Laravel Capstone Project – Requirements Document

Course: Laravel PHP Framework 12.x
Work Style: Individual Project

1. Introduction

This Capstone Project is the **final work** of the Laravel course.
You are required to design and build a **real-world full-stack web application** that demonstrates your mastery of:

- **Backend:** Laravel 12 (secure, maintainable, industry-standard).
- **Frontend:** Vue.js (can be built as part of a monolithic Laravel app or as a standalone SPA).
- **Database:** Relational database (MySQL or PostgreSQL).

Your project must integrate **authentication, authorization, CRUD operations, file uploads, notifications, analytics dashboards, and APIs**.

2. Project Goal

Deliver a fully functional, secure, and professional web application that shows:

• Strong **backend development** using Laravel 12 (MVC, ORM, REST APIs). •
Clean and modern **frontend** with Vue.js + Tailwind CSS.
• Core industry features: authentication, role-based access, file uploads, notifications. •
Extra polish with an **analytics dashboard** and **responsive design**.

3. Project Architecture Options

You may choose **one** of the following approaches:

1. **Monolithic App:** Use Laravel with Blade + Vue components inside the same project.
2. **Decoupled SPA:** Build a separate Vue.js SPA that consumes a Laravel REST API.

�� 👉Both are valid. Just document which approach you chose.

4. Mandatory Features

Backend (Laravel 12)

- Follows **MVC architecture**.
- Database: migrations, factories, seeders.
- CRUD operations with **Eloquent ORM** and relationships.
- RESTful API (/api/v1/...) if using a decoupled SPA.
- Authentication & Authorization:

  - Laravel Breeze or Sanctum/Fortify.
  - Role-based access control (Gates & Policies).
- File uploads (e.g., images, documents).
- Notifications:
  - Stored in database.
  - Sent via email.
  - Shown as **toasts** in the frontend.
- Error handling, input validation, and logging.
- Security measures: CSRF, password hashing, validation.

Frontend (Vue.js + Tailwind CSS)

- Dark/Light mode toggle.
- Fully **responsive design** (desktop & mobile).
- Dynamic UI integrated with backend APIs.
- Interactive forms with validation & error messages.
- Analytics dashboard with at least **2 charts** (e.g., user stats, activity logs, sales). • Toast notifications for success/error feedback.

Database

- At least **4 entities** in the schema.
- Includes relationships: One-to-One, One-to-Many, Many-to-Many. • Optimized queries with **eager loading**.

5. Suggested Project Ideas

Choose **one** of the following, or propose a new idea (with instructor

approval). **1. Humanitarian Aid Management Platform**

For NGOs to manage aid distribution.

- **Users:** Admin, Volunteers, Beneficiaries.
- Features:
  - Admin registers volunteers, manages donations & distributions.
  - Beneficiaries request aid.
  - Volunteers track deliveries.
  - File uploads: ID documents, aid receipts.
  - Notifications: aid request approved/denied.
  - Dashboard: donations, beneficiaries served, active volunteers.

2. Learning Management System (LMS)
A platform for online learning.

- **Users:** Admin, Instructors, Students.

- Features:
  - Instructors create/manage courses & lessons.
  - Students enroll, submit assignments.
  - File uploads: resources, assignments.
  - Notifications: grades, deadlines.
  - Dashboard: student enrollments, completion rates, submissions.

3. Event Management & Ticketing System

A platform for event organizers.

- **Users:** Admin, Event Organizers, Attendees.
- Features:
  - Organizers create events & tickets.
  - Attendees register and purchase tickets.
  - File uploads: event posters, tickets (PDF).
  - Notifications: ticket confirmation, event updates.
  - Dashboard: sales, revenue, attendee stats.

4. E-Commerce Platform with Order Tracking

An online store with full checkout.

- **Users:** Admin, Vendors, Customers.
- Features:
  - Customers browse, add to cart, checkout.
  - Vendors manage inventory & orders.
  - File uploads: product images.
  - Notifications: order confirmation, shipping updates.
  - Dashboard: sales, stock alerts, top products.

5. Multi-Vendor Marketplace

A marketplace with multiple sellers.

- **Users:** Admin, Vendors, Customers.
- Features:
  - Vendors manage shops, products, and orders.
  - Customers browse across vendors.
  - File uploads: logos, product images.
  - Notifications: vendor approvals, order status updates.
  - Dashboard: vendor performance, revenue growth.

6. Deliverables

You must submit:

1. **GitHub Repository** with meaningful commits.
2. **Documentation:**
   - o README (setup, usage).
   - o API documentation (if using SPA).
3. **Demo Presentation** (8–12 minutes).

7. Optional (Bonus) Features

For extra credit and portfolio strength, you may implement:

1. **Localization / Multi-Language Support**
   - o Support multiple languages (e.g., English & Arabic).
   - o Use Laravel's lang directory for translations.
   - o Add a language switcher in the frontend.
2. **Real-Time Features**
   - o Live chat between users.
   - o Real-time notifications with Laravel Echo + Pusher.
3. **Advanced Search & Filtering**
   - o Dynamic filtering (category, price, status).
   - o Full-text search with Laravel Scout + Meilisearch/Algolia.
4. **Mobile-Friendly PWA**
   - o Convert your Vue.js frontend into a PWA.
   - o Allow users to "install" it as a mobile app.
5. **Data Export & Reporting**
   - o Export data (CSV, Excel, PDF).
   - o Auto-generate weekly/monthly reports for admins.

8. Project Discussion & Evaluation Points

When presenting your **Capstone Project**, you will be evaluated on the following key aspects. Make sure your slides, demo, and explanations cover these areas clearly.

1. Backend System Design & Architecture

- **Laravel Best Practices**
  - o How did you apply Laravel's conventions (MVC, Service classes, etc.)?
  - o Why is your backend design maintainable, scalable, and robust?
- **Database Design & Eloquent Usage**
  - o Justify your chosen schema and relationships.
  - o How efficient is your design (indexes, normalization, eager loading)?

- How did you leverage Eloquent ORM for CRUD and relationships? • **API Design Quality**
    - Are your endpoints **RESTful, consistent, and well-named**?
    - How are resources modeled, and which HTTP methods were used for actions?
    - Did you version your APIs (/api/v1/...)?

2. Functionality & Completeness

- **Core Feature Implementation**
    - Demonstrate that all mandatory features (file uploads, analytics dashboard, notifications/toasts, auth) are working.
- **Full-Stack Integration**
    - Show how the **frontend (Vue.js)** and **backend (Laravel)** interact smoothly.
    - How efficient is your data flow?
- **Feature Completeness**
    - Are all chosen **optional/bonus features** (e.g., localization, real-time) implemented and functional?

3. Security & Reliability

- **Authentication & Authorization**
    - How robust is your login/registration flow?
    - How do you enforce **role-based access control** (Gates, Policies, Middleware)?
- **Input Validation & Error Handling**
    - How do you validate form input (server-side)?
    - How are errors handled and reported to the frontend?
- **Data Security**
    - How do you protect sensitive data (password hashing, CSRF tokens, secure cookies, .env secrets)?
    - Any steps taken to mitigate XSS/SQL injection?

4. Code Quality & Maintainability

- **Readability & Structure**
    - Is your code modular, consistent, and readable?
    - Did you follow **PSR-12 / Laravel coding standards**?
- **Code Reusability**
    - Where did you implement reusable components/services?
    - Did you avoid code duplication?
- **Version Control**
    - Show your GitHub repo: meaningful commits, branches, and version control best practices.

5. Problem-Solving & Technical Challenges

- **Challenges Faced**
    - What were the **biggest obstacles** in your project?
    - How did you **individually solve them**?
- **Debugging Process**
    - Walk through an example of how you debugged a backend or frontend issue.

- What tools or methods did you use (logs, Tinker, browser dev tools, etc.)?

6. Overall Presentation & Justification

- **Technical Explanation**
    - Be ready to explain your **architecture, API design choices, database schema, and Laravel-specific implementations**.
- **Project Demonstration**
    - Deliver a clear and engaging demo:
        - Show login/auth flow.
        - Perform a CRUD example (create/update/delete).
        - Demonstrate file upload + notification.
        - Show dashboard charts.
        - Show dark/light mode toggle.
    - Keep it **8–12 minutes** with focused storytelling: problem → solution → features → demo → conclusion.

Pro Tips for Success

- Prepare **slides** that summarize each of the above categories.
- Show **live demo**, but also have **screenshots/videos** ready in case of technical issues.