LiftDrop

Mobile App for Deliveries

Gonçalo Morais, n.º 49502, e-mail: a49502@alunos.isel.pt, tel.: 927468061
João Ramos, n.º 49424, e-mail: a49424@alunos.isel.pt, tel.: 919222551

Orientador: Miguel Gamboa, e-mail: miguel.gamboa@isel.pt

Co-Orientador: Diogo Silva, e-mail: diogo.silva@lyzer.tech

March of 2025

1 Resume

The rise of the **gig economy** has transformed various industries, particularly the food and package delivery sector. This economic model relies on flexible short-term work arrangements, often facilitated through digital platforms. Companies such as Uber Eats, Glovo, and Bolt Food have leveraged this framework to revolutionize urban logistics, enabling customers to order food and services on demand while offering couriers independent work opportunities.

Although these platforms prioritize customer convenience, they also present significant challenges for couriers. High commission fees, lack of job stability, and inefficiencies in delivery logistics create an imbalance between platform profitability and worker sustainability. In addition, customers often experience delays and inconsistent service quality.

To address these challenges, LiftDrop proposes a new approach to the gig economy by offering a more fair, efficient, and transparent delivery solution. This mobile app aims to improve courier conditions while improving the customer experience through optimized delivery processes and fairer economic policies.

2 Introduction

3 Scope

Developing a mobile application focused on delivery presents challenges, particularly in real-time data handling and order assignment. Since multiple couriers interact with the platform simultaneously, the system must manage updates effectively while ensuring a smooth user experience. After some research and consideration, we identified two primary approaches to handling real-time updates and concurrency:

- **Polling-based Request-Response Model** – The client periodically requests updates from the server. While simple to implement, this approach results in unnecessary network requests, increased server load, and delayed updates.
- **WebSockets** – A bidirectional communication protocol that allows both the client and server to send messages dynamically. While WebSockets are ideal for interactive, two-way communication (e.g., chat applications), SSE offers a simpler and more reliable approach for our needs, focusing on server-driven updates.
- **Event-Driven Server-Sent Events (SSE)** – The server pushes updates to clients as they occur, enabling real-time communication without the inefficiencies of constant polling. This is well-suited for unidirectional data flow (server-to-client updates).

Given the requirements of this project, we chose **SSE** as the preferred method for real-time updates due to its simplicity, maintainability, and efficient handling of one-way event streams. To implement SSE on Android, we will use OkHttp since Android lacks native SSE support.

For order assignment, the platform will consider factors like **Proximity-based allocation**, **Fair distribution**, and **Traffic-aware routing**. An external geospatial API will handle location-based calculations (distance, routing, and traffic), simplifying the implementation by reducing the need for complex in-house geospatial logic.

Several geospatial APIs can provide direct distance calculations between a courier and a pickup location. We chose the **Google Maps API** as it offers the most suitable free plan for the required functionalities.

To enhance safety, the platform will feature **neighborhood safety ratings** based on feedback from couriers who have completed deliveries in those areas. Only couriers can submit ratings, ensuring assessments reflect real delivery conditions.
Couriers will receive alerts for high-risk areas, and **nighttime safety measures** will highlight risk-prone locations for late-night deliveries.

4 Features

The mobile application will have the following features:

4.1 Client Features

1. **Client orders something:**
    - Request details:
        - ClientID
        - Restaurant
        - Order
2. **Client checks order status:**
    - Request details:
        - ClientID
3. **Client gets ETA of the order:**
    - Request details:
        - ClientID

4.2 Courier Features

1. **Courier accepts order**
    - Request details:
        - CourierID
        - OrderID
2. **Courier declines order**
    - Request details:
        - CourierID
        - OrderID
3. **Courier starts waiting for orders:**
    - Request details:
        - CourierID
4. **Courier stops waiting for orders:**
    - Request details:
        - CourierID
5. **Courier cancels order in progress:**
    - Request details:
        - OrderID
6. **Update order status:**

- Request details:
  - OrderID

7. **Courier confirms delivery:**

   - Request details:
     - CourierID
     - OrderID

4.3 User Features

1. **User registers as a client:**

   - Request details:
     - Name
     - Address
     - Phone number
     - Email
     - Password

2. **User registers as a courier:**

   - Request details:
     - Name
     - Phone number
     - Email
     - Password
     - Means of transportation