**Project Report: GBC Event Booking System**  
**Group Number**: 47

**Overview of the System Architecture**:  
The GBC Event Booking System adopts a microservices architecture to provide scalability and high availability. Each core functionality, such as Room Booking, Event Management, and Approval, operates as a distinct microservice. Communication and routing are managed through an API Gateway powered by Spring Cloud Gateway, ensuring streamlined and centralized traffic management.

* **Authentication & Authorization**: Keycloak has been integrated to secure the system, enabling both authentication and authorization.
* **Event Communication**: Kafka handles asynchronous messaging between services, while Schema Registry ensures message compatibility and consistency across microservices.
* **Resilience Mechanism**: Resilience4J’s Circuit Breaker design protects the system against failures in service-to-service communication.
* **API Documentation**: Swagger provides comprehensive API documentation for each microservice, enhancing accessibility and usability for developers.

**Challenges Faced During Development**:

1. **Role-Based Access Control (RBAC) via Keycloak**:  
   Setting up Keycloak for role-specific access was a significant challenge. Defining roles and ensuring that they were correctly enforced across different services, particularly the ApprovalService, was problematic. The configuration of roles and permissions through the API Gateway required considerable testing and debugging. Despite multiple attempts, syncing the roles across services remained incomplete due to token management issues.
2. **Schema Registry in a Dockerized Environment**:  
   Configuring the Schema Registry while running services in Docker was complex. The main problem was resolving the “shared-schema not found” error when containerizing the Booking and Event microservices. Although the schema files were properly configured, the issue persisted, halting the integration process.

**Key Takeaways**:

1. **Keycloak Authentication Success**:  
   Successfully implemented user authentication using Keycloak, providing secure access control for all services. This setup involved a detailed understanding of Keycloak’s functionality and configurations.
2. **Importance of Schema Registry**:  
   Learned that the Schema Registry is vital for managing message schemas in a Kafka environment. It ensures compatibility between producers and consumers, preventing issues related to mismatched data formats.
3. **Asynchronous Messaging with Kafka**:  
   Kafka’s asynchronous communication system is highly effective but requires meticulous planning, especially when managing schemas. Ensuring compatibility between services is critical for reliable message exchange.
4. **Resilience Through Circuit Breakers**:  
   The Circuit Breaker pattern implemented using Resilience4J proved invaluable in building a fault-tolerant system. Fallback mechanisms ensured service continuity during failures, preventing disruptions from cascading across the system.