INSTITUTO POLITÉCNICO DE BRAGANÇA
Escola Superior de Tecnologia e Gestão

Mobile Application for Fleet Management

Kazim Ahmad

Project work presented to the Escola Superior de Tecnologia e Gestão to obtain the Master's Degree in Information Systems

Supervisor:

Professor Doutor Paulo Alexandre Vara Alves

Bragança

October 2020

Abstract

Fleet management is not difficult if your company only has five to ten vehicles. But if there are thousands of vehicles, then you need a fleet management system, or you will be overwhelmed by the sheer amount of work for you.

With the help of a fleet management system, you can monitor any vehicle in real-time. You can find out whether the customer has received the item they purchased by or is on the way. You can also know whether your driver is working or just idling by.

Without a vehicle management system, you only rely on the trust you have between you and your drivers. And more often than not, your trust is betrayed by your drivers.

With a vehicle management system, routine service schedules information can be obtained anytime and anywhere. You only have to enter the required data into the system and it will arrange the schedule.

Fleet management certainly involves drivers. The involvement starts with determining work schedules and reporting. Staff who arrive late at the office will certainly affect your customer satisfaction because the delivery will definitely be delayed.

The fleet management system will eliminate these things. The system will automatically inform the driver of their work schedule so he/she can come to work on time.

A fleet management a website and a Android mobile application to facilitate the in-house management, viewing options and communications for the associated employees. In this project, an iOS mobile application is needed to manage the driver, vehicle and intercommunication of drivers. The application then further provides the functionalities for driver to view and track the Alarms, Events and other vehicle based on the geo location of the vehicle. Furthermore, it provides the facility for management to keep track of all the vehicles and their trips which helps to analyze the data. This report covers the development of the project of Fleet Management application.

Many companies want to stay small and keep their contracts manageable, and they’re happy with a few cars or trucks. Even so, they would still like to do more business. The Fleet Management application has been designed to provide solutions that will help them function

more efficiently. A company can greatly increase its profits for relatively low investment in these solutions.

Whether large or small, any business can find a fleet management software product on the market that meets its needs.

We all know that it is difficult, if not impossible, to predict the future. Generally, there are so many different variables to consider, many of them very difficult to predict, that make the whole exercise quite complex.

Predicting the future of fleet management is this type of exercise. If you think someone in the world already knows how it will be, think twice. From truck manufacturers to telematics companies, from large technology companies to fleet managers with 30 years of experience, nobody knows for sure.

we added some cutting-edge technologies to this Fleet Management system. With that in mind, we are in fact proposing a new future for fleet management. Or, at least, contributing to it.

Table of Contents

Table of Figures

1. Introduction

Companies, which deal with the fleets and their management, need both proper and thorough oversight of the organization's vehicle fleet maintenance activities to ensure the overall value and health of their fleet assets.

This responsibility typically falls on the organization's fleet manager. As a general rule, most fleet management plans and policies include routine inspections of the vehicles as per their working life. Here, the addition of customized Fleet Management system can easily help and troubleshoot any issues when fleet vehicle maintenance might be required.

With this software, companies will be able to keep an eye on the vehicle inspection and maintenance requirements of their extensive fleet.

This project is a fleet management solution that combines true fleet intelligence and great customer service for companies worldwide. Smartphone App is a mobile application, part of the client software product. It is already in the App Store and usable for company's clients.

The Smartphone App is made to be used by fleet managers. After login, the user is able to consult the fleet vehicles, drivers, trips, paths, etc. The user can also interact with drivers, find the nearest vehicles, block vehicles and receive activity notifications of each vehicle.

Initially a team of two members from the company were assigned to the fleet management application. The work was split into two areas:

- The local database and Parsing services (JSON is the most commonly used format to send and receive data from the web services. The data is in the form of key-value pairs. Using Swift Dictionaries, we fetch the values from the keys.

  The JSONSerialization class is used to parse a JSON data into a dictionary of key-value pairs by converting the Data object).

- Screens development.

The work was split in such way that by the time parsing services are being implemented the user interfaces of other features will be under development simultaneously, which includes setting up Bottom navigation bar and Home screen according to user's permission and

developing screens with responsiveness. And by the end of parsing services the user interface will be connected to the Web services JSON APIs with the help of parsing services.

Later on, after the finishing of the local database implementations the project had to be re-assign to a single person with all the responsibilities of the application development. This report covers the application development side of the fleet management project.

This report is divided into several sections that include: background chapter which reviews general mobile application development, requirement analysis that covers user requirements for the mobile application, design and implementation chapters that cover mobile development and web-service development, user testing and evaluation, future work and conclusion which includes a discussion of certain problems encountered in the project.

2. **Background Chapter**

When starting a mobile development, few questions may be raised. What differentiates mobile application development and traditional application development? What options are available? What are the important design principles for mobile application? What is a web- service? These questions are discussed in this chapter.

2.1 Software Engineering Issues in Mobile Development

Software engineering for mobile application shares similar practices with traditional application, but some specific issues need to be addressed in mobile development.

The first point is the potential interaction of applications with each other. Mobile devices have many applications from various sources with the possibility of interaction between them. Secondly, the accelerometers that respond to device movements, numerous touch screen gestures, global positioning system, microphones that are usable in applications other than voice calls, cameras, and multiple networking protocols are all in a single device, allowing many feature options for the application.

Mobile applications are required to support multiple devices with various screen sizes and different hardware. In addition, different versions of the operation systems are released much more frequently than the embedded devices complicating the support.

Lastly, many aspects of an application may affect the device's power, draining the battery life of the device.

The following issues were raised [1]:

- Determining which functions should be present in a mobile version of a traditional application.
- Providing techniques that can assure maximum reuse of code among different versions.
- Determining if the mobile user interface requires a different contextual design process to support a different set of use cases.
- Integrating the various forms of input and sensor data in application design.

- Determining if the synchronization techniques from traditional client-server computing would suffice in the potential loss of connectivity or battery power running out.
- Different application design depending on the speed of the network.
- Creating an application that can maximize battery life and resource usage.

The best practice suggested was to follow an agile methodology that can quickly adapt to changing user requirements and to follow development guidelines published by various platforms.

2.2 Human-Computer Interaction in Mobile

User interface is an important part of any software application that has user interaction. Even if the industrial design and their aesthetic are appealing, if it does not address real user needs, it will be of no use to the user [13].

Constant testing and refinement of the design by engaging with the actual users are vital. Various methods and activities can be used to find out what matters for the users. The most effective way to get user feedback on the design is through a prototype

There were a few design guidelines that stand out. The limiting of the number of items in the menu to a maximum of seven was one of them. Use of icons (symbols) was suggested to replace text whenever possible to free-up space. Generally, pictures are easier to remember than text. The authors argued [13] that well-designed overviews are powerful features that can overcome restrictions of small screen sizes. Overviews are a zoomed-out version of the displayed contents. By having an overview, the users can easily orientate themselves relative to the overall content. Other design guidelines included quick navigation to frequent functions, limiting excessive scrolling or page-to-page navigation and stroking not poking.

User experience (UX) focuses on having a deep understanding of users, what they need, what they value, their abilities, and also their limitations. It also takes into account the business goals and objectives of the group managing the project. UX best practices promote improving the quality of the user's interaction with and perceptions of your product and any related services.

3. State-of-the-Art

In this chapter the report discusses some applications that has the same goal to solve problems of fleet management, that are at the most recent stage in the development of a product, incorporating the newest technology, ideas, and features.

3.1 Tailwind TMS

Tailwind Transportation Software

Award-winning web-based transportation management software and does not have set-up fees, needed contracts and it has a free trial. Tailwind helps manage operations, customers, dispatch, admin & accounting, the main features include Billing & Invoicing, Carrier Management, Customer Management, Discount Management, Dispatch Management, Fleet Management, Live Driver Tracking, Order Management, Routing, Scheduling and Shipment Tracking [7].

3.2 The Vehicle Refurbishment

SMART

Field service management software for every industry. From streamlining routes to in-depth reporting and analysis, SMART Software helps the company services thrive and expand. Streamline the company operations while keeping all the important customer and business data in one place. The inventory management, equipment tracking, time management, and reporting boosts the bottom line. Keep track of all aspects of Service Issues, from ticketing and technician routing to inventory control and vehicle maintenance. SMART forecast vault cash differently than any processor or software on the market. Companies are returning 20% of their vault cash using SMART. SMART includes a fully integrated, powerful, and successful CRM solution to manage the relationships with the customers, vendors, and sales prospects [8].

3.3 Software for Billing, Dispatching, Scheduling and More

CTS Software

Industry-Leading NEMT & Paratransit Scheduling & Dispatching Software. TripMaster provides efficient, cost-effective NEMT, demand-response, and paratransit management tools.

It’s a full-service transit suite, including modules for: Automated scheduling, Powerful custom reporting, Integrated voice response, Mobile solutions and an automated vehicle locator, Web-based rider portal. CTS Software offers complete auditing support, manpower and vehicle resource management, cost control, payroll tracking, route management, statistical reporting, computer-assisted scheduling, electronic billing, and much more [9].

3.4 Schedule and Dispatching for Trucks

Razor Tracking

Razor Tracking is recognized as the one most-advanced Fleet Tracking Software. It provides powerful and easy to use equipment, fleet tracking and operations management platform. With Razor Tracking you can receive alerts on speeding vehicles, harsh braking, and poor driving. Never lose a tank, pull cart, or trailer. Some of the features include: Vehicle Grouping, Geofencing & Points of Interest, Live Vehicle Status and Automate Reports.

Also offer 6-months seasonal suspension, no contracts, free training, an unlimited number of users, and a 60-day money-back guarantee [10].

3.5 Efficiency and Scalability for Bookings

SWIVEL Software

Swivel Software a single cloud platform global logistics systems software management solution. It manages the supply chain daily operations, integrating business with customers and partner efficiently, ensuring gap variations components are set by the technical team providing the values and diversity within the supply chain (SOR) standard operations requirements. Systems solutions modules are: 360 (Single supply chain visibility operations platform) ERP (Global Logistics Systems Air, Ocean, Trucking) POM (Purchase Order) WMS (Warehouse DC) Swivel Trak (Shipments Tracking Status Milestone) CRM (Sales Retention Management) [11].

3.6 ParcelTrack

ParcelTrack Technologies

ParcelTrack keeps up to date on the status of the shipments, delivery forecasts and who accepted a parcel. It supports to follow the progress of the shipments in an easy way, on tablet,

in the car with mobile phone or at the desk of the office. ParcelTrack is available on all type of devices. The synchronization of all the devices keeps all the orders in one place. Regardless of the delivery service that ships the packages, ParcelTrack provides a single interface to keep track of the shipments. Is not needed anymore to visit various tracking sites and getting lost in technical details of their tracking systems. Follow the package in real-time with with ParcelTrack using GPS-location of the delivery van [12].

4. Development Technologies

4.1 Xcode

Xcode is an integrated development environment (IDE) for macOS containing a suite of software development tools developed by Apple for developing software for macOS, iOS, iPadOS, watchOS, and tvOS. It was first released in 2003; the latest stable release is version 12.1, released on October 20, 2020.

Xcode supports source code for the programming languages C, C++, Objective-C, Objective C++, Java, AppleScript,Python, Ruby, ResEdit (Rez), and Swift, with a variety of programming models [2].

4.2 Swift

Swift is general-purpose, multi-paradigm, compiled programming language developed by Apple Inc. and the open-source community, first released in 2014. Swift was developed as a replacement for Apple's earlier programming language Objective-C, as Objective-C had been largely unchanged since the early 1980s and lacked modern language features. Swift works with Apple's Cocoa and Cocoa Touch frameworks, and a key aspect of Swift's design was the ability to interoperate with the huge body of existing Objective-C code developed for Apple products over the previous decades. It is built with the open source LLVM compiler framework and has been included in Xcode since version 6, released in 2014 [3].

4.3 Realm Database

Embedded on the client, the Realm Database is a full-featured, object-oriented, cross-platform database that persists data locally on device. It's available for major mobile languages, such as Swift and Objective-C (iOS), Java (Android), C# (Xamarin, .NET), and JavaScript (React Native and Node.js). The Realm Database is lightweight and highly performant, capable of handling very large data loads and running queries in fractions of a second. Based on shared live objects, it syncs data seamlessly in real-time with the Realm Object Server without the need to write networking, serialization, or object-relational mapping code. This means that applications will be able to refresh data as fast as needed to provide an enjoyable, engaging

user experience. It supports the development of reactive applications due to the 'live object' nature of the database [4].

4.4 Firebase

Firebase is a toolset to “build, improve, and grow applications”, and the tools cover a large portion of the services that developers would normally have to build themselves, but don’t really want to build, because they’d rather be focusing on the app experience itself. This includes things like analytics, authentication, databases, configuration, file storage, push messaging, and so on. The services are hosted in the cloud, and scale with little to no effort on the part of the developer [5].

5. System Specifications

5.1 Platform requirements

- minimum iOS version: 10.0
- programming language: Swift
- XCode version: Should always work at the latest version

5.2 Architecture

The code is structured according with Model-View-Presenter architecture.

MVP uses passive View pattern. it means all the actions will be forwarded to the presenter, which will trigger the UI updates using delegates. so the view will only pass actions and listen to the presenter updates.

The components description becomes as the following:

- View: The view now consists of both views and view controllers, with all UI setup and events;
- Presenter: The presenter will be in charge of all the logic, including responding to user actions and updating the UI (via delegate). and the most important is that our presenter will not be UIKit dependent. which means well isolated, hence easily testable;
- Model: the model role will be exactly the same [6].

5.3 User Interface

The interface implementation is done dynamically, in code. Use of storyboards has been avoided but since the launch images were deprecated, a storyboard is used in this case.

The user interface designs were designed and provided by the company in which the internship was made.

5.4 Local Database

Realm database is used to storage local data in the app.

Realm Swift is the first database built for mobile. An alternative to SQLite and Core Data that's fast, easy to use, and open source it is a cross-platform mobile database solution designed for mobile applications that you can integrate with your iOS projects. Unlike wrappers around Core Data, Realm doesn’t rely on Core Data or even an SQLite back end.

The Realm Database is lightweight and highly performant, capable of handling very large data loads and running queries in fractions of a second. Based on shared live objects, it syncs data seamlessly in real-time with the Realm Object Server without the need to write networking, serialization, or object-relational mapping code. This means that applications will be able to refresh data as fast as needed to provide an enjoyable, engaging user experience. It supports the development of reactive applications due to the 'live object' nature of the database [10].

5.5 Translations

The API provides translations for several locations. A list with all the translations are saved in a local database (Realm) when the user logins into the app. The translation table is accessed every time a translation needed. For this to be achieved, when a certain text needs to be written, a "translation tag" is used (the key on the Translations table). If the translation is not available (key does not exist), the translation tag is printed like this: [translation-tag].

5.6 Notification Services

Firebase is used to process the notification service.

Firebase Cloud Messaging (FCM) is a cross-platform messaging solution that lets app reliably send messages at no cost.

Using FCM, the system can notify the app that new email or other data is available to sync. You can send notification messages to drive user re-engagement and retention. For use cases such as instant messaging, a message can transfer a payload of up to 4KB to a client app.

An FCM implementation includes two main components for sending and receiving:

- A trusted environment such as Cloud Functions for Firebase or an app server on which to build, target, and send messages.
- An iOS, Android, or web (JavaScript) client app that receives messages via the corresponding platform-specific transport service.

5.7 Client

The requests to the API is implemented with Alamofire (a third party library). The received JSONs should be parsed with Decodable or similar. Currently, when there is a need to mock an API call, a custom session manager is created and the responses are loaded from files (for testing purposes). Other ways to achieve this should be investigated since this creates the need of having extra code just for the tests to run.

5.8 Others Technologies

Google Maps API should be used.

The network layer should be built using Rxswift (with Alamofire). The version of Alamofire can be updated if needed, but in that case, the parsing of JSONs should be reimplemented.

6. Functional requirements

- **Splash Screen**

The screen is presented when the application is starting is the Splash Screen. This screen shares the company image together with a loading bar while loading application components.

Below is the activity diagram of the splash screen:

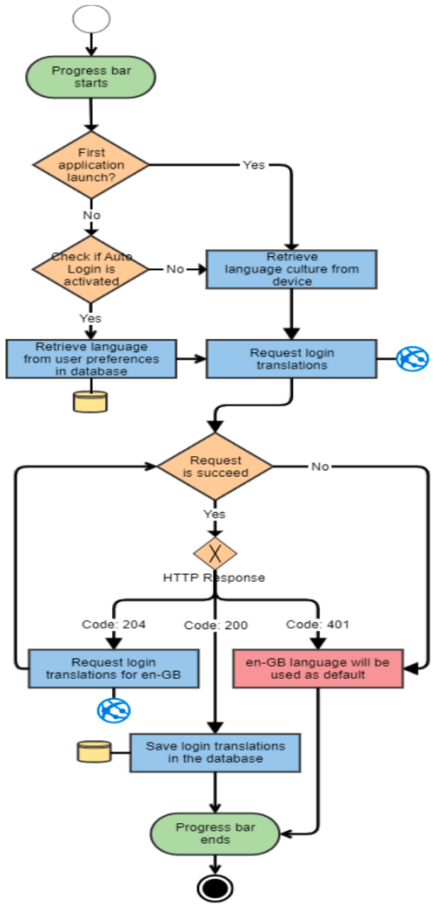

Figure 1: Splash Activity Diagram

The welcome screen appears when a user starts the application.

A progress bar is presented in order to provide user feedback regarding application's loading.

This progress bar is endless. This means that the inner progress bar has a fixed size and is always moving forward and backwards, depending on whether it reaches the end or start of the progress bar respectively. This behaviour is required because the loading time is unknown.

Launch Screen contains also a copyright text at the bottom.

Fleet Manager application starts with the Splash screen. Thus, the progress bar animation starts and application’s components begin to load.

Database creation of update is performed if required. After that translations for the Login screen are retrieved using the API request.

- ## Login Screen

After the splash screen, the login screen opens. Users should enter a valid username and password.

In the password field, an eye icon allows users to turn the password visible.

The Keep me signed in check box allows an automatic login when the token session expires.

After clicking the Login button, a loading symbol is visible, and the authentication API is called. The Login button is disabled at that moment, to avoid multiple calls to the API.

If login is successful, the app proceeds normally.

If the API returns an error, a message is presented to the user with a message.

To perform the auto login, it should save this preference on the database, and if auto login is enabled and the token hasn't expired it'll be OK, otherwise it can't do auto login.

When clicking the login button, the following request are sent:

- Post authorize
- Register notification service
- Get user
- Get translations

Below is the activity diagram of the login screen:

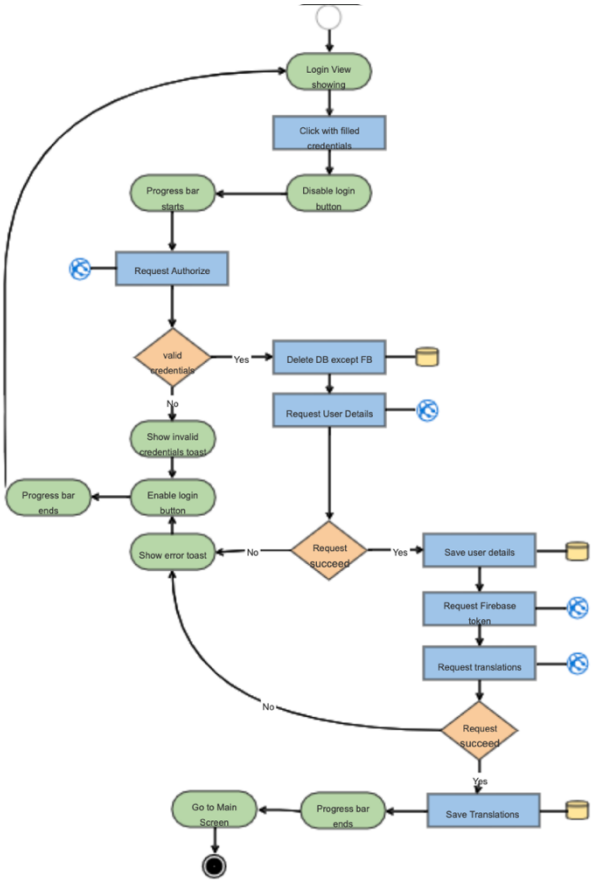

Figure 2: Login Activity Diagram

- **Home Screen**

The purpose of the Home Screen is to give users a general view of the status of the fleet, through a dashboard.

After a successful login, the home screen opens and the company logo is visible in the top bar.

In the Home Screen users are able to navigate to sub- menus like Fleet, Distribution and Nearest vehicles. A bottom tab bar is also presented to navigate through Messages, Alarms and user Account.

When opening the home screen the badges for messages and alarms counters are also updated.

Below are the permissions for home screen configuration:

“permission-summary-distribution”
Without this permission, the user cannot see the fleet distribution by country/district and the corresponding button is not visible on the Home Screen

“permission-map-view”

The Nearest vehicle option depends on this permission View Map, without this permission, the user will not see the button Nearest vehicle on the Home Screen.

“permission-alarms-view-occurrences”

Without this permission, the user will not see Alarms occurrences on the bottom navigation bar.

The user needs at least one of the following three permissions to have access to the messages screen

“permission-nav-messaging” “permission-sms-messaging” “permission-forms”

Without any of these three permissions the
button Messages will not be visible on the bottom navigation bar.

Below is the activity diagram of the home screen:

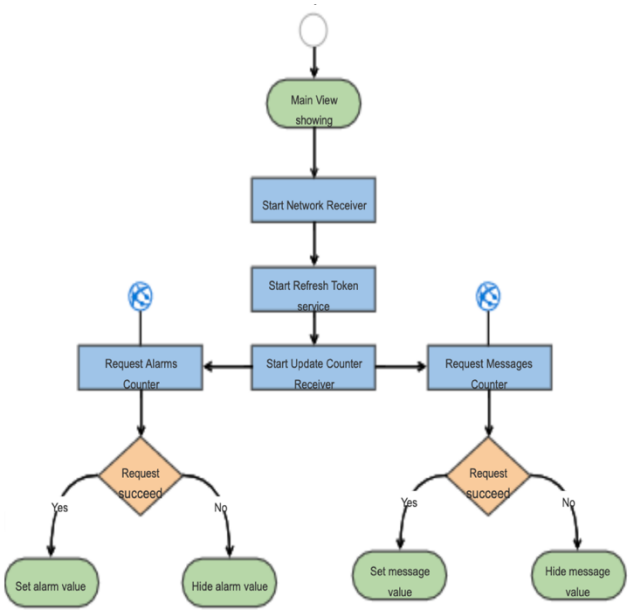

Figure 3: Home Activity Diagram

- ## Distribution Screen

Show distribution (number of vehicles, mileage) of the fleet by country and by districts, when clicking on a selected country.

If Backend returns an error or a "no content", both scenarios are handled with error message.

When a slice of a chart is clicked, the correspondent subtitle is changed with the information of that piece (mileage or number of vehicles). If nothing is selected, the total number of vehicles and mileage are set.

The colours of which the countries are represented, are retrieved from an array of colours. Since this array is created with a finite length, the colours are retrieved sequentially and if the length is maxed out, the first colours are used again.

We are using the country ISO to make a new request on the API to get the districts of that country, and we are using the location to put in the top bar.

When clicking on a country on the Distribution Country Screen, it sends via bundle the country ISO, text location, text distance, text units and number of vehicles.

When a slice of a chart is clicked, the correspondent subtitle is changed with the information of that piece (mileage or number of vehicles). If nothing is selected, the total number of vehicles and mileage are set

The app should use the country ISO to make a new request on the API to get the districts of that country, and use the location to put in the top bar.

When clicking on a country on the Distribution Country Screen, it'll send via bundle the country ISO, text location, text distance, text units and number of vehicles.

The activity diagram for both distributions is the following:

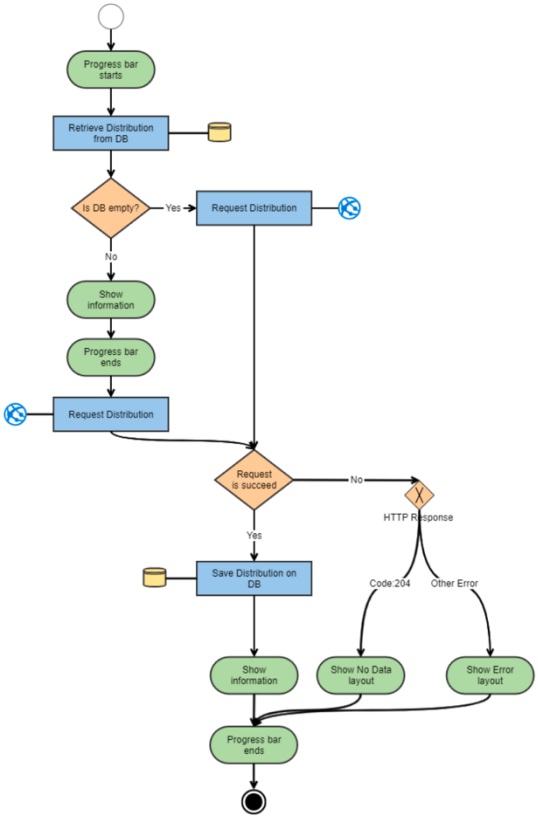

Figure 4: Distribution Activity Diagram

- **Nearest Vehicles Screen**

To achieve the Nearest Vehicle Screen, user should click in Nearest Vehicle Button from Home Screen.

To access this functionality the user has to give permission to his location, without this none of the features in this screen will work.

When user enter in the screen, should be validate if user already had granted the access to device location.

If yes, the flow proceeds normally.

If no, a dialog will be presented to the user.

The application flow proceeds normally if user allow the permission.

If user press “Deny”, a snackbar appear, explaining the impact. Feature is disable until manually activation, in phone settings, or clicking in “Allow“ button snackbar.

Snackbar is visible every time user enters the screen.

When the user already accepted the location permission and no location information available, a dialog will be visible requiring user to turn on the GPS.

If user press “Ok“ the phone settings open, to allow the device location manually.

Below is the activity diagram of the nearest vehicle screen:

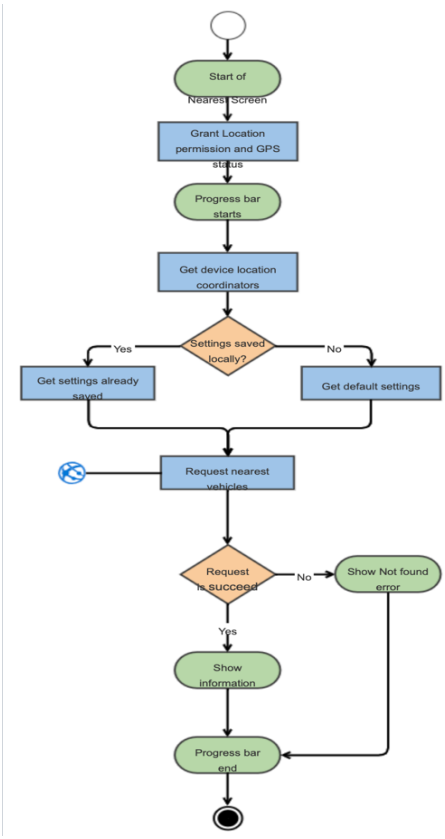

Figure 5: Nearest Vehicles Activity Diagram

- **Nearest Vehicles Screen – Settings**

When Nearest Settings button clicked, a new screen opens:

- The available options depend on user permissions;
- The options should be saved locally as a user preference;
- "Use driving time restriction rules" is a group. If it disables all subgroup items are disabled. If it is enabled, the items in the group are enable and user are able to turn on/off each one;

- “Avoid highways“ and “Avoid tolls“ are mandatory and by default with status OFF. The other options depend of the API endpoint returned data.

- If “mapProvider” is **Google** then the app should use “Vehicle class” label translation from the API to show the user.
- If “mapProvider” is **Here** the the app should use “Vehicle profile“ label translation from the API to show the user.

By default, the string to show in the drop down vehicle list is the first object of the vehicles list.

If the field “useDrivingTimeRestrictions” from API is true, the app should show the group of the following options:

- Use team driver
- Cover maximum distance
- Current driving time

When user click on “apply“, the values should be saved locally and applied in the next nearest vehicle call.

Every time user enter on the Nearest Settings Screen, the settings saved locally should be combined with the API nearest settings call, according with the following diagram.

Below is the activity diagram of the nearest vehicle screen:

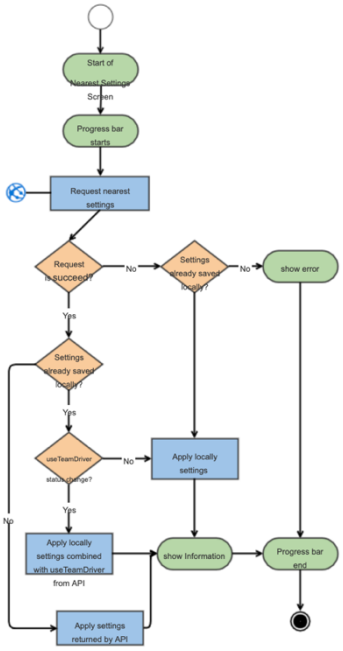

Figure 6: Nearest Vehicles Settings Activity Diagram

- **Account Screen**

The account screen must show the user profile picture, username, and company name on the top as mockup shown.

A list of actions appear:

Notifications → A new screen open when clicked Support → System call app will open when clicked Feedback → A new screen open when clicked Version → No click action

Logout → Perform the logout API call when clicked

All section area is clickable. The phone and arrow symbol are only illustrative.

Values presented in account screen are already loaded (after login)

The Support field only appear if API returns content on the field “supportContact“.

The “Notifications“ field only appear if Alarms permission allowed.

When user click on “Support“, the FleetManager app must call the system intent in order to set the support number on the system call application, keeping the backstack properly to allow the return to the FleetManager App.

The version field show the app version. The value must be dynamic. When Logout pressed:

- Delete session token
- Unregister notification service

Below is the activity diagram of the account screen:

Figure 7: Account Activity Diagram

- # Alarms Screen

If alarms permitted to the user, the icon appears in the bottom navigation bar.

A red circle appears with the number of unprocessed alarms.

A list with alarms appears.

Alarms not processed appear with a blue left label and text in bold format.

Alarms already processed appear without the blue label and with text in regular format.

A search tool and a filter tool appear on the top bar. (Search criteria: Driver name and License plate).

The search tool only looks in data already loaded by the app. Each cell includes the Date of alarm and License plate of vehicle, and Driver name (if a driver is associated with the vehicle).

If the endpoint returns an error or a "no content", both scenarios must be handled. The translations for the alarms type will be available in the general translations endpoint that is used on the login.

Below is the activity diagram of the alarms list screens:

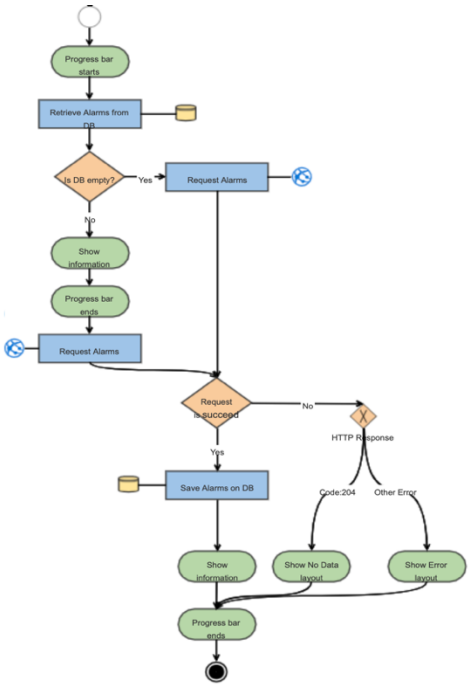

Figure 8: Alarms Activity Diagram

- **Alarms Details Screen**

When users click on an alarm cell, a new view will open with the details about that alarm.

If no driver is associated, a “-” sign appears in the Driver field.

If the alarm is unprocessed, a button will be available to process it.

The additional information displayed depends on the alarm type. Some types of alarms have no additional information whilst others have several parameters to be displayed. For this reason the information should be presented in a list.

The note is not mandatory, so the process button must be enabled even when the Notes field is empty.

When users click on the Save button, a loading view will be visible, and an API call will be made.

If the API call is successful, the view is closed, and a success message is presented to users (through a toast in Android and a dialog in iOS)

If an error happens, an error message appears to the user and the view is kept open.

If in edit mode, the “Notes” field will present the text recorded previously.

Below is the activity diagram of the alarm details screen:

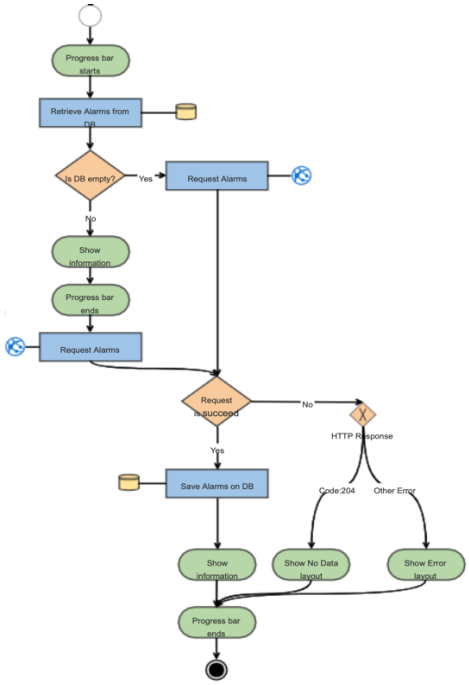

Figure 8: Alarm Details Activity Diagram

- **Messages Screen**

If messages permitted to the user, the icon appears on the bottom navigation bar.

A badge with a number of unprocessed messages will appear in a blue circle shape, located on the top right side of the message icon.

The Messages Screen is accessible from the bottom navigation bar.

The Conversation list will be loaded from the Backend, with pagination.

Most recent messages appear on top of the list. As users scroll down, the app loads older messages.

Conversations with messages unprocessed appear with a blue marker on the left side of the cell, with text in bold. (if any unprocessed message inside the group).

“status” assumes two values string: “Processed“ or “Unprocessed”.

Below is the activity diagram of the messages screen:

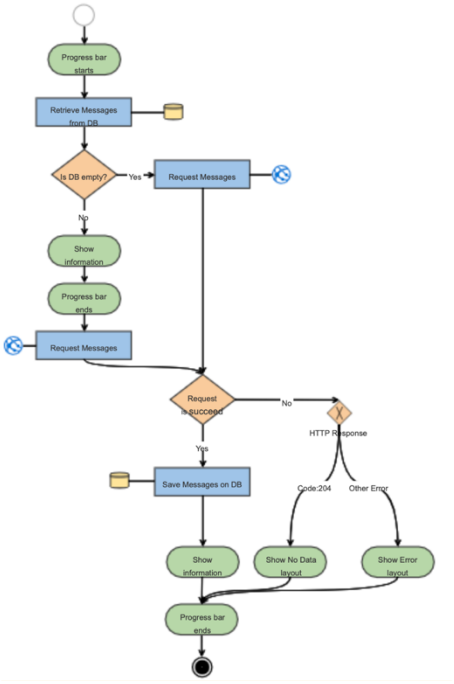

Figure 9: Messages Activity Diagram

- # Messages - Conversation details

To see all messages of a conversation, users should click on a conversation cell.

Unprocessed messages appear on a blue background.

Processed messages appear on a grey background. On the top right end side of each message, an icon is visible:

- Clicking on it shows the possible actions for the message.

- It is possible to mark messages as processed if they are unprocessed. The text visible in the action menu will be "Mark as processed".
- If the message is already processed, the text visible in the action menu will be "Mark as unprocessed".
- It is not possible to process and unprocessed messages from the office/user, located on the right side of the screen.
- After clicking on "Mark as processed", a confirmation dialog will open.
- The background color of a message changes color to grey after a successful return from the API call.
- A feedback message with the API response will be visible to the user.

Below are the permissions app should use for messages:

permission-delete-message

The Delete feature is available if permitted

After clicking on "Delete", a confirmation dialog will open.

A feedback text with the API response will be visible to the user.

If successful, the text will automatically disappear.

If an error happens, it will be possible to retry the API call (if desired by the user)

To mark a message, to delete a received message and to delete a sent message an API is called, with the message id.

- ## Messages Screen – New

To open the New message screen, users must click the "plus" button in the bottom right end side of the screen.

The list of contacts will be loaded when the view is open.

It is possible to select contacts individually, or all contacts, by clicking the "Select All" option.

The contacts selected will appear in the destination bar.

Clicking on the cross symbol or on a cell in the list, removes the contact from the destination bar.

If no contacts are selected, the "Apply button" option will be disabled.

On the top bar users can find two features:

- search tool.
- filter tool.

Search criteria → By Contact name or Device Type

- After users click on the funnel icon, a new view will open.
- Users can filter by device type.
- After selecting the contacts and clicking on "Apply", a new screen opens.
- Here, users are able to write their messages in the bottom bar.
- The Send button is disabled if the message bar is empty.
- It is possible to remove contacts from the destination bar.
- After clicking “Send” a feedback message is presented to the users.

• Fleet Screen

User should click in Fleet Button from Home Screen to consult the fleet vehicles.

The Fleet Screen shows a list of all the vehicles.

Vehicles are split by Type (bus, car, motorcycle, van, etc).

It is possible to expand or collapse each section - with the arrow icon, in the right end side of each section.

The API provides a list with all vehicles but grouping the vehicles by type and order by alphabetical criteria is to be done locally.

Each vehicle cell has a left marker with red, green or grey colour, depending on the vehicle **ignition status**.

If the vehicle is in **Immobilizer mode**: *active*, a label with red background will appear in the bottom right end side of the cell.

If the vehicle is in **Immobilizer mode:** *pending*, a label with orange background will appear in the bottom right end side of the cell.

Other information related with vehicle will be visible in the cell: License plate code, Driver name, Beginning of the daytime, Last trip finished time, Mileage and number of trips for the current day.

The flow of the map is the same, with the exception that the logic only runs when the map is loaded.

The below is the activity diagram of the fleet screen:

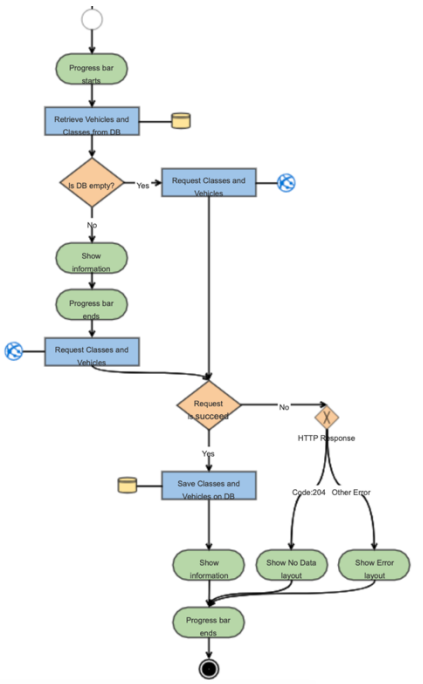

Figure 10: Fleet Activity Diagram

- ## Vehicle Details Screen

When users click on a vehicle (from the list of vehicles or from the map), a new view opens with the vehicle details information.

The app bar on the top shows the vehicle license plate and a date picker selector.

By default, the date picker selects the last 7 days:

- Data represented on the map and inside each tab is related to the vehicle activity inside the interval selected on the date picker.
- Vehicle Details Screen includes a map with the path of the vehicle and a Tab layout with the following data:
    - Vehicle details information
    - List of events
    - List of trips
    - The driving times
    - Graphs visual information

When Vehicle details Screen open, the path of the vehicle will appear on the map.

By default, the path of the last 7 days appears, with the current day marked with blue color (All other days are represented with gray color).

To allow the navigation through the days, a left and right buttons appear on the screen.

On the top right corner of the map view, a label appears with the range of selected dates.

If the user has more than one event and/or alarm at the same location, a “plus“ icon should be visible at the point.

When we click on a marker of the type alarm we have to make the below API request and show the info on an infobox.

How to make the polyline:

- Order every marker by date (events, locations, alarms)

- Create Google Polyline using the above markers

On the map we'll have buttons to iterate through the select days, this feature will have the following actions:

- Change the selected day:
    - The polyline must be split in multiple polylines in a way that the polyline will have a different color for the selected day
    - The number of markers that appear on the map depends on the current zoom and if we are on the selected day or not.
- We have 2 linear functions that relate this:
    - For the selected day we have:
        - **var percentageMarkers1Day: Float = ((99 * zoom) / 16) - (95 / 4)**
    - For the other days we have:
        - The above functions will give us the % of total markers we have to show.
        - **var percentageMarkersRestDay: Float = ((1.24688 * zoom) - 4.9375)**
        - The minimum zoom to show markers is 4, because in Android zoom goes from 1 to 20.

The below is the activity diagram of the vehicle details screen:

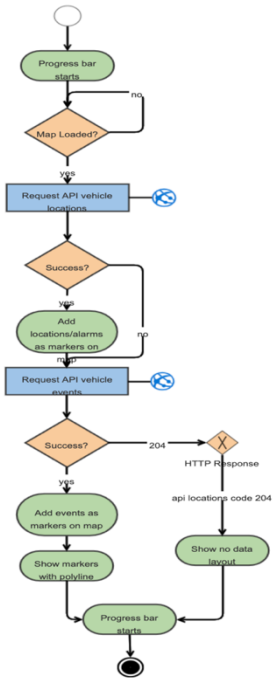

Figure 11: Vehicle Details Activity Diagram

- **Vehicle Details - Info Tab**

The Information tab contains general information about the vehicle. Missing fields must be hidden, except the driver name.

When no driver name available show ”-” .

Icons to send SMS message or call the driver will be visible only if permitted.

When users click on the SMS or call icon, the Smartphone App calls the system trying to make the call or write an SMS to that destination (phone number of the driver).

The Car details section also shows the Immobilizer status label and a clickable icon to change the status. This feature is only available if permitted.

The below is the activity diagram for the vehicle info screen:

Figure 12: Vehicle Info Activity Diagram

- **Vehicle Details - Events Tab**

The Events tab shows all the events within the selected date.

The Event details shows the details related to each event. This view is reached by clicking on a cell of the events list. The icon depends on the event type.

Location will be clickable, to show the location on the map.

The vehicle events are already loaded from API (needed to draw the path).

The app passes the Event object as an Object to the new View Controller, because we already have the necessary information.

Without this permission events tab is not visible in Vehicle screen.

Below is the activity diagram of vehicle events:

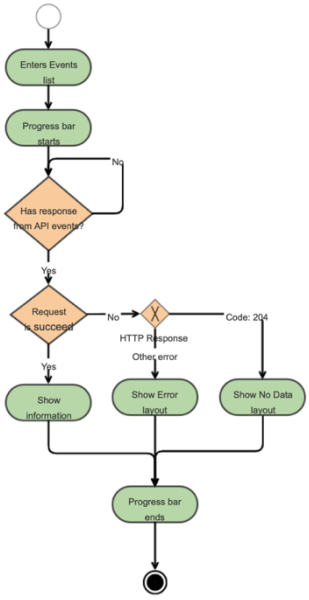

Figure 13: Vehicle's Events Activity Diagram

- **Vehicle Details - Trips Tab**

The Trips tab shows all the trips for the selected date.

The trips are presented in a scrollable list, and the trips are loaded by pagination. When users scroll down, more data are loaded from API and added to the list.

The Trips detail view shows the details related to each trip. This view is reached by clicking on a cell of the trips list. The fields without data must be hidden.

The interval of dates used to the request is the ones selected on the Calendar.

The app passes the Trip object as an Object to the new View Controller, because we already have the necessary information.

Below is the activity diagram of vehicle trips:

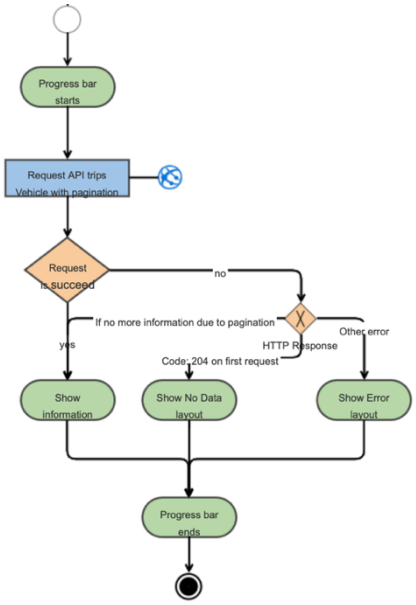

Figure 14:Vehicle's Trips Activity Diagram

- **Vehicle Details - Driving Time Tab**

The Driving Times tab shows all the clocks for the selected date. The tab is visible if permission allowed.

The drop-down contains 5 options as user desires to see the data:

- Elapsed driving time
- Remaining driving time
- Regular daily period
- Reduced daily period
- Complete driving times

The drop-down items should be set with the according translation.

The drop-down is a user preference. The driver will choose the way how he prefers to see the data.

By default the "Elapsed driving time" configuration is visible to the user.

If the user change to another visualization option, the preference should be saved locally and set in the next session.

In the “Complete driving times“ visualization option, when no data in “Beginning of the period”, “Duration of the break“ or “Amplitude“ the field should be invisible (like in the “Amplitude“ mockup example).
Below is the activity diagram of vehicle driving times screen:

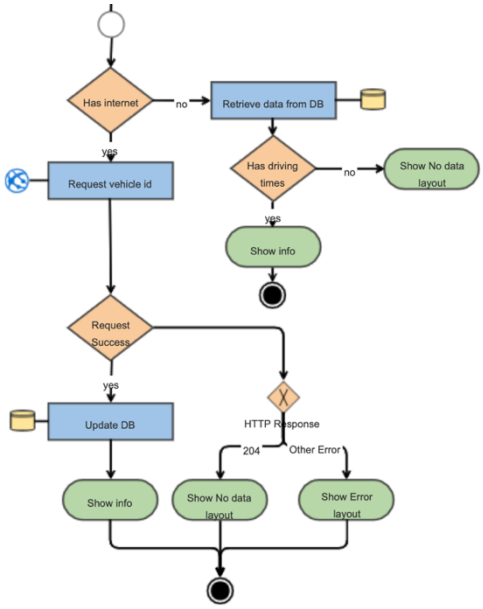

Figure 15: Vehicle’s Driving Time Activity Diagram

- **Vehicle Details - Graphs Tab**

The Graphs tab shows the graphs available for each vehicle.

Type of graphs:

- Digital graph → with on/off values. This type of graph is used to draw:
  - Ignition data;
  - Analog graph → with a range of values, used to represent:
  - Vehicle Speed
  - Engine Speed

    - Engine Temperature
    - Battery Voltage;
- Series graph → with a set of data. Is used to draw:
    - Fuel data.
    - Custom graph → to represent the 4 status provided by Tachograph data.
    - When the graph is clicked, a label must be shown.

7. Development process of the Fleet Management App

These specifications are just simple product guidelines, each time a new development started a Story was created with the expected flow, design guides and mockups and test scenarios. These stories were added to a shared Board (which will be updated while working at them) where the team performed code reviews, functional and design validation and documentation check. A shared Bitbucket repository was available where the code was committed daily, small doubts were clarified using the provided Slack Channel. A meeting took place each week to discuss the progress made, if necessary, daily meetings were arranged to discuss impediments. In that case, if there is something that prevents someone of working effectively or there is something blocking their path, a meeting was requested and scheduled according to the time zone and the team's availability.

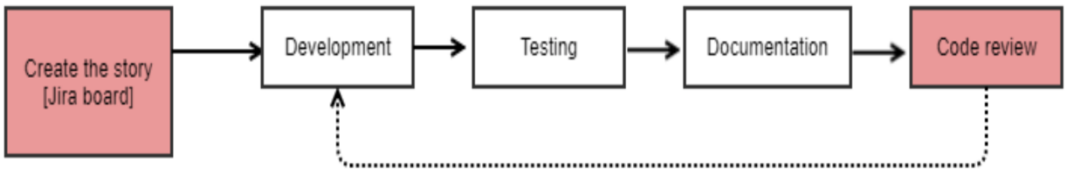

Figure 16: Development Process

7.1 Expected workflow

1. The story added in the Jira board will contain all the information to make the task possible

    1.1. Functional requirements

    1.2. Mockups, visual guidelines and icons

    1.3. Tests scenarios

2. Start of Developement

3. Tests Implementation
4. Create documentation in confluence
5. Code review

7.2 Scrum

This project used scrum methodology. Scrum is a framework that helps teams work together. Much like a rugby team (where it gets its name) training for the big game, Scrum encourages teams to learn through experiences, self-organize while working on a problem, and reflect on their wins and losses to continuously improve.

The scrum framework is heuristic; it's based on continuous learning and adjustment to fluctuating factors. It acknowledges that the team doesn't know everything at the start of a project and will evolve through experience. Scrum is structured to help teams naturally adapt to changing conditions and user requirements, with re-prioritization built into the process and short release cycles so your team can constantly learn and improve.

A scrum team needs three specific roles: product owner, scrum master, and the development team. And because scrum teams are cross-functional, the development team includes testers, designers, UX specialists, and ops engineers in addition to developers.

- Product owners are the champions for their product. They are focused on understanding business, customer, and market requirements, then prioritizing the work to be done by the engineering team accordingly.
- Scrum masters are the champions for scrum within their teams. They coach teams, product owners, and the business on the scrum process, and look for ways to fine-tune their practice of it.
- Team members have differing skill sets, and cross-train each other so no one person becomes a bottleneck in the delivery of work. Strong scrum teams are self-organising and approach their projects with a clear 'we' attitude. All members of the team help one another to ensure a successful sprint completion.

The scrum framework itself is simple. The rules, artifacts, events, and roles are easy to understand. Its semi-prescriptive approach actually helps remove the ambiguities in the development process, while giving sufficient space for companies to introduce their individual flavor to it.

7.3 Splash Screen

The welcome screen appears when a user starts the application.

A progress bar is presented in order to provide user feedback regarding application's loading.

This progress bar is endless. This means that the inner progress bar has a fixed size and is always moving forward and backwards, depending on whether it reaches the end or start of the progress bar respectively. This behavior is required because the loading time is unknown.

Launch Screen also contains a copyright text at the bottom.

when the Splash screen is loaded it is expected that the database will be created and updated. Language culture must be already saved in the database. If an error occurs retrieving the language from the endpoint, the English language (en-GB) existing in the database will be used.

Below is the user interface developed for the splash screen

Figure 17: Splash Screen

7.4 Login Screen

For the edit text's, TextFields were used in the app, which allows to show error messages and show/hide the password.

No validations were made on the textview's, everything is checked on the API side.

The last successful login username is saved in local memory.

When the translations don’t exist, the English language is used. When they are needed, they are retrieved one by one from the database.

When clicking the login button, the following request is sent:

Post authorize - Validates user credentials (username and password) and using Frotcom provider generate a valid Auth-Token that can be used to authenticate user in future API interactions.

Below is the user interface developed for the login screen:

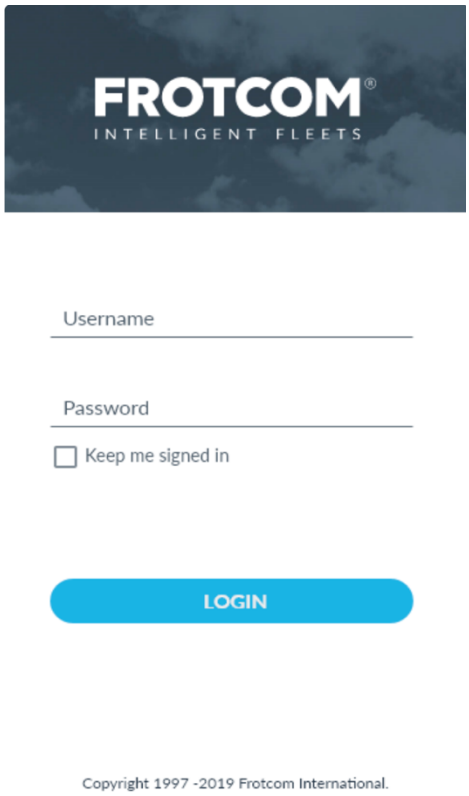

Figure 18: Login Screen

7.5 Home Screen

In this screen, the top navigation bar has no title, replaced by an Image View with company icon.

A custom button was created to avoid code duplication for each button. The icon is set in design. The button label is set dynamically in code, to allow translations.

Also, on this class, a Network Broadcast Receiver was implemented to check for connectivity changes and open a red bar saying that no internet is available. The refresh token service starts and a receiver for the badge counter is implemented.

The receiver for the counters is called when new notifications are received or when the user opens the home screen.

Below is the user interface developed for the home screen:

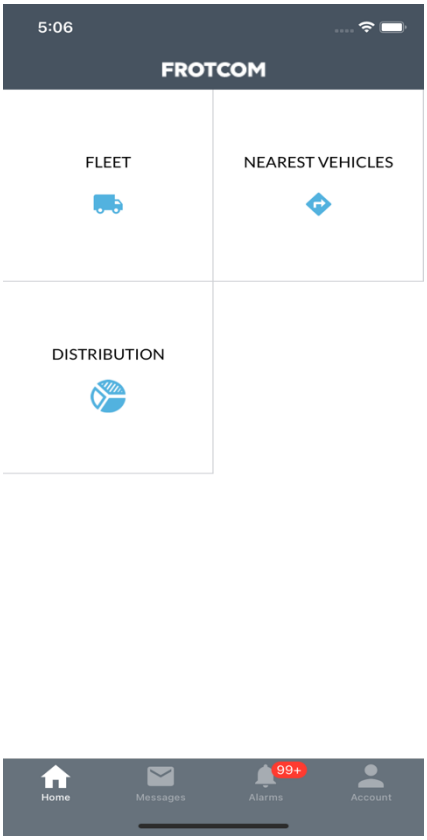

Figure 19: Home Screen

7.6 Distribution Screen

Fleet distributions are retrieved by the Country or a District of a country.

The colors of which the countries are represented, were retrieved from an array of colors. Since this array is created with a finite length, the colors are retrieved sequentially and if the length is maxed out, the first color are used again.

The app should use the country ISO to make a new request on the API to get the districts of that country, and use the location to put in the top bar.

When clicking on a country on the Distribution Country Screen, it sends via bundle the country ISO, text location, text distance, text units and number of vehicles.

When a slice of a chart is clicked, the correspondent subtitle is changed with the information of that piece (mileage or number of vehicles). If nothing is selected, the total number of vehicles and mileage are set

Below is the user interface developed for the distribution screen:

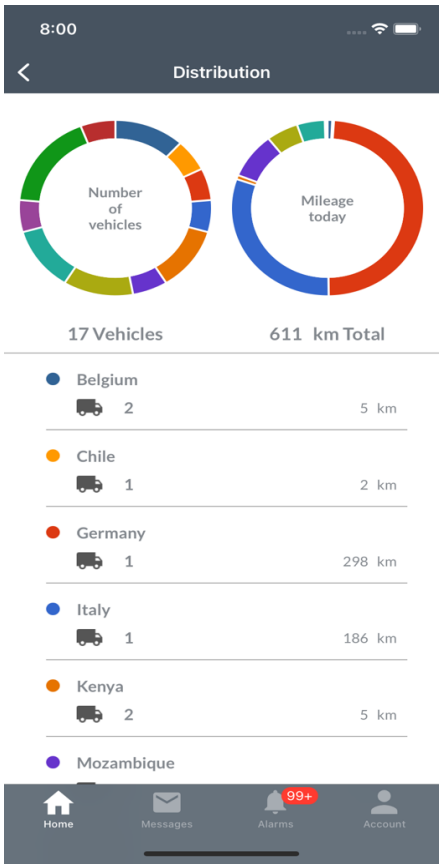

Figure 20: Distribution Screen

7.7 Nearest Vehicles Screen

When the user enters the Nearest vehicles screen, permissions to use the GPS tool will be required (first time).

The five nearest vehicles will be presented in the bottom card view (5 cards, navigated through by a horizontal swipe in the cards area).

If the API only returns less than 5 vehicles, only the number of results returned are shown.

On the top bar users can find three features:

- **Search** - Users can search by License plate number, by Driver name and by Places.
- **Pin** - The pin is used to indicate a specific location on the map and then calculate the nearest vehicles to that specific location. After clicking on the pin icon, a pin appears on the map and users will place it on the desired position.

- **Settings** - Users are able to set properties for nearest research.

When users click on the blue button (in a card), a new view opens, with the details of the path (itinerary).

Below is the user interface developed for the nearest vehicle screen:

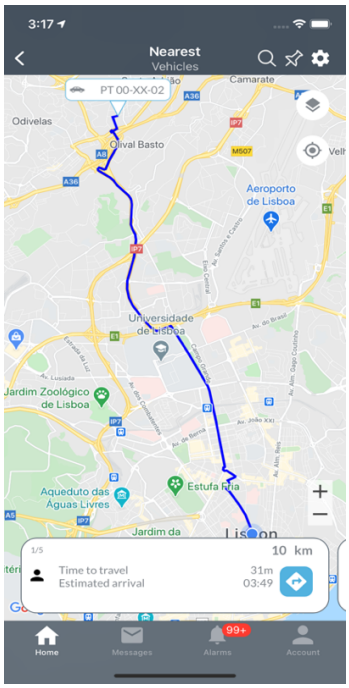

Figure 21: Nearest Vehicle Screen

7.7.1 Nearest Vehicles Screen – Settings

The API returns nearest vehicles settings:

Account map provider type (‘GOOGLE’ OR ‘HERE’)

- Vehicle class;
- Vehicle profile list.

The value of the company setting: Use driving time restrictions rules The value of the company setting: Use team drivers

Every time user enters on the Nearest Settings Screen, the settings saved locally should be combined with the API nearest settings call, according with the following diagram.

Below is the user interface developed for the nearest vehicle settings screen:

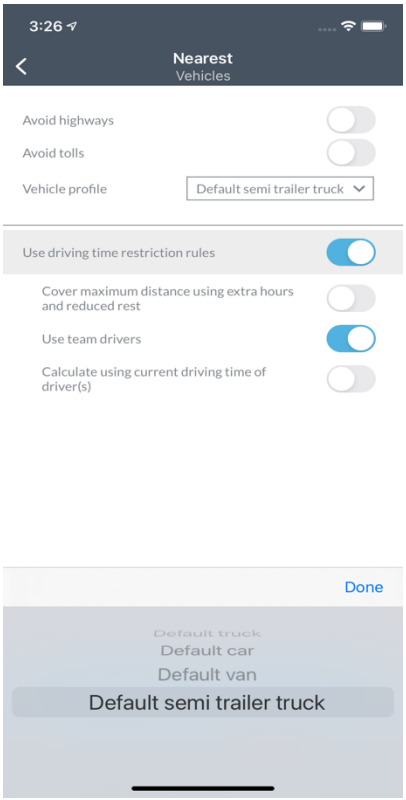

Figure 22: Nearest Vehicles Settings Screen

7.8 Account Screen

Values presented in account screen are already loaded (after login).

A list of actions appears:

- Notifications → A new screen open when clicked;
- Support → System call app will open when clicked;
- Feedback → A new screen open when clicked;
- Version → No click action;
- Logout → Perform the logout API call when clicked;
- All section area is clickable. The phone and arrow symbol are only illustrative.

Below is the user interface developed for the account screen:

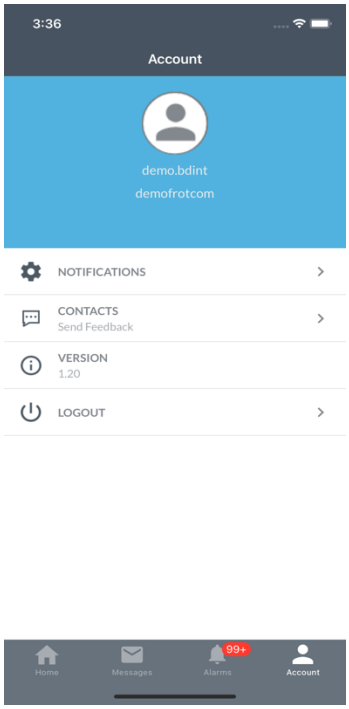

Figure 23: Account Screen

7.9 Alarms Screen

The Alarms Screen is accessible from bottom navigation bar.

In the bottom navigation bar, a red circle appears with the number of unprocessed alarms.

On the top bar users can find two features:

- Search tool;
- Filter tool.

When clicking on the search button, an input text box appears on the bottom of the screen.

Users are able to search by Driver name and by License plate.

The search tool only looks in data already loaded by the app.

When users click on the funnel icon, a new screen opens.

The Alarm type entries are provided by an API endpoint.

When users press the “Apply” button, the filter options are saved local and the view is closed.

Below is the user interface developed for the alarms screen:

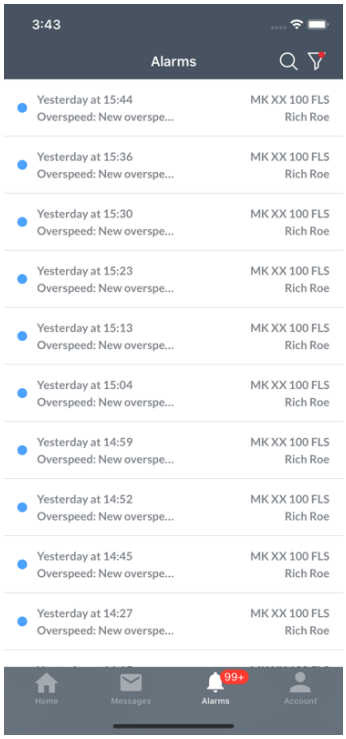

Figure 24: Alarms Screen

7.9.1 Alarms Details Screen

When users click on an alarm cell, a new view opens with the details about that alarm.

If the alarm is already processed a new field called “Notes” will be visible, with the note added.

If no note has been added, a “Click to add” text must be visible inside the field.

To edit or add a note the user will press the Notes cell. A new screen opens, to edit or add a note.

If the alarm is unprocessed, a button to process it will be visible in the bottom of the screen.

When the “Process” button is clicked, a new screen opens.

Below is the user interface developed for the alarm details screen:

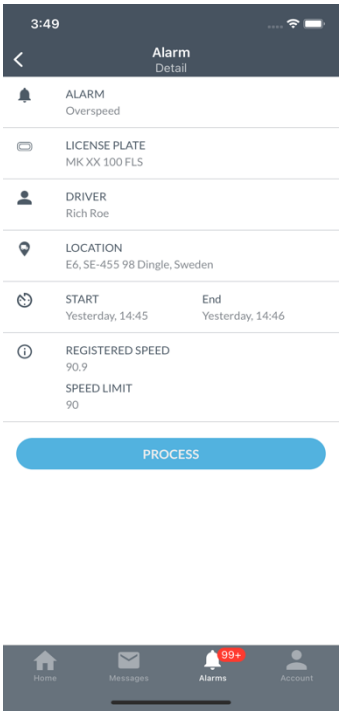

Figure 25: Alarm Details Screen

A new screen opens, to edit, add a note or process the alarm.

If in edit mode, the Notes field presents the text recorded previously.

The process button calls the API endpoint to save the alarm, if the API is successful then user goes back to the details view and the screen is updated. If the API gives error then alert view is shown.

Below is the user interface developed for the alarm update screen:

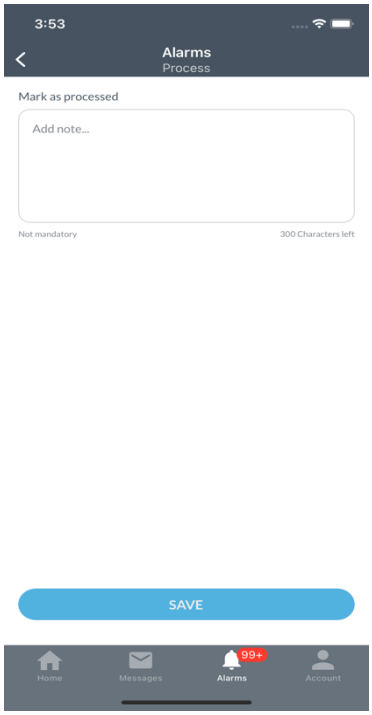

Figure 26: Alarm Update Screen

7.10 Messages Screen

The Messages Screen is accessible from bottom navigation bar.

A badge with number of unprocessed messages appear in a blue circle shape, located in the top right side of message icon.

On the top bar users can find three features:

- Search tool. Filter tool. New message.
- Search tool → The search tool (magnifying glass) is only applied to messages already loaded.
- Search criteria: Name of sender, license plate code, and text of loaded messages.

The search tool (magnifying glass) is only applied to messages already loaded.

Search criteria is the Name of sender, license plate code and text of loaded messages.

To process a set of messages, a swipe right on top of the cell will be performed.

- After swiping gesture to the right, view freeze with Mask as processed button visible.
- User are able to click in the blue button, or skip the process.
- After the button click, an API call to process the message is made.
- A success or error message appear to the user.
- To Mark as unprocessed, repeat the swiping gesture and the button should be updated to Mark as unprocessed status.

Below is the user interface developed for the messages screen:

Figure 27: Messages Screen

7.10.1 Messages - Conversation details

To see all messages of a conversation, users clicks on a conversation cell.

- Left ballons → messages with “direction” = “Received”
- Right ballons → messages with “direction“ = “Sent”

After clicking on "Mark as processed", a confirmation dialog will open.

The background color of a message changes color to grey after a successful return from the API call.

A feedback message with the API response will be visible to the user.

An API is called with the message identifier for:

- Process message
- Unprocess message
- Delete received message
- Delete sent message

Below is the user interface developed for the message conversation screen:

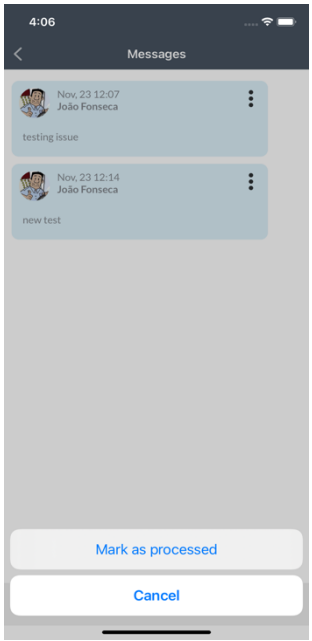

Figure 28: Conversation Screen

7.10.2 Messages Screen – New

The list of contacts will be loaded when the view is open.

It is possible to select contacts individually, or all contacts, by clicking the "Select All" option.

After selecting the contacts and clicking on "Apply", a new screen opens.

Here, users are able to write their messages in the bottom bar.

The Send button is disabled if the message bar is empty.

It is possible to remove contacts from the destination bar.

To add more contacts, users only need to click the back-arrow icon on the top bar and complete the selection. Throughout this process, the message is still vailable when users come back.

After clicking “Send” a feedback message is presented to the users.

Below is the user interface developed for the create new message screen:

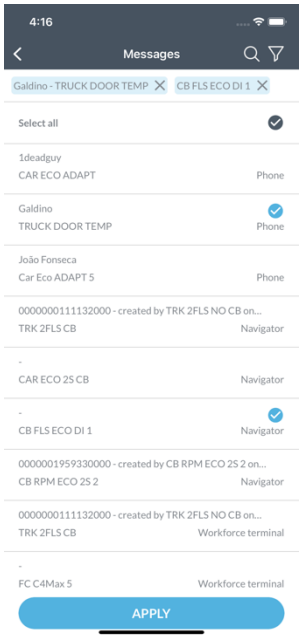

Figure 29: New Message Screen

7.11 Fleet Screen

This screen has the ability to see, search and filter vehicles on the list and on the map.

Users are able to see the vehicles in a list or on a map. This visualization preference will be saved locally. So, the next time the user comes to this screen, the last view option (list or map) will be chosen automatically.

When user enter in map mode permissions to use the GPS tool will be required.

In both modes every 1 minute everything is refreshed, meaning that if there are updates, we'll see them on the list and on the map.

Below is the user interface developed for the fleet list screen:

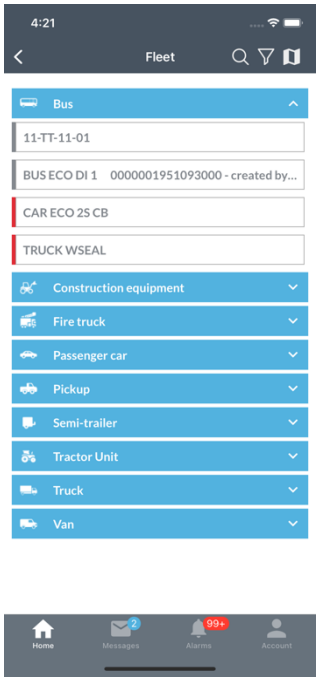

Figure 30: Fleet Screen

The flow of the map is the same, with the exception that the logic only runs when the map is loaded.

In the **map** view, vehicles are represented with an icon and the vehicle's license plate.

In the top right end side of the map, users can change the map settings (choose the type of map and activate/deactivate traffic details on the map).

Below is the user interface developed for the fleet map screen:

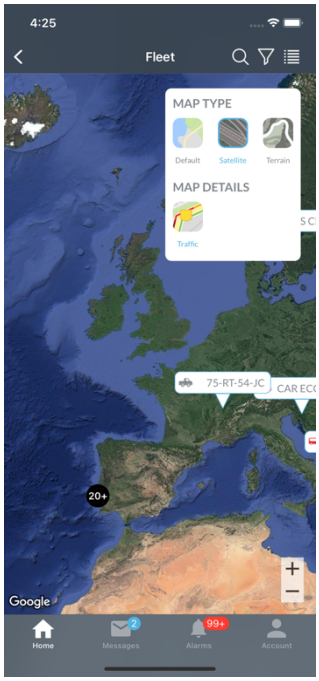

Figure 31: Fleet Screen

7.12 Vehicle Details Screen

When users click on a vehicle (from the list of vehicles or from the map), a new view opens with the vehicle details information.

Tab Layout construction is done dynamically because it depends of the user permissions.

Only permitted features are shown.

The Vehicle Details screen assumes different layout configurations:

- Map collapsed: Map and tab layout together. (default)
- Map hide: Map hide and tab layout fill all screen.
- Map expanded: Map fill the screen and tab layout hide.

To allow the navigation through the days, a left and right button appear on the screen:

left button to navigate to the previous day (disable when no more days available)

right button to navigate to the next day (disable when the user is on the current day)

On the top right corner of the map view, a label appears with the range of selected dates.

Below is the user interface developed for the vehicle details screen:

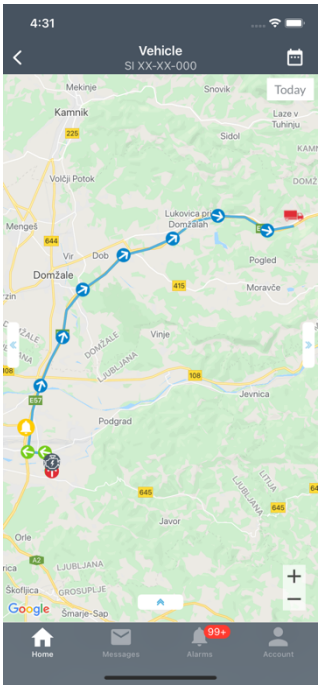

Figure 32: Vehicle Details Map Screen

The Date Range Picker feature is available at the top right. When clicking it, a calendar (Material Date Range Picker) opens with the current day marked with a circle and without the possibility of choosing days in the future.

At the beginning, the save button is disabled until the user selects a valid range (< 7 days). If the user fails to do it, an alert dialog will show up with an error message.

If the user clicks the back button or the “X” at the top left, the selection will be discarded.

Below is the user interface developed for the date selection calender screen:

Figure 33: Vehicle Details Map Screen

The icons drawn on the map must be clicked and an info-box will open providing more details about

When the “plus” marker clicked (multiple events and/or alarms at the same position)

Below is the user interface developed for the vehicle info box on map screen:

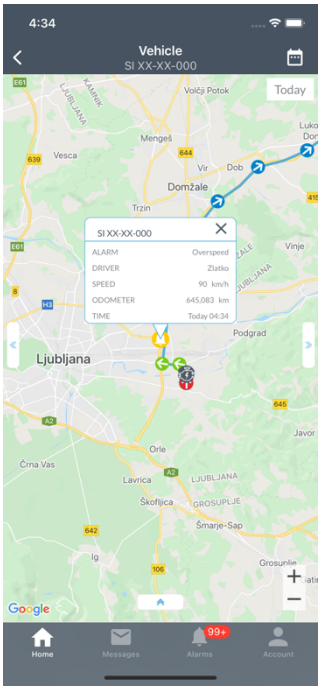

Figure 34: Vehicle Details Map Screen

7.12.1 Vehicle Details - Info Tab

The Information tab contains general information about the vehicle. Missing fields are hidden, except the driver name.

The Car details section also shows the Immobilizer status label and a clickable icon to change the status. This feature is only available if permitted.

If the immobilizer feature is not permitted for the vehicle, no label and icon will be visible.

If the immobilizer feature is permitted, the label and icon will be visible. The icon is clickable, according to the immobilizer status.

The icon click behaviour shows a dialog to activate the immobilizer status.

The request is not made if the password entered is empty or doesn't match the regex:

[0-9a-z]{4,}

An API call is done when users click the OK button.

Below is the user interface developed for the vehicle info screen:

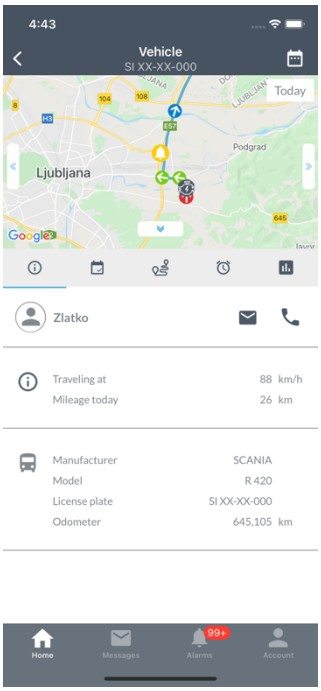

Figure 35: Vehicle's Info Screen

7.12.2 Vehicle Details - Events Tab

The Events tab shows all the events within the selected date.

The vehicle events are already loaded from API (needed to draw the path).

Observers are implemented to observe the change in the selected dates and events. According to the observers, the screen is updated automatically.

Below is the user interface developed for the vehicle event screen:

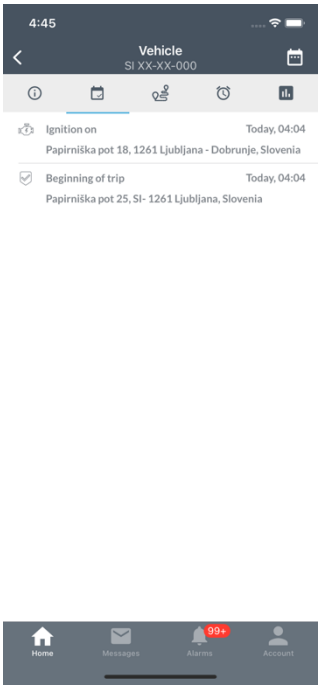

Figure 36: Vehicle's Events Screen

The Event details shows the details related to each event. This view is reached by clicking on a cell of the events list. The icon depends on the event type.

Location is clickable, to show the location on the map.

The app passes the Event object as an Object to the new View Controller, because we already have the necessary information.

Below is the user interface developed for the vehicle event details screen:

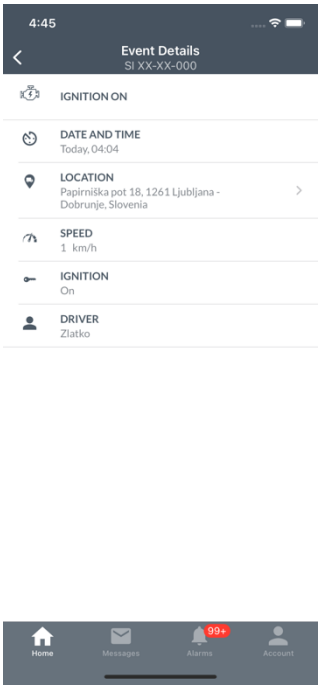

Figure 37: Vehicle's Event Details Screen

7.12.3 Vehicle Details - Trips Tab

The Trips tab shows all the trips for the selected date.

When the user enters in the screen, the trips are loaded form the API endpoint and the screen is updated.

Observers are implemented to observe the change in the selected dates and new API is called according to the observers and the screen is updated automatically.

Below is the user interface developed for the vehicle trips screen:

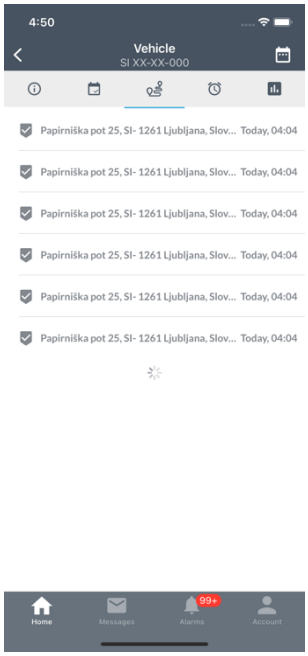

Figure 38: Vehicle's Trips Screen

The app passes the Trip object as an Object to the new View Controller, because we already have the necessary information.

The Trips detail view shows the details related to each trip. This view is reached by clicking on a cell of the trips list. The fields without data are hidden.

Below is the user interface developed for the vehicle trip details screen:

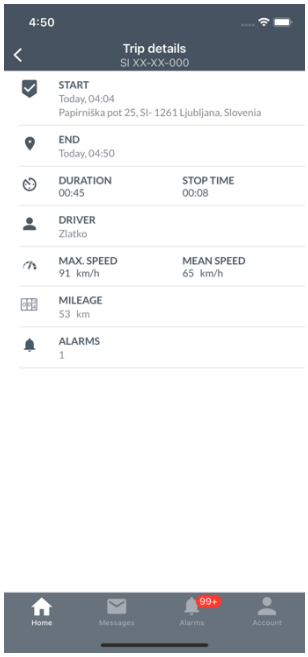

Figure 39: Vehicle's Trip Details Screen

7.12.4 Vehicle Details - Driving Time Tab

The Driving Times tab shows all the clocks for the selected date. The tab is visible if permission allowed.

- The green clock means that the extra hours are not used yet.
- The gray circle means that the extra hour is already spent.
- The green circle means that the user didn't use the extra allowed yet
- The red circle means that the user already used the extra.
- The "data resource" is represented by a label.
- The background color depends of resource type.

Below is the user interface developed for the vehicle driving time screen:

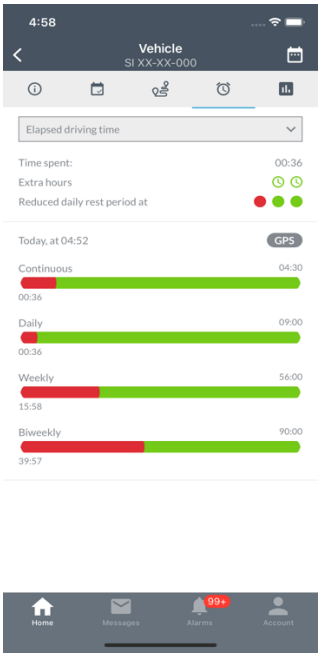

Figure 40: Vehicle's Driving Time Screen

7.12.5 Vehicle Details - Graphs Tab

The Graphs tab shows the graphs available for each vehicle.

Type of graphs:

- Digital graph → with on/off values. This type of graph is used to draw:
    - Ignition data;
    - Analog graph → with a range of values, used to represent:
    - Vehicle Speed
    - Engine Speed
    - Engine Temperature
    - Battery Voltage

Below is the user interface developed for the vehicle graphs screen:

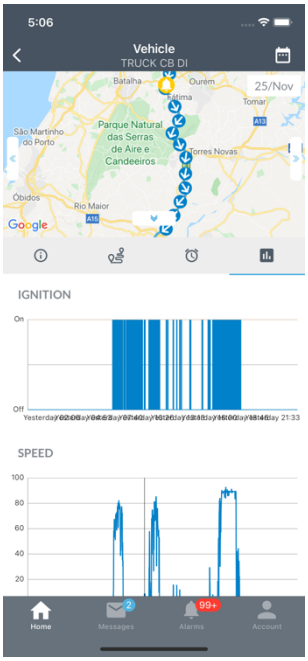

Figure 41: Vehicle's Graphs Screen

- Series graph → with a set of data. Is used to draw:

- Fuel data.
- Custom graph → to represent the 4 status provided by Tachograph data.
- When the graph is clicked, a label must be shown.

Below is the user interface developed for the vehicle graphs screen:

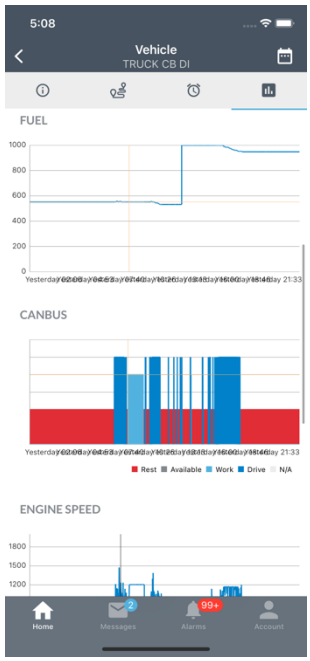

Figure 42: Vehicle's Graphs Screen

8. Testing

The test cases scenarios are exactly described in the development phase.

They are based on following:

8.1 Integration tests:

- Make successfully mocked request and verify what it does if success.
- Make error request and verify what it does.

8.2 Automatic tests:

- Verify that everything appears.
- Verify that the data that appears is coherent with the mocked data.
- Verify actions on clicked items.

8.3 Unit tests:

To test the logic of presenter classes, unit testing approach was used.

Generally, tests should cover:

- Core functionality: Model classes and methods and their interactions with the controller
- The most common UI workflows
- Boundary conditions
- Bug fixes

The acronym FIRST describes a concise set of criteria for effective unit tests. Those criteria are:

- Fast: Tests should run quickly.
- Independent/Isolated: Tests should not share state with each other.

- Repeatable: You should obtain the same results every time you run a test. External data providers or concurrency issues could cause intermittent failures.
- Self-validating: Tests should be fully automated. The output should be either “pass” or “fail”, rather than rely on a programmer’s interpretation of a log file.
- Timely: Ideally, tests should be written before you write the production code they test (Test-Driven Development).

Following the FIRST principles will keep your tests clear and helpful, instead of turning into roadblocks for your app.

Unit Testing in Fleet Management App

The Test navigator provides the easiest way to work with tests; we use it to create test targets and run tests against the app.

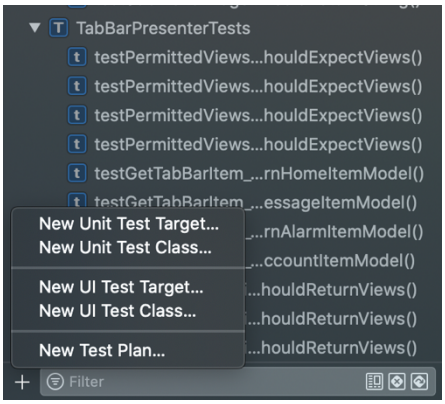

Figure 43: Unit Test Navigator

The way of how the unit tests are written and debugged is following:

A test method’s name begins with “test”, followed by a description of what it tests.

It’s good practice to format the test into given, when and then sections:

Given: Here, app sets up any values needed.

When: In this section, app executes the code being tested

Then: This is the section where app asserts the result a developer expects with a message that prints if the test fails.

Below is the code demonstration of one Unit Test function:

```
class TabBarPresenterTests: XCTestCase {

    var sut: TabBarPresenter!

    private var userModel: UserModel {

        let preference = PreferenceModel(timezone: "",
                                         language: "",
                                         mileageSpeedUnit: "",
                                         fuelConsumptionUnits: "",
                                         fuelUnits: "",
                                         decimalSeparator: "",
                                         groupSeparator: "",
                                         useDrivingTimeRules: "")

        return UserModel(username: "",
                         companyName: "",
                         preference: preference,
                         permissions: [],
                         photoURL: "",
                         supportContact: "")

    }

    private let keyValuePersistenceMock = KeyValuePersistenceMock()

    override func setUp() {

        super.setUp()

        sut = TabBarPresenter(keyValuePersistence: keyValuePersistenceMock)
        keyValuePersistenceMock.showError = false

    }

    func testPermittedViews_withUserPermissions_shouldExpectViews() {
        // given
        let userPermissions: [PermissionObjectModel] = []

        //when
        let views = sut.exposedPermittedViews(with: userPermissions)

        //then
        XCTAssertGreaterThan(views.count, 0, "Views computed are nil")
    }
```

Figure 44: Unit Test

After running the tests, the following success message is shown:

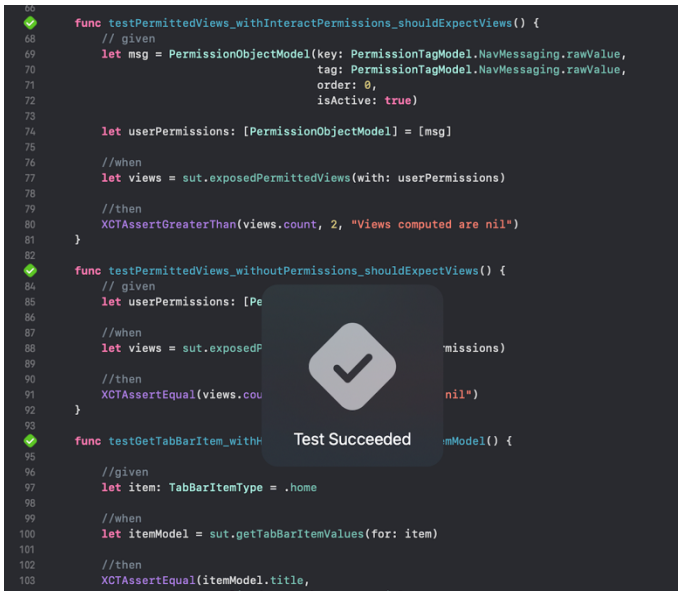

Figure 45: Unit Test Success

9. Conclusion

This project turned out to be a valuable experience in software development. The theory of how the development should be carried out and the things that can go wrong in the development process were experienced firsthand. Many little challenges were encountered and the understanding of why managing software development is not an easy task grew during the course of this project.

The core functionality of vehicle details screen maps with custom markers was successfully implemented in the final prototype of fleet management application. Other functionalities: nearest vehicles, messages, alarms etc were also implemented and tested to be working correctly. The scope of the features and quality of the features were not compromised to be scaled down to finish the project. Internal tests and client testing were used to eliminate the bugs. Redmine and Jira were used to create tickets for issues.

The successful factor in the project was the iterative nature of the scrum methodology. Having sprints planned out and iteratively implementing the features allowed fast adaptation to changing scope and requirements. Following the scrum methodology ensured that a working feature was produced and any changes or add-ons to the existing feature could be handled easily.

Software development is a complicated process and developing a full software product for a client, gave valuable lessons in managing time, understanding and capturing user requirements, defining scope, estimating task completion, designing for user satisfaction, evaluating and thorough testing and mitigating the changing requirements, which will contribute to a better management of the software development.

We all know that it is difficult, if not impossible, to predict the future. Generally, there are so many different variables to consider, many of them very difficult to predict, that make the whole exercise quite complex.

Predicting the future of fleet management is this type of exercise. If you think someone in the world already knows how it will be, think twice. From truck manufacturers to telematics companies, from large technology companies to fleet managers with 30 years of experience, nobody knows for sure.

New technologies are developing quickly, companies are eager to get to know autonomous vehicles, the Internet of Things, Artificial Intelligence or Linked Vehicles for the usage of logistics and fleet management systems will adopt with them.

However, in one respect everyone is unanimous: the future of fleet management will change; and the change will be quick. A plexus of technologies will soon enter the world of fleet management. And its impact will certainly be enormous.

10. References

1. Wasserman, A. I. 2010. Software Engineering Issues for Mobile Application Development. FoSER '10: Proceedings of the FSE/SDP workshop on Future of software engineering research. ACM.
2. https://developer.apple.com/xcode/
3. https://docs.swift.org/swift-book/
4. https://realm.io/docs/swift/latest/
5. https://firebase.google.com
6. https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ff649571(v=pandp.10)?redirectedfrom=MSDN
7. https://www.tailwindtransportationsoftware.com
8. https://www.smartfleetsolutions.com
9. https://tripmastersoftware.com
10. https://www.razortracking.com
11. https://swivelsecure.com
12. https://www.parceltrack.de/en/
13. Jones, M., and Marsden, G. 2006. Mobile Interaction Design. John Wiley & Sons, Ltd.