CS4067 – Assignment 01: Microservices-Based Online Event Booking Platform with Jira & GitHub Integration

Course Code: CS4067

Course Title: DevOps and Cloud Native

Total Marks: 100

Due Date: Feb 24, 2025

Objective

This assignment requires students to develop an **Online Event Booking Platform** using a **microservices architecture**. The goal is to demonstrate:

- **Synchronous & Asynchronous communication** between microservices
- **MongoDB & PostgreSQL integration** for data storage
- **Jira integration** for issue tracking
- **GitHub for version control, project management, and documentation**

Important Notes

- This assignment emphasizes the development process of microservices and their interactions, offering valuable insights into the true benefits of DevOps practices.
- The same application will be used throughout the course to apply various tools that we will cover. At a later stage, the application will be modified to seamlessly integrate these tools, ensuring it can adapt to the necessary changes as we progress.
- Please note that Dockerizing the microservices is NOT part of this assignment and will be addressed in a later phase.

Application Overview

Real-World Use Case:

Users can **browse events, book tickets, and receive confirmation notifications**. The system will manage event listings, user accounts, booking payments, and real-time notifications.

Microservices Details

Microservice	Functionality	Tech Stack	Database	Communication
User Service	Manages user authentication & profiles	FastAPI / Express.js	PostgreSQL	REST API (Sync)
Event Service	Manages event listings & details	Spring Boot / Node.js	MongoDB	REST API (Sync)
Booking Service	Handles ticket bookings, payments & status updates	Flask / Express.js	PostgreSQL	REST API (Sync), RabbitMQ (Async)
Notification Service	Sends email/SMS notifications for confirmations	Flask / Express.js	MongoDB	RabbitMQ (Async)

Notes:

- The technology stack is flexible and can be chosen based on your preferences. The table above provides suggested options.
- You are not limited to these microservices only, you can include more with a valid justification.

Communication Between Microservices

User Service → **Event Service** (Sync via REST API)

- Users retrieve available events from the **Event Service**.
- **Example API Call:** `GET /events`

User Service → **Booking Service** (Sync via REST API)

- Users create a booking by calling the **Booking Service**.
- **Example API Call:** `POST /bookings` with `{ user_id, event_id, tickets }`

Booking Service → **Notification Service** (Async via RabbitMQ)

- When a booking is confirmed, the **Booking Service** publishes an event to RabbitMQ.

- The **Notification Service** consumes this event and sends a confirmation email.
- **Example RabbitMQ Event:** `{ booking_id, user_email, status: "CONFIRMED" }`

Booking Service → Payment Gateway (Mock Service) (Sync via REST API)

- The **Booking Service** processes payments before confirming a booking.
- **Example API Call:** `POST /payments` with `{ user_id, amount }`

Event Service → Booking Service (Sync via REST API)

- Before confirming a booking, the **Booking Service** checks event availability.
- **Example API Call:** `GET /events/{event_id}/availability`

Implementation Considerations

While developing the application, please ensure the following best practices are followed:

- Use **try-catch blocks** to handle unexpected behaviors and exceptions effectively, ensuring the application remains resilient.
- Implement **comprehensive logging** for various events and activities (e.g., errors, informational messages).
- Logs should be stored in a single file, with the name of the microservice and function included for easier traceability.
- Ensure consistent error handling across all microservices to provide clear, actionable error messages.
- Adhere to **coding standards and best practices** to maintain code readability, reusability, and maintainability.
- Use **environment variables** (e.g., for sensitive data like API keys, database credentials, and endpoints of other microservices) and ensure secure handling of such configurations.
- Ensure proper API documentation for all exposed endpoints, specifying request/response formats and error codes.
- Follow **version control best practices**, including meaningful commit messages and the use of branches for feature development and bug fixes.

Jira & GitHub Integration

Step 1: Jira Setup

- Sign up for a free Jira account.
- Create a **Jira project** named `CS4067_EventBooking_RollNo_YourName`
  - e.g., `cs4067_i221080_Abdul_Munim`
- Define the workflow with **To Do, In Progress, Review, Done** states.
- Create tasks for each microservice and integration step.

- Take some screenshots of your Jira dashboard with the newly created project and tasks.

Step 2: GitHub Project Setup

- Create a **GitHub project** named `CS4067-Assgt-EventBooking-RollNo-YourName`.
- Add **Kanban columns** similar to Jira (`To Do`, `In Progress`, `Review`, `Done`).
- Link **GitHub issues with Jira tasks** for synchronization.
- Take some screenshots of the project and synced issues.

Step 3: GitHub Repository Setup

- Create a GitHub repository named `CS4067-Assgt-EventBooking-RollNo-YourName-repo` to serve as the monorepo for managing the codebase of the microservices.
- Organize the repository by creating separate directories for each microservice.
- Ensure that relevant code, configuration files (such as `.env`), and other necessary resources are appropriately stored within their respective directories.

Step 4: Automate Jira-GitHub Sync (optional)

- When a **GitHub commit or PR** references a Jira issue (`CS4067-123`), it should update Jira automatically.
- **Example Commit Message:** `Implemented booking API (CS4067-123)` → Moves issue to "In Progress" in Jira.

Expected Deliverables

1. **Fully functional microservices** (User, Event, Booking, Notification).
2. **Working Jira board** with tasks, progress tracking, and GitHub integration.
3. **GitHub repository** with:
    a. `README.md` (Architecture, API docs, setup guide)
    b. Source code for each microservice (as monorepo)
    c. API specification (Swagger/OpenAPI) (optional)
4. **Architectural Diagram** (Illustrating microservice communication)
5. **Screenshots of Jira-GitHub Integration** is the form a document (README.md recommended)

Evaluation Criteria

Criteria	Points
Microservices Development (Functionality)	50
Proper Communication (Sync & Async)	20
Jira & GitHub Integration	20
Code Quality & Documentation	05
Overall System Design	05

Reference Links

- https://medium.com/swlh/building-your-first-microservice-80c90af74d9b
- https://kinsta.com/blog/python-microservices/
- https://shakuro.com/blog/rabbitmq-in-microservice-architecture
- https://www.geeksforgeeks.org/microservices-communication-with-rabbitmq/