UNIVERSITY OF THE PELOPONNESE & NCSR “DEMOCRITOS”
MSC PROGRAMME IN DATA SCIENCE

Sodasense: A Framework for Collecting and Managing Data from the Sensors of Mobile Devices

by

Athanasios Vakouftsis

A thesis submitted in partial fulfillment
of the requirements for the MSc
in Data Science

Supervisor: Spiros Skiadopoulos
Professor

Athens, July 2023

UNIVERSITY OF THE PELOPONNESE & NCSR "DEMOCRITOS"
MSC PROGRAMME IN DATA SCIENCE

Sodasense: A Framework for Collecting and Managing Data from the Sensors of Mobile Devices

by

Athanasios Vakouftsis

A thesis submitted in partial fulfillment
of the requirements for the MSc
in Data Science

Supervisor: Spiros Skiadopoulos
Professor

Approved by the examination committee on July, 2023.

(Signature)	(Signature)	(Signature)
....	....	....
Spiros Skiadopoulos Professor	Christos Tryfonopoulos Associate professor	Theodoros Giannakopoulos Researcher

Athens, July 2023

Declaration of Authorship

(Signature)

. . . .

Athanasios Vakouftsis

Athens, July 2023

Acknowledgments

With the completion of this thesis, in the context of the postgraduate program that I attended at the National Centre for Scientific Research Demokritos in collaboration with the University of Peloponnese (M.Sc. in Data Science), as a sign of gratitude for his valuable help, I would like to thank my supervising professor Spiros Skiadopoulos who entrusted me this subject for this thesis and Konstantinos Vasilakis for his valuable help on providing me ideas. I would also like to warmly thank my family for the psychological support they provided, as well as my friends, the fellow students for their suggestions and ideas, which helped in the implementation of this dissertation and especially my friends from the undergraduate degree for their on point advices. Finally, I would like to thank the colleagues who contribute to open source software for helping me create a more robust and polished software.

To my family.

Περίληψη

Σκοπός της εργασίας αυτής είναι ο σχεδιασμός και η υλοποίηση ενός συστήματος για την συλλογή, την διαχείριση, την επεξεργασία και την αποθήκευση δεδομένων μετακίνησης. Αρχικά περιγράφουμε σύντομα το πρόβλημα και τη λύση την οποία προσφέρουμε. Έπειτα αναλύουμε τα εργαλεία πολλαπλών πλατφορμών στα οποία αποφασίσαμε να υλοποιήσουμε μία δοκιμαστική εφαρμογή για να δοκιμάσουμε τις δυνατότητες του κάθε εργαλείου και την εμπειρία χρήσης τους, τα πλεονεκτήματα και τα μειονεκτήματά τους και τους λόγους που απορρίψαμε κάποια από αυτά. Μετά αναλύουμε την αρχιτεκτονική του εργαλείου στο οποίο αναπτύχθηκε η εφαρμογή, στα δομικά του μέρη όπως τον τρόπο με τον οποίο δημιουργεί τη διεπαφή, την ενσωμάτωση του κώδικα σε δύο διαφορετικά λειτουργικά συστήματα και τα εργαλεία που προσφέρει για την δημιουργία της εφαρμογής. Στη συνέχεια παρουσιάζουμε τις δυνατοτήτες της εφαρμογής η οποία λειτουργεί σε φορητές συσκευές που έχουν λειτουργικό σύστημα Android και IOS. Ο χρήστης θα συνδέεται στο σύστημα και θα διαμοιράζει τα δεδομένα των μετακινήσεών του. Τα δεδομένα αυτά θα αποθηκεύονται σε ένα κεντρικό σύστημα το οποίο θα τα επεξεργάζεται και θα τα διαχειρίζεται. Ακολούθως, μελετάμε ενδελεχώς την ανάπτυξη όλων των λειτουργιών της εφαρμογής, των διαφόρων οθονών, της διεπαφής που βλέπει ο εκάστοτε χρήστης και τα διάφορα προβλήματα που προέκυψαν και τον τρόπο αντιμετώπισής τους. Επίσης, αναλύουμε τον τρόπο με τον οποίο η εφαρμογή συνδέεται στη βάση δεδομένων που δημιουργήσαμε και επιλέξαμε μετά από έρευνα καθώς και στον τρόπο με τον οποίο τα δεδομένα απόστελονται δημιουργώντας μια δική μας λύση. Τέλος αναφέρουμε τα συμπεράσματα που προέκυψαν από την ανάπτυξη ολόκληρου του συστήματος που δημιουργήσαμε από την αρχή, τα προβλήματα που δημιουργήθηκαν κατά την δημιουργία του συστήματος και τέλος τους στόχους που έχουμε θέσει για την περαιτέρω ανάπτυξη της λύσης που δημιουργήσαμε.

Abstract

The aim of this thesis is to design and implement a Framework for collecting, managing and processing movement data. Initially, a brief description of the problem and the solution we offer. Then we analyze the cross-platform tools we decided to implement a test application to investigate each tool's capabilities and user experience, their advantages and disadvantages, and the reasons we rejected some of them. After, we examine the architecture of the tool in which the application was developed, its structural parts such as the manner it creates the interface, the integration of the code in two different operating systems and the tools it offers to create the application. Next, we present the features of the application which works on mobile devices that have Android and IOS operating systems. By using the application the user will be able to log into the system and share the data of their movements. The data will be stored in a central system which will process and manage them. Subsequently, we thoroughly study the development of all the functions of the application, the various screens, the user interface that each user sees and the various problems that arose and how to solve them. Later, we analyze how the application connects to the database system we created and selected after research and how the data is sent by creating a custom solution. Finally, we report the conclusions that emerged from the development of the entire framework that we created from scratch, the problems that arose during the creation of the Framework and finally the goals that we have set for the further development of the solution that we created.

Contents

List of Tables

List of Figures

List of Abbreviations

IDE	Integrated Development Environment
UI	User Interface
PC	Personal Computer
SDK	Software Development Kit
OS	Operating System
GPS	Global Positioning System

Chapter 1

Introduction

In modern age the technology rapidly evolves every day and we have reach the point where a mobile device is more powerful compared to desktop computers from 15 years ago. The mobile devices have gone from a point where 15 years ago their primary functionality was to make calls and send messages. Today mobile devices can connect to the internet, have a camera at the front and at the back to take photos with flash, the keypad was replaced with touch displays and have the ability to recognise a person face through facial recognition with the help of the camera. Fingerprint sensors are also common and there are sensors like accelerometer, gyroscope, magnetometer and pedometer. The GPS is also included from early development of the first smart devices, so each device can recognise the geographical point of the user from the coordinates longitude and latitude. Finally, sensors like barometer, digital thermometer or proximity are common too. At the market most of the devices has at least the following sensors:

Accelerometer. It is a sensor that tracks the acceleration of the device at each of the x, y and z axis.

Gyroscope. It is a sensor that tracks the rotation of the device at each of the x, y, and z axis.

Magnetometer. It is a sensor that tracks the magnitude and direction of the Earth's magnetic field in the x, y, and z direction.

GPS (Global Positioning System). It is a sensor that register the real time location of a device with longitude and latitude.

Pedometer. It is a sensor that tracks the steps of device user by the abrupt movement of the device.

Proximity. It is a sensor that detects when a user is holding the device closer than 1cm.

Barometer. It is a sensor that tracks the atmosphere pressure that the device is in.

Some devices also have bio-metric sensors (for monitoring the temperature and the heart rate of the user), ambient light sensor or even humidity sensor. Based on the availability of the sensors, there are multiple applications that utilize them in order to give the users the metrics and the data that produce. The companies that have such applications, use these sensors to monitor the activity of each user to create a more focused product for them, but sometimes users do not know about the usage of the data from the sensors concluding that the companies use the data with unethical ways and are not following the proper ways of user's security and privacy.

1.1 Purpose - Problem description

The purpose of this thesis was to create a complete framework called Sodasense to track the user's data, send and save them to off-site database. So the framework was divided in three parts: the first one was to create a mobile application for Android and IOS operating systems, that tracks with accuracy the movement of the user, the environment that the user is currently in, the route that the user has followed, the distance and the real time speed of the user. The available sensors of each device contribute to this goal. The second part is to find the proper technique to find and set up an authentication and authorization service that will keep track of our users and build web-services which will be responsible for sending and receiving data from the application to the database on our university server and the third part is to find the most suitable database for our framework and set it up to save and send the data that are created from the application. There was the need of creating an application on both Android and IOS that uses all of the sensors of the device alongside with the GPS to register useful data about the environment and the movement of the user. All of the data then will be available to the user without any compromises. The aim is to respect user's data by being transparent about the data the device collects by having prompts for permissions. For better understanding starting of what we have to do, is to create an application that the user must connect with credentials in order to create a secure connection between the application and the database. The application must take full advantage of each sensor available on the device (e.g. accelerometer, gyroscope, magnetometer, proximity, pedometer, barometer) and the GPS sensor to track the real time location of the user, The data must be store first locally on a database in the device and when the device is connected to the internet it should send the save data automatically on the database on our server. Also the application should contain a map to illustrate the user's location and also the route that is registered selectable by a calendar. Likewise, a compass must be included and a place where the user can see the available sensors of the device and their real time metrics of each sensor. A settings menu is also required on every application in our days, that has all the basic functionalities like changing the sampling rate of the sensors, changing the daily target steps and the height of the user and also a button to extract the local database to a folder of the device in order for the user to have full control of the created data. The application must work in the background if the user also wants to have another application open in order for our application to collect data and must be offline first (should work flawlessly when is not connected to the internet). The middle layer of the framework (the web-services that will be designed and created from scratch without the use of any framework) should be responsible to securely connect the user to the authorization/authentication system that we will research and set up for our needs and the connect to our database and also exchange data to and from our database. Finally, we must select the most suitable database to use on the university server and configure it to connect and save data properly.

1.2 Thesis structure

Starting with Chapter 2, a reference is made to all the Software Development Kits and the Frameworks that are available for building applications on both Android and IOS. Also a comparison to their pros and cons and which SDK's we selected to create the same application on every SDK called DEMO application. Another reason to create the DEMO was to test the capabilities of each SDK and what does it take to be built on each one. Finally we list the arguments of our choice to select Flutter over every other SDK.

In Chapter 3 we take a deep look of Flutter SDK's architecture, we analyze how it is built, what are the widgets, the rendering technique with which the UI is made by using its custom built engine. We describe Flutter's connection to the native code using custom integration techniques for each supported platform.

In Chapter 4 we have an in detail look at the structure of the whole framework, how each layer communicate with the each other and the way our framework works. We compare the two best Integrated Development Environments (IDE), what they offer each one to help us and what we chose. Finally, we analyze the testing on devices and the manner that was done by configuring the Flutter on each OS.

In Chapter 5 we have a quick demonstration of the application. We have a tour of what are the functionalities of the application, what a user should expect to see from every screen (Login, Registration, Main, Route, Compass, Sensors and Settings) and how to interact with every UI object. Also we give instructions of the use of the application and what actions to make in order to take full advantage of all the capabilities of the application.

Chapter 6 is the biggest section of work because we have a complete analysis of the development of the whole framework, the application, the middle layer which is the web-services and the hosted database. We start by analyzing the development of how we built the application, the packages we used and the various functionalities we created in every screen. Then we proceed to the selection of the databases (both on the local and the server side) and then we proceed to the creation of the web-services, the way we build them and are their purpose.

Finally, in Chapter 7 we have the conclusion of this work, the problem we faced and solved and the future extensions of the framework.

Chapter 2

Cross-platform mobile applications development

In the past few years the smartphone market had 4 competitors of operating systems namely, Android, IOS, Windows phone OS and Symbian. Windows phone OS and Symbian were not able to keep up with the trends of the smartphone market. Progressively users stopped selecting them and they moved to Android and IOS. Additionally, the development for mobile applications were not easy for programmers so they stop supporting Windows phone OS and Symbian and likewise, they moved to developing mobile applications for Android and IOS. The native language for developing applications in IOS is Swift. Likewise, Android applications are natively build in Java [2, 3]. Developing the same application for Android and IOS simultaneously on their native languages is very time consuming. Each platform has its own difficulties and its unique ways of building the same application, so a developer must know both platforms in depth in order to make the same application. Fortunately, with the evolution and creation of new technologies the development on both platforms has become easier by creating a single project focusing on the execution of both platforms on the same time using software development kits (SDK). There are various and different software development kits available providing different capabilities to compile code with native design of each platform or with web design packaged into a mobile application [4, 5, 6, 7, 8]. Table 2.1 shows some popular SDK's that developers use.

In this thesis we consider to compare the Ionic, Xamarin, Cordova and Flutter SDKs since they are the most popular choices among cross-platform tools for development.

SDK	Programming languages
Cordova [9]	HTML, CSS, Javascript
Ionic [10]	Javascript, Angular, React, Vue
Native Script [11]	Javascript, Typescript, Angular, Vue, React, Svelte, Capacitor, HTML
React Native [12]	Javascript
Flutter [13]	Dart
Xamarin [14]	.Net Framework, C#
Felgo [15]	Javascript, C++, QML
Rho Mobile [16]	Ruby, HTML, CSS, Javascript
Sencha [17]	ExtJS
Framework7 [18]	HTML, CSS, Javascript, Vue, React, Svelte
Jasonette [19]	Javascript, Json

Table 2.1: Software development kits with their programming languages

2.1 General information and installation of SDKs

In order to install and test every SDK we select Windows 10 as our main operating system (OS). Thus, the following instructions regard normal Windows 10 but with slight modifications can be adopted for other OSs.

2.1.1 Cordova

We start with Cordova (formely known as PhoneGap), which is a mobile application development framework that supports Android, IOS and Web applications created by Nitobi and purchased by Adobe in 2011. The PhoneGap was in released as an open-source version of the software called Apache Cordova. Cordova uses HTML, CSS, Javascript for the user interface (UI) and the functionality. We used Visual Studio Code as our IDE.

To work with Cordova we had to install the following software:

1. The latest version of Java SDK [20]. This will offer the essential tools to work with Java language.

2. Android Studio [21]. This will offer useful command line tools, the android emulator and the Gradle which is an advanced build toolkit, to automate and manage the build process of building an Android application.

3. The latest version of Git [22]. This will offer the ability to download files and also create and build the application.

To configure the Android SDK we followed the steps of the respective documentation [23]. We opened the Android Studio and selected the Android SDK Manager (Tools → SDK Manager) and installed:

1. The desired API levels for developing in Android (from Android 7 to 12).
2. The latest version of Android SDK build tools.

To create an new application, we executed the installed launcher of Git and type the following command:

```
cordova create "name of the application"
```

The next step is to add the target mobile application platform. Specifically, for Android we type:

```
cordova platform add android
```

and for IOS we type

```
cordova platform add ios
```

2.1.2 Ionic

Ionic is a complete open-source SDK for hybrid mobile application development and supports Android, IOS and Web applications. It was created by Max Lynch, Ben Sperry, and Adam Bradley of Drifty Co. in 2013. Ionic uses Javascript, Angular, React and Vue. We used Visual Studio Code as our IDE. To work with Ionic we had to install the following software:

1. The latest version of Java SDK [20]. This will offer the essential tools to work with Java language.
2. Android Studio [21]. This will offer useful command line tools, the android emulator and the Gradle which is an advanced build toolkit, to automate and manage the build process of building an Android application.
3. The latest version of Git [22]. This will offer the ability to download files.

4. The latest version of nodejs [24]. This will offer the ability to create and build the application.

In command line of Windows 10 we entered the command:

```
npm install -g @ionic/cli
```

to install Ionic. After the download and installation is complete we could start working on the mobile application.

2.1.3 Xamarin

Xamarin is an open-source platform for building modern and performant applications for Android, IOS, tvOS, watchOS, macOS and Windows 10 applications with .NET. Xamarin is a Microsoft-owned San Francisco-based software company founded in 2011 (acquired by Microsoft in 2016) and the first version of Xamarin was also released in 2011. To install Xamarin we proceed as follows:

1. Download [25] and install the Visual Studio installer. This will offer the ability to install Visual Studio IDE.

2. We selected the desired version of Visual Studio which was the Visual Studio Community 2019.

3. We selected the package Mobile development with .Net.

After the download and installation of Visual Studio and the package is complete we could start working on the mobile application.

2.1.4 Flutter

Flutter is an open-source UI software development kit created by Google released in 2017 and supports Android, IOS, Mac, Windows, Google Fuchsia and Web applications. For Flutter the installation we perform the following steps:

1. Download the latest version [26] and we created a folder to copy the contents of the .zip file.

2. Update the path of the file to the Windows 10 environment variables by creating a link to the Flutter folder we created before in order to be accesible from the command line.

3. Entered the command

   ```
   flutter doctor
   ```

   in the terminal of Windows 10 to check if the Flutter installation is completed.

4. Install Android Studio [21]. This will offer useful command line tools, the android emulator and the Gradle which is an advanced build toolkit, to automate and manage the build process of building an Android application which is crucial part of the Android OS.

5. Install from Android Studio market the Flutter and Dart plugins in order to connect the folder we created before containing the contents of the SDK with the Android Studio.

Then we were ready to start developing the application.

2.2 Developing the demo application

To better understand the capabilities of each platform we decided to make the same application across the four platforms under consideration (namely Ionic, Cordova, Xamarin, Flutter). We chose to built a very simple application on every of these SDKs. We call this simple application DEMO. DEMO has a single button. Each time we press it an alert window appears showing the total number of button clicks within the current DEMO execution.

2.2.1 Cordova

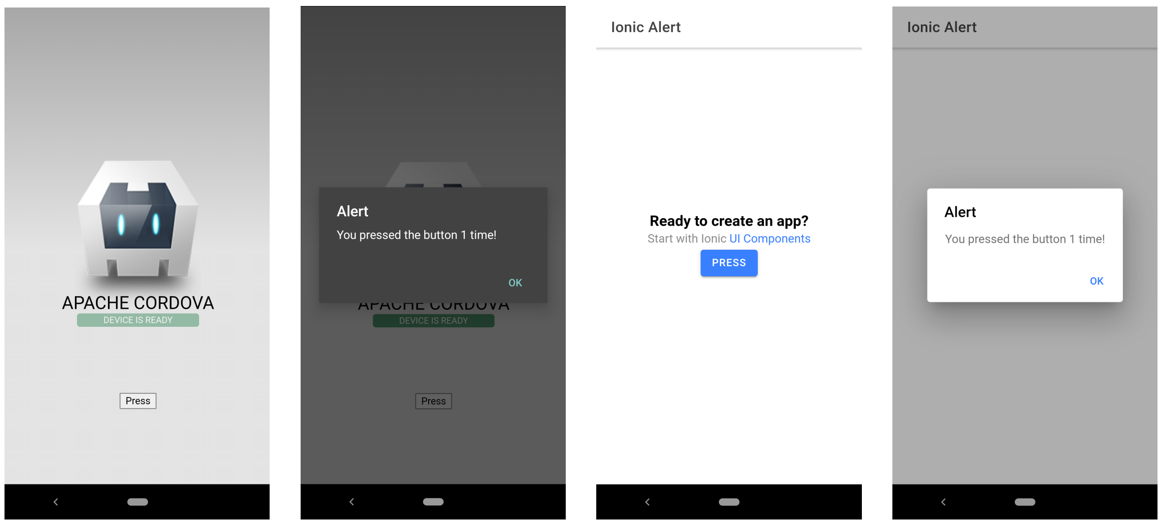
Starting with Cordova, we used Visual Studio Code as the IDE. To create a blank project we enter the command inside the terminal of Git:

```
cordova create "name of the application"
```

The above command creates a folder with all the necessary files for the DEMO to run. In order for the DEMO to be installed on mobile devices we have to add the Android and IOS platforms by typing the following commands when opening the Git and setting the path of work area inside the DEMO folder:

```
cordova platform add android
```

For IOS we type:



(a) (b) (c) (d)

Figure 2.1: Demo build using Cordova (a) and (b) and Ionic (c) and (d)

```
cordova platform add ios
```

After completing all the changes in Javascript and HTML we entered the command:

```
cordova build
```

In order to install and execute the application to a mobile device the Cordova gives two options, by building and executing automatically the application to an Android device with the command

```
cordova run android
```

and for IOS the command

```
cordova run ios
```

or (only for android) by building the .apk file that was made with the command

```
cordova  build
```

and then in order for the application to be executed the user must install manually the .apk file. The second way of testing the application is by testing it on android emulator with the command

```
cordova emulate android
```

(for the android application we used the Android Studio built-in emulator which comes pre-installed) and for IOS the testing on emulator must be done on Mac operating system. Figure 2.1 (a) and (b) shows the implementation of the DEMO application using Cordova.

2.2.2 Ionic

Ionic uses multiple frameworks for building applications like Angular, React, Vue and vanilla Javascript. We used Angular for building the application. Angular is a TypeScript based free and open-source web application framework. We created a blank project with the command:

```
ionic start myApp blank
```

so the home page was an empty HTML file. The function with the button is inside a TypeScript file which is the default file for containing the functionality of the application and then the function is called in the HTML page with an ion button which also has a container containing the text in home page. The ionic application can be deployed on an Android device with the command:

```
ionic capacitor run android
```

and on IOS device with the command:

```
ionic capacitor run ios
```

To see all the connected devices and their IDs on PC we enter the command

```
ionic capacitor run android --list
```

and to select a device we enter

```
ionic capacitor run android/ios -target="device-id"
```

Ionic has a deployment local server that shows in webview how the application would look like on a selected device, by pressing F12 we can select or even add a custom device (the webview isn't emulating the application, just how it looks from webview to a smaller screen). Ionic can also deploy applications on the same emulator as Cordova for Android with the command:

```
ionic capacitor emulate android
```

and for IOS with the command:

```
ionic capacitor emulate ios
```

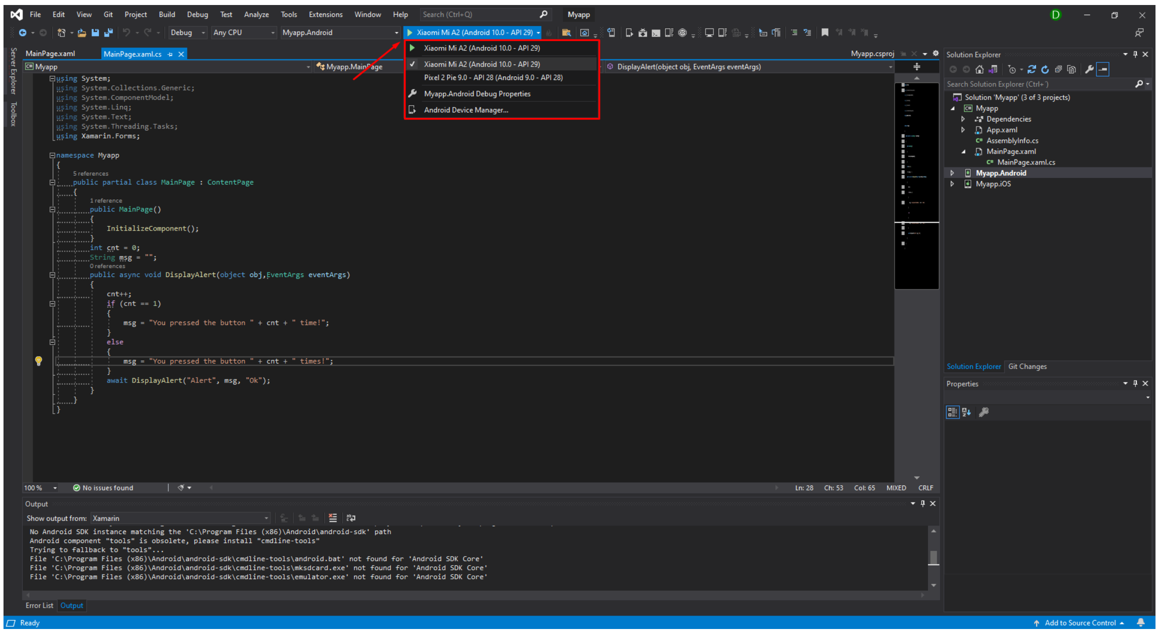
(for android application we used the Android Studio built-in emulator). Figure 2.1 (c) and (d) shows the implementation of the DEMO application using Ionic.

2.2.3 Xamarin

The development, we used Visual Studio 2019 as it is the more appropriate IDE for Xamarin development. The language we used was C# and once again we created a blank project by selecting a project type named Mobile App(Xamarin.Forms) through the Visual Studio project menu. Visual Studio creates a folder which contains three projects, the main project in which we coded the function and the button and two other projects for having specific code on every platform, even having different UI for each platform or customizing each UI to make the application look more like a native built application (Android project, IOS project). We selected the main project as our startup project and then we selected the .xaml file in which it was created the homepage of the DEMO, we added a button and then we created a function on the .xaml.cs file. For deployment Visual Studio supports deployment on Android and IOS devices or by building the .apk file for testing (only for Android devices). Visual Studio uses the emulator from Android Studio. Figure 2.2 (a) and (b) shows the location on which we can select the device or the emulator we want to deploy the DEMO for testing. Figure 2.3 shows the implementation of the DEMO application using Xamarin.

2.2.4 Flutter

The development, we used Android Studio as our IDE because it works best with it providing new layouts when creating a new project and giving many options for developing on specific platforms or all of the platforms simultaneously and also Flutter SDK has the best integration on Android Studio. We used the programming language Dart for creating the DEMO as it is the default programming language of Flutter. We created a blank project by selecting the option New Flutter Project in Android Studio and chose the boxes of Android and IOS to aim the development on these two platforms. Flutter just like any other SDK we selected creates a project


File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
Debug
Any CPU
Myapp.Android
Xiaomi Mi A2 (Android 10.0 - API 29)
Xiaomi Mi A2 (Android 10.0 - API 29)
Pixel 2 Pie 9.0 - API 28 (Android 9.0 - API 28)
Myapp.Android Debug Properties
Android Device Manager...
MainPage.xaml
MainPage.xaml.cs
Solution Explorer
Solution 'Myapp' (3 of 3 projects)
Myapp
Dependencies
App.xaml
AssemblyInfo.cs
MainPage.xaml
MainPage.xaml.cs
Myapp.Android
Myapp.iOS
Properties
No issues found
Ready

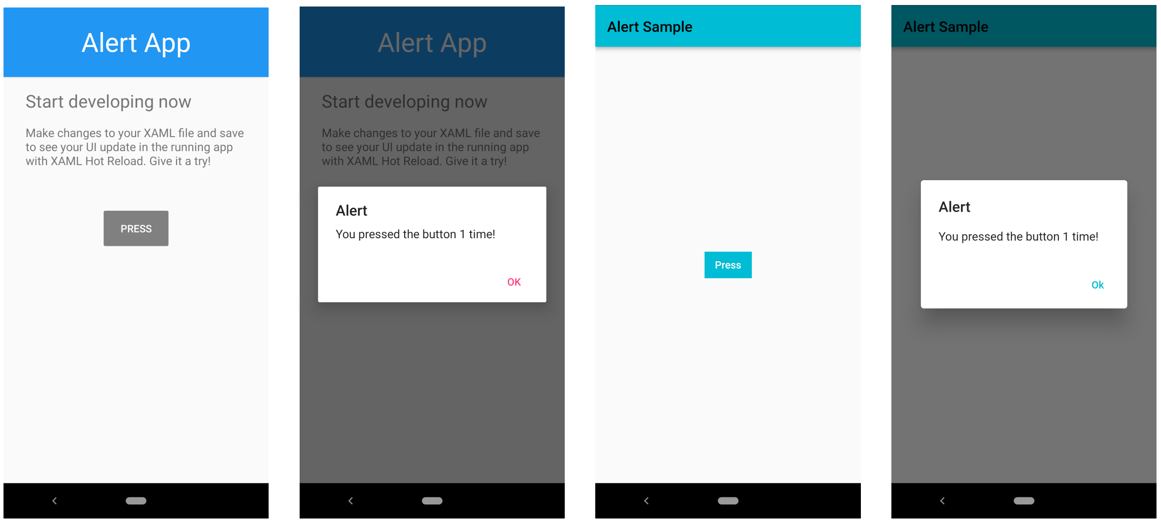
```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Xamarin.Forms;

namespace Myapp
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
        }
        int cnt = 0;
        String msg = "";
        public async void DisplayAlert(object obj,EventArgs eventArgs)
        {
            cnt++;
            if (cnt == 1)
            {
                msg = "You pressed the button " + cnt + " time!";
            }
            else
            {
                msg = "You pressed the button " + cnt + " times!";
            }
            await DisplayAlert("Alert", msg, "Ok");
        }
    }
}
```

Show output from: Xamarin

```
No Android SDK instance matching the 'C:\Program Files (x86)\Android\android-sdk' path
Android component "tools" is obsolete, please install "cmdline-tools"
Trying to fallback to "tools"...
File 'C:\Program Files (x86)\Android\android-sdk\cmdline-tools\android.bat' not found for 'Android SDK Core'
File 'C:\Program Files (x86)\Android\android-sdk\cmdline-tools\mksdcard.exe' not found for 'Android SDK Core'
File 'C:\Program Files (x86)\Android\android-sdk\cmdline-tools\emulator.exe' not found for 'Android SDK Core'
```

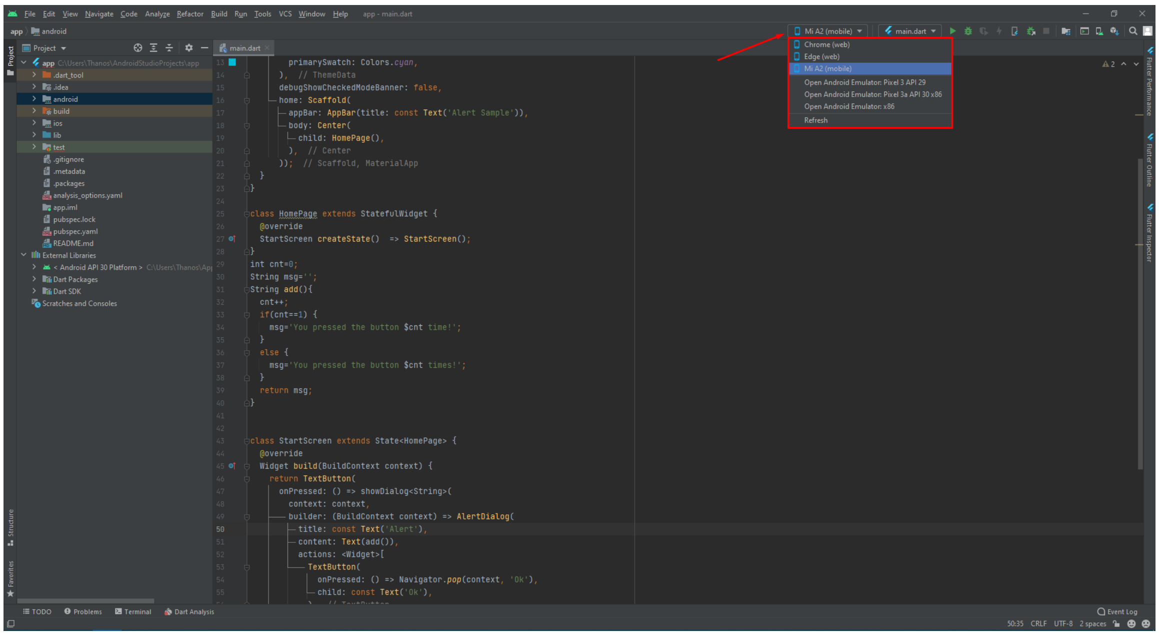
Figure 2.2: Visual Studio interface


Alert App
Start developing now
Make changes to your XAML file and save to see your UI update in the running app with XAML Hot Reload. Give it a try!
PRESS
Alert App
Start developing now
Make changes to your XAML file and save to see your UI update in the running app with XAML Hot Reload. Give it a try!
Alert
You pressed the button 1 time!
OK
Alert Sample
Press
Alert Sample
Alert
You pressed the button 1 time!
Ok

(a) (b) (c) (d)

Figure 2.3: DEMO build using Xamarin (a) and (b) and Flutter (c) and (d)

with every needed file to start creating the DEMO. Inside the lib folder of the project we created a .dart file in which we created a function for the alert and we called it within a button in the same file. Flutter supports deployment by building an .apk file (only for Android) or by connecting an Android and IOS device. For emulator Android Studio uses a built-in emulator which is used on most of the platforms for testing Android applications. On the Figure 2.4 we can see that we can select the device or the emulator we want to deploy the DEMO for testing. Figure 2.3 (c) and



```
            primarySwatch: Colors.cyan,
          ),  // ThemeData
          debugShowCheckedModeBanner: false,
          home: Scaffold(
            appBar: AppBar(title: const Text('Alert Sample')),
            body: Center(
              child: HomePage(),
            ),  // Center
          ));  // Scaffold, MaterialApp
  }
}

class HomePage extends StatefulWidget {
  @override
  StartScreen createState()  => StartScreen();
}
int cnt=0;
String msg='';
String add(){
  cnt++;
  if(cnt==1) {
    msg='You pressed the button $cnt time!';
  }
  else {
    msg='You pressed the button $cnt times!';
  }
  return msg;
}


class StartScreen extends State<HomePage> {
  @override
  Widget build(BuildContext context) {
    return TextButton(
      onPressed: () => showDialog<String>(
        context: context,
        builder: (BuildContext context) => AlertDialog(
          title: const Text('Alert'),
          content: Text(add()),
          actions: <Widget>[
            TextButton(
              onPressed: () => Navigator.pop(context, 'Ok'),
              child: const Text('Ok'),
```

Mi A2 (mobile)
main.dart
Chrome (web)
Edge (web)
Mi A2 (mobile)
Open Android Emulator: Pixel 3 API 29
Open Android Emulator: Pixel 3a API 30 x86
Open Android Emulator: x86
Refresh

Figure 2.4: Flutter interface

(d) shows the implementation of the Demo application using Flutter.

2.3 Conclusions

As we developed a simple application (app) called Demo in each one of the 4 frameworks we realised that for the project we could not continue with a web framework (like Cordova and Ionic) for the simple reason that they do not have adequate support for native libraries and it would be much harder to develop functions for getting data from the device sensors or for using what every device manufacturer can provide through the Operating System (OS). Another reason was the lack of translation to native code through an engine.

After examining Cordova and Ionic it is time to continue with Xamarin and Flutter. These frameworks provide an intermediate layer from writing code from Dart (on Flutter) and from C# (on Xamarin) translating both the written scripts to native code for each OS, Java for Android and Swift for IOS. This results of course as the application gets bigger, optimal execution even on low end devices when writing web scripts will be much slower as the application has more complicated functionalities.

Lastly both Flutter and Xamarin as they translate the written scripts on native code it can have provide native User Interface (UI) for both operating systems just by calling the already built functions on libraries. So it all come down to one of Flutter and Xamarin. Both have excellent documentation for building apps and both

Flutter	Xamarin
Easy to learn	Complete ecosystem to build many applications due to C#, **.Net**
One file contains UI and functionality	Specific files for in depth customization
Provides hot reload	Provides hot reload
High performance	High performance
Free to use	Free to use

Table 2.2: Advantages of Flutter and Xamarin

Flutter	Xamarin
Lack of third-party libraries	Slow updates of the framework
Flawed IOS support	Heavy graphics due to must have platform specific code
Platform specific optimizations must be done sometimes on each platform	Platform specific optimizations must be done sometimes on each platform
Large application size	Large application size

Table 2.3: Disadvantages of Flutter and Xamarin

have custom engines as intermediate layer, both have its pros and cons to consider before proceeding. Tables 2.2 and 2.3 shows the advantages and disadvantages of each platform [27]. After comparing Xamarin and Flutter thoroughly we realized that Flutter was the best option for the project as it has a more community based approach with loads of projects already built and is faster to build a UI as it uses widgets with dart scripts on one file compared to Xamarin which requires one .xaml file for UI and one file for writing the C# code. Flutter also uses packages to use as libraries, in which there are many already built code for run to be used.

Chapter 3

Flutter: Our developing framework

In Chapter 2 we have in detail reviewed the cross platform application SDKs. Our evaluation indicated that Flutter is the best option among the other three SDKs we tested (Ionic, Cordova and Xamarin) on both building and testing an application on Android and IOS. Let us now present the details of how the Flutter SDK is built and what does it take to translate Dart scipts in native code. The current chapter contains useful information from [1] and is adapted to the specific work.

3.1 Architectural layers of Flutter

Flutter is designed as an extensible, layered system. Figure 3.1 shows the structure of the layers [1]. It consists of three layers and no layer has privileged access to the layer below, every part of the framework layer is designed to be optional and replaceable. The first layer consists of Dart framework which contains widgets, rendering technique, cupertino and material (which are libraries that offer comprehensive sets of controls that use the widget layer's composition primitives to implement the Material or IOS design languages), animation, painting, gestures and foundation. The second layer contains the Flutter engine which is written in C# and is responsible for rasterizing composited scenes whenever a new frame needs to be painted. It provides the low-level implementation of Flutter's core API, including graphics through Skia, text layout, file and network I/O, accessibility support, plugin architecture, and a Dart run-time and compile tool-chain. Flutter applications are packaged in the same way as any other native application. The last layer is the embedder which is written in a language that is appropriate to the underlying operating system that is Java and C++ for Android, Objective-C/Objective-C++ for IOS and MacOS, and C++ for Windows 10 and Linux. Using the embedder, Flutter code can be integrated into an existing application as a module or as the entire

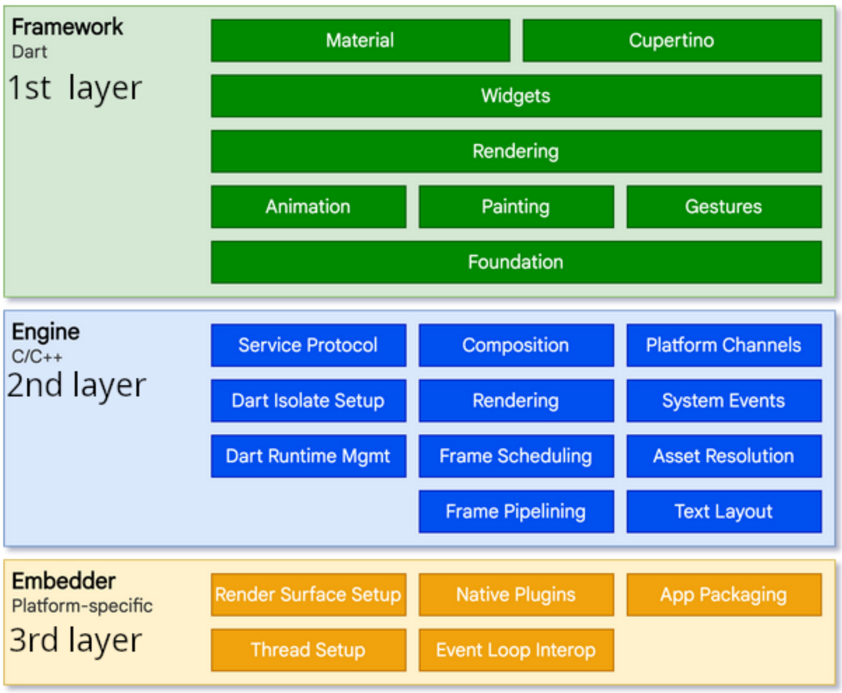


Figure 3.1: Flutter's layers [1]

content of the application.

3.2 Reactive user interfaces

On the surface, Flutter is a reactive, pseudo-declarative (by declarative by mean is the manner in which the developer describe the current User Interface state and leaves the transitioning to the framework [28]) User Interface (UI) framework, in which the developer provides a mapping from application state to interface state, and the framework takes on the task of updating the interface at run-time when the application state changes. Flutter decouples the UI from its underlying state which means that the user only creates the UI description and the framework takes care of using that configuration to both create and update the user interface appropriately. A widget declares its UI by overriding the build() method, which is a function that converts state to UI.

3.3 Widgets

Flutter widgets are included as a unit of composition. Widgets are the building blocks of a Flutter application's UI. Each widget is an immutable declaration of part of the UI. Widgets form a hierarchy based on composition. Each widget nests inside its parent and can receive context from the parent. This structure carries all the way up to the root widget.

3.3.1 Composition

Widgets are typically composed of many other small, single-purpose widgets that combine to produce powerful effects. There is a class hierarchy which is deliberately shallow and broad to maximize the possible number of combinations, focusing on small, composable widgets that each do one thing well. Core features are abstract, with even basic features like padding and alignment being implemented as separate components rather than being built into the core of the layout component.

3.3.2 Widget state

In Flutter there are two major classes of widgets: **stateful** and **stateless** widgets. If widgets do not change their state and they do not have any properties that change over they are called **StatelessWidgets** otherwise the unique characteristics of a widget need to change based on user interaction or other factors, that widget is **stateful**. When an element on the UI changes (e.g. a textfield that changes its context when a button is pressed), the widget needs to be rebuilt to update its part of the UI. These widgets subclass **StatefulWidgets**, and they store mutable state (the state that can change after the initialization) in a separate class that subclasses state. **StatefulWidgets** do not have a build method, instead their UI is built through their state object (an object e.g. mounted that determines if a widget is shown on screen or not). Whenever the programmer change a state object the **SetState()** method must be called to signal the framework to update the UI by calling the state's build method again.

3.3.3 State management

The state is managed and passed around with a constructor in a widget to initialize its data. The method `build()` can ensure that any child widget is instantiated with the data it needs. As widget trees get deeper, however, passing state information up and down the tree hierarchy becomes cumbersome. So, a third widget type, **InheritedWidget**, provides an easy way to grab data from a shared ancestor. **InheritedWidget** can be used to create a state widget that wraps common ancestor in the widget tree. **InheritedWidgets** also offer an `updateShouldNotify()` method, which Flutter calls to determine whether a state change should trigger a rebuild of child widgets that use it. As applications grow, more advanced state management approaches that reduce the ceremony of creating and using stateful widgets become more attractive. Many Flutter apps use utility packages like **Provider**, which is a package that makes it easy to share information through the widget tree and

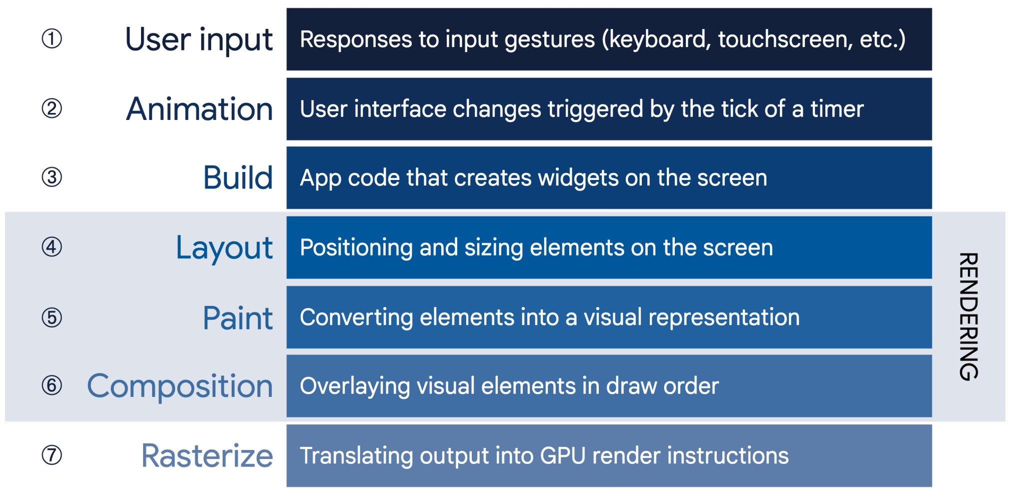

① User input
Responses to input gestures (keyboard, touchscreen, etc.)
② Animation
User interface changes triggered by the tick of a timer
③ Build
App code that creates widgets on the screen
④ Layout
Positioning and sizing elements on the screen
⑤ Paint
Converting elements into a visual representation
⑥ Composition
Overlaying visual elements in draw order
⑦ Rasterize
Translating output into GPU render instructions
RENDERING

Figure 3.2: Render pipeline [1]

provides a wrapper around **InheritedWidget**.

3.4 Rendering and layout

Cross-platform frameworks typically work by creating an abstraction layer over the underlying native Android and IOS UI libraries, attempting to smooth out the inconsistencies of each platform representation. Application code is often written in an interpreted language like Javascript, which must in turn interact with the Java-based Android or Objective-C-based IOS system libraries to display UI. All this adds overhead that can be significant, particularly when there is a lot of interaction between the UI and the application logic. Figure 3.2 shows the pipeline of dataflow of the system.

3.5 Platform embedding

Flutter UI built, laid out, composited, and painted by Flutter itself. The mechanism for obtaining the texture and participating in the application lifecycle of the underlying operating system inevitably varies depending on the unique concerns of that platform. The engine is platform-agnostic, presenting a stable Application Binary Interface (ABI) that provides a platform embedder with a way to set up and use Flutter. The platform embedder is the native OS application that hosts all Flutter content, and acts as the glue between the host operating system and Flutter. When a Flutter application starts, the embedder provides the entrypoint, initializes the Flutter engine, obtains threads for the UI and rastering, and creates a texture that Flutter can write to. The embedder is also responsible for the application lifecycle, including input gestures (such as mouse, keyboard, touch), window sizing, thread management, and platform messages. Flutter currently includes platform embedders

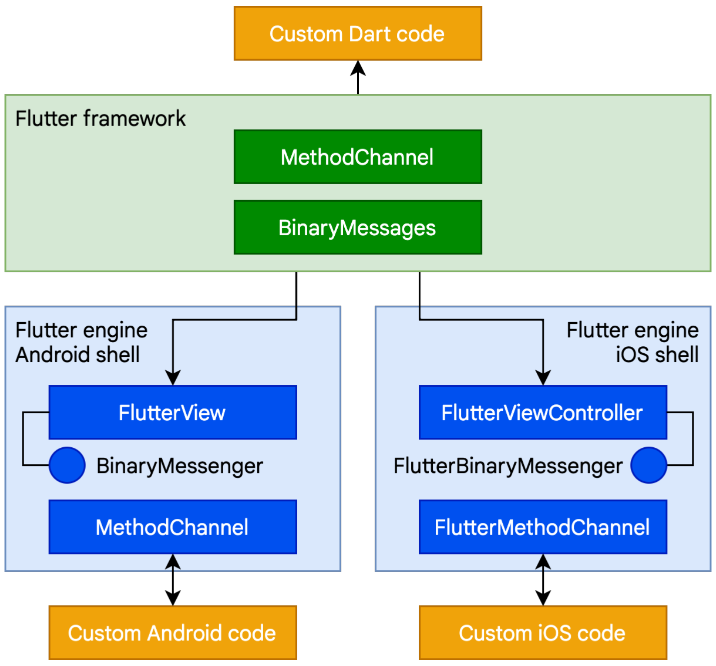


Figure 3.3: Platform channels [1]

for Android, IOS, Windows 10, MacOS, and Linux.

3.6 Integrating with other code

Flutter provides a variety of interoperability mechanisms, when the programmer accessing code or APIs written in a language like Kotlin or Swift, calling a native C-based API, embedding native controls in a Flutter application, or embedding Flutter in an existing application. For mobile and desktop apps, Flutter allows the programmer to call into custom code through a platform channel, which is a mechanism for communicating between the Dart code and the platform-specific code of the host application. By creating a common channel (encapsulating a name and a codec), messages can be send and received between Dart and a platform component written in a language like Kotlin or Swift. Data is serialized from a Dart type like Map into a standard format, and then deserialized into an equivalent representation in Kotlin (such as HashMap) or Swift (such as Dictionary). Figure 3.3 shows the exact procedure of integration.

3.7 Conclusions

As we chose Flutter SDK for our developing framework we realised that it is a well built ecosystem of tools to make it easy for every programmer to develop applications. Flutter team used best practices to create a robust architecture which consist of three layers closely related to each other in order to work well. The first layer contains the Dart language which is the developing language of Flutter, the second layer is the Flutter's engine which is responsible for the most of the functionalities required for every application and lastly is the Embedder layer which is responsible for every platform specific functionality of every supported device - OS. With Flutter's reactive user interfaces and widget tools it is easier than ever to create an application to many platforms. Using the platform embedding and integrating with other code from every supported platform with the help of platform channels the programmer can built one main application and with small changes can execute it on multiple platforms.

Chapter 4

Architecture of the Sodasense Framework and tools used for developing and testing

The architecture of the application was something that changed a lot of times until we ended up choosing the best practises to store our data. First in order to begin developing the application we had to choose the best IDE to work with, examining all the available options to debug, build and test the application. While finding the best IDE was not hard, testing the application on IOS required some troubleshooting in order to install the proper software.

4.1 Structure of the application

One of the biggest struggles of this thesis was to find the way to store our data locally and then finding a way to upload them on a university server using an authorization system in order to prevent unauthorized access. After we found the most appropriate databases for our project to work with which were SQLite for local database and MongoDB for the server, the schema was simple. The user has a mobile device which has all the needed sensors (such as GPS, accelerometer, magnetometer, gyroscope, pedometer, proximity and barometer) and also has a stable connection to the internet. After the user opens the application and connects, the application sends the username and password to a web-service that is responsible for user login and then the web-service sends a request to Keycloak, which is an open source authorization software to authenticate that the credentials are correct and let the user proceed to main page [29]. As the user moves the device, the sensors of the device are collecting data and store them locally to an SQLite (as it has a small footprint under 1MB of space [30, 31]) database and if the user is connected to

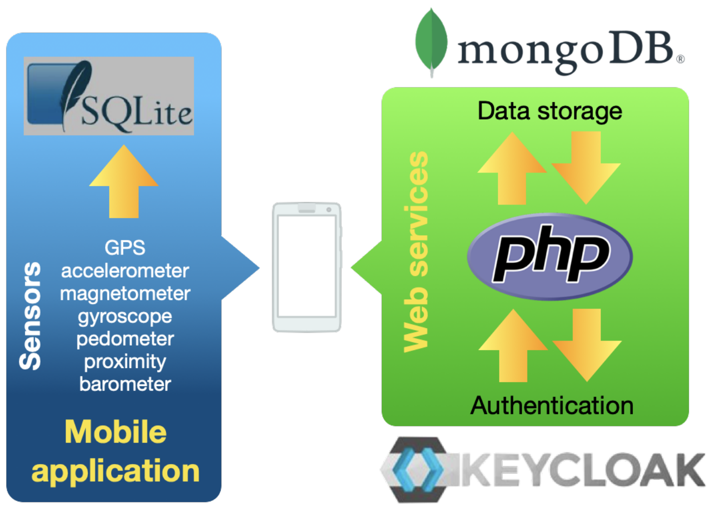


Figure 4.1: Structure of application

the internet the application uses another web-service (there are 3 web-services that sends data to the server, more information on Chapter 6), which is responsible for making sure the data are sent with a proper format, the data contains the unique id and then sends the current data from each sensor to each collection on our database. Then, after each checking is completed the web-service sends the data on our server. On our server we chose to have the NoSQL MongoDB database. It is an open source NoSQL data management system used for big data storage. The primary data entity in MongoDB is called *document* and is formed by a single JSON file. Documents are the logical counterpart of records in relational databases. Similar documents are organized in *collections* (which logically correspond to relational tables). MongoDB is easy to use and very scalable [32]. Figure 4.1 shows the structure of application.

4.2 Choosing the best IDE

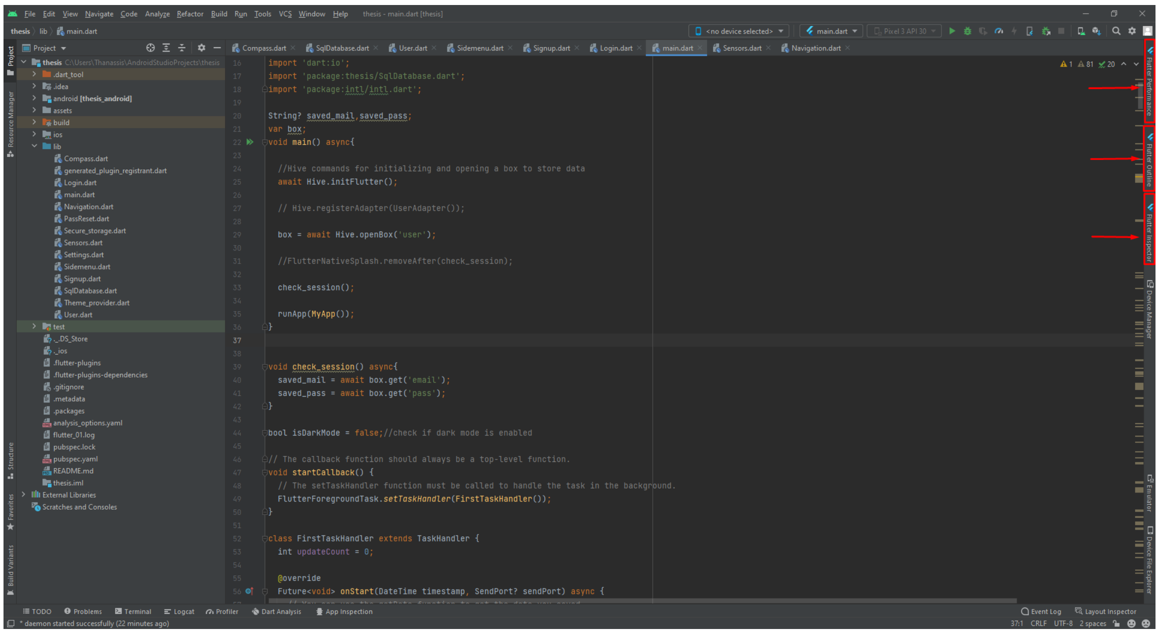
When it comes to developing applications for Flutter the are two choices for Integrated Development Environments (IDE). The first one is the Visual Studio Code and second is the Android Studio. The Visual Studio Code is a more lightweight IDE and comes with all the things a programmer would expect (like debugging, breakpoints), but it does not contain an Android emulator. The Android Studio on the other hand comes with all the things the Visual Studio Code has and also include an Android emulator built-in with a tool to select the version of Android device the programmer wants to debug the created application. We had experience with Android Studio, so we selected it as Flutter application development tool. To integrate Flutter, we followed the steps that we described on the Chapter 2. For Flutter integration on Android Studio we installed two plugins through the Android Studio plugin store, the first one was Dart to be able to write, compile and debug

Dart code and the second one was Flutter plugin to connect the installed framework on the computer with the Android Studio environment. Android Studio combined with Flutter supports 3 types of builds for deployment on a device [33]:

1. debug
2. profile
3. release

In more depth:

1. The debug mode, the application is set up for debugging on a physical device, emulator, or simulator. Debug mode for mobile applications means that:
    - Assertions are enabled (assertion is a statement that disrupts normal execution if a boolean condition is false to help with debugging).
    - Service extensions are enabled (service extensions are a set of tools that provide additional debugging capabilities e.g.debugAllowBanner, debug-DumpApp, debugDumpRenderTree,debugPaint).
    - Compilation is optimized for fast development and run cycles.
    - Debugging is enabled, and tools supporting source level debugging can connect to the process.
2. The profile mode, some debugging ability is maintained—enough to profile your application's performance. Profile mode is disabled on the emulator and simulator, because their behavior is not representative of real performance. On a mobile device, profile mode is similar to release mode, with the following differences:
    - Some service extensions, such that enabling the performance overlay, are enabled. Tracing is enabled, and tools supporting source-level debugging such as DevTools can connect to the process.
    - Tracing is enabled, and tools supporting source-level debugging such as DevTools can connect to the process.
3. The release mode, is used when the programmer wants the maximum optimization and minimal footprint size. This mode is not supported on the simulator or emulator. For mobile device, release mode means that:
    - Assertions are disabled.
    - Debugging information is stripped out.



```dart
import 'dart:io';
import 'package:thesis/SqlDatabase.dart';
import 'package:intl/intl.dart';

String? saved_mail,saved_pass;
var box;
void main() async{

  //Hive commands for initializing and opening a box to store data
  await Hive.initFlutter();

  // Hive.registerAdapter(UserAdapter());

  box = await Hive.openBox('user');

  //FlutterNativeSplash.removeAfter(check_session);

  check_session();

  runApp(MyApp());
}


void check_session() async{
  saved_mail = await box.get('email');
  saved_pass = await box.get('pass');
}

bool isDarkMode = false;//check if dark mode is enabled

// The callback function should always be a top-level function.
void startCallback() {
  // The setTaskHandler function must be called to handle the task in the background.
  FlutterForegroundTask.setTaskHandler(FirstTaskHandler());
}

class FirstTaskHandler extends TaskHandler {
  int updateCount = 0;

  @override
  Future<void> onStart(DateTime timestamp, SendPort? sendPort) async {
```

Figure 4.2: Flutter interface

- Debugging is disabled.
- Compilation is optimized for fast startup, fast execution, and small package sizes.
- Service extensions are disabled.

Figure 4.2 shows the UI of the Android Studio and the available tools provided by the Flutter framework such as performance, outline and inspector tools.

4.3 Testing on devices

Since, we use Windows 10 as our main OS and MacOS as our secondary OS we installed Flutter on each OS in order to test it on Android and IOS. Even though we can build the application for Android and IOS on MacOS (due to the nature of the Apple software the testing on IOS can only be done on MacOS) we found a **MacMini** on later development stage so we started testing the application on the early stages only on Android devices on Windows 10. The first 5 steps for Flutter installation are the same on both Windows 10 and MacOS and we perform the following steps:

1. Download the latest version [34] and we create a folder to copy the contents of the .zip file.
2. Update the path on Windows 10 and MacOS by creating a link to the Flutter folder we created before in order to be accessible from the terminal.
3. Enter the command

```
flutter doctor
```

in the terminal of each OS to check if the Flutter installation is completed.

4. Install Android Studio [21]. This will offer useful command line tools, the android emulator and the Gradle which is an advanced build toolkit, to automate and manage the build process of building an Android application which is crucial part of the Android OS.

5. Install from Android Studio market the Flutter and Dart plugins in order to connect the folder we created before containing the contents of the SDK with the Android Studio.

We had to make some adjustments on the application for the iPhone, so we installed Xcode IDE from the App store of MacOS. Then in order to install the application on a IOS device we had to install Cocoapods. We perform the following steps only on MacOS

1. On terminal we entered the following command

```
curl -L https://get.rvm.io | bash -s stable
```

2. Reopen terminal to enter the following command

```
rvm install ruby-2.6
rvm --default use 2.6.6
gem install cocoapods
```

Then we were ready to install the application on both Android and IOS device.

4.4 Conclusions

The architecture of the Framework was something that changed many times as we were searching the best way to combine all the three main components: The best practices to built the UI of the application in order to be user friendly and easy to use, the most suitable database system to set on our university server and finally the best way to build the web-services, for the application to communicate, authorize/authenticate and exchange data with the database. After solving the mentioned issues, the procedure to find the appropriate tools and the testing on both Android and IOS devices was straight forward easy to be done.

Chapter 5

Application tour

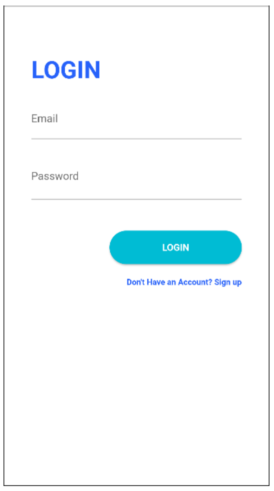
Sodasense Framework was built in mind with the principle of been simple to use, straightforward and fast for the average user. We selected to create a UI that will help the user understand every capability of the application in order to be used to its fullest. The privacy of the user's data was also a critical part for us, so we chose in order to use the application the user should create an account first.

5.1 Login screen

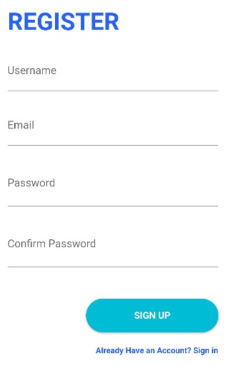
Login screen is the first screen the user sees after he/she opens the application. Figure 5.1 shows on the left Login screen of the application. From there it has two options, if user has already registered to the framework, he/she enters the e-mail and the password and then presses the button Login in order to connect to the application, else he/she must presses the text beneath the Login button Don't Have an Account? Sign up to proceed to the registration screen. The e-mail text-field has built-in email validator to inform the user that the inputed e-mail is not valid. The user must be connected to the internet to proceed to the main screen. If the user has not logged out since the last time the application was closed on their device, the user will be redirected to the main screen without entering the credentials again.

5.2 Registration screen

Registration screen is where a new user can register to framework by providing username, email and password. Just like the **Login** screen the email text-field has an email validator to warn in case the user enters a non valid email. In the framework the e-mail must be unique, if the user enters an e-mail that already exist when the button is pressed a small text poping from the bottom will tell the user to enter another e-mail. The username is only for displaying it inside the application and



(a) Login



(b) Registration

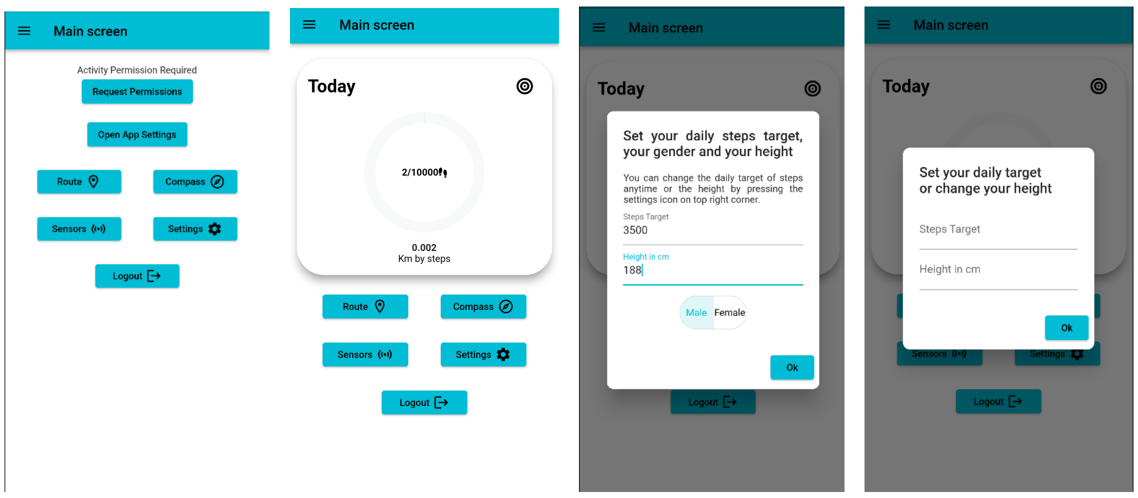
Figure 5.1: Starting the application

keep a copy in the database. The user must type a password and type the same password again on the confirmation text-field. If the two password text-fields do not match, the user will be warned with a red text under the second text-field. The password must be greater than 10 letters and less than 16 letters while containing at least a special character from these: !, @, #, $, %, ˆ, &, *. The user just like on the Login screen, must be connected to the internet in order to proceed to the main screen. Figure 5.1 shows on the right the Register screen of the application.

5.3 Main screen

Figure 5.2 (a) shows the main screen without activity permission (is the permission of which the user agrees to allow the application collect movement data). The main screen is the homepage of the application. On this screen the user gets redirected after the registration or the login. On the main screen the user can see the daily progress of the steps and five buttons to go to the other screens or to logout from the application.

If we want to collect daily steps we must give the permission of activity to the application (if the user has an android device with android version 8 or 9 it's not required to give permission). Figure 5.2 (c) shows the dialog that will appear after the user press the button 'Request Permission'. The user must enter the desired daily steps target, the height of the user (must be between 0-250cm else there will a warning which will not allow the user to press the 'ok' button) and the option to select between male or female for accurate tracking of kilometers walked by the user (the activity permission will be asked only the first time of opening the application). The selection of the gender is important because there is a formula for calculating



Set your daily steps target, your gender and your height

You can change the daily target of steps anytime or the height by pressing the settings icon on top right corner.

Steps Target: 3500

Height in cm: 188

[✓ Male] Female

Set your daily target or change your height

Steps Target

Height in cm

(a) (b) (c) (d)

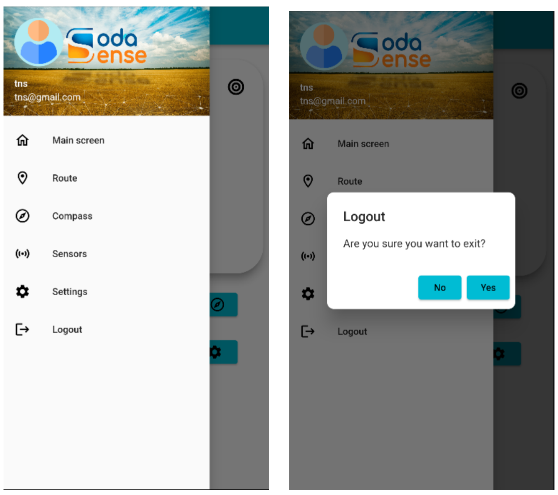
Figure 5.2: (a) Main screen without activity permission, (b) Main screen with activity permissions, (c) Activity permission dialog and (d) Target button dialog

the average length of step per gender [35]. After the user presses the button ‘ok’ the user can see the progress on a circular progress bar. Figure 5.2 (b) shows the card which is a widget (card is the widget name on Flutter which has all the text and the circular progress bar of the main screen) which contains all the information of the user’s daily steps.

The daily steps are saved on SQL database table (locally on device) for later synchronization with a central database as shown on image of application structure on Chapter 4. On the top right corner we can see a target icon which is a button. From there the user can change the daily steps target and the height of the user(in the case a child uses the application). In the main screen also the application collects data from the GPS and the rest of the sensors. Every time the user changes location a copy of the timestamp, latitude and longitude is saved on SQL database table (locally on device) for later synchronization with a central database. Also Every 10 seconds a copy of each sensor data is taken and with a timestamp is saved on SQL database table (locally on device) for later synchronization with a central database. Figure 5.2 (d) shows the dialog of the target button. On the top left corner there are 3 parallel lines, if the user presses it or by sliding from the left to the right of the screen the sidemenu will appear on which the user can navigate to the rest of the screens. Figure 5.3 (a) shows the sidemenu of the application.

5.4 Sidemenu

Sidemenu is where the user can navigate through the multiple screens of the application (Main screen, Route, Compass, Sensors, Settings). The Main screen is



Logout

Are you sure you want to exit?

(a) (b)

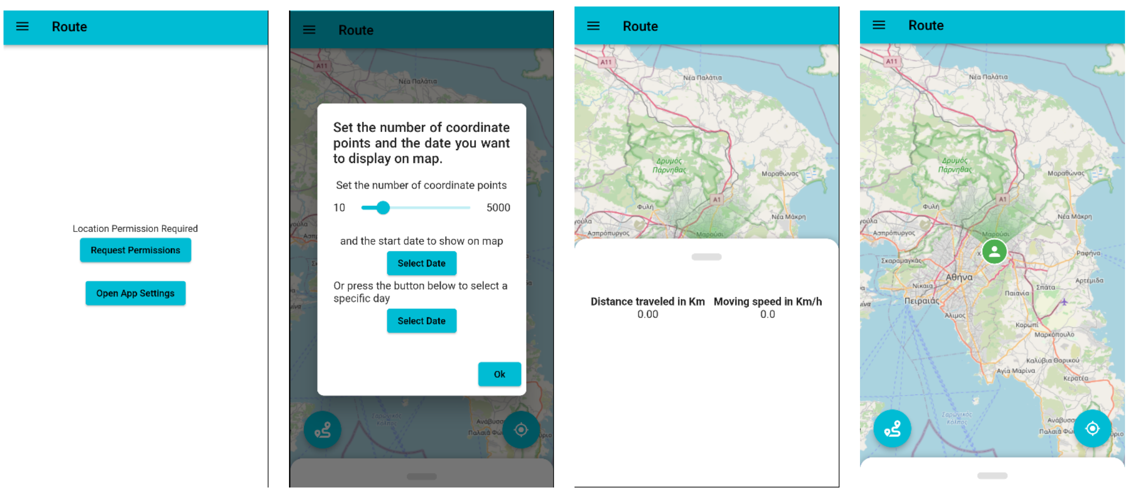
Figure 5.3: (a) Sidemenu (b) Logout button of the Sidemenu

the main screen of the application, the Route screen is where the user can see a map with the current position of the user and the path the user has created with a user icon with a red line, Compass screen is a screen with compass (which uses the magnetometer of the device), location, address and altitude. Sensors is the screen in which the user can see all the data from the device sensors. Settings is the screen in which the user can change between light and dark theme, change the rate of the sensors of collecting data and the Logout is a button that appears a dialog box in which the application asks if the user wants to logout from the application. If the user presses the button ‘Yes’, the user will be redirected to the Login screen and he/she will have to enter the credentials in order to login again or the next time the user opens the application. Figure 5.3 (b) shows dialog box of the Logout button.

5.5 Route screen

Route screen is the screen where the map is. The screen illustrates the current position of the user on the map. In order for the tracking to work properly the user must have enabled the GPS on the device before opening this screen, else the map will be loaded but the coordinates will not update each time the user changes location. After the user selects this screen for the first time the application will require real time tracking permission in order for the map to be loaded, so two buttons are on the screen, the first one to select without the type of permission the user wants to give to the application and the second one is to open application settings so that the user will give manually the real time tracking permission to the application. Figure 5.4 (a) shows how the screen looks without tracking permission.

After the user gives the required permission to the application, the map will



Location Permission Required

Set the number of coordinate points and the date you want to display on map.

Set the number of coordinate points

10 5000

and the start date to show on map

Or press the button below to select a specific day

Distance traveled in Km 0.00

Moving speed in Km/h 0.0

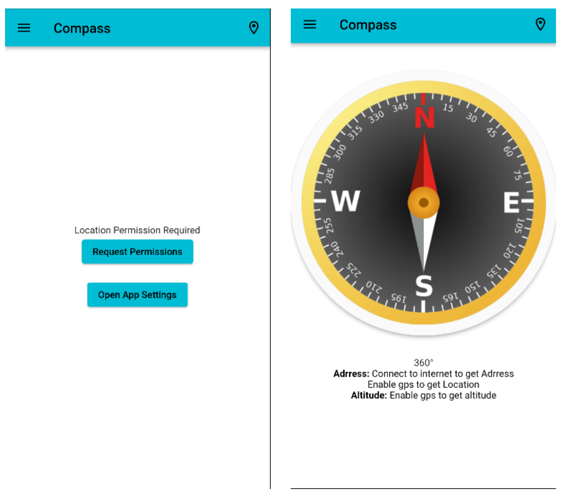
(a) (b) (c) (d)

Figure 5.4: (a) Route screen sliding panel and (b) Route screen with loaded map on the right.Route screen without location permission on the left and dialog of the left bottom button on the right

be loaded using the internet connection and will show the user's current position on map. On the bottom right corner there is the button which zoom to the user current location. By pressing the button for 3 seconds a popup will appear from the bottom to warn the user that the coordinates are saved to the clipboard. If the user navigate to the map losing the current location marker this button also centers the map with the center point being the user's current position marker. Figure 5.4 (b) shows the dialog of the left bottom button. On the bottom left corner there is a second button which appear a dialog to select a starting date and the number of coordinate points to be drawn on the map or to select only a specific day to draw the first 2000 coordinate points. By pressing the button 3 seconds the route will be erased from the map. Figure 5.4 (d) shows how the screen focus on the user when the user presses the button on the bottom right. Lastly, there is a sliding panel that shows the total distance in kilometers if a user has selected a route and the user's current moving speed. Figure 5.4 (c) shows the sliding up panel.

5.6 Compass screen

In Compass screen the user can see a compass which uses the magnetometer of the device, some geographical information about the user's location. In order to see the geographical information like latitude, longitude, address and altitude, requires from the user to give real time tracking permission. Figure 5.5 (a) shows how the screen look like before getting the permission by the user. So, in order for the user to have the full potential of the Compass screen, the user must give the permission to



Location Permission Required

360°
Adrress: Connect to internet to get Adrress
Enable gps to get Location
Altitude: Enable gps to get altitude

(a) (b)

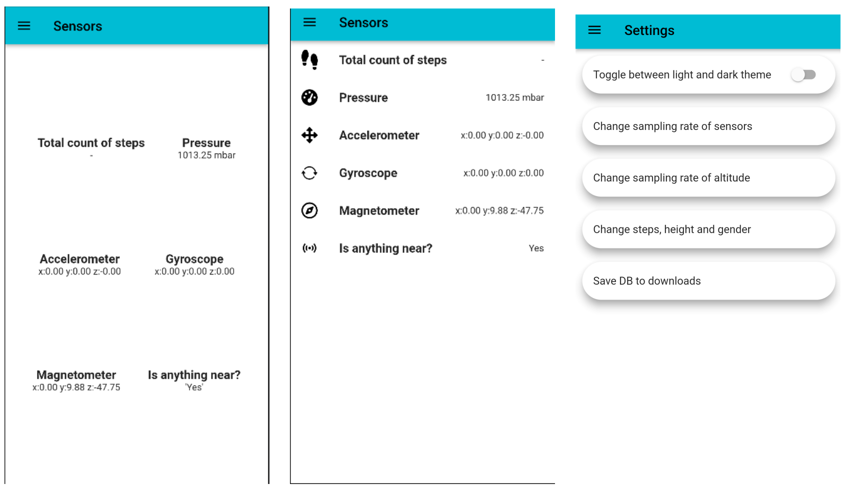
Figure 5.5: (a) Compass screen without location permission (b) Compass screen with location permission.

allow location tracking. If the user does not allow the location tracking, the compass screen will not work properly and also the altitude will not be shown on the screen and not registered on the local database.

In order for the application to show the coordinates (latitude, longitude) the user must enable the GPS and press the button on the top right corner (pin icon) to get the current coordinates. If the user also wants the current address and the current altitude the user must connect to the internet by opening the Wi-fi or the cellular data and then pressing again the pin button. Every some seconds (10 seconds is the default) a copy of the timestamp and altitude is saved on SQL database table (locally on device) for later synchronization with a central database (in order for the altitude to be saved an internet connection is required). The compass in order to work does not require enabled GPS or internet connection. The Figure 5.5 (b) shows the image after the user gives the real time tracking permission (the GPS and the internet are disabled).

5.7 Sensors screen

This is the screen where the user can see some sensors of the device such as pedometer, barometer, accelerometer, gyroscope, magnetometer and proximity. If the device does not have a sensor then next to its sensor there will be a message of the current sensor saying that the sensor is not available. The pressure sensor shows the pressure on milli bars. The proximity sensor show if something is close enough to the sensor, under 1 cm the sensor shows yes else is shows no. If the mobile device does not have pedometer sensor the total count of steps will be '-'. Due to design


Sensors

Total count of steps
-
Pressure
1013.25 mbar
Accelerometer
x:0.00 y:0.00 z:-0.00
Gyroscope
x:0.00 y:0.00 z:0.00
Magnetometer
x:0.00 y:9.88 z:-47.75
Is anything near?
'Yes'

Sensors

Total count of steps	-
Pressure	1013.25 mbar
Accelerometer	x:0.00 y:0.00 z:-0.00
Gyroscope	x:0.00 y:0.00 z:0.00
Magnetometer	x:0.00 y:9.88 z:-47.75
Is anything near?	Yes

Settings

Toggle between light and dark theme
Change sampling rate of sensors
Change sampling rate of altitude
Change steps, height and gender
Save DB to downloads

(a) (b) (c)

Figure 5.6: On the left the old Sensors screen, in the middle new Sensors screen and on the right the Settings screen.

of being not very attractive and not very user friendly we decided to completely change how the sensors screen look like. The Figure 5.6 (a) and (b) shows the old and new screen of the sensors.

5.8 Settings screen

Settings screen is the last screen of the application. The user here can change between light and dark theme to change while using the application, even though the application takes by default the theme that the device currently use, the user can change it anytime. Also the user can change the sampling rate of the altitude and the sensors such as accelerometer, barometer, pedometer, gyroscope, magnetometer and proximity. Finally the user can change the daily target steps, the height and the gender. The option 'Save DB to downloads' is for copying local database for debugging purposes. Figure 5.6 (c) shows how the settings screen look like.

5.9 Conclusions

As we take a look at the capabilities of each screen of the application we can see that each screen has a unique UI and has a different functionality. If the user allow all of the permissions to collect data from GPS and the various sensors the device has, the user can receive the maximum potential of the application. The route will be registered and later the user can see by date the registered route. Because of the nature of this framework, all of the data belong to each user and can extract the database to the device for later research on the data.

Chapter 6

Development of the Sodasense framework

The development of the application was the biggest part and the most difficult compared to the database on the server and the web-services. By starting this project from scratch we had the ability to create the application exactly as we wanted it and setting our goals on every version farther than previous time. The UI is also a part where we followed best practices to ensure that the user will never be lost on a screen of the application by giving the option to move a different screen by selecting it on the side menu. Every functionality is written within a function so it can be used on all files and is well documented for someone who wants to read the code. Each file of the project contains the code for each screen with the exception of the SqlDatabase.dart file. Every time we completed the goals we had set each time, a new version was born and was immediately send to friends for testing. Even though the testing on some devices may be successful, there is always the concern of the developer to exist some unidentifiable bugs that are not reported by the users. The complete project of this thesis can be found on Github [36].

6.1 User login

The Login screen (Login.dart) is extended with a stateful widget which means that this screen has states, because it has User Interface (UI) elements that changes if the user interacts with them. Also it has only 4 UI elements, two text-fields, one button and one gesture detector. To get the text from each text-field we used a TextEditingController for e-mail text-field and another one for the password text-field. The TextEditingController also requires a listener to work properly, so it must be initialized once every time the screen is opened for the first time the app is launched. To do this we used the function initState which is a core component of a stateful class. Then, when the app terminates the listeners must be disposed. Below we can see the initialization of the Controller.

```
@override
void initState(){
  super.initState();

  //Start listening to changes with listeners
  mail_txtController.addListener(mailvalue);
  pass_txtController.addListener(passvalue);
}

@override
void dispose(){
  //Clean controllers when the widget is
  //removed from the widget tree
  //and removes the values of both listeners
  mail_txtController.dispose();
  pass_txtController.dispose();
  super.dispose();
}
```

The first text-field in which the user can enter the e-mail has built-in an e-mail validator. We used the package 'email_validator' [37] which checks if the value in the current text-field is an e-mail. Basically it checks if the text has the necessary form of an e-mail (for example text@text.text). We built a function to check whether the e-mail text-field is empty or if the e-mail is valid. Below we can see the code for checking the if the e-mail is valid.

```
//Function for displaying the correct error message on email
   text-field
 String? Mail_Text-field_check(){
  String mail_msg='';
  if(mail_txtController.text.isEmpty==true){
    mail_msg='Email can\'t be empty';
    print(mail_msg);
    mail_check=false;
    return mail_msg;
 }
 else if(EmailValidator.validate(mail_txtController.text)==
   false){
  mail_msg='Enter a valid Email';
  print(mail_msg);
  mail_check=false;
  return mail_msg;
 }
 else if(EmailValidator.validate(mail_txtController.text)==
   true){
  mail_check=true;
  mail_msg='Valid email';
  print(mail_msg);
 }
}
```

When a user enters something and then deletes it, there is an error text (a small red text below the text-field) indicating that the text-field cannot be empty but the text will not appear if the text-field does not change for the first time. Figure 6.2 (a) shows the empty e-mail warning message.

In order to know if the text-field changes state we used the built-in function of Text-field widget called onChanged which monitors every change it happens on the current text-field. Then we have a boolean variable which calls a function returning true or false if the text-field changed for the first time. If it changed it calls the function to check the context of the text, if the text is an e-mail, if the text is empty or if the text is a valid e-mail (in which case the error text disappears). The password text-field works with the same logic. We used the same built-in function `onChanged()` of the Text-field widget to know if there is a text entered in the text-field for the first time by setting another boolean variable which calls a function returning true or false. Below we can see the function which is called to check if there is a text entered in the text-field or not.

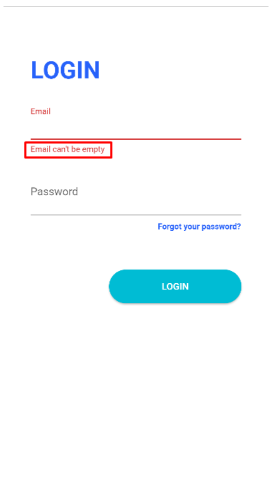
```
//Function for displaying the correct error message on
   password text-field
String? Pass_Text-field_check(){
 String pass_msg='';
  if(pass_txtController.text.isEmpty==true){
   pass_msg='Password can\'t be empty';
   print(pass_msg);
   print(pass_check);
   pass_check=false;
   return pass_msg;
  }
  else if(pass_txtController.text.isEmpty==false){
   pass_check=true;
```

```
    pass_msg='Valid password';
    print(pass_msg);
  }
}
```

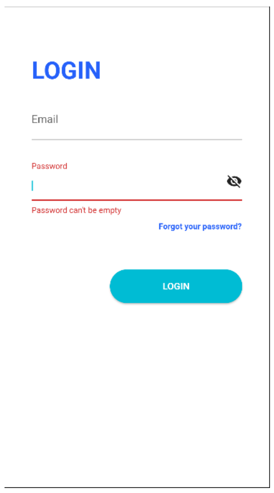
If the user types something in the password text-field and then deletes it an error text below the text-field will appear warning the user that the password text-field cannot be empty. Figure 6.2 (b) shows the warning message under password text-field.

The password text-field has also the option to hide and show the password as the user is typing it by clicking the eye button at the end of the text-field. This is feasible by using a built-in function of the text-field called `obscuredText()`. We used this function with a boolean variable that when the user presses the button the variable changes from true to false and vice versa. Figure 6.2 (c) and (d) shows how the button works.

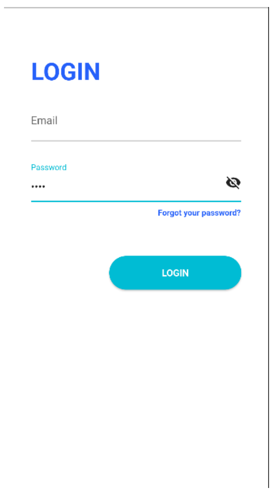
If the user does not have an account he/she must create one from the registration page by pressing the text 'Don't Have an Account? Sign up' which is a `GestureDetector` widget. It has a function called `onTap()` that triggers when the user touches the text. When the `onTap()` function is triggered we forward the user to the registration page to sign up. Finally the user must press the button Login in order to proceed. When the button is pressed the user must have already enabled the device Wi-fi or cellular data and has a stable connection to the internet or else



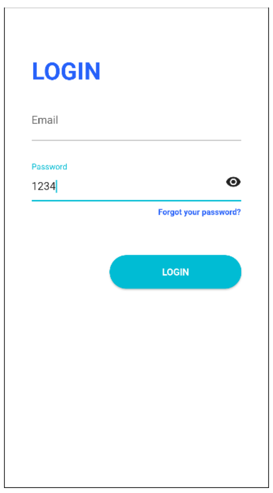
(a)



(b)



(c)

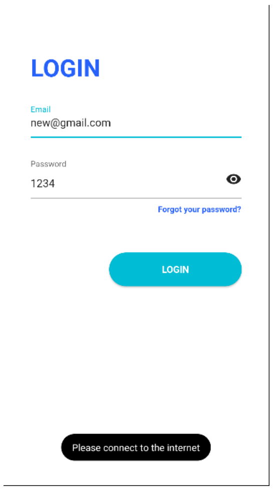


(d)

Figure 6.1

Figure 6.2: (a) Email text-field error text (b) password text-fields error text (c) Password Text-field show text button (d) Password hide text button

the user will not be able to login and there will be pop up text warning the user that the device is not connected to the internet. To check if the user has the device Wi-fi or cellular data enabled we used the package 'connectivity_plus' [38] and for checking if the user is connected to the internet the package 'internet_connection_checker' [39]. The pop up text is called `Toast` and we used the package 'fluttertoast' [40] in order to display a message with an animation to the user. Figure 6.3 shows the toast message that is displayed when the user is connected to the internet.


LOGIN

Email
new@gmail.com

Password
1234

Forgot your password?

LOGIN

Please connect to the internet

Figure 6.3: Toast pop up message

Also the two text-fields must be valid in order to login. If the above conditions are met then the user will be redirected to the main page of the application.

6.2 User registration

Registration page (Signup.dart) is extended with a stateful widget. It consists by 6 UI elements, four text-fields, one button and one gesture detector. To get the text from each text-field we used the same logic just like on the login screen, one `TextEditingController()` with its listener for each text-field. Inside the `initState()` function we have the initializations of the listeners and the dispose of them inside the dispose function as shown on below.

```
@override
void initState(){
 super.initState();
 //Start listening to changes with listeners
 user_txtController.addListener(uservalue);
 mail_txtController.addListener(mailvalue);
 pass_txtController.addListener(passvalue);
 confpass_txtController.addListener(confvalue);
}

@override
void dispose(){
 //Clean controllers when the widget is removed from the
 //widget tree and removes the values of listeners
 user_txtController.dispose();
 mail_txtController.dispose();
 pass_txtController.dispose();
 confpass_txtController.dispose();
 super.dispose();
}
```

The first text-field is a simple text-field for only saving and showing the username of the user. It shows an error message below the text-field in case that the user writes a username and then deletes it to warn the user that the text-field cannot be empty. We used the function `onChanged()` from the Text-field widget to know when the user types on the text-field and then we used the function below to warn the user if it is necessary. Below we can see the function for validating the user e-mail.

```
//Function for displaying the correct error message on
   username text-field
String? User_Textfield_check(){
 String user_msg='';
  if(user_txtController.text.isEmpty==true){
   user_msg='Username can\'t be empty';
   print(user_msg);
   user_check=false;
   return user_msg;
  }
  else if(user_txtController.text.isEmpty==false){
   user_check=true;
   user_msg='Valid username';
   print(user_msg);
  }
}
```

The e-mail text-field is built the exact same way as in the login screen using the function `onChanged()` to use a function for checking if the text-field is empty or if the e-mail is valid. The password text-field uses the `onChanged()` function of text-field widget with a function to check if the criteria for a strong password are met which are: the length of the password must be between 10 and 16 letters and use at least a special character like !, @, #, $, %, ˆ, &, . Below we can see the function that checks if the text-field is empty.

```
//Function for displaying the correct error message on
   password text-field
String? Pass_Text-field_check(){
 String pass_msg='';
  if(pass_txtController.text.isEmpty==true){
   pass_msg='Password can\'t be empty';
   print(pass_msg);
   pass_check=false;
   return pass_msg;
  }
 else if(pass_txtController.text.isEmpty==false){
  if(pass_txtController.text.length < 10 ||
   pass_txtController.text.contains(new RegExp(r'(?=.*[!@#$
   %^&*])')) == false){
   pass_msg='Password must be at least 10 letters\nand
   contain special characters (!@#\$%^&*)';
   pass_check=false;
   return pass_msg;
  }
  else if(pass_txtController.text.length > 16 ||
   pass_txtController.text.contains(new RegExp(r'(?=.*[!@#$
   %^&*])')) == false){
   pass_msg='Password must be maximum 16 letters\nand contain
    special characters (!@#\$%^&*)';
   pass_check=false;
   return pass_msg;
  }
  else{
   pass_check=true;
   pass_msg='Valid password';
   print(pass_msg);
  }
 }
}
```

The functionality for the hide and show button at the end of the text-field is the same as in the Login screen. The confirm text-field also uses the function onChanged() to call a function to check if the text that is written on the password text-field is the same as the one written on the confirm password text-field. The function also checks if the text-field is empty and warns the user. Below we can see the function for the confirmation text-field.

```
//Function for displaying the correct error message on
   confirmation password text-field
String? Conf_Text-field_check(){
 String conf_msg='';
  if(confpass_txtController.text.isEmpty==true){
   conf_msg='Password can\'t be empty';
   print(conf_msg);
   conf_check=false;
   return conf_msg;
  }
  else if(confpass_txtController.text.isEmpty==false){
   if(confpass_txtController.text.compareTo(
   pass_txtController.text) != 0){
    conf_check=false;
    conf_msg='Password isn\'t same as the one above';
    return conf_msg;
   }
  else{
   conf_check=true;
   conf_msg='Valid username';
   print(conf_msg);
  }
 }
}
```

The user finally must press the button ‘Sign up’ in order to create an account. The user must have enabled the Wi-fi or the cellular data of the device and have a stable connection to the internet before pressing the button just like the login button on the Login screen. Of course the four text-fields must have a valid context to proceed to get redirected to the Main screen or else a Toast message will appear to warn the user that some credentials may be missing or to connect to the internet. The text below the button named ‘Already Have an Account? Sign in’ is GestureDetector widget with a onTap() function that when it is triggered by touch the user will be redirected to the Login screen.

6.3 Foreground functionality

When the user closes the application, the application by default will stop all the functionality it has and will shut down. Before searching for the best package that does exactly what we wanted we had to search what does it take to keep an application running if the user changes to another application or if the user locks the device. We searched first on Android what is needed to make it work (writing native code) and we found out that the application must become a service. In the Android documents it states [41] that a service is an application component that can perform long-running operations in the background and it does not provide a user interface. If it starts, the service might continue running for some time even after the user switches to another application. Additionally, a component can bind to a service to interact with it and even perform interprocess communication (IPC). The service has three types: Foreground, Background, Bound.

- The foreground service performs some operation that is noticeable to the user and it must display a Notification. Foreground services continue running even when the user isn't interacting with the application.
- The background service performs an operation that isn't directly noticed by the user and it is not necessary to display a Notification.
- A service is bound when an application component binds to it by calling bindService(). A bound service offers a client-server interface that allows components to interact with the service, send requests, receive results, and even do so across processes with interprocess communication (IPC). A bound service runs only as long as another application component is bound to it. Multiple components can bind to the service at once, but when all of them unbind, the service is destroyed.

So we had to find a way to make the application a background service or a foreground service. After some search we found that an application like this which uses real time tracking must be a foreground service as Android and Apple documents suggest. But before we found this suggestion we installed two packages (both of them did not work as intended) that make the application a background service [42, 43]. After the mentioned suggestion we tried some packages that make the application a foreground service. We used the packages: foreground_service, flutter_foreground_plugin, flutter_foreground_service, flutter_foreground_service_plugin [44, 45, 46, 47]. None of these worked as we would like to. As a last resort we thought of writing native code using platform channels for both Android and IOS but that would be difficult and we had to make it ourselves because we did not found anything similar. We found a last package called 'flutter_foreground_task' [48]. Fortunately it worked as we would like to and we integrate it in the application. The whole code of the foreground functionality must be inside the main file which is the main screen and so the application will begin running in the foreground after the user enters the main screen for the first time. To achieve this functionality the home of the application (which is the first screen the user will encounter by default as Flutter provides) must be wrapped by the widget `WithForegroundTask` that the foreground package provides. Below we can see the code that is provided by the package and it must be included to initiate the functionality.

```
class MyTaskHandler extends TaskHandler {
 SendPort? _sendPort;
 int _eventCount = 0;

 @override
 Future<void> onStart(DateTime timestamp, SendPort? sendPort)
    async {
   _sendPort = sendPort;

   //You can use the getData function to get the stored data.
   final customData = await FlutterForegroundTask.getData<
   String>(key: 'customData');
   //print('customData: $customData');
 }

 @override
 Future<void> onEvent(DateTime timestamp, SendPort? sendPort)
    async {
   // FlutterForegroundTask.updateService(
   //     notificationTitle: 'MyTaskHandler',
   //     notificationText: 'eventCount: $_eventCount'
   // );

 // Send data to the main isolate.
 sendPort?.send(_eventCount);
 _eventCount++;
}

 @override
 Future<void> onDestroy(DateTime timestamp, SendPort?
   sendPort) async {
```

```
   // You can use the clearAllData function to clear all the
    stored data.
   await FlutterForegroundTask.clearAllData();
  }

  @override
  void onButtonPressed(String id) {
   // Called when the notification button on the Android
    platform is pressed.
   // print('onButtonPressed >> $id');
  }

  @override
  void onNotificationPressed() {
   // Called when the notification itself on the Android
    platform is pressed.
   //
   // "android.permission.SYSTEM_ALERT_WINDOW" permission must
     be granted for
   // this function to be called.

   // Note that the app will only route to "/resume-route"
    when it is exited so
   // it will usually be necessary to send a message through
    the send port to
   // signal it to restore state when the app is already
    started.
   FlutterForegroundTask.launchApp("/resume-route");
   _sendPort?.send('onNotificationPressed');
  }
}
```

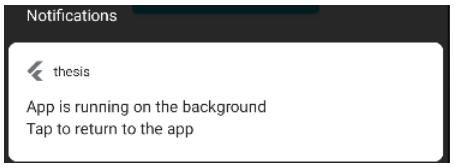
Below we can see the code that is used for configurating in each OS the Notifications options the programmer wants to have. On IOS the options are very limited compared to Android but we kept the notification simple on both operating systems.

```
Future<void> initForegroundTask() async {
 await FlutterForegroundTask.init(
  androidNotificationOptions: AndroidNotificationOptions(
   channelId: 'notification_channel_id',
   channelName: 'Foreground Notification',
   channelDescription:'This notification appears when the
   foreground service is running.',
   channelImportance: NotificationChannelImportance.LOW,
   priority: NotificationPriority.LOW,
   iconData: const NotificationIconData(
     resType: ResourceType.mipmap,
     resPrefix: ResourcePrefix.ic,
     name: 'launcher',
    ),
  ),
  iosNotificationOptions: const IOSNotificationOptions(
    showNotification: true,
    playSound: true
  ),
  foregroundTaskOptions: const ForegroundTaskOptions(
    interval: 1000,
    autoRunOnBoot: false,
    allowWifiLock: true,
   ),
   printDevLog: true,
  );
}

Future<bool> startForegroundTask() async {
 // You can save data using the saveData function.
 await FlutterForegroundTask.saveData(key: 'customData',
   value: 'hello');

 bool reqResult;
 if (await FlutterForegroundTask.isRunningService) {
  reqResult = await FlutterForegroundTask.restartService();
 } else {
   reqResult = await FlutterForegroundTask.startService(
    notificationTitle: 'App is running on the background',
    notificationText: 'Tap to return to the app',
    callback: startCallback,
   );
  }

  ReceivePort? receivePort;
  if (reqResult) {
   receivePort = await FlutterForegroundTask.receivePort;
  }
```


Notifications

thesis

App is running on the background
Tap to return to the app

Figure 6.4: Foreground notification

```
  return _registerReceivePort(receivePort);
}
```

As always the functions `initForegroundTask()` and `StartForegroundTask()` which are for notifications options and starting the notification service respectively are stated inside the `initState()` function on Main screen class. Figure 6.4 shows how the notification looks like.

6.4 Application lifecycle

Flutter provides a way of observing when the application changes state with the function didChangeAppLifecycleState. With this function the programmer can observe the states of the application:

- inactive - The application is in an inactive state and is not receiving user input (IOS only)
- paused - The application is not currently visible to the user, not responding to user input, and running in the background
- resumed - The application is visible and responding to user input
- suspending - The application will be suspended momentarily (Android only)
- detached – The application is closing

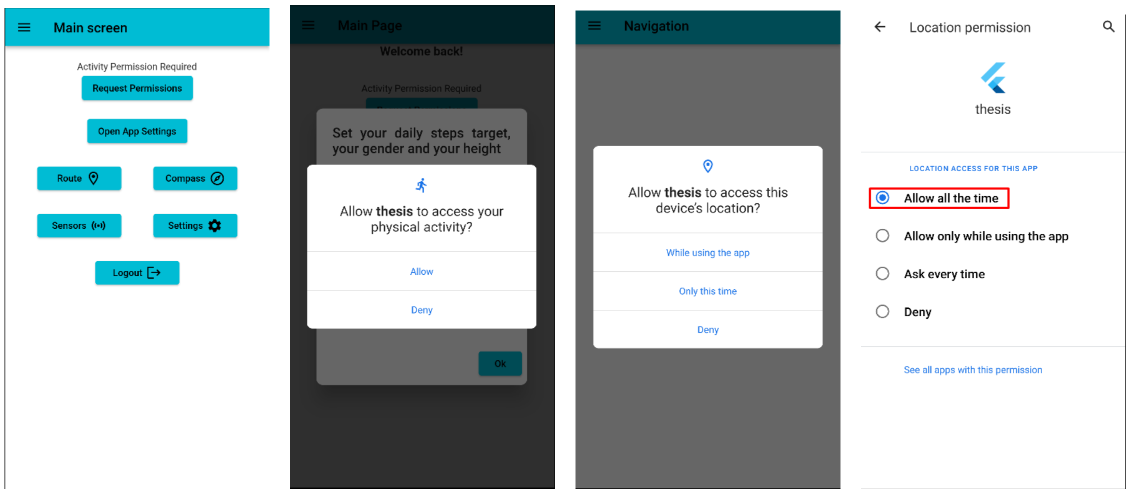
In order for this function to work the class must have an Observer by adding with `WidgetsBindingObserver()` after the name of the class. Then we initiated an Observer on `initState()` with `WidgetsBinding.instance?.addObserver(this)` and dispose it on `dispose()` function with `WidgetsBinding.instance?.removeObserver(this)`. It was not needed by default on every Flutter application to observe the state of the application but we wanted to know when the application go to the background or when the user is interacting with the application to know when we must enable or disable the background location functionality. Also we wanted to stop foreground functionality of the application when the user close the application.

6.5 Permissions

If the programmer wants to ask the user to collect data from the sensors of the device it is necessary to include permissions. Flutter does not provide how to ask permissions so we used the package 'permission_handler' [49]. We used this package for getting permissions for activity tracking (getting daily steps) and real time location (getting longitude and latitude). We added activity permission in the main file (main.dart) in which the user must accept the permission in order for the daily steps to be counted. The Figure 6.5 (a) shows the two buttons with two different options.

Figure 6.5 (b) shows the dialog that appears after the user presses the first button and asks if the user wants to give permission for activity tracking with a system pop up directly. The second button is for redirecting the user on the application settings in order for the user to manually accept the settings. The permission for the real time location is located in the navigation screen and in the compass screen. When the user selects for the first time the navigation screen the permission for the location will show up with a system pop up and the user have 3 options, to use location when the application is in use, to give one time permission of location to the application or to deny to give permission as shown on Figure 6.5 (c).

If the user deny on the pop up two buttons will appear on the screen, one for requesting permission for the location appearing the same pop up again and the other button will open the application's settings. Only on Android 10 and above in



Set your daily steps target, your gender and your height

Allow **thesis** to access your physical activity?

Allow **thesis** to access this device's location?

[✓ Allow all the time]
Allow only while using the app
Ask every time
Deny

(a) (b) (c) (d)

Figure 6.5: (a) Activity permission buttons and (b) Activity permission pop up (c) Location permission pop up (d) Application settings location permission on Android 10 and above

the options for the location permission there is a fourth option (Allow all the time) that the user must select through application's settings in order for the background location to work, if the user does not select this option the next time the user enters the navigation screen the application will redirect the user to the permissions options of the application to select it. Figure 6.5 (d) shows the application's settings for location permission.

If the user gives location permission for the first time to the application through navigation screen the application will not request the permission again on compass screen and if the user gives location permission on the first time on compass screen the application will not request the permission again on navigation screen.

6.6 Sensors

6.6.1 Pressure sensor

To get data from the pressure sensor we got through a small odyssey. We first searched if there was available a package that gets the pressure data from the sensor on both Operating Systems (Android and IOS) but there was not anything on the time we were building the functionality. So the next worst solution would be to write native code on Kotlin for Android and Swift for IOS and connect the native code with Flutter using platform channels. Thankfully we found exactly one Github project on which we were able to take some ideas and build on top of that [50]. So on Android side we created a main file with name MainActivity.kt and we created a variable a method channel and an event channel (in short both method channel and event channel exist for communicating with Dart, method channel is for sending information like status and event channel is for sending values even if they change dynamically). To connect two same types of channels between native and Dart the channels must have the same name (for example the variable for availability of pressure sensor is called pressure_sensor on Android, IOS and Dart file). The method channel purpose is for sending if the device has a pressure sensor or not. Below we can see the code on Android for the availability of the sensor.

```
//Channel for pressure
presschannel = MethodChannel(messenger, press_channel)
presschannel!!.setMethodCallHandler{
 call,result ->
  if (call.method == "isSensorAvailable") {
  result.success(sensorManager!!.getSensorList(Sensor.
    TYPE_PRESSURE).isNotEmpty())
 } else {
  result.notImplemented()
  }
}
```

The code for the data from the sensor we made a class `StreamHandler` in another file which basically registers a listener to get the events from the sensor as we can see below.

```
class StreamHandler(private val sensorManager: SensorManager,
    sensorType: Int, private var interval: Int =
   SensorManager.SENSOR_DELAY_NORMAL):
 EventChannel.StreamHandler, SensorEventListener {
 private val sensor = sensorManager.getDefaultSensor(
   sensorType)
 private var eventSink: EventChannel.EventSink? = null

 override fun onListen(arguments: Any?, events: EventChannel.
   EventSink?) {
  if (sensor != null){
   eventSink = events
   sensorManager.registerListener(this, sensor, interval)
  }
 }

 override fun onCancel(arguments: Any?) {
  sensorManager.unregisterListener(this)
  eventSink = null
 }

 override fun onSensorChanged(event: SensorEvent?) {
  val sensorValues = event!!.values[0]
  eventSink?.success(sensorValues)
 }

 override fun onAccuracyChanged(sensor: Sensor?, accuracy:
   Int) {
 }
}
```

Below we can see the call of the `StreamHandler` class in the main file.

```
pressureChannel = EventChannel(messenger, pressure_channel)
```

```
pressureStreamHandler = StreamHandler(sensorManager!!, Sensor
    .TYPE_PRESSURE)
pressureChannel!!.setStreamHandler(pressureStreamHandler)
```

For IOS we followed the same steps by making a .swift file having the main function inside. Then we made a variable for the method channel (pressure_sensor) and a variable for event channel (pressure_channel). Below we can see the function for checking if the pressure sensor is available on the current device.

```
let presschannel = FlutterMethodChannel(name: press_channel,
    binaryMessenger: controller.binaryMessenger)

presschannel.setMethodCallHandler({
  (call: FlutterMethodCall, result: @escaping FlutterResult)
    -> Void in
    switch call.method {
     case "isSensorAvailable":
       result(CMAltimeter.isRelativeAltitudeAvailable())
     default:
       result(FlutterMethodNotImplemented)
    }
})
```

We followed the same methodology for getting data from the sensor making another class inside the file to handle the events of the sensor as shown below.

```
class PressureStreamHandler: NSObject, FlutterStreamHandler {
 let altimeter = CMAltimeter()
 private let queue = OperationQueue()

 func onListen(withArguments arguments: Any?, eventSink
    events: @escaping FlutterEventSink) -> FlutterError? {

 if CMAltimeter.isRelativeAltitudeAvailable() {
    altimeter.startRelativeAltitudeUpdates(to: queue) { (data,
    error) in
    if data != nil {
    //Get pressure
    let pressurePascals = data?.pressure
    events(pressurePascals!.doubleValue * 10.0)
    }
```

```
  }
 }
 return nil
 }

 func onCancel(withArguments arguments:Any?) -> FlutterError?
    {
  altimeter.stopRelativeAltitudeUpdates()
  return nil
 }
}
```

Below we can see the function we called inside the main file.

```
let pressurechannel = FlutterEventChannel(name:
   pressure_channel, binaryMessenger: controller.
   binaryMessenger)
pressurechannel.setStreamHandler(pressureStreamHandler)
```

After this procedure the native code on both Android and IOS is done and we moved to connect the native code with Dart. The pressure functionality is inside the Sensors screen in which also the user can see if the device has pressure sensor or not and the data it gets from the sensor. First we created the corresponding method channel and event channel and then we made a function to check if the device has a pressure sensor. Below we can see the function for getting the availability of the pressure sensor.

```
//Future for checking the availability of pressure sensor
Future<void> check_pressure_availability() async {
 try {
  var available = await press_channel.invokeMethod('
   isSensorAvailable');
  setState(() {
  press_check = available;
 });
 } on PlatformException catch (e) {
  print(e);
 }
}
```

Then we called the function in `initState()` to be called once every time the application opens this screen to check if there is pressure sensor on the device and also listen to the `StreamHandler` to get data from the sensor in case there is as shown on below.

```
//pressure initialization event
pressureSubscription = pressure_channel.
   receiveBroadcastStream().listen((event) {
 setState(() {
  if(press_check == true){
   pressure=event;
   pmsg = '${pressure.toStringAsFixed(2)} mbar';
   if(press_check == false)
   {
     pmsg = 'Pressure not available';
   }
  }
  else{
   pmsg = 'Pressure not available';
  }
 });
});
```

6.6.2 Proximity sensor

To get data from proximity sensor it was much easier because there was a package called 'proximity_sensor' [51] which has pre-build functions to get data from the sensor. On Android to know if there is a proximity sensor on the device (or at least a virtual sensor) we made a method channel writing native code in the same file we made for pressure sensor (MainActivity.kt) to send the status of the sensor. Below we can see the Kotlin function for the proximity method channel.

```
//Channel for proximity
proxchannel = MethodChannel(messenger, prox_channel)
 proxchannel!!.setMethodCallHandler{
 call,result ->
  if (call.method == "isSensorAvailable") {
   result.success(sensorManager!!.getSensorList(Sensor.
   TYPE_PROXIMITY).isNotEmpty())
  } else {
     result.notImplemented()
```

```
    }
}
```

Then in the Sensors screen (Sensors.dart) we created a function for getting the availability of the sensor from the method channel. Below we can see the Dart function for getting from native code the availability of the sensor.

```
//Future for checking the availability of proximity sensor
Future<void> check_proximity_availability() async {
 if(Platform.isIOS){
  prox_check = true;
 }
 try {
  var available = await prox_channel.invokeMethod('
   isSensorAvailable');
  setState(() {
  prox_check = available;
 });
} on PlatformException catch (e) {
   print(e);
  }
}
```

Below we can see the function to get the data from the sensor.

```
//Future for gettind data from proximity sensor
Future<void> listenSensor() async {
 FlutterError.onError = (FlutterErrorDetails details) {
  if (foundation.kDebugMode) {
   FlutterError.dumpErrorToConsole(details);
  }
 };
 _streamSubscription = ProximitySensor.events.listen((int
   event) {
  setState(() {
   if(prox_check == true) {
    _isNear = (event > 0) ? true : false;
    if (_isNear == true) {
     nmsg = "'Yes'";
    }
    else {
     nmsg = "'No'";
    }
   }
   else{
    nmsg = 'Proximity not available';
   }
   print(nmsg);
  });
 });
}
```

On IOS there is no need to check for proximity sensor because every supported device to run this application has the sensor.

6.6.3 Accelerometer - Magnetometer – Gyroscope sensors

To get data from acceleration sensor, magnetometer sensor and gyroscope sensor we used the package 'sensors_plus' [52] which has pre-build functions to get data from the sensors but not the availability of them in the device. To get the availability on Android of the sensors we followed the same strategy as in proximity and pressure sensors by writing native code and using method channels for communicating with the Dart files. Below we can see the native code used for checking availability.

```
//Channel for accelerometer
accchannel = MethodChannel(messenger, acc_channel)
accchannel!!.setMethodCallHandler{
 call,result ->
 if (call.method == "isSensorAvailable") {
  result.success(sensorManager!!.getSensorList(Sensor.
   TYPE_ACCELEROMETER).isNotEmpty())
 } else {
    result.notImplemented()
   }
}

//Channel for gyroscope
gyrochannel = MethodChannel(messenger, gyro_channel)
gyrochannel!!.setMethodCallHandler{
 call,result ->
 if (call.method == "isSensorAvailable") {
  result.success(sensorManager!!.getSensorList(Sensor.
   TYPE_GYROSCOPE).isNotEmpty())
 } else {
    result.notImplemented()
  }
}

//Channel for magnetometer
magnchannel = MethodChannel(messenger, magn_channel)
magnchannel!!.setMethodCallHandler{
 call,result ->
 if (call.method == "isSensorAvailable") {
  result.success(sensorManager!!.getSensorList(Sensor.
   TYPE_MAGNETIC_FIELD).isNotEmpty())
 } else {
    result.notImplemented()
   }
}
```

Below we can see the Sensors screen, after we created three `Future` functions to get the availability of each sensor from the native code.

```
//Future for checking the availability of accelerometer
   sensor
Future<void> check_acc_availability() async {
 if(Platform.isIOS){
  acc_check = true;
}
 try {
  var available = await acc_channel.invokeMethod('
   isSensorAvailable');
  setState(() {
   acc_check = available;
  });
} on PlatformException catch (e) {
   print(e);
  }
}

//Future for checking the availability of gyroscope sensor
Future<void> check_gyro_availability() async {
 if(Platform.isIOS){
  gyro_check = true;
 }
 try {
  var available = await gyro_channel.invokeMethod('
   isSensorAvailable');
  setState(() {
   gyro_check = available;
  });
} on PlatformException catch (e) {
   print(e);
  }
}

//Future for checking the availability of magnetometer
Future<void> check_magn_availability() async {
 if(Platform.isIOS){
  magn_check = true;
 }
 try {
  var available = await magn_channel.invokeMethod('
   isSensorAvailable');
  setState(() {
    magn_check = available;
  });
} on PlatformException catch (e) {
  print(e);
  }
}
```

We called the functions inside the `initState()` function and to get data from each sensor and we also initiated inside the `initState()` the events listeners for the data streams as shown below.

```
//accelerometer initialization event
userAccelerometerEvents.listen((UserAccelerometerEvent event)
    {
  setState(() {
    if(acc_check == true){
      ax = event.x;
      ay = event.y;
      az = event.z;
      amsg='x:${ax.toStringAsFixed(2)} y:${ay.toStringAsFixed
   (2)} z:${az.toStringAsFixed(2)}';
    }
    else{
      amsg='Accelerometer not available';
    }
  });
});

//gyroscope initialization event
gyroscopeEvents.listen((GyroscopeEvent event) {
  setState(() {
    if(gyro_check == true) {
      gx = event.x;
      gy = event.y;
      gz = event.z;
      gmsg = 'x:${gx.toStringAsFixed(2)} y:${gy.
   toStringAsFixed(2)} z:${gz.toStringAsFixed(2)}';
    }
    else{
      gmsg='Gyroscope not available';
    }
  });
});

//magnetometer initialization event
magnetometerEvents.listen((MagnetometerEvent event) {
  setState(() {
    if(magn_check == true){
      mx = event.x;
      my = event.y;
      mz = event.z;
      mmsg='x:${mx.toStringAsFixed(2)} y:${my.toStringAsFixed
   (2)} z:${mz.toStringAsFixed(2)}';
    }
    else{
      mmsg='Magnetometer not available';
    }
  });
});
```

Again for IOS there is no need to check for accelerometer, gyroscope and magnetometer sensors because every supported device to run this application has all these sensor.

6.6.4 Pedometer sensor

For counting steps we had to get data from the pedometer sensor and once again we were lucky because there is a package named ‘pedometer’ [53] that does this exact functionality. Below we can see the functions we created to get the number of steps, to show the availability and to initialize them.

```
void onStepCount(StepCount event) {
 print(event);
 setState(() {
  if(box.get('today_steps')==null){
    box.put('today_steps',0);
  }
  else{
    box.put('today_steps',box.get('today_steps') + 1);
    dist = double.parse(((box.get('today_steps') * box.get('
   steps_length'))/ 1000).toStringAsFixed(3));
  }
 });

 box.put('date',date_once);
}

 void onStepCountError(error) {
  print('onStepCountError: $error');
  setState(() {
   steps = 'Pedometer not\navailable';
  });
 }

 void initPlatformState() {

  _stepCountStream = Pedometer.stepCountStream;
  _stepCountStream.listen(onStepCount).onError(
   onStepCountError);

  if (!mounted) return;
 }
```

The `onStepCount()` function counts every step the user does and saves it to a local database (more on database section), the `onStepCountError()` function checks if the device has a pedometer sensor and the `initPlatformState()` function initializes the pedometer and registers the changes, it is called in the `initState()` function.

6.7 Main screen

Main screen (main.dart) is extended with a stateful widget. It consists by seven buttons at first. If the user's device is Android 10 and above the screen will show two buttons in which the first one is requesting the permission to register physical activity in order to count the daily steps, the second button to open the application settings to manually give the physical activity permission (if the user's device is Android 9 or 8 the two buttons won't appear) and the other five buttons is to redirect the user to each screen and to logout the user. After the user gives the required permission an `Alert Dialog` will appear asking for the user to enter height, daily steps target and the gender. As seen on previous text-fields, each text-field requires a `TextEditingController` with a listener, a function to know if the user interacts with the current text-field for the first time and a function to check the context of the text-field. On the second text-field for example the user must enter a height between 0 and 250cm and requires also the text-field to not be empty. Below we can see the function we have created to check if the text-field is empty.

```
//Function for displaying the correct error message on height
//text-field
String? Height_Text-field_check(){
 String height_msg='';
 if(heightController.text.isEmpty==true){
  height_msg='Height can\'t be empty';
  print(height_msg);
  height_check=false;
  return height_msg;
 }
 else if(int.parse(heightController.text) > 250){
  height_msg='Height must be less than 250cm';
  print(height_msg);
  height_check=false;
  return height_msg;
 }
 else if(int.parse(heightController.text) <= 250){
  height_check =true;
  height_msg='Valid height';
  print(height_msg);
  }
}
```

After entering all the requirements a widget called `Card` will appear on the screen with the daily target of steps and the progress of them in a circular progress bar. The `Card` uses an elevation option inside the widget to be give the user a more 3d aspect of the widget (the daily steps are saved in a database, more on that topic on database section). On the top right corner there is a target icon which is a button, when the user presses the button another Alert Dialog will show up to give the option to the user to change the height and the daily steps target. On the bottom of the `Card` the user can see the kilometers by foot calculated by the number of steps and the height that the user has given. If the device does not have a pedometer sensor the package we used named 'pedometer' will try and calculate the number of steps by the motion of the device using the accelerometer and the gyroscope of the device, not very accurate though.

6.8 Route screen

Route screen (navigation.dart) is also extended with a stateful widget. When the user enters the screen for the first time if has not already given the permission to track real time location before showing the map two buttons will appear, the first one to request permission for real time location and the second one to open application's settings to manually give permission. After giving the permission the map will appear with a button on the left bottom corner. It's important that the user must have enabled the GPS if opening the screen for the first after a full shutdown and be connected to the internet in order for the map to load, due to a bug in initialization of positioning the marker will not show the correct location of the user if enters the screen with the GPS disabled and enable it after entering. For maps there are not many options if the programmer does not want to implement native maps. The first and the most popular choice is to use the package 'google_maps_flutter' [54] which is the best package for implementing maps on flutter due to having official support from Google and a special functionality which is camera animation. The biggest flaw of this package is that it requires to get an API key from google maps platform which is free for only a specific amount of API calls which can easily be reached, after the limit it is not free. So the next best choice is the package 'flutter_map' [55] which is the one that we have used and it is free. It is community based which means that is less supported and the updates are less often. It is a Dart implementation of Leaflet Maps with tiles from Open Street Maps which are also free. This plugin by itself does not provide other functionalities other than showing the map with a small dot where the user is currently on the map for the first time opening the map. If for example the user moved from the point was before he/she has to select another screen in the application and select again the navigation screen. In order to show on the map every time the user changes location we used the package 'flutter_map_location_marker' [56] which also has a marker with the heading of the device. To get the changes of position we integrated the package in the FlutterMap widget with a LocationMarkerLayer widget as shown below.

```
LocationMarkerLayerWidget(
 plugin: LocationMarkerPlugin(
  centerCurrentLocationStream:
   center_current_location_StreamController.stream,
  centerOnLocationUpdate: center_on_location_update
),
 options: LocationMarkerLayerOptions(
  marker: DefaultLocationMarker(
   color: Colors.green,
   child: Icon(
   Icons.person,
   color: Colors.white,
   ),
  ),
  markerSize: const Size(40, 40),
  accuracyCircleColor: Colors.green.withOpacity(0.1),
  headingSectorColor: Colors.green.withOpacity(0.8),
  headingSectorRadius: 120,
  markerAnimationDuration: Duration(milliseconds: Duration.
   millisecondsPerSecond),
 ),
)
```

We also used the package 'location' [57] to get the latitude and longitude as value because the flutter_map_location_marker did not provide the information we needed but only the representation on the map. To get the location in the foreground or in the background the location package provides a function and all we had to do was to enable it through the lifecycle (as mentioned in the Lifecycle section above) when the application goes to the background and disable it when the application is resumed. We initialiazed the function `onLocationChanged`.listen in `initState()` to get the location changes and called a function to save the coordinates and check for the GPS status as shown below.

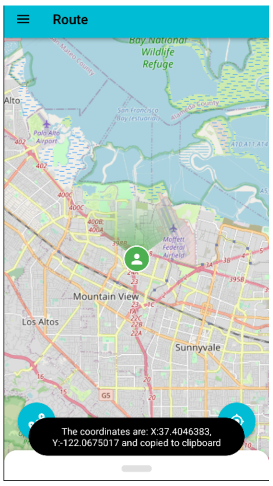
```
location.onLocationChanged.listen((loc.LocationData cLoc) {
 currentLocation = cLoc;
 setState(() {
  setpoint(cLoc.latitude, cLoc.longitude);
  speed = cLoc.speed! * 3.6;
 });
 insert_toDb();
});
```

```
void setpoint(latitude,longitude) async{
 serviceEnabled = await geo.Geolocator.
   isLocationServiceEnabled();

 lat=latitude;
 lng=longitude;
}
```

We added a button on the right bottom corner in case that the user wanted to navigate the map without losing where the user is on the map, when the user presses the button the screen will show the user current location with the marker in the center of the screen. We also wanted to have a way for the user to see his/hers route on map with a line (called polyline), so we used the package 'flutter_map_tappable_polyline' [58] which shows with a red line the route that the user has travelled. Below we can see the code of how we showed the line using a temporary list with coordinates with the package as a widget inside the `FlutterMap` widget. When the user presses the button for around 3 seconds the coordinates are copied to clipboard with a small pop-up warning the user. Figure 6.6 shows how the toast popup looks like.

```
TappablePolylineLayerWidget(
 options: TappablePolylineLayerOptions(
  polylineCulling: true,
  pointerDistanceTolerance: 20,
  polylines: [
  TaggedPolyline(
   tag: 'My Polyline',
   // An optional tag to distinguish polylines in callback
   points: polylineCoordinates,
   color: Colors.red,
   strokeWidth: 9.0,
```



The coordinates are: X:37.4046383, Y:-122.0675017 and copied to clipboard

Figure 6.6: Coordinates toast pop-up

```
    ),
  ],
  onTap: (polylines, tapPosition) => print('Tapped: ' +
    polylines.map((polyline) => polyline.tag).join(',') + 'at'
     + tapPosition.globalPosition.toString()),
  onMiss: (tapPosition){
    print('No polyline was tapped at position' + tapPosition.
    globalPosition.toString());
  }),
),
```

This package [58] has also a function called onTap in which the user can tap a polyline and in our case print the position of the polyline on map. The user can use the functionalities of the map without connecting to the internet and can see a part of the map that was saved the last time the user used the map with the device connected to an internet connection. We used the package 'cached _network_image' [59]. So we built a function to get an image of what the user sees on screen and we save it based on the coordinates and the options we have set on the map widget as shown below.

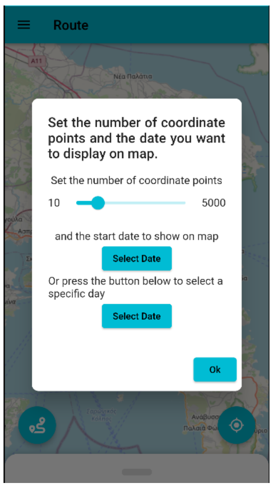
```
class CachedTileProvider extends TileProvider {
  const CachedTileProvider({customCacheManager});
  @override
  ImageProvider getImage(Coords<num> coords, TileLayerOptions
    options) {
   return CachedNetworkImageProvider(
    getTileUrl(coords, options),
    //Now you can set options that determine how the image
    gets cached via whichever plugin you use.
   );
  }
}
```

Basically when the user uses the map with the device connected to the internet this package saves a number of tiles around the user and the map is saved as an image inside the package of the application and the image is kept for 30 days, after that the image gets deleted automatically. This method in order to work requires to built a Cache Manager to save the image, so we used the package 'flutter_cache_manager' [60] in which we can configure the number of objects we want to save and the duration of the data we want to be kept as seen below.

```
static final customCacheManager = CacheManager(
 Config(
  'customCacheKey',
  stalePeriod: Duration(days:30),
  maxNrOfCacheObjects: 200
 ),
);
```

On the bottom left corner we added another button to give the option to user to select the route he/she wants to load on the map with polylines. There are two options available, the first one is to select one date until the current date and load with a slider a number of coordinates points from 10 to 5000 (the user will be warned that selecting more than 2000 coordinates points are only suggested if the user has a high end device). If the number of points on the given dates are more than the number that the user has selected then it will be loaded the first number of coordinate points that the user has selected. The second option is to select a specific day and load the route of this specific date. Again if the coordinate points are more than 2000 points it will load the first 2000 points on the map of the specific day. If on both selected options there are not any coordinate points saved a toast pop-up will appear and warn the user that there are not any saved route on these dates. When the user presses the button for around 3 seconds the polylines will reset and disappear from the map. Figures 6.7 (a) and (b) shows the Alert dialog with the two options as mentioned above and the DatePicker for selecting the date.

Finally, on the bottom of the screen there is a small grey circular button, when it is pressed there is sliding panel appearing from bottom to top. We used the package 'sliding_up_panel' [61]. When it is opened, it shows the total distance in kilometers


Route

Set the number of coordinate points and the date you want to display on map.

Set the number of coordinate points

10 5000

and the start date to show on map

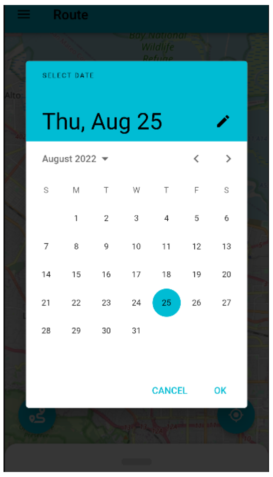
Select Date

Or press the button below to select a specific day

Select Date

Ok

(a)


SELECT DATE

Thu, Aug 25

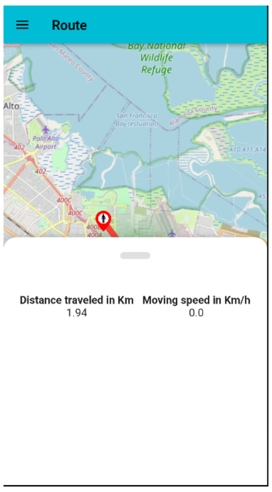
August 2022

S	M	T	W	T	F	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

CANCEL OK

(b)

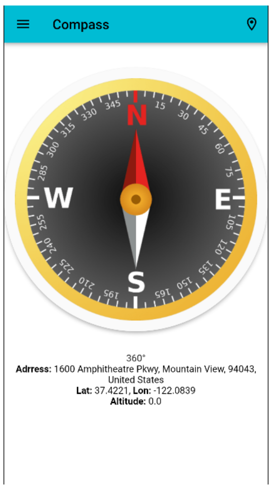
Figure 6.7: (a) Alert dialog of the left button (b) Route date picker


Route

Distance traveled in Km 1.94

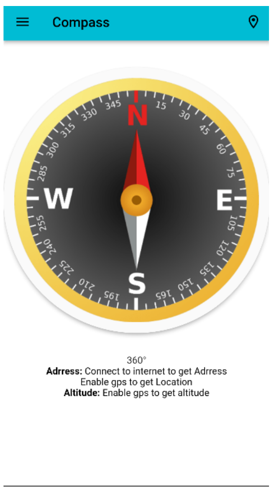
Moving speed in Km/h 0.0

(a)


Compass

360°
Adrress: 1600 Amphitheatre Pkwy, Mountain View, 94043, United States
Lat: 37.4221, **Lon:** -122.0839
Altitude: 0.0

(b)


Compass

360°
Adrress: Connect to internet to get Adrress
Enable gps to get Location
Altitude: Enable gps to get altitude

(c)

Figure 6.8: (a) Sliding panel (b) Compass screen with enabled GPS and internet connection (c) Compass screen with disabled GPS and disabled internet connection.

if the user has selected to show a route on the map and the moving speed in km/h in real time. Figure 6.8 (a) shows the Sliding panel of the Route Screen.

6.9 Compass screen

Compass screen (Compass.dart) is extended with a stateful widget due to having changing values. In order to use this screen the user must give the location permission to the application (we used the permission_handler package mentioned on the permissions section above). As in the navigation screen, when the user enters for the first time on this screen will see two buttons, the first one to ask permission with a native pop up message to give directly the location permission and the second button to direct the user to the application's settings to give the location permission manually. If the user has already given the location permission to the application through the navigation screen then the two buttons will not show up. After the user gives the location permission four text-fields show up and an image of a compass. The first text-field shows the angles from the north, the second text-field show the address that the user is currently, the third one shows the latitude and longitude and the fourth one shows the altitude. In order to get the latitude and longitude the user must enable the GPS on the device. If the user enters the screen without the GPS enabled he/she has to press the button on the top right corner to get the coordinates. In order to get the address and the altitude the user must have an internet connection. Figure 6.8 (b) and (c) shows the screen, on the left when the device has internet connection and an enabled GPS and on the right when the device has not an internet connection and the GPS is disabled. To get the coordinates we used the package 'geolocator' [62]. Below we can see the function we created to get the status of the GPS and also the current position of the user.

```
//Function for getting the status of GPS
Future<Position> getGeoLocationPosition() async {

 serviceEnabled = await Geolocator.isLocationServiceEnabled()
   ;

 return await Geolocator.getCurrentPosition(desiredAccuracy:
   LocationAccuracy.best);
}
```

The function returns a variable of type position which contains information of the current latitude, longitude and altitude. Then to find the current address from the coordinates we used the package 'geocoding' (https://pub.dev/packages/geocoding) and then we created the function below to get the address given the coordinates we found with the previous function (we print on the screen not only the address but also the country and the postal code) as shown below.

```
//Function for getting lat lng and
Future<void> GetAddressFromLatLong(Position position)async {
 List<Placemark> placemarks = await placemarkFromCoordinates(
   position.latitude, position.longitude);
 double alt_placemarks = await position.altitude;
 print(placemarks);
 Placemark place = placemarks[0];
 setState(()  {
  Address = '${place.street}, ${place.locality}, ${place.
   postalCode}, ${place.country}';
 });
}
```

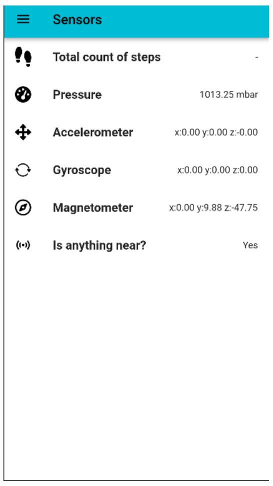
Then we created the function shown below to call the two functions mentioned above and initialized it in `initState()` in order every time the user enters the screen to show the corresponding messages.

```
//Function for setting address and location
  void getData() async {
    Position position = await getGeoLocationPosition();
    lat=position.latitude.toStringAsFixed(4);
    lng=position.longitude.toStringAsFixed(4);
    Altitude = position.altitude;

    print('$location');
    GetAddressFromLatLong(position);
  }
```

For the compass we used the package 'flutter_compass' [63]. Just like a physical compass this package uses the magnetometer of the device and shows the angles from north that the device is heading to. The compass is a single image of a compass and the package uses the gyroscope of the device to rotate the image. The package provides a stream of the angles and sets as 0 when the device is heading to north,0-179 when the device is to the right of the north and -180-0 when the device is heading to the left of the north. We did not like this approach so we created a simple function to show to the user the angles from 0-359. Below we can see the simple function we created to get the angles.

```
//Function for getting the angles of compass
void get_angle(event) {
 setState(() {
  if(event.heading>0){
   angle = event.heading;
  }
  else{
   angle = event.heading + 360;
  }
 });
```



Total count of steps	-
Pressure	1013.25 mbar
Accelerometer	x:0.00 y:0.00 z:-0.00
Gyroscope	x:0.00 y:0.00 z:0.00
Magnetometer	x:0.00 y:9.88 z:-47.75
Is anything near?	Yes

Figure 6.9: Sensors screen

```
}
```

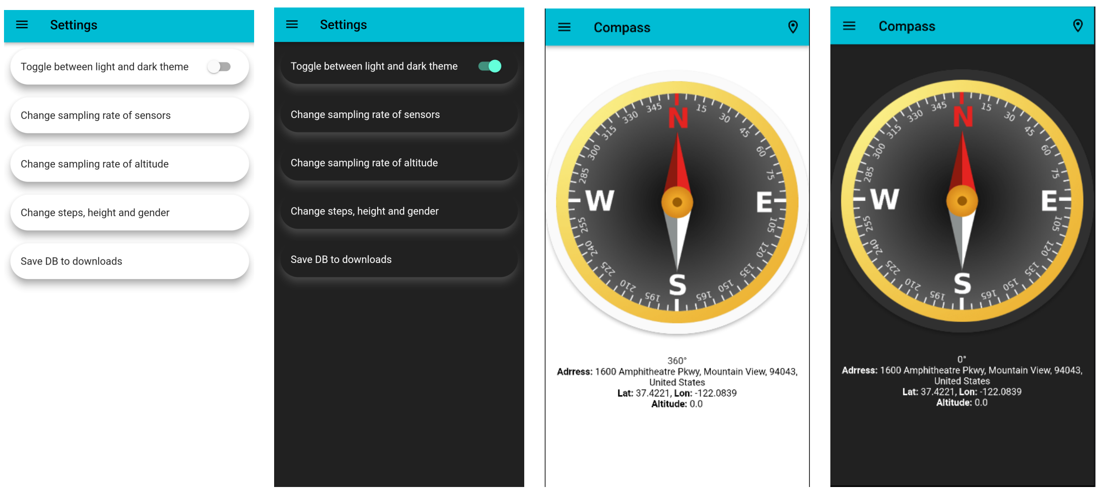
We integrated the package as widget inside the screen and is build every time the user enters the screen and is initialized in `initState()` function.

6.10 Sensors screen

Sensors screen (Sensors.dart) is extended with a stateful widget. It consists by six ListTile. Next to each sensor, the user can see the availability of the sensor or if the device has a sensor the data it creates (the functionality of each sensor is mentioned on the Sensors section above). This screen exists only for showing the user the available sensors on the device and their values when an event is occurred (for example when the user spins the device), so the user can not interact with this screen. Figure 6.9 shows the sensors screen.

6.11 Settings screen

Settings screen (**Settings.dart**) is extended with a stateful widget. It consists by five card widgets, the first one has a `SwitchListTile` widget inside the `Card` widget which changes the theme of the whole application from light to dark and vice versa. It has a field which holds a value, that the user can change between the two values by pressing the toggle button (SwitchListTile wiget). To manage the theme, we created the file Theme_provider.dart in which we have the functionality to change the theme of the application. To easily use this file we used the package 'provider' [64] which is a wrapper around `InheritedWidget`. In Theme_provider.dart file we have two classes, in the first one we have a function to set a boolean variable to change between light and dark theme so we can use it in the settings screen and the



Settings

Toggle between light and dark theme

Change sampling rate of sensors

Change sampling rate of altitude

Change steps, height and gender

Save DB to downloads

Settings

Toggle between light and dark theme

Change sampling rate of sensors

Change sampling rate of altitude

Change steps, height and gender

Save DB to downloads

Compass

360°
Adrress: 1600 Amphitheatre Pkwy, Mountain View, 94043, United States
Lat: 37.4221, **Lon:** -122.0839
Altitude: 0.0

Compass

0°
Adrress: 1600 Amphitheatre Pkwy, Mountain View, 94043, United States
Lat: 37.4221, **Lon:** -122.0839
Altitude: 0.0

(a) (b) (c) (d)

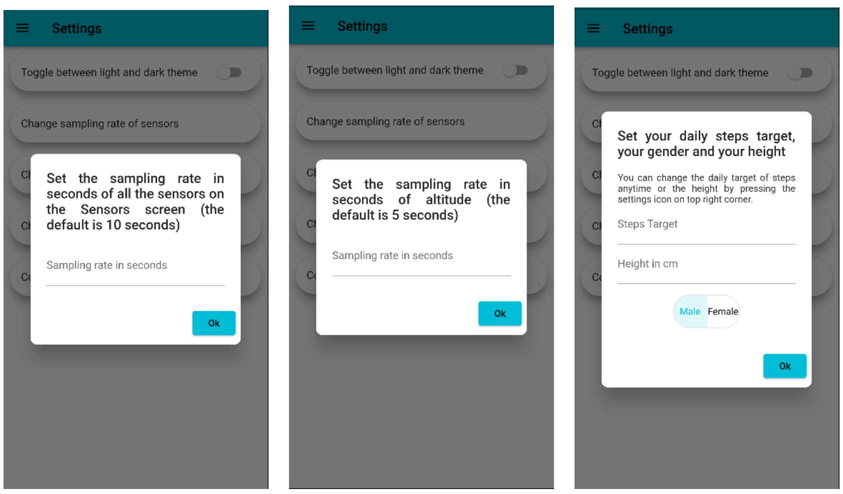
Figure 6.10: (a) Settings screen with light theme (b) Settings screen with dark theme (c) Compass screen with light theme (d) Compass screen with dark theme

second class is where we have the light and the dark theme colors for everything we use in the application. The color of the background change from white to dark grey and the color of the text changes from black to white. By default the selected theme is the selected theme that the device is using, for example if a user's device is set to dark theme then when the user opens the application, the application's theme will be also dark. This is done by getting the system's theme in the Theme_provider file and setting it in the main file when we build the whole application, before the user encounters the first screen when opening the application as shown below.

```
return MaterialApp(
 title: 'Sodasense',
 themeMode: themeProvider.themeMode,
 theme: MyThemes.lightTheme,
 darkTheme: MyThemes.darkTheme,
 debugShowCheckedModeBanner: false,
 home:WithForegroundTask(
  child: (saved_mail !=null && saved_pass!=null)? MyHomePage
   () : Login()
 )
);
```

Figures 6.10 (a) with (c) and (b) with (d) shows on the left the application using light mode and on the right using the dark mode.

The second and the third card widget gives the ability to the user to change the


Settings

Toggle between light and dark theme

Change sampling rate of sensors

Set the sampling rate in seconds of all the sensors on the Sensors screen (the default is 10 seconds)

Sampling rate in seconds

Ok

Settings

Toggle between light and dark theme

Change sampling rate of sensors

Set the sampling rate in seconds of altitude (the default is 5 seconds)

Sampling rate in seconds

Ok

Settings

Toggle between light and dark theme

Set your daily steps target, your gender and your height

You can change the daily target of steps anytime or the height by pressing the settings icon on top right corner.

Steps Target

Height in cm

Male Female

Ok

(a) (b) (c)

Figure 6.11: (a) Sensors sampling rate dialog (b) Altitude sampling rate dialog (c) Alert dialog of changing the height, daily steps and the gender of the user

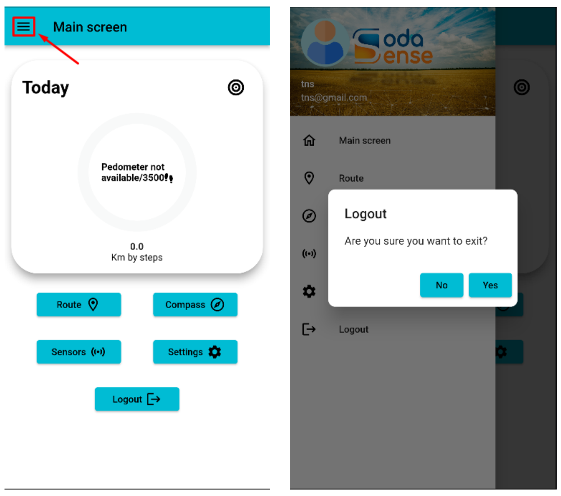
sampling rate of the sensors(pressure, accelerometer, gyroscope, magnetometer and proximity) and the sampling rate of the altitude accordingly with and Alert dialog. Figures 6.11 (a) and (b) shows the Alert dialogs of the aforementioned card widgets.

The card widget 'Change steps, height and gender' gives the ability to the user to change the height of the user, the daily steps target and the gender of the user just like on the Main Screen. Figure 6.11 (c) shows the Alert Dialog of this card widget. Finally, the last option 'Save DB to downloads' gives the option to the user to extract a copy of the local database of the device to the Downloads folder of the device and it was built mainly for debugging purposes.

6.12 Sidemenu

Sidemenu (Sidemenu.dart) is the only 'screen' that is extended with a stateless widget because we do not have a state to change. We chose to use Drawer widget to navigate through our screens as we though it was more convenient and more beautiful for the user. The next option was tabs but because we have multiple screens on the application it would be more difficult for the user to select a tab. To open the sidemenu the user must press the three parallel lines on the top left corner of each screen or slide from the left of the screen to the right to show up. Figure 6.12 (a) shows the three parallel lines the user must press in order for the sidemenu to appear.

It consists by 2 Texts widgets and 6 ListTiles widgets. In the 2 texts we show the username and the e-mail that the user has used to log in, and the first 5 ListTiles are used to navigate to each one of our screens (Homepage, Route, Compass, Sensors,



(a) (b)

Figure 6.12: (a) Button that opens the Sidemenu (b) Dialog message when the user presses the Log out button

Settings). The last ListTile is used to log out the user from the application and also the database with a pop up dialog, it also terminates the background location functionality of the application. In order for the user to log out he/she must press the log out ListTile and select "Yes" in the dialog, else the "No" option will close the dialog and won't disconnect the user. Figure 6.12 (b) shows the logout Alert dialog.

6.13 Databases

For saving the data that the application creates was one of the most difficult tasks we encountered on the creation of the application. We wanted the application to save the data locally on device and when the device is online to synchronize the data to a server that is hosted to our university. We though to use a NoSQL database so we searched for a solution (a package) that provides offline and online database self hosted (not on cloud). The most popular choice that provides the above requirements is ObjectBox [65]. ObjectBox is a database that the programmer can create objects and store them to the database. ObjectBox for the local database is easy to install, all we had to do was to install the package and start working. But to synchronize the data to our hosted database we had to request a key with the program from the company in order to work. After requesting the key, we answered to a Google Forms questionnaire for the reasons we selected the ObjectBox and why we wanted to use a hosted database and after this form we had answer another form about some personal data and billing address even though the hosted database was free only to realize that this database did not have built-in authorization and authentication functions. So we searched for other solutions and we found Couchbase lite which is also a NoSQL database that offers the same functionalities but not authentication and authorization. So we decided to take a different approach and have a separate database for saving the data locally and a separate database hosted on the server of the university. So we decided to host a Mongodb database on the server of the university and for the local database we searched for the best solution. First we tried NoSQL databases and we had multiple packages to choose:

- ObjectBox
- Hive
- SharedPreferences
- Sembast
- Couchbase Lite
- Realm

ObjectBox, Hive, Sembast, Couchbase Lite and Realm work similar and support creating objects to save them in their database and all them support CRUD operations. SharePreferences wraps platform-specific persistent storage for simple data (NSUserDefaults on IOS and macOS, SharedPreferences on Android). Data may be persisted to disk asynchronously, and there is no guarantee that writes will be persisted to disk after returning. So we decided to use Hive as it is the most lightweight and is written in pure Dart. The idea of the Hive database is simple, the programmer creates a box with a name and put data to the box with the function `put()`. Then in order to retrieve the data the programmer must use the function `get()`. The data can be either variables, list, maps and objects. In order to save objects to the database programmer must generate a type adapter. The update operation is only available if the programmer extends the object with `HiveObject`. When we started using the Hive database we noticed that the update operation of the database is not the same like an SQL database, because if for example we added a list to a box, the new list we added overwrites the previous one and we wanted to keep the data from the first time the user registers to the application until every time the user enters the application again. So we used Hive only for some basic tasks, like saving the e-mail and the password of the user to keep a session, so the user every time he/she closes the application and opens it again to be redirected to the main screen without the need of re-typing the credentials, for saving the target of steps and the daily steps and for saving the user's height and gender. To check if the user has already logged in to the application without logging out by pressing the ”Logout” button in the Sidemenu, we checked before running the main function of the application if the 'e-mail' and 'pass' fields in the box are empty to redirect the user to appropriate screen. Below we can see the function to check if there is an open session.

```
void main() async{
 //Hive commands for initializing and opening a box to store
   data
 await Hive.initFlutter();
 // Hive.registerAdapter(UserAdapter());
 box = await Hive.openBox('user');
 check_session();
 runApp(MyApp());
}
void check_session() async{
 saved_mail = await box.get('email');
 saved_pass = await box.get('pass');
}
```

```
return MaterialApp(
 title: 'Sodasense',
 themeMode: themeProvider.themeMode,
 theme: MyThemes.lightTheme,
 darkTheme: MyThemes.darkTheme,
 debugShowCheckedModeBanner: false,
 home:WithForegroundTask(
  child: (saved_mail !=null && saved_pass!=null)? MyHomePage
   () : Login()
 )
);
```

So we searched for available SQL packages and we found Drift, Floor and Sqflite. Drift is built on top of Sqlite, Floor is an abstraction of Sqlite and Sqflite is Sqlite package for Flutter. We ended up using Sqflite as it is the most stable package [66]. We used SQL database for saving the coordinates of the user, for saving data from the sensors (pressure, acceleration, gyroscope, magnetometer, proximity and pedometer) every 10 seconds and altitude every 5 seconds. We have created a file (SqlDatabase.dart) containing all the SQL statements (including creation of tables, insertion to tables and selection from tables) and the required functions for the database like the creation of the database file (db.db) and the initialization of it. We have 4 tables for saving the data we need (coordinates, altitude, daily_steps, sensors). The daily steps are inserted into the SQL database once per different day, we save the date of the last date and we check if the date of the last step is the different than the current day, if it is we insert the steps into the database with the current date. Of course in order for this function to work properly we execute it before we start counting the current's date steps. Below we can see the function of checking the dates which is in `initState()` function and the function to insert into the database table (these functions are inside the main.dart file).

```
if((box.get('date') != date_once) && (box.get('date') != null
   )){
 insert_toDb();
 box.put('today_steps',0);
}
```

```
insert_toDb() async{
 int stp = box.get('today_steps');
 date = DateTime.now().millisecondsSinceEpoch;
 await SqlDatabase.instance.insert_daily_steps(date,stp,0);
 List<Map> lista = await SqlDatabase.instance.
   select_daily_steps();
 print(lista);
}
```

For saving the coordinates the strategy was much simpler, we decided to save the coordinates to the database every time the user changes position, so this was done in the function we have the listener for changing position just as we have on the Route screen (the functions are inside the main.dart file). Below we can see the function for inserting the coordinates to database table.

```
location.onLocationChanged.listen((loc.LocationData cLoc) {
 currentLocation = cLoc;

 setState(() {
  setpoint(cLoc.latitude, cLoc.longitude);
  speed = cLoc.speed! * 3.6;
 });

 insert_toDb();
});
```

For saving the altitude again the strategy was simple, we decided to save the altitude every 5 seconds to the database using a timer (this was done inside the main.dart file). We followed the same strategy with saving the data, we checked for the availability of them and then saved the data every 10 seconds and the functionality is inside the main.dart file in initState function. Below we can see the function for inserting the sensors data to the database tables.

```
timer = Timer.periodic(Duration(seconds: srt), (Timer t) {
  if(acc_check == true){
    insert_acc_toDb();
  }
  else{
    ax=0;
    ay=0;
    az=0;
  }
  if(gyro_check == true){
    insert_gyro_toDb();
  }
  else{
    gx=0;
    gy=0;
    gz=0;
  }
  if(magn_check == true){
    insert_magn_toDb();
  }
  else{
    mx=0;
    my=0;
    mz=0;
  }
  if(press_check == true){
    insert_pressure_toDb();
  }
  else{
    pressure=0;
  }
  if(prox_check == true){
    insert_prox_toDb();
  }
  insert_sensors_toDb();
});
```

After we solved the problem of saving the data locally, we had to resolve the issue of copying the data from the device to a MongoDB database in a server of our university. Before that we wanted somehow to authenticate and authorizing the user and we ended up using Keycloak. Keycloak is an open source software to allow single sign-on with Identity and Access Management [67]. We selected Keycloak because it is a self hosted service and was easy to setup. In order to connect the application with the Keycloak because it is not safe to be directly connected with each other we had to make some web services (one for the login of the user, one for the registration of the user to the Keycloak etc.). After we built the web-services all we had to do in the application was to make an http post request to web-services in order for the data to be transferred to the Keycloak using the package 'http' [68].

6.14 Webservices

As we mentioned in the previous section, in order to connect the application with Keycloak, send and receive data to MongoDB we created 6 web services using PHP. To communicate with each web service we send http post requests and requires the data to be encoded as .json file. In every web service we check if the required field is empty and returns an error message. Starting with Register screen, we created the web service userRegister.php which requires email, username, lastname, firstname and password. If the email does not exist from other user then the registration of the user is successful. For Login screen we created the userLogin.php web service in which we send the username and the password of the registered user. If the login is successful the web service returns an acccess token that the Keycloak has sent to allow the user to connect to the application else if the login is not successful it returns an error message. The same web service (userLogin.php) is called after a successful user registration in order to get an access token. Finally, for the Main screen we created four web services for each of the SQLite tables we have on the device. altitudeData.php is for uploading the altitude data, dailystepsData.php for uploading the daily steps, userTrackingData.php for uploading the coordinates and sensorsData.php for uploading pressure, acceleration, gyroscope and magnetometer (for each one the x, y and z axis), proximity and steps. Every one of these web services (altitudeData.php, dailystepsData.php, userTrackingData.php, sensorsData.php) requires on each http post request the access token, the user id and then the specific list of each attribute, i.e. altitudeData.php requires the list with the altitude data, dailystepsData.php requires the list with the daily steps data etc.

6.15 Conclusions

After having a complete breakdown of the development of the framework, we can see the strong points of this work, the struggles we encountered and the problems we solved. On the parts of the application where we send and receive data like Login, Register and Main screen we followed the best security practices to ensure that a man in the middle cannot acquire access to our framework. The foreground functionality, although it was a difficult task to complete, we wanted to make sure that all the capabilities of the application like gathering data from sensors must be working. Finding also the best and most suitable database for both local and on server side, was solved after trying multiple packages and technologies. Finally, we built web-services to create a middle layer to our framework between the application and the database on server to take on the task of send, receive and process data to Keycloak and to MongoDB to be sure every task is completed.

Chapter 7

Conclusions and future work

In our days, a high percentage of the people have a smartphone both Android and IOS and want to know their daily movement. In the present thesis, a complete solution was given, a framework that includes a user's tracking application, a way of sending automatically the saved data of every device to our server and the manner of saving data to a hosted database of our selection to a university server. The application includes all the necessary functions that a user needs to register all of the movement on a daily basis, the user's route that can be selected using calendar and the ability of the user to manipulate the generated data by the user's need and by the way every user wants. Our focus was on the convenience and usability. During the development of the framework we encountered some major problems but were solved with the help of the Flutter community, forums, with videos and with lots of troubleshooting. Even though the thesis journey has come to an end the framework will grow as seen on the section below.

7.1 Future extensions

This is only the start of this project and many changes have been initiated. We have gathered some rough thoughts for future expansion starting by adding functionality to count the daily user's route distance and by adding animations to the application. We should also fill the settings screen with more options for example to change the user's username and password. If we want to expand the application we should try different databases like **Timescale** and **Influx** in order to understand which one is the most suitable for our framework. Also, a web application for checking all of the data that a user's device has created would be a good addition. Likewise, in our days a good addition to our framework would be a chat bot, to make it easier for the users e.g. to count the total days of using the application or the total distance of the registered route or even the habits of the user by using the data of the movement by the sensors. Finally a crucial part of expansion would be to find and fix bugs that may or may not be present in the future due to the use of the application by a large number of users, as its extensive use will lead to the detection of any bugs and the testing on real environment and testing on a high volume of users.

References

[1] https://docs.flutter.dev/resources/architectural-overview#architectural-layers.

[2] Matt Neuberg. *iOS 15 Programming Fundamentals with Swift: Swift, Xcode, and Cocoa Basics.* O'Reilly Media, 2021.

[3] Dawn Griffiths. *Head First Android Development: A Brain-Friendly Guide.* Shroff/O'Reilly, 2017.

[4] Raymond K. Camden. *Apache Cordova in Action.* Manning, 2015.

[5] Andreas Dormann. *Ionic 6: Create awesome apps for iOS, Android, Desktop and Web.* D&D Verlag Bonn, 2022.

[6] Alessandro Del Sole. *Xamarin with Visual Studio.* BPB Publications, 2022.

[7] Eric Windmill. *Flutter in Action.* Manning, 2019.

[8] Priyanka Tyagi. *Pragmatic Flutter.* CRC Press, 2021.

[9] https://cordova.apache.org/.

[10] https://ionicframework.com/.

[11] https://nativescript.org/.

[12] https://reactnative.dev/.

[13] https://flutter.dev/.

[14] https://dotnet.microsoft.com/en-us/apps/xamarin.

[15] https://felgo.com/.

[16] https://tau-platform.com/en/products/rhomobile/.

[17] https://www.sencha.com/.

[18] https://framework7.io/.

[19] https://jasonette.com/.

[20] https://www.oracle.com/java/technologies/downloads.

[21] https://developer.android.com/studio.

[22] https://git-scm.com/.

[23] https://cordova.apache.org/docs/en/10.x/guide/platforms/android/index.html.

[24] https://nodejs.org/en/.

[25] https://visualstudio.microsoft.com/downloads/.

[26] https://docs.flutter.dev/get-started/install/windows.

[27] https://blog.logrocket.com/xamarin-vs-flutter/.

[28] https://docs.flutter.dev/get-started/flutter-for/declarative.

[29] Stian Thorgersen and Pedro Igor Silva. *Keycloak - Identity and Access Management for Modern Applications*. Packt Publishing, 2021.

[30] Sunny Kumar Aditya and Vikash Kumar Karn. *Android SQLite Essentials*. Packt Publishing, 2014.

[31] Gene Da Rocha. *Learning SQLite for iOS*. Packt Publishing, 2016.

[32] Shannon Bradshaw. *MongoDB: The Definitive Guide: Powerful and Scalable Data Storage*. O'Reilly Media, 2019.

[33] https://docs.flutter.dev/testing/build-modes.

[34] https://docs.flutter.dev/get-started/install.

[35] https://marathonhandbook.com/average-stride-length/.

[36] https://github.com/ThanosVk/Sodasense.

[37] https://pub.dev/packages/email_validator.

[38] https://pub.dev/packages/connectivity_plus.

[39] https://pub.dev/packages/internet_connection_checker.

[40] https://pub.dev/packages/fluttertoast.

[41] https://developer.android.com/guide/components/services.

[42] https://pub.dev/packages/flutter_background.

[43] https://pub.dev/packages/flutter_background_service.

[44] https://pub.dev/packages/foreground_service.

[45] https://pub.dev/packages/flutter_foreground_plugin.

[46] https://pub.dev/packages/flutter_foreground_service.

[47] https://pub.dev/packages/flutter_foreground_service_plugin.

[48] https://pub.dev/packages/flutter_foreground_task.

[49] https://pub.dev/packages/permission_handler.

[50] https://github.com/nhandrew/platformcode.

[51] https://pub.dev/packages/proximity_sensor.

[52] https://pub.dev/packages/sensors_plus.

[53] https://pub.dev/packages/pedometer.

[54] https://pub.dev/packages/google_maps_flutter.

[55] https://pub.dev/packages/flutter_map.

[56] https://pub.dev/packages/flutter_map_location_marker.

[57] https://pub.dev/packages/location.

[58] https://pub.dev/packages/flutter_map_tappable_polyline.

[59] https://pub.dev/packages/cached_network_image.

[60] https://pub.dev/packages/flutter_cache_manager.

[61] https://pub.dev/packages/sliding_up_panel.

[62] https://pub.dev/packages/geolocator.

[63] https://pub.dev/packages/flutter_compass.

[64] https://pub.dev/packages/provider.

[65] https://pub.dev/packages/objectbox.

[66] https://pub.dev/packages/sqflite.

[67] https://www.keycloak.org/.

[68] https://pub.dev/packages/http.