Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingeniería y Sistemas de Telecomunicación

Mobile Devices Programming

RideSOS

Supervisor:

Miguel Ángel Valero Duboy
Ana Belén García Hernando

Authors:

Corpaci Ana Daria
Franzoso Antonio
Morano Stefano

Madrid, November 18, 2024

Contents

1 Introduction

People love adrenaline. It's satisfying, energizing, entertaining. But to continuously seek it can be extremely dangerous, too. One sure way to get it comes from speed. Since young ages children nowadays love racing games, cars, motors. The latter are our main focus.

Driving can be wild and the driving safety rules are underestimated by a lot of people. People are rushing, doing illegal things as they are in a hurry. This is how a lot of motorcycle or bicycle accidents happen. The gravity of these depends on many factors, including the location and the life condition of the driver.

2 Motivation

In many cases, accidents happen due to factors like speeding, poor road conditions or distracted driving. Furthermore, when a crash occurs in remote areas or during late hours, it can be challenging for the victim to get timely help. The lack of immediate assistance can turn minor injuries into life-threatening situations.

RideSOS aims to bridge this gap by leveraging modern smartphone technology to provide an automatic and reliable safety solution. With features like real-time crash detection, automatic emergency alerts and GPS-based hospital navigation, RideSOS offers a comprehensive safety net for riders, by providing real-time monitoring and emergency assistance, especially obtaining the ambulance service in an immediate way, but also signaling the crash and making the driver aware of its personal life condition.

3 App User Requirements

To ensure that RideSOS effectively enhances rider safety and provides a seamless user experience, several key requirements have been identified based on the needs of potential users. These requirements are split into functional and non-functional categories, as shown below:

1. **Functional**

   - **Profile Setup and Personalization**

     Once the app is downloaded, users will be able to create a profile with their name, birthdate, gender, blood type and phone number. There is automatic checking of all of these fields to assure users complete the form with accurate data. Additionally, users will be asked to answer 3 predefined personal safety questions, which will later be used to assess their consciousness in case of an accident.

   - **Crash Detection and Emergency Response**

     After the setup is complete, the app will continuously monitor the accelerometer data to detect potential crashes, if the user signals the app that a new ride will begin, through a slide button from the home activity. The app will immediately prompt the user with a pop-up notification to assure app is currently monitoring the ride. In case of an accident, if the user fails to respond or indicates they need help, the app should automatically alert emergency services thorugh MQTT and even call 911.

   - **Nearby Hospital Information**

     The app should have access to the user's location to identify nearby hospitals quickly. This information will be crucial if the user needs to find medical assistance immediately after a crash or wants to navigate to the nearest hospital. Otherwise, the data will be displayed randomly.

   - **Background Operation**

     The app must be able to function in the background without interrupting the user's ride. The crash detection service should continue to run even if the screen is locked or the app is minimized, ensuring that monitoring is continuous.

   - **User-Friendly & Accessible Interface**

     The app must provide an intuitive and simple interface. Users should be able to easily start and stop ride monitoring, navigate between features (e.g., view nearby hospitals, access their profile), and receive clear notifications and prompts in case of emergencies.

2. **Non-Functional**

   - **High Reliability and Accuracy**

     The crash detection algorithm must be highly accurate to minimize false positives and false negatives. This ensures that users are not unnecessarily alarmed and that real accidents are promptly detected.

   - **Offline Functionality**

The core crash detection should work offline. This is especially important for riders in remote areas with limited connectivity. To have the hospital accessible at all times, even in areas without internet connection, the JSON list should be downloaded once and stored in an internal database and then rendered anytime the user requests the data. Re-download logic should be implemented to assure the list is up to date.

4 App Design

To design the application's layouts, we utilized Figma, a dedicated interface design software, to create a detailed prototype. The complete prototype is available at [1]. Once downloaded on a device, the user will be asked to complete their profile and to fill 3 different safety questions, all of them personally decided. These questions will be then asked in case of an accident to assess the consciousness state of the user. When everything is completed, the user will be redirected to the main activity. Users can switch between all of these activities through a bottom navigation bar.

4.1 Main Activity

The primary scope of this is to allow the user to record a new ride, by turning on a **switch**. This will trigger 2 actions: the start of a background service that will read the accelerometer data on all 3 axis, to detect a potential crush during the ride, as well as the pop of an icon in the notification bar that will stay on until the switch is turned back off, to assure the driver that the app is monitoring.

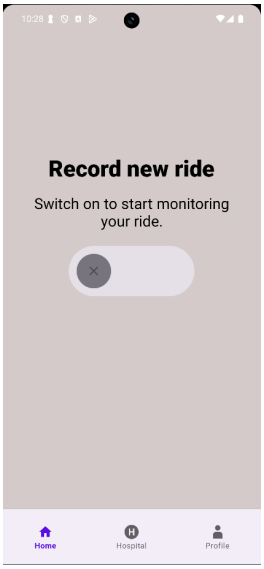

Figure 1: Main Activity Screenshot

4.2 Hospital Activity

This UI displays the list of nearby **hospitals** based on the current location of the user. Hospitals' data is downloaded once per app lifecycle and then stored in an internal Room Database. The database is instanced through the **Singleton pattern** and used in the adapter to display it. For the activity to display data correctly, the user must grant the app permission to access their location. We consider this functionality essential because, in the event of an accident, if the user is conscious and able to reach the hospital on their own, we want to provide the closest options first. Without these permissions, the data will appear randomly. Considering the current location can be read by the app, the hospitals list is sorted ascending using the **Haversine formula**.

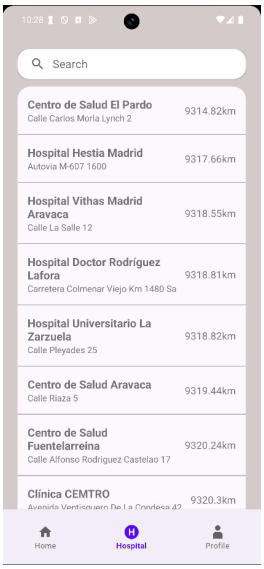

Figure 2: Hospital Activity Screenshot

4.3 Profile Activity

This page displays the user's **main data**, including basic information such as name, surname, phone number, and date of birth, as well as more **personal details** like blood type and the personal response that will be shown during the crash detection system. The user can also upload an image to complete their avatar.

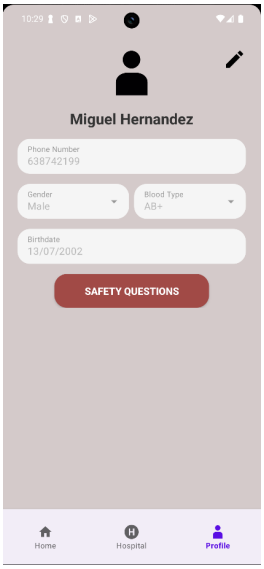

Figure 3: Profile Activity Screenshot

4.4 Question Pop-Up

When the crash detection system detects an incident, it triggers a **pop-up** with a background color that changes to alert anyone nearby of a problem. The alarm also activates the phone's flashlight and plays a sound to help locate the device.

The pop-up contains two buttons: **"I'm fine**" and **"I'm not okay"**. If the user selects

"I'm fine", they must respond to a quiz-like questionnaire based on the personal questions they provided during registration.

If the user fails to answer correctly (perhaps due to loss of consciousness) or clicks *"I'm not okay"* or don't touch anything for 10 seconds, the app sends an **MQTT message** to the ambulance server, providing the accident's location along with the user's parameter data.

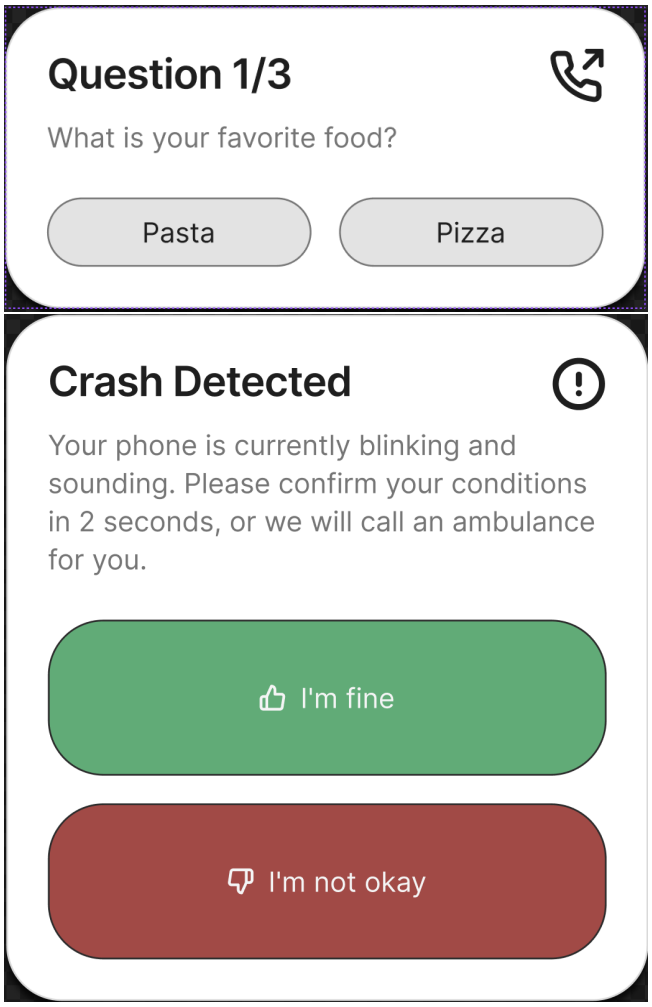

Figure 4: Alarm and Question Pop-Up Screenshots

5 App Development

5.1 Main Features

The app includes a **crash detection system** that can be activated by pressing the main switch button on the home page. The detector continues to operate even if the app is running in the background or the screen is locked.

When a crash is detected (Section 5.2.2), the phone automatically wakes up and unlocks, displaying a flashing pop-up with changing colors and activating the phone's flashlight. This feature is designed to alert people nearby, signaling that an accident has occurred and prompting them to slow down or offer assistance. The flashlight is particularly useful on poorly lit roads.

The pop-up asks the user whether they are okay. If the user selects "I'm not okay" or fails to respond within 10 seconds, the app automatically sends an MQTT message (Section 5.2.4) or calls an ambulance. If the user indicates they are okay, the app displays a personal question to ensure they are conscious and responsive.

If the user correctly answers all personal questions (Section 5.2.6), the app presents buttons that allow them to navigate to the **Hospital Activity**, return to the **Main Activity** or close the app. If the user fails to answer correctly, the app sends an MQTT message or, depending on the user choice, it calls emergency services.

5.2 Sensor and Services

5.2.1 Accelerometer

The crash detection system utilizes the phone's accelerometer sensor to monitor acceleration in real time and identify potential crashes. It leverages the **SensorManager** API in Android, which provides access to hardware sensors and allows efficient management of their operations [Figure: 5]. The system is designed to compute total acceleration by sampling data at regular intervals, such as every 500 milliseconds, and analyzing it for patterns indicative of a crash.

To ensure efficiency, the system includes a method for activating and deactivating the accelerometer sensor. This prevents the sensor from running unnecessarily, conserving power and reducing resource usage. When activated, the accelerometer readings are processed to calculate the total acceleration based on the x, y, and z components of movement. These calculations enable the detection logic to identify abnormal spikes in acceleration, which might signal a crash event.

```
public AccelerometerSensor(Context context, int sensDelayMs, CrashListener listener) {
    sensorDelayMs = sensDelayMs;
    crashListener = listener;

    sensorManager = (SensorManager) context.getSystemService(Context.SENSOR_SERVICE);
    if (sensorManager != null) {
        accelerometer = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
        assert accelerometer != null;
        boolean isWakeUp = accelerometer.isWakeUpSensor();
        Log.d(TAG, msg: "Accelerometer is wakeUpSensor? " + isWakeUp);
    }
}
```

Figure 5: Snippet of the accelerometer initialization code

5.2.2 Accelerometer Service

The Accelerometer service is a crucial class for handling the accelerometer sensor. It implements the accelerometer functionality and reads the sensor values in background threads, ensuring it continues to function even if the user switches apps or turns off the screen.

When the user activates the **detector**, the service sends a notification and then starts the background thread. Each time the thread reads the accelerometer values, it calculates the total acceleration. If it detects a **G-forc**e above a certain threshold, it triggers a pop-up question.

The **detectCrash** algorithm calculates the magnitude of the acceleration vector from the accelerometer's x, y, and z values using the formula

$$\text{magnitude} = \sqrt{x^2 + y^2 + z^2}$$

It then compares the current magnitude with the previous reading. If there's a significant change, like a **high-to-low G-force** (indicating sudden deceleration) or if the magnitude exceeds a predefined threshold (suggesting a crash), it triggers a crash alert. The algorithm helps detect **sudden braking or impacts** by analyzing acceleration **data over time**, ensuring quick response to potential emergencies such as vehicle accidents or sudden stops.

```
private boolean detectCrash(float[] values, long currentTime) {  1 usage
    // Extract the x, y, and z accelerometer values
    float x = values[0];
    float y = values[1];
    float z = values[2];

    float magnitude = (float) Math.sqrt(x * x + y * y + z * z);

    if (currentTime - lastTime >= TIME_THRESHOLD)
        return (lastMagnitude > BRAKING_THRESHOLD && magnitude < BRAKING_THRESHOLD);

    lastTime = currentTime;
    lastMagnitude = magnitude;

    return magnitude > CRASH_THRESHOLD;
}
```

Figure 6: Snippet of the detect crash code

5.2.3 GPS

The GPS sensor is a critical component to assure users fully benefit from complete functionalities of our app. To access data from the sensor, users must grant permissions for **ACCESS_FINE_LOCATION** and **ACCESS_COARSE_LOCATION**, which the app will request once during its lifecycle. With these permissions, the app can sort the list of hospitals in ascending order based on their distance from the user, using the Haversine formula. Without these permissions, the hospital list will be displayed in a random order.

If the permissions are granted, a *FusedLocationProviderClient* is initialized to obtain the user's location after configuring a *LocationRequest* object. This is set to request only a single update, as the user's location is needed just once per lifecycle of the Hospital Activity. The *requestLocationUpdates* method of the *FusedLocationProviderClient* is called with three parameters: the configured *locationRequest*, a *LocationCallback* to handle the result and the thread on which the request runs. Inside the callback, it verifies that *locationResult* is not empty and retrieves the last known location using *locationResult.getLastLocation()*. The retrieved location is then passed to a listener callback method via *listener.onSuccess(location)*. Once the location is obtained, the app proceeds to sort the hospital list accordingly.

5.2.4 MQTT

In the event of an accident where the user cannot access their phone, the app will automatically publish an **MQTT message** to a broker. This **broker** functions as an intermediary, connecting users in need of assistance with hospital servers.

In this setup, the app acts as a publisher during an accident, sending a message on the topic **"hospital/emergencies"**. The message contains the user's profile information and the accident's location. Upon successfully sending this message, the app displays a notification to inform the user that nearby hospitals have been alerted and an ambulance is being dispatched.

On the receiving end, a Python script simulates the hospital server by subscribing to the same topic. It waits for 10 seconds before publishing a response message on topic **"hospital/dispatch"**, which the app listens for. When the app receives this confirmation, it notifies the user that an ambulance is on route.

5.2.5 Shared Preferences

In the development of the application, we utilized Android's SharedPreferences to store and manage key pieces of user data and application status. While SharedPreferences is typically suited for lightweight data storage, it provided a convenient and efficient way to save both user-specific information and application state settings. Below, we detail how SharedPreferences is used in the application.

- **User Information:** SharedPreferences is employed to store personal information provided by the user, as well as the security questions and answers they configure. This approach enables the application to quickly retrieve user details and verify their identity if needed, without querying a remote database. This data includes:
  - *Personal Details:* name, surname, gender, blood type, birthdate, and phone number.
  - *Safety questions's answers:* To save user's answers to predefined questions.
- **Application State Management:** SharedPreferences also tracks various aspects of the application's state, ensuring smooth user experience by avoiding redundant operations:
  - *Hospital Data Download Status:* A boolean entry (hospitalsDownloaded) indicates whether hospital data has already been downloaded, preventing unnecessary downloads on subsequent app launches.
  - *First Start Status:* Another boolean entry (firstStart) is used to detect if the application is being opened for the first time, enabling initial setup processes.

5.2.6 Question section

When the phone detects a crash, a pop-up is displayed regardless of the smartphone's mode or state. This functionality is enabled by the **PowerManager** API, which allows the application to wake the screen, ensuring the pop-up is visible even if the phone is locked or in sleep mode.

To further enhance responsiveness, the app utilizes the **WindowManager** API to unlock the phone when the pop-up is triggered. Each pop-up is launched in a separate thread, preventing potential application crashes and ensuring high responsiveness during critical operations.

In parallel with the pop-up display, the application starts two additional threads to control the phone's camera flash, making it blink every 500 milliseconds. This feature is implemented using the **Camera** API, which provides the capability to manipulate the phone's flashlight. Additionally, the background color of the pop-up can be dynamically adjusted during this process.

The content of the pop-up is generated based on data stored in the **SharedPreferences** archive [5.2.5]. This archive not only holds user preferences but also stores predefined questions and answers, enabling the creation of a quiz-like interface within the pop-up. The stored data includes both correct and incorrect answers, ensuring a versatile and engaging user experience.

```
PowerManager powerManager = (PowerManager) getSystemService(POWER_SERVICE);
PowerManager.WakeLock wakeLock = powerManager.newWakeLock(
        levelAndFlags: PowerManager.SCREEN_BRIGHT_WAKE_LOCK | PowerManager.ACQUIRE_CAUSES_WAKEUP,
        tag: "CrashSimulator::CrashAlertWakeLock"
);
wakeLock.acquire( timeout: 10000);
```

```
Window window = getWindow();
window.addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON
        | WindowManager.LayoutParams.FLAG_TURN_SCREEN_ON
        | WindowManager.LayoutParams.FLAG_SHOW_WHEN_LOCKED
        | WindowManager.LayoutParams.FLAG_DISMISS_KEYGUARD);
```

Figure 7: Snippets Power Manager and Window API

5.2.7 JSON Parsing

To provide hospital data to the user, the free online JSON from Datos Madrid [3] is downloaded once per app lifecycle. Extra logic may be implemented to re-download data as fixed or dynamic intervals to update the list. To track whether the data has already been downloaded, a boolean flag is stored in the app-level shared preferences.

If this flag is not found, a download request is initiated on a background thread. The data is then stored locally using **Room DB**, an Android persistence library built on top of **SQLite**, saving it in the device's internal storage as an SQLite file. The parsing of this JSON file is done using the **JSONObject class**[Figure: 8].


```
"@graph": [

{
    "@id": "https://datos.madrid.es/egob/catalogo/tipo/entidadesyorganismos/5855213-centro-concertado-adicciones-ccad-centro-cruz-roja-.json",
    "@type": "https://datos.madrid.es/egob/kos/entidadesYorganismos/CentrosAtencionMedica/CentrosAsistenciaDrogodependientes",
    "id": "5855213",
    "title": "Centro Concertado de adicciones (CCAD) Centro (Cruz Roja)",
    "relation": "http://www.madrid.es/sites/v/index.jsp?vgnextchannel=bfa48ab43d6bb410VgnVCM100000171f5a0aRCRD&vgnextoid=c5dc3e042111c210VgnVCM2000000c205a0aRCRD",

    "address": {
        "district": {
            "@id": "https://datos.madrid.es/egob/kos/Provincia/Madrid/Municipio/Madrid/Distrito/Centro"
        },
        "area": {
            "@id": "https://datos.madrid.es/egob/kos/Provincia/Madrid/Municipio/Madrid/Distrito/Centro/Barrio/Justicia"
        },
        "locality": "MADRID",
        "postal-code": "28004",
        "street-address": "CALLE HERMANOS ALVAREZ QUINTERO 3"
    },
    "location": {
        "latitude": 40.42747404776084,
        "longitude": -3.697485377509971
    },
    "organization": {
        "organization-desc": "El Centro concertado de adicciones Centro forma parte de la red del Instituto de Adicciones de la Ciudad de Madrid, presta una
atención personalizada individual y/o grupal en régimen ambulatorio a las personas que presentan problemas de abuso o dependencia del alcohol y de otras drogas. Forma
de acceso: directo Metro: Alonso Martínez, L4, L5, L10",
        "accesibility": "1",
        "schedule": "De lunes a viernes de 9 a 17 horas",
        "services": "Valoración interdisciplinar del/de la paciente. Tratamiento integral que además de abordar el propio consumo, contempla aspectos sanitarios,
psicológicos, sociales y ocupacionales. Intervenciones de reducción del daño y riesgos dirigidas a disminuir los efectos del consumo de drogas y conseguir mejoras en
la salud cuando no se consigue la abstinencia. Tratamientos farmacológicos, incluyen la utilización de tratamientos con sustitutivos opiáceos como metadona. Apoyo a la
integración social y laboral del/de la paciente. Intervención con las familias, tanto con las que acompañan a la persona directamente afectada, como con aquellas otras
que solicitan ayuda sin que el consumidor haya decidido ponerse en tratamiento. Derivación a recursos externos de apoyo al tratamiento y a la reinserción social, que
complementan y apoyan el trabajo realizado desde los CAD.",
        "organization-name": "Centro Concertado de adicciones (CCAD) Centro (Cruz Roja)"
    }
},
```


Figure 8: Snippet of hospitals JSON

Therefore, the background thread iterates through the **@graph** object, extracting details such as the hospital's name ("title"), street address and latitude and longitude. For each entry,

a new **HospitalEntity** is created and added to a local list. Once the iteration is complete, the entire list is saved to the database.

5.2.8 APK

The code and the APK of the application can be found in GitHub: [2].

6 System Testing and Validation

We took advantage of the Figma prototype cited in Section 4 to present the application's design and functionality during the elevator pitch in class. The demonstration provided an opportunity for peer and instructor feedback, serving as an initial validation of the app's user interface and feature set. This validation process confirmed that the overall design and user flow aligned with the project goals and user requirements.

Due to time constraints, the testing and validation process for the application was limited. However, we ensured that the core features of the app were manually tested to verify their functionality. The testing focused on confirming that the app performed as expected under various conditions and that key features, such as crash detection, emergency alerts, and hospital location retrieval, were operational.

6.1 Manual Testing Approach

- **Emulator Testing:** The application was first tested on an Android emulator to simulate different devices and screen sizes. This allowed us to assess the app's general functionality, UI responsiveness, and behavior under typical user interactions.
- **Device Testing:** To ensure the app worked well on actual hardware, we tested it on our lent device, the BC Aquaris V. This device was used to validate real-world performance, including: **Crash Detection:** Ensuring the accelerometer properly detected simulated crashes. **Emergency Response:** Verifying that the app correctly triggered emergency alerts and notifications. **Location Features:** Confirming that the app accurately displayed nearby hospitals based on the user's location.

6.2 Limitations

Given the limited time for testing, the app was not subjected to extensive automated tests or stress tests. Edge cases and performance under varying network conditions were not fully evaluated, and further testing is required to identify and resolve potential issues, particularly for offline functionality and crash detection accuracy.

7 Conclusions and future enhancements

RideSOS was developed with the goal of enhancing the safety and well-being of motorcyclists and bicyclists through an intuitive and reliable mobile application. The app successfully integrates features such as crash detection, emergency response automation, and hospital navigation, offering a comprehensive safety solution for the needs of riders.

Through its implementation of accelerometer-based monitoring, real-time emergency notifications, and user personalization features, RideSOS provides an essential safety net in situations where immediate assistance can make the difference between life and death.

However, the application also revealed areas for improvement, particularly in terms of detection algorithm robustness and expanded accessibility features.

7.1 Future Enhancements

While RideSOS achieves its primary objectives, several enhancements could further improve its efficiency, usability, and reliability:

- **Improved Crash Detection Algorithm:** Developing a more sophisticated and efficient detection algorithm would enhance the accuracy of crash detection. This could involve integrating machine learning techniques to reduce false positives and negatives, providing a more reliable safety service.
- **Text-to-Speech Integration for Emergency Calls:** Adding a text-to-speech feature for emergency calls would make the app more accessible and user-friendly, especially in situations where the user is unable to speak. This feature could automatically communicate critical details, such as the user's location and condition, to emergency responders.
- **Enhanced Database Management:** Moving from SharedPreferences to a more robust database system (e.g., SQLite or Room) for storing user data would improve scalability and data integrity, particularly for applications with expanding feature sets.
- **Integration with Wearable Devices:** Future versions could integrate with wearable devices such as smartwatches to provide additional crash detection mechanisms and ensure constant monitoring, even when the phone is not in use.
- **MQTT Integration:** An improvement for the MQTT system could involve sending the accident notification initially to the 10 nearest hospitals. If no response is received on the "hospital/dispatch" topic within a set time frame, the app would send the message to the next 10 hospitals, continuing this process until a response is received. Once an ambulance dispatch is confirmed, the app will stop sending further messages or unsubscribe from the topic. Similarly, hospital subscribers should have a coordination mechanism. When one hospital confirms ambulance dispatch, it will publish a message on an internal topic shared among hospitals. This message will indicate that the hospital is handling the incident, preventing other hospitals from responding to the same emergency.

By incorporating these enhancements, RideSOS could further establish itself as a vital tool for improving rider safety, making it a reliable companion for users in diverse scenarios.

References

[1] Stefano Morano Ana Daria Corpaci, Antonio Franzoso. Ridesos figma. https://www.figma.com/design/RQUeHKqQcJsCnpfm3DdWol/RideSOS?node-id=0-1&t=mGs6iBn3geUwGe53-1, 2024. Accessed: 2024-11-18.

[2] Stefano Morano Ana Daria Corpaci, Antonio Franzoso. Ridesos github. https://github.com/stefano-morano/SOSRide, 2024. Accessed: 2024-11-18.

[3] Ayuntamiento de Madrid. Portal de Datos Abiertos - Hospitales de Madrid, 2024. Accessed: 2024-11-18.