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

Março de 2025

Introduction

Contextualization

The rise of the **gig economy** has transformed various industries, particularly in the food delivery sector. This economic model is characterized by short-term, flexible work arrangements where individuals take on temporary jobs, 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 on demand while providing couriers with independent work opportunities.

By prioritizing **customer convenience**, these platforms have successfully streamlined food delivery processes through real-time technology, ensuring efficiency and reliability at scale. However, as demand for delivery services continues to grow, the gig economy also presents operational challenges, particularly affecting the couriers who sustain this system.

Project Goals

This project aims to develop a delivery-focused Android mobile application that assists couriers in handling deliveries efficiently while ensuring a seamless working experience. To support the app's functionality, we'll be implementing two separate web APIs:

- **Courier API** - A backend service that powers the mobile app, handling real-time courier updates, order assignments, and safety features.
- **Client Simulation API** - A backend service designed to simulate client orders.

Leveraging this architecture, we aim to enhance the delivery experience with the following key objectives:

- **Authentication and user management**, supporting secure courier and client registration with token-based authentication.
- **Real-time demand insights** through heat maps showing high-order density areas to help couriers position themselves effectively.
- **New order pop-ups**, sending real-time notifications to couriers when new delivery requests are available, allowing them to accept or decline orders.
- **Optimized order assignment** based on proximity, fairness, and traffic conditions.
- **Courier safety measures**, including a neighborhood safety rating system that alerts couriers about high-risk locations and nighttime safety measures highlighting risk-prone areas for late-night deliveries.
- **Earnings tracking** (optional), providing insights into income and performance trends.

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.

Architecture and Tech Stack

The system architecture will consist of the following components:

- **Application for Android**: Implemented using Jetpack Compose. The app is planned for deployment on the Google Play Store.
- **Backend**: Developed using Kotlin with Spring MVC, handling business logic, real-time interactions, and data management. Initially deployed using Docker on Render.
- **PostgreSQL Database**: Stores connections between entities along with additional order and user-related data.

The chosen tech stack was selected to meet the needs of a **courier-focused delivery app**. **Jetpack Compose** simplifies Android development with a modern UI framework, while **Kotlin with Spring MVC** ensures a structured and maintainable backend for handling RESTful services and real-time updates.

PostgreSQL was chosen for its **geospatial query support**, enabling efficient location-based order assignments. The backend is containerized with **Docker** and deployed on **Render** to facilitate **scalability and cloud-based deployment**.

Challenges Ahead

Although our project aims to improve the courier experience through innovative features, several challenges must be addressed during development and deployment.

- **Real-Time Data Synchronization**: Ensure consistent and reliable updates between couriers, the server, and the application without compromising performance or data integrity.
- **Scalability**: As the user base grows, maintain system performance and responsiveness while scaling up server resources and database capabilities.
- **User Interface Design**: Design an intuitive and efficient user interface using Jetpack Compose that meets the diverse needs of both couriers and administrators.

- **Integration Complexity**: Integrating third-party APIs for geospatial data and safety ratings while maintaining system stability and reliability.
- **Testing and Validation**: Rigorously testing the application using automatic tests.

Addressing these challenges proactively will be crucial to delivering a robust and effective mobile application that meets the demands of both couriers and users alike.

Project Plan

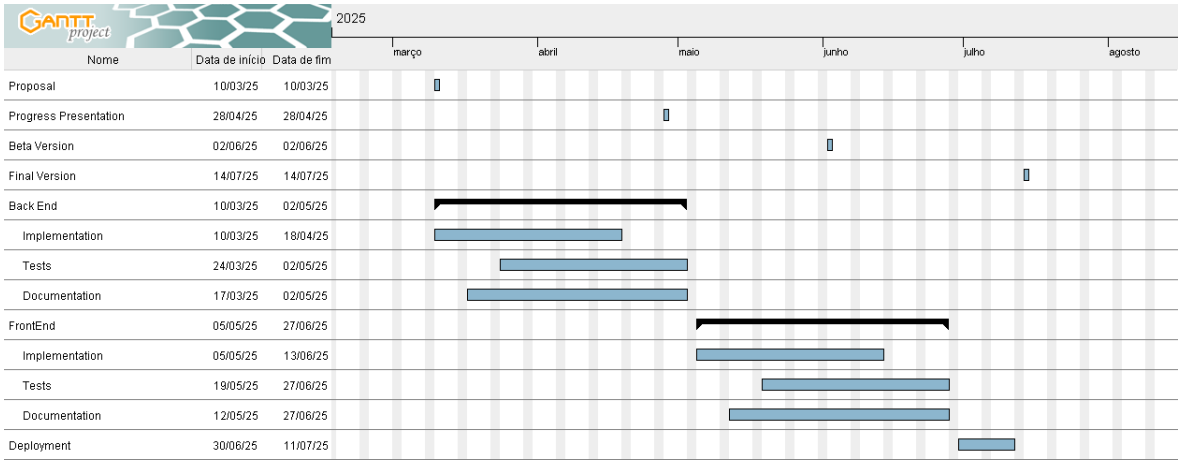

Figure 1: Project Plan

References

[1] Spring Framework. `https://spring.io/`.

[2] Kotlin Programming Language. `https://kotlinlang.org/`.

[3] PostgreSQL Database. `https://www.postgresql.org/`.

[4] Jetpack Compose. `https://developer.android.com/compose`.

[5] WebSockets. `https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API`.

[6] Server-Sent Events (SSE). `https://developer.mozilla.org/en-US/docs/Web/API/Server-sent_events/Using_server-sent_events`.

[7] Google Maps API. `https://developers.google.com/maps`.

[8] Docker Deployment on Render. `https://render.com/docs/deploy-docker`.