method_exists($controllerObject, $action)) {
        $controllerObject->$action();
    } else {
        // Action không tồn tại
        echo "404 Not Found: The action does not exist.";
    }
} else {
    // Controller không tồn tại
    echo "404 Not Found: The controller does not exist.";
}
```

2. Mô hình MVC

- Mô hình MVC mẫu:

```php
<?php

class DatabaseConfig {
    const HOST = 'localhost';
    const DB_NAME = 'your_database_name';
    const USERNAME = 'your_database_username';
    const PASSWORD = 'your_database_password';

    public static function getConnection() {
        try {
            $dsn = 'mysql:host=' . self::HOST . ';dbname=' .
self::DB_NAME;
            $connection = new PDO($dsn, self::USERNAME,
self::PASSWORD);
            $connection->setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);
            return $connection;
        } catch (PDOException $e) {
            echo "Kết nối cơ sở dữ liệu thất bại: " . $e->getMessage();
            exit;
        }
    }
}
```

```php
<?php

// Nạp file cấu hình cơ sở dữ liệu
require '../app/config/database.php';

// Kết nối đến cơ sở dữ liệu
$dbConnection = DatabaseConfig::getConnection();

// Tiếp tục với các bước khác của routing và MVC...
```

“Câu hỏi & Thảo luận”

THE END!