Group 02

G.U.A.R.D.



**Supervisors:**

Maria Chiara Lucatello

Mayra Soliz

**Team members:**

Boyan Dai

Erik Laurin

Gabriel Bulai

Joacim Eberlen

Justinas Stirbys

Shaun McMurray

[**1. Introduction**](#_p6qk2ewbj56) **2**

[**2. Backlog Summary**](#_2xf3j8abzr35) **3**

[2.1 Work Flow](#_94q0kg34o4zg) 3

[2.1.1 Sprint 1](#_h3zhv5kpbph4) 3

[2.1.2 Sprint 2](#_9nq7ah8to9dn) 4

[2.1.3 Sprint 3](#_qj7f652u8xf6) 5

[2.1.4 Sprint 4](#_bksaeukg7fvs) 6

[2.1.5 Sprint 5](#_eij95sxqme6q) 7

[2.1.6 Sprint 6](#_p3e6mobzo9rh) 8

[2.1.7 Sprint 7](#_zgoepx8l4s3p) 9

[2.2 Reflections](#_l5zmfowvnlfl) 10

[2.3 Improvements](#_xfa3h0v2mw78) 10

[2.4 Developer Activity](#_8tn75diqo5v3) 12

[**3. Database**](#_ipct60sxmcp8) **13**

[3.1 Table Structure](#_fzp90gghzh5d) 13

[3.2 Purpose](#_1pchl88yxuo4) 14

[**4. UML Diagrams**](#_ctjm2wavddw5) **15**

[**5. Technical Description**](#_phnd3o70yjay) **18**

[5.1 Language Overview](#_6t9e4jurpc9x) 18

[5.2 Class Roles](#_orm0w0j9gz5k) 18

[5.2.1 Java](#_bbhjzqq6ecgc) 18

[5.2.2 Python](#_tjgd9q7hdcpp) 20

[5.2.3 Arduino](#_t3dqmnmh4ihl) 21

[5.2.4 Node.js](#_7e6oefaunwpa) 21

[5.3 Important Functionality](#_s7hf0y4wxyev) 21

[5.3.1 Java](#_3pie4h6o0o9c) 21

[5.3.2 Python](#_3hu36e2jp4ly) 25

[5.3.3 Arduino](#_uber88a6on2z) 25

[5.3.4 Node.js](#_ub060xe436vp) 26

[5.4 Extra Libraries](#_bvs4xx87fx6k) 27

[5.4.1 Python](#_9z3r15o723y1) 27

[5.4.2 Arduino](#_g6ppg7ux20bw) 27

[5.4.3 Node.js](#_kjhuzxupptvd) 28

[5.5 SQL Queries](#_1yocjpqfcojx) 28

# 

# 1. Introduction

The G.U.A.R.D. project mainly aims to create a SmartCar capable of following a traveller through the Gothenburg area by using a GPS module. Secondary features include a manual controller for the purpose of close range movement control, which is expanded with the SmartCar’s camera support. The video feed is provided to the android application for a better overview of the SmartCar’s surroundings.. Lastly, the remaining secondary feature is database incorporation, which is used for the maintenance of user accounts.

The aim of this document is to provide a detailed overview of the project process and the resulting product. The document illustrates the development teams work flow and their experiences throughout the sprint iterations. Additionally, the database and project structure is provided, for the purpose of expanding on the project’s maintainability and the readers’ understanding of the project. UML diagrams are provided to easier illustrate code structure for java and python classes. Lastly, the document contains a technical description section. Overview of individuals classes and important functionality can be found within. Moreover, the section contains a brief description of the purpose for why the different languages were used as well as their dependencies in terms of foreign libraries used within the project.

# 

# 2. Backlog Summary

This section of the manual was established to provide additional traceability of the workflow seen throughout the project course. It examines which user stories have been tackled throughout the project, while briefly describing them and their purpose. The user stories are linked to work item, which a complete list can be found under the Project Planning directory. ([Link](https://github.com/DIT524-V17/group-2/tree/master/documentation/ProjectPlanning))

## 2.1 Work Flow

### 2.1.1 Sprint 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sprint 1 | | | | |
| ID | User Story | Referring Work Item | Target Sprint | Status |
| 1 | As an Admin I want Travelers to create an account, so that I could monitor who is using the SmartCar | User registration | Sprint 1 | Completed on time |
| 2 | As a Traveler I would like to establish a bluetooth connection to the SmartCar, so that I would have access to the SmartCar’s controls | Bluetooth connection | Sprint 1 | Late  Estimated 75% completed |
| 3 | As a SmartCar I would like to have collision prevention, so that I would stop before hitting objects | Collision prevention | Sprint 1 | Completed on time |
| 4 | As a Traveler I would like to have access to the SmartCar’s controls via a mobile application, so that I could control the SmartCar from multiple locations or while traveling | Initial mobile application | Sprint 1 | Completed on time |
| 5 | As a Traveler I want a digital joystick present in mobile application to control the SmartCar’s movement, so that I can specify the movement speed and angle | Analog controller | Sprint 1 | Completed on time |
| 6 | As an Admin I want a test environment to test new features, sto that I could prevent system faults and failures at implementation | Test environment | Sprint 1 | Completed on time |
| 7 | As a Traveler, I would like to see the current battery status of the Smart Car in the mobile application, so that I know how much battery is left | Battery display | Sprint 2 | Estimated 75% completed |

### 2.1.2 Sprint 2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sprint 2 | | | | |
| ID | User Story | Referring Work Item | Target Sprint | Status |
| 8 | As a Traveler I would like to have access to the SmartCar’s location via a mobile application, so that I could view its position in multiple locations or while traveling | Maps | Sprint 3 | Estimated 75% completed |
| 7 | As a Traveler, I would like to see the current battery status of the Smart Car in the mobile application, so that I know how much battery is left | Battery display | Sprint 2 | Completed on time |
| 9 | As a Traveler, I would like to see a warning before the SmartCar hits an object, so that I would get a better understanding of the SmartCar’s surroundings | Parking sensors | Sprint 3 | Estimated 75% completed |
| 10 | As a Traveler I would like to see the SmartCar’s video feed in the app's Controller view, so that I can have easier control of the SmartCar's movements. | Video streaming | Sprint 3 | Too hard to estimate remainder |
| 11 | As an Admin I would like to incorporate the Raspberry Pi with the SmartCar, so that I could increase the potential of the SmartCar | RPi incorporation | Sprint 2 | Completed on time |
| 12 | As a SmartCar I would like to have a continuous connection via WiFI between myself and the mobile device, so that I could handle more heavy data | WiFi Direct connection | Sprint 2 | Completed on time |
| 13 | As an Admin I would like to create a schema and formalize the hardware, so that I could improve the documentation and maintainability of the car | Hardware Schema | Sprint 2 | Completed on time |
| 14 | As a SmartCar I would like to find the correct degrees for turning, so that I could follow the Traveler in the right direction | Degree turning calculation | Sprint 2 | Completed on time |

### 2.1.3 Sprint 3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sprint 3 | | | | |
| ID | User Story | Referring Work Item | Target Sprint | Status |
| 15 | As an Admin I would like to refactor the analog controller to work on all devices, to improve the applications usability | Analog Controller | Sprint 1 | Late  Scale error.  Refactoring |
| 9 | As a Traveler, I would like to see the distance of objects from my SmartCar, so that I would get a better understanding of the SmartCar’s surroundings | Parking sensors | Sprint 3 | Late  Estimated 80% completed |
| 10 | As a Traveler I would like to see the SmartCar’s video feed in the app's Controller view, so that I can have easier control of the SmartCar's movements. | Video Streaming | Sprint 3 | Late  Estimated 75% completed |
| 8 | As a Traveler I would like to have access to the SmartCar’s location via a mobile application, so that I could view its position in multiple locations or while traveling | SmartCar coordinates | Sprint3 | Completed on time |
| 16 | As a SmartCar I would like a gyroscope implemented, so to that I know the relative direction I am heading in | Gyroscope implementation | Sprint 4 | Estimated 67% completed |
| 17 | As a SmartCar I would like to have the ability to follow Travelers, so that I could ensure their safety | GPS coordinates,  GPS following | Sprint3/ Sprint 6 | Completed on time/ hard to estimate |
| 18 | As an Admin I would like to establish code conventions, so that in the future the project is more maintainable | Code convention refactoring | Sprint 3 | Continues throughout all project |

## 

## 

### 2.1.4 Sprint 4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sprint 4 | | | | |
| ID | User Story | Referring Work Item | Target Sprint | Status |
| 8 | As a Traveler I would like to have access to the SmartCar’s location via a mobile application, so that I could view its position in multiple locations or while traveling | Maps | Sprint 3 | Late. Found error  Estimated 90% completed |
| 10 | As a Traveler I would like to see the SmartCar’s video feed in the app's Controller view, so that I can have easier control of the SmartCar's movements. | Video Streaming | Sprint 3 | Completed a sprint late |
| 17 | As a SmartCar I would like to have the ability to follow Travelers, so that I could ensure their safety | GPS following | Sprint 6 | Estimated 60% completed |
| 16 | As a SmartCar I would like a gyroscope implemented, so to that I know the relative direction I am heading in | Gyroscope Implementation | Sprint 4 | Completed on time |
| 18 | As a SmartCar I would like to use the LIDAR and the gyroscope for obstacle avoidance, so that I could avoid obstacles | LiDAR turning script | Sprint 5 | Estimate 50% completed |
| 19 | As an Admin, I would like to refactor the parking sensors, so that the application is more user friendly and oriented | Parking sensors | Sprint 3 | Late  Refactoring for user friendliness |

# 

# 

### 2.1.5 Sprint 5

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sprint 5 | | | | |
| ID | User Story | Referring Work Item | Target Sprint | Status |
| 20 | As an Admin I want to refactor the Contoller Activity, so that I could lower its complexity making the application more maintainable | Refactor Controller Activity | Sprint 5 | Completed on time |
| 8 | As a Traveler I would like to have access to the SmartCar’s location via a mobile application, so that I could view its position in multiple locations or while traveling | Maps, SmartCar coordinates | Sprint 6 | Difficulties sending data. Target moved to Sprint 6. Estimate 74% done |
| 17 | As a SmartCar I would like to have the ability to follow Travelers, so that I could ensure their safety | Send commands to SmartCar to follow phone | Sprint 5 for this work item. Sprint 6 overall | Late. Estimated 80% done |
| 21 | As an Admin I would like to save relevant information in a remote database, so that I could improve data security | Node.js server | Sprint 6 | Estimated 25% done |
| 19 | As an Admin, I would like to refactor the parking sensors, so that the application is more user friendly and oriented | Parking sensors | Sprint 3 | Completed 2 sprints behind  Refactored for user friendliness |
| 18 | As a SmartCar I would like to use the LIDAR and the gyroscope for obstacle avoidance, so that I could avoid obstacles | LiDAR turning Script, Rotate car to specific angle | Sprint 5 | Late. Estimated 75% done. More difficult than initially thought |

## 

## 

### 2.1.6 Sprint 6

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sprint 6 | | | | |
| ID | User Story | Referring Work Item | Target Sprint | Status |
| 8 | As a Traveler I would like to have access to the SmartCar’s location via a mobile application, so that I could view its position in multiple locations or while traveling | Maps, SmartCar coordinates | Sprint 6 | Completed on new time |
| 10 | As a Traveler I would like to see the SmartCar’s video feed in the app's Controller view, so that I can have easier control of the SmartCar's movements. | Video Streaming | Sprint 7 | Fixing issue with router.  New estimate 80% done |
| 17 | As a SmartCar I would like to have the ability to follow Travelers, so that I could ensure their safety | GPS following | Sprint 6 | Late. Estimated 70% done |
| 21 | As an Admin I would like to save relevant information in a remote database, so that I could improve data security | Node.js server | Sprint 6 | Completed on time |
| 22 | As an Admin I would like a Navigation Drawer in the app, so that I could offer a more user friendly UI | Navigation Drawer | Sprint 6 | Late. Estimated 60% done |
| 23 | As an Admin I would like UML diagram, so I can Maintain the software more easily | UML diagrams | Sprint 7 | Estimated 50% done |
| 24 | As a Traveler I would like to know what functionalities are available to me with or without a connection to the car, so that I could have a better understanding of the available SmartCar features | SmartCar function control | Sprint 7 | Estimated 50% done |
| 25 | As a Traveller I would like to access to my profile, so that I could maintain the most relevant information | User profile | Sprint 6 | Completed on time |
| 26 | As an Admin I would like to have better connectivity overview of the SmartCar so that the users could connect to it more easily | Information overview | Sprint 7 | Estimated 50% done |
| 2 | As a Traveler I would like to establish a bluetooth connection to the SmartCar, so that I would have access to the SmartCar’s controls | Bluetooth connection | Sprint 6 | Fixed connection error. Found in sprint 4 |

### 2.1.7 Sprint 7

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sprint 7 | | | | |
| ID | User Story | Referring Work Item | Target Sprint | Status |
| 22 | As an Admin I would like a Navigation Drawer in the app, so that I could offer a more user friendly UI | Navigation Drawer | Sprint 6 | Completed a sprint late |
| 27 | As an Admin I would like to offer a developer manual as documentation, so that I could make the project system more understandable and maintainable | Developer Manual | Sprint 7 | Completed on time |
| 28 | As an Admin I would like to offer a user manual as documentation, so that I could make the project more user friendly and understandable | User Manual | Sprint 7 | Completed on time |
| 24 | As a Traveler I would like to know what functionalities are available to me with or without a connection to the car, so that I could have a better understanding of the available SmartCar features | SmartCar function control | Sprint 7 | Completed on time |
| 26 | As an Admin I would like to have better connectivity overview of the SmartCar so that the users could connect to it more easily | Information overview | Sprint 7 | Completed on time |
| 17 | As a SmartCar I would like to have the ability to follow Travelers, so that I could ensure their safety | GPS following | Sprint 6 | Completed a sprint late |
| 8 | As a Traveler I would like to have access to the SmartCar’s location via a mobile application, so that I could view its position in multiple locations or while traveling | Map Markers | Sprint 7 | Revamped Map maker.  Completed on time |
| 29 | As an Admin I would like to provide a Vision and Budget of the project, so that I could provide a better understanding of the project | Vision and Budget | Sprint 7 | Completed on time |
| 30 | As an Admin I would like to provide documentation on the user interface test used, so that I could improve maintainability | Test case documentation | Sprint 7 | Completed on time |
| 31 | As a SmartCar I would like to map my surroundings in the room, so that I could avoid obstacles | LiDAR mapping | Sprint 7 | Completed on time |

## 2.2 Reflections

This section revolves around developer comments surrounding the project’s workflow. The purpose of which is to elaborate and detail some of the common pitfalls that arose within the development process in order to avoid these same mistakes in the future. Most time consuming mistakes were resulted due to a lack of acceptance criteria and poor work division. Moreover, a lack of software structure and quality resulted in time relocation to refactor certain tasks.

Developers must set a unified definitions for done to improve the project’s outcome. Little focus was put on planning at the beginning of the project. This resulted in differing definitions amongst developers of what it means to be completely done with an assigned task. The lack of unified completion criteria resulted in multiple tasks from the first iterations of the project to later be modified via refactoring sessions. Acceptance criteria was begun to be set with the aid of all developers around the third iteration of the project. This had resulted in visible improvements of what was required of the developers. The time spent refactoring due to this issue could have been diminished by developers establishing unified standards at the inception phase of the project.

A sensible way of dividing work items has a significant impact on the development process. As seen in the workflow, new items were added to the sprints before some of the old items were fully completed. Although, this is most likely better illustrated within the sprint iteration plans. This was due to developers working on user stories separately, unless the stories are rather large. This issue has significant consequences if multiple developers and/or user stories require to work on the single SmartCar. Furthermore, more problems arose by developers not seeking aid when needed. This communication issue resulted in low productivity. A solution to this problem would be to incorporation of pair programming. The tasks found be completed quicker by assigning multiple developers to it. Additionally, by improving project planning the effects of this issues could be diminished. This would ensure that avoidable clocking states do not occur within the project.

Lastly, for a successful project a software structure and definition of quality need to established early on. Multiple items in the workflow required refactoring to comply to the established quality threshold. For example, refactoring ControllerActivity was aimed at reducing the cyclomatic complexity to make the class more maintainable. A similar effect can be seen with project structure. Items pertaining to the initial mobile application required refactoring to improve the android application structure. A possible solution would be doing additional research into software architecture coupled with a focus on technical depth before beginning to code. This would aim to produce a maintainable and reliable application while avoiding common pitfalls. Additionally, the developers should aim to establish and follow deadlines more closely. By integrating the tasks on specific deadlines, a quality analysis and group code inspection could be performed in regular intervals. Another possible solution would have been to ask the teaching aids for android development guidance, which would have hopefully lead to a better understanding of software architecture in the android environment,

## 2.3 Improvements

Several ideas for possible improvements to the process were brought up after an evaluation of the project process and results. These improvements are geared towards planning, communication and assistance for developers. Specific focus was put on these areas, as they were concurrently identified as the most problematic by the development team.

The opening suggestions are specifically aimed towards improving the evaluation phase of the project. In retrospect, the inception phase was not utilized to its utmost extent. Specifically, more time should have been spent on researching desired features and how to make them come to fruition. The knowledge gained from research would be used to evaluate the feature feasibility and if needed rethink the approach. The feasibility examination would be followed be making a more established initial plan. The plan would center around estimating and managing the time required to achieve “feasible” features. In particular, which features are the least riskiest, how many developers would require to work on it and for how long, and how to balance the individual sprint based on the overall time estimate. Lastly, before the development phase begins time would be allocated to establishing modular definitions for software architecture and quality. The acceptance criteria and definitions of done would be established during the initial sprint planning meeting.

Other planning improvements are meant to increase the productivity of sprint planning sessions. Sprint iteration plans were deemed quite beneficial and it was decided to incorporate them in future projects. However, the iteration plans were only incorporated towards the end of this project. In the future, these plans would be originally made during the initial scrum meetings. The plans would contain the work items and estimated hours. At the end of the sprint additional information would be added to the plans detailing their current status and the actual hours spent working on the tasks. This information would be used to further improve the project plan and estimate the project status. Additionally, it was thought that more time should be spent on incorporating practices with the project. Pair programming could be easily incorporated when assigning multiple people to the same work items. Release management could be incorporated with aid of git, which would help improve the project traceability. Code inspection and code review should be used in regular intervals to make sure the that each developers has an understanding of the code. It is believed that these practices would help yield positive results in terms of quality and architecture, which should be the main focus as opposed to the amount of features.

Several suggestions for improving the means of communication were made as well. Based on the project experience it seems that face to face meetings seem to be the most fruitful. As these meetings are able to provide definite answer on the current state of affairs and project status, Furthermore, the team thought it would be beneficial to expand on the use of Trello. Specifically, make sure to keep the most up to date information, to use percentage indicators to show the progress for tasks. Additionally, Trello was thought to be useful in terms of keeping task information. In particular, acceptance criteria, definition of done, and brief description of the task. However , both of these improvements are dependent on the team members continuously keeping track of the communication channels.

Lastly, it was thought that the developers would benefit from reaching out for guidance. The development team did not utilize the resources provided for them effectively. For example, the team rarely reached out to the teaching assistants for guidance. In the future, the development team will reach out more to possibly learn from the teaching assistant experience. By doing so perhaps some common pitfalls could have been avoided and possibly a better perspective could have been obtained. Moreover, the team thought that they could rely more on each other. This could be done be seeking help more actively, something that was not done often. By doing so the developers could gain a fresh perspective on their tasks. Although this is somewhat hard to do seeing as how it would require someone to postpone their work and time. The developers could also have asked the teaching assistants for guidance in terms of software development as well.

## 2.4 Developer Activity

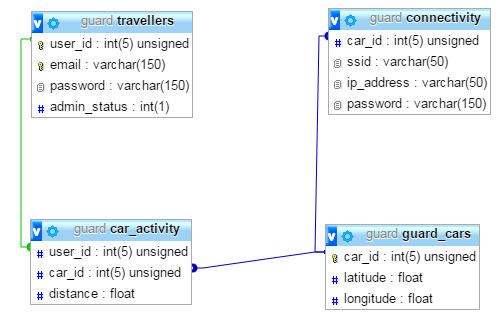
This section aims to provide information of developer meetings. The detailed information about the team meetings can be found under the Meetings board on Trello. [Link](https://trello.com/b/F38nP7MZ/meetings)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Missed meetings | Sprint 1 (out of 7) | Sprint 2 (out of 7) | Sprint 3 (out of 4) | Sprint 4 (out of 6) | Sprint 5 (out of 6) | Sprint 6 (out of 5) | Sprint 7 (out of 6) | Total (out of 41) |
| Axel Granli | 1 | 1 | 1 | Drop out | Drop out | Drop out | Drop out | 26, dropped ou |
| Boyan Dai | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 2 |
| Erik Laurin | 1 | 0 | 0 | 0 | 2 | 0 | 0 | 3 |
| Gabriel Bulai | 0 | 0 | 0 | 0 | 2 | 2 | 1 | 5, due to work |
| Joacim Eberlen | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 5, due to work1 |
| Justinas Stirbys | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Shaun McMurray | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 2 |

# 

# 

# 3. Database



## 3.1 Table Structure

The ***travellers*** table is the most used table in the database. The purpose of which is to maintain G.U.A.R.D. user data. The table consists of 4 fields. The first one, “user\_id” is an integer acting as a primary key, a unique identifier for the table. The field was given an “unsigned” attribute, as the id will never be a negative integer”. Additionally, the “auto\_increment” attribute was used for the purpose of removing responsibility of changing id values from the developers hands. These two attributes are used together to save space and computational power from within the database. A unique identifier was needed in order to recognize different users during registration/login. The “email” field was designated as unique for this purpose, since the the application users are unaware of the user id throughout the registration. The field “password” stores the user entered password. Both the password and email are hashed before being added to the “travellers” table, therefore their type is the to varchar. The “admin\_status” tracks the users’ privileges. A simple one digit is used to identify if the user is an admin, hence the fields short length.

This ***guard\_cars*** table is used for the purpose of storing the project SmartCars’ locations for the retrieval from remote locations.The “car\_id” field acts as a primary key used to identify individual SmartCars. Similarly to “user\_id” the car identification value will never be a negative integer, therefore it was set to “unsigned” to save space. The fields “latitude” and “longitude” refer to individual SmartCar coordinates and are therefore assigned float values.

The ***car\_activity*** table is intended for admin use. The table was created for admins to keep track of which users have utilized the SmartCar service and for how long. The table’s use is mainly geared towards a future, where hypothetically the G.U.A.R.D. project has access to more than a single SmartCar. The “user\_id” is a foreign key which refers to the “travellers” table and is used to keep tack of the user identity. Similarly, the “car\_id” is foreign key refering to “guard\_cars” table for the purpose of tracking which car was used. The “distance” field was set to float, so it could track how far the car has driven.

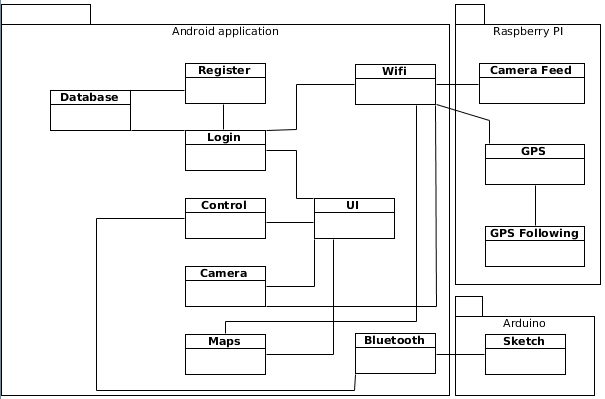
The ***connectivity*** table stores WiFi connection information of different SmartCars. To keep track of the network name “ssid” is used. The “ip\_address” and “password” field are rather self-explanatory. They are used to keep track of the of the router’s ip and password respectively. The network password is hashed, hence it’s length is larger than the other fields. The “car\_id” is a foreign key and links to the “guard\_car” table for the purpose of identifying network information for specific SmartCars’.

## 3.2 Purpose

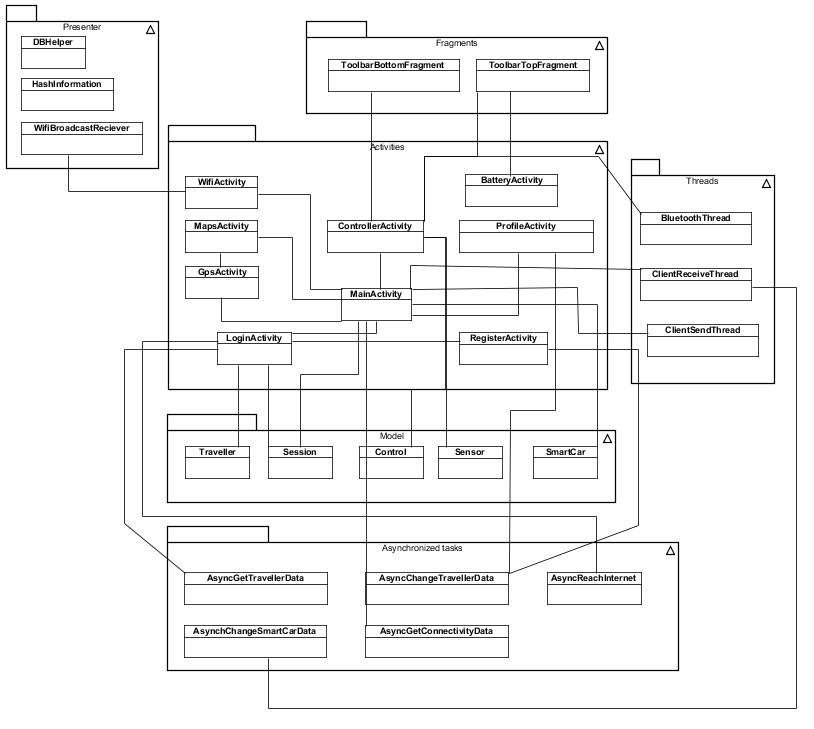
The G.U.A.R.D. project incorporates a database in order to provide more value to the consumer. The main reason behind database incorporation with the project was to expand on its capabilities by providing additional features, such as user registration. Additionally, the SmartCars’ locations are stored in the database, so that users are able to access and/or see the SmartCars through multiple remote locations or while traveling, eliminating the dependencies of bluetooth and wifi connections to the SmartCar itself. Moreover, traveller information and SmartCar usage is stored in the “guard” database for liability purposes in case of damage to the G.U.A.R.D. car. Lastly, the information is held more securely on a remote database and it lowers the memory usage of the application user devices.

# 4. UML Diagrams

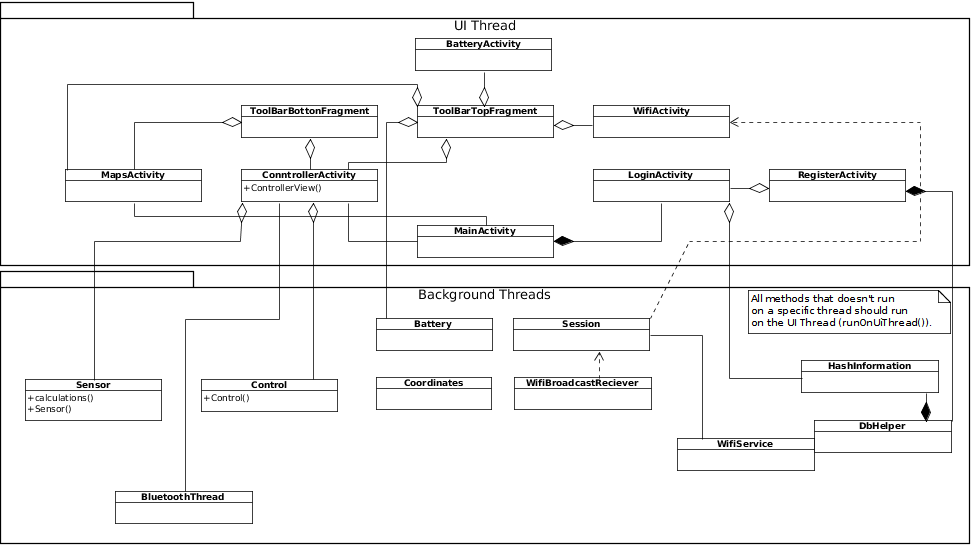
The following diagram demonstrates the overall project domain. The domain model is a simplified version of the application aimed to visualize the project’s capabilities and functionalities, and the connection amongst them.



The following figure shows the final class diagram reflecting of the project structure, which unfortunately is lacking.The diagram showcases the activities, fragments, presenters and miscellaneous tasks used within the project.



The following figure shows the interactions between the UI and background threads for the android application.



# 

# 

# 5. Technical Description

## 5.1 Language Overview

For the project to achieve its desired functionality the development team opted to incorporate 4 different languages. The languages include Java, Python, Arduino and Javascript.

***Java*** was used to create the mobile application. The application was built using the Android Studio IDE. The application handles all of the travellers interactions with the SmartCar. Some functions handled by the application; a manual controller, the video stream, GPS following, a Map that marks the location of the traveller and all of the SmartCars. WiFi and Bluetooth were used to establish a connection from SmartCar to our application.

***Python***was used for GPS following of a user as well as object avoidance when following a user. Mapping of a room was created using python. Communication between the application and the arduino were important to achieve the goals

***Arduino*** was used for SmartCar movement control. Specifically, the SmartCar is controlled by a microcontroller, an Arduino Mega, which runs Arduino code (C/C++). The code was developed using Arduino IDE and Atom. In accordance with the format of runnable Arduino code, our code contains both a setup as well as a loop method. At boot of the micro controller, the setup method is automatically executed. Thereafter, the loop method start to execute repeatedly until the micro controller is turned off or until it receives a new sketch. Thus, all code that ought to be executed must somehow be called from the loop method.

***JavaScript*** was used to build the Node.js section of the project. It is comprised of 4 main parts. These parts are differing files and are comprised of “package.json”, “server.js”, “guardRest.js” and lastly “authentication.js”. Combined these files create a node.js framework that is used to communicate between the android application and the MySQL database, in which the information is stored.

## 5.2 Class Roles

### 5.2.1 Java

The brunt of the application was made using java. The java code can be split up into different sections to hopefully better represent the architecture. The sections include activities, fragments, threads, asynctasks, helper classes and lastly object models.

Activities:

The activities comprise the majority of the application. They have accompanying xml files that create the GUI of the application. Each of the classes represents a different view and functionality. Activities include:

***BatteryActivity,*** which gives an overview of the stats of the battery pack that is attached to the SmartCar.

***ControllerActivity*** that contains the manual control, video feed and displays the ultrasonic sensors to the traveller.

***LoginActivity*** a basic login screen requesting account credentials and providing an option to register.

***MainActivity*** contains the main menu. All of the functions can be reached from here, as long as the traveller has a connection to the SmartCar through Bluetooth or WiFi.

***MapsActivity***, A map view that marks the coordinates for the SmartCars and the phone using the application.

***NavigationActivity*** a deprecated class that creates a navigation drawer which displays an option menu.

***ProfileActivity*** handles the changes the traveller want to apply to their profile, this includes updating email and creating a new password or removing their account altogether.

***RegisterActivity*** a basic register view, where the traveller can create their account by providing an email and password.

***WifiActivity*** another deprecated activity that was used during early development to establish a WiFi-Direct connection between the android application and the SmartCar.

Fragments:

Fragments in this application are created as parts of the UI in the activities, they are retainable to make sure they save the state they are in when turned and even the handling of closing and reopening activities.

***ControllerFragment*** a deprecated fragment that is the counterpart for the ControllerActivity, meaning it is intended and used to accomplish the same purpose.

***MainFragment*** a deprecated fragment counterpart to MainActivity, used to provide the main menu within the app.

***ProfileFragment*** a deprecated fragment that handles the changes the traveller want to apply to his/her profile, this includes updating email and creating a new password.

***ToolbarBottomFragment*** is the toolbar at the bottom of the screen, it contains intents for changing activity to the main menu, the gps activity or the maps activity.

***ToolbarTopFragment*** creates a simple toolbar that houses app title and SmartCar battery display.

Threads:

Threads, as usual, handle the background work of the application. In the G.U.A.R.D application the threads with classes are aimed towards data transfer between multiple devices.

***BluetoothThread***: this thread handles the reading and writing over bluetooth. It connects through the UUID and the MAC-address.

***ClientReceiveThread***: thread used to receive and read the location from the Raspberry Pi using a Java Server. It opens a TCP socket which is used for the transmission of data from and to the Pi.

***ClientSendThread***: thread used to send current phone location updates using the same Java server located on the Pi.

AsyncTasks:

Asynchronous tasks were used when smaller background work needs to be conducted. In this project, the async tasks are used for interaction with the remote database to alter and maintain the stored information.

***ChangeTravellerData*** this task is used to create, update and delete the traveller's account information, such as email and password. The information is stored within the “travellers” table.

***ChangeSmartCarData*** changes the the SmartCar’s coordinates. The table affected by the class is “guard\_cars”.

***GetConnectivityData*** gets the connection information from the “connectivity” table and uses it to make a SmartCar object once the main menu is created.

***GetTravellerData*** is used to search and return the information from “travellers” table and checks to see if the values entered in login are correct.

***ReachInternet*** an async tasks that makes sure the phone has internet access.

Helpers classes:

This section contains miscellaneous methods that were used as helpers in other classes.

***DBHelper*** helper methods for the Java server connection.

***HashInformation*** contains hashing methods for passwords in the application.

***WifiBroadcastReciever*** deprecated. Was used monitor changes in WiFi connection early on in the development process.

Models:

Data containing classes, for cohesion and easier development for the entire development team.

***Control*** used to display the manual control, and for calculations of angle and speed of the motors.

***Sensor*** contains the parking sensors with data returned to the application from the SmartCar.

***Session***, checks whether the traveller is currently signed in to the application.

***Traveller***, profile information of the logged in traveller.

***SmartCar***, all information regarding the SmartCar received from the remote database. If there is no internet connection the application creates a basic SmartCar with preset values.

### 5.2.2 Python

***DataGraph*** is a script to be used whenever a user would like see a current map of the immediate area generated by a LiDAR and a servo.

***GpsAngle*** is used to calculate the direction of travel for the SmartCar using the GPS location of the user and SmartCar.

***Drive*** is where all driving of the car is calculated and sent to the arduino to be executed.

***set\_coords\_phone*** is used to set the coordinate location of a user to be used when moving to the user.

***Smartcar\_gps*** writes the coordinates for the car to be calculated by GpsAngle and executed by drive.

***Lidarlite*** is a class to receive data from the lidar and interputting it.

***servo*** is a servo controlling class

***Mapping*** is a class that autonomously navigates a room and and simultaneously maps out the area

### 5.2.3 Arduino

The Arduino sketch contains all methods, apart from methods that originate from libraries.

### 5.2.4 Node.js

Firstly, ***package.json*** contains the relevant metadata pertaining to the Node.js files. Most importantly, it includes the foreign libraries used marked under dependencies. Additionally, it contains version numbers for traceability accompanied by a link to the repository.

Secondly, ***server.js*** is used to create a web server through which all the HTTP request will go through. Additionally, the file is used to establish connection pooling to the MySQL database, this allows multiple SQL to be executed concurrently as well as connection caching, which results in virtually permanent connections. Lastly, this script is used to set up URI routes for the API.

Thirdly ***guardRest.js*** refers to a file that creates the API used in the project in accordance to RESTful principles. Its responsible for enabling the application users to manipulate data within the database. SQL queries are contained and executed here as well.

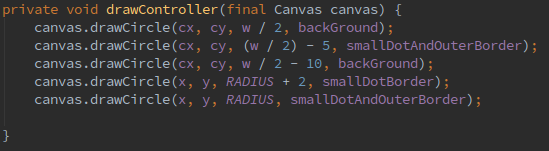
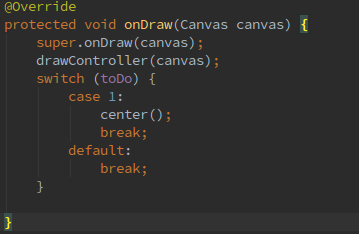
Lastly ***authentication.js*** is used to grant additional data security to the stored information. This file creates a Basic Authentication method that is exported to the “guardRest.js”, which requires additional set of credentials in order to be able to carry out the API functions.

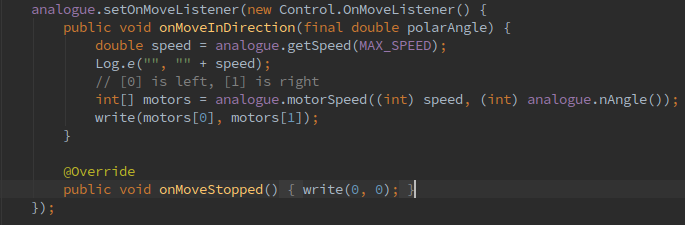
## 5.3 Important Functionality

### 5.3.1 Java

Manual Controller, is a circular controller. It is created as an extended View. When the Control class is instantiated, developers needs to add com.group2.guard.Control in the activities or fragments layout file, an id, width and height are required.

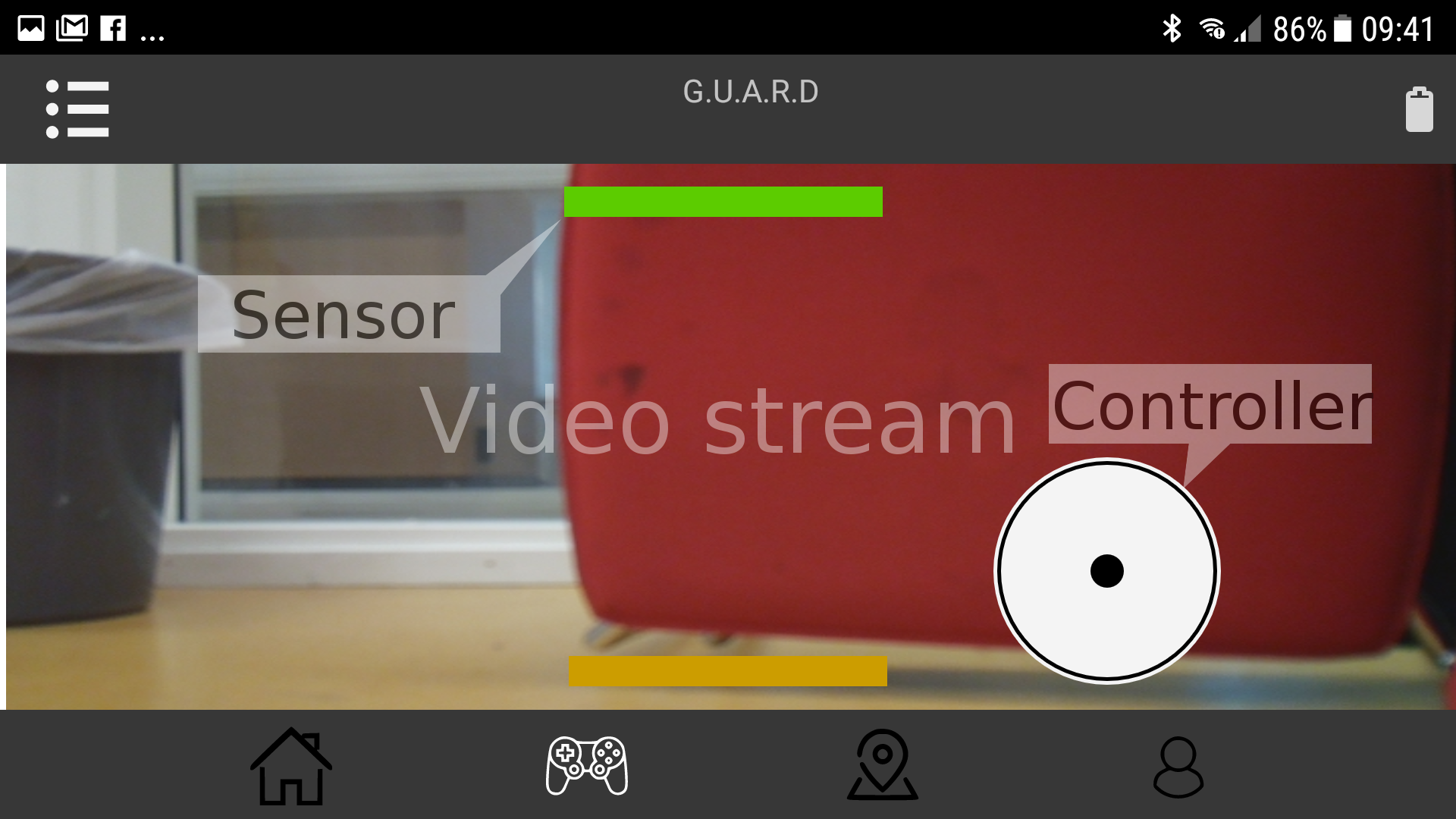
With Control [controllers id in layout] = (Control) findViewById(R.id.[controllers id in layout]) the onDraw method is called:



When the application has a bluetooth connection the controller writes to the Arduino board with an OnMoveListener: 

Video Stream is a WebView with an IP camera with little to none delay.

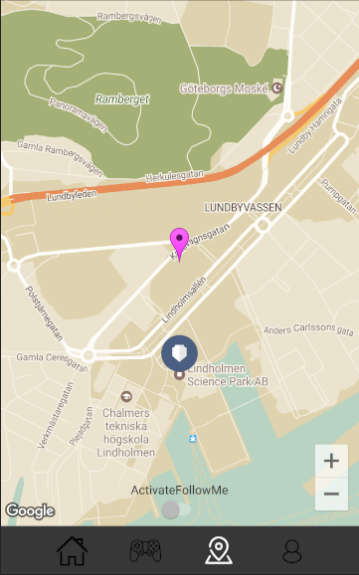
The ip address is gotten from the SmartCar, and the SmartCar gets the connection information from the node js database.



GPS following is not fully implemented due to hardware limitations.

There is communication an established between the application and the raspberry pi, the gps coordinates are compared and scripts written in Python sends values to the Arduino board.

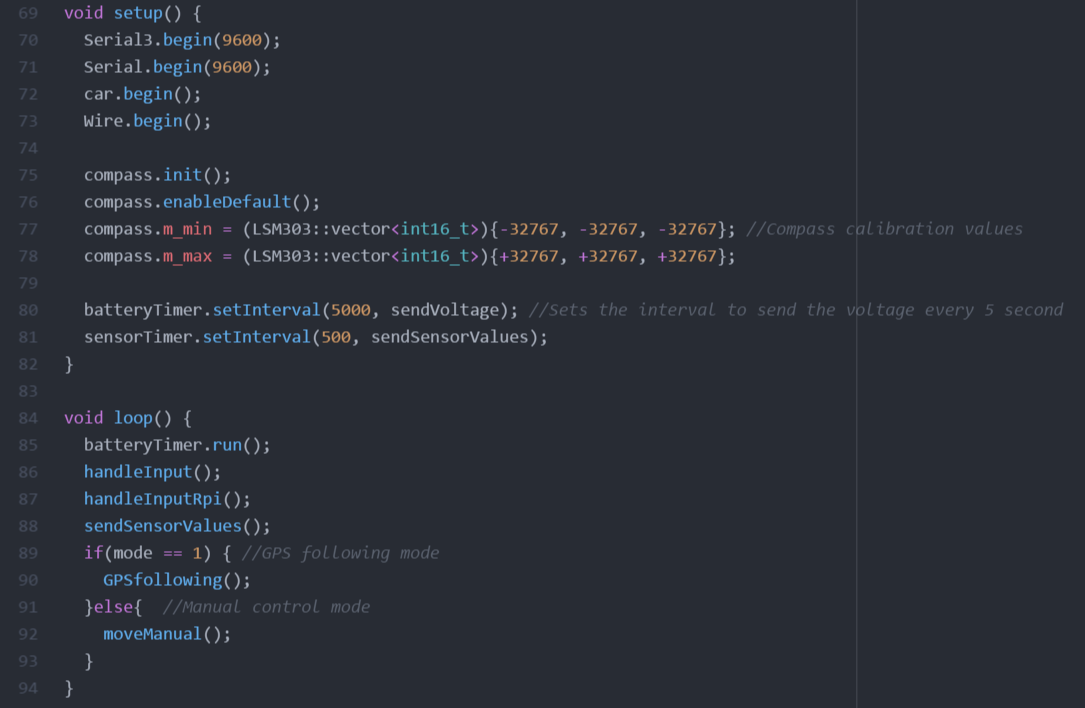
The GPS Following script takes control of the SmartCar, and follows the phone running the follow me part of the application.



Location Map, shows the travelers location and the location of the SmartCars of the system. Because it only exists one SmartCar at the time of writing only one marker is displayed. The project is ready to add new SmartCars when needed through the SmartCar class. The locations are saved to our database.

### 5.3.2 Python

### 5.3.3 Arduino

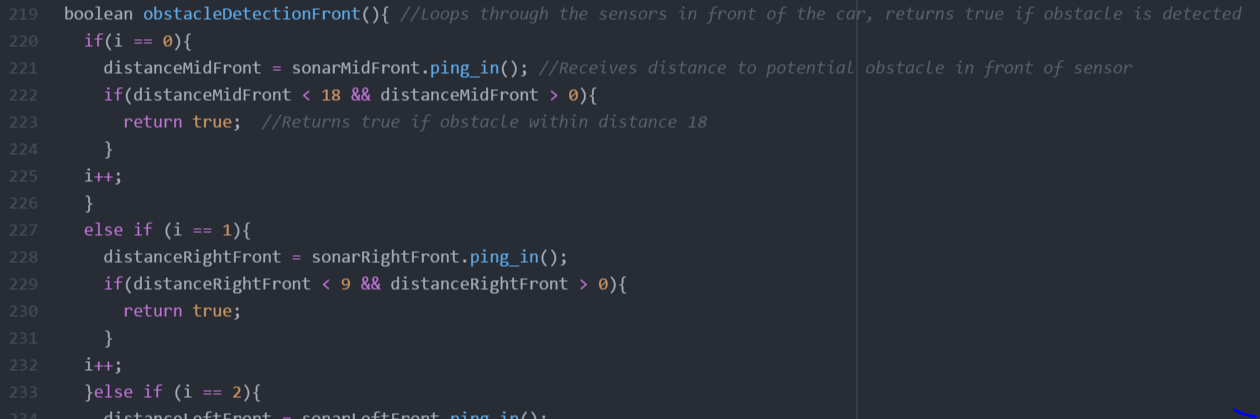
As aforementioned, the sketch has to include both a setup as well as a loop method to be runnable for the microcontroller. 

***setup*** method that, apart from initializing and specifying baud rates, sets the calibration values for the magnetometer, line 77 and 78, as well as sets the interval for sending the battery voltage (every 5 second) and the sensor values (every 0.5 second) on line 80 and 81 (see figure above).

***loop*** is the main method that executes repeatedly as soon as setup has been executed. Depending on mode, either GPSfollowing() or moveManual() will be executed along with the rest of the methods listed above (see figure above).

***handleInput*** methods are of critical importance since they handle all the incoming data. Depending on prefix of the string received, the methods assign the existing variable with the value that follows the prefix. For instance, “L45” would change the “motorSpeedLeft” to 45. Yet, if the handleInput methods receive either a “M” or a “G”, the mode which the Arduino are set to specifies (manual control or GPSfollowing).

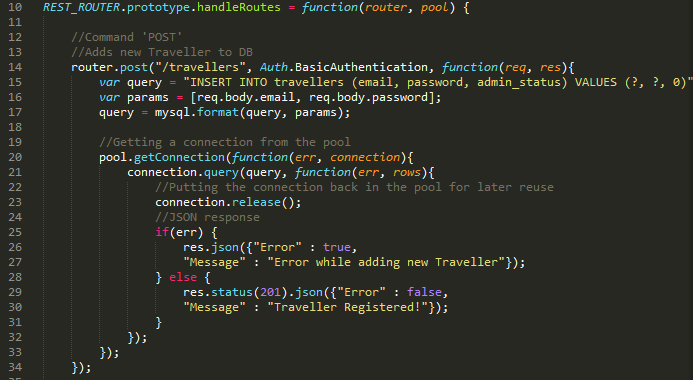
***obstacleDetectionFront*** returns a boolean which indicates whether obstacle detected in front of the SmartCar within the specified threshold or not (see figure below). Due to the microcontroller’s single thread of execution, an approach of only checking one sensor for obstacle per execution of the method was vital (“i” is used as a counter). Checking all sensors for obstacles during every execution of the method lead to a noticeable delay that decreased the user experience when using the joystick in the application significantly.



***moveManual*** evaluates whether the SmartCar is permitted to move in certain directions depending on what the obstacleDetection methods return. The method will also stop the car even if no input is acquired from the user but the applicable obstacleDetection method returns true.

***rotateOnSpot*** takes an integer as argument, the heading (target degree) which the SmartCar ought to rotate to, and rotates until such condition is fulfilled. The first ten lines are used to determine which way the SmartCar should rotate to turn the least. The method then goes into a while loop that terminates once the car faces the target degree (programmatically +/- 1°).

### 5.3.4 Node.js



***Server connection*** is functionality main part can be found in server.js under the node.js directory. The pool variable is created in the connectMysql function. The host IP and credentials are stored within the pool variable, alongside the maximum connection limit and the waitForConnection boolean, used in case the attempted connection number exceeds the limit. Once the connection pool is created it is passed to a separate function; configureExpress, which sets up the middleware used for the web services. From there the pool variable is used when creating a new router. The use of pooling can be seen in the figure above at line 20. The connection is wrapped in pool.getConnection ensuring that it will be received from the pool if there are any connections not currently used. After the query is executed the connection.release method is called to free up the connection and o be put back into the pool.

***Authentication*** function makes sure that the data stored in the MySQL database is only accessible if a separate form of credentials are inputted. The function can be found the authentication file under node.js directory and its use can be seen in the figure above at line 14. An authentication variable is declared at the beginning of guardRest. It’s exported method is called after declaring the endpoints for the router thus stating the route must require a second form of authentication. The function it self creates an authentication variable utilizing a foreign, “basic-auth”. The variables fields are then checked if they are null, if not then it continues to compare the fields entered by users with the username and password found in the script.

***Web services*** that bridge the connection between the android application and the database can be found in the guardRest file. The API are declared in the handleRoutes, a function that takes the previously mentioned connection pool and the middleware router as parameters (see figure above).The project decided to incorporate CRUD principle that states users should be able to create, read, update and delete data. This is done by declaring the URI endpoint and the type of method the router uses, pictured in line 14. The individual methods continue from there with SQL query variable that are necessary to alter the MySQL database information. Additionally, variable params is used in case the SQL query contains prepared parameters. The query is the formated and a connection is retrieved from the pool so it could be executed, line 17. After which a JSON object is returned depending on the success of the execution.

## 5.4 Extra Libraries

### 5.4.1 Python

**serial** is a library used for communication between the arduino and raspberry via USB.

**matplotlib** is a library for graphing data, which is used for creating maps based on data from the LiDAR.

**smbus** is a library for the raspberry pi to communicate and use the pins on the raspberry pi.

**pynmea2** is used for parsing of data from the GPS USB module.

### 5.4.2 Arduino

The Arduino sketch utilizes in total five different libraries.

**SimpleTimer** – www.playground.arduino.cc/Code/SimpleTimer

Due to Arduino’s single thread execution and limited performance, code that does not necessarily provide value by constantly being executed can instead be advantageously scheduled. Our sketch includes two instances of such code; sending battery voltage as well as sending sensor values (from ultrasonic sensors, distances to objects). With SimpleTimer, these values are instead sent regularly to the phone application minimizing the work for the Arduino.

**Smartcar shield** – www.github.com/platisd/smartcar\_shield

This project is based on the Smartcar platform provided by Dimitris Platis. It also utilizes his library, Smartcar shield for an easy and intuitive way of controlling the smartcar.

**NewPing** – www.playground.arduino.cc/Code/NewPing

When developing the sketch, we found the NewPing library far superior to any other ways of getting values from the ultrasonic sensors in a fast and stable way. The NewPing library is used for all the ultrasonic sensors used in our SmartCar.

**LSM303** – www.github.com/pololu/lsm303-arduino

Even though we use a magnetometer from Adafruit, we use Polo’s library for getting the heading. The Polo library has shown to provide more accurate data and it also gives us the ability to calibrate the magnetometer for even more accurate readings.

**Wire** – built-in library

Lastly, the Wire library is used by the magnetometer library, LSM303, to communicate with the magnetometer over the I2C/TWI pins.

### 5.4.3 Node.js

The Node.js section of the project contains several different foreign libraries that were used to establish the desired features.The necessary files for the libraries can be found under “node\_modules” directory and the resulting dependencies in the “package.json” file.

Firstly, the ***express*** module is acts as middleware for the web services. The framework is used to create and manage the API and HTTP requests.

Secondly, ***body-parser*** module is used to extract the body portion of the web services. The body contains information as to what will be inserted into the database. /for example, the users’ email and passwords.

Thirdly, a ***mysql*** module is used to format and execute queries to and from the database. The queries can be found in “guardRest.js”

Lastly, ***basic-auth*** is the module responsible for implementing the secondary authentication needed to execute the API calls.

Additionally, a module with the name of ***pm2*** is used as a production managed for the Node.js web server. The module is globally installed on the remote server and therefore is not found within the Node.js sub project. The module is responsible for logging and maintaining the web server over prolonged period of time as well as restarting it, in case it crashes or stops.

## 5.5 SQL Queries

As aforementioned G.U.A.R.D. utilizes a MySQL database, which requires SQL queries for data manipulation. A separate database user was created to represent the majority of the application users. This user was granted table specific privileges to only insert, select update and delete information and omitted any structural or management capabilities. This was done to ensure data security purposes. Furthermore, to avoid possible SQL injections, the queries were comprised of prepared statements.

Queries:

1. INSERT INTO travellers (email, password, admin\_status) VALUES (?, ?, 0)

The query is used from within the RegisterActivity for the purpose of adding traveller information to the “travellers” table. The SQL query requires the email and password to be passed through the HTTP body. However, the admin\_status is not necessary as it will explicitly add a 0, representing simple user status.

1. SELECT \* FROM travellers

The query is intended for administrative purposes in the possible future. The purpose of which is to retrieve all registered users to be displayed for the admin. At the moment this has not yet been implemented.

1. SELECT \* FROM travellers WHERE email = ?

Retrieves all the fields for a specific traveller. The query is called from within the LoginActivity and uses the emailed entered in the activity to identify which data to retrieve.

1. SELECT \* FROM connectivity

Returns all of the network information from the connectivity table. Although the query works in the future it would need minor changes if the G.U.A.R.D. project adapted multiple SmartCars.

1. UPDATE travellers SET password = ? WHERE user\_id = ?

The query is called from within the ProfileActivity enabling the currently signed in users to update their password. The email field is used to identify which password to change in the database.

1. UPDATE travellers SET email = ? WHERE user\_id = ?

The query is called from within the ProfileActivity enabling the currently signed in users to update their email. The user ID is used to identify which email to change in the database.

1. UPDATE guard\_cars SET latitude = ?, longitude = ? WHERE car\_id = ?

Simultaneously updates the latitude and longitude for a specific SmartCar. Each values must be past to in the HTTP request’s body.

1. DELETE FROM travellers WHERE email = ?

The query is used to grant the application users the ability to willfully remove their account if they so chose so. The query is called from within the ProfileActivity.