LAB 4 REPORT: MICROSERVICES DECOMPOSITION & COMMUNICATION

Project Name: Movie Ticket Booking System

Member Name: Ngô Đình Quyền - 23010485

Trần Hữu Kiên - 23010258

---

1. MICROSERVICE DECOMPOSITION

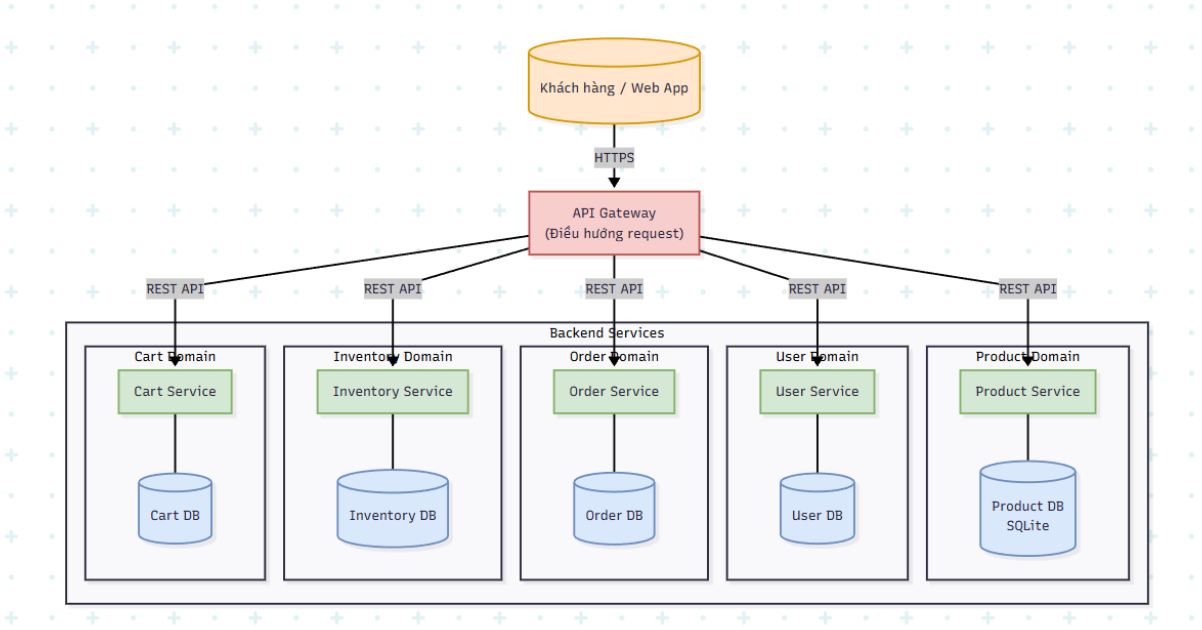
In this section, the monolithic application is decomposed into independent microservices based on **Business Capabilities**. Each service owns its specific data entities and operates loosely coupled from others.

1.1. Identified Microservices

Business Capability	Microservice Name	Data Owned (Entities)	Responsibility
Catalog Management	**Movie Service**	`Movie`, `Genre`, `Actor`	Manages movie details, metadata, and media (posters/trailers).
Scheduling	**Showtime Service**	`Showtime`, `TheaterRoom`, `SeatMap`	Manages screening schedules and real-time seat availability (Inventory).
User Management	**User Service**	`User`, `Role`, `MembershipPoint`	Handles registration, authentication (JWT), and user profiles.
Booking & Sales	**Booking Service**	`Ticket`, `BookingOrder`, `Invoice`	Processes ticket reservations and manages booking lifecycles.

Business Capability	Microservice Name	Data Owned (Entities)	Responsibility
Payments	**Payment Service**	`Transaction`, `PaymentMethod`	Interacts with external payment gateways (Momo, Visa) to process funds.

1.2. Architecture Logical View



---

2. SERVICE CONTRACTS (API DEFINITIONS)

This section defines the Interface (API) for the core service: **Movie Service**. This contract allows other services (like Booking Service) and the Frontend to interact with it.

2.1. Movie Service API (RESTful)

Endpoint	HTTP Method	Description	Request Body / Parameters	Response Data
`/api/movies`	**GET**	Retrieve a list of currently showing movies.	`?page=1&genre=Action`	`List<MovieDTO>`
`/api/movies/{id}`	**GET**	Get full details of a specific movie.	Path param: `id`	`MovieDetailDTO`
`/api/movies`	**POST**	Add a new movie (Admin only).	JSON: `{title, duration, ...}`	`MovieDTO` (Created)
`/api/movies/{id}`	**PUT**	Update movie information.	JSON: `{title, ...}`	`MovieDTO` (Updated)

2.2. Service Interaction Example

Scenario: A user wants to book a ticket.

- **Consumer:** `Booking Service`
- **Producer:** `Showtime Service`
- **Contract:** The Booking Service calls `GET /api/showtimes/{id}/seats` to check if the selected seats are still available before creating an order.

---

3. COMMUNICATION STRATEGY & SYSTEM CONTEXT

This section defines how the system interacts with the outside world and how internal services communicate.

3.1. Communication Strategy

Interaction	Type	Technology	Rationale
Frontend $\leftrightarrow$ API Gateway	**Synchronous**	REST (HTTP/JSON)	The user expects an immediate response when browsing movies or clicking "Book".
Booking Service $\rightarrow$ Payment Service	**Synchronous**	REST (HTTP/JSON)	Payment verification must happen immediately to confirm the booking.
Booking Service $\rightarrow$ Email Service	**Asynchronous**	Message Queue (RabbitMQ)	Sending the e-ticket email can happen in the background; the user doesn't need to wait for it to finish to see the "Success" screen.

3.2. C4 Model - Level 1: System Context Diagram

The diagram below illustrates the **Movie Ticket Booking System** as a black box and its interactions with external users and systems.