Room Booking and Event Management System

Team Size: 3 - Team Members (mandatory)
Project Name: **GBC_EventBooking-**<*Group Name*>

Objective:

The goal of this assignment is to design and implement a microservices-based room booking and event management platform for George Brown College. Your platform should allow students, staff, and faculty to book room resources for events and meetings. The focus will be on microservices creation, database integration using JPA, inter-service communication, containerization with Docker, and integration testing.

Instructions and Requirements:

This assignment will cover service development, database integration, inter-service communication, and deployment using Docker.

Environment Setup and Basic RESTful Service

Objective:

Set up your environment and develop the foundational microservices for the platform. Each microservice should be containerized and independently deployable.

Design the backbone of your event management platform by developing RESTful services. Learn how to containerize these services, ensuring they're scalable and deployable.

Environment Setup:

1. Download and install the necessary software specifically:
    - Java
    - Gradle
    - Spring / Spring Boot
    - Docker / Docker Desktop
    - MongoDB (docker container)
    - PostgreSQL (docker container)

Familiarize yourself with the Spring Initializr tool and prepare to set up your Spring Boot projects.

2. Using Spring Initializr, create five microservices:
    - **RoomService**: Manages room resources, such as room availability, capacity, and features.
    - **BookingService**: Handles room booking requests and manages booking information.
    - **UserService**: Manages user profiles for students, staff, and faculty.
    - **EventService**: Manages events linked to room bookings.
    - **ApprovalService**: Allows staff to approve or reject event requests based on college policy.

Each service must be containerized using Docker.

Part 2: Data Persistence and Service Functionality

Objective:

Each microservice should handle its own data using either PostgreSQL or MongoDB. Full CRUD operations should be implemented for managing the core entities in each service. Set up a robust data layer for your platform.

Relational Data Modeling:

- Seed sample data into your databases where/when deemed necessary/useful.
- Establish relationships between your entities:

1. **RoomService:**
   - Use **PostgreSQL / JPA** for data storage.
   - Manage room resources with attributes like roomName, capacity, features (e.g., projector, whiteboard), and availability. Of course, the final design is up to each individual group.
   - Provide endpoints to check room availability for bookings.
2. **BookingService:**
   - Use **MongoDB** for data storage.
   - Handle room booking requests. Each booking should include userId, roomId, startTime, endTime, and purpose. Of course, the final design is up to each individual group.
   - Prevent double-booking of rooms using appropriate validation logic.
3. **UserService:**
   - Use **PostgreSQL / JPA** to store user information.
   - Manage user profiles for students, staff, and faculty with attributes like name, email, role, and userType (student, staff, faculty). Of course, the final design is up to each individual group.
   - Implement role-based access (e.g., only staff can approve events etc…).
4. **EventService:**
   - Use **MongoDB** to manage events.
   - Create events linked to room bookings with attributes such as eventName, organizerId, eventType, and expectedAttendees.
5. **ApprovalService:**
   - Use **MongoDB** or **PostgreSQL** to store approval.
   - Allow staff to review and approve/reject event requests. Link event approvals to the user role.

Part 3: Inter-Service Communication

Objective:

Implement synchronous communication between microservices using REST APIs.

Booking Confirmation:

1. The **BookingService** should communicate with **RoomService** to verify room availability before confirming a booking.

User Access:

2. The **EventService** should communicate with the **UserService** to verify the role of the event organizer before an event is created (e.g., faculty can organize larger events, while students may have restrictions).

Approval Flow:

3. The **ApprovalService** should communicate with both **EventService** and **UserService** to fetch event details and verify if a staff member has the correct privileges to approve events.

Part 4: Testing & Containerization

Objective:

Write integration tests and ensure the system is fully containerized.

1. **Integration Testing**:
   - Write **integration tests** for each microservice to ensure they behave correctly and communicate as expected.
   - Use **TestContainers** to spin up PostgreSQL, MongoDB containers during testing.
2. **Containerization**:
   - Use **Docker** to containerize each microservice.
   - Use **Docker Compose** to orchestrate the deployment of all services needed services.
   - Ensure the system can be launched and run using a **single docker-compose.yml** file.

Remember for this course we primarily want to use containers, and for your assignment solution you need to create a containerized environment as demonstrated in class. Your containerized environment is the only environment that should be demonstrated in your assignment video. I will not be running your project using Intellij, meaning you demonstrations need to also running your solution, not using Intellij.

Deliverables:

- A **private** git repository containing all your code, properly documented.
    - Please ensure to add your professor as collaborator (ie. **Sergio.Santilli@georgebrown.ca**)
- A docker compose file to launch the entire system.
    - Make sure all components of your solution are containerized in docker.
    - I will be running your docker compose file to test your system **exclusively**. I will **NOT** run your solution environment in any other way (not using intellij), so you must make sure your docker compose brings up your **entire** working environment correctly.
    - Please note: If I cannot run your docker compose, you will **lose** marks
- You **must** demonstrate in your video recording, both running and brining up your docker-compose environment. This is the principal environment, and only environment that should be utilized during the demonstration.
- A Postman export collection, for each microservices tests, to showcase and test the various endpoints of your application. Each collection should be committed to your code repository.
- A brief report explaining the architecture, challenges faced, and lessons learned.
    - Include this in a parent folder of your project.
- **Please review the final page (below) for submission guidelines**

Evaluation Criteria:

- **Code Quality**: Clear, readable, and maintainable code.
- **Functionality**: Correct implementation of CRUD operations, communication between services, and room booking functionality.
- **Inter-Service Communication**: Reliable and accurate communication between microservices.
- **Testing**: Comprehensive integration tests with TestContainers.
- **Containerization**: Proper use of Docker and Docker Compose.
- **Documentation**: Well-documented code and a clear, concise report.

Final Word on Implementation

This information is detailed enough for you to complete the assignment. However, you'll need to use your best judgment on how to implement some of the requirement details. To do that, write the code, in such a way that you think is best.

For some requirements, the specifications may require you and your team to perform a certain amount of investigation and **research.** **Please note this is intended**. For example, some requirements will require to think about data persistence, CRUD operations etc., others may require you to think about data organization or storage, and yet others further still, may simply require you to learn a new library or concept, even if, those new concepts/libraries have not been formally covered in the course. I'm interested to see how your group approaches and solves problems.

Lastly, a requirement's document is used only to convey a perception of what is desired, but each developer is free to interpret differently, so long as the underlying requirements are met. Originality is always welcome, and always encouraged, and marks are often awarded accordingly.

Assignment Submission Guidelines:

1. Video Requirement
   a. Create a Short Video presentation. Your presentation should start with an introduction, where it must display a PowerPoint (or Google Presentation), that is 1 (single) slide. The slide introduces each member of your group, again, at the very start of your video.

   b. The first (and only) slide of your presentation must include current images of you and your partner(s) (no avatars allowed) that are displayed appropriately. You must also include your Full Names, Student IDs, the Course Code, Course Name, Course Section, and your Assignment information.

   c. Within the recording, you or your partner(s) will take turns demonstrating your program's functionality. You must show your solution working properly. You will also construct an assignment status report, a single page checklist/report. Use the report during the video, to facilitate communication confirming where requirements where successfully implemented and/or where requirements failed to be implemented and why.

   d. You and your partner(s) share the responsibility to demonstrate your functioning solution. Please note, the entire objective of the video is to demonstrate the functionality of your solution, this even more so then explaining your code. You will likely need to use Postman to demonstrate your services. When demonstrating your solution, I want to only see your docker compose environment running, that is, the environment that is demonstrated and tested, is running via your completely containerized environment.

   e. You and your partner(s) will each share the responsibility in describing the code in your solution that drives the functionality of your program – you will want to do this part well and be very clear. Be intelligent/selective on what code segments you describe; I do not need to know how every line of code works so keep this somewhat minimal.

   f. Sound for your video must at an appropriate level so that your voices may be clearly heard, and your screen resolution should be set so that your program's code and console details are clearly visible. In short, QA your videos. If your video is poor, assignment failures can/will be assigned.

   g. Your video should run no more than **~5-10** minutes. If you exceed this time, I simply will **not** be able to watch them... resulting in a grade of **zero**.

2. The **1 team lead**, must submit the following components to **Brightspace** on behalf of the entire group:
   a. The 1-page status report – **mandatory**
   b. The assignment video file - **mandatory**
      i. You may find Vento useful to create this
   c. The URL to the private GitHub repository (copy and paste the Brightspace as part of the submission)

3. Be cautious **DO NOT** share your application with others. Complete failures will be assigned if code is shared. All assignments will be reviewed and analyzed strictly within these regards.

4. Late assignments are assigned a penalty as described below:
   a. -20% per day for a maximum of 5 days.

Good luck, and remember to enjoy the process!