# [SET10101] Software Architecture

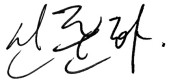
Coursework

Ihor Pisklov
40624131
November 28, 2024

**Declaration**

I declare, except where explicit reference is made to the contribution of others, that this assignment is the result of my own work and has not been submitted for any module or programme degree at the Edinburgh Napier University or any other institution. This is in accordance with Edinburgh Napier University's Academic Integrity Regulations.

☑ NO: I have not used such tools
☐ YES



# Contents


**1 Introduction** **1**

**2 Architectural alternatives for the system** **2**
2.1 Alternative 1: Client-Server architecture . . . . . 2
2.2 Alternative 2: Distributed Peer-to-Peer architecture . . . . . 3

**3 Justification for the selected architecture** **5**

**4 Design** **6**
4.1 Overview . . . . . 6
4.2 Component design . . . . . 6
4.3 Data flow and interaction . . . . . 8
4.4 Database schema design . . . . . 9
4.5 Scalability considerations . . . . . 9
4.6 Design challenges and resolutions . . . . . 10

**5 Evaluation** **11**
5.1 Functional requirements . . . . . 11
5.2 Non-functional requirements . . . . . 11
5.3 Strengths of the Client-Server architecture . . . . . 11
5.4 Limitations and challenges . . . . . 11
5.5 Recommendations for improvement . . . . . 12
5.6 Validation against requirements . . . . . 12

**6 Conclusion** **13**

**7 Appendix** **14**
7.1 Screenshots. Web portal . . . . . 14
7.2 Screenshots. iOS mobile app . . . . . 16
7.3 Screenshots. Firebase Firestore . . . . . 18

# 1 Introduction

KwikMedical are looking for an effective, robust, and scalable design solution for an ambulance dispatch system prototype. The goal is to create a system that supports real time data management and seamless communication between operators at the headquarters and rescue teams that work in ambulance vehicles. As specified, two architectural alternatives, Client-Server and Distributed Peer-to-Peer, were evaluated and compared based on how they meet the project investigator's requirements. This report also includes descriptions of the design and development processes, evaluates implementation and suggests future improvement strategies.

# 2 Architectural alternatives for the system

Two architectural alternatives were considered for implementing the prototype of KwikMedical ambulance system: Client-Server architecture and Distributed Peer-to-Peer architecture. Both have their advantages and disadvantages, so each alternative was analysed in detail and the architecture that best suits aims and goals of the project was chosen and implemented. These alternatives were evaluated based on their ability to fulfil the system's requirements: patient data creation, processing, and management, seamless connection between the web platform and the mobile device through a database, and potential for future implementation of GPS-related features.

## 2.1 Alternative 1: Client-Server architecture

The Client-Server architecture was a great potential choice for the project. The system would consist of a centralised server (Firebase Firestore) hosting patient data, vehicle data, and dispatch information. The clients — web and iOS applications — interact with it in order to retrieve or update data. It is a very robust architecture, as different user-facing platforms do not depend on each other, which is a guarantee for ease of further scaling, refactoring, or improvement of the system. The two platforms communicate with each other using Firebase SDK. This means that data is stored and accessed separately, and can be protected by high-standard security rules independently from user-facing clients.

### Components and connectors

Clients:

- Web portal for the phone operators
- iOS app for mobile devices used in ambulance vehicles

Server:

- Firebase Firestore

Connector:

- Firebase SDK

### Advantages

One of the major advantages of Client-Server architecture is that administrators can manage all patient, vehicle, and dispatch information from a single source of truth: the centralised database. Moreover, such an architecture is easy to understand and simple to implement: every component is responsible for its own independent functionality, e.g. Firebase only stores and serves data, while the web platform and iOS mobile app write and read data.

### Disadvantages

Client-Server architecture is heavily dependent on reliable and stable network connection, not allowing for offline functionality due to its nature that includes a server as the third component, the link between clients. Additionally, it has a single point of failure, which means that if the server experiences downtime, the whole system will be affected, and therefore no data will be accessible or updated.

## 2.2 Alternative 2: Distributed Peer-to-Peer architecture

Distributed Peer-to-Peer (P2P) architecture was the second one to be considered for this project. This architecture embodies decentralisation of data storage and processing. This way, the web portal and the iOS application communicate directly or via intermediary nodes. Following the architecture's nature, it is therefore more complex in terms of development, maintenance, and scaling, as the components directly depend on each other. The reason behind that is that as one platform evolves (e.g. fixing errors in code, refactoring or adding functionality, scaling etc.), the other part(s) of the system must evolve, too, in order to assure continuous functioning state. It is an attractive option if the project investigator is looking to eliminate any server usage due to various reasons, such as cost constraints, concerns about reliance on third-party providers, or avoiding having a single point of failure.

### Components and connectors

Peers:

- Web platform: shares and updates dispatch and patient data
- iOS mobile app: shares and updates dispatch and vehicle data

Data exchaning mechanism:

- A decentralised protocol (e.g., WebRTC or similar)

### Advantages

Possibly the biggest advantage of Peer-to-Peer architecture is having no single point of failure, which effectively makes the system very resilient, as data is distributed among peers. Additionally, in bigger scale systems with larger portions of data being transferred during the runtime, latency can be less significant, as opposed to not-as-rapid options that use a centralised server. Moreover, the architecture allows for significant scalability options as each peer adds to the common processing power of the system, and this reduces reliance on some '*links of the chain*'.

### Disadvantages

As one component is changed in a substantial manner, connection to the other side of the system might be affected in a negative way: distorted or even lost. Also, a Peer-to-Peer system is simply difficult to build and requires significant effort to maintain. Data consistency is also a big challenge for this architecture choice: ensuring that all peers have up-to-date information in a real-time context is a peculiar task that demands additional resources. Synchronising data and resolving conflicts between peers is another disadvantage that comes with this architecture: there is no centralised authority to enforce consistency. This means that each peer must independently handle updates to avoid erroneous data.

# 3 Justification for the selected architecture

As a result of careful comparison of the two architectures, Client-Server architecture was selected, as it best matches the requirements of the NHS dispatch response system, is simple and very scalable.

### Feasibility of implementation

The Client-Server architecture was much easier and more practical to use in the context of the project's timescale, resource constraints, and technical requirements. Google Firebase provided an easy and accessible solution for real-time data synchronisation and storage, so the need for complex and heavy custom backend development was omitted.

### System simplicity

The system per se is fairly simple because of how different components interact with each other: data can easily be kept well-maintained, correct, and synchronised. The system design was simplified by centralising data in Firebase. This allowed both the web platform and iOS mobile app to perform CRUD operations seamlessly without the complexity of maintaining a distributed network like P2P.

### Scalability

First of all, the database of choice, which is Firebase Firestore in this project, scales automatically to accommodate all kinds of bandwidth owing to its serverless architecture. This ensures that the system can perform well even when the amount of users and therefore user requests increases.

### Suitability for the use case

First of all, as per specification, the client is already using a database. Simply this fact is a very powerful influence on the decision, as moving a database from an SQL environment to Firebase would be a much easier and a more resource-efficient task than reformatting it and splitting it for Peer-to-Peer architecture. Moreover, there is no significant reason to deter from using a database when reliance on a third-party service is clearly not a problem for the project investigator.

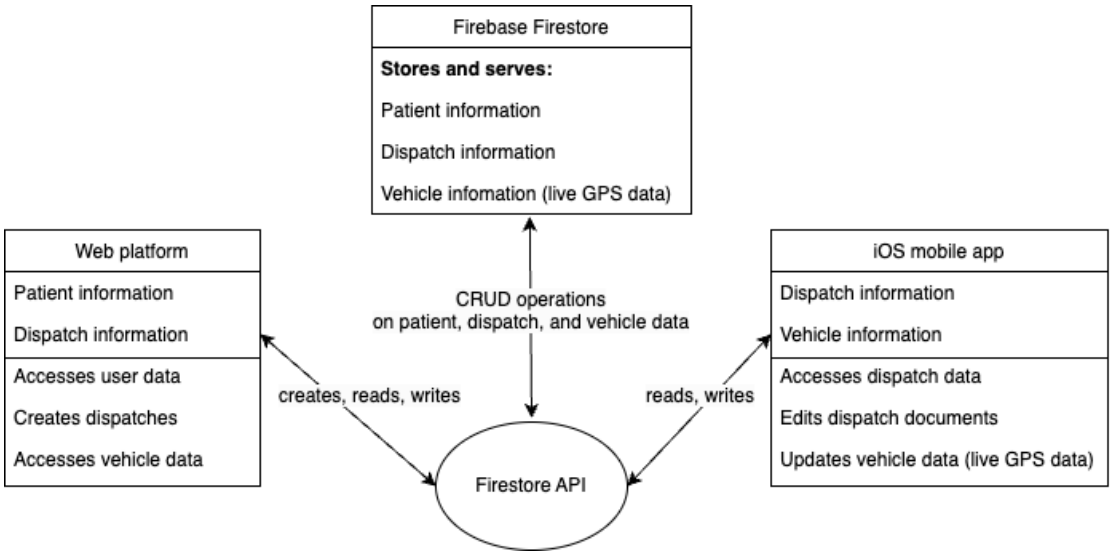
### Reduced complexity

Peer-to-Peer architecture, while very resilient, would introduce unnecessary complexity to the project, so going for Client-Server architecture was a great way to simplify and speed up the design, development, and testing processes. Maintenance — especially data maintenance — would create a need for more resources and the whole process would be more time-consuming.

# 4 Design

## 4.1 Overview

The system, as implemented using the Client-Server architecture, overall consists of three components and a connector: the web platform (client), the iOS mobile app (client), Firebase Firestore database (server), and Firebase Firestore API (connector). Every component has strictly outlined functionality: the matters are intentionally separated to keep the system simple, organised, and optimised.




## 4.2 Component design

### Web platform

The web platform is one of two user-facing components of the system. It is what the operators, per specification, would use in order to:

- **Look up existing patients**: the operators' website compares user-input NHS number of the patient against existing documents in the database.
- **Create new patients**: in case a patient does not yet exist in the system, operator can type details like first and last names, date of birth, address, and condition. With that data, two documents are created: patient document and dispatch document that the iOS mobile app will receive immediately.
- **Issue dispatches for rescue**: whether or not a patient exists or not, after all necessary steps (search or creating a new patient document) a dispatch is issued immediately and a new document is written to the database.

### iOS mobile app

The iOS mobile application is what the ambulance vehicle would be fitted with. The only functionality stated that is needed was to:

- **Access dispatch data**: the app automatically finds the longest-waiting dispatch and displays its details — the details that operator input on the web platform.
- **Update dispatch information**: functionality that was requested by project investigators, such as possibility to write down what actions were taken, how much time was spent on the call, and additional notes, was gracefully implemented. As a bonus, the concept of dispatch status was added to the system in order to differentiate between completed, active, and pending dispatches.
- **Write live location as coordinates to database when engaged**: by pressing the ”Start rescue” button the app starts updating the vehicle’s live coordinates value in the database. The coordinates were set up to refresh every 15 seconds, although due to how flexible code was made to be, it can be changed to 1, 5, 30, or any other amount of seconds.
- **Change status of vehicles**: although not requested in the specification, this useful feature can be crucially useful in the future. Using status of vehicles, and especially in combination with location-based functions, the hospitals might implement an effective, *scientifically supported* system of assigning ambulances to certain dispatches.

### Firebase Firestore

Firestore database is the one component of the system that is non user-facing, however it plays a leading role in every process: creating, reading, updating, and deleting—it is the heart of the system. It is the server, so its functionality includes:

- **Storing patient, dispatch, and vehicle data**: by having a database, the architecture becomes fairly simple to implement. This powerful tool that is set aside from other components with different functionality retains and synchronises all data with integrity and consistency.
- **Serving patient and dispatch data**: every time the web platform or the iOS mobile app are displaying information about existing patients or current dispatches, they send a read request to the database.
- **Writing (updating) patient, dispatch, and vehicle data**: during runtime, both the operator’s web platform and the ambulance vehicle app can send write requests, whether it is to create a new patient, initialise a rescue operation, update information about an active dispatch, or to update a

vehicle's current location.

– **Deleting information**: in theory, if any information will need to be deleted after use, such functionality is very easy to implement as Firebase Firestore API was designed with developers in mind. Currently, there are no reasons so far for the algorithm to be deleting any data, as dispatch documents that are marked as complete by the rescue team are retained in the database for further possible use, e.g. analysis.

## 4.3 Data flow and interaction

Structuring the workflow, the whole process starts with the operator at the headquarters using the web platform. Per specification, when they get a call, they ask for the patient's details. Purely for simplicity and for it being just the prototype, patient's NHS number was chosen as the main detail for patient lookup [see 7.1.1 and 7.3.2]. If a person does not yet have an NHS number [see 7.1.2], it is very easy for the operator to simply register a new patient with their first and last names, date of birth, address, and current condition that made them call an emergency service [see 7.1.3].

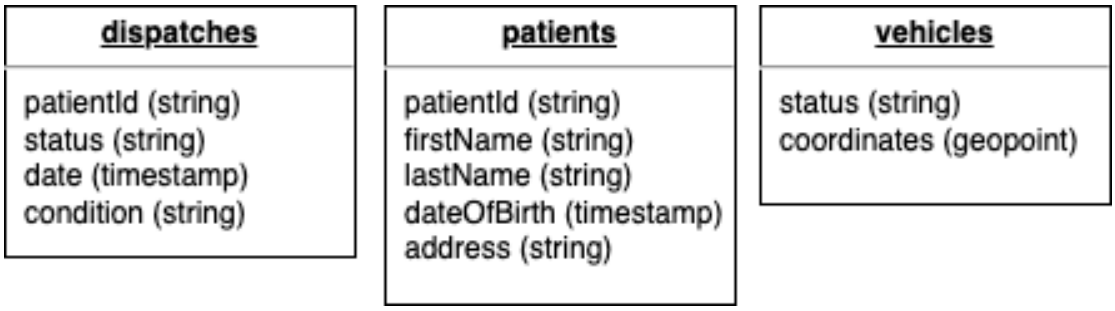
When a dispatch is issued, its initial status is assigned to be 'pending'. As soon as operator is finished initiating a dispatch, the emergency response team who are using the iOS mobile app [see 7.2.3] can see dispatch details: patient's full name, age, condition, and address. Them being an ambulance team, they have no other choice than to start the rescue process: they would simply click the "Start rescue" button and get to work.

While assessing the dispatch, they can add notes to the dispatch, just like the project investigators have requested. They can log actions taken, time spent on call, and additional notes. These are dynamically saved to the database [see 7.3.1] and can be reviewed as many times as needed while that exact dispatch is active. As the vehicle (the device in this case) is fitted with GPS, the iOS app writes live location of the device to the database every 15 seconds [see 7.3.3]. This can eventually facilitate location-based functions, although currently none are specified in the technical requirements.

As soon as the rescue team are finished with their job, they must press the "Finish rescue" button and confirm their choice. This will stop live location from being written to the database until the next dispatch. The status of the dispatch in the database will also be changed to 'completed', and status of the vehicle to 'available'.

## 4.4 Database schema design

To maintain a clean and sophisticated structure of the system as a whole, the database was structured as three tables (as called *collections* in Firebase):




Each of three collections are responsible for a different type of document:

- **patients**: the collection for all patients' documents. These are accessed by the web platform to quickly initiate dispatches for existing patients and to register new patients. Additionally, it is accessed by the mobile app to display patient information using patient's NHS number.
- **dispatches**: the collection for all dispatch documents. These are accessed by both the web platform and the mobile app in order to respectively write and view new emergency dispatch information, and change dispatch status.
- **vehicles**: the collection for all existing ambulance vehicles. These are accessed by the mobile app to change the status of the vehicle from 'available' to 'engaged' and the other way round, and to write live coordinates of the device for future potential location-based functions.

In actual code, these tables are treated as objects, and their key values are the unique IDs that were assigned by Firebase (patient and dispatch documents) or hard-coded on purpose (vehicle documents) for the sake of clarity and simplicity at the early prototype stage.

## 4.5 Scalability considerations

Leveraging Firebase Firestore's serverless architecture and real-time synchronisation capabilities, ambulance dispatch system was built with a big potential for scaling. As Firebase automatically manages resources to suit different workloads, performance can be excellent even during high-demand, busy scenarios, such as emergencies or disasters.

Timer-based updates for live GPS data were implemented to optimise database interactions. This approach even further reduces Firebase workload and still retains location accuracy at an appropriate level that might be needed at a hospital.

## 4.6 Design challenges and resolutions

One most difficult challenge during the development process was to understand how the app should be structured.

One of the main principles of software engineering is KISS, which stands for Keep It Simple, Stupid. This is exactly what was the hardest: initially, the app was split into three separate 'tabs', and each tab had certain functionality, such as switching to engaged mode on one page, accepting and finishing an actual dispatch on another page, and editing dispatch information on yet another separate page. Eventually the final idea came together: the simpler the better.

The resolution was to use one screen purely for functional features and leverage 'pop-up sheet screens', where the user can input dispatch details. The whole concept of manually setting availability for an ambulance vehicle did not make the cut either: this functionality turned out to be redundant per job's nature.

# 5 Evaluation

The system was evaluated based on its ability to fulfil the functional and non-functional requirements specified in the project brief, as well as its overall strengths, limitations, and areas for improvement. The system was designed and implemented with Client-Server architecture. It was selected for its simplicity, scalability, and ability to meet the project's demands efficiently.

## 5.1 Functional requirements

The system meets its primary functional requirements that were specified in project's description. The web platform enables operators to create and retrieve patient records and create new dispatches. The iOS mobile app features real-time access to dispatch details, allows to update dispatch status, and supports live location tracking. All these features are seamlessly implemented and facilitate efficient dispatch management, making sure that emergency teams have access to accurate and up-to-date information.

## 5.2 Non-functional requirements

The system demonstrates strong scalability, reliability, and usability. Firebase Firestore's serverless architecture provides potential for various workloads, from regular daily stream of requests to high-demand scenarios during emergencies. Data is synchronised across the web platform and mobile app in real time, which is crucial for both operators and ambulance teams. User interface follows the KISS principle: it is user-friendly, simplifies navigation, and enhances operational efficiency.

## 5.3 Strengths of the Client-Server architecture

The Client-Server architecture was proven to be effective for this type of system. Using a centralised database for data management (Firebase Firestore) has simplified data operations, offering a single source of truth for patient, dispatch, and vehicle data. Component interact in a straightforward manner, which reduces development complexity and makes maintenance easy. Furthermore, the serverless database infrastructure provides automatic resource scaling, which results in consistent performance.

## 5.4 Limitations and challenges

Despite its strengths, the system has some limitations, too. Areas with poor network coverage can easily affect performance, particularly for the ambulance teams using the iOS mobile app. Additionally, because the database can be a single point of failure, the system might experience downtime in rare circumstances.

## 5.5 Recommendations for improvement

Future improvements could focus on addressing existing limitations: adding offline functionality can eliminate the earlier mentioned single point of failure. One way to potentially do that is to introduce caching mechanisms, which would get a snapshot of the database when connection is active, store and manipulate it locally (whether the system is online or offline), and synchronise data with Firebase Firestore as soon as connection is restored or when it is time to (potential use of timed synchronisation). Security could also be enhanced by implementing additional encryption layers when moving data, and creating Firebase Firestore security rules to control what kind of requests to respond to. The web platform could benefit from further usage of existing GPS functionality, for example visualising vehicle position on a map in real time. This would support hospitals in decision making (e.g. preparing operation room before patient arrives).

## 5.6 Validation against requirements

The web platform and iOS mobile app were thoroughly tested and validated against the specified requirements. They consistently met the operational needs of both operators and ambulance teams, and confirmed their functional and non-functional capabilities. While there are areas for improvement, the prototype implemented is a robust and scalable solution for emergency dispatch management.

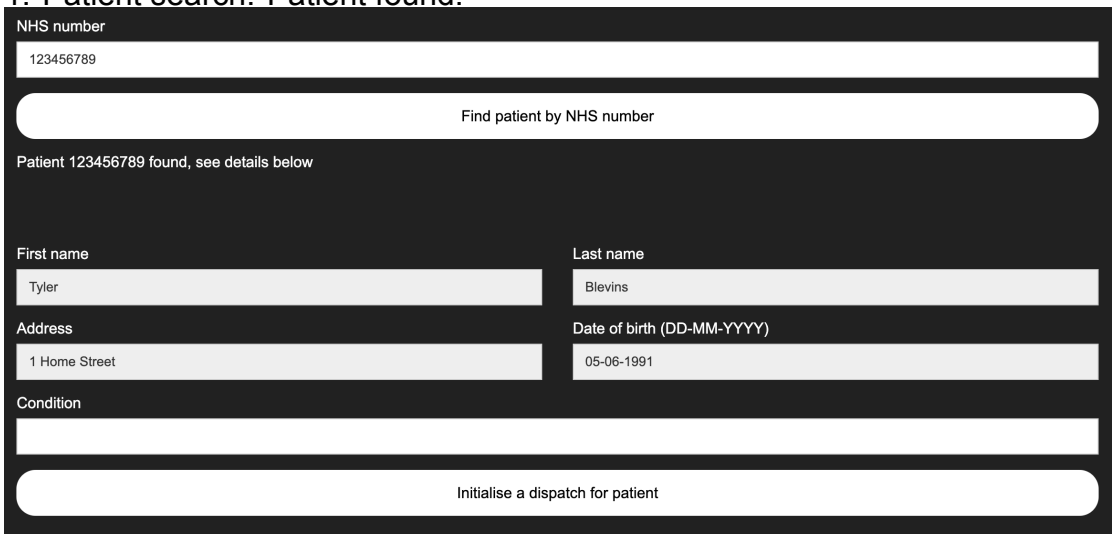
# 6 Conclusion

The ambulance dispatch system successfully meets the project's functional and non-functional requirements. Owing to Client-Server architecture with Firebase Firestore, the web platform and the iOS mobile app offer real-time synchronisation, scalability, and ease of use. While there are certain limitations related to extreme internet connection reliance, they can be addressed by implementing offline functionality and caching. Overall, the prototype is a robust solution that can already be used by a hospital if needed as soon as security features are implemented.

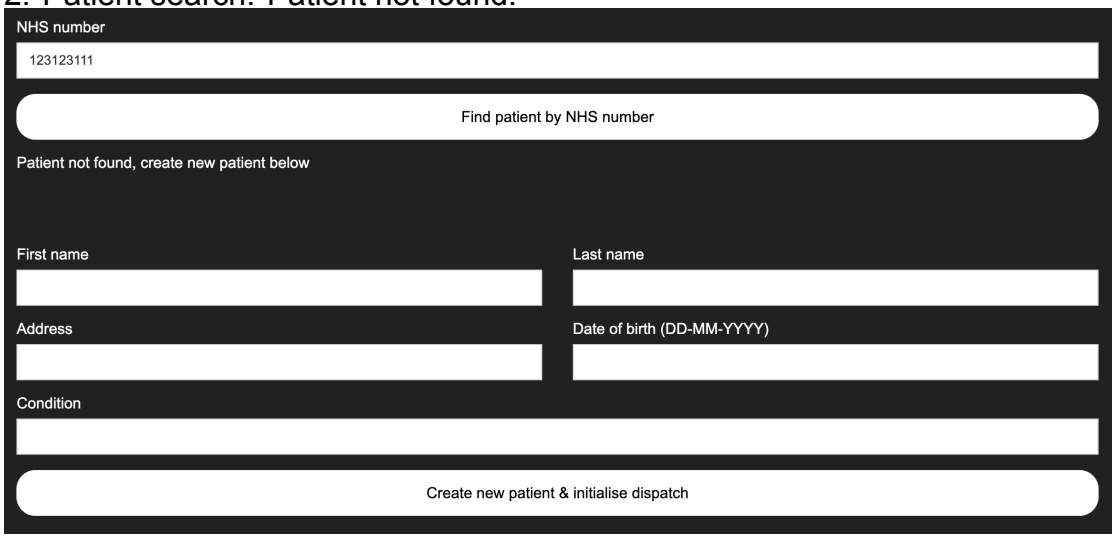
# 7 Appendix

## 7.1 Screenshots. Web portal

1. Patient search. Patient found:




2. Patient search. Patient not found:

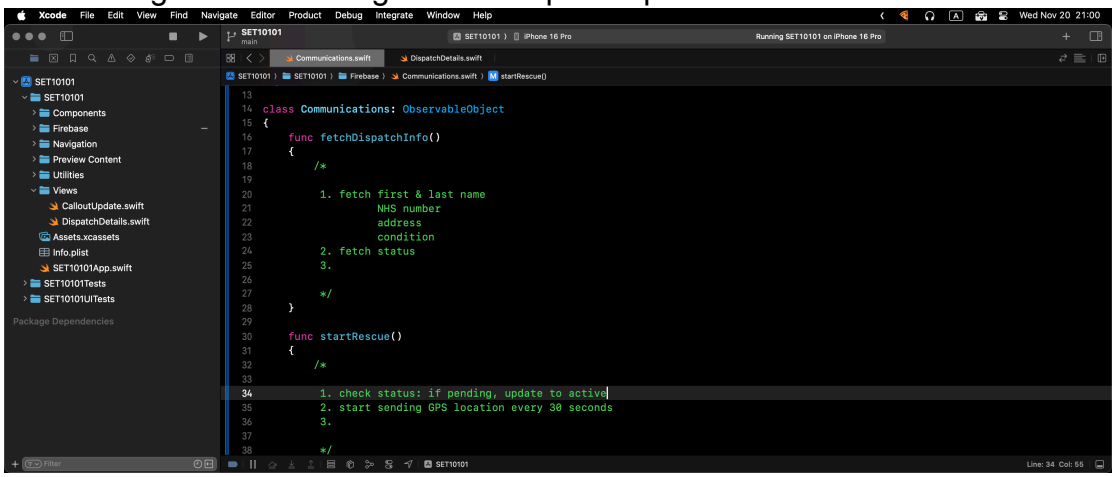


3. Creating a patient:

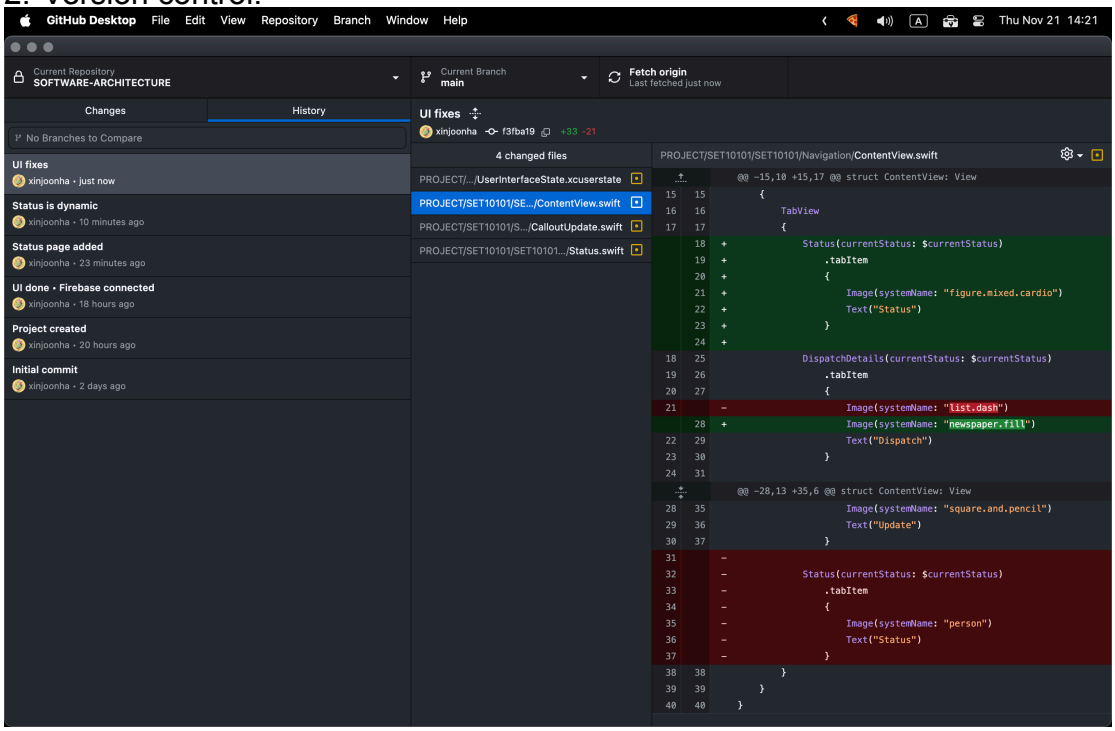

4. Field validation:

## 7.2 Screenshots. iOS mobile app

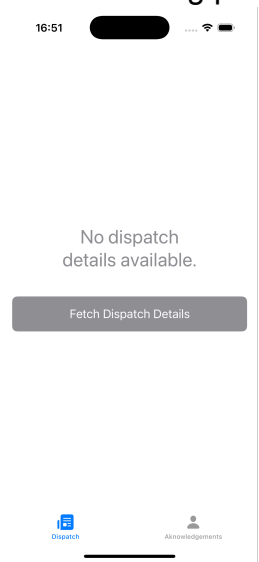
1. Outlining features during the development process:

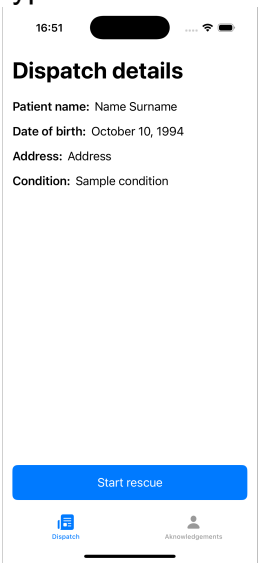


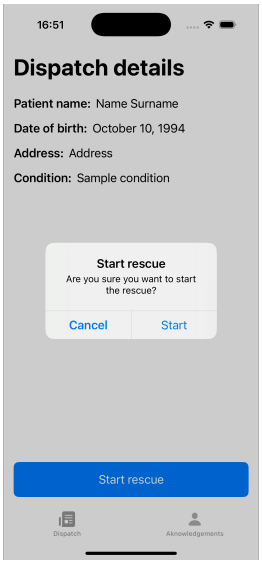

2. Version control:

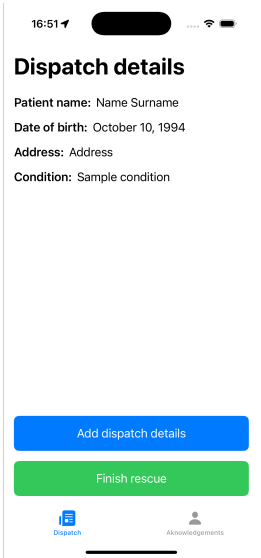


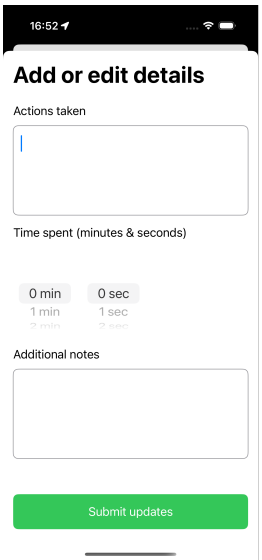
## 3. Functioning prototype:

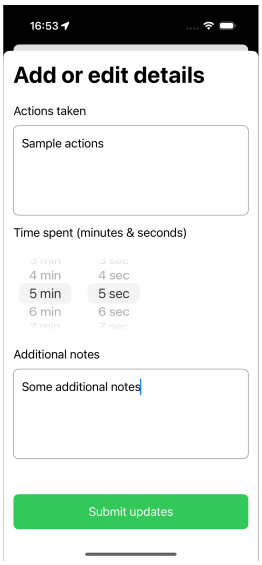


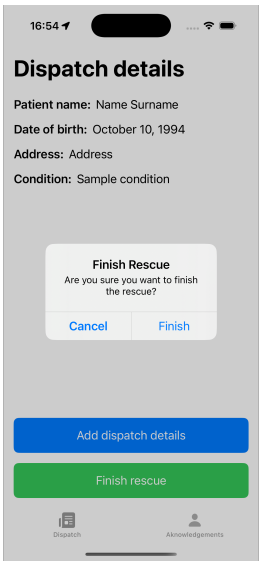



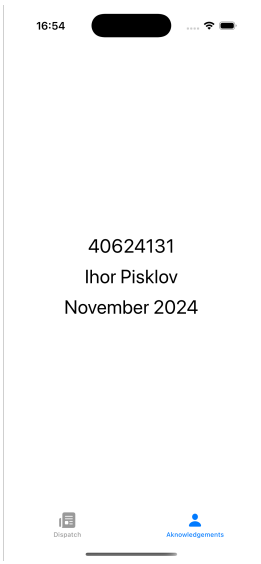



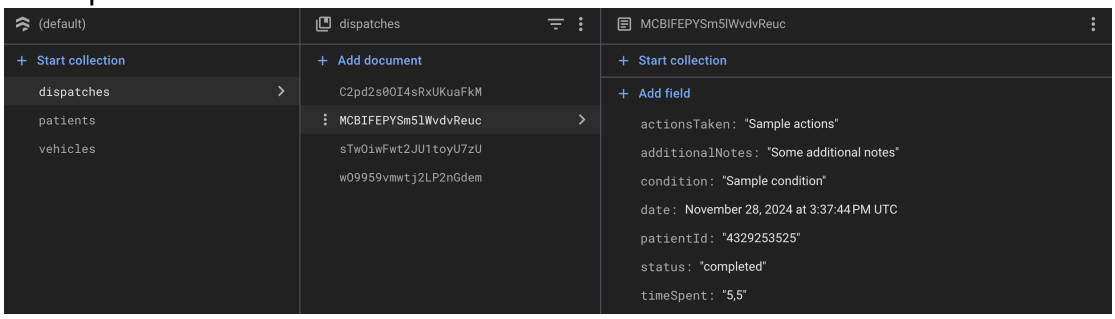



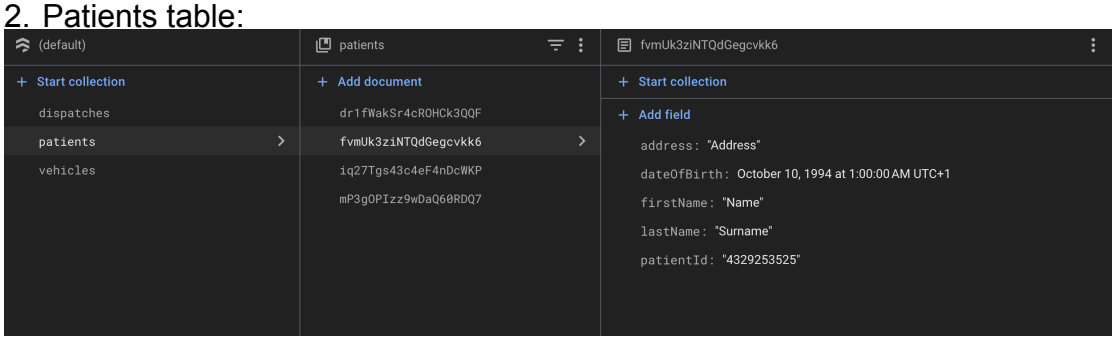



## 7.3 Screenshots. Firebase Firestore

1. Dispatches table:




2. Patients table:




3. Vehicles table: