Smart Parking Assistant

Mohammad Abdul Rafay - BSE183009

Saif Akhter BSE18303

Spring-2022

Supervised By

Dr. Aamer Nadeem

Department of Software Engineering

Capital University of Science & Technology, Islamabad

Submission Form for Final-Year

PROJECT REPORT

Version	4.0

NUMBER OF MEMBERS	2

TITLE	**Smart Parking Assistant**

SUPERVISOR NAME	**Dr. Aamer Nadeem**

MEMBER NAME	REG. NO.	EMAIL ADDRESS
Mohammad Abdul Rafay	BSE183009	99marafay@gmail.com
Saif Akhter	BSE183038	saifakhter1234@gmail.com

MEMBERS' SIGNATURES

_____

_____

_____

Supervisor's Signature

APPROVAL CERTIFICATE

This project, entitled as “Smart Parking Assistant” has been approved for the award of

Bachelors of Science in Software Engineering

Committee Signatures:

Supervisor: _____

(Dr. Aamer Nadeem)

Project Coordinator: _____

(Mr. Ibrar Arshad)

Head of Department: _____

(Dr. Nadeem Anjum)

DECLARATION

We, hereby, declare that “No portion of the work referred to, in this project has been submitted in support of an application for another degree or qualification of this or any other university/institute or other institution of learning”. It is further declared that this undergraduate project, neither as a whole nor as a part thereof has been copied out from any sources, wherever references have been provided.

MEMBERS’ SIGNATURES

DEDICATION

This project is especially dedicated to the teachers who helped and guided us to successfully complete this project work. Also, I would like to dedicate this project to my dear father, who has been a wonderful supporter until my research was completed, and to my beloved mother, who has been encouraging me for months.

Table of Contents

Table of Contents

Chapter 1 10
Introduction 10
1.1. Project Introduction 10
1.2. Existing Examples / Solutions 11
1.3. Business Scope 12
1.3.1 The Smart Parking Solution 12
1.4. Useful Tools and Technologies 13
1.5. Project Work Breakdown 14
1.6. Project Timeline 15
Chapter 2 17
Requirement Specification and Analysis 17
2.1. Functional Requirements 17
2.2. Non-Functional Requirements 18
2.3. System Use Case Modeling 19
2.4. System Sequence diagrams 30
Chapter 3 36
System Design 36
3.1. Software Architecture 36
3.2. Class Diagram 37
3.3. Sequence Diagram 38
3.3.1. Create Account: 38
3.3.2. Login 39
3.3.3. Logout: 40
3.3.4. Reservations Parking Spot: 41
3.3.5. Update Profile: 42
3.3.6. View Profile: 43
3.4. Entity Relationship Diagram 44
3.5. Database Schema 45
3.6. User Interface Design 46
3.7. Hardware to Software COTS 51
Chapter 4 52

Software Development....52
4.1. Development Environment....52
4.2. Software Description....53
4.2.1. Hardware Code:....53
4.2.2. Software Code:....53
Chapter 5....64
System Testing....64
5.1 Testing Methodology....64
5.2. Testing Environment....64
5.3. Test Cases....65
5.3.1 Hardware Test Cases....65
5.3.2 Software Test Cases....69
Chapter 6....75
Software Deployment....75
6.1. Installation / Deployment Process Description....75
6.2. Pre-requisites:....75
6.3 Deployment of Database:....75
6.4: Deploy the website....81
Chapter 7....89
Project Evaluation....89
7.1. Project Evaluation Report....89
References:....90

List of Figures

Figure 1: Work Breakdown 14
Figure 2: FYP part-1 Timeline 15
Figure 3: Timeline FYP-2 16
Figure 4: User Use-Case Diagram 19
Figure 5: Manager Use-Case Diagram 20
Figure 6: Create Account SSD 30
Figure 7: Login SSD 31
Figure 8: View Profile SSD 32
Figure 9: Update Profile SSD 33
Figure 10: Logout SSD 34
Figure 13: Domain Model 35
Figure 14: Architecture Model 36
Figure 15: Class Diagram 37
Figure 16: SD Create Account 38
Figure 17: Login System Diagram 39
Figure 18: Logout System Diagram 40
Figure 19: Reservation System Diagram 41
Figure 20: Update Profile System Diagram 42
Figure 21: View Profile System Diagram 43
Figure 22: ER Diagram 44
Figure 23: Database Schema 45
Figure 24: Home Page 46
Figure 25: Dashboard Page 47
Figure 26: Manager Login 47
Figure 27: Add User Page 48
Figure 28: View user Profiles 48
Figure 29: Sensor Location 49
Figure 30: Parking Map 49
Figure 31: About Page 50
Figure 32: About Page 50
Figure 33: hardware 51
Figure 34: Creating web server 81
Figure 35: Launching installer 82
Figure 36 Instance loading screen 82
Figure 37: Linux settings 83
Figure 38: Define resources 83
Figure 39: Setup networking 84
Figure 40: Allocating memory 85
Figure 41: Launching application 85
Figure 42: Uploading code 86
Figure 43: Setting connection 87
Figure 44: Connect VM Ware 87
Figure 45: Setup terminal 88

LIST OF TABLES

Table 1: Function Requirements 17
Table 3: Non-Functional Requirements 18
Table 4: Create Account Description 21
Table 5: Login Description 22
Table 6: View Profile Description 24
Table 7: Update Profile Description 25
Table 8: logout Description 27
Table 9: Reservation Description 28
Table 10: Parking Map Description 29
Table 11: Hardware test Case 01 65
Table 12: Hardware test Case 02 66
Table 13: Hardware test Case 03 67
Table 14: Hardware test Case 04 68
Table 15: Login test Case 69
Table 16: Data update test case 70
Table 17: Parking Detection test case 71
Table 18: Logout test case 72
Table 19: Space detection test case 04 73
Table 20: Parking map test case 74

Chapter 1

Introduction

1.1. Project Introduction

As of now maximum of the parking areas are manually controlled with the aid of human manpower and there is no automated device to manage the parking region in a green way. The concept is that when a motive force enters any manner of parking zone, he has to look for a few type of information board that tells him approximately the fame of the car parking zone whether or not the car parking zone is absolutely occupied, there are vacant parking spots. Most of the time the driver has to circle around the parking location searching for a vacant parking spot.

The system of looking the unfastened parking space is time eating and also wastage of fuel. Maximum of the instances the parking spaces continue to be unoccupied, however the total occupancy is low because of terrible management of parking zone. This reasons ineffective use of the parking region and also outcomes in traffic jams and congestion close to the parking masses.

To correctly control the parking lot and show every parking spot's statistic to the drivers earlier than entering the car parking zone have come to be an important issue to be resolved. In this mission, a gadget is proposed on the way to detect the total range of to be had parking spots and presentations the facts to the drivers so that it will easily park their vehicles. Sensors may be used to detect loose areas and occupied areas and gift them to the customers in real time through a cell utility.

The demand for the available parking place is a dynamic price, i.E., a value that modifications through the years, so logically there's a want to provide the information on the area and situation of parking offer in actual time to the drivers of passenger cars. The systems for imparting the facts on parking offer to customers already commenced to be used in some world cities extra than 20 years in the past and may be found below names like "A Guided Parking device" or "Dynamic Parking guidance machine". Through using Parking steering and statistics (PGI) structures it is possible to enhance to a degree the manner of parking and growth its success and efficiency. The ecological results of potential PGI structures application are glaring in the reduction of noise (the lowering range of vehicles in sure segments of the delivery community) and in the reduction of gasoline intake and exhaust emissions (shorter journey instances and the growth inside the visitors drift).

Normally, a PGI includes four additives, particularly, the automobile parking space tracking, information dissemination, conversation technology, and the manage system. The PGI structures could be the use of smart sensors for monitoring lots occupancy. The sensors primarily based monitoring is similarly classified as "On-Roadway". The on-avenue sensors are glued to the road surface; examples of such kind of sensors are magnetic sensors and acoustic sensors.

Features:

The features for the system are stated bellow:

- Parking Recommended System.
    - Collecting all of the data from using that data to give recommendation if an organization need to increase their parking space.
- Live Time Monitoring using different proximity sensors.

    - Using our Android application and Web interface the user and the customer can be able to get live feed location parking lot information.
- Reserving Parking Spot
    - Using our application, the customer can have reserved parking spot for multiple purposes, by paying a fee.

1.2. Existing Examples / Solutions

In current market there are a couple of system which are available but some of them are not very functional and practical to implement. Some market product names are stated below:

1. Best Parking
2. Parker
3. Park Whiz

There is some other application which have been developed but there are not very well commercialized, we are designing this application to save time, cost and improve the quality.
However, the main example of this project is to provide a parking assistant for specific, controlled parking spaces on demand. The parking assistant is made to cater to individual needs and is built according to budget, scenario, space, location and traffic.
A good example would be;

- CUST Parking
    - The parking at the main university front area is a controlled parking, it is covered, marked and under surveillance. It has pre embedded electronic systems, electricity, wiring and connections which make it a perfect environment for additional sensor testing.
- Mall parking
    - The underground parking of any commercial mall is a perfect environment for this system to be set up, as it is always controlled, has overhead wiring, vents, power supply and surveillance.

1.3. Business Scope

With extra than 88 million passenger and light automobiles sold each year, the economic zone and huge corporations face increasing parking issues. Imparting adequate parking space for customers, employees and visitors is a pinnacle precedence for stores, resorts, hospitals and corporations.

As call for parking escalates in the course of the day, uncoordinated and inefficient parking management can negatively affect an organization's revenue and overall performance. As an instance, a retailer wishes a consumer to stay in the shop for her hour or, all through which unfastened parking is supplied. The trouble is that even non-customers can gain from it and take up treasured parking area. Additionally, riding round blindly seeking out a parking space may be a very irritating enjoy. Customers, employees, and traffic regularly tour to one-of-a-kind places, ensuing in misplaced sales and reduced productiveness. Moreover, it increases emissions and ecological footprint whilst contributing to energy bills for car parking zone ventilation. Parking area control bottlenecks are not confined to peak hours. After hours, company parking plenty are often closed, in particular in city centers, growing wasted area as citizens run out of avenue parking.

1.3.1 The Smart Parking Solution

Inside the path of the IOT based totally clever-age shrewd parking systems will offer progressive solutions to these demanding situations. Low-energy magnetic, ultrasonic, or optical sensors embedded within the floor document whether or not and for how lengthy a selected area is occupied. This sensor information is wirelessly routed to a gateway and dispatched to a cloud platform for evaluation inside the corporation or constructing proprietors manage room. A smart parking solution that can provide exceptional parking zone visibility gives extensive enterprise benefits.

1. Pinpoint Inefficiencies through Parking length

Real-time information about character automobile parking times can help organizations identify uncommon or unlawful parking hobby so they can proactively reply. As an instance, supermarkets and stores can screen immoderate parking instances which can suggest unauthorized use of parking areas. Different difficult behaviors, including overtime parking and automobiles occupying his two parking areas, may be speedy recognized with automated signals. Further, a parking lot that has been empty for weeks indicates a capability hassle that wishes checking.

2. Enhance Parking studies and cut back Emissions

Streaming statistics and storing it in the cloud allows the improvement of devoted services for a pressure-loose parking enjoy. An API-enabled person app, digital signal, or light indicator retrieves real-time parking data from the cloud to notify drivers and navigate to the next available parking area. No longer best does this save you frustration and lost productiveness, it also reduces your carbon footprint and improves automobile parking space air pleasant.

3. Optimize Facility usage and Create New Revenue Streams

Sensor facts gives significant insight into which rooms or areas have the most or least parking. This permits facility proprietors to determine where to add parking and, therefore, to reduce. At the equal time, misuse of precise parking areas and emergency driveways may be extra without problems detected and controlled. Clever parking solutions additionally permit companies to generate extra sales from parking at some point of off-hours.

1.4. Useful Tools and Technologies

For this project the team has decided to use multiple programming language. The programming Application will be stated below:

Programming Languages:

For Front-End:

1. HTML
2. CSS
3. Java-script

Back-End:

1. MySQL

Tools:

1. Enterprise Architecture (Designing System)
2. Visual Studio Code
3. Xampp

OS Support:

This application can be run on Web Browsers.

Development OS:

The Development OS are Window 10 and Linux (Arch Linux)

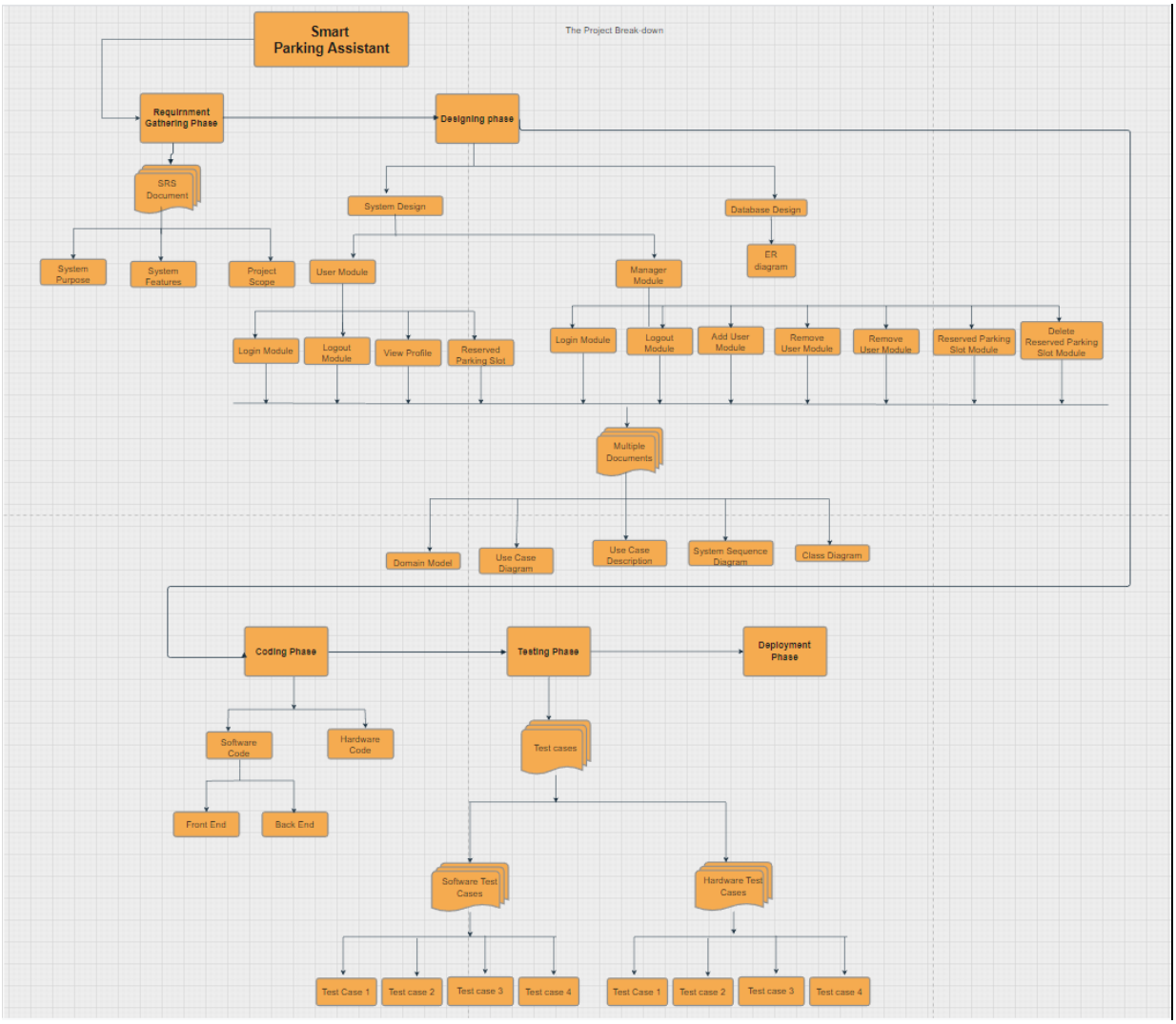
1.5. Project Work Breakdown

The venture can be divided into five components, those parts will tell us how the mission will start and give up, the components are said bellow.

1. Collecting necessities phase
2. Designing phase
3. Coding segment
4. Testing segment
5. Deployments segment

In every segment there will be a couple of steps with a purpose to applied inside the undertaking. In each phase the organization Member will be assigned with a particular undertaking. While every is completed and reviewed then next step could be beginning.

Figure 1: Work Breakdown



1.6. Project Timeline

FYP-Part 1

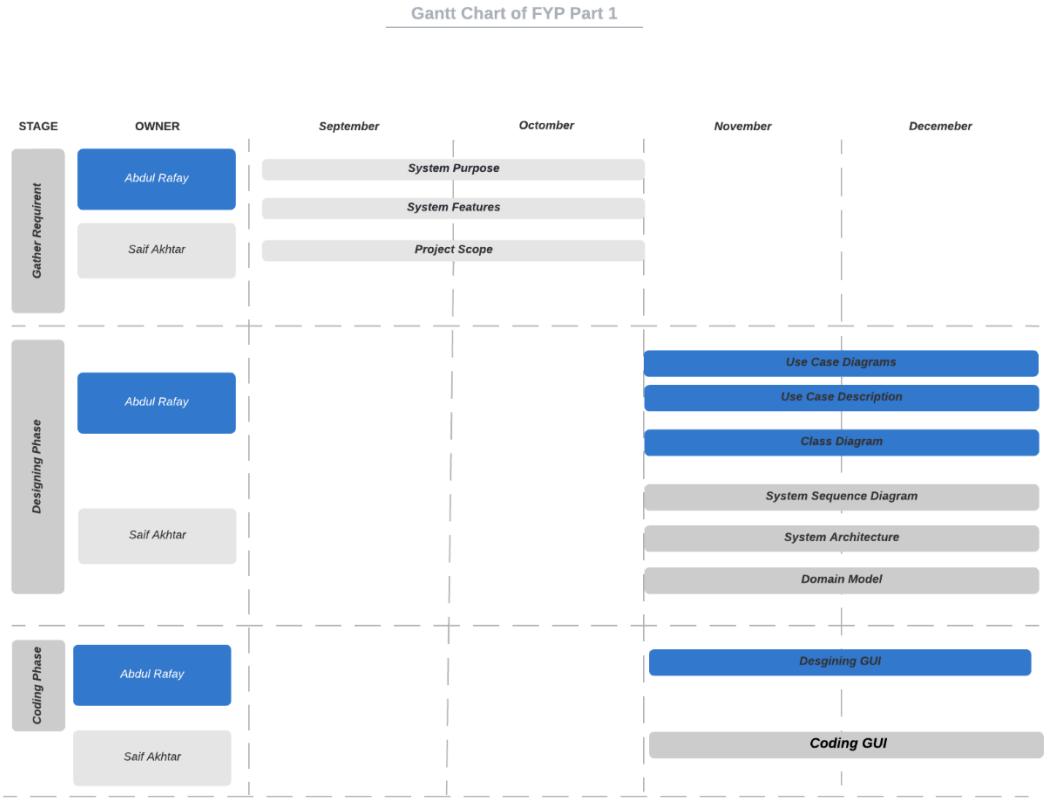


Figure 2: FYP part-1 Timeline

FYP-Part 2:

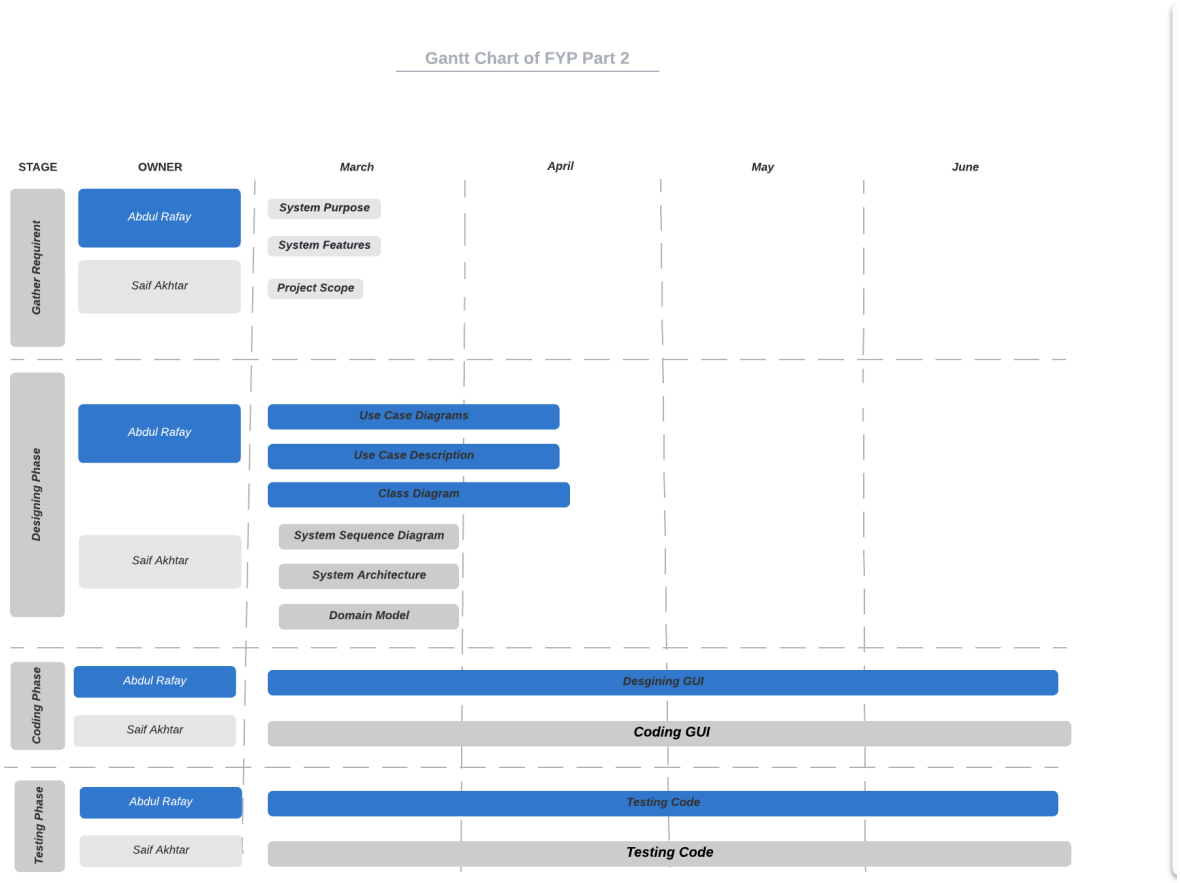


Figure 3: Timeline FYP-2

Chapter 2

Requirement Specification and Analysis

Description

The application will show the parking spaces using different sensors, the application will provide live time data environment and all of the data will be displayed in the live data in an android application. All of the features of the application are stated bellow.

2.1. Functional Requirements

Table 1: Function Requirements

S. No.	Functional Requirement	Type	Status
1	The manager shall be able to create an account for user	Core	Completed
2	The user shall be login in the system	Core	Completed
3	The user shall be able to view profile	Core	Completed
4	The consumer will be able to log off from the software.	Core	Completed
5	The Manager will be able to log into the system.	Necessary	Completed
6	The Manager shall able to view profile	Necessary	Completed
7	The manager will be able to log in to the machine.	Necessary	Completed
8.	The user and/or manager will be able to view the parking map	Core	Completed
9.	The user and/or manager will be able to reserve a parking slot	Core	Completed
10.	The Manager can reserve parking area for different people.	Core	Completed

2.2. Non-Functional Requirements

Table 2: Non-Functional Requirements

S. No.	Non Functional Requirements	Description	Type
1	Notification Alert	1. The system issues notifications to the manager about new account requests, errors, problems and/or reservation requests. 2. The system issues notifications to the user about approvals, errors, available spaces etc.	Interoperability
2	Google integrated maps	Locational display is integrated with google maps to provide location services, accurate to around 10 meters	Usability
3	Live Information	Parking space information is constantly displayed in live time and has a short refresh rate	Availability

2.3. System Use Case Modeling

The Usecase Diagram for User

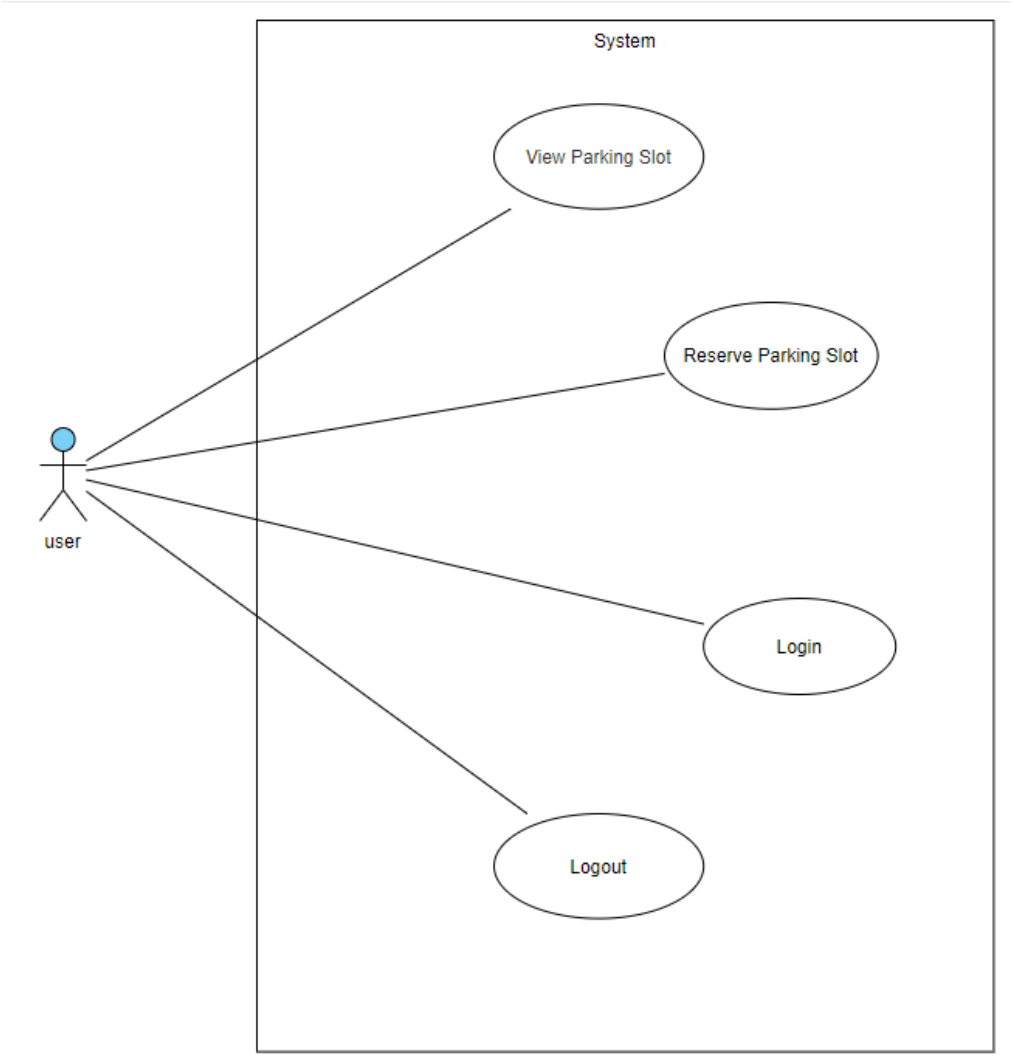


Figure 4: User Use-Case Diagram

The Use case diagram for Manager:

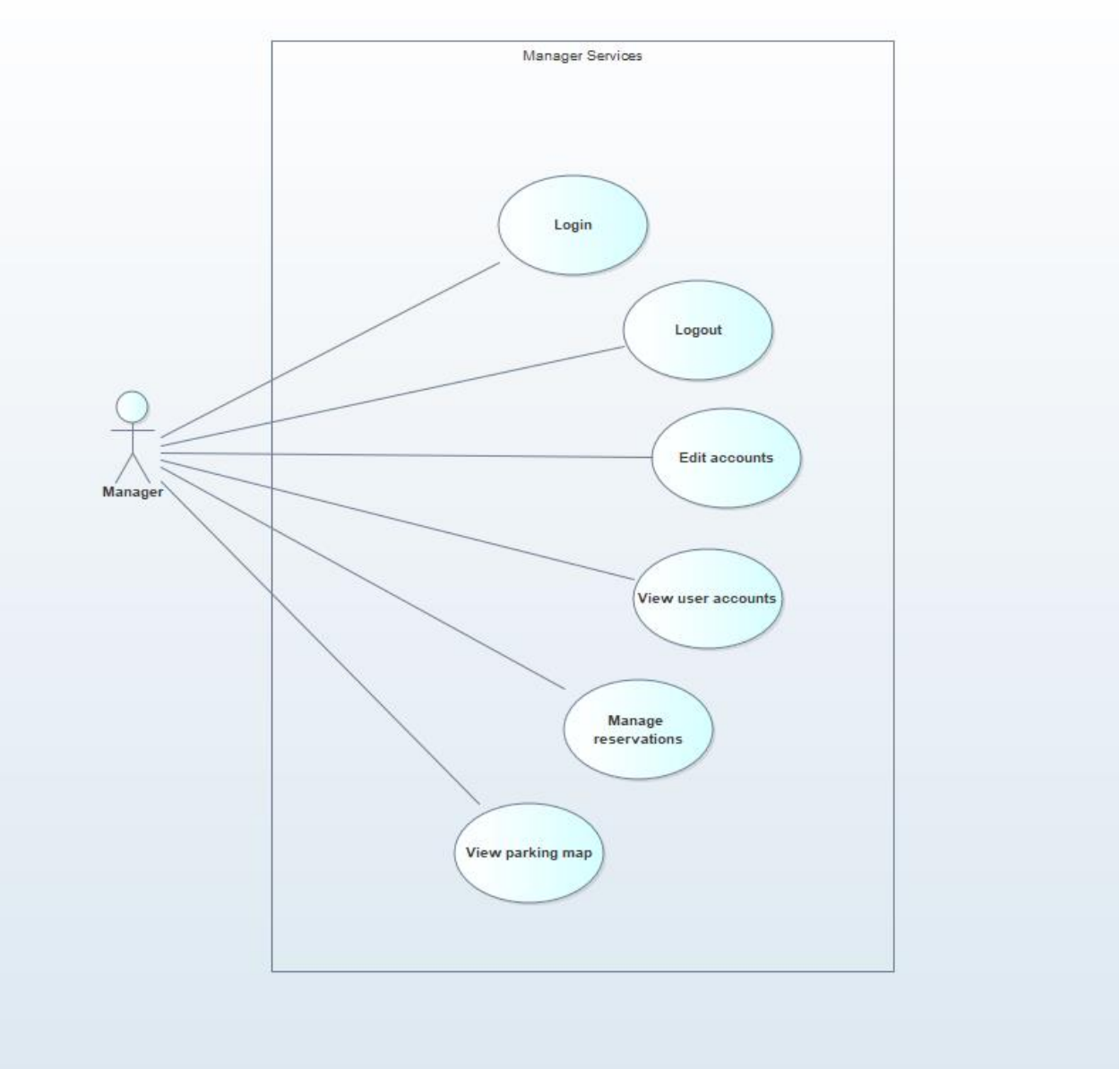


Figure 5: Manager Use-Case Diagram

Use Case Description:

1.1 Creating Account:

In this use case the manager shall be able to create an account by providing some information like email, name, password, once all of the information is provided then system will respond with a message that you are signed up for the application.

Use Case Description:

Table 3: Create Account Description

Use Case ID:	1.1		
Use Case Name:	Create Account		
Created By:	Mohammad Abdul Rafay Saif Akhter	**Last Updated By:**	Saif Akhter
Date Created:	05-December-2021	**Last Revision Date:**	30/7/2022
Actors:	Manager		
Description:	At any time, the manager may choose to cancel the account creation. At this point, the processing is discontinued, the manager account remains unchanged, and the manager is notified that the account management request has been canceled.		
Trigger:	"Create account" button is pressed		
Preconditions:	The manager must have an email address in-order to sign up for the application.		
Post conditions:	1. **Success** The manager enter the user correct data is stored in the user account. Confirmation is sent to the appropriate email address. 2. **Failure** The manager entered invalid data or chose to cancel the account creation request. In either case, no account will be created.		
Normal Flow:	**Actor**	**System**	
	1. The manager will click on add user button, Once Clicked then the manager will be redirected to the create account page. 3. The manager enters the required user information example Name, Email, Date of birth, Password, Gender.	2. The system will save the entered data provided by Manager. 4. The system will save all of the information and notify the user that account has been created.	
Alternative Flows:	1a. If any important column is missing or left blank. • System will present user with error message before proceeding		
Exceptions:	Account already exists		

1.2. Log in Section:

In this use case the user and Manager can log in into the system by providing a user-name and password, the system will validate the password and user-name if the user-name and password is correct then both user and Manager can log in the system if not then error message will be displayed

Use case Description:

Table 4: Login Description

Use Case ID:	1.2		
Use Case Name:	Account Login		
Created By:	Mohammad Abdul Rafay Saif Akhter	**Last Updated By:**	Saif Akhter
Date Created:	05-December-2021	**Last Revision Date:**	6/8/2022

Actors:	User, Manager	
Description:	The User entered data is stored in the user account. Confirmation is sent to the appropriate email address.	
Trigger:	"Login" button is pressed	
Preconditions:	The User must have an account in order to login into the system	
Post conditions:	**1. Success** The User is authenticated and the system displays all features available for the role the user is associated with as defined in his/her user account. **2. Failure** This can occur because the User repeatedly entered invalid sign in information. The User has been notified of the reason why he/she was not signed in. The User is not authenticated and remains in the Anonymous User role.	
Normal Flow:	**Actor**	**System**
	1. The User will click on login button. 3. Once clicked the user or the manager will enter the email and password for login.	2. The system validates the entered User or the manager email and password. 4. If the email and password is correct then system will notify the user or the manager that you are logged into the system.
Alternative Flows:	**2a. New User:** If the User does not have an account, the System will notify the user that account does not exist and will redirect to the sign-up form See the **Create Account** use case. Once the account is created, the User is considered signed in. **2b. Re-Entering Password:**	

	System will keep asking for password till user gets it right **2c. User Fails Authentication** If the User entered an invalid user-name and/or password, the following occurs: • The system describes the reasons why the User failed authentication. • The system prompts the User to re-enter the valid information.
Exceptions:	Already signed into account

1.3. Manage Profile:

In this use case the Manager and registered user can modified and view their profile.

View Profile Use Case Description:

Use case Description:

Table 5: View Profile Description

Use Case ID:	1.3		
Use Case Name:	View Profile		
Created By:	Mohammad Abdul Rafay Saif Akhter	**Last Updated By:**	Saif Akhter
Date Created:	05-December-2021	**Last Revision Date:**	6/8/2022

Actors:	Registered-User, Manager	
Description:	The Manager or the User can View their profile	
Trigger:	"View profile" button is clicked	
Preconditions:	The User or the manager must be signed in before the User can edit or deactivate his/her account. See the Sign in use case	
Post conditions:	**Title**	**Description**
	Success	The User entered data is stored in the user account.
	Failure	The User entered invalid data or chose to cancel the account management request. In either case, there is no change to the user account.
Normal Flow:	**Actor**	**System**
	1. The user or the manager will click the view profile button from dashboard to view the information.	2. The system displays the *User Account* information currently stored for the User or the manager.
Alternative Flows:	**Title** User Cancels Request	**Description** At any time, the user or manager can click on the back button to cancel the operation of view profile
Exceptions:	None	

1.4. Update Profile use case Description:

In this user can or the manager can update the user profile.

Use case Description:

Table 6: Update Profile Description

Use Case ID:	1.4		
Use Case Name:	Update Profile		
Created By:	Mohammad Abdul Rafay Saif Akhter	**Last Updated By:**	Saif Akhter
Date Created:	05-December-2021	**Last Revision Date:**	6/8/2022

Actors:	Registered-User, Manager	
Description:	The Manage Account use case allows the User to update the User Account Information maintained in the User's account.	
Trigger:	"View Profile" button is clicked	
Preconditions:	**User or Manager Log in:** **T**he User or the manager must be signed in before the User can edit or deactivate his/her account. See the Sign in use case	
Post conditions:	**Title**	**Description**
	Success	The User entered data is stored in the user account.
	Failure	The User entered invalid data or chose to cancel the account management request. In either case, there is no change to the user account.
Normal Flow:	**Actor**	**System**
	1. The user or the manager will click the view profile button. 3. The User or the manager will click on the edit profile button and a request will be sent to the system for editing the profile. 5. The user or the manager will update information according to the need.	2. The system displays the *User Account* information currently stored for the User or the manager. 4. The system accept the request and will load the edit the profile page. 6. The system notifies the User or the manager that the account has been updated.

Alternative Flows:	**Title** User Cancels Request	**Description** At any time, the User may choose to cancel the account update/deactivation. At which point, the processing is discontinued, the user account remains unchanged, and the user is notified that the account management request has been canceled.
	User Enters Invalid User Account Information	If during Modify Account, the system determines that the User entered invalid *User Account* information, the following occurs: 1. The system describes which entered data was invalid and presents the User with suggestions for entering valid data. 2. The system prompts the User to re-enter the invalid information. 3. The User re-enters the information and the system re-validates it. 4. If valid information is entered, the User Account Information is stored. 5. If invalid information is entered, the Entered Information is Invalid alternative flow is executed again. This continues until the User enters valid information, or chooses Cancel (see the User Cancels Account Management Request alternative flow). Invalid *User Account* information: • Missing information items • User name already exists in the system • *User Account* information entered does not comply to its definition in the glossary • Not well-formed e-mail address •
Exceptions:	Manager decides to cancel the operation	

1.5. Log-out of account:

In this use case the Manager and registered user can Log-out from there account by clicking the log out button when the button is clicked the system ask for confirmation and if the confirmation is Yes then system will save the files of user or the Manager and logout them out from the system.

Use case Description:

Table 7: logout Description

Use Case ID:	1.5		
Use Case Name:	Account logout		
Created By:	Mohammad Abdul Rafay Saif Akhter	**Last Updated By:**	Saif Akhter
Date Created:	05-December-2021	**Last Revision Date:**	6/8/2022
Actors:	Registered-User, Manager		
Description:	The registered user and The Manager can log-out from the system when they are done using the application.		
Trigger:	By clicking the View profile		
Preconditions:	**User or Manager Log in:** **T**he User or the Manager must be signed in before the User can edit or deactivate his/her account. See the Sign-in use case		
Post conditions:	**Title**	**Description**	
	Success	By click the Log-out the button, there shall be log-out from the system.	
	Failed to Log-out	In few case the user or the Manager might not able to log-out from the system.	
Normal Flow:	**Actor**	**System**	
	1. The user or the Manager will click the log-out button. 3. When User or the Manager shall click “OK” for log-out the they will be log-out from the system.	2. The system will generate a request for log-out from the application 4. The system shall save a files and detail of the user or the Manager.	
Alternative Flows:	**Title** User Cancels Request	**Description** At any time, the User may choose to cancel the log-out request, when the system will ask for confirmation.	
Exceptions:	User decides to stay logged-in		

1.6. Parking Reservation:

The user and/or manager will be able to select and reserve a slot from a number of parking spaces available.

Use case Description:

Table 8: Reservation Description

Use Case ID:	1.6		
Use Case Name:	Manage Reservations		
Created By:	Mohammad Abdul Rafay Saif Akhter	**Last Updated By:**	Saif Akhter
Date Created:	05-December-2021	**Last Revision Date:**	20/8/2022
Actors:	Manager, User		
Description:	The user is able to reserve parking slot at first availability and the manager is able to cancel the reservation in-case of any issues.		
Trigger:	Click “reserve” button in Dashboard page		
Preconditions:	**Manager Log in:** The manager must be logged into the system		
Post conditions:	**Title**	**Description**	
	Success	Selected parking is reserved and the timer has started	
	Failure	More than one user may reserve the same spot at the same time leading to both being cancelled	
Normal Flow:	**Actor**	**System**	
	1. User clicks “reserve” button 3. The manager is shown the list with times and importance and has options to reject.	2. System will redirect to reservations’ list 4. System will check if selected slot is available and inform the user of his options or vacancy	
Alternative Flows:	**Title** Manager Cancels Request	**Description** At any time, the manager may choose to cancel the request, when the system will ask for confirmation.	
Exceptions:	None		

1.7 View Parking Map:

In this use case, the manager is able to check and manage parking reservations made by customers. Users can find empty parking spots in their desired location and reserve them beforehand. A number of conditions will follow in this process.

Use case Description:

Table 9: Parking Map Description

Use Case ID:	1.7		
Use Case Name:	View Parking Map		
Created By:	Mohammad Abdul Rafay Saif Akhtar	**Last Updated By:**	Saif Akhter
Date Created:	05-December-2021	**Last Revision Date:**	30/7/2022
Actors:	Manager		
Description:	The manager can view the sensor grid of the selected parking spaces in case there is a need to customize the layout in sync with the backend code		
Trigger:	Error in the parking space grid		
Preconditions:	**Manager Log in:** The manager and user must be logged into the system		
Post conditions:	**Title**	**Description**	
	Success	Parking grid is visible and editable	
	Failure	Sensors are not showing or page is unresponsive	
Normal Flow:	**Actor**	**System**	
	1. Manager clicks the “view parking spaces” button.	2. System will redirect to sensor grid	
Alternative Flows:	none		
Exceptions:	Manager or user is unable to log-in, redirected to help page		

2.4. System Sequence diagrams

The system Sequence diagram for Use case 1.1 Create an Account:

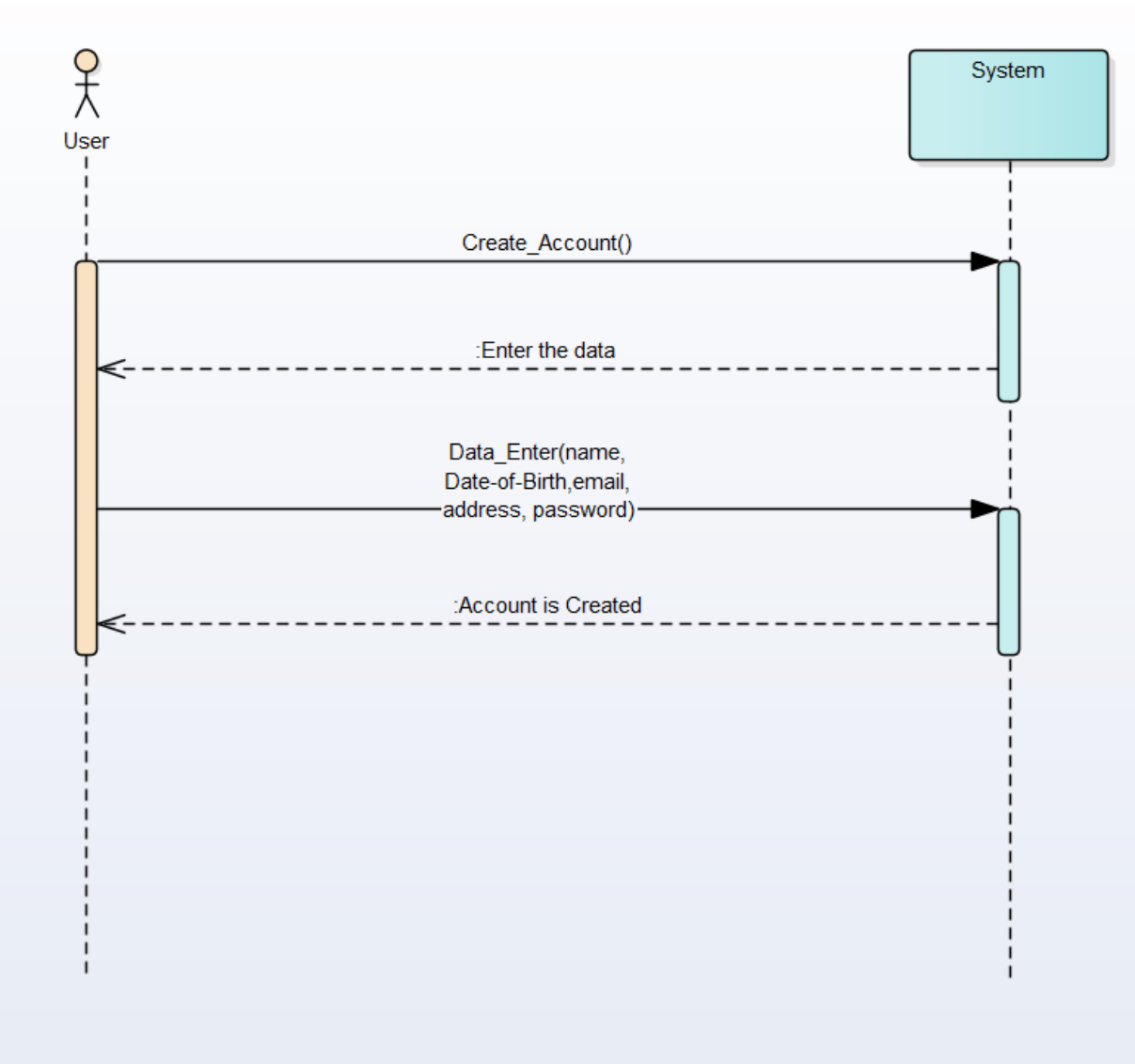


Figure 6: Create Account SSD

The System Sequence Diagram for Use Case 1.2 Login System:

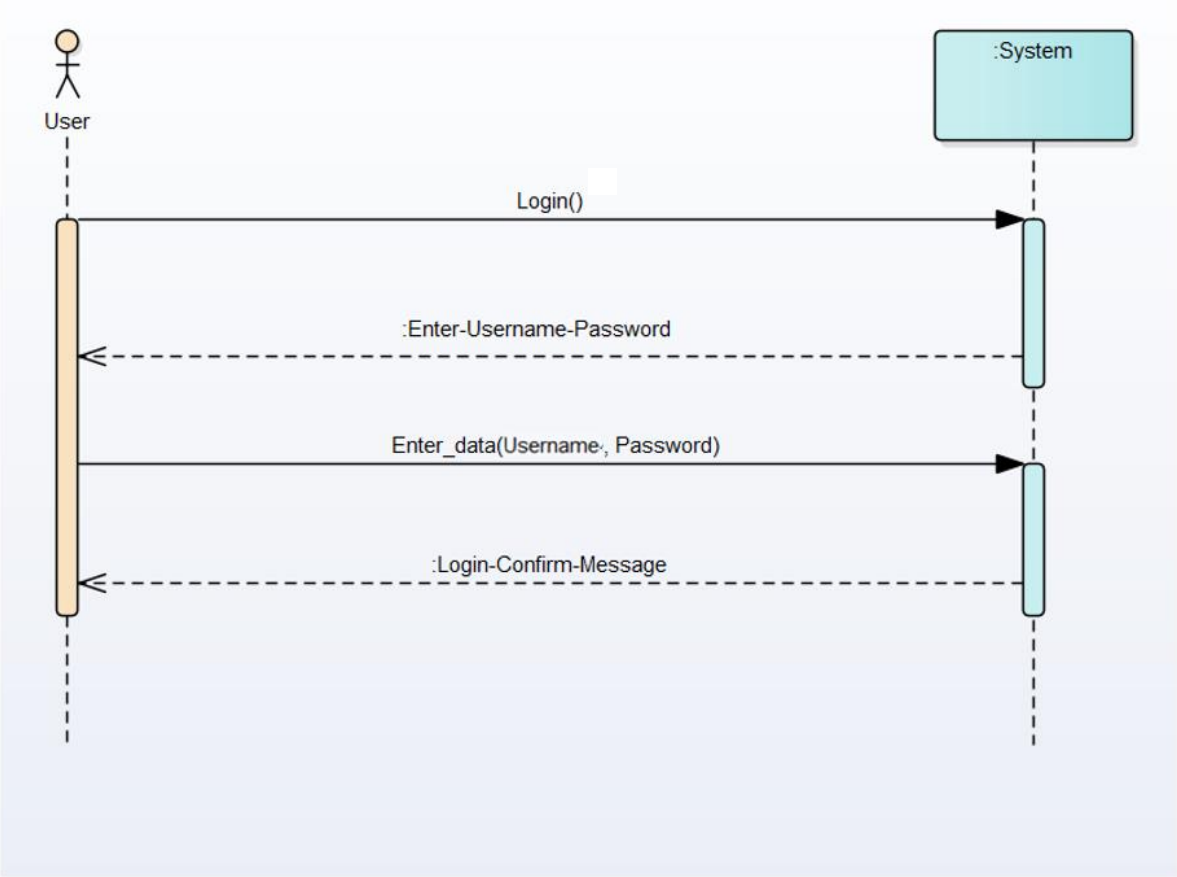


Figure 7: Login SSD

The System Sequence Diagram for Use Case 1.3 View Profile:

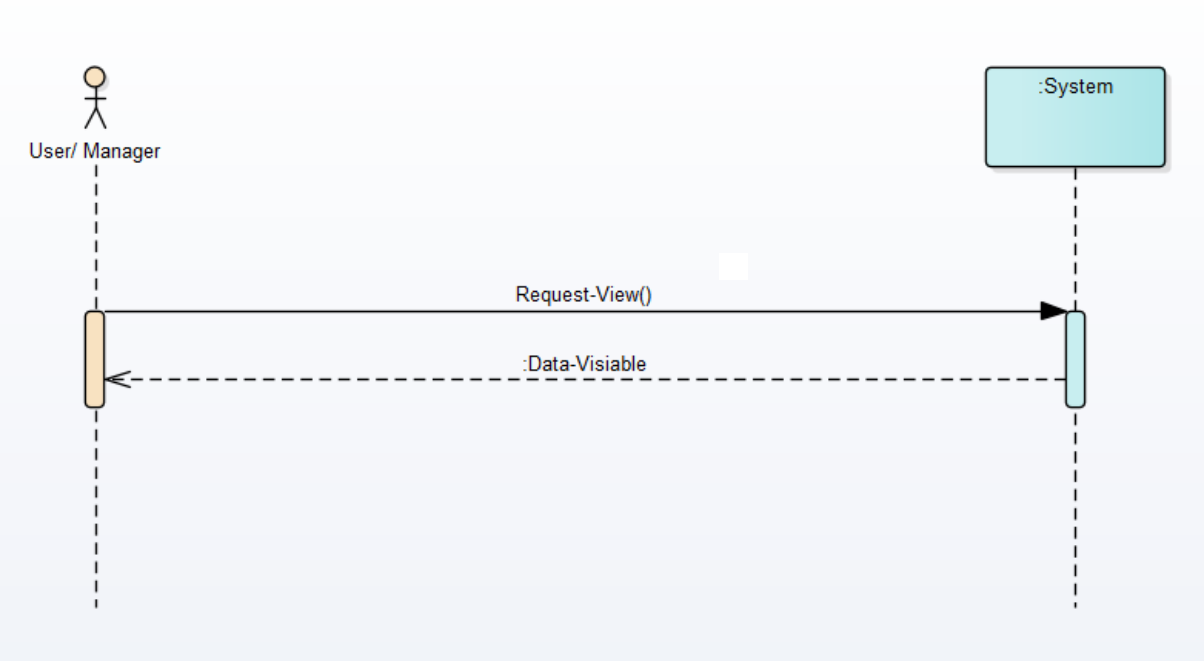


Figure 8: View Profile SSD

The System Sequence Diagram for Use Case 1.4 for Update Profile:

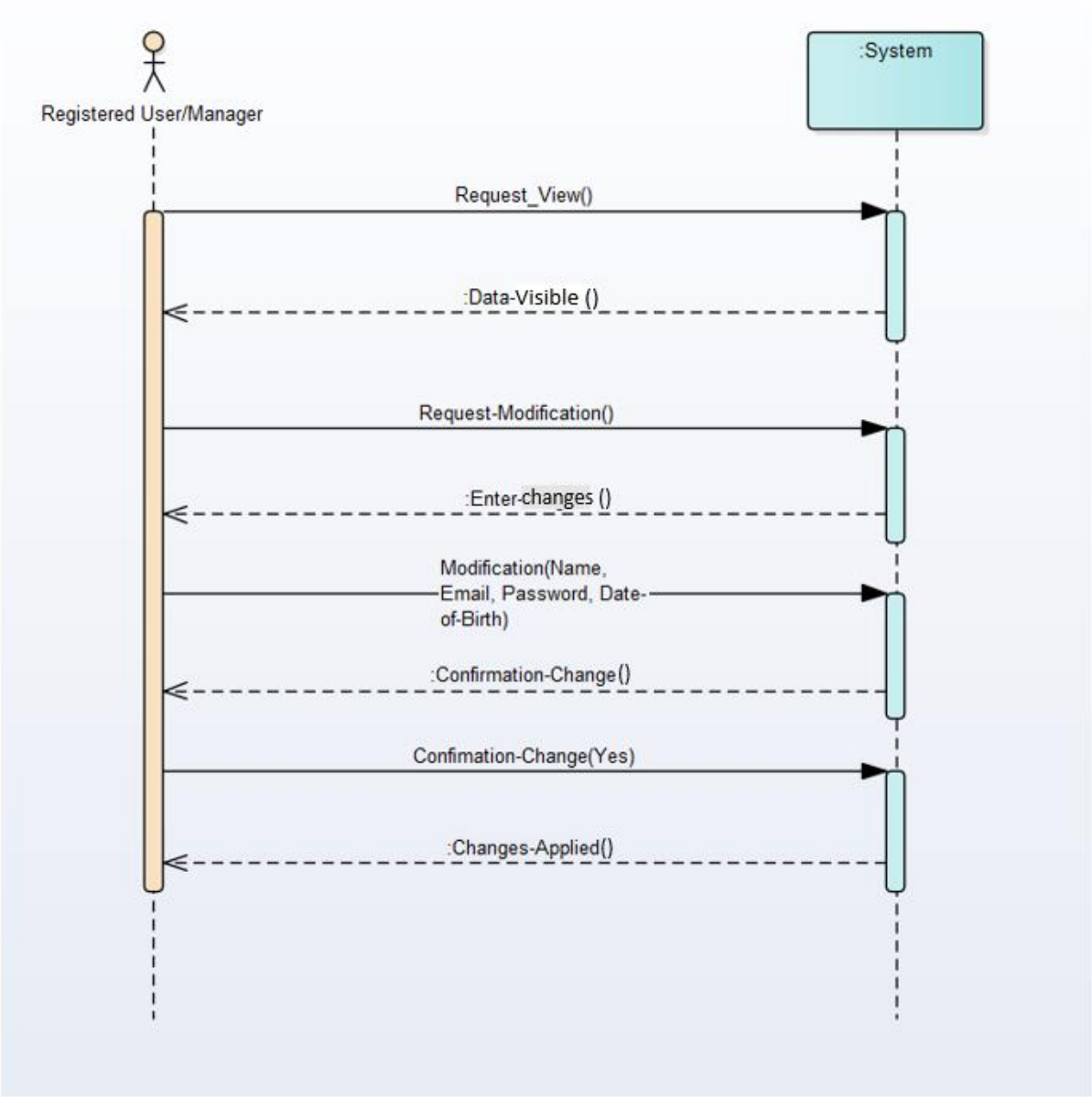


Figure 9: Update Profile SSD

The System Sequence Diagram for Use Case 1.5 – logging out:

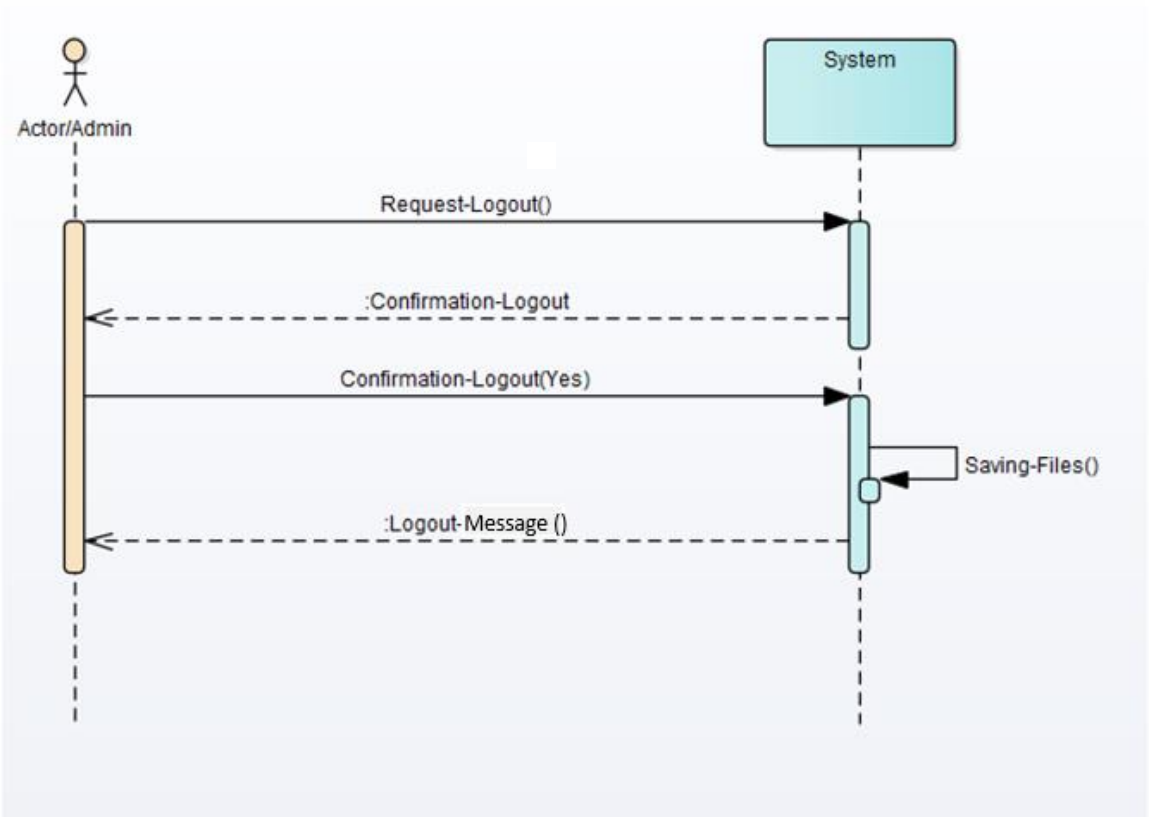


Figure 10: Logout SSD

Domain Model:

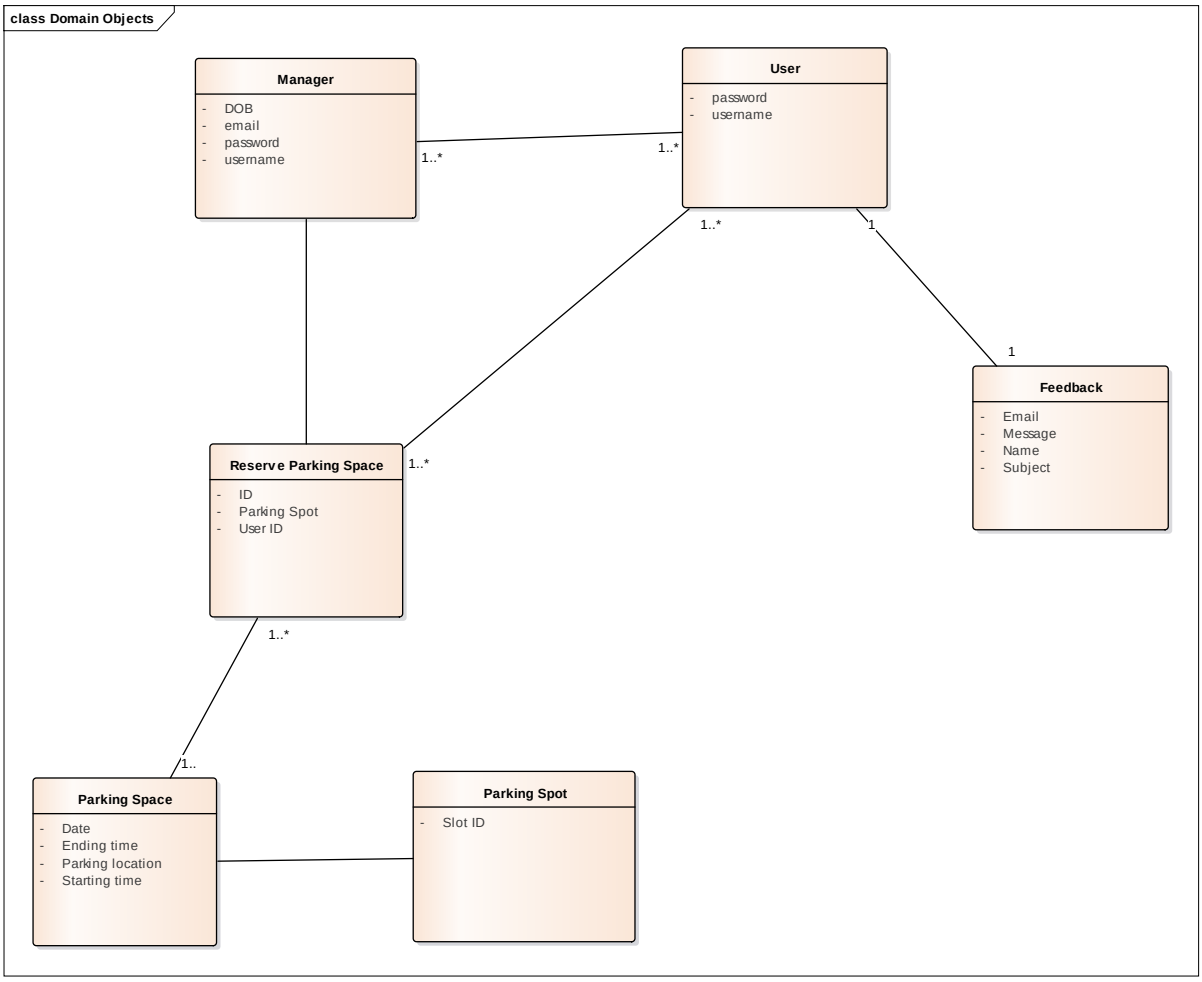


Figure 11: Domain Model

Chapter 3

System Design

3.1. Software Architecture

In our application the software architecture is very simple the user or the manager will interact with the GUI of the application. The working of the application is very easy, the sensors will detect the parking space and then once detected it will have sent the signal to a motherboard where the signal is transmitted to a server and from that server the signal will be displayed to the website or the android Application.

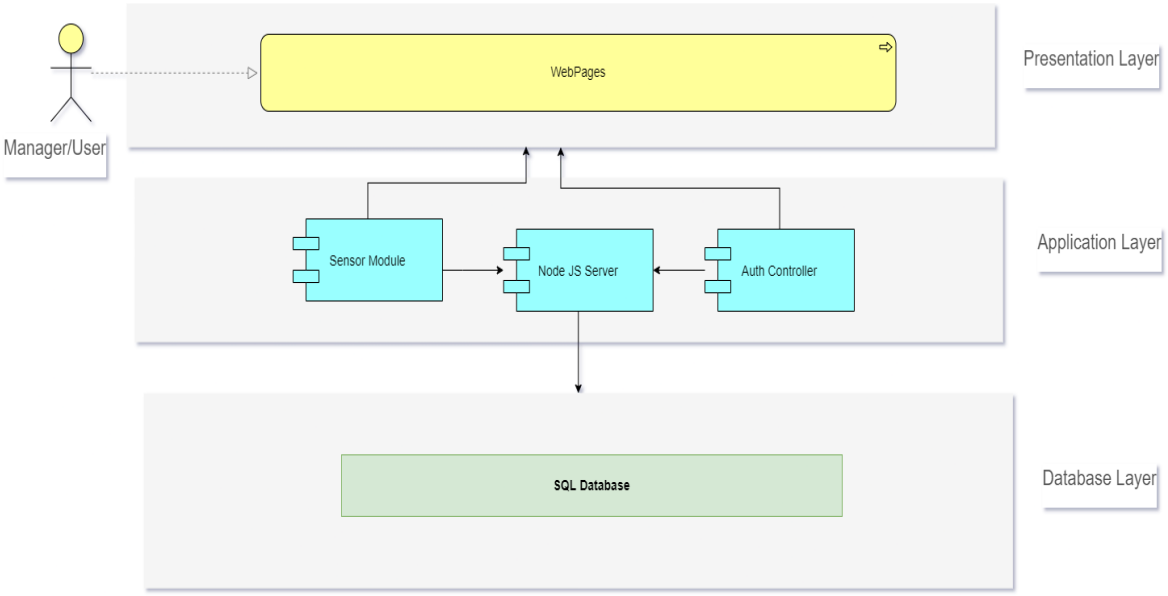


Figure 12: Architecture Model

3.2. Class Diagram

Class Diagram represent the structure and the behavior where our application, you can see all of the classes that will be in our application.

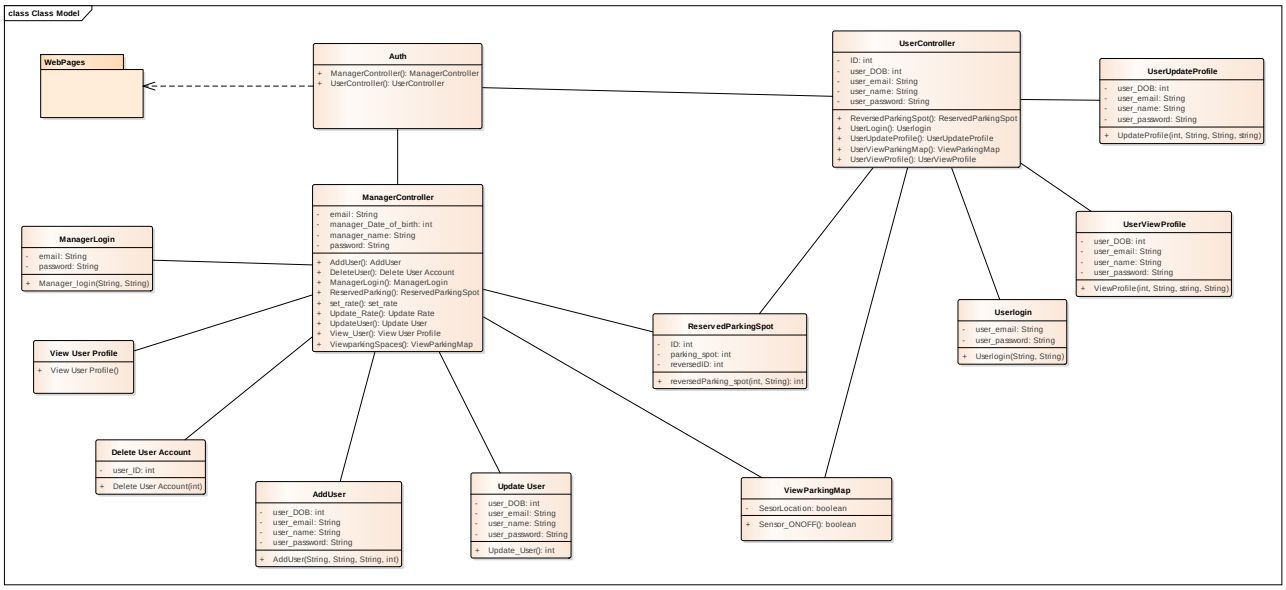


Figure 13: Class Diagram

3.3. Sequence Diagram

Sequence diagrams, when used in conjunction with class diagrams; provide an extremely effective communication mechanism. UML sequence diagrams as shown in Fig. 3.3 are used to show how objects interact in a given situation.

3.3.1. Create Account:

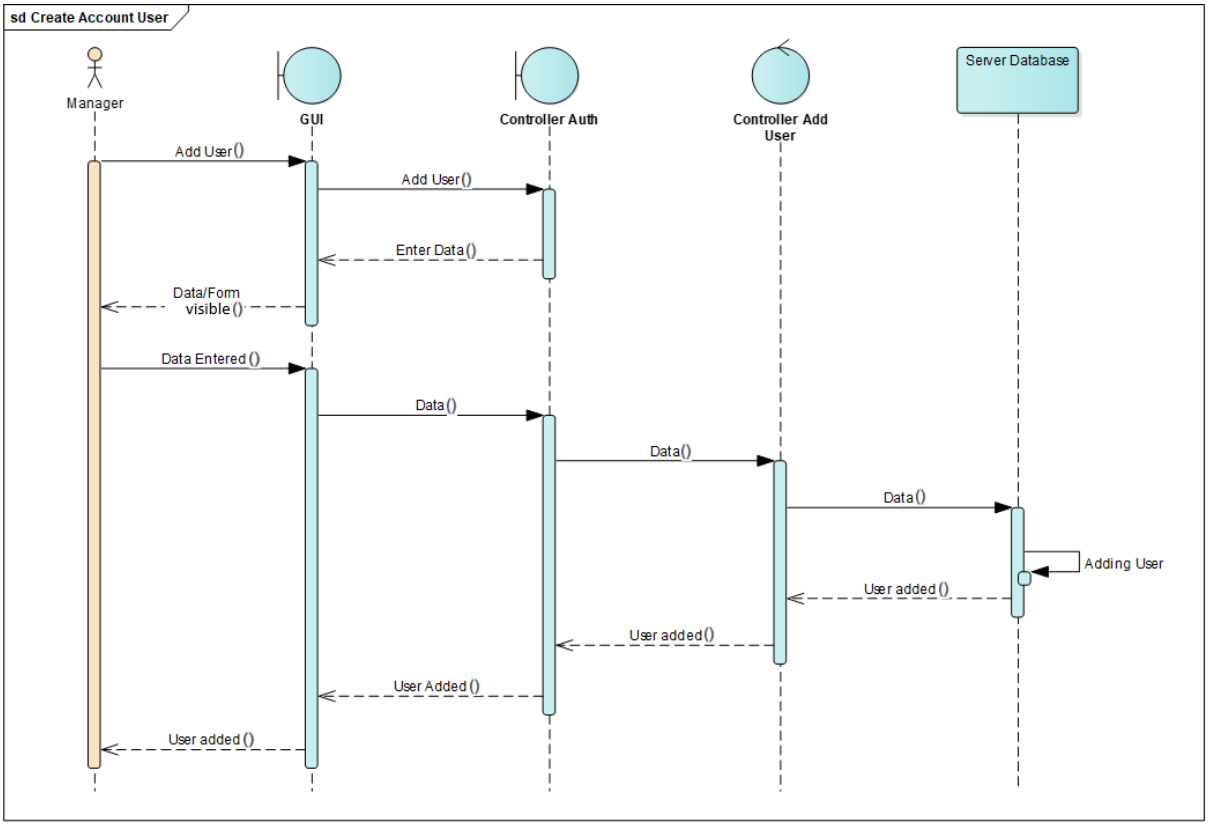


Figure 14: SD Create Account

3.3.2. Login

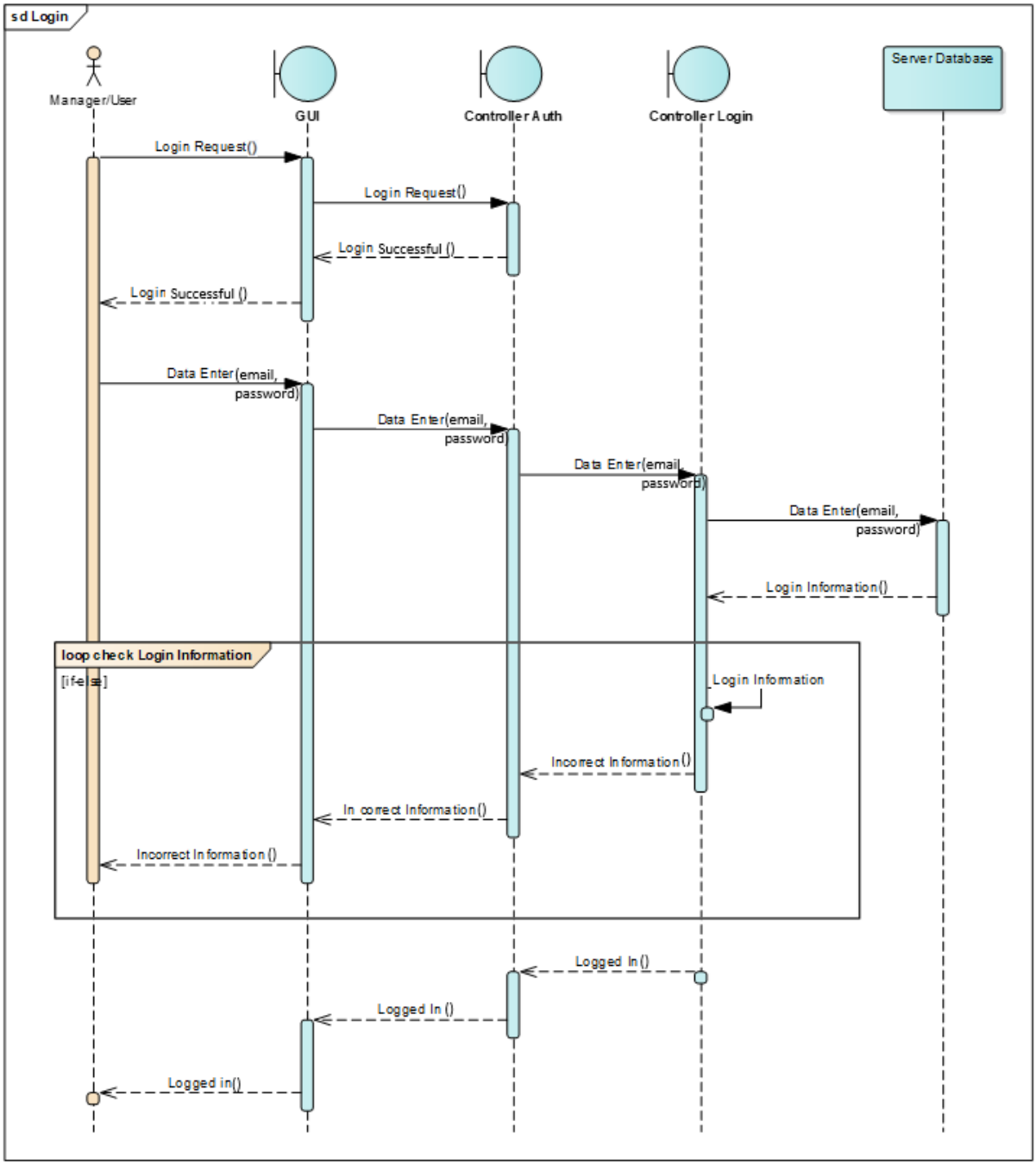


Figure 15: Login System Diagram

3.3.3. Logout:

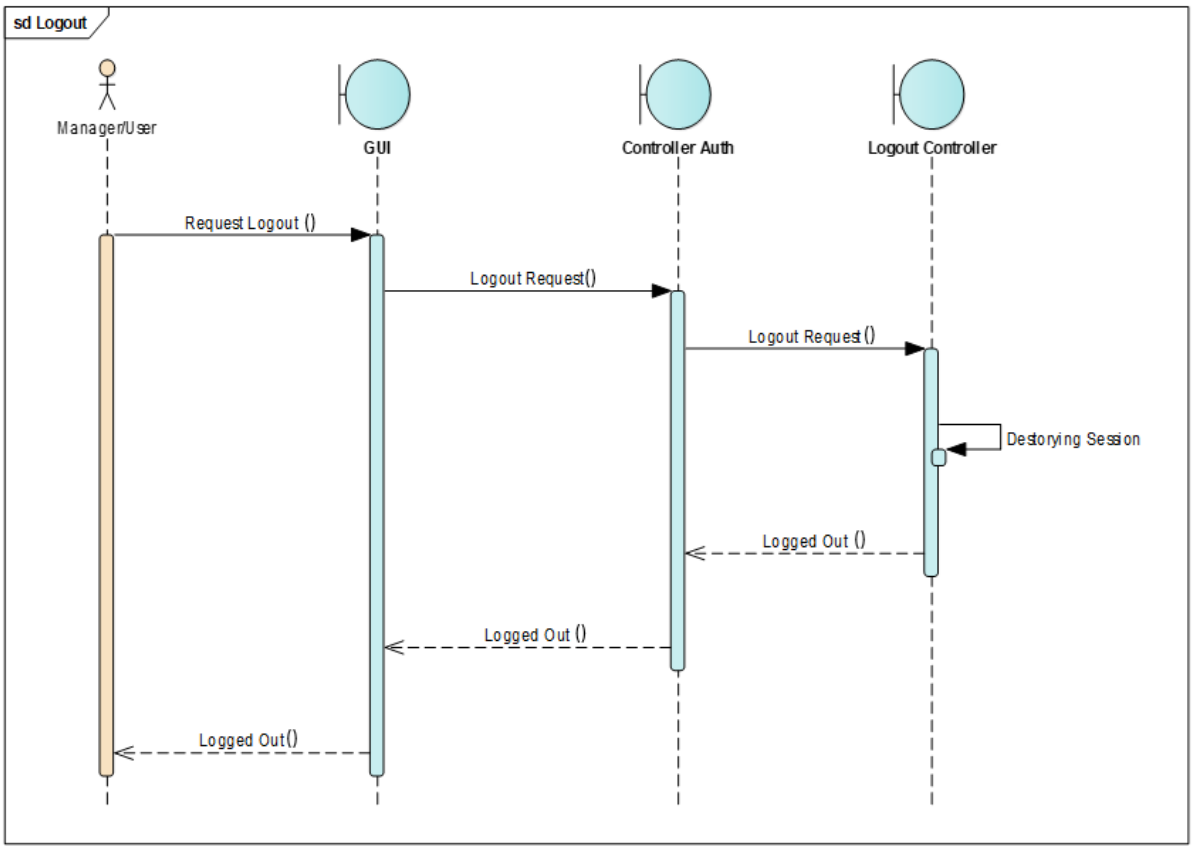


Figure 16: Logout System Diagram

3.3.4. Reservations Parking Spot:

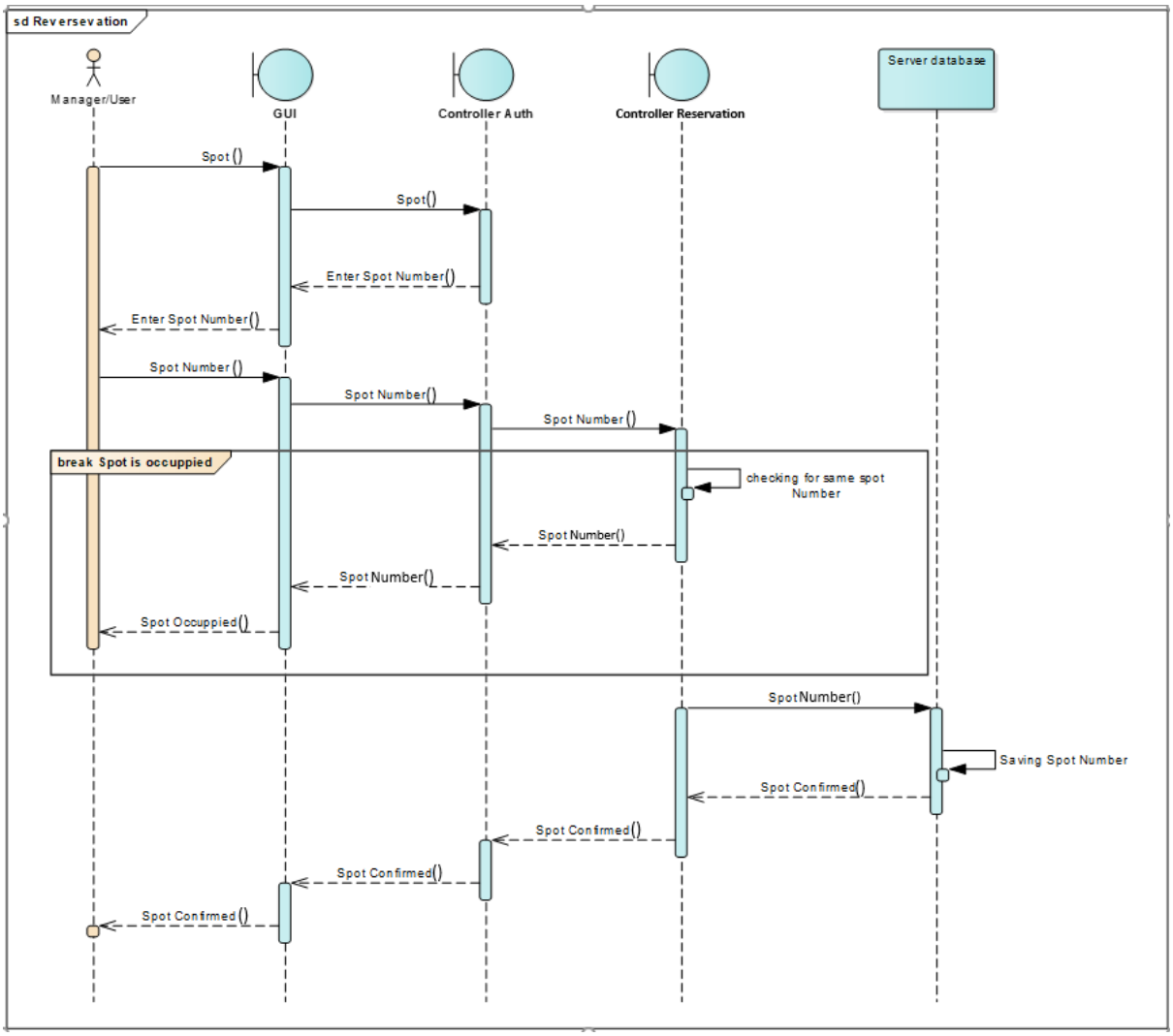


Figure 17: Reservation System Diagram

3.3.5. Update Profile:

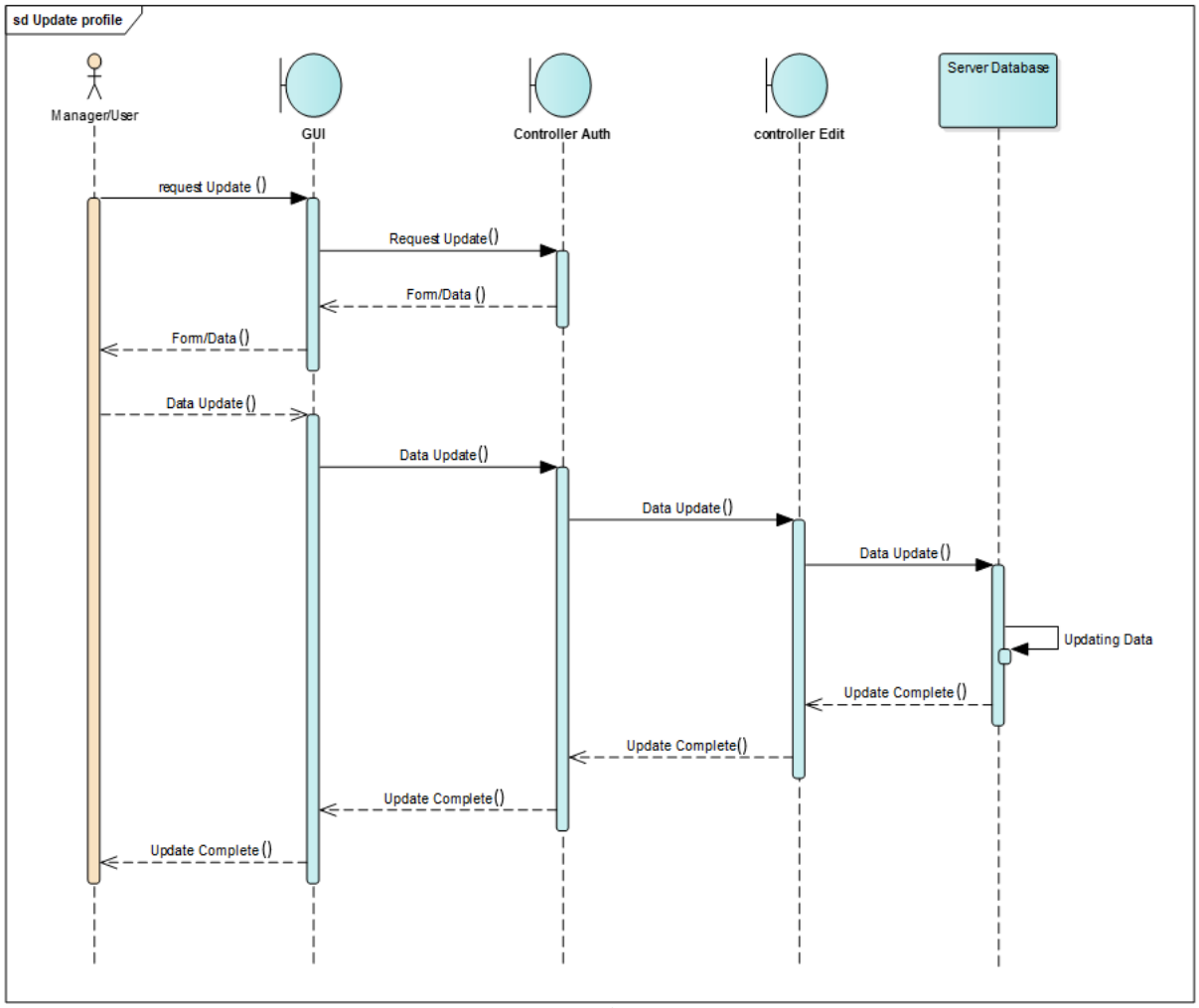


Figure 18: Update Profile System Diagram

3.3.6. View Profile:

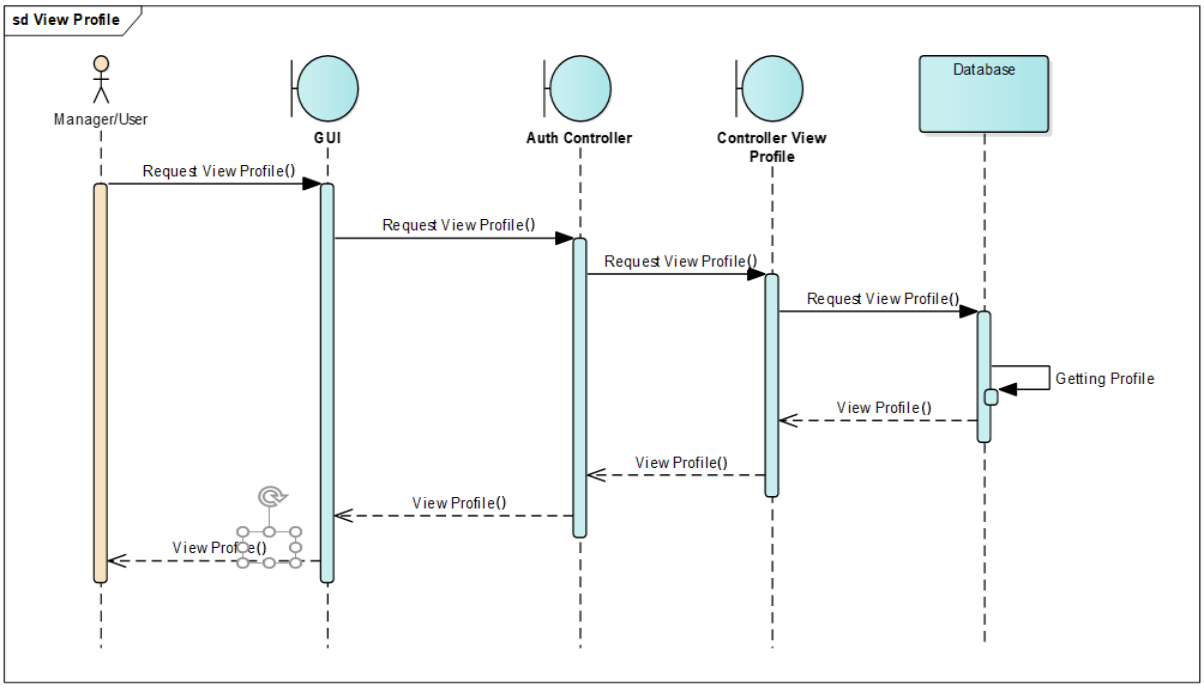


Figure 19: View Profile System Diagram

3.4. Entity Relationship Diagram

Entity Relationship Diagram is a graphically design of the data base, it represents the core feature and the structure of the database. The figure bellow represents the ERD of the application.

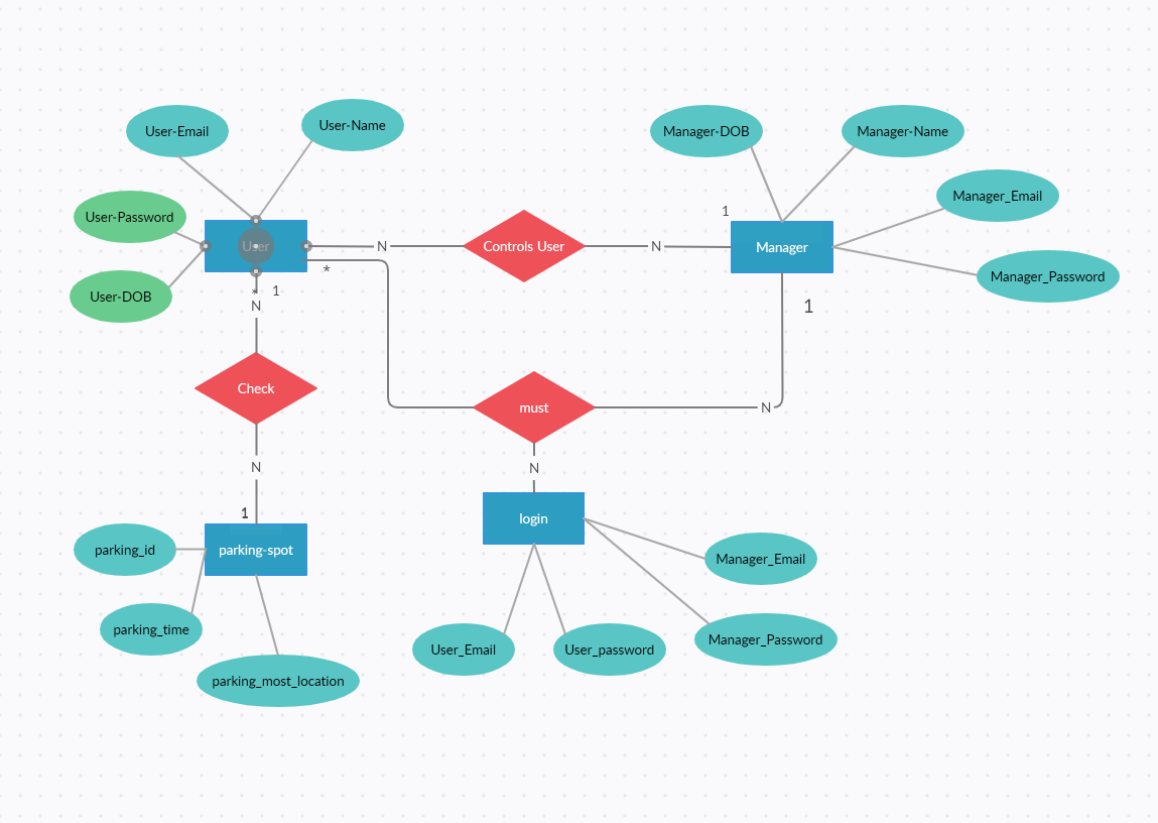


Figure 20: ER Diagram

3.5. Database Schema

In a system a database is very important, a database store all of the important data for the application. The application needs the data in order to run smoothly. The tables for the database is explained in the figure bellow.

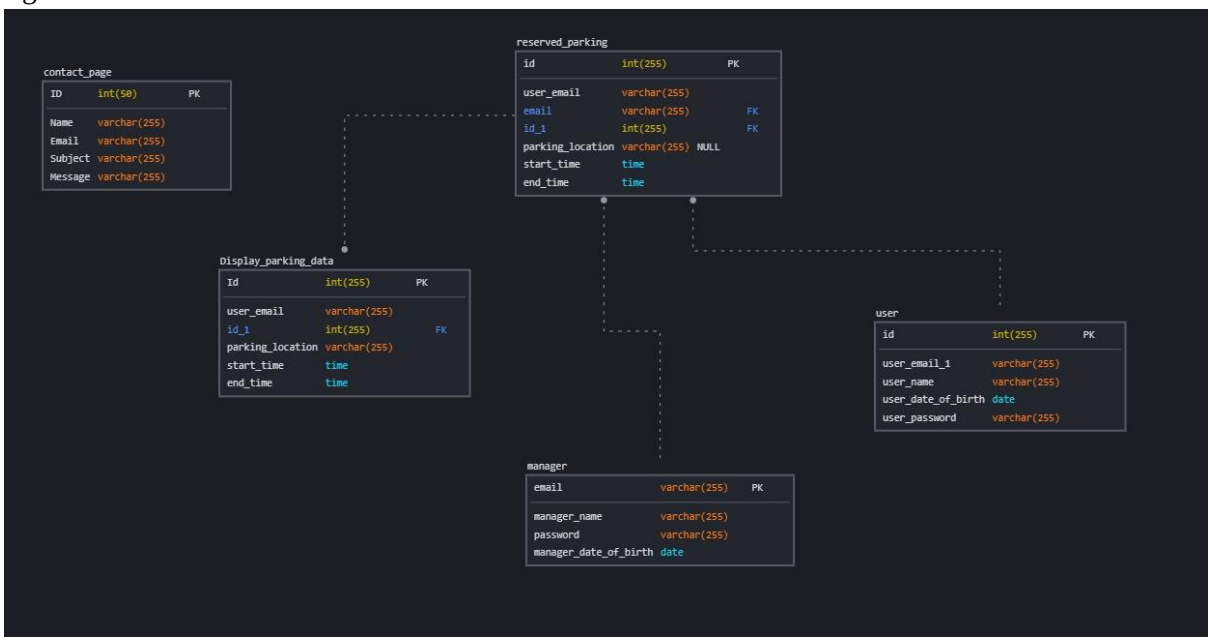


Figure 21: Database Schema

3.6. User Interface Design

We manger website is one most important components of the system so we make sure that the website is easy to use and understand.

Website GUI for Manager:

1. Homepage:

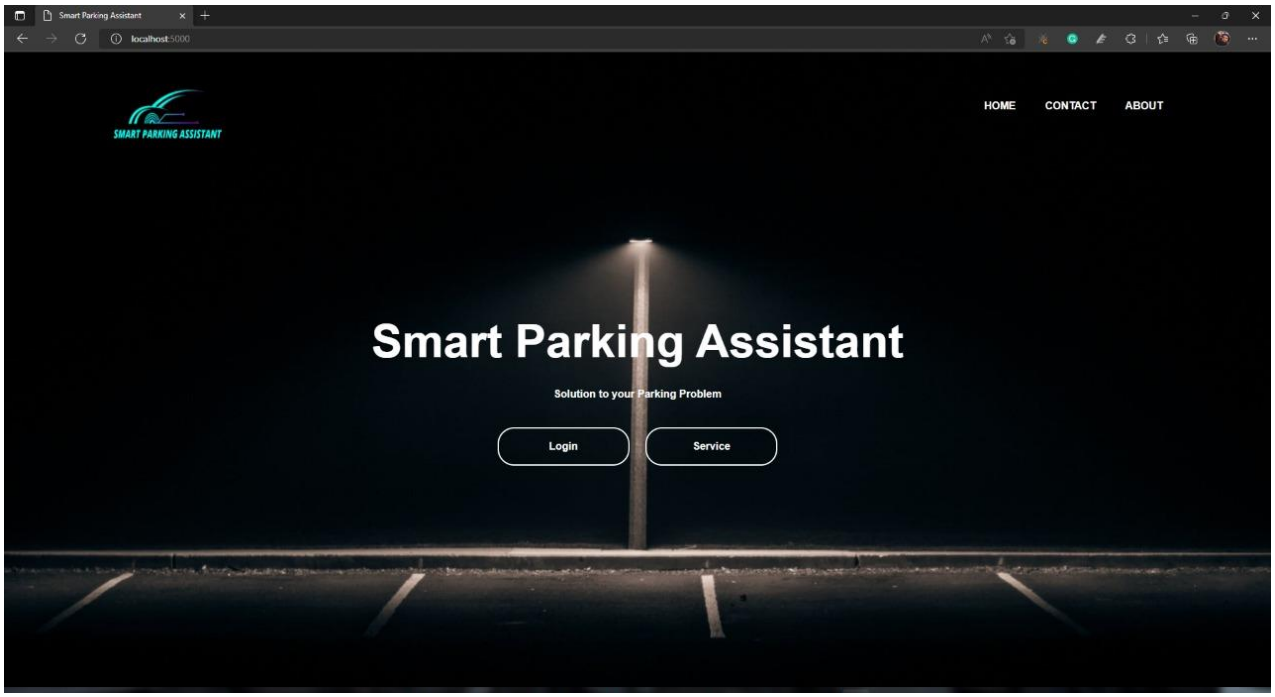


Figure 22: Home Page

2. Dashboard

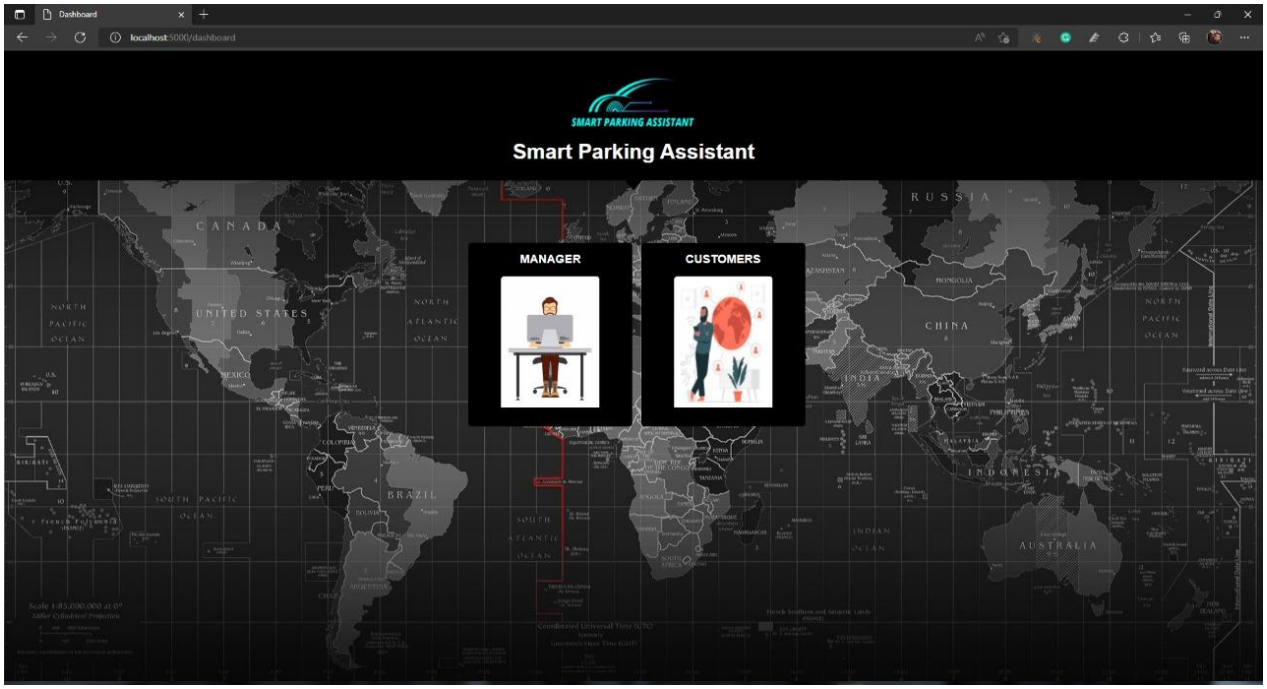


Figure 23: Dashboard Page

3. Login Page:

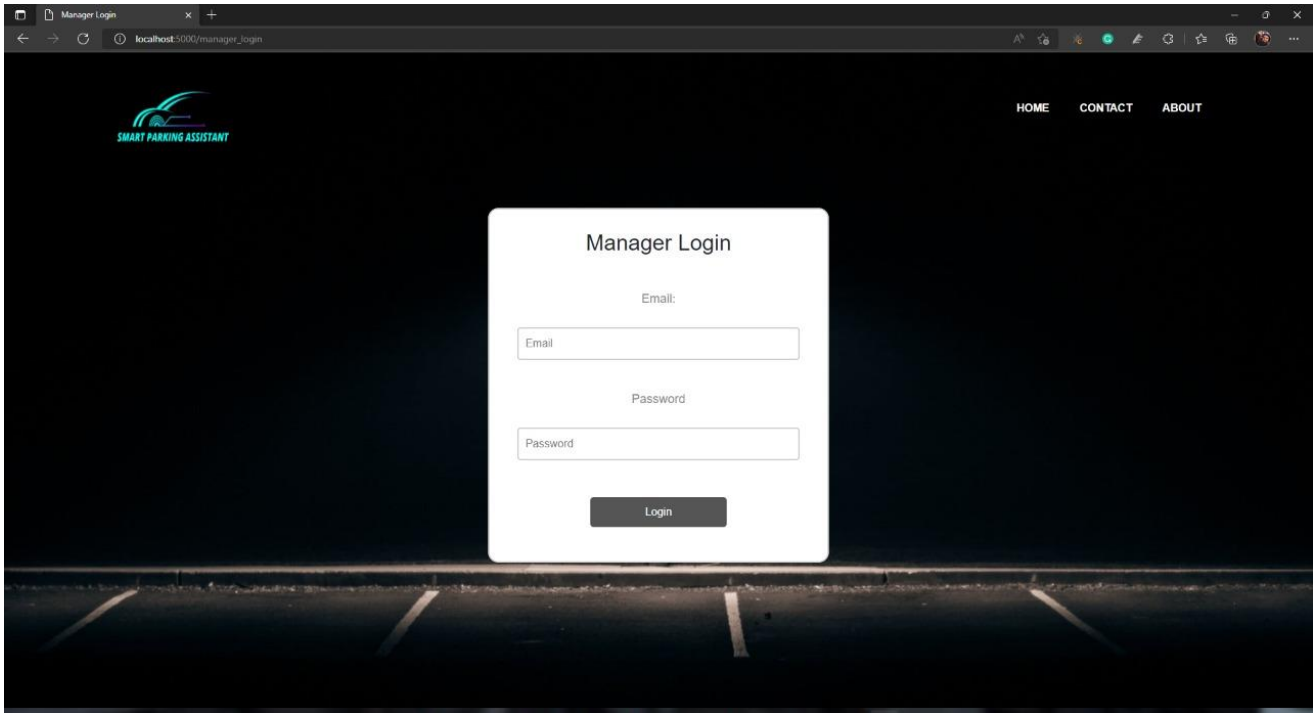


Figure 24: Manager Login

4. Add user Page:

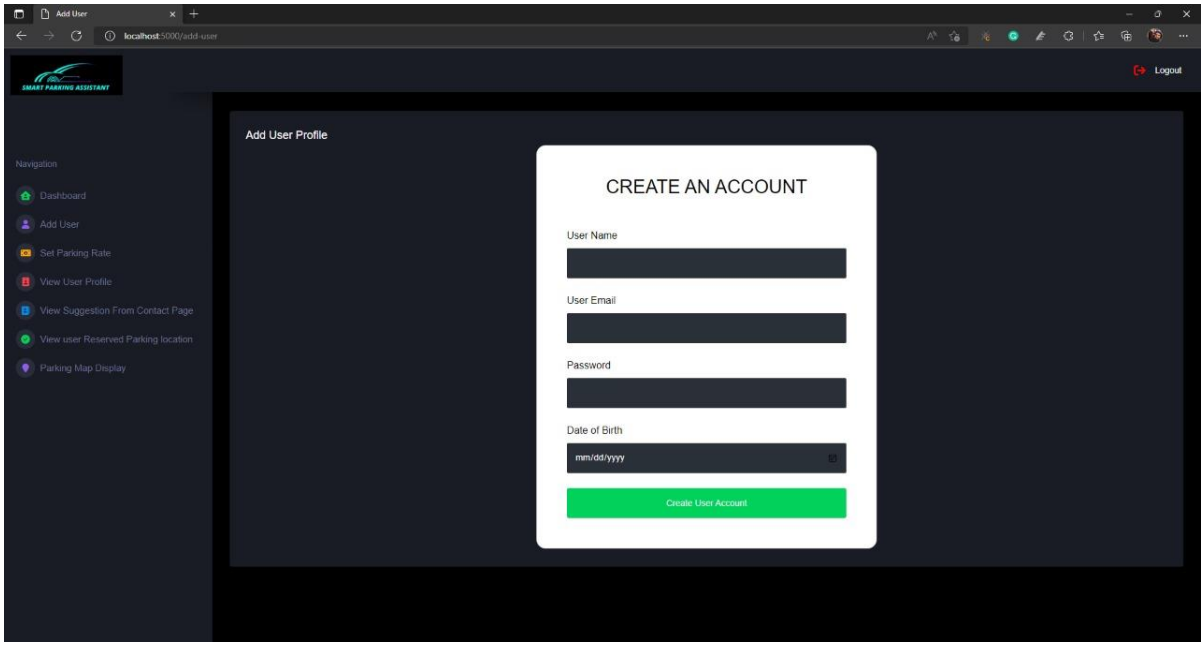


Figure 25: Add User Page

5. View user profile:

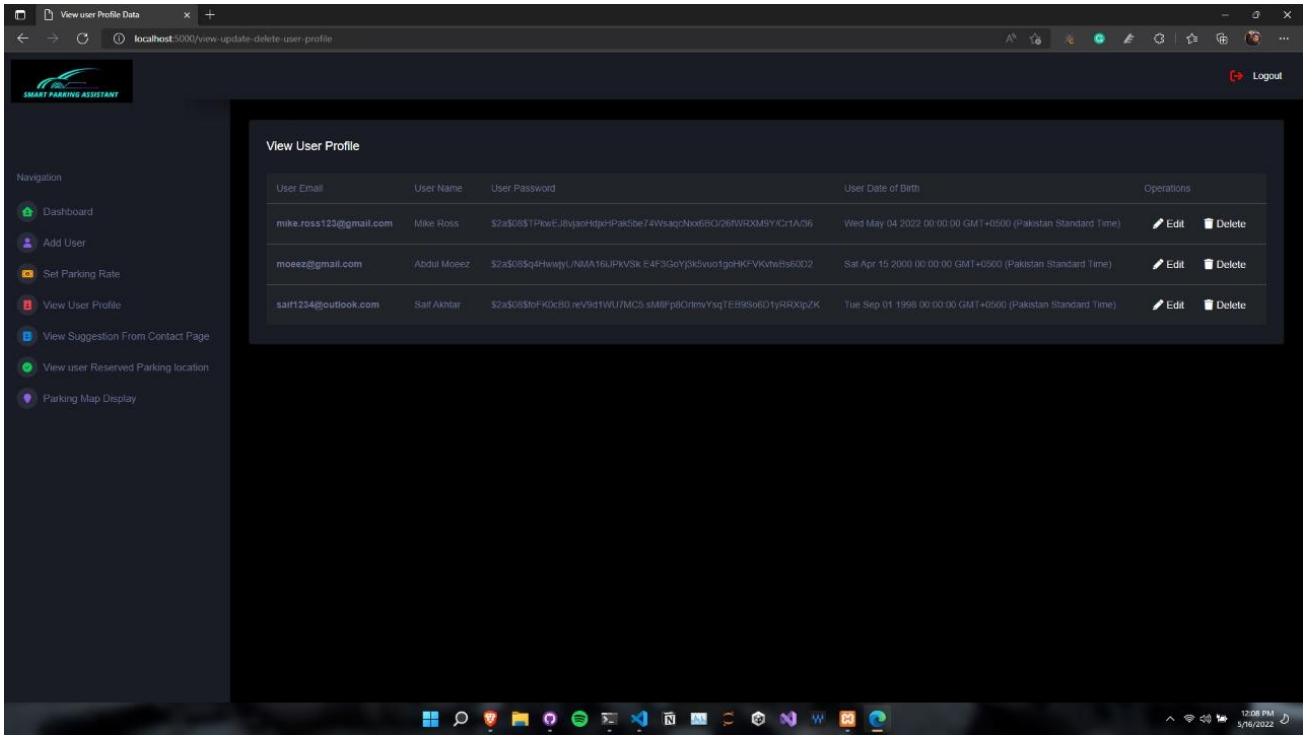


Figure 26: View user Profiles

6. View Sensor Location:

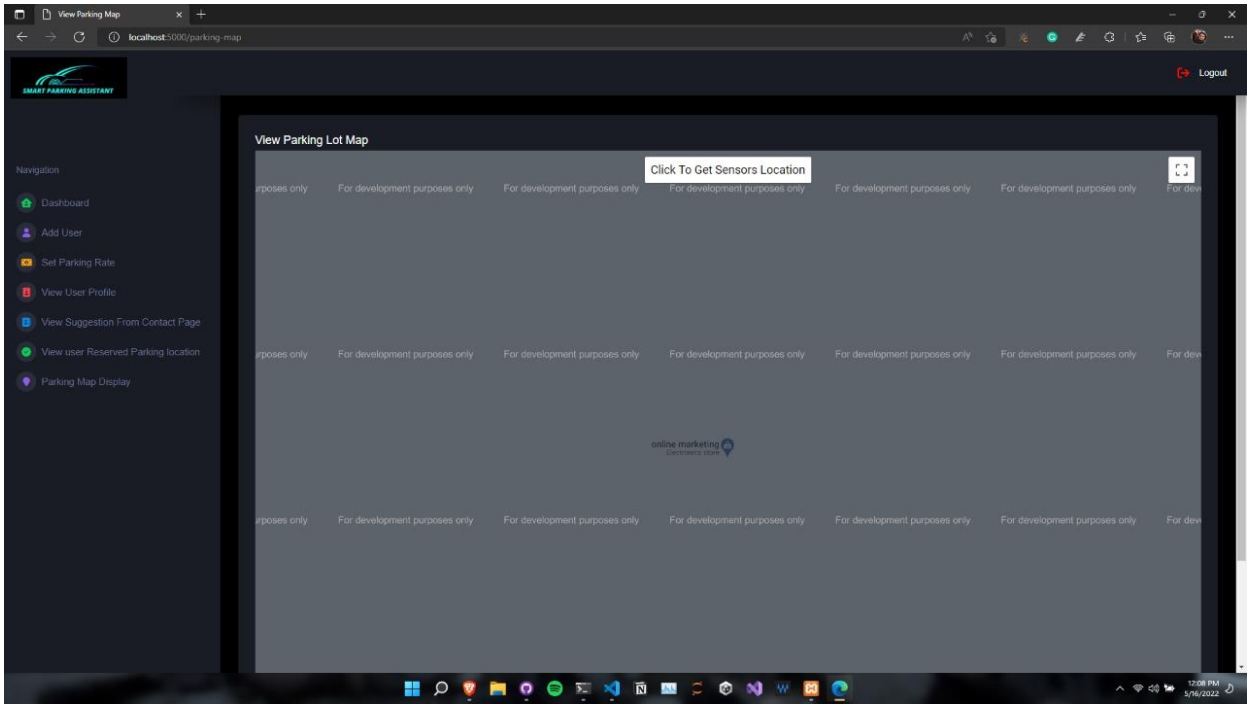


Figure 27: Sensor Location

7. Admin Control & Parking Map View:

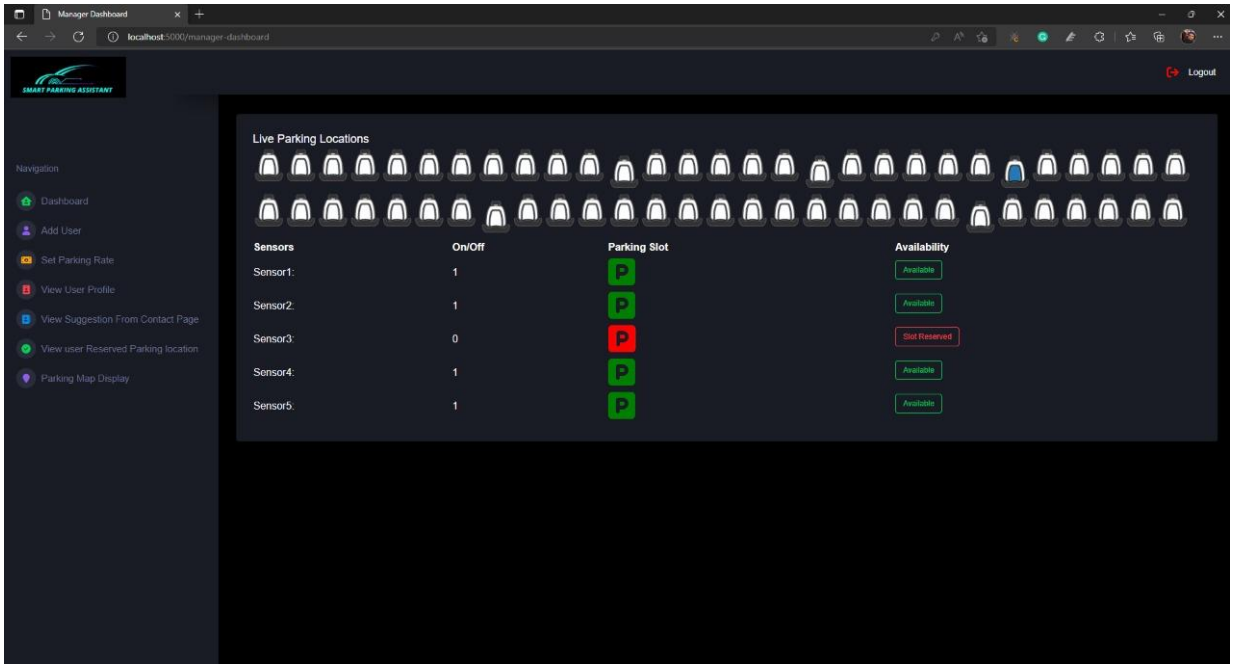


Figure 28: Parking Map

8. About Page:

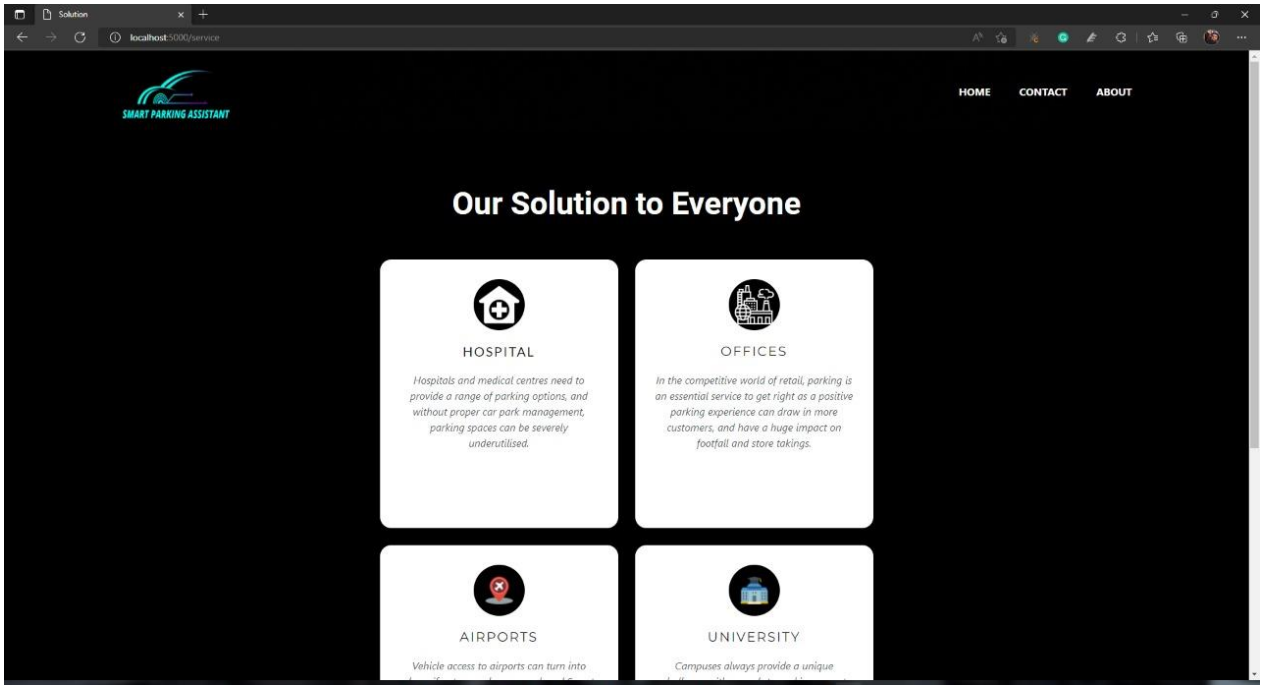


Figure 29: About Page

9. Contact Page:

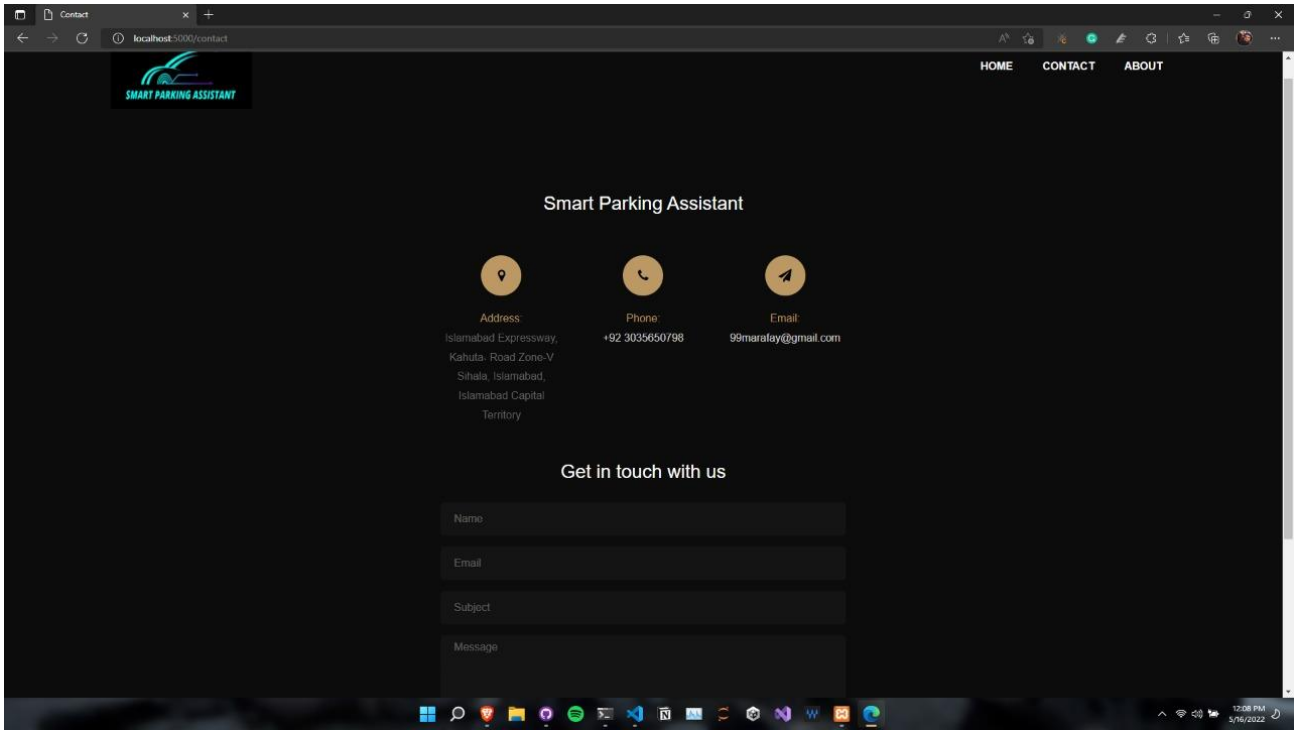


Figure 30: About Page

3.7. Hardware to Software COTS

In this we are using a couple of hardware and creating a server locally and a couple of hardware called Arduino

1. **Arduino Uno:**

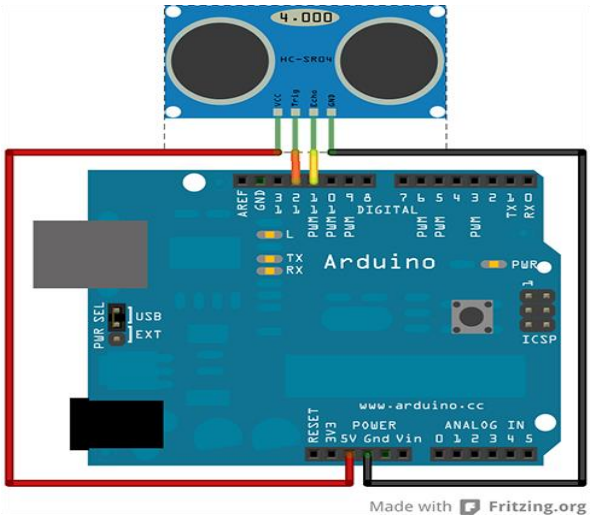


Figure 31: hardware

The Arduino UNO is a microcontroller board primarily based at the ATmega328. It has 14 digital enter/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz crystal oscillator, a USB connection, a electricity jack, an ICSP header, and a reset button. It incorporates everything needed to assist the microcontroller; sincerely join it to a pc with a USB cable or energy it with a AC-to-DC adapter or battery to get commenced.

Within the venture we used Arduino to application the sensors that connect with it, we simply were given keep of an HC-SR04 Ultrasonic Sensor Module. That is a quick publish on hooking it up to an Arduino Uno and getting distance facts from it. The sensor has 4 pins: - VCC (5V), GND, trigger and Echo. As proven inside the photograph above, the manner it works is:

- Ship a 10us excessive pulse on the trigger pin.
- The sensor sends out a "sonic burst" of eight cycles.
- Listen to the Echo pin, and the length of the subsequent high signal will give you the time taken by means of the sound to go to and fro from sensor to goal.

Chapter 4

Software Development

The implementation is one of the most important part of a project and the design need to be similarly to the code of the application, in our application we use standard coding statement and used correct statement.

4.1. Development Environment

During the development of this project we are using multiple programming languages and multiples libraries to achieve a perfect product with good quality.
The programming languages we used in this application are:

2. SQL
3.HTML
4. CSS
5. JavaScript
6.C

SQL:

In our application we are using SQL to generate s sequence of database and there are multiple tables in the database.

JavaScript:

In the application we are using JavaScript to develop a server that will communicate with the Arduino UNO board and will display the location of each parking spot on the web page.
In JavaScript we are using a framework called **Node JS**. In **Node JS** we are using two main Libraries and these are called "**Socket.io**" and "**Serial port**" using these libraries the Arduino board can communicate with the web page. To install these libraries, install node js from the google and then using a package manager called npm type the following command in the window terminal

1.1 socket.io: npm i socket.io

1.2 serial port: npm i Serial port

Once these libraries are installed and the code for the connection is written then start the server by enter the following command: **node server.js**

C Programming Language:

We us C programming languages to write code for the hardware which will be uploaded in the Arduino board.

4.2. Software Description

4.2.1. Hardware Code:

To run the code there are two main functions called void setup () and void loop ().
The setup function contains the basic input and out for each sensor where the loop will keep the run the code in a loop and will display the output.

Sensor 1 Code:

pin Mode (13, INPUT);
pin Mode (3, INPUT);
Serial. Begin (9600);
pin Mode (9, OUTPUT);

Description:

The above code displays the input line, the output line and beginning port for the sensor, pin Mode is the build in function that will take two inputs the pin number on the Arduino Board and the INPUT or the OUTPUT.

4.2.2. Software Code:

The software side is a website that is divided into three main parts Server Side, Client Side and APIs(controllers) all sides have multiple functions to perform and keep the website running.

Server Side:

For the server side there is just one file called app.js. The code is explained bellow:

Code Snippets 01:

```
const express = require("express");
const path = require("path");
const con = require("./database/connection");
const session = require("./session/session");
```

Code Description:

The above is importing multiple npm packages that are required for the server to run. The express package is a npm package that is used to start the server. Path is used to get the path of different directory so that files can render properly. Database connection is called and stored in a variable called con. Similarly, the session file is being called and the result is being stored in an object.

Code Snippets 02:

```
app.get("/static", (req, res) => {
  res.render("static");
});

app.use(
  session({
    secret: "ABCDefg",
    resave: false,
    saveUninitialized: true,
  })
);
```

Code Description:

In the above patch of code, we are telling the server to render the static images on the webpages and those images will be static.

Code Snippets 03:

```
const publicDirectory = path.join(__dirname, "./public");
```

Code Description:

In the above patch of code, we are declaring that our public directory will be called **public** and telling the location of public directory to the server.

Code Snippets 04:

```
app.set("view engine", "hbs");
app.use("/", require("./routes/pages"));
app.use("/auth", require("./routes/auth"));
const port_website= process.env.port || 5000;
app.listen(port_website, () => {
  console.log("Node Server is running at port 5000");
});
```

Code Description:

In the above patch of code, we ae telling the server to render webpages in the form of handlebars and then run the server on Port 5000,

Code Snippets 05:

```
var http = require("http");
var fs = require("fs");
var index = fs.readFileSync("./views/view-parking.hbs");
const { SerialPort } = require("serialport");
const { ReadlineParser } = require("@serialport/parser-readline");
var port = new SerialPort({
  path: "COM4",
  baudRate: 9600,
  dataBits: 8,
  parity: "none",
  stopBits: 1,
});
const parser = port.pipe(new ReadlineParser({ delimiter: "\r\n" }));
port.pipe(parser);

parser.on("data", function (data) {
   console.log(data);
});
var app = http.createServer(function (req, res) {
  res.writeHead(200, { "Content-Type": "text/html" });
  res.end(index);
});
```

```
const io = require("socket.io")(app);
parser.on("data", function (data) {
   console.log("Received data from port: " + data);
  io.emit("data", data);
});
// const port_hardware= process.env.port || 3000;
//THis will be port number through which the derever for hardware will be Listening.
app.listen(3000, () => {
  console.log("The server for the hardware is working on port 3000");
});
```

Code Description:

In the above patch of code, we are connecting a micro controller with the nodejs server. We are connecting the micro controller to the PC on port CM04 and using https, socket.io we are passing the data the nodejs server where it's being transferred or display to various part of the website. This part is also being executed and being running of port 3000.

API Side:

In this there are multiple APIs or controller. For this project we have one main controller that will pass database or get data from other controller the main controller is called **auth.js** the code explain is stated below:

Code Snippets 01:

```
const express = require("express");

const authController = require("../controllers/auth")
const suggestionController = require("../controllers/suggest")
const managerloginController = require("../controllers/managerlogin")
const userlogin = require("../controllers/userlogin")
const reverseparking= require("../controllers/reverseparking")
const permissionParking = require("../controllers/permissionParking")
const approved = require("../controllers/approved")
const usermanagementController = require("../controllers/user-management-curd")
const suggestion_view = require("../controllers/suggestion_view")
```

Code Description:

In the above patch of the code we are including different controller and each controller have a role for example one controller will be responsible for adding user to the database, for each feature there is one controller. The express is used to make this file a part of the server.

Code Snippets 02:

```
const router = express.Router();
router.post("/register",authController.register);
router.post("/suggestion",suggestionController.register);
router.post("/manager_login",managerloginController.register);
router.get("/view-update-delete-user-profile",usermanagementController.view);
router.get("/view-suggestions",suggestion_view.view);
router.get("/approved",approved.view);
router.post("/user-login",userlogin.register);
router.post("/reverseparking",reverseparking.register);
module.exports = router;
```

Code Description:

In the above patch of code, here are telling each controller what is the responsibility of each controller. If there .register at the end that means that data is being added, updated or deleted, if there is .view at the end that means the data is being read only. The last line module. export tells that this file cab be called by the **app.js** file.

Database Connection:

The is the file where the connection is established with the SQL server, All the configuration, user name and password are set and they are used for connection. The connection is checked in the **connection.js** file, if there is any error then connection.js file will report any error to the server **app.js.** The app.js file will through an error and will alert the user.

Code Snippets 01:

```
const express = require("express");
const mysql = require("mysql");
const dotenv = require("dotenv");
dotenv.config({path:  './.env'});
const db = mysql.createConnection({
   host: process.env.DATABASE_HOST,
   user: process.env.DATABASE_USER,
   password: process.env.DATABASE_PASSWORD,
   database: process.env.DATABASE,
   port: process.env.DATABASE_PORT
});
db.connect( (error, connection) => {
   if(error){
      console.log(error);
   }
   else {
      console.log("My SQL Database is connected successfully...");
   }
});

module.exports = db;
```

Code Description:

In the above patch of code, we are importing the express, SQL, dotenv npm packages and file. Once all the files are included then we are creating a connection using the build in function. The status of the connection is stored in an object, if there is any error then it will print the error on the terminal, if there is no error then a message will be displayed that database is connected.

Pages Render:

The file name is **pages.js.** This page is responsible when a URL is called then if URL is matched then server will render the following page. The render of each page is started bellow:

Code Snippets 01:

```
const express = require("express");
var con = require("../database/connection");
const router = express.Router();
```

Code Description:

In the above patch of code, we are including the connection file, router package and express package because it can detect different URL and tell server which page should be render.

Code Snippets 02:

```
router.get("/", (req, res) => {
  res.render("index");
});
router.get("/dashboard", (req, res) => {
  res.render("dashboard");
});
router.get("/manager_login", (req, res) => {
  res.render("manager_login");
});
router.get("/contact", (req, res) => {
  res.render("contact");
});
```

Code Description:

The above patch of code, will get the URL and then render the pages. The server will take a request and will sent a respond to the client. For example, In the above code /dashboard is being request and in its respond dashboard.hbs is being render. From the browser a URL will be sent as request and bases on that URL a respond will sent. All the webpages are render on the bases of URL.

SensorOnOff Side:

In this file the values of the sensor will stored and on the bases of that value the out will be changed. There are 5 sensors with default two values 0 or 1 and whatever is the value from the micro controller the values will be changed, this is hosted on port 3000. The code description is stated bellow:

Code Snippets 01:

```
var socket = io.connect("http://localhost:3000/");

socket.on("data", function (data) {
  var res = data.split(" ");
  if (res[0] + res[1] === "Sensor1:") {
    document.getElementById("sensor1").innerHTML = res[0] + res[1];
    document.getElementById("sensor1On/Off").innerHTML = res[2];
    if (res[2] === "1") {
      document.getElementById("sensor1ParkOn").innerHTML =
        " <i class='fa-solid fa-square-parking fa-3x' style='color:green;height: 50px;width: 50px;'></i>";
      document.getElementById("sensor1availability").innerHTML =
        " <h6 class='badge badge-outline-success'>Available </h6>";
      document.getElementById("sensor1ParkOnEmptyImage").innerHTML =
        " <img src='/image/empty.png' style='height: 50px;width: 50px;'></img>";
    }
    if (res[2] === "0") {
      document.getElementById("sensor1ParkOn").innerHTML =
        " <i class='fa-solid fa-square-parking fa-3x' style='color:red;height: 50px;width: 50px;'></i>";
      document.getElementById("sensor1availability").innerHTML =
        " <h6 class='badge badge-outline-danger'>Slot Reserved </h6>";
      document.getElementById("sensor1ParkOnEmptyImage").innerHTML =
        " <img src='/image/reserved.png' style='height: 50px;width: 50px;'></img>";
    }
  }
```

Code Description:

In the above patch of code, we are checking the Sensor ID, the sensor Id is stored in the micro controller and we know the Id, all the values are stored in a list and based on that list if value is 0 then display this output but if the value is one then display other output and keep checking until the value is static.

Client Side:

On the client there will be webpages, using these webpages the user will interact with the hardware and product as well. For the website we are using three programming Languages HTML, CSS and JavaScript. The section of the website is stated bellow:

Code Snippets 01:

```
<div class="banner">
        <div class="navbar">
            <img  src="/image/logo2.jpg"class="logo">
            <ul>
                <li><a href="/">Home</a></li>
                <li><a href="/contact">Contact</a></li>
                <li><a href="/about">About</a></li>
            </ul>
        </div>
        <div class="content">
            <h1 class="hover-underline-animation">Smart Parking Assistant</h1>
            <br>
            <p class="hover-underline-animation">Solution to your Parking Problem</p>
            <div>
                <a href="/dashboard"><button type="button"><span></span>Login</button></a>
                <a href="/service"><button type="button"><span></span>Service</button></a>
            </div>
        </div>
    </div>
```

Code Description:

The file name is index.hbs and this file is the main home page of the product. In the above code there is a company logo and a couple of options for the user. One of the options is to login and second option is checkout the services of the company. These options are linked with the URL and once they are clicked the URL are sent and as a respond webpages are render.

Code Snippets 02:

```
<div class="banner">
        <div class="navbar">
            <img  src="/image/logo2.jpg"class="logo">
            <ul>
                <li><a href="/">Home</a></li>
                <li><a href="/contact">Contact</a></li>
                <li><a href="/about">About</a></li>
            </ul>
        </div>
        <br>
        <br>
        <form method="POST" action="/auth/manager_login">
            <h2> Manager Login</h2>
            <label>Email: </label>
            <input type="text" id="email"name="email" placeholder="Email"><br>

            <label>Password</label>
            <input type="password" id="password" name="password" placeholder="Password"><br>

            <button class="btn"type="submit">Login</button>
        </form>
</div>
```

Code Description:

The above patch of code, is the login page for the manager, there is a form where manager will enter its email and password. When the manager press login button then a auth/manager/ URL is called and when this URL is called then in the background, manager email and password is checked, if the email and password is correct then manager will be redirected toward it's dashboard. If email or password is incorrect the error message will be displayed on the webpages.

Code Snippets 03:

```
<div class="row">
    <div class="col-12">
        <div class="preview-list">
            <table style="width:100%">
                <tr>
                    <th>Sensors</th>
                    <th>On/Off</th>
                    <th>Parking Slot</th>
                    <th>Availability</th>

                </tr>
                     <h3>Parking Map</h3>
                <tr>
                    <i id="sensor1ParkOnEmptyImage" style="margin-bottom:20px;"></i>
                    <i id="sensor2ParkOnEmptyImage" width="50px" height="50px"
                        style="margin-bottom:20px;"></i>
                    <i id="sensor3ParkOnEmptyImage" width="50px" height="50px"
                        style="margin-bottom:20px;"></i>
                    <i id="sensor4ParkOnEmptyImage" width="50px" height="50px"
                        style="margin-bottom:20px;"></i>
                    <i id="sensor5ParkOnEmptyImage" width="50px" height="50px"
                        style="margin-bottom:20px;"></i>
                </tr>
                <br>
                 <h3>Sensor Detials</h3>
                <tr>
                    <td id="sensor1"></td>
                    <td id="sensor1On/Off"></td>
                    <td><i id="sensor1ParkOn" class="img-responsive"></i></td>
                    <td id="sensor1availability"></td>


                </tr>
                <tr>
                    <td id="sensor2"></td>
                    <td id="sensor2On/Off"></td>
                    <td><i id="sensor2ParkOn" class="img-responsive"></i></td>
                    <td id="sensor2availability"></td>

                </tr>
                <tr>
                    <td id="sensor3"></td>
                    <td id="sensor3On/Off"></td>
                    <td><i id="sensor3ParkOn" class="img-responsive"></i></td>
                    <td id="sensor3availability"></td>

                </tr>
```

Code Description:

In the above patch of code, this is the manager dashboard once the manager is logged in then manager will be able to see the parking map and the location of sensors as well. There is a navbar where other options for manager exist.

Code Snippets 04:

```
<script type="text/javascript">
    document.getElementById('fetch').onclick = function () {
        var rows = document.getElementById('rows').value
        var cols = document.getElementById('cols').value
        if (rows == "" || rows == null)
            rows = 10
        if (cols == "" || cols == null)
            cols = 10
        var output = '<table class="table">'
        function createTable(rows, cols) {
            for (var i = 1; i <= rows; i++) {
                output = output + '<tr>'
                for (var j = 1; j <= cols; j++) {
                    output = output + '<td>' + "<i id='sensor" + j + "ParkOnEmptyImage' width=50px height=50px style='margin-bottom:20px;'></i> <br><br><br> <img src='/image/empty.png' width=50px height=50px style='m
                }
                output = output + '</tr>'
            }
            output = output + '</table>'
            document.getElementById('container').innerHTML = output
        }
        createTable(rows, cols)
    }
</script>
```

Code Description:

One of the main features of the manager is the custom parking map. So we designed and code a function in which the manager will take two input rows and columns and then on the bases of that input a custom parking map will be generated.

Chapter 5

System Testing

Software Testing is the most crucial part of Software Development Process. It is the investigation or evaluation of a software component, improving them, and finding bugs and defects. Testing is usually done by executing a system in such a way that it identifies any gaps, errors, or missing requirements in contrary to the actual requirements.

5.1 Testing Methodology

5.1.1 Black Box Testing

Testing a gadget in a "black field" is doing so without understanding some thing approximately the way it operates within. A tester inputs information and monitors the output produced by using the gadget being tested. This lets in for the identity of the system's reaction time, usability problems, and reliability worries in addition to how the system reacts to predicted and sudden person activities.

Because it checks a device from beginning to cease, black field checking out is a amazing checking out technique. A tester can imitate person action to test whether the gadget fulfils its guarantees, lots as stop users "do not care" how a machine is programmed or designed and assume to get a suitable response to their requests. A black box takes a look at exams all pertinent subsystems alongside the direction, including UI/UX, the web server or software server, the database, dependencies, and other additives.

5.2. Testing Environment

Our hardware was used extensively and the software was tested for different error checking. For hardware the environment was providing over usage of the sensors.

5.2.1 System Level Testing

Black box checking out is a sort of checking out method wherein the tester creates test instances to validate the capability of the software program whilst possessing unique know-how of the way the product functions.

Software programming skills isn't always necessary. Every test case is created through considering the enter and output of a selected characteristic. A tester is only aware about the unique outcome of a given enter; they're blind to the method with the aid of which the result is generated. Black field trying out employs a number of special trying out strategies, consisting of the choice desk method, the boundary fee analysis method, the kingdom transition, the All-Pair trying out approach, the cause-impact graph technique, the equivalence partitioning approach, the error guessing approach, the use case technique, and the person tale approach. Every of these strategies is described in.

5.3. Test Cases

5.3.1 Hardware Test Cases

Test Case 1

Test case description

Circuit is being tested for faults. This test case will focus on power distribution. Power is fluctuated between 3 – 5 volts to test for system or component failure. How test case was generated

Circuit was deliberately overloaded with power

Expected result of the test case

System should keep running but show a faulty end/component

Actual result of the test case

System shows that faulty sensor does not deliver values

Table 10: Hardware test Case 01

Date: 15-1-2021	
System*:* Smart Parking Detection	
Objective*:* View all component workings	**Test ID***:*1
Version*:*1	**Test Type***:* Hardware testing
Input*:* Voltage fluctuation (3.5v – 5v)	
Expected Result*:* Keeps working but presents error	
Actual Result*:* Sensor range decreases and a fuse fails but system keeps working. Shows fault.	

Test Case 2

Test case description

Circuit is being tested for faults. This test case will focus on power distribution. The parallel circuit system is tested by removing certain components to see if the system keeps working.

How test case was generated

Components are removed individually

Expected result of the test case

System should keep running but show missing component or no reading

Actual result of the test case

System shows that missing sensor does not deliver values

Table 11: : Hardware test Case 02

Date: 15-1-2021	
System*:* Smart Parking Detection	
Objective*:* View all component workings	**Test ID***:2*
Version*:*1	**Test Type***:* Hardware testing
Input*:* Parallel power test	
Expected Result*:* Keep working but presents error	
Actual Result*:* Other sensors keep working and circuit is intact	

Test Case 3

Test case description

System is being tested for detection. This test case will focus on obstructions. Different objects are placed at different ranges to the sensor to test which are detected best.

How test case was generated

Different objects are used to detect proximity results

Expected result of the test case

System should detect all or most objects

Actual result of the test case

System detects metallic objects very well, but synthetic materials less well

Table 12:: Hardware test Case 03

Date: 15-1-2021	
System*:* Smart Parking Detection	
Objective*:* View all component workings	**Test ID***:3*
Version*:*1	**Test Type***:* Hardware testing
Input*:* Object detection test	
Expected Result*:* System detects all objects significantly covering the sensors	
Actual Result*:* Metallic objects (metal chassis) detection is better than plastic/synthetics	

Test Case 4

Test case description

System is being tested for capacity. This test case will focus on multiple object detection.

How test case was generated

The system is pushed to maximum load, with all sensors occupied and all parking spaces being shown simultaneously.

Expected result of the test case

System should maintain speed and show all results

Actual result of the test case

System slows down to 3/4th speed but keeps working

Table 13: Hardware test Case 04

Date: 15-1-2021	
System*:* Smart Parking Detection	
Objective*:* View all component workings	**Test ID***:4*
Version*:*1	**Test Type***:* Hardware testing
Input*:* All parking spaced filled	
Expected Result*:* System works at normal performance	
Actual Result*:* System slows down once more than 3/4th parking is filled	

5.3.2 Software Test Cases

Test Case 1

Test case description

System is being tested for successful login and signup. User will try to login to the system or create a new account.

How test case was generated

Account creation was needed to access system.

Expected result of the test case

System should successfully log user in, or create a new account and store data to database.

Actual result of the test case

System completes login/signup successfully.

Table 14: Login test Case

Date: 2-2-2021	
System*:* Smart Parking Detection	
Objective*:* Check login process	**Test ID:***5*
Version*:*1	**Test Type***:* Software testing
Input*:* Email, password.	
Expected Result*:* User logs in without issues	
Actual Result*:* User login is successful	

Test Case 2

Test case description

System is being tested for data manipulation. Admin will have control over user accounts and information.

How test case was generated

Admin redirects to database management page.

Expected result of the test case

Database is changed, users can be updated, deleted or added.

Actual result of the test case

Users are updated, deleted or added.

Table 15: Data update test case

Date: 3-2-2021	
System: Smart Parking Detection	
Objective: Check if data is updated without issues	***Test ID:****6*
Version:1	***Test Type:*** Software testing
Input: New user data	
Expected Result: Data is updated	
Actual Result: Data is updated	

Test Case 3

Test case description

System is being tested for viewing parking spaces

How test case was generated

User selects the option for parking detection

Expected result of the test case

System should detect and show all spaces, occupied and vacant

Actual result of the test case

System detects spaces, suggests parking, shows reservations and optional switch for admin

Table 16: Parking Detection test case

Date: 3-2-2021	
System*:* Smart Parking Detection	
Objective*:* Check that parking detection is working	**Test ID***:7*
Version*:*1	**Test Type***:* Software testing
Input*:* Fill parking spaces	
Expected Result*:* System should detect and show all spaces, occupied and vacant.	
Actual Result*:* System detects spaces, suggests parking, shows reservations and optional switch for admin	

Test Case 4

Test case description

System is being tested for security. If user logs out, the following activity must be monitored.

How test case was generated

User presses logout button

Expected result of the test case

System logout and destroy the session on command

Actual result of the test case

System destroys the session and the user has to enter login details again

Table 17: Logout test case

Date: 4-3-2021	
***System**:* Smart Parking Detection	
***Objective**:* Check if user is logged out without issues	***Test ID**:8*
***Version**:*1	***Test Type**:* Software testing
Input: Click logout button	
Expected Result: Session expires immediately upon logout	
***Actual Result**:* System destroys session and does not reload page without details	

Test Case 5

Test case description

System is being tested to check if a registered user can successfully reserve a parking space upon availability

How test case was generated

User presses “reserve” button

Expected result of the test case

System redirects to reservation page, shows available parking spaces, space is reserved

Actual result of the test case

System redirects to reservation page, shows available spaces, space is reserved

Table 18: Space detection test case 04

Date: 24-7-2022	
***System**:* Smart Parking Detection	
***Objective**:* check if space is reserved successfully	***Test ID**:9*
***Version**:*1	***Test Type**:* Software testing
Input: Select desired space	
Expected Result: Parking space is reserved if available, for the next hour	
***Actual Result**:* Parking space reservation did not retrieve data from back end in the first phase	

Test Case 6

Test case description

System is being tested to check if the parking map is successfully displayed

How test case was generated

User/Manager presses “view parking map” button

Expected result of the test case

System redirects to parking map page and displays parking grid

Actual result of the test case

System redirects to parking map page and displays parking grid

Table 19: Parking map test case

Date: 4-7-2021	
***System**:* Smart Parking Detection	
***Objective**:* Check if parking map is displayed accurately	***Test ID**:10*
***Version**:*1	***Test Type**:* Software testing
Input: Click parking map button	
Expected Result: System redirects to parking map page and displays parking grid	
***Actual Result**:* System redirects to parking map page and displays parking grid	

Chapter 6

Software Deployment

6.1. Installation / Deployment Process Description

Our Project is a web base application and for the deployment there are multiple phases to deploy the application, there are a lot of options to deploy your application so for this project we decided to choose AWS. Using the service of AWS database was uploaded and connected to the application and then the application was deployed. The steps to deploy the application is stated bellow.

For our application there are two main steps to deploy the application.

1. Deploy the Database
2. Deploy the nodes application.

6.2. Pre-requisites:

There are two things that are required

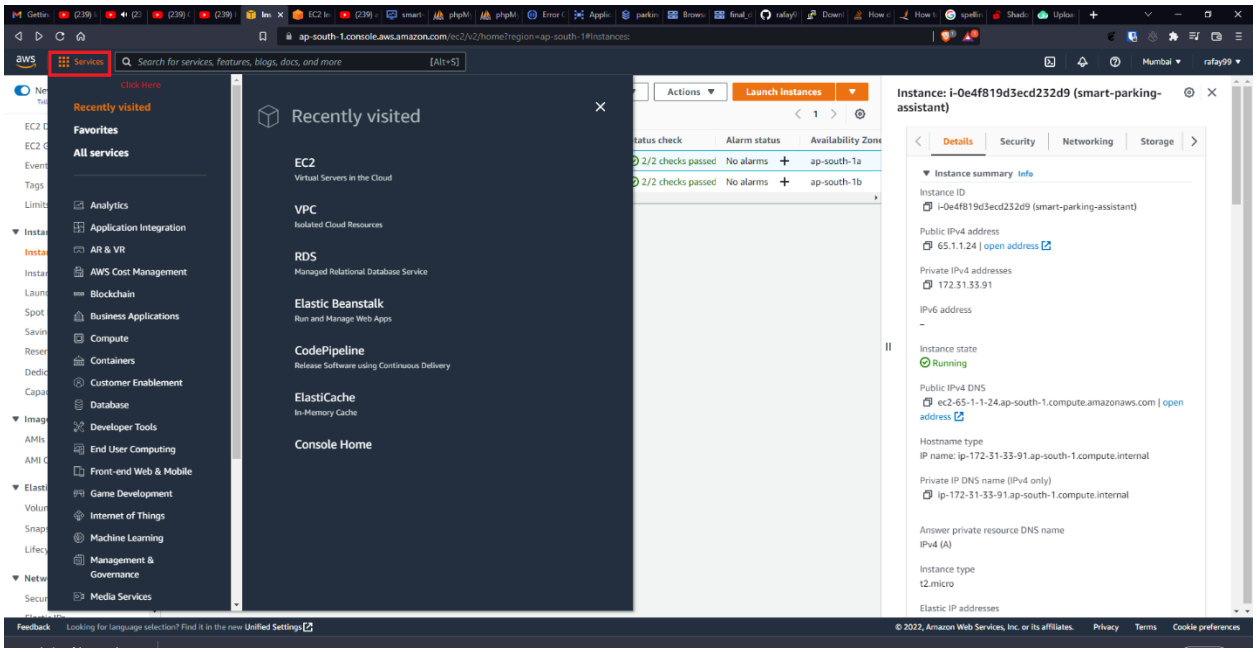
1. Credit Card or Debit Card with International Transmission.
2. AWS Account and Active account.

Precondition is that user is logged into the account.

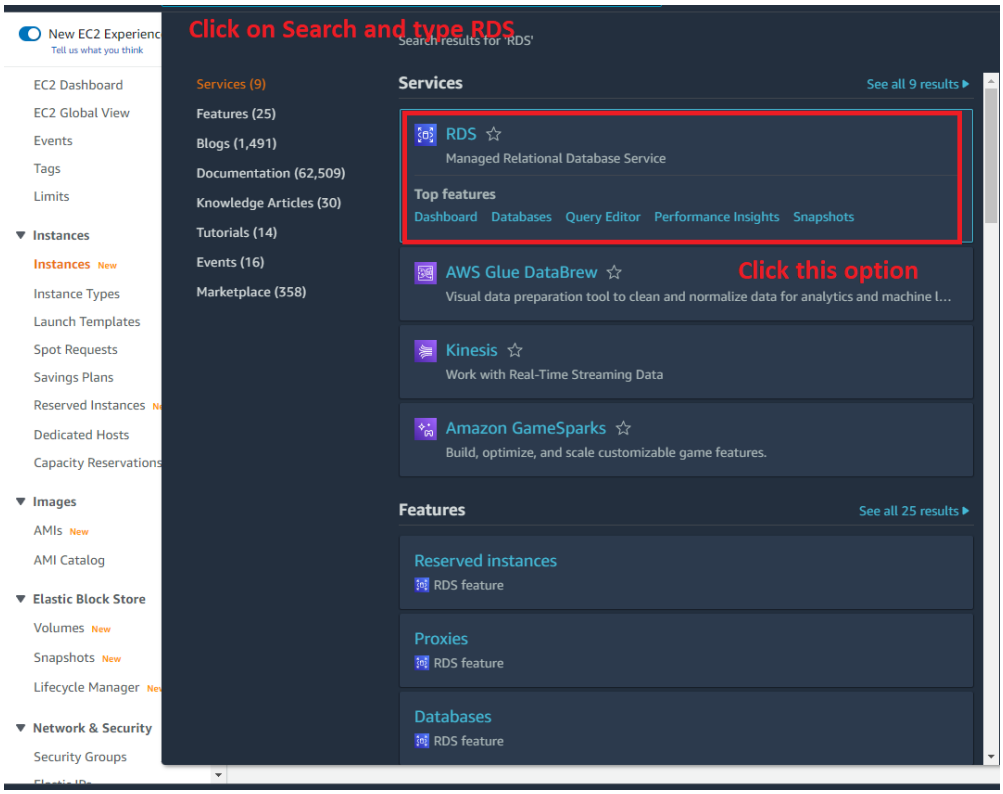
6.3 Deployment of Database:

In order to deploy the database, we are using RSB Service of AWS.

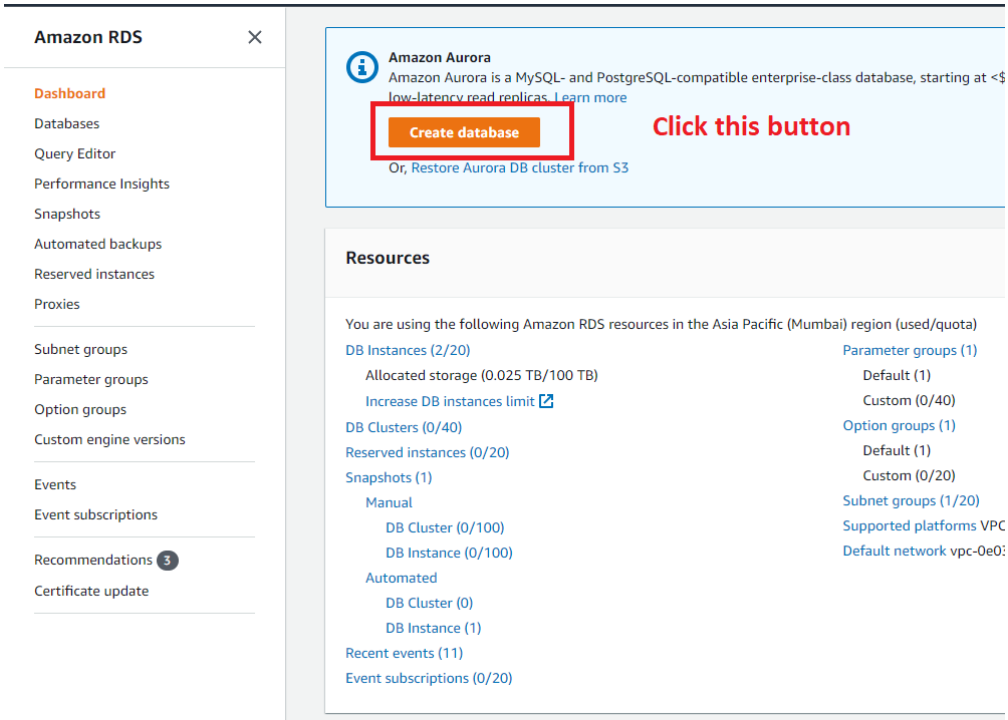
1. Click the Search Button.



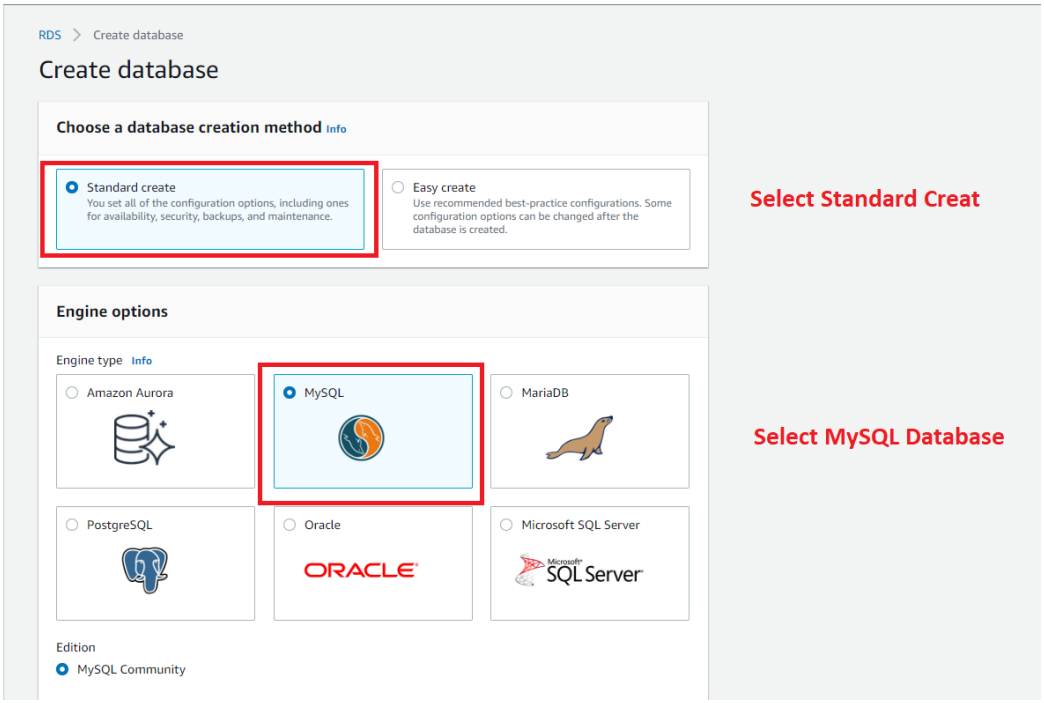
2. Search RDS in service Area and then click the first option



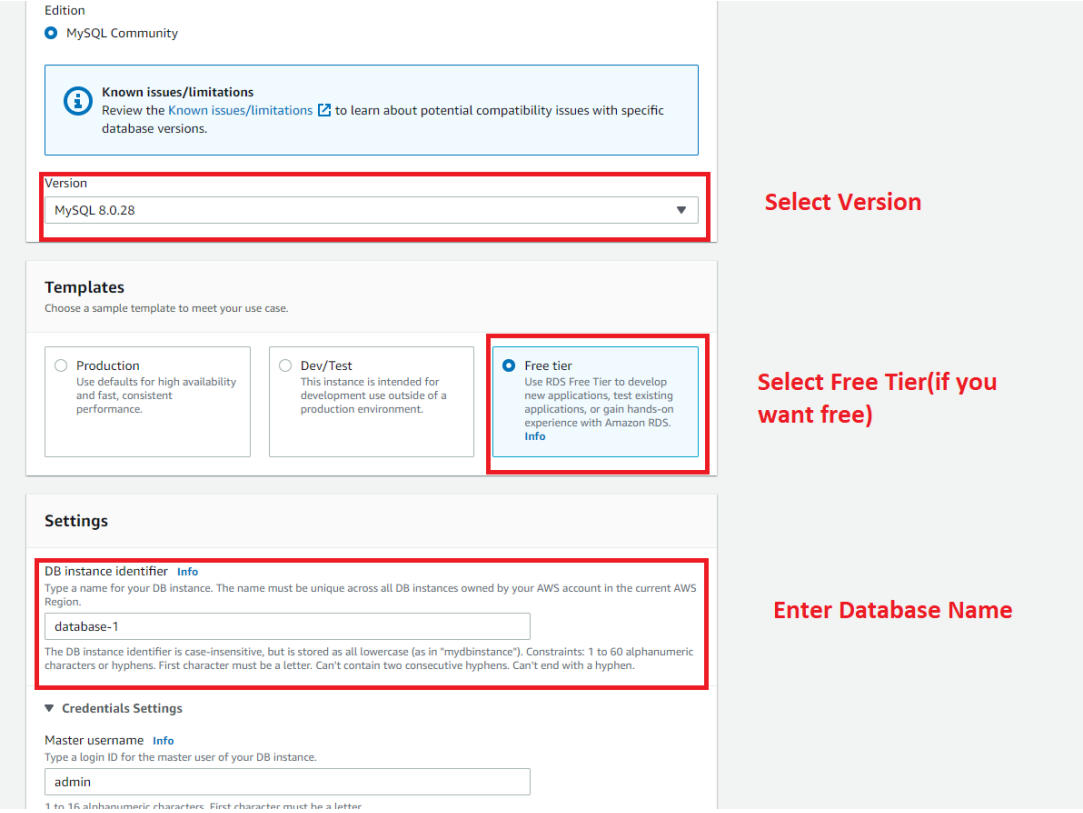
3. A Dashboard will appear and then click create Database:



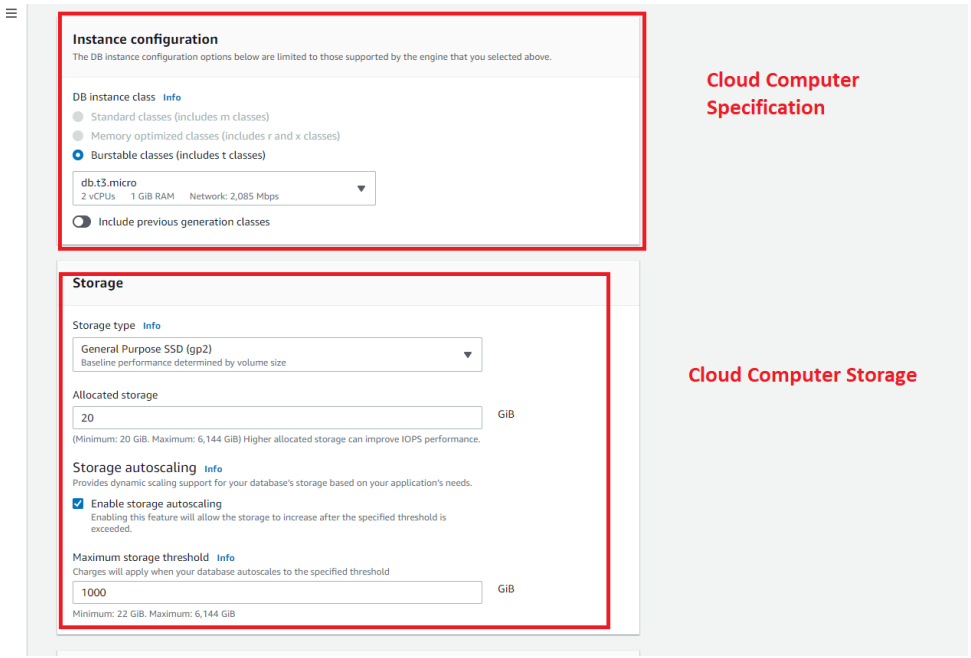
4. Select the Standard Pack and then Select the MySQL Version.



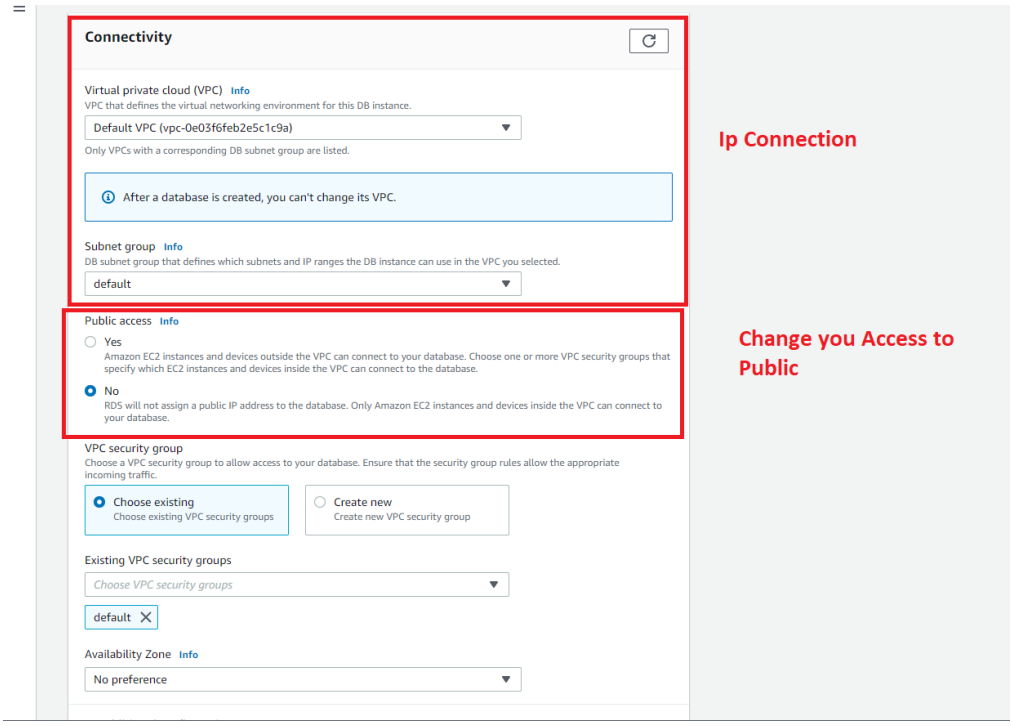
5. Select the version of SQL, Select the Free tier option and Enter the Database name



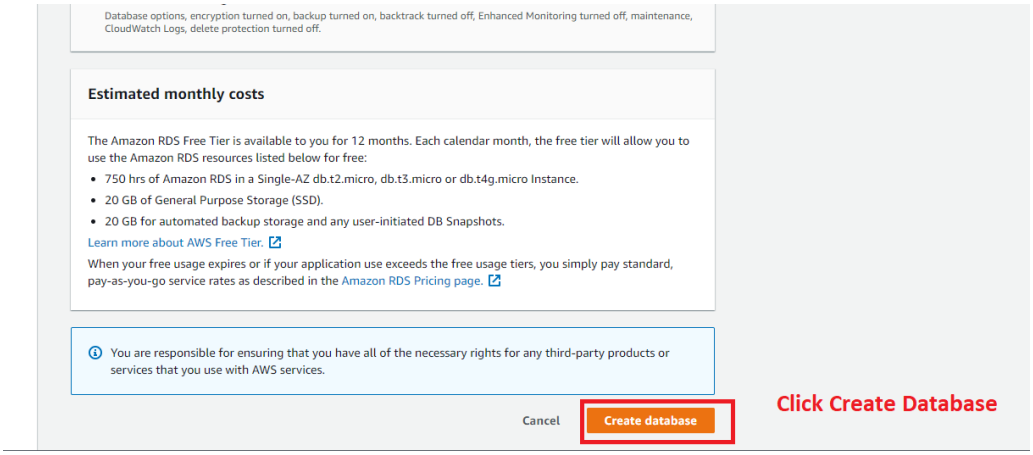
6. Enter the cloud computer Processing Power and select the storage option.



7. Select the default IP configuration and make sure to change the access point to Public by default it is Private, if it is private then database cannot be access.

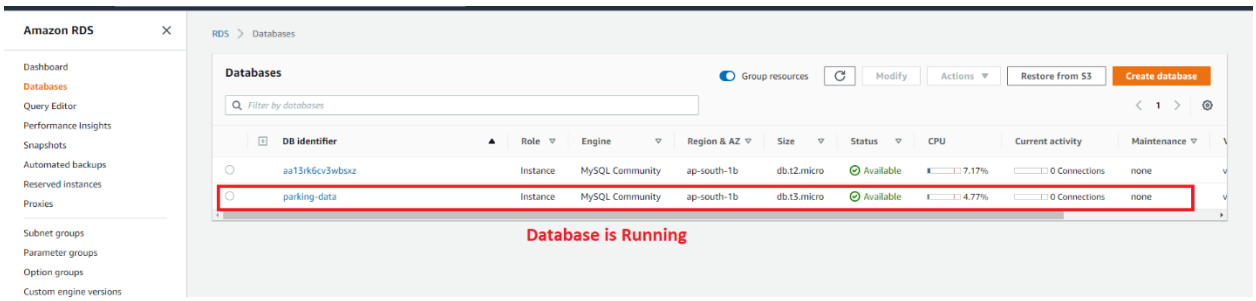


8. Then at the end click “Create Database”

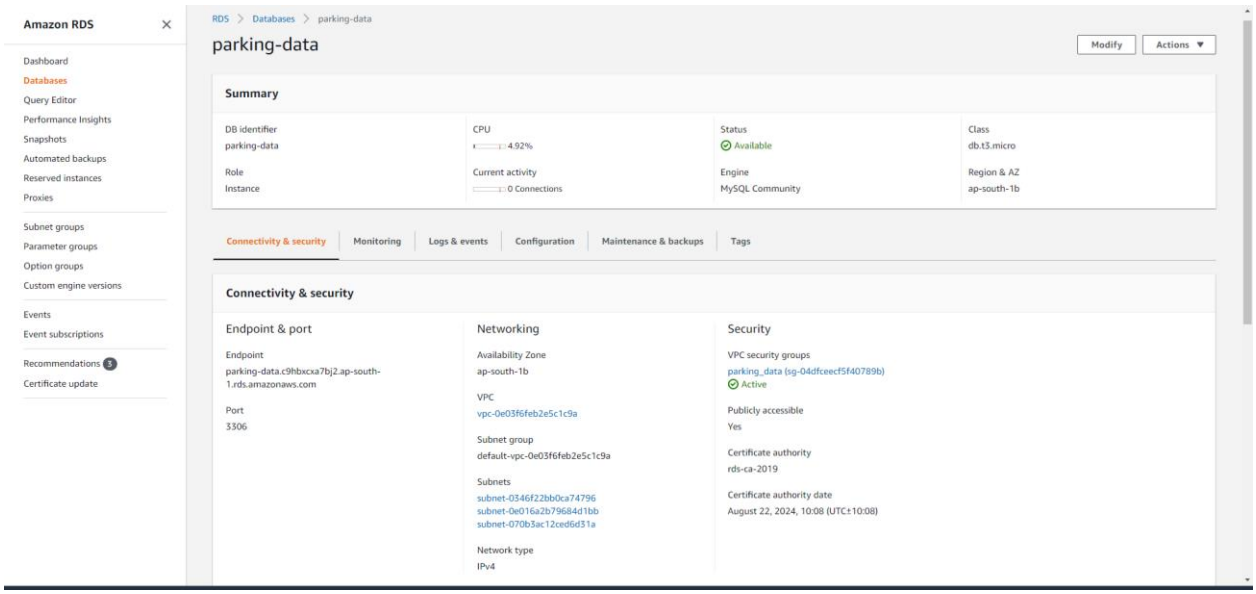


9. It will take some time but your database will be ready and with a public IP Address.

Once the database is created then it will appear in the RSD dashboard and will look like this:



To See the Database Dashboard, click on the Parking data or the name of the database, once clicked then a dashboard will appear.



6.4: Deploy the website

To deploy the website which is based on nodes, html and handlebars with bootstrap. So to upload this website we need a Linux Server that can run our run our application. There are multiple options as come to create Linux server. So for this project I am creating a Linux VM in AWS. To create this server, follow these steps,

6.4.1: Creating Web Server:

1. In the Search bar search “EC2” and select the EC2 option.

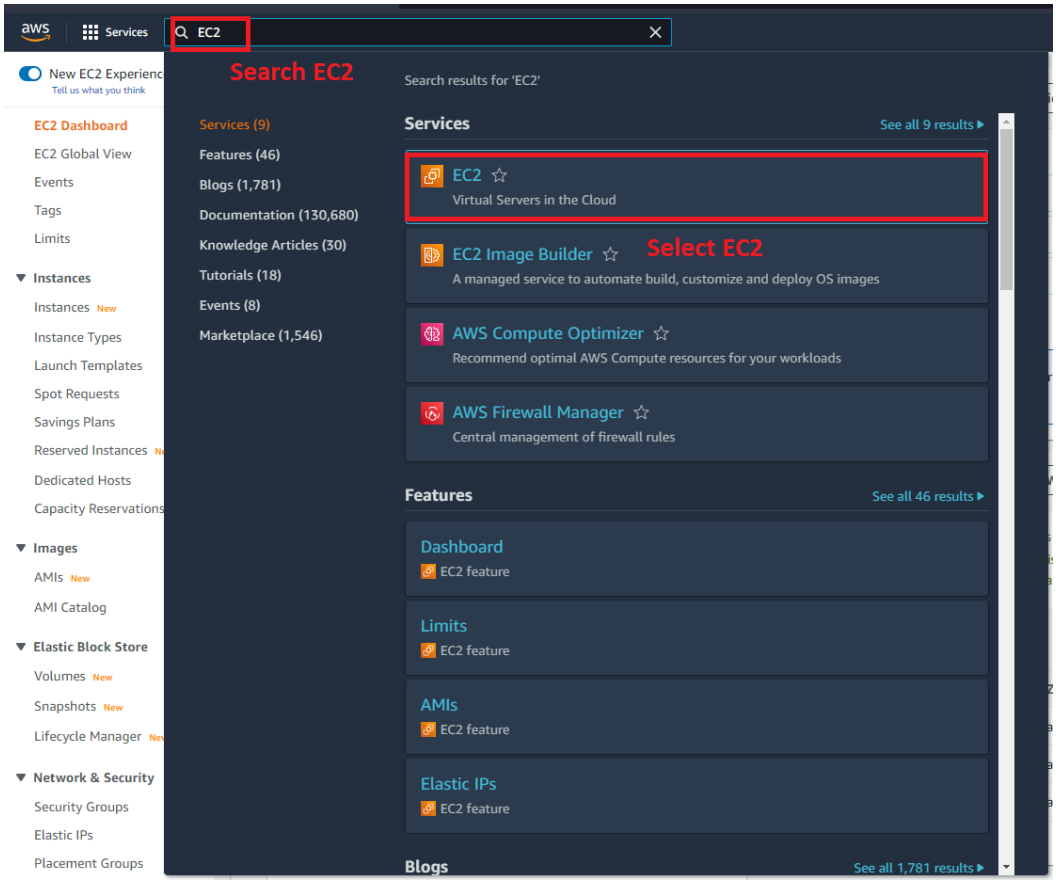


Figure 32: Creating web server

2. Click the Launch Instance button.

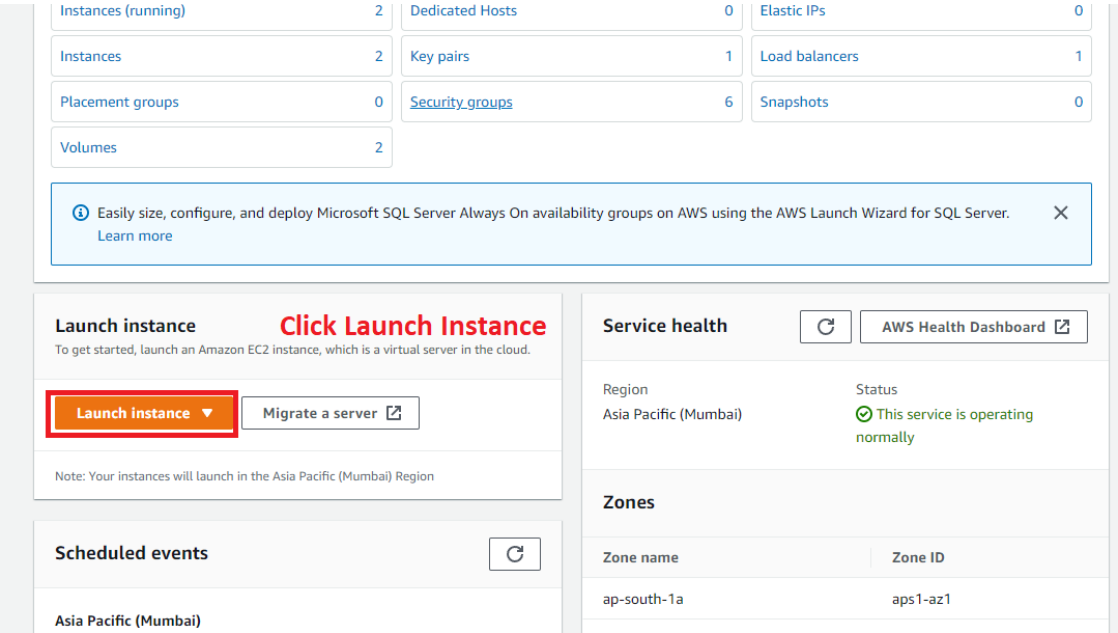


Figure 33: Launching installer

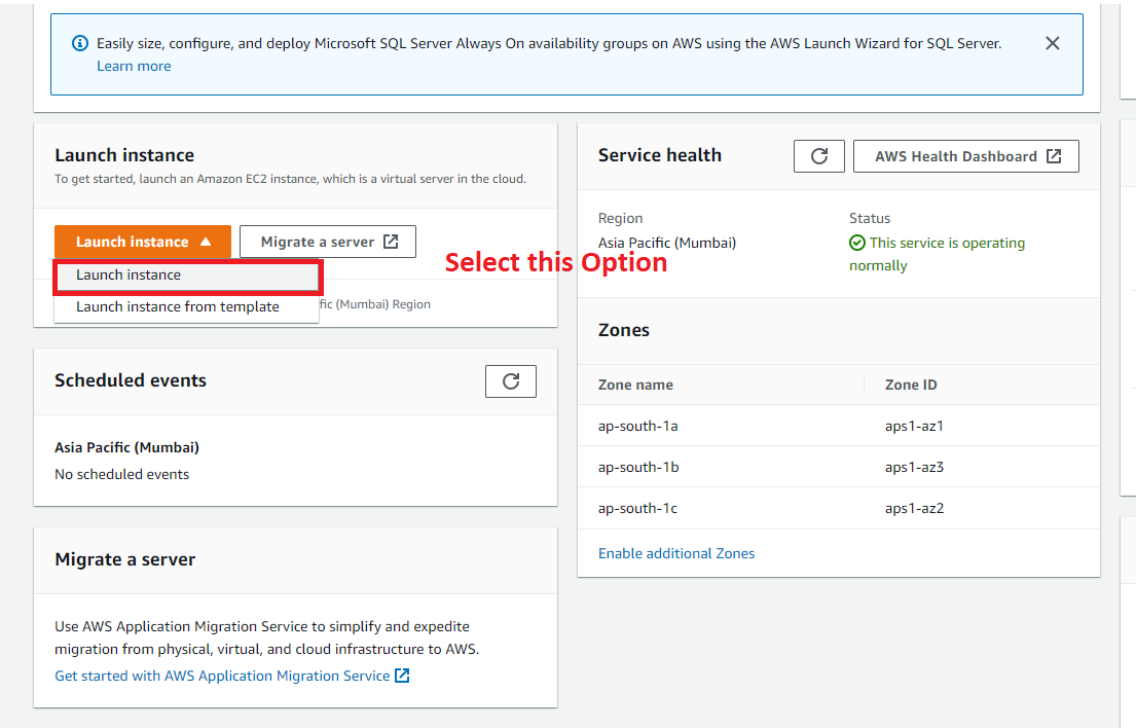


Figure 34 Instance loading screen

3. Now Set the detail for the server. Enter the name of the server and select the Ubuntu Linux Server.

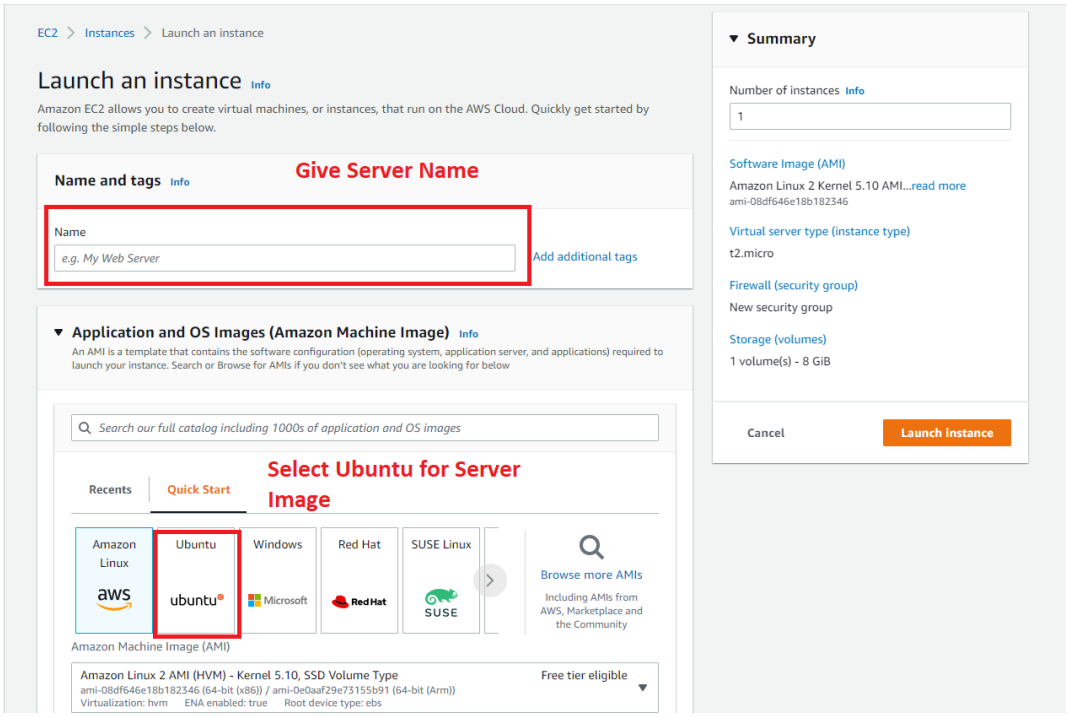


Figure 35: Linux settings

4. Define the Processing Power of the VM and Download the SSH key for connection with the connection with your computer.

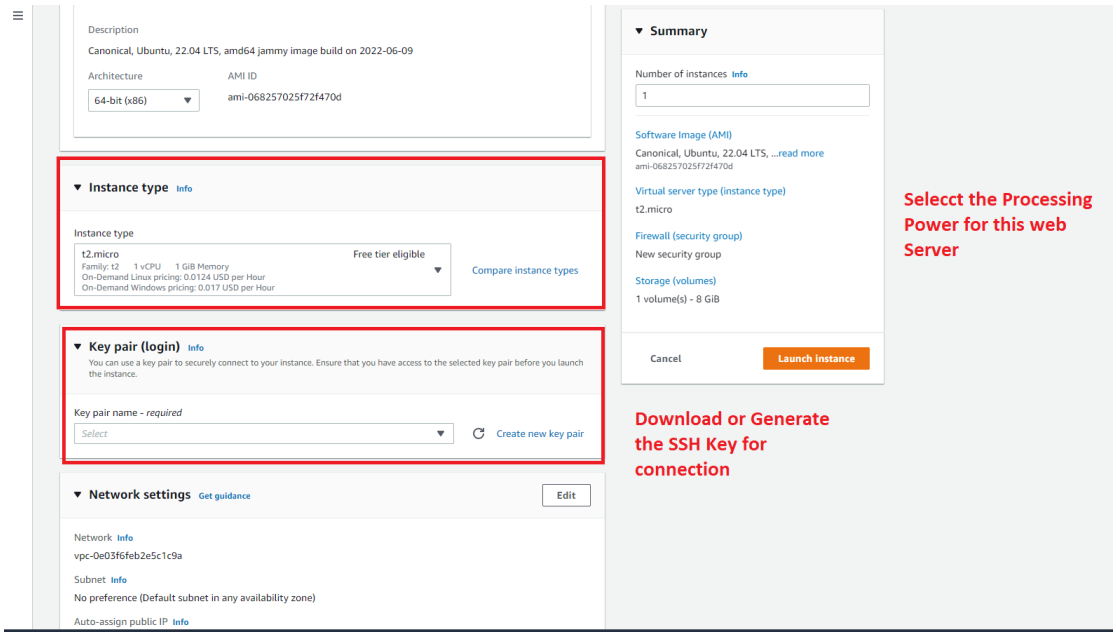


Figure 36: Define resources

5. In the networking section please enable the allow SSH key, HTTP for Internet Protocols and Connection.

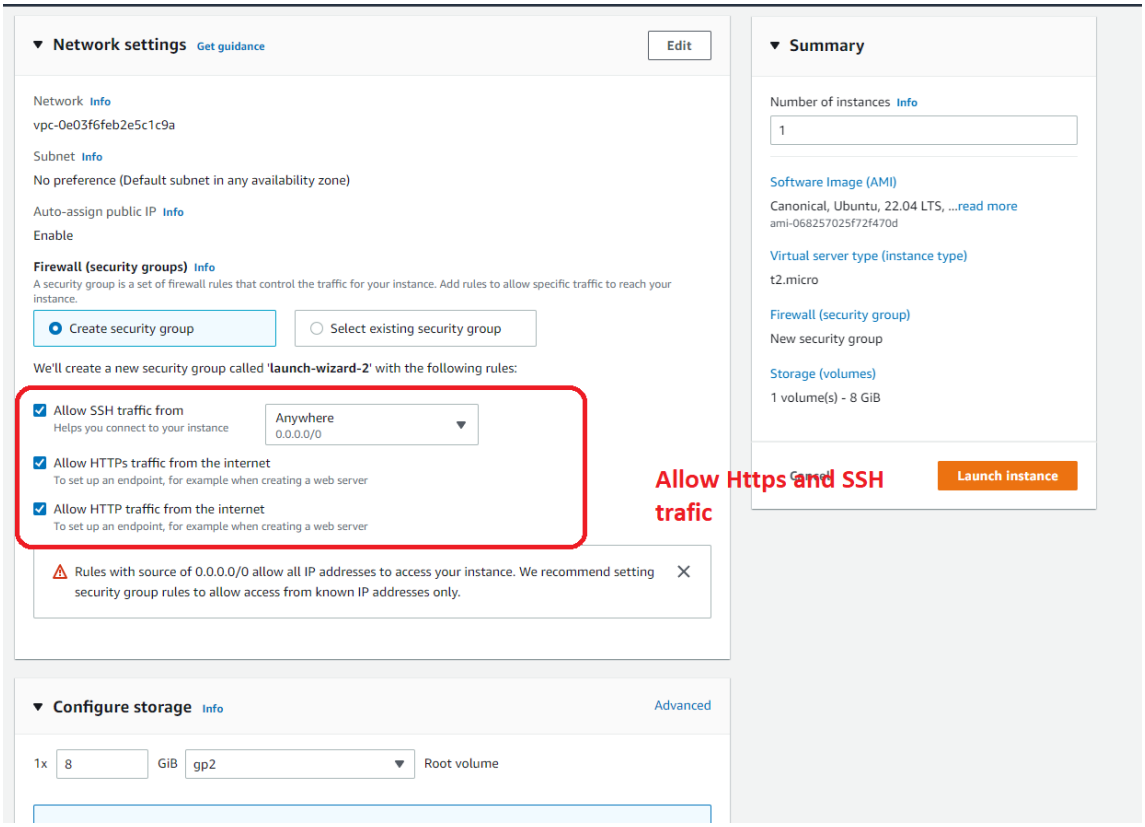


Figure 37: Setup networking

6. Change the storage but more storage will cost more for free tier there is only 30 GB

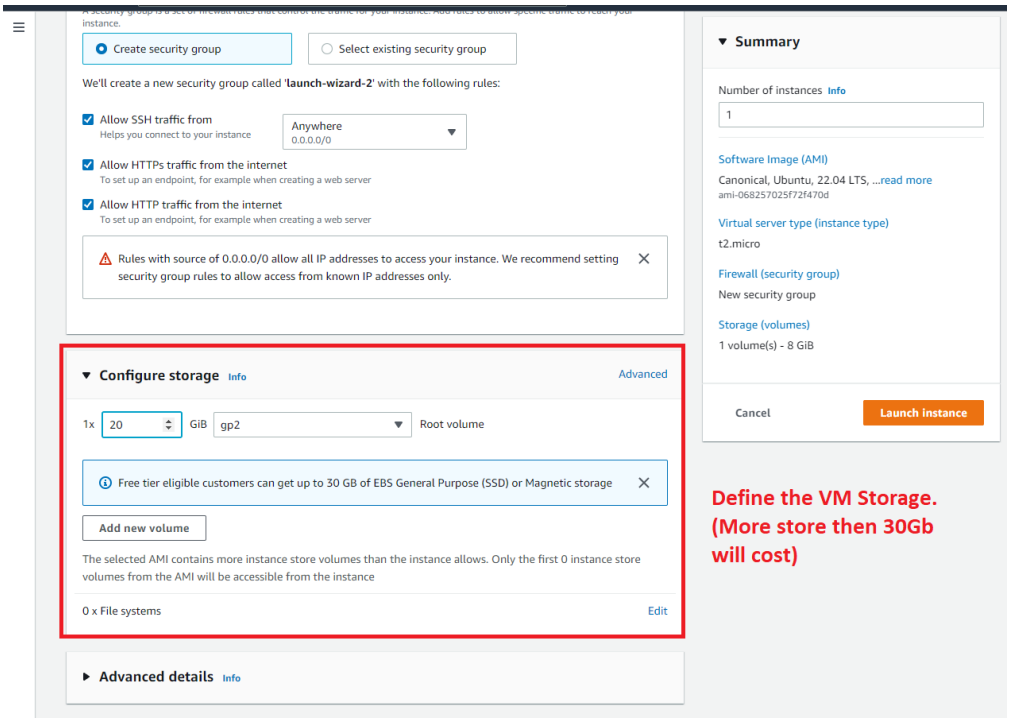


Figure 38: Allocating memory

7. Now click the Launch Button.

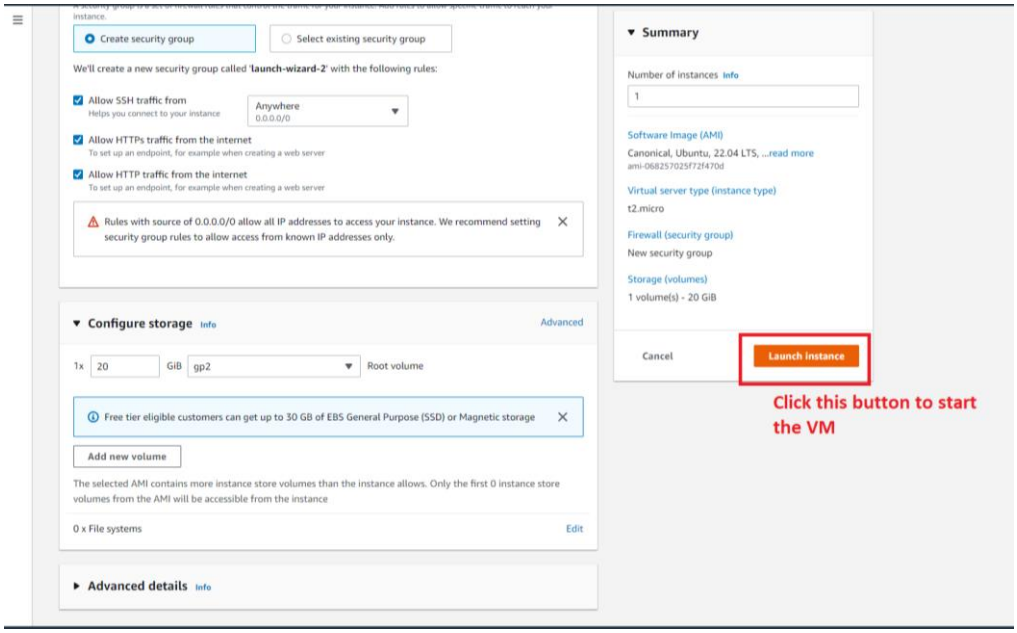


Figure 39: Launching application

Note: This process will take time and depending upon the internet connection it can take from 30 minutes to 40 minutes.

Now your VM is read now upload the files to this VM and your website will be deployed.

6.4.2. Uploading Coding to the website:

Now to upload the code to the VM but before upload we need a connection to the VM so for connection follow these steps:

1. Open the dashboard

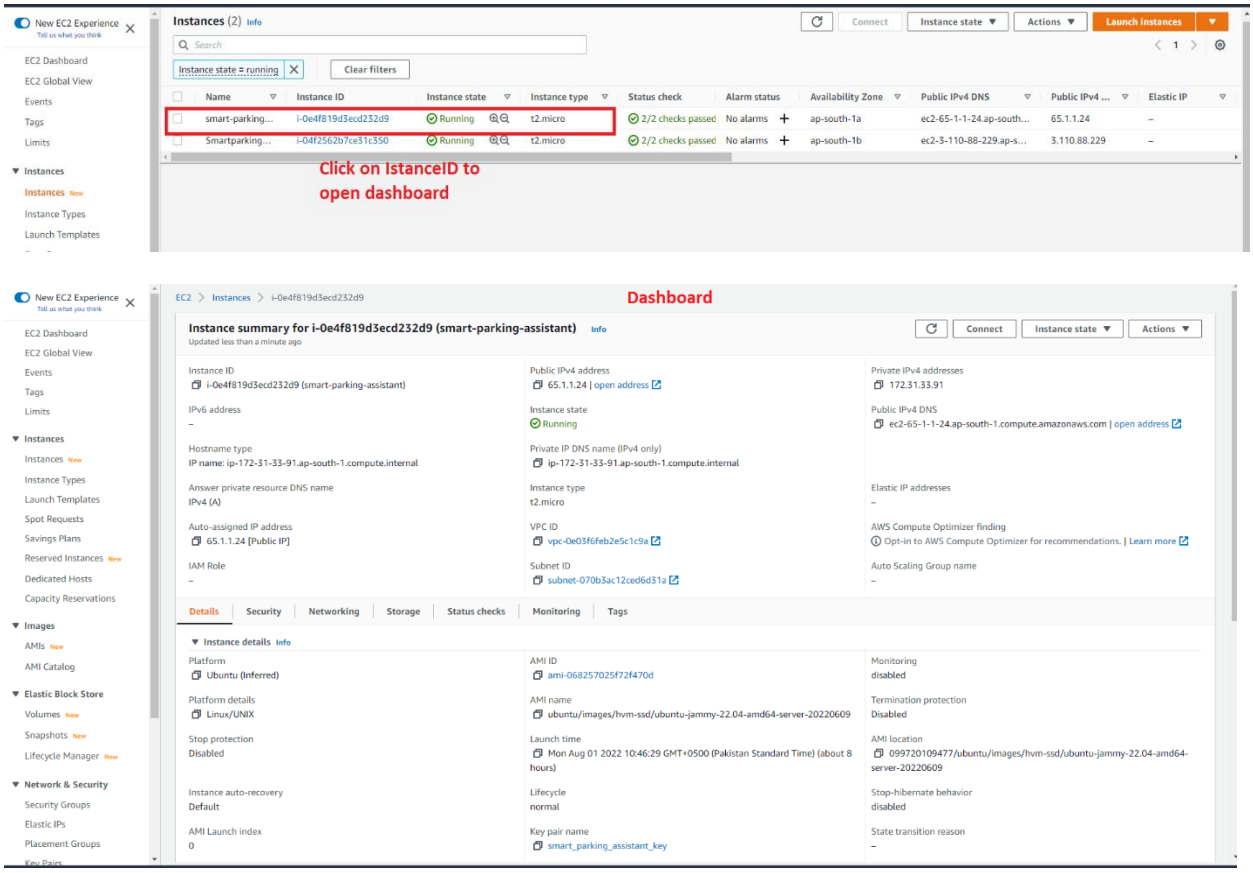


Figure 40: Uploading code

2. Click the connect button

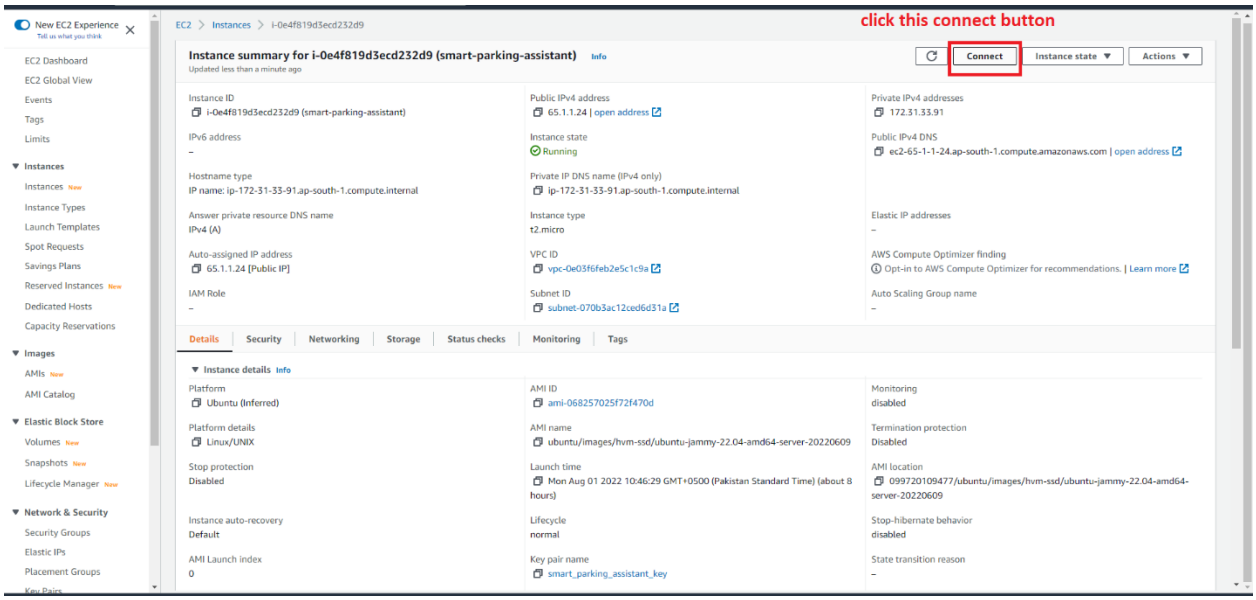


Figure 41: Setting connection

3. There are multiple options but for this we will be using the web terminal to interact with the VM. Then Press connect.

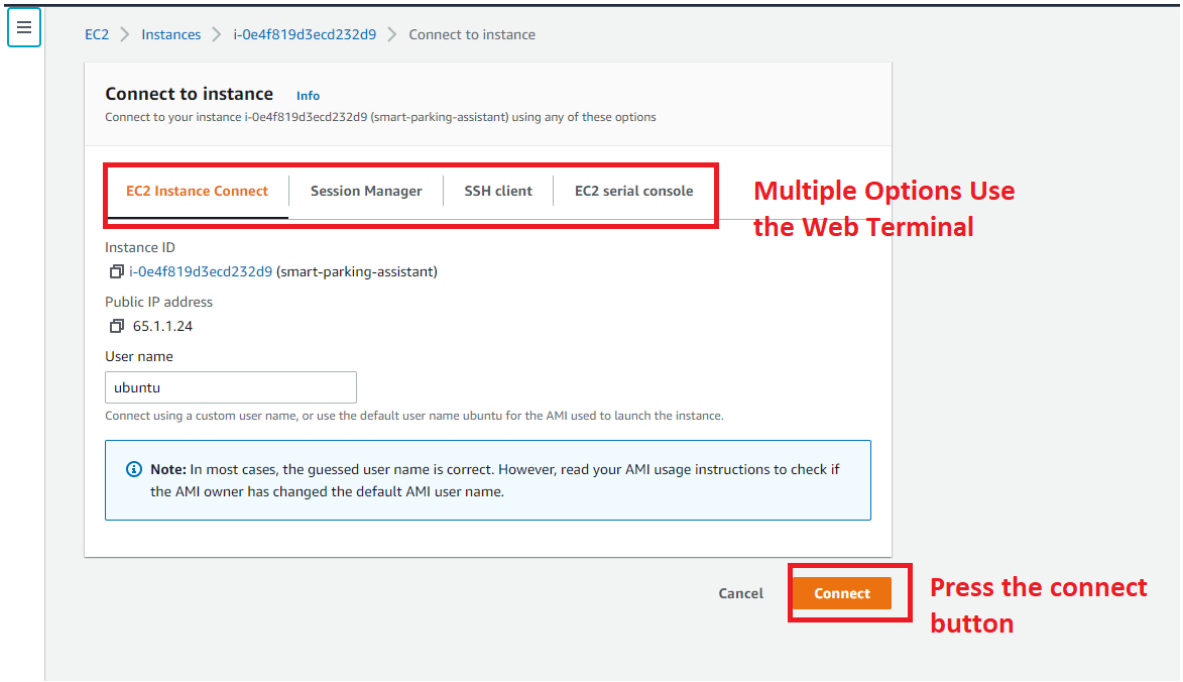


Figure 42: Connect VM Ware

4. Terminal will be opened in the new tab

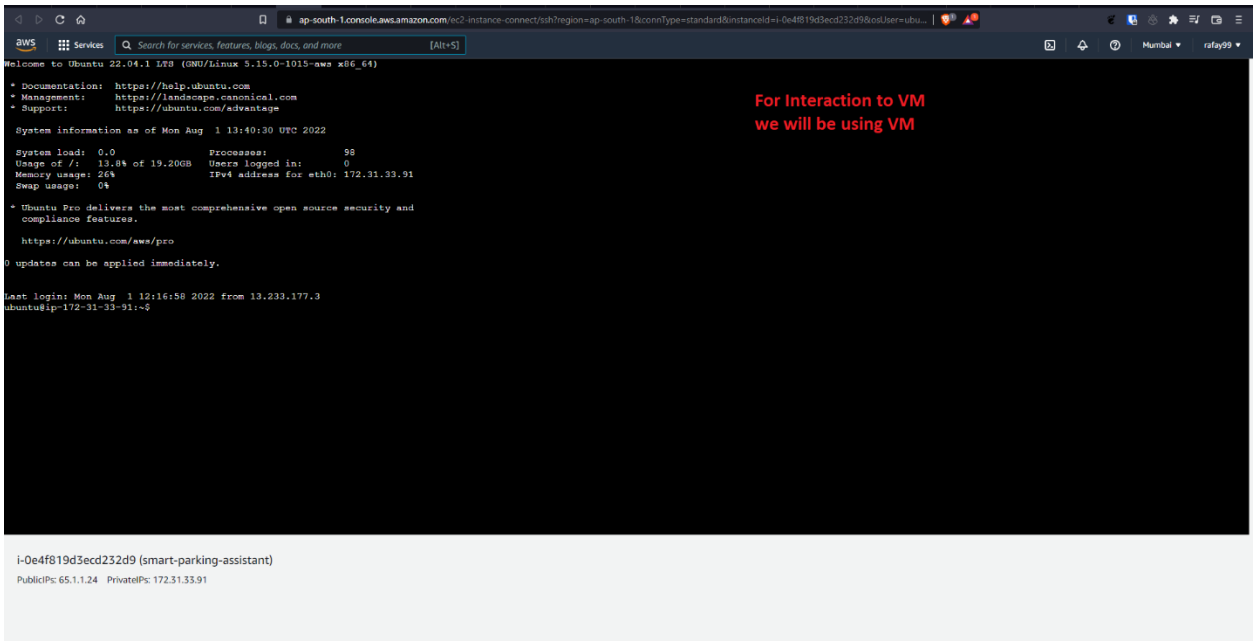


Figure 43: Setup terminal

5. Once the terminal is open then enter the following commands:

    - sudo apt update
    - sudo apt upgrade
    - sudo apt install git vim nodejs npm

6. Once the VM is update to date will all of the application Installed we need to download the files from GitHub. To download enter the following command.

    a. sudo git clone https://github.com/rafay99-epic/Parking-Assistant.git
    b. cd Parking-Assistant
    c. sudo npm install
    d. sudo npm start

7. Once all the commands are run, open to VM dashboard via URL and then access the application.
8. Application is now running.

Chapter 7

Project Evaluation

This chapter includes the examiners evaluation report, including the points to be revised/included along with the selected requirements in the next iteration.

7.1. Project Evaluation Report

Examiner Name:	
S. No.	**Suggestion**
1	Entity relation should be labeled.
2	Sequence diagram should be split.
3	English sentence structure should be improved.
4	Formatting should be improved.
5	Naming conventions are not followed in code artifacts.
6	Indention rules are not followed as described in chapter 4
7	Test cases for the incorrect behavior to be included.
8	Test cases should be added to test remaining functionalities.

Other Comments (If any):

_____
_____
_____

_____

Signature

References:

1. https://www.npmjs.com/package/bootstrap

2. https://www.npmjs.com/package/node

3. https://www.npmjs.com/package/handlebars

4. https://www.npmjs.com/package/socket.io

5.https://www.researchgate.net/publication/340487448_Smart_Parking_Assistance_Services_and_User_Acceptance_A_European_Model

6. https://www.researchgate.net/publication/224752535_3D_parking_assistant_system

7. https://www.geeksforgeeks.org/javascript/