UNIVERSIDAD POLITÉCNICA DE MADRID

TELECO. CAMPUS SUR-UPM ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y SISTEMAS DE TELECOMUNICACIÓN

RideSOS

Mobile Devices Programming

Corpaci Ana Daria, Franzoso Antonio, Morano Stefano

RideSOS was born from the desire to help people in case of motorcycle/bicycle accidents, monitoring them to detect potential crashes, and therefore providing instant support.

User Requirements

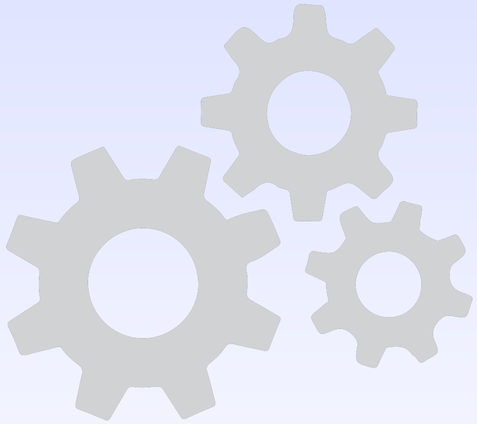
Functional

- Crash Detection and Emergency Response
- Background Operation
- Nearby Hospital Information
- Profile Setup and Personalization
- User-Friendly & Accessible Interface

Non-Functional

- High Reliability and Accuracy
- Offline Functionality

Main Features



Sensors

App uses the following sensors:

- **Accelerometer:** for crash detection
- **GPS:** use of the location to create the emergency MQTT message.

Accessibility

- Starting a new ride displays an icon in the notification bar, reassuring users that RideSOS is actively monitoring their journey.
- Simple interface, clear navigation and large, easy-to-read text, buttons and sounds.

Internet Functionality

The app downloads the official Madrid Hospital List in JSON format parsing it.

MQTT

- **Publish** - During a crash, if the user needs help, app will publish a MQTT message containing his info and location
- **Subscribe** - When an ambulance has started moving towards the user, app will receive a MQTT message.

App Design

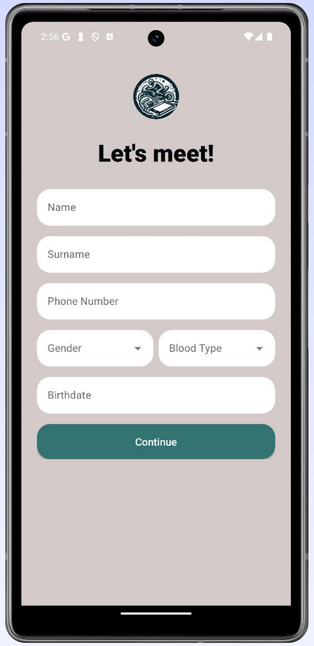
The whole design revolves around a bottom navigation bar-based layout, where three main features are accessible everytime:

- Home (1 activity)
- Hospital (1 activity)
- Profile (1+2 activities)

But the application has also other activities:

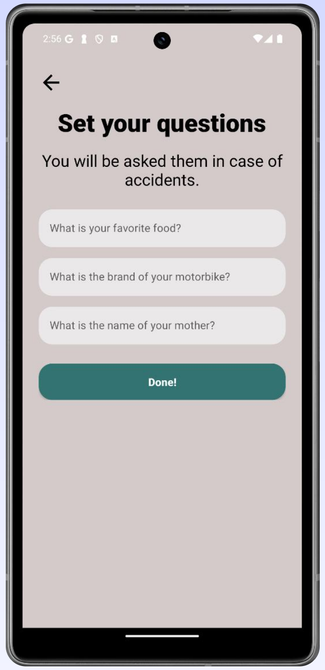
- First-Time Signup (2 activities)
- Crash Pop-Up (2 activities)

First-time Signup



Default

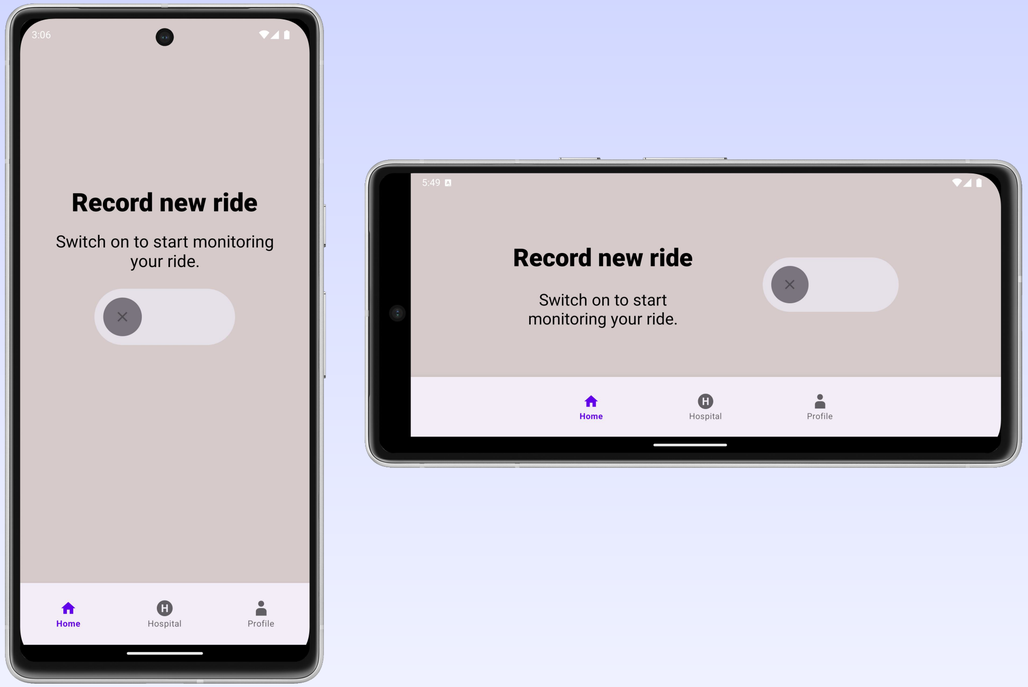
First time signup activity, where you will be asked to insert some personal informations. Every personal data will be stored in the app thanks to the *Shared Preference*.



Set Safety Answers

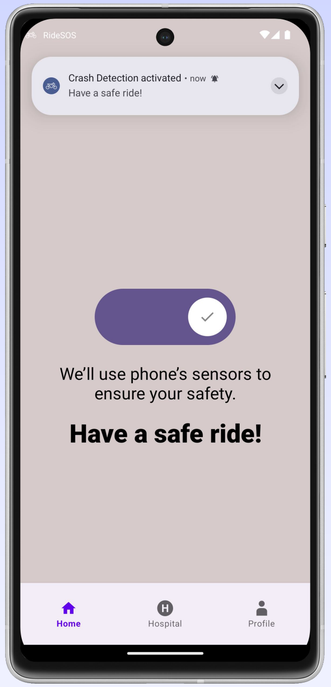
You can set the answers to a set of predefined safety questions.

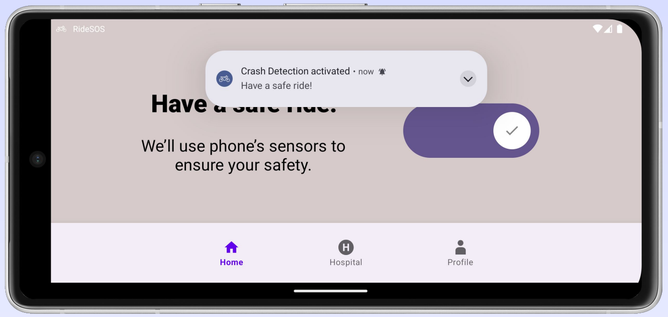
Home



Switch Off

Sensor’s tracking disabled: app will not be able to detect a crash.



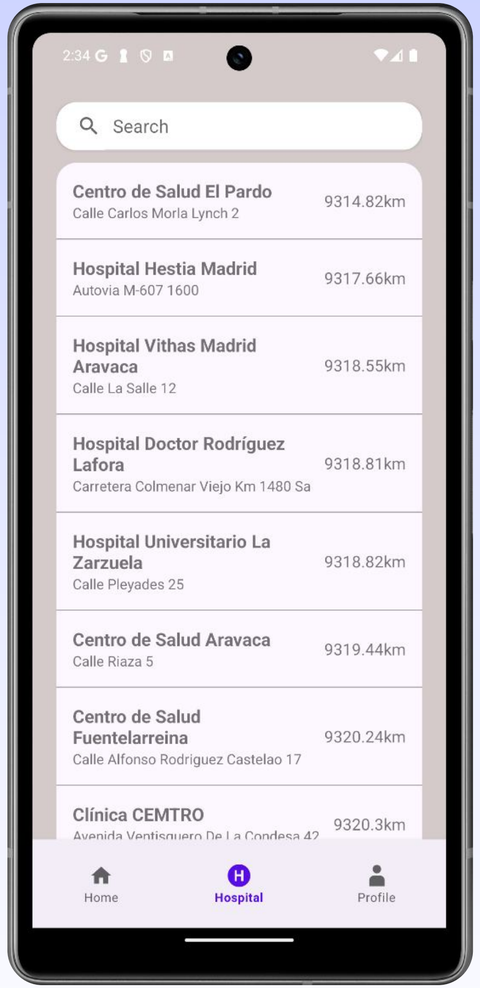


Switch On

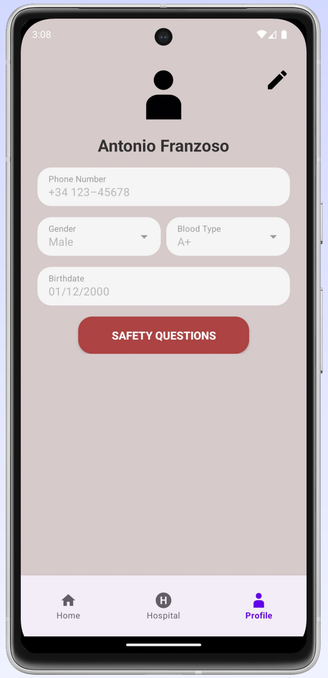
Sensor’s tracking enabled: app will detect a crash.

Hospitals

Show hospitals, sorting them based on distance to user's current location.
Clicking them will redirect you to the hospital's website.

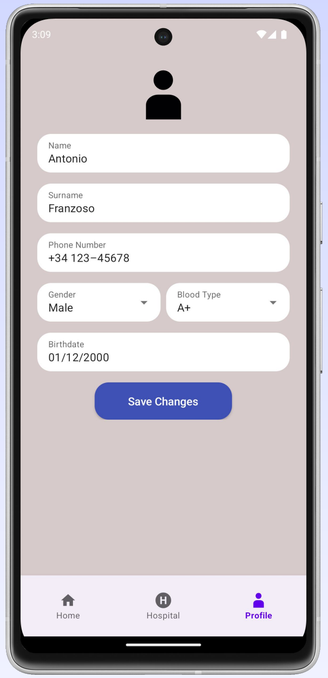


Profile



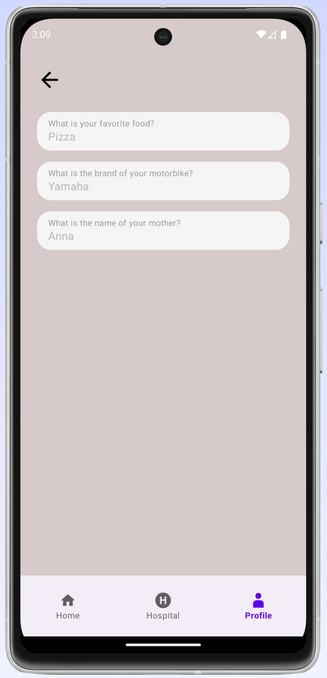
Default

To check your profile informations, allowing you to edit them and to review your safety questions.



Edit Profile

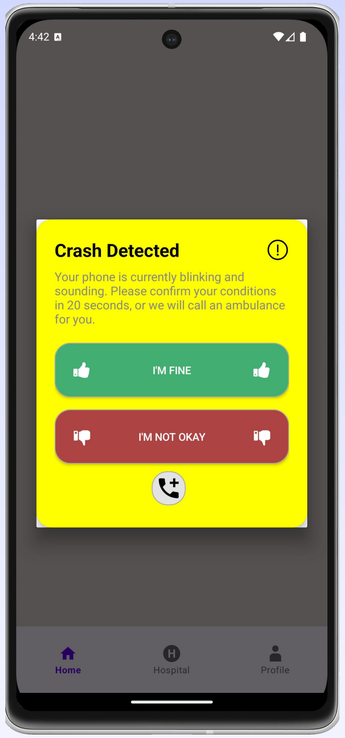
This activity allows you to edit your personal informations and save them.



Safety Questions

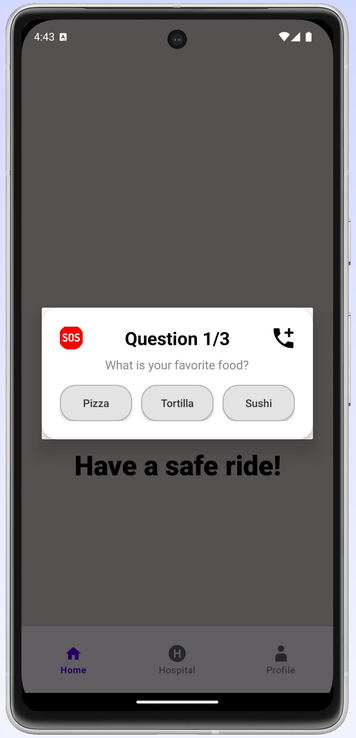
To check your previously defined safety questions and answers.

Crash Pop-Up



Crash Pop-Up

Will appear when app has detected a crashing: the app will automatically wake up and lock the phone to show the alarm popup window that will start changing color every second, emitting sounds and flashing camera light.



Questions Pop-Up

If after a crash the user select that he is fine, this popup will appear and will ask him to select the correct option to make sure he didn't lose consciousness.

Crash Detection Algorithm

If Home's switch is active, app will continuously reads accelerometer values every 500ms thanks to an Accelerometer Service running in a background thread that has the maximum level of importance hierarchy. For this reason, it works also when the phone is locked.

The algorithm works as follows:

- Calculate acceleration magintude:
  - $\sqrt{(x^2 + y^2 + z^2)}$
- Compares current and previous magnitudes against defined thresholds.
- Triggers alerts for quick emergency responses.

```
private void readAccelerometerData() {  1 usage
    readingSensorThread = new Thread(() -> {
        boolean isInterrupted = false;
        Log.d(TAG,  msg: "Reading accelerometer data");

        while (!isInterrupted) {
            try {
                float[] values = accelerometerSensor.getAccelerometerValues();

                if (detectCrash(values, lastMagnitude)) {
                    Log.d(TAG,  msg: "Crash detected!");
                    onCrashDetected();
                }

                lastMagnitude = calculateMagnitude(values);

                Thread.sleep(SENSOR_DELAY_MS);
            } catch (InterruptedException e) {
                isInterrupted = true;
            }
        }
    });
    // Launch the thread
    readingSensorThread.start();
}
```

```
private boolean detectCrash(float[] values, float lastMagnitude) {  1 usage

    float currentMagnitude = calculateMagnitude(values);

    if (lastMagnitude > CRASH_THRESHOLD && currentMagnitude < BRAKING_THRESHOLD) {
        return true; // Sudden deceleration detected
    }

    return false;
}
```

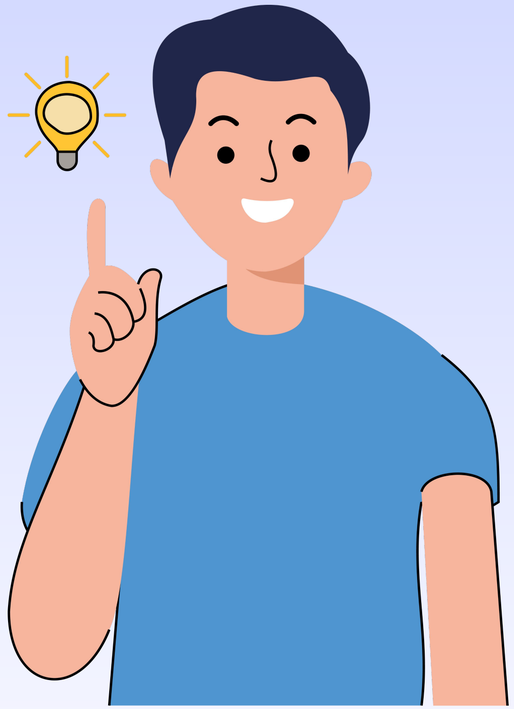
MQTT-Based Emergency System

Automatically connects users in accidents with hospitals using MQTT. When accidents occurs and user's needs help, workflow is the following:

- **App as Publisher:**
  - sends an MQTT message to the topic "*hospital/emergencies*" that includes user profile and accident location. Displays notification on user's phone.
- **Hospital Server Simulation:**
  - Python script subscribes to "hospital/emergencies". After 10 seconds, publishes a response on "hospital/dispatch".
- **App as Subscriber:**
  - Receives MQTT confirmation message. Displays notification on user's phone.

```python
from time import sleep

import paho.mqtt.client as mqtt

BROKER = "localhost"
SUBSCRIBE_TOPIC = "hospital/emergencies"
PUBLISH_TOPIC = "hospital/dispatch"


def on_connect(client, userdata, flags, rc):
    if rc == 0:
        print("Connected to MQTT Broker!")
    else:
        print("Failed to connect, return code %d\n", rc)


def on_message(client, userdata, msg):
    print(msg.payload)
    sleep(10)

    client.publish(PUBLISH_TOPIC, "Ambulance is on its way!", qos=2)

client = mqtt.Client()
client.on_message = on_message
client.on_connect = on_connect
client.connect(BROKER, 1883)
client.subscribe(SUBSCRIBE_TOPIC)
client.loop_forever()
```

Future Enhancements



- Improved crash detector algorithm
- Enhanced Database Management
- Text-To-Speech for emergency call
- Integration with wearable devices
- MQTT Improvement

Now it’s demo time!

We hope you’ll like it!