Qatar University

College of Engineering

Department of Computer Science and Engineering

Senior Project Report

ParQU - Parking System Using Cloud Computing Based on IoT

Project Group Members:
Ghareisa Mohammed Al-Kuwari (201402464)
Alaa Mousa Mousa (201305148)
Amal Khaled Haidarah (201302862)

Supervisor: Dr. Abdulaziz Al-Ali

2019

This project report is submitted to the Department of Computer Science and Engineering of Qatar University in partial fulfillment of the requirements of the Senior Project course.

Declaration

This report has not been submitted for any other degree at this or any other University. It is solely the work of us except where cited in the text or the Acknowledgements page. It describes work carried out by us for the capstone design project. We are aware of the university's policy on plagiarism and the associated penalties and we declare that this report is the product of our own work.

Student: Ghareisa Mohammed Al-Kuwari Date: 25-4-2019

Signature:

Student: Alaa Mousa Mousa Date: 25-4-2019

Signature:

Student: Amal Khaled Haidarah Date: 25-4-2019

Signature:

Abstract

Qatar University has become one of the most crowded places in Qatar, which with time is faced with a huge parking problem. That has led to a chaotic parking experience in the university. Faculty, staff and students complain about the negative effects that the lack of parking causes, such as not finding a vacant parking spot easily, which in turn makes them waste time and arrive late for their lectures. In the hopes of solving this issue, we planned to create a parking system based on the Internet of Things (IoT) concept. IoT is a network consisting of devices, machines or objects that are able to communicate and transfer data without the need for human-to-human or human-to-computer interaction.

The system allows users the ability to reserve a parking spot by subscribing our system. Also, the system allows even users without a subscription to view the availability of a parking spot. The parking system involves an Android mobile application, a website and a hardware prototype to be installed in the parking area. The hardware prototype consists of sensors such as Ultrasonic sensor and RFID (Radio Frequency Identification) for car detection and an embedded system to collect data from these sensors and communicate this information to the cloud. The on-board application and website demonstrates the functionalities of the system along with a cloud database for data storage.

Acknowledgment

Our project “ParQU - Parking System Using Cloud Computing Based on IoT” gave us a huge amount of experience and knowledge.

First and foremost, we would like to sincerely thank our supervisor Dr. Abdulaziz Al-Ali for his guidance, patience and motivation as well as providing us with the needed comments and suggestions that helped us improve and give our best work.

We would also like to thank our families, friends and colleagues who encouraged us to continue with the project.

Contents

List of Figures

List of Tables

1. Introduction and Motivation

Wherever a person goes to in a country with high population density areas, a parking issue will most likely be present. There is no difference in the case of Qatar as it is becoming more density populated, and one crowded place in the country is Qatar University (QU). Due to the lack of a good public transportation system in Qatar, most of the students in QU commute to the University by driving a car. The number of students at QU owning a car is increasing drastically each year, not to mention the increasing number of students QU accepts each year of which have led to the demand for more parking spots. As Qatar University students, we are personally motivated to come up with a solution to this problem.

1.1. Problem statement

Over the past few years, the number of students in Qatar University (QU) has been steadily rising each year, which in turn has led to an increase in the number of vehicles arriving on the campus. According to the Ministry of Education statistics from 2012, the number of students attending QU exceeded 10,000 students [1] and is only increasing with each passing year. Likewise, the number of staff and faculty that need parking for their vehicles has also increased. Today, the number of students at Qatar University has reached more than 20,000 [2]. As a result, there has been a rapid increase in the number of people and vehicles on campus, and with the limited available parking spaces, parking problems are bound to exist. An online survey was shared with Qatar University students to assess the severity of the parking problem. As shown in Figure 1.1, 94.3% of the survey respondents believed that there is a real problem in finding a vacant parking spot inside the campus.

Figure 1.1: Statistics from Survey

Frustrated students have taken to social media to air their criticism on the ongoing crisis of parking areas at Qatar University [3]. Students have circulated comments, pictures, and videos that represent their daily suffering due to the lack of parking spaces allocated to them in the University. Many sarcastic and critical tweets have discussed this issue, as displayed in Figures 1.2 until 1.5.

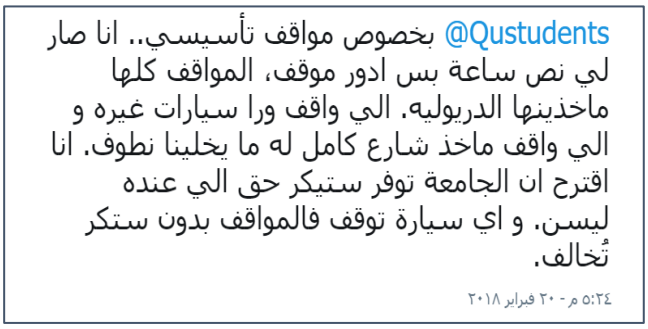


Figure 1.2: Tweet from Twitter about the parking problem

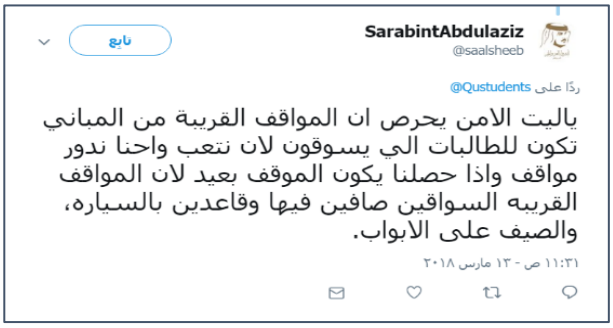


Figure 1.3: Tweet from Twitter about the parking problem

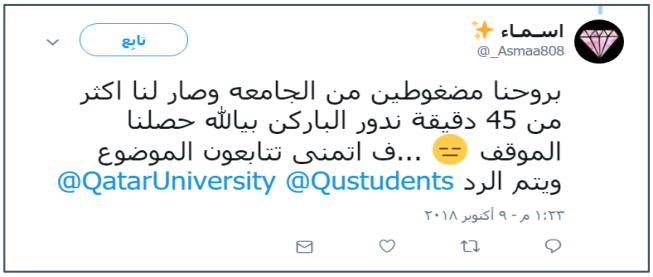


Figure 1.5: Tweet from Twitter about the parking problem

Figure 1.4: Tweet from Twitter about the parking problem

The problem has transcended student conversation to become the talk of the media in Qatar. Figures 1.6 up to 1.11 show headlines from prominent local newspapers (Al-Raya, Al-Watan, Al-Arab and Al-Sharq) that discuss the parking problem at Qatar University and its impact [4]. The articles made mention of the fact that the effects of this problem were not only limited to students and staff but also impacted the visiting guests who come for events, ceremonies, seminars or conferences.

Figure 1.7: Article from Al-Sharq Newspaper

Figure 1.6: Headline from Al-Raya Newspaper

Figure 1.8: Headline from Al-Arab Newspaper

Figure 1.9: Headline from Al-Arab Newspaper

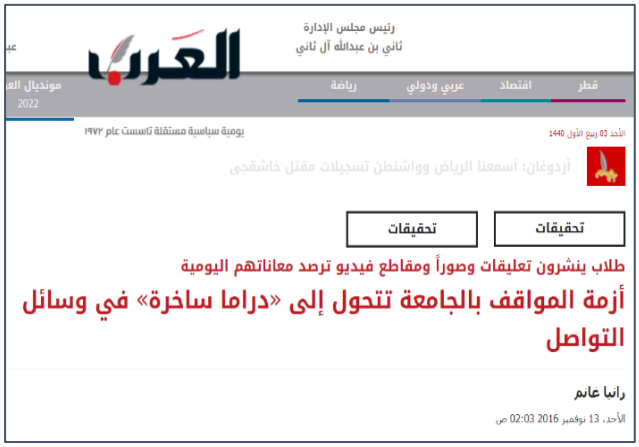


Figure 1.10: Headline from Al-Arab Newspaper

Figure 1.11: Headline from Al-Watan Newspaper

Impact

People looking for an empty parking spot constitutes one of the most significant causes of congestion and wastes the time of students and staff. As shown in Figure 1.12, more than 60% of the students who responded to our survey reported that they take more than 15 minutes to find a vacant parking spot. To a student, 15 minutes late could mean being late for an exam or a quiz. Also, if the professor takes attendance at the start of the lecture, the student may be counted as absent. Further, Qatar University follows an attendance policy that states that a student automatically fails a course if he/she is absent for more than 25% of the class sessions. Hence, if the problem continues, it is possible that students will exceed the allowable absences and fail one of their courses. As a result, many students have declared that they are forced to come to the University early so that they can get a parking slot for their cars [5].

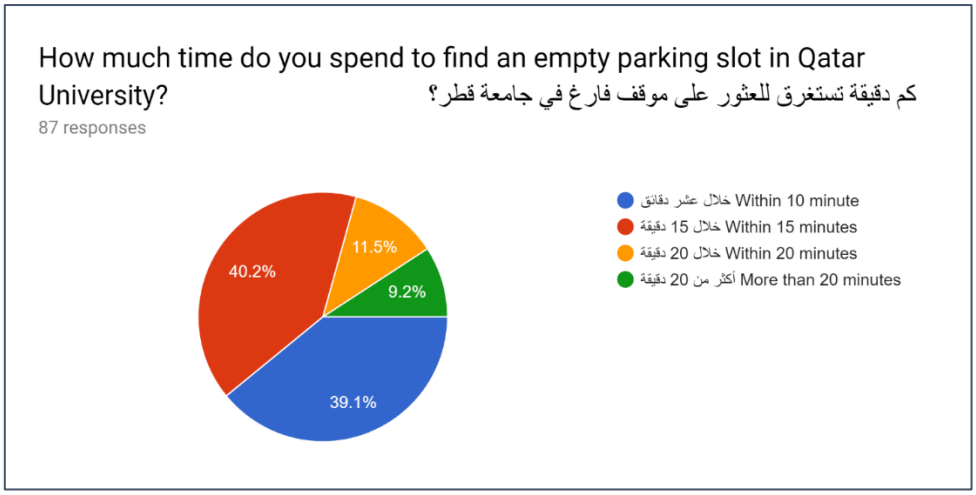


Figure 1.12: Statistics from Survey

Even though people rarely notice or discuss the environmental impact, the parking problem creates environmental effects as it causes vehicles to consume more fuel while driving around to search for parking spots. This increase in fuel consumption wastes the vehicle owner's money and leads to more CO2 emissions.

Additionally, hurried drivers are forced to park in illegal parking places which in turn leads to an increase in the number of traffic violations and accidents inside the campus. Figures 1.13 and 1.14 below show examples of illegal parking where cars are parked over sidewalks and on the grass field.

Figure 1.13: Vehicles in Qatar University parked illegally

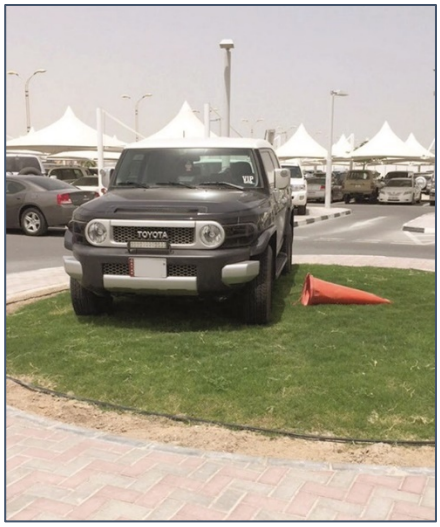


Figure 1.14: Vehicles in Qatar University parked illegally

In the end, the University's reputation is negatively affected by this situation, because it is the first thing that guests see during their visit as well as students and staff when they start their day.

1.2. Project significance

A parking system is needed to organize and utilize the parking area resources. An adequate parking system would be able to guide the vehicle owners at the University in such a way as to reduce traffic overcrowding by giving the users the location of the available parking spots. Such a system could reduce the vehicle owner's search time significantly by providing such information. As a result, traffic congestion would decrease, and the University would be able to fine vehicle owners when they park illegally.

A smart parking system would not only be beneficial for anyone entering the campus of Qatar University but would also be considered valuable for the environment [6][7]. The environment is more protected by the decreased CO2 emissions from cars that would reduce the level of pollution [6]. Also, for the security and other monitoring staff, the data gathered via the implementation of the parking system could be used to predict future parking patterns. Strategies for setting fines and parking fees could also be manipulated based on the information gained to increase the University's profit.

1.3. Project objectives

1. To design and implement a system that can manage parking areas.
2. To use sensors (Ultrasonic Sensor and RFID Reader) to collect data.
3. To use microcontroller boards (Arduino) to receive data from used sensors and send it to a database (Firebase) for storage.
4. To design an Android application and a website.

2. Background and related work

In this section, we introduce the parking problem background and some related existing solutions that aim to organize the parking spaces

2.1. Background

Finding a parking spot may seem like a superficial problem or an individual problem, but is, in fact, one that causes significant issues in the day to day life of people. In metropolitan areas worldwide, having a parking area is increasingly important, and a substantial amount of land and buildings are set aside to accommodate the growing demand for parking spaces. For instance, in a developed country like Australia, parking areas hold great importance especially in major cities Brisbane, Sydney, and Melbourne, where a vast number of parking spaces can range from 25,633 to 41,687 [8]. Furthermore, estimates show that around 30,000 square kilometers of land in Europe and 27,000 square kilometers in the US are devoted to parking spaces [9].

Moreover, parking can be very time consuming and costly. Hence, it comes as no surprise that "The U.S. economy bears the brunt of parking pain as 40% of drivers say they have avoided driving to shops due to parking challenges." [10]. According to a study from the car service company Inrix, Americans spend an average of 17 hours per year searching for a vacant parking spot, which leads to a loss of $345 per driver in wasted time and fuel. Similarly, drivers in the UK spend an average of 44 hours a year looking for an empty parking spot, with an estimated total loss of $954.

Contrary to the popular belief that generous parking allocations benefit users, the opposite can happen when there is a surplus of parking allocations. An enormous parking lot can prolong transportation times and is a waste of useable land. These effects, along with recent land-use, and socioeconomic and technological trends, are prompting towns to begin asking some critical questions about how to solve the parking problem smartly and cost-effectively. For this reason, there have been many attempts to provide smart parking systems as a solution, with each having its different approach as shown in the Related Work section.

2.2. Related work

Several studies conducted and presented in previous literature related to our project have assisted us in the development and implementation of our design.

A Secure Parking Reservation System Using GSM Technology

In [11], GSM technology was used to propose a secure and smart reservation system for parking areas. The system consists of two major modules which are the security reservation module and parking lot monitoring.

Secure Reservation Module

The main idea is that the user has to send an SMS message using GSM (Global System for Mobile communications) to a Visual Basic (VB) application in a laptop/PC. The VB then processes the requested data and checks the available spot. The authors of the study use open APIs in VB to handle SMS messages. If the requested parking spot is available, the user will receive a confirmed message that has the location of the spot and password. Otherwise, the user will receive a reject message.

Parking Lot Monitoring Module

The primary aim of this module is to show the status of the parking spots using Parking Layout Animation program through Visual Basic application in a laptop/PC. The program displays the status of each parking spot through colors, green for *empty*, yellow for *reserved*, and red for *occupied*. Initially, the state of all parking spots is *empty*. The status will change to *reserved* if the user successfully reserves a parking spot. When the user arrives at the entrance gate, he enters the password, which after will be verified by the controller. If the password is valid, the gate will open, and the Parking Layout module will update the spot status to *occupied*.

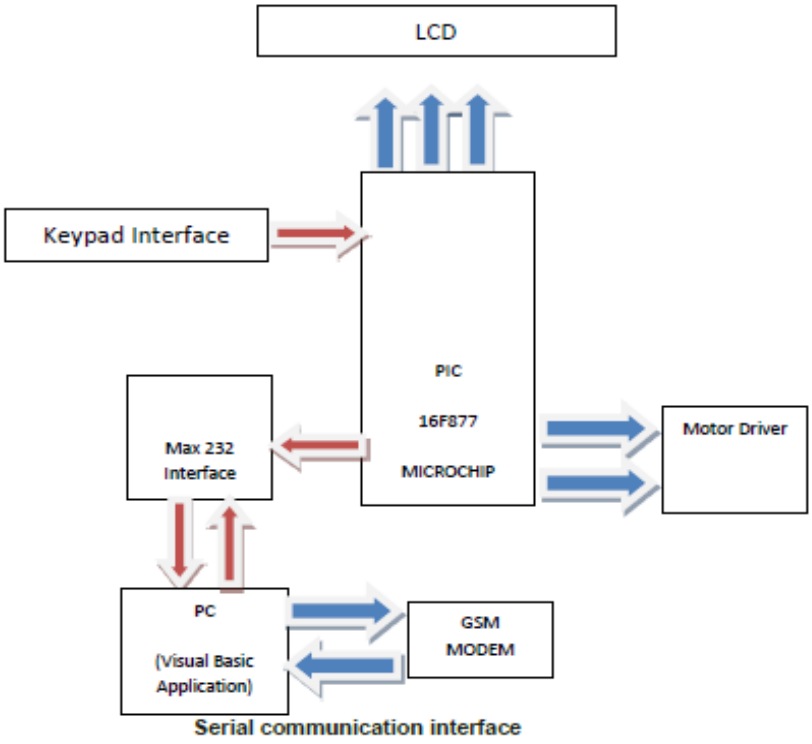


Figure 2.1: A Secure Parking Reservation System Using GSM Technology – System architecture

By using this approach, the user will be charged by the telecommunication company for sending the SMS messages. However, GSM is prone to be easily congested in crowded areas [12]. In this approach, all information is stored locally on one computer, which is inefficient as the computer memory will eventually be too full to save any more information from the system. Also, if the computer is somehow damaged, all the data will be lost.

Intelligent Parking Space Detection System Based on Image Processing

In [13], an image processing technique was used to detect whether the parking spot is empty. The proposed solution is carried out in 5 steps as follows:

1- System Initialization:
In the beginning, a rounded brown image is drawn at each parking spot manually to determine their positions.

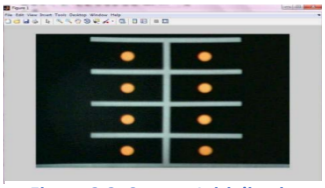


Figure 2.2: System Initialization

2- Image Acquisition:
The camera captures an RGB image of the car park scene and sends it to the processing unit that runs in MATLAB.

Figure 2.3: Image capture from camera

3- Image Segmentation:
MATLAB converts the RGB Image to a grey scale image. Then thresholding technique on the grey scale image is applied to create a binary image. The technique helps in separating the objects from the background.

Figure 2.4: Binary image

4- Image Enhancement:

The morphology functions such as dilation, erosion, opening, and closing are used to remove noise from the binary image as well as to trace the exterior boundaries of the object.

Figure 2.5: Image after Tracing object boundaries

5- Image Detection Module:

The system detects if the parking spot is empty or not by identifying the rounded brown image at every spot.

Figure 2.6: Detected parking spot

This approach may not function well in extreme weather conditions, as the camera might be damaged. Also, placing the camera in a suitable location such that all parking areas are visible, and there are no objects that might obstruct the camera's vision is essential for the camera to work as planned.

Monitoring Parking Space Availability via Zigbee Technology

In [14], the system's primary function is to allow its users to view the available parking spots through a screen at the gates of a parking area using Zigbee wireless technology for communication and digital infrared sensors for vehicle detection.

The system consists of two main modules:

Parking Lot Vacancy Monitoring Module

This module comprises of digital infrared sensors, Zigbee module, PIC microcontroller 18F4550, and an LCD. This module is used to detect the available parking spots. A digital infrared sensor is placed on top of each parking spot to detect the existence of a vehicle such that each sensor is used to monitor one parking spot. Infrared sensors work with the reflected light waves and consist of an IR transmitter and an IR receiver. The IR transmitter emits infrared light, which gets reflected when it meets a reflecting surface (e.g., white color). The IR receiver then detects the reflected light and calculates the distance between the sensor and the object. The infrared sensor interfaces with a PIC microcontroller, which also interfaces with a Zigbee module for wireless transmission. When the sensor senses a vehicle in the parking spot, it informs the microcontroller, and then the microcontroller notifies the master module through the Zigbee module.

Master Module

The master module consists of a GUI display shown through a PC or a laptop, and a Zigbee module. The GUI display, shown in Figure 2.7, allows the user to see precisely which parking spots are available. Whenever the Zigbee module receives data about a particular parking spot from the Parking Lot Vacancy Monitoring module, it informs the PC/laptop that it is interfaced with to update the status of a parking spot.

Nevertheless, this system has several disadvantages. The infrared sensor is entirely dependent on light as brighter surfaces are more accessible for detection than darker surfaces. Hence, changing

light conditions could yield wrong outputs. Also, the system does not have a mobile application for a remote user interface and restrict its users to a GUI interface at the gates of the parking lot.

Figure 2.8 shows the hardware architecture of the system.

Figure 2.7: GUI Display on PC or laptop

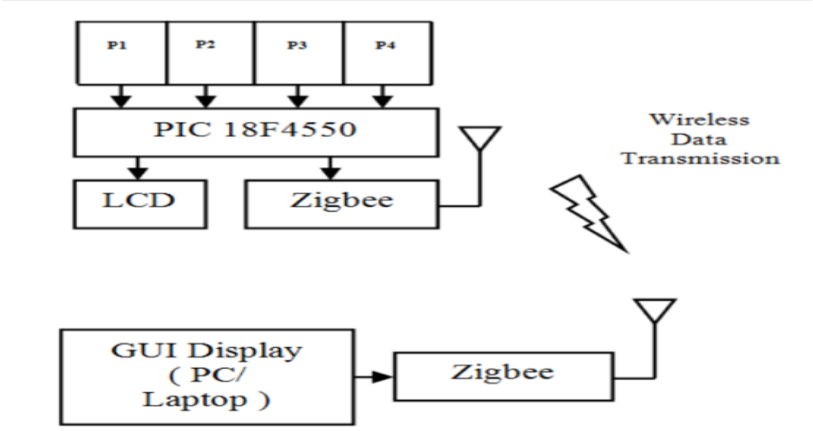


Figure 2.8: Hardware architecture of the system

Smart Parking System (SPS) Architecture Using Ultrasonic Detector

[15] is a smart parking system for multilevel parking lots, which uses ultrasonic sensors to detect the occupancy of a parking space and shows it to the user at the parking area using display boards. The display boards show the number of available parking spots, and they are implemented indoors, at the entrance to each level and the end of each aisle of the parking lot, and outdoors, at the entry and exit of the parking lot. Furthermore, an ultrasonic sensor is installed on the ceiling of each parking spot and is used to detect the presence of a car in that spot only. LEDs with different colors accompany each ultrasonic sensor to inform the user of the parking spot status (reserved, occupied, vacant or handicapped). Moreover, the system includes a monitoring software and line detection system to detect improper parking. The line detection system operates by two additional ultrasonic sensors horizontally on the right and left of each parking spot such that they face the parking spot

lines. Whenever a car goes over the detection line, an alarm will go off until the vehicle has moved out of the line.

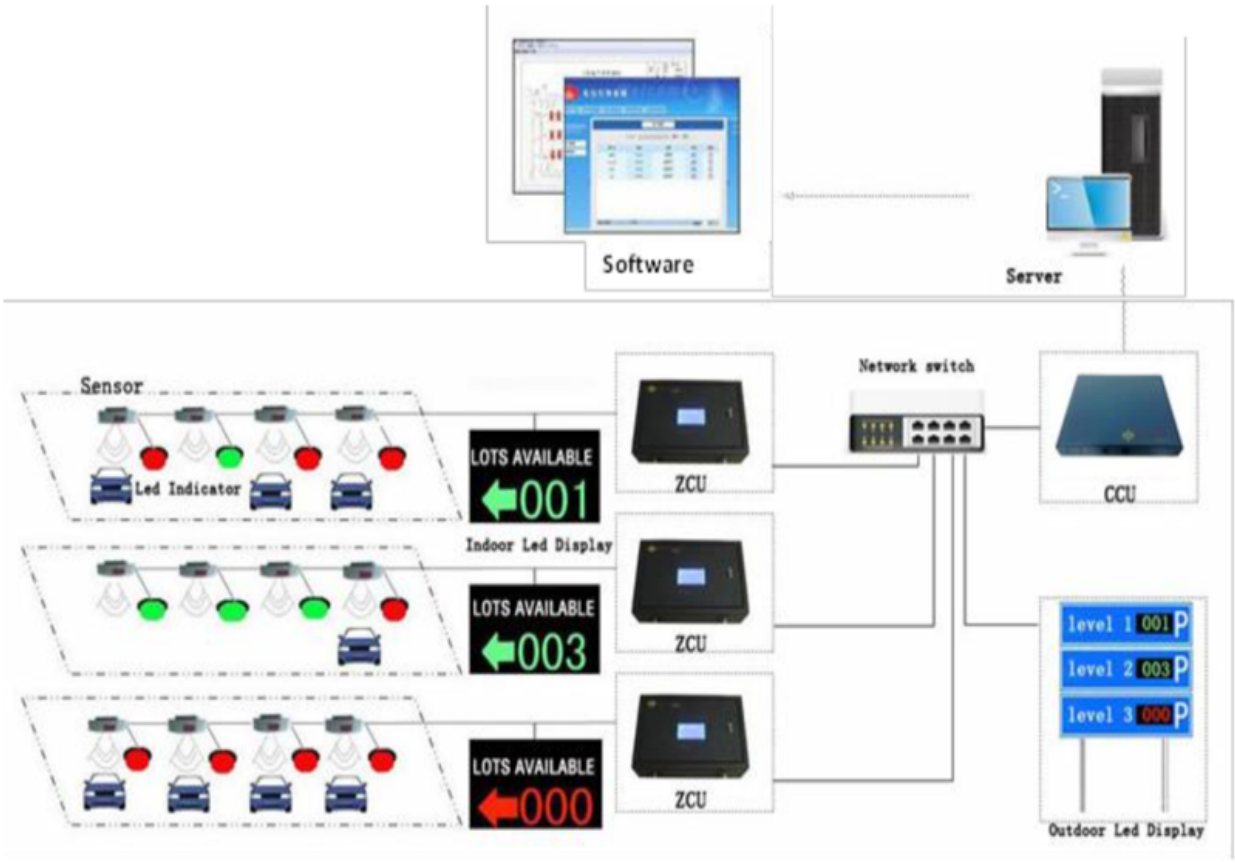


Figure 2.9: Hardware architecture of the system

Figure 2.9 shows the overall hardware architecture of the system. The system consists of ultrasonic sensors, LED indicators, indoor display boards, outdoor display boards, zone control units (ZCU), a central control unit (CCU), network switch, telephone cable, and management software. When the ultrasonic sensor detects a change in the status of a parking area, it transmits the status to the zone control unit (ZCU) through a telephone cable. The ZCU then forwards the information to the central control unit (CCU) with Cat5 cables and sends commands to the indoor display boards and the LEDs on the parking sports to change status accordingly. Both the indoor display boards and the LEDs connect to the ZCU through RS-485 ports. The ZCU connects to the central control unit (CCU) through network switches and LAN connections. The CCU upon receiving new information from any of the ZCUs processes the data with the whole parking lot data, then transmits commands to the outdoor display boards to update the parking area information.

The system is able to efficiently manage multilevel parking areas. However, the system contains many hardware components and connects its components through a great number of cables. Any extension to the system would require more cables and more complications on the system.

Intelligent Parking System

In [16], the system uses image processing techniques which capture, and process circles or rounded images drawn from the parking spot, which are then displayed in an android application. This rounded image indicates empty parking spaces to users.

A camera is used with a sensor to take photos to show the occupancy of car parks. The application displays whether a parking space is available or not. If the camera detects the vacancy of a parking area, a green circle appears to the user. If there is a car in a spot, it becomes unavailable, and no green circle is shown at that spot in the application.

The system contains three modules: A Monitoring module, Control module, and a Displaying unit. The monitoring module has ultrasonic sensors which identify the free parking spaces and transmit information to the control unit which processes and sends data to the administrative system. Besides that, there is a centralized system that supervises the received parking information from the controller. This information is then sent to the user's phone.

Below (Figure 2.10) illustrate the working process of the system.

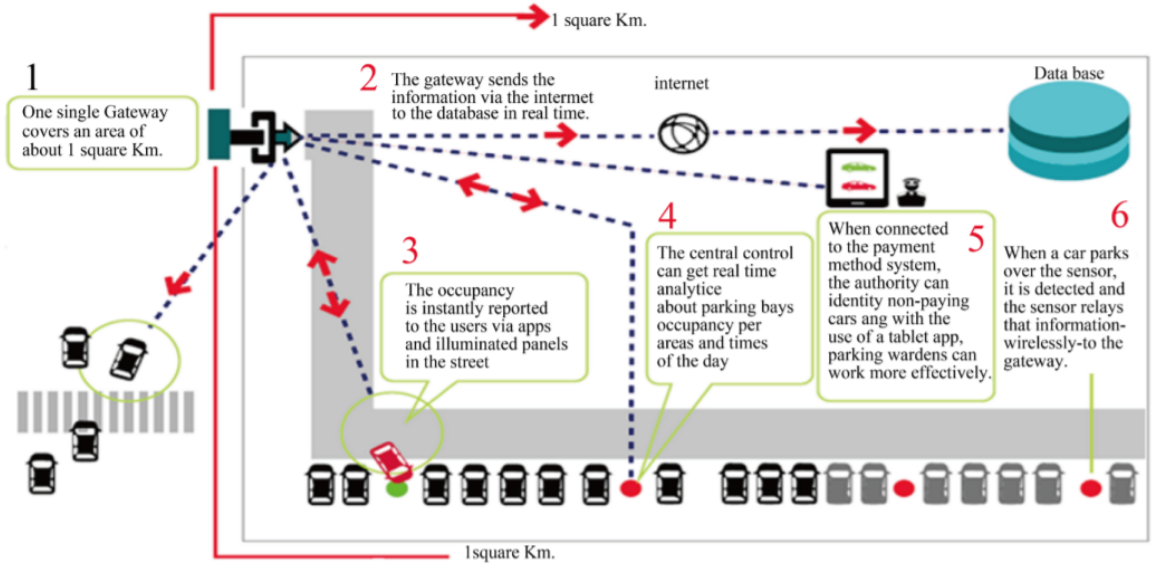


Figure 2.10: Working process of the system

Related work comparison

Table 2.1: Related work comparison

Paper	Hardware Used			Service			User Interface Through	Connection
	Microcontroller	Sensors	Others	Reservation	Availability checking	Payment		
[11]	Pic Microcontroller		LCD, Motor driver	✓			Visual Basic Application	GSM
[13]			Camera		✓			

[14]	Pic Microcontroller	Infrared Sensor	Zigbee-module LCD		✓		GUI displayed on PC or laptop	Wireless
[15]		Ultrasonic Sensor	Display Boards, LEDs, Network Switch		✓			Telephone cables, LAN connection
[16]		Ultrasonic sensors	Camera	✓	✓	✓	Android Mobile Application	
Our Method	Arduino Uno	Ultrasonic RFID	Servo motor, LEDs	✓	✓	✓	Mobile-Application Website	Wireless

Table 2-1 compares the existing solutions to ParQU. All the related works implement one of the following services: Checking Availability, Reservations, Payment except [16]. [16] gives all the above functions that we aim to provide in our system. However, [16] only implements an Android application while we will provide a website in addition to that. Furthermore, the above approaches use ultrasonic, infrared and camera for car detection only. Our method uses an ultrasonic sensor for car detection and RFID (Radio Frequency Identification) for car identification and detection.

3. Requirements analysis

In this section, the software development process used in this project is defined. The functional and nonfunctional requirements of the system are addressed.

3.1. Software development process

The software development process in our project is Scrum. Scrum is an iterative and incremental agile software development process that focuses on adapting to rapid changes. Scrum was chosen for our project because it does not require us to know precisely how and what to do from the beginning of the development process. The Scrum process is responsive to changes and improvements with repeated reviews and work inspection [17]. Figure 3.1 shows the Scrum process overview.

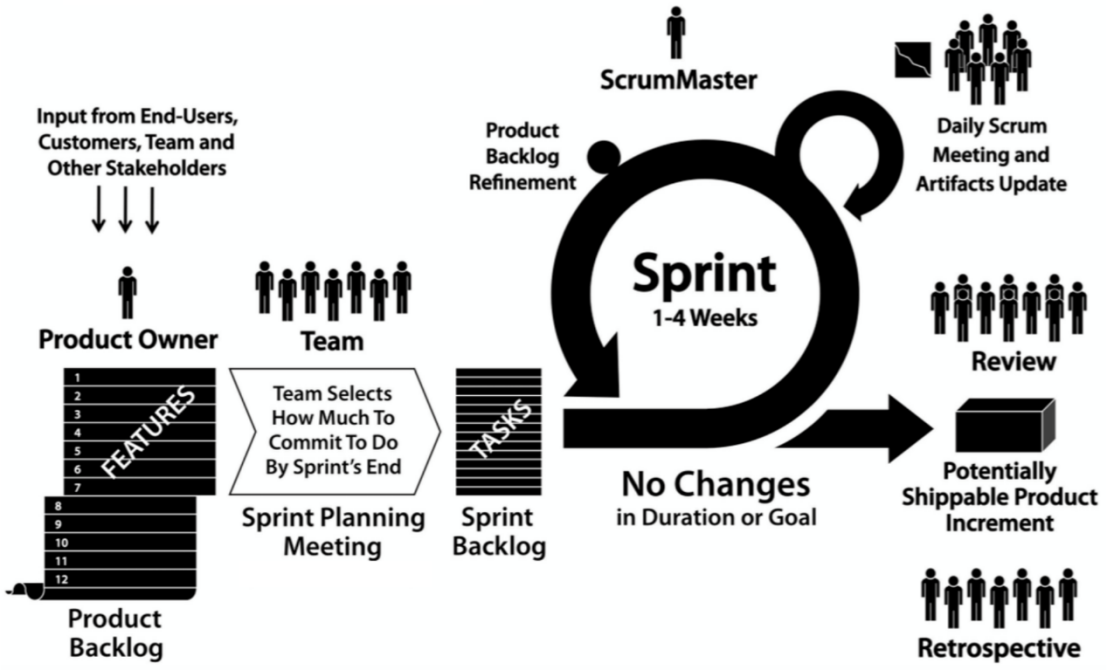


Figure 3.1: Scrum process overview

The central part of the Scrum cycle is the Sprint. Each Sprint is time restricted to one week and has four steps as shown in Figure 3.2.

Figure 3.2: Scrum cycle

1. Sprint Planning

At the beginning of the project, we collected the requirements into a Product Backlog from the users by researching the parking problem in Qatar University and the users' needs. Before starting a Sprint, we held up a meeting to decide on the top priority requirements from the Product Backlog and created a Sprint Backlog. The Sprint Backlog contained the tasks that we could complete during the Sprint which in our case was one week.

2. Sprint

In this stage, we started working on the tasks as specified in the Sprint Backlog. Further, a Daily Scrum Meeting was held to discuss the tasks that were done by each member so far, and the tasks for the next meeting. Also, if any member has faced any difficulties during the task, we tried to find a solution for it.

3. Sprint Review

At the end of a Sprint, a meeting is held with the ScrumMaster (supervisor) to demonstrate our work. The ScrumMaster suggested solutions to the issues we faced during the Sprint.

4. Sprint Retrospective

After the Sprint Review, we refined our tasks to meet the comments and suggestions given by the ScrumMaster and decided on the improvements for the next Sprint.

3.2. Functional requirements

- An RFID reader should be installed and kept operational at the gate of the parking facility.
- VIP users must have an RFID tag in their possession before entering the parking facility.
- Sensors should be hardwired to Arduino.
- Arduino must be connected to a power source.
- Arduino must be connected to a Wi-Fi module.
- Firebase must have a connection with the Arduinos, the mobile application and the website via Wi-Fi.
- The mobile must have internet connection.
- The user should be a University member or visitor.

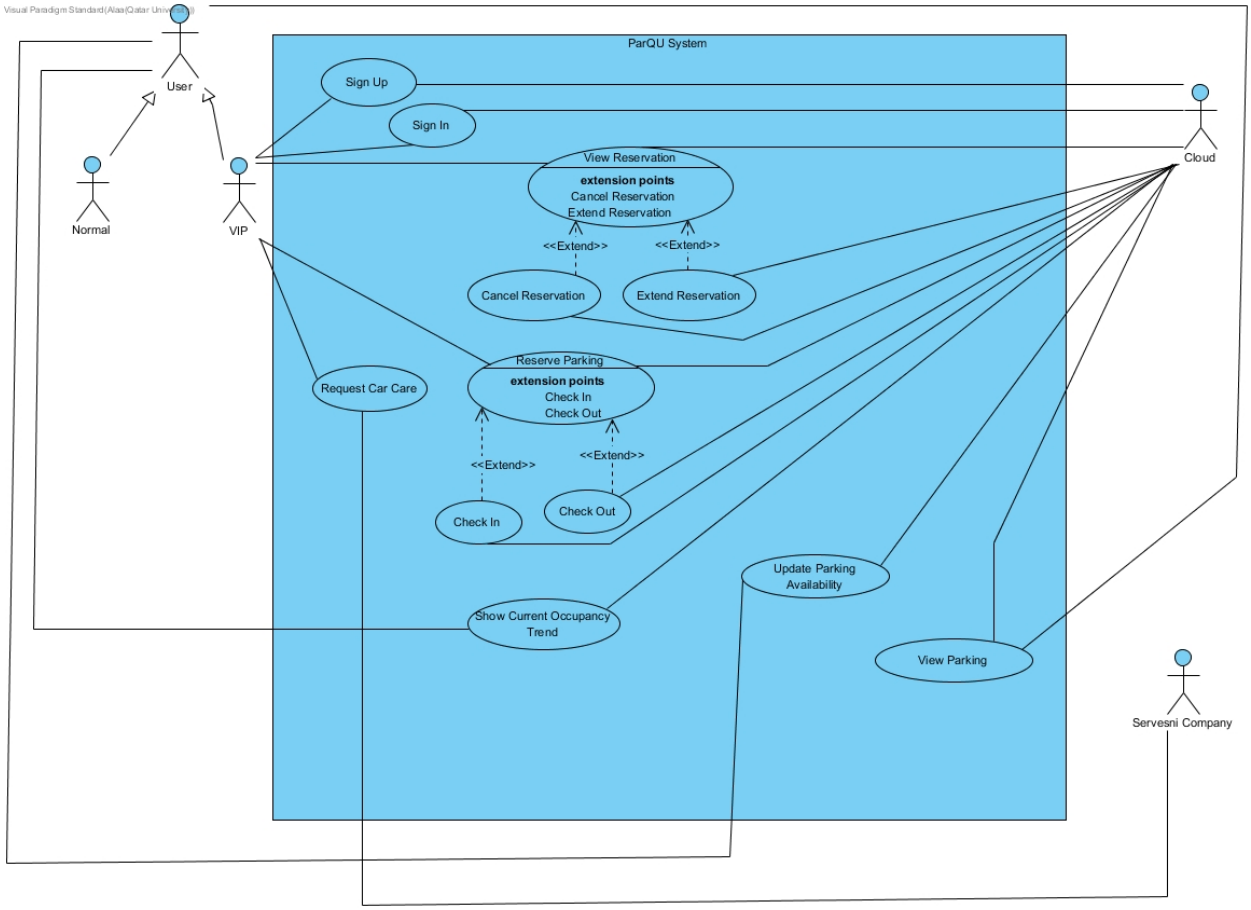


Figure 3.3: Use case diagram

Table 3.1: Actor type and descriptions

Actor types	Description
VIP User	Anyone subscribed to the VIP services such as reservation service and requests car care service.
Normal User	Anyone who is not subscribed to the VIP services.

Table 3.2: Use cases and descriptions

Use case	Brief description
Sign Up	VIP user registers in the system.
Sign In	VIP user logs into the system.
Reserve Parking	VIP user reserves a parking spot for several hours.
View Reservation	VIP user views reservation details.
Extend Reservation	The VIP user extends a reservation.
Cancel Reservation	The VIP user cancels a reservation.
Check In	The system reads the RFID tag on the car, checks its validity, then allows reserved VIP users only to enter the parking lot.
Check Out	The system reads the RFID tag on the car, checks its validity, then update available parking spots.
View Parking	The user views a map with the current status of parking spots.
Update Parking Availability	The system receives updated data from sensors through the Arduino board.
Request Car Care	The VIP user requests car care service which will redirect VIP user to Servesni (a car care company).
Show Current Occupancy Trend	The user views statistical data that represent occupied percentage in each hour for selected zone for last four weeks.

3.3. Non-functional requirements

In this section, non-functional requirements such as the technical design constraints and practical design constraints and design standards of the system are addressed.

Design constraints

Table 3.3: Technical design constraints

Name	Description
Reliability	Data is saved at all times on the Firebase database, so no data will be lost.
Scalability	The system can support the addition of sensors and components as well as have more users and the cloud can be upgraded accordingly.
Connectivity	Firebase database needs to be connected to the internet to collect updated data from the sensors through the Arduino board. Also, the mobile application needs to be connected to the internet to get the data

Availability	The system should be always available. During using the application, if any data is changed in the firebase database, the application should automatically display the updates without needing to refresh the page.
Mobility	The system can be accessed from different platforms through an Android application and a website
Power	Power source needed for: RFID Reader: 3.3V Ultrasonic Sensor: 5V NodeMCU: 7-12V Arduino: 7-12V

Table 3.4: Practical design constraints

Type	Name	Description
Economics	Cost	The prototype for availability module which consists of 4 parking spots for each Arduino Uno (maximum) should cost 270 QR on average. The prototype for reservation module should cost 260 QR on average.
Social	Usability	A normal user with minimal software knowledge should be able to use the mobile application and the website with ease
Sustainability	Maintenance	The system's component should be easy to replace, remove and implement.
Quality	Performance	The system should provide efficient information and accurate readings from the parking area.

Design standards

Table 3.5: Standard types and descriptions

Standard Types	Description
Networking Standard	HTTP (Hypertext transfer protocol)
Wireless Standard	IEEE 802.11 (Wi-Fi)
Serial Communication Standards	SPI (Serial Peripheral Interface)

3.4. Assumptions

Below are some assumptions that are beyond the scope of this project but necessary for the project to work properly:

- Arduino is always connected to the sensors.
- A power source is connected to the Arduino.
- Arduino always has a Wi-Fi connection.

- Firebase can always store data without limits.
- Application and website are always available.
- Mobile and PC that access our application and website is always connected to the internet.
- VIP user always has an RFID tag.

3.5. Ethics

Table 3.6: IEEE and ACM code of ethics

Code	Project Perspective
IEEE To seek, accept, and offer honest criticism of technical work, to acknowledge and correct errors, and to credit properly the contributions of others. ACM - Honor property rights including copyrights and patent. - Give proper credit for intellectual property.	Any project or work used is referenced in this paper. Feedback will be accepted by the supervisor and the examiners.
IEEE To treat fairly all persons and to engage in acts of discrimination based on race, religion, gender, disability, age, national origin, sexual orientation, gender identify, or gender expression. ACM - Be fair and take action not to discriminate.	Project members have all worked together to produce this work which can be used by all types of people.
IEEE To assist colleagues and co-workers in their professional development and to support them in following this code of ethics.	The project is collaborative; every team member will work on it and ensure the code of ethics is strictly followed.
ACM - Contribute to society and human well-being. - Manage personnel and resources to design and build information systems that enhance the quality of working life.	The design aims to achieve better standards and benefits.

4. Proposed solution

The possible solutions for our system are listed with their properties. Then our solution is presented with the justification.

4.1. Possible solutions and tradeoffs

The system has a software and hardware part. Each of these parts has its own possible solutions.

Hardware possible solutions

- Comparison between different sensors

There are many different sensors that can be used for detecting the presence of vehicles which are suitable for our smart parking system. One of them is the RFID (Radio Frequency Identification) method/technique/approach which comes with a reader and a tag. The idea is to use radio frequencies for detection and identification of each vehicle by installing an RFID tag on it. Another approach is to use an Infrared sensor which can detect a vehicle. This is done by detecting the reflected infrared light waves that the sensor emits. A third method is to use an Optical sensor that uses light rays that are converted into electrical signals for detection. Additionally, an Ultrasonic sensor can be used which emits and detects any reflected sound waves. Finally, a camera with image processing applied can be used for detecting vehicles.

In [18], a study was conducted which compared the different types of sensors used in existing approaches. A short version of their table is provided in Table 4-1 below.

Table 4.1: Comparison between different sensors

	Flexible	Small Size	Installation	Accuracy	Cost
Infrared	✓	✓	*	**	*
Ultrasonic	✓	✓	*	***	*
Camera	✓	✓	*	**	***
Optical	✓	✓	*	**	*
RFID	✓	✓	**	***	**

- Comparison between different development boards

Arduino and Raspberry Pi development boards can both be used as part of our solution. In [19], a comparative table was made between the two. A short version of their table is provided in Table 4.2 below.

Table 4.2: Comparison between Raspberry Pi and Arduino

SL	Raspberry Pi	Arduino
1	It is a mini computer with Raspbian OS. It can run multiple programs at a time.	Arduino is a microcontroller, which is a part of the computer. It runs only one program again and again.
2	It is difficult to power using a battery pack.	Arduino can be powered using a battery pack.
3	It requires complex tasks like installing libraries and software for interfacing sensors and other components.	It is very simple to interface sensors and other electronic components to Arduino.
4	It is expensive.	It is available at a low cost.

5	The processor used is from ARM family.	Processor used in Arduino is from AVR family Atmega328P.

Possible software solutions

- Comparison between services

Availability and Reservations are the two main services that can be provided to the user. Availability is mainly about showing the user a map that shows if a parking spot is available. Reservation allows the user to reserve a parking spot before coming to the parking area.

The two main criteria that we are going to use to compare between the two services are flexibility and guarantee of a parking spot.

Regarding flexibility, the availability service is more flexible than the reservation because whenever the user parks, he can stay in the parking for as long as he wants. However, in the case of reservation, the user should leave whenever his reserved time is over. Users of the Reservation service guarantee the availability of parking spots once they successfully place a reservation. However, in the Availability service, users may not find the parking spot available as they can be occupied at the time of arrival.

Table 4.3: comparison between services

Services	Flexibility	Guarantee of available parking spot
Availability	✓	X
Reservation	X	✓

Regarding the reservation period/duration, there are several options that can be taken into consideration, but they vary in realism and defects. As mentioned in Table 4-4, it is possible to extend with hour-based reservation whilst it is not with day-based reservation. Using the day-based reservation approach, the parking spots will not be utilized to the maximum, as opposed to the hour-based reservation. For instance, if a user wanted to reserve a parking spot for one hour in a day-based reservation system, he will have to reserve the whole day which will waste a parking spot for a day. This also means the system cannot serve the maximum number of users possible.

Table 4.4: Comparison between reservation duration options

Reservation Duration	Extension possibility	Serve maximum number of users	Resources utilization
Per hour	Possible	✓	✓
Per day	Not Possible	X	X

4.2. Proposed solution

Overview

After surveying the above options regarding the sensors, microcontrollers and services, we combined several components to give the best solution we think will satisfy the goals of the project. The proposed solution is a hybrid solution consisting of two main modules: A Reserved Parking module and a Reservation Free Parking module. The reasoning behind our design decisions will be provided throughout this section.

The modules will provide the user with the two main services discussed above: reservation and availability. However, each of these services along with some additional services will be given according to two types of users: VIP users and Normal users. On one hand, Normal users do not require registration nor any type of information from the user. On the other hand, VIP users require subscription registration and installation of an ID tag to be able to enter the parking area. Normal users will be given the check availability, availability percentage, get directions, currently looking and show occupancy trend services, whereas VIP users will be given all the services available for normal users in addition to reserve a parking, show reservations, extend a reservation, cancel a reservation, notification, and request car care services. The details of this services are provided in table 4.5.

Table 4.5: Description of services

Free Services (Available for both type of users)	
Service	**Description**
Check availability	One of ParQU's main features is to give users the ability to view the status of current parking areas. This is done by displaying the parking area map with a status shown for each parking spot. We applied concept on 4 spots only.
Availability Percentage	This feature shows the availability percentage of spots in each zone to the users. A progress bar is used to illustrate the availability percentage. The progress bar color will be changed based on the percentage, green if more than 75% of the spots are available, orange if 50% of the spots are available and red if 25% of the spots are available.
Get Directions	This feature gives users the ability to get directions for a specific parking spot. Each spot is associated with a specific coordination (latitude, longitude). By using Google Maps, we are able to give users directions to the wanted parking spot from his/her current location (device location).
Currently Looking	It gives the users the ability to know how many people are viewing a certain zone at the current moment.

Show occupancy trend	The user views statistical data that represent occupied percentage in each hour for selected zone for last four weeks.
Paid Services (Available for VIP users)	
Service	**Description**
Reserve a parking	Another main feature that ParQU provides to VIP users is the ability to reserve a parking spot for several hours. The VIP user is allowed to reserve 24 hours before the reservation time and can reserve up to 6 hours per day, for either a multiple or single reservation. The VIP user is charged with 5 QR per hour.
Extend a reservation	The VIP user can extend his current reservation by one hour at the last hour of his/her reservation only when there are available parking spots in the next hour. This is because it is unlikely that the parking will use its resources to the fullest within the next hour if it has available parking spots, hence as a way of utilizing our resources (parking spots), we allow the VIP user to extend his reservation. Additionally, the VIP user can extend (one hour per extension) as many times as he desires if the above conditions hold. The VIP user will be charged 5 QR per extension and the extended hours will not be part of the allowable reservation time (6 hours per day). This is because a VIP user may need more than 6 hours, so we allow the user to extend as much as he wants but only if there are available parking spots.
Cancel a reservation	Cancel whole reservation: A VIP user can cancel a reservation with 50% of the reservation price refunded. The reservation can be cancelled at the time when the reservation is made till the reservation start time Cancel part of the reservation: A VIP user can cancel part of the reservation (i.e. after the reservation has started). Moreover, 50% of the price of each cancelled hour is refunded to the VIP user Automatic cancellation: A VIP user can cancel his/her reservation without needing to cancel through an application or website. This is done when the VIP user checks out of his reservation by at least one hour before the reservation ends.
Show reservations	The VIP user can view a list of current and upcoming reservations. Each one of the reservations has 2 options which are “Extend” and “Cancel”.
Request car care	The VIP user is redirected to the Servesni application that provides various car care services.
Notification	A VIP user is aided with a reminder about his/her current reservation expiry time. The application/website notifies the VIP user 30 minutes before expiry time.

To help us make some design decisions, we collected information about the user needs in our system by conducting a survey for QU students. To establish a suitable number of hours to reserve per day, we found from the survey, as shown in Figure 4-1, that on average the most appropriate number of hours to reserve per day is 6 hours. However, we did take into consideration that the majority prefers to reserve more than 6 hours, and applied an extension feature depending on the parking's availability

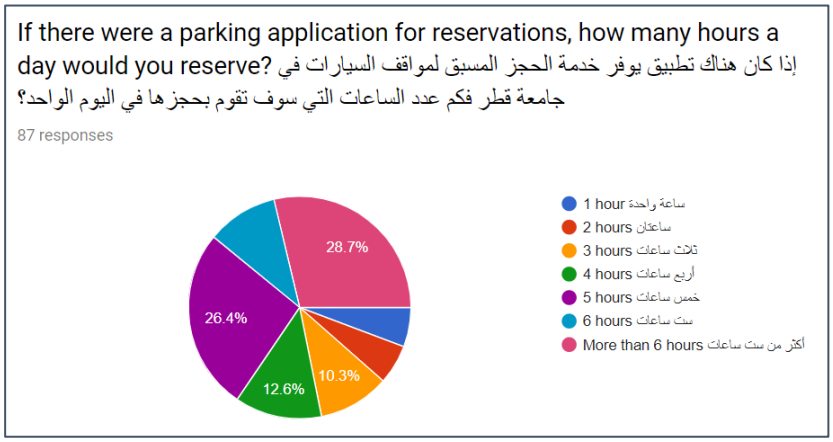
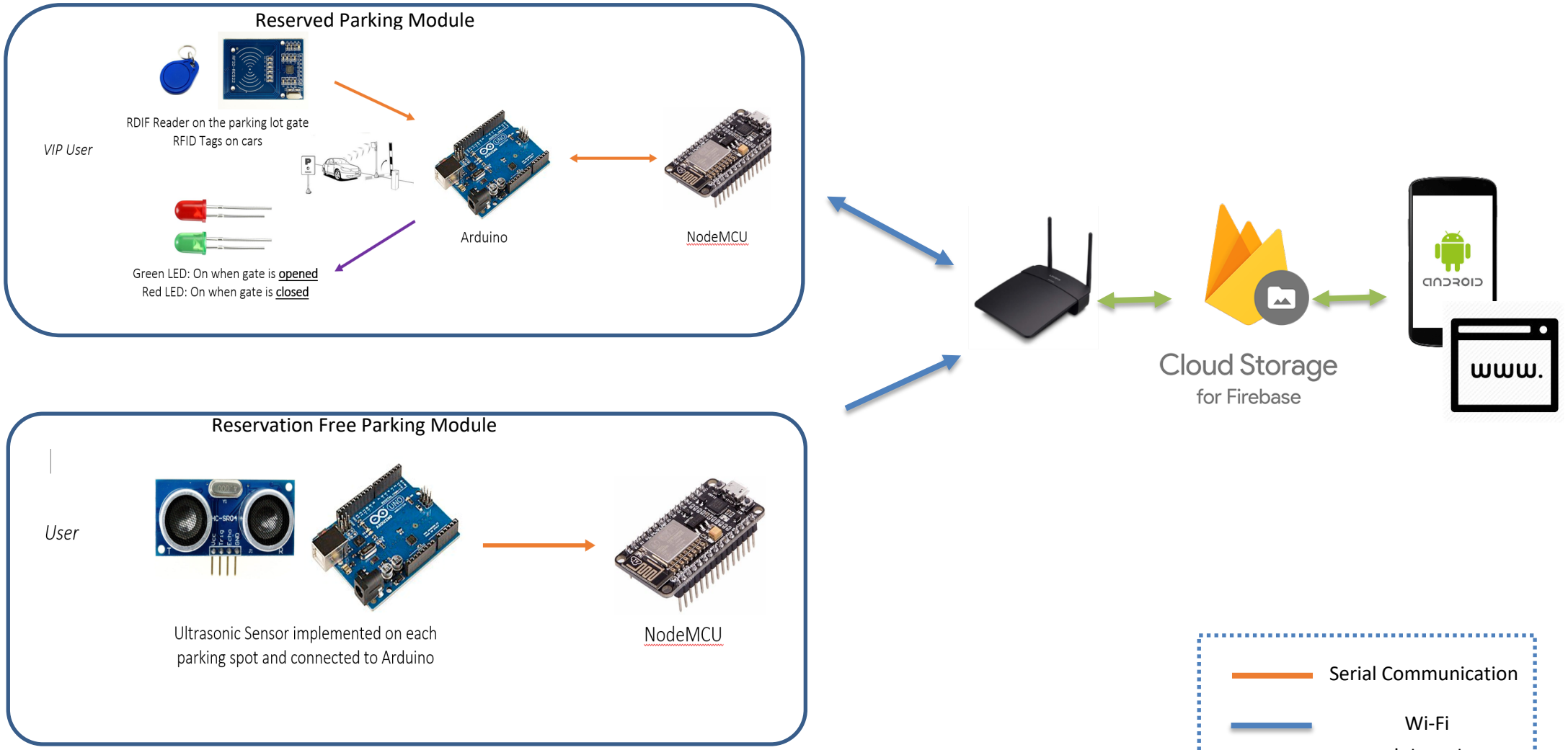


Figure 4.1: Statistics from Survey

High level architecture



- Reserved Parking Module

In this module the user is required to reserve through the mobile application or website. Whenever the user arrives at the gate of the parking area, the RFID reader will read the RFID tag and send the information to Arduino. The Arduino will send the acquired data via Wi-Fi to the (Firebase) cloud database service. The application running on the Arduino module will check the response received from Firebase and whether there is a reservation associated with this tag at this date and time. If the response is verified, then the Arduino will open or close the gate through a physical gate system; in our project, this is represented by a green led for (allow entry) and red led for (refuse entry). The connection between RFID sensor and Arduino is serial.

- Reservation Free Parking Module

The module's main purpose is to let the users know if there are available parking spots in a certain zone through a mobile application or website. The application will display the current parking area status in real time by showing a map. Whenever a car is parked in an available space, the Ultrasonic sensor will detect it and inform the Arduino which will then update the Firebase. The Ultrasonic sensor is physically connected to the Arduino.

Both modules have a Wi-Fi connection with Firebase whilst the application and website are connected to the Firebase through the internet.

4.3. Hardware/software to be used

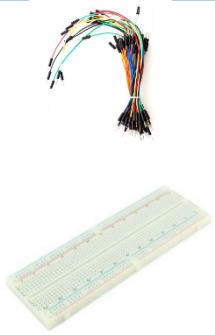
The hardware and software components used in this project are in the following two tables.

Hardware components

Table 4.6: Hardware components with descriptions

Component Name	Description	Price	Image
Arduino Uno Rev 3	Arduino Uno is a microcontroller board. Arduino Uno can be programmed to perform tasks. In the system, it is used to process the information given from the sensors and RFID reader and pass it on to NodeMCU.	115 QR	 Figure 4.2: Arduino Uno
NodeMCU V2 Model ESP8266	NodeMCU V2 is an open source IoT platform which is able to connect to Wi-Fi. The chip is also able to perform tasks with a certain program. NodeMCU is used to connect and update to Firebase database.	44 QR	 Figure 4.3: NodeMCU model ESP8266

Ultrasonic Sensor - HC-SR04	Ultrasonic sensor to detect whether a car is parked or not	17 QR	 Figure 4.4: Ultrasonic sensor - HC-SR04
RFID Kit - MFRC522	RFID Kit that includes an RFID Reader module, round tag, and then RFID card. The RFID tags are put on the cars as an identification, then the RFID reader is used to read the tags and identify cars at the gates of the parking lot.	35 QR	 Figure 4.5: RFID kit
DPDT Self Locking Button	Self-locking button to switch between zones (As it will be explained in section 5)	5 QR	 Figure 4.6: DPDT Self Locking Button
LEDs (Red and green)	LEDs to indicate the state of the parking lot gate. Red as closed and green as opened.	2 QR	 Figure 4.7: LEDs (Red and green)
9V to Barrel Jack Adapter	This cable is used to battery-power the Arduino that needs 9V and has an on-board barrel jack	4 QR	 Figure 4.8: 9V to barrel jack adapter

Full-Size Breadboard and Jumper Wires	For the connections between the different parts and Arduino, a full-size breadboard and jumper wires are used.	20+20 QR	 Figure 4.9: Full-size breadboard and jumper wires

Software components

Table 4.7: Software components and descriptions

Software	Image	Description
Firebase	 Figure 4.10: Firebase	Firebase is a complete backend solution that can manage authentication, real time database and hosting as well. It can be used in iOS apps, android apps and even in web apps.
Android Studio v.3	 Figure 4.11: Android Studio	Android Studio is the official IDE for Android development. It can be used to design user-friendly app as it has a variety of built-in functionality.
Arduino IDE	 Figure 4.12: Arduino IDE	Arduino software supports C/C++ based programming language. Used to write program to control the Arduino and NodeMCU microprocessor to perform tasks.
Fritzing	 Figure 4.13: Fritzing	Fritzing is an open-source that is used to develop the design of the electronics hardware. The software supports different kinds of hardware circuits diagrams.

WebStorm	 Figure 4.14: WebStorm	WebStorm is JavaScript IDE that is used to build website applications.
Postman	 Figure 4.15: Postman	A Google Chrome application interacting with HTTP, used as a tool for prototyping API's, in addition to several testing features.
Adobe Photoshop	 Figure 4.16: Adobe Photoshop	Graphics editor published by Adobe Inc, used to create, edit and enhance images.
Mockplus	 Figure 4.17: Mockpplus	Tool for user experience (UX) and user interface (UI) designers for creating prototype with drag-and-drop.

4.4. Hardware design

The proposed hardware solution is divided according to the module. The Reservation Free Parking module's hardware solution consists of an Ultrasonic sensor to detect if there is a car, Arduino board to collect information from the sensors and NodeMCU board to allow the Arduino to send and receive data to the Firebase. The Reservation Free Parking module needs a sensor implemented on each parking spot and so in total a huge number of sensors are needed depending on the number of parking spots. The Ultrasonic sensor has a high accuracy and a low cost. Hence, it was chosen among the other sensors in Table 4.1. Also, the Arduino was chosen over the Raspberry Pi because it is suitable for interfacing sensors with it and has a low cost which makes it more scalable. The connectivity circuit diagram for the module is shown in Figure 4.18.

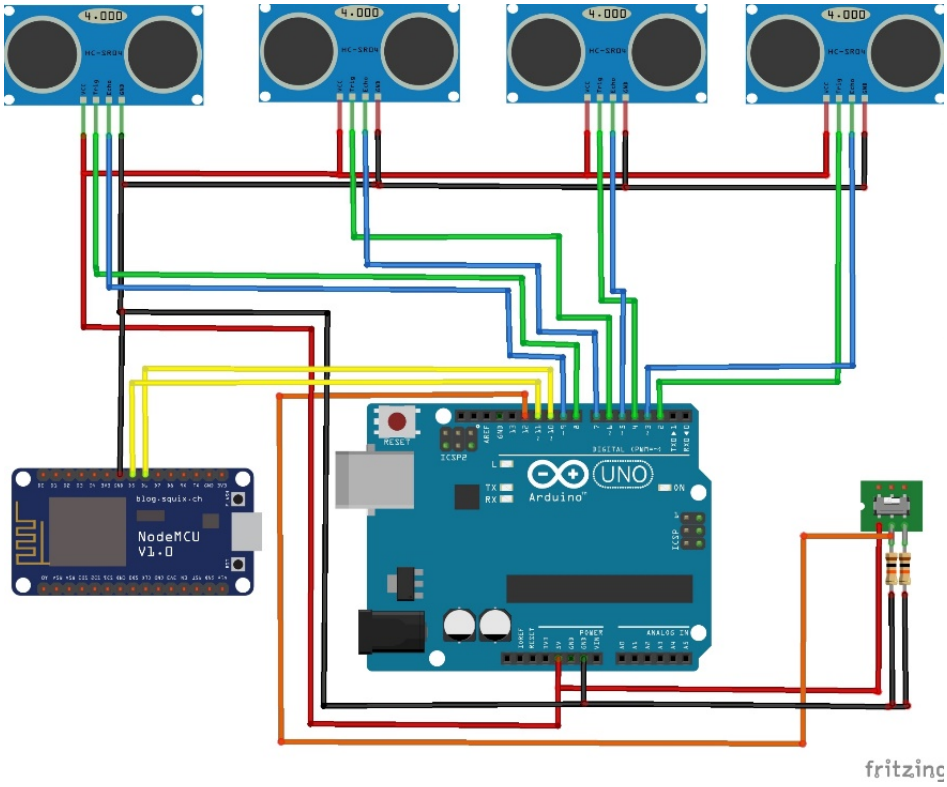


Figure 4.18: connectivity circuit diagram Reservation Free Parking module

In addition, the hardware solution for the Reserved Parking module consists of an RFID reader to read the RFID tag on the car, Arduino board to collect information from the RFID reader, LEDs (green and red) to indicate the status of parking area gates and NodeMCU to allow the Arduino to send and receive data to the Firebase. The RFID was chosen because it provides an important functionality over the other sensors in Table 4.1. Unlike the other methods, RFID is able to identify exactly the users entering or leaving the parking area. Hence, the RFID sensor can be implemented at the gates of the parking area and the system only allows those who have registered and reserved a parking spot to enter. The Arduino was also chosen in this module because it is more suitable for the RFID and the LEDs that are interfaced with. To power the system in our testing prototype, a 9V battery will be used. However, a power grid should be used in real life implementation. The connectivity circuit diagram for the module is shown in Figure 4.19.

Both modules include a switch button that is installed to switch between two zones for each module. This is to show the availability and scalability of our design without additional hardware. With each update from any of the two main modules, the Arduino informs the NodeMCU with the update along with which zone. After that, the NodeMCU updates the Firebase with the zone taken into consideration.

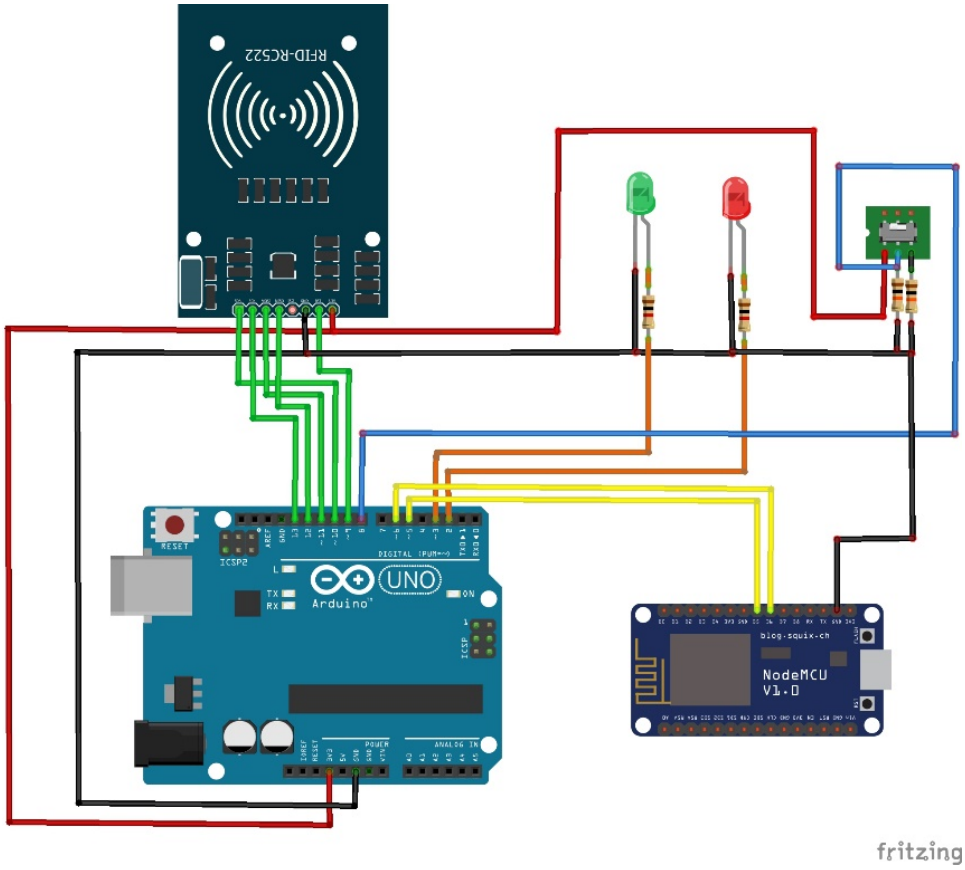


Figure 4.19: connectivity circuit diagram for Reserved Parking module

Hardware components

The components of the system besides the microcontrollers Arduino and NodeMCU are: Ultrasonic sensor, RFID sensor and the DPDT switch.

- Ultrasonic Sensor HC-SR04

HC-SR04 is a distance measurement sensor which is able to detect object and their distance. The sensor is able to give 2cm to 400cm non-contact measurement with an accuracy ranging up to 3mm or less. The sensor includes an ultrasonic transmitter, a receiver, and a control circuit. The four pins to connect the sensor with the Arduino are:

- VCC: 5v power
- Trig: Trigger (Transmitter) pulse
- Echo: Echo (Receiver) pulse
- GND: 0v ground

Figure 4.20: Ultrasonic Sensor HC-SR04

The working principle of the sensor is as follows:

- Transmitter (Trig pin) sends a signal (a high frequency sound).
- Signal is reflected when it detects an object.
- Receiver (Echo pin) receives the reflected signal.
- Time is measured by how long it took the signal from to reach the object and then go back to receiver.
- Object distance is calculated from measured time.

- RFID Reader MFRC522

Radio-Frequency Identification (RFID) uses electromagnetic fields to transfer data over short distances. The RFID system consists of two components:

  - RFID tags: The tags hold data known as unique identification (UID). The tag used in our project is the keychain tag (Figure 4.21) and should be attached to the object to be identified.

Figure 4.21: RFID tags

  - RFID reader: A two-way radio transmitter-receiver. The reader (Figure 4.23) sends a signal to the tag to power it up and reads its response which is its UID. The reader is powered with 3.3v, its read distance is 0-60mm. MFRC522 supports SPI, I2C and UART serial communication links. The reader communicates with Arduino using SPI interface.

There are 8 pins on the RFID reader (Figure 4.22) to connect with Arduino. Four of them connect SPI interface of reader (SDA, SCK, MOSI, MISO) and the other four are for power connections (IRQ – interrupt, RST – reset, GND – ground, 3.3v – power)

Figure 4.22: 8 pins on RFID reader

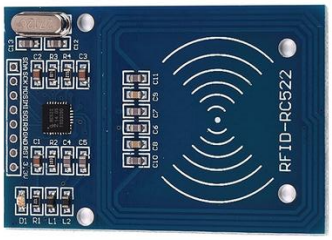


Figure 4.23: RFID reader

- DPDT Switch

Double Pole Double Throw (DPDT) Switch is two switch units in one. Figure 4.24 shows the schematic diagram of the switch.

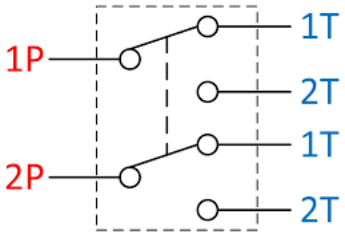


Figure 4.24: DPDT Switch

Each switch has three pins, the center pin which is the Common pin (or Pole), the other two pins are the Throw pins. Hence, the name Double Pole (two Poles as in two switch units) and Double Throw (two Throw pins per switch unit). Each Throw pin is a state that the Pole pin can switch to.

Our system only uses one switch unit to switch between zones. High (T1 supplied with 5V) being one zone, Low (T2 supplied with 0V) being the other zone and the switch (P1) switches between them.

4.5. Software design

The software solution consists of a mobile application, website and cloud service. Android studio is the software that was used to build the mobile application. For the cloud service we selected Firebase as it has many important features such as having a real time database. Moreover, Firebase provides APIs that supports Android and web development and that makes it easier to integrate them. In addition to that, the FirebaseArduino library is a library that allows the Arduino to communicate with the Firebase database directly without the need to have a server in between. For front-end web development, Vue.js, HTML, CSS and JavaScript are used and for the back-end development a Firebase real time database is used.

4.5.1. Structural model

In this section we present the class diagram for the system.

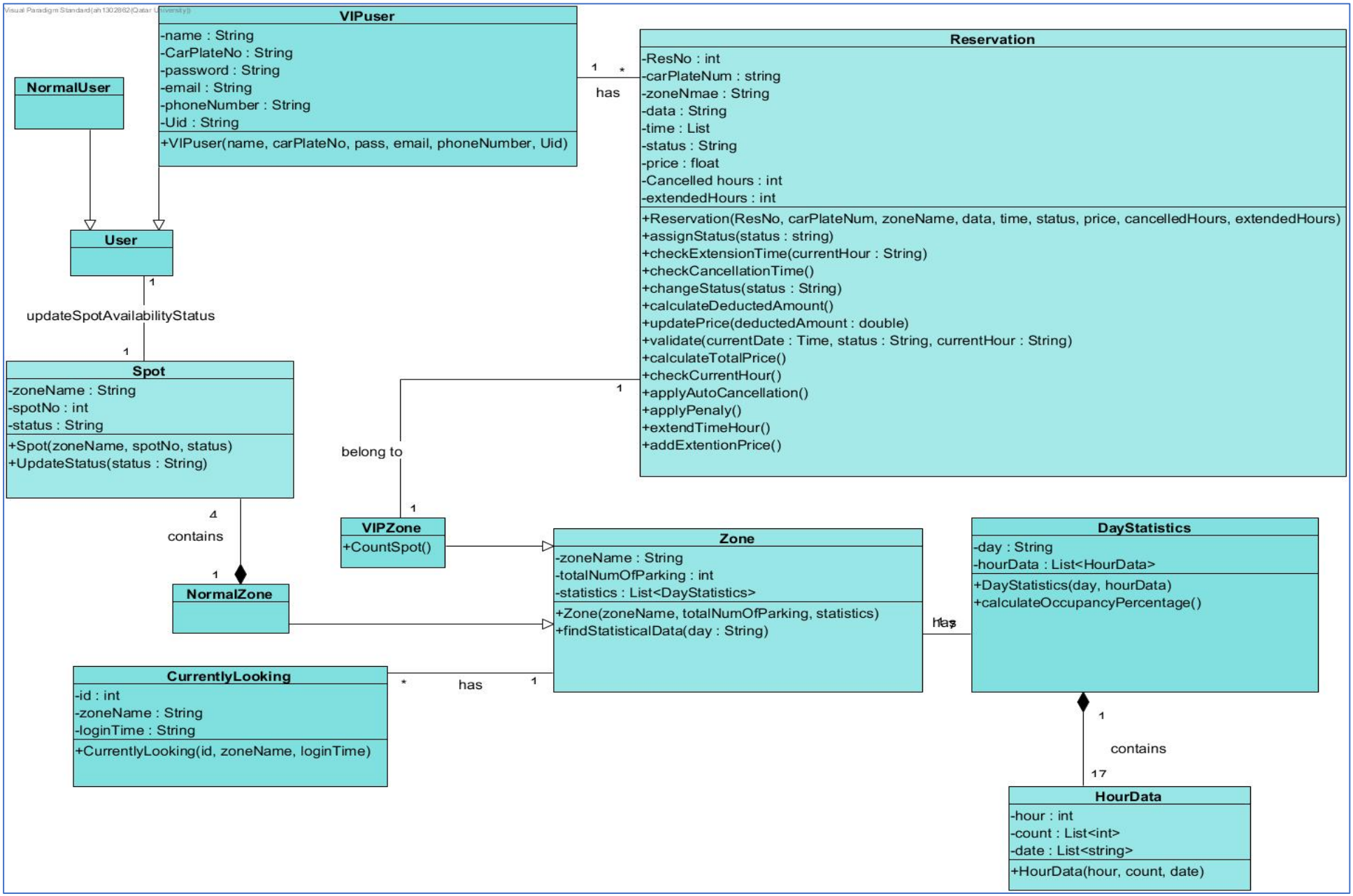


Figure 4.25: Class diagram

4.5.2. Behavioral model

In this section we present the sequence diagram for each use case.

- **Sign Up**

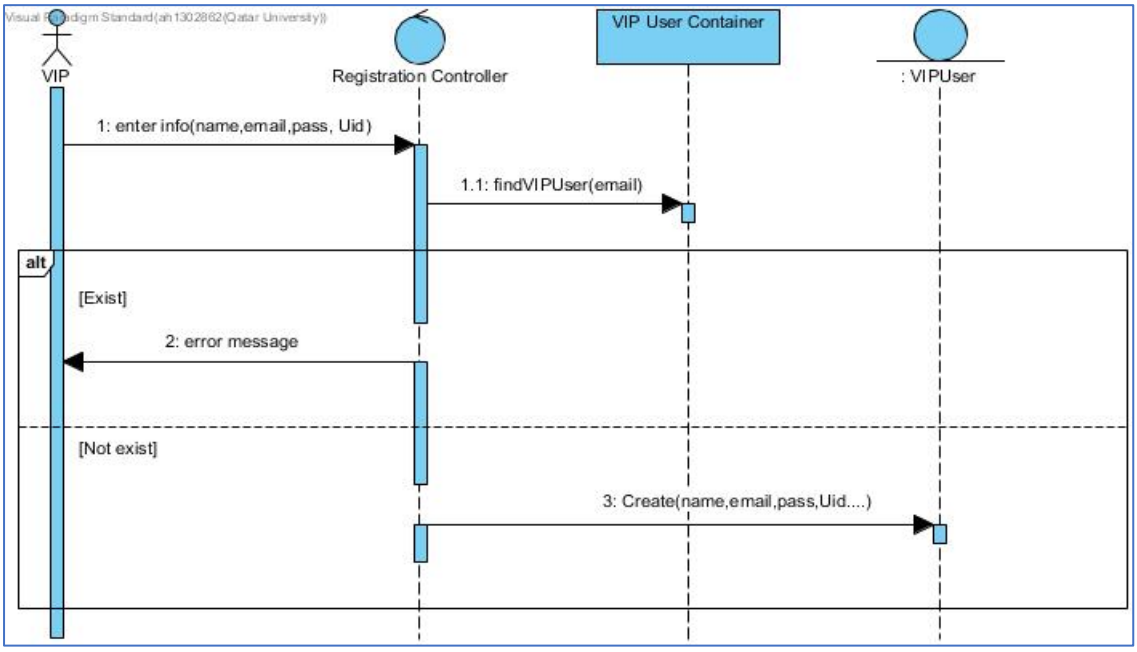


Figure 4.26: Sign up sequence diagram

- **Sign In**

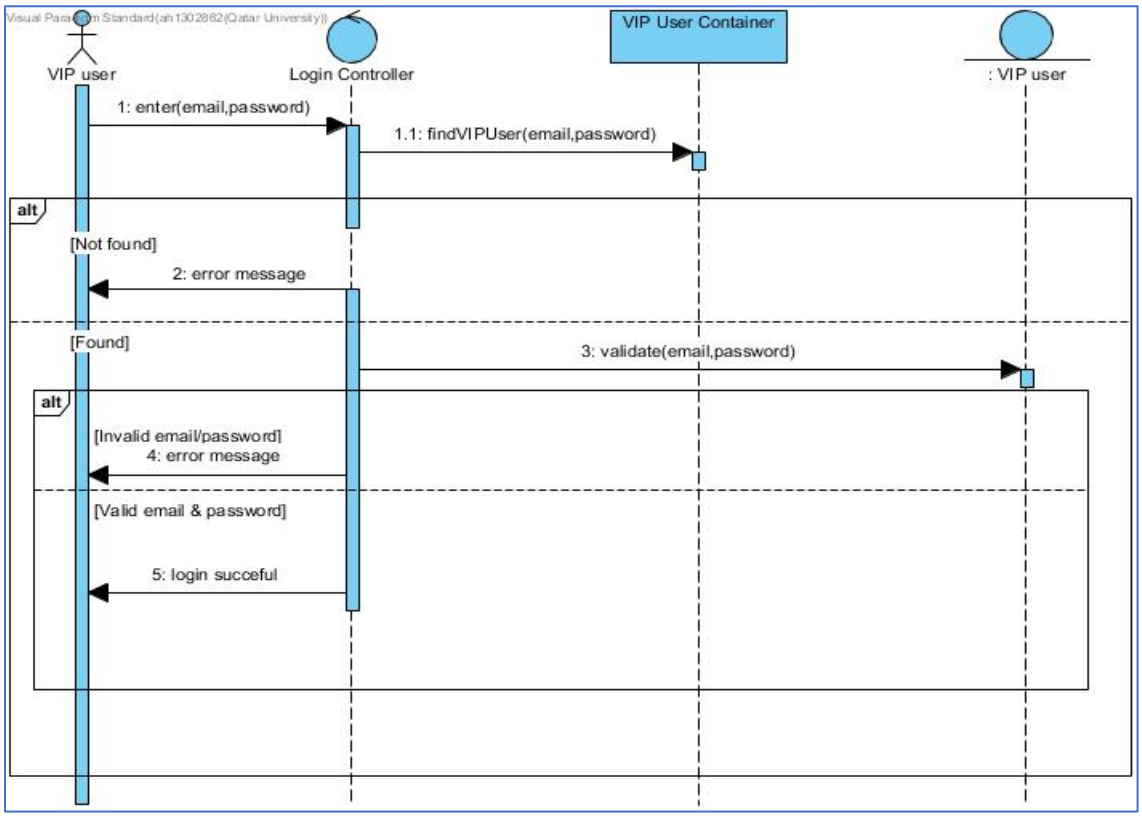


Figure 4.27: Sign in sequence diagram

- **View Parking**

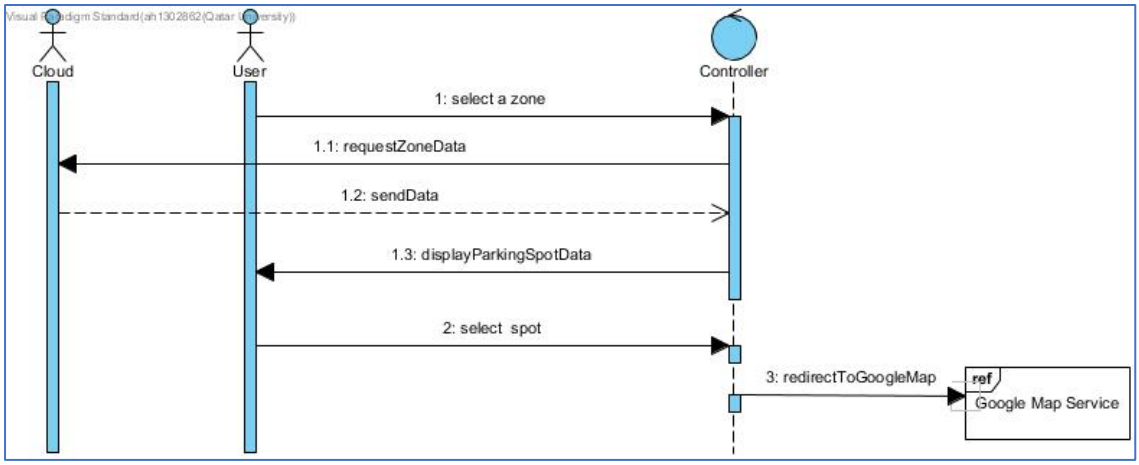


Figure 4.28: View parking sequence diagram

- **Update Parking Availability**

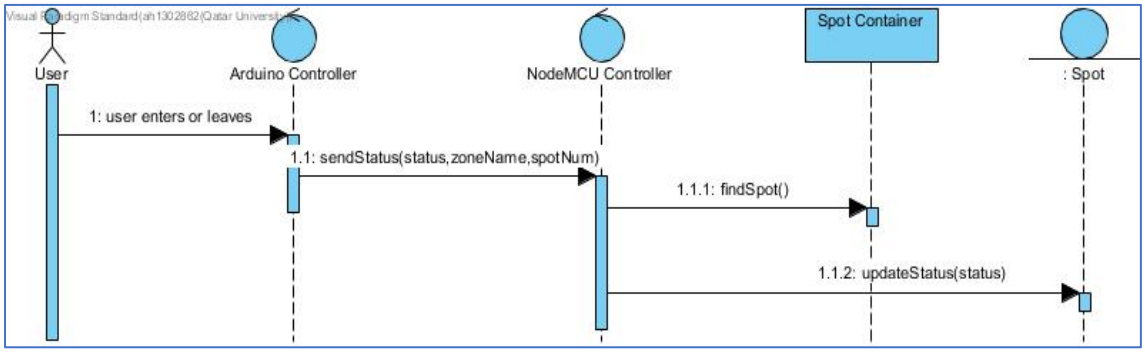


Figure 4.29: Update parking availability sequence diagram

- **Reserve Parking**

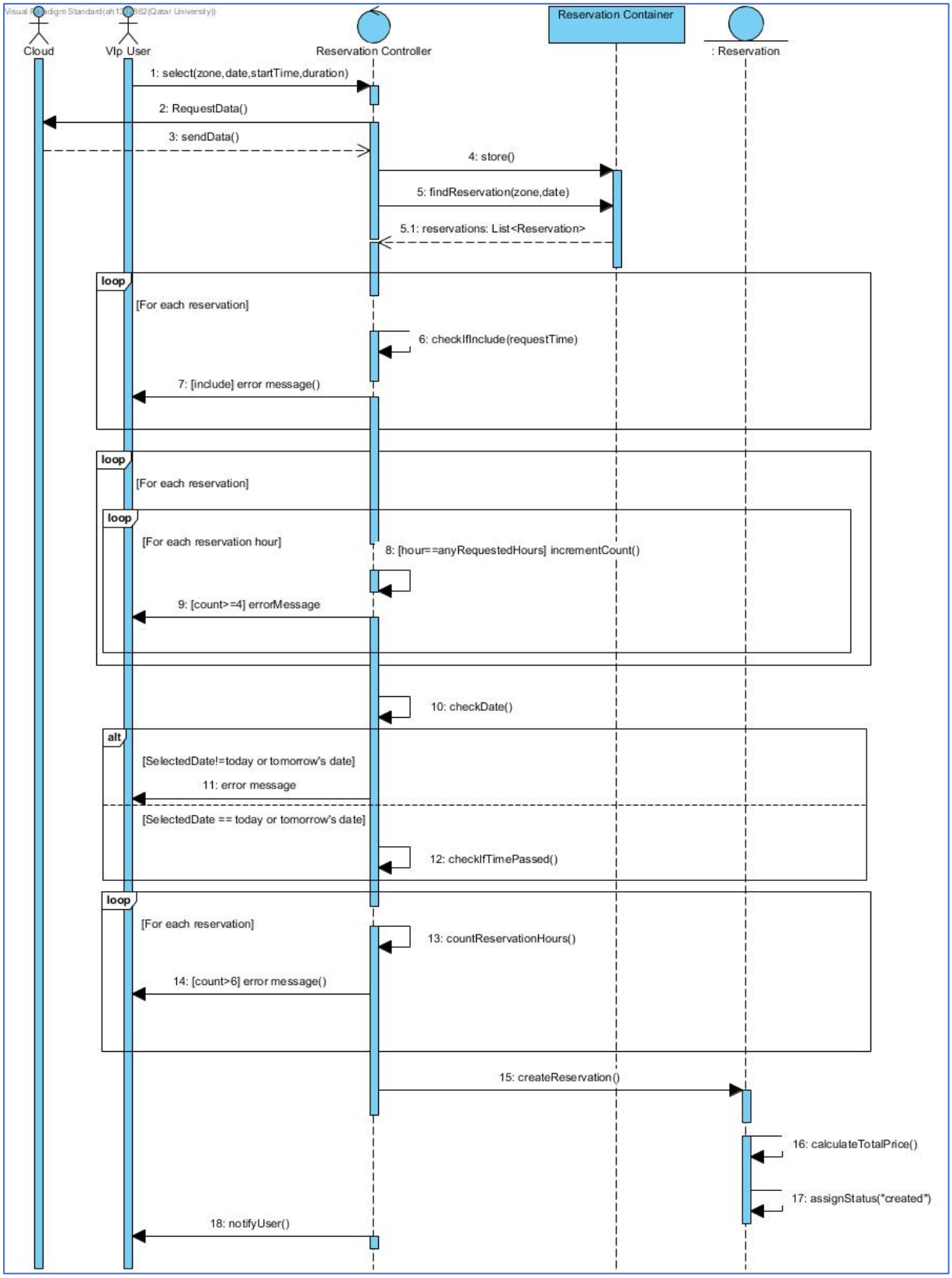


Figure 4.30: Reserve a parking

- **View Reservation**

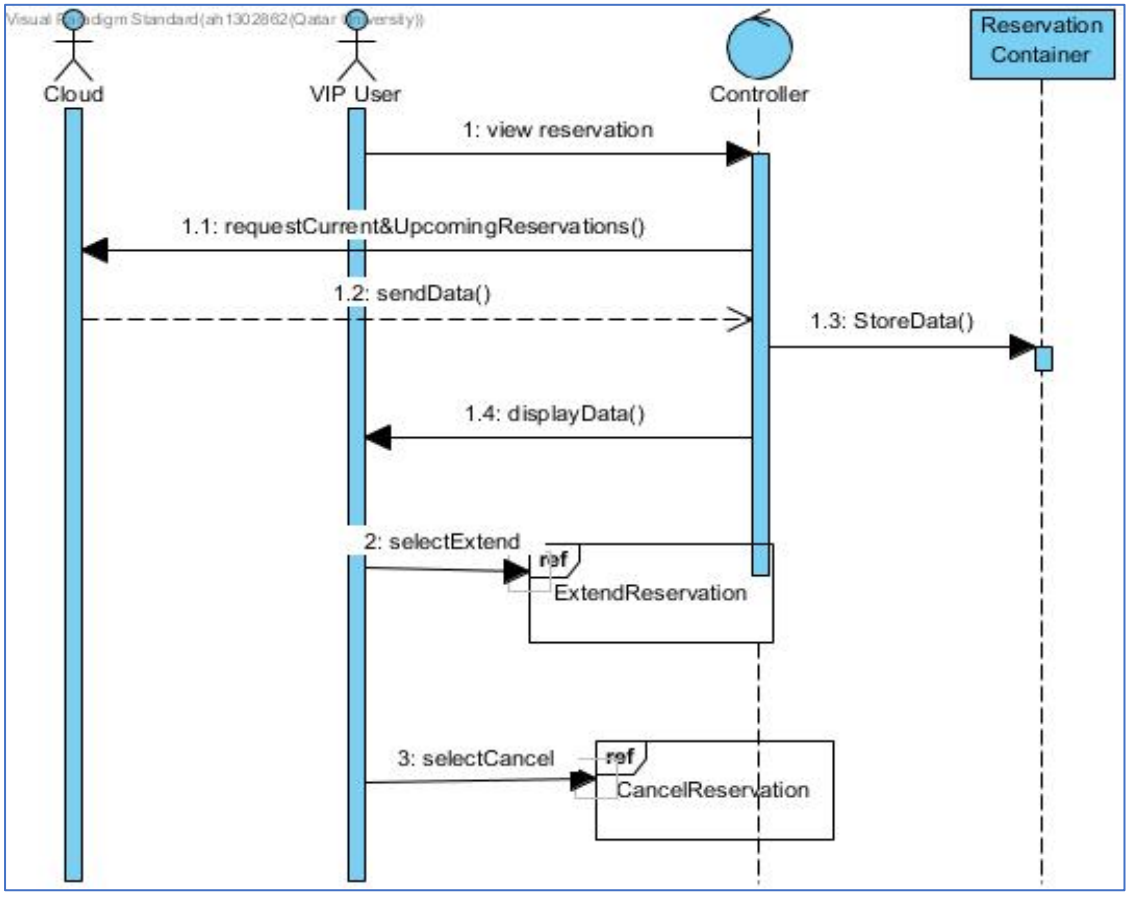


Figure 4.31: View reservation sequence diagram

- **Cancel Reservation**

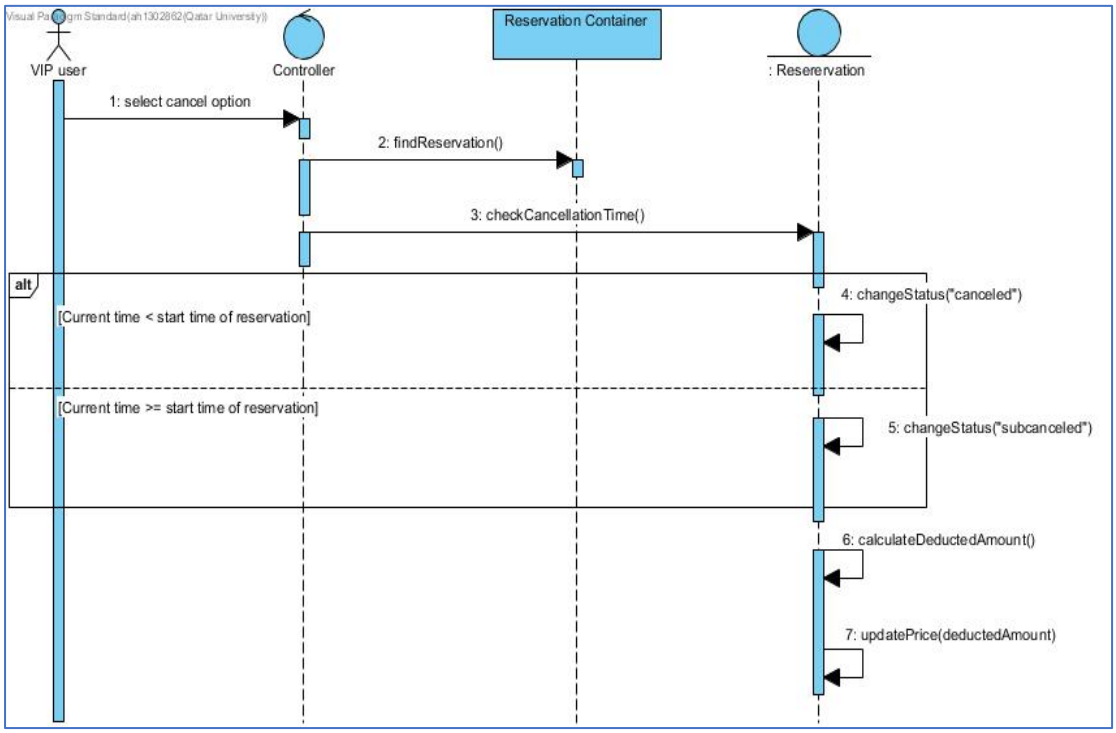


Figure 4.32: Cancel reservation sequence diagram

- **Extend Reservation**

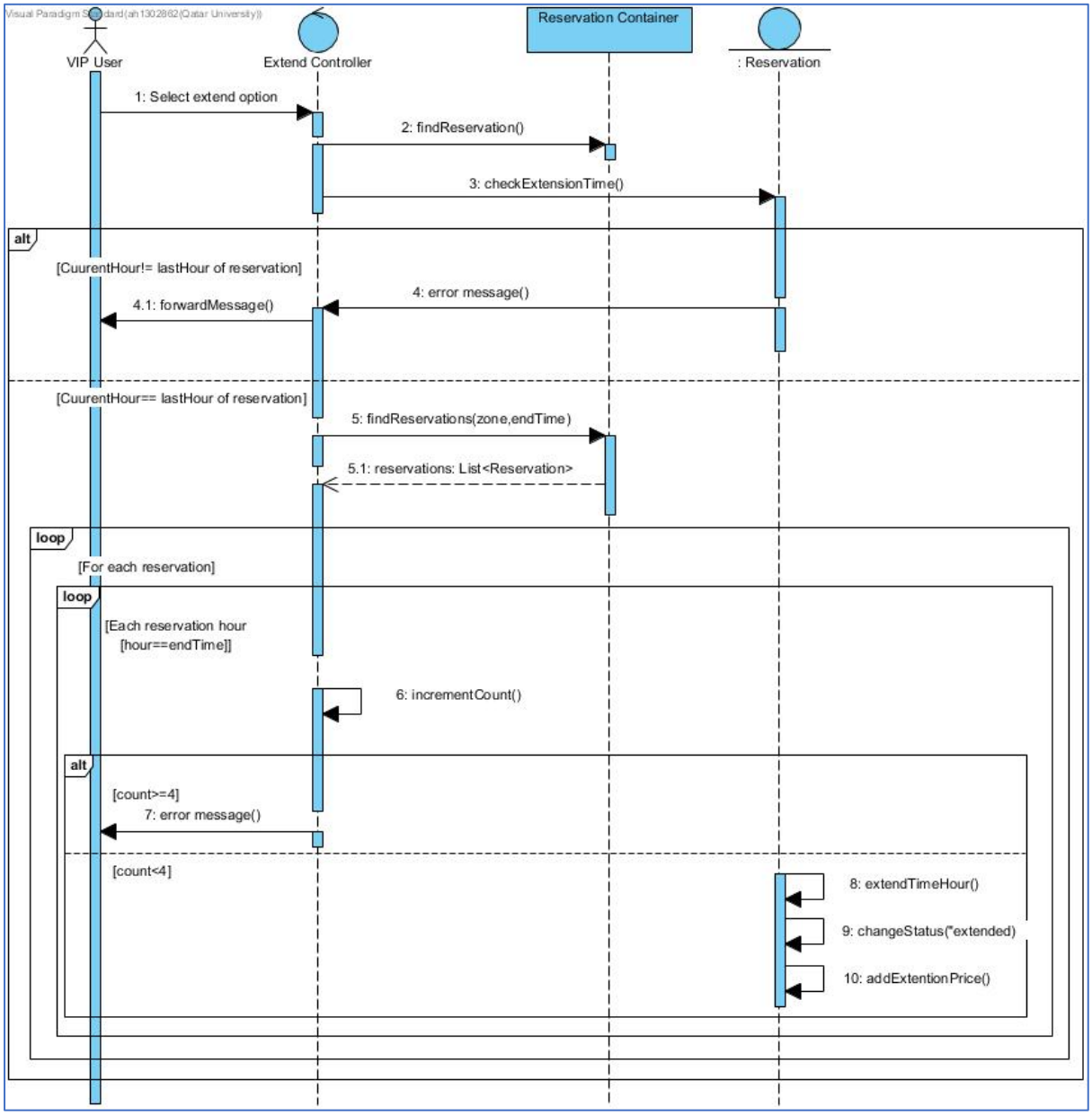


Figure 4.33: Extend reservation sequence diagram

- **Check In**

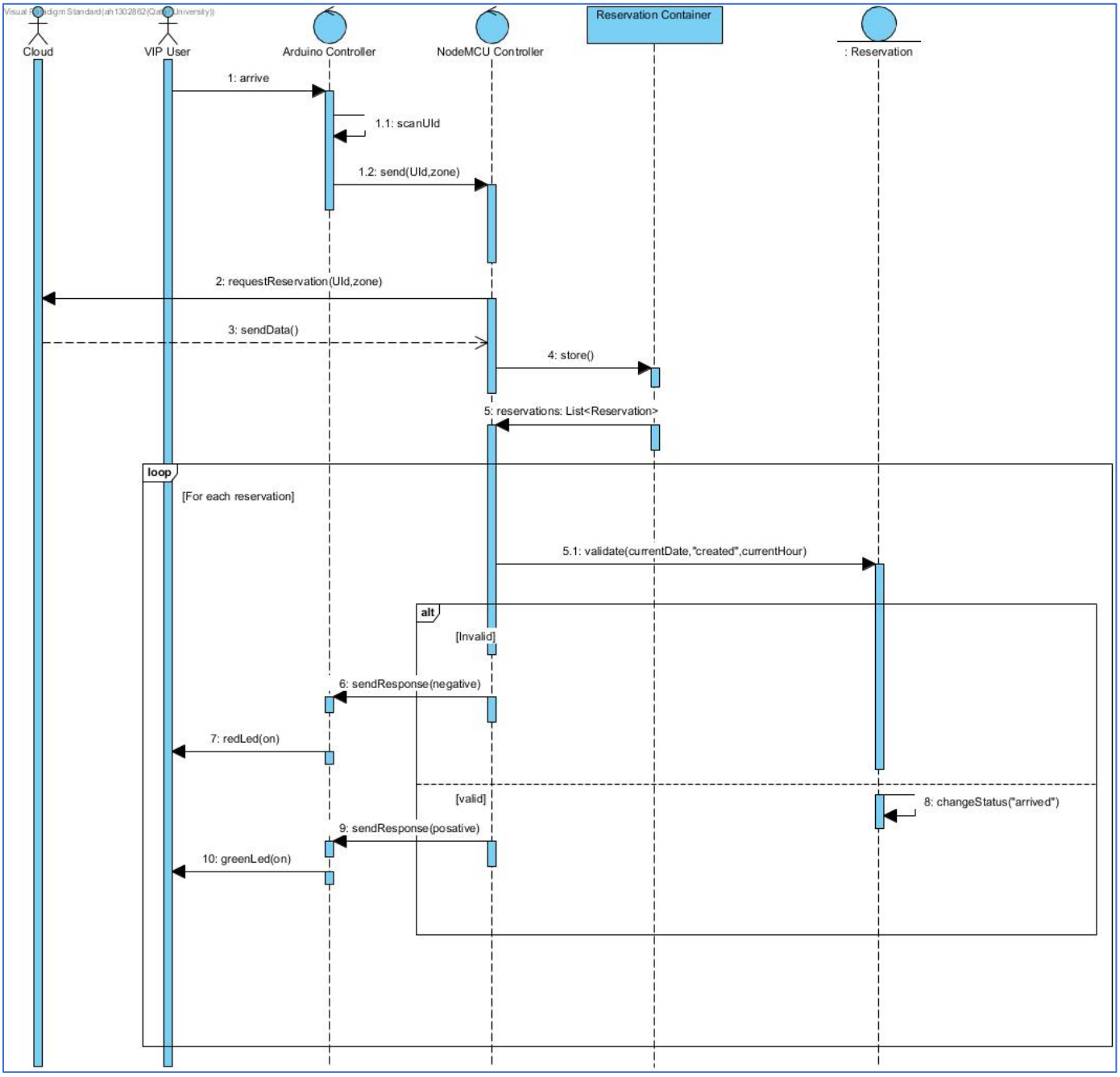


Figure 4.34: Check in sequence diagram

- **Check Out**

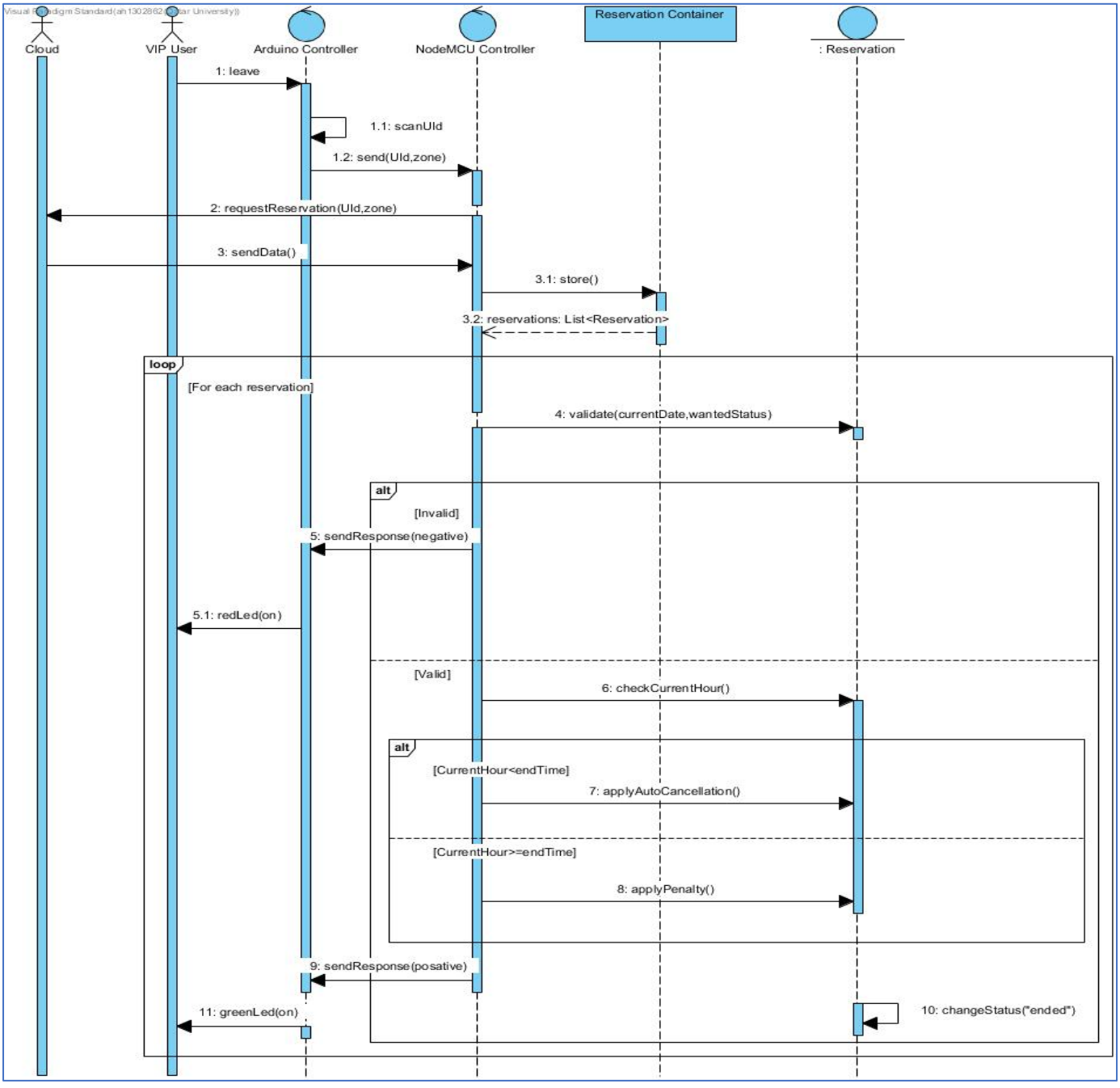


Figure 4.35: Check out sequence diagram

- **Request Car Care**

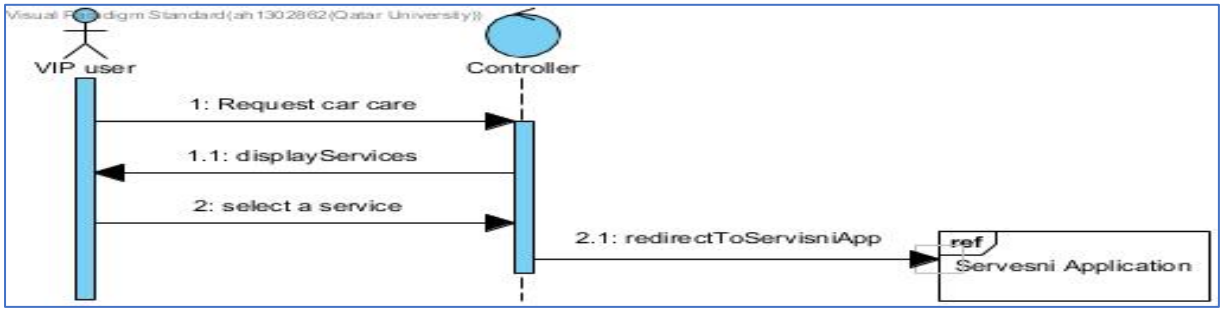


Figure 4.36: Request car care sequence diagram

- **Show Current Occupancy Trend**

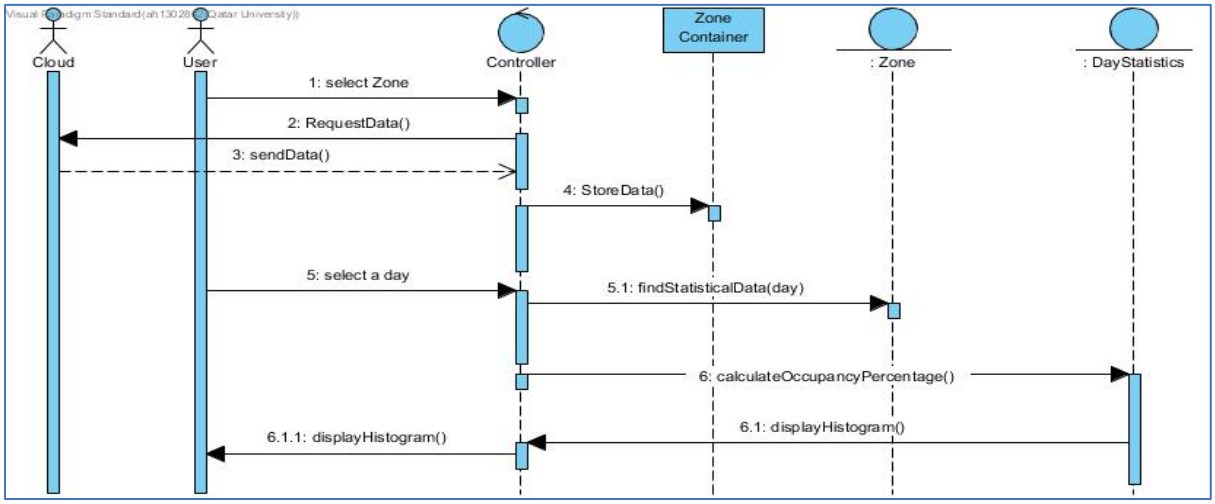


Figure 4.37: Show current occupancy trend sequence diagram

4.5.3. Database design

In the database design, the system stores information about 5 main components which are VIP User, Reservation, Parking area (Zone), Spot and Currently looking. The following figure illustrates the overall Entity Relationship diagram for the system:

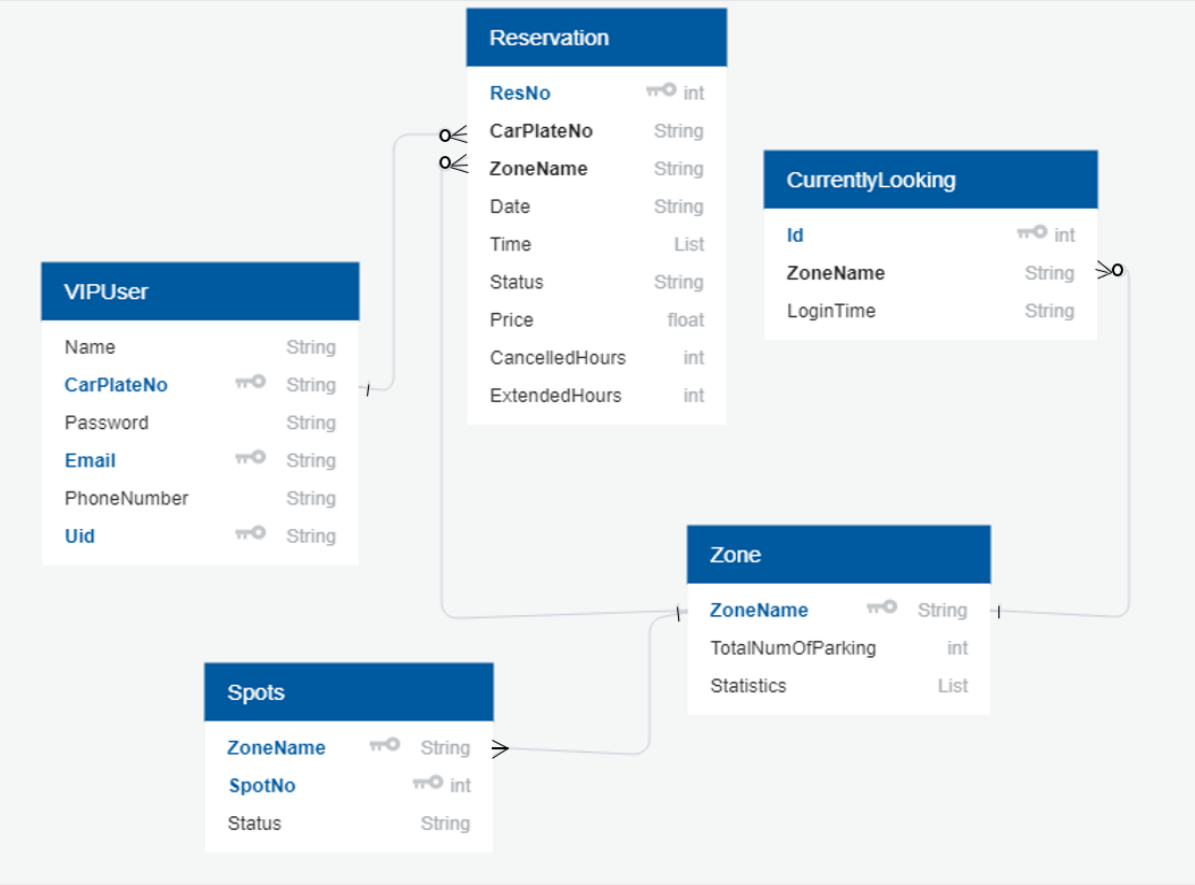


Figure 4.38: ER diagram

From the above ER diagram, we can figure out the relationships that exist between each two entities:

- VIPUser has zero or many Reservation.
- Each Reservation belongs to one VIPUser.
- Each Reservation is associated with one Zone.
- Zone is associated with zero or many Reservation.
- Zone has zero or many CurrentlyLooking.
- Zone has many Spot.
- Each CurrentlyLooking belongs to one Zone.
- Each Spot is assigned to one zone.

In Figure 4.38, an ER diagram is used to help visualize the relations between entities. However, Firebase Realtime Database that the system uses does not follow the ER relations. Instead, the data is stored in JSON format. Basically, the entire database is a big JSON tree with multiple nodes.

JSON syntax is derived from JavaScript object notation syntax:

- Data is in name/value pairs
- Data is separated by commas
- Curly braces hold objects
- Square brackets hold arrays (lists)

The following example illustrates how a reservation object is stored in Firebase Realtime Database

{

"cancelledHours" : 0,

"carPlateNo" : "123",

"date" : "2019-04-13",

"extendedHours" : 0,

"price" : 5,

"resNo" : 841,

"status" : "ended",

"time" : [17, 18],

"uid" : "97 D1 34 83",

"zoneName" : "CBAE Female & Male Zone"

}

4.5.4. User interface design

This section shows the initial mock ups of the user interface design. The actual implemented design is shown in section 5.4 and section 6.1.3.

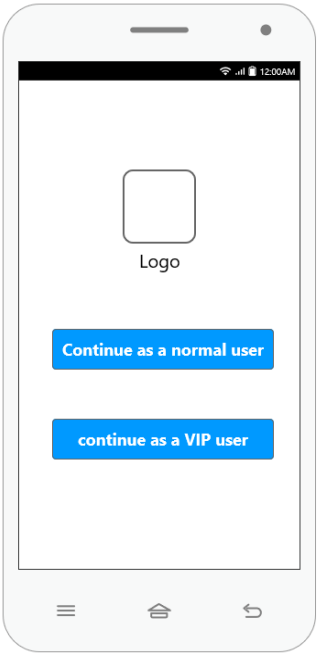


Figure 4.41: Home page

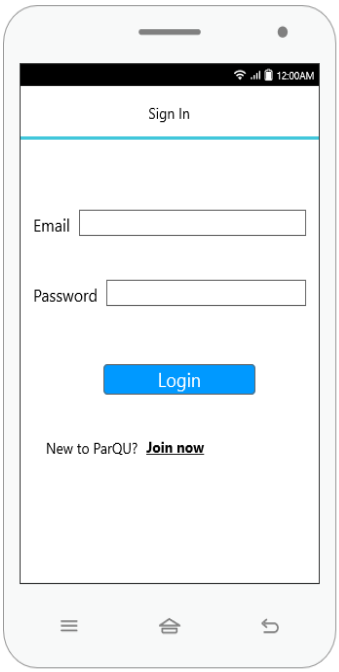


Figure 4.40: Sing in page

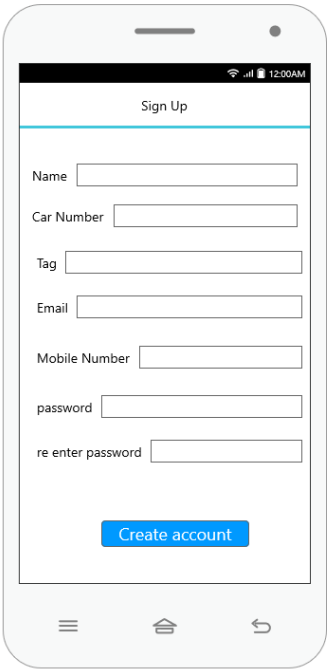


Figure 4.39: Sign up page

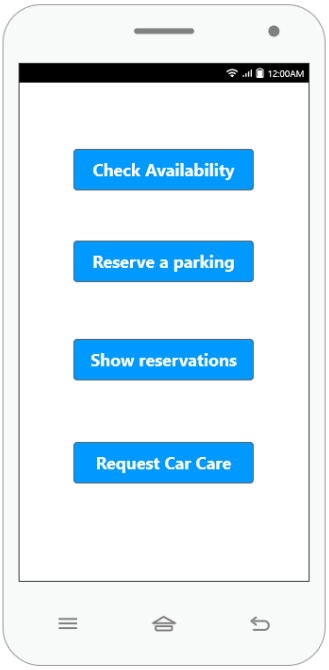


Figure 4.44: Services page

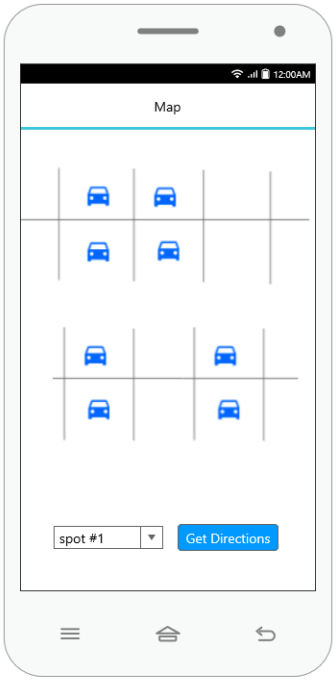


Figure 4.43: Check availability page

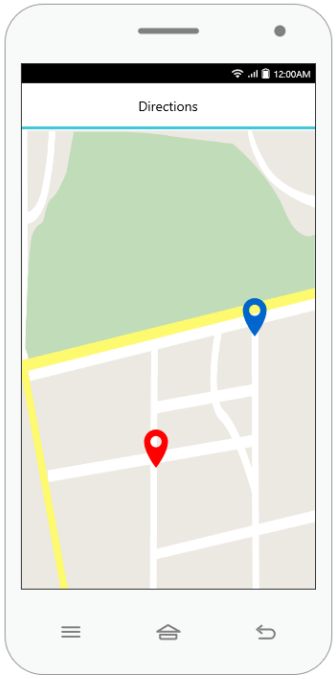


Figure 4.42: Google map page

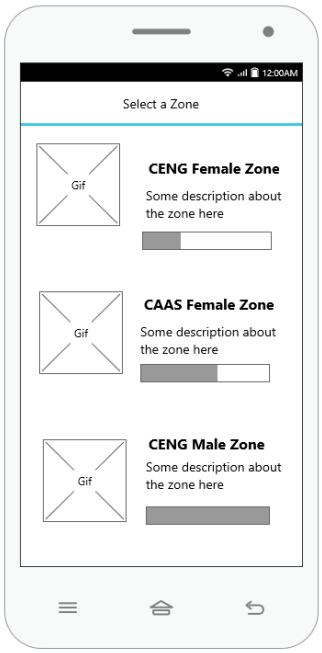


Figure 4.47: Zones page

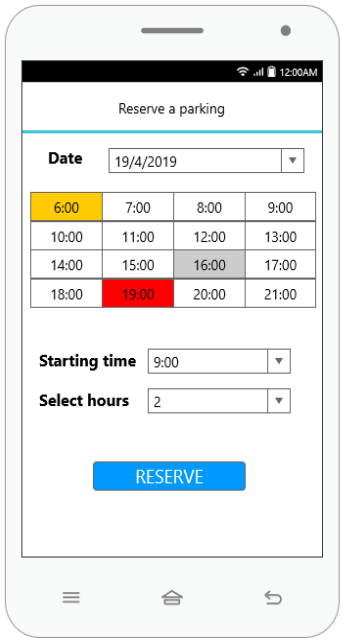


Figure 4.46: Reservation page

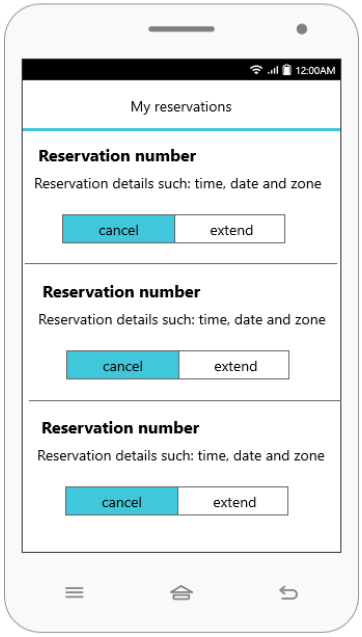


Figure 4.45: My reservation page

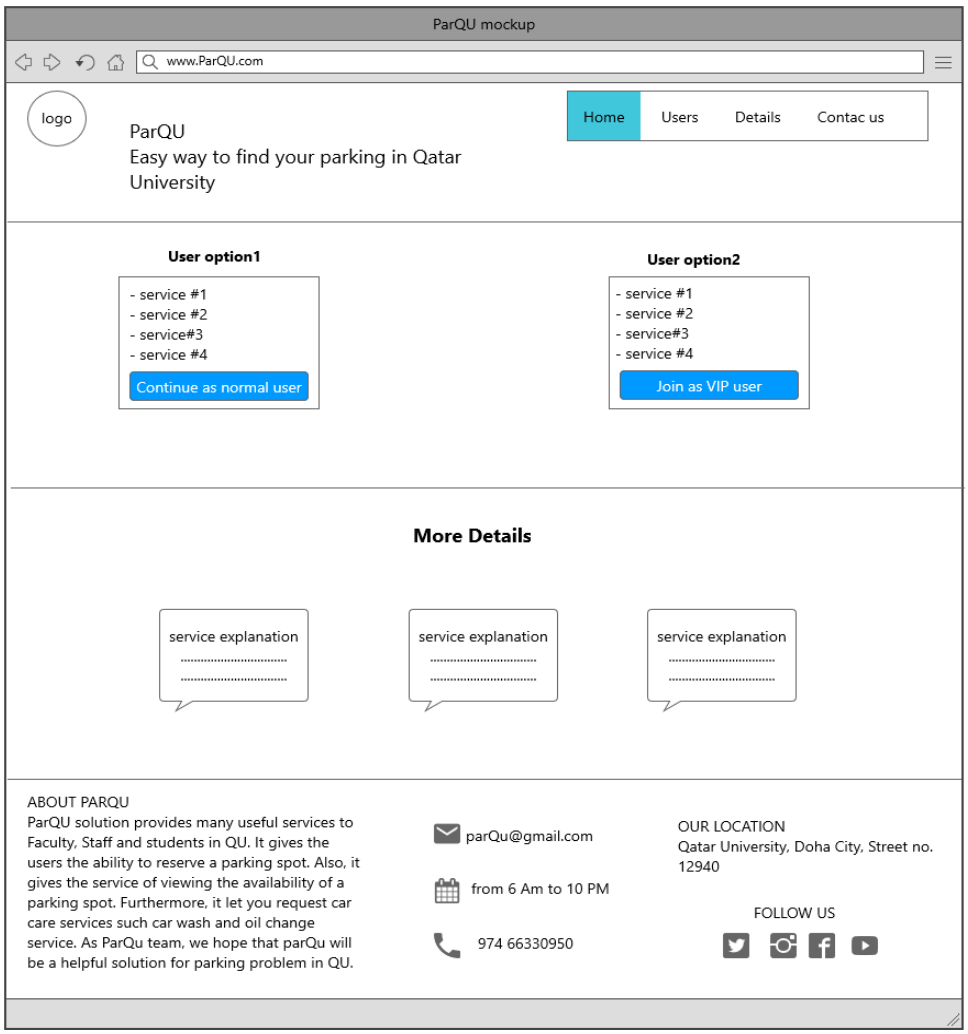


Figure 4.48: Home page

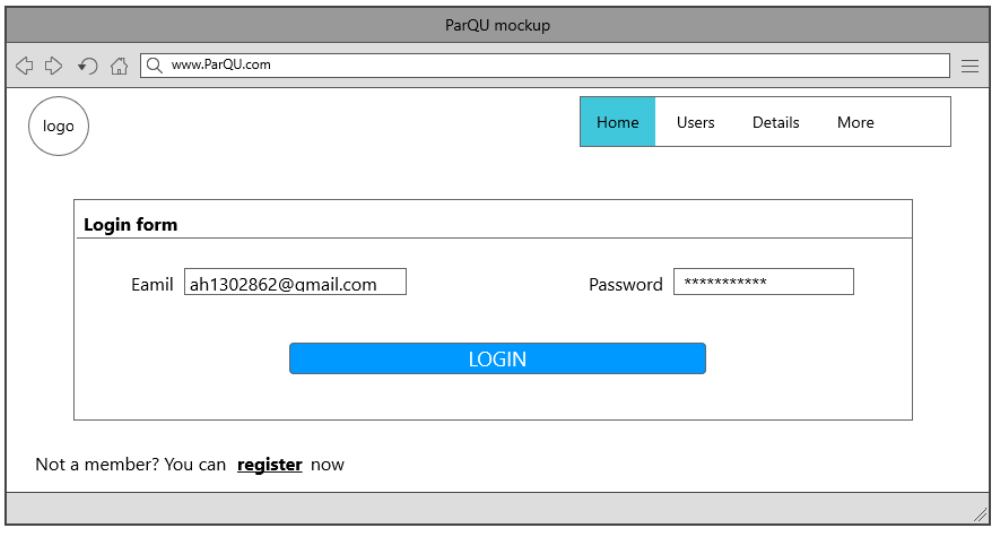


Figure 4.49: Login page

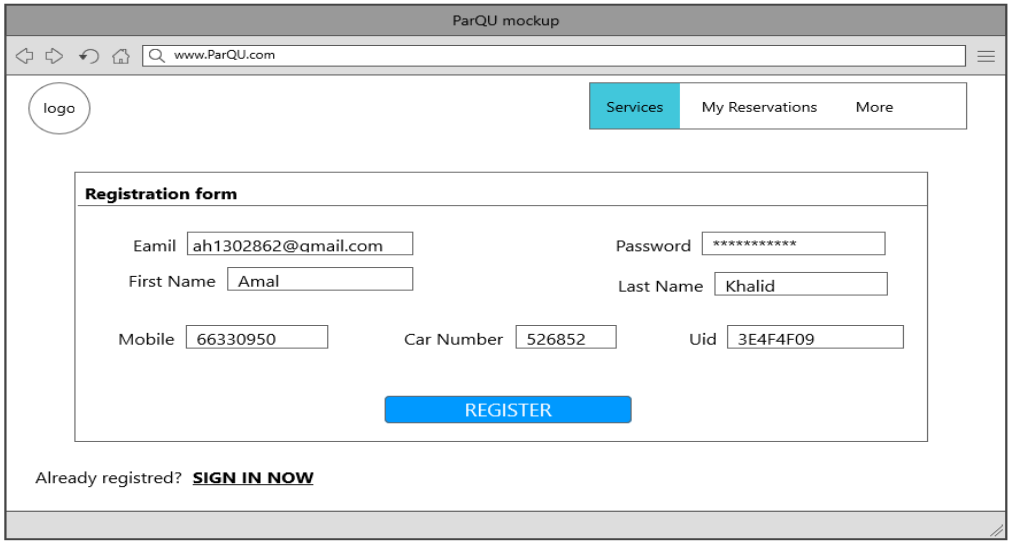


Figure 4.50: Sign up page

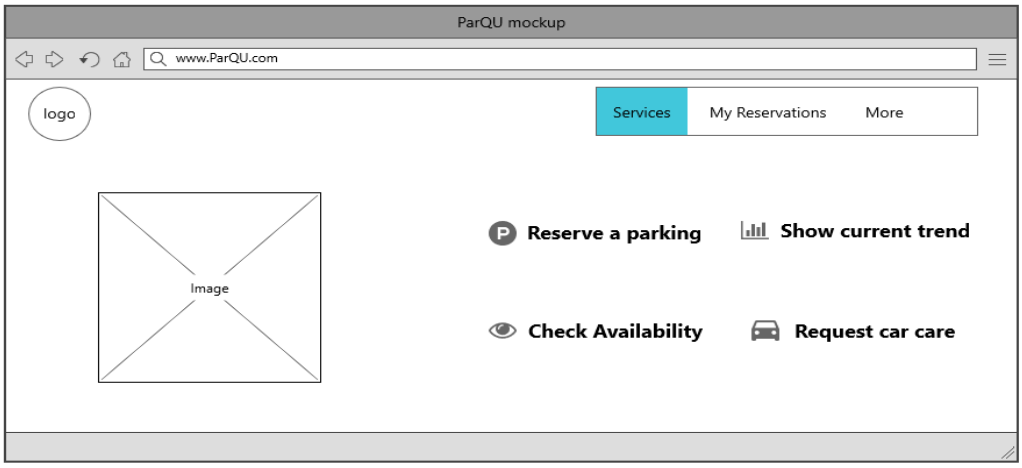


Figure 4.51: Services page

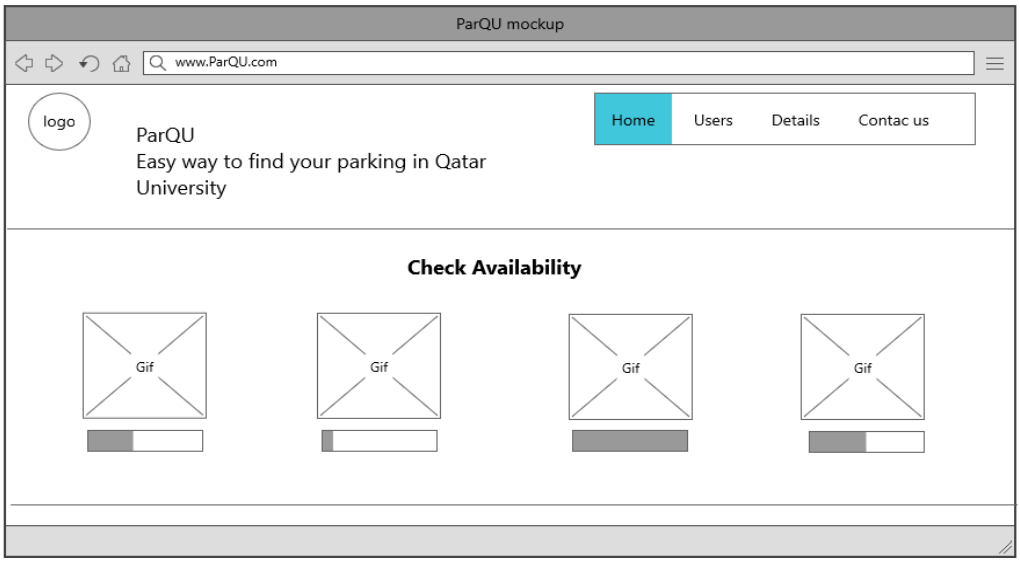


Figure 4.52: Zones page

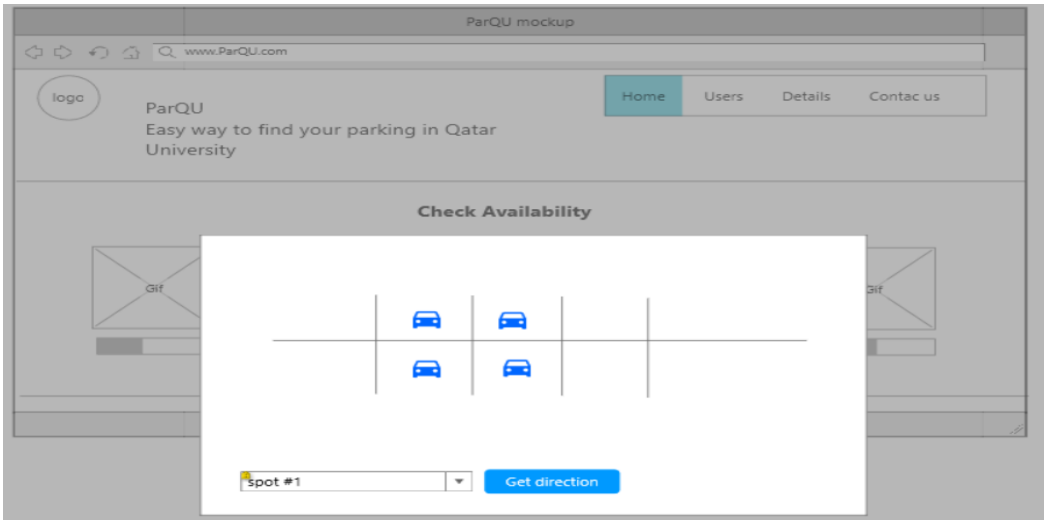


Figure 4.53: Check availability page

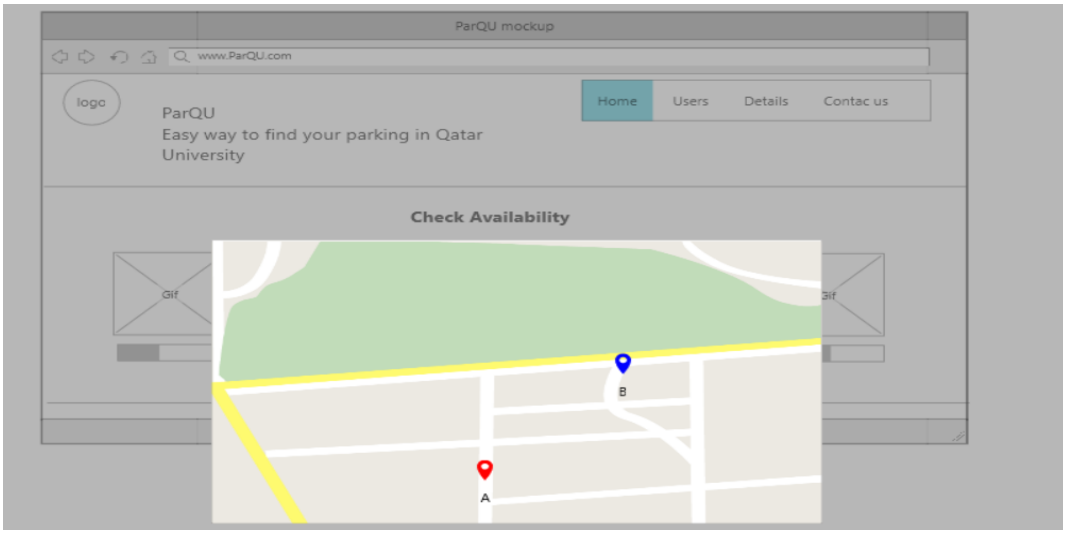


Figure 4.54: Map page

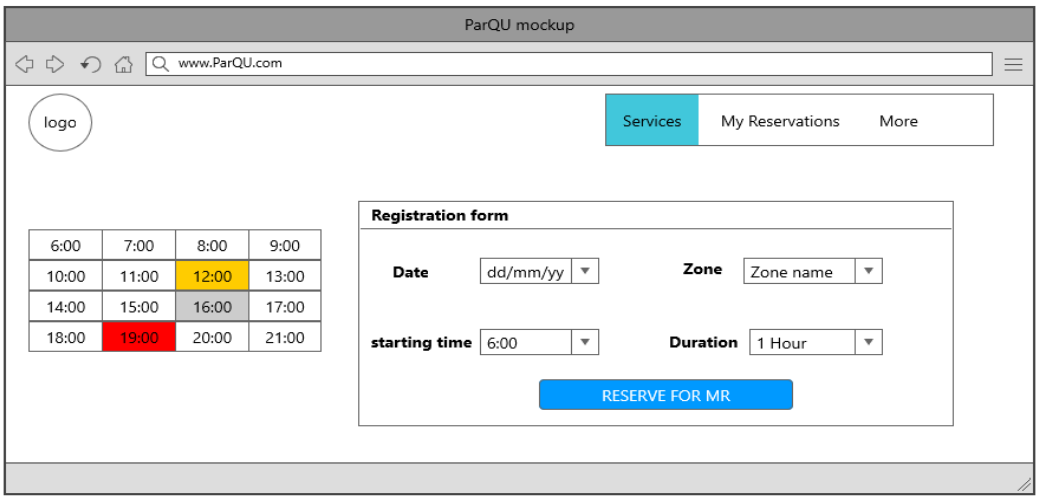


Figure 4.55: Reservation page

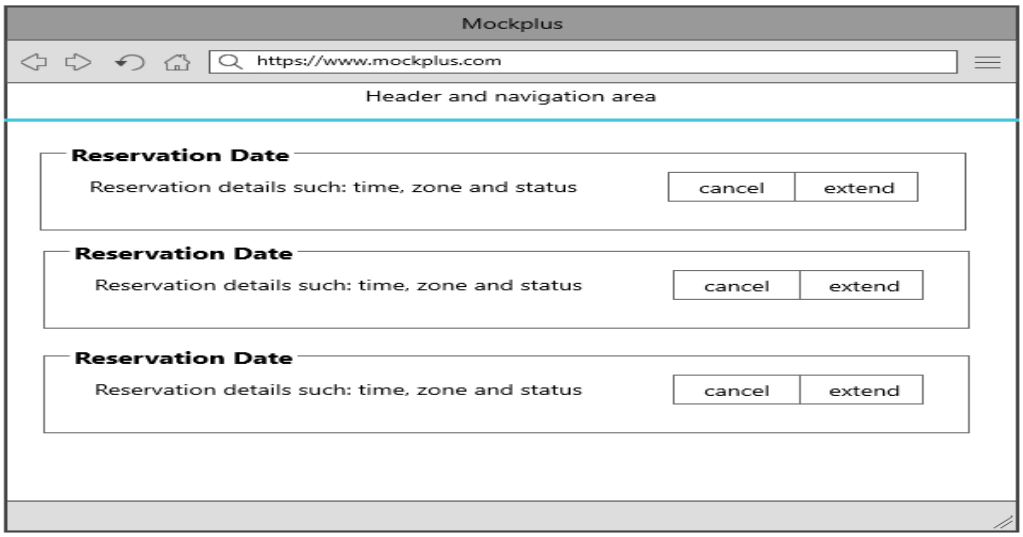


Figure 4.56: My reservation page

4.5.5. Design pattern

The design pattern our system follows and suits our implementation the most is the Model View Controller (MVC). The MVC pattern divides our software part into three main components:

- **Model**: Includes the classes that represents stored data. The Model can only see and interact with the Controller.
- **View**: Includes the interface that the user can view and interact with
- **Controller**: Includes the system logic and interacts with both the View and Model components by retrieving inputted data from user and passing it to the model after processing any required logic with it.

The MVC structure explains the main steps our system follows. The user interacts with the user interface of our system (application or website) and whenever the user requests to input or output a certain data from the view, the view sends that request to the controller along with any inputted data. Afterwards, the controller processes the request and retrieves the needed data from the model, and then sends the appropriate output (results) back to the view to be shown to the user.

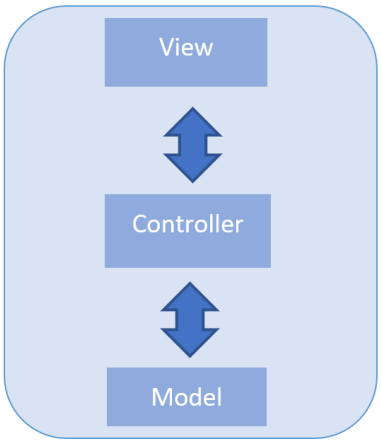


Figure 4.57: MVC

Advantage of using this pattern:

- Separation of Concerns:
  The MVC pattern separated our system into components that are independent from each other, this is called “separation of concerns”. A failure can easily be found because of the components independency, and an edit can be done on any of the components without disturbing other components.
- Re-usability and Flexibility:
  The MVC pattern can increase the scalability of our system, this is because a function can easily be added or changed in a component without changing other components. Hence, our system is easily flexible to scalability. Additionally, different component of our system can be reused in other systems to give the same service.
- High Cohesion and Low Coupling:
  The MVC model automatically makes the system have high cohesion and low coupling. This is because each component in the system must be related in functionality and serve the system with a specific job, which makes the functions in one component highly cohesive. Also, the different components in the system are lowly coupled, meaning changing one component, whether a major or minor change, does not interrupt the work of other

5. Implementation

In this section, we provide the detailed implementation of ParQU services and features. First the Firebase connection setup and commands are explained from the hardware’s, application’s and website’s point of views. Afterwards, we explain the detailed steps of our implementation in each of these three parts. Throughout our implementation process, the Scrum cycle (explained in section 3.1) was followed. Senior project 1 requirements were improved, and new features were added to the system with the Scrum process taken into consideration. With each Sprint we completed a task, solved a certain problem or came up with new ideas.

5.1. Setting up Firebase database

The first step in our implementation was to set up the firebase and to do that we first need to have a Firebase project in Firebase cloud. We have first created a Gmail account and then logged in to Firebase cloud by that account. Then we created a Firebase project and connected our hardware, application and website with it as explained below.

5.1.1. Connecting hardware (NodeMCU) to firebase cloud

This subsection describes the details of connecting the NodeMCU module to the firebase cloud database system.
In order for the NodeMCU module to issue commands (information retrieval or update) to the firebase cloud, the module must first be connected to the internet via a Wifi access point. Communication with the Firebase system was done using the publicly available library FirebaseArduino.h that supports clients using C++, which is the case with NodeMC. This library was first configured using the steps below:

1. In Firebase Database Rules, the read and write conditions are changed to true.

2. In the code, we add the database link 'FIREBASE_HOST' and its secret key 'FIREBASE_AUTH' for authentication (found in the database settings)

```
#define FIREBASE_HOST "fir-auth-45665.firebaseio.com"
#define FIREBASE_AUTH "oYCggxTfYvvMEwPoQN2vM59ZzTX2Lt2A7KFBT31U"
```

3. Then Firebase connection is started with

```
Firebase.begin(FIREBASE_HOST, FIREBASE_AUTH);
```

Once configured, the NodeMCU can start issuing commands to the firebase cloud server. FirebaseArduino.h functions can insert, update, append, retrieve or delete JSON values or objects. However, the project's hardware only handles three cases which are to update a value, retrieve a value and retrieve a whole node as an object. Below we detail the usage of these cases.

FirebaseArduino gets or sets/updates a value with a path. This means that a nested value or a complete node in the JSON tree can be retrieved or updated by passing the path as an argument to the function. For example the path below is the status of the first spot in CENG Female Zone.

```
pathValue = "spots/ -LZaE7RMP-v3D7gQ3-eb/status";
```

Or in the case of a node, the path would be the node's key

```
pathNode = "reservations"
```

Afterwards, FirebaseArduino functions are used as follows

Table 5.1: FirebaseArduino functions

	Command	Description
Retrieve a value	status = Firebase.getString(pathValue);	Retrieves a string value located at pathValue
Update/insert a value	Firebase.setString(pathValue,"not available");	Updates a string "not available" at pathValue location. If pathValue does not exist, the function creates the path and inserts "not available"
Retrieve a node	FirebaseObject nodeReserv = Firebase.get(pathNode); JsonObject& reservations = nodeReserv.getJsonVariant();	Retrieves a node located at pathNode. The function returns a FirebaseObject instead of a JSON structured object. To be able to go through the object as a JSON, FirebaseObject is converted to a JsonObject (with ArduinoJson.h) using the function getJsonVariant() in FirebaseArduino.

5.1.2. Connecting android application to firebase cloud

This subsection describes the details of connecting the Android application to the firebase cloud database system. We went through several configurations' steps in order to be able to read and write from the database.

Configuration steps:

1. A google-service.json file which is Firebase configuration file was given when the Firebase project was created to be added to the Android project. To enable Firebase products in our

app, we added the following google-services plugin to build.gradle file which is a file used by the android SDK to contain project configuration options:

- apply **plugin**: **'com.google.gms.google-services'**

2. The dependency for Real-time Database was also added to the build.gradle file:

- dependencies {
implementation **'com.google.firebase:firebase-database:16.0.4'** }

3. We configured the Firebase Database Rules. To be able to send and receive data to the database we defined the database rules of reading and writing as true which allows to read and write to it.
4. After doing all the above configuration, we can start reading or writing to the database.

Retrieving data from Firebase Real-time Database:

Data stored in a Firebase Real-time Database is retrieved by attaching an asynchronous listener to a database reference. The listener is triggered once for the initial state of the data and again anytime the data changes. The following table illustrates which listener was used:

Table 5.2: Type of Listener used

Listener	Event callback	Typical usage
ValueEventListener	onDataChange()	Read and listen for changes

Inserting data to Firebase Real-time Database:

The Real-time Database accepts multiple data types String, Long, Double, Boolean, List<Object> to store the data. It also allows us to use custom java objects to store the data which is very helpful when storing model class directly in the database.

For example: to store user information,

- Step1: we created a User model with all properties that we decided to store
- Step2: we got the reference to the 'users' node.
- Step3: we used the reference to generate a unique Id by calling *push()* method which creates an empty node with a unique key.
- Step4: we created a user object
- Step5: we used the generated unique Id in step3 to push the user to 'users' node

Deleting data from Firebase Real-time Database:

To delete data, we simply called removeValue() method on to database reference.

5.1.3.Connecting the website to firebase cloud

To enable reading and writing from and to Firebase database, the following steps were executed:

Configuration steps:

1. After the Firebase project was created, we started with configuration steps by obtaining the config object. The config object is unique identifiers for our Firebase project. In order to get the config object, we did the following steps:

a) Opening the overview page and selecting “Add app”
b) Choosing “Add Firebase to your web app”.
c) Copying the snippet shown below and adding it to the main HTML page.

```
var config = {
  apiKey: "AIzaSyCYgApFzAtXIVgdI4J09RWseIxlqfAS8hA",
  authDomain: "fir-auth-45665.firebaseapp.com",
  databaseURL: "https://fir-auth-45665.firebaseio.com",
  projectId: " fir-auth-45665",
  storageBucket: " fir-auth-45665.appspot.com",
  messagingSenderId: "777762792693",
}
```

2. Configuring Nodejs app by performing the steps below:
   a) Installing Firebase JavaScript SDK by running the following commands:

```
npm init
npm install --save firebase
```

   b) Using Firebase module after requiring them from JavaScript.

```
var firebase = require("firebase");
require("firebase/auth");
require("firebase/database");
```

3. Initializing the Realtime database

```
firebase.initializeApp(config);
```

4. Getting a reference to the database service

```
var database = firebase.database();
```

5. Configuring the rules as mentioned in (Section 5.1.2)

6. After doing all the above configuration, we can start reading or writing to the database.

Retrieving, inserting and deleting data in Firebase Real-time Database:

Retrieving, inserting and deleting data in the website have the same logic as application however website uses different type of listeners and functions.

The following table illustrate the type of listener used to retrieve data:

Table 5.3: Type of event used

Event	Event callback	Typical usage
Value	`On()` and `once()`	Read and listen for changes

As shown in the table above, we use the method **`on()`** or **`once()`** for retrieving the data which is taking “value” as the event type. Every time the data changes, the event is triggered, and all the data will be retrieved.

Below are short examples of reading and writing reservation from and to Firebase database:

- Reading reservation from the database

```
db.ref('/reservations').once('value')
.then((reservations) => {
      //Here goes the function
})
```

- Creating a new reservation to the database.

```
var newReserve = ref.push();
newReserve.set({
    carPlateNo: plateNo,
    date: req.body.date,
    price: price,
    status: 'created',
    zoneName: req.body.zones,
    resNo: resNo,
    extendedHours: 0,
    cancelledHours: 0,
    uid: uid
})
```

5.2. Software Tools

The Hardware tools are Arduino IDE and Fritzing. Arduino IDE is used to program both Arduino and NodeMCU, while Fritzing was used to create the hardware Connectivity and Schematic diagrams.

The main libraries used in the hardware implementation are:

Table 5.4: Libraries used in hardware implementation

Library name	Description
FirebaseArduino.h	Explained in Firebase connection section
SoftwareSerial.h	Allows serial communication on digital pins using software. The SoftwareSerial gives the same functionalities as the hardware serial communication (pins 0 and 1). The hardwired serial communication is restricted to one serial communication link and is always on pins 0 and 1. Even though we only use one serial communication link for each module, debugging can only be done through the hardware serial link. Hence, SoftwareSerial is added to allow us to debug on hardware serial whenever it is needed.
ArduinoJson.h	A C++ Json library for Arduino and Internet of Things (IoT). The library allows Arduino and NodeMCU to create JSON structured data (As explained in database schema 4.5.3). In our project, JSON data is used in: • Send/Receive data in Serial with SoftwareSerial.h • Send/Receive data in Firebase with FirebaseArduino.h
NTPClient.h	NTP is “Network Time Protocol” a standard Internet Protocol (IP) that synchronizes time on computers to UTC (Coordinated Universal Time) with the access of internet connection. As the name hints, the NTP

	applied in our project works in a server-client manner. The process of getting time in general is: • Client (NodeMCU) contacts the NTP server • Client gets the current UTC (time stamp) from NTP server • Client applies any local time zone offset (in Qatar UTC +3.00) Afterwards, certain functions in the library are called to get current time and date. For instance, getFormattedDate(), getHours(), getMinutes(), getDay().

Additionally, for android application, we used Android Studio as it is the official IDE for Android development. The language used to develop the entire logic of the application was java. Also, XML (Extensible Markup Language) used as Markup Languages to design the user interface.
For website development, the following table illustrate all tools and frameworks that we have used in different stage while implementing our website:

Table 5.5: Tools and frameworks used for website

Tools, frameworks and technologies	
Front-end Languages / Platforms	
HTML	Hyper Text Markup Language used to build the structure and the content of the website page
CSS	Style sheet language used for the presentation of web pages writing in HTML.
JavaScript	Programming language used for client-side scripting.
Front-end frameworks	
Bootstrap	An open-source CSS framework used to make the website responsive and allow the pages' style to act differently based on the browser size.
Back-end frameworks and Libraries	
Node.js	An open-source, JavaScript run-time environment used to execute server-side scripting.
Express Framework	Node.js framework used to make the HTTP requests such GET and POST requests.
Body-parser	Node.js middleware for handling JSON used to get the body from the request and work on it, and to get the data sent in the body.
Express-handlebars	Express module used to create dynamic front end pages using html.
Path	Node.js module used to set the css,js and images files and everything related to the frontend into public folder and access it from the pages.
App	JavaScript web application development framework used to Create instance form Express

Express-flash	Node.js module for flash messages for Express application. Used to display messages on success and on errors.
Express-session	Node.js module creates a middleware with the given options. We used it to save the session of the user.
Moment	A lightweight JavaScript date library for parsing and formatting dates. Used to make a lot of changes on the displaying, adding time or getting time difference.
Nodemailer	Node.js module used to send e-mail from Node.js
amCharts	JavaScript/ HTML5 charts libraries for websites
Task Runners / Package Managers	
NPM	Pack manager for JavaScript.

5.3. Hardware Implementation

The hardware implementation is divided according to the module and each module consists of two main parts: Arduino and NodeMCU.

5.3.1. Reservation Free Parking Module

As a follow up to the solution section (section 4.2), the schematic diagram in Figure 5.1 shows how the module is wired along with the pin names of the components. The module components are Arduino, NodeMCU, Ultrasonic sensor and the DPDT (Double Pole Double Throw) switch.

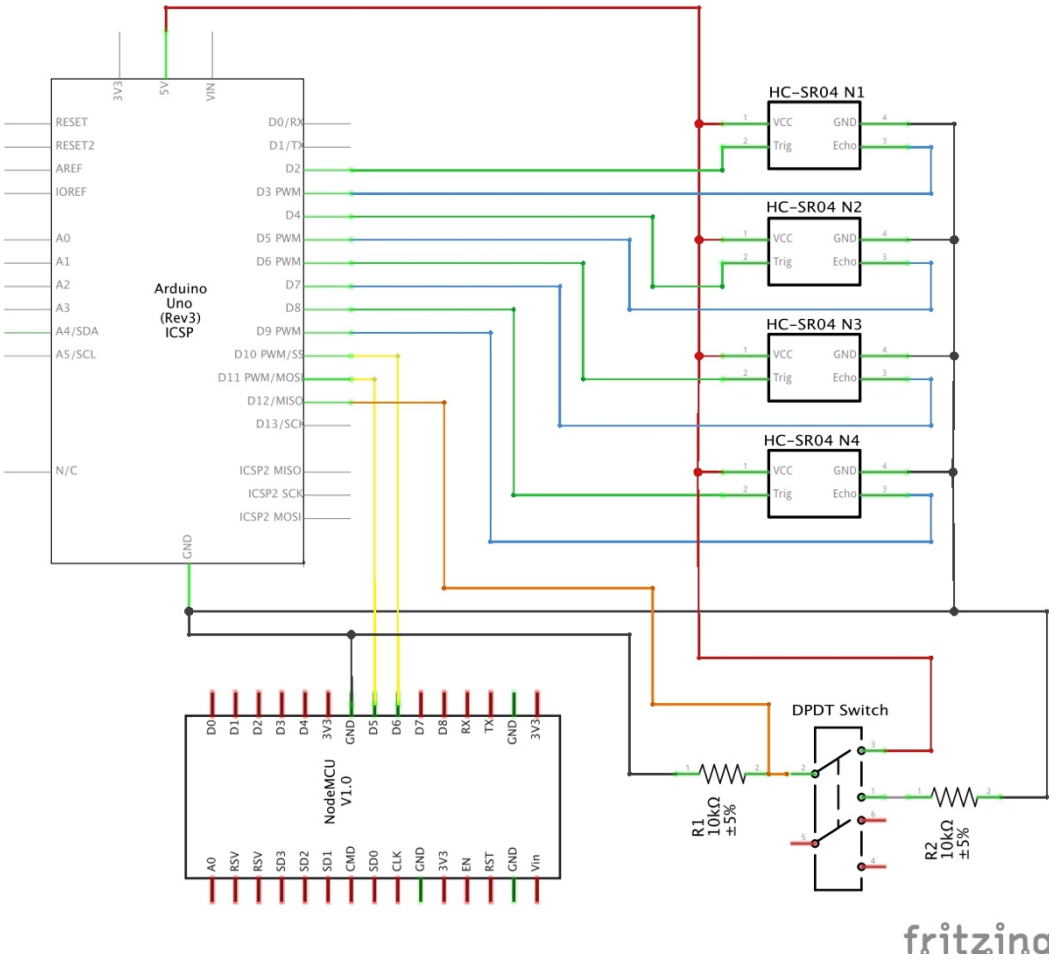


Figure 5.1: Schematic diagram for free reservation parking module

Furthermore, the module's software implementation of Arduino and NodeMCU follows the system's high-level architecture in section 4.2 in terms of the module's working flow. A simplified version is shown below. The module's flow is a one-way flow starting from Arduino to NodeMCU.

Element 1: Arduino

Element 2: NodeMCU

Element 1: Arduino

The first element in the working mechanism of the module is the Arduino. In Arduino IDE, the code is *always* executed repeatedly (a loop part) after setting up the microcontroller when it is turned on (a setup part). The loop in Arduino includes five steps explained below.

- ***Step 1:*** Read the current zone from the switch button
  - Low (not pressed): CENG Female Zone
  - High (pressed): CAAS Female Zone

- ***Step 2:*** Take a reading from the four Ultrasonic sensors
  For each ultrasonic sensor:
    - Step A: The trigger pin is held high for at least 10 us
    - Step B: Wait *till* the echo pin is held high (start of the pulse)
    - Step C: Measure how long the echo pin is held high. The time the echo is held high is the pulse width
    - Step D: Calculate the distance read from the pulse width.
      Distance (cm) = pulse width (us) / 58.0
    - Step E: The distance is checked (Distance can be from 2-400 cm)
      - Distance less than 20 cm: Car is present; new status of the parking spot is "not available"
      - Distance more than 20 cm: Car is not present; new status of the parking spot is "available"
- ***Step 3:*** Check if any of the new readings are different than the old readings (i.e. there is a change in status)
    - Both new and old readings are stored in an array and compared with respect to the index.
      - If new = old: No change in status
      - If new != old: Change in status
- ***Step 4:*** Send changes to NodeMCU serially
    - The data sent to NodeMCU is in a JSON format with the help of ArduinoJson.h library (as explained in the database design section 4.5.3) and is structured as follows:

```
{
    "zoneNumber" : 0,
    "spot1" : 1,
    "spot2" : 3,
    "spot3" : 2,
    "spot4" : 3
}
```

    - Zone Number: could either be 0 or 1
      - 0: CENG Female Zone
      - 1: CAAS Female Zone
    - Spot 1, 2, 3 and 4: Shows the updated status of each parking spot
      - 3: No update in status
      - 2: Status updated to "available"
      - 1: Status updated to "not available"
- ***Step 5:*** Mark the new readings as old readings
    - The old readings are deleted and the new readings are considered as old.

<u>Element 2: NodeMCU</u>
In this module, NodeMCU is responsible for two main tasks. First, to receive the new spots status from Arduino and update it in Firebase. Second, to update Firebase each hour of the current parking status as part of implementing the Current Occupancy Trend available in application and website (explained in sections 4.2 and 5.3). The NodeMCU loop is divided accordingly.

Part 1: Update changed parking spots status

- ***Step 1:*** Receive changes from Arduino serially
    - NodeMCU reads the received JSON format data using ArduinoJson.h library as explained in Arduino's Loop Step 4.

- ***Step 2:*** Check if there is at least one spot with an update.
    - If all the spot values are 3 (i.e. No update), step 3 is skipped
- ***Step 3:*** Update changes to Firebase.
    - All parking spots keys in Firebase are hardcoded.
    - As explained in Firebase section 5.1.1, The new status is updated using the parking spot path passed into FirebaseArduino.h library function setString(path, status)

Part 2: Send current parking spots status for Current Occupancy Trend

- ***Step 1:*** Get current time using NTP client
- ***Step 2:*** Check if an hour has passed since last update
    - 8 AM parking status is read at 8:30 AM. (To get the most efficient reading of the hour)
    - NodeMCU proceeds to step 3, if
        (currentMinutes >= 30) and (currentHourUpdated != true)
- ***Step 3:*** Get current parking spots status from Firebase
    - The parking spot path is passed to the FirebaseArduino.h function getString(path)
- ***Step 4:*** Send new parking spots status to Firebase
    - For each hour in each zone, there is a count and date array, where count is the number of cars parking in that specific hour at the corresponding date in date array. To illustrate
    - To illustrate, below is an example of hour 13 at zone “CENG Female Zone”

Date:	Count:
3: “2019-04-04”	3: 4
2: “2019-04-11”	2: 2
1: “2019-04-18”	1: 4
0: “2019-04-25”	0: 3

      The first line means on 4/4/2019 at 1pm, there were four cars parked at CENG Female Zone.

    - To update the above structure with the new read data
        - Step A: Oldest date and count at index 3 is deleted
        - Step B: Other dates and counts are shifted upwards (i.e. 2 becomes 3, 1 becomes 2 and 0 becomes 1)
        - Step C: New date and count is added at index 1

5.3.2. Reserved Parking Module

The schematic diagram of the module is shown in Figure 5.2. The components of this module are Arduino, NodeMCU, RFID reader and DPDT Switch.

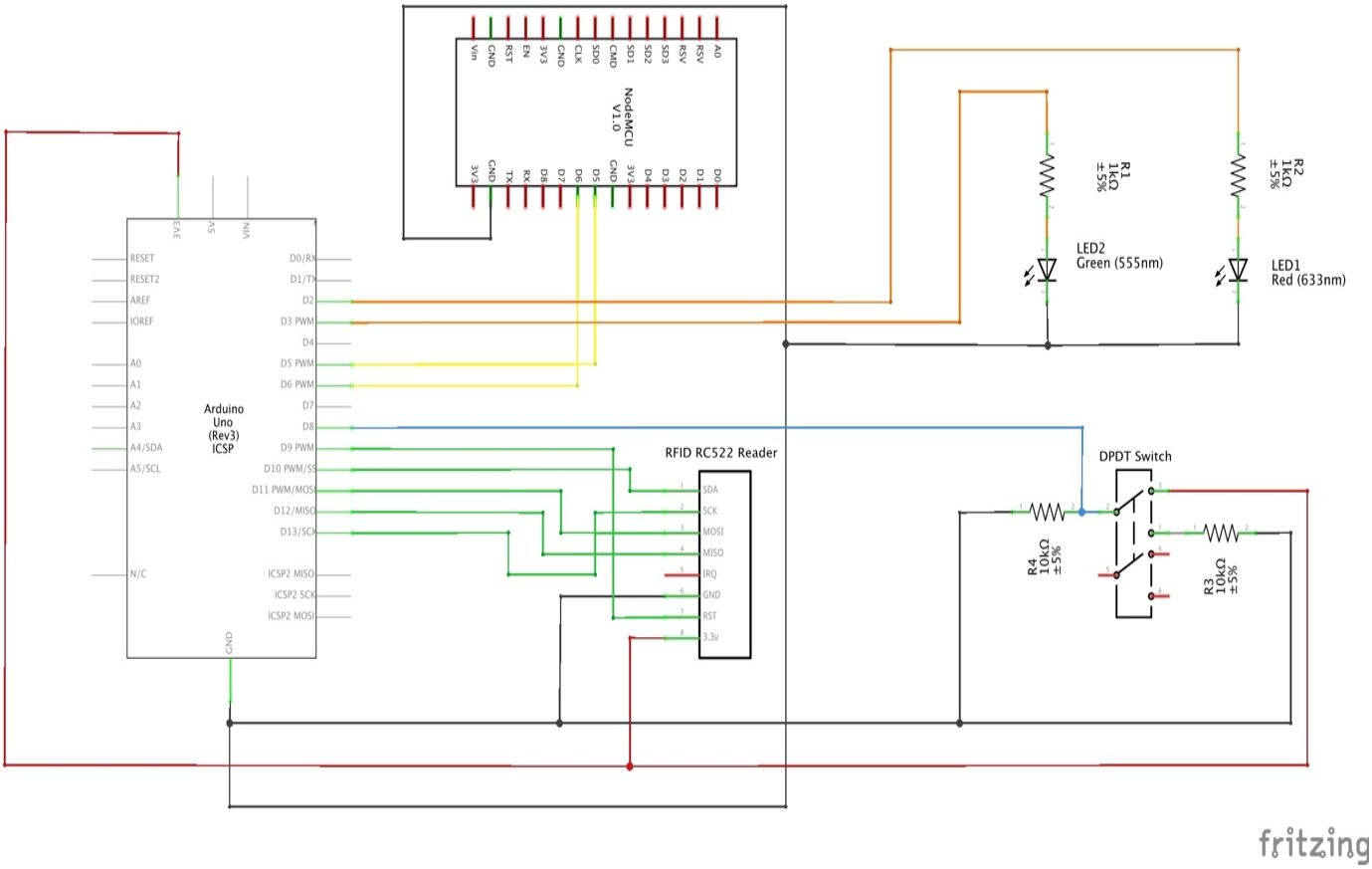
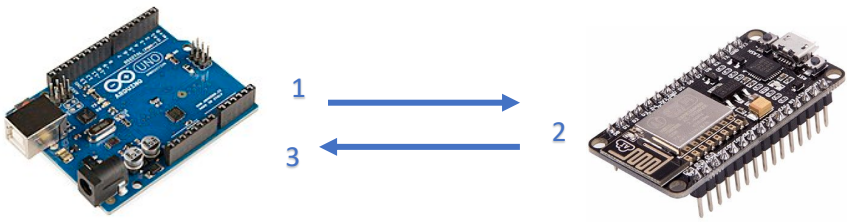


Figure 5.2: Schematic diagram for reserved parking module

The module's software implementation of Arduino and NodeMCU also follows the system's high level architecture in section 4.2 in terms of the module's working flow. A simplified version is shown below. The module's flow is a two-way flow starting from Arduino to NodeMCU then back to Arduino.



Element 1: Arduino

Element 2: NodeMCU

<u>Element 1: Arduino (first part)</u>

The first part of Arduino in the working flow is sending the scanned UID to NodeMCU. Below are the steps that the Arduino's loop.

- ***Step 1:*** RFID Reader scans UID tag if there is any within the reading range.
  - If a UID tag is scanned and read, Arduino proceeds to step 2
- ***Step 2:*** Read the current zone from the switch button
  - Low (not pressed): LIB Female & Male Zone
  - High (pressed): CBAE Female & Male Zone
- ***Step 3:*** Send UID to NodeMCU serially

- As explained above, the data is in JSON format when sent serially. This data is structured as follows:

```
{
    "zoneNumber" : 0,
    "A" : 01001010,
    "B" : 00001111,
    "C" : 01110001,
    "D" : 00111111
}
```

- Zone Number: could either be 0 or 1
  - 0: LIB Female & Male Zone
  - 1: CBAE Female & Male Zone
- A, B, C and D: The four parts combined gives the tag UID, the numbers stored are in binary (0s and 1s)

Element 2: NodeMCU

NodeMCU checks if the received UID has a valid reservation from Firebase. The NodeMCU's loop is as follows:

- ***Step 1:*** Receive tag UID from Arduino serially
  - Arduino reads the received JSON format data using ArduinoJson.h library as explained in Arduino's Loop Step 3.
  - Convert binary numbered UID to a hexadecimal number, then the UID number is converted to a string
- ***Step 2:*** Get current time and date using NTP client
- ***Step 3:*** Check if there is a reservation with the received UID
  - ***Step A:*** Get all reservations with the received UID from Firebase
    The following steps are looped through all received reservations
  - ***Step B:*** Check if
    - reservation's zone == zone sent serially
    - reservation's date == current date

    If any of the conditions is false, NodeMCU goes to the next reservation in the loop
  - ***Step C:*** Check if reservation has started
    Current time >= reservation's start time

    If it has not started yet, NodeMCU goes to the next reservation in the loop
  - ***Step D:*** Check if reservation status is "created" and reservation time has not ended
    (status == "created") & (current time <= end time)

    If *both* conditions hold, NodeMCU stores the reservation key in Firebase and goes to step 4
    If *first* condition fails, NodeMCU goes to step E
    If *second* condition fails, NodeMCU goes to the next reservation in the loop
  - ***Step E:*** Check if reservation status is either "extended" or "arrived" or "subcancelled"
    If not, NodeMCU goes to the next reservation in the loop
  - ***Step F:*** Check if Automatic Cancellation can be applied
    If there is at least one hour or more left to the reservation
    current hour < reservation's end time

if condition is true, automatic cancellation is applied by deducting half the cancelled hours price (2.5QR/hour)

price -= (2.5 * number of hours left)

then NodeMCU stores the reservation key in Firebase and goes to step 5

if condition is false, NodeMCU goes to step G

- ***Step G:*** Check if VIP user has exceeded his reserved time

  Current hour > reservation's end time

  Note: VIP user is left without any penalty in the first 5 minutes after his/her reservation

  If condition is true, the VIP user is penalized with triple the original price for one reservation hour (15 QR per extra hour)

  price += (15* number of extra hours)

  then NodeMCU stores the reservation key in Firebase and goes to step 5

  If condition is false, NodeMCU stores the reservation key in Firebase and goes to step 5

- ***Step H:*** If none of the received reservations goes into steps 4 or 5, then NodeMCU goes to step 6 with a negative response

- ***Step 4:*** Update reservation status to "arrived" in the Firebase
  - The status is updated using the reservation key
  - NodeMCU goes to step 6 with a positive response
- ***Step 5:*** Update reservation to "ended" in the Firebase
  - The status is updated using the reservation key
  - NodeMCU updates the reservation prices if there is any addition (Step G) or deduction (Step F), afterwards goes to step 6 with a positive response
- **Step 6**: NodeMCU sends a positive or negative response to Arduino serially
  - Positive: to open the gate of the parking lot
  - Negative: to not open the gate of the parking lot

Element 1: Arduino (second part)

The second part of Arduino receives a reply from NodeMCU on whether to open the parking lot gates to the VIP user or not. The loop includes one step.

- ***Step 4:*** Wait till a response is received from NodeMCU serially
  - Positive response: Light up the green LED (i.e. Send digital high to green led pin) to indicate that the gate is opened.
  - Negative response: Light up the red LED (i.e. Send digital high to red led pin) to indicate that the gate is closed.

  When a response is received Arduino goes back to step 1 (first part)

5.4. Software implementation

The software implementation consists of many services and feature. In this section, we are going to discuss the algorithm for each service. Regarding the application and website design, only the main screenshots are provided in this section. For more details about design please refer to section 6.1.3 where all the UI screenshots are illustrated there.

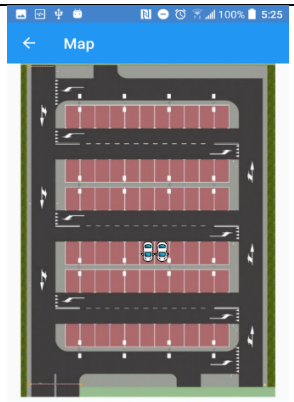
1. Check Availability:

- Algorithm:

Step 1: Wait till the user selects a zone
Step 2: Load all the spots that are in the selected zone
Step 3: Check each spot status, if the status is “available” set the visibility of the car image to be “invisible” and if it is “not available” set the visibility of the car image to be “visible”
Step 4: Wait till the user select spot number and click on “Get Direction” button
Step 5: Provide directions to the wanted parking spot from his/her current location (device location) by using google map

- Design:

Table 5.6: Check Availability Design

Application	Website
	
The map is designed by using image view.	Map is implemented by using Modal that pops up after “show Abaialbilty” buttons get cliked by a user.

2. Reserve a parking:

- Algorithm:
Step 1: Wait till the VIP user selects a date
Step 2: Load all the reservations that are in the selected date and the status not equal to cancel
Step 3: For each reservation, loop through the time array to find how many reservations in each hour
Step 4: For each hour find availability percentage (number of reservations in a specified hour/ total allowable reservation per hour) * 100. Based on the calculated percentage the color of each hour will be changed in the UI. Green if >75% is left, orange if 50% is left, red if 25% is left and gray if 0% is left.
Step 5: Wait till the VIP user selects start time and duration and clicks on the reserve button
Step 6: Whenever VIP user click on the reserve button, 5 conditions will be checked
    - Check if the VIP user does not have a reservation at the selected hours.
    - Check if there is an available parking spot at all selected hours
    - Check if selected date is equal to today or tomorrow (selected date == current date || selected date == current date +1) as the VIP user can only reserve at the same day or one day before the reservation date.

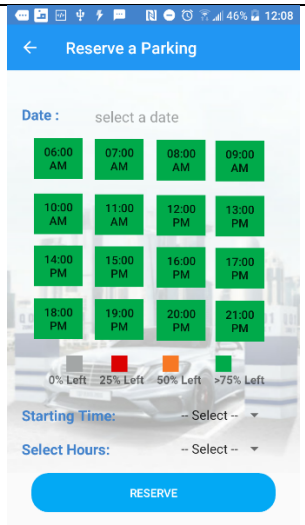
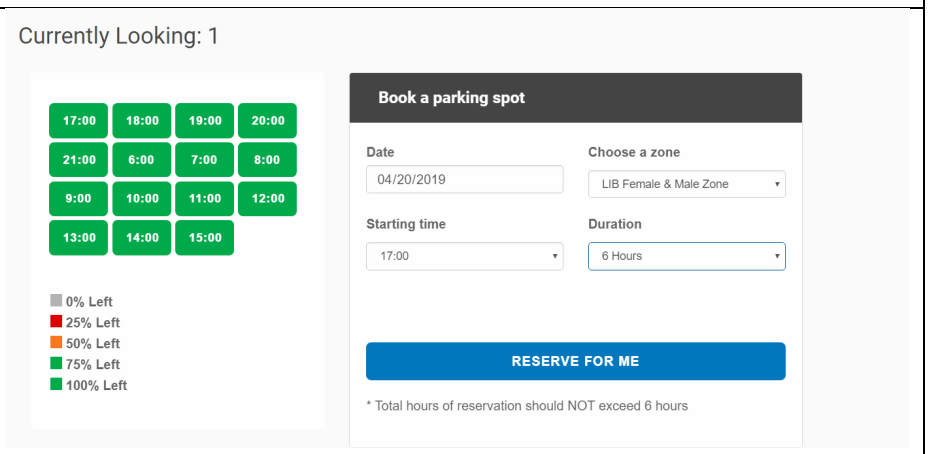
- Check if the reservation start time has passed, if the selected date is today by checking if the selected start hour is greater than the current hour
- Check if the number of selected hours with selected date's reserved hours is less than or equal to the number of allowable reservation hours per day (6 hours). Reserved hours in the selected date are calculated by going through the user reservations
  Total reservation hours += number of hours in a reservation – number of hours extended – number of hours cancelled

<u>If one of these conditions is not met an error message will appear to the VIP user</u>

Step 7: If all the above conditions are met then a reservation record with "created" status and 5 QR per each hour price will be added to the database.
Step 8: Notifies the VIP user 30 minutes before the expiry time

- Design:

Table 5.7: Reserve a parking design

Application	Website
	
Reservation page is designed by using buttons, spinner and text view components.	This page is implemented by using HTML forms,buttons and inputs.

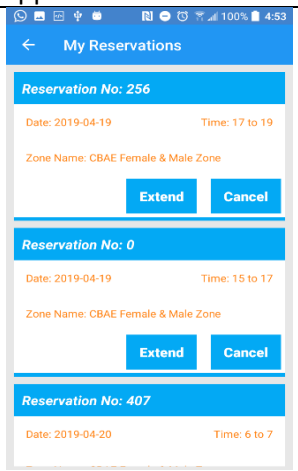
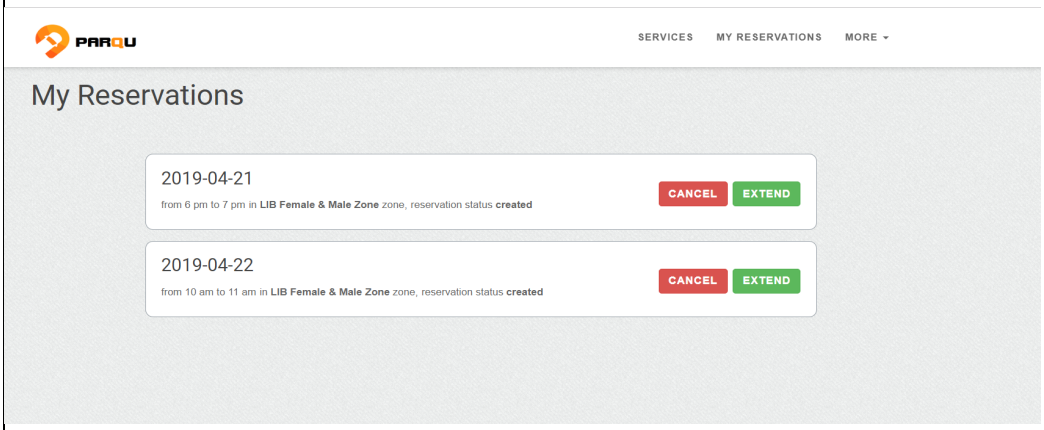
<u>3. Show reservations:</u>

- Algorithm:
  Step 1: Load all current and upcoming reservations related to the VIP user
  Loop through all reservations
  - If reservation date > current date (which mean no need to check the time)
    Check if (reservation plate number == user logged plate number && status != "cancelled" && status != "subcancelled")
  - If reservation date == current date (time need to be checked)

For each reservation check if (reservation plate number == user logged plate number && status != “cancelled” && status != “subcancelled” && current time <= reservation start time)

- Design:

Table 5.8: Show reservations design

Application	Website
	
Reservation list is designed by creating custom list view.	List of reservation is implemented using list element in HTML and is styled using Bootstrap.

4. Extend a reservation:

- Algorithm:
  Step 1: Wait till the VIP user selects “Extend” option
  Step 2: Whenever VIP user click on extend option, 2 conditions will be checked
  - Check if the selected reservation is at its last hour (current hour == last hour of reservation) as the VIP user can only extend his reservation in the last hour of the reservation
  - Check if there is an available parking spot after the reservation time

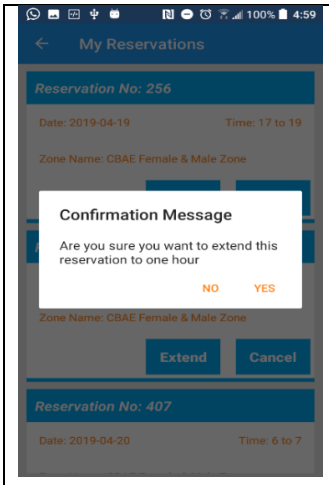
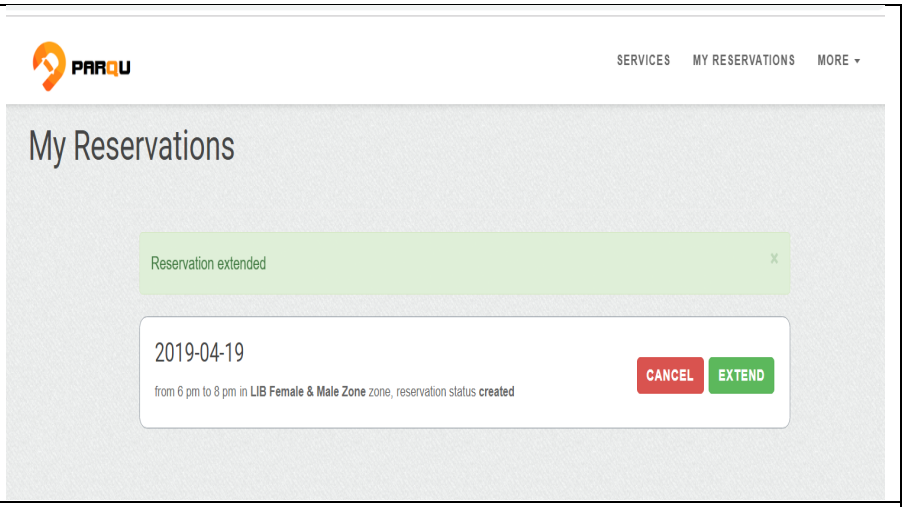
  If one of these conditions is not met an error message will appear to the VIP user

  Step 3: If all the above conditions are met then reservation is extended by updating the following fields in the database: status changed to “extended”, price incremented by 5 QR and the number of hours extended is incremented by 1

- Design:

Table 5.9: Extend a reservation design

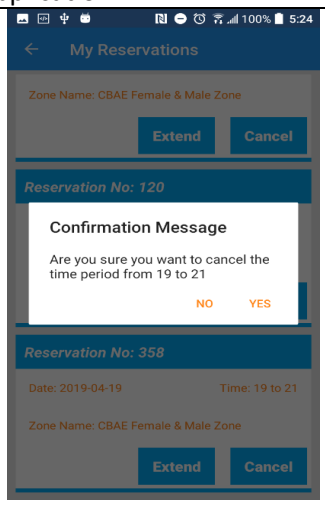
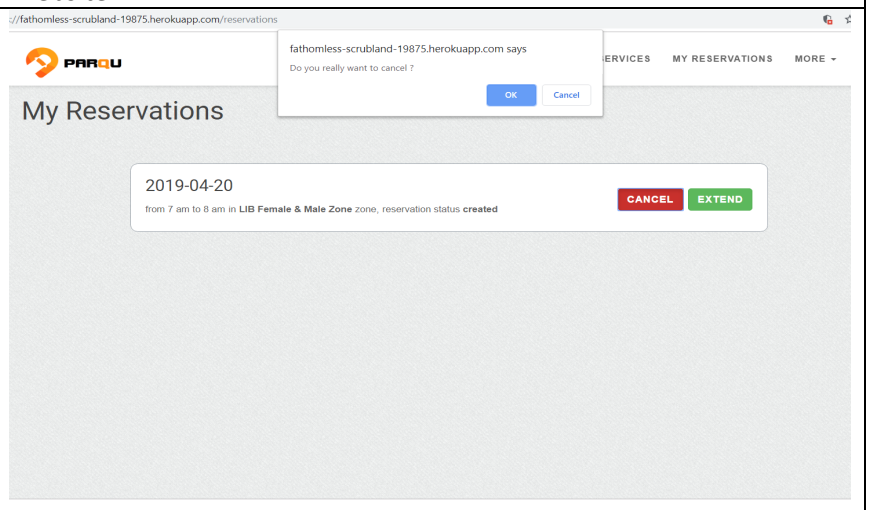
Application	Website

	
The confirmation message is designed by using alart dialog.	In the same page " My Resrvation", a success or error message is implemented by using **`Express Flash`** will be shown to the user.

5. Cancel a reservation:

- Algorithm:
  Step 1: Wait till the VIP user selects cancel option
  Step 2: Whenever VIP user click on extend option, 2 conditions will be checked
  - If the reservation has not started (current time < start time), cancel the whole reservation by updating the following field in the database: status changed to "cancelled", price decremented by 50 % and number of hours cancelled incremented based on reservation hours
  - If the reservation has started (current time >= start time), cancel remaining reservation hours by updating the following field in the database: status changed to "subcancelled", price decremented by 50 % for each cancelled hour and number of hours cancelled incremented based on number of hours that are cancelled.
- Design:

Table 5.10: Cancel a reservation

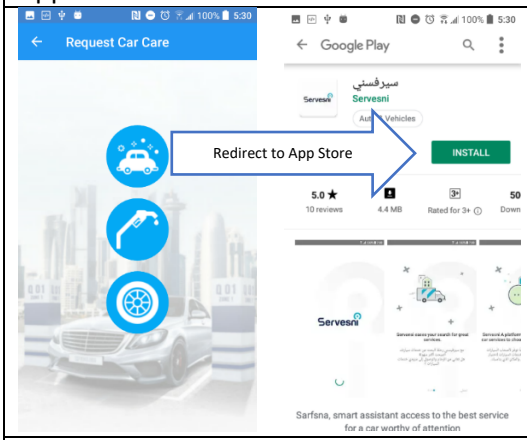
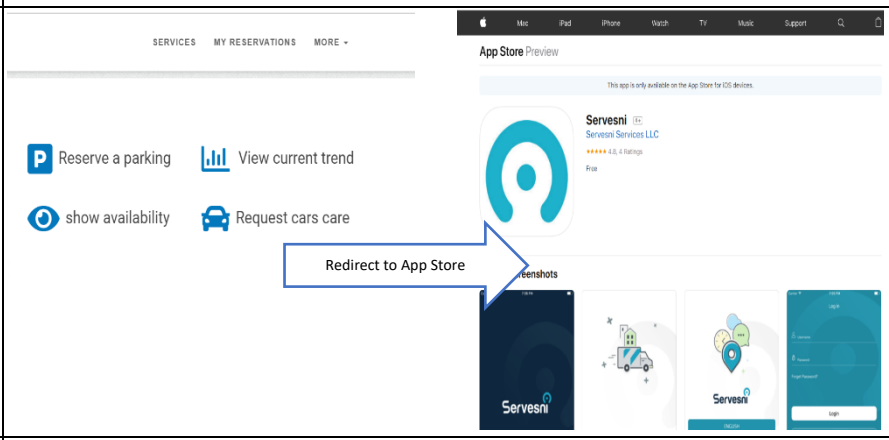
Application	Website
	

The confirmation message is designed by using alart dialog.	A pop up box for confirmation is implmented by JavaScript method **`window.confirm().`**

6. Request car care:

- Algorithm:
  Step 1: Wait till the VIP user selects one of the car care services options
  Step 2: Whenever VIP user click on extend option, 2 conditions will be checked
  - If the application is installed in the phone, VIP user is successfully redirected to the Servesni application.
  - If the application is not installed in the phone VIP user redirected to the Play Store
- Design:

Table 5.11: Request car care design

Application	Website
	
Option buttons is designed by using custom buttons with icons.	Linked “Request car care” option to Servesni application in App Store.

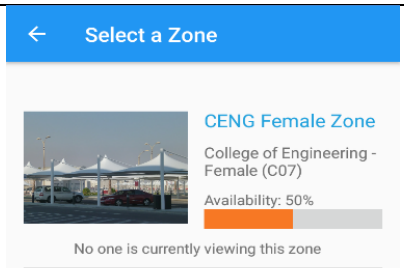
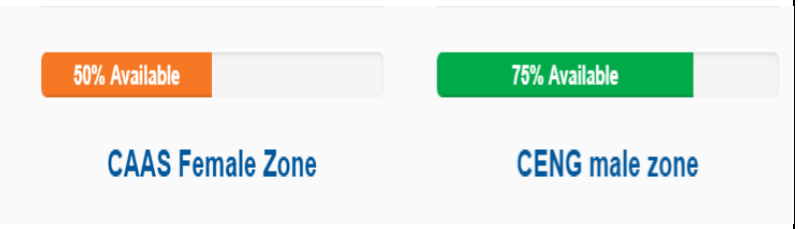
7. Availability percentage:

- Algorithm:
  Step 1: Load all the zones from the database
  Step 2: For each zone, the availability percentage of spots is calculated by looping through all the spots and count how many spots are available
  Step 3: Availability percentage is computed by the following equation
  availability percentage = (number of available spots (counted in step1) / total number of spots in zone) *100
  Step 4: Based on the computed percentage the progress bar color will be changed. Green if more than 75% of the spots are available, orange if 50% of the spots are available and red if 25% of the spots are available.

- Design:

Table 5.12: Availability percentage design

Application	Website

	
Availability percentage is designed using prograss bar and text view component.	Availability progress bar is implemented using Bootstrap CSS.

8. Currently looking:

- Algorithm:
  We expand our database to have a new relation called currently looking which has an ID, login time and zone name for each user who views a zone.
  Step 1: When a user views a zone, a new currently looking record specified for the user will be added to the database
  Step 2: When a user leaves the zone page, the user's currently looking record will be deleted from the database
  Nevertheless, the user may lose internet connection while viewing the zone. To handle this case, we applied the following logic:
  Step 1: For each zone, load all the currently looking records from the database
  Step 2: Remove all records older than 10 minutes
  Step 3: Show how many people are currently viewing each zone

9. Current Occupancy Trend:

For each zone, the entire occupancy data is stored in the statistics node as shown in Figure 5.4. The statistics structure holds an array of object where each element is a week day (0: Sun, 1: Mon, 2: Tue, etc...)

Each day object has a field called "hoursInfo" which is also an array of objects that holds hours data (6 AM – 9 PM). For example, the first index (0) stores data for 6 AM (See Figure 5.6). Each hour object stores 3 elements which are hour, count array and date array where count is the number of reservations in that specific hour at the corresponding date in date array (See figure 5.3). For example, in 2019-03-24 there are no reservations at 6 AM. We only store the data for recent 4 (Sunday) in weeks

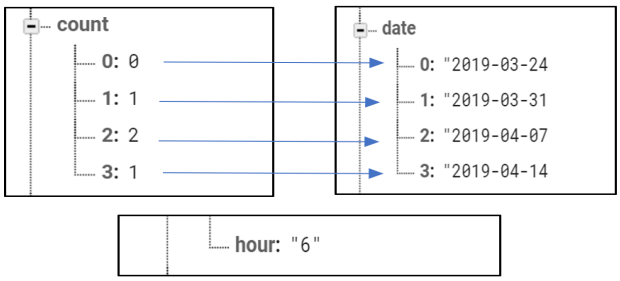


Figure 5.3: Hour object attributes

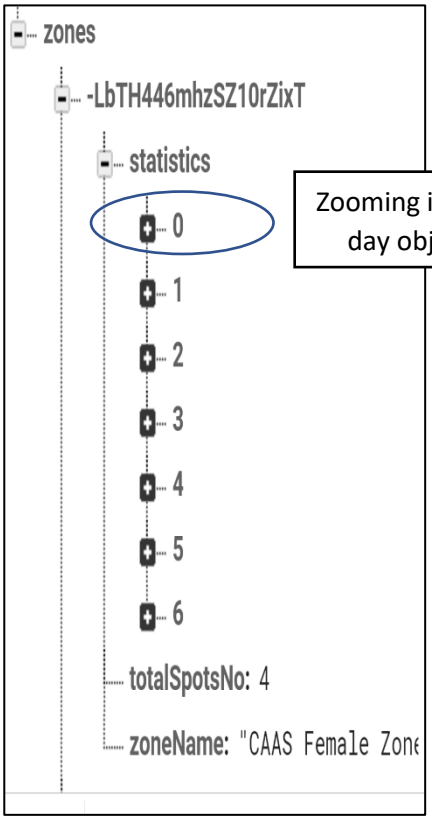


Figure 5.4: How statistical data stored in the database

Zooming in first day object

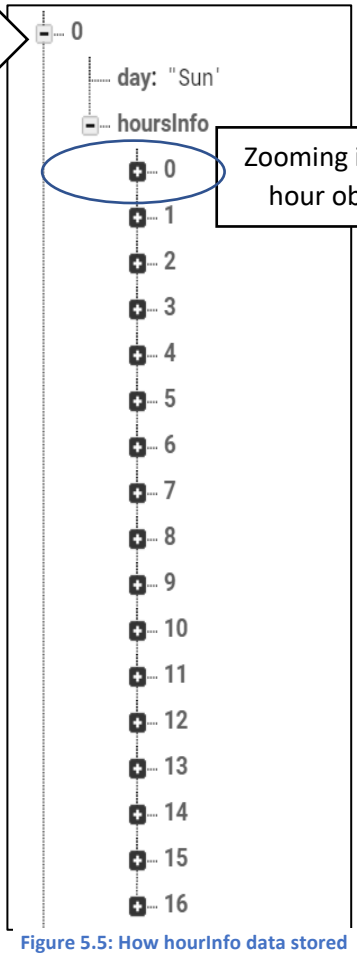


Figure 5.5: How hourInfo data stored in the database

Zooming in first hour object

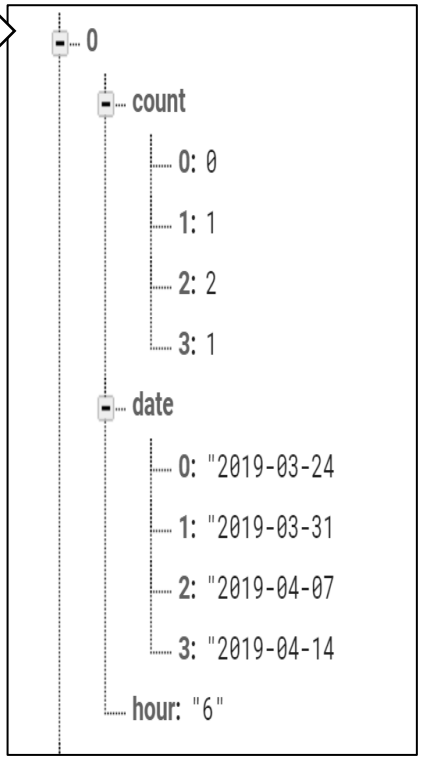


Figure 5.6: Zooming in one hourInfo object

Updating statistics data for reservation zones

- Algorithm:
  Step 1: Check which day of the week the date of reservation belongs to
  Step 2: Update all hours that are equal to reservation hours in that date by the following logic
  - if the date exists in the date array, increment the corresponding element in count array by 1
  - if the date does not exist, remove the oldest date with its count value (always in index zero), shift other dates and counts upwards and then add the new date and count (always in index two)

Updating statistics data for check availability zones

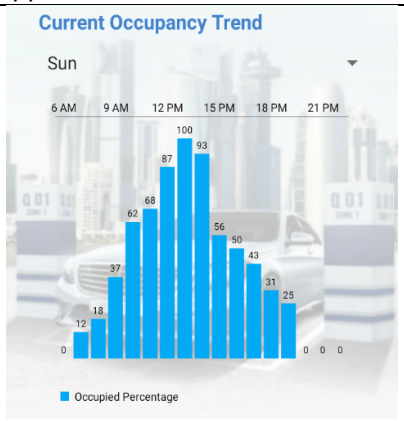
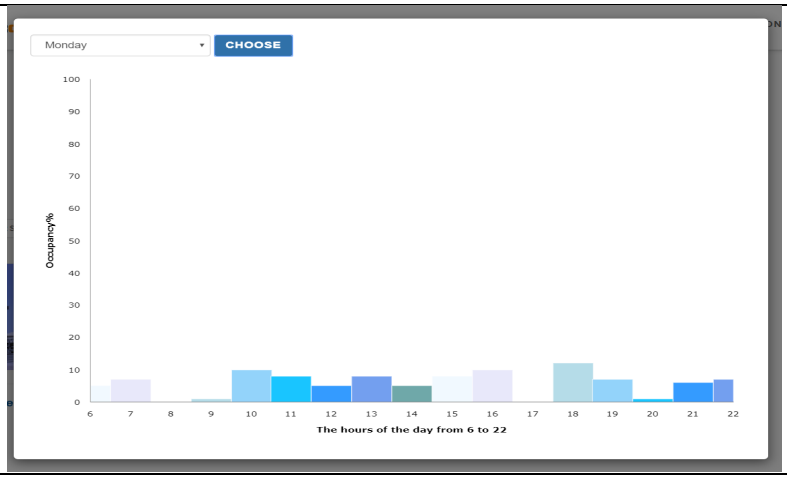
- Refer to section 5.3.1

Displaying data in term of a histogram:

To display data as a bar chart we used a library called MPAndroidChart [20]

- Algorithm:
  Step 1: Wait until a user selects a day of the week
  Step 2: For each hour, the occupancy percentage is calculated as follows
  % Occupancy = (total sum of count values / (number of dates * total number of spots)) * 100
  Step 3: Use setData function from MPAndroidChart to set the calculated percentage for each hour into a bar chart

- Design:

Application	Website
	
Bar chart is designed using MPAndroidChart library	Bar chart is designed using amCharts library

5.5. Challenges and Solutions

As we implement our project, we faced many challenges that required us to solve them to meet the project's requirements.

1. Currently looking implementation

- Challenge:
  It was quite challenging to implement this feature as the implementation logic in the application was different from the website. In the website, it is easy to know how many people are visiting a specific page by simply using the devices' IP addresses. However, this is not possible in an Android application.

  To implement the feature in the Android application, we must follow the "Activity Life Cycle" concept where each activity (page) goes through several stages through functions. The functions we used are the onStart() and onStop(). Whenever an activity starts, onStart() method is always called, so we invoked the method to place the logic needed for increasing the number of people looking. In contrast, the logic needed for decrementing the number of people looking was placed in onStop() as this method is called whenever the user leaves the page (activity).

After implementing this logic, we figured out that it will not work properly in the case that the user loses internet connection. Logically if the user lost connection the number of people looking should be decremented by 1. However, in reality, the number stays as it is. This is because there is no internet connection to update the number in the database, and if the user happens to restore connection, it would be considered as a new user (number incremented).

- Solution:
  The logic used to solve this issue discussed above, refer to section 5.4.

  By applying this logic, we give the number of people currently looking in a page. In some rare cases, the number can be an approximate as some users might view a page for more than 10 minutes.

2. Current Occupancy Trend implementation

- Challenge:
  We calculated the occupancy percentage based on all the history data stored in the database. However, with time, the huge accumulated data will affect the accuracy of new data and would not reflect what happens in real life. This is because the entire data would not be sensitive to changes after a long duration. For example, after one year, a huge data would have been accumulated that adding current data to previous data and averaging them, as the below equation, would not affect the trend and show what is happening currently to the users.
  (Today's data + all previous data) / number of days

- Solution:
  We decided to apply the moving average concept by calculating the occupancy percentage based on the recent 4 weeks data. The histogram is continually recomputed as new data becomes available, it progresses by dropping the earliest value and adding the latest value.
  The logic used to solve this issue discussed above, refer to section 5.4.

3. Serial communication

- Challenge:
  In Reservation Free Parking Module, the Arduino used to send the parking spots status to NodeMCU continuously. Afterwards, NodeMCU will detect if there is any change in status. This made the serial connection slow and required 15 seconds or so for a change to be available in the application/website.

- Solution:
  We changed how the data is sent from Arduino. Now, the Arduino only sends data serially when it detects a change, otherwise no data is sent to NodeMCU. This significantly increased the efficiency of the serial communication and the system overall.

4. Website design

- Challenge:

We started building the website design with bootstrap template in order to implement it faster and easier as it contains HTML and CSS-based design templates for forms, buttons, navigation and other interface components. After building the required pages, we figured out there were a lot of elements that can confuse the users. Creating a prototype again at that stage was a challenge. Moreover, extra features have been added during the implementation phase, which required re-structuring the website again and change both front-end and back-end elements.

- Solution:
  We re-revised the website design by using a sketch so we can quickly modify the design and it helped us to get the users' expectations on the working flow of the website without consuming much time and effort from ours side. So, after finalizing the prototype we removed any unnecessary components in order to minimize the number of user's click as much as possible and unified the colors and images between the website and the application to ensure the consistency and to create more streamlined and flawless experience for the user.

6. Testing

Several test mechanisms were applied to the prototype to verify that it has met the functional requirements and the design constraints mentioned in section 3. This section also proves how our prototype provides the solution to the problem stated in section 1.
The testing follows two main parts: First part is the black box testing which tests the system functionalities without inspecting the detailed internal processes and is applied through several stages. Second part is the outdoor testing where ultrasonic sensor was tested outdoor.
Part A: Black box Testing

1. Unit Testing: tests individual subsystems
2. Integration Testing: tests interface between groups of subsystems
3. System Testing: tests entire system
    A. Functional Testing: tests functional requirements
    B. Acceptance Testing: tests customer requirements
    C. Quality Testing: tests non-functional requirements

Part B: Outdoor Testing

6.1. Black Box Testing

6.1.1. Unit Testing

Tests the system components/subsystems individually to confirm that the component functions correctly and that the subsystem is coded correctly and performs the required functionality.

A. Unit testing for hardware

Components	
Ultrasonic Sensor Test	**RFID Reader and Tags Test**
Goal: Test if the sensor measures the distance of an object correctly.	Goal: Test if RFID reader reads the UID of a tag correctly. Expected results: Tag UID (97 D1 34 83) Actual results:

Expected results: Object's distance ~ 18 cm

Figure 6.1: Distance from sensor to object ~ 18 cm

Actual results:

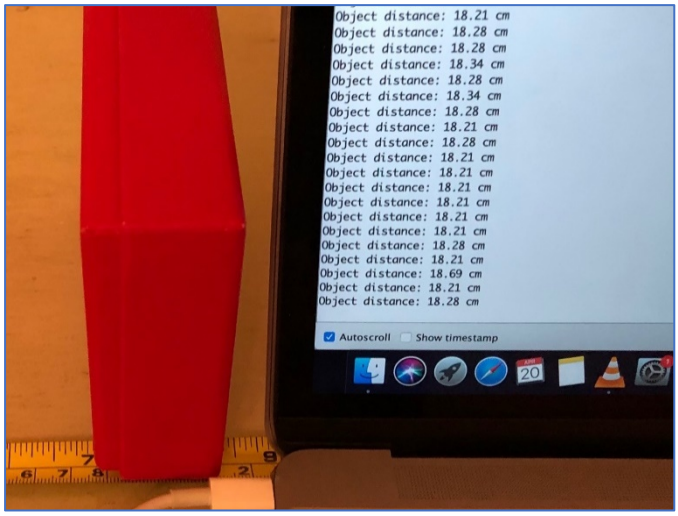


Figure 6.2: Output: object distance ~ 18 cm

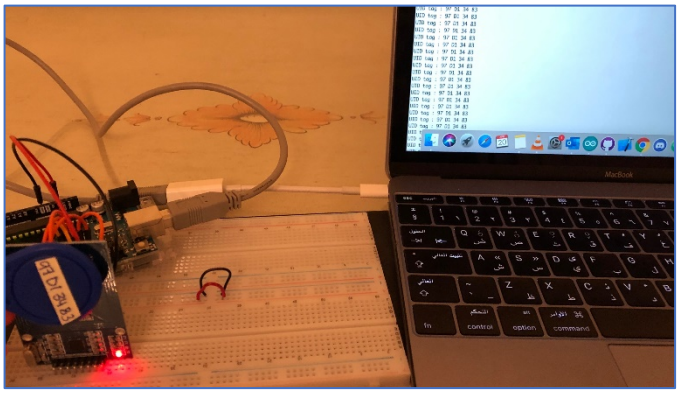


Figure 6.4: Tag (97 D1 34 83)

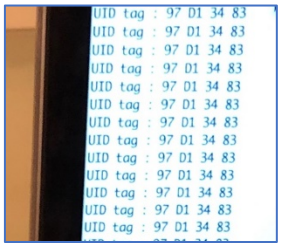


Figure 6.3: RFID reader reads the tag

B. Unit testing for subsystems

The implementation is divided into three subsystems for each module.

Subsystems
First Module: Reservation Free Parking Module
1. Arduino: Detect change in ultrasonic status

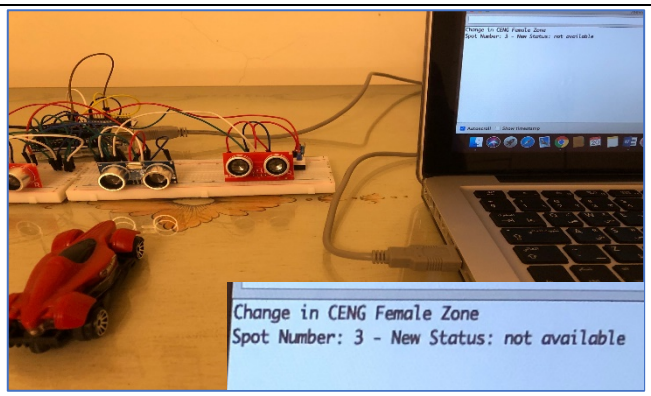


Figure 6.5: Arduino detect any change

2. Arduino & NodeMCU: Send status from Arduino to NodeMCU serially

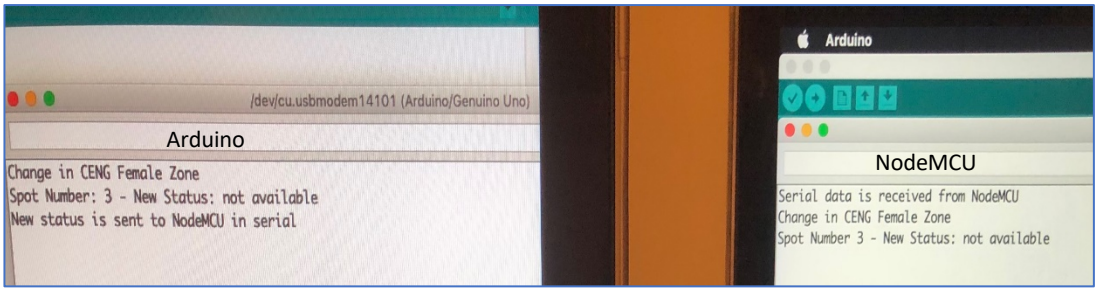


Figure 6.6: Arduiino send status to NodeMCU

3. NodeMCU: Update status in Firebase

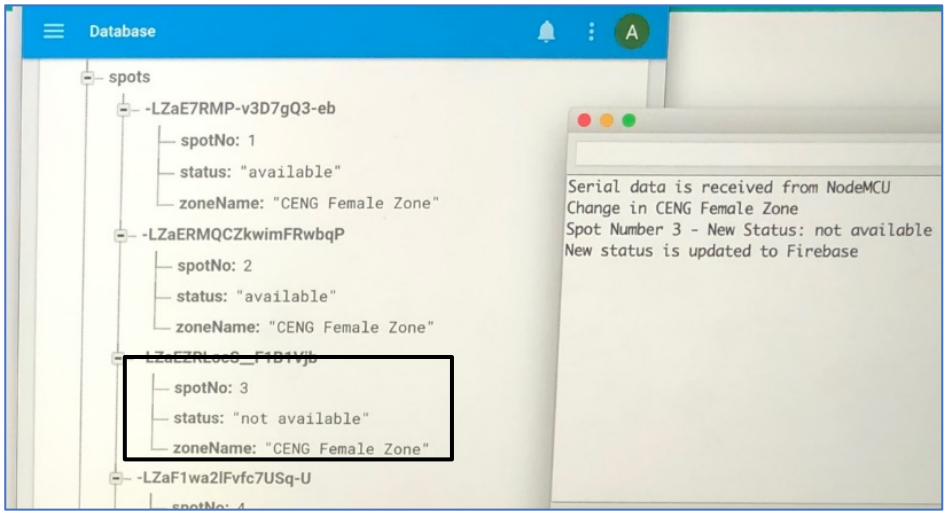


Figure 6.7: Status updated in Firebase

Second Module: Reservation Free Parking Module
1. Arduino: Get scanned UID tag from RFID Reader

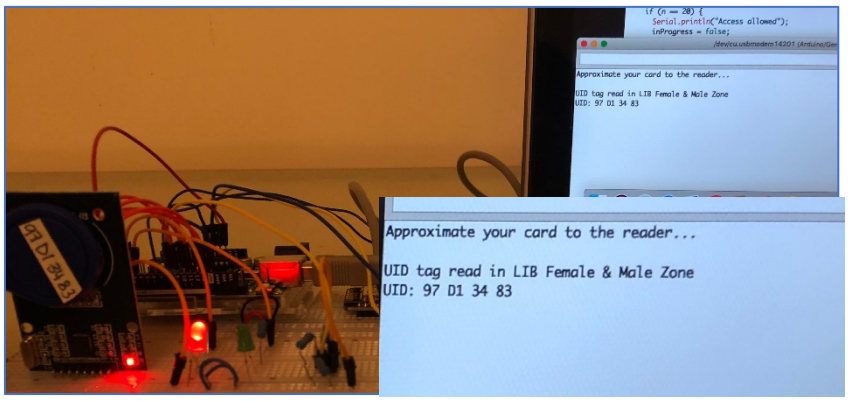


Figure 6.8: Arduino get tag from RFID Reader

2. Arduino & NodeMCU: Send UID from Arduino to NodeMCU serially

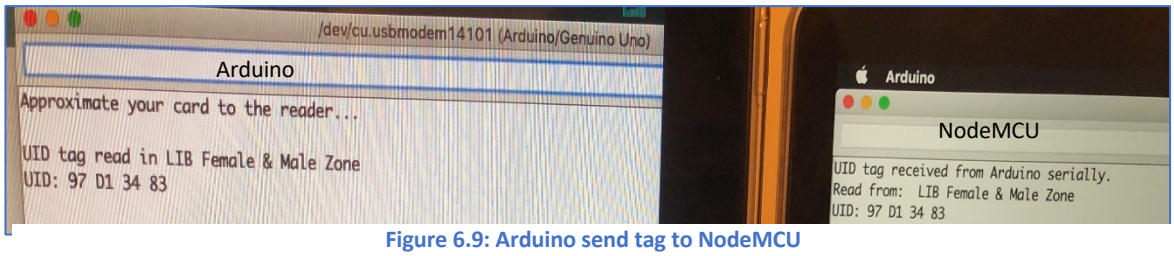


Figure 6.9: Arduino send tag to NodeMCU

3. NodeMCU & Arduino: Check if there is a valid reservation with the UID tag received from Firebase and change its reservation status to “arrived”. Arduino opens gate (Green LED)

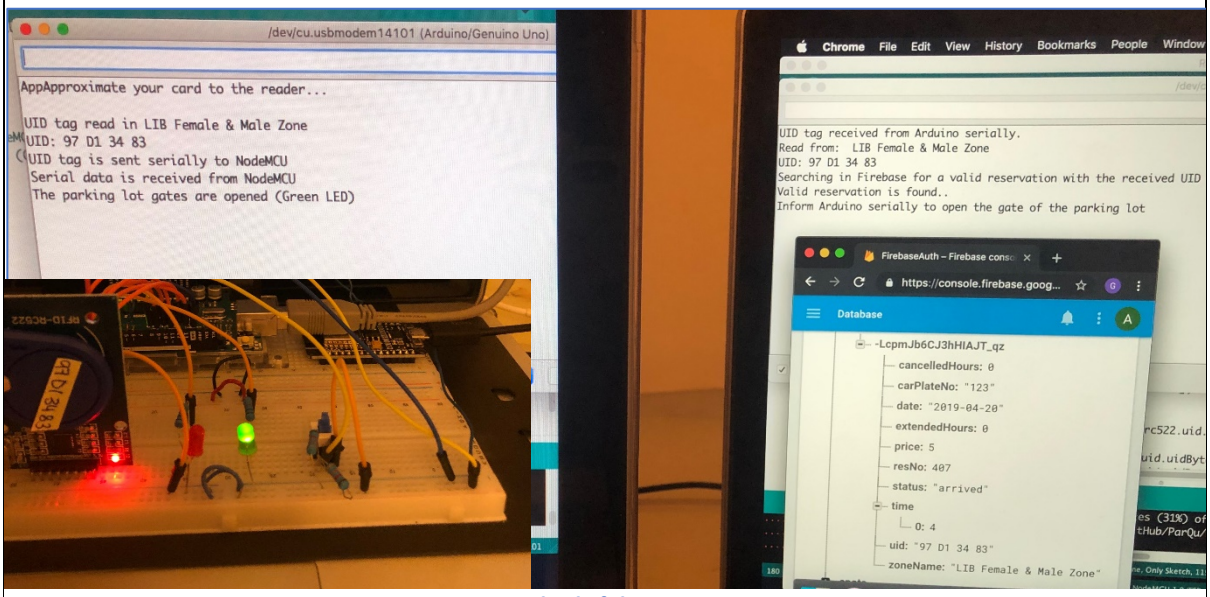


Figure 6.10: Check if there is reservation

6.1.2. Integration Testing

Tests a group of subsystems to ensure that they are interfaced correctly.

A. System Integration

During the implementation process, the hardware, application and website used different firebase projects while following one database schema. The three parts were tested to confirm that each one

worked as expected individually. Afterwards, the parts were combined to form the whole system by integrating them into one database. Figure 6.11 illustrates the whole system after integration through the availability percentage in Reservation Free Parking module. As explained in section 4, our prototype implements four parking spots, therefore each parking spot holds 25% availability.

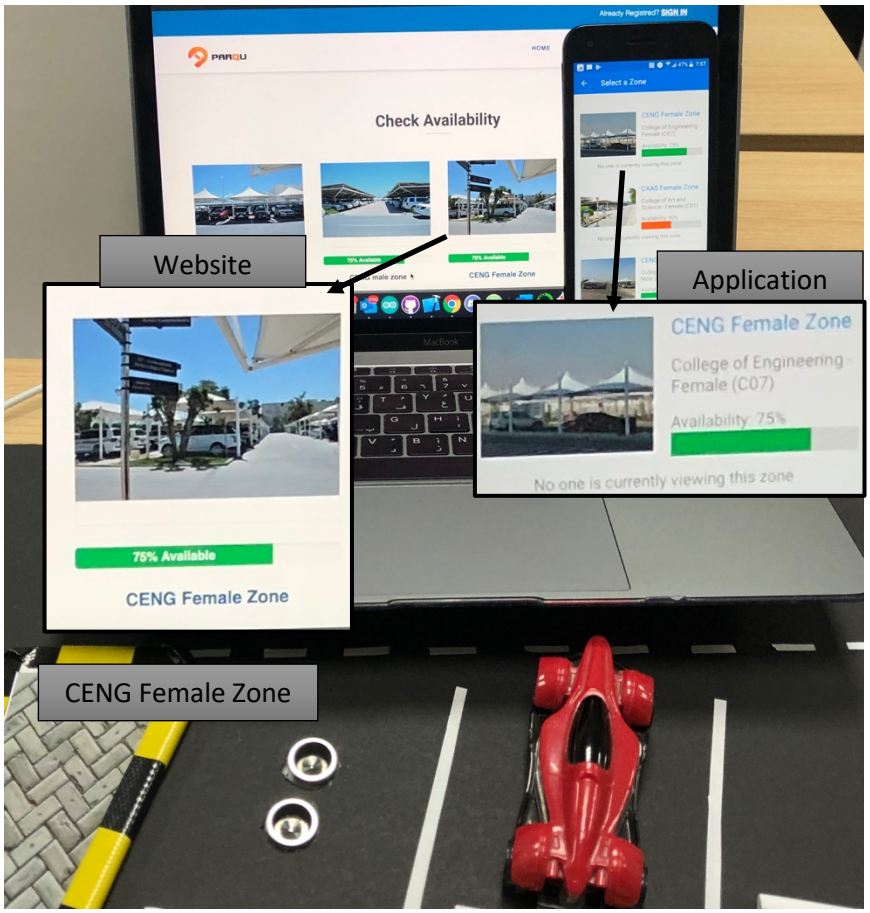


Figure 6.11: Integrate the 3 components together

B. Firebase Connectivity Testing

Firebase is an external subsystem that assists in keeping our system in real time. The connectivity test measures the average time it takes for a connection to pass data between a certain component in the system and Firebase. This test shows to what extent our system is real time and its speed. All tests were done in the same environment, internet connection and around the same time. In the application and NodeMCU, the time was recorded by the following algorithm:

- Step 1: read system time and store it in a variable called startTime
- Step 2: Preform needed operation with Firebase
- Step 3: read system time again and store it in variable called endTime
- Step 4: subtract the startTime from endTime and display result in the console

In contrast, the website uses a different algorithm to record the time:

Using Postman as a testing tool, by firstly creating new routes for reading/writing without displaying pages content and applying GET/POST requests using Postman as it automatically calculates the time it took for the response to arrive.

Four trials were computed for each connection, and then the average test time was calculated.

The test is done through several levels:

- Between NodeMCU and Firebase
- Between Application and Firebase
- Between Website and Firebase

Between NodeMCU and Firebase

The internet connection between NodeMCU and Firebase was tested by measuring the time NodeMCU needed to read from or write to the Firebase database. The system requires NodeMCU to deal with two main operations when communicating with the Firebase. First operation is to get all reservations in the database and second operation is to get or update (set) a specific value in a node. Both operations have different data sizes and thus the time taken for each operation differs. Table 6.1 shows the final results, and the detailed test is provided in the Appendix D.

Table 6.1: NodeMCU Average Connection Time

NodeMCU *Average* Connection Time to		
	Get	**Set**
Reservation (177 bytes)	0.353489 s	-
Update a value (15 bytes)	0.301054 s	0.388572 s

Between Application and Firebase

The internet connection between Application and Firebase was tested by measuring the time Application needs to read from or write to the Firebase database for several data sizes. Table 6.2 shows the final results, and the detailed test is provided in the Appendix D.

Table 6.2: Application Average Connection Time

Application Average Connection Time to		
	Post	**Get**
Reservation (177 bytes)	0.012112 s	0.007586 s
Currently Looking (52 bytes)	0.011627 s	0.007296 s
User (113 bytes)	0.011777 s	0.007429 s
Zone (8606 bytes)	0.017236 s	0.008666 s
Spot (49 bytes)	0.011526 s	0.007123 s

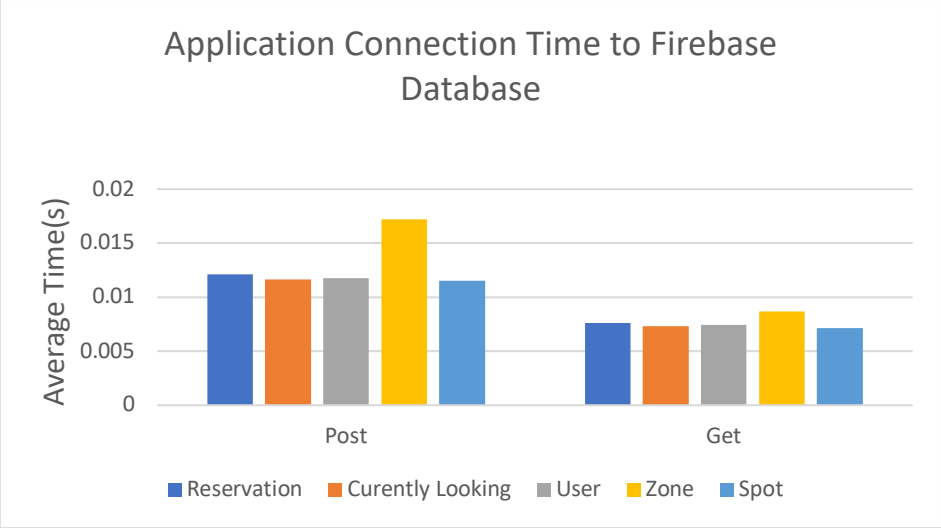


Figure 6.12: Application Connection Time to Firebase Database

In general, we observe that reading (get) data from Firebase database is much faster than posting data in the Firebase regardless of the data size. The results are to our advantage since our system is real time, we care more about fetching updated data from the Firebase and reflecting it to the UI as quick as possible than we care for sending data to the Firebase.

Between Website and Firebase

To test internet connection to and from Application and Firebase in several data sizes.

Table 6.3: Website Average Connection Time

Website Average Connection Time to		
	Post	**Get**
Reservation (177 bytes)	0.4838 s	0.4242 s
Currently Looking (52 bytes)	0.3824 s	0.3690 s
User (113 bytes)	0.4512 s	0.4048 s
Zone (8606 bytes)	0.8122 s	0.8303 s
Spot (49 bytes)	0.3306 s	0.3298 s

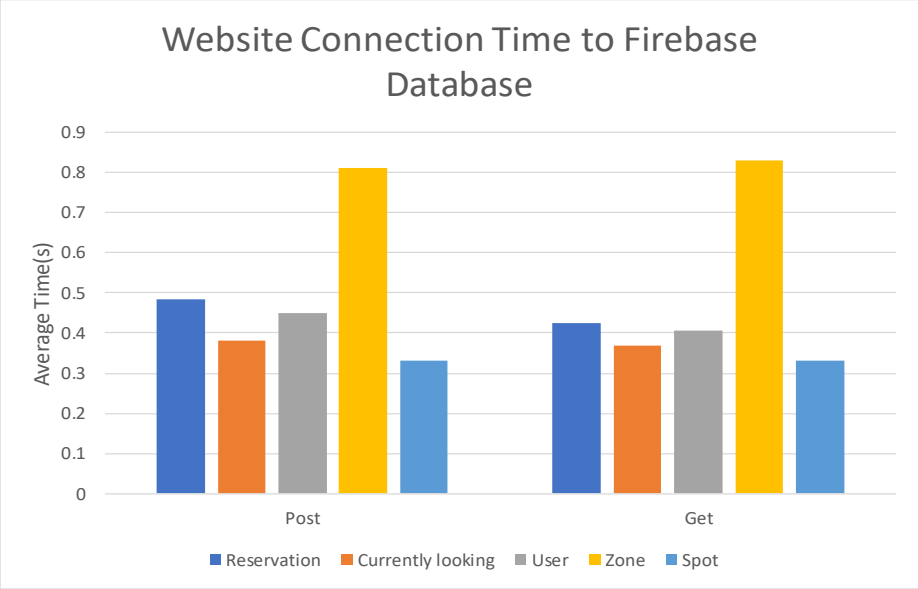
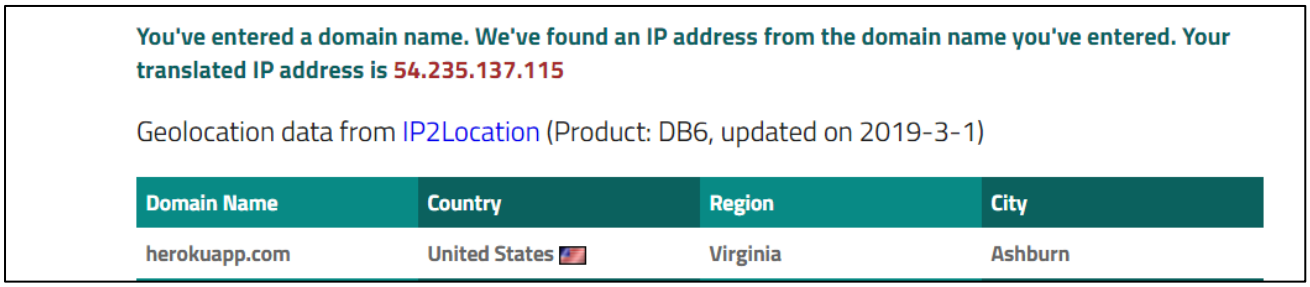


Figure 6.13: Website Connection Time to Firebase Database

In average, the total time needed to read/write to and from Firebase is considered a small number, which is good for our system as it is in real time. As shown in Table 6.3, it is clear that the time is directly proportional to the number of bytes. Moreover, the website needed more time to fetch data compared to the application and NodeMCU. This delay in response time may be due to some factors, one of them is the server location. By using iplocation.net [23], a service for finding the geolocation of the server IP address, we found that the server location based on its IP is located in United States (See Figure 6.14). This explains the time delay as it is known that a server location has an impact on response time and page loading if it is far from the user's location.



You've entered a domain name. We've found an IP address from the domain name you've entered. Your translated IP address is 54.235.137.115

Geolocation data from IP2Location (Product: DB6, updated on 2019-3-1)

Domain Name	Country	Region	City
herokuapp.com	United States	Virginia	Ashburn

Figure 6.14: Server location

6.1.3. System Testing

A. Functional testing

To test if the functional requirements of each use case specified in the use case diagram (section 3) has been met. All the details of the test cases will be found at the Appendix E.

- Sign-up:

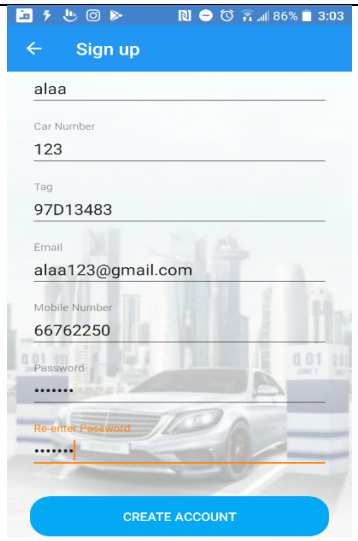
Table 6.4: Sign-up test cases

Test Case ID	Description	Screenshots

Sign-up 01	VIP user creates an account successfully.	For application see Table 6.5 For website see Table 6.8
Sign-up 02	VIP user uses an already existing email.	For application see Table 6.6 For website see Table 6.9
Sign-up 03	VIP user leaves required fields empty	For application see Table 6.7 For website see Table 6.10

Table 6.5: Sign-up 01 application screenshots

Sign-up 01



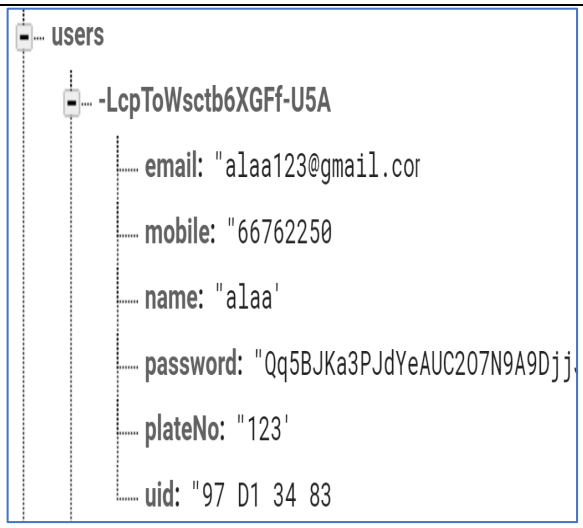


Table 6.6: Sign-up 02 application screenshots

Sign-up 02

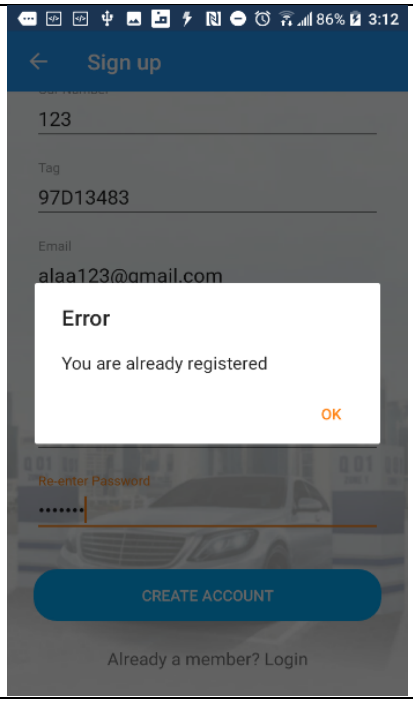


Table 6.7: Sign-up 03 application screenshots

Sign-up 03

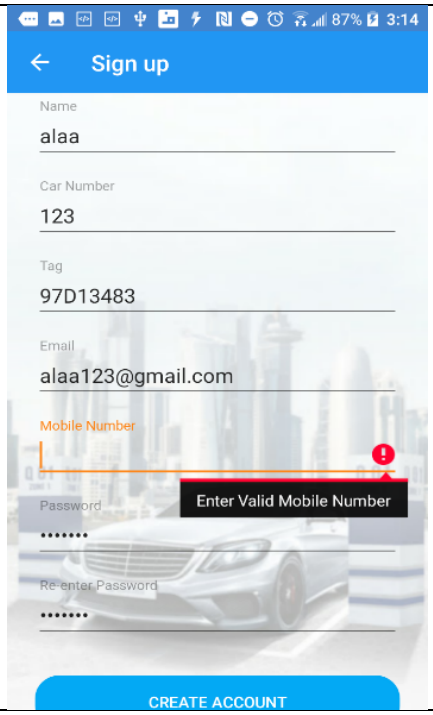


Table 6.8: Sign-up 01 website screenshots

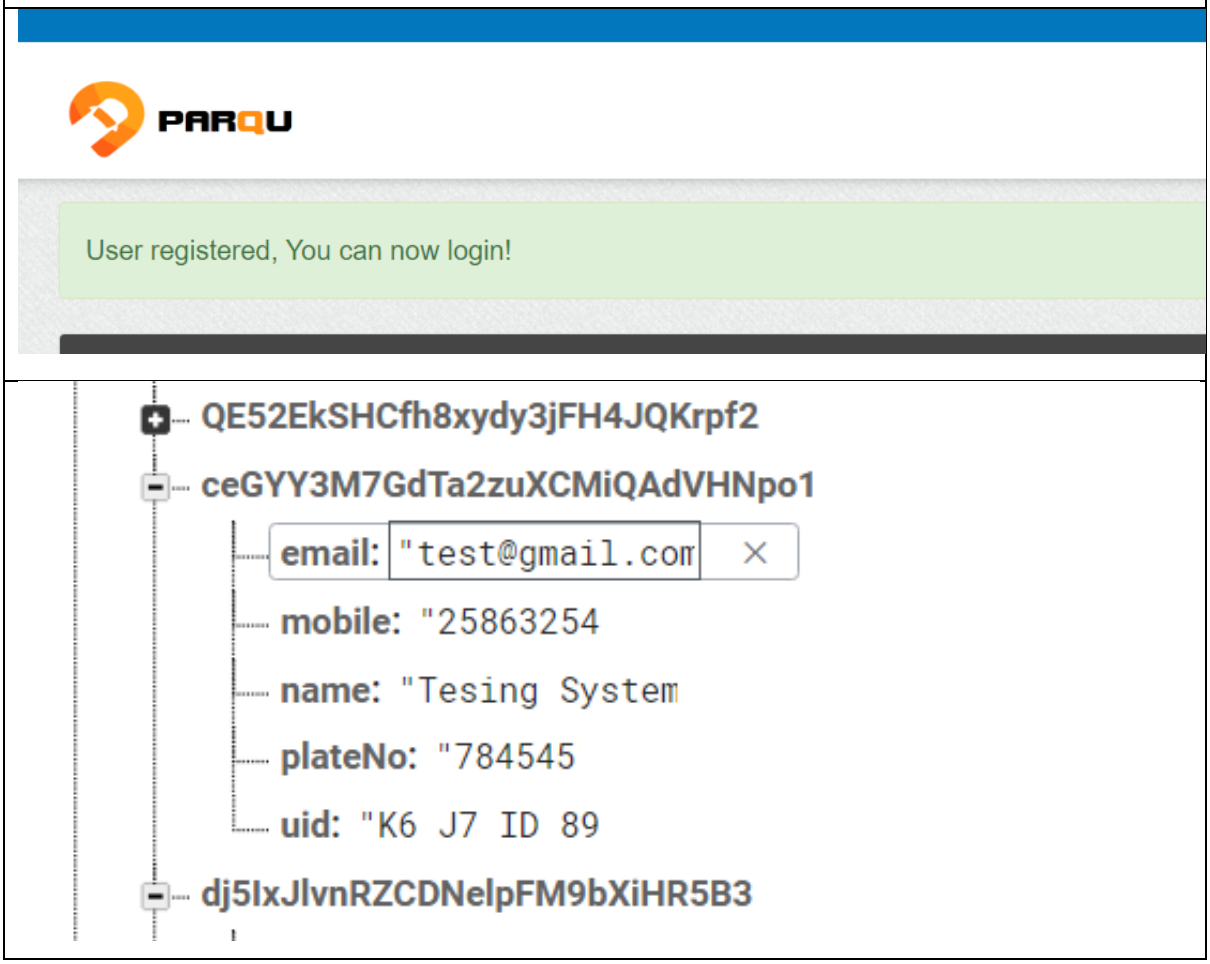
Sign-up 01


Table 6.9: Sign-up 02 website screenshots

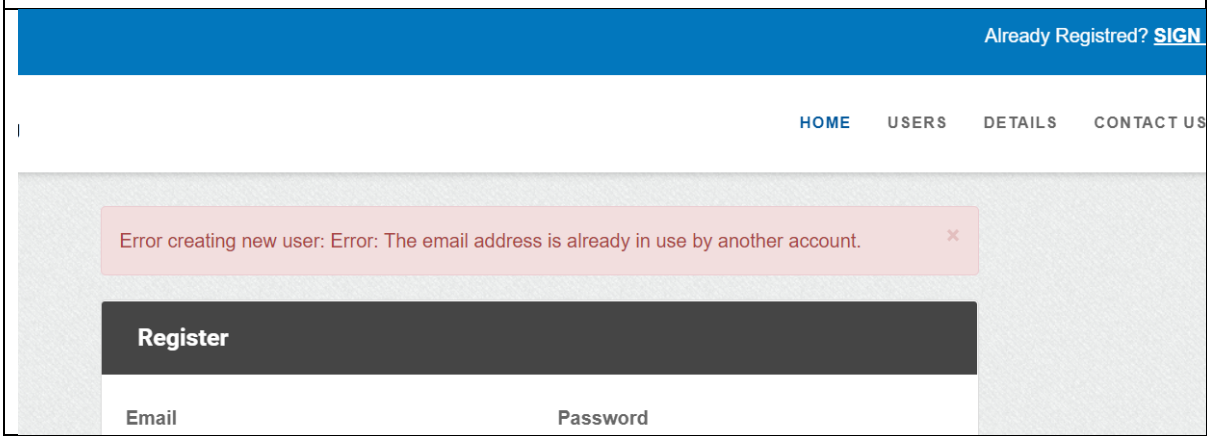
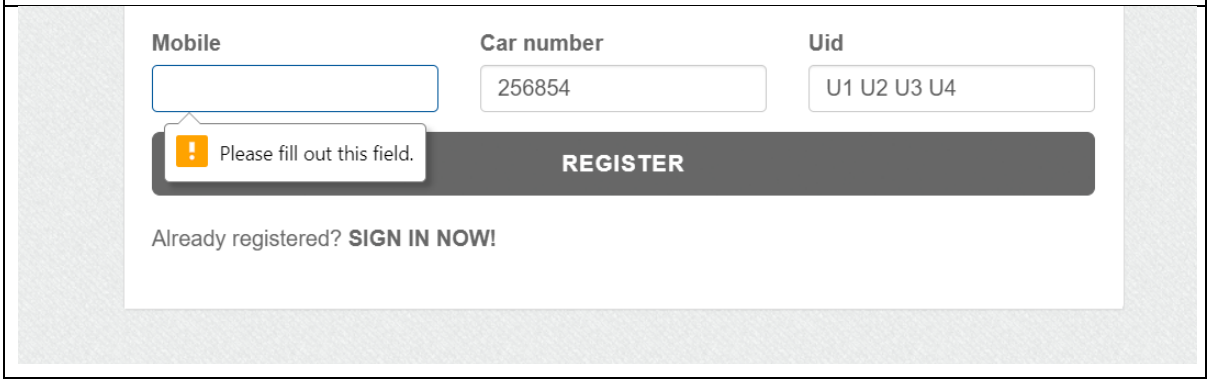
Sign-up 02


Table 6.10: Sign-up 03 website screenshots

Sign-up 03


- Sign-in

Table 6.11: Sign-in test cases

Test Case ID	Description	Screenshots
Sign-in 01	VIP user signs in successfully.	For application see Table 6.12 For website see Table 6.14
Sign-in 02	VIP user uses a wrong email address and/or password.	For application see Table 6.13 For website see Table 6.15

Table 6.12: Sign-in 01 application screenshots

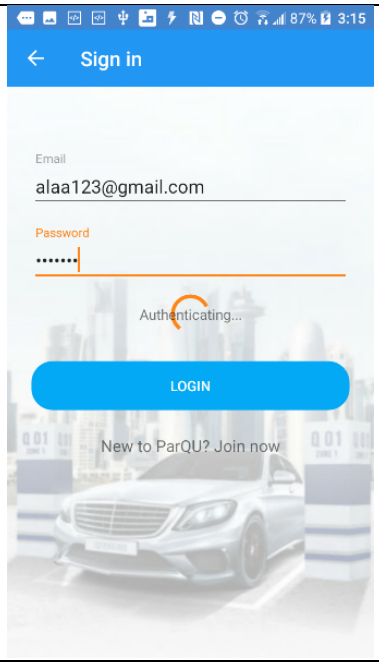
Sign-in 01


Table 6.13: Sign-in 02 application screenshots

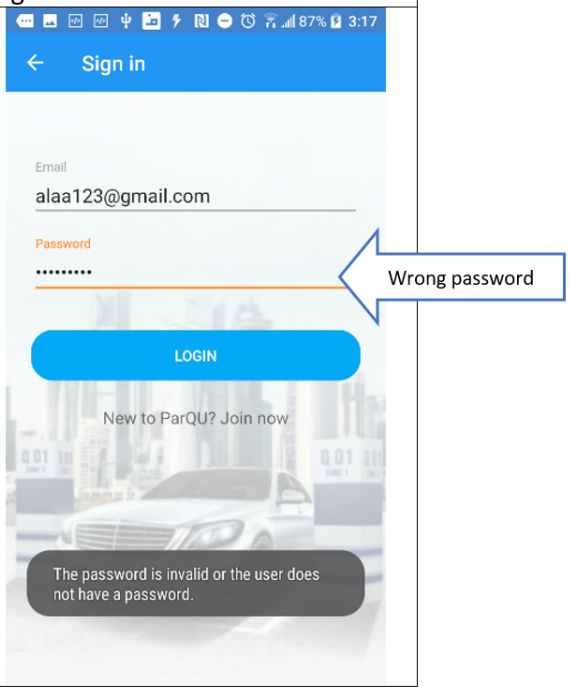
Sign-in 02


Table 6.14 sign-in 01 website screenshot

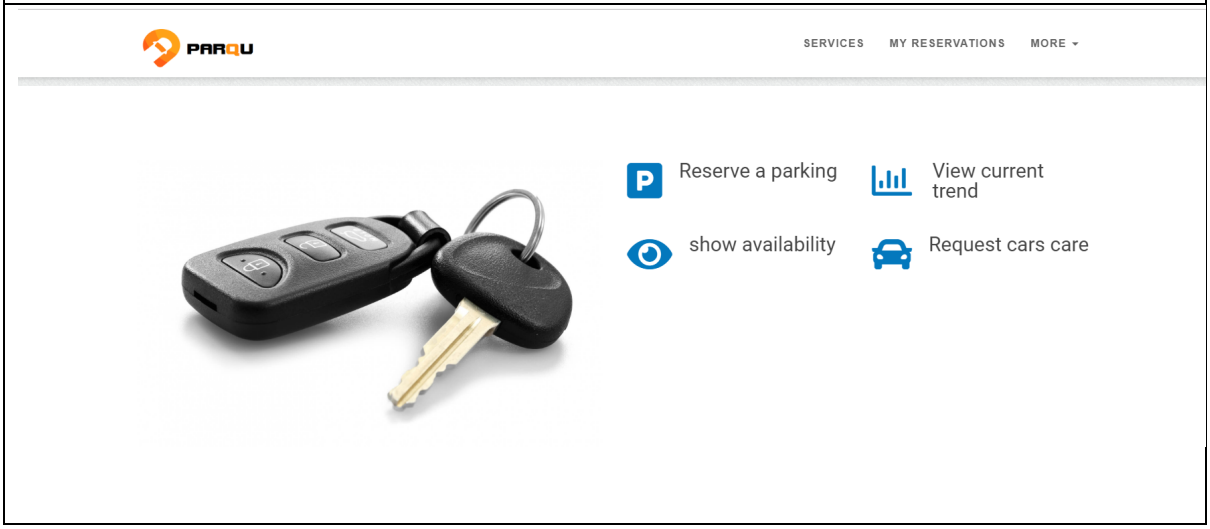
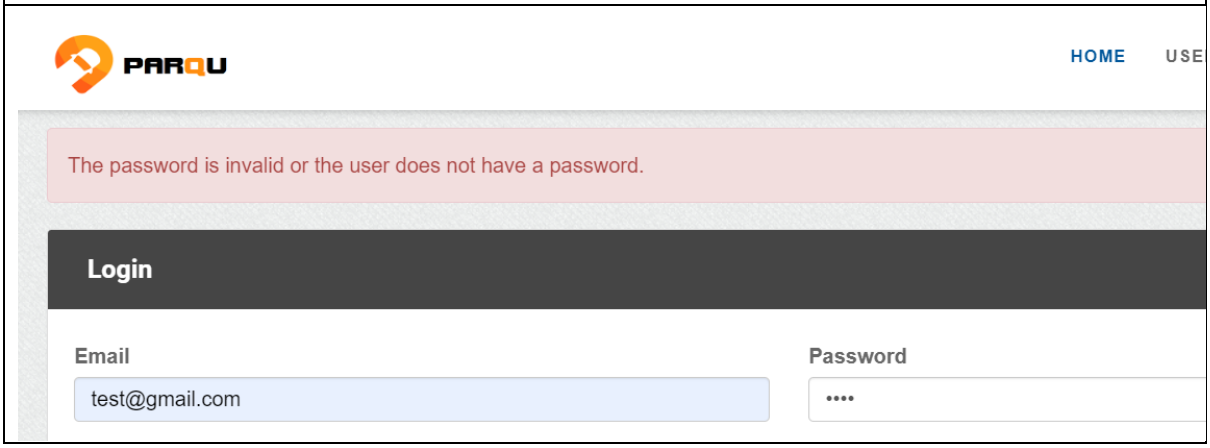
Sign-in 01


Table 6.15 Sign-in 02 website screenshot

Sign-in 02


- Reserve Parking:

Table 6.16: Reserve parking test cases

Test Case ID	Description	Screenshots
Reserve Parking 01	VIP user reserves a parking spot successfully.	For application see Table 6.17 For website see Table 6.24
Reserve Parking 02	VIP user already has a reservation at the selected time he/she wants to reserve on.	For application see Table 6.18 For website see Table 6.25

Reserve Parking 03	VIP user tries to reserve at a time where there is no available parking spot.	For application see Table 6.19 For website see Table 6.26
Reserve Parking 04	VIP user tries to reserve before the allowable reservation time which is same day or one day before.	For application see Table 6.20 For website see Table 6.27
Reserve Parking 05	VIP user tries to reserve at a time that has elapsed.	For application see Table 6.22 For website see Table 6.28
Reserve Parking 06	VIP user tries to reserve more than the number of allowable reservation hours per day (6 hours)	For application see Table 6.23 For website see Table 6.29
Reserve Parking 07	VIP user is notified 30 minutes before reservation expiring time.	For application see Table 6.21

Table 6.17: Reserve parking 01 application screenshots

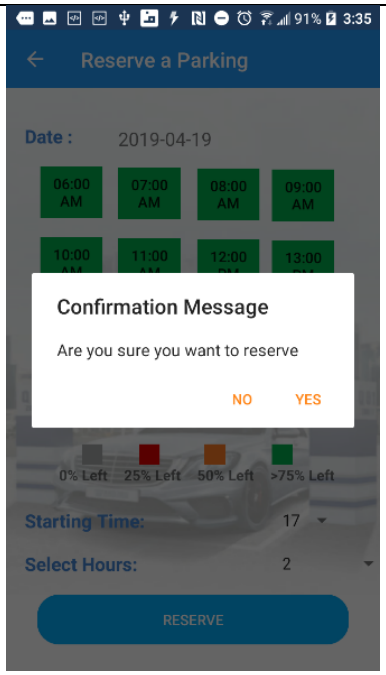
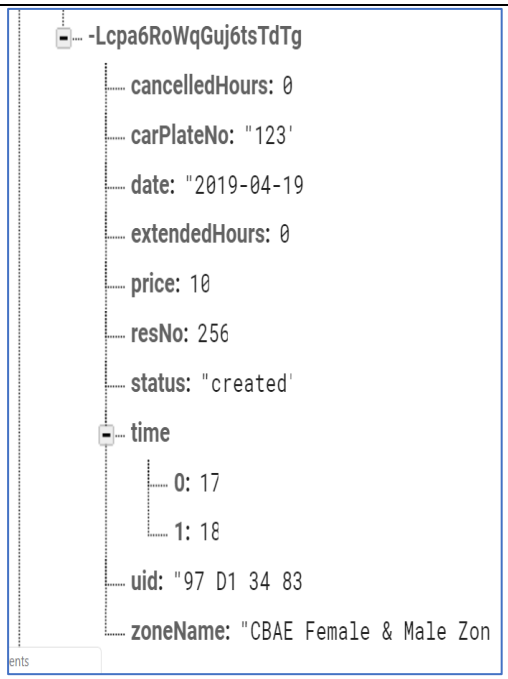
Reserve Parking 01	
	

Table 6.18: Reserve parking 02 application screenshots

Reserve Parking 02

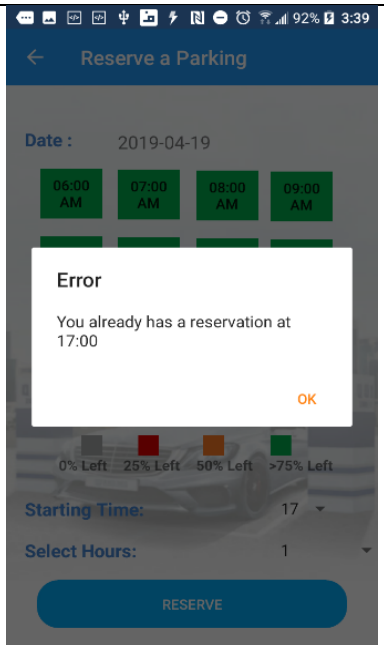
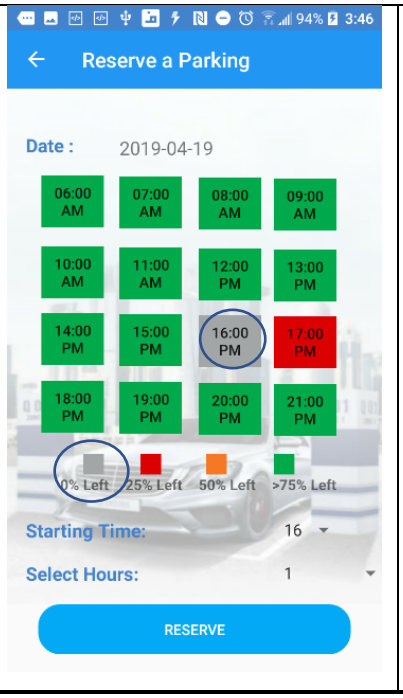
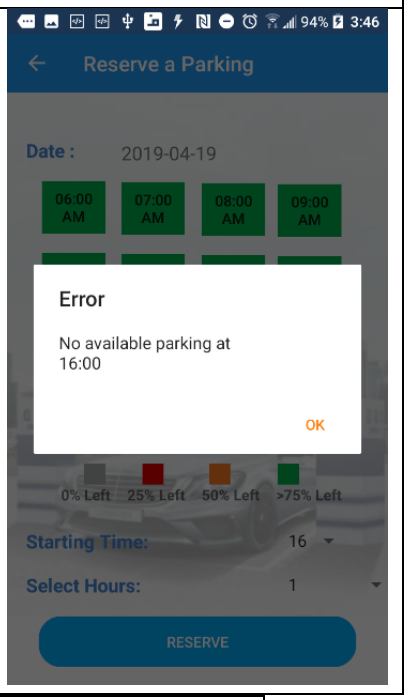


Table 6.19: Reserve parking 03 application screenshots

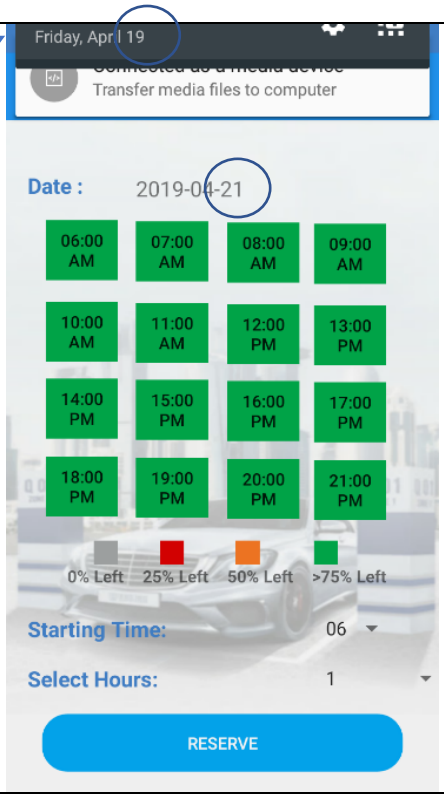
Reserve Parking 03





Reserve Parking 04

Friday, April 19



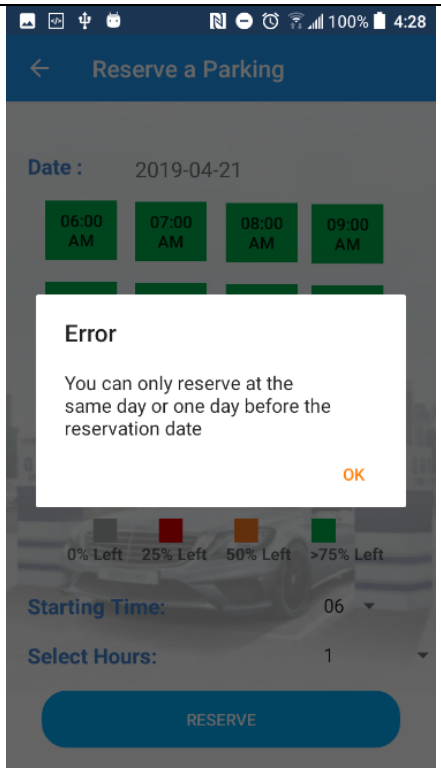


Table 6.20: Reserve parking 04 application screenshots

Table 6.22: Reserve parking 05 application screenshots

Reserve Parking 05

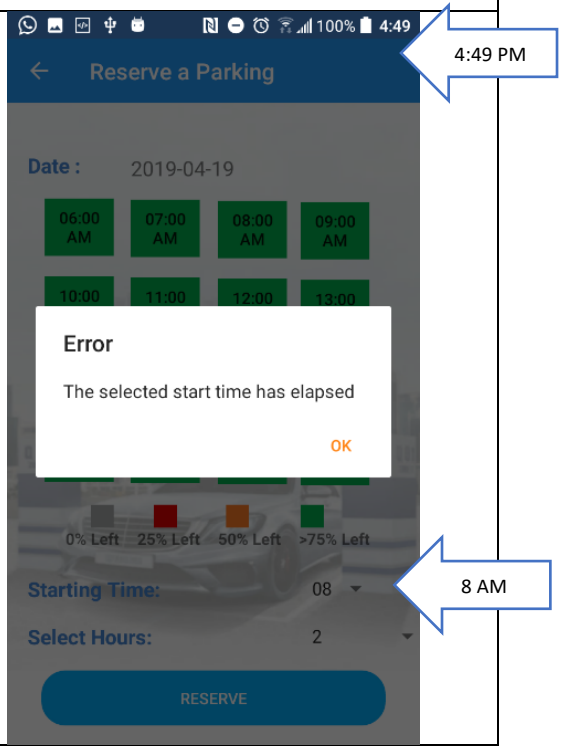


Table 6.21: Reserve parking 07 application screenshots

Reserve Parking 07

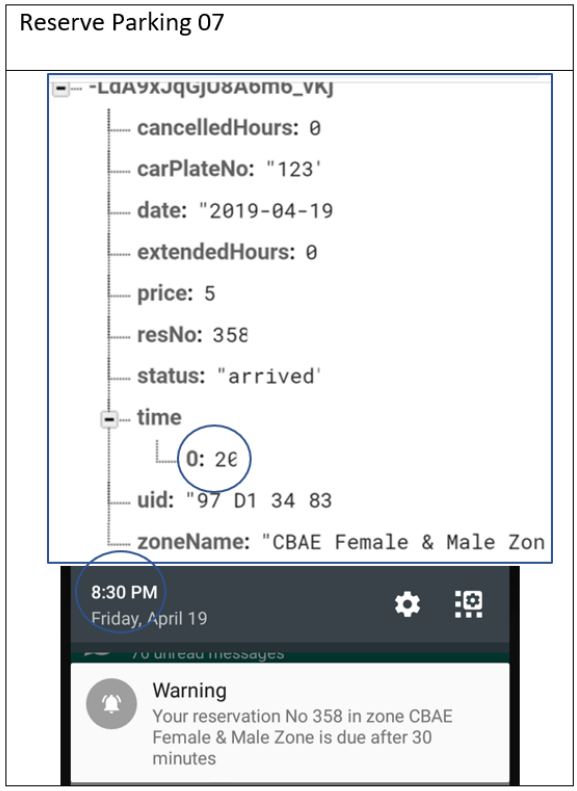


Table 6.23: Reserve parking 06 application screenshots

Reserve Parking 06

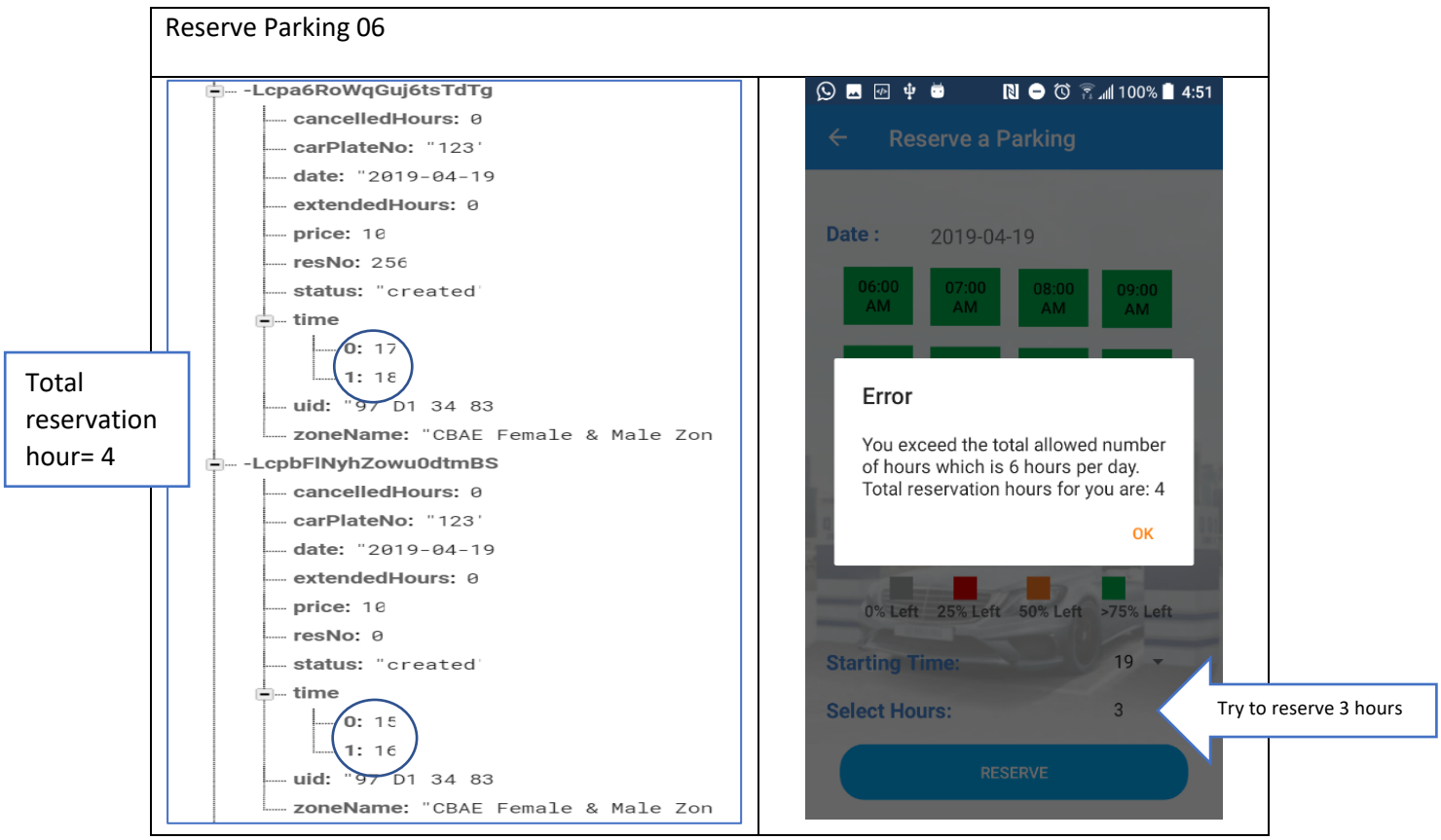


Table 6.24 Reserve Parking 01 website screenshot

Reserve Parking 01

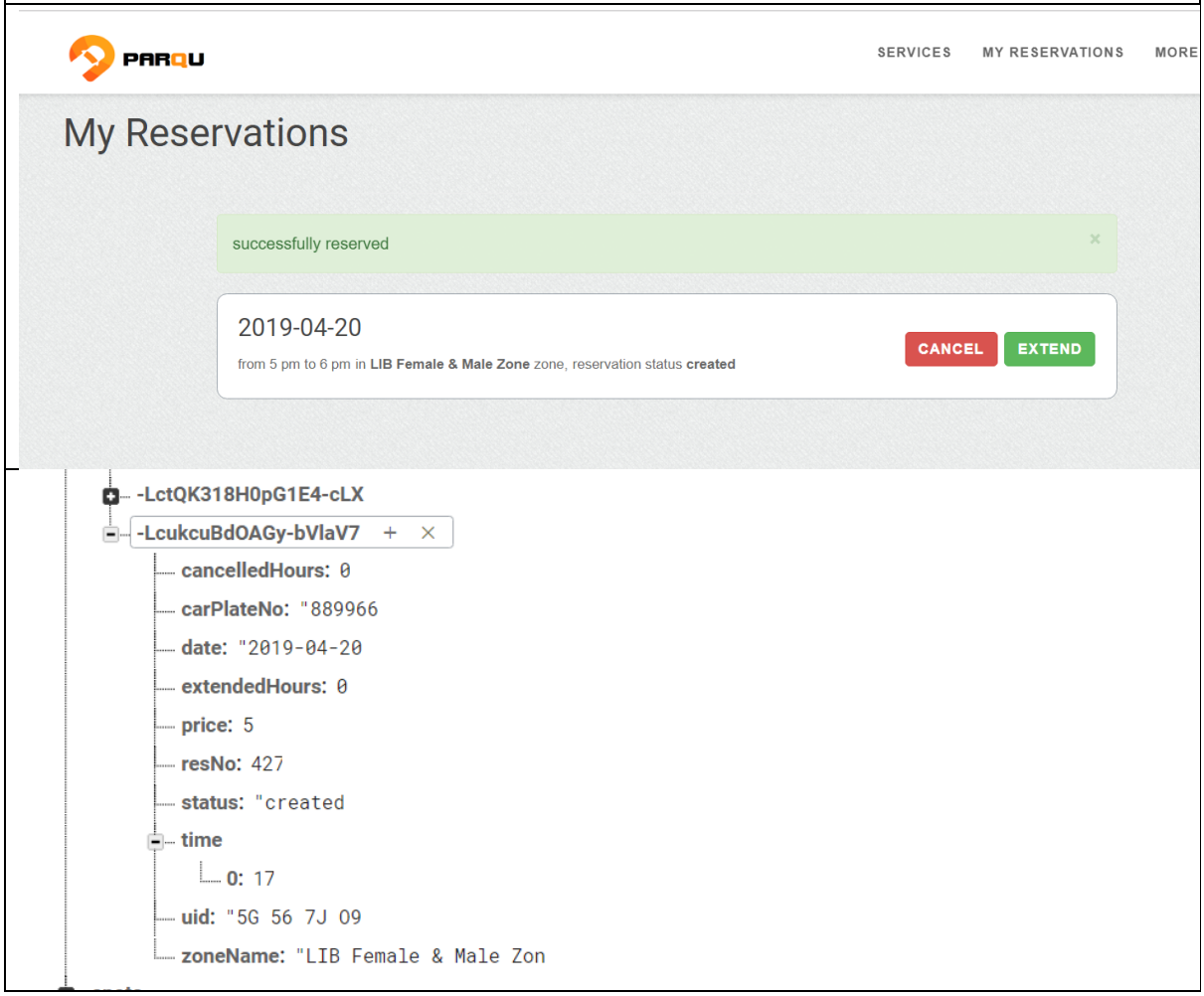


Table 6.25 Reserve parking 02 website screenshot

Reserve Parking 02

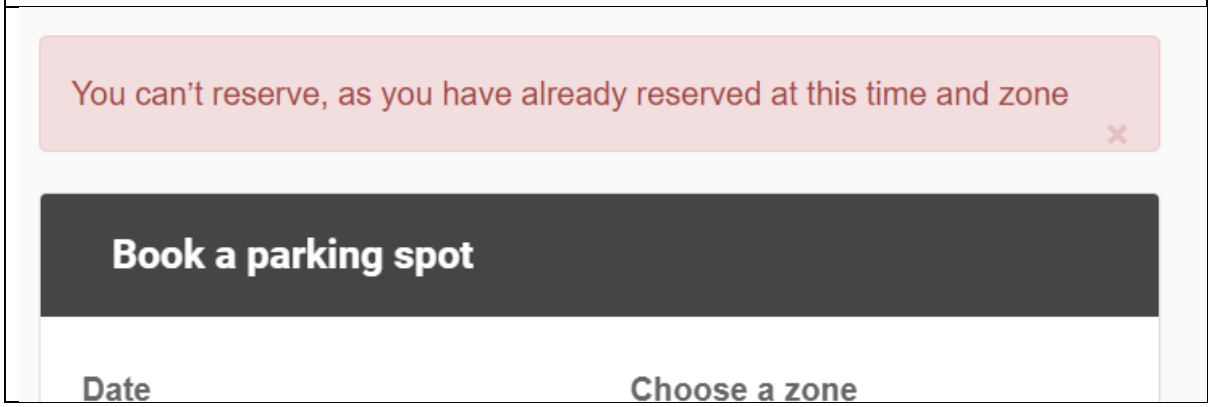


Table 6.26 Reserve Parking 03 website screenshot

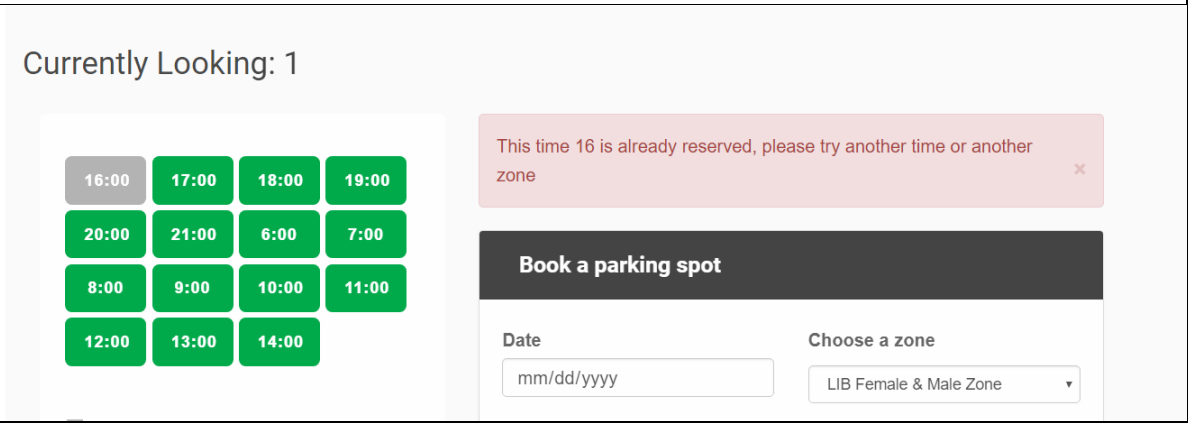
Reserve Parking 03


Table 6.27 Reserve Parking 04 website screenshot

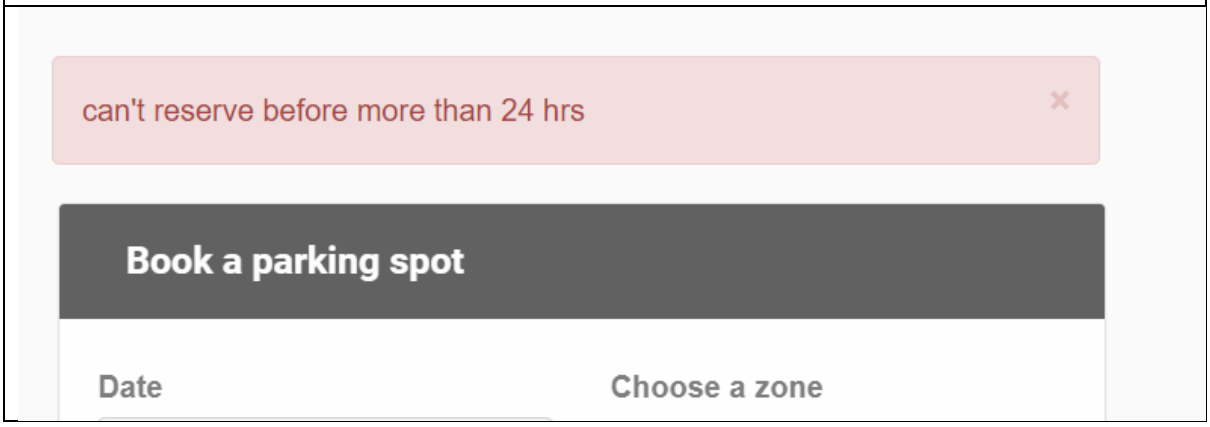
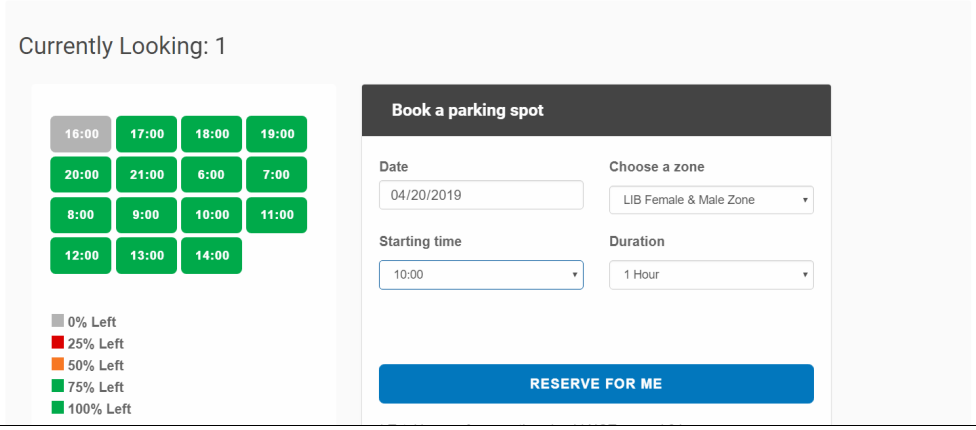
Reserve Parking 04


Table 6.28 Reserve Parking 05 website screenshots

Reserve Parking 05


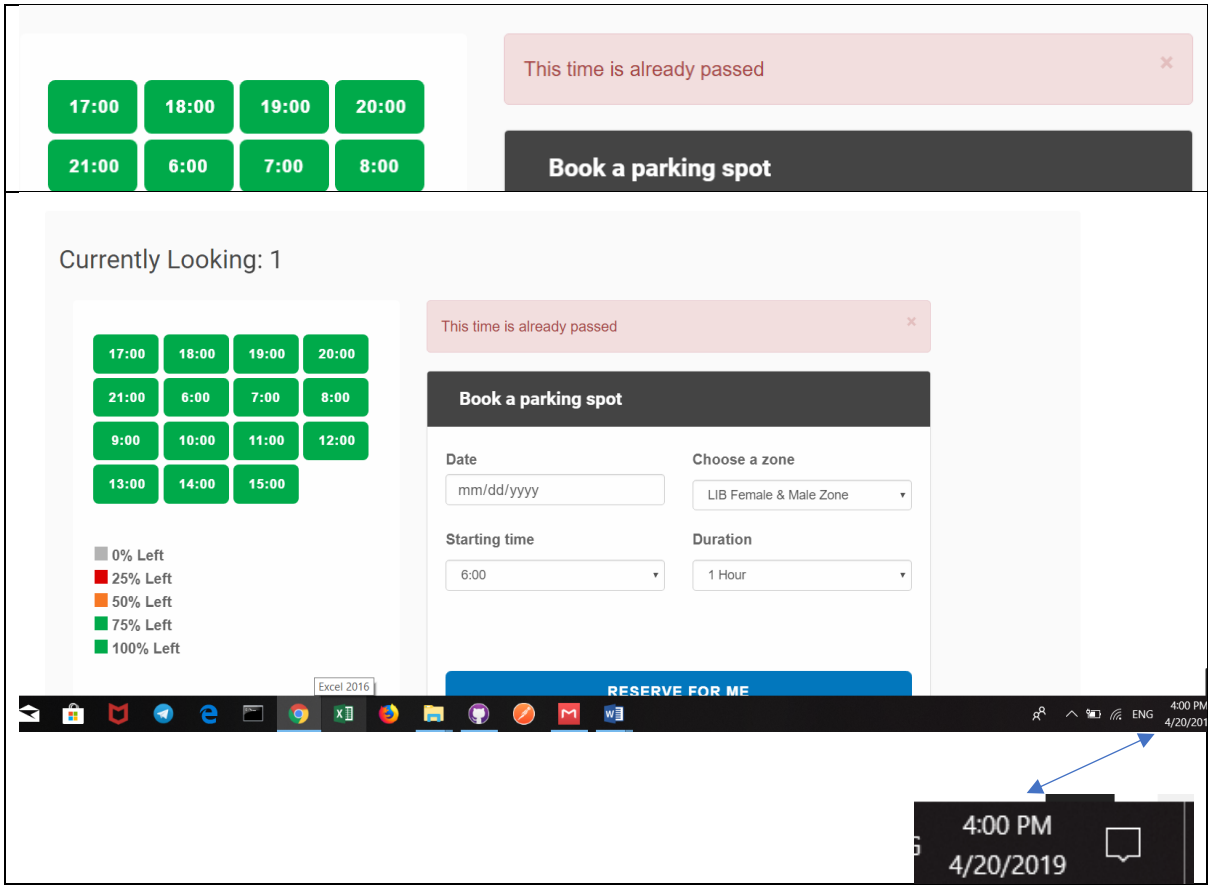
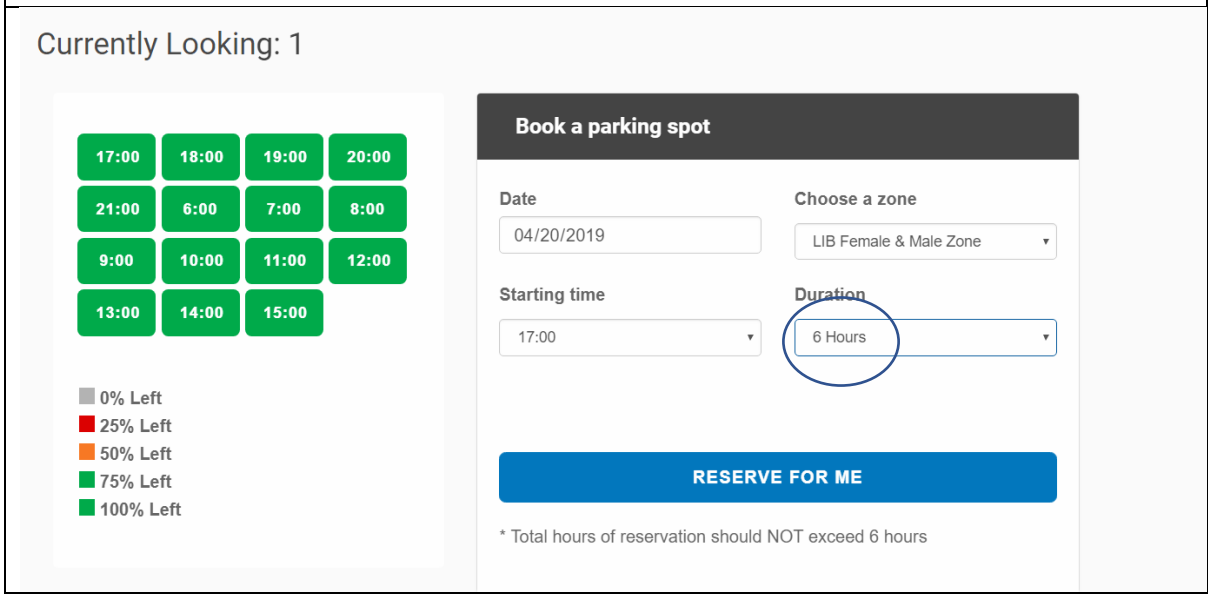
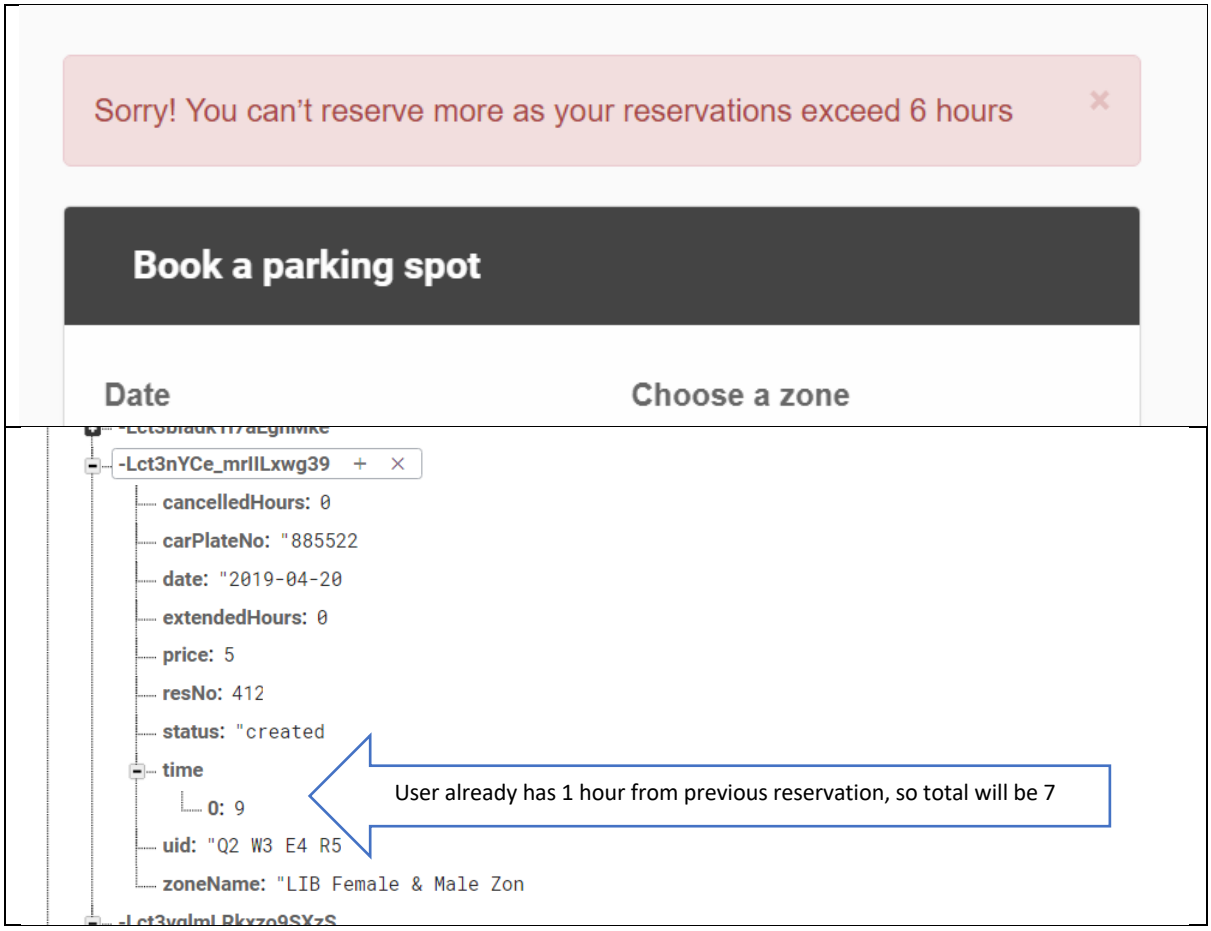


Table 6.29 Reserve Parking 06 website screenshots

Reserve Parking 06





- View Reservation:

Table 6.30: View reservation test cases

Test Case ID	Description	Screenshots
View Reservation 01	VIP user successfully views all his/her current and upcoming reservations.	For application see Table 6.31 For website see Table 6.32

Table 6.31: View reservation application 01 screenshots

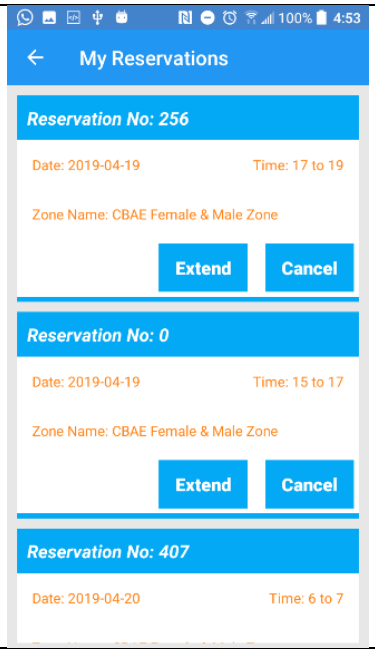
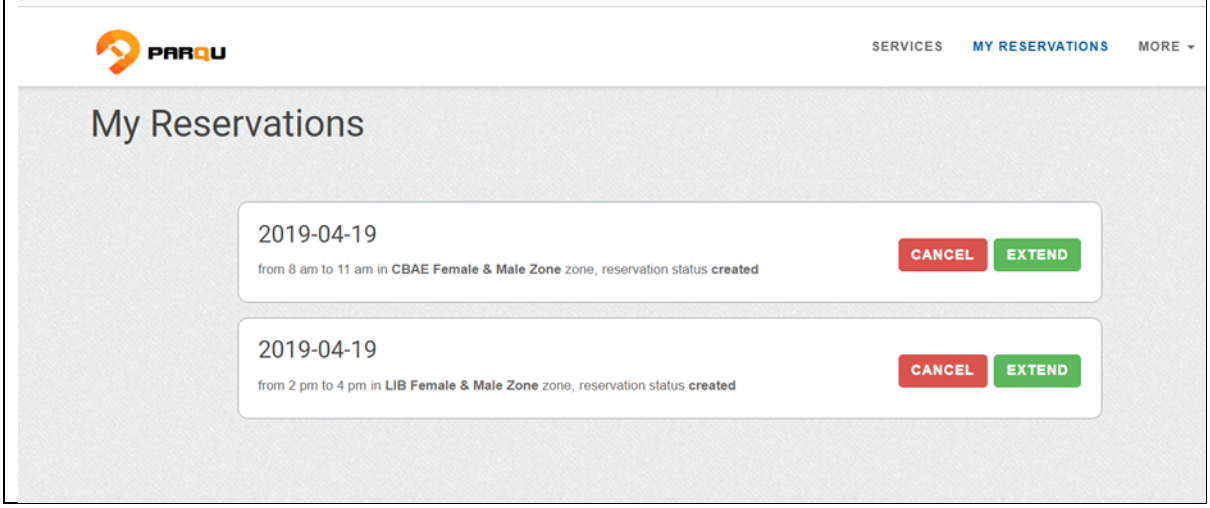
View Reservation 01


Table 6.32 View reservation 01 website screenshots

View Reservation 01


- Extend Reservation:

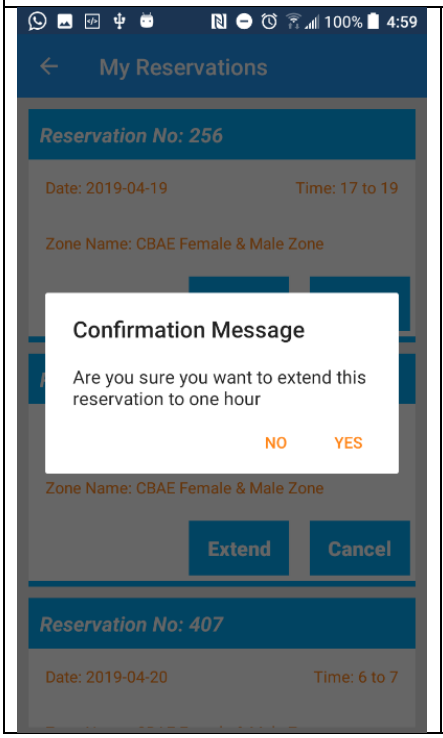
Table 6.33: Extend reservation test cases

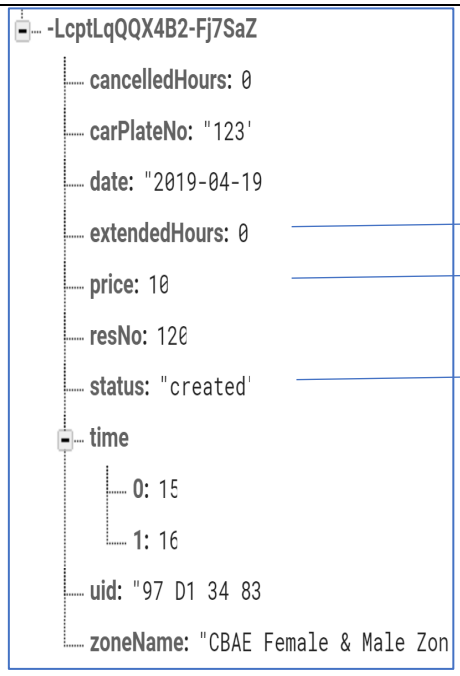
Test Case ID	Description	Screenshots
Extend Reservation 01	VIP user successfully extends a reservation and the extension price is added to the reservation.	For application see Table 6.34 For website see Table 6.37

Extend Reservation 02	VIP user tries to extend before the last hour of the reservation.	For application see Table 6.35 For website see Table 6.38
Extend Reservation 03	VIP user tries to extend where there is no available parking spot after the reservation time.	For application see Table 6.36 For website see Table 6.39

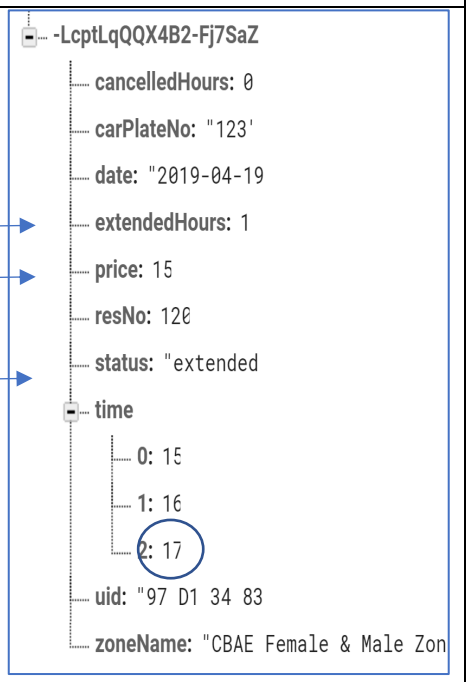
Table 6.34: Extend reservation 01 application screenshots

Extend Reservation 01





Before extending



After extending

Table 6.36: Extend reservation 02 application screenshots

Extend Reservation 02

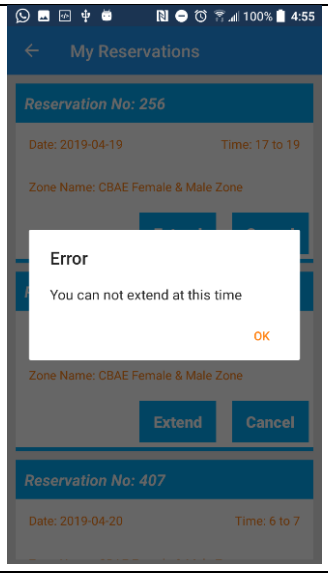


Table 6.35: Extend reservation 03 application screenshots

Extend Reservation 03

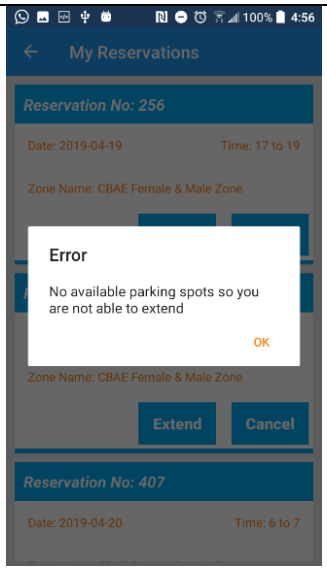


Table 6.37 Extend reservation 01 website screenshots

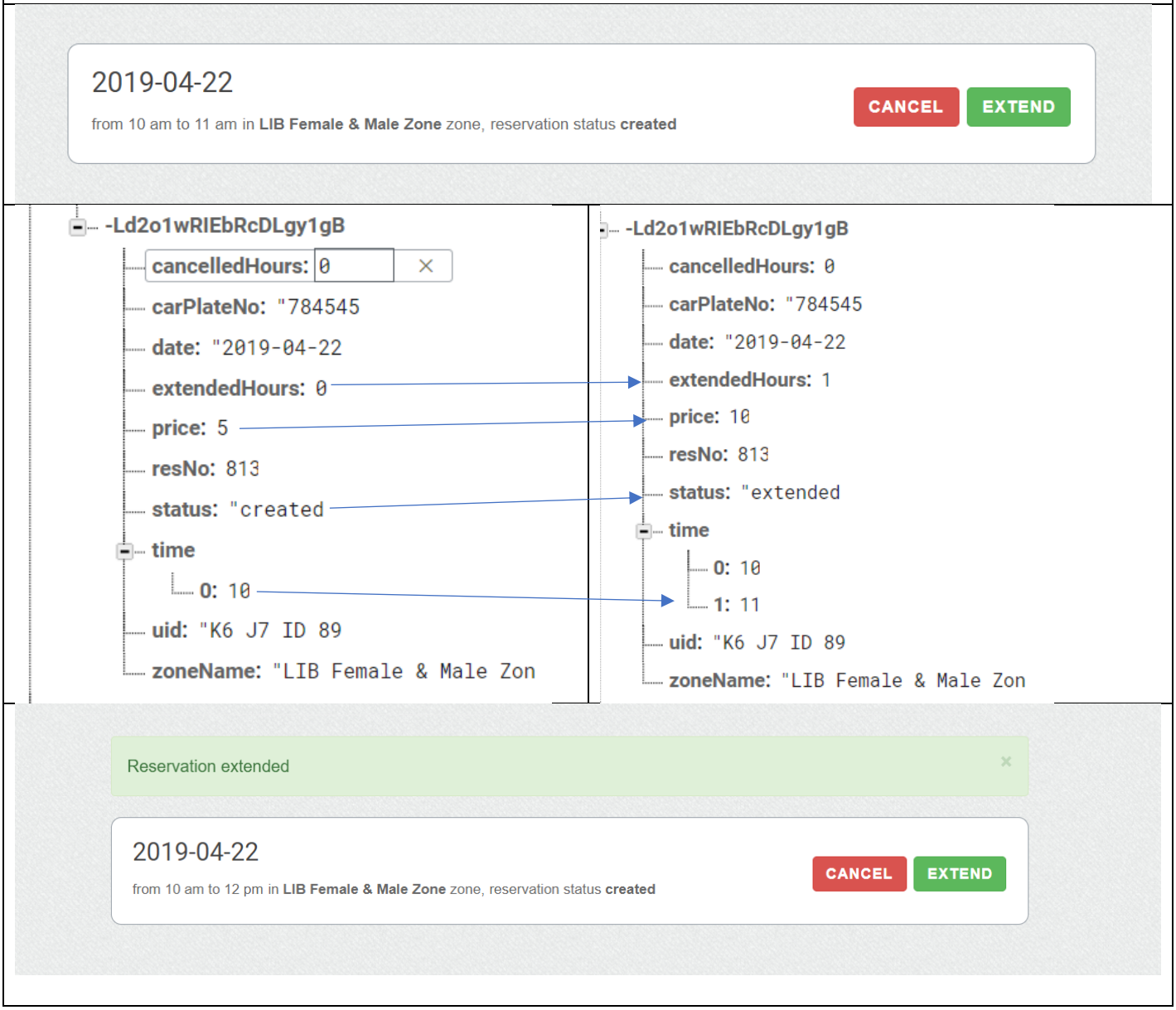
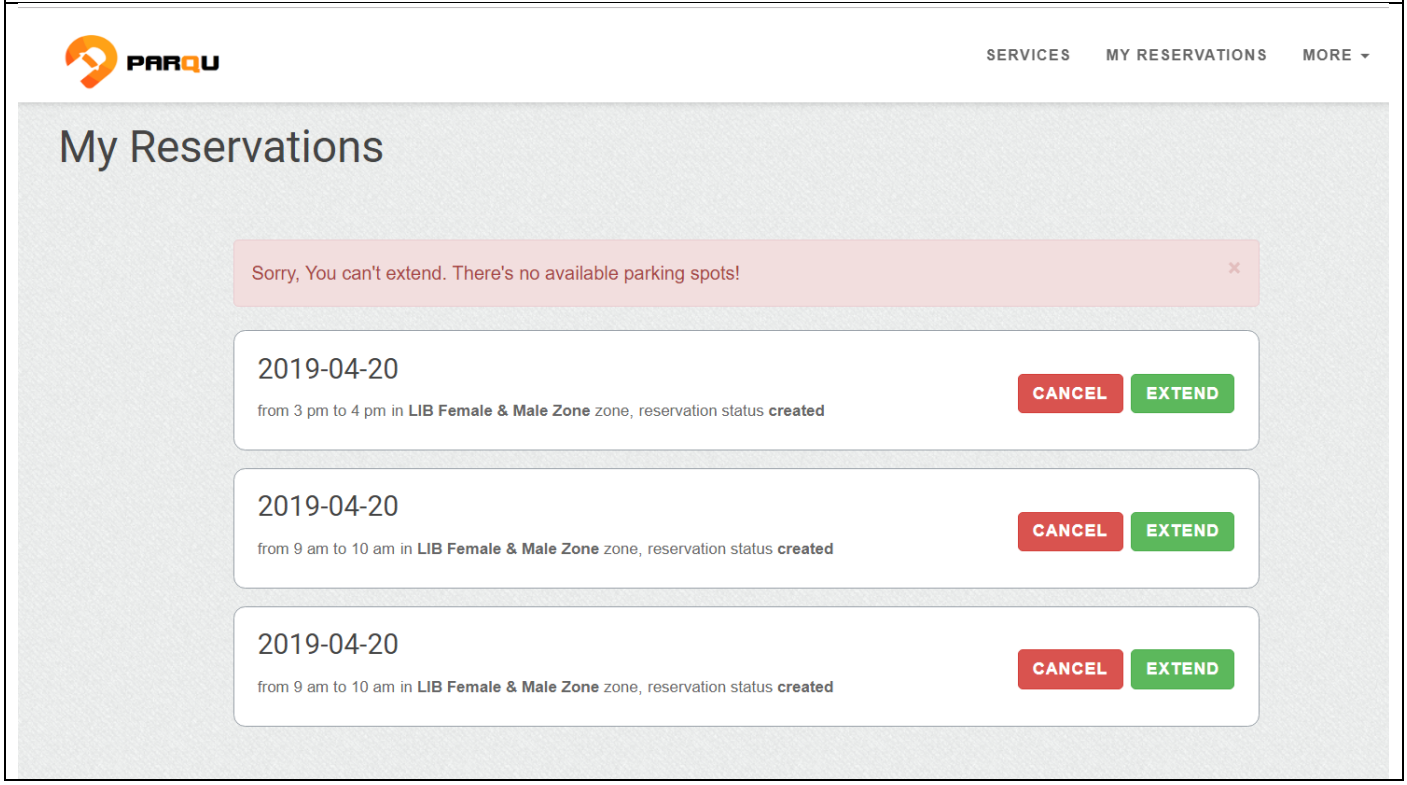
Extend Reservation 01


Table 6.38 Extend Reservation 02 website screenshot

Extend Reservation 02

Table 6.39 Extend reservation 03 website screenshot

Extend Reservation 03



- Cancel Reservation:

Table 6.40: Cancel reservation test cases

Test Case ID	Description	Screenshots
Cancel Reservation 01	VIP user successfully cancels the whole reservation and the deduction amount is calculated and deducted from the total price of the reservation, if the reservation has not started.	For application see Table 6.41 For website see Table 6.43

Cancel Reservation 02	VIP user successfully cancels remaining reservation hours from now and the deduction amount is calculated and deducted from the total price of the reservation, If the reservation has started	For application see Table 6.42 For website see Table 6.44

Table 6.41: Cancel reservation 01 application screenshots

Cancel Reservation 01

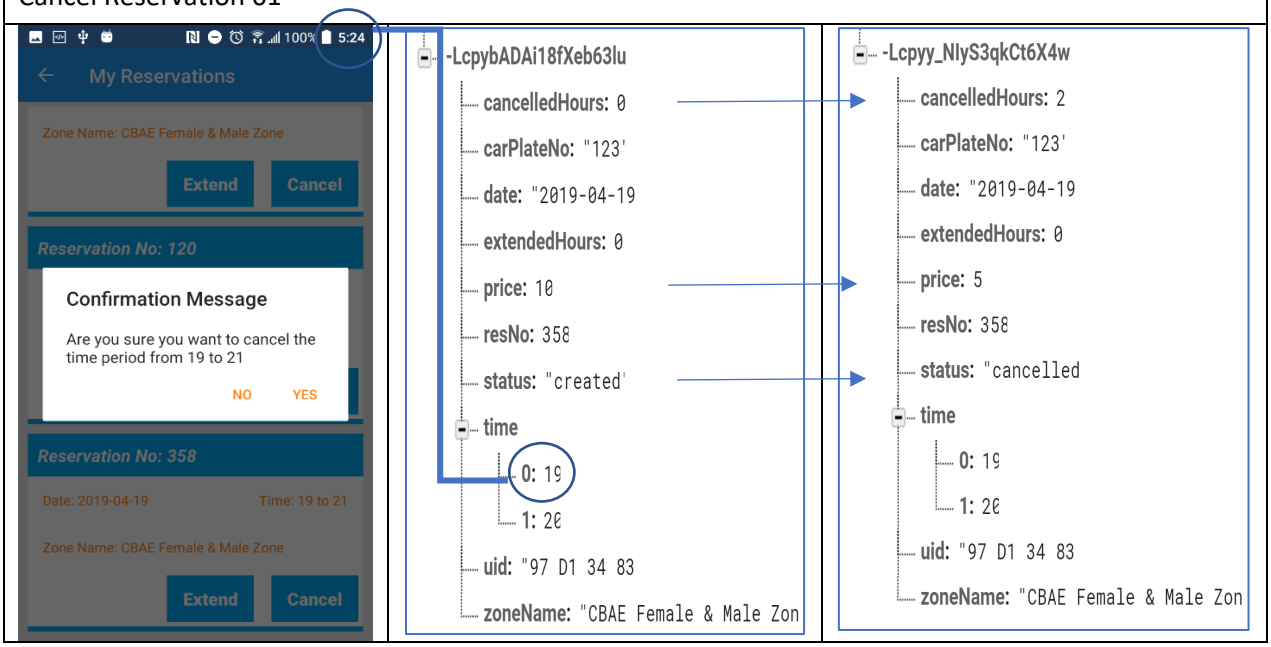


Table 6.42: Cancel reservation 02 application screenshots

Cancel Reservation 02

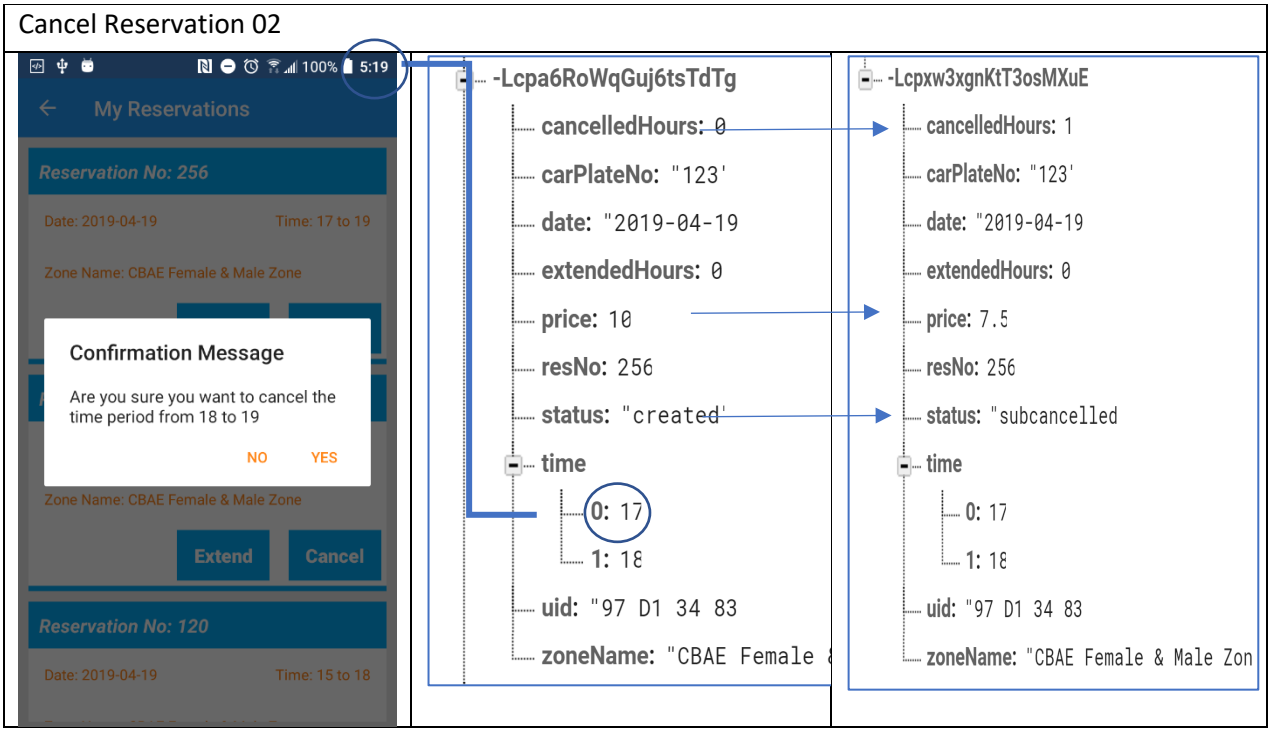


Table 6.43 Cancel Reservation 01 website screenshot

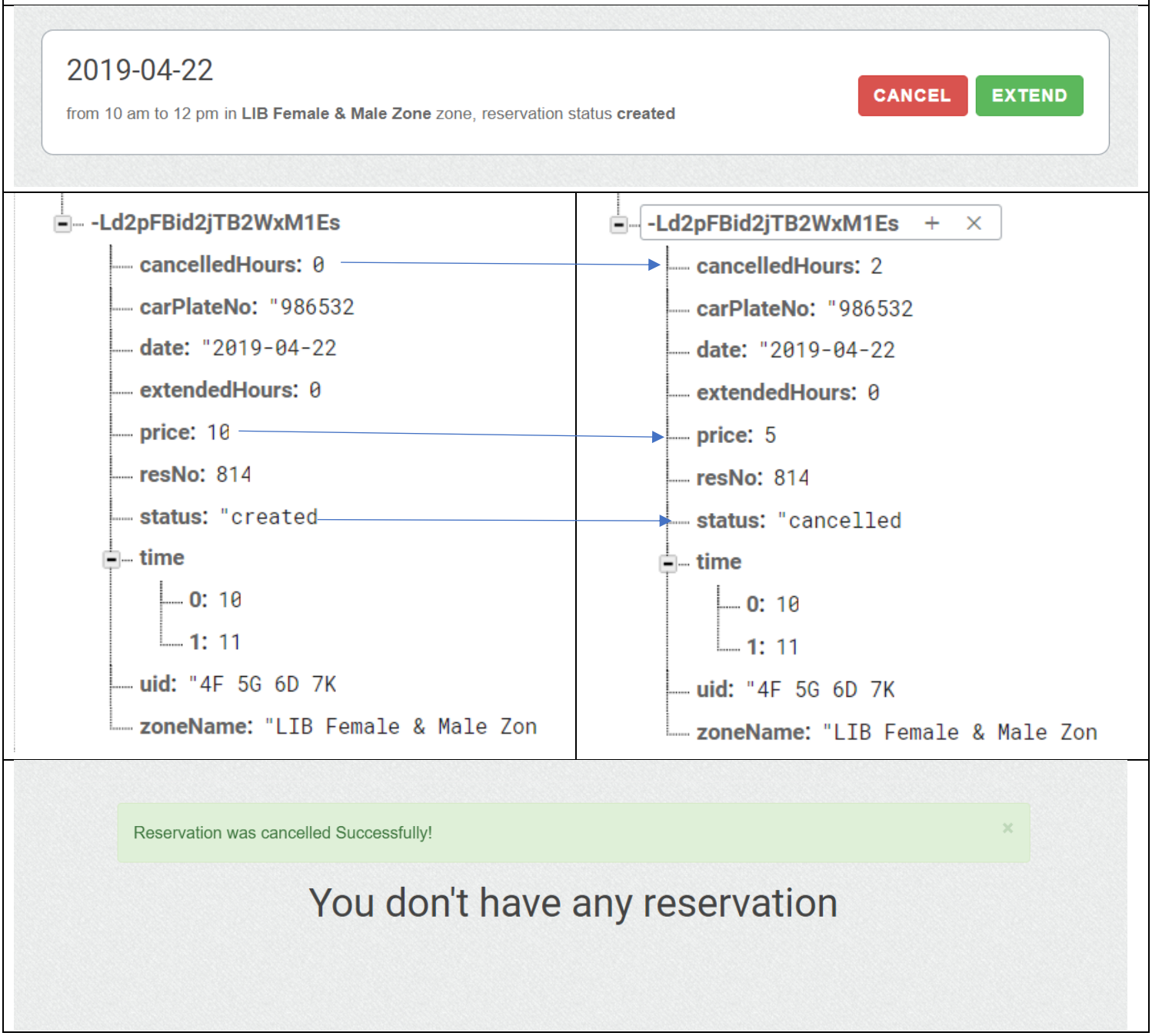
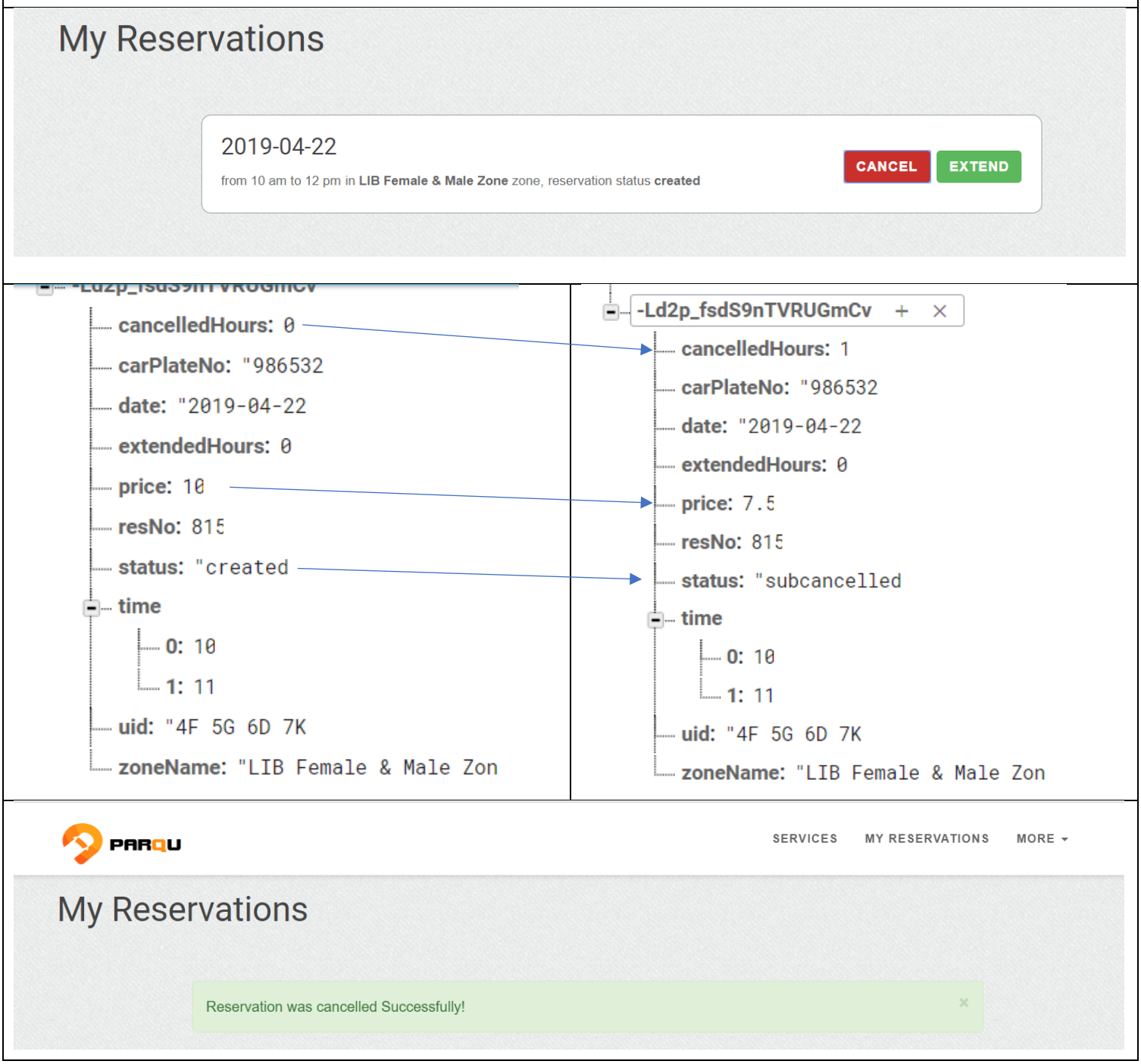
Cancel Reservation 01


Table 6.44 Cancel Reservation 02 website screenshot

Cancel Reservation 02



- View Parking:

Table 6.45: View parking test cases

Test Case ID	Description	Screenshots
View Parking 01	User successfully views a map of the parking spots current status.	For application see Table 6.46 For website see Table 6.48
View Parking 02	User successfully gets directions for a specific spot.	For application see Table 6.47 For website see Table 6.49

Table 6.46: View parking 01 application screenshots

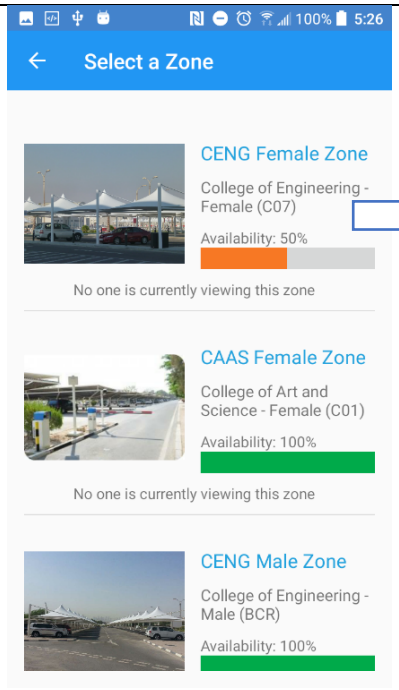
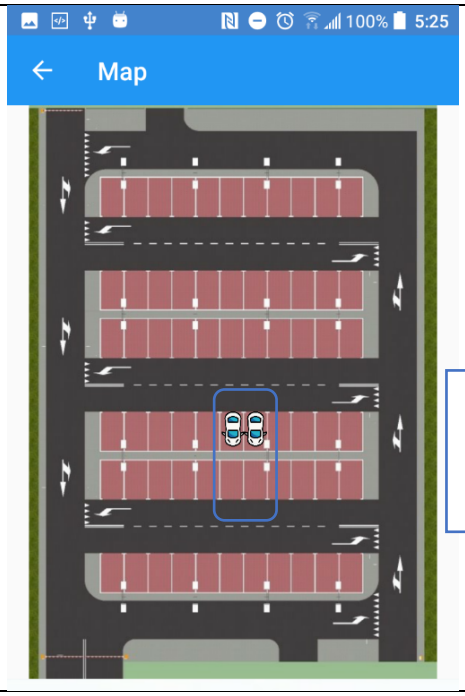
View Parking 01	
	 We applied the concept on 4 spots only

Table 6.47: View parking 02 application screenshots

View Parking 02

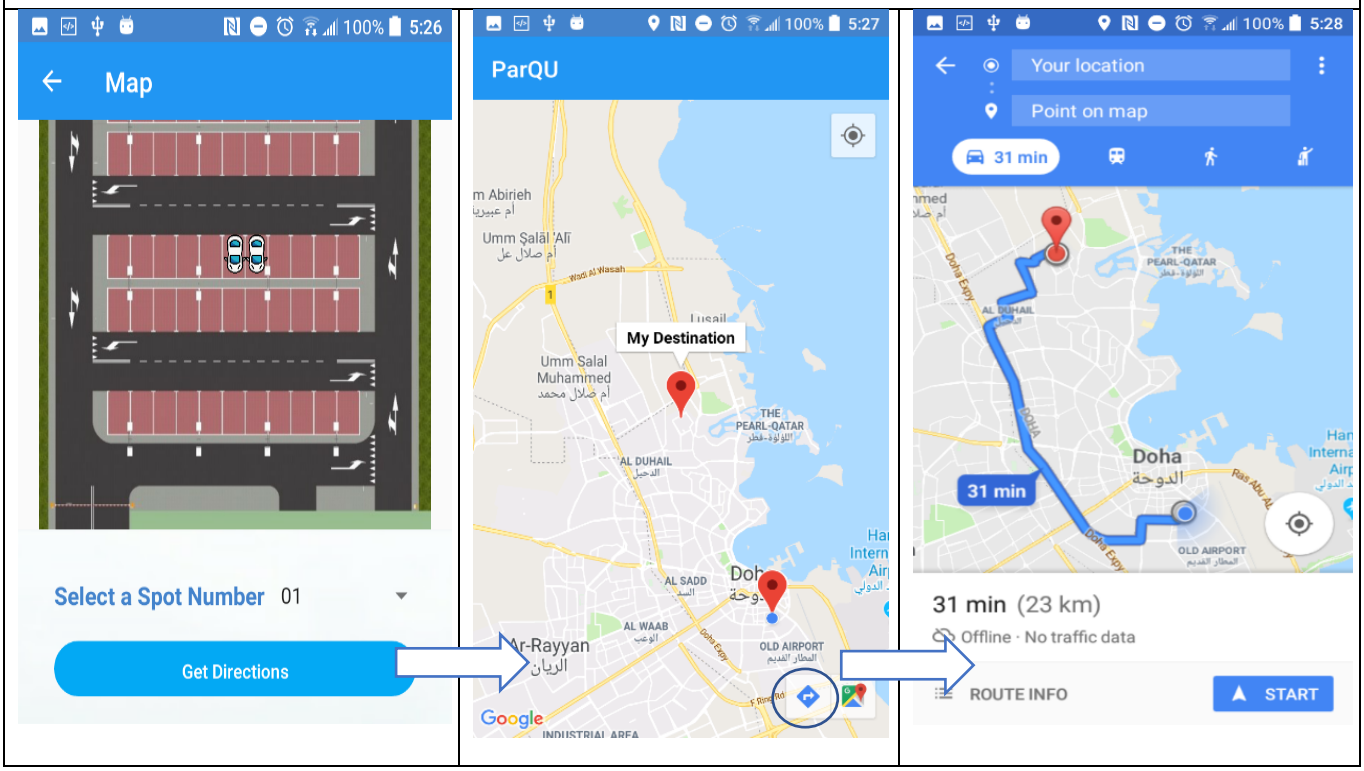


Table 6.48 View parking 01 website screenshots

View Parking 01

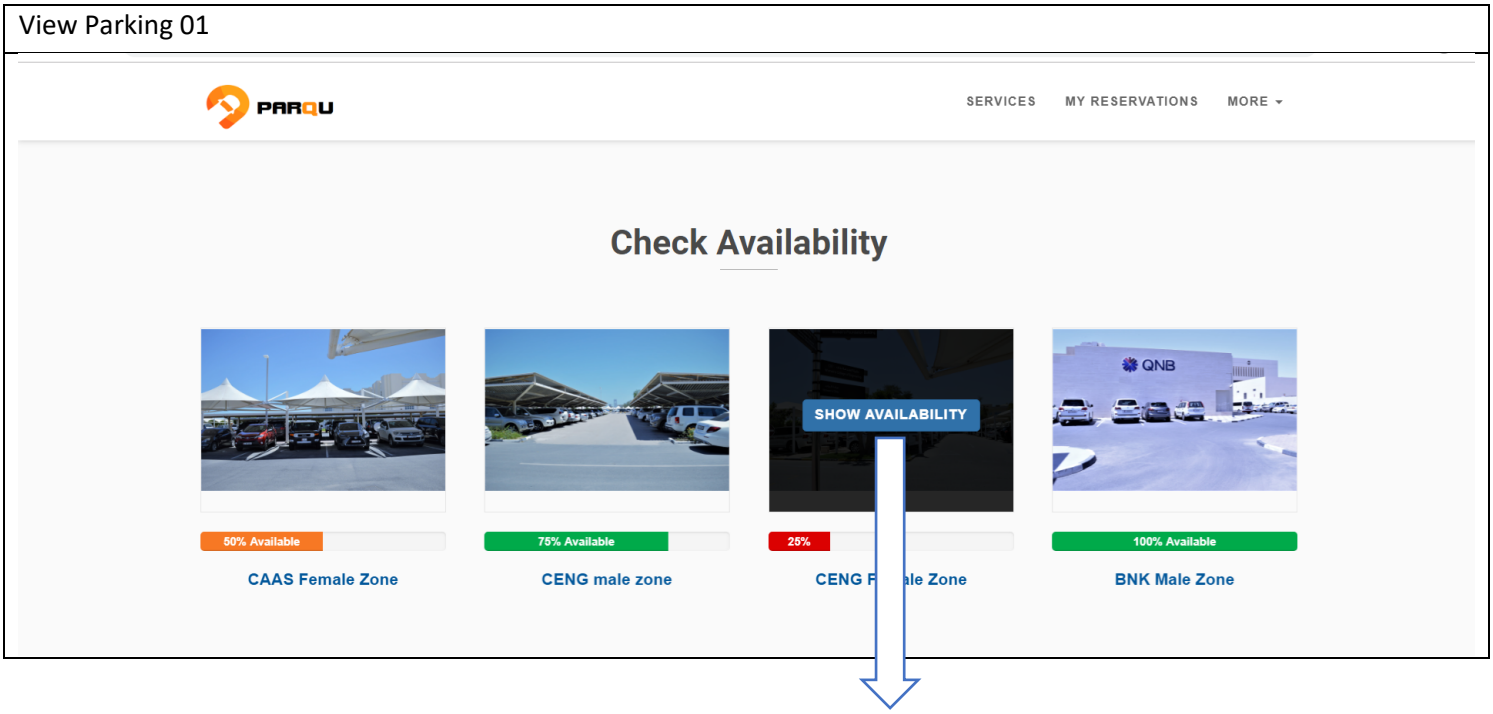
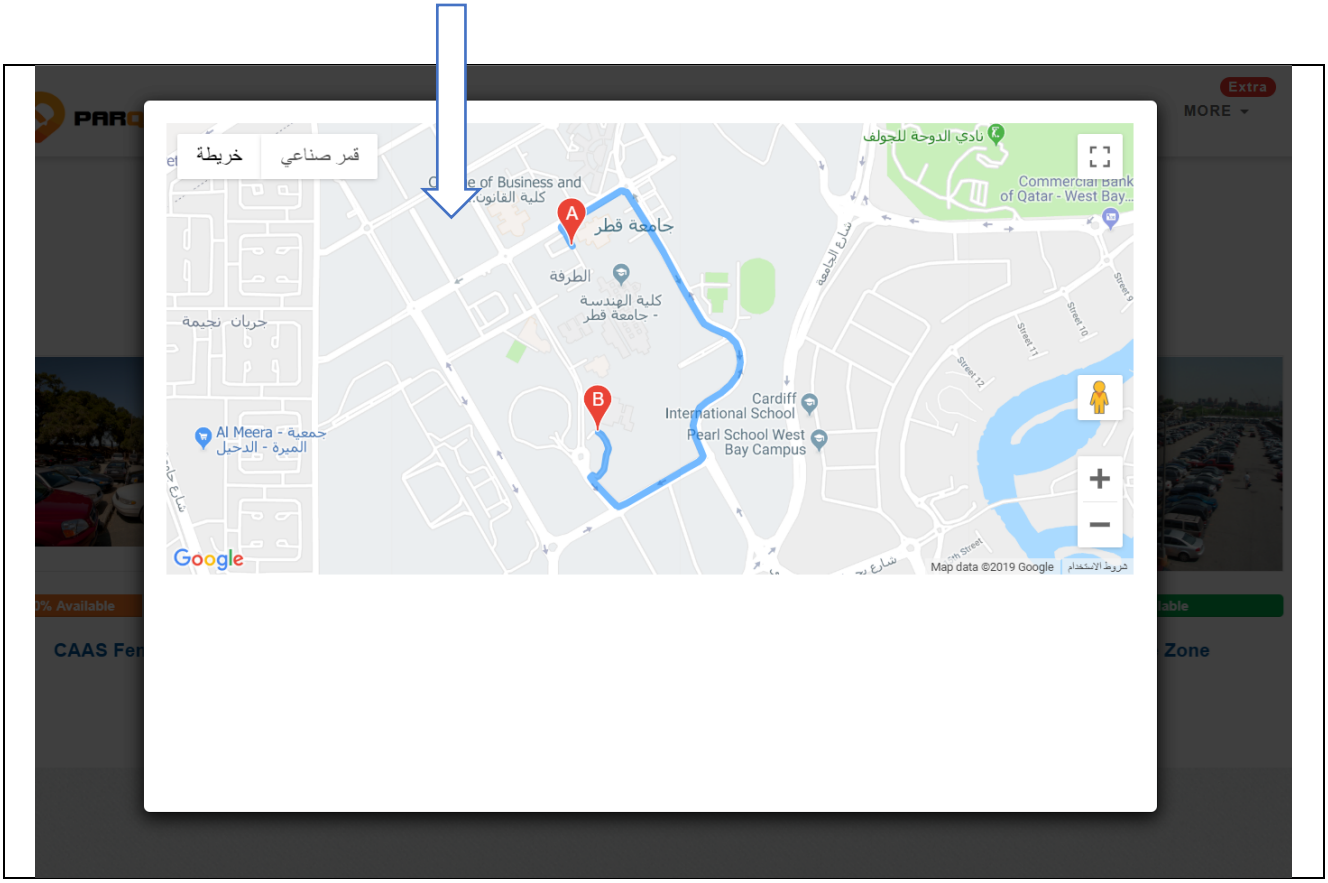


Table 6.49 View parking 02 website screenshots

View Parking 02

- Request Car Care:

Table 6.50: Request car care test cases

Test Case ID	Description	Screenshots
Request Car Care 01	VIP user is successfully redirected to the Servesni application or website, if the application is installed in the phone	For application see Table 6.51
Request Car Care 02	VIP user redirected to the Play store, if the application is not installed in the phone	For application see Table 6.52 For website see Table 6.53

Table 6.51: Request car care 01 application screenshots

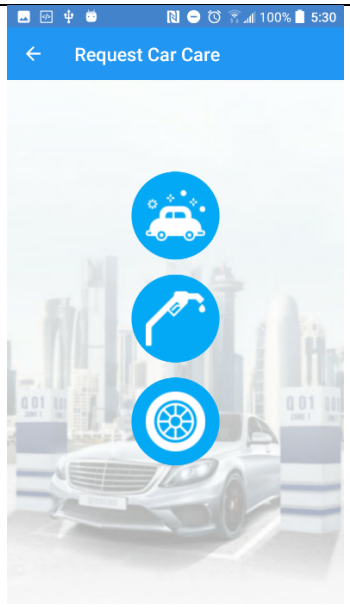
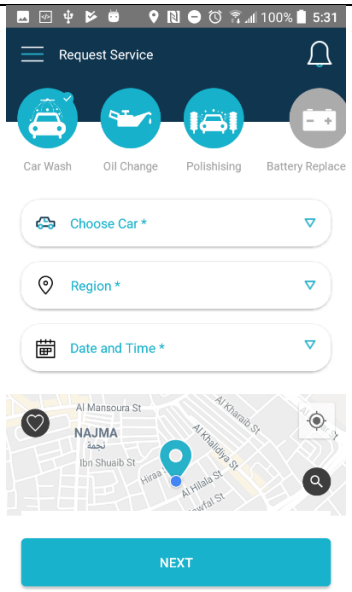
Request Car Care 01	
	

Table 6.52: Request car care 02 application screenshots

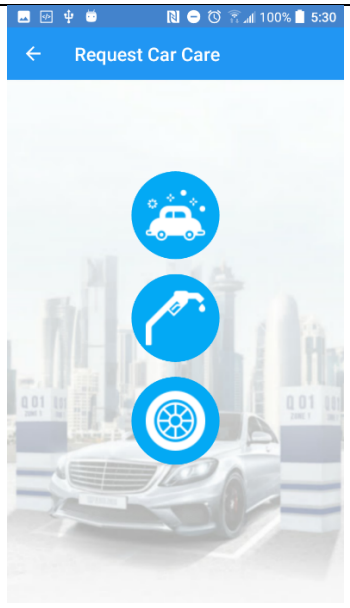
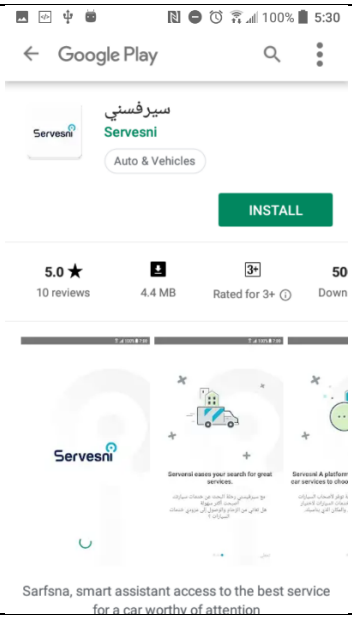
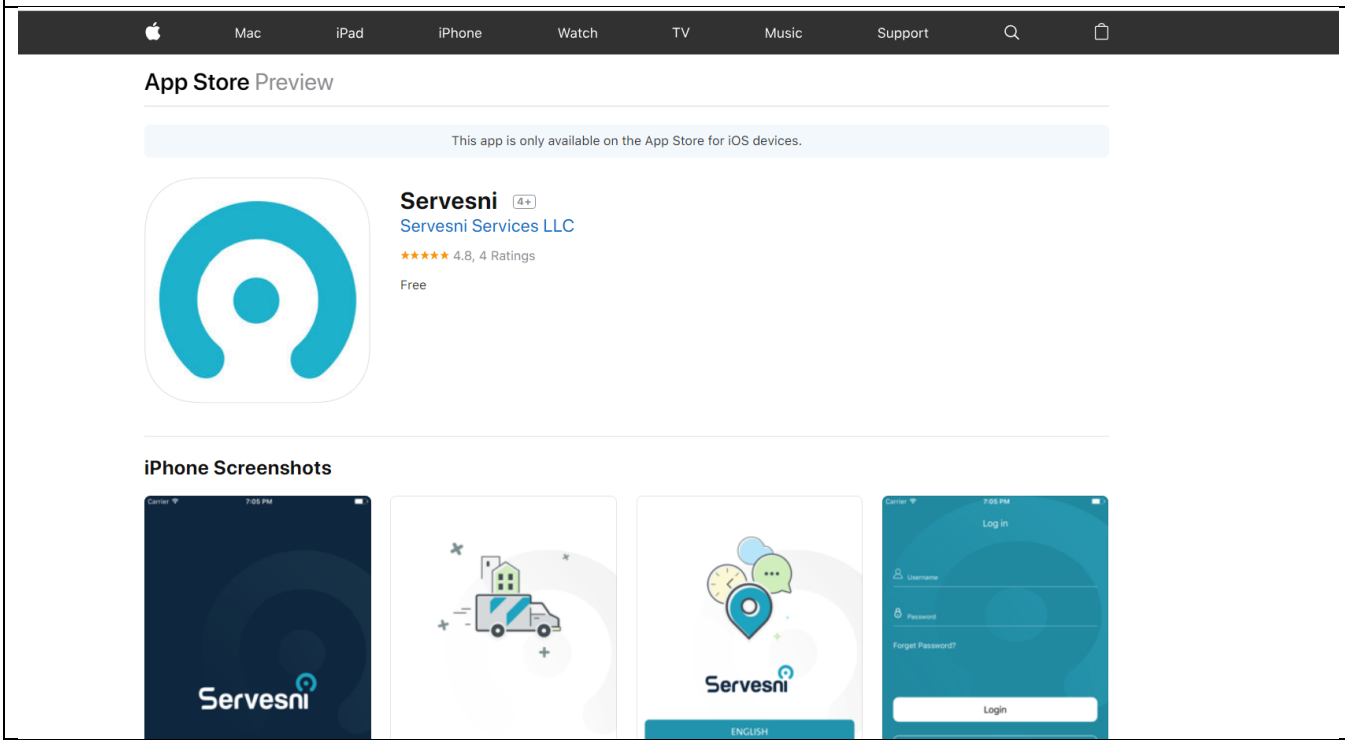
Request Car Care 02	
	

Table 6.53 Request car care 2 website screenshots

Request Car Care 2


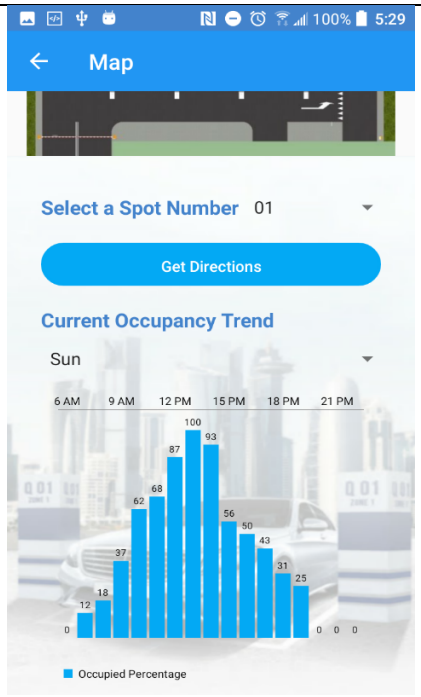
- View Current Occupancy Trend:

Table 6.54: View current occupancy trend test cases

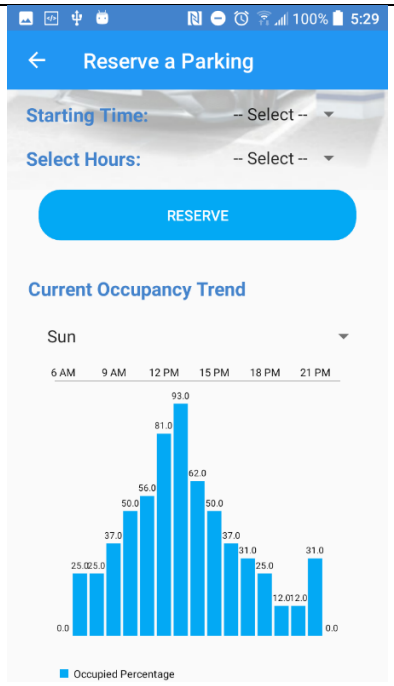
Test Case ID	Description	Screenshots
View Current Occupancy Trend 01	User successfully views statistical data that represent occupied percentage in each hour for selected zone for last four weeks.	For application see Table 6.55 For website see Table 6.56

Table 6.55: View Current Occupancy Trend 01 application screenshots

View Current Occupancy Trend 01

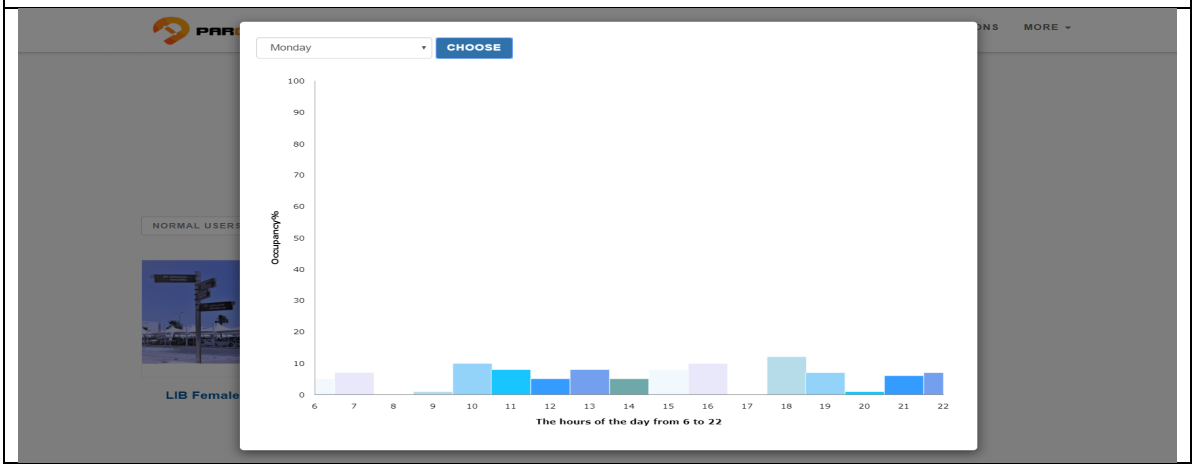


In check availability page



In reservation page

Table 6.56: View Current Occupancy Trend 01 website screenshots

View Current Occupancy Trend 01


- Check In

Table 6.57: Check in test cases

Test Case ID	Description	Screenshots
Check In 01	VIP user successfully checks in to zone if he has reservation at the current time and zone.	Table 6.58
Check In 02	VIP user tries to check in to zone when he does not have a reservation at the current time.	Table 6.59

Check In 03	VIP user tries to check in to zone when his reservation is in another zone	Table 6.59
Check In 04	VIP user tries to check in to zone when he cancels his reservation.	Table 6.60

Table 6.58: Check In 01 screenshots

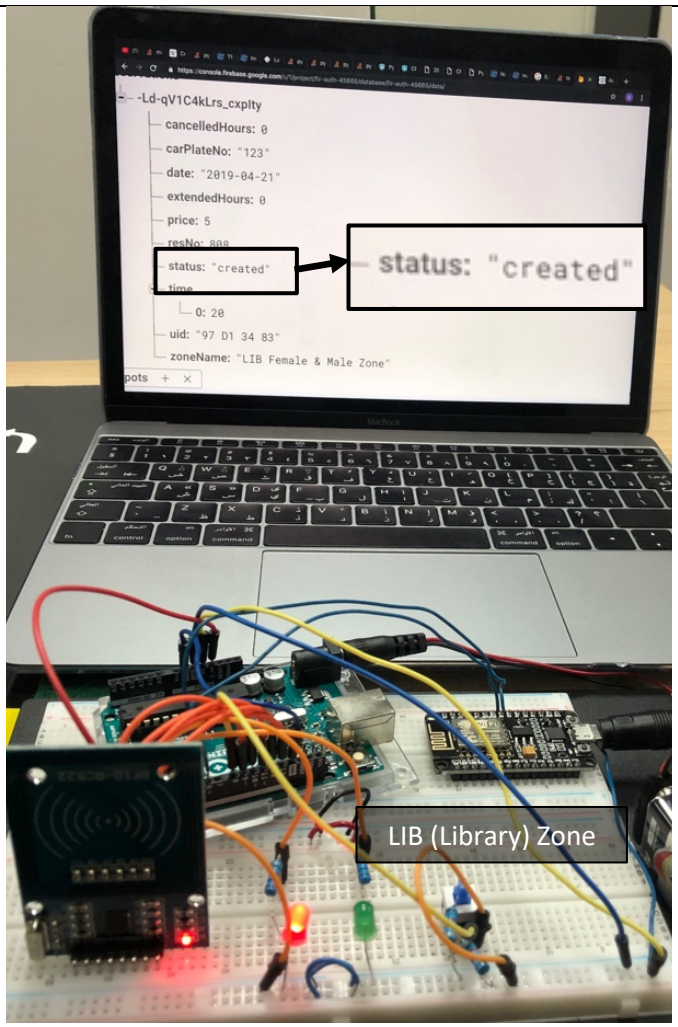
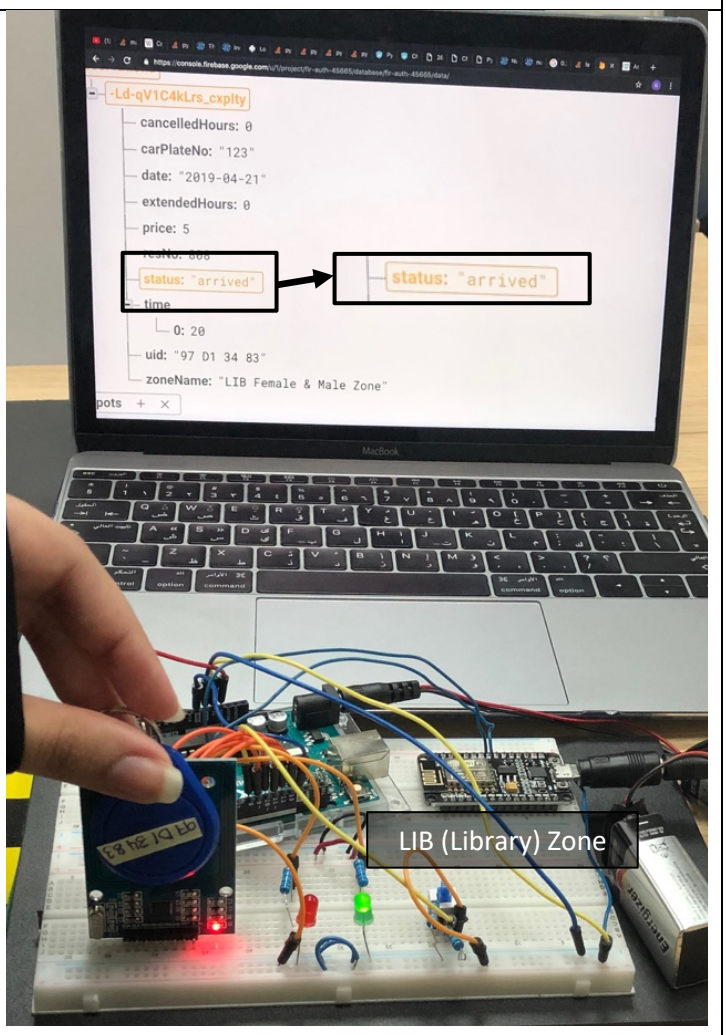
Check In 01 (Taken on 21-4-2019, at 8:06 PM)	
	

Table 6.59: Check In 02 and 03 screenshots

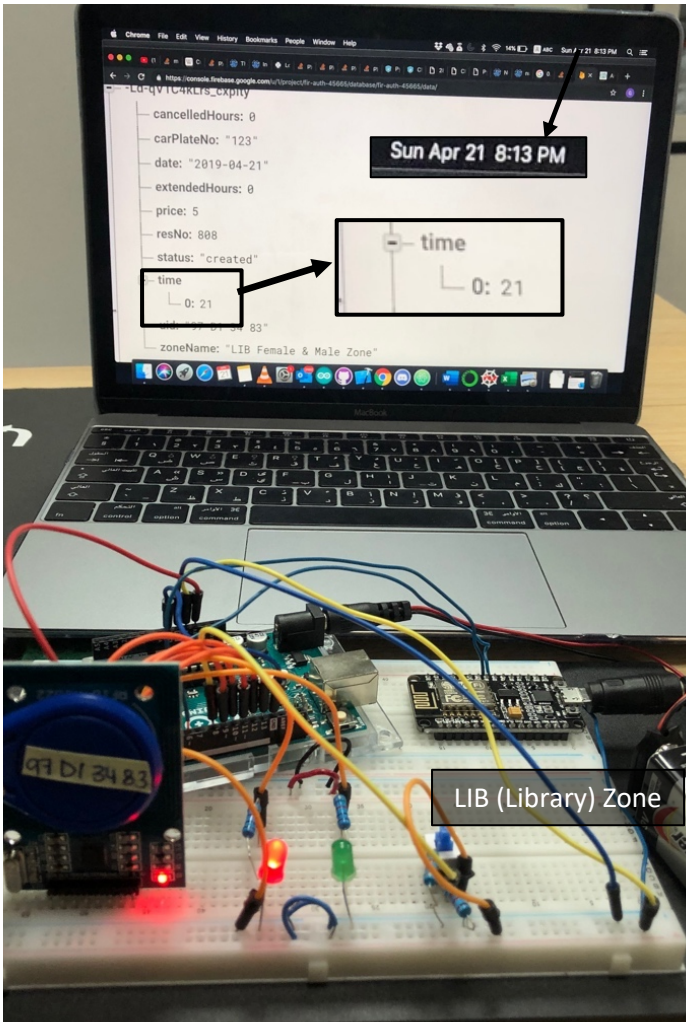
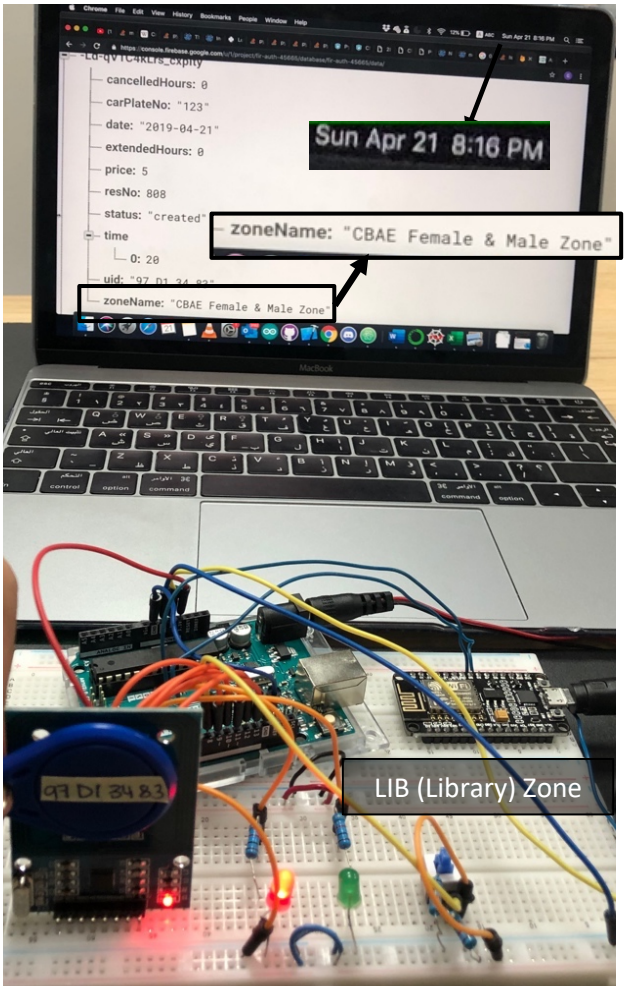
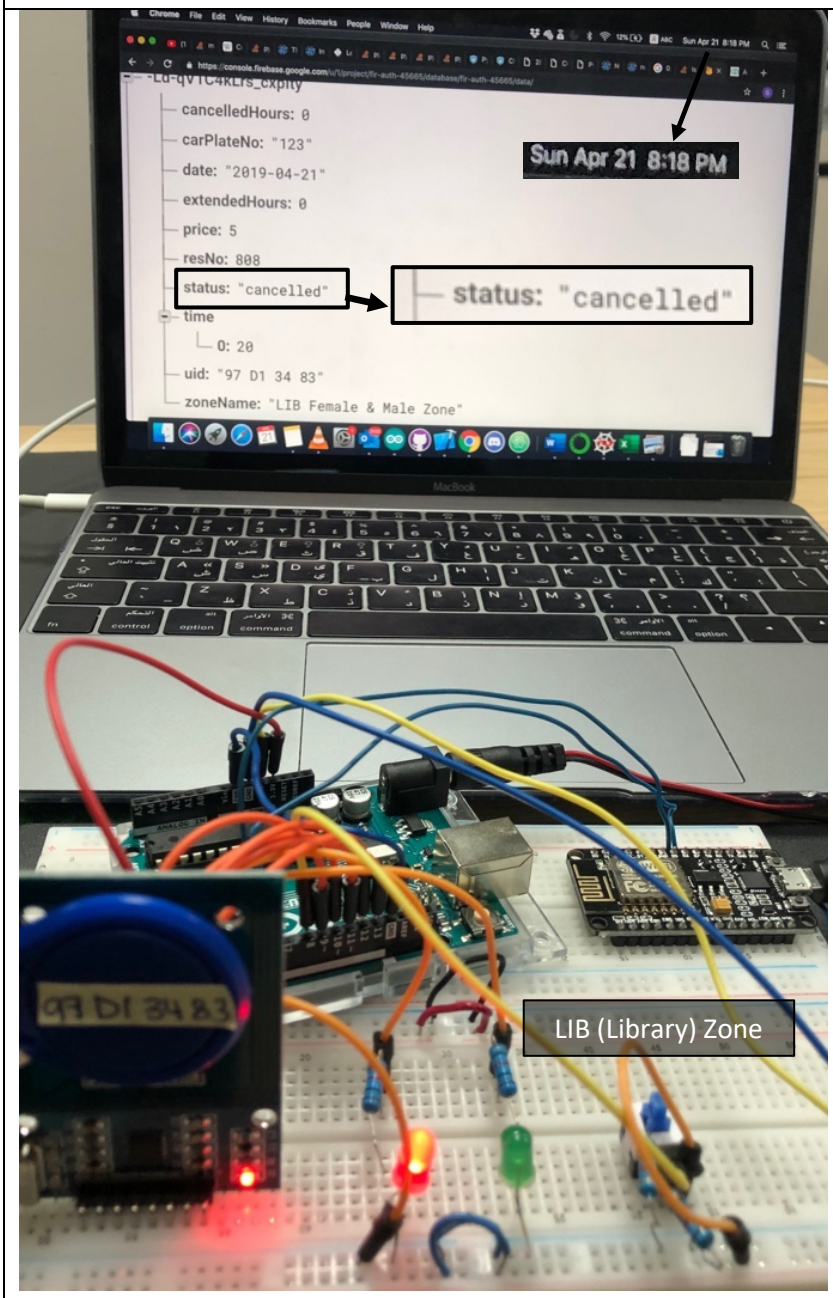
Check In 02 (Taken on 21-4-2019, at 8:13 PM)	Check In 03 (Taken on 21-4-2019, at 8:16 PM)
	

Table 6.60: Check In 04 screenshots

Check In 04 (Taken on 21-4-2019, at 8:18 PM)


- Check Out:

Table 6.61: Check out test cases

Test Case ID	Description	Screenshots
Check Out 01	VIP user successfully checks out of zone *when/before* his reservation time has ended (without penalty)	Table 6.62
Check Out 02	VIP user checks out of zone *after* his reservation time has ended (with penalty)	Table 6.63
Check Out 03	VIP user cancels part of his reservation using application/website and checks out of zone *when/before* his reservation has ended (without penalty)	Table 6.64
Check Out 04	VIP user cancels part of his reservation using application/website and checks out of zone *after* his reservation has ended (with penalty)	Table 6.65
Check Out 05	VIP user checks out of zone *when/before* his extension time has ended (without penalty)	Table 6.66
Check Out 06	VIP user checks out of zone *after* his extension time has ended (with penalty)	Table 6.67
Check Out 07	VIP user checks out of zone and automatic cancellation is applied when his reservation time has an hour or more left.	Table 6.68

Table 6.62: Check Out 01 screenshots

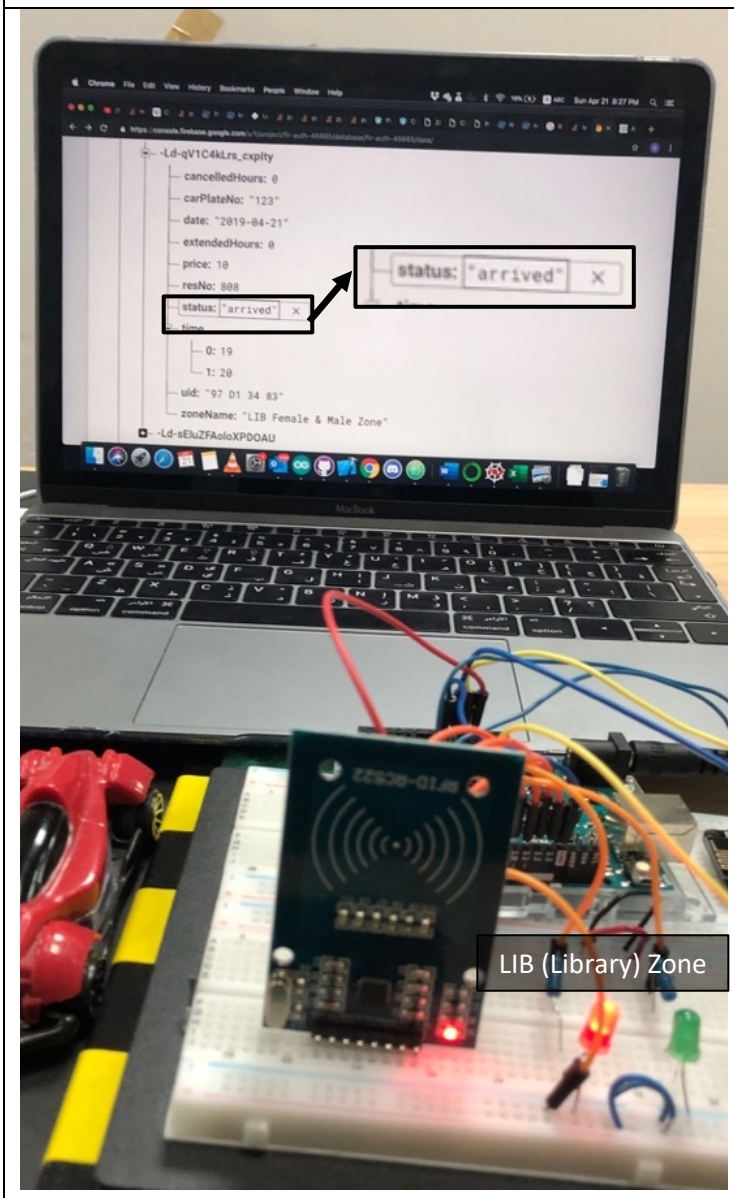
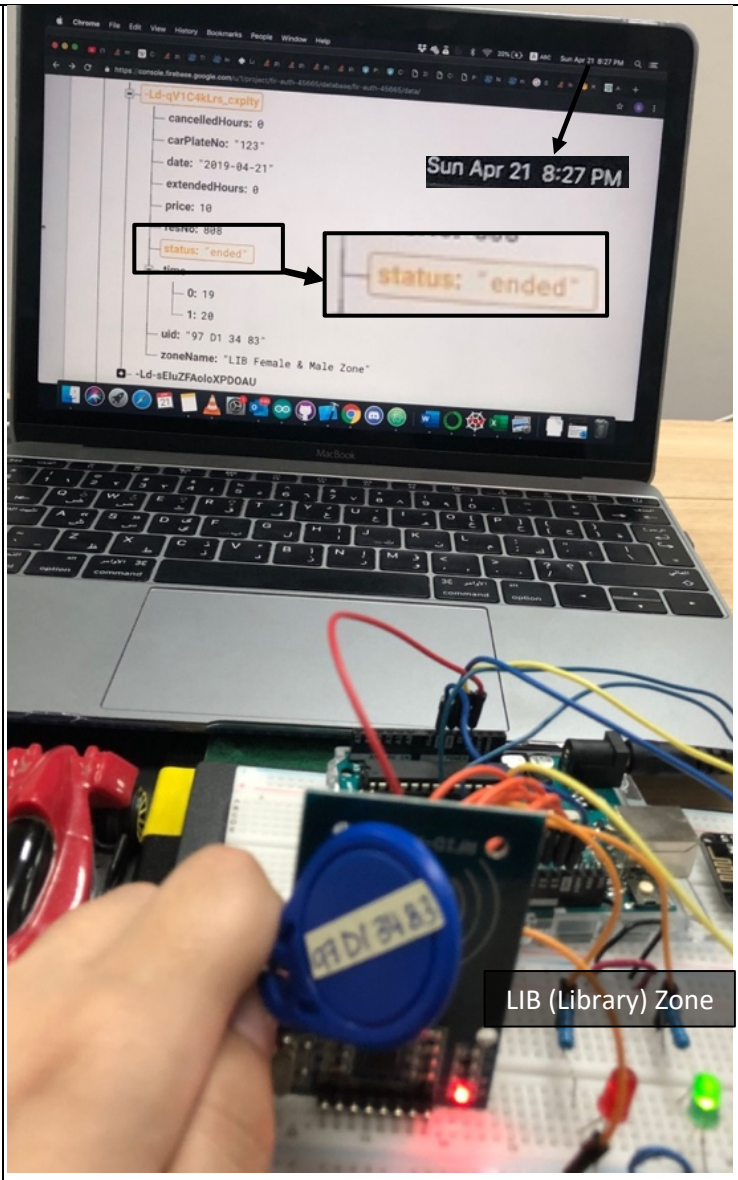
Check Out 01 (Taken on 21-4-2019, at 8:27 PM)	
	

Table 6.63: Check Out 02 screenshots

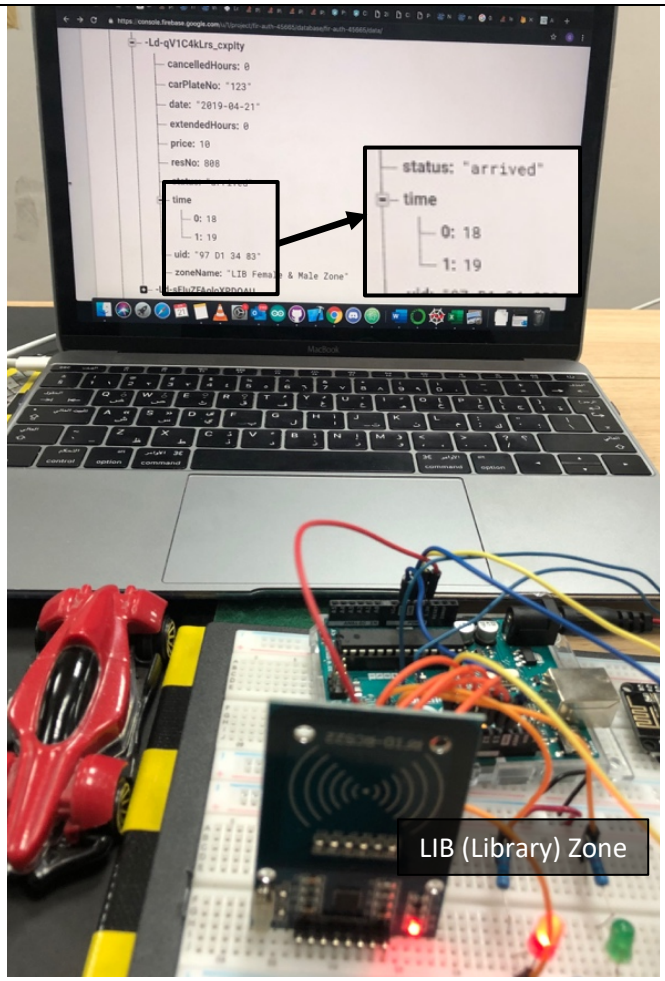
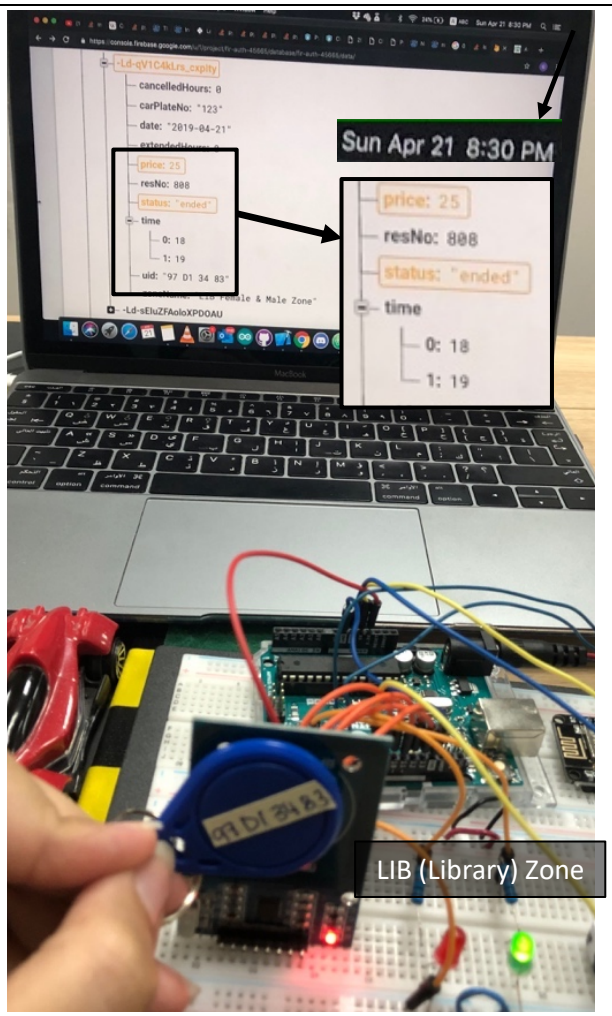
Check Out 02 (Taken on 21-4-2019, at 8:30 PM)	
	

Table 6.64: Check Out 03 screenshots

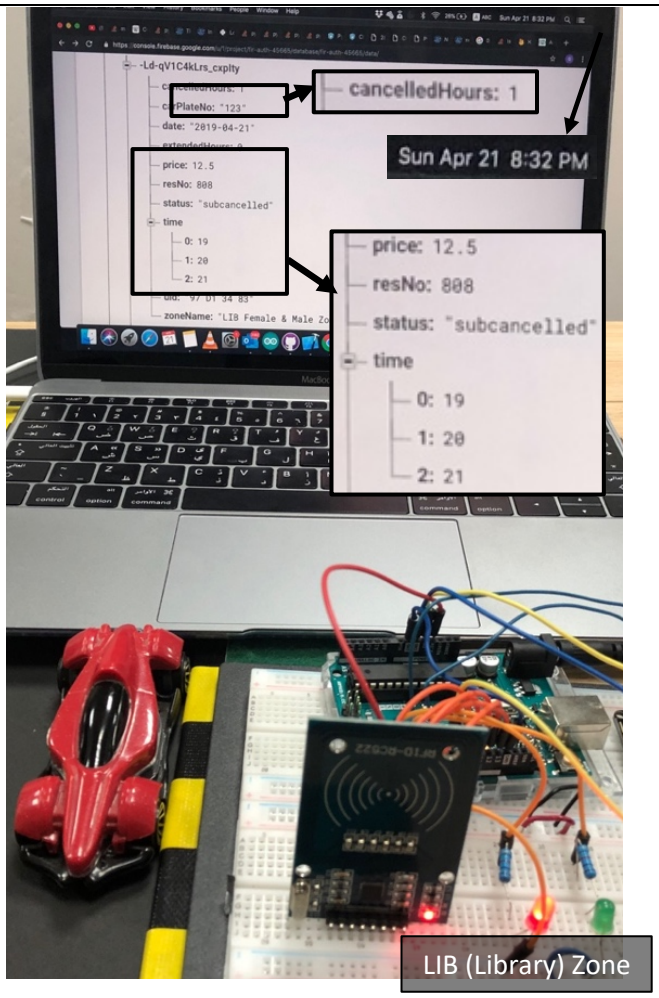
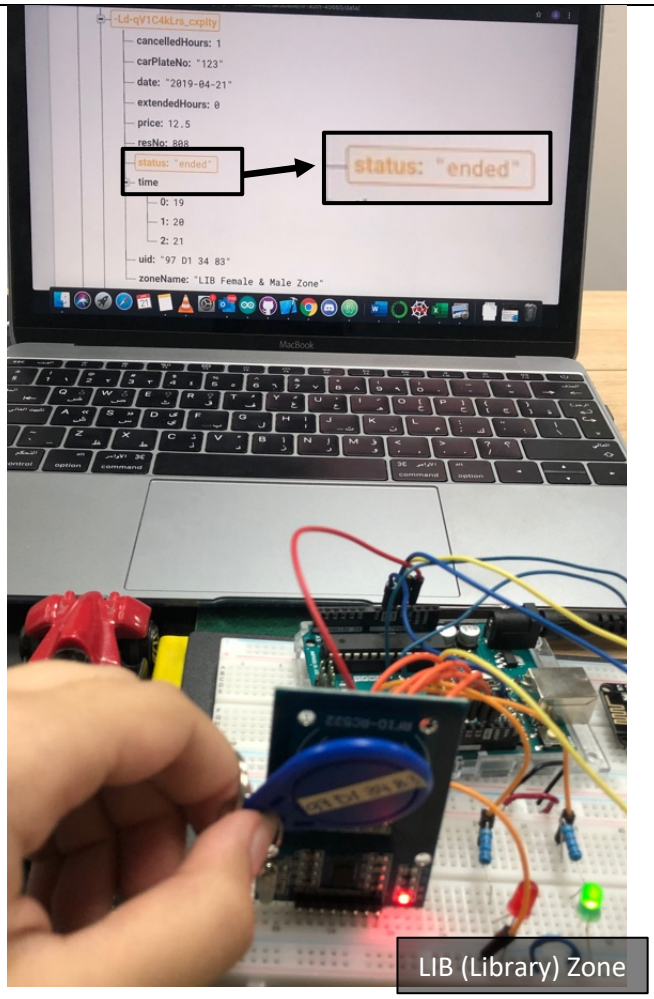
Check Out 03 (Taken on 21-4-2019, at 8:32 PM)	
	

Table 6.65: Check Out 04 screenshots

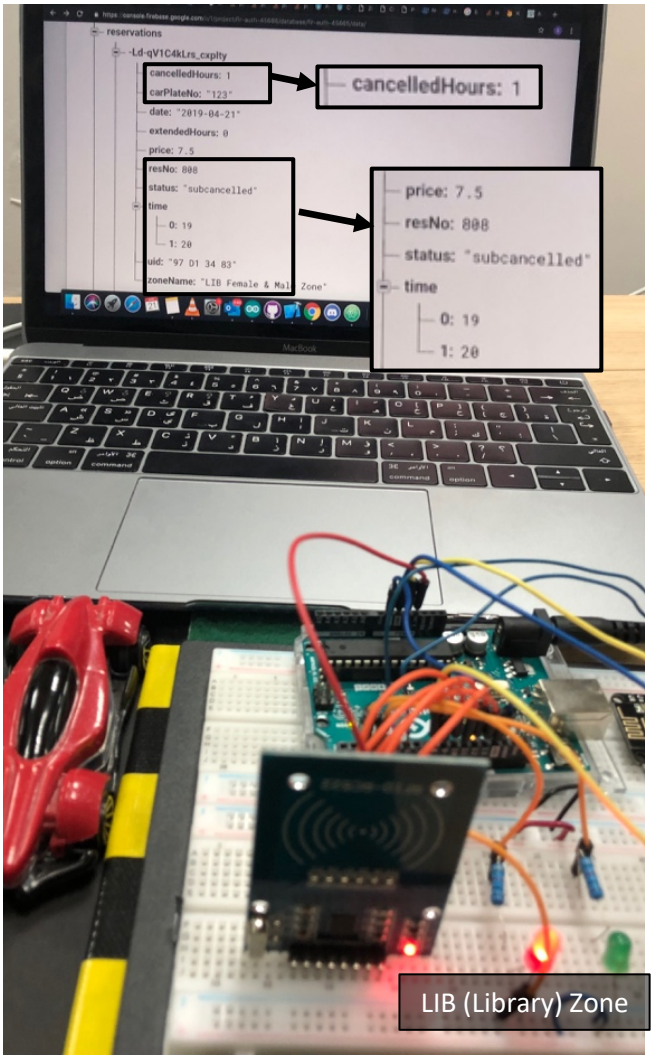
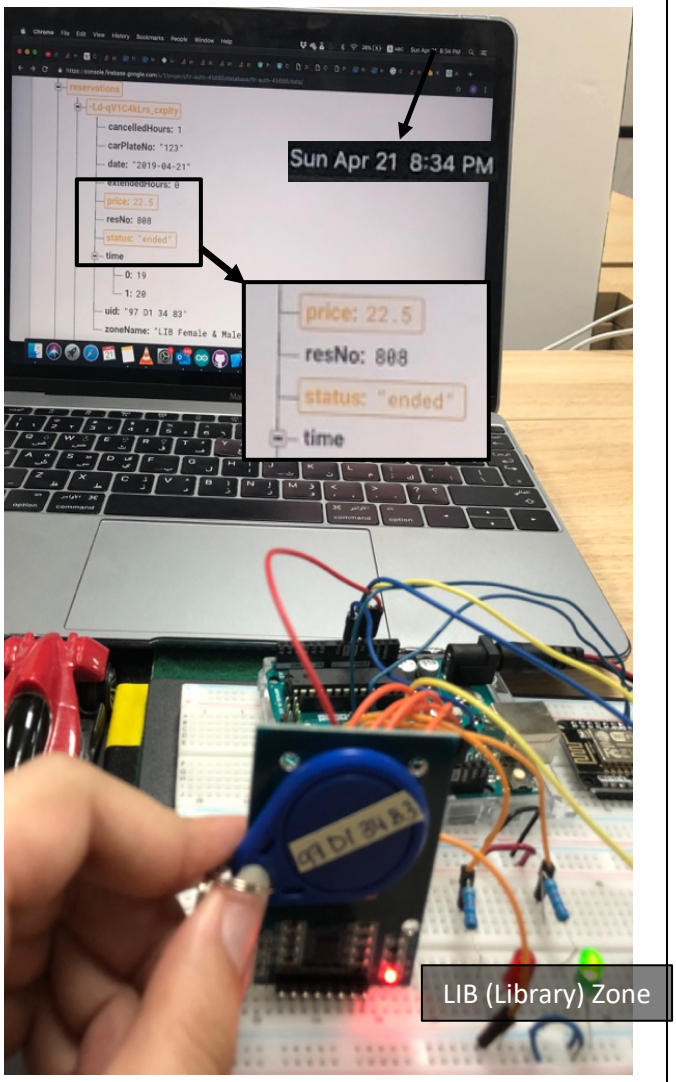
Check Out 04 (Taken on 21-4-2019, at 8:34 PM)	
	

Table 6.66: Check Out 05 screenshots

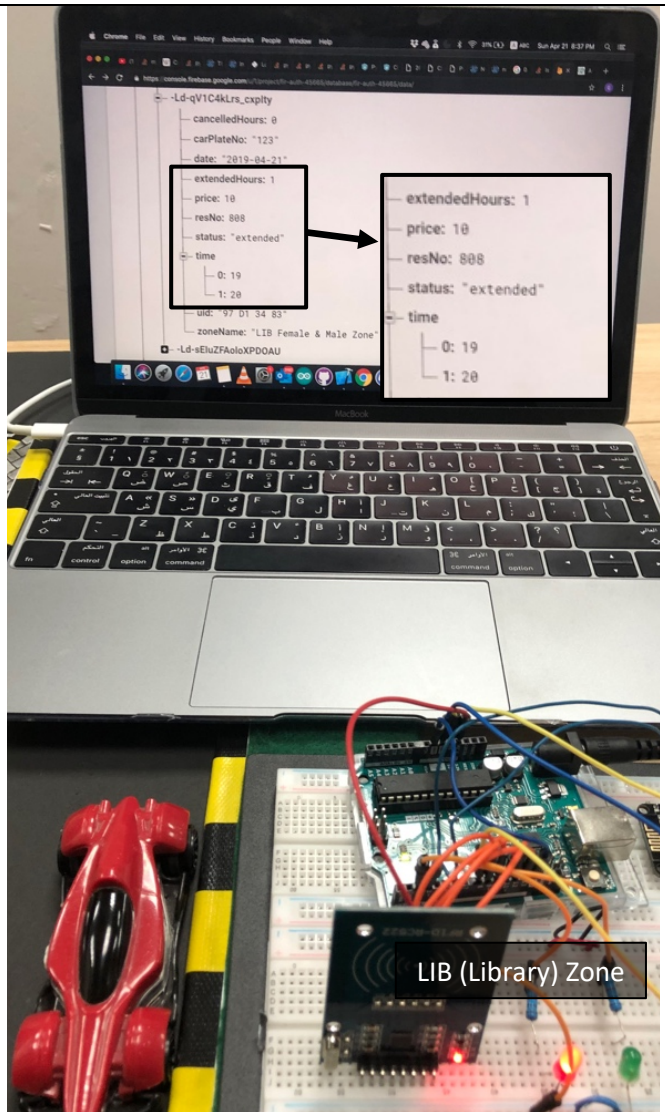
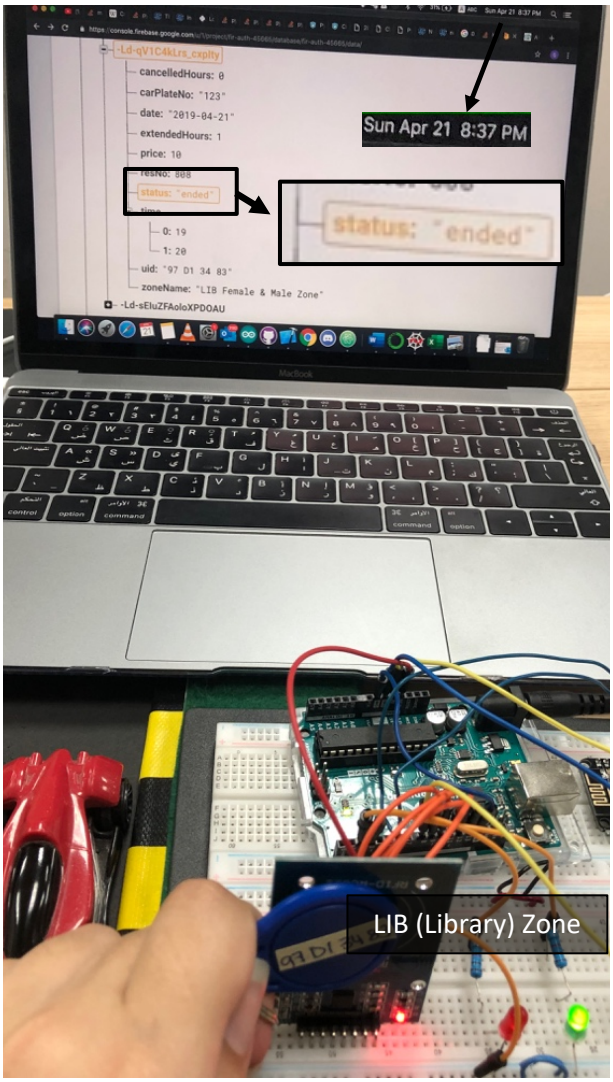
Check Out 05 (Taken on 21-4-2019, at 8:37 PM)	
	

Table 6.67: Check Out 06 screenshots

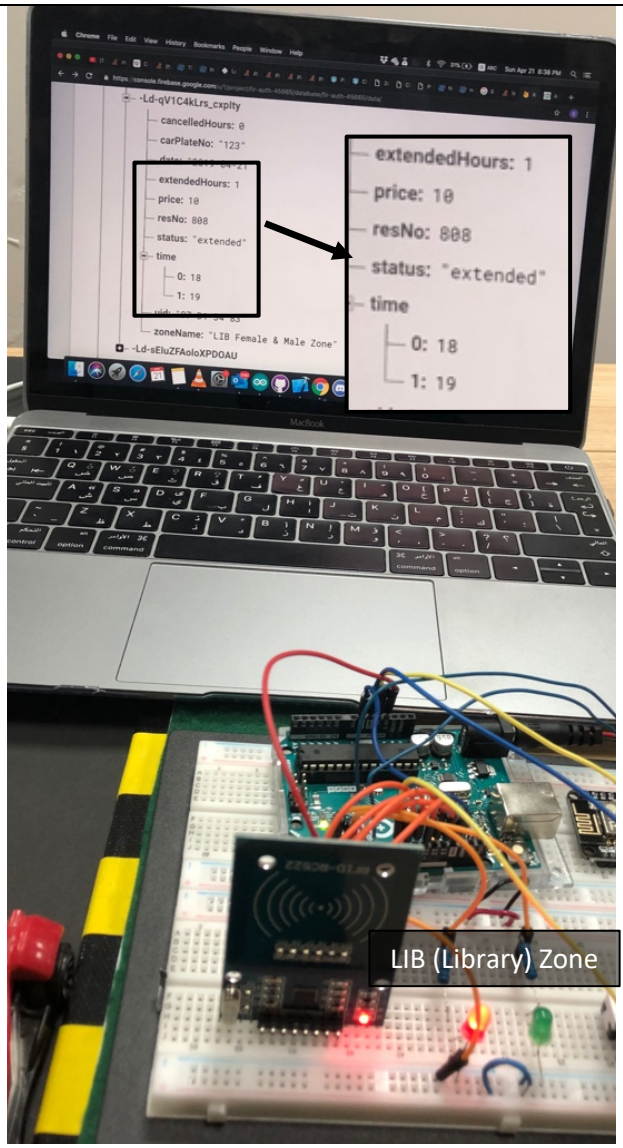
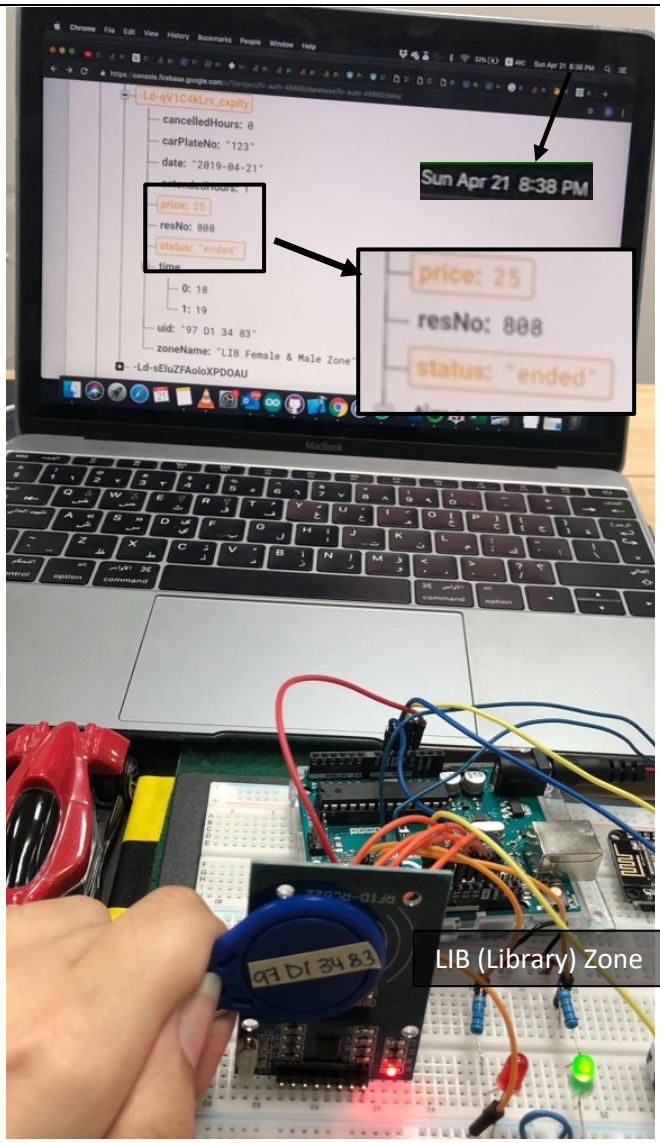
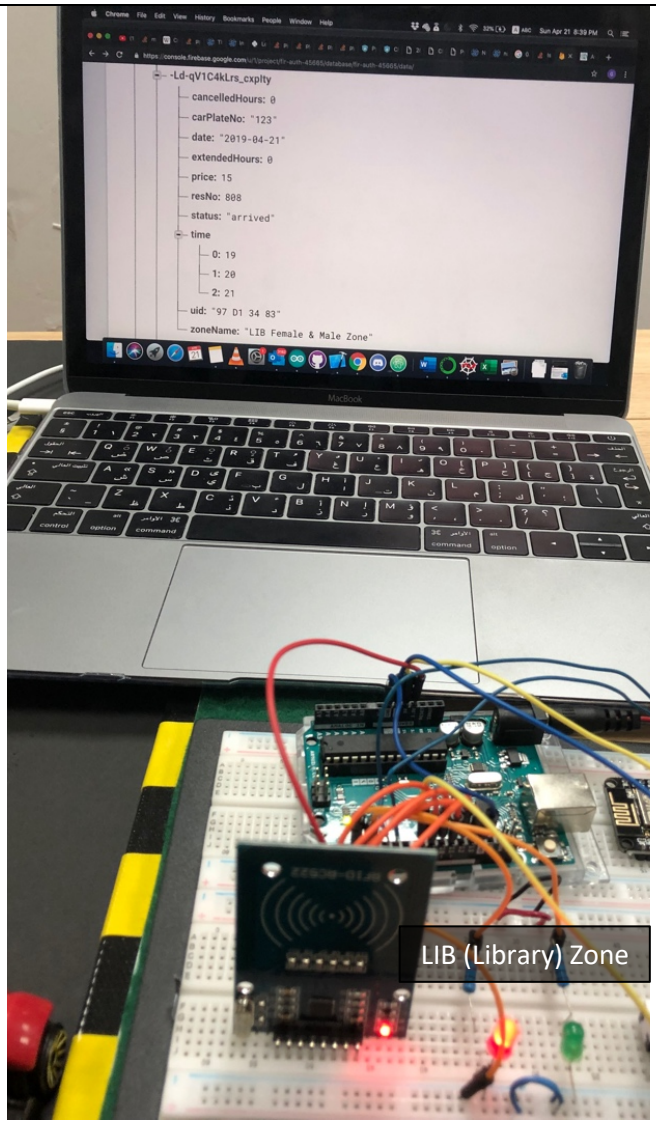
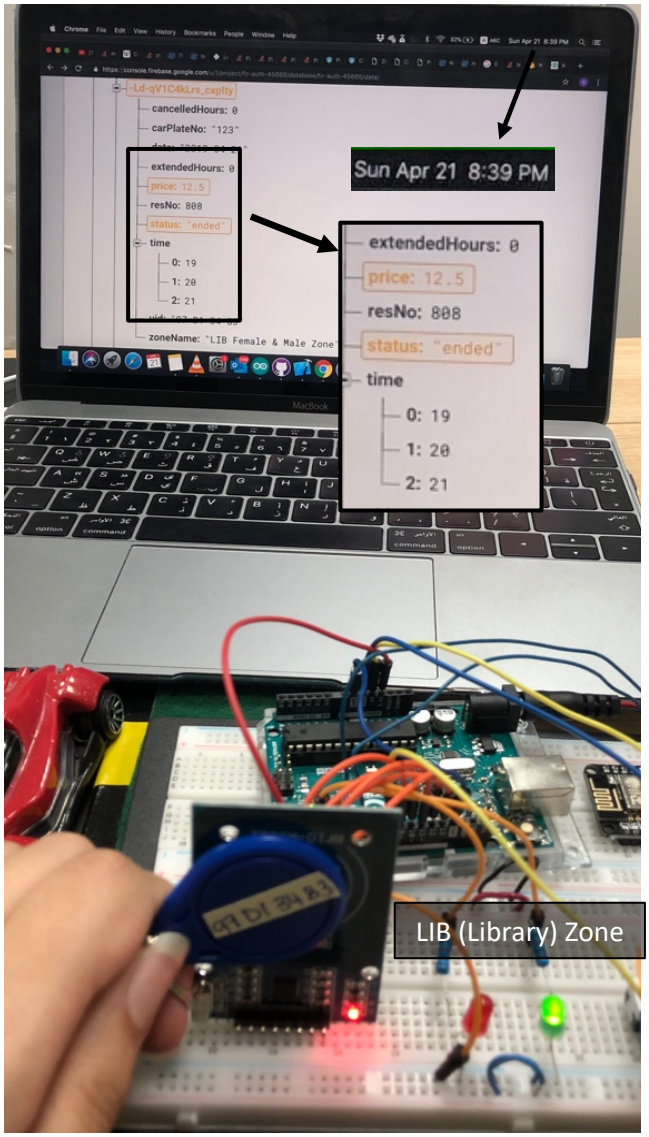
Check Out 06 (Taken on 21-4-2019, at 8:38 PM)	
	

Table 6.68: Check Out 07 screenshots

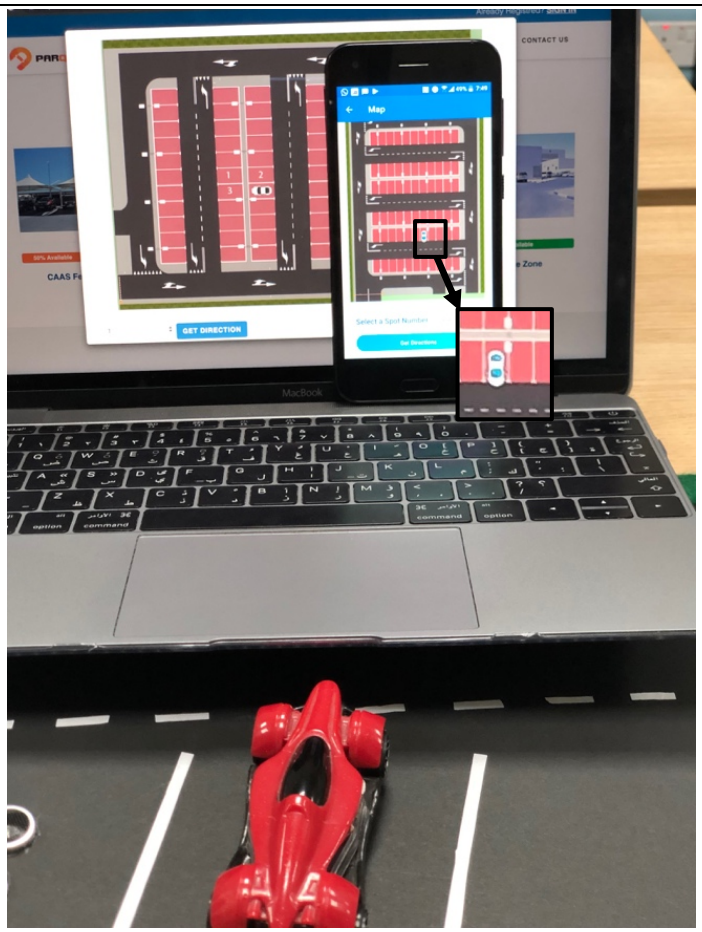
Check Out 07 (Taken on 21-4-2019, at 8:37 PM)	
	

- Update Parking Availability

Table 6.69: Update parking availability

Test Case ID	Description	Screenshots
Update Parking Availability 01	User successfully enters/leaves a parking spot and the availability status of the parking spot is updated.	Table 6.70

Table 6.70: Update Parking Availability 01 screenshots

Update Parking Availability 01


B. Acceptance Testing

As a way to ensure that our system would be accepted in real life. We managed to find 20 of our university colleagues and asked them to use ParQU system. Afterwards, we also asked them to fill a short online survey in order to get their feedback about what they think of the overall system. During this process, we received many comments on the user interface of the application and website to improve the user experience of our system. All the comments were taken seriously, and some were critical that they have been incorporated in the revision process of the system design and in subsequent further system improvement. For instance, one comment was that the application views were fixed and not scrollable when the mobile device's keyboard was on. The keyboard would obscured the rest of the items on the screen and the user had to scroll down the keyboard in order to see the other screen items. Another comment we received is that the buttons do not appear clickable. For this, we added a fading effect such that when the user clicks the button, the button is animated with a fading effect. Some of the student suggested to add in the application the descriptions for each service provided in the website to give them overall idea about the system so we added the "more details" option in the side menu. Another suggestion was to add additional services such as having valet parking. In contrast to "self-parking", where users find a parking space on their own, users' vehicles are parked for them by a person called a valet. This feature will be one

of the future works (refer to section 9). These were most of the comments received by the 20 users and what we have done in order to improve the application design and user experience. After looking at the users' survey answers (Appendix F) we can conclude that their experience with ParQU system was good overall.

C. Quality Testing (Design Constraints Evaluation)

To test if the non-functional requirements in section 3 has been met.

Table 6.71: Technical design constraints

Name	Description	Met/Not Met	How the design was met?
Availability	The system should always be available and real time.	√	The android application and website can be available at all times given that the devices accessing the application or website is connected to the internet. Also, for the system to work, NodeMCU must be connected to the internet. The system is real time by using Firebase Realtime Database that allows any change in the sensors to be immediately updated in the database, which in turn automatically displays the changes in the android application and website without the need to refresh the page
Reliability	No data loss is allowed.	√	NodeMCU, Arduino, Android and the website do not save any data locally and automatically save data in the Firebase, hence if any malfunction occurs, the data will not be lost.
Connectivity	NodeMCU needs to be connected to the internet to collect updated data from the sensors through the Arduino board then save it in the Firebase. In addition, the mobile application and website needs to be connected to the internet to get the data from the Firebase	√	Refer to section 6.1.2.
Scalability	The system can support the addition of sensors and components as well as having more users and the cloud can be upgraded accordingly.	√	ParQU can handle more VIP users with more RFID tags. Additionally, zones can be added with more components (In our prototype, a switch button is used to illustrate the scalability of our system, as explained in section 4.4). All of the previous additions require more database storage, the Firebase

			Realtime Database allows us to store up to 1GB for free (Free Plan). Any more storage would require us to upgrade to higher plans with a specific pricing [22]
Mobility	The system can be accessed from many different platforms.	√	Website can be accessible from any type of devices and platforms such as iOS, Android and Windows, etc. Android applications can be accessible from Android devices only whether it was a mobile or tablet.
Power	Power source needed for: Motor: 3-7V Sensor: 5V Arduino: 7-12V	√	Arduino is supplied by a 9V battery. Sensors and Servo Motor are both supplied by Arduino

Table 6.72: Practical design constraints

Quality	Name	Description	Met/Not Met	Verification
Economic	Design Cost	The prototype uses high quality component with affordable prices	√	The whole prototype costed us on average 550 QR (Prototype consists of two zones; each zone has four parking spots)
Social	Usability	A normal user with minimal software knowledge should be able to use the mobile application and the website with ease	√	Refer to section 6.1.3 Acceptance testing
Sustainability	Maintenance	The system components should be easy to replace, remove and implement.	√	The components could easily be replaced by uploading our code to a new component.
Quality	Performance	The system should complete the required functionalities within a short response time, providing accurate readings from sensors and efficient information from Android application and website.	√	We measured the time needed to create account, login, create reservation, extend reservation

				and cancel reservation by using stopwatch. • Create account takes less than 2 minutes • Login takes less than 12 seconds • Create reservation takes less than 30 seconds • Cancel reservation takes less than 15 seconds • Extend reservation takes less than 15 seconds

6.2 Outdoor Testing

ParQU is a system that should be able to be implemented outdoor in a real parking area. As an initial test to verify that our sensors are able to withstand the harsh environments of being outdoors in Qatar University and satisfy its purpose, we tested the Ultrasonic Sensor outdoors to check the availability of a parking spot.

The sensor was placed below the parking spot. Figure 6.15 shows the distance from the ground till the bottom of the car, this is needed in programming the sensor detection distance. Figures 6.16 and 6.17 shows our test accompanied with the database showing the current status of the parking spot. Figure 6.17 the parking is empty, so the database shows that the status is “available”, while Figure 6.16 has a car, so the database shows that the current status is “not available”.

Figure 6.15: Distance

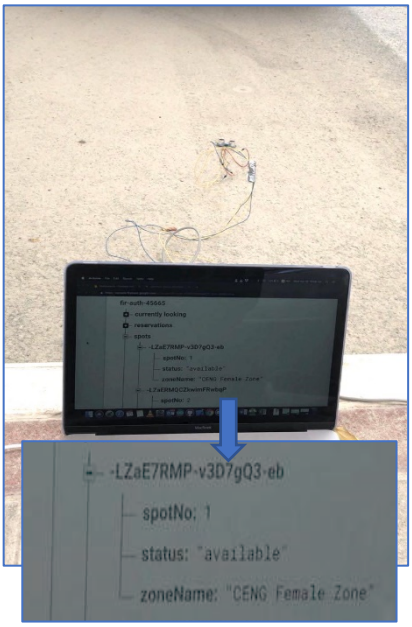


Figure 6.17 Status "available"

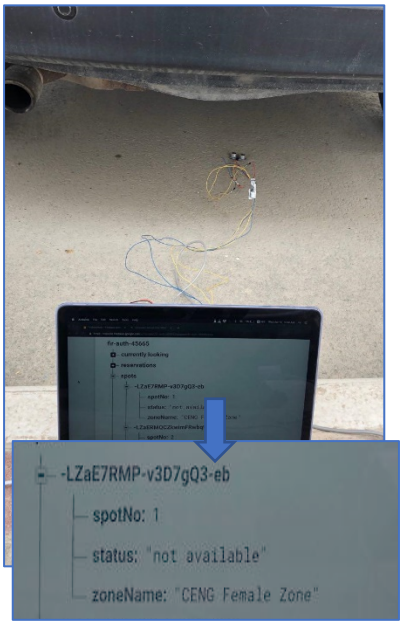


Figure 6.16 Status "not available"

Furthermore, when the system is implemented in real life, the sensor installation positions must be studied to pick the best position with the environment taken into consideration. Table 6.73 shows the different positions a sensor could be placed in real life along with its pros and cons.

Table 6.73: Possible installation positions

Position in a parking spot	Distance limit	Where	Pros	Cons
Above	Shortest possible car height	Indoors and outdoors	- Sensor protected from rain and dirt. - Less susceptible to breaking	- Needs a parking shade to place the sensor under it
Below	Highest possible car height	Indoors	- Cheapest	- Easily subjected to dirt if outdoors - Easily breakable (Sensor needs some kind of shield which is costly)
In front	Farthest possible distance a car could park	Indoors and outdoors	- Maintenance is easy (Easier to replace and clean)	-Cars may bump the sensor (Sensor needs some kind of shield which is costly) -needs a stand (Costly) -Most expensive

For indoor parking lots, the most suitable sensor position is above the parking spot because the sensor will be less susceptible to breaking and the parking spot will always have a ceiling to place the sensor under it. On the other hand, for outdoor parking lots, the best position can either be above or below the parking spot depending on the environment. Placing the sensor above the parking has a huge benefit compared to below the parking spot, as the sensor will be protected from rain and dirt. However, if the parking lot does not have a shade that is already implemented, then placing the sensor above can be expensive compared to placing it below the parking spot. With that said, installing the sensor below a parking spot will require a shield to protect the sensor from being bumped by cars or covered with dirt.

7. Evaluation of the impact of the engineered solution

The proposed solution contributes, if implemented, to the solving of some economic, environmental, and societal problems on different levels.

	Evaluation Context	Specific Contribution through the Project	level of impact (High, Medium, Low)
1	**Global**	The project can benefit people globally. Crowded parking lots is a widespread problem across the world as shown in the background section 2.1 and our project can assist people to have a nice	High

		parking experience. Furthermore, the project's main language is English and hence can be applied to several countries.	
2	**Economical**	ParQU saves the users time and fuel consumption when looking for an empty parking which saves the users money. Also, the project components are affordable for associations to implement.	Medium
3	**Environmental**	All project components do not affect the environment negatively. Also, less fuel consumption leads to higher CO2 emissions (as explained in section 1).	Low
4	**Societal**	The project can change people's view on parking through ParQU by providing them a comfortable and efficient parking experience.	Medium

8. Conclusion

One of the students and faculty problems in Qatar University is overcrowded parking areas where it is hard to find an empty parking spot. The main objective of our project ParQU is to make its users parking experience as smooth as possible by solving this issue. ParQU, a QU parking system, provides its users with two main services, showing the available parking spots in real time and reserving a parking spot. Furthermore, new features were added to the system in senior project 2 like the Currently Looking and Current Occupancy Trend features and others were improved like the reservation service. ParQU is available to the users through an Android application and a website. Even though we added new features as an improvement, we will still seek to improve our system in the future and add features to the user like the ones explained in Future Work (section 9).

9. Future work

As the time was not sufficient for us to implement further improvement in ParQU system, future work that can be done on this project are listed below:

- Develop an iOS application for iOS users.
- Add other languages for the application, most importantly Arabic for elderly people to understand the application.
- Implement pricing packages such as week, monthly or semester subscription.
- Make our system more scalable to the users by allowing them to register multiple cars with the same account because our current system allows users to register only one car with an account.
- Develop an admin interface where the admin has several functionalities such as adding additional zones to the system.

- Secure the system from all perspectives. Securing the channels of communication as well as the database.
- Adding some additional parking services such as valet parking.
- Implement a complete payment service as our current system already calculates the total price of a reservation and stores it in the database. Hence, as a future work, the total price for each user can be calculated and paid using a real payment service
- Nearest empty parking spot to the building entrance suggestion.
- Allocate parking spots for handicaps and people with special needs.

10. Student reflections

Ghareisa's reflection

The senior project throughout this year greatly helped me enhance my knowledge as a 5 year CE undergraduate student. I think this project showed me how difficult the process of implementing a real life project can be. Even though our practical implementation was only a prototype, we still did struggle in making clear project requirements at the beginning, designing how the project should be and making mistakes till the very end.

Furthermore, my teammate Alaa affected me beneficially to understand the project, as we were always discussing with each other any ideas and different ways to implement the project. I think me and Alaa worked efficiently together and knew how to communicate and understand the weaknesses and strengths of each other to then divide the work equally and efficiently between us. Dr. Abdulaziz taught me how to be creative with our projects and to always thrive for improvements.

Alaa's reflection

Working on our senior project was a very helpful and enjoyable experience for me. Since the project I worked on was so much big in scale compared to the other projects I worked on in other courses, I had to work hard to integrate most of what I have learned during my four years in studying computer science to make this project come to reality. Though I completed an Android application and development course, the skills and knowledge learnt in this year was furthermore than what I have learned in the past. Dealing with an external, cloud-based database, the communication with two software to be integrated into the same system were some challenges that I overcame during the process of implementation of the ParQU system.

Working with Dr. Abdulaziz was of great help to us as well as he was always available to help us finish different parts of the project. And not to forget my amazing brilliant colleague Ghareisa who was always helpful and hardworking throughout the year. At the end, I am sure that all the skills and knowledge I have gained will be helpful for me in the future.

Amal's reflection

My experience with this project was a unique experience that cannot be forgotten. Starting from the previous semester and preparing for. to the last minutes of submission. In fact, the impact of this rich experience has exceeded my expectations. In the beginning, I have to divide my experience learnings into two parts, a technical side, that touches my specialty, and another that is some general knowledge and skills. This project gave me the opportunity to experience and discover many programs, algorithms, libraries and technologies I never tried before. The senior project allows me not only to try these technologies but also to gain and develop good skills on using them. For instance, this experience laid the foundation stone for building web applications. It was really complicated at many stages to search for the information, learn, and implements without any

previous background. I faced many restrictions during the implementation. I learned a lot about sharing ideas and communication between teams' members.

References

[1] "Qatar-Education Statistics, 2013," *Arab Development Portal*, 23-Aug-2017. [Online]. Available: http://arabdevelopmentportal.com/publication/qatar-education-statistics-2013. [Accessed: 02-Oct-2018].

[2] "About", *Qatar University*. [Online]. Available: http://www.qu.edu.qa/about. [Accessed: 24-Nov-2018].

[3] Al-Arab, (2016). [online] Available at: http://www.alarab.qa/story/1010307. [Accessed 17 Sep. 2018].

[4] Al-Watan, (2018). [online] Available at: http://www.al-watan.com/news-details/id/156085. [Accessed 12 Oct. 2018].

[5] Al-Watan (2018). [online] Available at: http://www.al-watan.com/news-details/id/129483. [Accessed 12 Oct. 2018].

[6] S. A. Shaheen and C. Kemmerer, "Smart Parking Linked to Transit," *Transportation Research Record: Journal of the Transportation Research Board*, vol. 2063, no. 1, pp. 73–80, 2008. [online]. Available: https://journals.sagepub.com/toc/trra/2063/1

[7] J. Chinrungrueng, U. Sunantachaikul and S. Triamlumlerd, "Smart Parking: An Application of Optical Wireless Sensor Network," *2007 International Symposium on Applications and the Internet Workshops*, Hiroshima, 2007, pp. 66-66. doi: 10.1109/SAINT-W.2007.98. available: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4090136&isnumber=4090057

[8] D. Pojani, I. Mateo-Babiano, J. Corcoran, and N. Sipe, "Should we get rid of parking spaces to free up land in our cities?" *ABC News*, 06-Nov-2017. [Online]. Available: https://www.abc.net.au/news/2017-10-30/should-we-get-rid-of-car-parks-to-free-up-land-in-our-cities/9099272. [Accessed: 26-Oct-2018].

[9] D. Pojani, I. Mateo-Babiano, J. Corcoran, and N. Sipe, "Freeing up the huge areas set aside for parking can transform our cities," *The Conversation*, 09-Nov-2018. [Online]. Available: http://theconversation.com/freeing-up-the-huge-areas-set-aside-for-parking-can-transform-our-cities-85331. [Accessed: 12-Nov-2018].

[10] INRIX, "Searching for Parking Costs Americans $73 Billion a Year - INRIXINRIX," *INRIX*. [Online]. Available: http://inrix.com/press-releases/parking-pain-us/. [Accessed: 19-Nov-2018].

[11] Y. Rahayu and F. N. Mustapa, "A Secure Parking Reservation System Using GSM Technology," *Int. J. Comput. Commun. Eng.*, pp. 518–520, 2013.

[12] B. Haider, M. Zafrullah, and M. K. Islam, “Radio Frequency Optimization & QoS Evaluation in Operational GSM Network,” p. 6, 2009.

[13] R. Yusnita, F. Norbaya, and N. Basharuddin, “Intelligent Parking Space Detection System Based on Image Processing,” *Int. J. Innov.*, vol. 3, no. 3, p. 4, 2012.

[14] H. Chien Yee and Y. Rahayu, “Monitoring Parking Space Availability via Zigbee Technology,” *Int. J. Future Comput. Commun.*, vol. 3, no. 6, pp. 377–380, Dec. 2014.

[15] A. Kianpisheh, N. Mustaffa, P. Limtrairut, and P. Keikhosrokiani, “Smart Parking System (SPS) Architecture Using Ultrasonic Detector,” *Int. J. Softw. Eng. Its Appl.*, vol. 6, no. 3, p. 8, 2012.

[16] A. Ahad, Z. R. Khan, and S. A. Ahmad, “Intelligent Parking System,” *World J. Eng. Technol.*, vol. 04, no. 02, pp. 160–167, 2016.

[17] R. S. Pressman, *Software engineering: a practitioner’s approach*, 7th ed. New York: McGraw-Hill Higher Education, 2010.

[18] T. Lin, H. Rivano, and F. Le Mouel, “A Survey of Smart Parking Solutions,” *IEEE Trans. Intell. Transp. Syst.*, vol. 18, no. 12, pp. 3229–3253, Dec. 2017.

[19] “What are the differences between Raspberry Pi and Arduino?” 2017. [Online]. Available: https://www.electronicshub.org/raspberry-pi-vs-arduino/. [Accessed: 24-Nov-2018].

[20] P. Jahoda, *A powerful Android chart view / graph view library, supporting line- bar- pie- radar- bubble- and candlestick charts as well as scaling, dragging and animations.: PhilJay/MPAndroidChart*. 2019.

[21] “Storage size calculations,” *Firebase*. [Online]. Available: https://firebase.google.com/docs/firestore/storage-size. [Accessed: 23-Apr-2019].

[22] “Firebase,” *Firebase*. [Online]. Available: https://firebase.google.com/pricing/. [Accessed: 23-Apr-2019].

[23] “IP Location Finder - Geolocation.” [Online]. Available: https://www.iplocation.net/. [Accessed: 24-Apr-2019].

Appendix A – Project Plan

A.1. Project milestones

Phase-1

Milestone	Tasks
Milestones -1 Information gathering	• Research about the project problem to prove the necessity of a solution. • Agree on a problem statement • Write a brief overview abstract
Milestones -2 Introduction and motivation	• Determine the technical and non-technical challenges and finalize the problem statement • Perform a survey to study the possible parking problems in Qatar University • Determine the significance of the project and how the outcomes of the project will benefit the users • Identify the aim and objective of the project
Milestones -3 Hardware and software to be used	• Search about all possible hardware components that might be used to solve the problem. • Prepare an excel sheet that has the decided-on hardware with its price • Search about Firebase and how it can be connected to the mobile application and Arduino
Milestones -4 Related work	• Read multiple papers with similar approaches to our problem in addition to the survey papers about same topic • Analyze the method used in on each selected paper • Deliver a table that lists all the similarities and differences between our approach and the other approaches.
Milestone-5 Requirement analysis	• Determine functional requirements • Design a use case diagram • Write use case specifications • Determine the technical and practical design constrains • List the design standards • List code of ethics
Milestone-6 Proposed solution	• List all the possible solutions for hardware and software parts and analyze their tradeoffs • Selected solution overview • Design high level architecture for the selected solution
Milestone-7 Complete interim report	• Gather all the work done in the interim report • Revise all parts of the report with the supervisor and get feedback • Complete and submit the report

Milestone-8 **Presentation**	• Prepare the presentation • Demonstrate the project to the examiners

Phase-2

Milestone	Tasks
Milestones -1 **Hardware and software design**	• Finalize use cases • Design a rough diagram of the circuit and how all the hardware components will be connected to each other • Work on structural model (class diagram) • Work on behavioral model (sequence diagram) • Work on database design • Work on user interface design • Work on design pattern
Milestones -2 **Implementation**	• Implement hardware • Build hardware prototype • Implement mobile application • Implement website
Milestones -3 **Testing**	• Work on unit testing • Work on integration testing • Work on system testing • Work on acceptance testing
Milestones -4 **Documentation**	• Gather all the work done in the report • Complete and submit the report
Milestones -5 **Presentation + Poster**	• Prepare the presentation • Prepare the poster and video • Demonstrate the project to the examiners

A.2. Project timeline

Tasks	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10	Week 11	Week 12	Week 13	Week 14	Week 15
Milestones -1 Information gathering															
Research about the project problem to prove the necessity of a solution															
Agree on a problem statement															
Write a brief overview abstract															
Milestones -2 Introduction and motivation															
Determine the technical and non-technical challenges and finalize the problem statement															

Perform a survey to study the possible parking problems in Qatar University															
Determine the significance of the project and how the outcomes of the project will benefit the users															
Identify the aim and objective of the project															
Milestones -3 Hardware and software to be used															
Search about all possible hardware components that might be used to solve the problem															
Prepare an excel sheet that has the decided-on hardware with its price															
Search about Firebase and how it can be connected to the mobile application and Arduino															
Milestones -4 Related work															

Read multiple papers with similar approaches to our problem in addition to the survey papers about same topic															
Analyze the method used in on each selected paper															
Deliver a table that lists all the similarities and differences between our approach and the other approaches															
Milestone-5 Requirement analysis															
Determine functional requirements															
Design a use case diagram															
Write use case specifications															
Determine the technical and practical design constrains															

List the design standards															
List code of ethics															
Milestone-6 **Proposed solution**															
List all the alternative solutions for hardware and software parts and analyze their tradeoffs															
Selected solution overview															
Design high level architecture for the selected solution															
Milestone-7 **Complete interim report**															
Gather all the work done in the interim report															
Revise all parts of the report with the supervisor and get feedback															
Submit the report															
Milestone-8															

Presentation															
Prepare the presentation															
Demonstrate the project to the examiners															

Tasks	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10	Week 11	Week 12	Week 13	Week 14	Week 15
Milestones -1 Hardware and Software design															
Finalize use cases															
Design a rough diagram of the circuit and how all the hardware components will be connected to each other															
Work on structural model (class diagram)															
Work on behavioral model (sequence diagram)															
Work on database design															

Work on user interface design															
Work on design pattern															
Milestones -2 Implementation															
Implement hardware part															
Build hardware prototype															
Implement website															
Implement mobile application															
Milestones -3 Testing															
Work on unit testing															
Work on integration testing															
Work on system testing															
Work on acceptance testing															
Milestones -4 Documentation															
Gather all the work done in the report															
Submit the report															
Milestones -5															

Presentation + Video + Poster																
Prepare the presentation																
Prepare the poster and video																
Demonstrate the project to the examiners																

	Amal
	Ghareisa
	Alaa
	All
	Alaa & Ghareisa

A.3. Anticipated risks

Risks	Proposed solutions
Obtaining wrong results from the sensor	Search for high quality sensors that would give correct results
Data is not transmitted in real-time (there is a delay)	Try to identify the source of the delay, and search for methods to minimize the delay
Obtaining wrong results from the RFID sensor	Change the type of the RFID to one that suits the wanted distance
User does not leave the parking when his reservation time ends	Have several back up parking spots
The application and/or the website receive an unexpectedly huge number of users such that the database storage reaches its limit	Upgrade the Firebase to have more data storage

Appendix B – Use cases specification

Use case Id: UC**01**	Register/ Sign-up
Brief Description:	VIP user registers in the system.
Primary Actors:	VIP user
Secondary Actors:	Cloud
Trigger:	VIP user asks to register in the system.
Preconditions: - VIP user must have a car plate number.	
Post-conditions: - A new record is created for the VIP user in the database.	
Normal Scenario	
Actor Action	**System Response**
1. VIP user enters his/her data (ID, name, car plate number, etc.).	2. check for the validity of the information. (see 2.a)
	3. Create the account and save it in the cloud.
Alternative Flows: 2.a. Ask VIP user to renter any missing information.	

Use case Id: UC**02**	Login / Sign-in
Brief Description:	VIP user logs into the system.
Primary Actors:	VIP User
Secondary Actors:	Cloud
Trigger:	VIP user asks to login to the system.
Preconditions: - The VIP user has a valid account.	
Post-conditions: - The VIP user is logged into the system.	
Normal Scenario	
Actor Action	**System Response**
1. VIP user enters username and password.	2. Validate the entered username and password

	with stored username and password in the cloud. (See 2.a.)
	3. log VIP user into the system.
Alternative Flows: 2.a. If VIP user enters an invalid username and/or password, the system displays an error message.	

Use case Id: UC**03**	Reserve Parking
Brief Description:	VIP user reserves a parking spot for several hours.
Primary Actors:	VIP user
Secondary Actors:	Cloud
Trigger:	VIP user asks to reserve a parking spot.
Preconditions: - VIP user must be registered to the system.	
Post-conditions: - Reservation is created.	
Normal Scenario	
Actor Action	**System Response**
1. VIP user selects zone.	
2. VIP user selects a date.	3. Request parking data from cloud.
4. Cloud sends data.	5. Display if there are available parking spots at selected zone for each hour.
6. VIP user selects start time and duration.	7. Check if the user does not have a reservation at the selected hours. (see 7.a)
	8. Check if there is an available parking spot at all selected hours. (see 8.a)
	9. Check if selected date is equal to today or tomorrow (selected date == current date || selected date == current date +1). (see 9.a)
	10. If selected date is today, check if the reservation start time has passed. (selected start hour > current hour). (see 10.a)
	11. Check if the number of selected hours with the total reservation hours for this date is less than or equal to the number of allowable reservation hours per day (6 hours). (see 11.a)
	12. Create a reservation record in the cloud.
	13. Calculate the total price and assign it to the reservation.
	14. Assign “created” to the reservation status.
	15. Start background notification service that notifies the VIP user 30 minutes before expiring time.
Alternative Flows: 7.a. If the user has a reservation at the selected time, display error message. 8.a. If there is no available parking spot at one of the selected hours, display error message. 9.a. If selected date is not equal to the current date or greater than the current date by more	

than 1 day, display error message to inform user that he can only reserve at the same day or one day before the reservation date. 10.a. If the selected start hour is less than or equal to current hour, display error message to inform user that selected start time has elapsed. 11.a. If the number of selected hours with the total reservation hours for this date is more than the number of allowable reservation hours per day, display error message.

Use case Id: UC**04**	View Reservation
Brief Description:	The VIP user views a reservation.
Primary Actors:	VIP user
Secondary Actors:	Cloud
Trigger:	VIP user asks to view reservation.
Preconditions: - The VIP user must have a reservation.	
Post-conditions: - Reservation details was displayed to the VIP user.	
Normal Scenario	
Actor Action	**System Response**
1. VIP user asks to view a reservation.	2. Request current and upcoming reservations related to the VIP user.
3. Cloud sends data.	4. Display requested data. (see 4.a)
	5. If user select extend option, <extend: Extend Reservation use case>.
	6. If user select cancel option, <extend: Cancel Reservation use case>.
Alternative Flows: 4.a. If there are no reservations associated with the user, display a message.	

Use case Id: UC**05**	Extend Reservation
Brief Description:	The VIP user extends a reservation and the extension time is by default one hour.
Primary Actors:	VIP user
Secondary Actors:	Cloud
Trigger:	VIP user asks to extend reservation.
Preconditions: - The VIP user must have reservation.	
Post-conditions: - Reservation status was extended.	

- Total price was updated.	
Normal Scenario	
Actor Action	**System Response**
1. VIP user selects extend option.	2. Check if the selected reservation is at its last hour (current hour == last hour of reservation). (see 2.a)
	3. Check if there is an available parking spot after the reservation time. (see 3.a)
	4. Add one hour to the end of the current reservation time.
	5. Change the status of the reservation to "extended" in the cloud.
	6. Add the extension price to the reservation.
Alternative Flows: 2.a. if the current hour is not equal to the last hour of the selected reservation time., display an error message to inform VIP user that he can only extend his reservation in the last hour of his reservation. 3.a. If there is no available parking, displays a message to the VIP user.	

Use case Id: UC**06**	Cancel Reservation
Brief Description:	The VIP user cancels a reservation.
Primary Actors:	VIP user
Secondary Actors:	Cloud
Trigger:	VIP user requests to cancel reservation.
Preconditions: - The VIP user must already have a reservation.	
Post-conditions: - Reservation status is cancelled. - The price of the reservation is deducted.	
Normal Scenario	
Actor Action	**System Response**
1.VIP user selects cancel option.	2. Check if the reservation has not started in order to cancel the whole reservation (current time < start time). (see 2.a)
	3. Change the status of the reservation to "cancelled" or "subcancelled" in the cloud.
	4. Calculate the deducted amount (depending on the number of hours cancelled) and deduct it from the total price of the reservation.

Alternative Flows: 2.a. If the reservation has started, cancel remaining reservation hours by going to step 3 with status “subcancelled”.

Use case Id: UC**07**	VIP Check-In
Brief Description:	The system reads the RFID tag on the car, checks its validity, then allows reserved VIP users to enter the parking lot.
Primary Actors:	VIP user
Secondary Actors:	Cloud
Trigger:	VIP user arrives at the gates of the parking lot.
Preconditions: - The parking lot must have at least one reserved available parking spot. - The VIP user must be registered to the system. - The VIP user must have reserved a parking spot.	
Post-conditions: - The parking area gate is opened for the VIP user. - Reservation status is updated to “arrived”	
Normal Scenario	
Actor Action	**System Response**
1. VIP user arrives at the gates of the parking lot.	2. Read the car tag (UID) at the parking lot gates by RFID sensor reader.
	3. Receive UID from reader by Arduino and send it with the zone name to NodeMCU.
	4. Request all the reservations with a selected UID and zone from the cloud by NodeMCU.
5. Cloud sends data.	6. Search for all the reservations where date is today by NodeMCU. (see 6.a)
	7. Search for all the reservations where status is “created” by NodeMCU. (see 7.a)
	8. Find a reservation where the current hour is at the reservation time by NodeMCU (start time <= current hour <= last hour). (see 8.a)
	9. Change reservation status to “arrived” by NodeMCU
	10. Send response (“open” or “close”) to Arduino by NodeMCU.
	11. Check response in order to open the gate or not by Arduino.
Alternative Flows: 6.a. If no reservation exists, go to step 10 with “close” response. 7.a. If no reservation exists, go to step 10 with “close” response.	

8.a. If no reservation exists, go to step 10 with “close” response.

Use case Id: UC**08**	VIP Check-Out
Brief Description:	The system reads the RFID tag on the car, checks its validity, then allow VIP users to exit the parking lot
Primary Actors:	VIP user
Secondary Actors:	Cloud
Trigger:	VIP user exits the parking lot.
Preconditions: - The VIP user must be registered to the system. - The VIP user must have reserved a parking spot. - The VIP user must have arrived to his reservation.	
Post-conditions: - The parking area gate is opened for the VIP user. - Reservation status is updated to “ended”	
Normal Scenario	
Actor Action	**System Response**
1. VIP user arrives at the gates of the parking lot.	2. Read the car tag (UID) at the parking lot gates by RFID sensor reader.
	3. Receive UID from reader by Arduino and send it with the zone name to NodeMCU.
	4. Request all the reservations with a selected UID and zone from the cloud by NodeMCU.
5. Cloud sends data.	6. Search for all the reservations where date is today by NodeMCU. (see 6.a)
	7. Search for all the reservations where status is “arrived” or “extended” or “subcancelled” by NodeMCU. (see 7.a)
	8. Check if the reservation has more than or equal to 1 hour left by NodeMCU (current hour < end time) (see 8.a)
	9. Apply automatic cancellation deduction by NodeMCU, then go to step 13
	10. Find a reservation where its time has passed by NodeMCU (current hour >= end time) (see 10.a)
	11. Apply penalty for exceeding reservation time by NodeMCU.
	12. Change reservation status to “ended” by NodeMCU.
	13. Send response (“open” or “close”) to Arduino by NodeMCU.

	14. Check response in order to open the gate or not by Arduino.
Alternative Flows: 6.a. If no reservation exists, go to step 13 with “close” response. 7.a. If no reservation exists, go to step 13 with “close” response. 8.a. If the reservation has less than 1 hour left, go to step 12 10.a. if no reservation was found, go to step 12	

Use case Id: UC**09**	View Parking
Brief Description:	The user views a map with the current status of parking spots and get directions for a specific spot.
Primary Actors:	User
Secondary Actors:	Cloud
Trigger:	User asks to view current available parking spots.
Preconditions:	
Post-conditions: - A map with the current status of parking spots is displayed to the user.	
Normal Scenario	
Actor Action	**System Response**
1. User select parking zone.	2. Request zone data from cloud.
3. Cloud sends data.	4. Display a map with the current status of parking spots.
5. User select a parking spot to get direction.	6. Redirect to the google map
Alternative Flows:	

Use case Id: UC**10**	Update Parking Availability
Brief Description:	The system receives updated data from sensors through the Arduino board.
Primary Actors:	User
Secondary Actors:	Cloud
Trigger:	Sensor senses a user’s car entering or leaving the parking spot.
Preconditions:	
Post-conditions: - Status of a specific parking spots is updated.	
Normal Scenario	
Actor Action	**System Response**

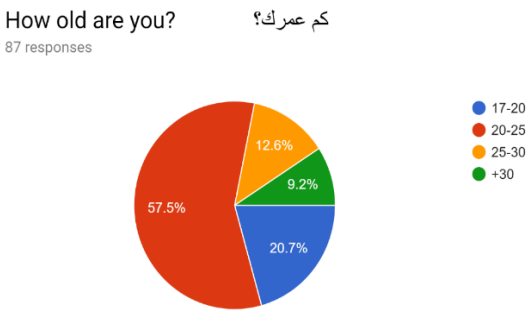
1. User enters or leaves a parking spot in a specific zone.	2. Sense a car entering or leaving the parking spot in a specific zone by Ultrasonic sensor.
	3. Receive data (status, spot number and zone name) from sensors by Arduino and send it to the NodeMCU.
	4. Update the status of the parking spot in a specific zone to “available” or “unavailable” in the cloud by NodeMCU.
Alternative Flows:	

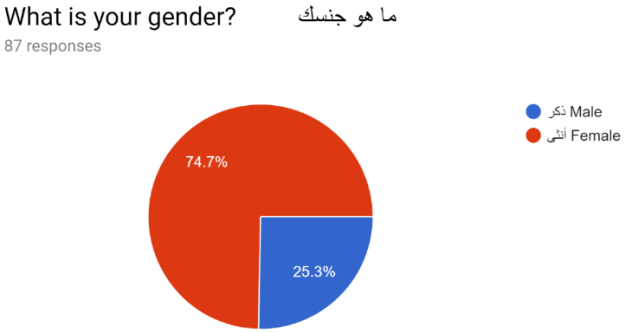
Use case Id: UC**11**	Request Car Care
Brief Description:	The VIP user request car care services.
Primary Actors:	VIP user
Secondary Actors:	Servesni company
Trigger:	VIP user asks to request car care service.
Preconditions: - VIP user must be registered to the system.	
Post-conditions: - VIP user is redirected to the Servesni application or website.	
Normal Scenario	
Actor Action	**System Response**
1. VIP user asks to request car care service.	2. Display services that Servesni company provides.
3. VIP user selects one of the services.	4.Redirect user to the Servesni application or website.
Alternative Flows:	

Use case Id: UC**12**	View Current Occupancy Trend
Brief Description:	The user views statistical data that represent occupied percentage in each hour for selected zone for last four weeks.
Primary Actors:	User
Secondary Actors:	Cloud
Trigger:	User select to view current occupancy trend option (Histogram).
Preconditions:	
Post-conditions: - Statistical data (Histogram) is displayed to the user.	
Normal Scenario	
Actor Action	**System Response**

1. User selects zone.	2. Request zone data from cloud.
3. Cloud send data.	
4. User select a day in the week (Sun, Mon, Tue, …. etc.).	5. Search for statistical data where day equal to selected day.
	6. Calculate for each hour, % Occupancy = total sum of occupancy in selected day for past four weeks / (number of weeks * total number of spots).
	7. Display % Occupancy in each hour in the form of a histogram.
Alternative Flows:	

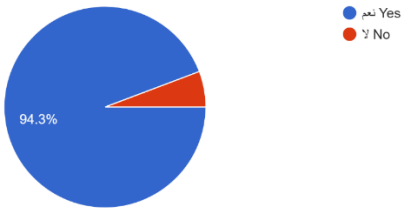
Appendix C – Survey Statistics





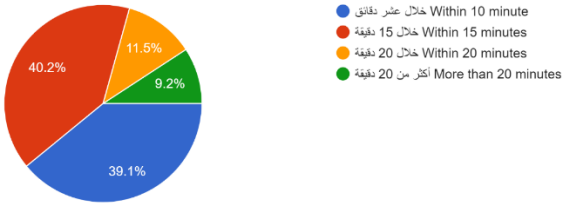
Do you think there is a parking problem in Qatar University?
هل تعتقد أن جامعة قطر تعاني من مشاكل في مواقف السيارات؟

87 responses



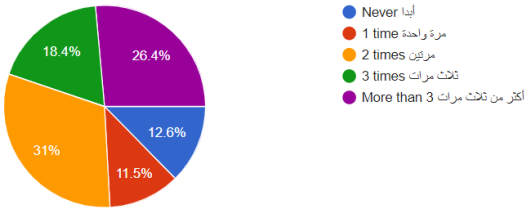
How much time do you spend to find an empty parking slot in Qatar University?
كم دقيقة تستغرق للعثور على موقف فارغ في جامعة قطر؟

87 responses



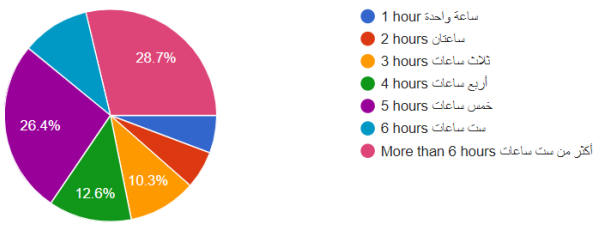
How many times a week do you arrive late to your classes because you spent a lot of time looking for an empty parking spot?
كم مرة في الأسبوع تصل متأخراً الى محاضراتك بسبب استغراقك وقتاً طويلاً في العثور على موقف فارغ؟

87 responses



If there were a parking application for reservations, how many hours a day would you reserve?
إذا كان هناك تطبيق يوفر خدمة الحجز المسبق لمواقف السيارات في جامعة قطر فكم عدد الساعات التي سوف تقوم بحجزها في اليوم الواحد؟

87 responses



Appendix D – Connectivity Test

1st Operation: Get reservations (Node of Objects)

NodeMCU connection time to Firebase	
Trials	**Get Reservation (177 Bytes)**
1	0.350258 s
2	0.401654 s
3	0.330333 s
4	0.331711 s
Average	**0.353489 s**

2nd Operation: Update a value

NodeMCU connection time to Firebase		
Trials	**Get a value (15 Bytes)**	**Set a value (15 Bytes)**
1	0.306381 s	0.383343 s
2	0.295115 s	0.390443 s
3	0.296623 s	0.387329 s
4	0.306095 s	0.393172 s
Average	**0.301054 s**	**0.388572 s**

Application connection time to		
Trials	**Post User (113 Bytes)**	**Get User (113 Bytes)**
1	0.011711 s	0.007428 s
2	0.011810 s	0.007435 s
3	0.011822 s	0.007423 s
4	0.011734 s	0.007430 s
Average	**0.011777 s**	**0.007429 s**

Application connection time to		
Trials	**Post Currently Looking (52 Bytes)**	**Get Currently Looking (52 Bytes)**
1	0.011628 s	0.007298 s
2	0.011623 s	0.007295 s
3	0.011625 s	0.007294 s
4	0.011630 s	0.007299 s
Average	**0.011627 s**	**0.007296 s**

Application connection time to		
Trials	**Post Reservation (177 Bytes)**	**Get Reservation (177 Bytes)**
1	0.012106 s	0.007584 s
2	0.012110 s	0.007590 s
3	0.012109 s	0.007588 s
4	0.012115 s	0.007583 s
Average	**0.012112 s**	**0.007586 s**

Application connection time to		
Trials	**Post Zone (8606 Bytes)**	**Get Zone (8606 Bytes)**
1	0.017234 s	0.008668 s
2	0.017236 s	0.008664 s
3	0.017239 s	0.008663 s
4	0.017235 s	0.008666 s
Average	**0.017236 s**	**0.008666 s**

Application connection time to		
Trials	**Post Spot (49 Bytes)**	**Get Spot (49 Bytes)**
1	0.011525 s	0.007122 s
2	0.011524 s	0.007121 s
3	0.011526 s	0.007120 s
4	0.011528 s	0.007124 s
Average	**0.011526 s**	**0.007123 s**

Website connection time to		
Trials	**Post Reservation (177 Bytes)**	**Get Reservation (177 Bytes)**
1	0.490 s	0.411 s
2	0.445 s	0.420 s
3	0.480 s	0.451 s
4	0.482 s	0.423 s
5	0.522 s	0.416 s
Average	**0.4838 s**	**0.4242 s**

Website connection time to		
Trials	**Post Currently Looking**	**Get Currently Looking**

	(52 Bytes)	(52 Bytes)
1	0.383 s	0.370 s
2	0.370 s	0.386 s
3	0.313 s	0.406 s
4	0.410 s	0.360 s
5	0.436 s	0.323 s
Average	**0.3824 s**	**0.3690 s**

Website connection time to		
Trials	**Post User (113 Bytes)**	**Get User (113 Bytes)**
1	0.489 s	0.418 s
2	0.409 s	0.373 s
3	0.456 s	0.436 s
4	0.488 s	0.369 s
5	0.414 s	0.428 s
Average	**0.4512 s**	**0.4048 s**

Website connection time to		
Trials	**Post Zone (8606 Bytes)**	**Get Zone (8606 Bytes)**
1	0.662 s	0.722 s
2	0.666 s	0.716 s
3	0.656 s	0.652 s
4	0.674 s	0.712 s
5	0.712 s	0.698 s
Average	**0. 8122 s**	**0.8303 s**

Website connection time to		
Trials	**Post spot (49 Bytes)**	**Get Spot (49 Bytes)**
1	0.362 s	0.349 s
2	0.287 s	0.320 s
3	0.342 s	0.324 s
4	0.317 s	0.328 s
5	0.345 s	0.328 s
Average	**0.3306 s**	**0.3298 s**

Appendix E – Test Cases for Functional Testing

- Sign-up:

Test Case Id: Sign-up 01	
Test Purpose	VIP user successfully signs up
Success Criteria	VIP user account is created
Test Steps for Android Application: 1. Click on “New to ParQU? Join now” link 2. Fill all the fields 3. Click “Create account” button **Test Steps for website:** 1. Click on “Join as VIP user” button. 2. Fill all the fields 3. Click “Register” button	
Post-Conditions: • User information is added to the database correctly. • User is redirected to “Services” page. • User is authorized to use the system.	

Test Case Id: Sign-up 02	
Test Purpose	VIP user uses an already existing email.
Success Criteria	VIP user is informed that he/she tried to use an already existing email.
Test Steps for Android Application: 1. Click on “New to ParQU? Join now” link 2. Enter a used email address 3. Fill all other fields 4. Click “Create account” button **Test Steps for website:** 1. Click on “Join as VIP user” button. 2. Enter a used email address. 3. Fill all the fields. 4. Click “Register” button	
Post-Conditions: • Error message “You are already registered” appears to VIP user	

Test Case Id: Sign-up 03	
Test Purpose	VIP user leaves required fields empty
Success Criteria	VIP user is informed that he/she tried to leave required fields empty.
Test Steps for Android Application: 1. Click on “New to ParQU? Join now” link 2. Intentionally leave empty fields while filling the form 3. Click “Create account” button **Test Steps for the website:** 1. Click on “Join as VIP user” button. 2. Intentionally leave empty fields while filling the form 3. Click “Register” button	
Post-Conditions: • VIP user is prompted to fill all information	

- Sign-in:

Test Case Id: Sign-in 01	
Test Purpose	VIP user successfully signs in
Success Criteria	VIP user is redirected to “Services” page
Test Steps for Android Application: 1. Click on “join as VIP user” button 2. If you do not have account, create one as shown in Sign-up 01 test case 3. Fill email and password field correctly 4. Click “LOGIN” button **Test Steps for the website:** 1. Click on “join as VIP user” button 2. If you do not have account, create one as shown in Sign-up 01 test case 3. Fill email and password field correctly 4. Click “LOGIN” button	
Post-Conditions: • User is redirected to “Services” page. • User is authorized to use the system.	

Test Case Id: Sign-in 02

Test Purpose	VIP user uses a wrong email address and/or password.
Success Criteria	VIP user is informed that he/she tried to use a wrong email address and/or password.
Test Steps for Android Application: 1. Click on “join as VIP user” button 2. If you do not have account, create one as shown in Sign-up 01 test case 3. Intentionally enter wrong email or password 4. Click “LOGIN” button **Test Steps for the website:** 1. Click on “join as VIP user” button 2. If you do not have account, create one as shown in Sign-up 01 test case 3. Intentionally enter wrong email or password 4. Click “LOGIN” button	
Post-Conditions: • Error message appears to VIP user	

- <u>View Parking:</u>

Test Case Id: View Parking 01	
Test Purpose	User successfully views a map with the current status of parking spots
Success Criteria	User is redirected to “Map” page
Test Steps for Android Application: 1. If you are VIP user, sign in as shown in Sign-in 01 test case 2. If you are Normal user, click on “Continue as Normal User” button 3. Click on “Check Availability” button 4. Select a zone from zones list **Test Steps for the website:** 1. If you are VIP user, sign in as shown in Sign-in 01 test case 2. If you are Normal user, click on “Continue as Normal User” button from the home page. 3. Select a zone from zones images. 4. Click on “Show Availability” button	
Post-Conditions: • User is redirected to “Map” page.	

Test Case Id: View Parking 02	
Test Purpose	User successfully gets directions for a specific spot.
Success Criteria	User is redirected to Google map

Test Steps for Android Application: 1. If you are VIP user, sign in as shown in Sign-in 01 test case 2. If you are Normal user, click on “Continue as Normal User” button 3. Click on “Check Availability” button 4. Select a zone from zones list 5. Select a spot number 6. Click on “Get Directions” **Test Steps for the website:** 1. If you are VIP user, sign in as shown in Sign-in 01 test case 2. If you are Normal user, click on “Continue as Normal User” button from the home page. 3. Select a zone from the zone’s images. 4. Click on “Show Availability” button. 5. Select spot number from the drop-down list. 6. Click on “Get Directions”
Post-Conditions: • User is redirected to Google map

- Reserve Parking:

Test Case Id: Reserve Parking 01	
Test Purpose	VIP user reserves a parking spot successfully.
Success Criteria	Reservation record is created in the database
Test Steps for Android Application: 1. Sign in as shown in Sign-in 01 test case 2. Click on “Reserve a Parking” button 3. Select a zone 4. Select date, start time and duration 5. Click on “Reserve” button 6. Click “Yes” to confirm your reservation **Test Steps for the website:** 1. Sign in as shown in Sign-in 01 test case 2. Click on “Reserve a Parking” link. 3. Select a zone from the drop-down list. 4. Select date, start time and duration 5. Click on “Reserve for me” button	
Post-Conditions: • Reservation record is created in the database	

Test Case Id: Reserve Parking 02	
Test Purpose	VIP user already has a reservation at the selected time he/she wants to reserve on.
Success Criteria	VIP user is notified that he/she has a reservation at the selected time.

Test Steps for Android Application:

1. Sign in as shown in Sign-in 01 test case
2. Click on “Reserve a Parking” button
3. Select a zone
4. Select date, start time and duration
5. Click on “Reserve” button
6. Click “Yes” to confirm your reservation

Test Steps for the website:

1. Sign in as shown in Sign-in 01 test case
2. Click on “Reserve a Parking” link.
3. Select a zone from the drop-down list.
4. Select a date, start time and duration
5. Click on “Reserve for me” button

Post-Conditions:

- Error message “You already has a reservation at the selected time” appears to VIP user

Test Case Id: Reserve Parking 03	
Test Purpose	VIP user tries to reserve at a time where there is no available parking spot.
Success Criteria	VIP user is notified that there is no available parking spot at one of the selected hours.

Test Steps for Android Application:

1. Sign in as shown in Sign-in 01 test case
2. Click on “Reserve a Parking” button
3. Select a zone
4. Select date, start time and duration
5. Click on “Reserve” button
6. Click “Yes” to confirm your reservation

Test Steps for the website:

1. Sign in as shown in Sign-in 01 test case
2. Click on “Reserve a Parking” link.
3. Select a zone from the drop-down list.
4. Select a date, start time and duration
5. Click on “Reserve for me” button

Post-Conditions:

- Error message “No available parking spot at one of the selected hours” appears to VIP user

Test Case Id: Reserve Parking 04	
Test Purpose	VIP user tries to reserve before the allowable reservation time which is the same day or one day before.
Success Criteria	VIP user is notified that he can only reserve at the same day or one day before the reservation date.

Test Steps for Android Application:

1. Sign in as shown in Sign-in 01 test case
2. Click on “Reserve a Parking” button
3. Select a zone
4. Select date, start time and duration
5. Click on “Reserve” button
6. Click “Yes” to confirm your reservation

Test Steps for the website:

1. Sign in as shown in Sign-in 01 test case
2. Click on “Reserve a Parking” link.
3. Select a zone from the drop-down list.
4. Select a date, start time and duration
5. Click on “Reserve for me” button

Post-Conditions:

- Error message “You can only reserve at the same day or one day before the reservation date” appears to VIP user

Test Case Id: Reserve Parking 05	
Test Purpose	VIP user tries to reserve at a time that has elapsed.
Success Criteria	VIP user is notified that the selected start time has elapsed.

Test Steps for Android Application:

1. Sign in as shown in Sign-in 01 test case
2. Click on “Reserve a Parking” button
3. Select a zone
4. Select date, start time and duration
5. Click on “Reserve” button
6. Click “Yes” to confirm your reservation

Test Steps for the website:

1. Sign in as shown in Sign-in 01 test case
2. Click on “Reserve a Parking” link.
3. Select a zone from the drop-down list.
4. Select a date, start time and duration

5. Click on “Reserve for me” button
Post-Conditions: • Error message “The selected start time has elapsed” appears to VIP user

Test Case Id: Reserve Parking 06	
Test Purpose	VIP user tries to reserve more than the number of allowable reservation hours per day (6 hours)
Success Criteria	VIP user is notified that he/she cannot reserve more than the number of allowable reservation hours per day (6 hours)
Test Steps for Android Application: 1. Sign in as shown in Sign-in 01 test case 2. Click on “Reserve a Parking” button 3. Select a zone 4. Select date, start time and duration 5. Click on “Reserve” button 6. Click “Yes” to confirm your reservation **Test Steps for the website:** 1. Sign in as shown in Sign-in 01 test case 2. Click on “Reserve a Parking” link. 3. Select a zone from the drop-down list. 4. Select a date, start time and duration 5. Click on “Reserve for me” button	
Post-Conditions: • Error message “You exceed the total allowed number of hours which is 6 hours per day” appears to VIP user	

Test Case Id: Reserve Parking 07	
Test Purpose	VIP user is notified 30 minutes before reservation expiring time.
Success Criteria	Notification appear to the user before 30 minutes of the reservation expiring time.
Test Steps for Android Application: 1. Sign in as shown in Sign-in 01 test case 2. Click on “Reserve a Parking” button 3. Select a zone 4. Select date, start time and duration 5. Click on “Reserve” button 6. Click “Yes” to confirm your reservation	

Test Steps for the website: 1. Sign in as shown in Sign-in 01 test case 2. Click on “Reserve a Parking” link. 3. Select a zone from the drop-down list. 4. Select a date, start time and duration 5. Click on “Reserve for me” button
Post-Conditions: • Notification message “Reservation No XX in zone XX is due after 30 minutes” appear to the user before 30 minutes of the reservation expiring time.

- View Reservation:

Test Case Id: View Reservation 01	
Test Purpose	VIP user successfully views all his/her current and upcoming reservations.
Success Criteria	VIP user is redirected to “Reservations” page
Test Steps for Android Application: 1. Sign in as shown in Sign-in 01 test case 2. Click on “Show Reservations” button **Test Steps for the website:** 1. Sign in as shown in Sign-in 01 test case. 2. Click on “My Reservations” link on navigation bar.	
Post-Conditions: • VIP user is redirected to “Reservations” page	

- Extend Reservation:

Test Case Id: Extend Reservation 01	
Test Purpose	VIP user successfully extends a reservation
Success Criteria	Reservation is extended by one hour
Test Steps for Android Application: 1. Sign in as shown in Sign-in 01 test case 2. Click on “Show Reservations” button 3. Click on “Extend” option **Test Steps for the website:** 1. Sign in as shown in Sign-in 01 test case 2. Click on “My Reservations” link on navigation bar. 3. Click on “Extend” button.	

4. Click on “ok” button to confirm extending.
Post-Conditions: • Reservation status changed to “extended” • Extension price is added to the reservation

Test Case Id: Extend Reservation 02	
Test Purpose	VIP user tries to extend before the last hour of the reservation.
Success Criteria	VIP user is notified that he/she can only extend his reservation in the last hour of his reservation
Test Steps for Android Application: 1. Sign in as shown in Sign-in 01 test case 2. Click on “Show Reservations” button 3. Click on “Extend” option **Test Steps for the website:** 1. Sign in as shown in Sign-in 01 test case 2. Click on “My Reservations” link on navigation bar. 3. Click on “Extend” button. 4. Click on “ok” button to confirm extending.	
Post-Conditions: • Error message “You can only extend at the last hour of your reservation” appears to VIP user	

Test Case Id: Extend Reservation 03	
Test Purpose	VIP user tries to extend when there is no available parking spot after the reservation time.
Success Criteria	VIP user is notified that there is no available parking spot after the reservation time
Test Steps for Android Application: 1. Sign in as shown in Sign-in 01 test case 2. Click on “Show Reservations” button 3. Click on “Extend” option **Test Steps for the website:** 1. Sign in as shown in Sign-in 01 test case 2. Click on “My Reservations” link on navigation bar. 3. Click on “Extend” button. 4. Click on “ok” button to confirm extending.	

Post-Conditions: • Error message “no available parking spots” appears to VIP user

- Cancel Reservation:

Test Case Id: Cancel Reservation 01	
Test Purpose	VIP user successfully cancels the whole reservation, if the reservation has not started.
Success Criteria	The whole reservation is cancelled, and the deduction amount is calculated and deducted from the total price of the reservation
Test Steps for Android Application: 1. Sign in as shown in Sign-in 01 test case 2. Click on “Show Reservations” button 3. Click on “Cancel” option **Test Steps for the website:** 1. Sign in as shown in Sign-in 01 test case 2. Click on “My reservation” link on the navigation. 3. Click on “Cancel” option. 4. Click on “ok” button to confirm cancellation.	
Post-Conditions: • Reservation status changed to “cancelled” • Deduction amount is calculated and deducted from the total price of the reservation	

Test Case Id: Cancel Reservation 02	
Test Purpose	VIP user successfully cancels remaining reservation hours from now, if the reservation has started
Success Criteria	A part of the reservation hours is cancelled, and the deduction amount for cancelled hours is calculated and deducted from the total price of the reservation
Test Steps for Android Application: 1. Sign in as shown in Sign-in 01 test case 2. Click on “Show Reservations” button 3. Click on “Cancel” option **Test Steps for the website:** 1. Sign in as shown in Sign-in 01 test case 2. Click on “My reservation” link on the navigation. 3. Click on “Cancel” option. 4. Click on “ok” button to confirm cancellation.	
Post-Conditions: • Reservation status changed to “subcancelled”	

• Deduction amount is calculated and deducted from the total price of the reservation

- Request Car Care:

Test Case Id: Request Car Care 01	
Test Purpose	VIP user is successfully redirected to the Servesni application if the application is installed in the phone
Success Criteria	VIP user is redirected to Servesni application
Test Steps for Android Application: 1. Sign in as shown in Sign-in 01 test case 2. Click on “Request Car Care” button 3. Select one of the services	
Post-Conditions: • VIP user is redirected to Servesni application	

Test Case Id: Request Car Care 02	
Test Purpose	VIP user redirected to the Play Store, if the application is not installed in the phone
Success Criteria	VIP user is redirected to Play Store
Test Steps for Android Application: 1. Sign in as shown in Sign-in 01 test case 2. Click on “Request Car Care” button 3. Select one of the services **Test Steps for the website:** 1. Sign in as shown in Sign-in 01 test case 2. Click on “Request Car Care” link from the services page.	
Post-Conditions: • VIP user is redirected to Play Store	

- View Current Occupancy Trend:

Test Case Id: View Current Occupancy Trend 01	
Test Purpose	User successfully views statistical data.
Success Criteria	Statistical data shown to the user

Test Steps for Android Application: 1. If you are VIP user, sign in as shown in Sign-in 01 test case 2. If you are Normal user, click on “Continue as Normal User” button 3. Click on “Check Availability” button 4. Select a zone from zones list **Test Steps for the website:** 1. If normal user, go to “Check availability” page as shown in “view parking” test case. 2. Select a zone from the zone’s images. 3. Click on “show current trend” button. 4. If VIP user, sing in as shown in Sign-in 01 test case 5. Click on “View current trend” option in “Services” page. 6. Select a zone from the zone’s images. 7. Click on “show current trend” button.
Post-Conditions: • A modal with statistical data appears to the user.

- Check in:

Test Case Id: Check In 01	
Test Purpose	VIP user successfully checks in to zone if he has reservation at the current time and zone.
Success Criteria	Parking gate is opened and green LED is on
Test Steps: 4. VIP user scans his RFID tag on RFID reader	
Post-Conditions: • Parking lot gate is opened and green LED is on • Reservation status is changed to “arrived”	

Test Case Id: Check In 02	
Test Purpose	VIP user tries to check in to zone when he does not have a reservation at the current time
Success Criteria	Parking gate is closed and red LED is on
Test Steps: **1.** VIP user scans his RFID tag on RFID reader	
Post-Conditions: • Parking lot gate is closed and red LED is on	

Test Case Id: Check In 03	
Test Purpose	VIP user tries to check in to zone when his reservation is in another zone
Success Criteria	Parking gate is closed and red LED is on
Test Steps: **1.** VIP user scans his RFID tag on RFID reader	
Post-Conditions: • Parking lot gate is closed and red LED is on	

Test Case Id: Check In 04	
Test Purpose	VIP user tries to check in to zone when he cancels his reservation
Success Criteria	Parking gate is closed and red LED is on
Test Steps: **1.** VIP user scans his RFID tag on RFID reader	
Post-Conditions: • Parking lot gate is closed and red LED is on	

- <u>Check out:</u>

Test Case Id: Check Out 01	
Test Purpose	VIP user successfully checks out of zone *when/before* his reservation time has ended (without penalty)
Success Criteria	Parking gate is opened and green LED is on
Test Steps: 1. VIP user scans his RFID tag on RFID reader	
Post-Conditions: • Parking lot gate is opened and green LED is on • Reservation status is changed to “ended”	

Test Case Id: Check Out 02	
Test Purpose	VIP user checks out of zone *after* his reservation time has ended (with penalty)
Success Criteria	Parking gate is opened and green LED is on

Test Steps: 1. VIP user scans his RFID tag on RFID reader
Post-Conditions: • Parking lot gate is opened and green LED is on • Reservation status is changed to “ended” • Penalty is added to reservation (+15QR per hour)

Test Case Id: Check Out 03	
Test Purpose	VIP user cancels part of his reservation using application/website and checks out of zone *when/before* his reservation has ended (without penalty)
Success Criteria	Parking gate is opened and green LED is on
Test Steps: 1. VIP user scans his RFID tag on RFID reader	
Post-Conditions: • Parking lot gate is opened and green LED is on • Reservation status is changed to “ended”	

Test Case Id: Check Out 04	
Test Purpose	VIP user cancels part of his reservation using application/website and checks out of zone *after* his reservation has ended (with penalty)
Success Criteria	Parking gate is opened and green LED is on
Test Steps: 1. VIP user scans his RFID tag on RFID reader	
Post-Conditions: • Parking lot gate is opened and green LED is on • Reservation status is changed to “ended” • Penalty is added to reservation (+15QR per hour)	

Test Case Id: Check Out 05	
Test Purpose	VIP user checks out of zone *when/before* his extension time has ended (without penalty)
Success Criteria	Parking gate is opened and green LED is on
Test Steps: 1. VIP user scans his RFID tag on RFID reader	

Post-Conditions: • Parking lot gate is opened and green LED is on • Reservation status is changed to “ended”

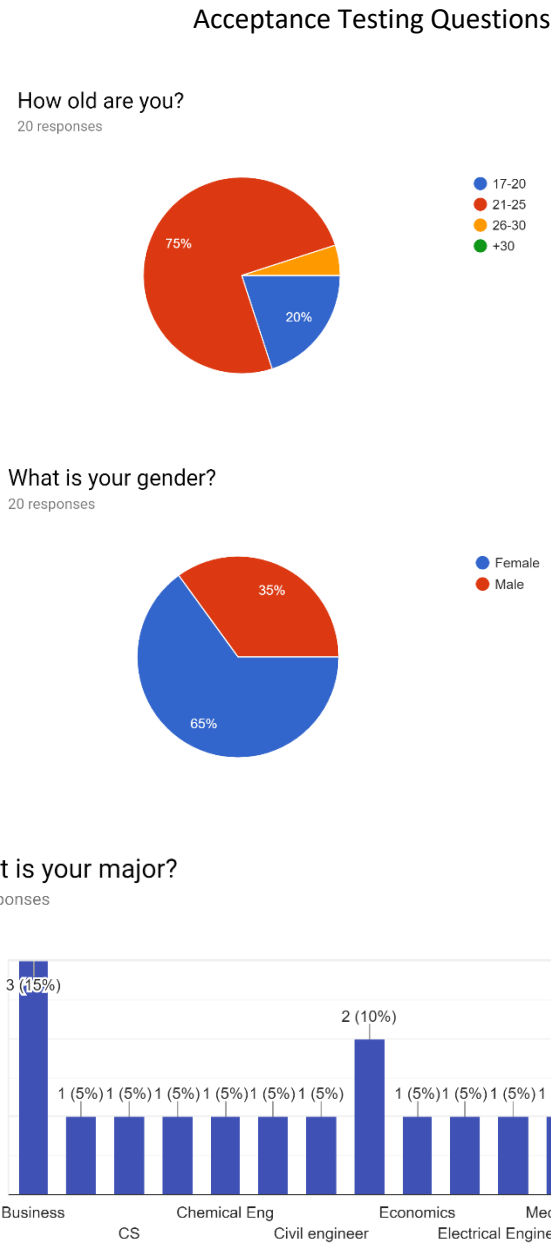
Test Case Id: Check Out 06	
Test Purpose	VIP user checks out of zone *after* his extension time has ended (with penalty)
Success Criteria	Parking gate is opened and green LED is on
Test Steps: 1. VIP user scans his RFID tag on RFID reader	
Post-Conditions: • Parking lot gate is opened and green LED is on • Reservation status is changed to “ended” • Penalty is added to reservation (+15QR per hour)	

Test Case Id: Check Out 07	
Test Purpose	VIP user checks out of zone and automatic cancellation is applied when his reservation time has an hour or more left.
Success Criteria	Parking gate is opened and green LED is on
Test Steps: 1. VIP user scans his RFID tag on RFID reader	
Post-Conditions: • Parking lot gate is opened and green LED is on • Reservation status is changed to “ended” • Automatic cancellation is applied (-2.5 per hour)	

- <u>Update Availability:</u>

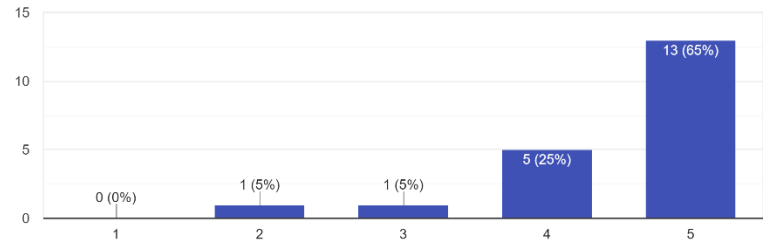
Test Case Id: Update Parking Availability 01	
Test Purpose	User successfully enters/leaves a parking spot and the availability status of the parking spot is updated.
Success Criteria	Parking spot status is updated
Test Steps: 1. User enters/leaves a parking spot	
Post-Conditions: • Parking spot status is updated	

Appendix F – Acceptance Testing Questions



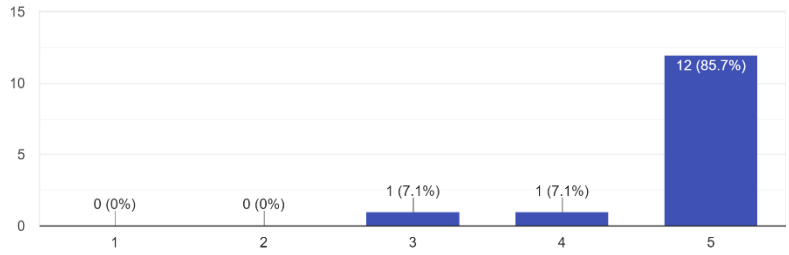
How easy did you find ParQU to be? Rate on a scale of 1–5 (1 being very complex, 5 being very easy)

20 responses



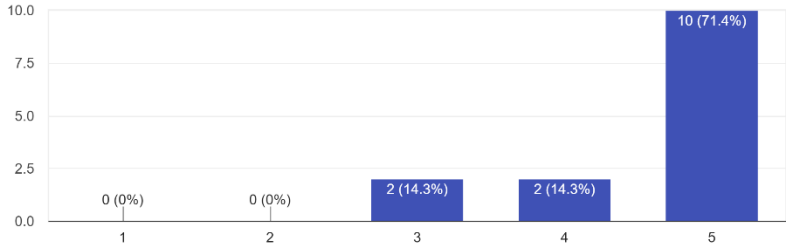
Is the user interface of the Android application pleasing to look at and work with? (Rate the UI design on a scale of 1–5 based on design preference)

14 responses



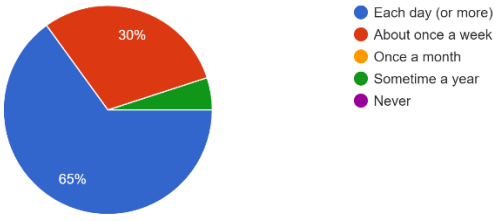
Is the user interface of the website pleasing to look at and work with? (Rate the UI design on a scale of 1–5 based on design preference)

14 responses



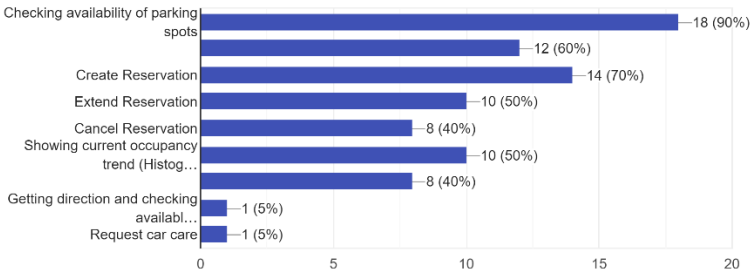
How often would you use this service if it was provided to you in real life?

20 responses



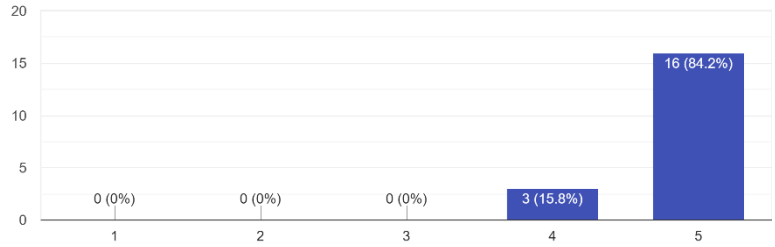
Which feature(s) of ParQU is the most useful for you and why? (Select all that applies)

20 responses



Would you recommend this app to your colleagues and friends? (1 being never, 5 being definitely)

19 responses



Please provide any additional suggestions and comments

7 responses

Reserve the parking two days before
The app colors and organization could be modified to be more clear for the user
I hope it works as new inventory to QU students, it would fit our needs
All service are good an helpful except get directions no need for it
ترجمة الموقع إلى اللغة العربية .. :) ، واتاحته لمستخدمي الايفون
I like it good job
Valley services