**Document:**

Design Document

**Product Name:**

Driver Monitor System

**Course:**

Professional Practice in IT

**Participants:**

Kevin Delassus

Michael Curley

Shane Daniels

**Supervisor:**

Martin Hynes

|  |
| --- |
|  |

# 1. Introduction

The project is to create a driver monitor system that monitors your driving performance per journey. The systems uses a raspberry pi to connect to a OBDII dongle which pull all the data . Once the journey has concluded a report is generated and available to view on your smart phone. The system should store every report generated in the last month.

This design document presents the designs used or intended to be used in implementing the project. The designs described, follow the requirements specified in the Project Requirements Specification document given to us at the start of semester 2.

## Purpose

The purpose of this document is to present a detailed description of the designs of the driver monitor system , created for the Professional Practice in IT module project. Firstly, this document is intended for the team, to use the designs as guidelines to implement the project. Lastly, this document is also one of the project requirements.

# 2. System Requirements

The system requirements for this system are as follows

* A Raspberry Pi running Raspbian and a laptop with either Intellij Idea or Eclipse EE to run the Spring application
* Either a Android or Apple device to run the Ionic Application
* An OBDII Dongle

# 3. Technology Used and Why

The application running on the Raspberry Pi will be implemented in Spring boot (Java). The reason we have decided to program this in Spring Boot is because Spring has a great way of connecting to a database. In our case we are using MongoDB and both work very well together. Our Spring application connects to a car by a OBDII dongle. The connection is done by the COM ports between P.C and OBDII dongle.

On the server side of things we will be running MongoDB to store the data from the Spring Boot application. MongoDB stores data as documents in a binary representation called BSON (Binary JSON). This makes life a lot easier when creating an API as the API needs to be in JSON format. For creating the API we will be using Node.js. Node will be using Mongoose to connect to the MongoDB, Express to create the sever, Body Parser which parses the data from the MongoDB to JSON format. Lastly Cors will be used to assign the appropriate headers so applications can make requests to the API. The reasons for using Node.js is because it is fast, NPM is the Node.js packet manager and it is really good to use. It does a great job specifying and installing project dependencies, but also obscures a great deal of complexities. All this technology will be hosted on a Digital Ocean Droplet. Digital Ocean make it easy to set up a server which already has MongoDB pre-installed. You can create a student account which gives you $50 free credit.

For the mobile application side we have decided to go with Ionic. Ionic is an HTML5 mobile app development framework targeted at building hybrid mobile apps. Hybrid apps are essentially small websites running in a browser shell in an app that have access to the native platform layer. Reasons for using Ionic are that we wont have to develop different mobile applications to run on different mobile operating systems. We can just develop the one Ionic app and this will up on all mobile operating systems.

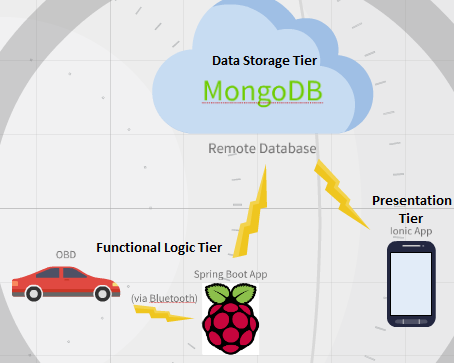
# 4. Architecture of the Solution

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.



# 5. 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.

# 6. 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.

**Spring Boot**

We decided to implement this solution using a spring boot application which has its own configured embedded servers meaning there was no need to deploy WAR files or set up servers. We could have used Java Database Connectivity (JDBC) but the correct drivers have to be installed, connections have to be opened and closed manually, it’s not as fast as it doesn’t return objects and overall it would be more cumbersome than the automated spring configuration.

**OBD Class**

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 utilized 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 organizes the data into arrays and sends the data to the database.

**Ionic**

On the front end, we chose to use Ionic 3 as an application development platform as it offered a suitable User Interface (UI) as well 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 of the rpm and speeds of that journey.

# 7. Limitations

During the project we faced many limitations which hindered our abilities. First and foremost was the issue of time. The limited timeframe made it difficult to both implement all of the features we had hoped we would get to in the project, and also increased the overall pressure placed on us to create a working and robust product.

Another limitation we faced earlier on in the testing of the spring boot application was having a decent network connection. Before we had implemented off-line recording, we had to ensure that the connection to the server was not broken.

We were also limited by a number of smaller finance-based issues including the cost of hardware such as the OBDII Bluetooth adapter and the cost of fuel used whilst running the tests.

# 8. Known Bugs

Across the different programs that make up the project, there is a small number of bugs that we have come across. In the front end Ionic app the graphs won’t load when page is navigated to and on a second time loading data on graphs page renders speed and distance graphs blank.

The OBDII command for VIN is not returning a value.

# 9. Recommendations for Future Development

With regards to future objectives, we would like to develop the system so the Spring Boot program can directly communicate with the Ionic application allowing the user to start and stop the program manually via the mobile app.

Extending the utilization of the possible commands would be another recommendation to generate elaborate reports and return more data to the application to give the user a more accurate indication of their driving habits.

We would have also liked to be able to extract the vehicle identification number (VIN) to reference each report with the corresponding driver and then possibly authenticate the application with user login requiring credentials to gain access to the system.

# 10. Conclusions

Conclusions goes here