COMP 301 – Project Description

Faculty of Computer and Information Sciences
Department of Software Development
Fall 2025

Course: Software Architectures and Tools

Project Title: Building a Microservices-Based Web Application Using Spring Boot

Overview

In this project, you will embark on an immersive journey into the **world of Microservices Architecture**, leveraging the power of **Spring Boot** to design and develop a **scalable, resilient, and modular web application**.

This hands-on experience will enable you to understand the **principles of modern software architectures**, implement distributed systems, and finally **deploy your project to the Render cloud platform**.

Project Objective

The main objective of this project is to **design, implement, test, and deploy** a microservices-based web application.
Each microservice will perform a **distinct business function**, communicating through **RESTful APIs** to achieve full system functionality.

Microservices encourage:

- Loose coupling between components
- Independent deployment and scaling
- Improved maintainability and fault isolation

Technologies and Tools

Category	Recommended Tools
Backend Framework	**Spring Boot**
API Design	**RESTful APIs**
Databases	**MySQL**, **PostgreSQL**, or **MongoDB**
Testing	**JUnit**, **Mockito**
Containerization	**Docker**
Deployment	**Render Cloud Platform**

💡 **Tip:** Choose tools that align with your microservice's requirements. For example, use MongoDB for an event catalog service and PostgreSQL for user accounts.

⚙️ Project Phases

1. Design Phase

- Analyze business requirements.
- Identify individual components and define their responsibilities as microservices.
- Design communication patterns (e.g., RESTful APIs) and data flows between services.
- Prepare architecture diagrams and a high-level data model.

> 💡 **Deliverable:** Submit your **System Design Document** including architecture diagrams, API specifications, and database schemas.

---

2. Implementation Phase

- Develop microservices using **Spring Boot** ensuring **clear separation of concerns**.
- Implement RESTful endpoints for each service.
- Use **Spring Data JPA** or **Spring Data MongoDB** for database access.
- Apply **Spring Security** for authentication and authorization.

> ⚙️ **Task Example:**
> Implement a `UserService` microservice that supports registration, login, and role-based access.

---

3. Integration Phase

- Connect all microservices to enable inter-service communication.
- Implement **service discovery** using **Spring Cloud Netflix Eureka**.
- Handle configuration management using **Spring Cloud Config**.
- Ensure secure communication using **JWT** or **OAuth2** tokens.

> 💡 **Tip:** Use Postman or cURL to verify that inter-service APIs are functioning correctly.

---

4. Testing and Validation

- Write **unit tests** for each microservice using **JUnit** and **Mockito**.
- Conduct **integration tests** to validate service communication.
- Perform **end-to-end testing** on the entire system.

> 📦 **Deliverable:** A testing report including screenshots, test results, and coverage summary.

---

5. Deployment Phase

All projects must be **deployed to Render Cloud Platform**.
A separate document titled **"Spring Boot Deployment Guide for Render"** provides detailed step-by-step

instructions for building, configuring, and deploying your application.

> 🚀 **Deliverable:** Provide the **Render service URL** in your final submission package.

---

6. Documentation and Presentation

Prepare comprehensive documentation including:

- Architectural diagrams and API documentation
- Database schemas and deployment configurations
- Testing procedures and results
- Lessons learned during development

> 🎤 **Final Presentation:** Each team will demonstrate their deployed application on Render, explaining architectural decisions, challenges faced, and how they were resolved.

🏘️ Example Scenario – EventPlanner Application

Design Phase

Scenario:
Design an EventPlanner application that allows users to browse, organize, and register for events. The system should handle event listings, user management, booking, and payment operations.

Identify Microservices:

- **Event Catalog Service:** Manages events, including title, category, date, location, and capacity.
- **User Service:** Handles user registration, authentication, and profile management.
- **Booking Service:** Manages ticket booking, cancellations, and seat availability.
- **Payment Service:** Handles payments and stores transaction history.

Architectural Design:

- Define communication protocols (RESTful APIs) between microservices.
- Decide on database technology for each service (e.g., PostgreSQL for User, MongoDB for Events).
- Plan inter-service interactions, considering service discovery, load balancing, and asynchronous messaging if necessary.

Implementation Phase

- `EventCatalogService`: Provides APIs to add, list, and update events.
- `UserService`: Manages user accounts and authentication using **Spring Security**.
- `BookingService`: Handles user bookings, ensures seat availability, and updates event status.
- `PaymentService`: Simulates or integrates payment handling with basic validation and transaction tracking.

> ⚙️ **Hint:** Make sure each service exposes its REST endpoints under a distinct URL path and uses appropriate HTTP status codes.

Integration Phase

- Use **Spring Cloud Eureka** for service discovery and registration.
- Configure secure communication between services using **JWT** tokens.
- Use **Spring Cloud Config Server** for centralized configuration management.

> 💡 **Tip:** Test each service independently before integrating to avoid cascading errors.

Testing and Validation

Perform tests on:

- **Event Creation and Listing** (EventCatalogService)
- **User Authentication** (UserService)
- **Booking Confirmation and Cancellation** (BookingService)
- **Payment Transactions** (PaymentService)

> 📦 **Deliverable:** Submit a test report with success criteria and screenshots showing service interactions.

Deployment

Each microservice should be **deployed as an independent Render Web Service**.
Use appropriate environment variables for database URLs and secrets.

> 📄 Refer to the **"Spring Boot Deployment Guide for Render"** for full deployment instructions.

🧩 Expected Outcomes

Students are expected to deliver:

- A **fully functional** microservices-based EventPlanner web application.
- A **Render-deployed system** accessible through a public URL.
- A **documentation package** (design, implementation, testing).
- A **final presentation** demonstrating functionality, scalability, and resilience.

🔗 Helpful Resources

- Spring Boot Documentation
- Spring Guides
- Baeldung Tutorials
- Render Deployment Guide
- Docker Documentation

🏁 Good Luck!

Design, build, and deploy with confidence — let your **EventPlanner** microservices shine on Render!