PHENIKAA UNIVERSITY

PHENIKAA UNIVERSITY: SCHOOL OF COMPUTING

COURSE: SOFTWARE ARCHITECTURE

Lab 4: Microservices Decomposition & Communication

Instructor:	Vu Quang Dung
Group 8:	Nguyen Gia Bao (23010383)
	Pham Xuan Bach(23010118)
	Bui Minh Duc (23010513)
	Vu Duc Hieu (23010223)
Class:	**CSE703110-1-2-25(N01)**

Hanoi, 2025 December

INDEX

1. Abstract/Summary..... 2
2. Decomposition by Business Capability.....3
2.1. Identify Core Business Capabilities:..... 3
2.2. Define Microservices:..... 3
2.3. Define External Dependencies:..... 4
3. Defining Service Contracts..... 4
3.1. Define the Movie Service's API..... 4
3.2. Define the Booking Service's Interaction (Consumer)..... 5
4. System Context Diagram Description (C4 Model - Level 1)..... 5
5. Conclusion & Reflection..... 6

1. Abstract/Summary

In this lab, the Movie Ticket Booking System project transitioned from the monolithic Layered Architecture to a Microservices Architecture to address previous scalability limitations. We applied the Decomposition by Business Capability pattern to identify independent microservices (User, Movie, Booking, Payment) and formally defined their Service Contracts using RESTful API specifications.

The system's high-level boundaries and external interactions were modeled using the C4 System Context Diagram (Level 1). Furthermore, a hybrid communication strategy was established, utilizing synchronous mechanisms (HTTP) for real-time operations and asynchronous patterns (Message Queues) for decoupled tasks, providing a solid blueprint for independent service development.

2. Decomposition by Business Capability

2.1. Identify Core Business Capabilities:

Based on the functional requirements, the system's core domains are identified as follows:

- **User Management:** Handling customer registration, authentication, and profiles.
- **Movie Catalog Management:** Managing movies, theaters, halls, and showtime schedules.
- **Booking & Reservation:** Handling seat selection, seat locking, and ticket generation.
- **Payment Processing:** Managing financial transactions.
- **Notification:** Sending booking confirmations via Email/SMS.

2.2. Define Microservices:

We map the capabilities above to dedicated microservices to ensure loose coupling:

Business Capability	Proposed Microservice	Data Owned
User Management	User Service	User Profile, Account Credentials, Role (Admin/Customer)
Movie Catalog	Movie Service	Movie Details, Genres, Theater Rooms, **Showtimes**, Seat Layouts (Static).
Booking & Seats	Booking Service	**Tickets**, Reservations, **Seat Status** (Available/Locked/Booked) for specific showtimes.
Payments	Payment Service	Transaction Records, Payment Status, Invoice.
Notifications	Notification Service	Email Templates, SMS Logs.

2.3. Define External Dependencies:

The system must interact with the following external third-party systems:

- **Payment Gateway (e.g., VNPay, Momo, Stripe):** To process secure credit card or e-wallet transactions.
- **Email Provider (e.g., SendGrid, Gmail SMTP):** To send ticket confirmations and QR codes to users.
- **SMS Gateway (e.g., Twilio):** To send OTPs for login or booking reminders.

3. Defining Service Contracts

3.1. Define the Movie Service's API

In this architecture, the **Movie Service** acts as the Producer. It manages data related to movies and showtimes.

Endpoint	HTTP Method	Description	Data Returned
/api/movies/{id}	**GET**	Retrieve full details of a single movie	Movie object (Title, Genre, Duration, Description)
/api/showtimes/{id}	**GET**	Retrieve details of a specific showtime (essential for pricing)	Showtime object (Hall, Start Time, Price, Seat Layout)
/api/movies	**GET**	Search/ list movies based on criteria (e.g., currently	List of movies object (summary view)

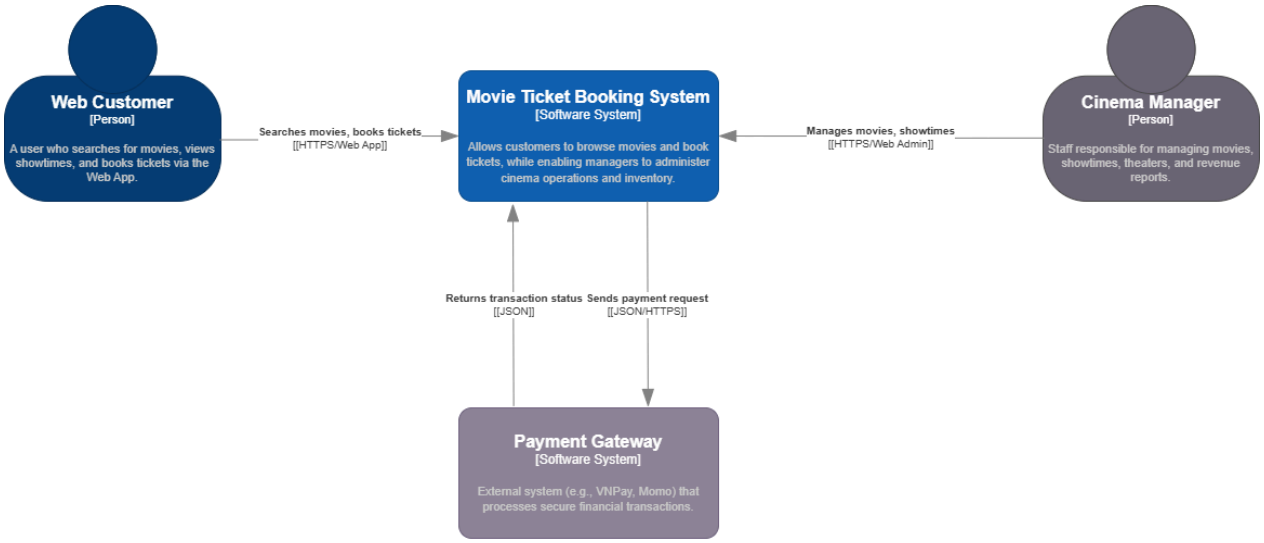
		showing)	
/api/movies	**POST**	Add a new movie (Admin function)	The created movie object

3.2. Define the Booking Service's Interaction (Consumer)

The **Booking Service** acts as the Consumer. This interaction occurs when a user attempts to book a ticket for a specific showtime.

- **Action**: The Booking Service needs the Price and Hall information of the specific showtime ID requested to calculate the total cost and generate the ticket.
- **Requirement**: The Booking Service must not access the Movie Service's database directly (this violates the microservice boundary). It must use the defined API endpoint (GET /api/showtimes/{id}).

4. System Context Diagram Description (C4 Model - Level 1)



- **Central System:** This is the "Movie Ticket Booking System." At Level 1, we do not separate the Web App and Database; instead, we treat it as a "Blackbox" that provides full functionality.
- **Web App Interaction:** Although the assignment requires "Showing the Web App," in the standard C4 Level 1 model, the Web App is typically understood

as the interface of the "System." However, if the instructor explicitly requires separating the Web App at Level 1 (which leans slightly towards Level 2), you can draw an arrow from the Customer with a label stating "Access via Web App."

- **Payment Gateway:** This is a crucial component for satisfying non-functional requirements regarding security and interoperability (ASR 3).

5. Conclusion & Reflection

Conclusion In this lab, we successfully transitioned the **Movie Ticket Booking System** design from a Monolithic Layered Architecture to **Microservices**. We decomposed the system into independent capabilities (User, Movie, Booking, Payment) and defined their **API Service Contracts**. The **C4 System Context Diagram** was also created to visualize the system's boundaries and external interactions.

Reflection The move to Microservices addresses previous scalability issues but introduces new complexity in **inter-service communication**. We learned that strictly defining API interfaces and choosing the right communication strategy (Synchronous vs. Asynchronous) is critical to maintaining loose coupling between services like Booking and Movie.