**Food Delivery System**

**API-Based Products (S1-24\_SEZG504)**

[**2023mt93202@wilp.bits-pilani.ac.in**](mailto:2023mt93202@wilp.bits-pilani.ac.in) **(NAKUL KUMAR)**

[**2023mt93333@wilp.bits-pilani.ac.in**](mailto:2023mt93333@wilp.bits-pilani.ac.in) **(ISHAN JAIN)**

[**2023mt93028@wilp.bits-pilani.ac.in**](mailto:2023mt93028@wilp.bits-pilani.ac.in) **(SWEETY)**

**1. Overview and Key Design Choices**

The API product designed for the food delivery system aims to cater to various user roles, including customers, restaurant owners, delivery personnel, and administrators. Each role has a set of functionalities tailored to their interactions with the platform. This report outlines the key design choices, role-based access implementation, and data security measures embedded in the API.

**Key Architectural Decisions:**

**Framework Selection:** The project was developed using Java and Spring Boot framework due to its scalability, ease of integration with third-party libraries, and strong support for RESTful design.

**Database:** MySQL was chosen as data is structured, making it ideal for content like menus, orders and payments etc. Also, ACID (Atomicity, Consistency, Isolation and Durability) is guaranteed with MySQL.

**Authentication:** Implemented JWT (JSON Web Token) for secure, token-based authentication across all endpoints.

**API Style:** A RESTful approach was adopted to ensure stateless interactions, making the API scalable and suitable for integration with various client-side applications if required.

**2. Role-Based Access Control (RBAC) Implementation**

To ensure that only authorized users access specific functionalities, RBAC was implemented. Each role (Customer, Restaurant Owner, Delivery Personnel, Administrator) has unique permissions defined as follows:

**Customer:** Can browse restaurants, search menus, place and track orders, view past orders, and manage account details.

**Restaurant Owner:** Can manage their menu, update restaurant details, view and manage orders.

**Delivery Personnel:** Can view and accept delivery requests, update the status of deliveries, and manage their availability.

**Administrator:** Has comprehensive access to manage user accounts, monitor orders, generate reports, and oversee platform activity.

**Implementation Strategy:**

Each user role is assigned a set of permissions stored in the database.

Middleware checks user roles and permissions before granting access to protected endpoints.

Role-based checks are integrated using Express middleware, ensuring modular and secure access control.

**3. Data Security Measures**

Security is a primary concern, especially when handling sensitive information like personal user data and accessing restricted resources. The following measures were implemented:

**Encryption:** Passwords are stored securely using bcrypt hashing.

**Authentication & Authorization:** JWT tokens are used for stateless authentication, with tokens stored client-side to prevent unauthorized access.

**Validation:** User input is rigorously validated at each endpoint to mitigate injection attacks, ensure data integrity, and prevent application crashes.

**CORS Management:** Configured CORS policies to allow controlled cross-origin requests from trusted web and mobile clients, enhancing security while maintaining usability.

**4. API Documentation**

Comprehensive documentation was created using Swagger. Each endpoint is detailed with:

**Request Methods:** GET, POST, PUT, DELETE.

**Parameters:** Path, query, and body parameters.

**Response Codes:** Success (200, 201), Client errors (400, 401, 404), Server errors (500).

**Error Messages:** Custom error messages help developers identify and debug issues.

**5. Testing and Deployment**

**Testing Strategy:**

**Unit Tests:** JUnit and Mockito were used for backend unit tests, covering core functions like user registration, order placement, and menu management.

**Coverage:** Jacoco plugin is used for test cases to ensure code coverage. Tests verified the functionality of endpoints, including role-based restrictions and data validation.

**Manual Testing:** Postman was used for manual testing to simulate real user interactions and ensure expected responses.

**Deployment:** The API was deployed on AWS for easy accessibility, with an endpoint URL provided for testing purposes.

**6. Database Schema**

The database schema was designed with separate tables for Users, Restaurants, Menus, Orders, and Deliveries:

User table: Stores user profiles, roles, and authentication data.

Restaurant table: Contains restaurant details, associated menus, delivery zones and opening hours.

Order table: Includes order details, customer information, and status updates.

Schema has been added in the github Readme.md at below location:

https://github.com/nakulkumarbits/apib-assignment/blob/master/fooddeliverysystem/Readme.md#database-schema

**Conclusion**

The API product designed and implemented for the food delivery system emphasizes robust functionality across user roles, security through RBAC and JWT authentication, and comprehensive testing. Its modular design promotes code maintainability, scalability, and seamless deployment, ensuring a reliable experience for all platform users. Comprehensive documentation provides transparency and ease of integration for developers and stakeholders.