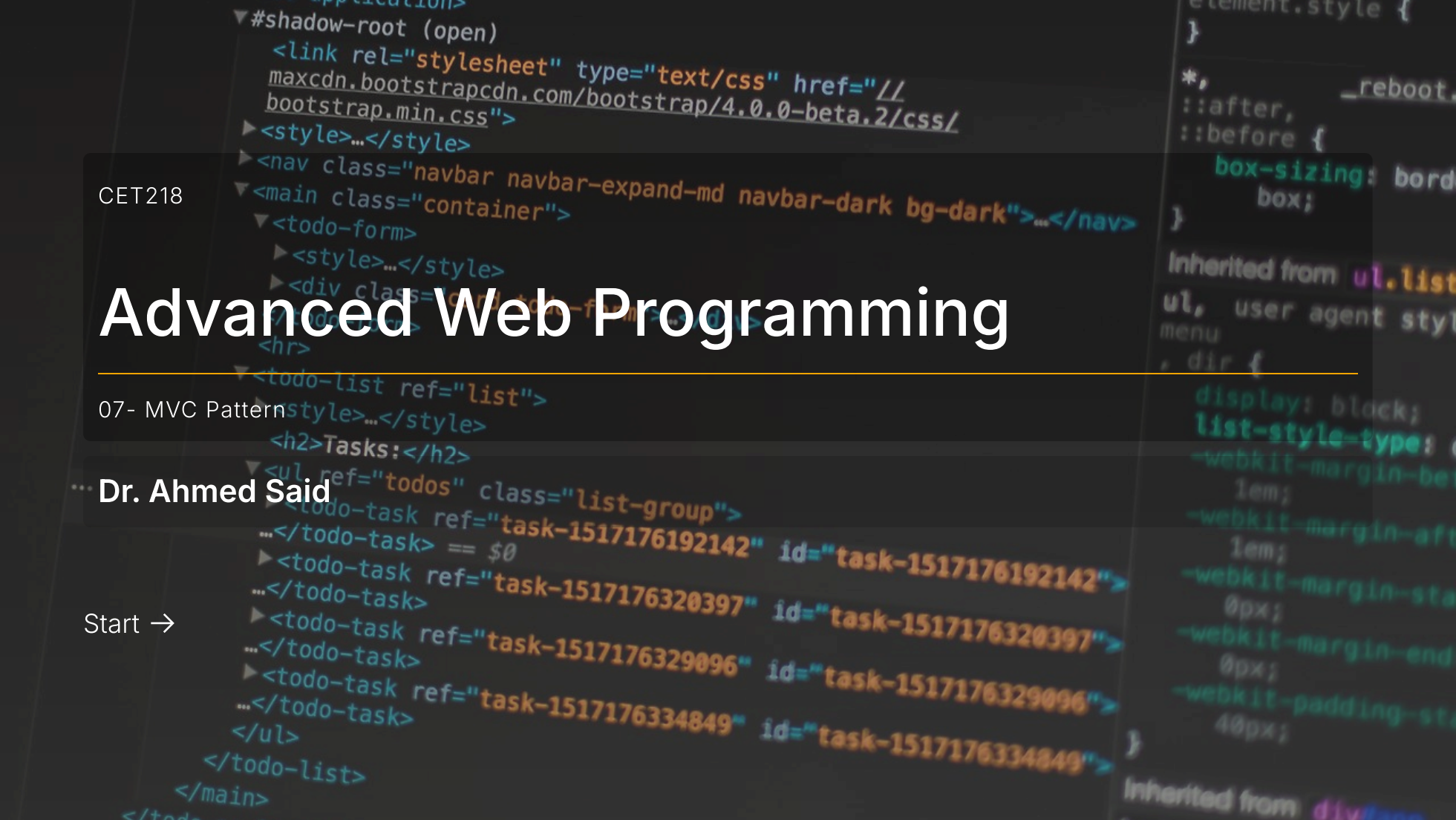
CET218

Advanced Web Programming

07- MVC Pattern

Dr. Ahmed Said

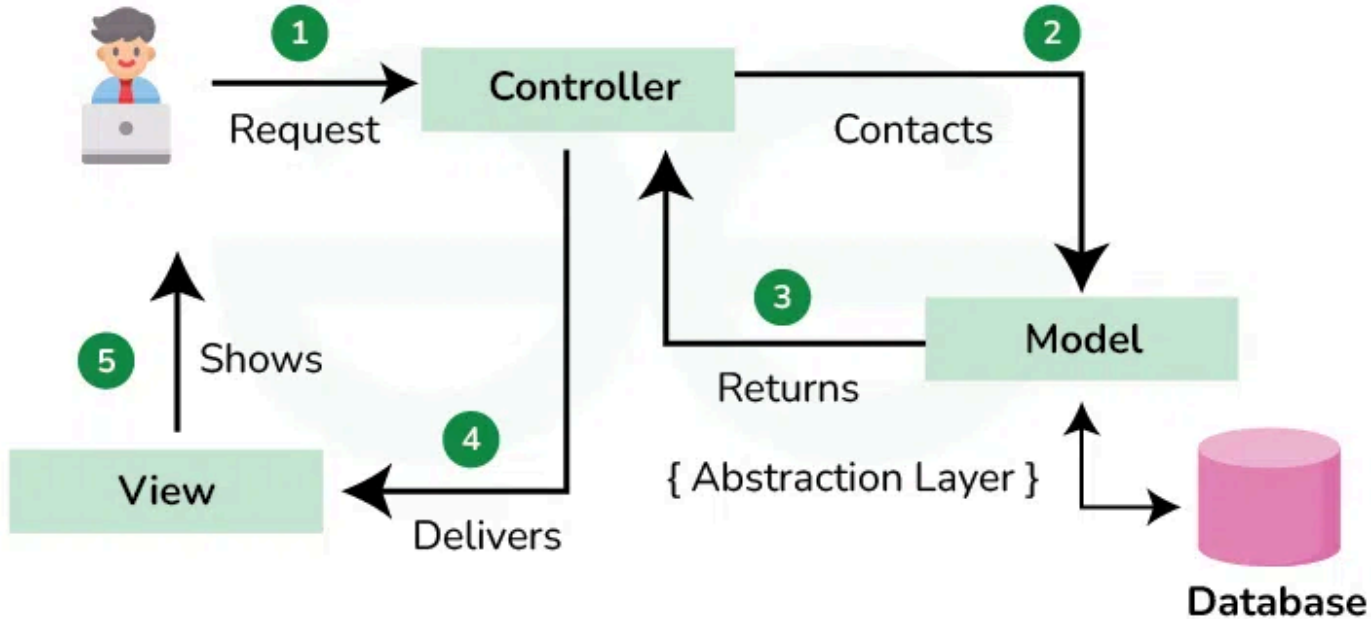
Start →



MVC Pattern

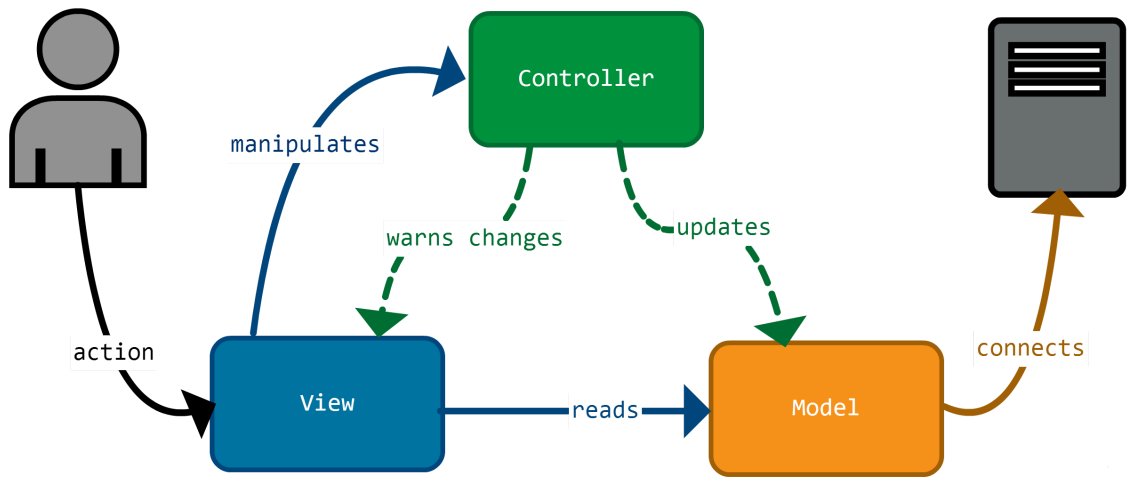
What is MVC?

- The **MVC** architecture pattern turns complex application development into a much more manageable process. It allows several developers to simultaneously work on the application.
- **MVC** stands for model-view-controller. Here's what each of those components mean:
  - Model: The backend that manages data and business logicc
  - View: The frontend or graphical user interface (GUI) "Displays data to the user"
  - Controller: The brains of the application that controls how data is displayed "Handles user input and coordinates Model and View."


1
Request
Controller
2
Contacts
3
Returns
Model
{ Abstraction Layer }
Database
4
Delivers
View
5
Shows

MVC Pattern: Overview

- The concept of MVCs was first introduced by **Trygve Reenskaug**, who proposed it as a way to develop desktop application GUIs.
- Today the MVC pattern is used for modern web applications because it allows the application to be scalable, maintainable, and easy to expand.



Why should you use MVC?

- **Separation of Concerns**: Each component has a distinct responsibility, making it easier to manage and maintain.
- **Collaboration**: Multiple developers can work on different components simultaneously, speeding up development.
- **Flexibility**: You can change the View without affecting the Model or Controller, allowing for easier updates and redesigns.
- **Reusability**: Components can be reused across different parts of the application or even in different projects.
- **Maintainability**: Code is easier to read and understand, making it simpler to fix bugs and add new features.

MVC: Model

- The **Model** represents the data layer, often interacting with a database.

```php
class UserModel {
    private $name;
    public function __construct($name) {
        $this->name = $name;
    }
    public function getName() {
        return $this->name;
    }
    public function save() {
        // Simulate database save
        echo "Saving $this->name to database.";
    }
}
```

Note: Models encapsulate data operations, keeping logic centralized.

MVC: View

- The **View** renders the data for the user, typically as HTML.

```php
// user_view.php
<h1>User Profile</h1>
<p>Name: <?php echo $user->getName(); ?></p>
```

> **Tip**: Keep Views simple and focused on presentation, avoiding business logic.

MVC: Controller

- The **Controller** processes user input and updates the Model and View.

```php
class UserController {
    private $model;
    public function __construct() {
        $this->model = new UserModel("Alice");
    }
    public function showProfile() {
        include 'user_view.php';
    }
    public function updateName($newName) {
        $this->model = new UserModel($newName);
        $this->model->save();
    }
}

$controller = new UserController();
$controller->showProfile(); // Displays profile
$controller->updateName("Bob"); // Updates and saves
```

> **Note**: Controllers act as the glue between Model and View.

MVC: Putting It Together

- A simple workflow:
  1. User requests a profile (Controller).
  2. Controller fetches data via Model.
  3. Controller passes data to View for display.

```php
$controller = new UserController();
$controller->showProfile(); // Outputs: <h1>User Profile</h1><p>Name: Alice</p>
```

Tip: In real applications, use frameworks like Laravel for robust MVC implementations.

PHP MVC Frameworks

- **Laravel**: A popular PHP framework that follows the MVC pattern, offering a clean and elegant syntax.
- **Symfony**: A robust PHP framework that provides reusable components and a strong MVC architecture.
- **CodeIgniter**: A lightweight PHP framework that is easy to set up and follows the MVC pattern.
- **CakePHP**: A rapid development framework that uses MVC and provides code generation features.
- **Yii**: A high-performance PHP framework that follows the MVC pattern and is suitable for large-scale applications.

The Laravel Framework

Your Elegant PHP Framework for Web Artisans

Today's Agenda

1. **What is Laravel?** - Why it's popular.
2. **Installation & Setup** - Getting started.
3. **Configuration** - The `.env` file.
4. **Directory Structure** - Navigating your project.
5. **MVC Pattern** - The core architecture.
6. **Routing** - Handling requests (Web & API).
7. **Controllers** - Logic handlers.
8. **Creating Your First Routes & Controllers** - Practical examples.

1. What is Laravel?

- A **free, open-source PHP web framework** created by Taylor Otwell.
- Follows the **Model-View-Controller (MVC)** architectural pattern.
- Designed for developing web applications with **expressive, elegant syntax**.
- Aims to make development **enjoyable** and **creative** by easing common tasks (routing, authentication, sessions, caching, etc.).

Why Choose Laravel?

- **Developer Experience:** Elegant syntax, helpful tools (Artisan, Tinker).
- **Large Community:** Extensive documentation, tutorials, and packages (Packagist).
- **Robust Features:** Built-in features for common web development tasks.
- **Scalability:** Suitable for small projects to large enterprise applications.
- **Security:** Protects against common web vulnerabilities (SQL injection, XSS).
- **Blade Templating Engine:** Simple yet powerful templating.
- **Eloquent ORM:** Makes database interactions intuitive.

2. Installation & Setup

Prerequisites

- **PHP:** >= 8.1 (Check Laravel version requirements)
- **Composer:** PHP dependency manager. https://getcomposer.org/
- Database (MySQL, PostgreSQL, SQLite, SQL Server)
- Web Server (Nginx, Apache - often handled by dev tools)

Laravel Versions

Installing Laravel

Use Composer to create a new Laravel project:

```bash
# Via Laravel Installer (Recommended first time)
composer global require laravel/installer
laravel new my-awesome-app

# OR Via Composer Create-Project
composer create-project --prefer-dist laravel/laravel my-awesome-app "10.*" # Specify version if needed

cd my-awesome-app
```

- The first command installs the Laravel installer globally on your system:
- After installation, you can create a new Laravel project by running: `laravel new my-awesome-app`
  - This method is recommended for first-time users as it simplifies the process.
- Or Using Composer's create-project Command:
  - This method directly creates a new Laravel project without requiring the Laravel installer: composer create-project --prefer-dist laravel/laravel my-awesome-app "10.*"
  - The "10.*" specifies the Laravel version to install (e.g., version 10.x).

Running the Development Server

Laravel comes with a simple development server using Artisan:

```bash
php artisan serve
```

Your application will typically be available at `http://127.0.0.1:8000`.

3. Configuration

The `.env` File

- Located in the project root.
- Stores **environment-specific** configuration (database credentials, API keys, app settings).
- **Crucial:** `.env` should **NOT** be committed to version control (security risk!).
- Laravel uses the `DotEnv` library to load these variables.
- `.env.example` provides a template. Copy it to `.env` : `cp .env.example .env`

Example `.env` Snippet

```dotenv
APP_NAME=Laravel # Your App Name
APP_ENV=local # Environment (local, production, testing)
APP_KEY= # Will be generated
APP_DEBUG=true # Enable debug mode (NEVER true in production)
APP_URL=http://localhost # Your app's base URL

LOG_CHANNEL=stack
LOG_LEVEL=debug

DB_CONNECTION=mysql # Database type
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=laravel # Your database name
DB_USERNAME=root # Your database username
DB_PASSWORD= # Your database password

# Other settings like Mail, Cache, Queue drivers...
```

Application Key

- A unique, random string used for encryption.
- Generate it after creating `.env` :

```bash
php artisan key:generate
```

Note This command updates the `APP_KEY` in your `.env` file.

4. Directory Structure

A brief overview of key directories:

- `app/`
  - Core application code.
  - `Http/Controllers` : Handles requests.
  - `Models` : Represents database tables.
  - `Providers` : Service container bootstrapping.
  - (And Console, Exceptions, etc.)
- `bootstrap/`
  - App bootstrapping scripts, cache files.

4. Directory Structure, cont.

A brief overview of key directories:

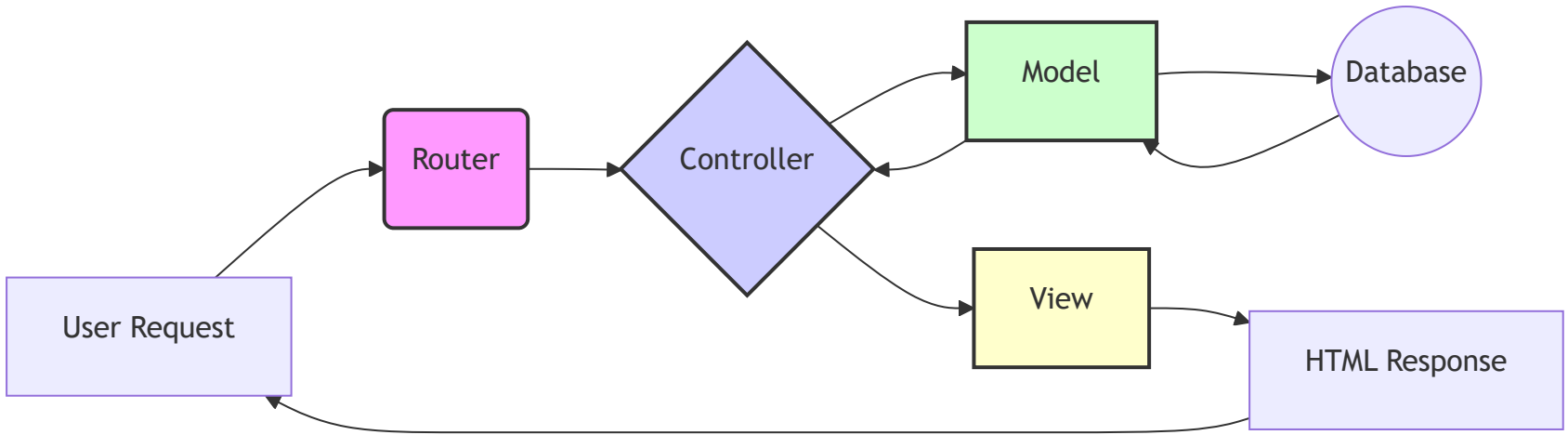
- `config/`
  - Application configuration files (database, cache, mail, etc.). Values often sourced from `.env`.
- `database/`
  - `factories`: Define model factories for testing/seeding.
  - `migrations`: Database schema version control.
  - `seeders`: Seed database with initial data.
- `public/`
  - **Web server document root.**
  - `index.php`: Entry point for all requests.
  - Assets (CSS, JS, images - often managed by Vite/Mix).

4. Directory Structure, cont.

- `resources/`
  - `css` , `js` : Raw frontend assets (processed by Vite/Mix).
  - `lang` : Language files for localization.
  - `views` : Application views (Blade templates).
- `routes/`
  - `web.php` : Routes for web interface (session state, CSRF protection).
  - `api.php` : Routes for stateless APIs (token authentication).
  - `console.php` : Artisan console commands.
  - `channels.php` : Broadcasting channels.
- `storage/` : Compiled Blade templates, file caches, logs, user-uploaded files.
- `tests/` : Application tests (Unit, Feature).
- `vendor/` : Composer dependencies (Framework core, packages). **Do not edit!**

5. The MVC Pattern

Model - View - Controller is a fundamental architectural pattern for separating concerns in web applications.



5. The MVC Pattern, Cont.

- **Model:**
  - Represents the application's data and business logic.
  - Interacts directly with the database (often via Eloquent ORM).
  - Example: `app/Models/User.php`
- **View:**
  - Responsible for presenting data to the user (UI).
  - Typically HTML generated by Blade templates.
  - Example: `resources/views/users/index.blade.php`

5. The MVC Pattern, Cont.

- **Controller:**
  - Acts as an intermediary between Model and View.
  - Handles incoming user requests (via Routing).
  - Fetches data from the Model.
  - Passes data to the View for rendering.
  - Example: `app/Http/Controllers/UserController.php`

Thank You!