Laravel

This 3-day intermediate Laravel program helps working Laravel developers strengthen architecture, performance, API quality, security, testing, and production readiness through practical best-practice implementation.

Overview

This 3-day intermediate Laravel program is designed for developers who already have working knowledge of Laravel and want to elevate their skills to professional standards. The course focuses on clean code practices, modular architecture for large systems, performance optimization, security hardening, and UI/UX best practices. Through hands-on exercises and real-world scenarios, participants will learn how to build scalable, maintainable, and production-ready Laravel applications.

Introduction

This program emphasizes practical implementation of industry's best practices in Laravel development. Participants will refactor existing code, design modular systems, optimize performance, implement secure APIs, and improve user experience. The goal is to transform working developers into professional-level engineers capable of building high-quality systems.

Topics Covered

- Clean Code Principles in Laravel
- Service Layer and Repository Pattern
- Modular Architecture (Domain-based / Feature-based)
- Advanced Eloquent and Query Optimization
- Caching Strategies and Performance Tuning
- Queue Systems and Background Jobs
- API Design Best Practices
- Security Hardening (OWASP, API Security)
- Authentication and Authorization Patterns
- Validation and Form UX Best Practices
- Error Handling and Logging
- Testing Strategies (Unit / Feature)
- Deployment and Production Best Practices

Prerequisites

Participants should have prior experience working with Laravel (routing, controllers, Blade, and database operations) and basic understanding of web application development.

Audience:

Intermediate Laravel developers, backend developers, and teams aim to improve code quality, scalability, and system performance.

Duration: 3 Days

Course Outline

Day 1 - Foundations, Environment Setup, and Modern PHP for Laravel

1. Clean Code and Modular Architecture
    - 1.1 Naming conventions and code readability
    - 1.2 Single Responsibility Principle (SRP)
    - 1.3 Refactoring controllers and services
    - 1.4 Avoiding fat controllers and models
    - 1.5 Writing maintainable and self-explanatory code
    - 1.6 Organizing reusable business rules

2. Service Layer and Repository Pattern
    - 2.1. Separating business logic
    - 2.2. Implementing service classes
    - 2.3. Repository pattern for database abstraction
    - 2.4. Keeping controllers thin and focused
    - 2.5. Structuring reusable application services
    - 2.6. Deciding when repository pattern is appropriate

3. Modular Architecture
    - 3.1. Domain-driven structure
    - 3.2. Feature-based folder organization
    - 3.3. Scaling Laravel for large applications
    - 3.4. Defining module boundaries clearly
    - 3.5. Organizing shared components and utilities
    - 3.6. Maintaining consistency across larger teams

4. Hands-On Lab
    - 4.1. Refactor a CRUD application into modular structure
    - 4.2. Introduce service layers into an existing codebase
    - 4.3. Move business rules out of controllers
    - 4.4. Improve readability and maintainability of existing modules

Day 2 - Performance, APIs, and Security

5. Performance Optimization
    - 5.1. Query optimization
    - 5.2. Eager loading vs lazy loading
    - 5.3. Caching strategies
    - 5.4. Using Redis
    - 5.5. Reducing unnecessary database round trips
    - 5.6. Profiling common performance bottlenecks

6. Queues and Background Jobs
    - 6.1. Queue fundamentals
    - 6.2. Handling async tasks
    - 6.3. Real-world use cases
    - 6.4. Dispatching and processing jobs
    - 6.5. Retry and failure handling concepts
    - 6.6. Choosing suitable queue use cases in Laravel systems

7. API Design Best Practices
    - 7.1. REST API design
    - 7.2. Consistent response structures
    - 7.3. Versioning APIs
    - 7.4. Designing predictable endpoints
    - 7.5. Returning useful validation and error responses
    - 7.6. Maintaining API clarity for frontend and mobile teams

8. Security Best Practices
    - 8.1. OWASP overview
    - 8.2. CSRF, XSS, SQL Injection prevention
    - 8.3. Secure API authentication
    - 8.4. Protecting sensitive application data
    - 8.5. Common Laravel security hardening practices
    - 8.6. Reviewing risky coding patterns

9. Hands-On Lab
    - 9.1. Optimize an API endpoint
    - 9.2. Implement caching and queue processing
    - 9.3. Improve query performance in an existing module
    - 9.4. Review and harden a sample API flow

Day 3 - UI/UX, Testing, and Production Readiness

10. UI/UX Best Practices
    - 10.1. Form validation UX
    - 10.2. Error handling UX
    - 10.3. API-driven UI considerations
    - 10.4. Designing user-friendly feedback messages
    - 10.5. Improving form flow and usability
    - 10.6. Aligning backend validation with frontend experience

11. Testing Strategies
    - 11.1. Unit testing
    - 11.2. Feature testing
    - 11.3. Test-driven development basics
    - 11.4. Testing controllers, services, and business rules
    - 11.5. Testing API responses and validation flows
    - 11.6. Building confidence for refactoring and release

12. Deployment and Production
    - 12.1. Environment configuration
    - 12.2. CI/CD basics
    - 12.3. Monitoring and logging
    - 12.4. Production deployment checklist
    - 12.5. Handling environment-specific configuration safely
    - 12.6. Observability and troubleshooting in production

13. Final Project
    - 13.1. Build and refactor a Laravel system
    - 13.2. Apply all best practices
    - 13.3. Deploy and present
    - 13.4. Improve architecture and code quality
    - 13.5. Optimize performance and security
    - 13.6. Explain technical design decisions clearly