Instituto Politécnico de Viana do Castelo

ipvc estg

Graduation in Informatics Engineering

Project III Curricular Unit

REAL-TIME LOCATION TRACKING FOR ELECTRICAL BUSES - CMVC

António Marques, n.º 21674

Nelson Campinho, n.º 21751

2020 - 2021

Supervised by: Professor Sara Paiva | sara.paiva@estg.ipvc.pt

Table of contents

Table of figures

1. Introduction

For the course unit of Project III alongside the City Hall of Viana do Castelo, it was proposed the creation of a mobile application, using React Native, that gives real-time information about Viana do Castelo's electrical buses regarding their location and time until they get to the users' current position. This application aims to provide not only an easy to use and easy to understand interface but also provide enough features to assist people who are visually impaired, so that anyone and everyone can use it. For the execution of the project features like buses GPS integration, Google maps and Talkback and Voice Over functionalities are going to be used to achieve the milestones of this project.

The result of this application may assist with the reduction of the CO2 emissions in the city since it can attract the attention of other public transportation clients.

2. Objectives

These electrical buses have the particularity of stopping when asked regardless of if the user is at a stop, an element that is important for this application. When the user opens the app, it calculates the nearest point from the user's current position to the buses' route and outputs how long until the user gets to that point. The application also gives information in real-time about the buses' position and calculates the distance and estimated arrival time to the point previously mentioned. The combination of each information allows users to get an estimate about how long for them to get to a place where they can catch a bus and how long that bus takes to reach that spot. Related to the bus, the application uses physics to understand its situation, that is, the application constantly monitors the speed of the buses to evaluate if it's are moving normally or if there are some impediments – like transit or accidents – that can influence its arrival time. If the bus is stopped for some time, the user is notified about that and there will not be any estimated arrival time to give to the user, until the bus resumes its movement.

For usability matters, the application must have a notification activated when the users' selected a bus so that the user can constantly get feedback about the current selected bus position and arrival time without having necessity of unlock their phones and open the application.

Considering visually impaired people, the application not only has to provide Talkback and Voice Over functionalities (so that the application can be used in both IOS and Android) so that it's possible for them to fully interact with the application but it also requires to have access to the coordinates of fixed stops dedicated to visually impaired people.

With the incorporation of this application on Viana do Castelo it is possible to promote the usage of the electrical buses that exist on the city whilst contributing for the reduction of CO2 emissions on the city. Simultaneously the app can assist visually impaired people living their day-to-day lives making it easy to catch public transportations.

2.1. Use Cases

Using the objectives described on chapter 2 it is possible to create a list of use cases has shown on the following table:

Use Case	Application	System	Description
Getting GPS information for each bus		X	The application can get the operational buses position, in real-time, to give a perception to the user about their positioning.
Talkback/Voice over functionalities		X	The system outputs various types of information for those who are impaired visually.
Information about fixed stops position	X		For those who present visual impairment the application contains information about fixed stops, so it is easier for them to catch the bus
Show bus route	X		By default, the application shows on the map the route the bus makes.
Allow bus selection	X		The user can select a bus icon, that moves constantly on the map simulating the real bus position and can easily track that specific bus. The application outputs a warning when the bus is approaching the specified location.
Calculate bus distance and estimate arrival time		X	The application can calculate the distance and estimate an arrival time using the bus real-time positioning tracking and the user's selected route place.
Allow easy access to selected bus information		X	So that the user does not have to unlock the phone and open the application each time the user wants to see where the selected bus is, the application gives a notification to the user that updates the information about the bus position and estimated time of arrival.

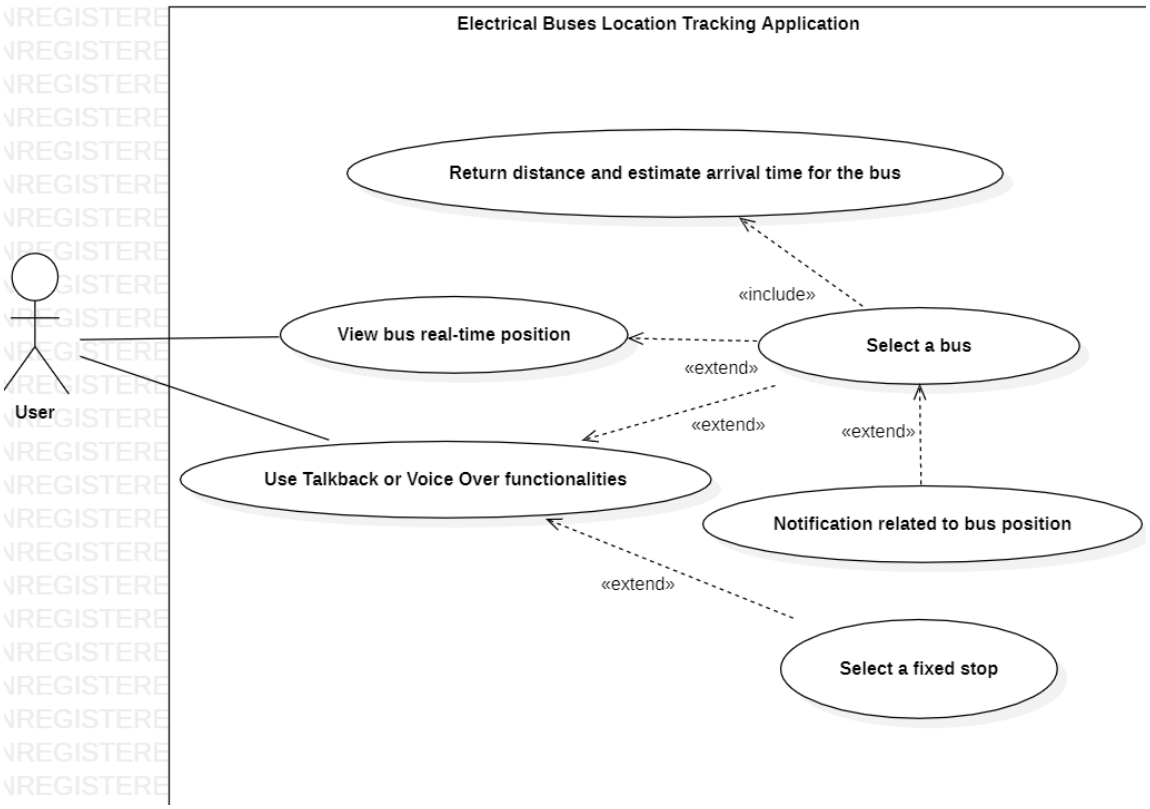


Figure 1 - User Use Cases

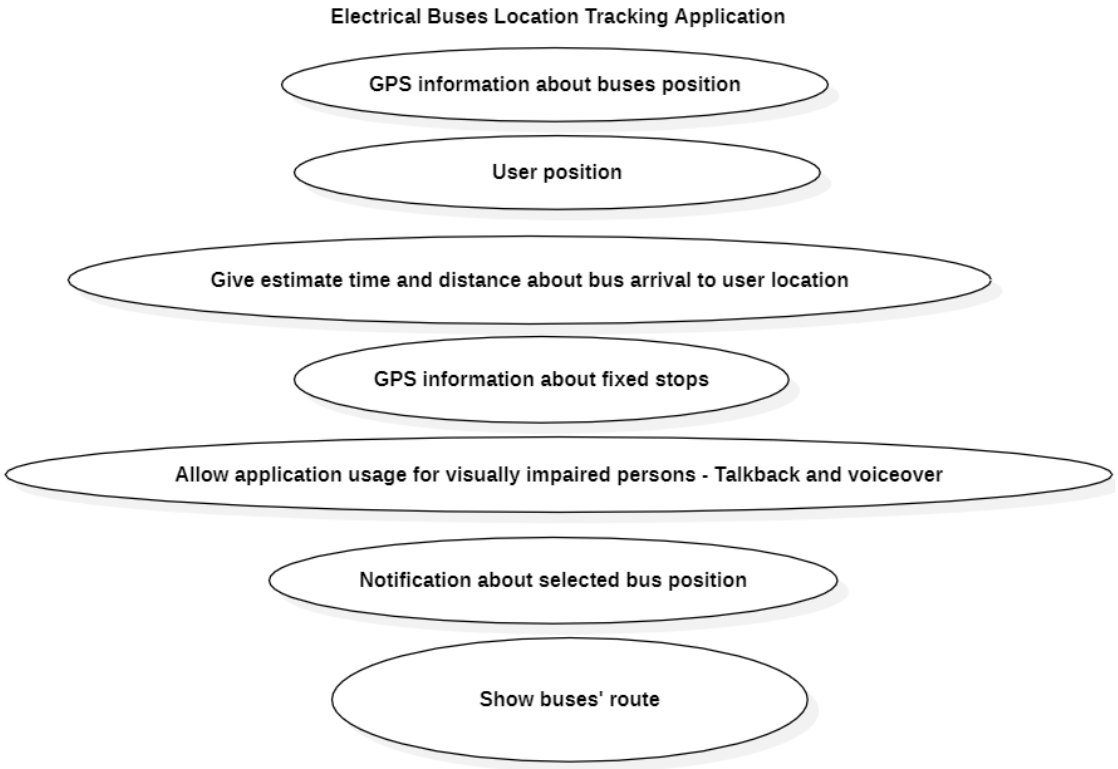


Figure 2 - System Use Cases

3. State of the art

Global app downloads by quarter (Sensor Tower), billions

	Google Play Store	iOS App Store	Overall
Q1 2015	10.4	5.4	15.8
Q2 2015	11.3	5.2	16.5
Q3 2015	10.2	5.5	15.7
Q4 2015	10.5	5.9	16.4
Q1 2016	11.1	6.1	17.2
Q2 2016	12.6	6.3	18.9
Q3 2016	14	6.6	20.6
Q4 2016	15.2	6.5	21.7
Q1 2017	16.7	6.9	23.6
Q2 2017	16.1	6.5	22.6
Q3 2017	17.1	7.3	24.4
Q4 2017	17.4	7.1	24.5
Q1 2018	17.4	7.7	25.1
Q2 2018	18.6	7.2	25.8
Q3 2018	19.4	7.6	27
Q4 2018	20.1	7.2	27.3
Q1 2019	20.7	7.4	28.1
Q2 2019	21.3	7.4	28.7
Q3 2019	21.6	8	29.6
Q4 2019	20.9	7.8	28.7
Q1 2020	24.3	9.3	33.6
Q2 2020	28.7	9.1	37.8

Source: Sensor Tower

Figure 3 - Key App Download Statistics

Source: https://www.businessofapps.com/data/app-statistics/

Mobile applications have been around for years and its usage has been increasing due to their availability, convenience. Mobile phones nowadays have the possibility of process huge amounts of data allowing the construction of apps heavier apps with more integration without losing efficiency.

There are apps for everything ranging from gaming and educational categories to parenting, fitting the needs of any type of user that enjoys exploring and incorporating mobile apps on their daily routines **(IQBAL, 2020)**.

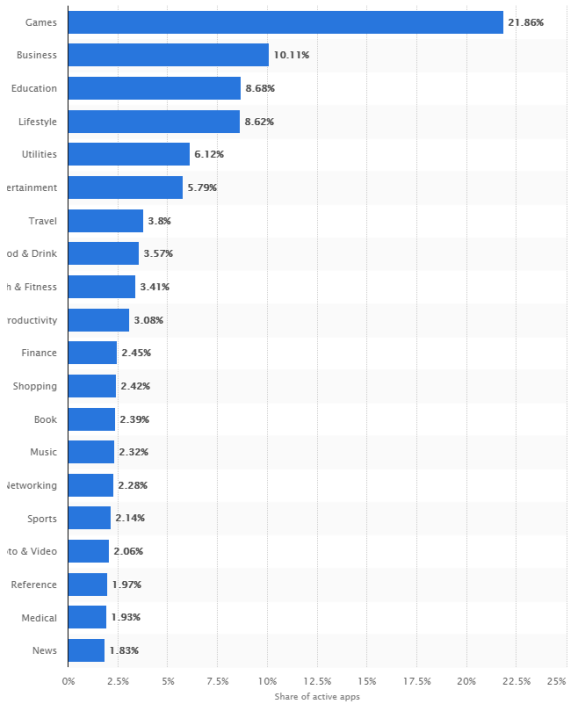


Figure 4 - Apple most popular app store categories until August of 2020

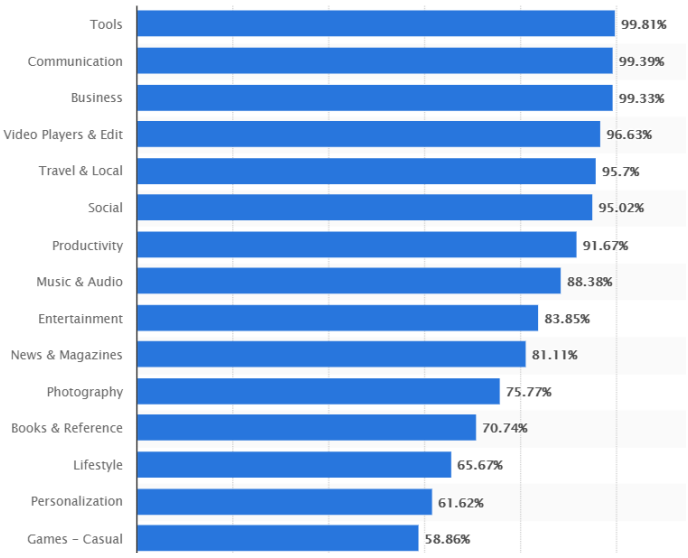


Figure 5 - Google most popular app store categories as of third quarter of 2020

Mobile apps for transportation have an important and informative role on the life of users that use those types of apps and according to Statista this category is well placed in both Apple App Store (Figure 4) **(Clement, Most popular Apple App Store categories in August 2020, by share of available apps, 2020)** and Google App Store (Figure 5) **(Clement, 2020)**:

These types of apps allow users to easily have access to transportation or schedules if the app dedicates to public transportation.

On the market exists apps like Bolt and Uber that allows users to select a destination and automatically is assigned to them a driver that drives to the users' current location and takes them to the destination. The same app lets users perform the respective payment, due to integration with bank services, which not only is useful for the user because it does not require them to have physical money but also for the company where the payments of the service are assured. But there are also apps for transportation dedicated to informing the users about schedules and traffic problems. These apps are dedicated to public transportation information.

Existing applications for public transportation like Citymapper, Moovit integrate information from different transportation methods and company to proportionate multiple options to users so they can choose which one fits better to their needs. They integrate the available information for each public transportation company that adheres to the apps functionalities and objectives and constantly updates the information according to possible obstacles or changes that may occur.

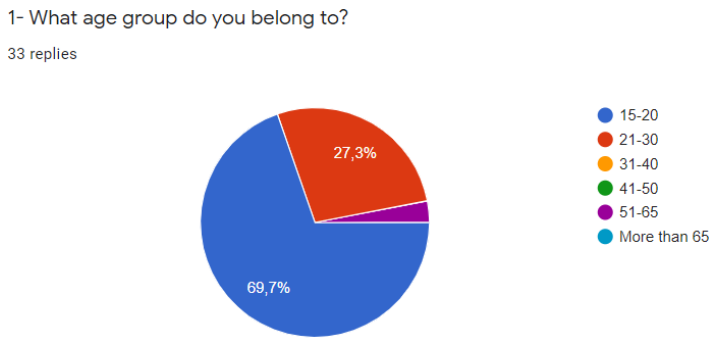
Most of these apps require constant usage of the GPS functionality (constant satellite communication) of the cellphone so that it is possible to obtain not only the user's current position but also position information about the transportation method that the user intends to use (that also needs to communicate with the satellite). The communication between GPS and satellite needs to be as precise as possible to give correct information to the users.

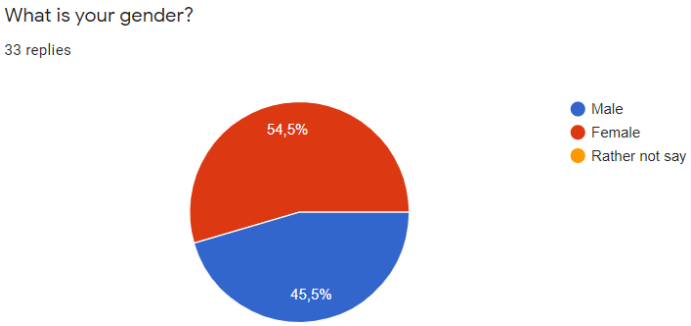
4. Collected Statistics

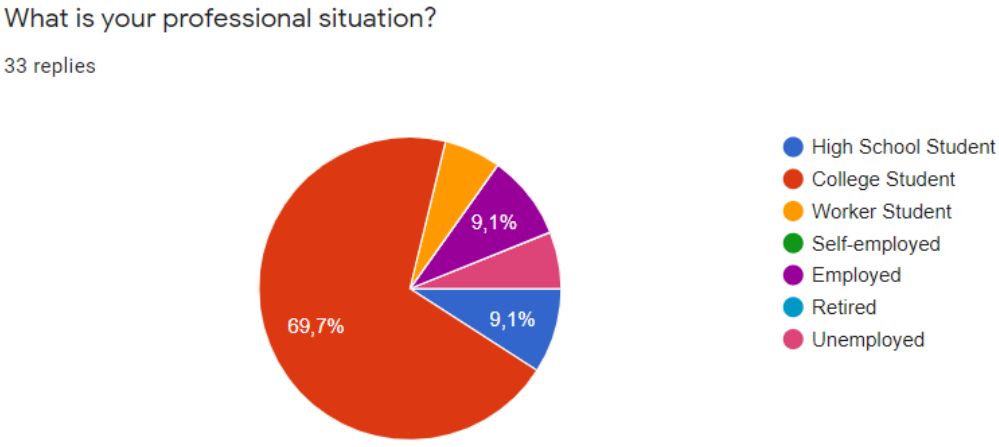
4.1. People interaction with public transportation

To get a better perception about how the community of Viana do Castelo, that represents the target audience for the application being developed, fit public transportations on their routines. It was created a Google Form questionnaire, provided to the academic community from School of Business and Technology - Polytechnic Institute of Viana do Castelo (Escola Superior de Gestão e Tecnologia do Instituto Politécnico de Viana do Castelo - ESTG - IPVC) to get a better perception about how regularly they use public transportations, which transportation methods are used by the community to circulate in the city, and if they have knowledge about the existence of electrical buses on the historic center of the city. It was received 33 (thirty-three) responses overall, which is enough to provide good statistics conclusions.

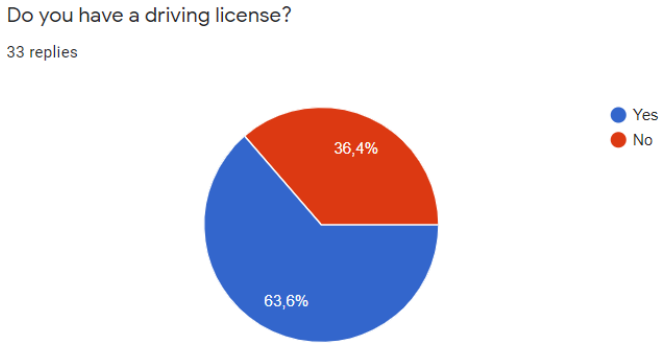
One of the characteristics of the surveyed people is that they are mostly between the ages of 15 and 20, mostly college students and mostly have a driver's license.

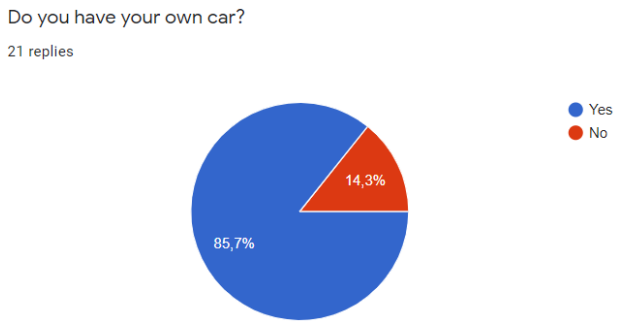




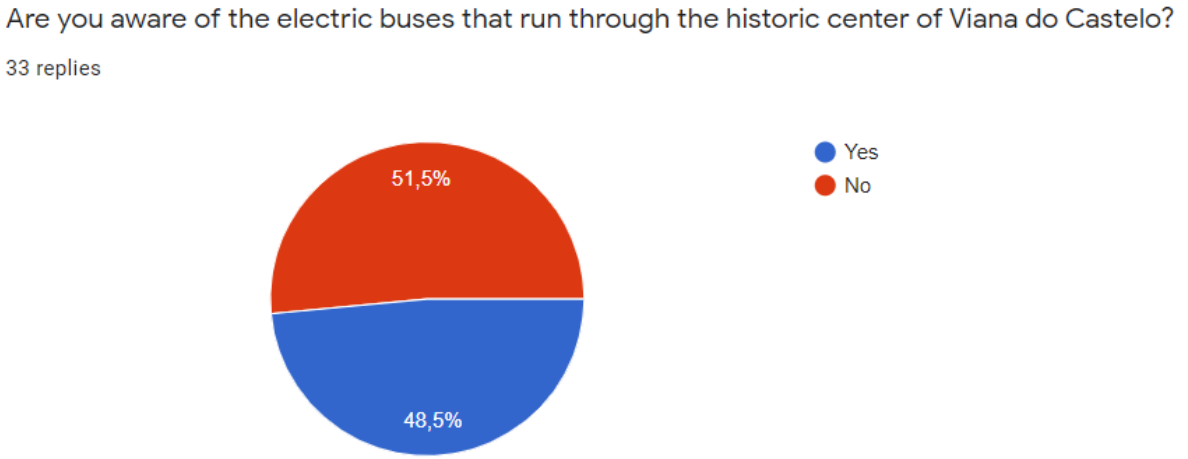


Most of those who answered that possess a driver's license also possess their own vehicle, having greater chances of not needing to use public transportations.

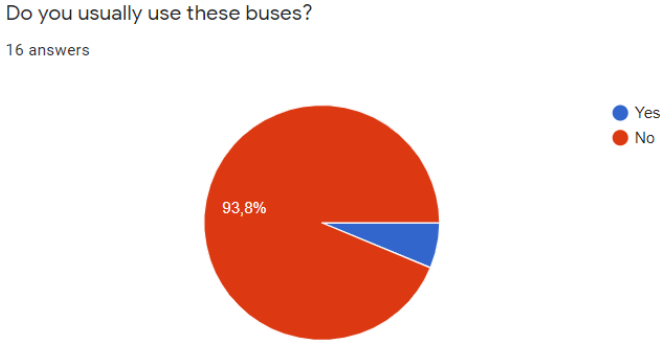


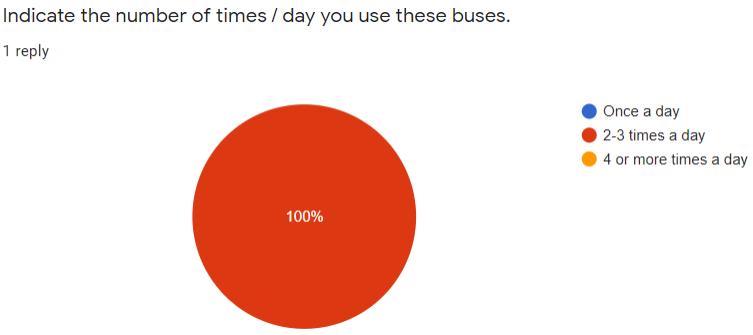


Unfortunately, when asked about if they have knowledge about the electrical buses that circulate on the historic center of the city, most answered that they have no knowledge.

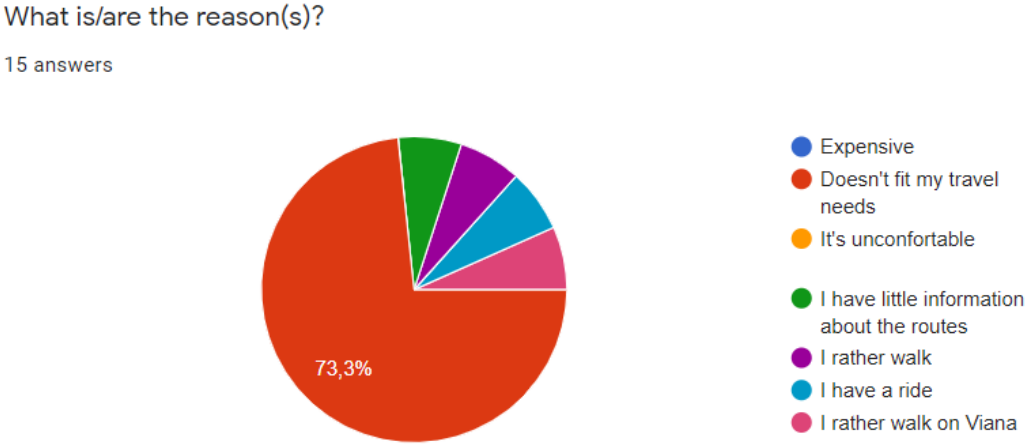


For the 48,5% of the people that know about the electrical buses all of them use this public transportation method more than once a day.

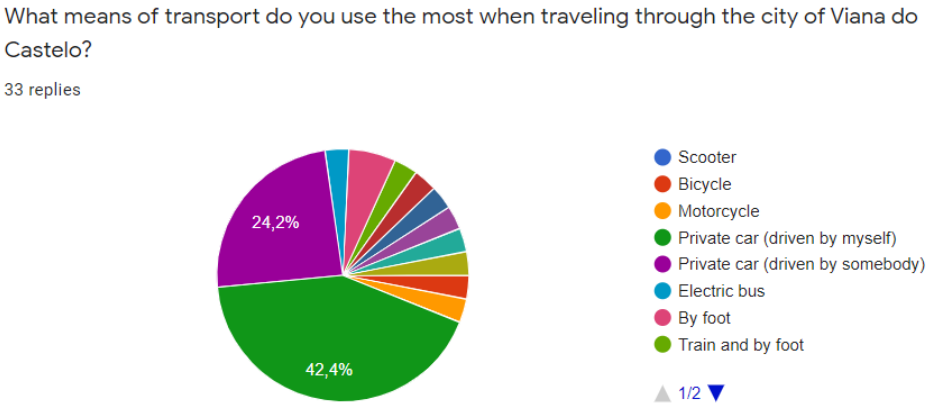


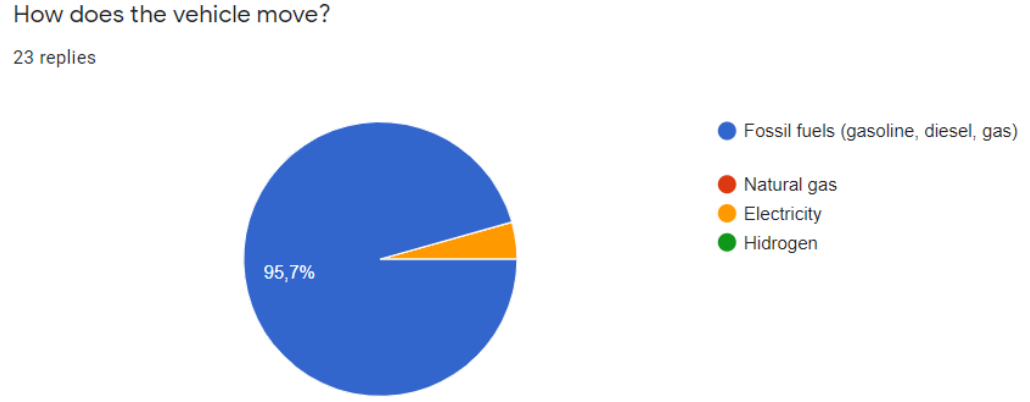


For the 51.5% of the people that do not use this type of public transportation the reason most described to justify the answer was that the route that these electrical buses perform every day, that is fixed, does not match to the needs of their daily lives, while others claim that they are not acquainted about the route of these buses.



The preferred transportation method referred are vehicles that required fossil fuel to move, while a small percentage prefers to walk or use bicycles to move around the city.





4.1.1. Conclusions from the survey

With the answers obtained on the survey, it is possible to verify that approximately half of the answer point to the fact of people not knowing about the existence of Viana do Castelo's electrical buses, but for those who do know about its existence but still do not use this public transportation is because the route does not match with the public's need. The lack of information and variety of routes lead to people to search for other ways to move around the city. The amount of surveyed people that use a vehicle that consumes fossil full is almost 100%, meaning that a bet on the promotion of electrical buses can greatly impact the ecological footprint of the city to the planet.

4.2. Ecological footprint

Of the thirty-three people surveyed, twenty-two use a car, which consumes fossil fuels as energy source, to move around the city. In this chapter is going to be made a small assumption about how much CO2 emissions can be reduced if these twenty-two people would start using Viana's electrical buses as a transportation method.

According to a study made in 2018 by **(Transport & Environment, 2018)**, CO2 emissions by car are around 170g/km, and if value still stands in the year 2020 it's going to be the base for this analysis. On figure 6, it is represented the CO2 emissions on Europe by each industry being noticeable that the most contribution comes from the transportation industry (considering not only public but also private transportations). By its impact on the emission of CO2 should represent a focus of investment to promote the usage of less vehicles that run using fossil fuels.

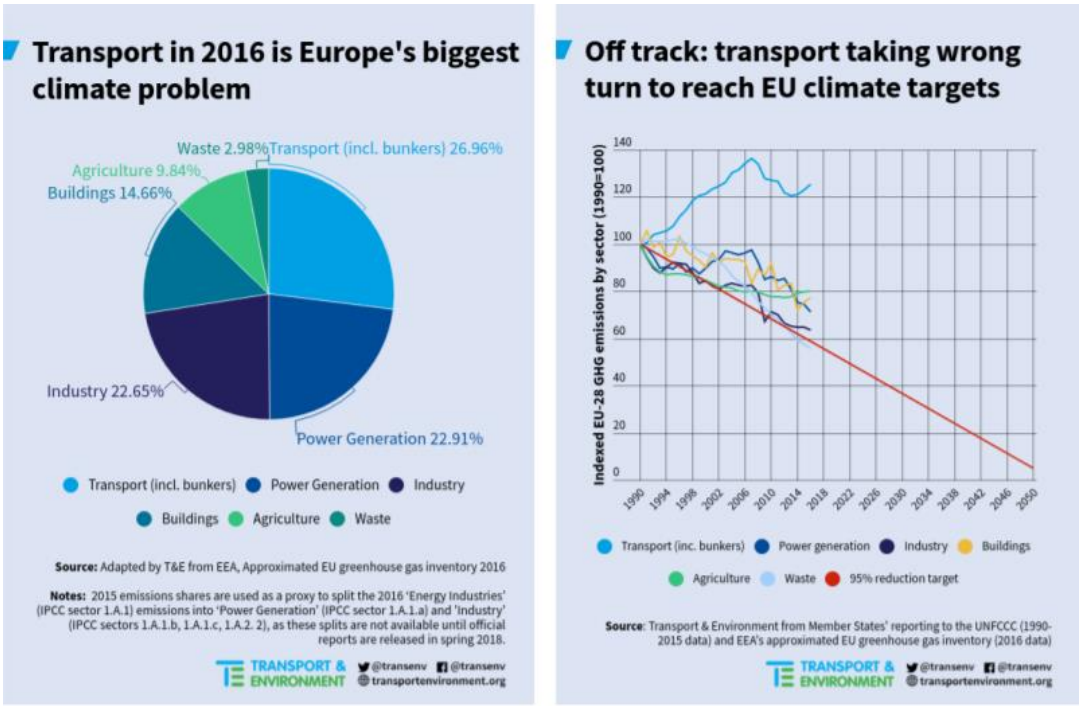
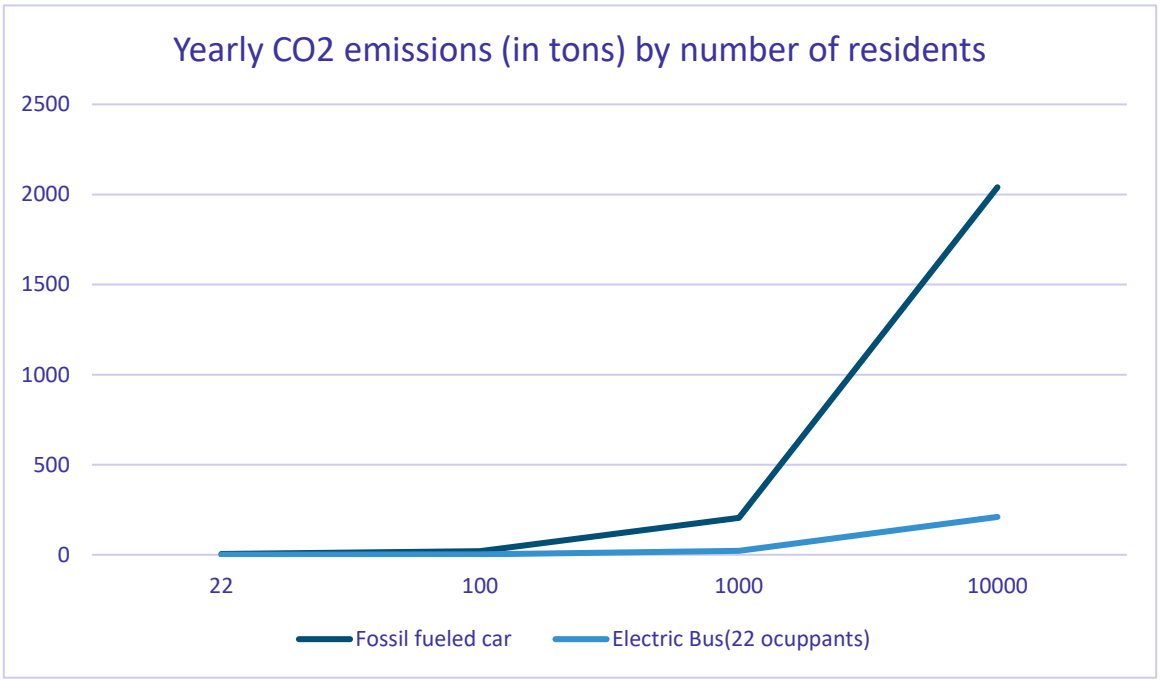


Figure 6 – 2016 climate impact in Europe by each industy

By considering an average of 5km travel by working day for each resident of Viana do Castelo, to perform an estimate of how much it is possible to reduce CO2 emissions by using Viana do Castelo's electrical buses, we can conclude that by each day of the week, a driving resident contributes with 850g of CO2 for the atmosphere, 4250g each week, 17 000 each month and 204 000g a year.

With 204 000g of CO2 emissions per person, and considering the values previously mencioned, represents a total of 4 488 000g by year from the twenty-two surveyed that use a fossil fueled vehicle to move around.

The same line of thought can be applied to electrical buses. According to the study from **(Spector, 2018)** by mile each electrical bus contributes with 1072g of CO2 emissions that correspond to approximately 1736g per kilometer. That represents a total of 416 640g of CO2 emissions (considering that the bus only performs 5km daily like the residents). Viana do Castelo's has a capacity of 22 occupants, the same number of people that were surveyed that drive a fossil fuel vehicle. It is possible to conclude that if those twenty-two people would take these electric buses instead of their own private vehicle the yearly CO2 gas emissions would go down from 4 488 000g to 416 640g, a huge difference in value.



By analyzing the graph, it is possible to see that the reduction on CO2 emissions increases in significance the more people switch from fossil fueled vehicles to electric public transportations. Thus, it is important to keep promoting, investing and encourage the usage of these types of planet friendly transportations to leave a big ecological footprint for the future.

5. Architecture of the solution

For the application to get the position of the buses of the city it is necessary that the application makes a request to a tracking server about the electrical buses of Viana do Castelo. This tracking server that constantly updates the position of these buses retrieves the information required by the application. There is a constant communication between the satellite that retrieves the position (latitude and longitude) of given bus of which the latter sends that information to the tracking server via GSM/GPRS communication, allowing the tracking server to constantly have an updated version of the bus positions.

At the same time this request happen the mobile device also performs a request to the satellite in order to obtain the current position of the user. This information alongside the position of the buses allows the application to perform calculations about distance and estimate time.

To further assist on these calculations, the app integrates with an API that returns information about traffic that can be significant enough to cause changes on the calculations.

The mobile app also integrates a Google Map API so it is possible to obtain a map where the information about buses, user location and bus routes (information got from a SQLite database) are going to be displayed.

Finally, for visually impaired users to be able to use the app, there is going to be user an API that integrates Talkback and Voice Over functionalities.

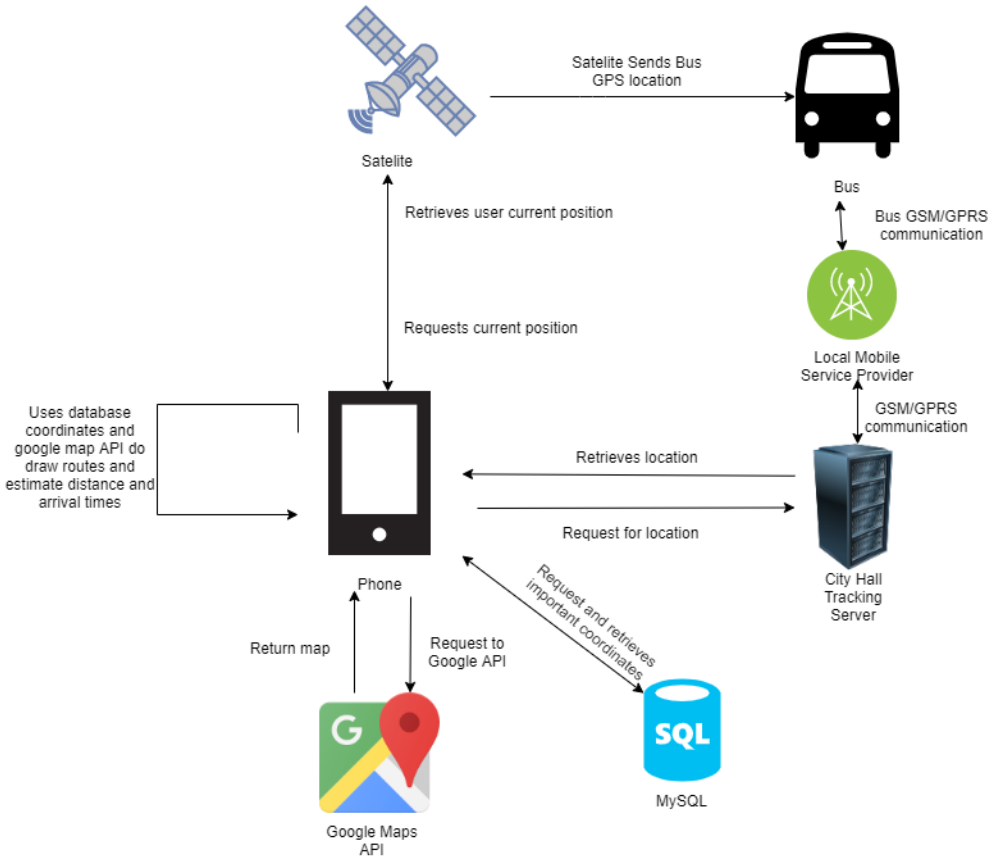


Figure 7 - Architecture of the solution

6. Mockups

To give a scope about how the application is going to be displayed to the users, and show the different screen that they can access, some mockups where created. These mockups work as a draft and intent to show, not only the information that is going to be displayed to the user when the application run, but also the different kinds of buttons available and their functionalities.

In the figure 7 is shown the first screen that users are going to have access to.

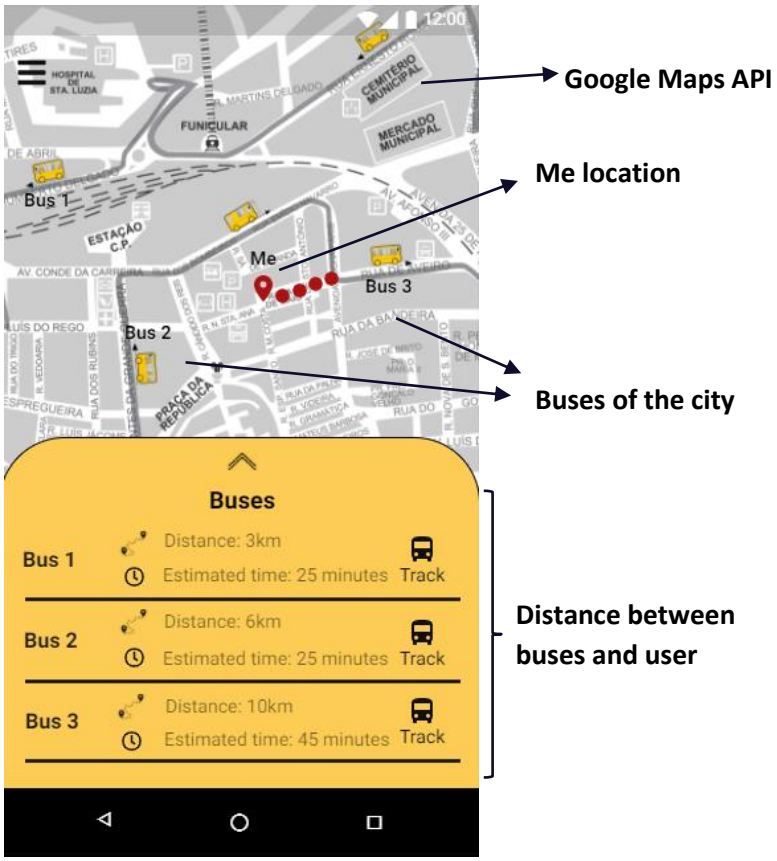


Figure 8 - Confirmation Dialog

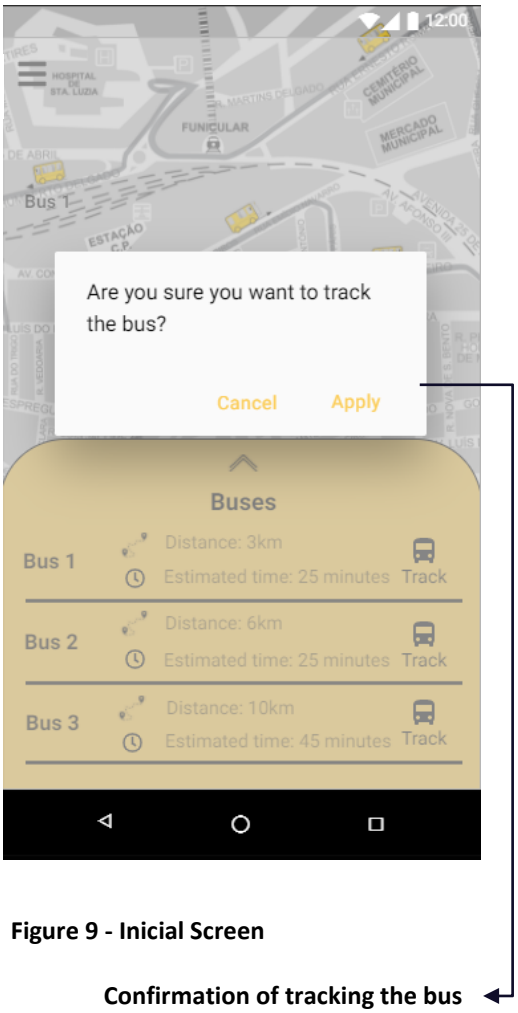
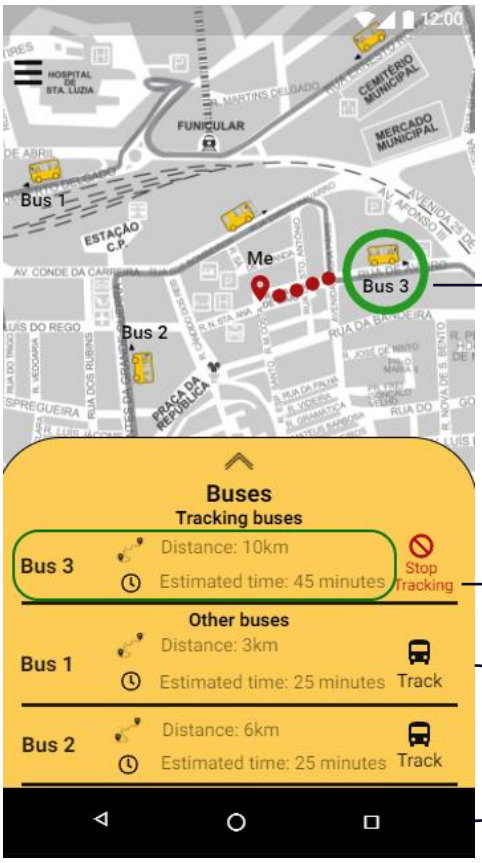


Figure 9 - Inicial Screen



Bus with tracking activated

Selected bus tracking. User can stop tracking by pressing "Stop tracking" button

The remaining buses information. Clicking on "Track" button will switch the bus that is being tracked.

Figure 10 - Initial screen with route place selected.

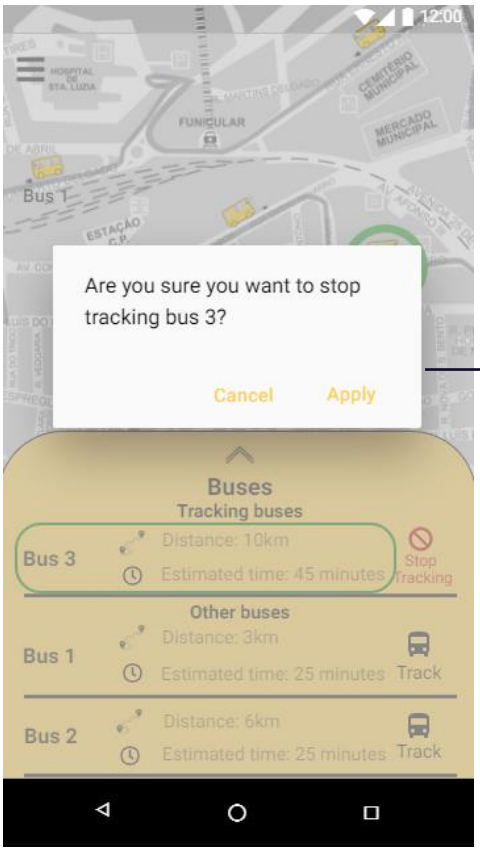
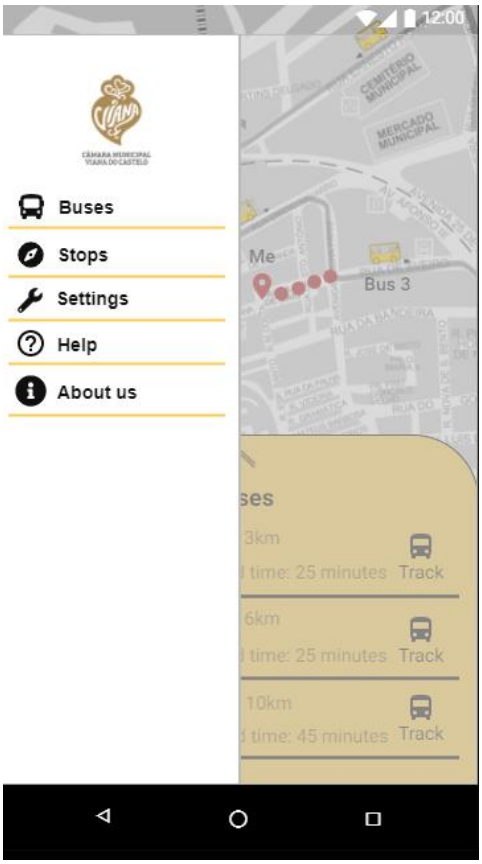


Figure 11 - Dialog stop confirmation.

Cancel confirmation of tracking the bus

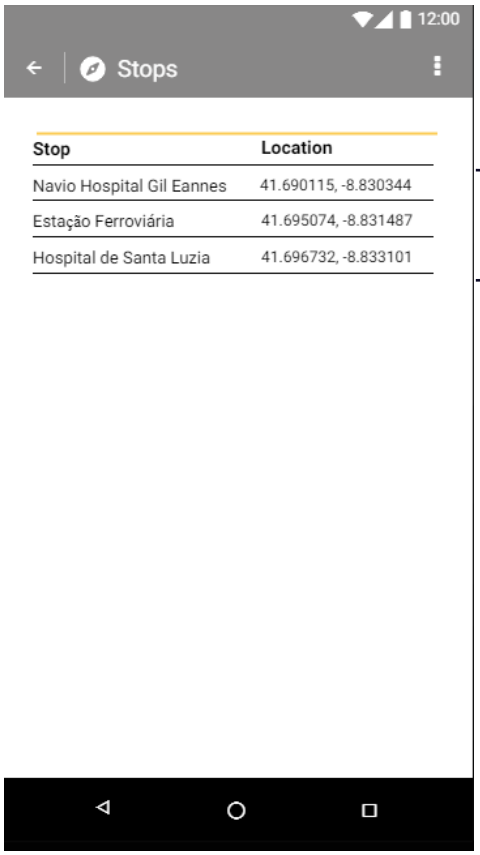
When the user locks the phone, a notification remains so that the user can see the information without having to open the application again (application runs on the background).

Figure 12 - Notification Screen



Drawer Navigation

Figure 13 - Drawer Navigation Screen



Stops of buses on table format

Figure 14 - Stops Screen

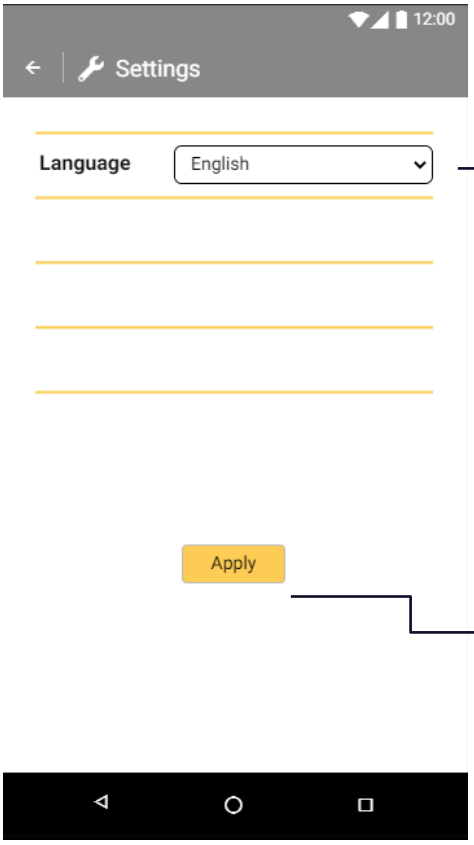


Figure 15 - Settings Screen

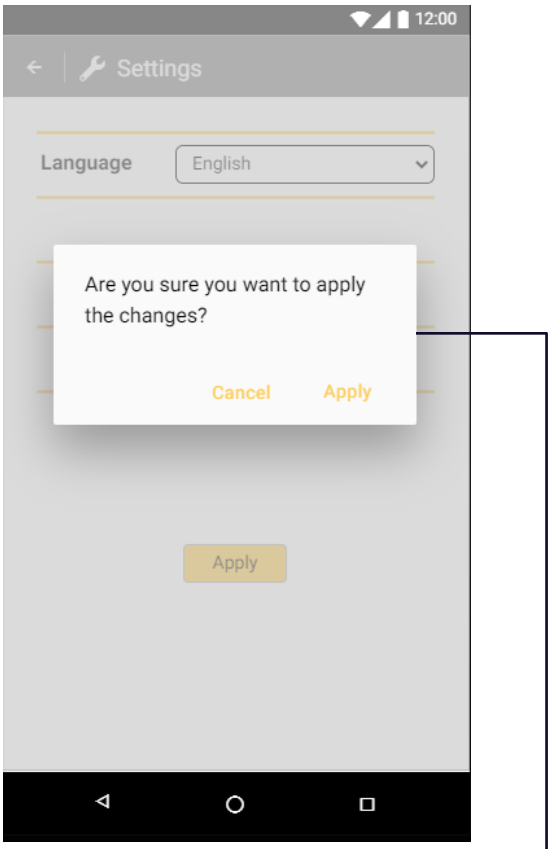


Figure 16 - Confirmation dialog

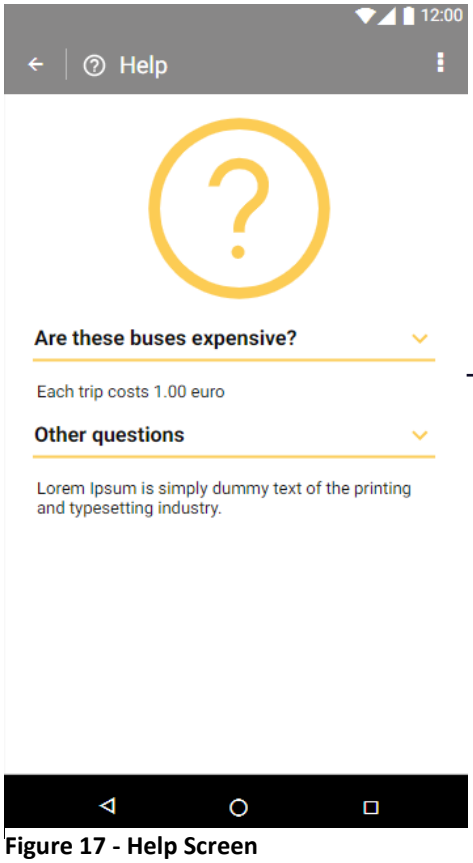
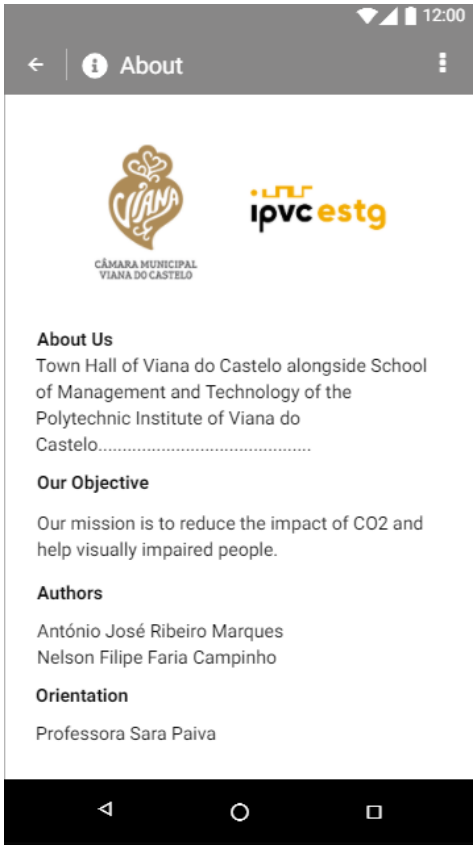


Figure 17 - Help Screen



Contains information about the application, creators, and objectives of the app.

Figure 18 - About screen

7. Project Management, Technologies, and Database

To better organize the workflow of this project it was required some planning regarding which tasks should be executed, by who and how long should which one of those tasks should take until they must be concluded. It was used Trello **(Trello, 2020)**, a collaboration tool that assists on project planning, allowing the users to write tasks, apply tags and due dates and even allows different users to share files and communicate with one another. Using this tool assisted on using an agile development method throughout the creation of the project, and thus being prepared for eventual changes on the workflow without compromising schedules and milestones.

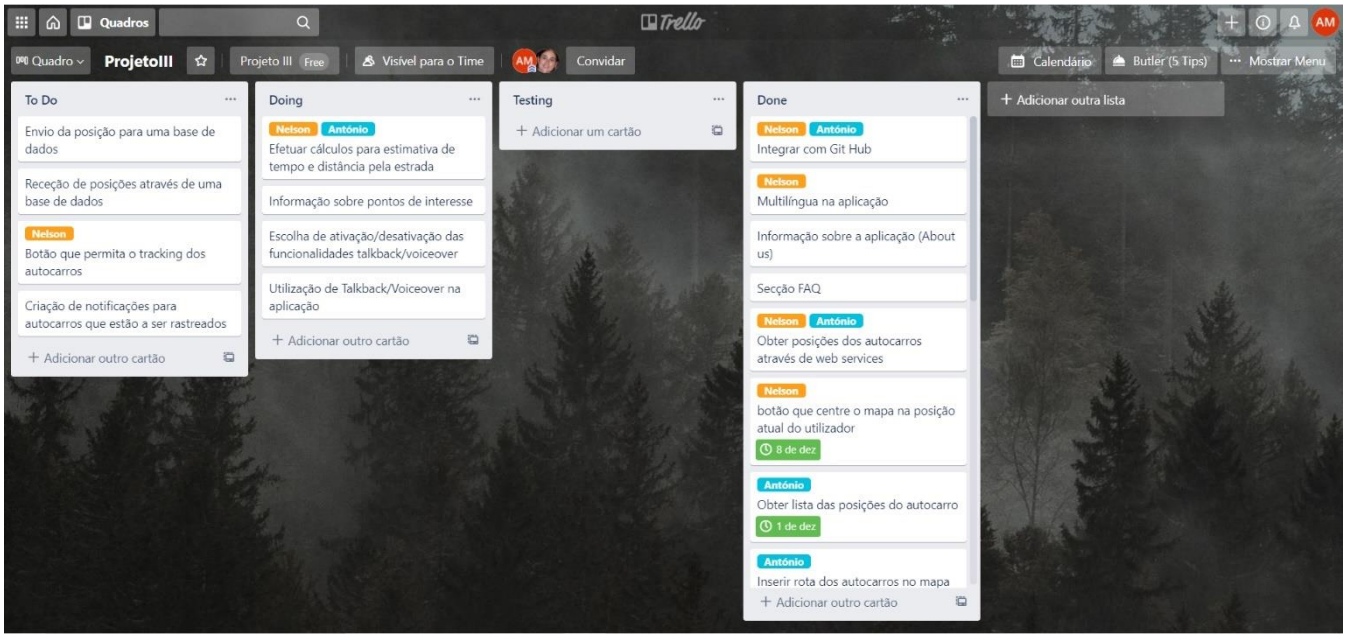


Figure 19 – Trello Overview

With schedules and task defined the implementation of the project itself can begin. All the technologies involved, previous and after planning, will be described on the following subchapter.

7.1. Technologies

7.1.1. StarUML

StarUML is a modeler software that supports the creation of diagrams like use cases diagrams, ER diagrams and much more. It supports teams in agile and concise modeling enabling easy updates to diagrams when necessary **(Niklauslee, 2020)**. This software was used to create the use cases diagrams for this project, seen in subchapter 2.1.

7.1.2. MockingBot

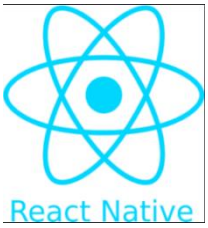
MockingBot **(MockingBot, 2020)** is a software that allows the creation of dinamic mockup with simple examplificative functionalities. These mockups helps figure out what features should be created and the GUI's displaying to the user **(Mockup, 2020)**. They are a prototype of the final product that works as both an example to present to the client and as a guideline for the developers. The way the mockups are structured can suffer changed throughout development.

7.1.3. Visual Studio Code

Visual Studio Code is a souce code editor that, by default, supports JavaScript, TypeScript and Node.js but has extensions for other languages **(Microsoft, 2020)**.

7.1.4. React Native

React Native **(Facebook, React Native, 2020)** is a Javascript framework that allows to natively create mobile applications for both Android and iOS. Some features of web development are visible on React Native, like JavaScript, HTML tags and CSS **(Oreilly, 2020)**.



7.1.5. NPM

Is a Software Registry that allows the installation of new packages into our developing environment **(W3Schools, 2020)**.

7.1.6. MySQL

MySQL is a database service to store and manage, locally or not, data in data structures. These tables can be related to one another or they can simply exist by themselves. These data structures can store any type of data, from strings and number to files and images **(Oracle, 2020)**.

7.1.7. PHP

This scripting language is server oriented and can be embedded into HTML and thus is possible to apply it to the client-side. With this language can be created server that return information to clients' requests **(Group, 2020)**.

7.1.8. 000WebHost

This website provides a hosting service. It allows to import databases and webservices so that they go from only being accessible locally to being accessible to everyone **(000webhost, 2020)**.

7.1.9. Slim

Slim is a PHP micro framework that assists on the creation is web application and API's. It's responsible to receive requests, invoke the correct callback and return the response **(Josh Lockhart, 2020)**.

7.1.10. NotORM

It's a PHP library that provides a simple way of working with data contained on a database. It treats each database as an object and each table of that database as methods **(Vrána, 2010)**.

NotORM

7.1.11. Postman

Postman is a collaboration platform for API development. It assists on API development and testing, creating an environment where users can test their API. Postman can identify if the request requires headers or not, allows simple filling of those same headers, and provides detailed information about retrieved data (case request was successful) or error generated (case request unsuccessful).

7.1.12. GitHub

GitHub is a software that allows versioning control via graphic user interface. Github uses Git, the open-source version control system, to manage versions. With GitHub it's possible to track each updated version and what changed between versions. It also allows to public submit code and manage contributors and suggestions **(GitHub, 2020)**.

7.1.13. Google Maps API

This API provides multiple resources to build a map, getting directions, drawing routes and calculate distances **(Google, 2020)**.

7.2. Data Model

Data had to be collected in order to progress with the development of this project. Summarizing the application, the biggest focus of the app is a user interacting with a map in order to access multiple functionalities. To do so the database stores information about places descriptions along side with latitude and longitude of multiple interest points and other places that were relevant to development. Even though MySQL is a relational database, a database whose focus is to store data in tables that relate to one another, the geospacial information collected have no relation between tables meaning that each table that makes up the database have no connections to one another.

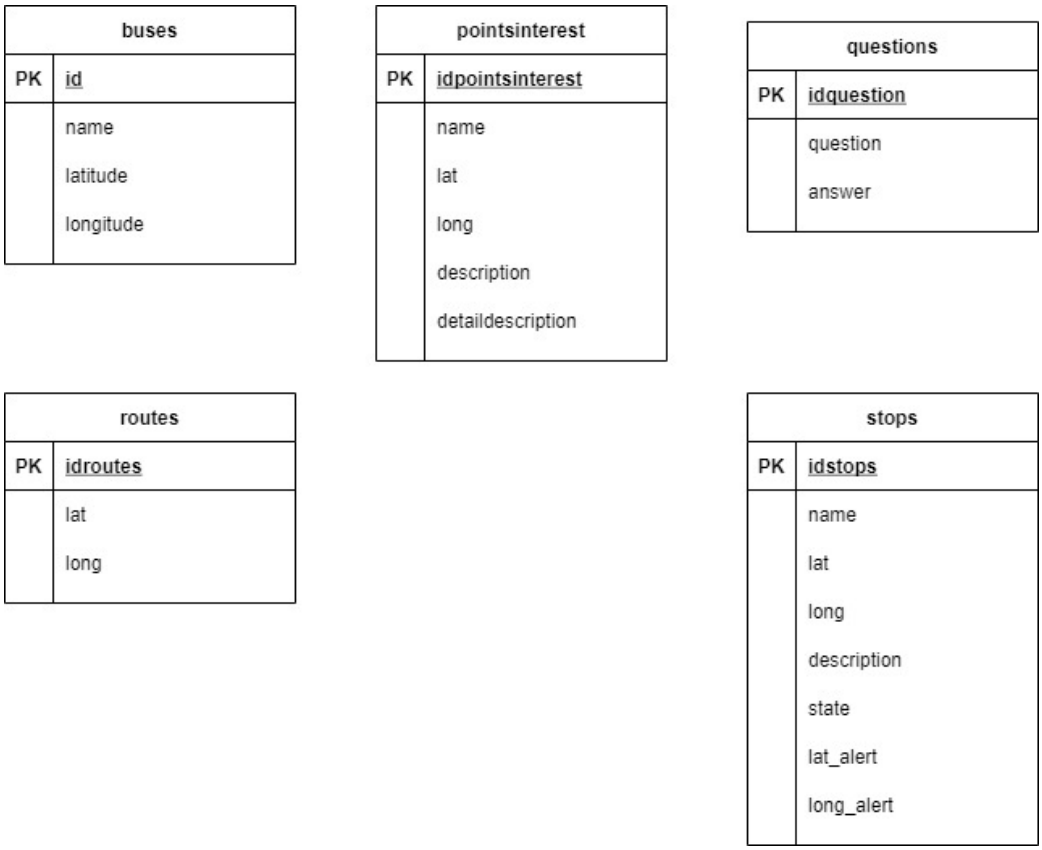
City Hall provided information about points of interest spread accross Viana do Castelo that manly will be used to give Talkback/VoiceOver feedback to visually impaired persons in order to notify them where they are while riding the bus. In the first steps of development, in order to test buses moving on the map, functionalities like a user taking a bus and calculate distances and estimate times, was created tables which served the porposes of creating the route of the buses, simulate buses circulations and better indicate which paths the user should use to get to the route.

7.2.1. Database Tables

The tables used to store relevant data for the creation of this project were:

- Buses – To simulate signals being transmitted and updated, simulating a real-life bus moving in real time while the application is running.
- Pointsinteress – Stores information about points of interests like its position and description. Used to update the application bottom sheet when the user gets inside a bus. With Talkback/Voiceover functionalities the database description for each element of the table is used to give feedback to visually impaired persons.
- Questions – This table stores questions, and respective answers, for the help section of the application, with information that may answer some frequent doubts the user may have.
- Routes – Stores multiple latitude and longitude values that are necessary to trace the buses' route.
- Stops – This table stores information about fixed stops for visually impaired users. This table contains information about the name and a description that helps these users to better situate the stop.

These tables are independent of one another, their focus in to provide data to build the app, and all of these data is later connected when the application is compiled.



8. Programing Environment and Developed Features

The project was developed using React Native, so it was possible to build natively and simultaneously for Android and iOS. Using NPM some external packages needed to be install like Google Maps that allows the application to incorporate a map from google and some other features related to it (example: Polylines, Markers, ...) **(Google, 2020)**, Google Maps Directions, that allows to trace a route using the maps path while simultaneously calculate the distance and required time to get from the origin to the destination **(Bramus, 2020)** and Geolib that allows to perform geospacial calculations, not only related to distances but also with object velocity and polyline length **(Bieh, 2020)**.

This packages alongside the collected data makes most of the application. The data, stored in a MySQL database, needed to be accessed by the application remotely, via APIs, so that its usage during development could be used without being connected to the local machine. To achieve that it was used a web host service from 000webhost **(000webhost, 2020)** which objective was to store on a cloud the database in use with all the necessary data and all created APIs. With the database on a cloud the application could perform requests (via fetch) to the database to retrieve the data. The APIs for simplicity, that are consumed via fetch where created using PHP language and both Slim framework (assists on API creation) **(Josh Lockhart, 2020)** and NotORM (simplified database quering) **(Vrána, 2010)**.

Combining these technologies it was possible to create the following features, described on subchapter 7.1, to match the use cases defined on chapter 2.

8.1. Hooks Usage

React Native's Hooks play an important role on maintaining the application's integraty. Hooks **(Facebook, 2020)** allow creating applications without the use of classes and allows users to 'hook into' React's states and lifecycles. There are different types of Hooks but on this project two where used: useState and useEffect.

useState Hook **(Facebook, 2020)** allows variable declaration. This hook creates the variable to be used and a function to set the value of the variable. useStates allow the content displayed on a component to dinamically change according to updates to these variables, meaning that any changes that occur to a useState variable is instantly updated to the user.

While useStates are related with variables, useEffects **(Facebook, 2020)** are related to the application's component lifecycle. This hook is used to perform actions when the component is mounted and when when any update occurs to the component. Since this project uses latitude and longitude positioning where any update to positions alters other variables in the project, useEffect were used. Each time, for example, a bus position would change, a certain amount of function would be performed.

8.2. Application Features

By default, the application provides the user a map from googles, the buses' routes (created using multiple coordinates and polylines) and all active buses position. With these core features all the others were created. All source code for the features described can be found on chapter 8.

8.2.1. Drawer Navigation

Drawer navigation allows the creation of menus with items that changes screens with different intentions. Each time the user selects an item from this menu the screens changes displaying other types of information. The application gives the following menu items:

- Buses: Refers to the first screen the user as access too. Displays the map and circulating buses, alongside the functionalities of calculating distances and estimate arrival time for a selected bus.
- Stops: List about fixes stops that also correspond to points of interest.
- Schedules: List of different buses' companies' schedules.
- About: Information about the application, who created it and why.
- Help: List of frequently asked questions that can answer to most of the user questions about the electrical buses or application usage.

8.2.2. Select a bus

The user can select one of the moving buses icon on the map, that represent an active in circulation. The selected bus changes colors to give a visual feedback to the user about which one was selected.

8.2.3. Bus route

At launch, the application fetches from the database all coordinates that make up these buses' route.

8.2.4. Distance and estimate time calculation

When a bus is selected a bottom sheet slides up, so the user has access to distance and estimate arrival time of that same bus. The distance is calculated from the point of the route where the bus is situated until the closest point from the user to the route. The distance is calculated by measuring the length of the polyline between those two points, using getPathLength method from geolib package. The path from bus to user is highlighted as green.

The time calculation uses the previous selected bus position and the current position to check at what speed it currently is and then uses that speed to estimate the arrival time using physics' formulas.

For the velocity: The **distance** represents the distance from the previous point to the current one and **time** represents how long the application waits until it makes a new request.
velocity = distance/time

For the estimated arrival time: The **distance** is the result from getPathLength and the **velocity** calculated as mentioned previously.
time = distance / velocity

Each time a new request is made to the database requiring buses' positions the values are updated.

8.2.5. Getting in and out of the bus

On the bottomsheet exists a button, which initial text is "Get into the bus". This button changes the information displayed to the user. When the user is outside of the bus, the relevant information to be shown is about the distance and estimate arrival time of a selected bus. When the user gets inside the bus this type of information is irrelevant. When the bus encounters at a certain distance of the user, this button when pressed triggers a change to the information displayed on the bottom sheet.

The text changes into 'Leaving the bus's and the user has access to the current point of interest (that the bus is going through), the previous point of interest and the next. These points of interest exist to inform the user what is the following place that it will be encountered from the buses' route and is used alongside the accessibility functionality of each mobile operating system.

8.2.6. User Location and User Location Button

For some functionalities of the application to work users' location is also a relevant factor. User position if displayed by a blue marker (default from React Native Google Maps API) that constantly changes position while the user is moving. All changes to an user position is tracked.

8.3. React Native Accessibility

Accessibility refers to the assistance that can be provided to people that would normally struggle because they present some health-related diagnosis. React Native's Accessibility, **(Facebook, Accessibily, 2020)** corresponds to a set of options/features that can be implemented natively into the code and assisting on the making of applications for both Android and iOS that can assist users that could have more difficulties using these apps. The way that React's accessibility works is by allowing the implementation of attributes that, when Talkback/VoiceOver are activated, perform audio feed back to users that present visual impairment.

One of the focus of the project was to be as inclusive as possible, but mainly to assist users that would normally struggle to catch buses due to their difficulty in seeing, making accessibility on the application very important.

The way these features were implemented took two phases:

- Automatically verifying if the user has the mobile accessibility feature activated, and thus automatically allowing the app to perform adjustments to visually impaired users.
- Finding which components and what type of feedback each one of them would give to the user to give visually impaired user the best usability possible and assisting their day to day lives when using this application.

To detect if Talkback/VoiceOver is currently turned on the the following code, already provided by React Native library was implemented.

```
//Check is accessibility is turned on the mobile device
  AccessibilityInfo.fetch().then((isEnabled) => {
    //True if Accessibility is turned on
    setAccessibility(isEnabled)
  });
```

If accessibility is turned on, some other components on the map are rendered, that users with Accessibility turned off don't require to have access. These components are markers that represent fixed stop, dedicated to visually impaired users. For feature development, using these users' feedbacks some other components can be created that are only available for them.

Following this process some attributes were implemented on React Native's components to define what should be said on the audio feedback to the users. These attributes added were accessibility (accepts boolean as value) to indicate if certain View component can be accessed by accessibility, accessibilityLabel (accepts string as value) that represents a title for the selected component and accessibilityHint (accepts string as value) that represents a description for said component. These strings are read by the accessibility turning into the audio feedback that visually impaired users need.

Another factor is that these users must learn the application all by themselves. They need to know what makes the application, all the feedbacks that it can give, and which components they need to be paying attention. To also aid with parts of the application that have its data constantly updated it was used the attribute accessibilityLiveRegion that provides audio feedback every time the component where it is implemented suffers and update to its data. This feature is restricted to Android users.

9. Source Code

9.1. Drawer Navigation

The library react-navigation allows the implementation of a lateral menu for the user to navigate. By default, each of the components incorporated into the Drawer Navigation have a header were a title can be inserted.

```
const Drawer = createDrawerNavigator()

export default function DrawerRoute () {
  const {translations} = useContext(LocalizationContext)

  return (
    <NavigationContainer>
      <Drawer.Navigator inicialRouteName='Buses' drawerContent={props => <DrawerContent {...props}/>}>
        <Drawer.Screen name={translations.BUSES} component={Buses}  /*options={navigationOptionsHeader}*/ />
        <Drawer.Screen name={translations.STOPS} component={Stops} />
        <Drawer.Screen name={translations.SCHEDULES} component={Schedules} />
        <Drawer.Screen name={translations.SETTINGS} component={Settings} />
        <Drawer.Screen name={translations.ABOUT} component={About} />
        <Drawer.Screen name={translations.HELP} component={Help} />
      </Drawer.Navigator>
    </NavigationContainer>
  )
}
```

Alongside with the drawer navigation was created another component which may focus is to stylize the items that appear on the menu with icons and color changes.

```
export function DrawerContent (props) {
  const {translations} = useContext(LocalizationContext)
  return (
    <View style={{flex: 1}}>
      <DrawerContentScrollView {...props}>
        <View style={stylesDrawerContent.drawerContent}>
          <Image
            style={{flex:1, height: 150, width: 120, marginLeft: 70, marginTop:20, marginEnd:20, alignItems:'center'}}
            source={require('./../assets/cmvc.png')}
          />
          <Drawer.Section style={stylesDrawerContent.drawerSection}>
            <DrawerItem
              icon={({color, size}) => (
                <Icon name='bus' color={color} size={size} />
              )}
              label={translations.BUSES}
              onPress={() => {props.navigation.navigate(translations.BUSES)}}
            />
            <DrawerItem
              icon={({color, size}) => (
                <Icon name='bus-stop' color={color} size={size} />
              )}
              label={translations.STOPS}
              onPress={() => {props.navigation.navigate(translations.STOPS)}}
            />
```

9.2. Bus Route, Buses and Selecting a Bus

To obtain either the buses available and the route that they perform firstly it is require to fetch the information to the database.

```
//Fetches Buses Route
fetch(
  'http://projetoiii-busapp.000webhostapp.com/slimFramework/index.php/api/routes',
)
  .then(response => response.json())
  .then(function (json) {
    const listOfCoords = json.map(function (point) {
      const lat = parseFloat(point.lat)
      const long = parseFloat(point.long)

      return {latitude: lat, longitude: long}
    })
    //useState to be used by the application containing all coordinates of the route
    setRouteCoords(listOfCoords)
  })
  .catch(error => console.error(error))
```

The following code corresponds to a simulation of getting the buses and make those buses move from time to time. Because webhosts present some limitations, the simulation of a bus signal had to be performed on the application itself. In the future the following code needs to be replaced by the City Hall's webservice that should retrieve all active buses signals.

Without this webservice the signal is simulated by fetching from an API a value from the route that will correspond to the bus position. Each iteration (that occurs every 2.5 seconds) fetches a new point of the route that is positioned five coordinates ahead from the previous one.

```
//Executed when routeCoords is not null -> fetches buses for the first time
 useEffect(() => {
  if (routeCoords.length > 0) {
   var x = [...buses]
   fetch(
    `http://projetoiii-busapp.000webhostapp.com/slimFramework/index.php/api/busPosition/${temp}`,
   )
    .then(response => response.json())
    .then(function (json) {
     const latlong = json.map(function (coord) {
      const key = '1'
      const latitude = parseFloat(coord.lat)
      const longitude = parseFloat(coord.long)

      return { key: key, latitude: latitude, longitude: longitude }
     })
     x[0] = latlong[0]
     setBus(latlong[0])
     setTemp(temp + 2)
    })
    .catch(error => console.error(error))
   fetch(
```

```
      `http://projetoiii-busapp.000webhostapp.com/slimFramework/index.php/api/busPosition/${temp1}`,
    )
      .then(response => response.json())
      .then(function (json) {
        const latlong = json.map(function (coord) {
          const key = '2'
          const latitude = parseFloat(coord.lat)
          const longitude = parseFloat(coord.long)

          return { key: key, latitude: latitude, longitude: longitude }
        })

        x[1] = latlong[0]
        setTemp1(temp1 + 2)

        setBuses(x)
      })
      .catch(error => console.error(error))
    setLoading(false)
  }
 }, [routeCoords])
//Occurs each time buses variable changes its value (timer can be changed)
 useEffect(() => {
  const timer = setTimeout(() => {
    var x = [...buses]
    /**Keeps track of the buses' previous positions in order
     * to assist on the calculation of each bus speed and thus
     * calculate estimate arrival time
     */
    var previousPos = [...busesPreviousPosition]

    previousPos[0] = x[0]
    previousPos[1] = x[1]

    setBusesPreviousPosition(previousPos)
    fetch(
      `http://projetoiii-busapp.000webhostapp.com/slimFramework/index.php/api/busPosition/${temp}`,
    )
      .then(response => response.json())
      .then(function (json) {
        const latlong = json.map(function (coord) {
          const key = '1'
          const latitude = parseFloat(coord.lat)
          const longitude = parseFloat(coord.long)

          return { key: key, latitude: latitude, longitude: longitude }
        })

        x[0] = latlong[0]
      })
```

```
    .catch(error => console.error(error))

  fetch(
    `http://projetoiii-busapp.000webhostapp.com/slimFramework/index.php/api/busPosition/${temp1}`,
  )
    .then(response => response.json())
    .then(function (json) {
      const latlong = json.map(function (coord) {
        const key = '2'
        const latitude = parseFloat(coord.lat)
        const longitude = parseFloat(coord.long)

        return { key: key, latitude: latitude, longitude: longitude }
      })
      x[1] = latlong[0]
      setBuses(x)
    })
    .catch(error => console.error(error))

  if (temp >= 380) {
    setTemp(1)
  } else {
    setTemp(temp + 2)
  }
  if (temp1 >= 380) {
    setTemp1(1)
  } else {
    setTemp1(temp1 + 2)
  }
 }, 5000)

 return () => clearTimeout(timer)
}, [buses])
```

After having the route coordinates and fetching the buses' position when the application is turned on, the line that corresponds to the route and the icons that correspond to the bus must be created on the map. For the route is used Polyline, multi-line drawing component available on react-native-maps library and Marker also available on the same library.

```
{/**Component that returns route of the buses*/}
<Polyline
  coordinates={routeCoords}
  strokeWidth={7}
  strokeColor='grey'
/>
```

On source attribute for Image component the images changes depending on if the bus icon was selected or not. If the user selects a bus it turns red giving visual feedback to the user about which of the bus is currently being tracked.

```
{/**Component that creates a Marker for each bus active. Each marker has a bus as an icon*/}
{buses.map((bus, index) => (
  <Marker
    key={bus.key}
    coordinate={{latitude: bus.latitude, longitude: bus.longitude}}
    onPress={() => getBusInfo(bus, index)}>
    <Image
      style={{width: 60, height: 40, resizeMode: 'contain'}}
      source={
        activatedMarker.key === buses[index].key
          ? require('../assets/bus1.png')
          : require('../assets/bus.png')
      }
    />
  </Marker>
))}
```

9.3. Distance and Estimate Time Calculation

Has seen on the previous image, each marker has a onPress attribute that triggers a function. That function is responsible to assign a key, according to the selected icon, to the useState activatedMarker (so that the application knows which marker needs to change colors)

```
//Starts the process of getting distance and estimated time of selected bus to user's closest point to the route
  const getBusInfo = (bus, index) => {
    bs.current.snapTo(0)
    setActivatedIndex(index)
    setActivatedMarker({key: bus.key})
  }
```

Using useEffect the calculations are performed and the route that the bus is going to take is highlighted in green.

```
useEffect(() => {
  if (activatedMarker.key !== '') {
    const ourBus = buses[activatedIndex]
    setSelectedBus(ourBus)
    if (inRoute) {
      const closestToBus = findNearest(
        {latitude: ourBus.latitude, longitude: ourBus.longitude},
        routeCoords,
      )
      const closestToMarker = findNearest(markerPosition, routeCoords)

      const startWaypoint = routeCoords.findIndex(
        route =>
          route.latitude == closestToBus.latitude &&
          route.longitude == closestToBus.longitude,
      )
```

```
    const endWaypoint = routeCoords.findIndex(
      route =>
        route.latitude == closestToMarker.latitude &&
        route.longitude == closestToMarker.longitude,
    )

    if (startWaypoint < endWaypoint) {
      const listaWaypoints = routeCoords.slice(
        startWaypoint + 1,
        endWaypoint + 1,
      )
      setWaypoints(listaWaypoints)
    } else {
      const temp = routeCoords.slice(
        startWaypoint + 1,
        routeCoords.length - 1,
      )
      const listaWaypoints = temp.concat(routeCoords.slice(0, endWaypoint + 1))

      setWaypoints(listaWaypoints)
    }
  } else {
    const closestToBus = findNearest(
      {latitude: ourBus.latitude, longitude: ourBus.longitude},
      routeCoords,
    )
    const closestToMarker = findNearest(markerPosition, referencePoints)
    const closestToReference = findNearest(
      {
        latitude: closestToMarker.latitude,
        longitude: closestToMarker.longitude,
      },
      routeCoords,
    )

    const startWaypoint = routeCoords.findIndex(
      route =>
        route.latitude == closestToBus.latitude &&
        route.longitude == closestToBus.longitude,
    )
    const endWaypoint = routeCoords.findIndex(
      route =>
        route.latitude == closestToReference.latitude &&
        route.longitude == closestToReference.longitude,
    )

    if (startWaypoint < endWaypoint) {
      const listaWaypoints = routeCoords.slice(
        startWaypoint + 1,
        endWaypoint +1 ,
```

```
        )
        setWaypoints(listaWaypoints)
      } else {
        const temp = routeCoords.slice(
          startWaypoint + 1,
          routeCoords.length - 1,
        )
        const listaWaypoints = temp.concat(routeCoords.slice(0, endWaypoint +1 ))

        setWaypoints(listaWaypoints)
      }
    }
  }
 }, [buses])
```

Each time a new bus position is fetched and if there is a activated marker the useEffect constantly splits the route into a smaller portion (waypoints) that correspondes to the route from the selected bus to the user. When these waypoints are activated another useEffect is activated which performs the necessary calculations for time and distance.

```
useEffect(() => {
  if (waypoints.length > 0 && !enterState) {
    getBusDistance()
    getEstimatedTime()

    setIsTracking(true)
  }
 }, [waypoints])
```

```
const getBusDistance = () => {
  const distanceCalculation = getPathLength(waypoints)
  setDistance(convertDistance(distanceCalculation, 'km'))
 }
const getEstimatedTime = () => {
  const velocity = getVelocity()
  setEstimateTime((distance /velocity) * 60)
 }
const getVelocity = () => {
  const velocity =
    getPreciseDistance(
      busesPreviousPosition[activatedIndex],
      buses[activatedIndex],
    ) / 2.5
  return convertSpeed(velocity, 'kmh')
 }
```

When all these operations end, a variable named isTracking is set to true. This allows the route highlighted in green to be drawn.

```
{isTracking ? (
          <View><Polyline
          coordinates={waypoints}
          strokeWidth={7}
          strokeColor='green'
         />
          </View>) : (<View/>)}
```

9.4. Getting in and out of the bus

When a moving bus is close enough of the user's position, and when the user intends to get inside of the bus, a button can be clicked meaning that the user got inside the bus. By pressing this button all the bottomsheet information is changed, since informing the user about distance and estimate arrival time is unnecessary when inside the bus. Instead, a new set of information is displayed to the user regarding the current point of interest that the bus is going through while simultaneosly informing which point of interest the bus just passed and which point of interest lays ahead.

```
  {/**Button to get inside or out of the bus */}
    <TouchableOpacity style={styleBuses.panelButton} onPress={changeState}>
     <Icon name='bus' size={30} color='white' />
     <Text style={styleBuses.panelButtonTitle}>
      {enterState ? translations.exit : translations.enter}
     </Text>
    </TouchableOpacity>
```

```
<View style={styleBuses.panel} accessible={true}>
    {/**If the button 'Go in the bus' is pressed and bus close to user -> enterState = true */}
    {enterState ? (
     <View style={{ alignItems: 'center' }} >
      <View style={styleBuses.panelHandle} />
      <View style={styleBuses.iconRow}>
       <Text style={styleBuses.panelPar}>{translations.ptInt}</Text>
      </View>
      <Text style={styleBuses.panelTitle} accessible={true} accessibilityLiveRegion="polite">{info.title}</Text>
      <Text style={styleBuses.panelSubtitle} accessible={true} accessibilityLiveRegion="polite">{info.desc}</Text>
      <View style={styleBuses.iconRow}>
       <Icon name='arrow-left' color={'gray'} size={30} accessibilityHint={translations.previousStop} />
       <Text style={styleBuses.panelPar}>{points.pointA}</Text>
      </View>
      <View style={styleBuses.iconRow}>
       <Icon name='arrow-right' color={'gray'} size={30} accessibilityHint={translations.nextStop} />
       <Text style={styleBuses.panelParv2}>{points.pointB}</Text>
      </View>
     </View>
```

When the variable “enterState” the information changes and the following code the condition for the following code to be executed is satisfied.

```
{/**If enterState = true then markers
        * that represent points of interess are rendered */}
        {enterState ? (pointsOfInterest.map(marker => (
          <Marker key={marker.idpointsinterested}
            coordinate={{ latitude: marker.latitude, longitude: marker.longitude }}
            title={marker.title}
            description={marker.det}>
            <Icon name='information-variant' color="red" size={20} />
          </Marker>))) : (<View />)}
```

9.5. User Location

To get a user position using the GPS from mobile phones the following code was implemented.

```
Geolocation.getCurrentPosition(handleSuccess, handleError)
```

If the position is successfully obtained it is set to a variable markerPosition that is used to update by executing the function handleSuccess.

```
const handleSuccess = position => {
  var lat = parseFloat(position.coords.latitude)
  var long = parseFloat(position.coords.longitude)

  var initialRegion = {
    latitude: lat,
    longitude: long,
    latitudeDelta: 0.01,
    longitudeDelta: 0.01,
  }
  //Set new map position
  setInitialPosition(initialRegion)
  //Set new marker position
  setMarkerPosition(initialRegion)
```

Even though the map component from Google API does not require the user position stored on the variable markerPosition, since the map has an attribute called showUserLocation that if set to true displays a blue spot that is constantly updated when the user moves, this constant update of the markerPosition value is necessary to perform calculations during runtime of the application.

To update this variable automatically over time it is used the watchPosition function of Geolocation to watch for changes on user’s position.

```
useEffect(() => {
   const watchId = Geolocation.watchPosition(handleSuccess, handleError, {
    enableHighAccuracy: true,
    timeout: 20000,
    maximumAge: 1000,
    accuracy: 'high',
    distanceFilter: 10,
   })
   return () => Geolocation.clearWatch(watchId)
 }, [])
```

Every second the user position is updated with new data.

10. Final Implementations

10.1. Usability Tests

To get a feedback about the usability of the app and get a better perception of what can and should be improved, the application was made available to four testers that after testing responded to a questionnaire about their perception of the app when exploring it and using it. The questions provided are based on the System Usability Scale (SUS) **(Teixeira, 2021)**

Table 1: LIST OF QUESTIONS (SUS)

Question 1	I would like to use the system frequently
Question 2	The system was unnecessary complex
Question 3	The system is easy to use
Question 4	The support of a technical person was needed to use the system
Question 5	The various functions of the system were well integrated
Question 6	There was too much inconsistency in the system
Question 7	People would learn to use the system very quickly
Question 8	The system was very cumbersome to use
Question 9	I am very confident to use the system
Question 10	I needed to learn a lot of things before I could get on with the system

Table 2: SUS SCORES FOR ALL FOUR TESTERS

	Tester 1	Tester 2	Tester 3	Tester 4
Question 1	3	2	4	3
Question 2	1	2	1	2
Question 3	4	4	5	5
Question 4	2	1	1	1
Question 5	4	5	4	4
Question 6	1	1	1	1
Question 7	4	4	4	5
Question 8	1	4	1	1
Question 9	4	4	4	4
Question 10	2	1	1	2
Score	50	65	70	70

The average of the score obtained, using the calculation method defined by SUS, resulted in a score of approximately 64 points, a little less that the desired one. Commonly between testers they present the same opinion about not thinking they would use the application since the electrical buses do not correspond to their needs. Also, they all agreed that the application is simple to use, they believe people would adjust well to the application and presents a good structure for usability.

10.2. Final prototype

Based on the mockups demonstrated on Chapter 6 the visual aspect of the prototype was created. The general idea represented on the mockups was implemented, having only changed the aesthetic of the prototype to give it a more modern aspect and to be visually pleasing to the user.

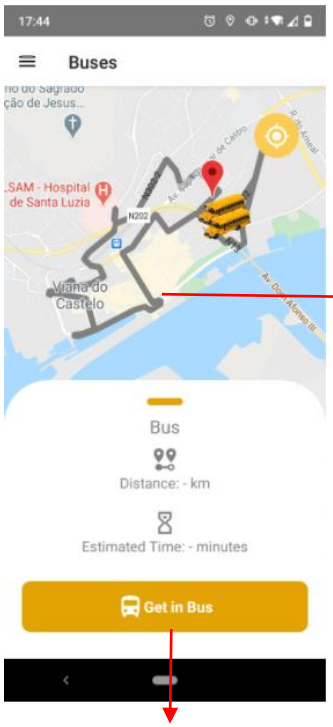
When the user turns on the application it starts rendering, fetching information about active buses, coordinates to build their route, fetching user's current location and checking how far away the user is from the route. This last aspect is important to restrict the user's functionalities on the app. The first screen that the user has access to contains a map with buses' route, icons that represent the bus, a blue marker that represents the user and a top left menu icon that opens a menu to redirect the user to other screens.

Menu Button

Button that takes centers the map on the user's location

User's current position. Updates each time the user moves.

By sliding up on the screen this bottomsheet appears. Has information about a selected bus distance and estimate time.

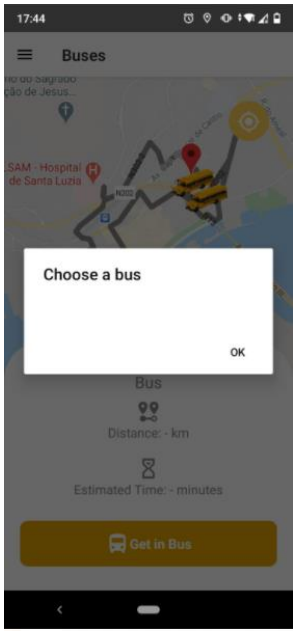


Polyline that represents the route

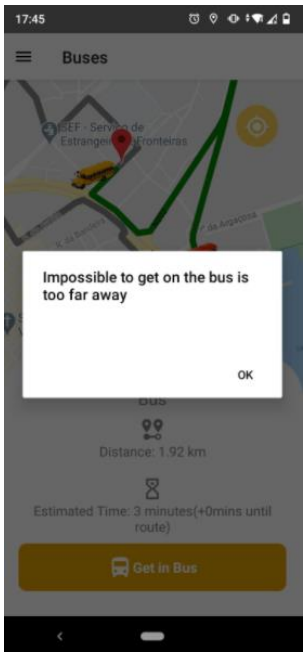
Button to be clicked when the user catches a bus. Changes the information that appears on the bottomsheet. Has three distinct actions.

1 **2** **3**

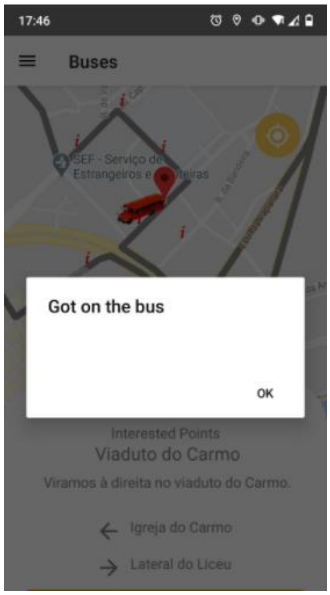
1 If a bus was not selected.



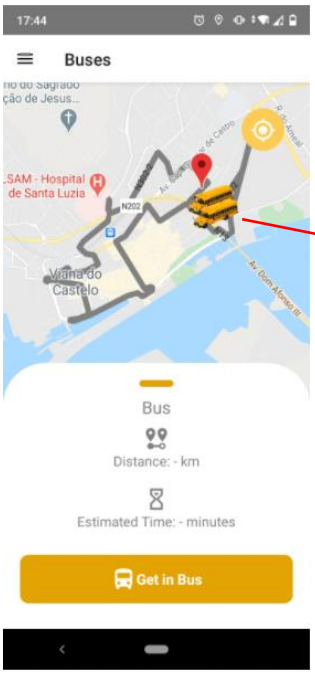
2 bus was selected but too far away.

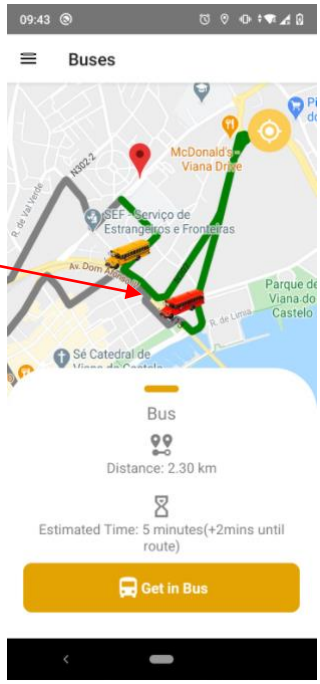


3 If a bus was selected and near the user



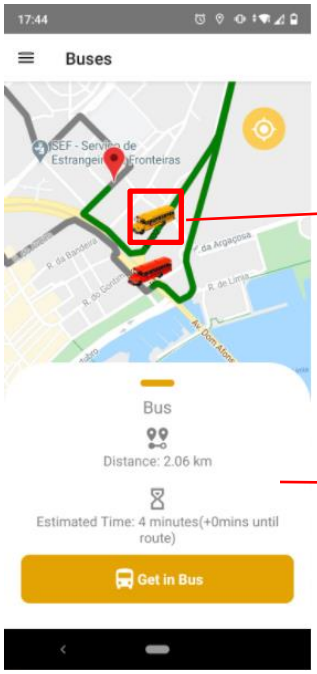
As an example, a marker (red icon) was created to test the features of the application. That represents a user's location. To start tracking a bus, meaning get the information about the distance from the selected bus and the user position (or closest point to the route in case the user is far from the route) and estimate time the user just needs to click on a bus icon. To change the selected bus, it is only required to click on another icon. The selected bus has its color changed from yellow to red.





Calculates the distance from the selected bus and the user position (marker). The estimate time is calculated using physics formulas. Acessibility reads the entire text when user slides the finger to the bottomsheet

Since the route is obtained by rendering a polyline using multiple coordinates, to obtain the length is used the library 'geolib' method 'getPathLength'. This method uses an array of points can calculates the distance. The array of coordinates provided results from a slice between the coordinates of the route that start on the bus and ends on the user marker. That path from the selected bus and user is rendered as a green polyline on the map. That polyline gets constantly updated while the bus is moving.



With accessibility turned on by click once on the icon it gives instructions to the user about what will happen by selecting a bus

No only the distance is getting constantly updated but also the estimate time

In a real scenario the calculations involved to calculate the estimate arrival time would proceed as follows:

1. Get the distance between the previous coordinate of the bus and the current bus position.
2. With that distance obtained it is possible to calculate the speed of the bus (in Km/h).

Velocity (Km/h) = Distance(Km)/Time(h)

Distance -> represents the distance between previous and current bus coordinate.

Time -> represents the time interval between a new API fetch (5 seconds)

3. After getting the estimate velocity from the bus it is proceeded with the calculations of the estimate arrival time, using the same physics formula represented previously.

Velocity (Km/h) = Distance(Km)/Time(h) -> Time(h) = Distance(Km)/Velocity(Km/h)

Velocity -> represents previously obtained velocity.

Distance -> represents distance between bus position and user marker.

4. The time calculated is then multiplied by 60 to obtain the value in minutes.

Since at this point there was no web service to test with a real scenario, the validity of the code incorporated needed to be tested in a hypothetical scenario. It was considered that the bus would constantly be moving at 30Km/h. With that in consideration it was only necessary to apply the physics formula once to obtain the estimate time.

On the previous figure the distance from the bus to the user is 1.96km and as mentioned before the speed if 30Km/h then:

Time = 2.06/30 => 0.068666666(6)h * 60 minutes => 4.12 that rounded up is 4 minutes.

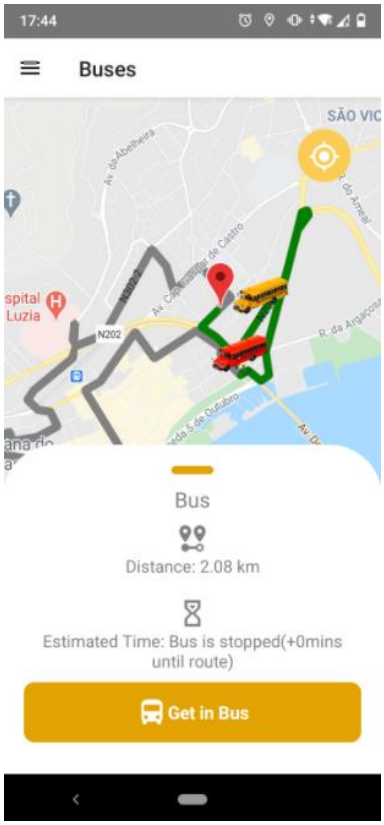
Before the bus was 2.06 km away from the user it began at 2.30km.

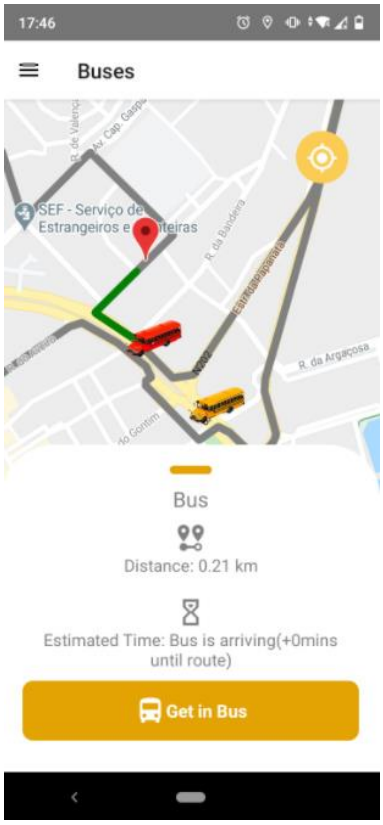
Time = 2.30/30 => 0.076666666(6)h * 60 minutes => 4.6 that rounded up is 5 minutes.

Beside the estimate time of arrival in minutes it is also presented to the user the following messages:

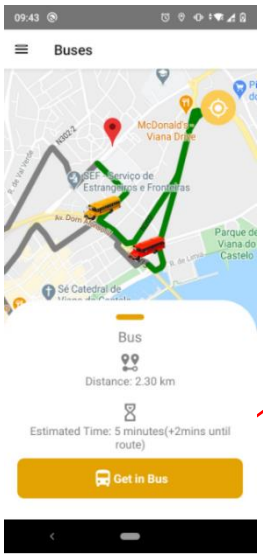
- If the bus is not moving -> Bus is stopped.
- If the bus is close to the user's location -> The bus is arriving

These messages differ on the decisive factor. If the message is "Bus is stopped" that means the speed is 0 and the time value is not a finite number. If the message is "The bus is arriving" it means that estimate time of arrival is less than a minute.

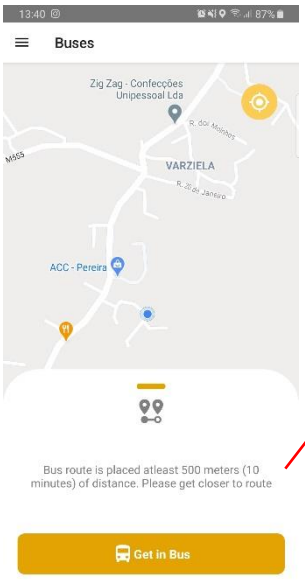


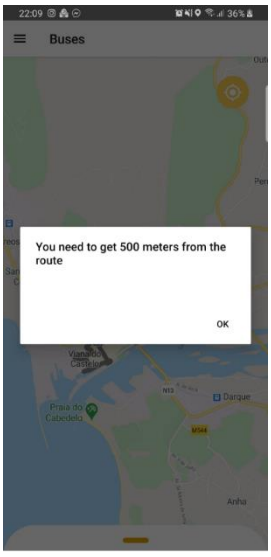


It can also be seen that after the estimate time there is another value in minutes, inside the parenthesis. That value corresponds to how long the user takes to get to the route. If the user is on the route the time that the user takes to get to it is 0. The further the user gets from the route the bigger the value. If the user is more than 500 meters from the route the bottomsheet will present a message warning the user.



It is used the same physics formula that is being used to calculate the estimate arrival time for the buses. The difference is that is being used a average walking speed of a person (3Km/h) and the distance considered is between the user's marker and the closest coordinate to the route.

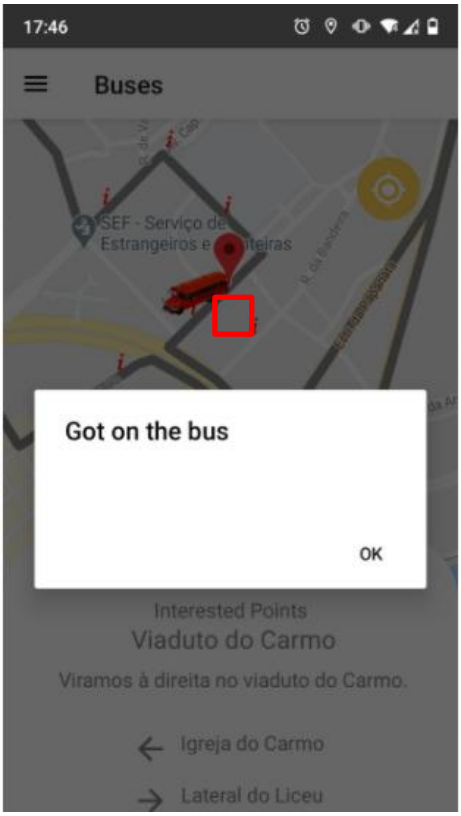


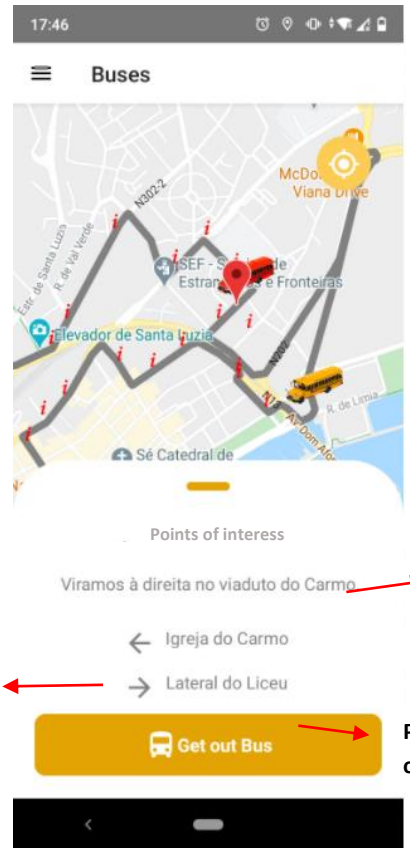


If the user is more than 500 meters it can't select a bus

When the user gets into the bus the feature from the button 'Get in Bus' can be activated without any problems. This button performs the following functionalities:

- Changes the information that appears in the bottom sheet from information about the bus distance and estimate time to information about the current point of interest and the following and previous one.
- Disables the green route because it is no longer required.
- Renders points of interest on the map.
- Changes button text (to 'Get out bus') and functionality.

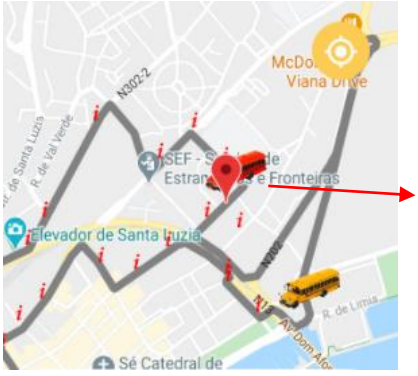




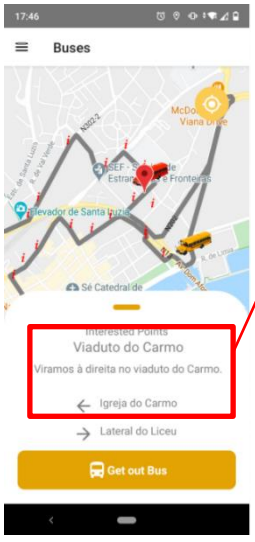
Current point of interest

Next point of interess

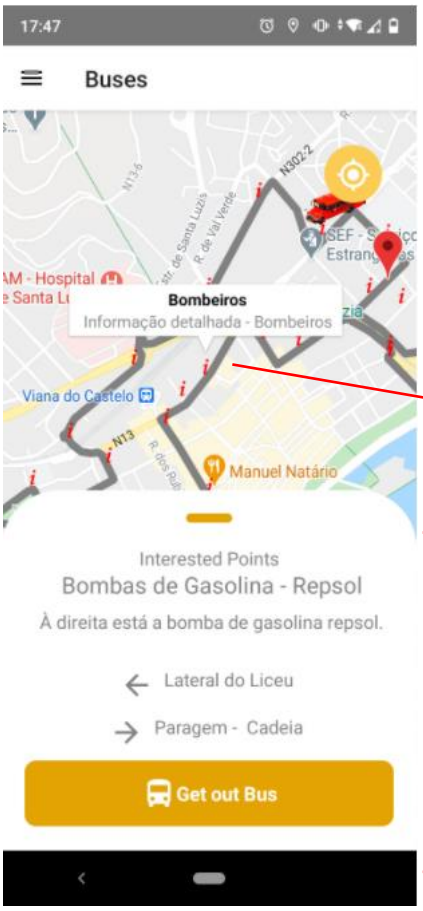
Previous point of interess



Markers that represent the point of interest



When this value is updated, if the Accessibility is activated it give feedback, automatically, to the user without requiring the user to select the bottom sheet

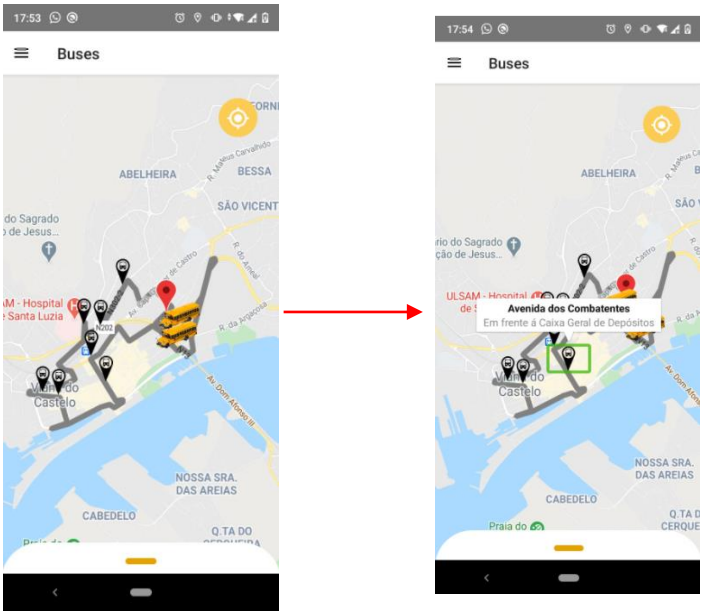


Clicking on the marker provides information to the user. Accessibility reads the information

Information updated.

When clicking on the 'Get out Bus' button the application's initial features (mentioned previously) return to the user. The distance and estimate time get calculated again and the green path renders.

If the Accessibility is turned on in the mobile phone of the user, the application automatically detects it and performs an adjustment for visually impaired people. The initial map renders additional markers that represent the fixed stops for these users. By sliding the finger on each marker, the accessibility feature reads the text that comes with which marker.



Referring to the menu button, when opened it becomes available to the user multiple options so that they can navigate through the application. Each option represents the following:

- Buses – The initial screen that the user has access to. Contains information about the moving buses.
- Stops – Information about fixed stop in Viana do Castelo. This screen is especially useful to visually impaired users.
- Schedules – Contains information about schedules from other types of bus companies.
- About – Information about the app and its creators.
- Help – Contains answers to questions that some users may come across when using the application.

10.3. Conclusion

The focus of the application was to provide a tracking application that is inclusive of everyone, people that usually take public transportations to move around the city, tourists that visit Viana do Castelo and visually impaired people to assist them on their daily routines. It is provided to all users an easy-to-use interface to be appealing for the user and easy to learn by its organization.

Developing the application using React Native allowed not only the development of an application natively for both Android and iOS, being available for most people to use the application, but also the incorporation of libraries that allow performing geospacial operations that were required to present relevant information to all user, like distance and estimate arrival time. Those geospacial operations not onlye assisted on finding the closest point on the route from where the user was standing and thus allowing to perform calculations but also allowed to determine if the user is close enough to the route and enable or disable some features considering that factor.

Creating the app also encouraged the exploring of accessibility features, included by default on React Native, and to learn about of visually impaired people use applications, having to consider different usability rules for these types of users. All of that to make sure that the application is not only pleasant and easy to use but also secure.

It was developed a functional prototype, where some features are local simulations while other features use webhost database access to fetch information to implement on the application. The objective met was to provide all utility to the user using a single application.

References

000webhost. (2020). *Alojamento Web Grátis*. Fonte: 000webhost: https://pt.000webhost.com/?__cf_chl_jschl_tk__=2a2af15ae6c1d1698bcc9ec62b36013a1f57c538-1608499722-0-AUelGNd8iydViKEbdUcZEsJXM-4FKb_kgOLh_VPK6b1W07EbUWV_6Zho1WQwRyXChTMcmVz56b8XMadZeTF9JRQKHDRP8djLNJlvjsbKy6Rn8Jz8GFE-tXt9iHZp8fpHQ_SoZNgnTUP-Mlcf_VZr5YY43n

Bieh, M. (2020). *geolib*. Fonte: GitHub: https://github.com/manuelbieh/geolib

Bramus. (2020). *react-native-maps-directions*. Fonte: GitHub: https://github.com/bramus/react-native-maps-directions

Clement, J. (30 de 10 de 2020). *Most popular Apple App Store categories in August 2020, by share of available apps*. Fonte: Statista: https://www.statista.com/statistics/270291/popular-categories-in-the-app-store/

Clement, J. (30 de 10 de 2020). *Most popular Google Play app categories as of 3rd quarter 2020, by share of available apps*. Fonte: Statista: https://www.statista.com/statistics/279286/google-play-android-app-categories/

Facebook. (10 de 12 de 2020). *Accessibily*. Fonte: React Native: https://reactnative.dev/docs/accessibility

Facebook. (2020). *Introducing Hooks*. Fonte: React: https://reactjs.org/docs/hooks-intro.html

Facebook. (2020). *React Native*. Fonte: React Native: https://reactnative.dev/

Facebook. (2020). *Using the Effect Hook*. Fonte: React: https://reactjs.org/docs/hooks-effect.html

Facebook. (2020). *Using the State Hook*. Fonte: React: https://reactjs.org/docs/hooks-state.html

GitHub. (2020). *GitHub*. Fonte: GitHub: https://github.com/

Google. (2020). *react-native-maps*. Fonte: GitHub: https://github.com/react-native-maps/react-native-maps

Group, T. P. (2020). *What is PHP?* Fonte: PHP: https://www.php.net/manual/en/intro-whatis.php

IQBAL, M. (30 de 10 de 2020). *App Download and Usage Statistics (2020)*. Fonte: BusinessofApps: https://www.businessofapps.com/data/app-statistics/

Josh Lockhart, A. S. (2020). *Slim 4 Documentation*. Fonte: Slim: https://www.slimframework.com/docs/v4/

Microsoft. (2020). *Getting Started*. Fonte: Visual Studio Code: https://code.visualstudio.com/docs

MockingBot. (20 de 12 de 2020). *MockingBot*. Fonte: MockingBot: https://mockingbot.com/

Mockup. (20 de 12 de 2020). *Mockup*. Fonte: seobilitywiki: https://www.seobility.net/en/wiki/Mockup

Niklauslee. (20 de 10 de 2020). *Introduction*. Fonte: StarUML documentation: https://docs.staruml.io/

Oracle. (2020). *MySQL*. Fonte: MySQL: https://www.mysql.com/

Oreilly. (2020). *Chapter 1. What Is React Native?* Fonte: Oreilly: https://www.oreilly.com/library/view/learning-react-native/9781491929049/ch01.html

Spector, J. (23 de 07 de 2018). *Study: Electric Buses Already Emit Less Carbon Than Diesel Buses, in Any State*. Fonte: GreenTechMedia: https://www.greentechmedia.com/articles/read/study-electric-buses-already-emit-less-carbon-than-diesel-buses-in-any-stat

Teixeira, F. (10 de 01 de 2021). *O que é o SUS (System Usability Scale) e como usá-lo em seu site*. Fonte: UX Collection: https://brasil.uxdesign.cc/o-que-%C3%A9-o-sus-system-usability-scale-e-como-us%C3%A1-lo-em-seu-site-6d63224481c8

Transport & Environment. (2018). *CO2 Emissions From Cars: the facts.* Bruxelas: European Federation for Transport and Environment AISBL. https://www.transportenvironment.org/sites/te/files/publications/2018_04_CO2_emissions_cars_The_facts_report_final_0_0.pdf

Trello. (11 de 2020). *Trello*. Fonte: Trello: https://help.trello.com/article/708-what-is-trello

Vrána, J. (2010). *NotORM*. Fonte: NotORM: https://www.notorm.com/

W3Schools. (2020). *What is npm?* Fonte: w3shchools.com: https://www.w3schools.com/whatis/whatis_npm.asp