

**Design and implementation of Mobile Applications**

**Delevery**

**Design Document**

**1 Project Description**

**Delevery**

****

**Description:**

Delevery is a centralized delivery mobile application that supports vendors who run small businesses, through a delivery system that links drivers who can be dedicated drivers or just working part-time for some extra cash.

The system utilizes an integrated process that links both drivers and sellers in complete different user interfaces to link a driver with a seller to handle a delivery. The novelty here is that the delivery does not have to be a single item; to save time and money on both ends, this allows the seller to bundle orders together and just set a delivery address for each customer, and the driver only needs to pick up the packages and the rest is handled by the application.

The application uses a smart algorithm based on a private Geo-server that calculates the distance between the driver, the customers, and the return path; to optimize the route that the driver must take in order to finish the order.

**2 Requirements Analysis and Specification**

**2.1 Project overview**

The objective of this application is to provide small-scale vendors with small, growing business, a platform that ensures reliable delivery of their orders to the customers. The deliveries must be in sequence and done in the most optimized way possible, exploiting external services like Google Maps. MapBox and other geolocation services to do that.

The project will have a cloud based NoSQL database using Google’s Firebase, the database includes services that will be used such as Firebase authentication to keep track of registered sellers and drivers, it will also use the Firebase Firestore cloud service to store the information about the orders, clients, and much more, and finally it will also use Firebase’s cloud storage to store pictures and files.

The application will be developed in Flutter framework; Flutter is an open-source mobile application development framework created by Google. It uses the Dart programming language and provides a reactive programming model for building high-performance, high-fidelity apps for both iOS and Android. Flutter's widgets incorporate all critical platform differences such as scrolling, icons, fonts, and navigation to provide full native performance on both iOS and Android. It also includes a rich set of customizable widgets, which can be used to compose high-quality native interfaces, and it also supports Hot Reload which allows developers to experiment, build UIs, add features, and fix bugs faster. Overall, Flutter is a powerful tool for building mobile apps that have a fast development cycle, expressive and flexible UI, and high performance.

Finally, the project will support 2 different user experiences, based on the type of the user, the different types will be seller and driver profiles, which will also have different registration procedures, and different user interfaces and functionalities.

**2.2 Project goals**

The online business is currently dominated by the big companies that offer high-end services in customer service and delivery, the goal of this project is to also allow small business owners to contribute to the market even though they don’t have the resources necessary to build a private delivery system.

**2.3 Domain Analysis**

**2.3.1 Domain entities**

a. Users

* Sellers
* Buyers

b. External services

* Maps
* Geoservices and locators
* Cloud-based database
* Tracking and order management system

**2.3.2 Domain analysis table**

Table (2.1): Domain analysis table

|  |  |  |
| --- | --- | --- |
| Phenomena | Location | Controlled by |
| User receives orders on his  Platform  User opens the application  User creates a seller account  User inputs his information  User fills the data of the orders  User finishes the orders  User publishes the orders  The orders are georeferenced and  Their location is stored in the  Database  The orders are stored in the database and  Are linked to the user  The user’s profile is saved into the database  The application recognized the user’s  Location provided by him  The location of the user is transformed  Is transformed into latitude and  Longitude and is stored in the  Database  The user’s orders are displayed to drivers  Who reside in the same area  The drivers can view the orders on  Their screen and can accept the orders  The orders are assigned to the drivers  The database links the orders of the seller with  The diver  The database links the seller with the driver  The driver can see that he has been  Assigned with the orders and can  See their details  The application prompts the driver to  Pick up the orders from the seller  The driver is shown the seller’s location in  Order to pick up his orders  The driver picks up the orders  The application updates the driver’s UI  The driver now can see the orders and  The locations he has to deliver to  The application optimizes the paths based on  The distances and displays the suggestion to  The driver  The driver proceeds to deliver the orders to the  Customers  The driver has to take a picture of the delivery  That he made  The system pushes a notification to the seller  That a delivery has been made  The seller can view on a map what orders have  Been delivered  The driver can visualize on the map which routes  He is advised to take  The driver completes all deliveries  The seller is notified that all deliveries have  Been made  The seller is prompted to review the driver  Performance  The driver performance is received and  His rating is adjusted based  On the feedback  The seller’s user interface is reset to  The initial page where he can then  Create new orders  The driver can accept new orders | World  Shared  Shared  Shared  Shared  Shared  Shared  Machine  Machine  Machine  Machine  Machine  Machine  Shared  Machine  Machine  Machine  Shared  Machine  Machine  World  Machine  Shared  Machine  World  Shared  Machine  Shared  Shared  World  Machine  Machine  Machine  Machine  Machine | World  World  World  World  World  World  World  Machine  Machine  Machine  Machine  Machine  Machine  World  Machine  Machine  Machine  Machine  Machine  Machine  World  Machine  Machine  Machine  World  World  Machine  Machine  Machine  World  Machine  Machine  Machine  Machine  World |
|  |  |  |

**2.3.3 User cases & scenarios**

In order to explain the software functionalities, this section is going to address an explanation about the actions taken by the software and the user in a list of cases that are useful to explain the internal processes of the application. In this section we describe what is going on from server-side and client-side on when the user cases happen by specifying the different actions that take place in these situations.

**Actors:**

1. **Sellers**
2. **Drivers**

## User Case 1: Normal User Registration

Use case name: NormalUserRegistration.

Actors: All actors.

For registration, the flow is:

1. User enters page of registration.
2. User fills the required data
3. Application inspects the rules of password and display the result.
4. Information mentioned above are sent to the server then stored in the database.
5. Alert the registration result, redirect to login page.

Exit condition: The user successfully registers.

## User Case 2: Registered User Login

Use case name: RegisteredUserLogin.

Actors: Registered Users.

For login, the flow is:

1. User enters page of login.
2. User fills the email and password.
3. Information mentioned above are sent to the server then check the correctness of username and password.

Exit condition: Alert the login result, redirect to main page

## User Case 3: Seller/Driver process

Use case name: SellerOrders.

Actors: Sellers.

Flow is:

1. Seller creates the requests and publishes them.
2. Driver can see the requests and accepts them.
3. The seller is notified that a driver is assigned to them.
4. The seller is notified that the driver is heading to them to pick up the orders
5. The sellers give the driver the orders.
6. The driver delivers the orders.
7. The seller is notified for each delivery.
8. The driver finishes the orders
9. The seller is notified that the driver has finished the deliveries.
10. The seller can restart the process.

Exit condition: The driver delivers all orders.

## User Case 4: Map process

Use case name: MapOrders.

Actors: All.

Flow is:

1. The map always is active.
2. If no orders are present, the map is empty.
3. If the orders are present, the map should display them.
4. The driver can see the optimized path.
5. The seller can see the delivered orders and the order that are not.
6. The map supports both actors.

Exit condition: The driver delivers all orders.

## User Case 5: Exception Handling

Use case name: ExceptionHandling

Actors: Can happen with all Actors.

Flow:

1. When an exception is detected, the program will catch it and enter the handling phase. Distinguish the exception type.
2. If this exception type was predicted in the program, run the corresponding exception handling function.

Exit condition: When the exception is handled, and user has been redirected to the right page with clear instruction of dealing with this exception.

* + 1. **Requirements**
       1. **Technical requirements**

1. The application will be delivered in Flutter.
2. The application has to have maps as a supported external service.
3. The application must support a geo-server to optimize paths.
4. The application will use Firebase as a backend.
5. The application will most likely work with internet connection only.
6. The application has to support 2 different UI’s
   * + 1. **Functional requirements:**
7. The application must support two authentication processes, to cope with the presence of both the sellers and the drivers.
8. The application must support a map for both types of users.
9. The application must support a navigation system for the driver that optimizes the distance based on the shortest route.
10. The application must be consistent when it’s handling the interactions between the sellers and the drivers.
11. The cloud based database must be able to support different types of registration, for example Google’s Gmail and Facebook login.
12. The users can have the option to upload pictures and change their data.
13. The drivers can see requests outside of their region.
14. The map must also support a view for the sellers to show them all the locations of the orders and in the case where they are in delivery, it must show which deliveries are delivered and what orders are under ye to be delivered.
15. The application must be transparent and ask for the user’s permissions to access their locations and their internal storage.
16. The application must support the user with a page that allows them to express their opinions and complaints.
17. The application must provide a terms and agreements page.
18. The application must support Android and IOS.

**3 Design process**

**3.1 Design introduction**

In this phase, the design process provides a specification on the architecture of our application system. In fact, this document is corresponding to the Requirements Analysis and Specification.

**3.2 Architectural design**

Based on the requirements posed in the previous section, certain design choices concerning the build of the application must be taken to allow the inception of the criteria made in the requirements.

The design choices facilitate a robust and comprehensive solution to the requirements of the application, some of the design choices are going to be discussed in this section regarding the interaction between the different components and their proposed implementation.

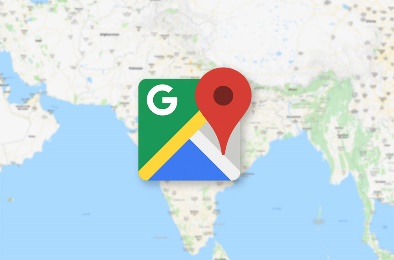


Figure (3.1): Framework, Database and services graph

**3.2.1 Database**

Since it is a technical requirement to use Flutter as the framework to build the application, the most obvious solution for the database was Firebase, since it is well supported in the Flutter framework, and offers a smooth integration process within the Dart programming language and great maintenance.

Firebase is a mobile and web application development platform created by Google. It provides a variety of tools and services for building and maintaining apps, including real-time databases, authentication, hosting, storage, and more. Firebase Real-time Database is a cloud-hosted NoSQL database that allows data to be stored and synchronized across multiple clients in real-time. Firebase Authentication enables developers to authenticate users with email and password, phone number and social media accounts like Google, Facebook, Twitter, and GitHub. Firebase Hosting provides a simple way to host and serve web assets, while Firebase Storage allows developers to easily upload and store files. Additionally, Firebase provides services like Cloud Firestore, Cloud Functions, Cloud Messaging, and Cloud Storage. It also provides machine learning services like ML Kit, Cloud Firestore and Firebase ML. Overall, Firebase is a comprehensive and powerful platform that makes it easy for developers to add backend services to their apps without having to build and maintain their own servers.

**3.2.2 External services**

For the different functionalities that the application must support, the use of external services is a must, such as the map functionality, the geo-locators and navigation, we decided to use Google Maps and Mapbox to this end.

Google Maps is a web mapping service developed by Google. It offers satellite imagery, street maps, street view, real-time traffic conditions, and route planning for traveling by car, foot, bicycle, or public transportation. It also provides information about millions of places, such as businesses, landmarks, and points of interest. Google Maps also offers an API that allows developers to integrate its functionality into their own websites and mobile apps.

Mapbox is an open-source mapping platform for developers. It allows developers to create and customize maps using their own data, styles, and functionality. Mapbox offers a variety of services such as maps, geocoding, routing, and turn-by-turn navigation. It also provides APIs and SDKs for both iOS and Android, allowing developers to easily add maps and location-based features to their mobile apps. Mapbox also provides a number of pre-built map styles which can be customized with different colors, fonts, and icons.

Overall, Both Google Maps and Mapbox are powerful tools for creating and integrating maps into web and mobile applications. Google Maps is more widely used and offers more features, while Mapbox is more customizable and allows developers to work with their own data and styles.

**3.2.3 Architecture and tools**

Flutter is an integrated framework, which means that everything occurs at the same time inside it within the same code base, this means that the front-end and the back-end are all designed using Dart and within the same project.

To this end, there are some conventions in Flutter to provide a multi-layered architecture within the same code base, while at the same time handling the state of the UI with the use of some famous Flutter tools like BLOC.

The idea is to make the system look like a typical client-server architecture, to this end we already have the UI which is developed using the Flutter framework, and the database is in Firebase.

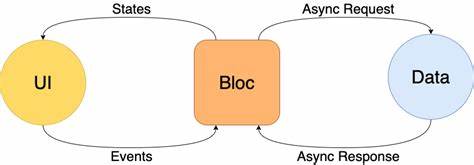
To handle their interaction and make the UI responsive to the changes in the database, a BLOC architecture is used to handle the state of the widgets. This is done very intuitively, where the widgets in flutter will react to the states they receive after they send out an event they’ve encountered in the UI, BLOC will react to the changes and translate the event-state couple to an async-request, async-response couple

Figure (3.2): The interaction between the UI and the database and BLOC

The implementation is actually quite intuitive and allows for a smooth design process, with needs the developer to code the events and the states, and within the event-state interaction, one must define the necessary actions “the final state” of the UI should look like, this is an example of state management techniques in Flutter.

**3.2.4 preliminary design and wire framing**

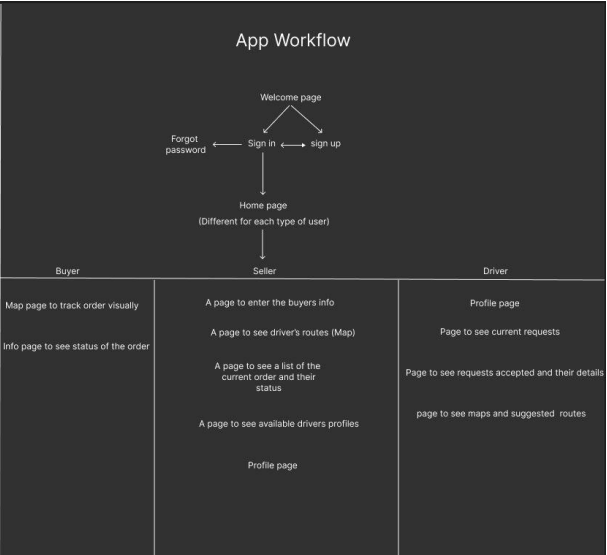
For the early stages of development, we were focused on how the app should roughly look like, and how it should go about the different UI’s for the seller and the driver. This section includes the initial workflow graphs and the initial wireframes presented below.

Figure (3.3): preliminary application component interaction

As can be seen in the above figure, the initial idea was to also include pseudo account for each individual customer constituting a third UI. However, this idea was scraped since it will also require the handling of their data and the creation of redundant accounts not necessary nor beneficial to the application’s lifecycle.

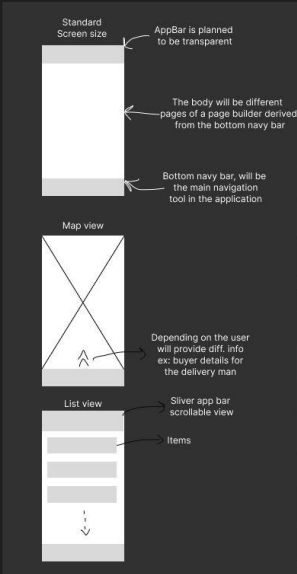


Figure (3.4): preliminary wire framing

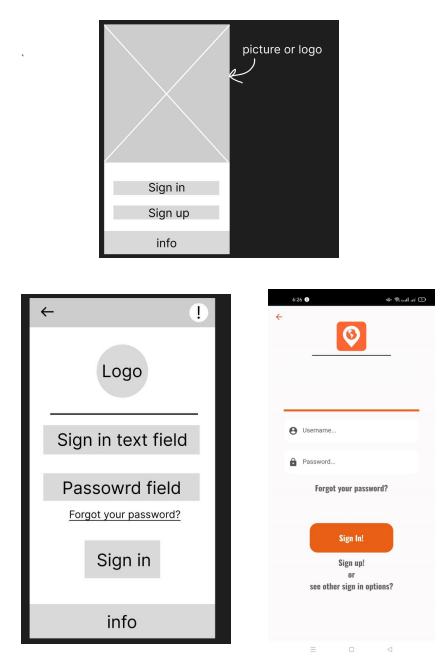


Figure (3.5): Additional preliminary wire framing and a test design

**3.2.5 Components and workflow**

Once the preliminary design stage was concluded, it was by then the time to actually design the components of the application and their interactions, this includes the full lifecycle of the application to facilitate the requirements and the use cases intended for the application.

The technologies, services, and packages used will also be discussed below.

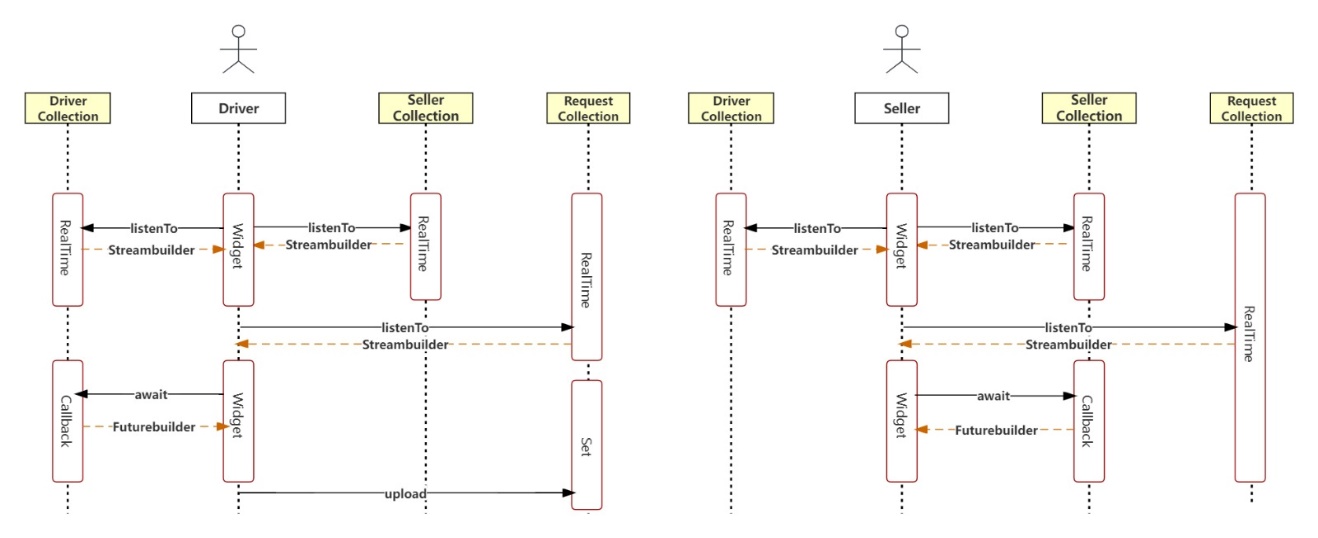


Figure (3.6): Logic design

As can be seen in figure 3.6, the lifecycle of the application is a shared interaction between the seller and the driver, while both have different UI’s and functionalities, they have to interact with each other to move along the flow of the delivery process.

The progression of the order start with the seller, where he posts his orders, where he can edit and delete orders, where he has to upload images of the orders, information regarding the customers, and the location of the delivery, which is supported by Google’s geo-locator, which ensures that the location abides the same naming conventions so that it can be fed later to the same locator to convert it to latitude and longitude.

After the seller is satisfied with the orders, he can publish them, which will make them appear for the nearby drivers.

From this point we will look at the driver part since the seller can no longer do anything except visualize the orders on the map.

For the driver from this point, he can visualize the orders posted by different sellers, and choose the order that fits him best, the orders’ location can be seen on a Google map, and the revenue that the driver will make.

Once the driver accepts the order, his UI will change and he will be assigned with the orders, but at this point the only option he has is that he has to take the orders from the seller, which will be apparent on the map.

Once he collects the orders from the seller and marks them collected, he can then visualize the optimized paths present on his map.

He can then start delivering the orders, to mark a delivery complete, the driver must take a picture to the delivered ordered in the site of delivery for Insurance purposes.

**5- Testing**

For the testing part, which is on our view the most important, we prepared different types of testing for the application, and since Flutter has great potential for testing such as unit testing and widget testing, we also arranged some test cases and ran through them in the final version of the application to make sure we had a stable bug-free version.

The first part of the testing which was the widget testing, which can also be seen as a unit test since it was focused on one field, which was the authentication process, since every other widget is either a UI widget or logic, they were going to be tested on the End-To-End (e2e) part of the testing process.

5.1 Widget & unit testing

To start off the testing environment in flutter, some packages have been installed which are fully maintained by flutter and offer great performance.

The widget tests were run on the authentication process, by creating mock authentication provider as can be seen below, where the auth provider is tested for each of the functionalities that it has to ensure that it is performing as intended and is robust enough.

the tests are comprised of 150 lines of tests in the test folder in the application directory, once they were they proved to all pass the tests, meaning that they work as intended.

Text

Description automatically generated

5.2 E2E testing & test cases.

In this part, the methodology will be to simulate the logical process of the application’s lifecycle and go through all the possible cases to guarantee that the application has no bugs or errors, the test cases will be run on Android and IOS mobile phones, and as a plus, we will also test on an Android tablet to ensure that the application is robust to bigger displays and can maintain integrity when the display ratios are different.

The structure of this phase is the introduction of the test case, the expected outcome, and the comparison with the actual outcome. Since we were doing E2E testing as we were developing the application, we didn’t run into many issues during this phase, some minor UI bugs were found but were maintained.

## Test case 1: Splash screen

Logo

Description automatically generated with medium confidenceE2E testing: Android (left), IOS (center), Tablet (Right)

Test Outcome: The test is passed.

## Test case 2: Welcome Page

E2E testing: Android (left), IOS (center), Tablet (Right)

Graphical user interface, application

Description automatically generatedA picture containing chart

Description automatically generated

Test Outcome: The test is passed.

## Test case 3: Sign up functionality.

Expected Outcome: The outcome of this test is that the sign-up process offers 2 types of registration for both drivers and sellers, with exception handling like email check, password check, fields are all full.

E2E testing: Android seller page (left), Android driver page (Right), Tablet (Bottom)

Graphical user interface, text, application

Description automatically generatedGraphical user interface, application

Description automatically generatedGraphical user interface, application

Description automatically generated

## Test case 4: Sign in and exception handling.

Graphical user interface, application

Description automatically generatedGraphical user interface, application, PowerPoint

Description automatically generatedExpected Outcome: The user is expected to be shown error messages when they enter a wrong password, a user is not found error when they enter a user that is not present, and a loading screen when they successfully enter their credentials.

E2E testing: Android wrong credentials (left), Android wrong email (Center), Android correct credentials (Right).

Graphical user interface, application

Description automatically generated

Test Outcome: The test is passed.

## Test case 5: User first time login

Expected Outcome: The user is asked to verify their email address via an email link, they can resend the email if they didn’t receive it.

E2E testing: Android e-mail verification.

Graphical user interface, application

Description automatically generatedText

Description automatically generated

Test Outcome: The test is passed.

## Test case 6: Handling empty lists.

Expected Outcome: The user is shown placeholders of empty lists when they don’t exist.

E2E testing: Android notifications empty (left), Android map orders empty (centre), Android orders empty (right).

Graphical user interface, text, application

Description automatically generatedA picture containing text, electronics, screenshot

Description automatically generated

A screenshot of a phone

Description automatically generated with medium confidence

Test Outcome: The test is passed.

## Test case 6: Edit/ Delete entries.

Expected Outcome: The user can edit or delete orders.

Graphical user interface, application

Description automatically generated

Test Outcome: The test is passed.

## Test case 7: Driver can see seller orders.

Expected Outcome: The Driver can see the sellers’ orders and choose the one he sees fit.

E2E testing: Android orders published by seller (left), Android driver sees the seller profile (centre), Android driver can visualize the orders and he revenue (right).

Graphical user interface, text, application, chat or text message

Description automatically generatedGraphical user interface, text, application

Description automatically generated

Graphical user interface, application, map

Description automatically generated

Test Outcome: The test is passed.

## Test case 8: Driver/ Seller interaction on handling of delivery.

Expected Outcome: The Driver is assigned the orders and can see he orders and the seller he is delivering to. The seller is notified that the driver is assigned and can see his progress.

E2E testing: Android driver can see the orders and the seller (left), Android seller can see his assigned driver (right).

Graphical user interface, text, application

Description automatically generatedGraphical user interface, application

Description automatically generated

Test Outcome: The test is passed.

## Test case 9: Driver is prompted to retrieve the items.

Expected Outcome: The driver can see the location of the seller and the address and is asked to collect the item.

Map

Description automatically generated

Test Outcome: The test is passed.

## Test case 10: Driver completes delivery.

Expected Outcome: The driver delivers the last entry in the orders bundle, and the seller is notified about this, now the seller can post new orders, and the driver can accept new orders.

E2E testing: Driver delivers the last order (left), Seller orders are archived after delivery, he can rate the driver based on customer feedback (right).

Text

Description automatically generated with low confidence

Graphical user interface, application

Description automatically generated