Delivery App Simulator

Acknowledgement

We would like to express our sincere gratitude to everyone who supported us throughout the duration of this project. As a team of two, we especially acknowledge each other Lakshmi Priya, Khajabi for our dedication, collaboration, and joint contributions toward the successfully completion of this project.

Project Guide

V. Sai Chandana

FAANG CEO

J.Venkat

Technology Stack: Java, Spring Boot, Maven, Rest APIs, Git

INTRODUCTION:

The **Delivery App Simulator** is a Spring Boot–based application designed to emulate the core functionality of a real-world delivery platform. It enables customers to place orders, track delivery status, and view order history while ensuring accurate order processing and delivery management.

The system leverages **RESTful APIs** for modular and scalable backend design, enabling easy integration with frontend applications or third-party services. For data persistence, it supports **MongoDB** or relational databases, storing detailed order and delivery information for quick retrieval and historical analysis.

For development workflow, the project uses **Git for version control**, allowing collaborative development, code tracking, and rollback if necessary. **Postman collections** are included for API testing, ensuring that endpoints function correctly and meet expected requirements.

OBJECTIVES:

Customer Order Management:

Enable customers to easily place, update, and cancel delivery orders through a user-friendly interface. The system ensures smooth order handling and efficient processing.

Real-Time Delivery Tracking:

Provide live updates on the status of deliveries(PENDING, IN_TRANSIT, DELIVERED) so that customers and administrators can monitor orders at any time.

Order and Delivery History:

Maintain a comprehensive history of all orders and deliveries, allowing users to review past transactions and track repeat orders.

Modular API Design with Spring Boot:

Use Spring Boot to build modular, scalable, and maintainable APIs. The architecture follows clean coding principles and ensures easy integration with other systems.

Development and Testing Support:

Leverage Git for version control and Postman collections for testing APIs. This ensures collaborative development, efficient debugging, and reliable deployment.

KEY FEATURES:

- **Order Management** – Place, update, cancel, and view orders.
- **Delivery Tracking** – Track delivery status in real time.
- **Item Management** – Add, update, and fetch delivery items with pricing.
- **Order History** – Retrieve complete order history for any customer.
- **Modular API Design** – Built with Spring Boot REST APIs following OOP principles.
- **Database Persistence** – Store and retrieve orders, deliveries, and items reliably.
- **Development Tools** – Uses Maven for build lifecycle, Git for version control, and Postman for testing.

TOOLS AND TECHNOLOGIES:

SpringBoot :

Used for building the RESTful backend API with modular, scalable architecture.

MongoDB / Relational Database:

Stores order, delivery, and user information with efficient retrieval and query support.

Postman:

For API testing and validation of endpoints, including GET, PUT, and POST requests.

Git :

Version control system for code management,collaboration, and maintaining project history.

Maven / Gradle :

Build tools for dependency management and project automation.

Spring Data JPA / Spring Data MongoDB:

Simplifies data access and repository implementation.

Output Format:

Items List:

1.POST Request (Add New Item):

- **Method**: POST
- **URL**: http://localhost:8080/items/add

2. GET Request (Retrieve All Items)

- **Method**: GET
- **URL**: http://localhost:8080/items

3. PUT Request (Update Existing Item)

- **Method**: PUT
- **URL**: http://localhost:8080/items/{id}

Order List:

1.POST Request (Add New Item):

- **Method**: POST
- **URL**: http://localhost:8080/orders/adds

2.Get Order by Id

- **Method**: POST
- **URL**: http://localhost:8080/orders/{id}

3.Get Order History by User

- **Method:** Get
- **URL:** http://localhost:8080/orders/history/{userId}

4.Update Order

- **Method:** put
- **URL:** http://localhost:8080/orders/{id}

5.Delete Order

- **Method:** Delete
- **URL:** http://localhost:8080/orders/{id}

6.Get Delivery Status

- **Method:** GET
- **URL:** http://localhost:8080/delivery/{orderId}/status

7.Update Delivery Status

- **Method:** PUT
- **URL:** http://localhost:8080/delivery/{orderId}/update

8.Cancel Order:

- **Method:**PUT
- **URL:** http://localhost:8080/delivery/{orderId}/cancel

9.Update Delivery Status

- **Method:** PUT
- **URL:** http://localhost:8080/delivery/{orderId}/update

10.Cancel Order:

- **Method:** PUT
- **URL:** http://localhost:8080/delivery/{orderId}/cancel