

DOKUZ EYLÜL UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

**REAL-TIME VEHICLE MONITORING AND ON-
BOARD DIAGNOSTIC SYSTEM**



by
Emin VİLGENOĞLU

November, 2019

İZMİR

REAL-TIME VEHICLE MONITORING AND ON-BOARD DIAGNOSTIC SYSTEM

**A Thesis Submitted to the
Graduate School of Natural and Applied Sciences of Dokuz Eylül University
In Partial Fulfillment of the Requirements for the Degree of Master of Science
in Electrical and Electronics Engineering**

**by
Emin VİLGENOĞLU**

**November, 2019
İZMİR**

M.Sc THESIS EXAMINATION RESULT FORM

We have read the thesis entitled “**REAL-TIME VEHICLE MONITORING AND ON-BOARD DIAGNOSTIC SYSTEM**” completed by **EMİN VİLGENOĞLU** under supervision of **ASST. PROF. DR. REYAT YILMAZ** and we certify that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.


Asst. Prof. Dr. Reyat YILMAZ

Supervisor


(Jury Member)

Prof. Dr. Emin ANARIM


(Jury Member)

Asst. Prof. Dr. Yavuz SENOL


Prof. Dr. Kadriye ERTEKİN

Director
Graduate School of Natural and Applied Sciences

ACKNOWLEDGEMENT

I would first like to express my gratitude to my supervisor of this project, Asst. Prof. Dr. Reyat Yılmaz for his valuable guidance and advice. It would not have been possible to write this thesis without his support. He inspired me greatly to work in this project and encouraged me to overcome the problems I encountered during this period.

I would also like to thank my fiancée Simge Dizdar for her assistance and patience. She encouraged me all the time and helped me whenever she thought I needed it. And an honorable mention goes to my best friends Vural Avşar and Fatih Mehmet Dağ for their understanding and contribution in completing this thesis.

Above all, I would like to thank Dokuz Eylül University for providing me with a good environment and facilities to complete this thesis.

Finally, an honorable mention goes to my parents and siblings for their support during this period as though it were my whole life.

Emin VİLGENOĞLU

REAL-TIME VEHICLE MONITORING AND ON-BOARD DIAGNOSTIC SYSTEM

ABSTRACT

Despite the existence of the vehicles' communication standards and protocols with the external world, the importance of the data exchange in vehicles is having a lot of meaning in recent years with the digitalization of the world and getting connected of the things and people. Besides, with the evolution of Long Term Evolution (LTE) and 5G mobile communication systems, Internet of Things (IoT) concept and cloud technologies have been growing increasingly since the data exchange and connectivity require reliable and fast communication systems. All those consequences and developing vehicle technologies have inspired to do work of this thesis.

In this paper, vehicles' On-Board Diagnostics (OBD) system has been researched and a new concept developed to obtain and use the data from the vehicles basically as for the security, controllability, and energy consumption topics. Within the scope, the following subjects are studied: (i) regardless of the communication protocol of the vehicles a generic embedded system is designed to communicate and adapt to the vehicles OBD interface, (ii) a cloud-based platform is used to store and analyze the vehicle data so as to make it reachable and connected from anywhere, (iii) an IoT concept is implemented by connecting the users and drivers to the cloud through services and Application Interfaces (API). The mentioned End-to-End (E2E) system allows us to analyze the obtained data and serve it to the end-user applications.

Consequently, a solution, which covers all the mentioned concepts and technologies, is designed and developed. The solution includes vehicle data exchange module, location and communication module, and cloud system applications module. It's observed that the developed system provides the drivers and end-users to monitor the vehicle's energy consumption rates and let them conclude efficient driving behavior, to be notified about diagnostic issues in advance, and to monitor the location.

Keywords: OBD-II, Vehicle Diagnostic, Vehicle/Car Monitoring System, Cloud, IoT

GERÇEK ZAMANLI ARAÇ İZLEME VE ARIZA TESPİT SİSTEMİ

ÖZ

Araçlardaki iletişim standart ve protokollerinin varlığına rağmen, araçlarda veri alışverişinin önemi son yıllarda dünyanın dijitalleşmesi, nesnelerin ve insanların birbirlerine bağlanmasıyla birlikte daha da anlam kazanıyor. Ayrıca LTE ve 5G mobil iletişim sistemlerinin gelişmesiyle beraber, Nesnelerin İnterneti (IoT) ve bulut teknolojileri hızlı bir şekilde büyümektedir çünkü veri alışverişi ve bağlantı güvenilir ve hızlı iletişim sistemlerine ihtiyaç duymaktadır. Tüm bu sebepler ve gelişen araç teknolojisi bu tez çalışmasını yapmak için ilham kaynağı oldu.

Bu çalışmada araçların OBD sistemi araştırıldı ve temel olarak güvenlik, kontrol edilebilirlik ve enerji tüketim konuları için araçlardan veri toplama ve kullanma konsepti geliştirildi. Bu kapsamda şu konular çalışıldı: (i) araçların iletişim protokollerine bağlı olmaksızın, OBD ara yüzüyle uyumlu ve haberleşebilecek genel bir gömülü sistem tasarlandı, (ii) her yerden erişilebilirlik ve bağlantı adına araç verilerinin saklanması ve analizi için bulut tabanlı bir platform kullanıldı, (iii) kullanıcı ve sürücüler servisler ve uygulama ara yüzleri ile bulut sistemine bağlayarak bir nesnelerin interneti konsepti uygulandı. Bahsi geçen uçtan uca sistem elde edilen verileri analiz etmeye ve son kullanıcı uygulamalarına sunmaya yardımcı olmaktadır.

Sonuç olarak bahsedilen tüm kavram ve teknolojileri kapsayan bir çözüm tasarlandı ve geliştirildi. Bu çözüm araç veri alışverişi, konum, haberleşme ve bulut sistemi uygulamaları modüllerini içermektedir. Geliştirilen sistemin sürücü ve son kullanıcılara aracın enerji tüketim oranlarını sunduğu ve verimli sürüş davranışlarını tespit etmelerini sağladığı, arızalarla ilgili önceden bilgilendirilmelerine ve aracın konumunu izlemelerine olanak sağladığı gözlemlenmiştir.

Anahtar Kelimeler: OBD-II, Araç Arıza Tespiti, Araç İzleme ve Takip Sistemi, Bulut, Nesnelerin İnterneti

CONTENTS	Page
M.Sc THESIS EXAMINATION RESULT FORM.....	ii
ACKNOWLEDGEMENT	iii
ABSTRACT.....	iv
ÖZ	v
LIST OF FIGURES	viii
LIST OF TABLES.....	ix
 CHAPTER ONE - INTRODUCTION	 1
1.1 Objectives	2
1.2 Thesis Organization.....	3
 CHAPTER TWO - LITERATURE REVIEW	 4
2.1 On-Board Diagnostic Background	4
2.2 Mobile Communication Systems Background.....	6
2.3 IoT and Cloud Systems Background.....	7
 CHAPTER THREE - METHODOLOGY AND DESIGN.....	 10
3.1 System Specification	12
3.2 On-Board Embedded System Design.....	13
3.2.1 OBD-II UART Module.....	14
3.2.2 SIM808 Module.....	18
3.2.3 Embedded System Software Architecture	19
3.3 Back-End Connected Services Design	20
3.3.1 GPS Location Service	21
3.3.2 Vehicle Information Service	22
3.3.3 Trend Data Service	23
3.3.4 DTC Service	25
3.4 Cloud System Design	27

3.4.1 Elastic Cloud Computing Architecture	28
3.4.2 Database Design	32
3.5 End-User Application Design.....	34
CHAPTER FOUR - RESULTS	36
CHAPTER FIVE - CONCLUSION	42
REFERENCES.....	43



LIST OF FIGURES	Page
Figure 1.1 End-to-End (E2E) system design representation	2
Figure 3.1 Block diagram of End-to-End (E2E) system	13
Figure 3.2 Circuit diagram of the system	14
Figure 3.3 OBD Type-A connector pinout representation	15
Figure 3.4 Embedded system flow diagram	19
Figure 3.5 Vehicle identification number representation	23
Figure 3.6 DTC character representation	26
Figure 3.7 EC2 AMI properties	29
Figure 3.8 EC2 general instance properties	30
Figure 3.9 EC2 volume representation	31
Figure 3.10 EC2 inbound security group	32
Figure 3.11 MongoDB context and cluster	33
Figure 3.12 Vehicle DB collections	34
Figure 3.13 User application login page	35
Figure 3.14 User application main dashboard	35
Figure 4.1 OBD port connection	36
Figure 4.2 In-car embedded circuit connection	37
Figure 4.3 GPS location dashboard	39
Figure 4.4 Trend data dashboard	39
Figure 4.5 DTC dashboard	40
Figure 4.6 VIN dashboard	41

LIST OF TABLES	Page
Table 3.1 OBD Type-A connector pinout description	15
Table 3.2 OBD-II services	16
Table 3.3 Bitwise encoding representation	17
Table 3.4 Response data bytes representation	17
Table 3.5 SIM808 AT command definition	18
Table 4.1 Back-End services endpoint URLs	38



CHAPTER ONE

INTRODUCTION

Road transportation has evolved rapidly immediately after the invention of the Internal Combustion Engine (ICE). The first commercial ICE was developed by Etienne Lenoir around the year 1859 and Nikolaus Otto has created the first modern ICE in 1876 (Internal Combustion Engine, 2019).

In years, with the developing electronic and computer technologies, the vehicle and the car industry has adapted to the technology and made use of electronic devices in the vehicle systems. In this sense, Engine Control Unit (ECU), which is also called Engine Control Module (ECM), was first used by BMW in 1939 for its 801 14-cylinder aviation radial engine (Engine Control Unit, 2019).

Today, particularly the car industry is improving with the developing chip and sensor technology especially in terms of fuel efficiency, vehicle safety, and vehicle's electronic systems. As well as the improving interior radio, navigation, and entertainment systems, the adaptive cruise control and adaptive steering control systems are also some of the hot topics in the vehicle industry.

In the near future, the vehicle industry is going to be adapted to the IoT concept by means of connected vehicle terminology. In other words, it's expected to get all the vehicles connected and communicate with each other. Besides, smart parking (Hans et al., 2015) and adaptive traffic signal controlling (Jing et al., 2017) are also new topics in the connected vehicle area.

The proposed work in this thesis is aimed to provide real-time vehicle connectivity to the drivers in the aspects of monitoring and controlling the vehicle's location, trouble codes, service maintenance due, general vehicle information, and so on. Also, the proposed solution will provide to check and interpret vehicle's basic trend parameters such as speed, rpm, fuel tank level, oil temperature, energy consumption, and Malfunction Indicator Light (MIL) from remote.

1.1 Objectives

Advancing technology is leading to systems getting smart and therefore helping human life to get easier. In this sense, the development of vehicle systems is inevitable and day by day vehicles are getting smarter. Electric Vehicles (EV) and autonomous vehicles are the destination of today's automotive world.

The concept of this work has arisen from the idea of connecting the vehicles to cloud systems and creating an IoT based infrastructure in order to reach out vehicle's information remotely and monitor it from anywhere. As depicted in the following figure 1.1, the vehicles are aimed to connect to the cloud systems with the help of GSM technology. Hereby, the drivers will be able to reach out their vehicle information and to monitor its location and to know in advance about the diagnostic issues.

In this design, data connectivity has been achieved via GPRS feature of GSM mobile communication as a prototype, however, advanced mobile communication systems like 3G/4G/5G would be used so as to take advantage of wide bandwidth.

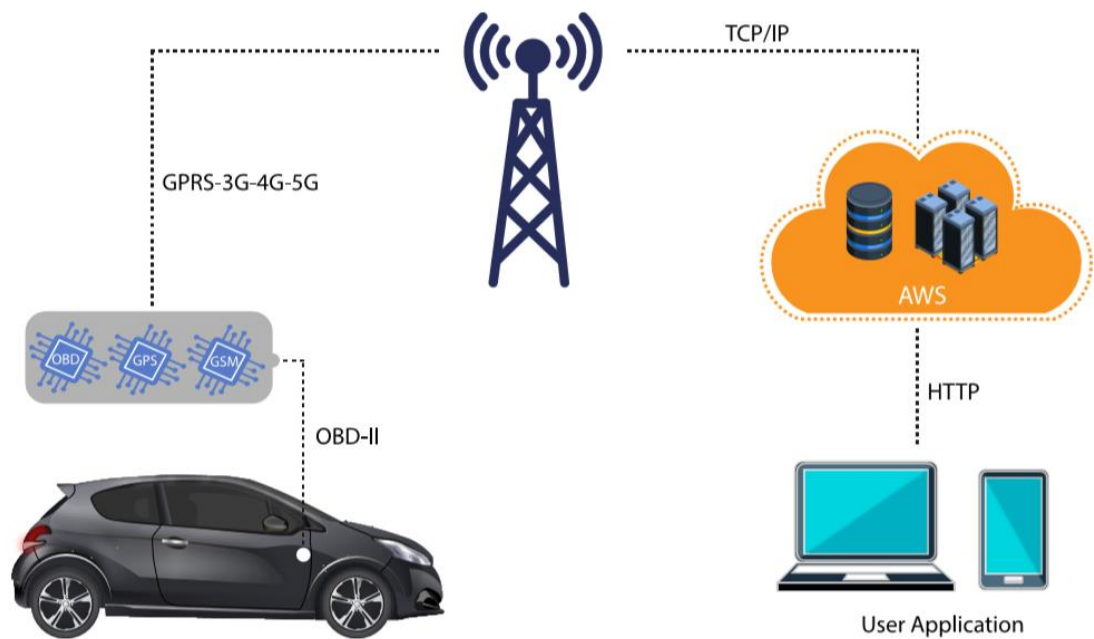


Figure 1.1 End-to-End (E2E) system design representation

1.2 Thesis Organization

This thesis consists of five chapters:

- **Chapter One, Introduction:** In the first chapter, the motivation behind this thesis is discussed and thesis objectives are defined.
- **Chapter Two, Literature Review:** In the second chapter, brief information is given about the project components as well as the background of the components is clarified based on the literature review.
- **Chapter Three, Methodology and Design:** In the third chapter, way of working and methodology is defined. Based on the specified solution, design and implementation are detailed.
- **Chapter Four, Results:** In the fourth chapter, results are evaluated and compared with similar designs.
- **Chapter Five, Conclusion:** In the fifth chapter, the design is summarized. Also, possible future enhancement is discussed.
- **References:** All applied resources including books, journals, digital files, etc. are cited in the references section.

CHAPTER TWO

LITERATURE REVIEW

In this section, the literature review of the used systems and main components will be discussed. The E2E system will be reviewed in the means of already proposed background studies until now.

2.1 On-Board Diagnostic Background

Modern vehicles today have an Engine Control Unit (ECU) that controls and manages the vehicles' subsystems. The vehicle manufacturers, in fact, develop the ECU in order to optimize the engine performance by collecting data from various subsystems (sensors) in the vehicle. The mentioned optimizations would be like fuel efficiency, reduced CO₂ emissions, ease of diagnostics, and so on (Sim et al., 2014). On the other hand, the most known and reliable communication protocol in-vehicle technology is Controller Area Network (CAN) bus protocol. The CAN Bus protocol allows the ECU and subsystems to communicate with each other without having a computer system.

The CAN Bus protocol was designed and developed by Robert Bosch in 1986 especially for automotive, however, it's used even in some industrial applications nowadays (Khorsravinia et al., 2017). In addition, the CAN Bus protocol, which has a very high level of security, was first commercially used as a bus system in 1991. The CAN Bus protocol is defined by 3 relevant standards that are ISO 11898 (CAN), ISO 15766 (Diagnostic on CAN), and ISO 15031 (Legislated OBD on CAN) (Khorsravinia et al., 2017).

Besides the existing electronic control system and communication protocol in the vehicles, there is a need for communicating with the vehicles from external computer systems. For the very purpose, the On-Boar Diagnostic (OBD) standard has been defined and developed basically to get the state of health information from the vehicle

subsystems. The OBD standard was first introduced in the 1980s and the amount of provided information has varied since then (Yun et al., 2011).

Throughout history, the OBD standard has been developed gradually and there have been several versions of it. The vehicle companies and manufacturers have contributed to the standard as well. For example, General Motors designed its own standard as Assembly Line Diagnostic Link (ALDL) in the late 1970s and early 1980s, as well as Toyota, has developed Multiplex OBD (M-OBD) as an alternative protocol that complies with OBD-II (On-board diagnostics, 2019).

In the United States the OBD standard has become a mandatory requirement for the light-duty vehicles that are 1996 model and newer (OBD-II PIDs, 2019). Basically, the standard can be analyzed in as the version and types of OBD-I, OBD-1.5, OBD-II, EOBD, and JOBD.

The OBD-I standard was regulated to encourage the vehicle manufacturers to control emission systems and it's known as 1991 and later California standard, not a USA Federal standard. On the other hand, OBD-1.5 corresponds to a pre and partial implementation of OBD-II (On-board diagnostics, 2019).

The OBD-II is an enhanced version of previous OBD standards and improved in terms of capability. The standard's connector pinout is defined by the Society of Automotive Engineers (SAE) J1962 (Yun et al., 2011). Also, the EOBD and JOBD refer to the equivalent OBD-II standard in Europe and Japan, respectively.

According to the regulations and standards, there are 3 mandated rules of OBD-II (Sim et al., 2014)

- Communication standard with the ECU
- The standard inquiry commands (Parameter Ids)
- The standard error codes (DTC)

There are 5 communication protocols for the OBD-II to communicate with vehicles' ECU that are listed as follows and all vehicle manufacturers should comply with at least one of them (Sim et al., 2014)

1. J1850 PWM
2. J1850 VPW
3. ISO 9141-2
4. ISO 14230 (KWP 2000)
5. ISO 15765-4-CAN

In this proposed work, an OBD-II Universal Asynchronous Receiver-Transmitter (UART) module is used to communicate with the vehicle's ECU system. Also, the module is capable of communicating with all these 5 types of communication protocols.

2.2 Mobile Communication Systems Background

Mobile communication has started with the first generation (1G) and followed by 2G, 3G, 4G, and 5G technologies. While 1G is an analog system that used for public voice service, a digital technology network infrastructure that also supports text messaging, is used in 2G. On the other hand, depending on the increasing demand for information via the internet data connectivity has provided and bandwidth expanded with 3G, 4G, and 5G mobile communication systems (Li et al., 2009).

In addition, the Global System for Mobile (GSM) network is a digital mobile network used by mobile phones in the world. In 2G GSM network, it's already possible to reach to the internet with General Packet Radio Services (GPRS). With this technology data packets can be transmitted and received over Transmission Control Protocol/Internet Protocol (TCP/IP) (El-Ata, 200).

As well as the mobile communication systems are used for mobile phones to let people communicate, the technology also is widely used for electronic devices to get

connectivity over the internet. For this purpose, the GSM modules are used in systems and the data connectivity is achieved by a Subscriber Identity Module (SIM) over operators GSM network.

2.3 IoT and Cloud Systems Background

The designed system is basically based on IoT and Cloud Systems infrastructure that makes the connectivity and provide the vehicle information observation from remote applications. In this sense, cloud computing and IoT background systems are studied as follows.

Cloud computing basically corresponds to the availability of remote computer resources by means of data storage and computing power without having to manage the resources directly (Cloud computing, 2019). The word of “Cloud Computing” was firstly born in 1996 in a document released by Compaq Inc. However, it got popularized by Amazon.com in 2006 with the release of Elastic Compute Cloud (Cloud computing, 2019).

In the consequent years, cloud computing has developed and the organizations started to adopt the system and transform their (Information Technology) IT infrastructures. Therefore, the organizations could simply purchase the resources like server, data storage systems online and instantiating a virtual image on the cloud, it has become more preferable and popular (Kirda, 2012).

Despite, there are a few big Cloud Computing service providers in the market, Amazon is one of the most popular and trusted provider because of the various aspects like computing power, security, prices, warranty of system uptime, redundancy, and so on (Kotas et al., 2018).

Amazon Web Services (AWS) is affiliated of Amazon that serves on-demand cloud computing platforms for the organizations, companies, individuals, etc. through the internet. The AWS technology can be thought as an implementation of server farms

all around the world and fees are charged based on the usage of hardware/OS/software/networking options chosen by the subscriber. Also, AWS operates from many global geographical regions including America, Europe, and Asia (Amazon Web Services, 2019).

In addition, the Cloud Computing Systems are scalable on-demand and therefore are cost-effective. Besides, it's classified as Public, Private, and Hybrid Cloud. This classification is also known as deployment models. The Cloud Computing is called Public Cloud when the services are shared over the network among the organizations, on the contrary, it's called Private Cloud if the infrastructure separated for a single organization. On the other hand, Hybrid Cloud is a combination of Public and Private Clouds and it is more cost-effective and scalable (Narula et al., 2015).

Apart from the Cloud Computing System classification, it is divided into types of service model. The Cloud Computing Providers offer 3 standard models that are Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). In the IaaS model, the principles of Cloud Computing is used. The provided services are related to the offered hardware including virtual servers and storage services as well. In the PaaS model, a development environment on the cloud is provided by the Cloud Computing Providers to the application developers. The provided computing platform basically involves operating system, database, program execution environment based on programming-language, and the webserver. In the SaaS model, the cloud providers manage the infrastructure and platforms. The developers or users get access to software and database (Cloud computing, 2019).

Furthermore, the AWS provides many services that all correspond to a specific requirement like IoT, Machine Learning, Data Analytics. Elastic Cloud Computing (EC2) is one of those services and EC2 is used in this work in order to meet the requirement of running back-end and front-end applications and store the data on the cloud. EC2 service is based on the IaaS model, and as per the definition of the IaaS model, a virtual machine is provided with an operating system. The hardware resources of the instances are chosen while creating the instances with EC2 service, however,

it's very flexible and scalable that means increasing and decreasing can be applied later on.

Since the internet has been invented and started to be used by people, it has made human life more comfortable and easier year by year. The first internet has begun in the 1960s as a link between a few computers (International Telecommunication Union Internet Reports, 2005). Then the internet usage was dominated with e-mail and file transfer operations. In the preceding years with the development of web and web browsing, the internet network has improved and the number of users has increased accordingly. Furthermore, with the invention of smart mobile phones and developed GSM, 3G, and LTE mobile network the internet usage has changed in a different way and there has been a dramatic increase in the number of internet users. All these processes are ended up with the concept of ubiquity (International Telecommunication Union Internet Reports, 2005).

In addition to connecting people to the internet, the thought of connecting things to the internet on behalf of people has resulted in the idea of Internet of Things (IoT). In other words, if this idea happened, the number of connections on the internet would be around hundreds of billion.

In this sense, regarding the idea of connecting the vehicles to the internet in this prototype work, the IoT concept is achieved. According to the definition of the IoT concept defined in different sources; it manages intelligently to identify, track, and monitor the components of the vehicles and manages the network by connecting things to the others based on the agreed protocol (Zhu et al., 2011).

To sum up, briefly, the AWS EC2 instances are used to run back-end services and front-end applications and to store obtained data as well. Also, the cloud connection of the vehicles is achieved through GPRS mobile communication and accordingly the IoT concept is implemented as a result.

CHAPTER THREE

METHODOLOGY AND DESIGN

Design of Real-Time Vehicle Monitoring and On-Board Diagnostic System can be divided into 4 main sections:

- On-Board Embedded System Design
- Back-end Connected Services Design
- Cloud Computing System Design
- End-User Application Design

Embedded System Design consists of OBD-II UART communication module, GPS location module, and GSM/GPRS communication module. In the OBD-II communication module vehicle's On-Board Diagnostic port is used to have a connection to the vehicles as specified in the introduction chapter. The OBD port may have different communication protocols such as CAN Bus, SAE J1850 PWM, ISO 9141-2, etc.

The communication protocol that a vehicle exposes to external systems varies based on the vehicle model, manufacturing year, and brand. The OBD-II UART module allows communicating with all the OBD protocols regardless of which one the vehicle supports. The OBD-II UART device is connected to the port in the vehicle with an OBD Diagnostic Connector (DLC). Therefore, it enables us to interpret the ECU by Attention (AT) commands and the supported communication protocol of the vehicle can be detected in this way. Then, regarding the protocol type, the communication is started and relevant PIDs are requested from the OBD port. The OBD Parameter IDs will be explained in detail in the methodology and design section. Besides, the OBD-II UART module lets us make communication (send command and receive data) over serial communication with a Recommended Standard 232 (RS232) interface.

On the other hand, for the GPS and GSM/GPRS part of the embedded design, a SIM808 module is used. The module contains GPS, GSM, and GPRS engines that all are AT command compatible. While the GPS engine is used to obtain the location info, the GPRS engine is used to send the location and OBD data to the cloud over Transmission Control Protocol / Internet Protocol (TCP/IP). The TCP/IP protocol protocols provide the internet connectivity through the GSM base station, however, the data sending operation is achieved by using Hyper Text Transfer Protocol (HTTP) POST technology.

Among all parts of the system, one of the main objectives is to connect the vehicles to cloud systems in order to obtain real-time data and have real-time connectivity. Therefore, the connectivity is done by designing and developing back-end services which work on a server in the cloud. Based on the available data obtained from the vehicles OBD port, a classification should be done in order for the system to make it useful and meaningful. In this sense, a number of back-end services have been designed depending on the data classification in terms of abstraction of each data model. Basically, the data abstraction has been thought in 4 main titles. The first one is the GPS Location data model that is handled in GPS service. The second, third, and fourth ones are Vehicle Information, Diagnostics Trouble Code, and Trend data models.

There are several cloud systems nowadays provided by Google, Microsoft, IBM, and Amazon, etc. All are serving similar cloud services but in different scalability and prices. In this project, Amazon Web Services (AWS) cloud system is used by Amazon.com Inc. Because it's one of the most popular and secure cloud systems and provides free of charge usage for a limited disk and memory size for educational purpose and non-profit organization or end-users. The designed cloud system is divided into three sections; one is the application server for the back-end services, the second one is the Next Generation Database system which is known as NoSQL database, and the third one is the application server for the front-end web application that is designed for the purpose of monitoring the vehicle and knowing the vehicle specific information, trouble codes, and trend data.

Although the designed embedded circuit that operates in the vehicle and interprets the vehicle ECU, the GPS module which provides the location info, the GSM/GPRS module which makes the connectivity with the TCP/IP protocol, the back-end system which handles the orchestration between the vehicle, and the cloud system are the main components of the design; serving the obtained data in a meaningful and readable format to the users and drivers with the help of a user interface application is inevitable. Thus, there will be a chance to render the interpreted data to show and take necessary actions accordingly.

The end-user applications would be various depending on the requirement and design specification such as desktop, mobile, and web applications, etc. After all, in this work, a web application has been designed to serve for the purpose of the end-user and driver usage. The web application consists of the following menus; general vehicle information menu, GPS location on a live map, vehicle trouble codes menu, and trend data menu.

3.1 System Specification

Before designing the real-time onboard diagnostics and monitoring system, defining specification of vehicle OBD, back-end services, cloud system, and end-user application is critical and an important step of the development. Specifications define how the vehicle computer will be interpreted and how it will work with all the components properly. Also, defining the constraints of the research and design process is needed in order to understand how much the system is comprehensive. Based on the vehicle specification, hardware and software systems are developed accordingly.

In this work, a Real-Time Vehicle Monitoring and On-Board Diagnostic System are aimed to be designed and implemented. Due to the thesis goals, the design will be limited to build a prototype in order to highlight project objectives. The following figure represents the block diagram of the entire system.

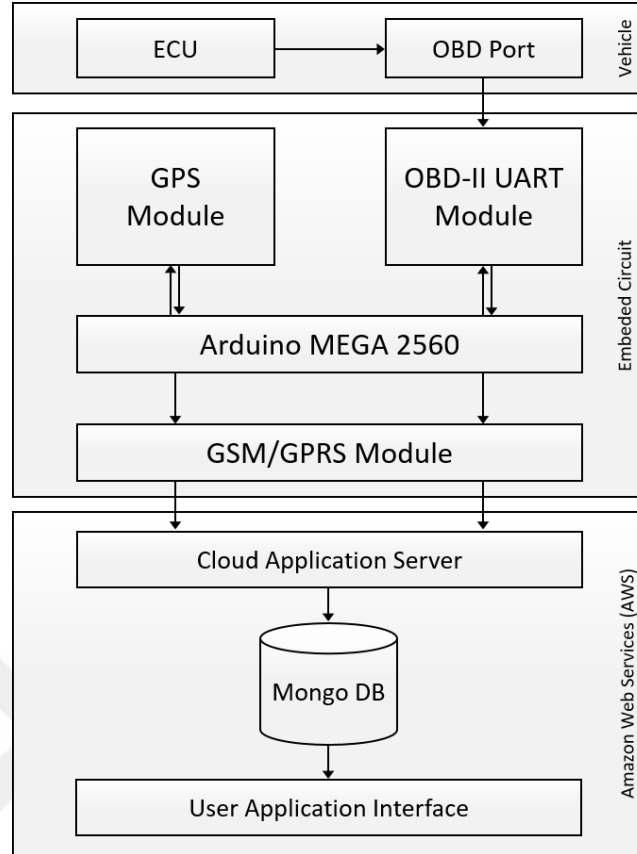


Figure 3.1 Block diagram of End-to-End (E2E) system

3.2 On-Board Embedded System Design

As specified in the introduction of the methodology section, the on-board embedded system consists of OBD-II UART, GPS, and GSM/GPRS modules. All these components are programmed and controlled by an Arduino Mega board which is built on ATmega 2560 microcontroller. The Arduino Mega board operates at 16MHz clock speed and has 8Kb Static Random Access Memory (SRAM) and 4Kb Electrically Erasable Programmable Read-Only Memory (EEPROM). The microcontroller board communicates with both the OBD-II UART and the SIM808 modules over serial communication pins on it. The following figure represents the circuit diagram of the embedded system.

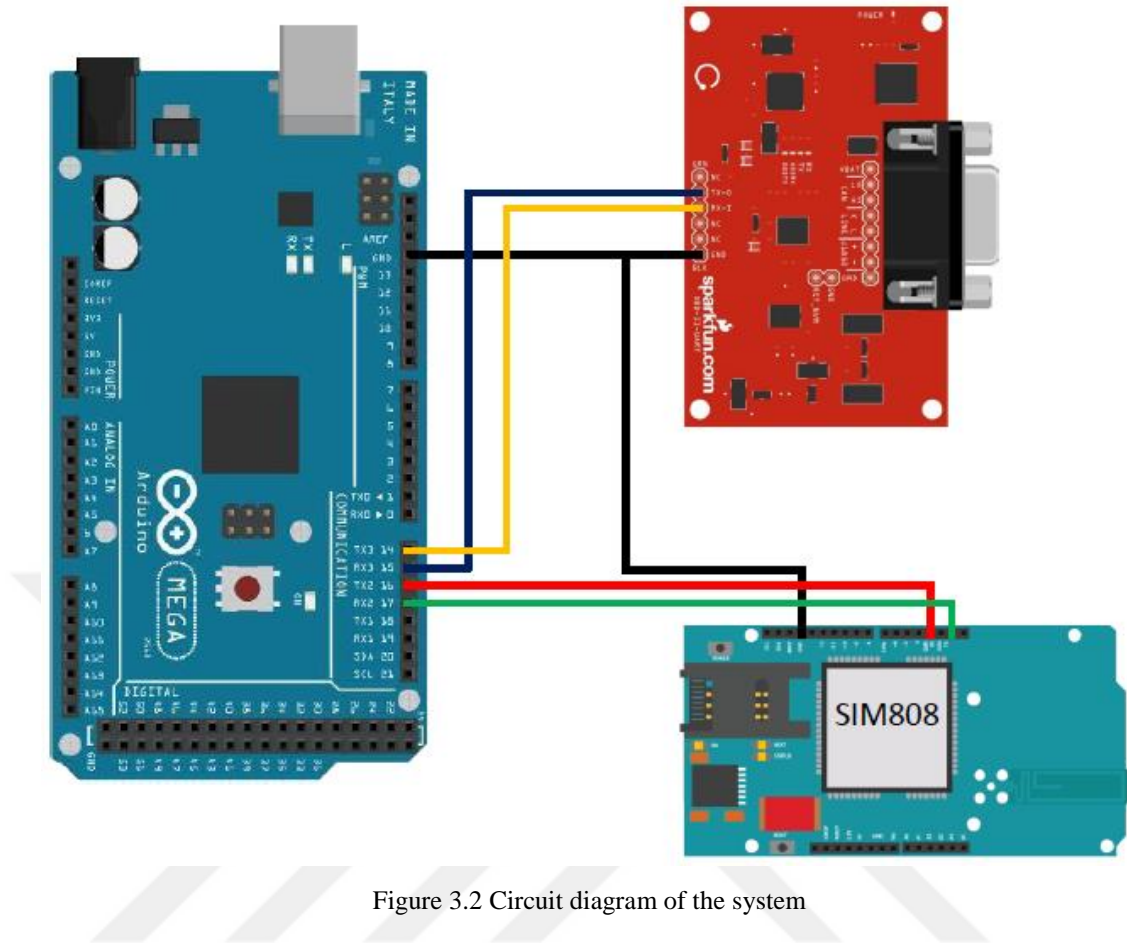


Figure 3.2 Circuit diagram of the system

3.2.1 OBD-II UART Module

OBD-II UART module is a board that is designed to connect and interpret the vehicle ECU. The module provides an OBD-to-DB9 connector. While the OBD end corresponds to Type-A socket as a connection type of OBD-II standard, the DB9 end corresponds to the RS232 connection. In terms of the available OBD connectors, there are two types as A and B. Whilst, Type A connector is used for the four-wheel vehicles and cars that use 12V supply voltage, whereas Type B connector is used for the vehicles that use 24V supply voltage. In this design, Type A connector has been used since the system has been tested in a 4-wheel passenger car. The Type-A connector's pinout schematic diagram is shown in the following figure.

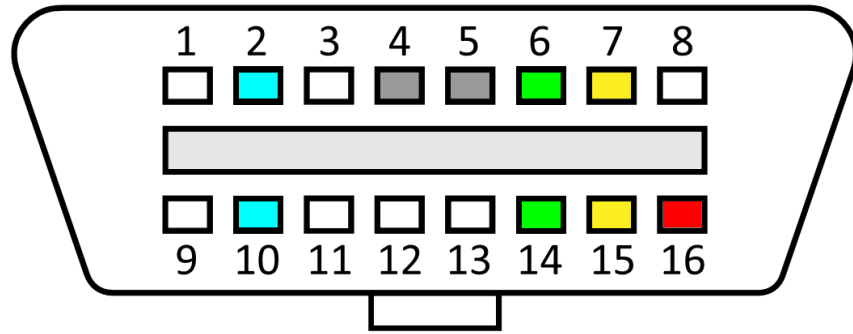


Figure 3.3 OBD Type-A connector pinout representation

The connector provides power supply pins as well as communication pins for the mentioned ECU communication protocols such as CAN Bus and SAE J1850 PWM. According to SAE J1962 standard, Type-A connector's pinout representation should be as given in the following table.

Table 3.1 OBD Type-A connector pinout description

1	Manufacturer discretion	9	Manufacturer discretion
2	Bus Positive Line of SAE J1850 PWM and VPW	10	Bus Negative Line of SAE J1850 PWM (not only VPW)
3	Manufacturer discretion	11	Manufacturer discretion
4	Chassis ground	12	Manufacturer discretion
5	Signal ground	13	Manufacturer discretion
6	CAN-High (ISO 15765-4 and SAE J2284)	14	CAN-Low (ISO 15765-4 and SAE J2284)
7	K-Line of ISO 9141-2 and ISO 14230-4	15	L-Line of ISO 9141-2 and ISO 14230-4
8	Manufacturer discretion	16	Battery voltage

On the OBD-II UART board, both the STN1110 and the MCP2551 chips populated. The MCP2551 provides to access only CAN protocol, whereas the STN1110 chip interprets any other protocols including CAN, ISO, J1850 Transceivers, and so on (OBD Solutions, 2018). Also, the STN1110 chip is compatible with ELM327 command set. The ELM327 is another popular OBD to RS232 Interpreter chip that owns a set of AT commands available for serial communication (Elm Electronics Inc., 2010).

3.2.1.1 OBD Parameter IDs (PID)

OBD protocols provide vehicle data such as Malfunction Indicator Light (MIL), Diagnostic Trouble Code (DTC), Inspection and Maintenance (I/M), Freeze Frames (FF), Vehicle Identification Number (VIN), and hundreds of real-time parameters by tapping into the OBD bus.

The requested data from vehicles are classified into 10 modes. These modes are called service modes and each contains its PIDs available. In the following table, the list of OBD services with its definition is represented.

Table 3.2 OBD-II services

Mode	Definition
0x01	Show current data
0x02	Show freeze frame data
0x03	Show stored Diagnostic Trouble Codes
0x04	Clear DTC and stored values
0x05	Test results and oxygen sensor monitoring (<i>no CAN only</i>)
0x06	Test results and other components/system monitoring (<i>CAN only</i>)
0x07	Show pending DTC (<i>detected during current or last driving cycle</i>)
0x08	Control operation of on-board component/system
0x09	Request vehicle information
0x0A	Permanent DTC (<i>Cleared DTC</i>)

Each service has its own PID codes defined according to the OBD protocol. The PID codes might vary based on the vehicle model and each manufacturer might have its own additional PIDs, whereas most of them are standard PIDs. As specified in the introduction and methodology sections, standard PIDs are defined by SAE J1979 (SAE International J1979_201202).

In addition, a request of 0x00 (PID 00) returns 4 bytes of data for each service. The response provides the information of which PIDs are supported in that service. In the following bitwise encoding table, an example response analyzed and the supported PIDs of a specific service obtained as an example representation (OBD-II PIDs, 2019).

Table 3.3 Bitwise encoding representation

Data Byte (HEX)	B				E				7				F			
Binary	1	0	1	1	1	1	1	0	0	1	1	1	1	1	1	1
PID number	1	2	3	4	5	6	7	8	9	0A	0B	0C	0D	0E	0F	10
Data Byte (HEX)	B				8				1				3			
Binary	1	0	1	1	1	0	0	0	0	0	0	1	0	0	1	1
PID number	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	20

3.2.1.2 OBD-II Response Data Structure

The ECU module of the vehicles process the incoming request, fetch the requested data and send it in the response. The response data byte could be in different lengths based on the request and the length of data, however, the first 2 bytes of the returned response have in common structure that the 1st byte combines the requested service with the response header, whereas the 2nd byte corresponds to the requested PID number. The following table shows the general request and response of the OBD.

Request: 0100 → Service 01 and PID 0x00 (for the supported PIDs in Service 01)

Table 3.4 Response data bytes representation

Response	41	00	BE	7F	B8	13
Meaning	0x40 + 0x01 0x40: default response header 0x01: Service 01		PID 00	Returned data byte		

On the other hand, some vehicles might have more than one ECU depending on the manufacturer's design. In that case one of the available ECUs processes the incoming request and returns the response. In such a situation, the first 2 bytes of the response

corresponds to the ECU name if the header bit of the ECU is set to true. Otherwise, the ECU name cannot be returned in the response.

3.2.2 SIM808 Module

SIM808 board is basically a GSM module that is designed in order for electronic and software applications to be used. The board is preferred to be used in this design since it has GPRS service availability and as well as a GPS and Bluetooth (BT) antenna mounted on it. BT module has not required to be used in this design but the GPS is required. The GSM/GPRS engine of the SIM808 board is quad-band and allows to work on 850MHz, Enhanced GSM (EGSM) 900MHz, Digital Cellular System (DCS) 1800MHz, and Personal Communication Service (PCS) 1900MHz (SIMCom, 2014). On the other hand, the module requires a SIM card in order to have connectivity through operators' base station as well as Internal Mobile Equipment Identity (IMEI) number is required to be defined to get connected on the GSM network. The GPS engine of the SIM808 board cannot be run by itself, whereas it's controlled by the GSM engine. Therefore, the GSM shouldn't be in SLEEP mode in order to reach out GPS. The GSM modes and operations are controlled by AT commands through serial port so as the same is applied for GPS modes and operations. A number of GSM/GPS AT commands are shown in the following table as an example.

Table 3.5 SIM808 AT command definition

Command	Description
AT+CGPSPWR	GPS power control
AT+CGPSRST	GPS mode reset
AT+CGPSSTATUS	Get current GPS status
AT+CGPSINF	Get current GPS location info
AT+CGATT	Attach or detach from GPRS service
AT+CIPSTART	Startup TCP connection
AT+CIPSEND	Send data through TCP connection
AT+CIPCLOSE	Close TCP connection

3.2.3 Embedded System Software Architecture

The Arduino Mega 2560 board is preferred in this design because its programming language is C/C++ based and it provides to implement data exchange communication protocols and handle byte data as for the communication as well as the availability of serial com port makes the testing much easier than most of the other boards. All the GPS, GSM/GPRS, and OBD-II UART modules are controlled by the Arduino microcontroller. The Arduino board has 4 TX/RX serial communication interface that allows up to 4 devices to be controlled and operated. In this sense, the microcontroller was programmed to handle the devices simultaneously. The flow diagram of the software design is shown in the figure below.

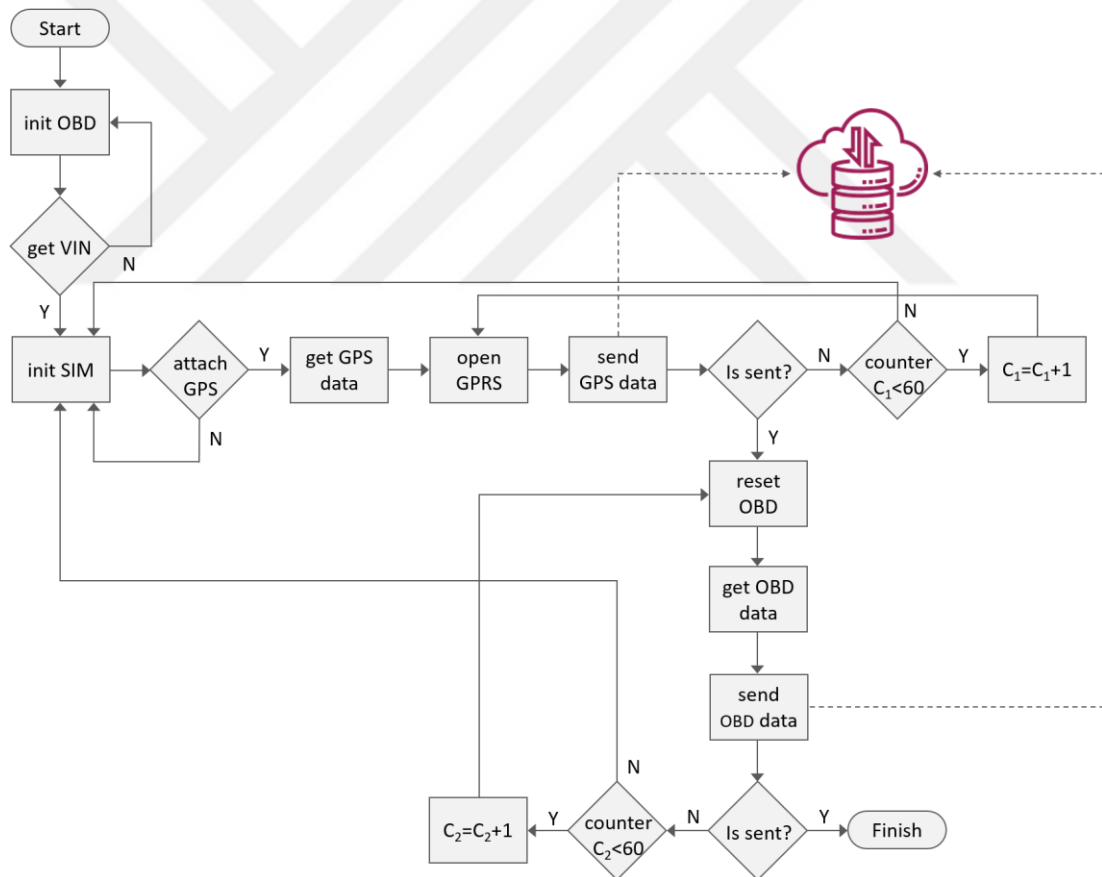


Figure 3.4 Embedded system flow diagram

As illustrated in the flow diagram, the program starts with opening the serial communication ports for both the SIM808 and OBD-II UART modules and in the

meanwhile communication baud rates are set to 9600bps. On the other hand, the Vehicle Identification Number (VIN) is requested from the OBD port in order to identify the vehicle. The vehicle identification should be done for each vehicle once the program is started. The vehicle identification number contains the information of vehicles such as model, year, and engine specification. That information is shown in the user application as the vehicle general information and specifies the vehicle uniqueness.

In the loop method of the program, first the SIM808 board initialized and GPS module is opened to capture the location. As soon as the location data is gathered, the GPRS mode is opened to send the location data. The data sending is done based on internet HTTP POST method over TCP/IP. The internet connection is achieved via mounted SIM card on the SIM808 module. The vehicle-specific data is also gathered from the OBD port and sent to the cloud as the same approach used for GPS data gathering and sending method.

3.3 Back-End Connected Services Design

In the programming world, web programming has been developed to connect devices and integrate applications on the internet. At this very purpose, web services and web applications (APIs) are used to meet those requirements. Most of the Object-Oriented Programming (OOP) languages support to develop web APIs and services. Java and C# are one of the most known OOP languages nowadays. On the other hand, JavaScript, which is a front-end script language for web programming, is used for web applications and pages are very popular because of its strength and capabilities. Despite the fact that the JavaScript was born for the need for client-side applications, server-side JavaScript frameworks are being developed recently. NodeJS has been created in order to be used in server-side application development.

The NodeJS framework has been built upon Google Chrome's V8 engine that actually consists of C++ libraries and classes. On the other hand, as well as NodeJS is an OOP language it works perfectly with JavaScript Object Notation (JSON) format.

JSON is a light-weight data exchange format contrary to Simple Object Access Protocol (SOAP). The SOAP protocol is based on Extensible Markup Language (XML) which is also designed for storing and transporting data.

In the back-end services design of this implementation, each data type has been classified and built in a specific JSON format that will be shown in detail in the next sections. Applying this concept provides to send and receive data in object form rather than building and parsing XML data which is a legacy and traditional system. In the XML data exchange systems generally, a relational database (DB) is used and each XML element corresponds to an attribute of an entity. Therefore, to build an XML data from a DB requires to collect each attribute and put in XML format and in the same way, to store XML data requires to parse each XML element and save it in the entity. However, in JSON format things are not like in XML, moreover, NoSQL database systems are designed to store JSON data in object form. In this way, sending and receiving JSON data doesn't require to build it up or parse it. These properties make JSON and JavaScript easier to implement and provide faster data exchange applications.

3.3.1 GPS Location Service

The GPS Location Service is designed to meet the requirement of receiving GPS location data in JSON format. In this sense, a GPS entity has been created in the MongoDB that stores the GPS location data. The location data is gathered from the GPS module in the embedded system and sent to this services in JSON format. The service is obliged to receive the GPS location data and save it in the GPS entity in JSON format. Following is an example data format of the collected GPS location data.

Latitude, longitude, heading, and speed are the parameters that correspond to the GPS location data gathered from the GPS module.

User id corresponds to the identification number of the data from which car it's sent. Vehicles' license plate is used as an identifier in the GPS data service.

Transaction id and date-time parameters are used to specify each transaction and the time when the data is received, respectively. On the other hand, the id parameter itself is generated by MongoDB for each record.

As seen the gathered GPS location data has a form of JSON.

```
{
  "latitude": {
    "lat": 38.23921,
    "deg": 38,
    "min": 14,
    "sec": 21.1496
  },
  "longitude": {
    "lon": 27.04872,
    "deg": 27,
    "min": 2,
    "sec": 55.40359
  },
  "_id": "5d0f61785d9d270a95d14e60",
  "user_id": "35DD1961",
  "transaction_id": "201962311233087",
  "date_time": "2019-6-23T11:23:30",
  "heading": 49.58,
  "speed": 1.852
}
```

3.3.2 Vehicle Information Service

The Vehicle Information service is basically designed to identify each vehicle in the cloud applications since the OBD provides a vehicle identification number (VIN), which is also known as chassis number, is unique for all the manufactured vehicles. VIN is a 17 character length identifier and it complies with two standards. One is the United States (US) standard Federal Motor Vehicle Safety Standard (FMVSS) 115 (NHTSA, 2011) and the other one is ISO 3779:2009 (ISO, 2019).

In the vehicle identification number, the following information could be found engine specification, sequential serial number, World Manufacturer Identifier (WMI), where the vehicle was produced, safety equipment, model year, some technical

specification, and features. An example VIN given below in the figure shows the meaning of each character based on the specified standards.

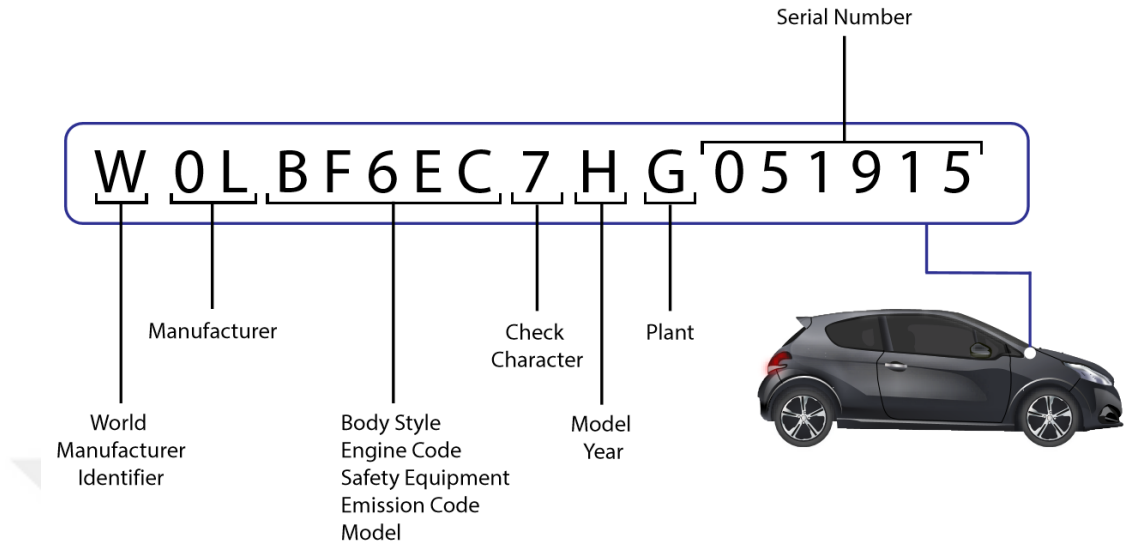


Figure 3.5 Vehicle identification number representation

The VIN is an entry point of the embedded software program as it's already shown in the flow diagram. In the setup method of the program, VIN is read from the OBD and sent to the cloud in the Vehicle Information Service in order to have identification of each vehicle in the back-end system. As well as VIN is a parameter of the Vehicle Information Service, fuel type is another key parameter. An example Vehicle Information Service JSON data form is shown as follows.

```
{
  "_id": "5d6af72b9ae8692f70b5eb1e",
  "user_id": "35DD1961",
  "transaction_id": "201908158000000",
  "date_time": "2019-7-01T11:12:20",
  "vin": "W0LBF6EC7HG051915",
  "fuel_type": "gasoline"
}
```

3.3.3 Trend Data Service

The Trend Data Service has been designed to obtain a vehicle's sensor data such as vehicle speed, engine rpm, fuel tank level, and so on. All the data are captured from

the OBD port and sent to the cloud via this service. Following is a representation of the service JSON data as an example.

```
{
  "_id": "5d6d65d5cb100f1cd0f8f291",
  "user_id": "35DD1961",
  "transaction_id": "201908158000000",
  "v_speed": 5,
  "e_rpm": 900,
  "e_oil_temp": 90,
  "e_fuel_rate": 5,
  "f_tank_lvl": 75,
  "f_pressure": 1.6,
  "rt_s_estrt": 3600,
}
```

As seen, the service has 7 OBD parameters read from the vehicle. The responses from the OBD are always in hexadecimal format. The integer correspondence of the hex bytes is calculated to obtain the desired values. All the parameters and their calculation are explained in detail as follows;

- Vehicle speed: it's a real-time obtained data and the OBD service 1 provides the PID "0D" to request it. The OBD returns 1-byte data (A) in return and it gives the vehicle speed in km/h.

$$v_speed = A \text{ (km/h)} \quad (2.1)$$

- Engine revolutions per minute (rpm): it's also a real-time obtained data within the OBD service 1 PID "0C". The response is 2-byte data (A and B) and the bytes gives the engine rpm with the following formula.

$$e_rpm = \frac{256A + B}{4} \text{ (rpm)} \quad (2.2)$$

- Engine oil temperature: The oil temperature is provided by OBD service 1 PID "5C". In response, 1-byte data (A) is returned and the result is calculated with the below formula.

$$e_oil_temp = A - 40 \text{ (°C)} \quad (2.3)$$

- Engine fuel rate (L/h): Engine fuel rate is a data that shows the engine fuel efficiency in liter per hour. This data can be obtained from both the OBD service 1 and 2 with PID “5E”. The response is 2-byte data as A and B the calculation is shown in the following formula.

$$e_fuel_rate = \frac{256A + B}{20} (L/h) \quad (2.4)$$

- Fuel tank level (%): the fuel tank level shows the percentage of the remaining fuel in the tank. It's provided by OBD service 1 PID “2F”. The response is 1-byte data (A) and calculated as below.

$$f_tank_lvl = \frac{100A}{255} (\%) \quad (2.5)$$

- Fuel pressure (kPa): Fuel pressure is the value that shows the flowing fuel's pressure through the system. Its calculation is shown below based on the returned 1-byte data (A) in response.

$$f_pressure = 3A \text{ (kPa)} \quad (2.6)$$

- Run time since engine start (seconds): This is a parameter shows the engine run time in seconds. The ECU also provides to get the run time since engine start with the following formula.

$$rt_s_estart = 256A + B \text{ (sec)} \quad (2.7)$$

3.3.4 DTC Service

The Diagnostic Trouble Codes (DTC) service has been developed in order to receive and save the vehicle trouble codes in the cloud. Thus, there will be a chance to keep the backlog of the trouble codes as well as the drivers or end-users will be notified with the existing issues about their vehicle.

The OBD in the vehicles is capable of providing almost all troubles with a specified code. However, it's known that all the trouble codes are not standard and there might

be manufacturer based trouble codes. Having this truth at hand makes us manufacturer dependent, however, the standard trouble codes are accessible according to SAEJ2012 standard (SAE International J2012_200204, 2019). In this sense, based on the specified structure trouble codes is divided into 4 modes which are Body (B), Chassis (C), Power Train (P), and Network (U). In terms of this separation, vehicles provide the trouble codes specifying in which part the problem exists.

On the other hand, the trouble codes are made of 5 characters. The first character specifies that where the problem exists like in the Body, Chassis, Power Train, or Network. The second character defines whether it is a generic OBD fault code or manufacturer specific fault code (0 – Generic OBD fault code, 1 – Manufacturer-specific fault code). Finally, the last 3 characters define the specific problem. The DTC 5 digit fault code representation and the meanings are shown in the following figure in depth.

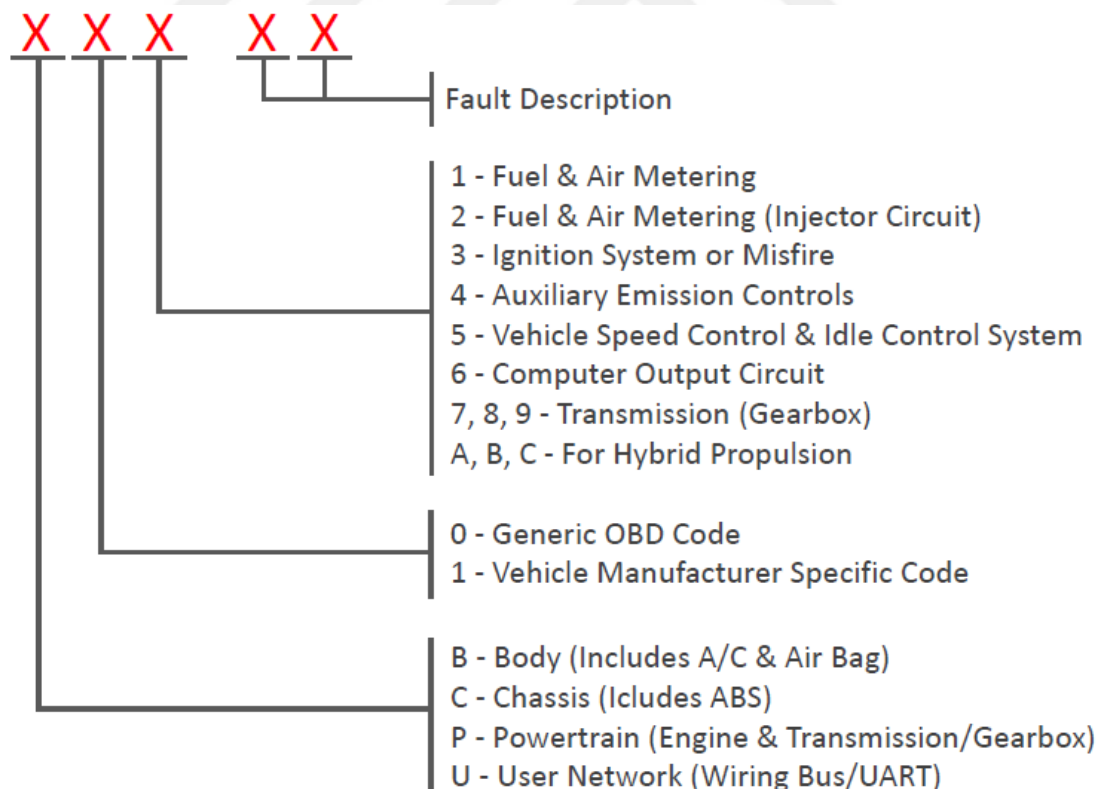


Figure 3.6 DTC character representation

The designed DTC service has a form of these trouble codes accordingly. Following is an example data format of the collected DTC data. As seen the gathered DTC data has a form of JSON as well.

```
{
  "_id": "5d6ac1bb8da80d3fb4e4d577",
  "user_id": "35DD1961",
  "transaction_id": "201908157000000",
  "date_time": "2019-7-12T09:13:38",
  "body": "B1200",
  "chassis": "C1091",
  "power_train": "P1100",
  "network": "U1000",
}
```

Body, Chassis, Power Train, and Network parameters in the JSON form correspond to the fault codes of 4 different types. For each transaction, the service is capable of receiving the 4 trouble codes if exists. Besides, the id parameter is generated by MongoDB as a primary key, the user_id parameter is the identifier of the vehicle, and the transaction id and date-time parameters are combined to discriminate the transaction itself and the time of the receiving the data respectively.

3.4 Cloud System Design

Cloud systems, which are indeed virtual computer systems, are being used nowadays especially for data storage and computation purposes. One of the main advantages of using cloud systems is to avoid the burden of hardware management and operation. In addition to computing and data storage options, the provided cloud systems are already capable of serving the networking, database, analytics, media, mobile, machine learning, IoT, AR & VR, blockchain, and so on.

Cloud computing systems can be classification under two subjects that are the basis of location and basis of services. On the basis of location, Private, Public, and Hybrid Clouds are classified. The Private Cloud is allocated to particular organizations with higher security and higher cost as well. In the Public Cloud, computing foundation is shared between organizations and companies as well as it's located at the vendor's premises. On the other hand, Hybrid Cloud is cost-effective and more scalable and it

is a combination of Public and Private clouds. On the basis of provided services, the classification is divided into 3 types that are Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). In the concept of IaaS principles of computing and services are used related to offered hardware. Besides, while the PaaS is a development platform on the cloud, the SaaS is a complete offered software service on the cloud (Narula et al., 2015).

As specified in the previous sections, AWS has preferred to be used in this design. To meet the requirement of the design, EC2 and Atlas MongoDB services are designed to be used. While EC2 allows hosting the NodeJS back-end services and end-user application, Atlas MongoDB is used for data storage.

3.4.1 Elastic Cloud Computing Architecture

Elastic cloud computing can be thought of as a number of virtual computers hosted in the AWS cloud. The meaning of elastic refers to the flexibility and scalability of the virtual machine. In addition, based on the requirement and the complexity of the system that is going to be designed, as many instances as can be instantiated in EC2 virtual machines. Also, AWS sets cost per instance use, so if an instance is stopped you stop paying for it. On the other hand, hardware optimization can be adjusted on-demand. For example, if high performance is required, resources such as CPU, memory, storage can be increased. Moreover, AWS EC2 provides instances optimized CPU, memory, storage, and GPU processing to enable right price-performance combination for the workload of the system.

As security is one of the most considered concerns of the cloud systems, AWS EC2 provides a Virtual Private Cloud (VPC) that has a logically isolated network can be controlled. In this design, VPC has been preferred so as to meet the security requirement of vehicle data.

In order to create an EC2 virtual machine, an AWS account should be created then the following few steps should be configured to get a running instance. The first step

is to select an Amazon Machine Image (AMI) for EC2. The AMI is basically a template for creating a new instance and it contains software information, operating system information, volume information, and so on. In this design, Amazon Linux AMI has been selected, since the back-end service developed in NodeJS and it can easily be hosted and maintained in a Linux machine. A screenshot of the AMI is shown in the figure.

```
NAME="Amazon Linux AMI"
VERSION="2018.03"
ID="amzn"
ID_LIKE="rhel fedora"
VERSION_ID="2018.03"
PRETTY_NAME="Amazon Linux AMI 2018.03"
ANSI_COLOR="0;33"
CPE_NAME="cpe:/o:amazon:linux:2018.03:ga"
HOME_URL="http://aws.amazon.com/amazon-linux-ami/"
```

Figure 3.7 EC2 AMI properties

The second step is choosing an instance type that is actually a hardware specification. There are 5 instance types that are fixed and their configuration cannot be changed. The instance types are classified as computing optimize, memory-optimized, Graphical Process Unit (GPU) optimized, storage optimized, and general-purpose (Kotas et al., 2018). Compute-optimized instance type is preferred for the applications that require more resources for computing, however, memory-optimized instances are used for the applications that need cache memory. On the other hand, while the GPU optimized instance type is preferred for gaming, the storage optimized instance is used mostly for the size and storage critical applications. Lastly, general-purpose instance type can be used for all generic applications. In this design, a general-purpose instance of AWS has chosen and its specification summarized in the following snapshot.

Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks
Vehicle Instance	i-06410420c1b02bfbb	t2.micro	eu-central-1b	running	2/2 checks ...

Instance: **i-06410420c1b02bfbb (Vehicle Instance)**
Public DNS: **ec2-18-184-26-197.eu-central-1.compute.amazonaws.com**

Description
Status Checks
Monitoring
Tags

Instance ID	i-06410420c1b02bfbb	Public DNS (IPv4)	ec2-18-184-26-197.eu-central-1.compute.amazonaws.com
Instance state	running	IPv4 Public IP	18.184.26.197
Instance type	t2.micro	IPv6 IPs	-
Elastic IPs		Private DNS	ip-172-31-38-147.eu-central-1.compute.internal
Availability zone	eu-central-1b	Private IPs	172.31.38.147
Security groups	launch-wizard-2 , view inbound rules , view outbound rules	Secondary private IPs	
Scheduled events	No scheduled events	VPC ID	vpc-1d2f3d76
AMI ID	amzn-ami-hvm-2018.03.0.20190514-x86_64-gp2 (ami-03a71cec707bfc3d7)	Subnet ID	subnet-5031012d
Platform	-	Network interfaces	eth0

Figure 3.8 EC2 general instance properties

The third step of the configuration is to add storage for the instance. There are storage types like Ephemeral Storage which is temporary and free, Amazon Elastic Block Store (EBS) which is permanent and paid, and Amazon S3, etc. (Amazon Elastic Compute Cloud, 2019). In the design architecture, Amazon's General Purpose SSD (gp2) volume type has been used with 8Gb size. The EC2 instance volume specification is shown in the below figure.

Name	Volume ID	Size	Volume Type	IOPS	Snapshot	Created
Vehicle Instance	vol-0ceca688cf51c...	8 GiB	gp2	100	snap-03abcba741f...	June 7, 2019...

Volumes: vol-0ceca688cf51cd0b5 (Vehicle Instance)	
Description	Status Checks Monitoring Tags
Volume ID	vol-0ceca688cf51cd0b5
Size	8 GiB
Created	June 7, 2019 at 10:40:54 PM UTC+3
State	in-use
Attachment information	i-06410420c1b02bfbb (Vehicle Instance):/dev/xvda (attached)
Volume type	gp2
Product codes	marketplace: {"billingProducts": [], "marketplaceCodes": [], "intrinsicAmiProperties": {"virt-method": "hvm", "sriov-net-
Alarm status	None
Snapshot	snap-03abcba741f52053c
Availability Zone	eu-central-1b
Encryption	Not Encrypted
KMS Key ID	
KMS Key Aliases	
KMS Key ARN	

Figure 3.9 EC2 volume representation

The fourth and last step of the instance configuration is creating a security group and generating a key pair for the private connection as part of VPC. The generated key pair is used on remote connection to the instance. On the other hand, the security group should be defined to allow which inbound and outbound connection protocols. For the Vehicle Instance in this architecture, the TCP protocol is chosen as a security group with Secure Shell (SSH), HTTP, and HTTPS types. The inbound security group is shown in the figure below. As seen from the defined inbound rules, the port “5000” has been set in order for the embedded system to connect and make HTTP POST operation to send the obtained data.

<input checked="" type="checkbox"/>	Vehicle Instance	sg-04d3fe5bca7f07b49	launch-wizard-2	vpc-1d2f3d76	691302247047
<input type="checkbox"/>	Vehicle Instance	sg-8bfc1eeb	default	vpc-1d2f3d76	691302247047

Security Group: sg-04d3fe5bca7f07b49

Description
Inbound
Outbound
Tags

Edit

Type ⓘ	Protocol ⓘ	Port Range ⓘ	Source ⓘ	Description ⓘ
HTTP	TCP	80	0.0.0.0/0	
SSH	TCP	22	0.0.0.0/0	
Custom TCP Rule	TCP	5000	0.0.0.0/0	
Custom TCP Rule	TCP	5000	::/0	
HTTPS	TCP	443	0.0.0.0/0	

Figure 3.10 EC2 inbound security group

3.4.2 Database Design

Relational database management systems (RDBMS) are in tabular format and Script Query Language (SQL) is used to query data. Contrary to RDBMS, NoSQL database is not tabular based and it provides storage and retrieval mechanism for the purpose (NoSQL, 2019). In addition to this, the NoSQL database system has an advantage over RDBMS in scalability and availability. The data structure in the NoSQL database is key-value paired and because of its horizontal scaling and object-oriented aspects, it's more flexible and easy to manage.

There are several NoSQL database systems available in the market and some are open source. MongoDB is one of them among all and provided by MongoDB Inc. (MongoDB, 2019). There are many products and services provided by MongoDB and MongoDB Atlas is one of them and it's used in this context as for the database design. Because the MongoDB Atlas is available on AWS regions, it's been preferred.

In MongoDB Atlas, a context with the name of "vehicle" and inside the vehicle context a cluster with the name of ClusterVehicle have been created. The context and cluster are shown in the following figure.

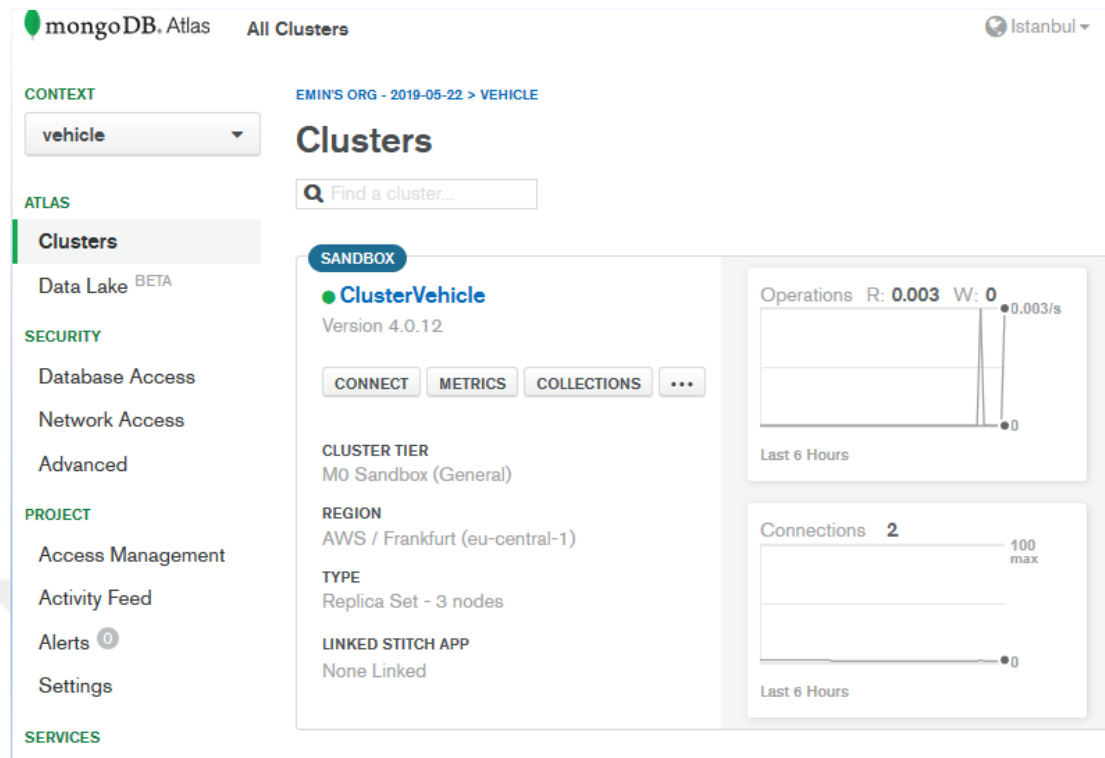


Figure 3.11 MongoDB context and cluster

In addition, the ClusterVehicle contains a database named “vehicleDB”. In the vehicleDB there are 4 collections named dtcs, gps, trends, and vins corresponds to the DTC service, GPS location service, Trend data service, and Vehicle Information service, respectively. The database collections are used both the back-end services and end-user application in order to store the obtained data and manipulate it in the application interface. The representation of the vehicleDB and collections are as follows.

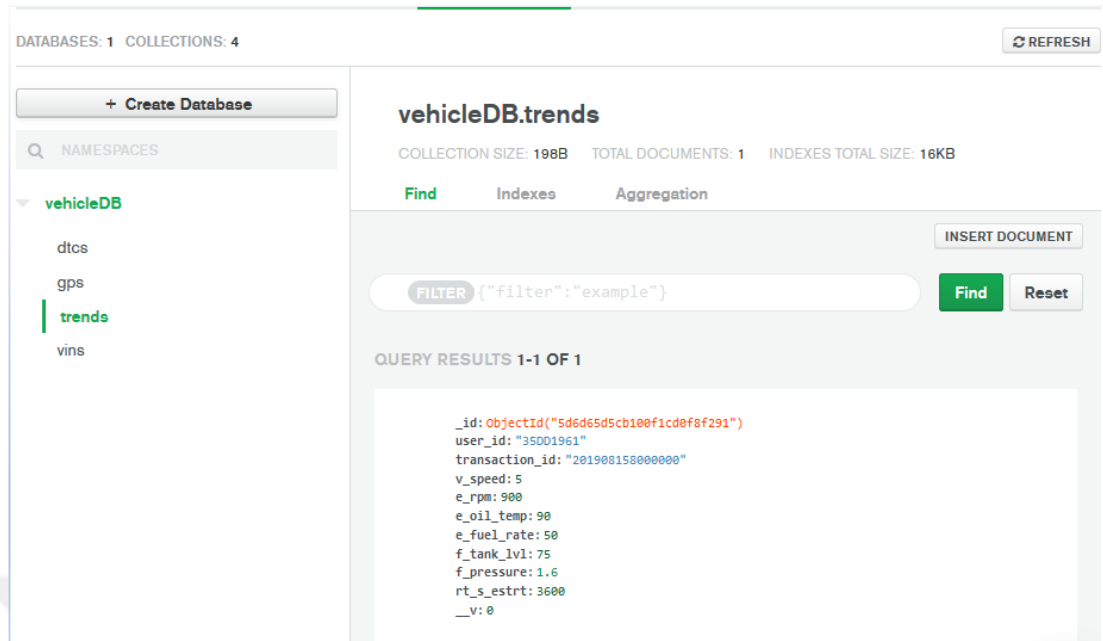


Figure 3.12 Vehicle DB collections

3.5 End-User Application Design

An End-User Application has been required to be designed because of the need for presenting the obtained data and to let the users monitor the vehicle's related information and location. The designed application is actually a front-end web page that has a login page and the main dashboard. The dashboard contains 4 pages that are Location, VIN, DTC, Trend. Each of these 4 pages corresponds to a back-end service so that each gathered data is shown in the proper section.

In legacy web pages, web form application frameworks were being used, however, Model-View-Control (MVC) structure has been developed in recent years and is being used in modern web applications. In this design, an MVC based responsive web application has been developed in Vue.js which is a JavaScript framework for single-page User Interfaces (UI). Vue.js is open-source and because of its responsive structure, it runs on any device that has a web browser. Therefore, Vue.js is preferred to be used in this prototype in order not to develop mobile and web application separately.

The web page has a login page as shown in the following figure and the users signup to the system with their vehicle's plate id and create a password.

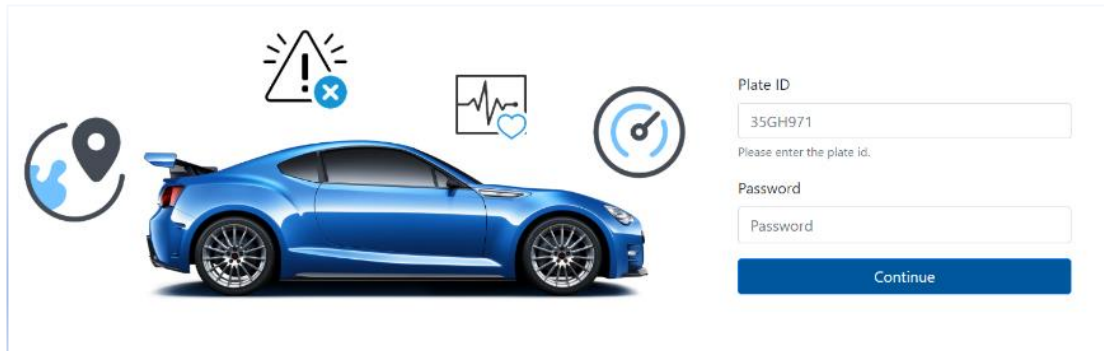
The login page features a blue sports car in the center. Above the car are four icons: a location pin, a warning sign with an exclamation mark and a blue 'x', a heart rate monitor, and a speedometer. To the right of the car is a form with two input fields. The first field is labeled 'Plate ID' and contains the text '35GH971'. Below it is a smaller text label 'Please enter the plate id.'. The second field is labeled 'Password' and contains the text 'Password'. Below the password field is a blue button labeled 'Continue'.

Figure 3.13 User application login page

When the users' login to the page they are directed to the main dashboard as shown in the following figure. The dashboard contains 4 pages that each presents particular information.



Figure 3.14 User application main dashboard

CHAPTER FOUR

RESULTS

The designed prototype has been applied and tested in an Opel Astra K 2016 model vehicle. The test car has ISO 15765-4 (CAN Bus 11 bit ID 500 Kbaud) OBD protocol. The OBD-II UART module is connected to the OBD port in the car and all the test has been realized during the development and design period of the system.

The OBD port in almost all the vehicles is placed under the steering wheel as shown in the following picture. The DB9-to-RS232 cable connection from OBD port to the embedded circuit is done as shown in the picture.



Figure 4.1 OBD port connection (Personal archive, 2019)

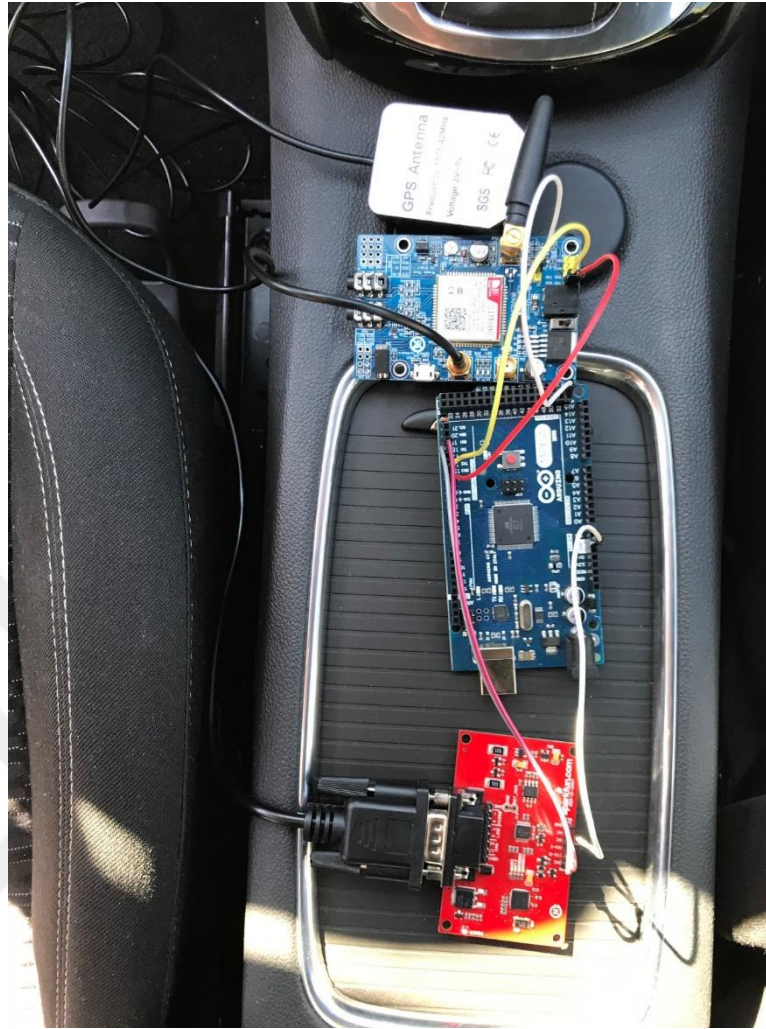


Figure 4.2 In-car embedded circuit connection (Personal archive, 2019)

Based on the Embedded System Software Architecture, OBD data read starts by requesting the vehicle identification number and vehicle fuel type. As soon as the software is installed and the embedded board connected to the vehicle, these two information are requested and it helps to identify the vehicle and receive its basic information. Then the GPS location, DTC, and trend data are retrieved continuously from the system and sent to the cloud in order to let provide real-time connectivity.

The afore-mentioned back-end services are ready to receive the incoming post requests and handle the provided data keep it in the database. The designed back-end services not only handle the post requests, but they also handle the get requests. This means that the services are bi-directional and the stored data can be requested by sending HTTP GET requests. Table 4.1 contains the list of post and get request URLs

of the services. The IP address and port number are seen in the URLs have been configured while the EC2 instance was created in the AWS.

Table 4.1 Back-End services endpoint URLs

Service	End-point URL
GPS	http://18.184.26.197:5000/vehicle/getGpsData
	http://18.184.26.197:5000/vehicle/postGpsData
VIN	http://18.184.26.197:5000/vehicle/getVIN
	http://18.184.26.197:5000/vehicle/postVIN
Trend Data	http://18.184.26.197:5000/vehicle/getTrend
	http://18.184.26.197:5000/vehicle/postTrend
DTC	http://18.184.26.197:5000/vehicle/getDTC
	http://18.184.26.197:5000/vehicle/postDTC

Since the services have been designed depending on the RESTful API structure, the data can be requested by concatenating the plate ID in the URL.

In the user application interface, there are 4 main pages as already described in chapter three. Each page corresponds to the related end-point service and represents the related vehicle information. On the one hand, both the vehicle's real-time and last location can be seen from the Location page. The page not only provides the location coordinates but also shows it in an embedded map. On the other hand, the system also provides the ability to draw a driving-route on the map as shown in the Figure 4.3.

In addition, the other 3 pages represent the vehicle information, diagnostic codes, and some trend data respectively. The handy and user-friendly GUI makes to monitor that information easily, in this sense some results are represented as in the following Figure 4.4 Trend data dashboard, Figure 4.5 DTC dashboard, and Figure 4.6 VIN dashboard respectively.

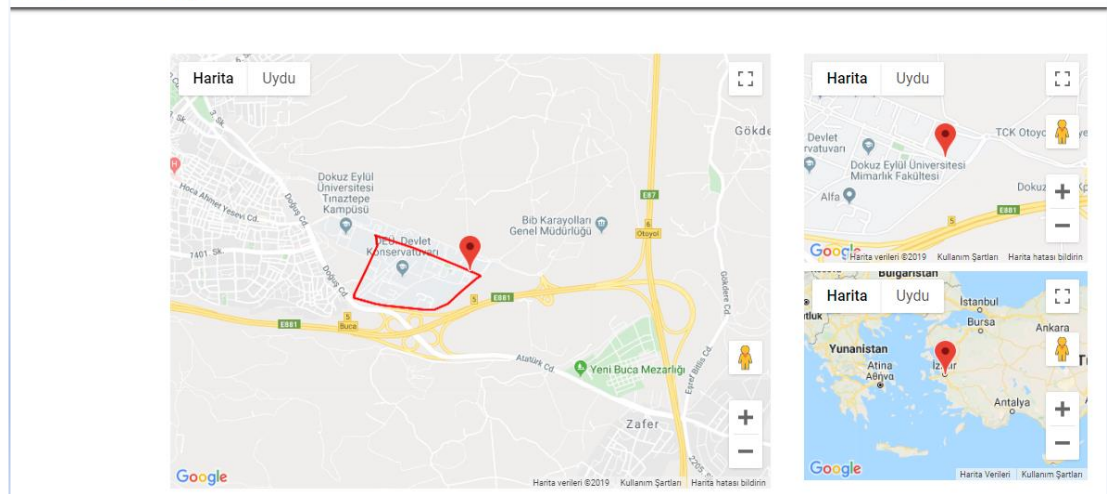


Figure 4.3 GPS location dashboard

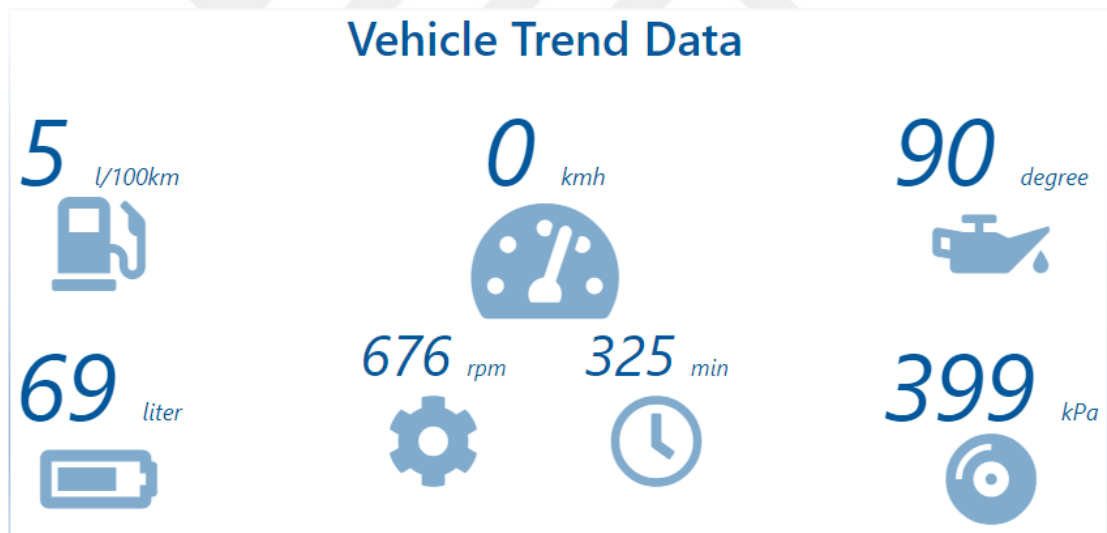


Figure 4.4 Trend data dashboard


Vehicle DTC			
			
35GH671			
Date	Fault Group	Fault Code	Fault Description
15.08.2019	Network	U1173	SCP (J1850) Invalid or Missing Data for Remote Button Control
29.10.2019	Power Train	P1183	Synchronization Fault
03.06.2019	Chassis	C1200	ABS Inlet Valve Coil LF Circuit Short To Battery
01.09.2019	Body	B1200	Climate Control Pushbutton Circuit Failure

Figure 4.5 DTC dashboard

Vehicle VIN Information



35GH671

MADE_IN	Spain
MODEL	Leon
MAKE	SEAT
YEAR	2016
VIN	VSSZZZ5FZGR183705

Figure 4.6 VIN dashboard

CHAPTER FIVE

CONCLUSION

In this thesis, a Real-Time Vehicle Monitoring and On-Board Diagnostic System have been designed in order to present the vehicle diagnostic and location information to drivers/users. Thus, the drivers/users will be able to monitor their vehicle's real-time location for the security aspect and knowing the vehicle's engine and electronic units troubles in advance as well. Besides, in the proposed web application they will have a chance to interpret their driving behavior, fuel consumption rates, driving ranges per week/month/ year, etc. based on the obtained trend data log.

The context of the thesis started by asking a research question other components how would the drivers know about their vehicle information remotely and be notified for the troubles exist in the vehicle. In this aspect, vehicles' On-Board Diagnostic port has been researched and studied for several 4-wheel vehicle manufacturers and the know-how of vehicle data exchange standards has been carried out during the period.

According to the ISO standards and already accomplished literature, there are 5 protocols that all the vehicle manufacturers should comply with at least one of them. However, the main problem was to design a system that can communicate with all the vehicles having one of these 5 protocols. For this purpose, an OBD board that contains the STN1110 chip has been used to differentiate the types of protocols regardless of vehicle model, and already existing ELM327 chip's command set has been made of use in the design.

REFERENCES

- Amazon Elastic Compute Cloud. (2019). Amazon elastic compute cloud user guide for Linux instances. Retrieved September 05 2019, from <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-ug.pdf>
- Amazon Web Services (2019). In *Wikipedia*. Retrieved September 10 2019, from https://en.wikipedia.org/wiki/Amazon_Web_Services
- Cloud computing (2019). In *Wikipedia*. Retrieved September 12 2019, from https://en.wikipedia.org/wiki/Cloud_computing
- El-Ata, M. A. (2000). Evolution of mobile cellular communication systems. The journey to UMTS. In *Proceedings of the Seventeenth National Radio Science Conference* 1-17.
- Elm Electronics Inc. (2010). *ELM327 AT Commands*. Retrieved August 22, 2019, from https://www.sparkfun.com/datasheets/Widgets/ELM327_AT_Commands.pdf
- Engine Control Unit (2019). In *Wikipedia*. Retrieved September 08, 2019, from https://en.wikipedia.org/wiki/Engine_control_unit
- Hans, J., Sethi, P. S., & Kinra, J. (2015) An approach to IoT based car parking and reservation system on cloud. In *2015 International Conference on Green Computing and Internet of Things (ICGCIoT)*, 352-354.
- International Telecommunication Union Internet Reports. (20005). *ITU internet reports 2005: The internet of things*. Retrieved September 03, 2019, from <https://www.itu.int/pub/S-POL-IR.IT-2005/e>

Internal Combustion Engine (2019). In *Wikipedia*. Retrieved September 07, 2019, from https://en.wikipedia.org/wiki/Internal_combustion_engine

ISO. (2019). *ISO 3779:2009 Road vehicles – vehicle identification number (VIN) – content and structure*. Retrieved September 16, 2019, from <https://www.iso.org/standard/52200.html>

Jing, P., Huang, H., & Chen, L. (2017). An adaptive traffic signal control in a connected vehicle environment: A Systematic Review. In *Information 2017*, 8, 101

Khorsravinia, K., Hassan, M. K., & Rahman, R. Z. A., & Al-Haddad, S., A., R., (2017, October) Integrated OBD-II and mobile application for electric vehicle (EV) monitoring system. In *2017 IEEE and 2nd International Conference on Automatic Control and Intelligent Systems (I2CACIS)*, 202-206.

Kirda, E. (2012, June). A security analysis of Amazon's Elastic Compute Cloud service. In *IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)* 1-1.

Kotas, C., Naughton, T., & Imam, N. (2018). A comparison of Amazon Web Services and Microsoft Azure cloud platforms for high performance computing. In *2018 IEEE International Conference on Consumer Electronics (ICCE)* 1-4.

Li, X., Gani, A., Salleh, R., & Zakaria, O. (2009). The future of mobile wireless communication networks. In *2009 International Conference on Communication Software and Networks*, 554-557.

MongoDB (2019). In *Wikipedia*. Retrieved September 01, 2019, from <https://en.wikipedia.org/wiki/MongoDB>

NoSQL (2019). In *Wikipedia*. Retrieved September 01, 2019, from <https://en.wikipedia.org/wiki/NoSQL>

- Narula, S., & Jain, A. (2015). Cloud computing security: Amazon web service. In *2015 Fifth International Conference on Advanced Computing & Communication Technologies*, 501-505.
- NHTSA. (2011). Quick Reference Guide (2010 Version) to Federal Motor Vehicle Safety Standards and Regulations. Retrieved September 09, 2019, from <https://www.nhtsa.gov/sites/nhtsa.dot.gov/files/fmvss-quickrefguide-hs811439.pdf>
- OBD Solutions. (2018). Multiprotocol OBD to UART Interpreter Datasheet. Retrieved August 21, 2019, from <https://www.scantool.net/downloads/updates/stn1110/>
- On Board Diagnostics (2019). In *Wikipedia*. Retrieved September 17, 2019, from https://en.wikipedia.org/wiki/On-board_diagnostics
- OBD-II PIDs (2019). In *Wikipedia*. Retrieved September 17, 2019, from https://en.wikipedia.org/wiki/OBD-II_PIDs
- SAE International J2012_200204. (2019). Diagnostic Trouble Code definitions equivalent to ISO/DIS 15031-6: April 30, 2002 J2012_200204. Retrieved September 07, 2019, from https://www.sae.org/standards/content/j2012_200204/
- SAE International J1979_201202. (2019). E/E Diagnostic Test Modes J1979_201202. Retrieved September 07, 2019, from https://www.sae.org/standards/content/j1979_201202/
- Sim, A. X. A., & Sitohang, B. (2014, November). OBD-II standard car engine diagnostic software development. In *2014 International Conference on Data and Software Engineering (ICODSE)*, 1-5.

SIMCom. (2014). SIM808_Hardware Design_V1.00. Retrieved August 08, 2019, from https://cdn-shop.adafruit.com/datasheets/SIM808_Hardware+Design_V1.00.pdf

Yun, H. J., Lee, S. K., & Kwon, O. C. (2011). Vehicle-generated data exchange protocol for remote OBD inspection and maintenance. In *2011 6th International Conference on Computer Sciences and Convergence Information Technology (ICCIT)*, 81-84.

Zhu, X., & Zhang, Y. (2011). An IOT based Car-bus for the 4WIDIS EV. In *2011 International Conference on Electrical and Control Engineering*, 3343-3345.