Riding Mechanic: An iOS App for Vehicle Monitoring and Analysis

A Manuscript

Submitted to

the Department of Computer Science

and the Faculty of the

University of Wisconsin-La Crosse

La Crosse, Wisconsin

by

Dezheng Wang

in Partial Fulfillment of the

Requirements for the Degree of

Master of Software Engineering

May 2017

Riding Mechanic: An iOS App for Vehicle Monitoring and Analysis

By Dezheng Wang

We recommend acceptance of this manuscript in partial fulfillment of this candidate's requirements for the degree of Master of Software Engineering in Computer Science. The candidate has completed the oral examination requirement of the capstone project for the degree.

Dr. Elliott Forbes Examination Committee Chairperson	Date
Dr. Kasi Periyasamy Examination Committee Member	Date
Dr. John Maraist Examination Committee Member	Date

Abstract

Dezheng Wang, "Riding Mechanic: An iOS App for Vehicle Monitoring and Analysis", Master of Software Engineering, May 2017, Dr. Elliott Forbes.

The On-Board Diagnostics II (OBD-II) standard is a vehicle-mounted system used to monitor the running status of engine components and the working status of emission control systems. OBD-II was first mandated for all cars and light trucks from the 1996 model year onward. Early OBD-II applications (apps) were developed to view information from OBD-II systems, but many did not make it very clear what problems existed with a car because most app developers had little knowledge of OBD-II systems. Newer apps have become more comprehensive. These modern OBD-II apps provide drivers with detailed information of their cars, but drivers need to pay for them. The goal of this work is to develop a free mobile OBD-II application to gather detailed diagnostic information over a long period in order to monitor a vehicle's operation and discover potential concerns before they become serious faults. The application can also help those drivers who like to modify their vehicles to see the effects that modifications have by recording driving behaviors from one route to another route.

Acknowledgements

I will like to express my sincere thanks to my project advisor Dr. Elliott Forbes for his patience, kindness and valuable guidance. I will also like to express my thanks to the Computer Science Department and the University of Wisconsin-La Crosse for providing the computing facilities for my project.

Table of Contents

List of Figures

List of Tables

Glossary

Attention command set

Attention Command set (AT command set) is a command language used to control modems.

ELM327

The ELM327 is a programmed microcontroller produced by ELM Electronics for translating the on-board diagnostics (OBD-II) interface found in most modern cars [1].

Diagnostic Trouble Codes

Diagnostic trouble codes (DTCs) are alphanumeric characters saved by a vehicle's on-board diagnostic system to point out different types of faults.

On-board Diagnostic II Parameter IDs

On-board Diagnostic II Parameter IDs (OBD-II PIDs) are hexadecimal codes sent to a vehicle's On-board Diagnostic system to request data.

On-board Diagnostic II scanner

On-board Diagnostic II scanner (OBD-II scanner) is a tool used to access the data provided by the OBD-II system.

On-board Diagnostics

On-board Diagnostics (OBD-II) is a vehicle-mounted system used to monitor the running status of engine components and the working status of emission control systems.

1. Introduction

It is difficult to imagine life without driving. To work in a different city, to attend school far away from your house, to travel to another state, or to speed transportation, taking a car is often a preferred choice.

People drive every day, but they typically have little knowledge about their cars. On-board diagnostics (OBD) systems can help people get a greater understanding of their vehicles. It is possible for people to use an OBD application (app) along with their mobile devices and an OBD scanner to read data from their cars. Most OBD applications on the market can provide drivers with their cars' real-time information as well as Diagnostic Trouble Codes (DTCs) - which are the fault codes detected and stored by OBD systems. Instead of investigating those trouble codes without having any professional knowledge about cars, those apps simply tell people the categories and meanings of those problems.

The goal of this work is to develop a free mobile OBD application to gather detailed diagnostic information over a long period of time in order to monitor a vehicle's operation and discover potential concerns before they become serious faults. Some drivers enjoy modifying their vehicles. The app developed for this project can also help those drivers identify the effects of their modifications by recording driving behaviors from one route to another route.

1.1 Onboard Diagnostic systems

On-Board Diagnostics (OBD) systems were first introduced in early 1980s. Early OBD systems were developed to warn drivers of malfunctioning engine components. As time went by, a more advanced version, On-Board Diagnostics II (OBD-II), was adopted, making it mandatory for all cars and light trucks manufactured in the United States from 1996 onward, as required by the Clean Air Act Amendments [2]. Since that time, OBD-II systems began to play an important role in fighting emissions of greenhouse gasses as well as providing information on potential failures of vehicle components.

OBD-II systems are vehicle-mounted computer hardware and software that monitors the emission control and emission-related components/systems, along with certain vehicle components that provide overall vehicle operation information [2]. OBD-II systems can monitor both running status of engine components and working status of the emission control system. If the OBD-II system detects emissions over specified thresholds, the OBD-II system will alert drivers. If any system fails, a “Check Engine” or “Malfunction Indicator Lamp” will light on the dashboard of the car. When this occurs, the OBD-II system will also store a Diagnostic Trouble Code specific to the failed system. By using a standard scan tool connected to vehicle’s OBD-II connector, an automotive technician can quickly know what the trouble is and in which system it occurs. This can save time for both drivers and technicians.

The biggest difference between the OBD-II system and previous OBD systems is that the OBD-II system is extremely strict with emissions of gasses such as carbon dioxide. It can detect emissions over a specified threshold by monitoring emissions sensors. OBD-II system also has the following features:

- The data link connector (DLC) in every vehicle is a standardized 16-pin connector.
- The OBD-II DLC can help to transmit data between cars and scan tools by using standard OBD-II protocols.
- OBD-II defines universal OBD-II Diagnostic Trouble Codes(DTCs) for all types of vehicles.
- Users can use scan tools to clear DTCs.
- OBD-II can show the cleared DTCs.

Although it is very efficient to monitor emissions by using the OBD-II system, sometimes drivers ignore alerts from the OBD-II system. For this reason, a more advanced version, OBD-III, has been developed. OBD-III will prevent users from driving when high emissions have been detected. However, OBD-III systems are not widely used.

The application developed for this work could easily be extended to use potential future OBD-III systems.

1.2 Onboard Diagnostic II scanner

Since the OBD-II scanner is a standard tool for accessing the data provided by OBD-II systems, it has played an important role in the process of reading data from a vehicle. An OBD-II scanner cannot be used by itself - it must be used in conjunction with an OBD-II system. The OBD-II scanner can connect to the OBD-II port provided by the OBD-II system. OBD-II ports are defined by the SAE J1962 standard. When users want to use an OBD-II scanner to connect to an OBD-II port, the OBD-II port must be plugged into an OBD-II scanner. There are two types of wireless OBD-II scanners commonly used with other OBD-II applications. The first type is a Wi-Fi scanner, which provides access to the OBD-II system via a Wi-Fi interface. The other type is a Bluetooth scanner, which likewise provides access to the ODB-II system, but through a Bluetooth interface. With the OBD-II scanner plugged in, it acts as a server. Communication between a mobile device and an OBD-II scanner can be easily established by simply connecting to the scanner’s Wi-Fi or Bluetooth server. After building the connection successfully, the mobile application acts as a client, allowing the app to read data from the vehicle.

Figure 1 shows how an OBD-II scanner acts as a medium between the application and a car [3]. As shown in the picture, the OBD-II scanner used for this project has two hardware components; one is the ELM327 and the other is a Wi-Fi module which can send and receive TCP/IP packets. The ELM327 is the core part of our OBD-II scanner, and it is designed to translate messages between the vehicle’s OBD-II port and a standard serial port. This scanner supports almost all standard OBD-II protocols and it has a common Universal Asynchronous Receiver/Transmitter (UART) serial interface.

A mobile device also has a Wi-Fi module, which is responsible for sending and receiving signals. Once a connection to the scanner has been made, the application can send commands to the Wi-Fi module on the phone. The Wi-Fi module on the phone receives those commands and sends them to the Wi-Fi module on the scanner. The Wi-Fi module on the scanner receives those commands and sends them to the ELM327. Then the ELM327 uses those commands to interact with OBD-II system in the car through the universal serial port. The OBD-II system can identify those commands and quickly send response back to the scanner. As the scanner receives response from the car, it sends them to the application using the same mechanism but in the reverse order (from the scanner to the mobile device). The application will continually send requests for information and receive the reply messages. Once each message is received, the application can analyze those messages and display the relevant information to the user.

Many modern cars have built-in Bluetooth connectivity to allow drivers to use their mobile devices for maps, route information, making phone calls, sending text messages, and for music and entertainment. A Wi-Fi scanner was chosen for this project, since a Bluetooth scanner would not allow users to simultaneously connect to the vehicle's Bluetooth and a scanner's Bluetooth at the same time. Therefore, the app developed for this project uses an OBD-II Wi-Fi scanner.

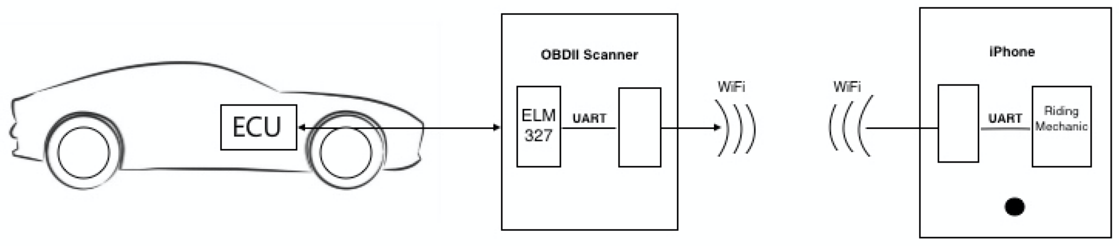


Figure 1. How to use an OBD-II scanner

1.3 Onboard Diagnostic Parameter IDs

OBD-II Parameter IDs (PIDs) are ASCII encoded hexadecimal codes sent to a vehicle's OBD-II system to request data. SAE J1979 defines standard universal OBD-II PIDs that are used by many car manufacturers. Most car manufacturers also define their own specific PIDs for use with their vehicles. The application developed for this project will focus on the universal PIDs instead of all OBD-II PIDs because manufacturer-specific PIDs are very diverse. There exist databases that contain these manufacturer-specific PIDs, but use of these databases requires subscription fees.

As mentioned previously, SAE J1979 describes standard universal OBD-II PIDs. Table 1 shows all modes of operations defined in the standard [4]. Mode ‘01’ and ‘03’ are the two most important codes for this project because the app only needs to read current data and Diagnostic Trouble Codes from the car.

Mode (hex)	**Description**
01	Show current data
02	Show freeze frame data
03	Show stored Diagnostic Trouble Codes
04	Clear Diagnostic Trouble Codes and stored values
05	Test results, oxygen sensor monitoring (non CAN only)
06	Test results, other component/system monitoring (Test results, oxygen sensor monitoring for CAN only)
07	Show pending Diagnostic Trouble Codes (detected during current or last driving cycle)
08	Control operation of on-board component/system
09	Request vehicle information
0A	Permanent Diagnostic Trouble Codes (DTCs) (Cleared DTCs)

Table 1. Ten modes of operation

As defined by SAE J1979, OBD-II PIDs consist of ASCII codes. There are many kinds of PIDs under mode ‘01’ while ‘03’ does not have any subset. Table 2 shows OBD-II PIDs used in the application used in this project [4].

OBD-II PIDs	Description
0100	Get supported PIDs [01-20]
0120	Get supported PIDs [21-40]
0140	Get supported PIDs [41-60]
0105	Get engine coolant temperature
010C	Get engine RPM
010D	Get vehicle speed
0110	Get MAF air flow rate
0142	Get control module voltage
03	Get stored DTCs

Table 2. OBD-II PIDs used in the application

1.4 Diagnostic Trouble Codes

Diagnostic Trouble Codes (DTCs) are fault codes detected and stored by OBD-II systems. Reference documentation exists to gather diagnostic information associated with DTCs. The OBD-II DTCs themselves are alphanumeric codes consisting of five characters – an alphabetic letter followed by four numeric digits (e.g., P1000). The first character always describes which system has the fault. Table 3 gives each letter's meaning [5].

Letter	System
P	Powertrain (Includes engine, transmission, drive shafts, differentials and the Final Drive)
U	User Network (Wiring Bus or UART)
B	Body (Includes A/C and Air Bag)
C	Chassis (Includes ABS)

Table 3. Meaning of each DTC prefix

The second character is a binary number that indicates whether the trouble code is an OBD-II generic code or not. If the second digit is zero, it is a generic code. If the second digit is one, it is a manufacturer-specific code. The third character points out which subsystem in the car has the fault. The fourth and fifth characters are used together to explain the fault.

Suppose, for example, that a vehicle has a DTC of C0300. The first letter ‘C’ indicates a chassis problem. The first zero indicates that it is a generic code. The number three indicates that the failure is in the misfire monitoring system. And the last two digits indicate that there is a malfunction in the rear prop shaft circuit.

The application can receive the DTCs from a vehicle and then provide the corresponding diagnostic message. However, those messages need to be encoded to become universal DTCs. The prefix of a DTC is encoded as two bytes and the second character of a DTC is also described by another two bytes. Table 4 and Table 5 show how the two characters are encoded [6]. The last three digits are hexadecimal characters and each of them can be either a number (0-9) or a letter (A-F). All three characters are encoded by four bytes and they follow rules when a hexadecimal character is converted into a binary character.

First two bytes of a diagnostic code	**First DTC character**
00	P- Powertrain
01	C- Chassis
10	B- Body
11	U- User Network

Table 4. How the first DTC character is encoded

Second two bytes of a diagnostic code	**First DTC character**
00	0
01	1
10	2
11	3

Table 5. How the second DTC character is encoded

1.5 AT command set

The scanner tool also uses the Attention (AT) command set. The AT command set is a command language originally developed by Dennis Hayes for the Hayes Smartmodem 300 baud modem in 1981 [7]. This command set became popular over time as a commonly used command language consisting of a series of short text strings that can be used to manage modems' behaviors. Although AT commands and OBD-II commands (OBD-II PIDs) are both sent from the application, they are use for different purposes. The application can send AT commands to change settings of the OBD-II scanner. As mentioned above, ELM327 is the core part of an OBD-II scanner. It is fully configurable using AT commands. An AT command is composed by the application and then it is transferred from the phone to the scanner using the sequence of steps described above. Once the scanner has received that AT command, it can change its configuration parameters accordingly. It will also give a response to the sender to let it know whether the command succeeded or not.

Every AT command begins with "AT", and the characters that follow provide the details of the command. Table 6 shows the AT commands used in this project [8].

Version	**Command**	**Description**	**Group**
1.0	@1	Display the device description	General
1.3	@2	Display the device identifier	General
1.3	SP 00	Set protocol to auto and save it	OBD
1.0	Z	Reset all	General

Table 6. AT commands used in this application

1.6 Thesis outline

The remainder of this thesis is organized as follows:

- Chapter Two describes the users' requirements from several different perspectives. Furthermore, functions that can be performed by users are described in detail in the functional requirements. Also, non-functional requirements describe some of the system qualities.

- Chapter Three describes both the goals and the constraints of the system architecture design. It also gives an overview of how the application will look. It also points out some key issues and its solutions.
- Chapter Four presents the detailed system design, including database table design and the UML diagrams for user scenarios, the relationship between classes and the interactions between each component in the system.
- Chapter Five shows the implementation of main functions using pseudocode to make logic of each process clear.
- Chapter Six presents testing strategies in this project. It also shows the process of testing and testing data included in each kind of testing.
- Chapter Seven provides concluding remarks.

2. Requirements analysis

This section lists specific requirements for Riding Mechanic. Requirements are divided into the following sections:

1. User requirements. Requirements written from the point of end-users. These requirements describe the functions that the product must have. User requirements are usually expressed with user stories.
2. Functional requirements. Detailed specifications describe the basic functions the system must have. Developers define functional requirements in order to satisfy user requirements.
3. Non-functional requirements. Requirements define characteristics the system should have. Non-functional requirements are also called system features (e.g., usability).

2.1 User requirement analysis

In section one, we discussed why OBD applications are needed in our daily life. This part will describe the requirements in detail from the end-user's perspective. A smart vehicle can optimize its performance to reduce fuel consumption, show the best with considering current external conditions and detect errors in the engine. It is hard for users to maintain the performance of cars because users can only view some basic information (e.g., speed) which gave users little help to improve the performance. What should people do if they want to get familiar with their cars without going to the auto shop? As mentioned above, users can use an OBD-II adapter to read information from a car's OBD-II system. However, to handle all possible data form OBD-II device users still need OBD-II applications to understand those categories. The question is what capabilities users want those applications to have.

To communicate with a car, the very first step for users is to connect to an OBD-II scanner through Wi-Fi or Bluetooth. Users need an OBD application to help them get access to the car. After building connection successfully, users can communicate with the car through the scanner. It is necessary for users to think about what kinds of information are needed so that users can take advantage of those information to help them. To improve driving habits, the best way is to analyze driving records to draw conclusions. Users need the application to have functions of recording driving information. All data in driving records must be specific and accurate to guarantee correctness of those conclusions. Speed and RPM are two kinds of significant information in driving records. Although users can easily view speed and RPM on the dashboard of a car, they are not specific. Users need the application to have specific speed and RPM in driving records. In addition, driving time and driving distance are very important for users to know how long and how far they have been driving. Users need the application to count time and distance in records. Besides, average speed can help users to identify comfortable diving speed in different situations. Users need that information in records. What’s more, real-time MPG can let users know how fuel consumption depends on road conditions and average MPG can help users assess fuel consumption of a car. Therefore, real-time MPG and average MPG are both needed in driving records. Moreover, fuel consumption and fuel cost give user intuitional feelings of how much a car cost. Those two kinds of information should be included in records. Furthermore, engine coolant temperature can help user identify if a car is at risk of damage from overheating and control module voltage can help user identify if a car is at normal voltage level. Users need engine coolant temperature and control module voltage to be included in records. At last, driving records should contain bad driving habits like sharp acceleration times and sharp braking times. Users need those two kinds of information to help them overcome bar driving habits.

When users are driving, bad driving habits or unknown malfunctions may result in accidents. Users need the application to remind them of those bad driving habits or unknown malfunctions. Users need to set up speed limit to build over speed protection. Users also need to limit driving hours in case of tired driving. In addition, users also need to set up water temperature limit to remind them of high engine temperature. Users need to set up fuel price to help them calculate how much they will pay for a journey. What’s more, different users use different units of measurement. They want to find their familiar units of measurement in the application. Because users want to record all kinds of driving information, they need to register accounts and then use them to log into the application. Meanwhile, they need the application to memorize their accounts at very first time so they do not need to type their username and password again and again.

Once users have multiple driving records, they may want to search records for a date to see total fuel consumption, total fuel cost and total driving distance on that day. The application should show all records on that day. Another important requirement for users is a full inspection of cars. Users want to check their cars to find if there is a problem. External problems (e.g. headlight malfunction) are easy to find, but internal problems are hard to detect without professional tools. Users need the application to help them identify those malfunctions.

2.2 Functional requirements analysis

In this section, we will analyze user requirements from developers' perspectives. It indicates the functions that they system must have to satisfy user requirements. The format is taken from one of my classes [9]. Index is used for functional requirements retrieval. Name represents the name of the function. Purpose indicates the purpose to design this function. Input parameters are input data required by this function and Output parameters are output data created by this function. Action indicates constrains and scenarios of this function. Exceptions are those wrong operations that will have influence on this function. Remarks remind developers of something important. Cross-references refer to anther function's index which is related to this function. This section will show part of functional requirements. Appendix will include the rest of them.

Index: Access.1

Name: Register

Purpose: To register an account for Riding Mechanic.

Input parameters: Username, password, confirm password

Action: Ensure that username doesn't exist in the system.

Ensure that password and confirm password should not be empty.

Ensure that password and confirm password are same.

Show information of registering an account successfully.

Output parameters: None

Exceptions: Username has already existed.

Password or confirm password is empty.

Password and confirm password are different.

Remarks: Application should show proper error message when users make mistakes.

Cross-references: None

Index: Connection.1

Name: Connect to a car

Purpose: To connect to a car.

Input parameters: None

Action: Ensure that an OBD-II adapter has already plugged into a car.

Ensure that Riding Mechanic is using the OBD-II adapter's Wi-Fi.

Jump to main screen and change status to connected.

Output parameters: None

Exceptions: The OBD-II adapter hasn't plugged into a car.

Riding Mechanic is not using the OBD-II adapter's Wi-Fi.

Remarks: Let users perform actions step by step.

Cross-references: Access.2

Index: Get.1

Name: Get supported PIDs

Purpose: To get supported OBD-II PIDs of a car.

Input parameters: None

Action: Ensure that Riding Mechanic has already connected to a car

Store those information into database.

Output parameters: None

Exceptions: Riding Mechanic hasn't connected to a car.

Remarks: Perform this activity in background.

Cross-references: Connection.1

Index: Save.1

Name: Save driving information

Purpose: To save driving information

Input parameters: None

Action: Ensure that Riding Mechanic has already connected to a car

Ensure that Riding Mechanic has already got supported PIDs.

Ensure that Riding Mechanic is keeping reading data from the car.

Choose to stop reading data and save it.

Output parameters: None

Exceptions: Riding Mechanic hasn't connected to a car.

Riding Mechanic hasn't got supported PIDs.

Riding Mechanic is not keeping reading data from the car.

Remarks: Give users choice to save or not save.

Cross-references: Get.2

2.3 Non-functional requirements analysis

In this section, we discuss non-functional requirements for this application from following aspects.

The first one is security. This application will read data from a car. Taking data protection into consideration, the application will allow users use their personal accounts to login. The last one is usability. This application will give users appropriate message when users go to a different screen. Users can easily know what functions this screen has and how to go to other screens. What's more, this system will allow users perform functions step by step instead of giving all functions at very beginning.

The first one is platform constraint. Because this application is developed for iOS users, it can only be installed and run successfully on an iPhone. Meanwhile, this application has used some application programming interfaces(APIs) supported by Apple Inc. and those APIs are only included in software development kit (SDK) whose iOS version is equal to version 8.0 or higher than 8.0. If users want to use this application without having any troubles, they need to upgrade their iPhones' system to iOS 8.0 or higher.

The last one is operating constraint. Users need to register an account for this app and then used the account to login. The third one is accuracy and precision. This application will show real-time information of a car. Some of real-time data comes from the car and the other data is decided by developers. Therefore, the accuracy and precision of the data is in allowable error scope. The fourth one is reliability. In case of system failure, this application will give proper error message to users when they have unexpected operations.

3. System architecture design

This section will firstly identity goals and constraints of architecture. Then, it will focus on figuring key scenarios out. After that, it will overview whole application including architecture, tools and technologies used in this project. Finally, it will identify key issues that may occur during implementation and talk about how to deal with those issues.

3.1 Goals and constraints of architecture

One of the most important goals is to build a prototype for this project which can help users to understand developers' suggestions for the final product. Meanwhile, users can easily describe and prove requirements that have not been considered and then communicate with developers to figure issues out. This system has used incremental model and by the help of incremental prototyping the time gap between user and software developer has reduced a lot. During development, Dr. Elliott Forbes, advisor of this project, played a role of a customer and met with me very week to give his advice for each development phrase in this project. Users have opportunities to provide feedback while some components are still in development. In addition, that way can influence the outcome of further development.

Another important goal is to identify key technical risks. The first key technical risk is to identify how to connect to a car through a mobile application. This process may involve hardware issues and will take some time to make it clear. After building connection successfully, another key technical risk is to identify how to read data from a car which will take long time to figure out both sending requests to a car and receiving data from a car. The next key technical risk is to guarantee accuracy of the data read form the car. The last key technical risk is to take thread safety into consideration. While the application is running, both UI changing and communication between a car and the application must be executed in different threads and those threads should not have conflicts with each other.

As mentioned in proposal, this application will use an OBD-II Wi-Fi scanner to read data from a car. In the other word, users must use a Wi-Fi type scanner to get access to the car and cannot connect to normal network. This is the biggest constraint in this project. The second constraint is that the application doesn’t have enough storage to keep data and it should be saved in server database. The last constraint is that users need to buy an OBD-II Wi-Fi adapter and then they can use this application. There is no way to connect to a car directly without using any hardware products.

3.2 Key scenarios

A scenario is a broader and more encompassing description of a user's interaction with the system, rather than a path through a use case [11]. Key scenarios in this system include connecting to a car, reading data from a car, saving data into database and searching history data.

When a user wants to connect to a car, he or she should have logged in. The first step is to open the device’s Wi-Fi. Then an OBD-II Wi-Fi scanner need to be plugged into an OBD-II port on the car. Finally, connections can be built by connecting to the OBD-II scanner’s Wi-Fi.

When a user wants to read data from a car, he or she is required to connect to the car first. After building connections successfully, the application will keep reading data from the vehicle and show it to the user. The user can also record driving behaviors for a route.

When a user wants to save data into local database, the system will save data into temporary database instead of server database because network is unavailable when the user is using the scanner to read data from the car. Local database will upload data to server database after login.

When a user wants to search history data, he or she is required to go to trip analysis screen. Local database will request data from server database after login. The user can select specific date to search run history. If the user does not select any date, system will automatically show current date’s trip information.

3.3 Application overview

This section will create an overview of what the application will look like when it is completed. The biggest deployment constraint is that this application cannot have access to the server database while it is running on an iPhone. Some running data can only be stored in local database. Fetching and saving data must be executed between after login.

This application will use a combination of architecture styles. It will involve a separate server system and multiple clients. When a client sent one or more requests, the client will receive those requests and send corresponding responses to the client. There are two main benefits to use the client/server architectural style. The first one is higher security. Because all data is stored on the server, it is easy for database administrations to get a better control of security than client machines. The second one is ease of maintenance. Roles and responsibilities of a computing system are distributed among several servers that are known to each other through a network [11]. Another important architecture style is object-oriented architectural style which will divide system responsibilities into individual objects. Each individual module will take responsibilities for separate functionalities which can be used by other modules by providing corresponding interfaces. There are several advantages to use objective-oriented design. The first one is reusable. Some classes can be used by other classes through polymorphism and abstraction. The second one is testable. Encapsulation of classes can help users to do both unit testing and integration testing. The last one is highly cohesive. By locating only related methods and features in an object, and using different objects for different sets of features, you can achieve a high level of cohesion [11].

This software is a mobile application running on iPhone. The programming language used for client part is objective-c and the integrated development environment is Xcode. As for the server part, the main programming language is Node.js and the integrated development environment is Visual Studio Code. This product will use MySQL as server database and use MySQL Workbench to manage the database.

3.4 Key issues and solution

It is meaningful for developers to identify key issues based on quality attributes where problems often occur when designing an application.

The first quality attribute is system quality which will analyze the whole system from supportability and testability. This product cannot run individually without using an OBD-II Wi-Fi scanner. This is hardware restriction and cannot be solved in software development. Besides, this product is required to run on an iPhone. There may exist unknown problems when users launch it on an iPad. As for testability, it is not too hard for developers to do functional testing, unit resting, integration testing and system testing because this product will take object-oriented architecture design.

The second quality is run-time quality which will analyze the system from manageability, performance, reliability, scalability and security. Manageability describes how easy it is for system administrators to manage the application. This system will take client/server architecture design to make it easy to manage the server database. In general, a system should give proper message when the system is downloading data or uploading data. Some actions which took tens of seconds or a few minutes will be executed in the background instead of keeping users waiting for a long time. Reliability defines how stable the system is when it is running. The system should give corresponding error message instead of breaking down suddenly whenever users make mistakes. Scalability is the ability to create a more advanced version of the system. This product will use generic OBD-II PIDs which can request some basic information from a car. More functionalities can be designed if developers have access to manufactures specific OBD-II PIDs. This product uses incremental model which can help users communicate with developers directly. Users have more information about the product while it is under development.

Another important key issue is thread safety. This application will send multiple standard OBD-II PIDs to the vehicle but the car cannot receive more than two commands at the same time. The main thread of the application will take charge of updating UI and background threads will be responsible for sending OBD-II PIDs. The orders of sending different OBD-II PIDs should be sent based on their priorities. Next chapter will talk how to implement it.

As mentioned in users' requirements, a user can use this application to record vehicle information for a journey. Some information, such as speed and RPM, can be easily retrieved by sending corresponding OBD-II PIDs. But other information like driving distance cannot be read directly from the vehicle. So, this kind of information will be handled by developers. Next chapter will describe how the system dealt with that kind of information.

As for communication between server side and client side, the system uses the Hypertext Transfer Protocol (HTTP) to send request. Figure 2 shows a high-level design for this application. Both information from server side and information from client side must be configurable to be meaningful. What's more, the server side should give client side proper response whenever a client submitted an HTTP request to the server. For example, if a client sends a request to do account verification, the client receives that request and deals with that request. If no such account exists, the server will respond no such account existed. If network crashes suddenly, the server will respond internet occurred error. If a request is in wrong format, the server will respond method is not supported. If that account exists in database, the server will give response together with that account information to the client.

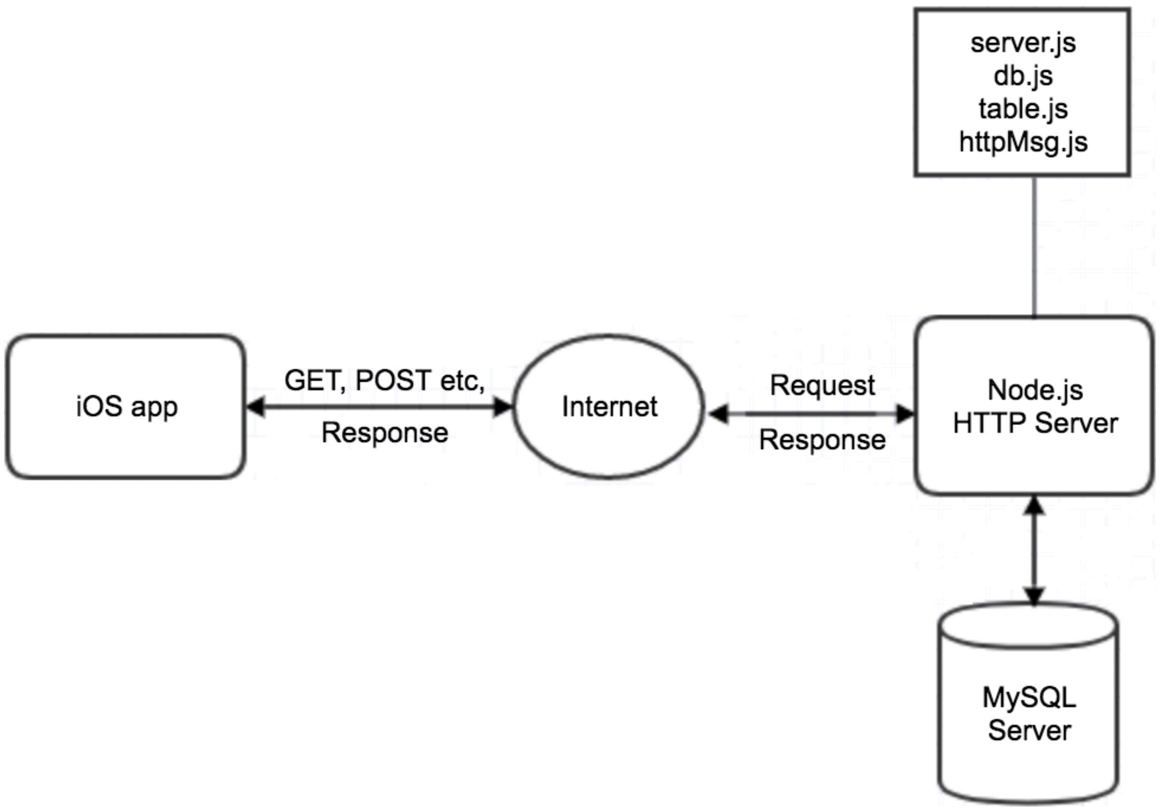


Figure 2. High-level architecture

4. Detailed system design

Detailed design focuses on all implementation details necessary to implement the architecture that is specified. Detailed system design is a low-level design and the main goal of it is to prepare a blue print of a system that meets the goals mentioned in system architecture design. This section described procedures and methods which will be used in detailed system design. According to the requirements of Rational Unified Process (RUP) [13] and Unified Modeling Language (UML) [14], detailed system design will define the behavior of each architectural component in detail. This section will take advantage of use case diagram to give a graphic overview of the actors involved in a system, different functions needed by those actors and how these different functions are interacted. Class diagram will be used to shows internal classes in the system, attributes and operations of each class and the relationship between each class. Sequence diagram will be used to show interactions between objects, functions taken by each object and sequence of messages sent or receive by each object.

4.1 Database design

4.1.1 Database table design

A well-structured database can save disk space by eliminating redundant data, maintain data accuracy and integrity and provide access to the data in useful ways [15]. The first step in database design is to find out "entities" which exist in four kinds: people, things, events, and locations. Everything people can want to put in a database fits into one of these categories. In this project, user is absolutely one of the entities. Another entity is trip which contains information of a car during this trip.

The next step is to determine the relationships between the entities and to determine the cardinality of each relationship. In this project, a user can record a trip or multiple trips. Multiple users cannot record the same trip.

The third step is to find out "attributes" which will be saved as data elements for each entity. For user table, it uses auto-increment id as primary key. Unique "username" and “password” are also needed. Besides, “totalAlertSwitch” should be one of the “attributes” because it can determine if a user wants to open or close alert during a trip. In addition, “speedLimit” and “speedAlertSwitch” are also “attributes” because “speedLimit” determines the warning speed value and “speedAlertSwitch” determines if a user wants to open speed limit alert or not. Similarly, we can get “attributes” like “tiredDrivingHour”, “tiredDrivingAlertSwitch”, “waterTemperatureLimit” and “waterTemperatureAlertSwitch”. What’s more, “fuelPrice” should be one of the “attributes” because users can use it to calculate fuel cost. “unit” should also be one of the “attributes” because users should be able to change unit of measurement.

To design a trip table, it should use auto increment id as primary key and uses “userID” as a foreign key. In addition, it should have “startDateTime” and “endDateTime” to represent when users recorded the trip. Besides, “drivingDistance”, “averageMPG”, “averageSpeed”, “fuelCost”, “sharpAccelerationTime” and “sharpBrakingTime” are also “attributes” because they are necessary information for users to analyze a trip.

After finishing above steps, Entity Relationship Diagram (ERD) can be easily retrieved. ERD gives a graphical overview of the database. Figure 3 shows ERD for this project.

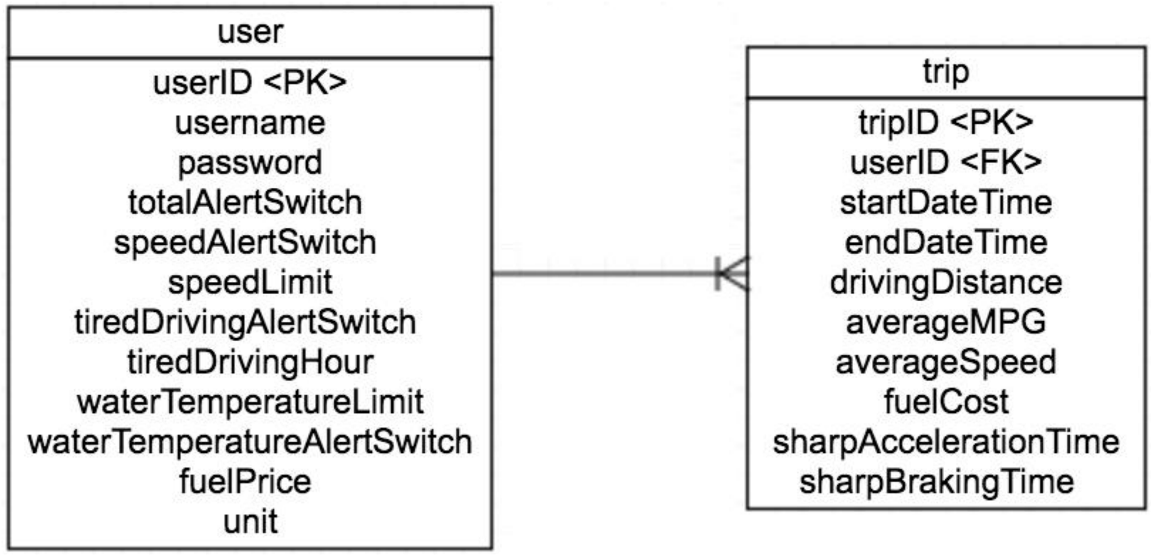
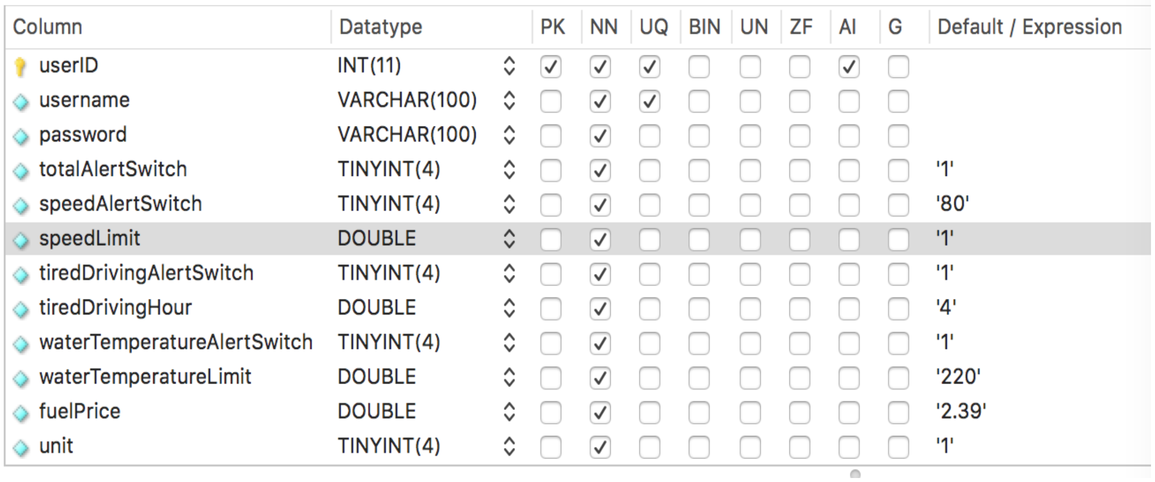


Figure 3. Entity relationship diagram

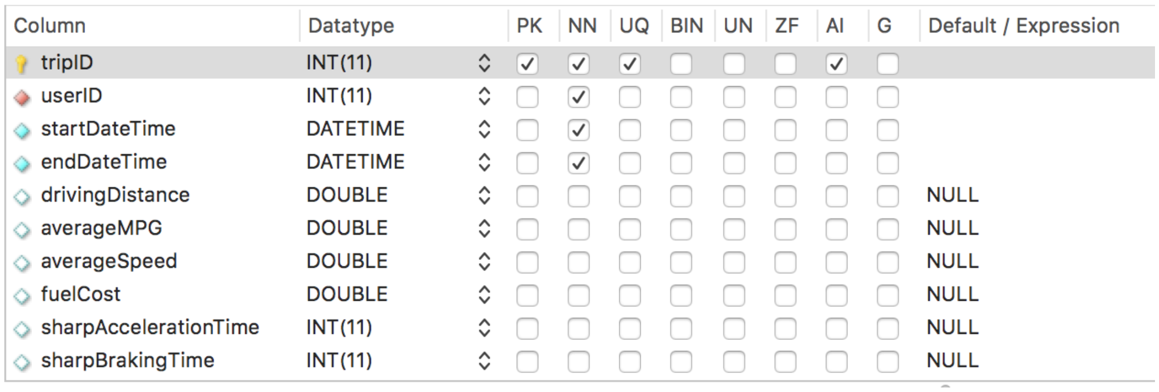
The final step is to define attribute’s data type. There are a lot of different data types. This project will use MySQL as a server database, so all types of data are defined by MySQL. For user table, all attributes should not be null. “userID” should be unique positive self-increasing integer. Type of “username” and “password” are “VARCHAR” and “username” should be unique. “totalAlertSwitch”, “speedAlertSwitch”, “tiredDrivingAlertSwitch”, “waterTemperatureAlertSwitch” and “unit” have the same type. The type is “TINYINT” which is equal to Boolean type. The reason to use “TINYINT” instead of using “Boolean” is that MySQL did not have internal “Boolean” data type. “speedLimit”, “tiredDrivingHour”, “waterTemperatureLimit” and “fuelPrice” used the type of “Double” because all of them are float values. For table trip, “tripID” should be unique positive self-increasing integer. It used “userID” as foreign key and “userID” should be not null and unique. Besides, “startDateTime” and “endDateTime” represented specific date and time so their type is “DATETIME”. “drivingDistance”, “averageMPG”, “averageSpeed” and “fuelCost” are float values and their type is “DOUBLE”. “sharpAccelerationTime” and “sharpBrakingTime” record how many times users have sharp acceleration and sharp braking, so their type is “INT”.

After those steps finished, we can build the database through MySQL Workbench. Figure 4 shows user table and figure 5 shows trip table.



Column	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	G	Default / Expression
userID	INT(11)	✓	✓	✓				✓		
username	VARCHAR(100)		✓	✓						
password	VARCHAR(100)		✓							
totalAlertSwitch	TINYINT(4)		✓							'1'
speedAlertSwitch	TINYINT(4)		✓							'80'
speedLimit	DOUBLE		✓							'1'
tiredDrivingAlertSwitch	TINYINT(4)		✓							'1'
tiredDrivingHour	DOUBLE		✓							'4'
waterTemperatureAlertSwitch	TINYINT(4)		✓							'1'
waterTemperatureLimit	DOUBLE		✓							'220'
fuelPrice	DOUBLE		✓							'2.39'
unit	TINYINT(4)		✓							'1'

Figure 4. User table in database



Column	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	G	Default / Expression
tripID	INT(11)	✓	✓	✓				✓		
userID	INT(11)		✓							
startDateTime	DATETIME		✓							
endDateTime	DATETIME		✓							
drivingDistance	DOUBLE									NULL
averageMPG	DOUBLE									NULL
averageSpeed	DOUBLE									NULL
fuelCost	DOUBLE									NULL
sharpAccelerationTime	INT(11)									NULL
sharpBrakingTime	INT(11)									NULL

Figure 5. Trip table in database

4.1.2 Database password encryption and decryption

Password encryption is designed to grantee account safety. When a user has created an account for this application, the password will be encrypted when it is insert into server database table. This technology is supported by MySQL. The encryption method is "AES_Encypty (password, key)". By using this method, the password in the table will be a messy code and no one can read useful information form that messy code. The key of password worked as an index. The application can only find the password through the key. This project will use unique username as the key of the password.

It is easy for developers get the password which is not encrypted. Once it has been encrypted, the most important thing is decryption. The decryption method is "AES_Decypty (password, key)". This decryption key should be same with the key for encryption. As mentioned above, it is the username. It is not enough to get the real password because the password will be converted into binary value after using that method. So, another method, "cast (AES_Decypty (password, key)", AS CHAR)", should be used after initial decryption. So, the first step of decryption is to convert a messy code into binary value by using "AES_Decypty (password, key)". The second step is to convert that binary value into char value by using "cast (AES_Decypty (password, key)", AS CHAR)". Then, the real password can be retrieved.

4.2 Use case diagram

All use cases are retrieved from user requirements and the use case diagram gives an overview of usage requirements for a system. Figure 6 shows the use case diagram of this project.

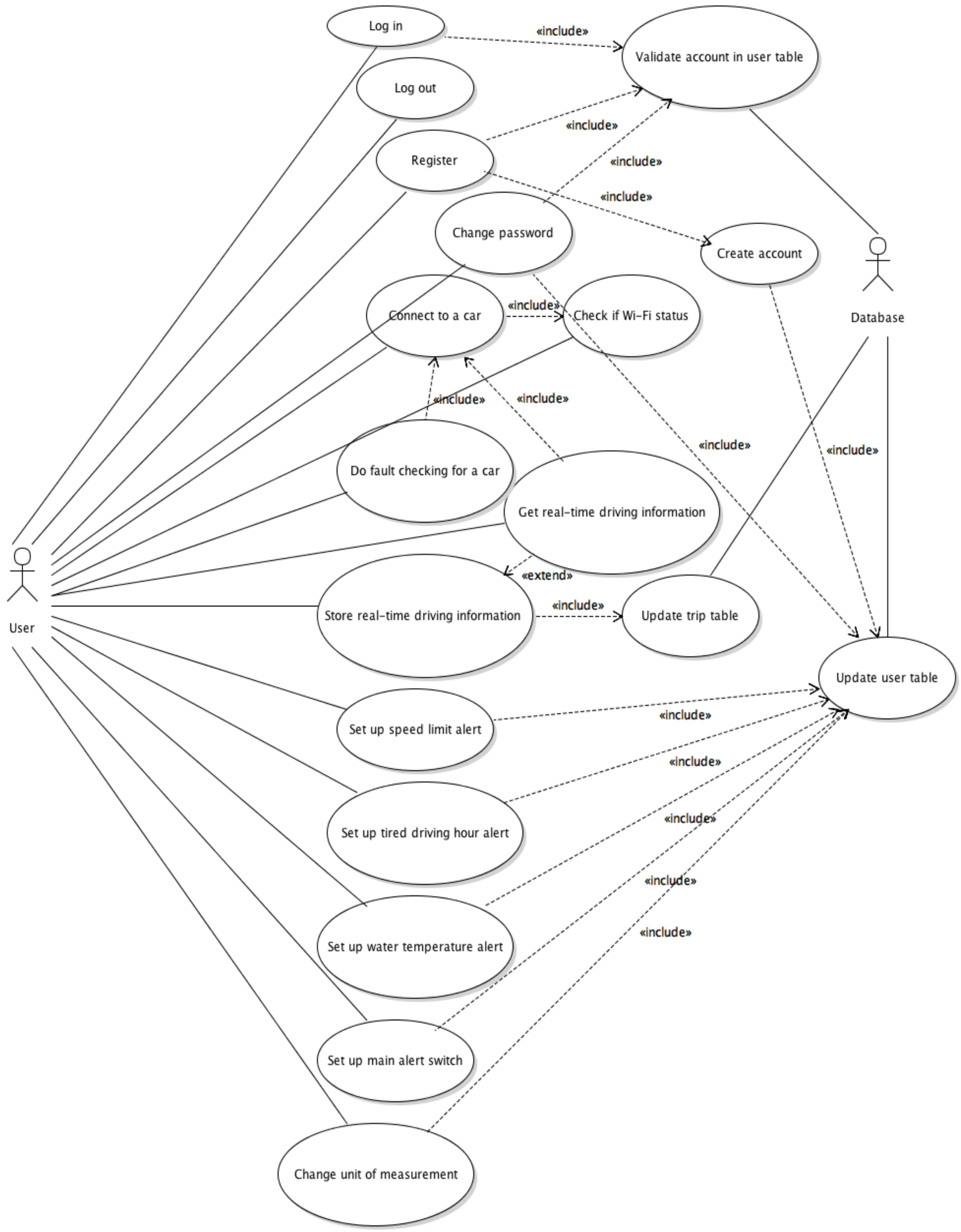
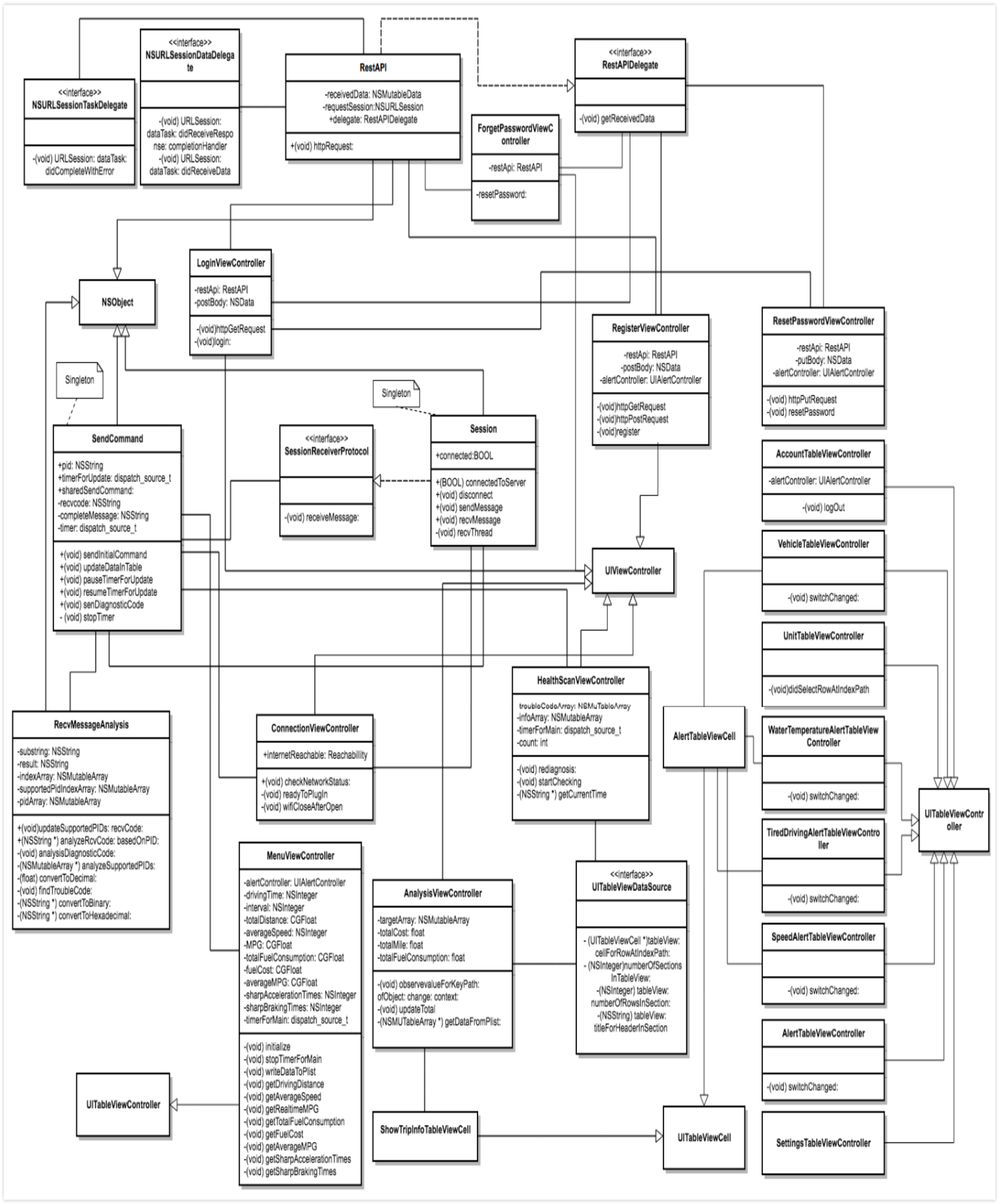


Figure 6. Use case diagram

4.3 Class diagram

Class diagram is the core part of UML diagram and it describes the structure of a system by modeling its classes, attributes, operations, and relationships between objects. Figure 7 shows the class diagram of this project.



<<interface>>
NSURLSessionTaskDelegate
-(void) URLSession: dataTask: didCompleteWithError
<<interface>>
NSURLSessionDataDelegate
-(void) URLSession: dataTask: didReceiveResponse: completionHandler
-(void) URLSession: dataTask: didReceiveData
RestAPI
-receivedData: NSMutableData
-requestSession:NSURLSession
+delegate: RestAPIDelegate
+(void) httpRequest:
<<interface>>
RestAPIDelegate
-(void) getReceivedData
ForgetPasswordViewController
-restApi: RestAPI
-resetPassword:
LoginViewController
-restApi: RestAPI
-postBody: NSData
-(void)httpGetRequest
-(void)login:
NSObject
RegisterViewController
-restApi: RestAPI
-postBody: NSData
-alertController: UIAlertController
-(void)httpGetRequest
-(void)httpPostRequest
-(void)register
ResetPasswordViewController
-restApi: RestAPI
-putBody: NSData
-alertController: UIAlertController
-(void) httpPutRequest
-(void) resetPassword
Singleton
SendCommand
+pid: NSString
+timerForUpdate: dispatch_source_t
+sharedSendCommand:
-recvcode: NSString
-completeMessage: NSString
-timer: dispatch_source_t
+(void) sendInitialCommand
+(void) updateDataInTable
+(void) pauseTimerForUpdate
+(void) resumeTimerForUpdate
+(void) senDiagnosticCode
- (void) stopTimer
<<interface>>
SessionReceiverProtocol
-(void) receiveMessage:
Singleton
Session
+connected:BOOL
+(BOOL) connectedToServer
+(void) disconnect
+(void) sendMessage
+(void) recvMessage
-(void) recvThread
AccountTableViewController
-alertController: UIAlertController
-(void) logOut
UIViewController
VehicleTableViewController
-(void) switchChanged:
UnitTableViewController
-(void)didSelectRowAtIndexPath
RecvMessageAnalysis
-substring: NSString
-result: NSString
-indexArray: NSMutableArray
-supportedPidIndexArray: NSMutableArray
-pidArray: NSMutableArray
+(void)updateSupportedPIDs: recvCode:
+(NSString *) analyzeRcvCode: basedOnPID:
-(void) analysisDiagnosticCode:
-(NSMutableArray *) analyzeSupportedPIDs:
-(float) convertToDecimal:
-(void) findTroubleCode:
-(NSString *) convertToBinary:
-(NSString *) convertToHexadecimal:
ConnectionViewController
+internetReachable: Reachability
+(void) checkNetworkStatus:
-(void) readyToPlugIn
-(void) wifiCloseAfterOpen
HealthScanViewController
troubleCodeArray: NSMuTableArray
-infoArray: NSMutableArray
-timerForMain: dispatch_source_t
-count: int
-(void) rediagnosis:
-(void) startChecking
-(NSString *) getCurrentTime
AlertTableViewCell
WaterTemperatureAlertTableViewController
-(void) switchChanged:
UITableViewController
TiredDrivingAlertTableViewController
-(void) switchChanged:
MenuViewController
-alertController: UIAlertController
-drivingTime: NSInteger
-interval: NSInteger
-totalDistance: CGFloat
-averageSpeed: NSInteger
-MPG: CGFloat
-totalFuelConsumption: CGFloat
-fuelCost: CGFloat
-averageMPG: CGFloat
-sharpAccelerationTimes: NSInteger
-sharpBrakingTimes: NSInteger
-timerForMain: dispatch_source_t
-(void) initialize
-(void) stopTimerForMain
-(void) writeDataToPlist
-(void) getDrivingDistance
-(void) getAverageSpeed
-(void) getRealtimeMPG
-(void) getTotalFuelConsumption
-(void) getFuelCost
-(void) getAverageMPG
-(void) getSharpAccelerationTimes
-(void) getSharpBrakingTimes
AnalysisViewController
-targetArray: NSMutableArray
-totalCost: float
-totalMile: float
-totalFuelConsumption: float
-(void) observevalueForKeyPath: ofObject: change: context:
-(void) updateTotal
-(NSMUTableArray *) getDataFromPlist:
<<interface>>
UITableViewDataSource
- (UITableViewCell *)tableView: cellForRowAtIndexPath:
- (NSInteger)numberOfSections InTableView:
-(NSInteger) tableView: numberOfRowsInSection:
-(NSString) tableView: titleForHeaderInSection
SpeedAlertTableViewController
-(void) switchChanged:
AlertTableViewController
-(void) switchChanged:
UITableViewController
ShowTripInfoTableViewCell
UITableViewCell
SettingsTableViewController

Figure 7. Class diagram

4.4 Sequence diagram

This section shows the sequence diagrams of some important activities in this application, illustrating different components interact with each other and how messages passed between components in different activities.

Figure 8 shows sequence diagram of login, it describes login scenario; Figure 9 shows sequence diagram of registration, it describes registration scenario; Figure 10 shows sequence diagram of resetting password, it describes a user forgets password and uses the system to reset password; Figure 11 shows sequence diagram of connection, it indicates the scenario to connect to a car; Figure 12 shows sequence diagram of reading data from a car; Figure 13 shows sequence diagram of searching data for a specific date; Figure 14 shows sequence diagram of checking a car

While the system is sending OBD-II PIDs to the car to get real-time information, it is very important to decide the order of sending different OBD-II PIDs because the OBD-II system can only receive and give response to one OBD-II PID for each time.

The application will receive message and deal with that message every half second. In the other word, the application can only receive two kinds of message each second which meant the application can only send two kinds of message each second. Table 7 shows the frequency of sending each kind of OBD-II PIDs.

	1s	2s	3s	4s	5s	6s	7s	8s	9s	10s	Rule
0105 (Coolant Temperature)										√	Time%10==0
010C (RPM)	√		√		√		√		√		Time is odd
010D (Speed)	√	√	√	√	√	√	√	√	√	√	Time%1==0
0110 (MAF Air flow rate)		√		√				√			Time is even and! Time%10==0 and! Time%6==0
0142 (Control module voltage)						√					Time%6==0

Table 7. Frequency of sending different OBD-II PIDs

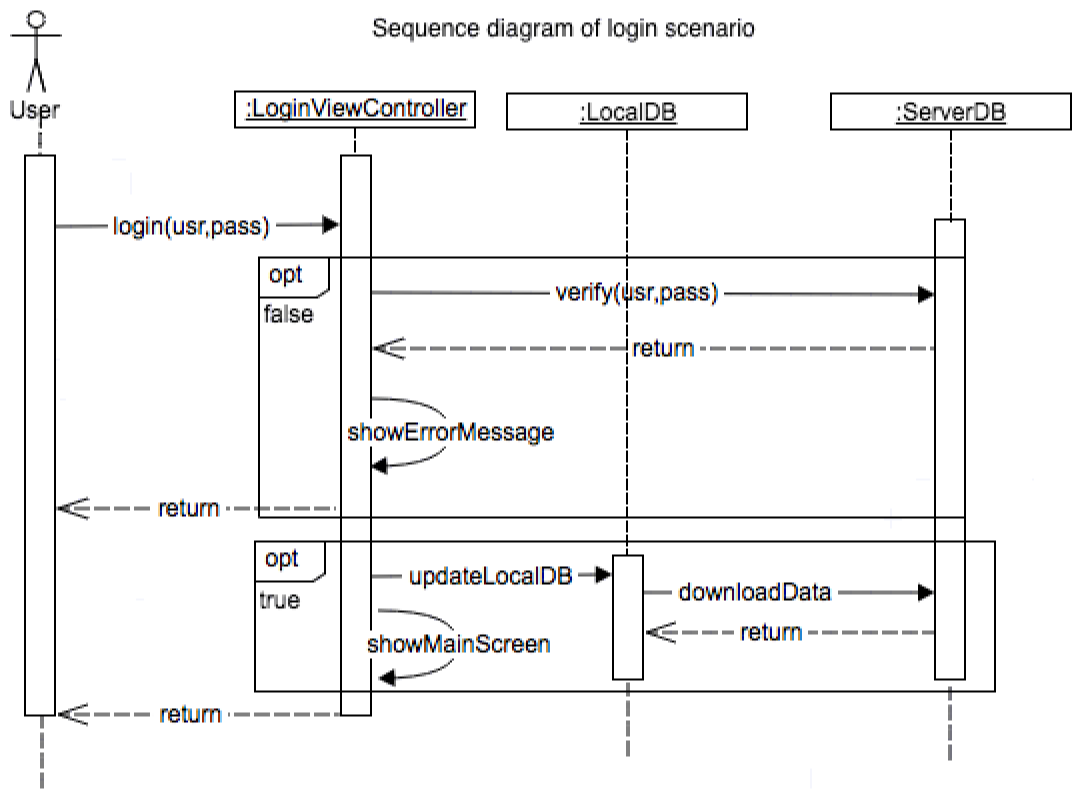


Figure 8. Sequence diagram of login

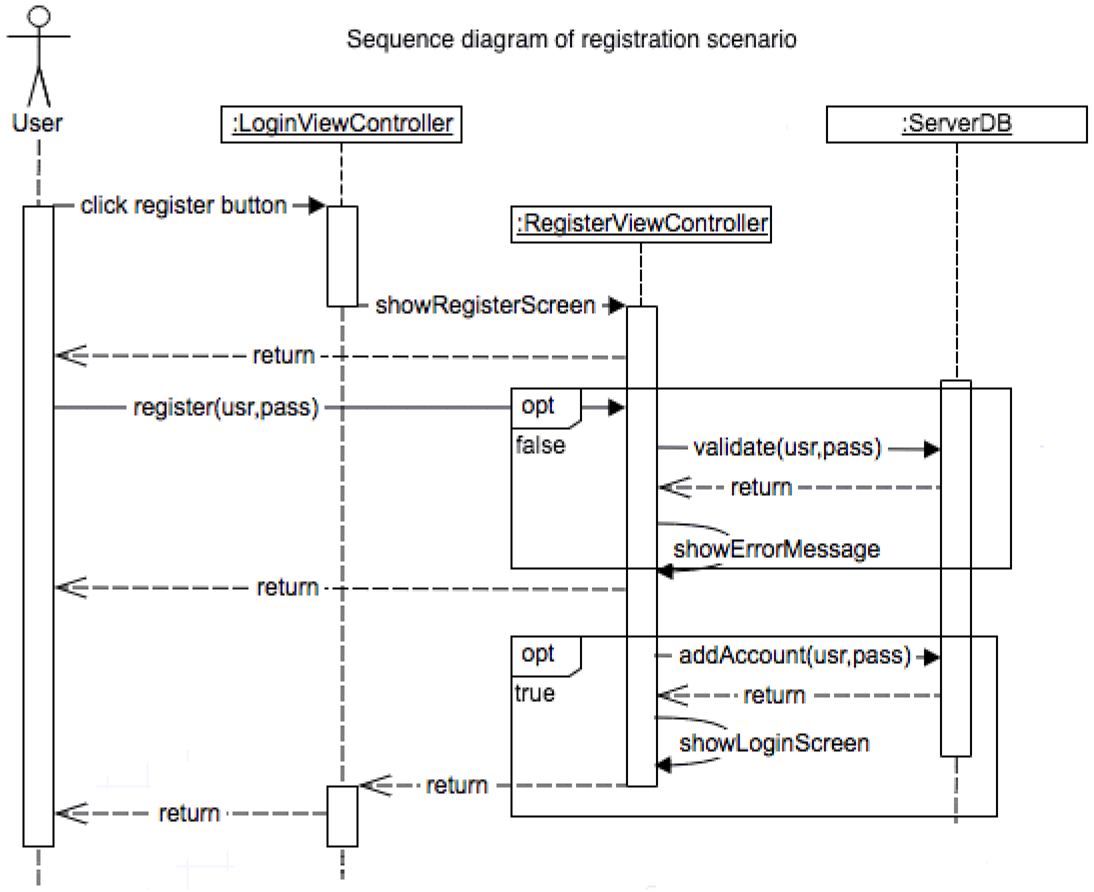


Figure 9. Sequence diagram of registration

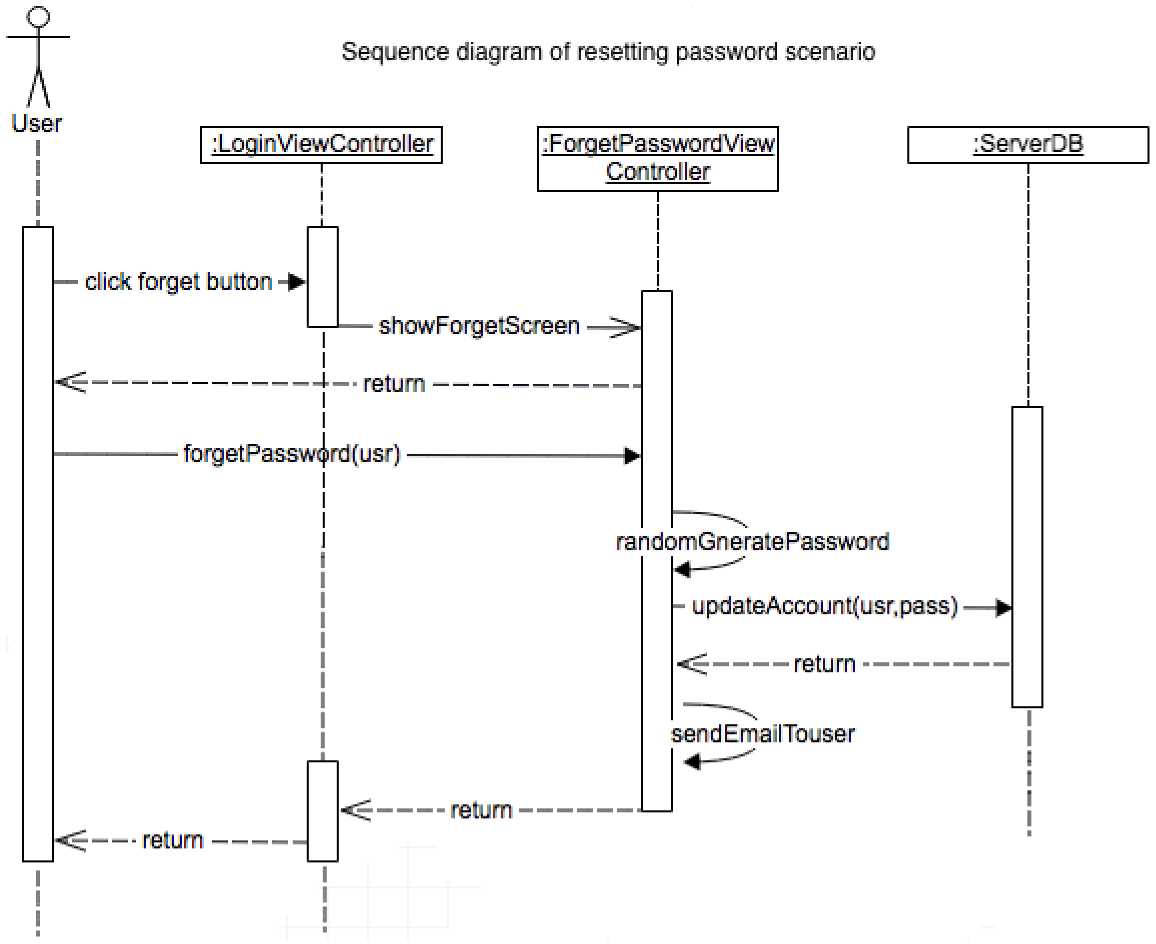


Figure 10. Sequence diagram of resetting password

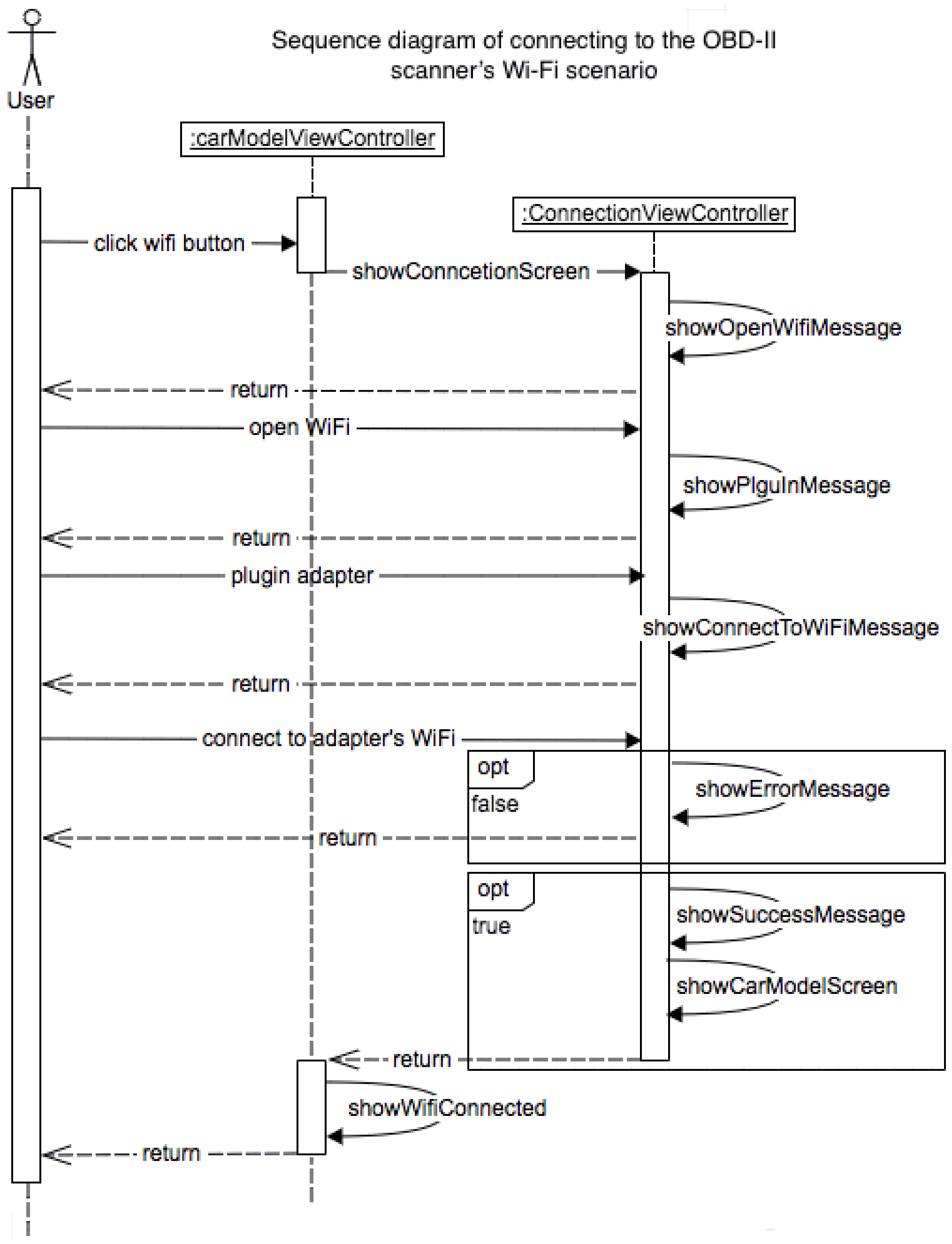


Figure 11. Sequence diagram of connection

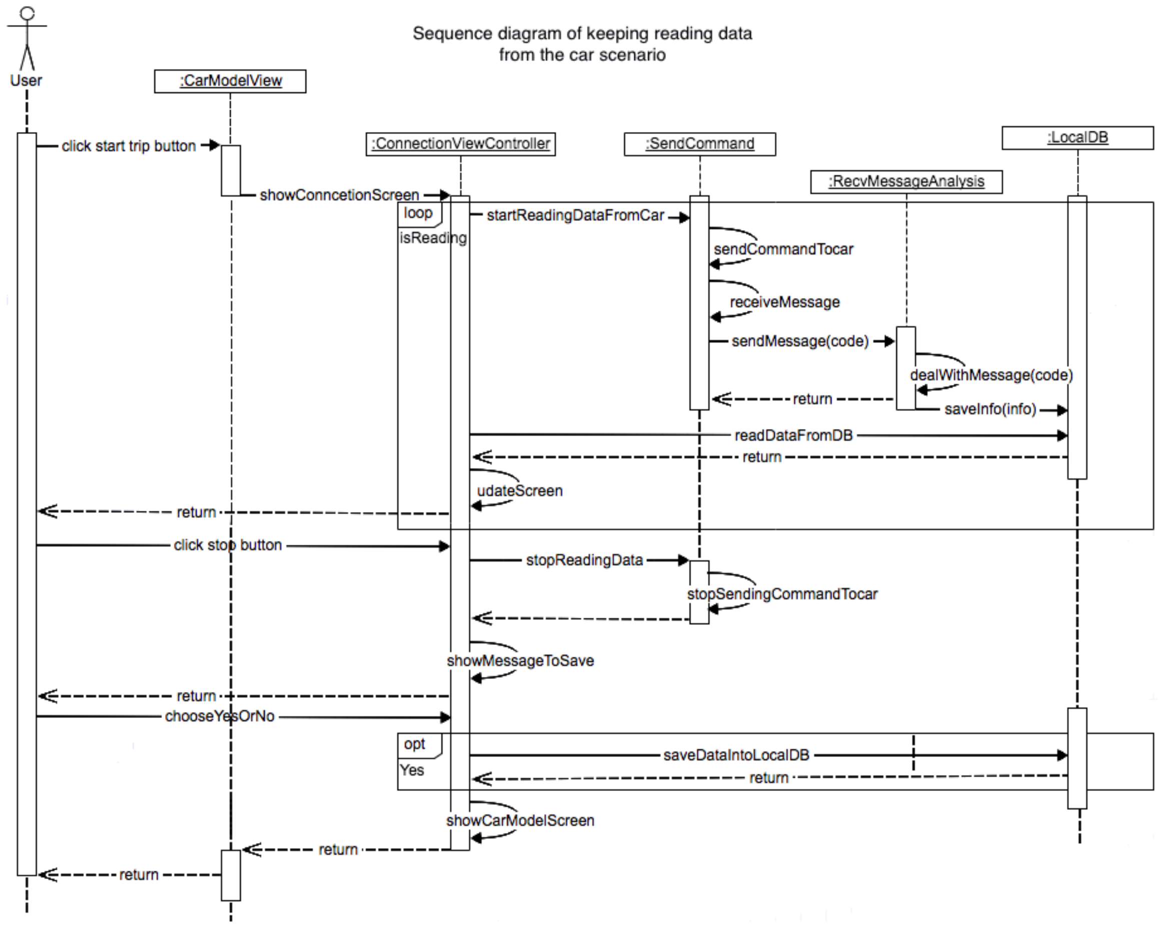


Figure 12. Sequence diagram of reading data

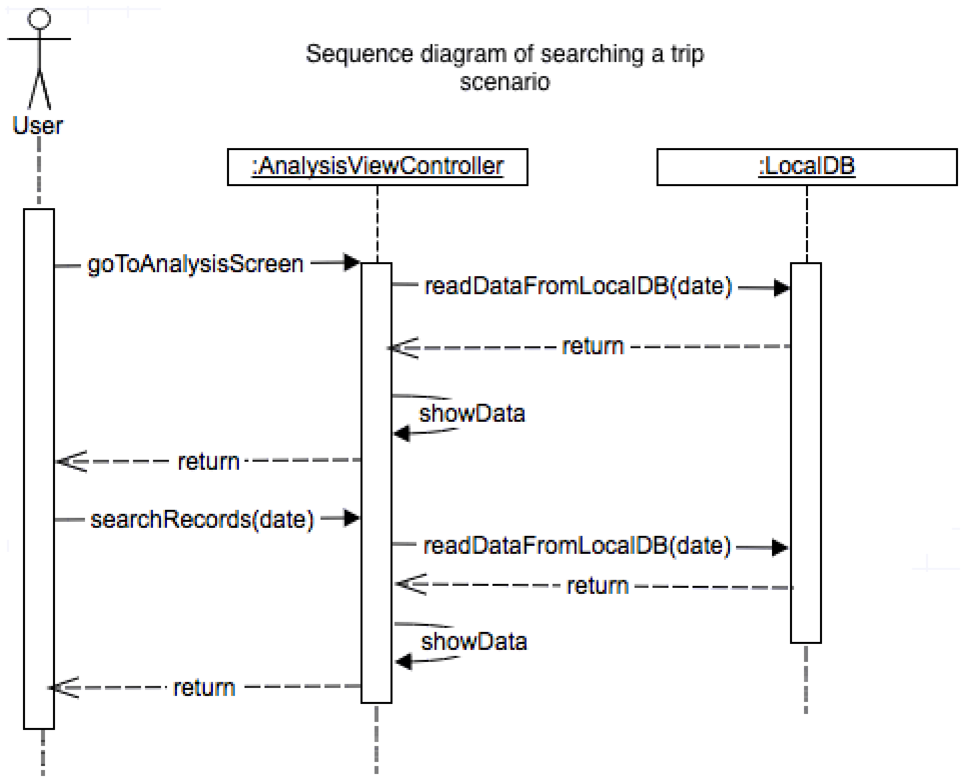


Figure 13. Sequence diagram of searching a trip

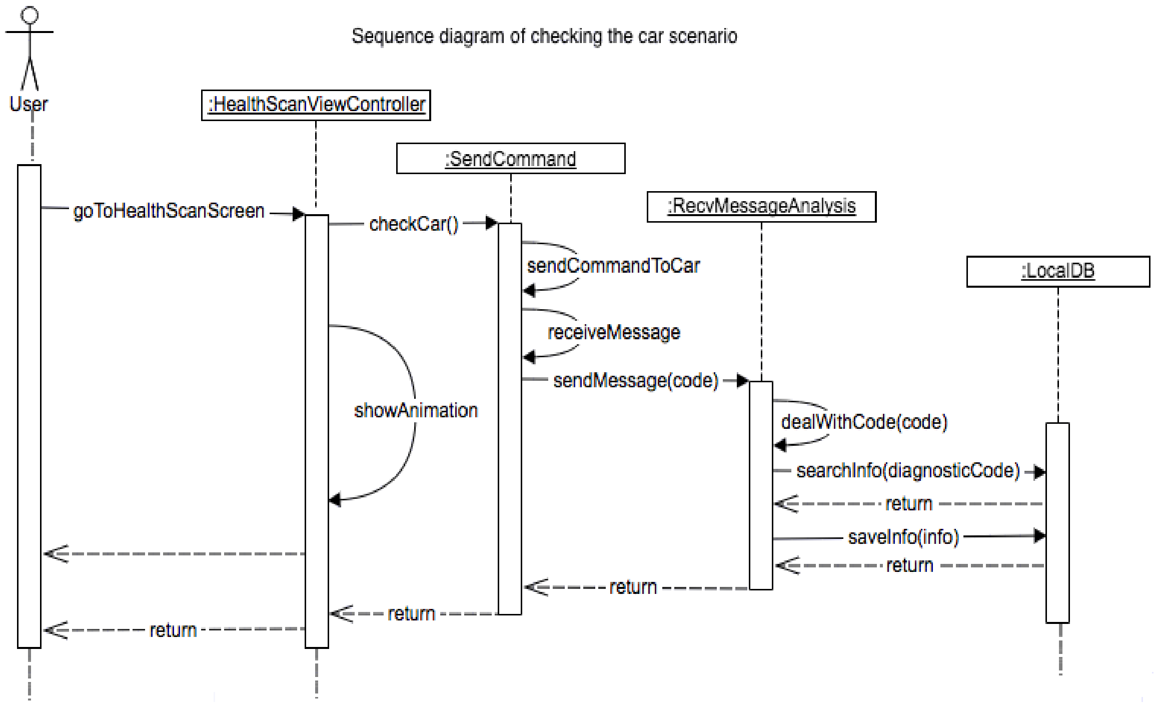


Figure 14. Sequence diagram of checking a car

4.5 How driving distance is calculated

As mentioned above, some information of a journey cannot be read directly from the vehicle and that kind of data will be worked out by the system. This part will describe the process of calculating the driving distance. Physics formulas has taught us distance is equal to elapsed time plus average velocity. We can apply it into calculating driving distance. The system can read the current speed form the vehicle every second. By using the theory of differential calculus [16], the error of the distance will be reduced by using smaller time interval and the smallest time interval for reading current speed is one second.

The first step for calculating distance is to get average speed in one second. For example, the speed of the third second is ten miles per hour and the speed of the fourth second is twenty miles per hour, so the average speed between the third second and the fourth second is fifteen miles per hour. Then, the second step is to calculate the distance in one second by using the physics formula (s=v*t). The last step is to add every second's distance together to get total distance.

4.6 User interface design

User interface design is one of the most important parts in software development. The quality of user interface has huge influence on user experience of the product. A good GUI (graphical user interface) not only catches a first-time user's eye but also motivates the user to use the product again. We consider introduce four principles of UI design [17], reading gravity and minimization of cognitive load, using pre-existing mental models, and use of color to direct attention.

In addition, we further emphasize [18] that GUI should be usable to as many as possible, GUI must report appropriate error messages at the right time, and providing appropriate help at the right time to motivate the user to continue, and we must try to protect the users from introducing errors into the system (error prevention). Those UI design principles will be used as references to design UI of this product.

I will first introduce some basic patterns for mobile navigation. The tab bar pattern is one of the most frequently used patterns in mobile applications. Many popular applications, like Facebook, Instagram and Twitter, use the tab bar pattern in their design. It originated in desktop design. The tab bar usually put navigation icons at the bottom of a screen to fit finger touch. Users can navigate directly to the screen associated with the item. With tab bar navigation, users can access all main functionalities without multiple touches; in addition, users can know where they are and what to do when they are using the application. In this product, all main functions are represented by bottom icons in tab bar. Figure 15 shows an example.

Another popular UI design pattern is navigation bar. Almost all popular mobile applications have navigation bars. In general, a navigation appeared between the status bar and an application screen. Users can view the status and application screens without having any troubles. A navigation bar usually has a title, a back button and a control button. The back button is always associated with previous screen and appears on the left side of the bar. The title always shows information of current screen and appears on the middle of the bar. The control button is related to a Cancel or Done button, and used mainly to manage users' actions. This product makes use of navigation bars. Figure 16 shows an example.

In addition to UI design patterns mentioned above, some popular UI elements are also frequently used in famous mobile applications. For example, Messenger used table view to show contract list. The table view is used to manage data which has similar data structure or belongs to similar category. Table view can help us to show additional contents by scrolling with it. Some table views also have headers and footers which can make it easy to tract a long table. Figure 17 shows an example in this product.

Alert is another important UI element in mobile application design. Many applications used alerts to remind users of something important. An alert usually contains a title and optional information. The title indicated what that the alert's meaning and optional information told users what they can do. A widely-used example for using alert is AMBER Alert. Almost all people can view that information immediately once an

AMBER Alert occurred. From this example, we can easily find out the advantages of using alerts in mobile application. Figure 18 shows an example in this product.

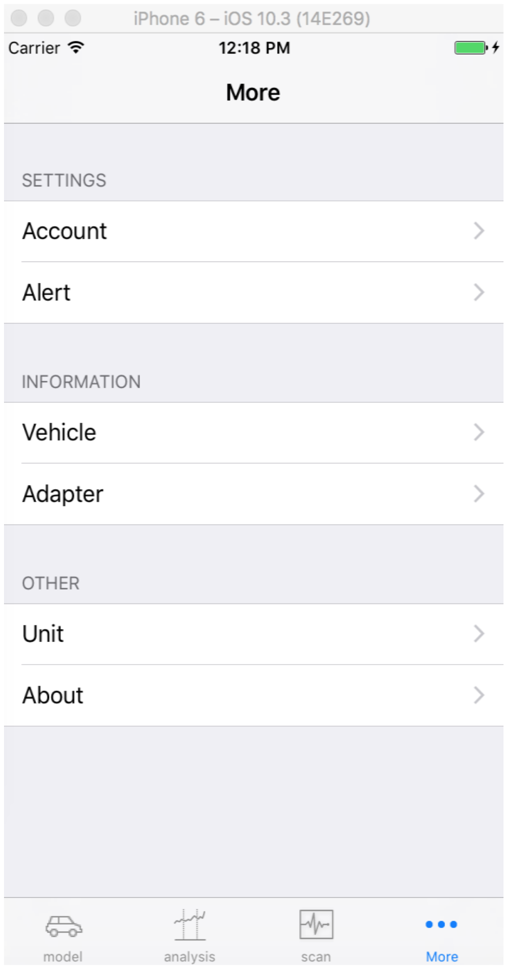


Figure 15. Example of using tab bar

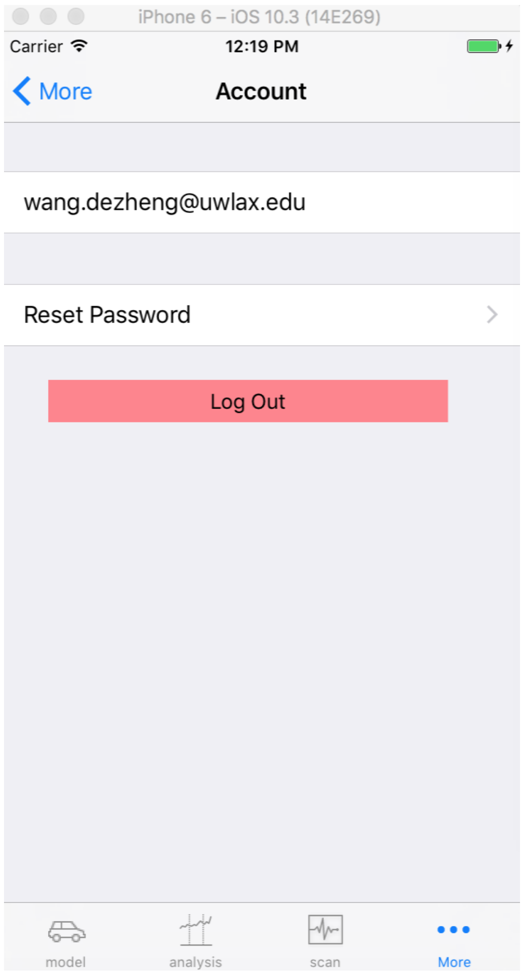
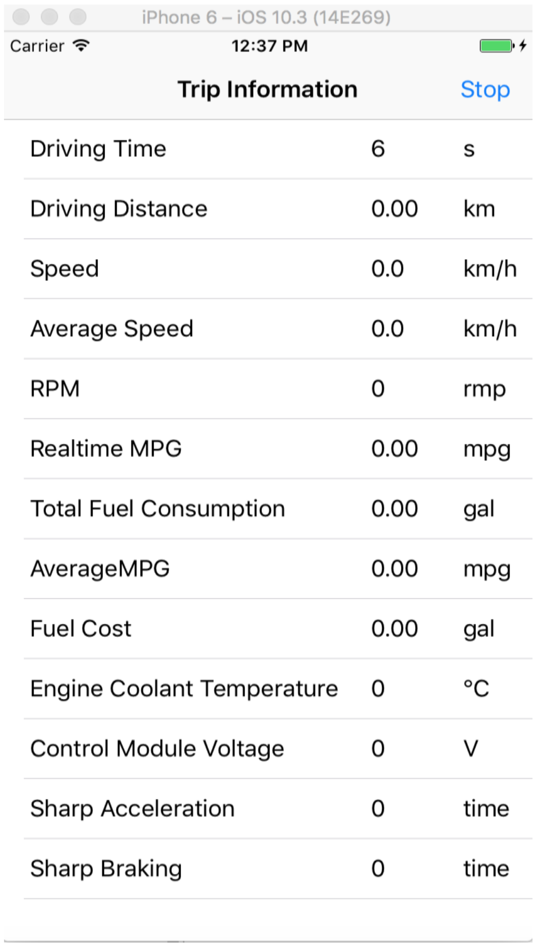


Figure 16. Example of using navigation bar



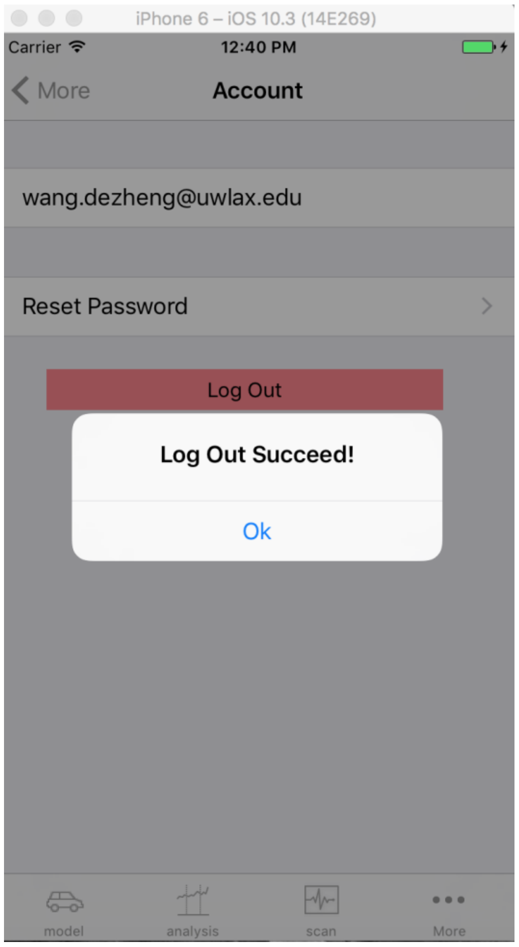
iPhone 6 – iOS 10.3 (14E269)

Carrier 12:37 PM

Trip Information Stop

Driving Time	6	s
Driving Distance	0.00	km
Speed	0.0	km/h
Average Speed	0.0	km/h
RPM	0	rmp
Realtime MPG	0.00	mpg
Total Fuel Consumption	0.00	gal
AverageMPG	0.00	mpg
Fuel Cost	0.00	gal
Engine Coolant Temperature	0	°C
Control Module Voltage	0	V
Sharp Acceleration	0	time
Sharp Braking	0	time

Figure 17. Example of using table view



iPhone 6 – iOS 10.3 (14E269)

Carrier 12:40 PM

More Account

wang.dezheng@uwlax.edu

Reset Password

Log Out

Log Out Succeed!

Ok

Figure 18. Example of using alert

5. System implementation

Instead of using objective-c codes, this section will use pseudocode to talk about the main algorithm and logic in this application. Pseudocode can be understood by most people.

5.1 Login module

This part used pseudocode to describe the login procedure. It involved following functions: Login (), SetInitialStatus (username), SetUser(username), SetTrip (userID).

```
If username and password had been memorized
Go to main screen
Else if users click login button
    Login () //User click login button will call this function
Else if users click register button
    Go to register screen
Else
    Go to forget password screen

Login () {
   Set username to username text field value
   Set password to password text field value
   If username or password is empty
        Alert “Username or password should not be empty”
   Else
        Send get request to the server //request body contains username
        Query password based on username in user table
        If return value is empty
               Alert “Incorrect username or password”
        Else if return value is not equal to password
               Alert “Incorrect username or password”
        Else if return value is equal to password
            Store username, password, userID in local database
            SetInitialStatus (username) // background thread
            Go to main screen
}

SetInitialStatus (username) {
    SetUser(username) // upload local data (update local database if needed)
    SetTrip (userID) // upload local data (update local database if needed)
}

SetUser (username) {
    If user table is empty in local database
        Send get request to the server //request body contains username
        Query all related information based on username in user table in server
        Set a delay // need time to download data
        Set userInfo to return value
        Update user table in local database based on userInfo
    Else
        Query all related information based on username in local database
        Set userInfo to return value
        Send put request to the server // request body contains userInfo
        Update user table in server database based on userInfo
}

SetTrip (userID) {
    If new trip table is empty in local database
        Send get request to the server //request body contains userID
        Query all related information based on userID in trip table in server
        Set a delay // need time to download data
        Set tripInfo to return value
        Update trip table in local database based on tripInfo
    Else
        Query all related information based on userID in new trip table in local
        Set newTripInfo to return value
        Send post request to the server // request body contains newTripInfo
        Add newTripInfo to trip table in server database
}
```

5.2 Registration module

This part used pseudocode to describe the registration procedure. It involved following functions: Verify ().

```
If users click submit button in register screen
    Verify () //User click submit button will call this function
Else users choose to go back login screen
    Go to login screen

Verify () {
       Set username to username text field value
       Set password to password text field value
       Set confirm to confirm password field value
       If username is not in email format
              Alert “Username should in email format”
       Else if password or confirm is empty
              Alert “Password or confirm password should not be empty”
       Else if password is not equal to confirm
              Alert “Passwords must match”
       Else
               Send post request to the server // request contains username and
password
              Insert username and password into user table
              If return value is false
                     Alert “Sorry, this username had been used!”
              Else
                     Alert “Register Succeed”
                     Go to login screen
}
```

5.3 Forget password module

This part used pseudocode to describe how to reset password when a user forgot his password.

```
Set username to username text field value
If username is empty
    Alert “You must enter username”
Else
    Send get request to the server //request body contains username
    Query that username in user table in server database
    If return value does not have content
        Alert “Username does not exist”
    Else
        Set newPassword to random six-digit number
        Send email to username // username is in email format
Send put request to the server
        Update user table in server database based on username and newPassword
        Go to login screen
```

5.4 Device connecting module

This part used pseudocode to describe the procedure of connecting to an OBD-II Wi-Fi scanner. It involved flowing functions: SendInitialCommand (), ReveiveMessage ().

```
while Wi-Fi is open
Do
    Show plug in button
    Users click plug in button
    Show connecting to Wi-Fi button
    If device is connected to adapter’s Wi-Fi
        Set wifiStatus to connected
        Store wifiStatus in local database
        SendInitialCommand () // this method comes from SendCommand class
        Go back main screen
    Else
        Alert “Connection failure”

SendInitialCommand () {
    Create timer
    Set timer priority to background
    Timer:
        If last command had not been sent
              Use Session class
              Session send commands
              ReveiveMessage ()
        Else
              Stop timer
}

```

```
ReveiveMessage () {
    Set message to receive message
    Use regular expression to deal with message
    Analyze message and get value// value represents vehicle supported command
    Store that value in local database
}
```

5.5 Trip information module

This part used pseudocode to describe how to read real-time information from a vehicle. It involved following functions: UpdateDataInTable (), ReveiveMessage ().

```
while Wi-Fi status is connected && users don not click stop button
Do
    UpdateDataInTable () //this method comes from SendCommand class
    Create timer
    Set timer to priority to main
    Timer:
        Read data from local database
        Analyze and compute data to get other data
        Show data in table view
        Reload table
    If users click stop button
        Alert ("Save trip information? Yes: No")
        If users select yes
            Save those data and date in new trip table in local database
            Go back main screen

UpdateDataInTable () {
    Create timer
    Set timer priority to background
    Use Session class
    Timer:
        Get supported commands from local database
        Session send supported commands
        ReveiveMessage ()
}

ReveiveMessage () {
    Set message to receive message
    Use regular expression to deal with message
    Analyze message and get corresponding value // value represents vehicle data
    Store that value in local database
}
```

5.6 Health scan module

This part used pseudocode to describe how to read fault codes from a vehicle and show corresponding diagnostic information to users. It involved following functions: SenDiagnosticCode (), ReveiveMessage ().

```
 // this method will be executed automatically when users go to health scan screen
StartChecking () {
        SenDiagnosticCode ()
        Set timer to priority to main
        Timer:
                If timer ends
                        Read diagnostic information and code from local database
                        Show diagnostic information and code in table view
                        Show rediagnostic button
                If users click rediagnostic button
                        Set rediagnostic button and table view to hidden
                        StartChecking ()
                Else
                        Update UI to let users wait for scanning
}

SenDiagnosticCode () {
        Create timer
        Set backgroundTimer to priority to background
        backgroundTimer:
                Session send diagnostic commands
                ReveiveMessage ()
}

ReveiveMessage () {
    Set message to receive message
    Use regular expression to deal with message
    Analyze message and get corresponding value//value is diagnostic code
    Search diagnostic information in fault code table based on diagnostic code
    Store diagnostic information and diagnostic code in local database
}
```

6. Testing

Software development always involves software testing which is the process of evaluating correctness, integrity and quality of the software. In the other word, software testing is the process of fixing bugs to satisfy user requirements. This section will talk about each phrase in software testing in detail.

6.1 Testing plan

In unit testing, testing strategy is to test server side and client side separately. For server side, it will use Postman to send different kinds of requests and check response. After bugs fixed in server side, it will use UI testing in Xcode to test each individual user interface. Compared to other UI testing framework, the major benefit of UI testing in Xcode is that it can record developers' operations to the system. In the other words, developers can perform all kinds of operations in each user interface to test all kinds of situations and then UI testing will memorize those operations and transfer them into testing codes which can be executed automatically. In a word, UI testing in Xcode is a kind of scene simulation and can bring lots of convenience to testers. Integration testing will also use UI testing in Xcode to testing the whole system including server side and client side. As for system testing, it will mainly test data accuracy. A tester will use this application to record plenty of data. Another tester will use recording functions in the car to get plenty of data. Then a tester will use graphs to make comparisons and then get conclusions.

6.2 Unit testing

6.2.1 Testing for server module

This part is responsible for testing different requests which are sent from the client side to the server side. Test cases are generated based on requests which have been used in the application. Other types of requests will not be taken into consideration. Table 8 shows details of each test case. Scenario indicates the testing environment. URL represented all kinds of requests occurred in the application. Return value is a message from the server side. The status indicates whether the test case passed or not.

Test case	**Scenario**	**URL**	**Return value**	**Status**
1	User table has this account	138.49.101.87:9000/userSettings /wang.dezheng@uwlax.edu Method: GET	All user information related to username 'wang.dezheng@uwlax.edu'	Passed
2	User table doesn't have this account	138.49.101.87:9000/userSettings /wang.dezheng@uwlax.edu Method: GET	[]	Passed
3	Trip table has this trip	138.49.101.87:9000/tripInfo/1 Method: GET	All trip information related to userID '1'	Passed
4	Trip table doesn't have this trip	138.49.101.87:9000/tripInfo/1 Method: GET	[]	Passed
5	Add a new user to user table	138.49.101.87:9000/userInfo requesrBody: {username: yuan.zhihao@uwlax.edu, password :123} Method: POST	'true	Passed
6	Add an existed user to user table	138.49.101.87:9000/userInfo requesrBody: {username: yuan.zhihao@uwlax.edu, password :123} Method: POST	'false'	Passed
7	Add a new trip to trip table	138.49.101.87:9000/tripInfo requesrBody: {trip information} Method: POST	'true'	Passed
8	Update a user's information	138.49.101.87:9000/userInfo requesrBody: {updated user information} Method: PUT	'true'	Passed

Table 8. Test cases for the server side

6.2.2 Testing for client module

Testing for client module will be divided into five parts. The first part is to test the session module which is responsible for sending commands and receiving messages from the car. The second part is to test login module. The third part is to test registration module. The fourth part is to test resetting password function. The last part is to test searching history trips module.

Table 9 shows all test cases for testing session module. Test cases are generated based on those OBD-PIDs which have been used in this application. Each test case has three kinds of return messages and all three are right. The reason why one OBD-II PID has three return values is that the frequency of the system receiving messages will have influence on message sent by the OBD-II system. Some message cannot be sent during that time interval and it will be sent during next time interval.

Test case	**Scenario**	**Command**	**Return message**	**Status**
9	To get supported PIDs from 0101-0120	0100	"0100\r"	Passed
		0100	"41 00 BF 9F A8 93 \r\r>"	Passed
		0100	"0100\r 41 00 BF 9F A8 93 \r\r>"	Passed
10	To get supported PIDs from 0121-0140	0120	"0120\r"	Passed
		0120	"41 20 80 07 F1 19 \r\r>"	Passed
		0120	"0120\r 41 20 80 07 F1 19 \r\r>"	Passed
11	To get supported PIDs from 0141-0160	0140	"0140\r"	Passed
		0140	"41 40 FE F0 00 00 \r\r>"	Passed
		0140	"0140\r 41 40 FE F0 00 00 \r\r>"	Passed
12	To get engine coolant temperature	0105	"0105\r"	Passed
		0105	"41 05 88 \r\r>"	Passed
		0105	"0105\r 41 05 88 \r\r"	Passed
13	To get engine RPM	010C	"010C\r"	Passed
		010C	"41 0C 09 92 \r\r>"	Passed
		010C	"010C\r 41 0C 09 92 \r\r>"	Passed
14	To get vehicle speed	010D	"010D\r"	Passed
		010D	"41 0D 00 \r\r>"	Passed
		010D	"010D\r 41 0D 00 \r\r>"	Passed
15	To get	0110	"0110\r"	Passed

	MAF air flow rate	0110	“41 10 01 1B \r \r>”	Passed
		0110	“0110\r 41 10 01 1B \r \r>”	Passed
16	To get control module voltage	0142	“0142\r”	Passed
		0142	“41 42 36 41 \r \r>”	Passed
		0142	“0142\r 41 42 36 41 \r \r>”	Passed

Table 9. Test cases for session module

Testing for registration module will use UI testing form Xcode. Firstly, it will try to register for an existed account to see if the system will give error message. Then it will use a new username. It will try to have inconsistent passwords in password field and confirm password field to see if the system will give error message. Then it will use consistent password to register for a new account. Following codes are testing codes for registration module.

```
-(void)testRegisterFunction {
    XCUIApplication *app = [[XCUIApplication alloc] init];
    [app.buttons[@"Register"] tap];
    XCUIElement *emailAddressTextField = app.textFields[@"Email address"];
    [emailAddressTextField tap];
    [emailAddressTextField typeText:@"793013053@qq.com"];
    XCTAssert(app.staticTexts[@"Sorry, this email had been used!"].exists);

    [app.buttons[@"Clear text"] tap];
    [emailAddressTextField typeText:@"w793013053@gmail.com"];
    XCTAssert(!app.staticTexts[@"Sorry, this email had been used!"].exists);

    XCUIElement *passwordSecureTextField = app.secureTextFields[@"Password"];
    [passwordSecureTextField tap];
    [passwordSecureTextField typeText:@"123"];
    XCUIElement *confirmPasswordSecureTextField = app.secureTextFields[@"Confirm password"];
    [confirmPasswordSecureTextField tap];
    [confirmPasswordSecureTextField typeText:@"12"];
    XCUIElement *submitButton = app.buttons[@"Submit"];
    [submitButton tap];
    XCTAssert(app.alerts[@"Error!"].exists);

    [app.alerts[@"Error!"].buttons[@"OK"] tap];
    [confirmPasswordSecureTextField tap];
    [confirmPasswordSecureTextField typeText:@"123"];
    [submitButton tap];
    XCTAssert(app.alerts[@"Register Succeed"].exists);
    [app.alerts[@"Register Succeed"].buttons[@"OK"] tap];
}
```

Testing for login module will also use UI testing. Firstly, it will try to login with correct username and incorrect password to see if the system will give error message. Then it will incorrect username and correct password to see if the system will give error message. Finally, it will use correct username and password to login.

```
-(void) testLoginFunction{
    XCUIApplication *app = [[XCUIApplication alloc] init];
    XCUIElement *emailAddressTextField = app.textFields[@"Email address"];
    [emailAddressTextField tap];
    [emailAddressTextField typeText:@"w793013053@gmail.com"];

    XCUIElement *passwordSecureTextField = app.secureTextFields[@"Password"];
    [passwordSecureTextField tap];
    [passwordSecureTextField typeText:@"123456"];
    XCUIElement *loginButton = app.buttons[@"Login"];
    [loginButton tap];
    XCTAssert(app.staticTexts[@"Incorrect username or password"].exists);

    [emailAddressTextField tap];
    [app.buttons[@"Clear text"] tap];
    [passwordSecureTextField tap];
    [app.buttons[@"Clear text"] tap];
    [loginButton tap];
    XCTAssert(app.staticTexts[@"Incorrect username or password"].exists);

    [emailAddressTextField tap];
    [emailAddressTextField typeText:@"w79301305333@gmail.com"];
    [passwordSecureTextField tap];
    [passwordSecureTextField typeText:@"123"];
    [loginButton tap];
    XCTAssert(app.staticTexts[@"Incorrect username or password"].exists);

    [emailAddressTextField tap];
    [app.buttons[@"Clear text"] tap];
    [emailAddressTextField typeText:@"w793013053@gmail.com"];
    [passwordSecureTextField tap];
    [app.buttons[@"Clear text"] tap];
    [passwordSecureTextField typeText:@"123"];
    [loginButton tap];

    XCTAssert(!emailAddressTextField.exists);
    XCTAssert(!passwordSecureTextField.exists);
    XCTAssert(!loginButton.exists);
    XCTAssert(!app.buttons[@"Register"].exists);
    XCTAssert(!app.staticTexts[@"Incorrect username or password"].exists);
}
```

Testing for resetting password module will also use UI testing in. Firstly, it will try to enter wrong username to see if the system will give error message. Then it will click reset button without having any username to see if the system will crash. Finally, it will enter correct username and click reset button. The tester should also check the email to ensure new password has been sent. The new password should be same with password value stored in database.

```objc
-(void)testForgetPasswordFunction{
    XCUIApplication *app = [[XCUIApplication alloc] init];
    [app.buttons[@"Help"] tap];
    XCUIElement *emailTextField = app.textFields[@"Email address"];
    XCTAssert(emailTextField.exists);

    [emailTextField tap];
    [emailTextField typeText:@"w7930130533@gmail.com"];
    XCUIElement *resetButton = app.buttons[@"Reset"];
    [resetButton tap];
    XCTAssert(app.alerts[@"Error!"].exists);

    XCUIElement *okButton = app.alerts[@"Error!"].buttons[@"OK"];
    [okButton tap];
    XCUIElement *clearTextButton = app.buttons[@"Clear text"];
    [clearTextButton tap];
    [resetButton tap];

    XCTAssert(app.alerts[@"Error!"].exists);
    [okButton tap];
    [emailTextField typeText:@"w793013053@gmail.com"];
    [resetButton tap];

    XCTAssert(!resetButton.exists);
}
```

Testing for resetting connection module will also use UI testing in. Firstly, it will login. Secondly, it will go the connection screen. Thirdly, it will plug the 0BD-II scanner in. Finally, it will connect to scanner’s Wi-Fi and users can start recording trip information.

```objc
XCUIApplication *app = [[XCUIApplication alloc] init];
XCUIElement *emailAddressTextField = app.textFields[@"Email address"];
[emailAddressTextField tap];
[emailAddressTextField typeText:@"w793013053@gmail.com"];

XCUIElement *passwordSecureTextField = app.secureTextFields[@"Password"];
[passwordSecureTextField tap];
[passwordSecureTextField typeText:@"123"];
[app.buttons[@"Login"] tap];
[app.navigationBars[@"Riding Mechanic"].buttons[@"Wi Fi"] tap];
[app.buttons[@"Device plugged please scan now"] tap];
[app.buttons[@"Yes, Wifi conneted"] tap];

XCTAssert(app.buttons[@"Start Trip"].exists);
```

Testing for starting trip module will also use UI testing in. Firstly, it will connect login and connect to OBD-II scanner's Wi-Fi. Secondly, it will use star trip function to read data from the car and show vehicle information. Finally, it will stop recording and choose to save trip information.

```
XCUIApplication *app = [[XCUIApplication alloc] init];
XCUIElement *emailAddressTextField = app.textFields[@"Email address"];
[emailAddressTextField tap];
[emailAddressTextField typeText:@"w793013053@gmail.com"];

XCUIElement *passwordSecureTextField = app.secureTextFields[@"Password"];
[passwordSecureTextField tap];
[passwordSecureTextField typeText:@"123"];
[app.buttons[@"Login"] tap];
[app.navigationBars[@"Riding Mechanic"].buttons[@"Wi Fi"] tap];
[app.buttons[@"Device plugged please scan now"] tap];
[app.buttons[@"Yes, Wifi conneted"] tap];
[app.buttons[@"Start Trip"] tap];

XCUIElementQuery *tablesQuery = app.tables;
[tablesQuery.staticTexts[@"Driving Distance"] tap];
[tablesQuery.staticTexts[@"Realtime MPG"] tap];
[tablesQuery.staticTexts[@"Total Fuel Consumption"] tap];
[tablesQuery.staticTexts[@"Fuel Cost"] tap];
[tablesQuery.staticTexts[@"Sharp Braking"] tap];
[app.navigationBars[@"Trip Information"].buttons[@"Stop"] tap];
[app.alerts[@"Do you want to store trip information?"].buttons[@"YES"] tap];
```

Testing for trip analysis module will also use UI testing in. Firstly, it will connect login and go to analysis screen. Then, it will show trip information based on current date. Finally, it will select different date to search trip information.

```
XCUIApplication *app = [[XCUIApplication alloc] init];
XCUIElement *emailAddressTextField = app.textFields[@"Email address"];
[emailAddressTextField tap];
[emailAddressTextField typeText:@"w793013053@gmail.com"];

XCUIElement *passwordSecureTextField = app.secureTextFields[@"Password"];
[passwordSecureTextField tap];
[passwordSecureTextField typeText:@"123"];
[app.buttons[@"Login"] tap];

XCUIElementQuery *tabBarsQuery = app.tabBars;
[tabBarsQuery.buttons[@"analysis"] tap];

XCUIApplication *app2 = app;
[app2.buttons[@"8"] tap];
[app2.buttons[@"10"] swipeRight];
[app2.buttons[@"7"] tap];
[app2.buttons[@"6"] tap];
```

```
[app2.buttons[@"5"] tap];
[app2.buttons[@"4"] tap];
[app2.buttons[@"3"] tap];
[app2.buttons[@"2"] tap];
[app2.buttons[@"1"] tap];
[tabBarsQuery.buttons[@"More"] tap];
[app2.tables.staticTexts[@"Account"] tap];
[app.tables.buttons[@"Log Out"] tap];
[app.alerts[@"Log Out Succeed!"].buttons[@"Ok"] tap];
```

Testing for more module will also use UI testing. Firstly, it will connect login and go to more screen. Secondly, it will go to speed alert screen to change speed limit. Thirdly, it will go to tire driving screen to change tired driving limit.

Fourthly, it will go to engine coolant temperature screen to change coolant temperature limit. Fifthly, it will go to vehicle screen to change fuel price. Sixthly, it will go to adapter screen and about screen to view detailed information. Finally, it will use function of changing password and function of logout.

```
XCUIApplication *app = [[XCUIApplication alloc] init];
XCUIElement *emailAddressTextField = app.textFields[@"Email address"];
[emailAddressTextField tap];
[emailAddressTextField typeText:@"w793013053@gmail.com"];

XCUIElement *passwordSecureTextField = app.secureTextFields[@"Password"];
[passwordSecureTextField tap];
[passwordSecureTextField typeText:@"123"];
[app.buttons[@"Login"] tap];
[app.tabBars.buttons[@"More"] tap];

XCUIElementQuery *tablesQuery2 = app.tables;
XCUIElementQuery *tablesQuery = tablesQuery2;
[tablesQuery.staticTexts[@"Alert"] tap];
[tablesQuery.switches[@"Alert"] tap];
[tablesQuery.switches[@"Alert"] tap];
[tablesQuery.staticTexts[@"Speed Alert"] tap];
[tablesQuery.switches[@"Speed Alert"] tap];
[tablesQuery.switches[@"Speed Alert"] tap];

XCUIElement *textField = [[tablesQuery2.cells containingType:XCUIElementTypeStat
icText identifier:@"Speed Limit"] childrenMatchingType:XCUIElementTypeTextField]
.element;
[textField tap];
[textField typeText:@"80"];
[app.navigationBars[@"Speed Alert"].buttons[@"Alert"] tap];

XCUIElement *tiredDrivingAlertStaticText = tablesQuery.staticTexts[@"Tired Drivi
ng Alert"];
[tiredDrivingAlertStaticText tap];
[tablesQuery.switches[@"Tired Driving Alert"] tap];
[tablesQuery.switches[@"Tired Driving Alert"] tap];
```

```

XCUIElement *textField2 = [[tablesQuery2.cells containingType:XCUIElementTypeStaticText identifier:@"Tired Driving Hour Limit"] childrenMatchingType:XCUIElementTypeTextField].element;
[textField2 tap];
[textField2 typeText:@"4"];

XCUIElement *alertButton = app.navigationBars[@"Tired Driving Alert"].buttons[@"Alert"];
[alertButton tap];
[tiredDrivingAlertStaticText tap];
[alertButton tap];
[tablesQuery.staticTexts[@"Water Temperature Alert"] tap];
[tablesQuery.switches[@"Water Temperature Alert"] tap];
[tablesQuery.switches[@"Water Temperature Alert"] tap];

XCUIElement *textField3 = [[tablesQuery2.cells containingType:XCUIElementTypeStaticText identifier:@"Water Temperature Limit"] childrenMatchingType:XCUIElementTypeTextField].element;
[textField3 tap];
[textField3 typeText:@"220"];
[app.navigationBars[@"Water Temperature Alert"].buttons[@"Alert"] tap];
[app.navigationBars[@"Alert"].buttons[@"More"] tap];

XCUIElement *vehicleStaticText = tablesQuery.staticTexts[@"Vehicle"];
[vehicleStaticText tap];

XCUIElement *textField4 = [tablesQuery2.cells childrenMatchingType:XCUIElementTypeTextField].element;
[textField4 tap];
[textField4 typeText:@"2.35"];

XCUIElement *moreButton = app.navigationBars[@"Vehicle"].buttons[@"More"];
[moreButton tap];
[vehicleStaticText tap];
[moreButton tap];
[tablesQuery.staticTexts[@"Adapter"] tap];
[app.navigationBars[@"Connectivity"].buttons[@"More"] tap];
[tablesQuery.staticTexts[@"Unit"] tap];
[app.navigationBars[@"Unit"].buttons[@"More"] tap];
[tablesQuery.staticTexts[@"About"] tap];
[app.navigationBars[@"About"].buttons[@"More"] tap];
[tablesQuery.staticTexts[@"Account"] tap];
[tablesQuery.staticTexts[@"Reset Password"] tap];

XCUIElement *currentPasswordSecureTextField = app.secureTextFields[@"Current password"];
[currentPasswordSecureTextField tap];
[currentPasswordSecureTextField typeText:@"123"];

XCUIElement *newPasswordSecureTextField = app.secureTextFields[@"New password"];
[newPasswordSecureTextField tap];
[newPasswordSecureTextField tap];
[newPasswordSecureTextField typeText:@"111"];

XCUIElement *confirmNewPasswordSecureTextField = app.secureTextFields[@"Confirm new password"];
```

```
[confirmNewPasswordSecureTextField tap];
[confirmNewPasswordSecureTextField tap];
[confirmNewPasswordSecureTextField typeText:@"111"];
[app.buttons[@"Submit"] tap];
[app.alerts[@"Password change succeed"].buttons[@"OK"] tap];
[tablesQuery.buttons[@"Log Out"] tap];
[app.alerts[@"Log Out Succeed!"].buttons[@"Ok"] tap];
```

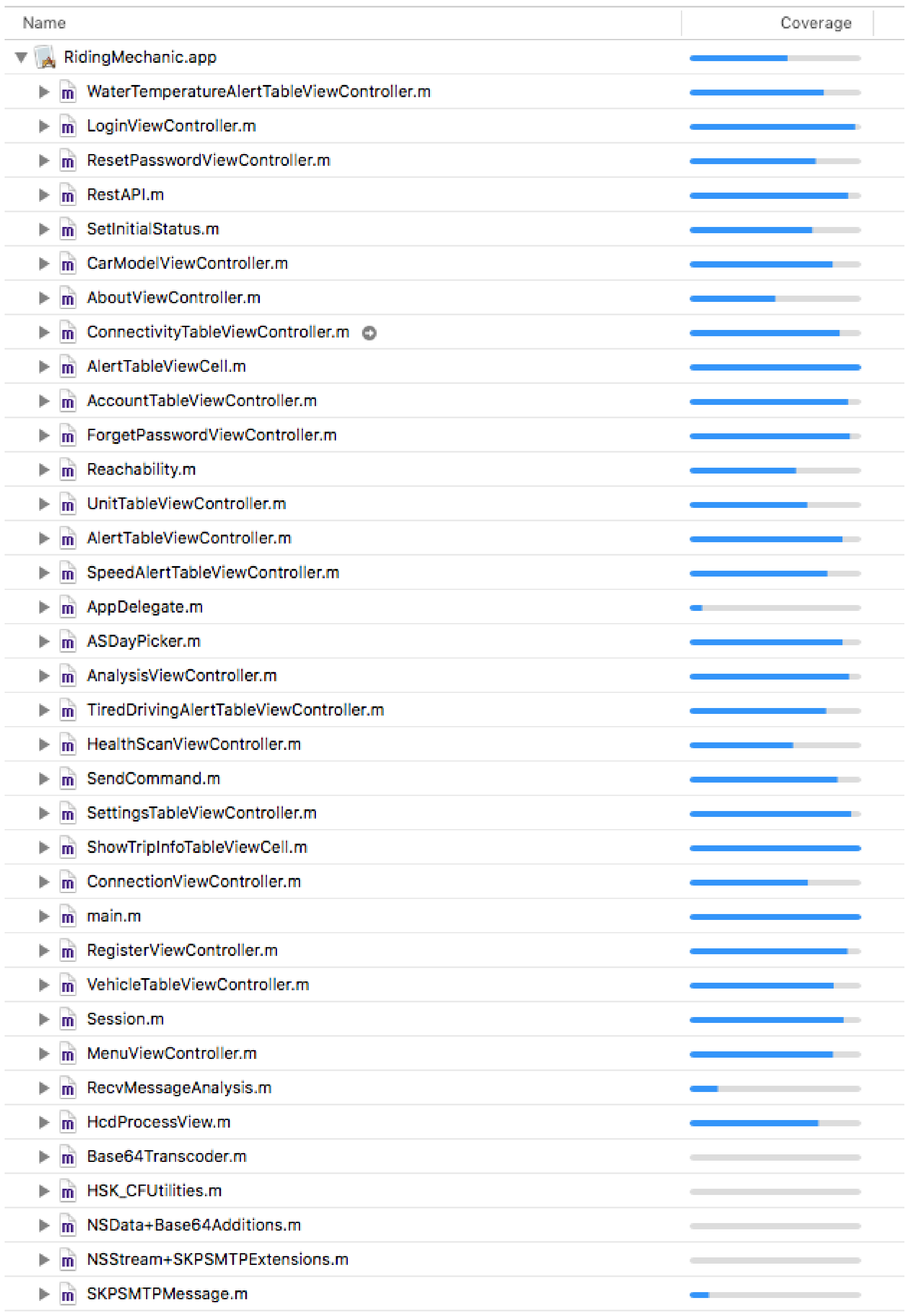
Testing for health scan module will also use UI testing. Firstly, it will connect login and go to health scan screen. The system will show diagnostic information when the screen is loaded. Finally, it will use re-diagnosis function to get newest diagnostic information from the vehicle.

```
XCUIApplication *app = [[XCUIApplication alloc] init];
XCUIElement *emailAddressTextField = app.textFields[@"Email address"];
[emailAddressTextField tap];
[emailAddressTextField typeText:@"w793013053@gmail.com"];

XCUIElement *passwordSecureTextField = app.secureTextFields[@"Password"];
[passwordSecureTextField tap];
[passwordSecureTextField typeText:@"123"];
[app.buttons[@"Login"] tap];
[app.tabBars.buttons[@"scan"] tap];
[app.buttons[@"Rediagnosis"] tap];
```

6.2.3 Code coverage

UI testing from Xcode also provids users with testing code coverage. Figure 19 has shown the testing code coverage from Xcode. As it indicated in the figure, some classes are not covered. The reason that they have little testing code coverage is some classes are external classes or from third-party library which has lots of functions. But only one or two functions will be used for this application among those functions.



Name	Coverage
RidingMechanic.app	
WaterTemperatureAlertTableViewController.m	
LoginViewController.m	
ResetPasswordViewController.m	
RestAPI.m	
SetInitialStatus.m	
CarModelViewController.m	
AboutViewController.m	
ConnectivityTableViewController.m	
AlertTableViewCell.m	
AccountTableViewController.m	
ForgetPasswordViewController.m	
Reachability.m	
UnitTableViewController.m	
AlertTableViewController.m	
SpeedAlertTableViewController.m	
AppDelegate.m	
ASDayPicker.m	
AnalysisViewController.m	
TiredDrivingAlertTableViewController.m	
HealthScanViewController.m	
SendCommand.m	
SettingsTableViewController.m	
ShowTripInfoTableViewCell.m	
ConnectionViewController.m	
main.m	
RegisterViewController.m	
VehicleTableViewController.m	
Session.m	
MenuViewController.m	
RecvMessageAnalysis.m	
HcdProcessView.m	
Base64Transcoder.m	
HSK_CFUtilities.m	
NSData+Base64Additions.m	
NSStream+SKPSMTPExtensions.m	
SKPSMTPMessage.m	

Figure 19. Code coverage

6.3 Integration testing

As mentioned above, this application should be used along with an OBD-II W-Fi scanner connected to the OBD-II port on the car. So, integration testing is performed while the tester is driving. As previously mentioned, this application can read real-time data from a car. Reading real-time data is the core part of this project and it will involve most modules of this application. So, integration testing will focus on data comparison between data from the application and data from the car. Table 10 uses those recorded data to compute the margin of error.

Driving distance recorded by the app (mile)	**Driving distance recorded by the car (mile)**	**Margin of error (%)**
0.00	0.0	0%
0.20	0.2	0%
0.82	0.8	2.5%
1.42	1.4	1.4%
1.67	1.7	1.7%
1.91	1.9	0.5%
2.39	2.4	0.4%
3.34	3.3	1.2%
3.78	3.8	0.5%
4.22	4.2	0.4%
5.07	5.1	0.5%
6.54	6.5	0.6%
11.56	12.0	0.3%
25.44	25.5	0.2%
28.64	28.7	0.2%
	Geometric Mean:	0.9%

Table 10. Data comparison and margin of error

Figure 20 shows those data in coordinate system. Green points are consisted of those two kinds of data. X value is data recorded by the app and Y value represented data recorded by the car. The orange line represented Y=X.

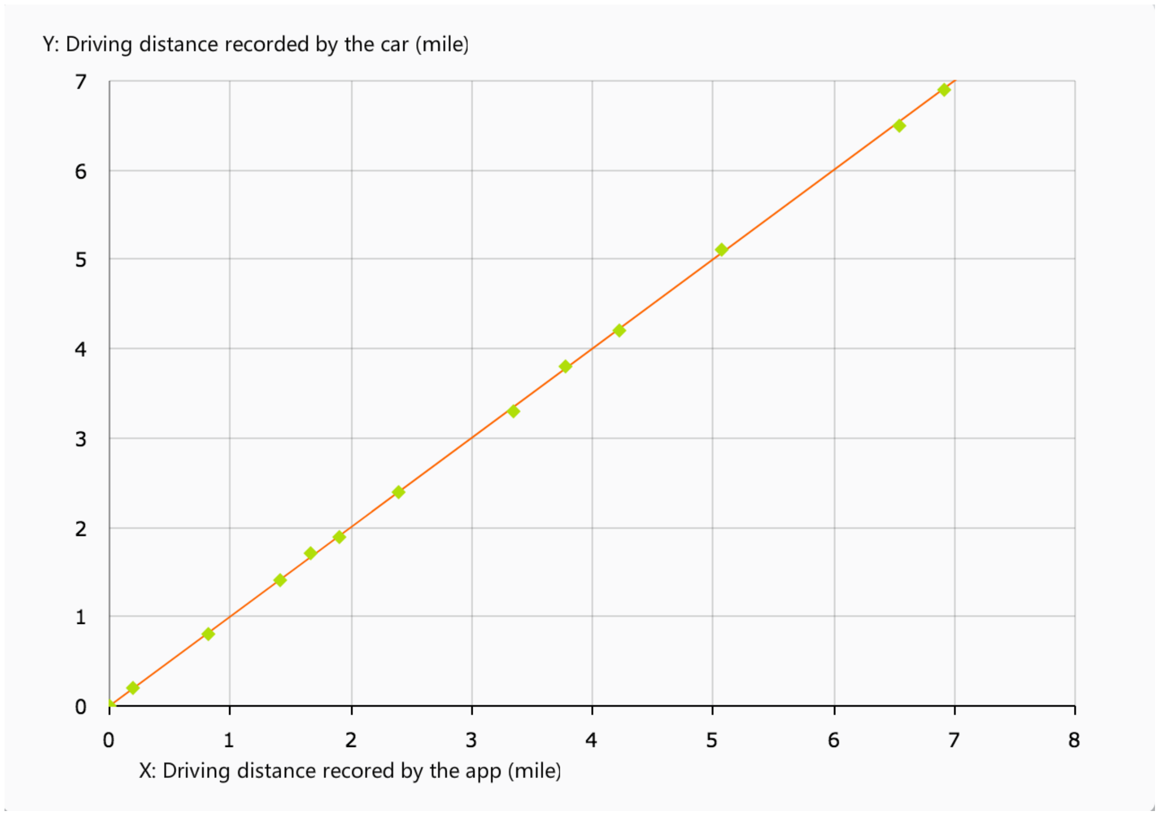


Figure 20. data comparison in coordinate system

6.4 System testing

System testing will focus on if the application satisfied users' requirements. Table 11 gives comparisons between required functions from users and functions in this application.

A user want to do (during a journey)	**Application**	**Status**
A user want to use an application to connect to a car	The app can help to build connection between the user and the car	Completed
A user want to get how long he had been driving	The app can provide the user with driving time	Completed
A user want to get specific speed	The app can provide the user with specific speed value	Completed

A user want to get specific RPM	The app can provide the user with specific RPM value	Completed
A user want to get average speed	The app can provide the user with average speed value	Completed
A user want to get fuel consumption	The app can provide the user with fuel consumption	Completed
A user want to get fuel cost	The app can provide the user with fuel cost	Completed
A user want to get engine coolant temperature	The app can provide the user with engine coolant temperature	Completed
A user want to get control module voltage	The app can provide the user with control module voltage	Completed
A user want to get sharp acceleration times	The app can provide the user with sharp acceleration times	Completed
A user want to get sharp braking times	The app can provide the user with sharp braking times	Completed
A user want to save related information of a journey	The app allows the user to register for an account to save related information of a journey	Completed
A user want to search journey information of specific date	The app allows the user to search journey information of that date	Completed
A user want to get diagnostic information of the car	The app can provide users with the car’s diagnostic information	Completed
A user want to set an alert for over speed	The app allows the user to set up speed limit	Completed
A user want to set an alert for engine being overheated	The app allows the user to set up engine coolant temperature limit	Completed
A user want to set an alert for control module voltage being abnormal	The app allows the user to set up control module voltage limit	Completed
A user want to change measurement of unit	The app allows the user to change measurement of unit	Completed

Table 11. Achievements of requirements

7. Conclusion and Future Work

7.1 Conclusions

The main achievement of this project is that it can help drivers identify the effect certain configuration changes have on their cars' operations by recording driving behaviors from one route to another route. We show the process of analyzing user requirements, designing system architecture, constructing the whole system, implementation and testing final product which will be used by end users.

- Chapter Two shows how to analyze users' requirements from different perspective. Functions that can be performed by users are described in detail in functional requirements. Non-functional requirements are related to system qualities.
- Chapter Three describes both goals and constrains of system architecture design. Then it gives an overview of what the application will look like when it is completed. It also pointed out some key issues and its solutions.
- Chapter Four presents detailed system design including database table design and use of UML diagrams to analyze user scenarios, relationship between classes and interactions between each component in the system.
- Chapter Five shows implementation of main functions. It uses advantages of pseudocode to make logic of each process clear.
- Chapter Six presents testing strategies in testing. It also shows the process of testing and testing data included in each kind of testing.

Although this product has been tested for many times, there are some defects in it. User interfaces does not look very nice and does not have any appropriate message while loading data from server database. The screen does not look very fluent while logging out. Users can not select their own vehicles and find out some manufacture-defined fault codes.

7.2 Future Work

As mentioned above, there exists some weaknesses in the final product. The first thing of future work is to overcome those shortcomings. In addition, user interface can be resigned and colored to give better user experience. More data can be worked out to provide more vehicle information for users. For example, giving total idle time during a route. Because this product only provides universal diagnostic information, some manufacturer-specific codes cannot be recognized. The database can gather more information for both universal diagnostic codes and manufacturer-specific codes. It should also classify those codes. Users should have choices to select model type of their vehicles and then database can provide corresponding diagnostic codes. In a word, this application still has lots of work to do to become a real product.

8. Bibliography

[1] KitoSDev-CMS, “OBD-II ELM327” [online]. Available: http://obd2-elm327.es/

[2] United States Environmental Protection Agency, “On-Board Diagnostic (OBD) Regulations and Requirements: Questions and Answer”, Dec 2003.

[3] Freelancer Technology Pty Limited, “Create an Outline sketch for a car as per given example” [online], 2017. Available: https://www.freelancer.com/contest/Create-an-Outline-sketch-for-a-car-as-per-given-example-195238.html

[4] Electric Auto Association, “Escape PHEV TechInfo - PIDs” [online], Dec 11, 2013. Available: http://www.eaa-phev.org/wiki/Escape_PHEV_TechInfo#PIDs

[5] AA1Car Auto Diagnosis Repair Help, “Trouble Codes” [online], 2008. Available: http://www.trouble-codes.com/

[6] Alex (Im) E, “Complete List of OBD Codes: Generic OBD2 (OBDII) & Manufacturer” [online], Jan 29, 2013. Available: http://www.totalcardiagnostics.com/support/Knowledgebase/Article/View/21/0/genericmanufacturer-obd2-codes-and-their-meanings

[7] Conexant Proprietary Information, “Hayes AT Command Reference Manual” [online], Apr 5, 2001. Available: http://www.zoomtel.com/documentation/dial_up/100498D.pdf

[8] Elm Electronics Inc., “ELM327 AT Commands” [online], October 2010. Available: https://www.sparkfun.com/datasheets/Widgets/ELM327_AT_Commands.pdf

[9] Mao Zheng, “CS-741 Software Engineering”, Sep 2015.

[10] Dezheng Wang, “Functional requirements specification”, Dec 2016.

[11] Microsoft Patterns & Practices Team, *Microsoft® Application Architecture Guide, 2nd Edition,* Microsoft Press, Nov 2009.

[12] Grady Booch, *Object-Oriented Analysis and Design with Applications (3rd Edition),* Addison Wesley Longman Publishing Co., Inc. Redwood City, CA, USA, 2004.

[13] Ahmad Shuja, Jochen Krebs, *IBM RUP Reference and Certification Guide*, IBM Press, Dec 29, 2007.

[14] Scott W. Ambler, Ronin International, *The Elements of UML™ 2.0 Style*, May 2005.

[15] Lucid Software Inc., “Database Structure and Design Tutorial” [online], 2017.

Available: https://www.lucidchart.com/pages/database-diagram/database-design

[16] Merriam-Webster, Incorporated, “Differential Calculus” [online].

Available: https://www.merriam-webster.com/dictionary/differential%20calculus

[17] Noah Stupak, Nicholas DiFonzo *, Andrew J. Younge, Christopher Homan, “SOCIALSENSE: Graphical user interface design considerations for social network experiment software”, Computers in Human Behavior 26(3):365-370, May 2010.

[18] Kasi Periyasamy, “CS-551 User Interface Design”, Sep 2015.

Appendix

The following functional requirements are referenced in body of the thesis.

Index: Access.2

Name: Login

Purpose: To login into Riding Mechanic.

Input parameters: Username, password

Action: Ensure that username exists in the system.

Ensure that username and password match with the corresponding entries stored in database.

If they match, jump to main screen.

Output parameters: None Exceptions: Username does not exist.

Username and Password don't match.

Remarks: Checking the format of inputs before validating them and input validation.

Cross-references: None

Index: Access.3

Name: Logout

Purpose: To logout from Riding Mechanic.

Input parameters: None

Action: Ensure that a user has already logged in.

Ensure that the user does not have access to the system.

Terminate all pending activities.

Output parameters: None

Exceptions: Logout action takes long time.

Remarks: None

Cross-references: Access.2

Index: Access.4

Name: Forget

Purpose: To reset password.

Input parameters: Username

Action: Ensure that a user has that username associated with an account
Ensue that that username is in email format

Output parameters: None

Exceptions: Username doesn’t exist Username is not in email format

Remarks: None

Cross-references: None

Index: Get.2

Name: Get driving information

Purpose: To get real-time data while driving

Input parameters: None

Action: Ensure that Riding Mechanic has already connected to a car
Ensure that Riding Mechanic has already got supported PIDs.
Keep sending PIDs to the car, process return data and show it to users.

Output parameters: Real-time information of the car

Exceptions: Riding Mechanic hasn’t connected to a car.
Riding Mechanic hasn’t got supported PIDs.

Remarks: None.

Cross-references: Get.1

Index: Search.1

Name: Choose specific date

Purpose: To choose a specific date for searching

Input parameters: None

Action: Ensure that users have already logged in.

Output parameters: None

Exceptions: Users haven’t logged in.

Remarks: None

Cross-references: Access.2

Index: Search.2

Name: Search driving information

Purpose: To search driving information for a specific date.

Input parameters: None

Action: Ensure that users have already logged in.
Ensure that users have already chosen a specific date.

Output parameters: Driving Information

Exceptions: Users haven’t logged in.
Users haven’t chosen a specific date

Remarks: None

Cross-references: Search.1

Index: Check.1

Name: Check a car

Purpose: To check a car to find malfunctions Input parameters: None

Action: Ensure that users have already logged in.

Ensure that Riding Mechanic has already connected to a car.

Start checking and show results.

Output parameters: Malfunctions Exceptions: Users haven't logged in.

Riding Mechanic hasn't connected to the car.

Remarks: Give hint if no malfunctions occur.

Cross-references: Connection.1

Index: Alert.1

Name: Set up alert

Purpose: To close or open an alert for Riding Mechanic

Input parameters: None

Action: Ensure that users have already logged in.

Open or close an alert.

Output parameters: None

Exceptions: Users haven't logged in.

Remarks: None

Cross-references: Access.2

Index: Set.1

Name: Set up fuel price

Purpose: To set up fuel price to calculate fuel cost

Input parameters: Fuel price

Action: Ensure that users have already logged in.

Enter valid fuel price.

Output parameters: None

Exceptions: Users haven’t logged in. Fuel price is invalid.

Remarks: None

Cross-references: Access.2

Index: Set.2

Name: Set up unit of measurement

Purpose: To set up unit of measurement

Input parameters: None

Action: Ensure that users have already logged in.

Select unit of measurement.

Output parameters: None

Exceptions: Users haven’t logged in.

Remarks: None

Cross-references: Access.2