**Project Report: GBC Event Booking System**

**Architecture Overview:**

The GBC Event Booking System is designed using a microservices architecture for scalability and fault tolerance. Each service, such as Room Booking, Event Registration, and Approval, is encapsulated in its own microservice. Communication is managed through an **API Gateway** built with Spring Cloud Gateway, ensuring centralized routing.

* **Keycloak** is integrated for secure authentication and authorization.
* **Kafka** facilitates asynchronous communication between services. **Schema Registry** is used to validate Kafka message formats and ensure compatibility.
* **Resilience4J** is implemented for handling service failures using the **Circuit Breaker** pattern, providing resilience during inter-service communication.
* **Swagger** is integrated to document all RESTful APIs.

**Challenges Faced:**

1. **Role-Based Access with Keycloak**: Configuring Keycloak for role-based access control was one of the most challenging parts. It involved properly linking Keycloak roles to access levels for various services through the API Gateway. Despite extensive documentation and tutorials, there were issues with syncing Keycloak roles between services, especially ensuring that only authorized staff members could access the **ApprovalService**. This required a deep dive into Keycloak’s configurations and extensive trial-and-error and still wasn’t able to configure it.
2. **Schema Registry with Docker**: Setting up **Schema Registry** when running the services in Docker. The issue was primarily with the **shared-schema**. I was able to setup the files correctly but couldn’t solve the error while containerizing booking and event-service that shared-schema couldn’t be found

**Lessons Learned:**

1. **Keycloak-authentication**: Was able to authenticate users
2. **Schema Registry**
3. **Asynchronous Messaging with Kafka Needs Rigor**: Kafka-based asynchronous communication is powerful but requires careful handling of schemas and event consumption. Schema Registry is indispensable for ensuring compatibility and preventing data corruption or loss.
4. **Resilience Through Circuit Breakers**: **Resilience4J’s** Circuit Breaker pattern was crucial for service resilience. Implementing fallback mechanisms helped in avoiding cascading failures during service downtimes, providing a seamless user experience.