Status Report for Room Booking and Event Management System

Project Overview

This project is a microservices-based system for managing room bookings, user information, and event approvals. The system leverages four independent services—**User Service**, **Room Service**, **Booking Service**, and **Approval Service**—supported by two databases: **PostgreSQL** and **MongoDB**. All components are containerized using **Docker Compose**.

This report outlines the progress, task distribution, and the assignment requirements fulfilled.

---

1. Task Distribution

The team collaboratively completed the project by dividing the tasks as follows:

Anar's Contributions:

1. **User Service**:
   - Developed the **User Entity** with attributes like name, email, and userType.
   - Implemented CRUD operations for:
     - Creating a user.
     - Deleting a user.
     - Retrieving user details.
   - Integrated the service with **PostgreSQL**.

    - Wrote and tested all endpoints.

2. **Room Service**:
    - Developed the **Room Entity** with attributes like roomId, roomName, capacity, and features.
    - Implemented the service using **Spring Data JPA REST Resources** for CRUD operations, eliminating the need for a custom controller.
    - Integrated the service with **PostgreSQL**.
    - Verified data persistence and endpoint functionality.

3. **Booking Service**:
    - Developed the **BookEvent Entity** for handling room bookings, including attributes like roomId, userId, and approval statuses.
    - Integrated with **MongoDB** for flexible data storage.
    - Created and tested endpoints for:
        - Creating bookings.
        - Updating booking approval statuses.
        - Fetching bookings (all, specific, or unapproved).

Onat's Contributions:

1. **Databases**:

    - Set up **PostgreSQL** for User and Room services.
    - Configured **MongoDB** for Booking and Approval services.
    - Ensured smooth database connections for all services using Docker containers.
2. **Postman Testing**:
    - Created and tested Postman collections for all endpoints across the four services.
    - Documented the request/response flows and ensured endpoints adhered to expected functionality.
3. **Docker and Deployment**:
    - Developed the Dockerfile for each service.
    - Wrote and configured the docker-compose.yml file to containerize and orchestrate all services and databases.
    - Verified the Docker Compose environment by running all containers and ensuring inter-service communication.

---

2. Assignment Requirements Fulfilled

1. **Git Repository**:
    - A private GitHub repository was created for the project.
    - The professor has been added as a collaborator.
    - The repository URL has been tested and is functional.

2. **Video Demonstration**:
   - A video showcasing the project has been created.
   - The video includes:
     - An introduction slide with group member details (names, photos, student IDs, course information).
     - A demonstration of the system using Postman and Docker Compose.
   - All team members contributed to the video.
3. **Functional Microservices**:
   - Each service (User, Room, Booking, Approval) is independently developed and functional.
   - Endpoints are tested for accuracy and reliability using Postman.
4. **Database Integration**:
   - PostgreSQL is used for structured data (User and Room services).
   - MongoDB is used for unstructured, flexible data (Booking and Approval services).
   - Both databases are containerized and linked via Docker Compose.
5. **Docker Environment**:
   - All services and databases are containerized using Docker.
   - Docker Compose ensures smooth orchestration of all containers.
6. **Documentation**:

- API endpoints are documented in Postman collections.
- This status report provides a summary of the requirements met.

---

3. Unfulfilled Requirements

The following requirements were not fulfilled or require further improvement:

1. **Performance Optimization**:
    - While all services are functional, optimization (e.g., load balancing, caching) was not implemented due to time constraints.
2. **Additional Testing**:
    - Unit tests for the backend logic were not comprehensively developed, as the focus was on functional endpoint testing with Postman.

---

4. Challenges and Lessons Learned

1. **Challenges**:
    - Configuring inter-service communication in Docker Compose required debugging.
    - Ensuring MongoDB and PostgreSQL integration across multiple microservices was complex.
2. **Lessons Learned**:
    - Leveraging microservices architecture enhances modularity but requires careful orchestration.

- Docker Compose is a powerful tool for containerization, especially in multi-service applications.

---

5. Final Deliverables

1. **GitHub Repository**: Submitted with full source code and configurations.
2. **Video Presentation**: A 5-10 minute video, including all required elements.
3. **Postman Collection**: Shared for endpoint documentation and testing.
4. **Status Report**: This document, summarizing the progress and requirements.