**INTERNSHIP PROJECT REPORT**

(Project Term January-April, 2018)

## Car Dash and Analytics app on Android Automotive

Submitted by

**Ayush Anand Registration Number: 11403385**

**Course Code: CSE441J**

Under the Guidance of

**Mr. Shetty Sachin Santosh, (Designation)**

**Robert Bosch Engineering and Business Solutions Private Limited (RBEI)**

**CERTIFICATE FROM THE COMPANY**

**CERTIFICATE OF ORIGINALITY**

This is to certify that the project report entitled **“Car Dash and Analytics app on Android Automotive”**, submitted to Lovely Professional University, Phagwara in partial fulfillment of the conditions for the award of B.Tech degree in Computer Science and Engineering from Lovely Professional University, Phagwara is an original work carried out by **Mr. Ayush Anand**, under guidance of **Mr. Shetty Sachin Santosh**. The matter embodied in this project is a genuine work done by Ayush Anand to best of my knowledge and belief and has not been submitted before, neither to this University not to any other University for the fulfillment of the requirement of any course of study.

**Signature of the Student Signature of the Guide**

**Designation**

**ACKNOWLEDGEMENT**

The satiation and euphoria that accompany the successful completion of project would be incomplete without the mention of the people who made it possible.

I would like to take the opportunity to thank and express my deep sense of gratitude to my corporate mentor Mr. Shetty Sachin Santosh and Project manager Mr. Sudheer Kumar L. I am greatly indebted to both of them for providing their valuable guidance at all stages of study, you advice, constructive suggestions, positive and supportive attitude and continuous encouragement, without which it would not have been possible to complete the project.

I owe my whole hearted thanks and appreciation to my entire team for their corporation and assistance during the course of my project.

I hope that I can build upon the experience and knowledge that I have gained and make a valuable contribution towards this industry in coming future.

Ayush Anand

LPU, Phagwara

Punjab

**Name of the Student Signature**

**TABLE OF CONTENTS**

Inner first page ………………………………………………………………….. (i)

Certificate form the Company ………………………………………………….. (ii)

Certificate of Originality……..…….……………………………………………. (iii)

Acknowledgement …………………...…………………………………………. (iv)

Table of Contents……………………...………………………………………… (v)

1. **INTRODUCTION 1**
   1. ANDROID AUTOMOTIVE **1**
2. **PROFILE OF PROBLEM 2**
3. **EXISTING SYSTEM 3**
   1. INTRODUCTION **3**
      1. ENSURING FAST BOOT-UP **3**
      2. EFFICIENT CPU USAGE **3**
      3. DISPLAY MANAGEMENT **4**
      4. DESIGN CONSIDERATIONS IN IVI (IN VEHICLE INFOTAINMENT) **4**
      5. TIME TO MARKET **4**
      6. REUSABILITY **4**
      7. EXTENSIBILITY **5**
   2. WHAT’S NEW IN THE SYSTEM TO BE DEVELOPED **7**
4. **PROBLEM ANALYSIS 8**
   1. PRODUCT DEFINITION **8**
   2. FEASIBILITY ANALYSI**S 8**
      1. TECHNICAL FEASIBILITY  **8**
      2. LEGAL FEASIBILITY **8**
   3. PROJECTPLAN **9**
5. **SOFTWARE REQUIREMENT ANALYSIS 10**
   1. INTRODUCTION **10**
   2. EXTERNAL INTERFACE REQUIREMENTS **10**
      1. USER INTERFACE **10**
      2. HARDWARE REQUIREMENTS **11**
      3. SOFTWARE REQUIREMENTS **11**
   3. FUNCTIONAL REQUIREMENTS **11**
      1. FUNCTIONAL REQUIREMENT 1 **11**
      2. FUNCTIONAL REQUIREMENT 2 **11**
      3. FUNCTIONAL REQUIREMENT 3 **11**
      4. FUNCTIONAL REQUIREMENT 4 **12**
      5. FUNCTIONAL REQUIREMENT 5 **12**
      6. FUNCTIONAL REQUIREMENT 6 **12**
      7. FUNCTIONAL REQUIREMENT 7 **12**
      8. FUNCTIONAL REQUIREMENT 8 **13**
      9. FUNCTIONAL REQUIREMENT 9 **13**
      10. FUNCTIONAL REQUIREMENT 10 **13**
      11. FUNCTIONAL REQUIREMENT 11 **13**
   4. PERFORMANCE **14**
6. **DESIGN 15**
   1. SYSTEMDESIGN **15**
      1. AUTOMOTIVEINTRODUCTION **15**
      2. AUTOMOTIVEARCHITECTURE **16**
      3. CARDASHAPP **16**
   2. PSEUDO CODE **17**
7. **TESTING 20**
   1. FUNCTIONAL TESTING **20**
      1. TESTING EACH FUNCTION INDEPENDENTLY **20**
      2. TESTING THE FUNCTIONS PARALLALLY **21**
   2. STRUCTURALTESTING **22**
   3. LEVELSOFTESTING **22**
      1. TESTINGMANUALLY **22**
      2. TESTING USING SIMULATOR **22**
      3. TESTING INSTALLING IN CAR **22**
8. **PROJECT LEGACY**
   1. CURRENT STATUS OF PROJECT AND REMAINING AREAS OF CONCERN **23**
   2. TECHNICAL AND MANAGERIAL LESSIONS LEARNT **23**
9. **SYSTEM SNAPS 24**
10. **BIBLIOGRAPHY 26**
11. **INTRODUCTION**

Car dash is a simple application designed to replace the old analog and on intuitive dashboards of cars and replace it with new HMI developed on Android OS (Android Automotive) specially built for the cars and displayed on a big displays installed in cars.

* 1. **ANDROID AUTOMOTIVE**

Android Automotive is first of its kind car-focused version of Android OS. It is a fully powered in-car infotainment system with touchscreen console. Android Automotive was a joint effort by Google and partners like Audi and Volvo to launch in the market, an advanced Android OS for car Infotainment. It then slowly evolved and got much more functionalities like Heating, Ventilation and Air Conditioning (HVAC) controls, radio controls and a data of every part can be get using sensors and could then be used for repairs, maintenance, or simple drive pattern analysis.

Android Automotive is an Android-based operating system designed for vehicles. Android Automotive is much more tightly integrated with the features of a car. Depending on the set-up, it can even control various sensors and switches in the vehicle. Android has successfully been deployed in production infotainment (IVI) systems as the primary operating system and also has been used in conjunction with other operating systems. The options for IVI systems includes porting Android to run directly on an SoC as the sole operating system, running Android in conjunction with another operating system such as Linux with use of a hypervisor, or utilizing Android via a Linux Container.

Car infotainment systems can be a mess when it comes to shipping modern technology. Android currently ships in several cars, but Google has never been involved before. Any Android car OSes out there today exist because a car manufacturer grabbed the Android source code on their own and reconfigured it into a car operating system, all without Google's help. The result is usually a lot of development time and something that is very out of date when it hits the market. For instance: the current 2017 Honda Accord runs Android as the car OS, but its version 4.2 Jelly Bean, an OS from 2012. Hyundai is even worse, shipping an Android 2.3 Gingerbread-based system on the 2017 Genesis.

With Google's involvement, car manufacturers will ship much newer versions of Android, and the development time will significantly go down.



Figure 1: Android Automotive based Car (IVI and dashboard)

1. **PROFILE OF PROBLEM**

Create an interactive HMI (Human Machine Interface), to display the sensor data in an interactive and simple way which could be easily understood by people.

The Challenge is to learn how the android automotive system functions by reading all the classes of the dummy apps available with the packages provided by google.

1. **EXISTING SYSTEM**
   1. **INTRODUCTION**

Currently the creating own Intelligent IVI (In-Vehicle-Infotainment) systems is a very tough job. And because different developers can implement different modules diversely, it again becomes difficult to develop applications which can run on all IVI systems.

The following are some of the challenges faced during the design and development of modern IVI solutions.

* + 1. **ENSURING FAST BOOT-UP**

The issue of boot-up time can be a major decision maker when it comes to IVI. The average boot-up time for IVI on the MeeGo platform was approximately 15 seconds. Imagine switching off your engine at a signal and attempting to restart when the signal turns green. An almost-zero delay is preferred, where the instrument cluster becomes functional in less than one second.

While Real-Time Operating Systems [RTOS] are capable of fast boot-up, but a full-fledged Linux OS like MeeGo needs customization to reduce the boot-up time. There are solutions that run a full-fledged Linux OS along with an RTOS which is capable of a fast boot-up to solve this issue for designers, but the designer has to ensure that all the hardware diagnostics are done to ensure smooth operation.

* + 1. **EFFICIENT CPU USAGE**

Instrument Clusters should be able to acquire information from the various sensors on the automobile and render the data immediately on the display without any delay. High-quality graphics require a high amount of CPU usage and processing, unless there’s a dedicated GPU available to crunch those numbers. Multi-threaded applications require an advanced CPU with multi-core architecture that can handle IVI applications efficiently.

* + 1. **DISPLAY MANAGEMENT**

Multiple displays in IVI prove to be a better option for displaying information of several types. For example, the front display shows the instrument cluster, while a movie plays on the left-rear display, and music plays on the right-rear display.

The ability to run numerous applications on a central device and handle multiple displays requires lot of processing, and an efficient graphics processor capable of handling high amounts of rendering. Many IVI-customized boards provide outstanding graphics processing and drivers capable of such operations.

* + 1. **DESIGN CONSIDERATIONS IN IVI**

IVI solutions, still being in a nascent phase, face considerable drawbacks in mass production. Listed below are some of the key design considerations.

* + 1. **TIME TO MARKET**

Every automotive manufacturer’s concern in developing an instrument cluster is to be able to provide a solution that matches the specifications of the car, and meets the expectation of the driver. Car manufacturers that have migrated to the use of digital displays can benefit from the technology innovations available in both IVI hardware and software. These solutions facilitate the customization of instrument clusters for various models of cars without causing delays in the designing of customized dashboards.

* + 1. **REUSABILITY**

While developing custom solutions, reusability is given very less consideration. If base components are reusable, then porting it to new designs, platforms and subsequent revisions become possible. Reusability was not considered as a key aspect in software design primarily because hardware and midlevel software platforms did not follow any specific standards, and varied mostly due to technological differences and cost factors. Aiming to make all base components reusable reduces cost and time-to-market of solutions for OEMs.

* + 1. **EXTENSIBILITY**

The high rate of advancements in technology makes it important to develop solutions that can be extensible. The advancements/upgrades, while adding value, must be capable of being incorporated into existing products and solutions. Constraints in hardware/software may not always allow this to be possible. Creating an entirely new system just to add minor technological improvements is not a workable solution. Solutions must always be extensible for improvement and must be able to include new features and technologies.

* 1. **WHAT’S NEW IN THE SYSTEM TO BE DEVELOPED**

The new app to be developed is on the new platform of google android automotive which gives us access to all sensor reading of the care and control over a lot of things apart from the infotainment system like the HVAC (Heating, Ventilation and Air Conditioning), radio, etc.

This app build will be able to not only picturize the sensor data but also help us analyze the data and inform about some part not functioning properly or if the driver is straining the engine and show the history of the drive.

Google already provide a Kitchen Sink Application with the package which shows usage of most of the classes already prepared by google. It shows some features like real-time sensor readings but in textual format, a low power garage mode, radio controls media controls, HVAC controls and a few more.

But our task is to read and go through all the classes used to build this application and build our own car dash app with the gained knowledge.

Earlier it was the manufacturers who changed the android packages to make the IVI software which were based on android but it led to long development time and old versions being used when several upgrades had already been released.

With android automotive, google is now directly involved and provide support for android to be used as a car’s Operating System. Hence now manufacturers can now make only specific changes like the launcher and some system apps and can simply put it to the car and also it is now possible to provide fast updates as the build time will be significantly reduced.

My app will just be another system app which will be displayed on the dashboard screen of the car.

1. **PROBLEM ANALYSIS**
   1. **PRODUCT DEFINITION**

Car dash will be an application with multiple task but a simple and intuitive UI. It will be representing the sensor data reading in a convenient and simple format and at the same time use the data to keep a record of the past and analyze the reading to give information like driving pattern, average speed, sudden brake, inoperative parts or info about parts not working properly, maintenance info, wheel balance and alignment etc.

* 1. **FEASIBILITY ANALYSIS**
     1. **TECHNICAL FEASIBILITY**

The project will be developed using android. It requires:

* Linux system with android environment set up with a fast SSD and processor and RAM of more than 16GB.
* Latest android version package available on google git to be pulled and kept on the same machine
* Knowledge about linux command line and development of android apk’s and complete package image using linux command line.

All the technical requirements can be easily fulfilled and all packages that are required and open sourced and can easily be pulled from git. Development doesn’t require a big team but just a group of 2 or maximum 3 developers so that the work can be divided and completed faster.

* + 1. **LEGAL FEASIBILITY**

All the packages used are open source and available to all for development and also the tools and OS used for development is open source. No software or packages used for development is got from unknown sources.

The packages are pulled from Google’s official git repository, linux operating system is used with the android development environment set up and both of these can be downloaded from official pages as both the OS and the development environment are open source.

* 1. **PROJECT PLAN**
* Go through already available classes and application file to get knowledge of how the communication between android and the car is established.
* Also getting familiar with the architecture of the android automotive.
* Gain knowledge about how android automotive works and how different layers of the new OS communicate.
* After this building a simple layout and making a successful connection with the car and getting the sensor manager and car info manager.
* Checking if the sensor reading could be received.
* Building widgets for the different sensors like the speedometer widget for the speed sensor and so on.
* Check the working of each widget separately.
* Making a cluster of all widgets into one layout screen and check if each cluster works perfectly together.
* After a successful build of the UI, making algorithms to use the logged data of the sensors and analyze the data to show useful information.
* Graphically representing the historical as well as the real time data from sensors.
* Finally testing the built application and check for its accuracy and bugs.

1. **SOFTWARE REQUIREMENT ANALYSIS**
   1. **INTRODUCTION**

This section will contain all functional and quality requirements of the system. It will give a detailed description of the system and all its features.

* 1. **EXTERNAL INTERFACE REQUIREMENTS**
     1. **USER INTERFACE**

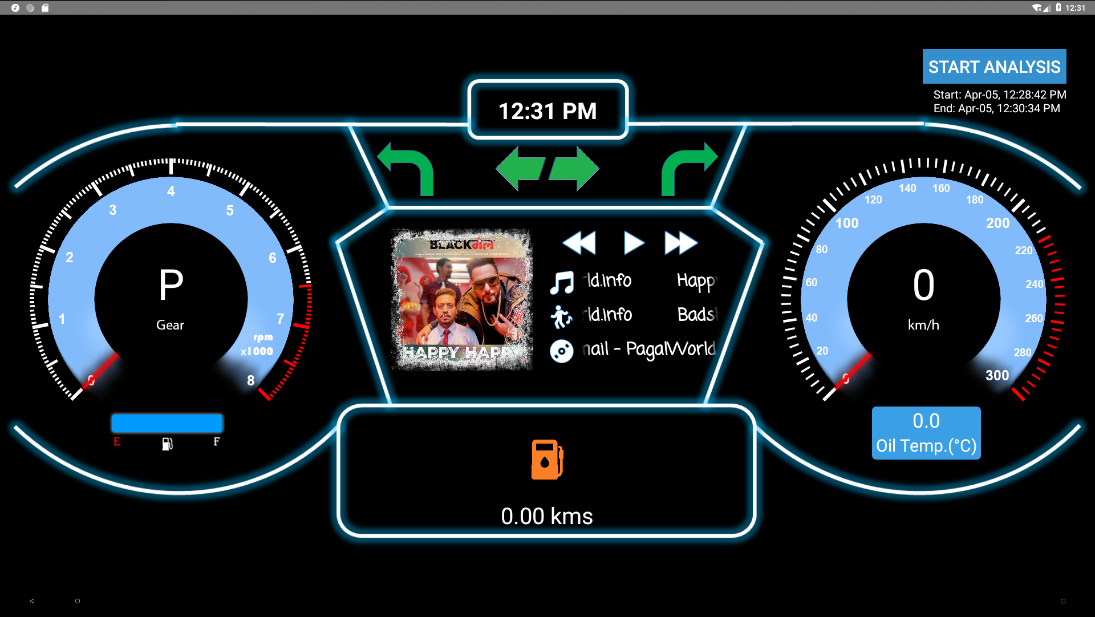
The user will be presented with a mobile like interface as the OS is based on android but will be a minimalistic and simple view so that drivers are not distracted while driving. Car Dash will be a simple system application on his car screen and will give him a traditional car dashboard like view.

Figure 2: Car Dash home screen

All the elements on the screen will actually be a widget which can be touched to see more information about that particular sensor data. E.g. the music can be controlled from the music widget and the fuel widget can show the fuel left in percentage and also that range up to which you can travel with the remaining fuel.

There is also an analysis button which starts the driving analysis and prepares a historical graph of the driving pattern until it is stopped.

* + 1. **HARDWARE REQUIREMENTS**

Because this app is created on android based system all the hardware, the sensors speakers, microphones, GPS, all that is required is present in the car and managed by the manufacturer of the car.

* + 1. **SOFTWARE REQUIREMENTS**

An Android Automotive system with all required middle layers written by the manufacturer or the support provider. A linux based system with android build environment set up for building the application or complete image which can be installed or flashed on the system.

* 1. **FUCTIONAL REQUIREMENTS**
     1. **FUNCTIONAL REQUIREMENT 1**

**ID: FR1**

TITLE: View all sensor data in simple format

DESC: The driver should be able to get the sensor data in a glance and not think much about the reading shown, so that he can concentrate on driving. All the components on the screen will have a graphical view of the sensor reading which is what every driver is familiar with and is quick to understand too.

DEP: None

* + 1. **FUNCTIONAL REQUIREMENT 2**

**ID: FR2**

TITLE: Start/Stop Analysis of the driving pattern

DESC: The driver when wants to analyze the driving pattern he can start the analysis by clicking on the start analysis button on the HMI and the system starts recording the data from the sensors and create a historical graph for easy understanding of the driver.

DEP: FR1

* + 1. **FUNCTIONAL REQUIREMENT 3**

**ID: FR3**

TITLE: Get RPM analysis graph

DESC: The historical as well as real time RPM data is visible as a graphical form. Real time data can be seen when the analysis is started and real time update stops one driver stops the analysis. The graph can be seen by clicking on the rpm widget (RPM meter).

DEP: FR1, FR2

* + 1. **FUNCTIONAL REQUIREMENT 4**

**ID: FR4**

TITLE: Get speed analysis graph

DESC: The historical as well as real time speed data is visible as a graphical form. Real time data can be seen when the analysis is started and real time update stops one driver stops the analysis. The graph can be seen by clicking on the speed widget (Speedometer).

DEP: FR1, FR2

* + 1. **FUNCTIONAL REQUIREMENT 5**

**ID: FR5**

TITLE: View fuel and range data

DESC: The remaining fuel can be seen when this functionality is used with the range the car can travel with the remaining fuel. This should be visible when the driver clicks on the fuel widget.

DEP: FR1

* + 1. **FUNCTIONAL REQUIREMENT 6**

**ID: FR6**

TITLE: Get current music details and control

DESC: The driver is able to see the current playing music details like, title, author, album name and the album art of the song and also get buttons to pause play or change song.

DEP: FR1

* + 1. **FUNCTIONAL REQUIREMENT 7**

**ID: FR7**

TITLE: Zoom in/out and scroll through graph one analysis is complete

DESC: The driver should be able to scroll through the prepared graphs and also should be allowed to zoom in or out to get a clear picture of the data shown.

DEP: FR1, FR2, FR3, FR4

* + 1. **FUNCTIONAL REQUIREMENT 8**

**ID: FR8**

TITLE: Fuel Low indication

DESC: The driver should be notified about the low fuel by simply indicating low fuel animation.

DEP: FR1

* + 1. **FUNCTIONAL REQUIREMENT 9**

**ID: FR9**

TITLE: Turning indications

DESC: When the turning indicator is triggered by the driver same should be notified in the car dash application if it is left, left signal should be shown and vice versa for the right.

DEP: FR1

* + 1. **FUNCTIONAL REQUIREMENT 10**

**ID: FR10**

TITLE: Parking Indicator

DESC: When the user triggers the parking signal same should also be indicated on the car dash application.

DEP: FR1

* + 1. **FUNCTIONAL REQUIREMENT 11**

**ID: FR11**

TITLE: Send historical data to server

DESC: All the data points from the servers is not only represented as graphical data but also stored in local database. These data is send to the server when the car is in garage mode and only if the user allows to.

DEP: None

* 1. **PERFORMANCE**

Because all the sensor reading keeps on changing the app should be capable of handling huge amount of data and also update its UI in real time. With doing so it should also be able to have quick response to touch events and function smoothly.

1. **DESIGN**
   1. **SYSTEM DESIGN**

Development of the application is easy but we need to understand the flow of data of the sensors and how it gets to the android layer and from what class or manager class will we be able to get it and what all do we need to have to be connected to before actually getting the data from the sensors. Let’s first discuss about the architecture of the Android Automotive system.

* + 1. **AUTOMOTIVE INTRODUCTION**

Many car subsystems interconnect with each other and the in-vehicle infotainment (IVI) system via various bus topologies. The exact bus type and protocols vary widely between manufacturers (and even between different vehicle models of the same brand); examples include Controller Area Network (CAN) bus, Local Interconnect Network (LIN) bus, Media Oriented Systems Transport (MOST), as well as automotive-grade Ethernet and TCP/IP networks such as BroadR-Reach.

The Android Automotive hardware abstraction layer (HAL) provides a consistent interface to the Android framework regardless of physical transport layer. This vehicle HAL is the interface for developing Android Automotive implementations.

System integrators can implement a vehicle HAL module by connecting function-specific platform HAL interfaces (e.g. HVAC) with technology-specific network interfaces (e.g. CAN bus). Typical implementations may include a dedicated Microcontroller Unit (MCU) running a proprietary real-time operating system (RTOS) for CAN bus access or similar, which may be connected via a serial link to the CPU running Android Automotive. Instead of a dedicated MCU, it may also be possible to implement the bus access as a virtualized CPU. It is up to each partner to choose the architecture suitable for the hardware as long as the implementation fulfills the interface requirements for the vehicle HAL.

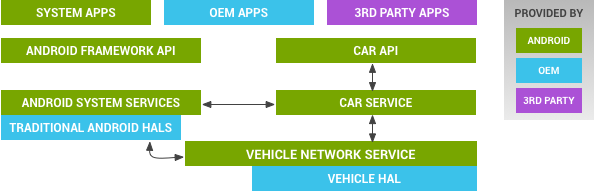
* + 1. **AUTMOTIVE ARCHITECTURE**

Figure 3: Vehicle HAL and Android automotive architecture

* Car API. Contains the APIs such as CarHvacManager and CarSensorManager.
* CarService.
* VehicleNetworkService. Controls vehicle HAL with built-in security. Access restricted to system components only (non-system components such as third party apps should use car API instead).
* Vehicle HAL. Interface that defines the vehicle properties OEMs can implement. Contains property metadata (for example, whether the vehicle property is an integer and which change modes are allowed).
  + 1. **CAR DASH APP**

Car Dash application is a system application which can directly contact the vehicle HAL by connecting to the car using service classes provided by google.

Car service class is used to make connection with the car and once the application has made the connection with the car we can make CarSensorManager and all required sensors and the CarHvacManager to gain access of the sensor data and the HVAC (Heating, Ventilation and Air Conditioning) controls and data about current settings.

This data can then be represented as best suited for the driver of the car and can be used for analysis of the drive. Also we can control the heating ventilation and air conditioning all from one place.

The Car Dash application is built to show the driver all necessary sensor readings required while driving the car in most apt way, so that driver can get the knowledge by just having a quick glance and concentrate on driving.

We have a view to show extra information like the current playing music or maps for navigation whatever the driver requires.

* 1. **PSEUDO CODE**

After going through all the classes and demo applications provided by google this was how the application was supposed to work or in exactly this order we are supposed to move in the application cycle to work properly:

1. Creating an xml file with layout of the view to be created.
2. On creating of the view the application must make a new connection with the car using the **Car** class written in /platform/packages/services/Car/car-lib
3. Create car connection callback
   1. Successful connection – call function to get all managers and add all required sensors.
   2. Failed connection – Show connection failed with car and show restart app message.
4. Create a function **initPermissions** to check for permissions and ask the driver for all required permissions if not yet given and then call function **initSensors**.
5. Create function **initSensors** and instantiate **CarSensorManager** and add all required sensors to it
6. Connection with car is to be made whenever the application opens (**onCreate**) and when it closes or it is destroyed the car connection is to be terminated (**onDestroy**).
7. Create and instantiate 2 handlers one for UI and the other for the rest of the task like (for sending data to database, update time, for background task of analysis feature).  
   **uiHandler** and **restHandler**  
   Also create a **ConcurrentHashMap< Integer,List<Objects> > hm** to store all the sensor data with their Enum integer value already assigned in CarSensorManager class as the key and List of objects as value, because values can be of any type.
8. Create a **refreshUi** function which synchronously gets data from the concurrent hash map hm and correspondingly change the UI. Like changing speedometer and rpm meter needle or displaying turning signal etc. All the updates on the UI is done using uiHandler to keep things smooth and lag free
9. Create **OnSensorChangedListener** and call refreshUi here.  
   This listener will execute each time there is a change in the sensor readings and all the UI will be updated in real time.
10. Create **onTouchEventListener** to handle all touch events and assign required action.
11. Create a **broadcast receiver** to get data of current playing song and for displaying this on the UI. Also display buttons to control music.
12. Create 2 Runnable **clock** and **analyze** first to update time and display on the UI cluster, and the second to take the data reading of some sensors and setting it to graphs as new points and sending these data to database at regular intervals of 200ms.
13. The callback ‘clock’ should be posted to restHandler onResume of the application and removed when application is paused.
14. Create a function start\_analysis to start the storage of sensor data and also to prepare graphs based on it. The function should:
    1. Clear the graph
    2. Post the analyze runnable to restHandler
15. Create a function stop\_analysis to stop the storage of sensor data. The function should remove analyse runnable from restHandler.
16. **TESTING**
    1. **FUNCTIONAL TESTING**

Testing all functions in app independently as well as parallally

* + 1. **TESTING EACH FUNCTION INDEPENDENTLY**

Each functions that do not have any dependency is checked.

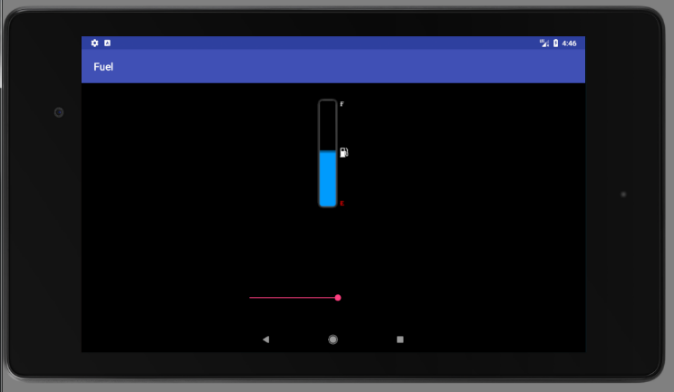
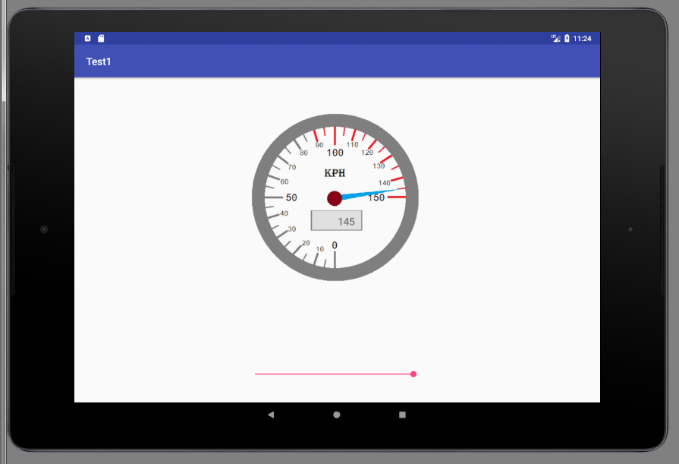
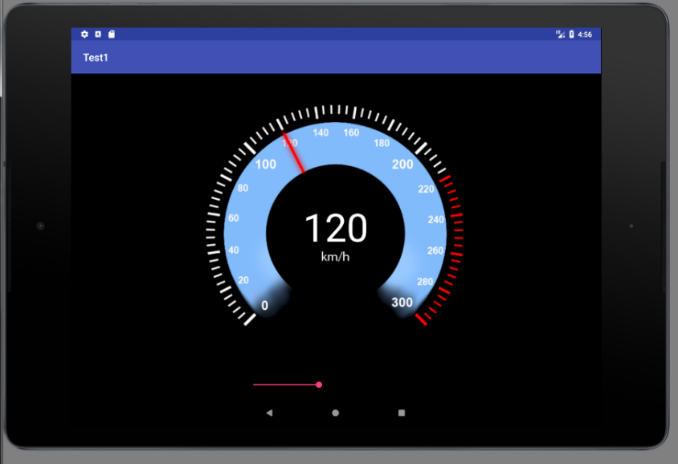
1. Testing all widgets data reading with the current sensors data one by one.
2. Sending multiple sensors data and checking for lag or glitch in the UI
3. Testing touch events.
4. Testing music widget by changing music from app as well as from the music player if music data updated in the app
5. Testing analysis feature if graph updates in real time and then check zooming in and out and swiping through created graph after stopping analysis.

Figure 4: Testing single widgets in emulator

* + 1. **TESTING THE FUNCTIONS PARALLALLY**

1. Allow all sensors to update data on UI and start the analysis and now check for lags and compare reading of widget data and sensor data.
2. Allow the previous functions to still perform and check touch events.
3. Check the graphs for real time update while the analysis is running.
4. Now also run music and check for update in music details when paused, played or changed from either the music player or the application itself.

Figure 5: Testing widgets parallally in emulator

* 1. **STRUCTURAL TESTING**

Simply feeding all sensors data together and running the analysis as well and check for touch events continuously and also change the music frequently and check for any lag or program crash or any bug.

System should be able to function normally and all functions should work properly.

* 1. **LEVELS OF TESTING**
     1. **TESTING MALUALLY**

All the widgets were checked for functioning while creation in the emulator of studio by giving values either by a progress bar or a function.

Then the widgets were tested parallally with other widgets the same way and then finally added to the main cluster.

* + 1. **TESTING USING SIMULATOR**

The application is build and installed onto a test target and the values of the sensors were simulated and sent using another backend Testing Software named Vehicle Simulator. And again checked for any lag jitter or crashes.

* + 1. **TESTING INSTALLING IN CAR**

After being certain that the Application runs properly and all the bugs were removed it is finally installed on a test car and tested there. The car is driven inside campus and checked if all data was displayed and the UI is not distracting and colors and the theme used is not irritating to eyes while driving.

1. **PROJECT LEGACY**
   1. **CURRENT STATUS OF PROJECT AND REMAINING AREAS OF CONCERN**

Currently all the functional requirements are fulfilled except the data being sent to the server (**FR11**). The data is now being saved in the local database and can easily be sent to the server when the car enters the garage mode.

This functionality is been tested though by sending the data to the connected system when testing on the test target using XAMPP server.

Another function as to be added in the next phase that is showing the navigation inside the application when driver is using maps for navigation.

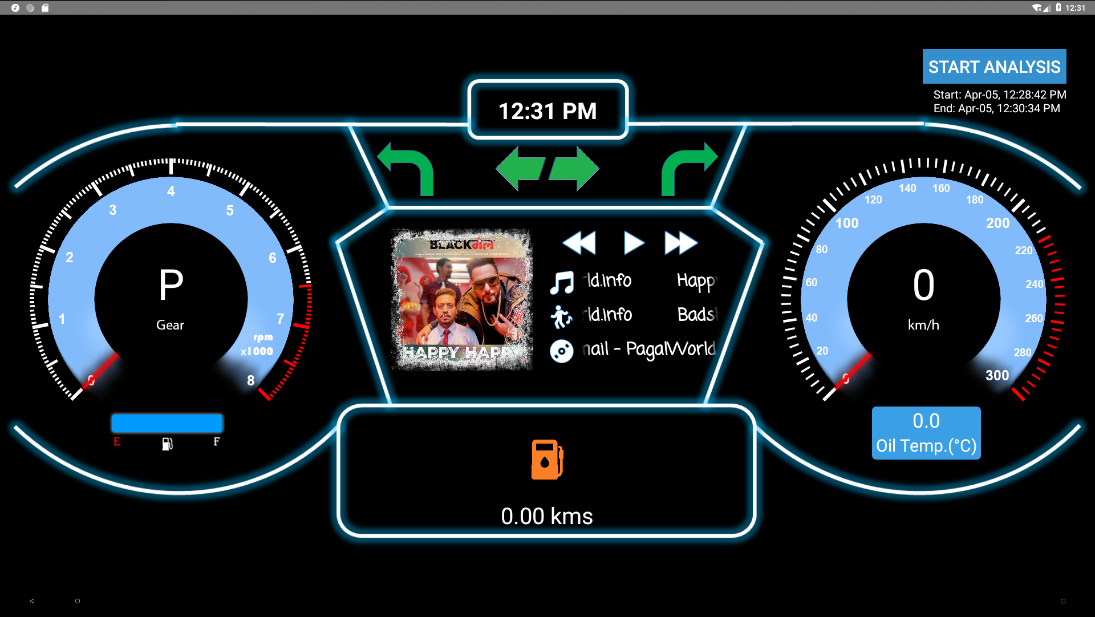
* 1. **TECHNICAL AND MANAGERIAL LESSONS LEARNT**

Working in this project was a challenge as the android automotive system and its working was new to me and also a new field in the market as well. Google provides a package which can be found on the google git but everything needs to be understood reading the classes and no other documentation can be found on the web.

It was because of my Mentor and my team who helped me at every step that I was able to grab the knowledge as quick as possible. Now I know the architecture of Android Automotive and how the data flows between different layers of the system.

I can now develop system apps for android automotive systems which can control heating, ventilation, and air conditioning of car or to get sensor data and make apps to show car configuration like wheel balancing etc. which can help in maintenance or show driver details about his driving pattern.

1. **SYSTEM SNAPS**



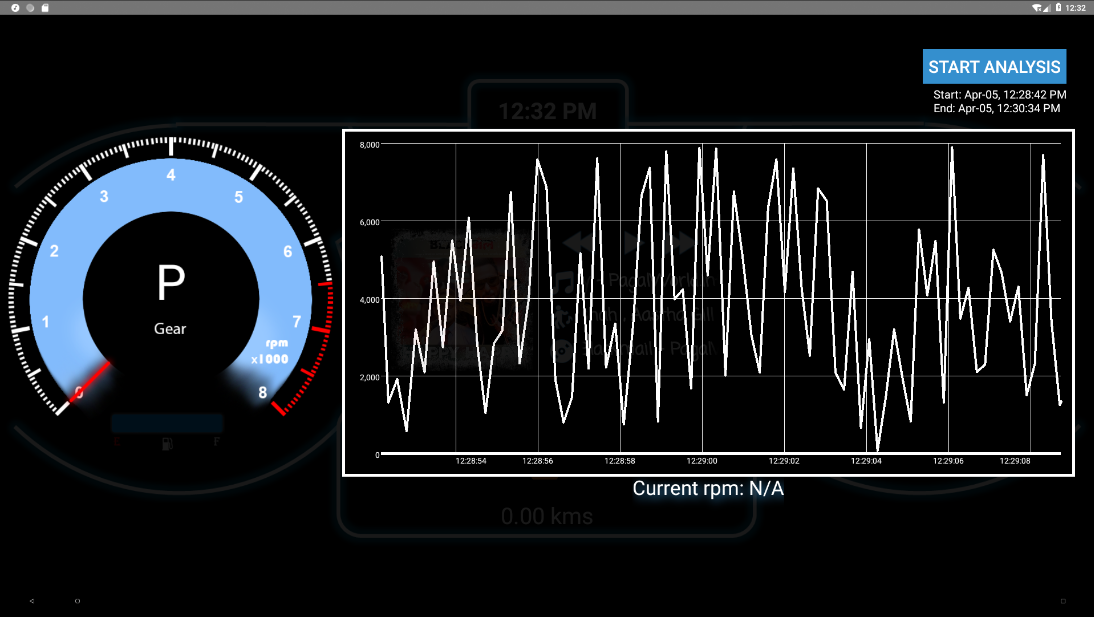
Figure 6: Application Home with all widgets and music control

Figure 7: RPM history graph visible when rpm meter is touched

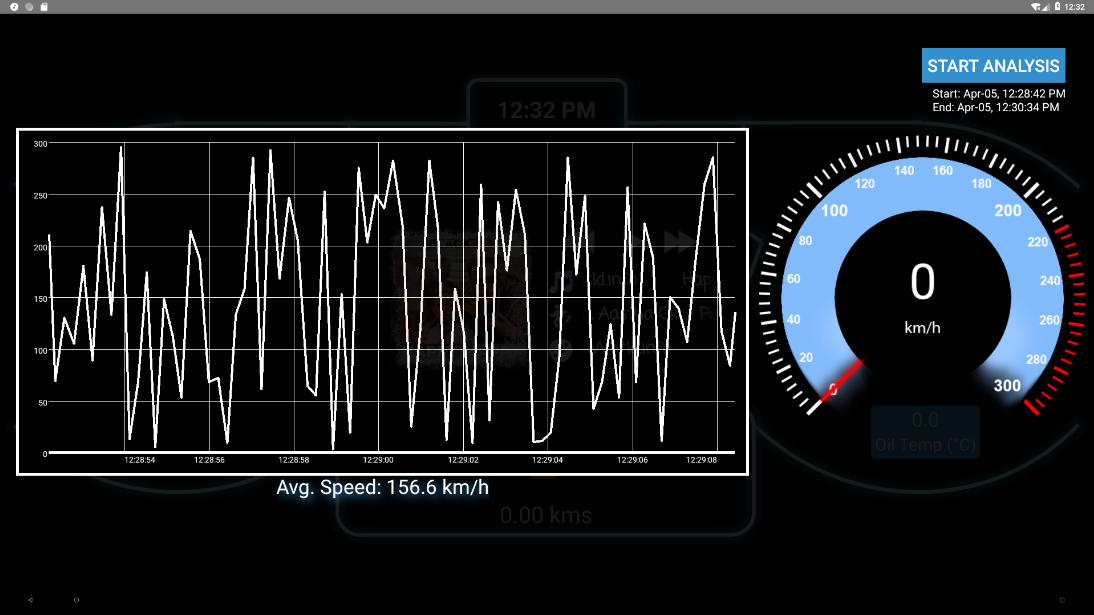


Figure 8: Speed history graph visible when speedometer is touched

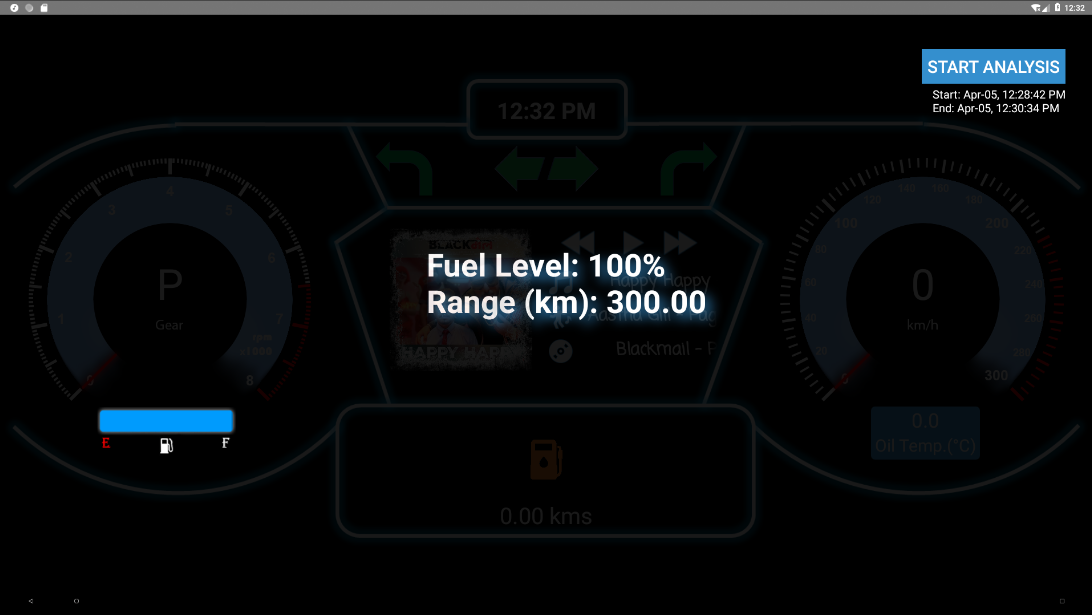


Figure 9: Fuel remaining and range data when fuel widget is touched

1. **BIBLIOGRAPHY**
2. <https://android.googlesource.com/platform/packages/services/Car/>
3. <https://source.android.com/devices/>
4. <https://source.android.com/devices/automotive/>
5. <https://www.android.com/auto/>
6. <https://developer.android.com/training/auto/index.html>
7. <https://www.embitel.com/blog/embedded-blog/no-android-auto-is-not-an-infotainment-os-make-way-for-android-automotive>