PROJECT REPORT

SOFTWARE ARCHITECTURE

Course Instructor: Vu Quang Dung

Group 14

Project Title: Cinema Management

Nguyen Que Bac
23010574

Hoang Tuan Kiet
23010517

Do Bao Long
23010561

1.Project Requirements

Core Functional Requirement

ID	Description
FR-01	The system must allow users to register and authenticate using a unique account.
FR-02	The system must allow users to view the list of available movies and their details (title, genre, duration, and description).
FR-03	The system must allow users to view movie showtimes based on date and cinema room.
FR-04	The system must allow users to select seats and create a ticket booking for a specific showtime.
FR-05	The system must validate seat availability and prevent double booking.
FR-06	The system must allow users to confirm, cancel, or view their booking history.
FR-07	The system must allow administrators to manage movies, showtimes, cinema rooms, and seating layouts.
FR-08	The system must allow administrators to view statistical reports including ticket sales, revenue, and booking status.

1.Project Requirements

Proposed Model of ASR

ASR: Performance and Scalability (Non-functional Requirement)

· **Statement**:
The system must support a large number of concurrent users (e.g., 500–1000 users) during peak booking periods while maintaining acceptable response times (under 2 seconds) for seat selection and booking confirmation.

· **Impact**:
This requirement influences the architectural choice. It drives the use of an optimized monolithic architecture with efficient database schema design, caching mechanisms (e.g., Redis for frequently accessed data like showtimes and seat availability), and proper concurrency control to ensure stable performance under increased load.

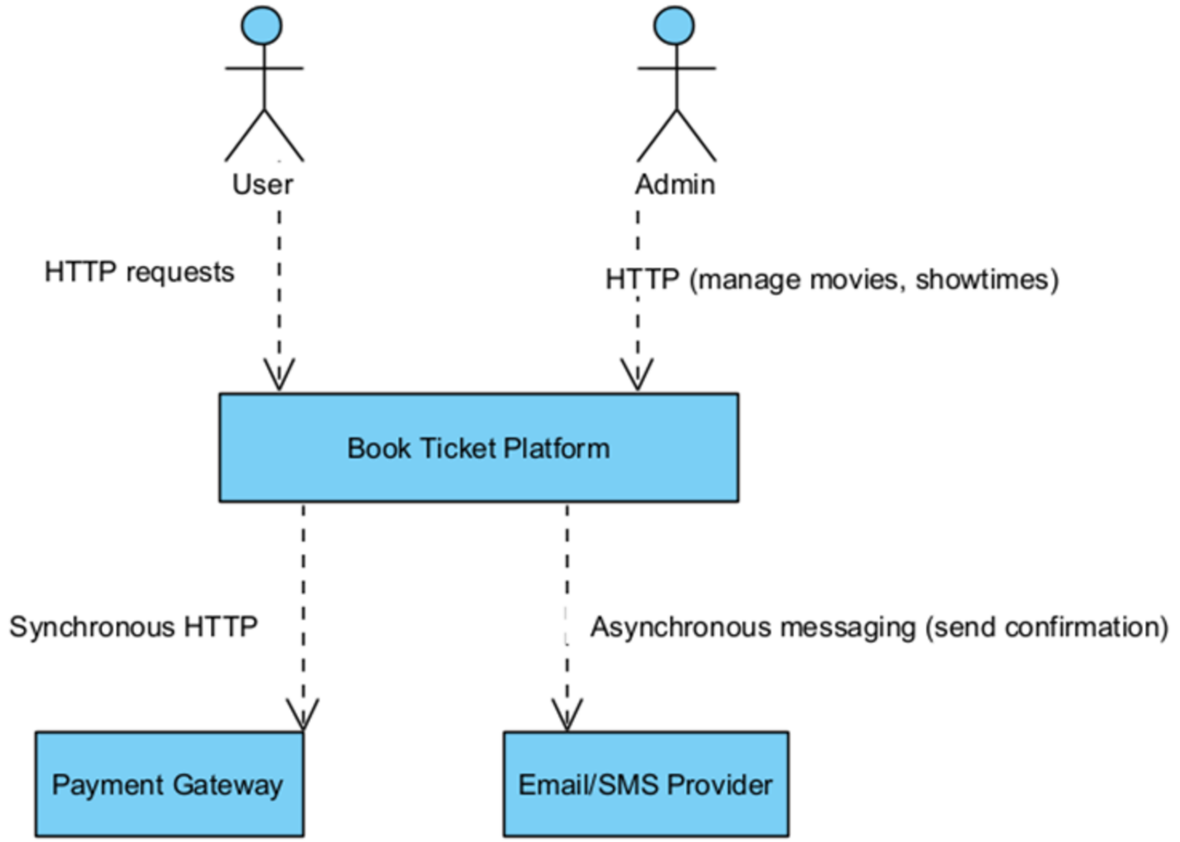
· **Rationale**:
A monolithic layered MVC architecture is chosen because the system is medium-scale, where simplicity and maintainability are prioritized. At the same time, performance and scalability can be achieved through database optimization, caching, and horizontal scaling if needed. This balances development speed with system reliability during peak usage.

2. Use Case Modeling

Use Case Diagram

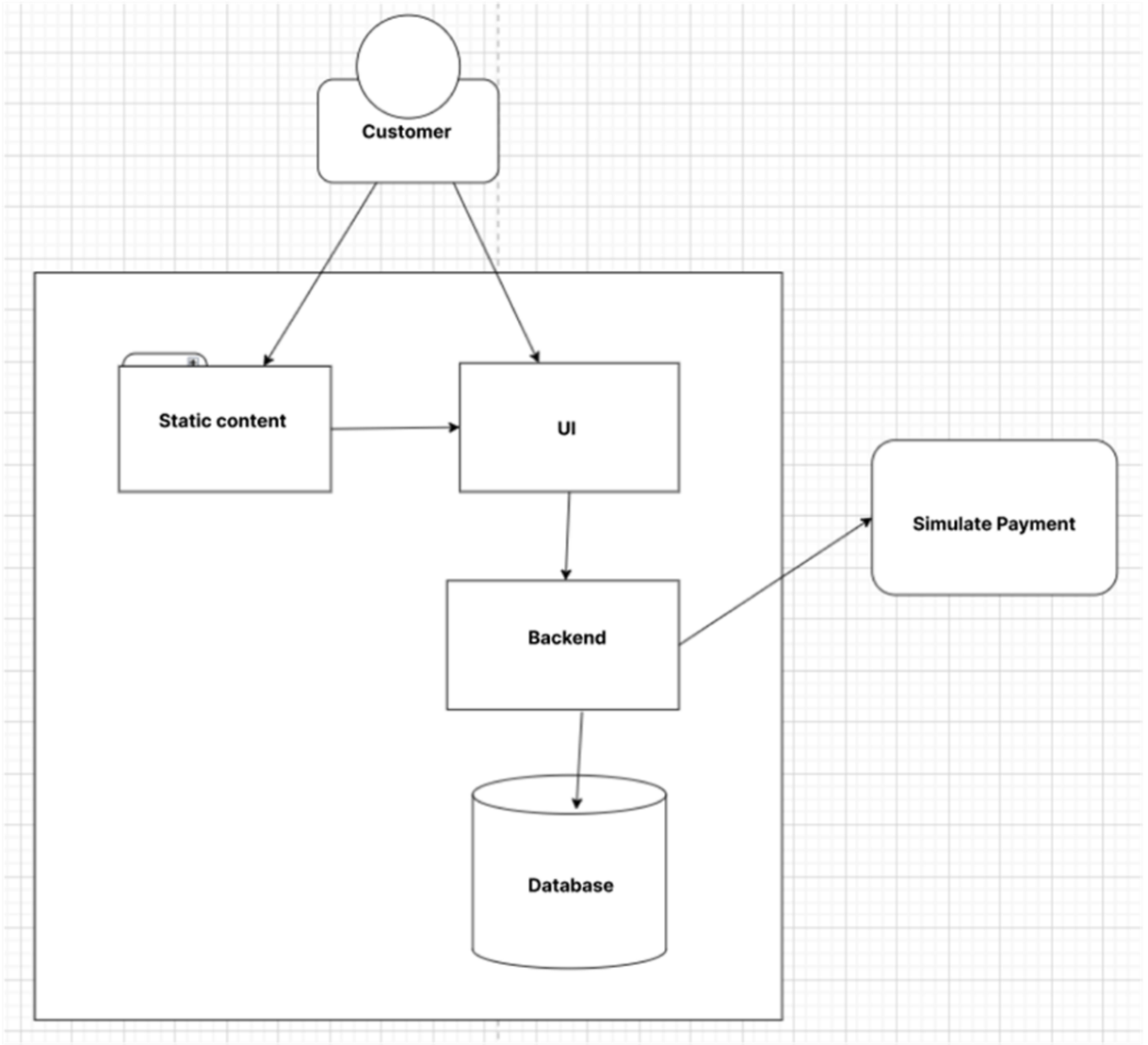
2. C4 Model

System Context Diagram (C4 Model – Level 1)



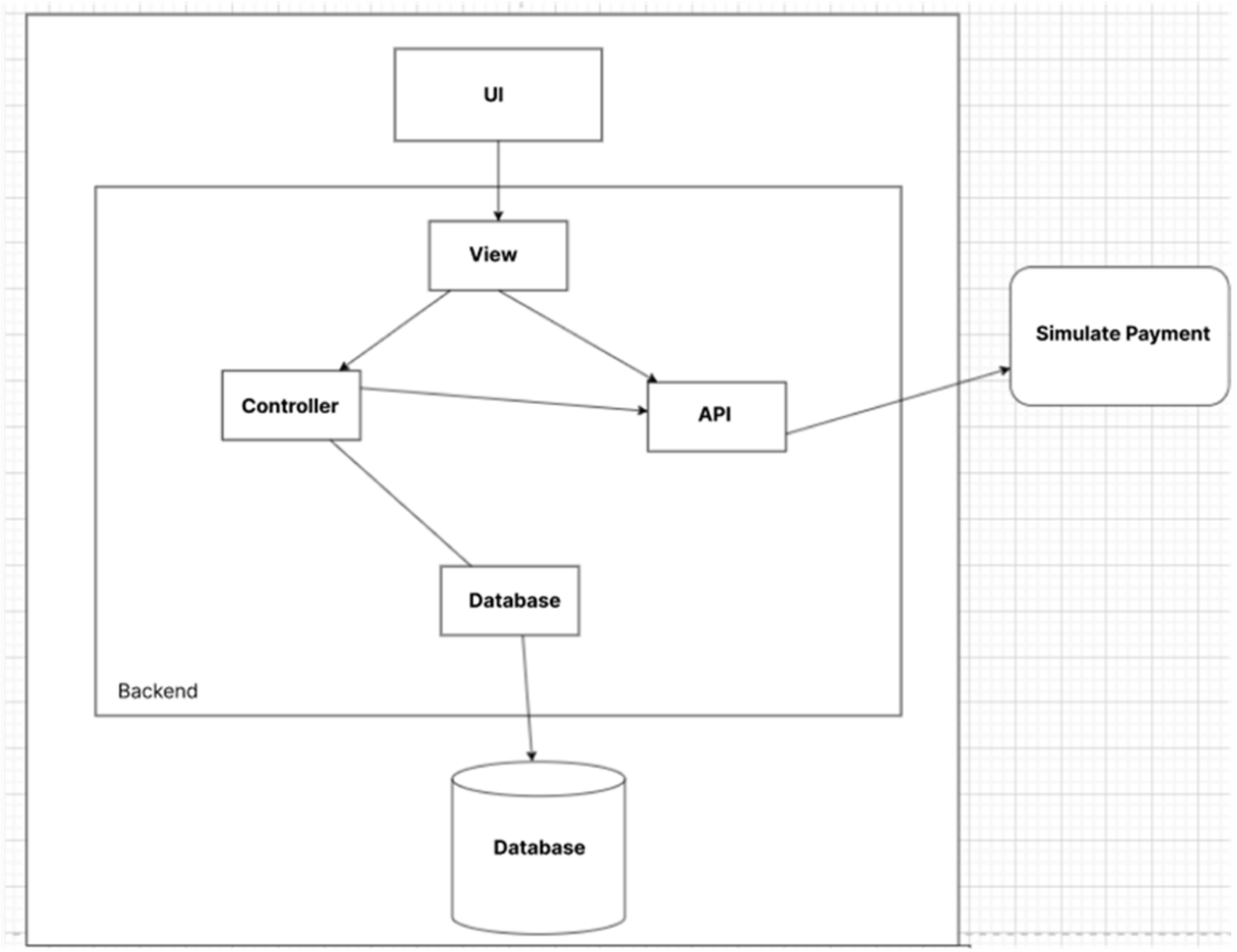
2. C4 Model

Container Diagram (C4 Model – Level 2)



2. C4 Model

Component Diagram (C4 Model – Level 3)



3. Detailed Architecture

Core Components

1. Web Browser / Client

o Users and administrators interact with the system through a web browser.

o Requests are sent via HTTP (GET or POST) to the Laravel backend.

o Pages are rendered using Blade templates, showing dynamic content such as movie lists, seat availability, and booking confirmations.

2. Laravel Application (Monolithic Backend)

· Controllers handle incoming requests and coordinate application logic.

§ BookingController: manages seat selection, booking creation,

§ MovieController: manages movie listing and details.

§ ShowtimeController: manages showtimes and seat availability.

§ AdminController: generates reports and manages users, rooms, and showtimes.

§ User, Movie, Showtime, Seat, Room , Booking, BookingSeet BookingCombo

3. Database (MySQL)

· Stores all persistent data: users, movies, showtimes, cinema rooms, seats, bookings, combo items.

· Supports transactional integrity, ensuring seat booking is atomic and preventing double booking.

· Relationships between entities are managed via foreign keys and Eloquent ORM.

3. Detailed Architecture

Flow 1: Movie Booking

1. User Interaction

o A user navigates to the booking page via the Web Browser (UI).
o The interface displays available movies, showtimes, and seat layouts using Blade templates.
o The user selects seats and optionally adds combo items (e.g., popcorn, drinks).

2. Form Submission

o The browser sends a POST request to BookingController@store.
o Request parameters include user ID, showtime ID, selected seats, and combo items.

3. Server Processing

o BookingController validates input and checks seat availability.
o A database transaction begins:
§ A new Booking record is created.
§ BookingSeat entries are inserted for each selected seat.
§ BookingCombo entries are recorded if applicable.
§ The total price is calculated.
o If all steps succeed, the transaction commits; otherwise, it rolls back to prevent inconsistencies.

4. Response

o The system returns a confirmation page (Blade template) or a JSON response for API endpoints.
o The UI updates to reflect booked seats and displays a success message.

4. Testing

Step	Action	Expected Result	Actual Result	Status
1	Open Browser as User A	Booking page for a specific showtime loads successfully	Booking page loads with available seats	Pass
2	Select one or more available seats and submit booking form	Seats are validated; booking is created in the database; user sees a confirmation message	Booking created successfully; confirmation displayed	Pass
3	Open Browser as User B and check the same showtime	Previously booked seats are no longer available	Booked seats disabled / marked as sold	Pass
4	Attempt to book already booked seats	System prevents booking and shows an error message	Error message displayed: "Some seats are already booked"	Pass

5.Conclusion & Reflection

This project successfully implemented a Cinema Management System using a Monolithic Laravel architecture. Core functionalities including movie browsing, seat selection, booking, and administrative reporting were fully implemented. The system ensures data integrity and prevents double booking through server-side validation and transactional handling. Verification confirmed that the booking workflow operates correctly, with the database updated accurately and the user interface reflecting seat availability.

The project demonstrates that a Monolithic MVC approach is effective for managing cinema operations, providing a simple, cohesive, and reliable solution for both customers and administrators.

Thank you for listening