Unit 8: PHP Framework

Presented By:
Bipin Maharjan

Learning Objective

- Understand the concepts of models, views, and controllers, and understand how they interact within the MVC architecture to handle user requests and manage application data.
- Learn about the benefits of using a framework for web development, such as code reusability, modularity, and rapid development.
- Explore one of the PHP frameworks available, such as Laravel, Symfony, CodeIgniter.
- Learn about common features provided by PHP frameworks, such as routing, templating, authentication, authorization, and ORM (Object-Relational Mapping).

Table of Contents

MVC Model

Introduction to MVC

MVC (Model-View-Controller) is a design pattern used in modern web development frameworks to separate application logic into three interconnected components:

- **Model** – Manages the data and business logic.
- **View** – Manages the presentation and user interface.
- **Controller** – Handles user input, updates the model, and renders the appropriate view.

This separation of concerns makes the code organized, reusable, and easier to maintain.

Real-World Analogy

Think of MVC as a restaurant system:

MVC Component	Analogy	Description
Model	Kitchen	Prepares the food (data and logic)
View	Waiter’s tray / Plate	Presents the food (data) to the customer
Controller	Waiter	Takes orders (user input) and coordinates between kitchen and customer

Why MVC Architecture?

Traditionally, PHP scripts used to mix HTML, database queries, and logic in a single file, making it hard to debug and scale.

MVC solves this by dividing the code into three logical parts.

Benefits:

- Separation of logic, presentation, and data
- Easier debugging and testing
- Promotes code reusability
- Enables multiple developers to work in parallel (Model, View, and Controller independently)
- Easier maintenance and scalability

Model Component

The Model represents the data and business logic of the application.

It directly interacts with the database and contains rules that define how data can be created, retrieved, updated, or deleted.

Responsibilities of the Model:

- Connect to the database
- Define data structures and relationships
- Implement business logic (e.g., validation rules, calculations)
- Return data to the controller

Model Example

```
// app/Models/Student.php

namespace App\Models;


use Illuminate\Database\Eloquent\Model;


class Student extends Model {

    protected $table = 'students';

    protected $fillable = ['name', 'email', 'course'];

}
```

Here, the Student model represents the students table in the database.

We can use it to perform CRUD operations:

```
$students = Student::all(); // Retrieve all students

Student::create(['name' => 'John', 'email' => 'john@email.com', 'course' => 'PHP']);
```

View Component

The View is responsible for the presentation layer — it defines how data is displayed to users.

Views are typically HTML templates combined with PHP or template engines like Blade (Laravel) or Twig (Symfony).

Responsibilities of the View:

- Display data passed from the controller
- Contain minimal logic (only presentation-related)
- Provide a user interface for interaction

View Example

```
<!-- resources/views/students/index.blade.php -->
<!DOCTYPE html>
<html>
<head>
    <title>Student List</title>
</head>
<body>
<h2>Student List</h2>
<ul>
    @foreach($students as $student)
        <li>{{ $student->name }} - {{ $student->course }}</li>
    @endforeach
</ul>
</body>
</html>
```

Here, the View simply presents data provided by the Controller in a readable format.

Controller Component

The Controller acts as the middleman between the Model and the View.

It receives the user request, interacts with the Model to retrieve data, and selects which View to display.

Responsibilities of the Controller:

- Receive and process user input
- Call Model methods to fetch or update data
- Pass data to the View for rendering
- Control application flow

Controller Example

```
// app/Http/Controllers/StudentController.php
namespace App\Http\Controllers;

use App\Models\Student;

class StudentController extends Controller {
    public function index() {
        $students = Student::all();  // Fetch data from Model
        return view('students.index', compact('students')); // Pass data to View
    }
}
```

Controller Example Contd..

When a user visits /students, the index() method runs:

- Fetches all students using the Model
- Passes the data to the View
- Returns rendered HTML to the browser

Flow of Control in MVC Architecture

1. **User Request:** The user types a URL or submits a form — e.g., http://example.com/students.
2. **Routing:** The framework routes this URL to the correct Controller action:

```
Route::get('/students', [StudentController::class, 'index']);
```

3. **Controller Processes Request:** The StudentController@index method executes.
4. **Model Interaction:** The controller requests data from the Student model.
5. **View Rendering:** The controller sends the retrieved data to the students.index view.
6. **Response to Browser:** The rendered HTML is sent back to the user's browser.

Advantages of MVC

- **Separation of Concerns:** Logic, data, and presentation are isolated.
- **Reusability:** Models and Views can be reused across different parts of the app.
- **Parallel Development:** Multiple developers can work simultaneously.
- **Maintainability:** Easier to update or modify individual parts.
- **Testability:** Components can be tested independently.

Benefits of Using the PHP Framework

PHP Framework

A PHP Framework is a collection of pre-built libraries, tools, and conventions that provide a structured way to develop PHP web applications.

Frameworks like Laravel, Symfony, and CodeIgniter are built upon the MVC architecture, promoting clean, organized, and maintainable code.

Instead of writing everything from scratch, developers can leverage reusable components, resulting in faster development, fewer bugs, and more secure applications.

Code Reusability and Modularity

Frameworks encourage writing modular and reusable code.

You can organize your application into modules or classes, making it easier to maintain or reuse parts of your project in other applications.

Example:

Instead of writing your own database connection code every time, Laravel provides:

```
$users = DB::table('users')->get();
```

This uses the built-in Database library, which can be reused across your entire project.

Contd…

Benefits:

- No repetitive coding
- Easier updates — fix one module and reuse it
- Faster team collaboration

Rapid Application Development (RAD)

Frameworks come with pre-built features (like routing, authentication, form validation, etc.), which greatly reduce development time.

Feature	Laravel	Symfony	CodeIgniter
Routing	✅ Yes	✅ Yes	✅ Yes
Authentication	✅ Built-in	✅ Bundle	✅ Manual
ORM	✅ Eloquent	✅ Doctrine	✅ Active Record
Templating	✅ Blade	✅ Twig	✅ View Parser

Result:

Developers can focus on the application logic, not repetitive tasks like handling sessions or connecting to the database.

Contd…

Example:

A simple route in Laravel:

```
Route::get('/home', function () {
    return view('home');
});
```

Without a framework, you'd manually handle URLs, parameters, and view rendering.

Maintainability and Scalability

PHP frameworks enforce a structured file organization, separating logic, data, and presentation using the MVC pattern.

This makes large projects easier to manage and extend over time.

Example:

In Laravel:

- All controllers go in app/Http/Controllers
- All models in app/Models
- All views in resources/views

This consistent structure helps teams work on different parts of the application without conflicts.

Contd…

Benefits:

- Easier debugging (locate problems quickly)
- Easier to add new features
- Smooth scaling as the project grows

Security and Best Practices

Frameworks include built-in protection mechanisms against common web vulnerabilities such as:

Threat	Description	Framework Protection
SQL Injection	Inserting malicious SQL into queries	ORM & Query Builder
CSRF	Cross-Site Request Forgery	CSRF tokens in forms
XSS	Cross-Site Scripting	Auto-escaped outputs
Password Hashing	Protecting stored passwords	Built-in hashing libraries

Contd…

Example (Laravel CSRF Protection):

```
<form method="POST" action="/profile">
    @csrf
    <button type="submit">Update</button>
</form>
```

Laravel automatically adds a token that prevents CSRF attacks.

Result:

Your applications become more secure without requiring manual implementation of these security features.

Community Support and Ecosystem

Popular frameworks have large communities, providing:

- Extensive documentation
- Open-source packages and extensions
- Frequent updates and bug fixes

Example:

Laravel’s Packagist offers thousands of community-built packages for:

- Payment gateways
- Email sending
- PDF generation
- File uploads

Benefit: You don’t need to “reinvent the wheel” — just install a package.

Consistency and Standardization

Frameworks promote coding standards and best practices, ensuring consistency across projects.

This makes it easier for new developers to understand existing code.

Example:

Laravel follows PSR (PHP Standards Recommendations) — a set of rules for file naming, autoloading, and structure.

Result:

Team projects stay consistent, readable, and collaborative.

Built-in Tools for Common Tasks

Frameworks often come with ready-to-use developer tools, such as:

Tool	Purpose
Artisan CLI (Laravel)	Run commands like migrations, seeding, etc.
Debugging tools	Show detailed error messages
Migration system	Manage database versions
Testing utilities	Perform unit and feature testing

Object-Relational Mapping (ORM)

Frameworks like Laravel (Eloquent) and Symfony (Doctrine) provide an ORM to simplify database operations.

Instead of raw SQL:

SELECT * FROM users WHERE id = 1;

You can write:

$user = User::find(1);

Contd…

Benefits:

- Easier to read and maintain code
- Prevents SQL injection
- Works across multiple database types (MySQL, PostgreSQL, etc.)

Routing Made Easy

- Routing in frameworks allows developers to define how URLs map to controllers and actions.

Example:

```
// routes/web.php

Route::get('/about', [PageController::class, 'about']);
```

- Without a framework, you'd have to manually parse the request URL and decide which script to run.
- Routing provides a clean, readable, and centralized way to manage all your application paths.

Templating System

Most frameworks include templating engines that make HTML rendering dynamic and clean.

Example (Blade in Laravel):

<h1>Welcome, {{ $user->name }}</h1>

Blade automatically escapes output to prevent XSS attacks and allows layout inheritance.

Benefit: Keeps frontend code organized, reusable, and secure.

Getting Started with PHP Framework

Introduction to Popular PHP Frameworks

Framework	Key Features	Use Case
Laravel	Elegant syntax, ORM (Eloquent), Routing, Blade templating	Large-scale web applications
Symfony	Reusable components, robust architecture	Enterprise-level applications
CodeIgniter	Lightweight, simple to learn	Small to medium projects

Start with Laravel, as it’s beginner-friendly, widely used, and rich in features.

Setting Up Your Development Environment

Before installing any PHP framework, ensure the following tools are installed:

Check installation:

php -v

composer -V

Tool	Description
PHP (≥ 8.1)	Core scripting language
Composer	PHP dependency manager
Web Server	Apache or Nginx
Database	MySQL or PostgreSQL
Text Editor/IDE	VS Code, PhpStorm, Sublime Text

Installing Laravel Framework

Laravel Documentation: https://laravel.com/docs/12.x/installation

Step 1: Install Composer

Composer is a package manager for PHP that handles dependencies.

Download & Install:

- Visit https://getcomposer.org
- Follow installation instructions for your OS.

Contd…

Step 2: Create a New Laravel Project

Use the command:

```
composer create-project laravel/laravel myapp
```

Step 3: Start the Development Server

Navigate to your project directory and start the built-in Laravel server:

```
cd myapp

php artisan serve
```

Understanding Laravel Directory Structure

After installation, your Laravel project looks like this:

```
myapp/
├── app/
│   ├── Http/
│   │   └── Controllers/
│   └── Models/
├── bootstrap/
├── config/
├── database/
├── public/
│   └── index.php
├── resources/
│   └── views/
├── routes/
│   └── web.php
```

Key Folders

Folder	Purpose
app/	Contains Models, Controllers, and core classes
resources/views/	Stores all View templates
routes/web.php	Defines web routes
database/	Migration and seed files
public/	Front controller (index.php) – entry point of app

Creating a Simple MVC Example in Laravel

Step 1: Define a Route

Open routes/web.php:

```
use App\Http\Controllers\WelcomeController;

Route::get('/welcome', [WelcomeController::class, 'index']);
```

Step 2: Create a Controller

Run the Artisan command:

```
php artisan make:controller WelcomeController
```

Now edit the file app/Http/Controllers/WelcomeController.php:

```
namespace App\Http\Controllers;

class WelcomeController extends Controller {
    public function index() {
        $name = "Students of Internet Technology";
        return view('welcome', compact('name'));
    }
}
```

Step 3: Create a View

Create a new file in resources/views/welcome.blade.php:

```
<!DOCTYPE html>
<html>
<head>
    <title>Welcome Page</title>
</head>
<body>
    <h1>Hello, {{ $name }}!</h1>
    <p>Welcome to your first Laravel application.</p>
</body>
</html>
```

Step 4: Test It

Run the server (only if server is not running):

```
php artisan serve
```

Visit http://127.0.0.1:8000/welcome

You'll see the message rendered by your controller and view.

Common Features of PHP Frameworks

Feature	Description	Example
Routing	Maps URLs to controllers	`/students → StudentController@index`
ORM (Object Relational Mapping)	Interacts with the database using objects	`$user = User::find(1)`
Templating Engine	Simplifies HTML generation	Blade (Laravel), Twig (Symfony)
Authentication	Handles login, registration, sessions	`php artisan make:auth` (Laravel)
Authorization	Restricts access based on roles	Middleware-based
Migration System	Manages database structure	`php artisan migrate`
Validation	Validates form input	`$request->validate([...])`
Testing Tools	Built-in unit testing support	PHPUnit
Artisan CLI	Command-line tool to automate tasks	`php artisan make:model`

Any Questions?

Learn Laravel in Detail

Series: https://laracasts.com/series/30-days-to-learn-laravel-11