

# **Development of “Track” Android Application Using “Ford AppLink” For Real-Time Car Data Storage**

A project final report submitted to The University of Essex for the degree of  
BSc (Hons) Information and Communication Technologies  
In the Faculty of Science and Health

2017/18

Evaldas Senavaitis

1402039

School of Computer Science and Electronic Engineering

Professor Nick Zakhleniuk

Professor Luka Citi

## *Abstract*

With every year fuel reserves in the world become smaller and air pollution is getting worse, making individuals and companies increasingly aware of how much fuel their cars use every day. That is where "Track" comes in, the application implements Smart Device Link API which connects likes of Ford and Toyota, opening the huge market to the car world. As well as using Android and Google, it gives application majority of the phone users around the world and while using Google APIs it provides many tools for monitoring application interactions and allows developers to see statistics of how many active users' application has.

Mobile application "Track" uses Bluetooth for connectivity to vehicle infotainment system (AppLink) and it provides users to review their recorded journeys with details like, how long they travelled, how much actual fuel was burned and the averages of speed and rpm. As well authentication details are recorded in the app to be available for the user like VIN and full car model details. Backlogs of trips are available as well, for later revision and it is possible to set reminder notifications for services needed on the vehicle.

Everything in the application is presented in friendly user way and having all this information application can indicate driving behaviour and patterns allowing to see efficient routes and give advice how to efficiently drive in the future.

## *Acknowledgements*

I would like to thank my supervisor, Professor Nick Zakhleniuk for his never-ending support and guidance, helping me to achieve the highest quality of work on the reports and objectives of the project as well as his confidence in my ability and knowledge of the system project undertakes. Nick provides students with the best possible supervisor experience imaginable, striving that students get all the resources and help they need and being friendly and professional at the same time during the meetings. I would like to thank Professor Luca Citi for his insightful feedback on the project planning software solution and overall amazing experience during the oral examination.

I must also thank, Ford Motor Company and specifically Mr Paul Elliot for providing support and insider knowledge about AppLink features and future developments during meetings at the university. As well as introducing me to Mr Leonard Lu the IT Manager at Ford Motor Company during the Open Project Day and his advice in pursuing carrier at Ford Motor Company.

# Table of Contents

<b>Abstract .....</b>	<b>1</b>
<b>Acknowledgements.....</b>	<b>2</b>
<b>Chapter 1 Introduction .....</b>	<b>5</b>
1.1 Outline of the project and the report .....	5
1.2 Aim of the project .....	5
1.3 Objectives.....	5
1.4 Requirements.....	6
1.5 Project schedule and deliverables.....	6
1.6 Project evaluation .....	6
<b>Chapter 2 Background research and review .....</b>	<b>7</b>
2.1 Driver assistance systems .....	7
2.2 Self-Driving cars .....	7
2.3 Driver distraction .....	8
2.4 HMI systems .....	8
2.5 "On road" driver assistance mobile application .....	8
2.6 Development of eco-driving and safe-driving components using vehicle information .....	9
2.7 Programming language Java.....	9
2.8 Layout design understanding .....	9
2.9 Android Studio and Android SDK .....	10
2.10 Fords' AppLink and SYNC3 system with emulator.....	10
2.11 Literature review conclusions .....	11
<b>Chapter 3 Design of the application layout screens .....</b>	<b>12</b>
3.1 Structure of application .....	12
3.2 "Connection and Data Review" button page concept.....	12
3.3 "Data Review" concept screen .....	13
3.4 "Trip" function concept screen layout for saved routes .....	13
3.5 "Vehicle Details" concept layout design .....	14
3.6 "Records" concept layout design .....	14
3.7 "User Settings/Notifications" concept layout design with actual push notification design.....	15
3.8 "Lock Screen Activity" application design requirement.....	15
3.9 "Human Machine Interface" screen template design .....	16
3.10 Saved data structure design .....	17
<b>Chapter 4 Current implementation of the project .....</b>	<b>18</b>
4.1 Getting started .....	18
4.2 "AndroidManifest.XML" .....	18
4.3 "Main Activity" Connection and Data Review button page.....	19

4.4 "Screen After Login" Data review page .....	21
4.5 "Select Trips" journey review page .....	24
4.6 "Car Details" vehicle details page .....	25
4.7 "User Notifications" settings class of notification system .....	28
4.8 "SdlService" Heart of the application and SmartDeviceLink.....	30
Chapter 5 Project planning .....	43
5.1 Methodology adopted .....	43
5.2 Sprints overview .....	43
5.3 Version Control for the software .....	44
Chapter 6 Conclusions, evaluation and future work .....	46
6.1 Evaluation and conclusions .....	46
6.2 Images of current implementation results .....	47
6.3 Future work.....	51
References.....	52

# Chapter 1 Introduction

## 1.1 Outline of the project and the report

This report features the development cycle and all the steps necessary to produce a mobile application on an Android operating system with Fords' AppLink technology. The functionality of the product will be to minimize driver distraction and at the same time, gather all possible data vehicle can provide to inform the user about vehicle status, service reminders and journey details. To accomplish this, many hours went into research and development to understand the dangers and solutions to driver's distraction and identify the best possible functions for a vehicle mobile application.

The product provides the user with the following functions and services; Trip Back Logs for simple journey review after it was finished, service notifications allowing the driver to set reminders for vehicle maintenance, vehicle records giving the ability to show vehicle identification number (VIN), odometer reading and button or voice activated command for trip recording.

The report will follow the lifecycle of the application containing design, implementation and evaluation process.

## 1.2 Aim of the project

Because of smartphone ownership and ever-growing sales, the project aims to develop an Android application for drivers to minimize their distraction and providing safety on the roads with some useful feedback to the driver about their made trips. Therefore, to follow the steps of a complete Agile software development cycle and the ways Android development tools can be used for such application design and development, the project hopes to offer the best solutions and support possible for future Ford developers and vehicle users.

## 1.3 Objectives

The main objectives of the project are to:

- I. Understand and research the effects of distraction a mobile phone creates while driving.
- II. Carry out a literature review and do market background research of similar applications that are available or in development.
- III. Collect requirements for the project and recognise the main functions of a friendly and familiar mobile application.
- IV. Design and develop a mobile application that meets requirements of the project, detailed in the following section
- V. Evaluate and conclude the final product if the project has met the objectives and requirements needed.

## 1.4 Requirements

The minimum requirements of the project are:

- I. Use Ford's AppLink technology (SmartDeviceLink)
- II. Provide Bluetooth or USB connection to the vehicle HMI system.
- III. Deliver exciting and attractive features to improve the driving experience.
- IV. Offer an interface to display gathered data from the vehicle to the user.

## 1.5 Project schedule and deliverables

In software development, management is an essential part of excellent implementation and delivery of the product. Planning the lifecycle of the development is vital to establish and achieve grounds on what project will be built upon, especially in the early stages.

For this project, a development plan was chosen with Agile mixed methodologies, to achieve maximum efficiency. The lifecycle of this project was split into 1 - 2 weeks Sprints, depending on complexity and amount of tasks sprint might have, with a possibly a tangible result after sprint was successfully finished. Due to unaccountable and unforeseen difficulties sprints might have been extended up to a maximum of 2 weeks period. Every 1-week period must have had a minimum of 15 hours put into them and every 2-week sprints must have had a minimum of 30 hours, to achieve optimal results.

At the end of the project, a fully functional mobile application running Android operating system with Ford's AppLink API will be delivered and minimal requirements of such application should be met.

## 1.6 Project evaluation

Following list of criteria were used as a benchmark to assess the completion of the project objectives:

- I. Project Schedule: To evaluate the usefulness of the schedule by examining the problems faced in the research or development and accordingly reviewing how feasible it was.
- II. Agile, Design and Implementation: To deliberate the effectiveness of the chosen methodology, design and implementation results.
- III. Project Evaluation/Conclusions: Final product evaluation to determine if the requirements have been met and the personal thoughts about the project with possible future development additions and improvements. Following with difficulties and learning curves during the project.

## Chapter 2 Background research and review

### 2.1 Driver assistance systems

“Three decades of driving assistance by Klaus Bengler” [1], in this paper he established the importance of mobility of transport in the modern world. Many delivery services rely on such applications to provide deliveries as fast as possible. In addition to that, it allows people to travel greater distances than ever before to reach their job site or their offices.

These come with an economic, environmental and social price, like fatal accidents, pollution and cost of maintenance of their vehicles [2].

To achieve these, people must first establish better safety environment on the roads, DAS was introduced in the early 1990s with their first invention being ABS [3] which prevents wheels of locking up in the emergency type breaking. Earlier records can be found of using ABS for aircraft and motorcycles starting from 1920-60s.

Next big development was in 1995, DAS introduced electronic stability control (ESC) [4] which was later been made as a legal requirement in USA, EU and rest of the world.

Nowadays such systems like Navigational systems and various sensor technologies come as a standard for vehicles. Sensor systems were first being sold in the early 1990s [5] as this allowed drivers to approximately know the distance between their car and obstacle in front or rear.

Such systems were studied by Engles and Dellen paper [6] and founded that non-locals are more likely to cause accidents than locals because they do not use Driver Assistance systems. One such system is Navigation allowing drivers to let the system decide their route and thus freeing up mental resources for attention awareness on the road. Also gives driver fastest route to the destination, saving fuel, time and stress for the driver.

All this long distance driving forced people to come up with a solution that would save energy for drivers during and after the trips. The solution was Adaptive Cruise Control (ACC) [7]. The first release of this system only allowed 30 km/h and above, but nowadays system advanced so much that even in traffic jams vehicles can maintain speed and distance to the vehicle in front of them [8]. Using ACC and sensor technologies together drivers are notified of an impending collision and if it is unavoidable, the system would automatically apply breaks to reduce damage. After EU recognition of ACC system, it is a required legislation that all trucks that are produced for European market come with such system as standard.

With all these developments in the automotive industry, a new advanced driver assistance system (ADAS) category emerged. As all major car manufacturers now give options on ADAS systems, that give drivers more information and more control over their vehicles [9].

### 2.2 Self-Driving cars

The best system for the driver would be the one that completely removes the driver and these are called self-driving cars. The most known brand is Tesla in autonomous vehicle category and for now, all their production vehicles are Level 2 autonomous system, which still requires driver attention and input to the road [10].

On the other hand, the most advanced self-driving car is made by Renault called Symbioz which is level 4 in the autonomous system category [11], although it is not a production model just a concept. This is because the technology for production is too costly and current legislation in most countries do not allow to drive a



car without having your hand on the steering wheel. All known car brand work with governments to change this law, to be more compliant with new car development for the future.

Having familiarised with levelling category of autonomous vehicles, level 4 and 5 vehicles are most likely to fall into interconnected vehicles, such cars would connect to the major systems of the roads and between each other. Having such information of roads and vehicles, traffic systems and controls could be introduced, using these systems travel times would significantly be reduced in cities and all major roads, because all vehicles would know all their destinations and would adjust accordingly for everyone needs. Accidents would be almost non-existent and traffic jams reduced to zero.

## 2.3 Driver distraction

Systems discussed above help drivers to behave on the road safer and be better, but some help to complete everyday tasks of the modern world and before moving to such systems we must consider what driver distraction is and what types are there.

Driver distraction is an act when a driver is involved in other activities while driving [1]. In 2015 distracted drivers caused over 3,400 deaths and 39000 injured in the USA alone [12]. Studies helped to remove miss concepts about phones being only distraction driver can have and is now established that there are many types of distractions [13]. Types are split into 4 different categories, visual, cognitive, auditory and biomechanical distractions [14]. Sometimes these distractions can be called secondary tasks and most of the time it involves multiple distractions to cause one secondary task.

However, these tasks can be measured by demand and willingness of the driver to engage with a distraction. Most tasks are talking, texting, eating, applying makeup, fiddling with the stereo, entertainment system, mirrors or navigational system as of this takes attention from the task of safe driving.

To conclude driver distraction-related accidents most, occur when inattention happens followed by an unanticipated event.

## 2.4 HMI systems

Human-machine interfaces are systems that help drivers minimize distractions as much as possible and provide quality of life necessities of the modern world. Systems are mainly menu based interfaces most likely with touchscreens [15].

With a high demand for drivers to have their phones to be an extension of their vehicles, Android and Apple IOS systems developed their own respected solutions. Android Auto and Apple CarPlay, these systems, in theory, provide same benefits and function as their counterparts. Allowing to extend users phones to vehicles Head Unit or HMI, both companies strive to provide as much applications as possible to their respected systems, but clear winner at this is Android in the number of applications available, on the other hand, Apple has very strict approval system and only most polished and best apps get approved. Although many popular apps are available on both systems and major car OEMs strive to provide both experiences in their HMIs, but some do have exclusive partnerships with Apple, one example would be Mercedes, as all their new cars come equipped as standard Apple CarPlay support [16].

## 2.5 “On road” driver assistance mobile application

This publication covers an Android application that is used to monitor cars different parameters for the driver to be aware of his driving habits and possible improvements he can make to be safer and more economical on the road [17].

To achieve these improvements application, have various features that help with economical gear changing, better fuel economy etc. Some of these features are “Eco-shift” which gives indicator when it is optimal time to change gear, “Fuel Efficiency calculator” giving driver their average fuel consumption, “Traffic Idle” shows how long driver has spent in traffic with an engine turned on. Another one is “Fault Detection” allowing the application to read and receive ECU, DTC fault codes, which are then matched against known codes in the database to identify a failure.

Furthermore, publication indicates that this application requires OBD-II port adapter to fully utilize functionality, needing this hardware device, application potential is very limited to almost non-usable as the common driver would not want to buy or add more special equipment just to use the app.

## 2.6 Development of eco-driving and safe-driving components using vehicle information

This 2-page [18] paper gives a brief overview of the mobile application, which focuses on eco-driving and safe-driving components. All of this is achieved by collecting data from the onboard computer and sending it to the phone over a Bluetooth connection.

Developers of the system also want to aid inexperienced drivers by changing their driving style and behaviour using all possible data provided by OBD-II port.

In the first page, it is discussed “Vehicle-IT convergence Platform”, why they are developing such system and what they want to achieve with it and development of low-cost IT vehicle solution which is the main part of their application.

The second page establishes what application elements of their system are, features of eco-driving and safe-driving. Looking deeper into these features Eco-driving focuses on CO2 emissions, fuel consumption, idle times and other driving habits that are reflected in collected data. Safe-driving features collect data on acceleration, braking, speeding etc. The system uses all that data to determine and inform the driver about driving style and safety on the road.

## 2.7 Programming language Java

Android is based on Java programming language which used its own application programming interfaces (APIs) that are top-level interfaces for the system. Language and syntax are almost identical, but Android has its own specific libraries that standard Java applications do not have.

One example would be Android Runtime (ART) that is used instead of Java virtual machine (JVM) or Dalvik as a runtime environment [19], being custom made Android Runtime environment has performance and battery life improvements over JVM.

Another would be Extensible Mark-Up language (XML). This code is like HTML and is used for design and key components of the system to identify and describe. XML is easy to read and is straightforward and without any complex queries [20]. A prime example and heart of build would be “Androidmanifest.xml” as this file contains build parameters, vital components, list of all activities (classes) and all requirements (permissions) for application to function.

## 2.8 Layout design understanding

Android uses “Material Design”, which includes tools and resources developed by Google. Launched with Android 5.0 [21].

Material Design components can be used on lower versions as well, but its resources were designed for 5 and higher. These tools and resources are available for free and included in Android Studio Environment, having this access to the ever growing database allow developers to always stay up to date with latest animations and icons.

Research on design elements for mobile applications [22] analyses how designers commonly place logos and menus in familiar or eye-catching places to develop a friendly user interface for easy navigation. Elements mentioned in “Application research of interface design element analysis for mobile platforms” and “Material Design” guidelines and theory supports the way the user interacts with some application elements.

Both Meis paper and “Material Design” agree that Dialogs and Lists are key design elements that all applications must have. Dialogs inform users about specific tasks and may contain critical information, requires a decision or involve multiple tasks. Includes types of alerts, simple menus and confirmation dialogs.

## 2.9 Android Studio and Android SDK

Android Studio is developed and released by Google, having very familiar look to IntelliJ IDEA software system, meaning developers cut down learning time with a new tool as their layout is almost identical, or if they do not like Android Studio, they have an option to install plugin of Android to the IntelliJ IDEA software giving best of both systems.

For the project, Android Studio was chosen as a primary build tool, because it comes with all Google supported interfaces and APIs like “Material Design” additional tools and resources already built in. Also coming with automatic activity (class) creator, helping developer save valuable time by not needing to write activities from scratch.

When using Android studio, application development never been easier, with features like design canvases with drag and drop functionality, ever updating Android SDK for most up to date resources and the best testing environment of Android simulators, allowing developer to pick and choose versions of Android, phone models with different configurations in RAM and screen sizes. All this coming with a minimal stress on the development machine and allowing more than one test to be carried out in the environment, compared to other Android development tools.

## 2.10 Fords’ AppLink and SYNC3 system with emulator

Development API and testing tools provided by Ford and are easily downloadable from their developers site, although registration to the site is required to download SYNC3 Emulator tool, site contains all guides of how to set up and start working with their system and as a requirement to run emulator tool it needs 64-bit Ubuntu 14 operating system, as all other versions of Ubuntu might have connection problems.

AppLink is a specially developed API, now called SmartDeviceLink (SDL) created by Ford Motor Company it strives to create standard API for connected vehicles. SDL allows multi-system connectivity and communication between a mobile phone and vehicle HMI [23], by sending various remote procedure calls (RPCs) to change behaviour or appearance in HMI of the vehicle [24].

SYNC and SYNC3 are integrated in-vehicle communications and entertainment systems that allows users to interact with SmartDeviceLink features, additionally having compatibility with Android Auto and Amazon Alexa, it is one of most advanced communication and entertainment systems for vehicles today. Currently running on QNX operating system from BlackBerry Limited [25].

## 2.11 Literature review conclusions

After an extensive literature review of source materials, it was clear that with ever-increasing driver amount every year car manufacturers and third-party companies try to keep roads as safe as possible. OEMs do it with new integrations of advanced driver assistance systems, third-party companies do it with the help of various mobile applications or hardware devices. Ford with their all-new SYNC3 hardware and AppLink technology can provide much needed friendly experience with a vehicle, additionally other major car manufacturers will try to copy or join Ford's group just like Toyota and Mazda does, with their Gold Partnership in SmartDeviceLink, because they can rebrand SmartDeviceLink technology into their own brand name just like Ford does. The benefit of SDL is the removal of hardware layer, SmartDeviceLink eliminates the OBD-II need in any future vehicle, but to see it coming true, the system still needs few years for development and major support for manufacturers to make it a standard for all vehicles.

While researching similar vehicle applications in the marketplace, it was obvious that almost all of them have the same or very similar functions, differences were the only possibility of connection, storage of data and design of application activities, this is very clear when examining this project as most applications were an inspiration for making this project, only difference would ease of use and OEM branding which would bring much more attention to the user rather than third-party application.

## Chapter 3 Design of the application layout screens

This part of the report focuses on design solution of the mobile application with the help of early prototype activity design features and layouts. Each part of the design will emphasise minimal requirements of the project and guide developer throughout implementation stage with the main idea of each activity and minimalistic layout design.

### 3.1 Structure of application

Before implementation, a getting started mobile application is provided by Ford from their developer site with their company minimal requirements, that being SmartDeviceLink API installed with all needed classes and Lock Screen activity. The application provided minimal activities are SdlService, SdlRouterService, SdlApplication, SdlReceiver, MainActivity and LockScreenActivity, these are vital to a mobile application running SmartDeviceLink connectivity and services.

### 3.2 “Connection and Data Review” button page concept

This page is the first one user will see after launching an application. The layout consists of the large background image of application name and company logo with two buttons on the bottom side of the screen. First button being “Data review” if an application has been used before it will activate another activity in application structure to launch new activity and redirect the layout to new one, the second button being “Connection” it is used to establish a link between mobile phone and the compatible vehicle which will immediately launch LockScreenActivity allowing user to interact with HMI unit of the car.



*Figure 1. Design of initial application screen*

### 3.3 “Data Review” concept screen

The “Data Review” page is main activity users will be using while the mobile application is not connected to the vehicle, it was arranged in two large sections with a navigation menu on the left side corner for the rest of application main functions. Both large sections represent recorded journey data user took and saved for future review, first part displays valuable data about the journey who’s recorded using SmartDeviceLink services, second part displays an image or map view of the journey displaying polyline if GPS data was recorded.

On the left side, activity has navigation menu where all main functions rest, this menu is activated by a button press on the top left side. This navigation design solution is a staple of all applications for Android and must feature if the application has more than one function.

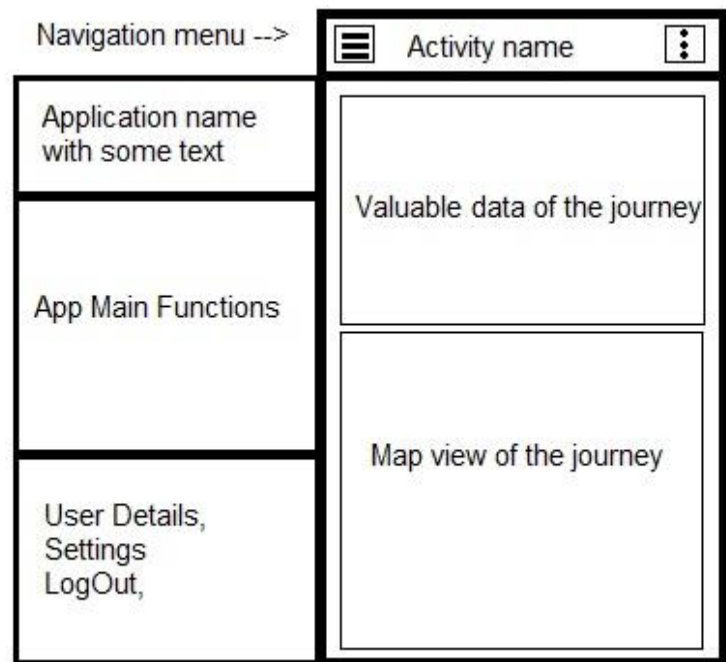


Figure 2. Design of the main application screen

### 3.4 “Trip” function concept screen layout for saved routes

This page is where all journeys will be visible for the user to review. Activity should consist of Android ListView [26] with a function to press on selected “Trip name/Date” and show very similar or the same page as “Data Review”. Possibly having navigation menu from previous activity or working “Back” button to return to “Data Review” page. In this page, the user should be able to delete records as well.

Activity page should work from internal memory storage, as all journeys will be stored locally on the device for security purposes. No other application has access to these files as well as user itself, because Android only allows reading these files for an application unless specified otherwise.

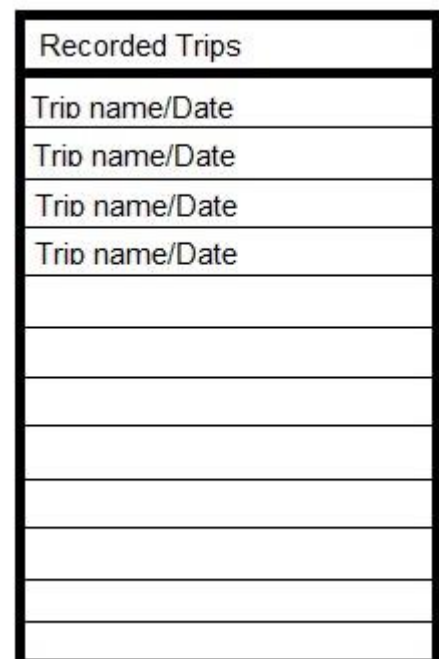


Figure 3. Design of list view for the application

### 3.5 “Vehicle Details” concept layout design

The “Vehicle Details” page has main details of last connected vehicle. Such details are updated once the application is connected.

The design consists of various text fields with automated completion based on car information and only a few areas where the static text will be used. Such areas are “VIN:”, “Car Tire Pressure Info”, “Tire info” named as tire location on the vehicle, “Fuel Percentage:”.

All this data will be read with the help of SmartDeviceLink services and stored locally on the device because the mobile application would not be able to display such information without a connection. Additionally, data will be stored in a single file, meaning if a user connects to other vehicles last information will be replaced with a new one keeping security intact.

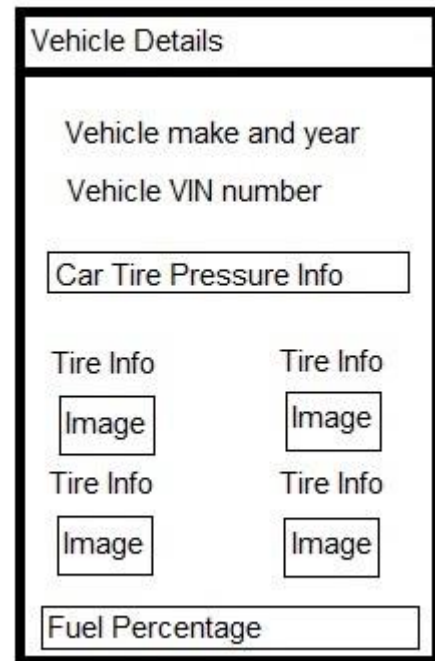


Figure 4. Design of the vehicle details screen

### 3.6 “Records” concept layout design

The “Records” page has few main details regarding journey highest and monthly recorded information such as speed, distance, Fuel Consumption as well as readings of total fuel consumed, total distance driven etc.

The design consists of two main large sections, top being monthly recorded data, this top section will use fragment [27] to be able to change with different months data if users wishes, by pressing the button on the top right corner of the screen. The bottom part of the screen is dedicated to highest achieved records, this section is read from internal storage file which is updated if upcoming journey records are higher than a last known highest.

Note: all unique data is handled separately for highest records.

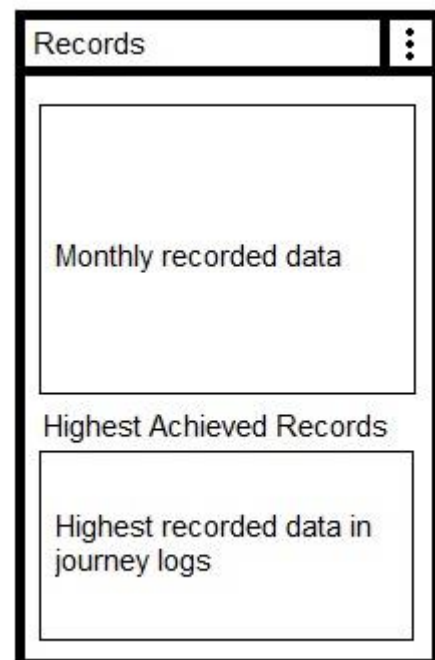


Figure 5. Design of the records screen

### 3.7 "User Settings/Notifications" concept layout design with actual push notification design

The "User Settings/Notifications" page is designed to allow the user to set notification settings for a fuel top up reminders, oil service and tire inspections for regular maintenance. This page is made in such way that user would be able to set and see their current reminders with mobile application always reminding how much is left until next inspection or top up. The complete integration of the activity, page system would only show push notification for the user when current readings are coming very close to the set settings by the user, meaning notifications would not be intrusive to the user experience.

Activity pages design consists of two main parts, the top part being notification settings environment where the user can set and change notification settings or do not set them at all, the bottom part design consists of setting active notifications with the ability to manually check how much is left to selected notification deadline.

Push notification itself consists of two main parts, the name of notification and message text field, with application image and name at the top of pop up.

Notifications/Settings page

Text Box

Notifications field to manage settings

Actual Settings

Figure 6. Design of the notification settings screen

Device settings shortcuts
Name of the notification
Message text field
Other device settings

Figure 7. Actual design of the push notifications

### 3.8 "Lock Screen Activity" application design requirement

The "Lock screen activity" is a vital part of mobile applications that are designed for vehicle market. This activity is activated when the application is connected to the vehicle and present at all time while the connection is live.

The main purpose of such screen is to prevent distractions to the phone and solely focus on the road and interact only with the cars HMI. Activity is design in such way that all incoming calls would be redirected to cars head unit where the driver could answer with steering wheel buttons or at a head unit of the car, all messages should be silent to prevent distractions.



### 3.9 "Human Machine Interface" screen template design

This screen in design is one of the Fords templates developers can use with 3 different parts and display information. Human-machine interface screen is the main screen users will interact with connected application besides dedicated menu button for additional functions.

The mobile application will use layout called "graphic\_with\_text\_and\_softbuttons" having split screen into three parts. First and the biggest allocation on the left side, large application graphic as the name implies, second is on right top side displaying application name and two text lines for additional information. Lastly right bottom part is where the main function of the application lies with two main buttons to launch ("start") function and finish ("stop") button being dynamic and staying at one place to minimize user confusion. With one place left for future developments and keeping space open, lastly a static button of the menu to access additional functions of application like vehicle details page of the application displaying tire information for the user on the go.



Figure 8. Ford template, image from SmartDeviceLink website.

Menu screen consists of a list of functions, displaying action on the same layout of original screen layout or adjusting the layout to the one that would suit best to display required information.



Figure 9. Menu screen in list type. Image from SmartDeviceLink website.

Note: human-machine interface can show only 3 lines of text and any more is limited by Ford layouts.

### 3.10 Saved data structure design

The data structure for the mobile application is stored based on an emulator at ".data/user/0/application.project.name/..." for this project application having few functions will need structured file folders for each function to store important data files.

To start with the application has "Final" folder where function "Trip" finalised data will be stored, then in structure application must be storing images or maps of each trip, based on implementation selection with the name of "GPS\_data" folder.

Finally, the structure will end with "Notes" folder for user notification settings and current reading for set settings.

These folders are only created when the application is used for the first time and not created with the installation.

Note: All application created folders outside of "files" folder will get the name "app\_'Name created'" to identify a system that these folders are created after the application was used for its functions.

## Chapter 4 Current implementation of the project

This chapter discusses in detail, functional parts of the design and current implementation of the mobile application based on design and requirements. During topic of the chapter, each will discuss very detailed implementation solution at each Java class.

### 4.1 Getting started

To start implementation of a mobile application with SmartDeviceLink services, developers will need to download "getting started" application from GitHub or Fords developer site with API already added [28]. Getting everything working is somewhat complicated, meaning close attention to SmartDeviceLink guide is necessary. Next, to start testing application, developer have few testing solution solutions, first one SYNC3 emulator from Ford, second download HMI with Core from GitHub [29] which will be generic with limited functionality or lastly for testing "Manticore" can be used. Manticore is cloud-based testing environment provided by SmartDeviceLink Company [30].

### 4.2 "AndroidManifest.XML"

The manifest file is the settings for application and device permission acknowledgements. Android manifest specifies minimal SDK version and target version on what application runs the best, followed by device hardware permissions such as Bluetooth, internet, network state, USB access, read and writes of external storage as well as phone state, all enclosed in "<use-permission />" clause.

```
<uses-sdk
    android:minSdkVersion="8"
    android:targetSdkVersion="26" />

<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.INTERNET" />
<!-- Required to check if WiFi is enabled -->
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />

<!-- Required to use the USB Accessory mode -->
<uses-feature android:name="android.hardware.usb.accessory" />

<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />

<application
    android:name=".SdlApplication"
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="Ford Track"
    android:theme="@style/AppTheme">
```

Figure 10. AndroidManifest.XML display of permissions and SDK.

Followed by "<application />" clause which specifies a mobile application name, icon, display label and main theme. Inside "<application />" all activities of application are specified (Java classes) and such clauses like <service/> for SmartDeviceLink router services as well as <receiver /> for USB, Bluetooth actions, receiver

being vital to service clause because "router.startservice" is inside <intent-filter/> when application try to connect it will search for nearest SmartDeviceLink router service.

```
<service
    android:name=".SdlRouterService"
    android:exported="true"
    android:process="com.smartdevicelink.router" />

<receiver android:name=".SdlReceiver">
    <intent-filter>
        <action android:name="com.smartdevicelink.USB_ACCESSORY_ATTACHED" /> <!--
        <action android:name="android.intent.action.BOOT_COMPLETED" />
        <action android:name="android.bluetooth.device.action.ACL_CONNECTED" />
        <action android:name="android.bluetooth.adapter.action.STATE_CHANGED" />
        <action android:name="sdl.router.startservice" />
    </intent-filter>
</receiver>
```

Figure 11. AndroidManifest.XML service and receiver clauses for SDL connection

Each activity inside "<application />" clause must have the name of the java class e.g. ".MainActivity", a label that shows on running application and theme stated.

```
<activity
    android:name=".ScreenAfterLogin"
    android:label="Screen after Login"
    android:theme="@style/AppTheme.NoActionBar" />
<activity
    android:name=".selectTrips"
    android:label="selectTrips"
    android:theme="@android:style/Theme.Holo.Light.DarkActionBar" />
```

Figure 12. AndroidManifest.XML Activities examples.

### 4.3 "Main Activity" Connection and Data Review button page

Starting application first activity is called main because it is the first one created for the project and as design implicates it is the first screen users can interact with. A class having 3 main methods, first for Android most important is "onCreate(...)", this method specifies key details and layout configuration for activity at first launch. Best practices tell developers to have only one "setContentView(...)" method for layout specification.

After that "connect(...)" method follows, which checks at the point of button press what TRANSPORT build configuration follows, right now 3 possibilities are available, "MBT", "TCP" and "LBT".

- MBT – Multiplexing Bluetooth
- LBT – Legacy Bluetooth
- TCP – Transmission Control Protocol

If first build option is "MBT", which starts SDL receiver class and "queryForConnectedService" method inside SdlBroadcastReceiver class placed in SmartDeviceLink API library, selecting this option is designed for a fully published application using the Bluetooth connection. "TCP" and "LBT" are used for testing purposes and start regular intent of SdlService class with a proxy of services if such option is set.

The final required method is "login(...)" which waits for button press and on activation it will start ScreenAfterLogin class which is the main screen of the application where journey data can be reviewed.

Button click does not have a listener as usual java implementation, Android can use on click events with correlating object to start method and it is specified inside "activity\_main.xml" file under the text.

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:background="@drawable/rectangle_button"
    android:layout_marginBottom="49dp"
    android:layout_above="@+id/connect_but"
    android:layout_alignStart="@+id/connect_but"
    android:textColor="#FFFFFF"
    android:textSize="17sp"
    android:text="Track Data"
    android:onClick="login"
/>
```

Figure 13. Android layout text file displaying: onClick option available for Android

Activity\_main.xml file is layout specific file with close design specification having image, text and two buttons followed each other. The layout is made in Relative layout [31] and used a background colour instead of the large image as specified in the Design chapter for simplicity.

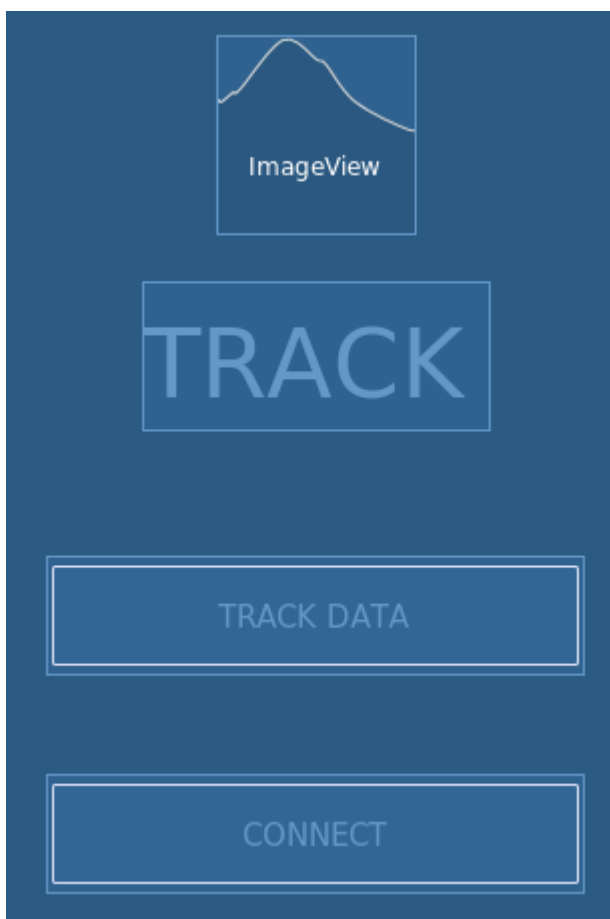


Figure 14. Current design implementation of the login screen

Finally, more methods can be found inside this class, but they are only for future expansion functions and have no functionality or meaning.

## 4.4 "Screen After Login" Data review page

This public class starts off like any other Android class having "onCreate(...)" method with the layout, toolbar [32] and navigation menu stated.

```
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_screen_after_login);  
    Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);  
    setSupportActionBar(toolbar);  
    getSupportActionBar().setTitle("Most recent Trip");  
}
```

Figure 15. Android activities main method.

First thing class asks is to check "Final" folder inside internal storage for any files and display their names in an array list, after that navigation drawer layout is set and drawer created with right placement in the main activity\_screen\_after\_login.xml layout.

```
fileTitles = getDir( name: "Final", Context.MODE_PRIVATE).list();  
System.out.println(Arrays.toString(fileTitles));  
drawer = (DrawerLayout) findViewById(R.id.drawer_layout);  
ActionBarDrawerToggle toggle = new ActionBarDrawerToggle(  
    activity: this, drawer, toolbar, "Open navigation drawer", "Close navigation drawer");  
drawer.addDrawerListener(toggle);  
toggle.syncState();
```

Figure 16. Displaying drawer widget code implementation

The final stage for navigation drawer is to specify its "View" for selected items.

```
NavigationView navigationView = (NavigationView) findViewById(R.id.nav_view);  
navigationView.setNavigationItemSelectedListener(this);
```

Figure 17. Code snippet of navigation view listener

Finally, "onCreate(...)" method calls "mainScreen(...)" method to fill in the main layout with the latest information from array list file.

"mainScreen(...)" method states String, TextView and WebView objects to be used for data review screen. The method works very primitively for Java, but it was selected as implementation because it always works and is not that CPU heavy as it appears to be. Primitivity in this solution is that method gets and reads the latest file from internal storage "Final" folder, reads it line by line and at each one having "if" statement, if true and line have data, it will fill in TextView or Webview objects.



```

public void mainScreen(String fileTitle) {
    String Map_name;
    TextView view;
    WebView myWebView;
    try {
        File iner = new File(getDir( name: "Final", Context.MODE_PRIVATE), fileTitle);
        int y = 0;

        List<String> data = FileUtils.readlines(iner, encoding: "UTF-8");
        String latitude = "", longitude = "";
        for (String line1 : data) {
            y++;
            if (y == 1) {
                view = (TextView) findViewById(R.id.showSpeed);
                view.setText(line1 + " " + "Mph/h");
            }
            if (y == 2) {
                view = (TextView) findViewById(R.id.showRPM);
                view.setText(line1 + " " + "Rev/min");
            }
        }
    }
}

```

Figure 18. Display of current implementation of main screen method for journey review screen

Objects are found inside layout by ID, and all such visible objects must have an ID to be identified by the application.

The final part of the method "mainScreen(...)" is to use Geocoder [33] to reverse GPS coordinates to actual street address. This is done using Geocoder Android class which has functions to convert coordinates to the actual location, this action is possible to be done back and forth.

```

Geocoder reversegeo = new Geocoder( context: this, Locale.ENGLISH);
List<Address> geo = reversegeo.getFromLocation(Double.parseDouble(latitude), Double.parseDouble(longitude), maxResults: 1);
Address fetched = geo.get(0);
StringBuilder strAd = new StringBuilder();
for (int i = 0; i < fetched.getMaxAddressLineIndex(); i++)
    strAd.append(fetched.getAddressLine(i)).append(" ");
System.out.println(strAd.toString());

```

Figure 19. Code snippet of the current implementation of geocoder.

Additionally, class has few automated methods created like "onBackPressed", "onOptionsItemSelected", "onOptionsItemSelected", all handling various interactions with the device clicks on the screen.

A last and final method in this class is "onNavigationItemSelectedListener(...)", it handles navigation view item clicks as mentioned before, though the process in this method is very simple having items IDs to match with "if" statements and starting the appropriate activity.

```

public boolean onNavigationItemSelectedListener(MenuItem item) {
    // Handle navigation view item clicks here.
    int id = item.getItemId();

    if (id == R.id.trip) {
        Intent listViewIntent = new Intent( packageContext: ScreenAfterLogin.this, selectTrips.class);
        ScreenAfterLogin.this.startActivity(listViewIntent);
        Log.i( tag: "Content", msg: "Select Trips Loaded");
    }
}

```

Figure 20. Current implementation and example of navigation item selection method

```

DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer_layout);
drawer.closeDrawer(GravityCompat.START);

```

Figure 21. Displaying that drawer must close.

Note: drawer must always close when the item is selected.

As class has drawers inside, developers must think carefully about the layout of activity, for example, this project java class layout consists of three different layout files besides drawer, which is not important as it can be created automatically.

First layout file must support android widget drawer layout and include another layout file as no more objects can be placed in the first layout configuration.

```
<include
    layout="@layout/app_bar_screen_after_login"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />

<android.support.design.widget.NavigationView
    android:id="@+id/nav_view"
    android:layout_width="wrap_content"
    android:layout_height="match_parent"
    android:layout_gravity="start"
    android:fitsSystemWindows="true"
    app:headerLayout="@layout/nav_header_screen_after_login"
    app:menu="@menu/activity_screen_after_login_drawer" />
```

**ndroid.support.v4.widget.DrawerLayout>**

Figure 22. Layout XML file snippet displaying additional layout and support for navigation view

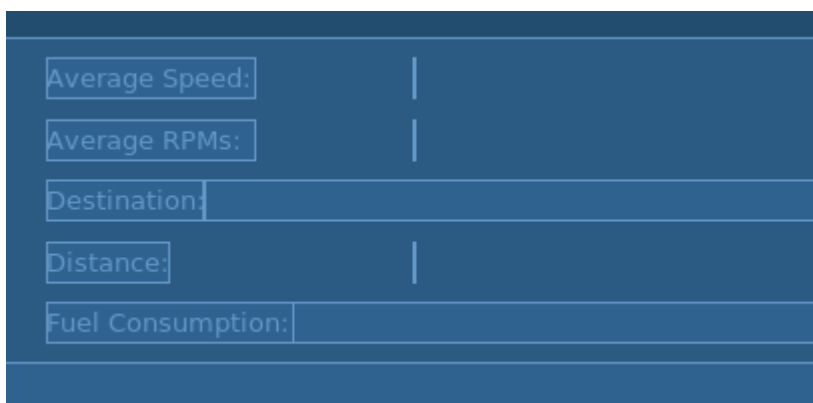
Second layout file something like toolbar can be placed and then further extended with the inclusion of final layout file which would represent main part of data review design page as mentioned in Design chapter.

```
<android.support.v7.widget.Toolbar
    android:id="@+id/toolbar"
    android:layout_width="match_parent"
    android:layout_height="?attr/actionBarSize"
    android:background="?attr/colorPrimary"
    app:popupTheme="@style/AppTheme.PopupOverlay" />
```

**</android.support.design.widget.AppBarLayout>**

```
<include layout="@layout/content_screen_after_login" />
```

Figure 23. Layout XML file snippet displaying additional layout and support for AppBar.



The screenshot shows a dark blue-themed user interface for a journey review. It contains five input fields, each with a label and a text entry area. The labels are 'Average Speed:', 'Average RPMs:', 'Destination:', 'Distance:', and 'Fuel Consumption:'. The 'Destination:' field is wider than the others. The input fields are arranged vertically with a small gap between them.

Figure 24. The current implementation of journey review page design.



## 4.5 "Select Trips" journey review page

This journey review page is an integration of "Trip" and "Data Review" pages, just making "Trip" list page into navigation type list view. This type of implementation design was chosen because it was much faster to implement and saving a user with unnecessary steps so going back and forth of activities as this current implementation save user one click and believed makes the user experience a little better. Both "Material design" and Mei's paper hint that saving user from unnecessary screens and making them as efficient as possible is a better design choice.

Diving deeper into detailed technical aspects, navigation drawer implementation takes different approach from "Data Review" page as mentioned before, as this drawer must have dynamically created list from the internal storage of the device.

```
<ListView
    android:id="@+id/left_drawer"
    android:layout_width="240dp"
    android:layout_height="match_parent"
    android:layout_gravity="start"
    android:choiceMode="singleChoice"
    android:divider="@android:color/transparent"
    android:dividerHeight="0dp"
/>
```

Figure 25. XML code snippet for ListView widget

First class must state private objects like DrawerLayout for activity, ListView for Drawer, ActionBarDrawerToggle for drawer toggle button, String array list for journey file names, CharSequence objects for title naming scheme and regular object for data reviews like TextView and WebView.

```
private String Map_name;
private TextView view, destination;
private WebView myWebView;
```

Figure 26. Code snippet displaying regular object necessary for the implementation

Next, as regular activity starts with "onCreate(...)" stating completely the same layout as "Data Review" page just making it much clearer implementation, using 2 file scheme instead of 3. Followed by setting Adapter [34] and Listener for drawer list.

```
// set a custom shadow that overlays the main content when the drawer opens
mDrawerLayout.setDrawerShadow(R.drawable.drawer_shadow, GravityCompat.START);
// set up the drawer's list view with items and click listener
mDrawerList.setAdapter(new ArrayAdapter<String>(getApplicationContext(),
    R.layout.drawer_list_item, mFileTitles));
mDrawerList.setOnItemClickListener(new DrawerItemClickListener());
```

Figure 27. The current implementation of ListView inside Navigation Drawer.

```
private class DrawerItemClickListener implements ListView.OnItemClickListener {
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
        selectItem(position);
    }
}
```

Figure 28. The current implementation of Drawer list listener.

Secondly, the class then has few necessary automated methods for a drawer for open and close functions, but this time "Title" in the action bar must change based on which file user is looking at.

```
private void selectItem(int position) {  
    // update the main content by replacing fragments  
  
    mainScreen(mFileTitles[position]);  
  
    // update selected item and title, then close the drawer  
    mDrawerList.setItemChecked(position, value: true);  
    setTitle(mFileTitles[position]);  
    mDrawerLayout.closeDrawer(mDrawerList);  
}
```

Figure 29. Code snippet displaying how application title changes based on the selected file.

At this class implementation stage, it was thought that users who review journeys must be mostly interested in places they visited or fuel consumptions, that is why it was chosen to implement "Petrol Prices.com" into one of the extras in the options menu. This is done by simple Intent and Search Manager [35] that automatically opens a browser window with the pre-recorded query, the third-party website was the primary choice for implementation because they specialise in that field area and are completely free.

Rest of the class has the same methods as "Data Review" page.

```
switch(item.getItemId()) {  
    case R.id.action_websearch:  
        // create intent to perform web search for this website  
        Intent intent = new Intent(Intent.ACTION_WEB_SEARCH);  
        intent.putExtra(SearchManager.QUERY, value: "https://www.petroprices.com/");  
        // catch event that there's no activity to handle intent  
        if (intent.resolveActivity(getPackageManager()) != null) {  
            startActivity(intent);  
        } else {  
            Toast.makeText(context: this, text: "Not Available", Toast.LENGTH_LONG).show();  
        }  
        return true;  
}
```

Figure 30. Code snippet displaying implementation of SearchManager

## 4.6 "Car Details" vehicle details page

This class implements the design ideas and suggestions from the "Vehicle details" page, specified in the design chapter, allowing the user to have saved data and double check the information whenever needed.

Class starts off like a standard Android activity class with a layout statement and widgets like toolbars etc. Design of this class specified that 4 images and dynamic text boxes are needed, that is what main class method starts with and logic behind is getting the file, named "CarDetails.txt" in Notes folder of the inner device storage location and reading the file line by line. If the file is filled with data, the system will fill in the textboxes with data based on the ID of the information and textboxes.

```

setContentViews(R.layout.carddetails);
Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
setSupportActionBar(toolbar);

```

```

TextView textbox;

```

```

ImageView fl = (ImageView) findViewById(R.id.imageView3);
ImageView fr = (ImageView) findViewById(R.id.imageView6);
ImageView rl = (ImageView) findViewById(R.id.imageView5);
ImageView rr = (ImageView) findViewById(R.id.imageView7);

```

Figure 31. Code snippet of the current implementation, displaying main layout file, toolbar support and necessary objects.

```

try {
    File iner = new File(getDir( name: "Notes", Context.MODE_PRIVATE), child: "CarDetails.txt");
    List<String> data = FileUtils.readLines(iner, encoding: "UTF-8");
}

```

Figure 32. Code snippet displaying, which file method targets in the inner storage.

E.g. If "CarDetails.txt" file is filled correctly with 8 unique data sets system will fill in a car model, year of manufacture, VIN, all current tire pressure information and fuel tank percentage.

All the tire pressure data can be present in few formats like "NORMAL", "FAULT", "LOW", "ALERT". This is present in the current configuration and has only "NORMAL", "LOW" or none of the two, meaning system will present vector image on the screen at each tire showing not just text with information, but an image as well. These formats of information have 3 vector images available, "NORMAL" will display tire with an "okay" image, "LOW" displaying need to inspect the tire and "FAULT" or "ALERT" will display flat tire symbol that attention is required immediately.

```

if(y==4){
    textbox = (TextView) findViewById(R.id.frontleft);
    textbox.setText(line1);
    if(Objects.equals( a: "NORMAL", line1)){
        fl.setImageResource(R.drawable.tire_pressure_ok);
    }else if(Objects.equals( a: "LOW", line1)){
        fl.setImageResource(R.drawable.tire_pressure_inspect);
    }else{
        fl.setImageResource(R.drawable.flat_tire);
    }
}

```

Figure 33. Code snippet displaying, dynamic image implementation.

The final functionality of the class is displaying fuel percentage with a progress bar if in any case, cars actual meter is not working.

```

if(y==8){
    textbox = (TextView) findViewById(R.id.fuel_level_round);
    textbox.setText(line1);
    ProgressBar progressBar = (ProgressBar) findViewById(R.id.progressBar);
    double n = Double.parseDouble(line1);
    progressBar.setProgress((int)n, animate: true);
}

```

Figure 34. Code snippet displaying fuel progress bar implementation.

Talking about layout design, activity has two layout types in component tree both being "Relative Layout" for ease of use in the design tool and not having many restrictions in the placement of the objects.

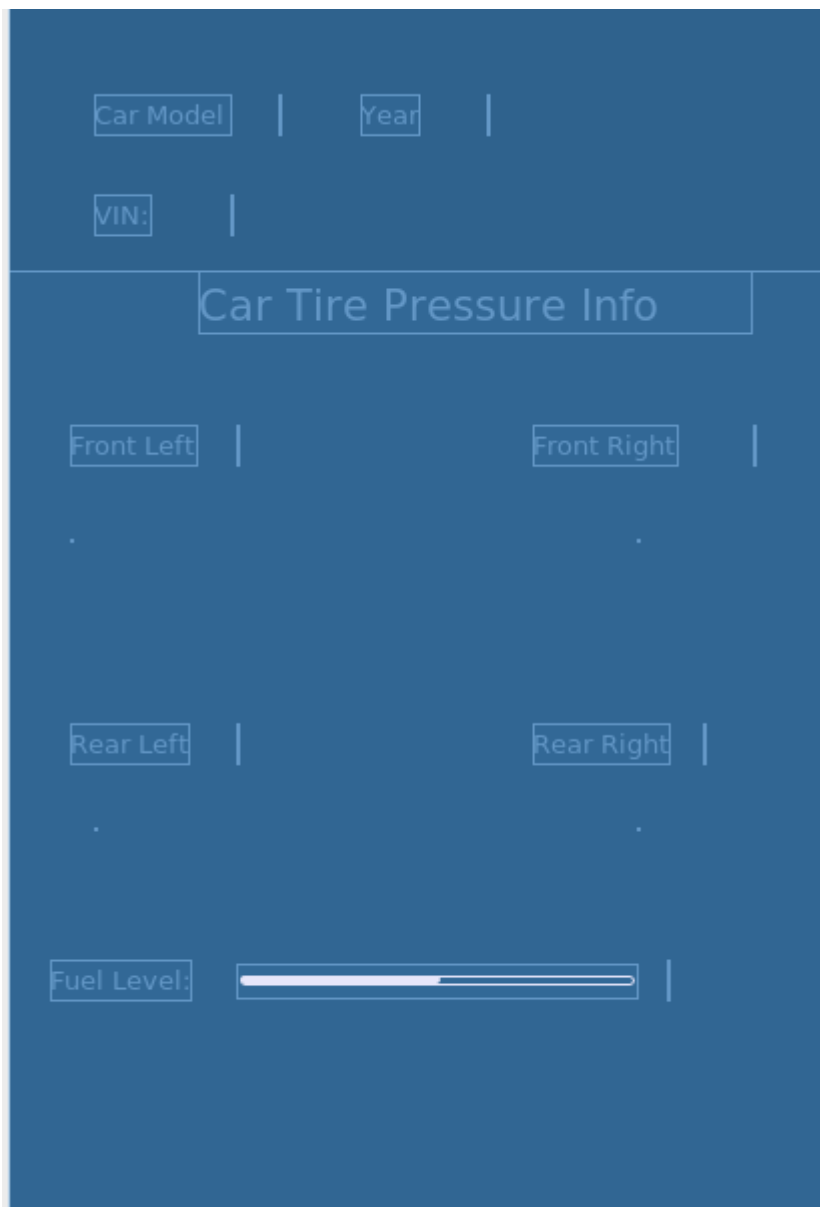


Figure 35. Current implementation design of car details activity.

## 4.7 "User Notifications" settings class of notification system

"User Notifications" class is another activity which will use toolbar for settings options at the top right-side corner of the screen, and implementation design was adjusted into faster, less CPU demanding workload than previously mentioned classes that use the reading of files.

The difference is that "if" statements are no longer used, the system already integrates array lists of strings when reading text file lines, this implementation just gets the information at a desired position in the list rather than iterating whole list and checking whenever information exists or not. Currently, the class implementation does not deal with specific error handling but if the system gets an error message at the text boxes it will display a text to set reminder first and the system was designed in such way, that it is almost impossible to not have the required information or error message displayed.

```
setContentView(R.layout.activity_user_notifications);
Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
setSupportActionBar(toolbar);

TextView textbox;

try {
    File iner = new File(getDir( name: "Notes", Context.MODE_PRIVATE), child: "TireNotes.txt");
    File iner2 = new File(getDir( name: "Notes", Context.MODE_PRIVATE), child: "OilNotes.txt");
    File iner3 = new File(getDir( name: "Notes", Context.MODE_PRIVATE), child: "fuelReminder.txt");
    List<String> data = FileUtils.readLines(iner, encoding: "UTF-8");
    List<String> data2 = FileUtils.readLines(iner2, encoding: "UTF-8");
    List<String> data3 = FileUtils.readLines(iner3, encoding: "UTF-8");

    textbox = (TextView) findViewById(R.id.tiresNotes);
    textbox.setText(data.get(0));
    textbox = (TextView) findViewById(R.id.OilNotes);
    textbox.setText(data2.get(0));
    textbox = (TextView) findViewById(R.id.fuelNotes);
    textbox.setText(data3.get(0));
} catch (IOException e) {
    textbox = (TextView) findViewById(R.id.tiresNotes);
    textbox.setText("Set Reminder");
    textbox = (TextView) findViewById(R.id.OilNotes);
    textbox.setText("Set Reminder");
    textbox = (TextView) findViewById(R.id.fuelNotes);
    textbox.setText("Set Reminder");
}
```

Figure 36. Code snippet displaying targeted files and their data extraction.

The class has 4 different methods to handle all the buttons as they all require slightly different outcome. Car tire inspection at 500 miles "write500" method is the first example when the user clicks on the desired mileage and method handles it by firstly getting or creating required text files in the "Notes" folder of inner device storage. Then it will read odometer reading file to know last known mileage of the vehicle, followed by writing new file ("TireNotes.txt") of the selected mileage into memory and lastly adding both last known mileage and selected mileage into new file ("TireSum.txt") for notification to keep checking against it, indicating how many miles are left till inspection. "write100(...)" method works in the same way, just mileage added is different.

```

void write500(View view) {
{
File iner = new File(getDir( name: "Notes", Context.MODE_PRIVATE), child: "TireNotes.txt");
File iner2 = new File(getDir( name: "Notes", Context.MODE_PRIVATE), child: "OdoReadings.txt");
File iner3 = new File(getDir( name: "Notes", Context.MODE_PRIVATE), child: "TireSum.txt");
List<String> data = FileUtils.readLines(iner2, encoding: "UTF-8");
try (PrintWriter pw = new PrintWriter(iner)) {
    pw.println("500");
}
try(PrintWriter ps = new PrintWriter(iner3)){
    int i = Integer.parseInt(data.get(0));
    int sum = i+500;
    ps.println(sum);
}
}
}

```

Figure 37. Code snippet for Tire notification set at 500 miles.

Oil check inspection reminder is designed to make user input mileage amount when the vehicle must go for maintenance and it works very simply by storing user inputted mileage and later notification checking the difference between real mileage and user set one.

```

void oilcheck(View view){
{
File iner = new File(getDir( name: "Notes", Context.MODE_PRIVATE), child: "OilNotes.txt");
try(PrintWriter pw = new PrintWriter(iner)){
    EditText input = (EditText) findViewById(R.id.editText2);
    pw.println(input.getText());
}
}
}

```

Figure 38. Code snippet for oil check notification.

Fuel reminder is designed and implemented if the vehicle has a faulty indicator or user is very irresponsible of his fuel levels. It works by the user, setting the preferred amount of when the user should be notified when his vehicle is close to or at the level of his desired amount. Works simply by storing user input into ("fuelReminder.txt") file and then checking whenever the user connects to the vehicle.

```

void fuelReminder(View view){
{
File iner = new File(getDir( name: "Notes", Context.MODE_PRIVATE), child: "fuelReminder.txt");
try(PrintWriter pw = new PrintWriter(iner)){
    EditText input = (EditText) findViewById(R.id.editText3);
    pw.println(input.getText());
}
}
}

```

Figure 39. Code snippet for fuel reminder notification.

Layout design for this class is very closely followed by initial design and is created with 2 different layout types first being Coordinator Layout [36] with AppBar Layout widget and its content area includes new layout file with Relative layout file which has 4 buttons that are activated by methods inside the class, 8 static text fields, 3 dynamic text boxes and 2 editable fields that only allow user to type integers making activity as error-free as possible.

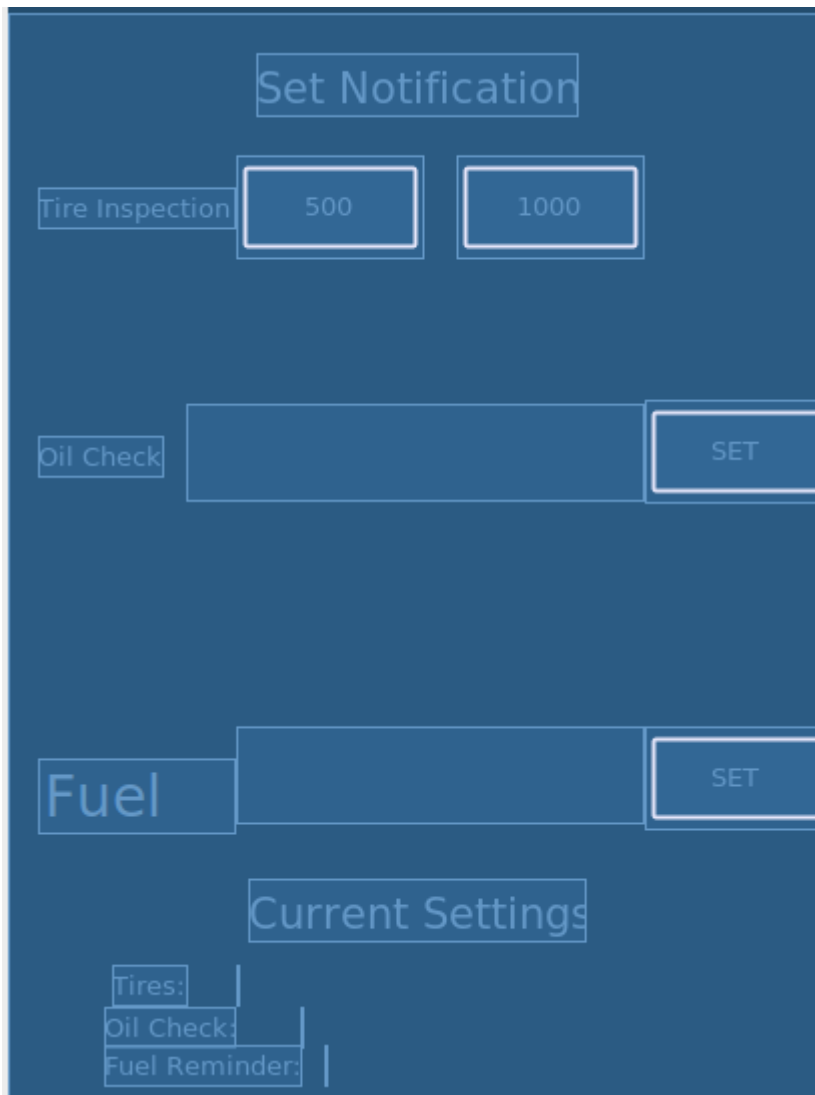


Figure 40. The current implementation of notification and settings screen.

#### 4.8 "SdlService" Heart of the application and SmartDeviceLink

This class contains all main functionality of the SmartDeviceLink (SDL) and where main development is taken place. Currently class contains over 1400 lines of code because class needs to extend Service and IProxyListenerALM which requires to implement all methods of listeners of all methods as system deals with actual hardware and needs to check if the system functions normally indicating any faults or errors while the environment is running.



"SdlService" is where application name can be declared for HMI unit, application ID stated for publishing or testing purposes and IP address if the application is in the testing phase.

```
private static final String APP_NAME           = "Ford TRACK";
private static final String APP_ID            = "8675309";
```

Figure 41. Code snippet of application display name and testing app-id

```
// TCP/IP transport config
private static final int TCP_PORT = 12345;
private static final String DEV_MACHINE_IP_ADDRESS = "155.245.20.210";
```

Figure 42. Code snippet for developers testing IP address.

As a class have an unknown number of methods; this report implementation will go over only required and written methods by the developer.

Class starts implementation with object statements and constant that will be used throughout the whole class and follows with first methods of starting a proxy of SDL, it works the same way as "Connect" button implementation does because button activates method "startProxy(...)" and checking what build configuration is running, difference being method here initiates security levels for "MBT" connection and rest of the method checks what version of Android is currently trying to run the application and checks if connection is using USB transport it will not allow for Android version to be lower than 11. Connection methods for proxy and all required methods for proxy were provided in the base application from SDL website or GitHub.

```
public void startProxy(boolean forceConnect, Intent intent) {
    Log.i(TAG, msg: "Trying to start proxy");
    if (proxy == null) {
        try {
            Log.i(TAG, msg: "Starting SDL Proxy");
            BaseTransportConfig transport = null;
            if (BuildConfig.TRANSPORT.equals("MBT")){
                int securityLevel;
                if (BuildConfig.SECURITY.equals("HIGH")){
                    securityLevel = MultiplexTransportConfig.FLAG_MULTI_SECURITY_HIGH;
                } else if (BuildConfig.SECURITY.equals("MED")){
                    securityLevel = MultiplexTransportConfig.FLAG_MULTI_SECURITY_MED;
                } else if (BuildConfig.SECURITY.equals("LOW")){
                    securityLevel = MultiplexTransportConfig.FLAG_MULTI_SECURITY_LOW;
                } else{
                    securityLevel = MultiplexTransportConfig.FLAG_MULTI_SECURITY_OFF;
                }
            }
        }
    }
}
```

Figure 43. Code snippet of current implementation for connecting the application to SDL

```
} else if (BuildConfig.TRANSPORT.equals("TCP")){
    transport = new TCPTransportConfig(TCP_PORT, DEV_MACHINE_IP_ADDRESS, autoReconnect: true);
} else if (BuildConfig.TRANSPORT.equals("USB")) {
    if (intent != null && intent.hasExtra(UsbManager.EXTRA_ACCESSORY)) { //If we want to support USB transport
        if (android.os.Build.VERSION.SDK_INT <= android.os.Build.VERSION_CODES.HONEYCOMB) {
            Log.e(TAG, msg: "Unable to start proxy. Android OS version is too low");
            return;
        }
    }
}
```

Figure 44. Code snippet for testing connection using IP address or USB.



Design of HMI states what display layout system should use for the main screen. Implementation of the layout is rather straightforward and easy, initialize "setDisplayLayout()", then set the preferred layout in this system case it is "GRAPHIC\_WITH\_TEXT\_AND\_SOFT\_BUTTONS", next method required to have response listener using Logs [] for testing and system health purposes. Followed, this method dynamic layout changer method is written for future development.

```
public void setDisplayLayoutRequest() {
    SetDisplayLayout setDisplayLayoutRequest = new SetDisplayLayout();
    setDisplayLayoutRequest.setDisplayLayout("GRAPHIC_WITH_TEXT_AND_SOFTBUTTONS");
    setDisplayLayoutRequest.setOnRPCResponseListener((correlationId, response) → {
        if(((SetDisplayLayoutResponse) response).getSuccess()){
            Log.i( tag: "SDLservice", msg: "Display layout set successfully");
            //More user interface RPCs
        }else{
            Log.i( tag: "SDLservice", msg: "Display layout request rejected");
        }
    });
    sendRpcRequest(setDisplayLayoutRequest);
}
```

Figure 45. Code snippet displaying layout request from SDL.

```
public void changeDisplayLayout(String layoutName){
    SetDisplayLayout setDisplayLayoutRequest = new SetDisplayLayout();
    setDisplayLayoutRequest.setDisplayLayout(layoutName);
    sendRpcRequest(setDisplayLayoutRequest);
}
```

Figure 46. Code snippet displaying dynamic layout request from SDL.

Next design requires two interface buttons that are named soft buttons in SDL implementation. Firstly, system uploads image from the HMI memory for the button to use the image and display neatly on the screen. The image is then initialized as dynamic and value set as required by the system.

```
uploadImage(R.drawable.ifroad, imageName: "ifroad.png", CorrelationIdGenerator.generateId(), isPersistent: true);
```

Figure 47. Code snippet displaying how the image is uploaded to the system for displaying on HMI.

```
Image trip_image = new Image();
trip_image.setImageType(ImageType.DYNAMIC);
trip_image.setValue("ifroad.png");
```

Figure 48. Code snippet displaying how the image is created for SDL head unit.

When soft buttons are created in SDL, the system requires to have them in a special array list, buttons can be in one array list or stored in different ones, the main difference being for the system is to have all buttons with unique IDs set, as that is how the system identifies each button on HMI. The current implementation of buttons uses both image and text for clarity.

```

List <SoftButton> softButtons = new ArrayList<>();
SoftButton Function_button = new SoftButton();
Function_button.setType(SoftButtonType.SBT_BOTH);
Function_button.setText("Start TRIP");
Function_button.setImage(trip_image);
Function_button.setSoftButtonID(0);
softButtons.add(Function_button);

```

Figure 49. Code snippet showing how soft buttons are created.

```

public void onOnButtonPress(OnButtonPress notification) {
    switch (notification.getCustomButtonName()){
        case 1:
            int ID = notification.getCustomButtonName();
            Log.i(TAG, msg: "OnButtonPress notification from SDL: " + ID);
            Writer_method();
            setFirstButton();
            break;
        case 0:
            subscribeRequest();
            subscribedRPM.clear();
            subscribedSpeed.clear();
            GPS_data_RAW.clear();
            GPS_data.clear();
            setStopButton();
            break;
    }
}

```

Figure 50. Code snippet showing where soft buttons are activated and their activated methods.

After these soft buttons were created for main display of the HMI, now the system will implement menu level buttons for more functionality of the application. Current buttons in the menu are created for testing purposes and here will be discussed what is needed to create one and how they structured in SDL environment.

First, all menu based buttons are created inside "sendCommands()" method, then each button must be initiated with "MenuParams" where menu button name, position, parent or submenu ids are set, followed by initialization of "AddCommand()" where command is given unique ID and setting the command in menu. Most commands can and should use one of the following interaction types. Manual, VR or both, manual meaning interaction only with a screen of HMI, VR is voice activate interaction or both. These menu buttons

are activated in the "onOnCommand(...)" method, by indicating "CmdID" of the button

```
MenuParams params1 = new MenuParams();
params1.setMenuName("Test Notification");
command1 = new AddCommand();
command1.setCmdID(2);
command1.setMenuParams(params1);

MenuParams params2 = new MenuParams();
params2.setMenuName("Graphics with text on right");
AddCommand command3 = new AddCommand();
command3.setCmdID(3);
command3.setMenuParams(params2);

MenuParams params4 = new MenuParams();
params4.setMenuName("Vehicle Tire Information");
AddCommand command4 = new AddCommand();
command4.setCmdID(4);
command4.setMenuParams(params4);
```

Figure 51. Code image, showing how menu options are created.

Note: manual option does not require to write any additional code and only VR interaction requires to add an additional line of code making application interactable with both types.

```
command.setVrCommands(Arrays.asList(new String[]{TEST_COMMAND_NAME}));
```

Figure 52. Code snippet displaying how voice commands are added to the menu options.

```
public void onOnCommand(OnCommand notification){
    Integer id = notification.getCmdID();
    if(id != null){
        switch(id){
            case TEST_COMMAND_ID:
                showTest();
                break;
            case 2:
                sendNotification();
                break;
            case 3:
                changeDisplayLayout( layoutName: "GRAPHIC_WITH_TEXT");
                break;
            case 4:
                odoReadings();
                vehicle_details();
                break;
        }
    }
}
```

Figure 53. Code image displaying where menu buttons are activated.

Now, implementation expands of how images are uploaded to the head unit or HMI, to perform this action "uploadImage(...)" method is required as it specifies what images can be actually can be uploaded. Firstly, system must initiate "PutFile" RPC request, set file type to PNG or any other, correlation id mostly generated one, persistence to the system and specifying that it is not actual system file, finally passing all the files to "contentOfResource(...)" method which helps to take resource file and turn them into byte array for hardware level read.

@param resource the R.drawable.\_\_\_ value of the image you wish to send  
 @param imageName the filename that will be used to reference this image  
 @param correlationId the correlation id to be used with this request. Helpful for monitoring putfileresponses  
 @param isPersistent tell the system if the file should stay or be cleared out after connection.

Figure 54. Important parameters explanations.

```

private void uploadImage(int resource, String imageName,int correlationId, boolean isPersistent){
    PutFile putFile = new PutFile();
    putFile.setFileType(FileType.GRAPHIC_PNG);
    putFile.setSdlFileName(imageName);
    putFile.setCorrelationID(correlationId);
    putFile.setPersistentFile(isPersistent);
    putFile.setSystemFile(false);
    putFile.setBulkData(contentsOfResource(resource));
  }

```

Figure 55. Code snippet displaying how image upload method functions in SDL system.

```

private byte[] contentsOfResource(int resource) {
    InputStream is = null;
    try {
        is = getResources().openRawResource(resource);
        ByteArrayOutputStream os = new ByteArrayOutputStream(is.available());
        final int bufferSize = 4096;
        final byte[] buffer = new byte[bufferSize];
        int available;
        while ((available = is.read(buffer)) >= 0) {
            os.write(buffer, off: 0, available);
        }
        return os.toByteArray();
    } catch (IOException e) {
        Log.w(TAG, msg: "Can't read icon file", e);
        return null;
    } finally {
        if (is != null) {
            try {
                is.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}

```

Figure 56. Code image displaying how the image is processed for HMI interface.

Note: both methods were provided in the base application but are vital to the system implementation.

Following, method "onOnHMIStatus(...)" is used for application display and the first run of the application on head unit, running methods like "setDisplayLayoutRequest()", performs welcome message and other HMI interactions, like button initialisation, vehicle details, odometer reading for notification and other commands based on HMI Levels.

```

public void onOnHMIStatus(OnHMIStatus notification) {
    if(notification.getHmiLevel().equals(HMILevel.HMI_FULL)){

        if (notification.getFirstRun()) {
            setDisplayLayoutRequest();

            // send welcome message if applicable
            performWelcomeMessage();
        }
        // Other HMI (Show, PerformInteraction, etc.) would go here
        setFirstButton();
        writer_details();
        odoReadings();
        sendNotification();
    }
}

```

Figure 57. Code image displaying interaction for the first run and other method activation.

```

if(!notification.getHmiLevel().equals(HMILevel.HMI_NONE)
    && firstNonHmiNone){
    sendCommands();
    //uploadImages();
    firstNonHmiNone = false;
    // Other app setup (SubMenu, CreateChoiceSet, etc.) would go here
}else{
    //We have HMI_NONE
    if(notification.getFirstRun()){
        uploadImages();
    }
}
}

```

Figure 58. Code image displaying rest of method activation during the first run.

Next, class focuses on the vehicle data subscription or retrieval requests, implementation has two iterations of subscriptions using "onOnPermissionsChange(...)" and "subscribeRequest()" methods, main method is "subscribeRequest()" and if it fails "onOnpermissionsChange(...)" will take over. In these methods, system subscribes to speed, rpm, GPS, odometer, fuel levels and instant fuel consumption.

```

public void onOnPermissionsChange(OnPermissionsChange notification) {
    Log.i(TAG, "msg: " + notification);

    List<PermissionItem> permissions = notification.getPermissionItem();
    for(PermissionItem permission:permissions){
        if(permission.getRpcName().equalsIgnoreCase(FunctionID.SUBSCRIBE_VEHICLE_DATA.name())){
            if(permission.getHMIPermissions().getAllowed()!=null && permission.getHMIPermissions().getAllowed().size()>0){
                if(!isVehicleDataSubscribed){ //If we haven't already subscribed we will subscribe now
                    //TODO: Add the vehicle data items you want to subscribe to
                    //proxy.subscribevehicledata(gps, speed, rpm, fuelLevel, fuelLevel_State, instantFuelConsumption, externalTemperature, prndl
                    try {
                        proxy.subscribevehicledata( gps: true, speed: true, rpm: true, fuelLevel: false, fuelLevel_State: false, instantFuelConsumption: true,
                    } catch (SdlException e) {
                        e.printStackTrace();
                    }
                }
            }
        }
    }
}

```

Figure 59. Code snippet displaying backup method for vehicle data subscription.



```

public void subscribeRequest(){
    com.smartdevicelink.proxy.rpc.SubscribeVehicleData subscribeRequest = new SubscribeVehicleData();
    subscribeRequest.setSpeed(true);
    subscribeRequest.setRpm(true);
    subscribeRequest.setGps(true);
    subscribeRequest.setOdometer(true);
    subscribeRequest.setFuelLevel(true);
    subscribeRequest.setInstantFuelConsumption(true);
    subscribeRequest.setOnRPCResponseListener((correlationId, response) -> {
        if(response.getSuccess()){
            Log.i(TAG, msg: "Successfully subscribed to vehicle data");
        }else{
            Log.i(TAG, msg: "Request to subscribe to vehicle data was rejected");
        }
    });
    sendRpcRequest(subscribeRequest);
}

```

Figure 60. Code snippet displaying the main method for vehicle data subscription.

Vehicle data method is called "getRequest()", which initialise RPC requests of getting vehicle data and request for vehicle type. Because of current bugs in the testing system, implementation requires to set most details, but in real-world use, the system would overwrite with actual information.

```

public void getRequest(){
    GetVehicleData vdRequest = new GetVehicleData();
    VehicleType vtRequest = new VehicleType();
    vtRequest.setMake("Ford");
    vtRequest.setModel("Fiesta");
    vtRequest.setModelYear("2013");
    vtRequest.setTrim("SE");
    vdRequest.setTirePressure(true);
    vdRequest.setVin(true);
    vdRequest.setFuelLevel(true);
    make = vtRequest.getMake();
    Log.i( tag: "Vehicle Type", msg: "Make: "+make);
    model = vtRequest.getModel();
    Log.i( tag: "Vehicle Type", msg: "Model: "+model);
    model_year = vtRequest.getModelYear();
    Log.i( tag: "Vehicle Type", msg: "Year: "+model_year);
    trim = vtRequest.getTrim();
    Log.i( tag: "Vehicle Type", msg: "Trim: "+trim);
}

```

Figure 61. Code image displaying request for vehicle data.

Vehicle tire information in SDL environment is only possible to retrieve such information inside vehicle "response listener". Firstly, needing to state Tire Status RPC with vehicle data response indicating to focus tire pressure information as a lot of other information is possible to retrieve from such response and if the response is successful system uses "Log" to store response and parses the information into local variables for "writer\_details()" method to write all the variable information to the file named "CarDetails.txt" inside "Notes" folder.

```

vdRequest.setOnRPCResponseListener((correlationId, response) → {
    if (response.isSuccess()){
        TireStatus tirePressure = ((GetVehicleDataResponse) response).getTirePressure();
        vin = ((GetVehicleDataResponse) response).getVin();
        double fuelLevel = ((GetVehicleDataResponse) response).getFuelLevel();
        Log.i( tag: "SdlService", msg: "GetVehicleData successful");
        Log.i( tag: "Response", msg: "Tire Pressure: "+tirePressure.getLeftRear().getStatus().toString());
        Log.i( tag: "Response", msg: "Tire Pressure: "+tirePressure.getRightFront().getStatus().toString());
        Log.i( tag: "Response", msg: "Tire Pressure: "+tirePressure.getLeftFront().getStatus().toString());
        Log.i( tag: "Response", msg: "Tire Pressure: "+tirePressure.getRightRear().getStatus().toString());
        Log.i( tag: "Response", msg: "Vin: "+vin);
        Log.i( tag: "Response", msg: "Fuel Level: "+Math.round(fuelLevel));
        LeftRear = tirePressure.getLeftRear().getStatus().toString();
        RightFront = tirePressure.getRightFront().getStatus().toString();
        RightRear = tirePressure.getRightRear().getStatus().toString();
        LeftFront = tirePressure.getLeftFront().getStatus().toString();
        fuelLevelRound = Math.round(fuelLevel);
    }else{
        Log.i( tag: "SdlService", msg: "GetVehicleData was rejected");
    }
});
sendRpcRequest(vdRequest);

```

Figure 62. Code image displaying, vehicle data response.

The same principle follows "odoReadings()" method, which records and writes vehicle actual mileage meter.

```

public void odoReadings(){
    File iner = new File(getDir( name: "Notes", Context.MODE_PRIVATE), child: "OdoReadings.txt");
    GetVehicleData vdRequest = new GetVehicleData();
    vdRequest.setOdometer(true);
    vdRequest.setOnRPCResponseListener((correlationId, response) → {
        if (response.isSuccess()){
            readings = ((GetVehicleDataResponse) response).getOdometer();
        }else{
            Log.i( tag: "SdlService", msg: "GetVehicleData was rejected");
        }
    });
    sendRpcRequest(vdRequest);
}

```

Figure 63. Code snippet displaying how odometer reading is extracted from SDL for vehicle data screen.

Note: to get human readable response information, it has to be retrieved using ".getStatus().toString()" method after specifying which tire to focus.

Now, focusing on one of the most important methods in the implemented system "onOnVehicleData(...)", this method gets data responses in a name of notifications in SDL system of actual vehicle data that application subscribes to. Implementation gets response information about speed, rpm, GPS data in both latitude and longitude degrees separately, then methods states the format in what coordinates must be passed to the system and fixing current bug on the SYNC3 emulator by comparing both values and "if" both are not equal, system will record coordinates and all other instances are will be rejected. This kind of solution is acceptable for real-world implementation because it is very rare for coordinates to match on the road.

```

public void onOnVehicleData(OnVehicleData notification) {
    //Log.i(TAG, "Vehicle data notification from SDL");
    //TODO Put your vehicle data code here
    //ie, notification.getSpeed().
    try{
        speed = notification.getSpeed();
        rpm = notification.getRpm();
        GPSTData coords = notification.getGps();
        double latitude = coords.getLatitudeDegrees();
        double gps_speed = coords.getSpeed();
        double longitude = coords.getLongitudeDegrees();
        Log.i(TAG, msg: "Speed status was updated to: " + speed + " " + "Gps speed: " + gps_speed);
        Type: In word 'longitude' more... (Ctrl+F1) + latitude + " " + longitude);
        Log.i(TAG, msg: "RPM status was updated to: " + rpm);

        DecimalFormat df = new DecimalFormat( pattern: "##.####");
        df.setRoundingMode(RoundingMode.DOWN);
        String lt = df.format(latitude);
        String lg = df.format(longitude);
        //Bug FIX
        if(!lt.equals(lg)){
            GPS_data.add("{lat:"+lt+", "+ "lng:"+lg+"}");
            GPS_data_RAW.add(lt);
            GPS_data_RAW.add(lg);
        }
        subscribedRPM.add(rpm);

    }catch (NullPointerException e){
        Log.i( tag: "Error", msg: "One of the values are passed as 'null'");
    }
    //showTest2(speed, rpm);
    if(speed>0.0){
        subscribedSpeed.add(speed);
    }
}

```

Figure 64. Code image displaying how subscribed data is extracted from SDL system.

As this method records speed data and system needs the data in calculations all speed values of "0.0" are ignored as it would alter the data in a negative way.

Note: this method is very prone to "Null Pointer Exception" as notifications responses are not passed to the system if they do not change.

As implementation comes to the last stages, this part will focus on how "Notifications" are launched in the current implementation with the "sendNotification()" method.

This method gets and reads files from the "Notes" folder of inner device storage, then asks SDL system for vehicle data firstly setting parameters (Odometer and Fuel Level) and send the RPC for a response. If the response is success system gets the readings and checks them against current notifications settings in appropriate files of each notification. Design of notifications is the same as in the design chapter.

```

public void sendNotification(){
    try {
        File iner = new File(getDir( name: "Notes", Context.MODE_PRIVATE), child: "TireSum.txt");
        File iner2 = new File(getDir( name: "Notes", Context.MODE_PRIVATE), child: "OilNotes.txt");
        File iner3 = new File(getDir( name: "Notes", Context.MODE_PRIVATE), child: "fuelReminder.txt");
        List<String> data = FileUtils.readLines(iner, encoding: "UTF-8");
        List<String> data2 = FileUtils.readLines(iner2, encoding: "UTF-8");
        List<String> data3 = FileUtils.readLines(iner3, encoding: "UTF-8");
    }
}

```

Figure 65. Code snippet displaying targeted files for notification check-up.



```

GetVehicleData vdRequest = new GetVehicleData();
vdRequest.setOdometer(true);
vdRequest.setFuelLevel(true);
vdRequest.setOnRPCResponseListener((correlationId, response) -> {
    if (response.getSuccess()){
        readings = ((GetVehicleDataResponse) response).getOdometer();
        fuelLevel = ((GetVehicleDataResponse) response).getFuelLevel();
    }else{
        Log.i( tag: "SdlService", msg: "GetVehicleData was rejected");
    }
});
sendRpcRequest(vdRequest);

```

Figure 66. Code image displaying how odometer and fuel levels are retrieved for the notification system.

```

if(Integer.parseInt(data.get(0))>readings){
    int sum = Integer.parseInt(data.get(0))-readings;
    android.support.v4.app.NotificationCompat.Builder mBuilder = new android.support.v4.app.NotificationCompat.Builder( context: this)
        .setSmallIcon(R.drawable.sdl_tray_icon)
        .setContentTitle("Tire Inspection")
        .setContentText("Left to inspection: "+sum);

    NotificationManager mNotification = (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);

    if (mNotification != null) {
        mNotification.notify( id: 1, mBuilder.build());
    }
}

```

Figure 67. Code snippet displaying how notifications are created and sent to the user.

Note: When creating such notifications and they display different information when building object ID of the notifications must be unique, otherwise last known information would be displayed.

Finally, implementation chapter closes out talking about how the system writes everything to the file system, the main method of writing data to files is "Writer\_method()", there are other methods like "writer\_details()" method etc.

```

public void writer_details(){
    File f = new File(getDir( name: "Notes", Context.MODE_PRIVATE), child: "CarDetails.txt");
    getRequest();
    try(PrintWriter pw = new PrintWriter(f)){
        pw.print(make+" ");
        pw.print(model+" ");
        pw.println(trim);
        pw.println(model_year);
        pw.println(vin);
        pw.println(LeftFront);
        pw.println(RightFront);
        pw.println(LeftRear);
        pw.println(RightRear);
        pw.println(fuelLevelRound);
    }
}

```

Figure 68. Code snippet showing how vehicle details are written in a text file.

mentioned before and that write information to inner device storage, but all of their functionality was passed from "Writer\_method()" and it starts off with a getting current date of the system and applying it to the file names (e.g. "Mon Mar 19 13\_40\_22 GMT+00\_00 2018.txt"), then it gets access to "Final" folder, "GPS\_data"

and assets [] folder, where firstly method handles assets folder with GPS coordinate data.

```
public void Writer_method() {
    String name = new java.util.Date().toString();

    File Final = new File(getDir( name: "Final", Context.MODE_PRIVATE), child: name+".txt");

    // Writing to Map template gps coordinates, and center as starting position
    File Map_template = new File(getFilesDir().getPath(), child: "GPS map.html");

    final AssetManager assets = getAssets();

    File newMap = new File(getDir( name: "GPS_data", Context.MODE_PRIVATE), child: name+"_"+"GPS_map_route.html");
```

Figure 69. Code image displaying start of the journeys writing method.

```
InputStream input = getAssets().open( fileName: "GPS map.html");
int size = input.available();
byte[] buffer = new byte[size];
input.read(buffer);
input.close();
```

Figure 70. Code snippet showing how template file is processed.

GPS data is saved on the map using template file provided by Google, which is slightly modified to act as template file, method opens modified template, converts content to byte array and then those bytes are placed in object "String" to rebuild it in one constant streamed line, same is done with "GPS\_data" array and finally replacing target string part named "\$GPS\_center" with first "GPS\_data" instance and targeting second string part named "\$GPS\_coords" with actual string of "GPS\_data" system placed in string object before. Final part and logic behind other writer method are compiling files for "Final" folder where main data is stored for journey review.

```
String html_string = new String(buffer);
//String html_string = FileUtils.readFileToString(Map_template);
String GPS_coords = GPS_data.toString();
html_string = html_string.replace( target: "$GPS_center", GPS_data.get(0));
html_string = html_string.replace( target: "$GPS_coords", GPS_coords);
FileUtils.writeStringToFile(newMap, html_string);
```

Figure 71. Code snippet showing how GPS information is injected into the template file.

This is done with Print Writers [37] allowing developers to easily create and close writing files saving valuable implementation time, first speed and rpm data averages are computed, by checking if both data lists are not empty, "if" true function will iterate throughout the array list adding all instances into one then dividing the sum by the number of instances in the specified array list and finally rounding up the results and writing them into file line by line.

```

try(PrintWriter pw = new PrintWriter(Final)){
    double sum =0.0; int sum2=0;

    // Average Speed results
    if(!subscribedSpeed.isEmpty()){
        for(Double number : subscribedSpeed){
            sum+=number;
        }
        double result = Math.round(sum/subscribedSpeed.size());
        pw.println(result);
        Log.i( tag: "Data Writer", msg: "Average Speed Written");
    }
}

```

Figure 72. Code image displaying show average speed is written for journey review screen.

Next GPS coordinates for the street name are saved, by accessing "GPS\_data\_RAW" list, extracting last two instances of the array list and writing data to the same file.

```

pw.println(GPS_data_RAW.get(GPS_data_RAW.size()-2));
pw.println(GPS_data_RAW.get(GPS_data_RAW.size()-1));

```

Figure 73. Code snippet displaying how last GPS data coordinates are retrieved for location identification.

The final step in this method is saving the name of the newly created GPS data map, where later Android activity reads the file name and looks for it, in the corresponding folder if the file exists and opens it's in Web View.

```

pw.println(newMap.getName());

```

Figure 74. Code snippet displaying how journey review screen knows which GPS map to open.

## Chapter 5 Project planning

### 5.1 Methodology adopted

Throughout the project agile methodology was used to help the development plan with ever-changing requirements needed for the project to succeed. The reason why agile was chosen over the waterfall was that agile does not require to plan features and structure of the system before the developer is familiar with the development tools and API. This was proven the correct methodology to choose because during early development sprints, all functions and features of the API and tools were learned and during these sprints, it was rather clear what API is capable to do and what features are not feasible with current API iteration.

Furthermore, agile having different types of methodology cycles than waterfall one initial path. Agile proven to be most effective way of developing software, as with agile it is possible to scale software at any point in the development cycle and during such project used few different agile methodology cycles, first was XP allowing development increments to be sprint plans and review of them after each sprint was done, using this methodology cycle, project can reflect the progress it is making at steady pace, second methodology cycle was adaptive software development, using this methodology theory and adapting it to the project proved to be lifesaving, as initial software development focused on the high-level idea of what the system should do and after learning all capabilities of the SDL API it was clear that most important feature of the project mentioned in the initial report will not going to work. With this discovery emergency meeting was arranged with a company representative, he confirmed that current technology used in the project is not capable without full real-life testing involvement that university cannot provide. Knowing these issues, the main idea of the system had to be changed from machine learned driving co-pilot to the real-time data storage solution for the research simulation purposes as well as user awareness program.

### 5.2 Sprints overview

The project consisted of total 13 sprints, starting in late 2017 summer where the focus was to start reading and thinking of ideas for the project, initial planning software was Microsoft Project and later was changed to Trello for ease of use and interim oral presentation second assessor advice. Trello proved to be friendlier for the development because it was easy to use and quick to update without a need to learn all functions of the web-based software.

To start with the development, summer and autumn sprints consisted of reading, learning and development of the main feature. These sprints totalled to 7 different sprints with planned about 130 hours and sprints ended with actual 180 hours within 10 weeks' time. Most notable sprint during this period was autumn sprint 4, where 77 hours were spent learning all the technicalities of SDL API and coding them to get used to the structure of the API.

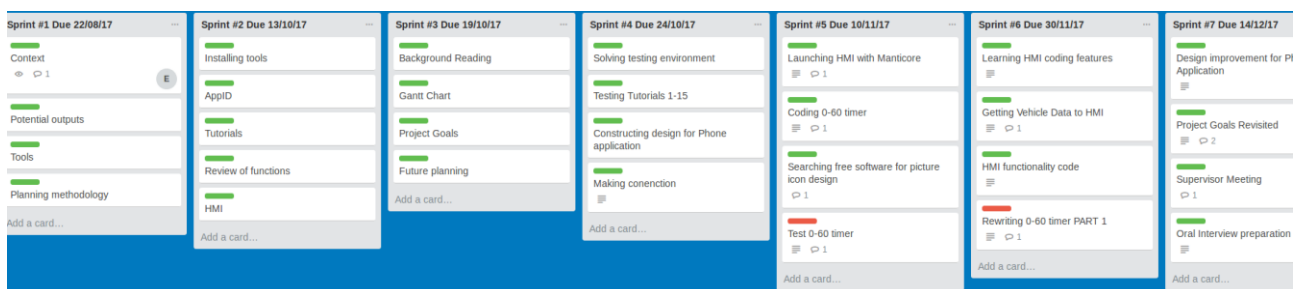


Figure 75. Sprints of summer and autumn terms

Moreover, during spring sprint plans, the focus was to finish developing the main feature of the mobile application, solve discovered bugs in the testing software and add additional features to the application. Spring sprint plans took 9 weeks to complete and all were completed in a close approximation of the plan with 135 hours planned and 160 hours actual for completion of all spring sprint plans.

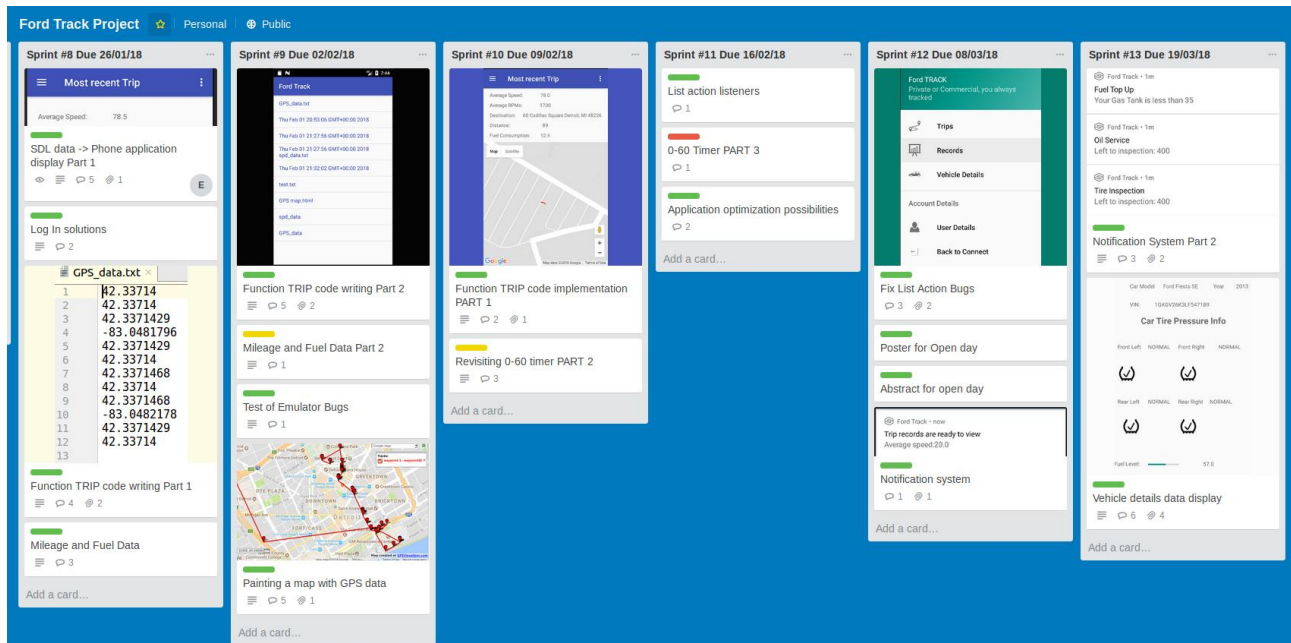


Figure 76. Sprints of spring term

Finally, sprint plans proved to be effective planning solution, by reflecting each task success and time taken to complete each of planned task and choosing agile methodology proved that project can be continually redefined if necessary by eliminating failed tasks and risks that come with them. All sprints can be found on Trello page at <https://trello.com/b/2SOLaa4c> each sprint has a number of tasks and all are colour labelled, green showing the complete success of the task, yellow shows difficulties found with the task and red colour means failure of task implementation. Please consider that Trello was adapted only after sprint 7, meaning minimal information can be found for sprint 1 to 7.

Note: link to first planning solution.

[https://essexuniversity-my.sharepoint.com/:u:/g/personal/esenav\\_essex\\_ac\\_uk/EWOvNRpu34pDqilNvIRYacUBRRBRPCO-sh1vnKMa-OCQuQ](https://essexuniversity-my.sharepoint.com/:u:/g/personal/esenav_essex_ac_uk/EWOvNRpu34pDqilNvIRYacUBRRBRPCO-sh1vnKMa-OCQuQ)

## 5.3 Version Control for the software

Use of version control for software development is essential when such projects take team of people and during this software development only one person was developing such project meaning all structure and design was developed by one person only, allowing to know all changes in the code and using iteration testing solution, software could not be broken due to lack of knowledge or miscommunication and version control software features were not essential for this project.

Iteration testing is known, when a sprint task is finished with implementation stage, tests would be carried out to see if the overall system still works and whole software solution functions as it was before, if any errors

occur during testing, implementation solution would be revisited of task and logs would be inspected at which stage system was broken.

GitLab link for main classes and layouts of the project: [https://gitlab.com/esenav/Final\\_year\\_project](https://gitlab.com/esenav/Final_year_project)

The Logcat window in Android Studio displays system messages, such as when a garbage collection occurs, and messages that you added to your app with the Log class. It displays messages in real time and keeps a history so you can view older messages [39].

To display just the information of interest, you can create filters, modify how much information is displayed in messages, set priority levels, display messages produced by app code only, and search the log. By default, logcat shows the log output related to the most recently run app only.

When an app throws an exception, logcat shows a message followed by the associated stack trace containing links to the line of code.

## Chapter 6 Conclusions, evaluation and future work

In this chapter, the report will consider 3 benchmark criteria that were set out in 1.6 and discuss possible future work that can be done for mobile application.

### 6.1 Evaluation and conclusions

Firstly, project schedule was implemented from the start and carried out throughout the project lifecycle, this proved to be great time management solution and feature implementation overview. Autumn and spring sprints reflect that there were some issues with one or two tasks throughout the sprint periods and from based conclusions in those sprints it should have been clear from first implementation failure of the feature not to go back and revisit the feature, but stubbornness of the feature to be implemented, few iterations were tried of the same function resulting in failed attempts due to bad solution or newly found solution not being up to the quality of the developer standards. The overall project schedule was a little bit underestimated and more hours had to be dedicated to featuring implementation stage, this is very clear in the early development schedule.

Secondly, methodology, design and implementation were carried out up to a standard, meaning all requirements, objectives and deliverables were achieved and the decision to use agile was the best approach to such project, because, at the start, the project did not have any specifications to follow, besides using SDL API and using its features. Although there has been a major change in main functionality of the application this obstacle did not prove to be a much of a challenge as considering market of similar applications, ideas of them were adopted to carry out this project and improved.

In the terms of design and implementation, the application was design to be future scalable, with new features easily added to the existing list, some methods in the implementation were already left to be used for future work. Regarding implementation language Java, it was a good choice to carry out the project in well-known and supported language. Unfortunately, no solutions could be found on SDL API and Java, forums and Google searches did not produce any results, needing for project to join SDL Slack community [38] for Android developers and even with community support it was very difficult to find help on bug fixing or Java implementation solutions, leaving only trial and error approach to SDL API implementation. Later in development, mobile application visual design had to be made for the user and at that stage, all Google Android developers site were used to the fullest to implement best possible design solution possible for interface with as little knowledge as possible of Android.

Finally, for first time Android developer it is not recommended to use SDL API as their first ever project, because the learning curve is not linear of such API, it is more of a jump because API can only be understood after all corners of API were learned. In terms of mobile application, usability for the user it was designed in such way that it is easily identifiable and each screen shows exactly what user expects to be showing. Reliability of the application is flawless and no crashes occur when using application, on the other hand, it is sometimes slow with "Data Review" page because it needs to load Google maps view. Overall requirements and objectives were achieved and the project was a success in terms of research and educational purpose.



## 6.2 Images of current implementation results



Figure 77. How connected application looks on HMI screen results

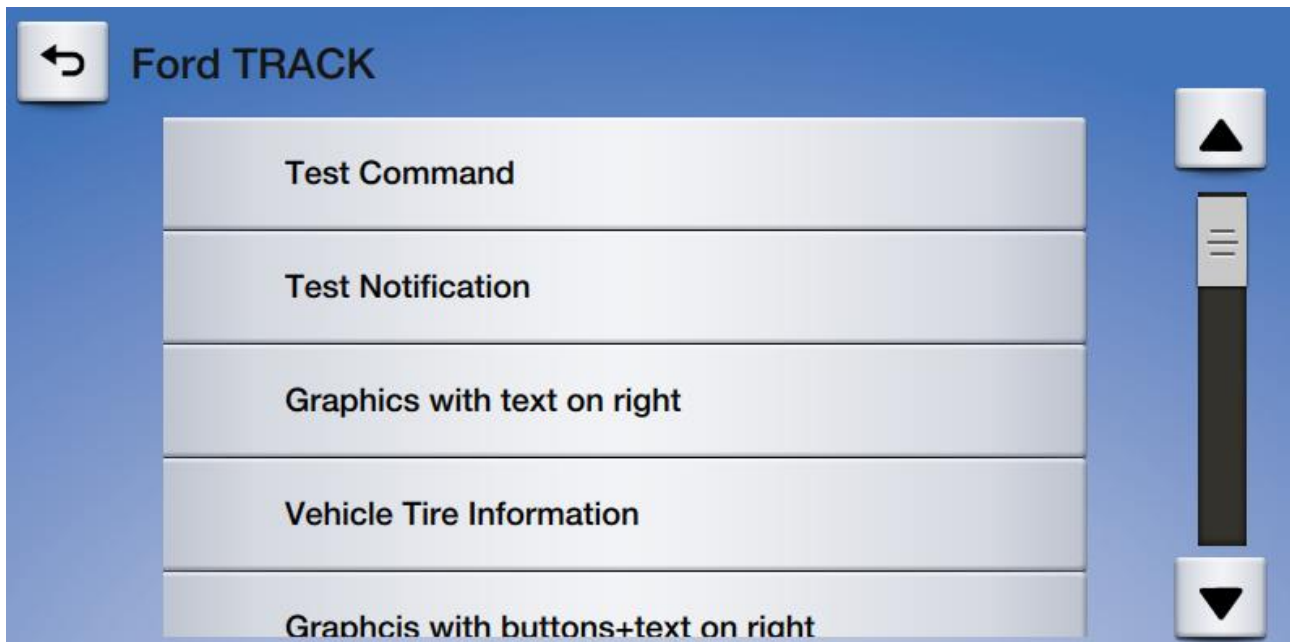


Figure 78. Menu view with a list design screen results






 Ford Track • now <b>Fuel Top Up</b> Your Gas Tank is less than 35
 Ford Track • now <b>Oil Service</b> Left to inspection: 27543
 Ford Track • now <b>Tire Inspection</b> Left to inspection: 20500

Figure 79. Current notification reminder results

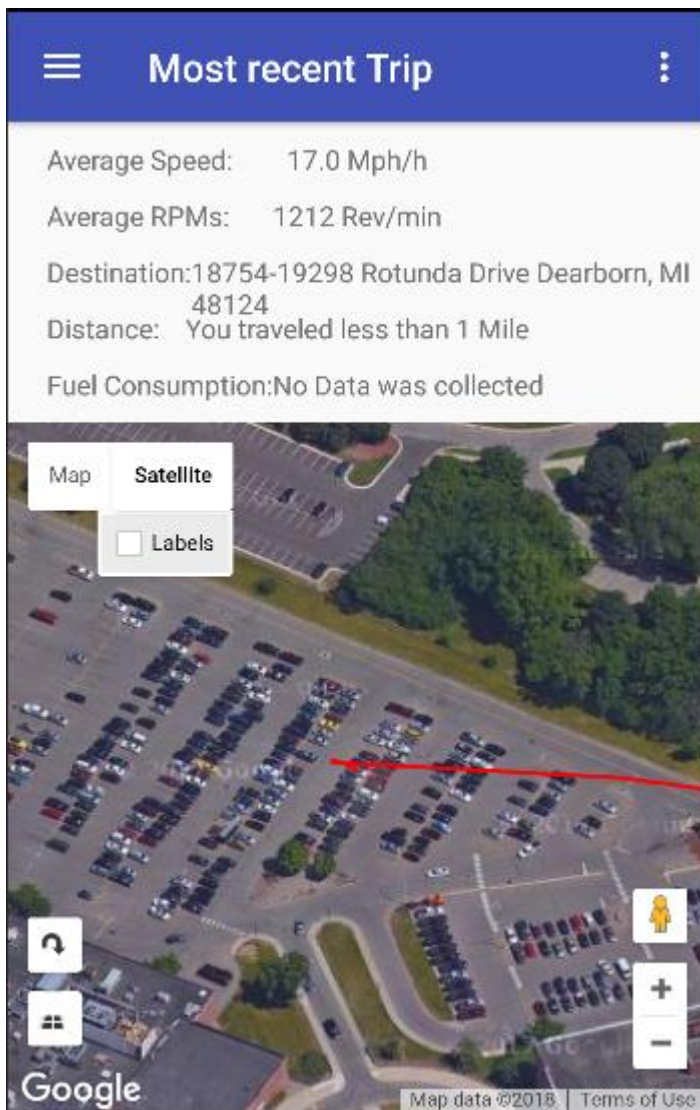


Figure 80. Current implementation of journey review results

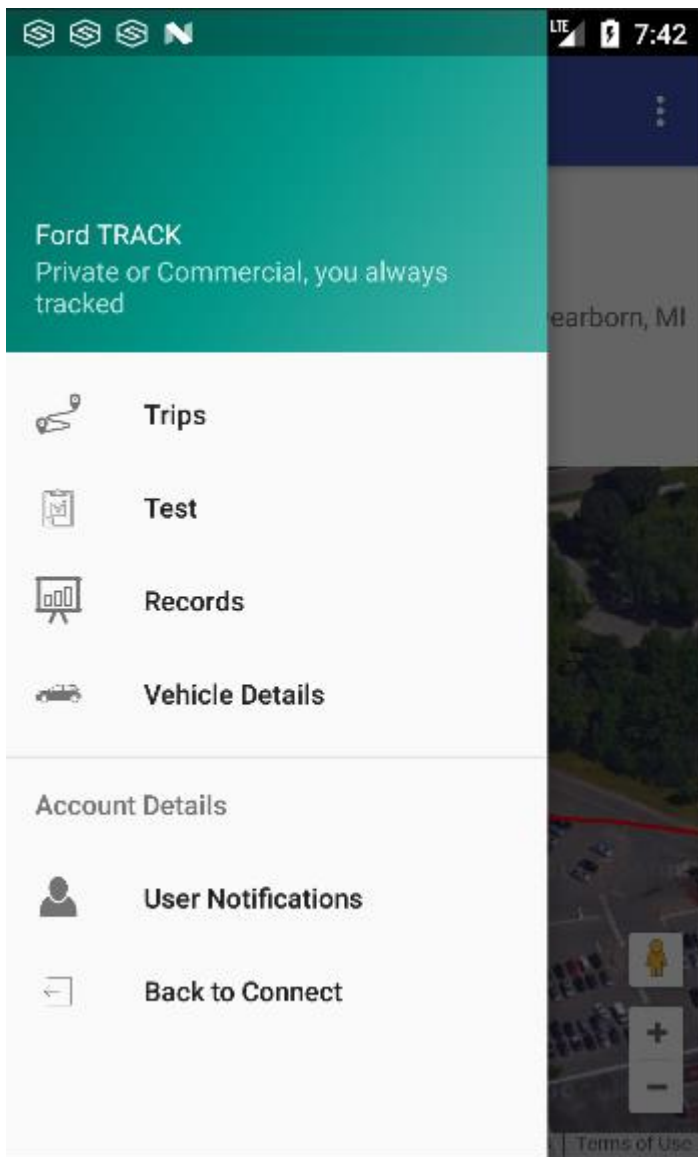


Figure 81. Navigation drawer current implementation results

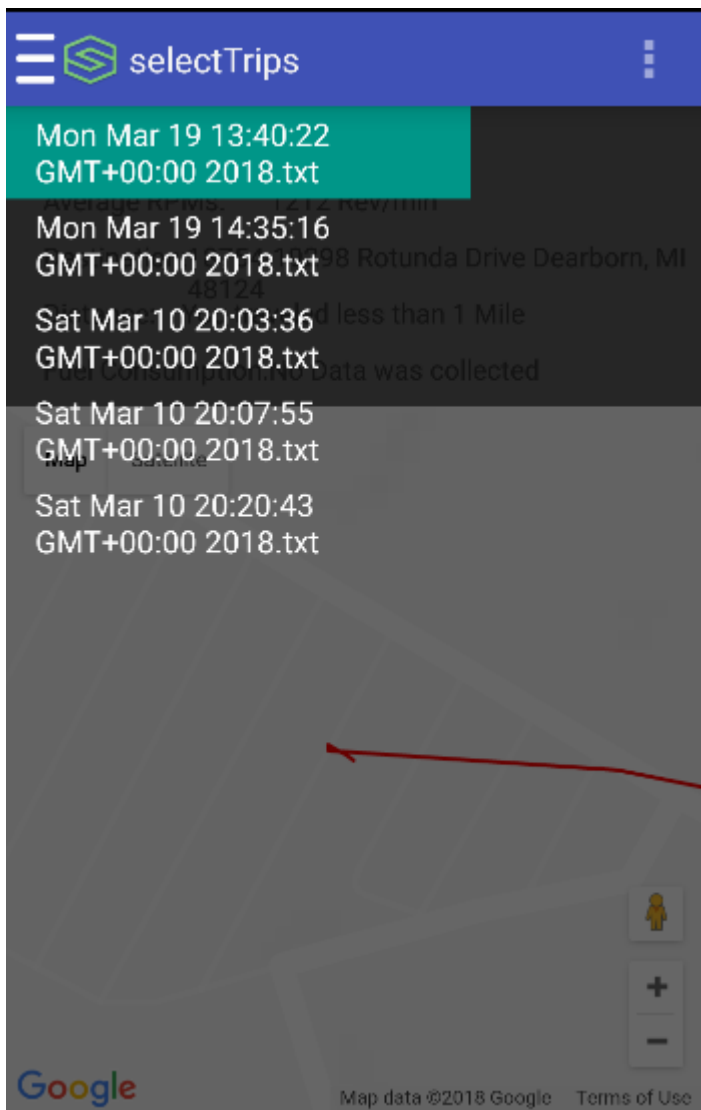


Figure 82. Results of drawer with a list view of saved journeys. Background is journey review screen.

## 6.3 Future work

When designing this project, it was in theory created to be future scalable that new features would be easily added and implemented not just for user interactions but for developer monitoring as well.

Some possible developments:

- **Google Log In:** allowing the user to register and log in with their Google account, allowing developer not just monitor the number of user's applications have but to have their information of application stored on the cloud and monthly overview of data could be emailed.
- **"Records" page:** allowing users to display the monthly history of journeys and highest records. Design of this feature can be seen at 3.6.
- **Driver behaviour implementation:** showing the user, his driving patterns during most frequent journeys or most visited routes.
- **Implementation of Dialogs:** for users to select which journeys to be saved before reviewing finished route information.
- **Implementation of timer:** to let users know how long they spent driving on a set route.
- **Implementation of location stops:** for users to display all locations their vehicle stopped during the journey and how long they spent at that location.
- **Implementation of details:** Additional display where the user could review they journey in most detailed way possible displaying all information retrieved by SDL in tabbed activity on one tab showing information on next one displaying complete map.

## References

- [1] Bengler, K., Dietmayer, K., Farber, B., Maurer, M., Stiller, C. and Winner, H. (2014). Three Decades of Driver Assistance Systems: Review and Future Perspectives. IEEE Intelligent Transportation Systems Magazine.
- [2] M. Aga and A. Ogada, "Analysis of vehicle stability control effectiveness from accident data," in Proc. 18th Int. Enhanced Safety Vehicles-Conf., Nagoya, AI, 2003.
- [3] Burckhardt, Manfred, Franz Brugger, and Andreas Faulhaber. "Anti-lock brake system." U.S. Patent No. 4,861,118. 29 Aug. 1989.
- [4] Anders Lie, Claes Tingvall, Maria Krafft & Anders Kullgren (2006) The Effectiveness of Electronic Stability Control (ESC) in Reducing Real Life Crashes and Injuries, Traffic Injury Prevention, 7:1, 38-43, DOI: 10.1080/15389580500346838
- [5] Yasui, Nobuhiko, and Atsushi Iisaka. "Parking assistance system." U.S. Patent No. 6,483,429. 19 Nov. 2002.
- [6] Sivak, Michael, and Brandon Schoettle. "Drivers on unfamiliar roads and traffic crashes." (2010).
- [7] Labuhn, Pamela I., and William J. Chundrlik Jr. "Adaptive cruise control." U.S. Patent No. 5,454,442. 3 Oct. 1995.
- [8] Marsden, Greg, Mike McDonald, and Mark Brackstone. "Towards an understanding of adaptive cruise control." Transportation Research Part C: Emerging Technologies 9.1 (2001): 33-51.
- [9] ADAS: Features of advanced driver assistance systems. [Online]. Available: <https://roboticsandautomationnews.com/2017/07/01/adas-features-of-advanced-driver-assistance-systems/13194> [Accessed April 2018]
- [10] Justin Hughes. (Nov 3, 2017). Car Autonomy Levels Explained. Thedrive.com. [Online] Available: <http://www.thedrive.com/sheetmetal/15724/what-are-these-levels-of-autonomy-anyway> [Accessed April 2018]
- [11] Renault's developed a full-on, level four autonomous car. And we've driven it. Topgear.com. [Online] Available: <https://www.topgear.com/car-news/big-reads/renault-symbioz-driven-car-future> [Accessed April 2018]
- [12] Olson, Rebecca L., et al. Driver distraction in commercial vehicle operations. No. FMCSA-RRR-09-042. 2009.
- [13] Stutts, Jane C., et al. "The role of driver distraction in traffic crashes." (2001).
- [14] Pettitt, Michael, Gary E. Burnett, and Alan Stevens. "Defining driver distraction." 12th World Congress on Intelligent Transport Systems. 2005.
- [15] Downs, Rick. "Using resistive touch screens for human/machine interface." Analog Applications Journal Q 3 (2005)
- [16] D. Reporter. (2015, July 6). New blood alcohol sensors could make cars shut down if they sense drivers are over the legal drinking limit. DailyMail.com. [Online]. Available: <http://www.dailymail.co.uk/news/article-3113951/New-blood-alcohol-sensors-make-cars-shutsense-drivers-legal-drinking-limit.html>

- [17] A. S. G. D. S. D. Akhila V Khanapuri, "On Road: A car assistant application," in International Conference on Technologies for Sustainable Development (ICTSD-2015), Mumbai, India, 2015.
- [18] J.-W. L. S.-K. L. O.-C. K. Doo Seop Yun, "Development of the Eco-Driving and Safe-Driving Components using Vehicle Information," in 012 International Conference on ICT Convergence (ICTC), Jeju Island/ Daejeon, Korea, 2012.
- [19] ART and Dalvik. Source.android.com [Online] Available: <https://source.android.com/devices/tech/dalvik/> [Accessed April 2018]
- [20] Write the XML. Developers.android.com. [Online] Available: <https://developer.android.com/guide/topics/ui/declaring-layout#write> [Accessed January 2018]
- [21] Google Inc, "Google Material Design," Google, [Online]. Available: <https://developer.android.com/design/material/index.html>. [Accessed February 2018].
- [22] P. Mei, "Application Research of Interface Design Element Analysis for Mobile Platforms," in 2015 IEEE 8th International Conference on Intelligent Computation Technology and Automation (ICICTA), Nanchang, 2015.
- [23] HMI Documentation. Smartdevicelink.com [Online] Available: <https://smartdevicelink.com/en/docs/hmi/master/overview/> [Accessed November 2017]
- [24] Remote Control Procedure Specification. Github.com. [Online] Available: [https://github.com/smartdevicelink/rpc\\_spec](https://github.com/smartdevicelink/rpc_spec) [Accessed April 2018]
- [25] Alisa Priddle and Chris Woodyard, Ford dumps Microsoft for Blackberry for Sync 3. Usatoday.com [Online] Available: <https://www.usatoday.com/story/money/cars/2014/12/11/ford-sync-3/20234131> [Accessed April 2018]
- [26] ListView. Developer.android.com. [Online] Available: <https://developer.android.com/reference/android/widget/ListView> [Accessed February 2018]
- [27] Fragment. Developer.android.com. [Online] Available: <https://developer.android.com/reference/android/app/Fragment> [Accessed February 2018]
- [28] hello\_sdl\_android. Github.com [Online] Available: [https://github.com/smartdevicelink/hello\\_sdl\\_android](https://github.com/smartdevicelink/hello_sdl_android) [Accessed October 2017]
- [29] A sample HMI to use with sdl\_core. Github.com. [Online] Available: [https://github.com/smartdevicelink/generic\\_hmi](https://github.com/smartdevicelink/generic_hmi) [Accessed November 2017]
- [30] Manticore Now Open to the Public. smartdevicelink.com [Online] Available: <https://smartdevicelink.com/news/manticore-now-open-to-the-public/> [Accessed October 2017]
- [31] RelativeLayout. Developer.android.com. [Online] Available: <https://developer.android.com/reference/android/widget/RelativeLayout> [Accessed January 2018]
- [32] ToolBar. Developer.android.com [Online] Available: <https://developer.android.com/reference/android/support/v7/widget/Toolbar> [Accessed February 2018]
- [33] GeoCoder. Developer.android.com [Online] Available: <https://developer.android.com/reference/android/location/Geocoder> [Accessed February 2018]

- [34] ArrayAdapter. Developer.android.com [Online] Available:  
<https://developer.android.com/reference/android/widget/ArrayAdapter> [Accessed February 2018]
- [35] SearchManager. Developer.android.com [Online] Available:  
<https://developer.android.com/reference/android/app/SearchManager> [Accessed March 2018]
- [36] CoordinatorLayout. Developer.android.com [Online] Available:  
<https://developer.android.com/reference/android/support/design/widget/CoordinatorLayout> [Accessed February 2018]
- [37] PrintWriter. Doc.oracle.com [Online] Available:  
<https://docs.oracle.com/javase/7/docs/api/java/io/PrintWriter.html> [Accessed December 2017]
- [38] Join SmartDeviceLink Community on Slack. [Online] Available: <http://sdlslack.herokuapp.com/>  
[Accessed December 2017]
- [39] Write and View Logs with Logcat. Android.developer.com. [Online] Available:  
<https://developer.android.com/studio/debug/am-logcat> [Accessed November 2017]