Design Document

# Introduction

Introduction goes here.

# System Requirements

System Requirements goes here

# Technology User and Why

Technology user and why goes here

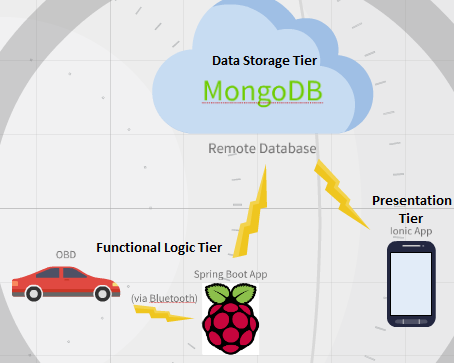
# Architecture of the Solution

For this project we implemented a three-tier architecture solution. It is a client-server model in which the functional logic (application tier), data storage (data tier) and the user interface (presentation tier) are developed as individual components on separate platforms.

We chose this style architecture for several reasons including the major benefit of being able to split up the workload and each member on the team could develop on the platform which they were most competent and comfortable with.

Another advantage of using this architecture is the ability to modify one component of the system without impacting on the other areas of the project which also saves a lot of potential work when configuring under-lying technologies and services.

Looking ahead to the future, if we were to scale the application up and out it would not be a major overhaul operation as changing the database would simply entail us swapping out the current data tier in use and replacing it with another pre-configured server/database for more users and data storage. With the same logic we can keep our system up to date with the most modern technologies used to continuously improve our application.



# Design Methodology

The design methodology we used for this project was AGILE. It is an incremental approach for articulating a structured project procedure allowing for recurrent alterations using the version control tool GitHub.

We chose this methodology as it limits the work load risks by creating software in short iterative bursts which happened every couple of weeks. These short iterations were beneficial to the development as we were able to adapt to changes in the requirements favourably.

The maintenance of the system is simplified significantly under this methodology as the discovery and fixing of bugs and defects were made early and they didn’t impact on the continuous integration of components developed later on.

The most advantageous aspect of using AGILE is the frequency of the prototypes as it gave clear indications of what changes needed to be made and it was also a good motivating factor from a developer’s perspective to see a working version early on in the development stage.

# Features of the Implementation

**On-Board Diagnostics (OBD)**

Using the spring framework we developed an application that extracted data via Bluetooth from an on-board diagnostics tool which in turn communicated with the vehicle’s Engine Control Unit (ECU). This OBD dongle which is compatible with the vast majority of cars made since (2003) allows our application to available to the majority of drivers.

We developed an OBD class that initiated a communication between our application and the dongle, the then made the requests -which we imported from an existing API- to the dongle which returned an input stream of data from the vehicle's ECU. We utilised the speed, distance and rpm commands as we decided them values were the most useful for a wider demographic of users. A handy feature in the application is the driver reports generated alert the user if they're an economic driver or not. The program then organises the data into arrays and sends the data to the database.

On the front end, we chose to use Ionic 3 as an application development platform as it offered a nice User Interface (UI) aswell as functional capabilities to deal with http requests and formatting the responses in the form of reports containing calculated values for that trip and graphs for a more visual context.

# Limitations

Limitations goes here

# Known Bugs

Known bugs goes here

# Recommendations for Future Development

Recommendations for future development goes here

# Conclusions

Conclusions goes here