TCSS 559 SERVICE COMPUTING

A PROJECT ON

CAR PARKING MANAGEMENT USING RFID

UNDER THE GUIDANCE OF

Eyhab Al-Masri, Assistant Professor

BY

Sri Vibhu Paruchuri

Ashwin Meena Meiyappan

Prathyush B Vuppala

TABLE OF CONTENTS

11		

ABSTRACT

Quick Park is a practical, streamlined solution to parking management. It aims to improve upon the shortcomings of the existing parking management systems, which are labor-intensive. Quick Park is an intelligent system that improves the user experience at the end-user level as well as the administrator level. The system comprises multiple web services that automate the process of parking, payment, and reservation of parking spots. The system also analyzes data in real time at the administrator level and displays the data in an easy-to-consume format. The system achieves a high level of automation by using RFID tags and readers at the entry and exit locations of the parking garage and at the individual parking spots. The various web services act on the data generated by these readers.

INTRODUCTION

The existing parking management systems require a crew to handle the day-to-day operations. Most of the tasks performed by the crew are trivial and can be automated. For parking garages with a crew at the entry/exit spots, they oversee handling the payment and opening/closing of gates. This process can easily be automated by using RFID tags on the car and readers at the entry/exit locations. When the readers detect the tags, the barriers can be opened/closed, and the payment can be processed based on the interval between the entry and exit time. This automation results in a reduction of operational costs, potentially saving millions of dollars depending on the scale of the operations.

In some other cases, there are unmanned stations where the user is expected to manually input the entry time and expected time and insert a credit card to pay for the service. An automated system would handle all these tasks with minimal human intervention resulting in a much better end-user-level experience.

Another issue with parking management systems is that users struggle to find free spots in busy parking lots, often resulting in users driving around in rounds looking for a free spot.

Quick Park enhances user- experience by guiding the end-users towards free parking spots by sending directions to the nearest free parking spots based on their location. Also, users can check for free spots in advance using the Reservation service. Using reservation service users can reserve parking spots with a premium fee.

Business stakeholders would like to access the operational data in an easy-to-consume format to make better business decisions. There are many parking management systems that focus on this aspect. Quick Park will provide real-time data analysis of data in the form of charts so that administrators can make quick decisions to improve operational efficiency.

OBJECTIVE:

In response to the problem statement above, our goal is to build an automated parking management system that results in a better user experience for the end-user as it cuts down the time spent on parking, reservation, and payment. The system also analyzes data and presents it to the administrator in a concise format. The system has further benefits for businesses as it reduces operating costs over the long run due to minimal human labor.

SCOPE:

In its current state, QuickPark is limited to following functionalities:

- Charges: The user is expected to park for at least one hour.
- The latest time to reserve/enter the parking lot is 9 PM and must exit before 11:59 PM.
- Vehicle is deallocated (towed) if it doesn't show up within 2 hours of the reservation time.
- One future reservation per vehicle and vehicle cannot make the reservation for the same day.
- Online requests reservation (web service) are allocated every one hour.

EXISTENCE AND PROPOSED

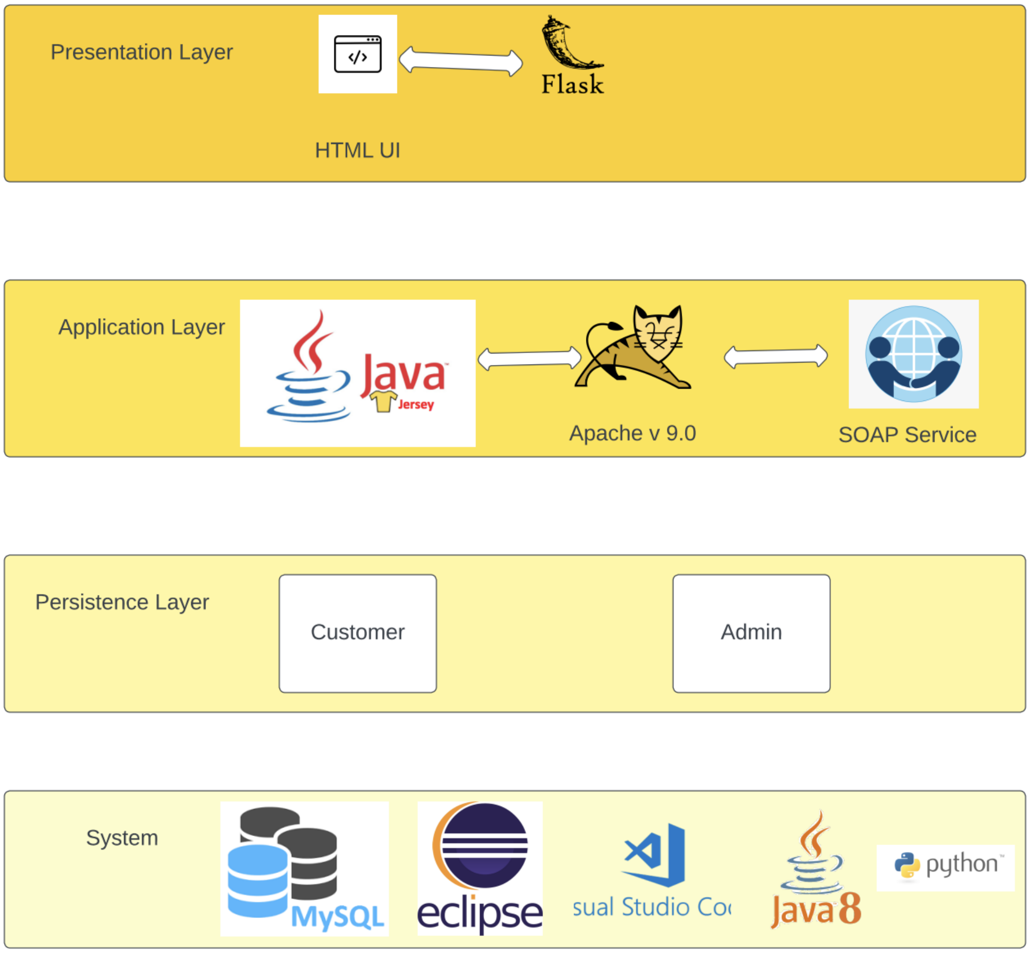
Does your idea already exist? How will yours be different? Yes, we have automated parking management systems. Quick Park has the following additional features:

Administration: The administrator will have access to a dashboard displaying real-time analytics of the parking lot. These insights will help future operational decisions.

Reservation: With Quick Park, we can book ahead for a parking spot if the user is within a mile.

Payment: With QuickPark, payment service is automated and users are charged as soon as they exit the parking lot.

Architecture:



Conclusion: Quick Park application is the Car Parking Management application using RFID , combination of REST and Non-Restful web services , XML and JSON data exchange with mysql database to store the data. QuickPark is a hands free fully automated solution for Car Parking Management to manage automated payments and insights about the vehicles parked in parking lot.

Contribution: All 3 of us jumped into zoom meetings for the past two weeks for an average of 6 hours/day and worked together like pair programming. It helped to progress in the project whenever we got stuck. We had constructive discussions to design and implement the application.

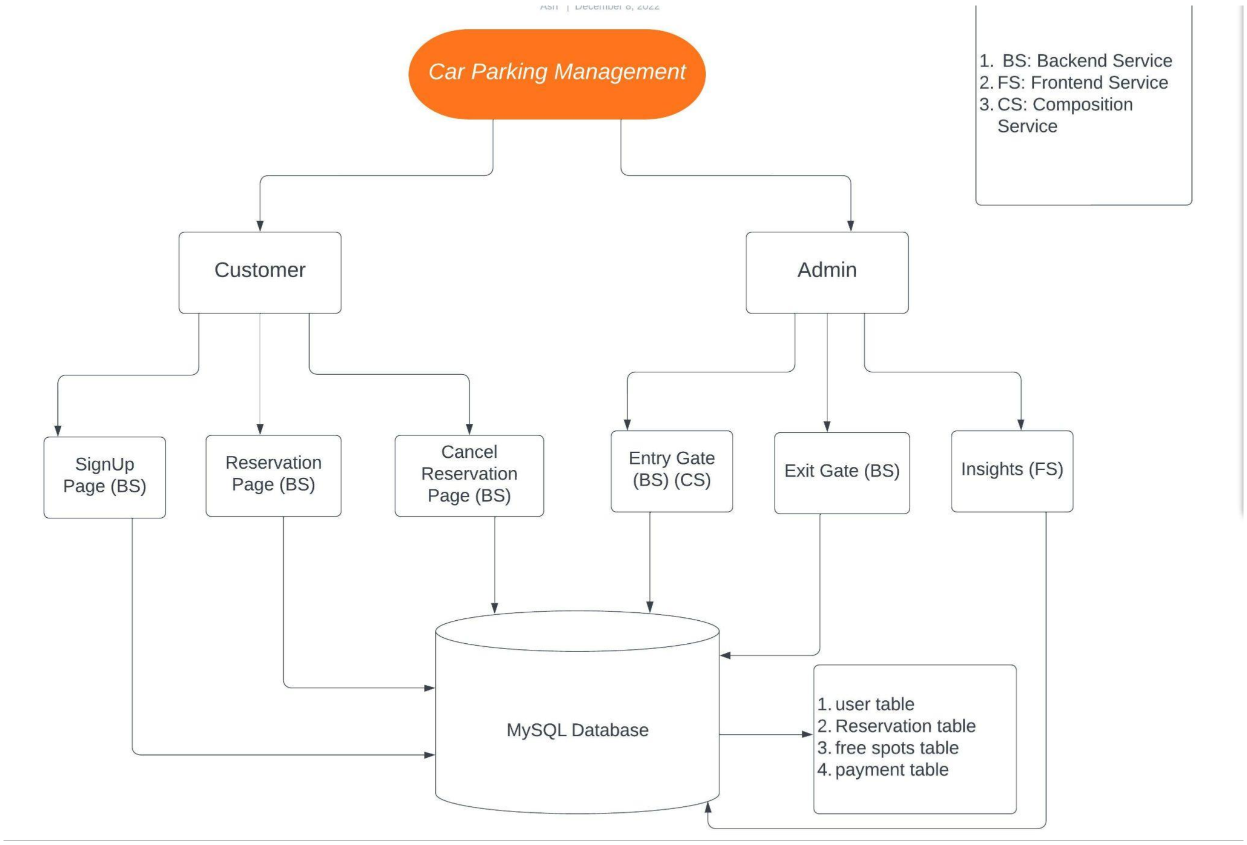
Future Work:

1. We can automate to run the scripts (noShow and bookSpotOnline) in the background.
2. Application could support managing multiple future reservations.
3. Processing entry/exit time and charging payments get more granular. Now the minimum parking time is 1 hour.
4. Application can have a front end or display that helps the user to navigate to the nearest free spot.

References:

1. https://flask.palletsprojects.com/en/2.2.x/#user-s-guide
2. https://www.digitalocean.com/community/tutorials/jersey-java-tutorial
3. Learn to style HTML using CSS - Learn web development | MDN
4. https://lucid.app/documents

Appendix A: Architectural design



High Level Design:

QuickPark: This is the high level overview design of Car Parking Management using RFID is the application with https://localhost:5000/ endpoint served using flask. It contains two tabs: Customer and Admin.

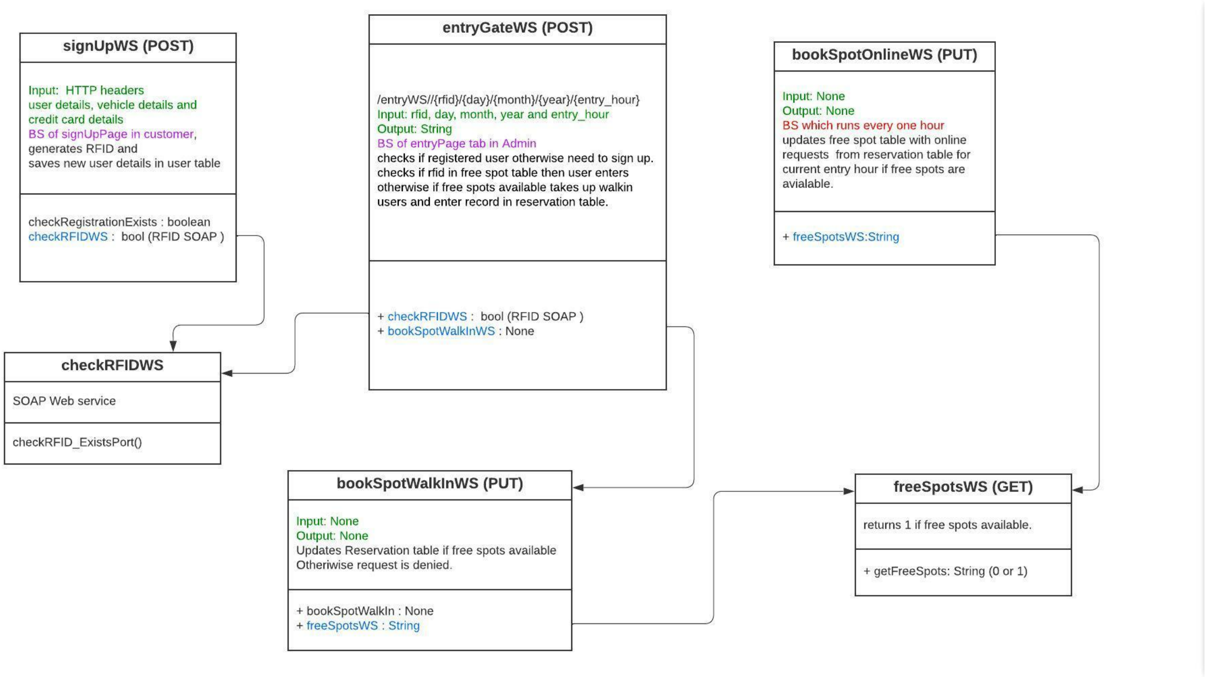
Customer: Customers can directly jump into this tab (https://localhost:5000/customer) if they would like to reserve (https://localhost:5000/reservation) or cancel previous reservations (https://localhost:5000/cancel). If a new user, then need to sign up https://localhost:5000/signup.

Admin: Admin tab (https://localhost:5000/admin1) can only be accessed by the management of QuickPark, hence needing to login and then redirects to (https://localhost:5000/admin2) which contains the following tabs. In order to simulate a real world scenario, we have an entry and exit gate as part of the Admin tab. Otherwise ideally we will have a sensor which detects the car near the entry gate and calls backend service with rfid details, entry/exit time etc.

Entry Gate: (https://localhost:5000/entry) Admin has to fill in the details like: rfid, entry date and entry hour when a user arrives at the entry gate.

Exit Gate: (https://localhost:5000/exit) Admin has to fill in details : rfid and exit hour when a user arrives at exit gate.

Insights tab: (https://localhost:5000/insights) Admin can have a look at the statistics like: currently occupied spots, total reservation requests, total walkin requests , revenue per rfid, denied requests.



signUpWS (POST)

Input: HTTP headers
user details, vehicle details and
credit card details
BS of signUpPage in customer,
generates RFID and
saves new user details in user table

checkRegistrationExists : boolean
checkRFIDWS : bool (RFID SOAP)

entryGateWS (POST)

/entryWS//{rfid}/{day}/{month}/{year}/{entry_hour}
Input: rfid, day, month, year and entry_hour
Output: String
BS of entryPage tab in Admin
checks if registered user otherwise need to sign up.
checks if rfid in free spot table then user enters
otherwise if free spots available takes up walkin
users and enter record in reservation table.

+ checkRFIDWS : bool (RFID SOAP)
+ bookSpotWalkInWS : None

bookSpotOnlineWS (PUT)

Input: None
Output: None
BS which runs every one hour
updates free spot table with online
requests from reservation table for
current entry hour if free spots are
avialable.

+ freeSpotsWS:String

checkRFIDWS

SOAP Web service

checkRFID_ExistsPort()

bookSpotWalkInWS (PUT)

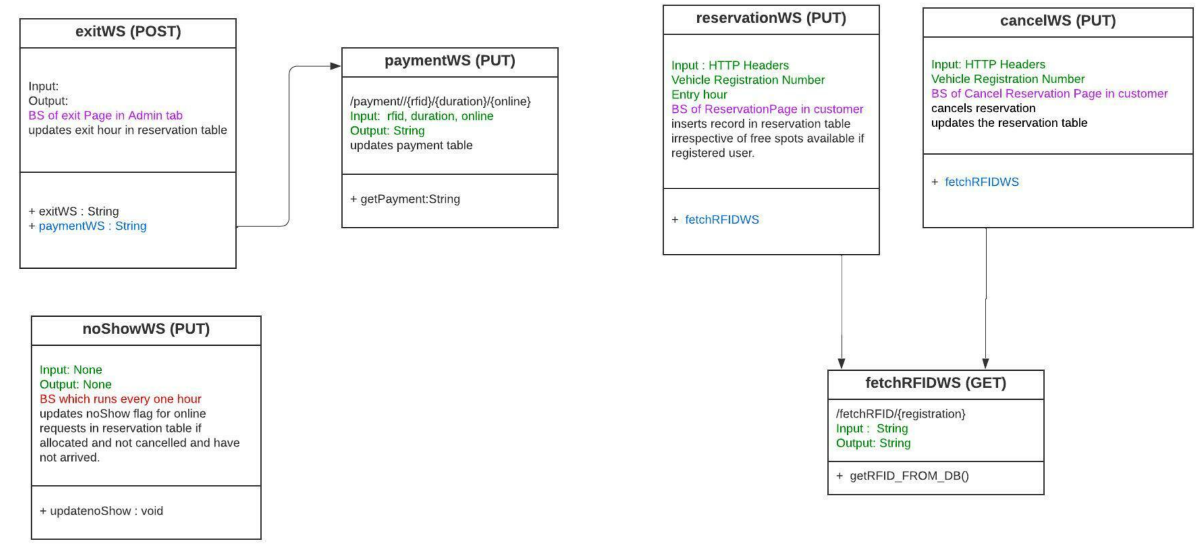
Input: None
Output: None
Updates Reservation table if free spots available
Otheriwise request is denied.

+ bookSpotWalkIn : None
+ freeSpotsWS : String

freeSpotsWS (GET)

returns 1 if free spots available.

+ getFreeSpots: String (0 or 1)



exitWS (POST)

Input:
Output:
BS of exit Page in Admin tab
updates exit hour in reservation table

+ exitWS : String
+ paymentWS : String

paymentWS (PUT)

/payment//{rfid}/{duration}/{online}
Input: rfid, duration, online
Output: String
updates payment table

+ getPayment:String

reservationWS (PUT)

Input : HTTP Headers
Vehicle Registration Number
Entry hour
BS of ReservationPage in customer
inserts record in reservation table
irrespective of free spots available if
registered user.

+ fetchRFIDWS

cancelWS (PUT)

Input: HTTP Headers
Vehicle Registration Number
BS of Cancel Reservation Page in customer
cancels reservation
updates the reservation table

+ fetchRFIDWS

noShowWS (PUT)

Input: None
Output: None
BS which runs every one hour
updates noShow flag for online
requests in reservation table if
allocated and not cancelled and have
not arrived.

+ updatenoShow : void

fetchRFIDWS (GET)

/fetchRFID/{registration}
Input : String
Output: String

+ getRFID_FROM_DB()

Low Level UML class diagram:

This UML diagram depicts the backend service developed using jersey framework used in the application. We have the following main APIs: signUpWS , reservationWS , cancelWS , entryGateWS and exitWS which also use other services.

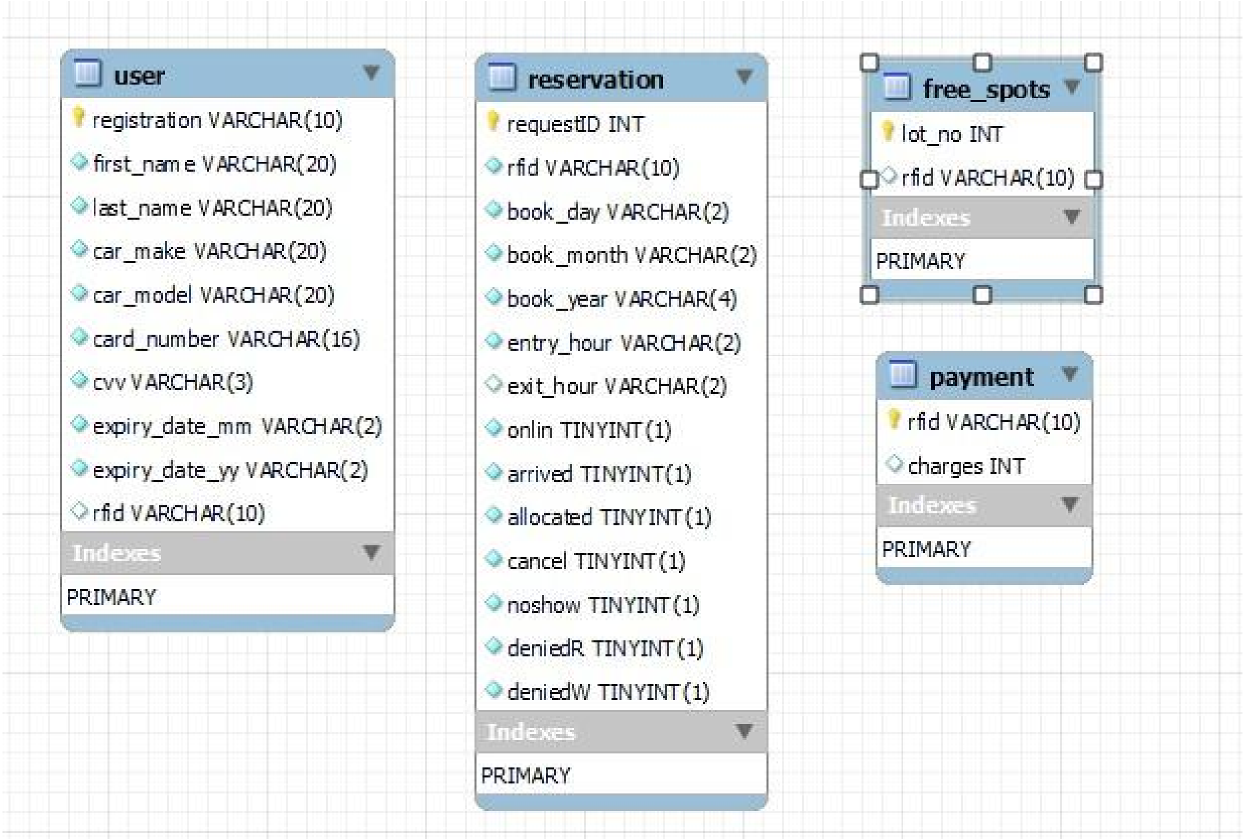
Composite service:
entryGateWS communicates with **chechRI DWS** web service and **bookSpotWalkInWS**.

XML data exchange:
chehRA IDWS : It is a SOAP web service and uses XML to communicate with other web services. Other web services are REST services and communicate using JSON.

noShowWS and bookSpotOnlineWS are web services which run every one hour as background jobs, which update the reservation table and free spots table respectively.

Appendix B: Database Design and Sample Data

Schema:



user
- registration VARCHAR(10)
- first_name VARCHAR(20)
- last_name VARCHAR(20)
- car_make VARCHAR(20)
- car_model VARCHAR(20)
- card_number VARCHAR(16)
- cvv VARCHAR(3)
- expiry_date_mm VARCHAR(2)
- expiry_date_yy VARCHAR(2)
- rfid VARCHAR(10)

Indexes
PRIMARY

reservation
- requestID INT
- rfid VARCHAR(10)
- book_day VARCHAR(2)
- book_month VARCHAR(2)
- book_year VARCHAR(4)
- entry_hour VARCHAR(2)
- exit_hour VARCHAR(2)
- onlin TINYINT(1)
- arrived TINYINT(1)
- allocated TINYINT(1)
- cancel TINYINT(1)
- noshow TINYINT(1)
- deniedR TINYINT(1)
- deniedW TINYINT(1)

Indexes
PRIMARY

free_spots
- lot_no INT
- rfid VARCHAR(10)

Indexes
PRIMARY

payment
- rfid VARCHAR(10)
- charges INT

Indexes
PRIMARY

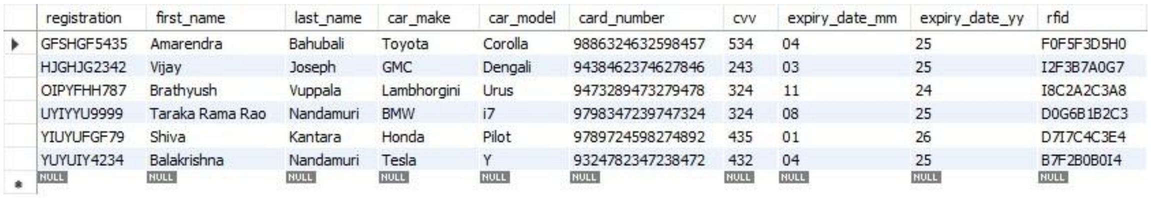
Description:

The QuickPark project utilizes 4 tables in the car parking MYSQL database. The first table is the user table. This table has all the necessary user details. The vehicle registration is the unique key (PRIMARY).

Reservation tables store all the reservation requests. It has several columns to keep track of the reservation details. This is useful for the data visualization performed in the insights page of the Administrative section in the QuickParksystem. The primary key in this table is the requestID. The free_spots table keeps track of the current status of the parking lots in the parking lot managed by the QuickPark system. The lot_no is the primary key. The final table is the payment table which keeps track of the total revenue generated by each vehicle. This is tracked based on the rfid of each vehicle. The primary key in this table is the rfid.

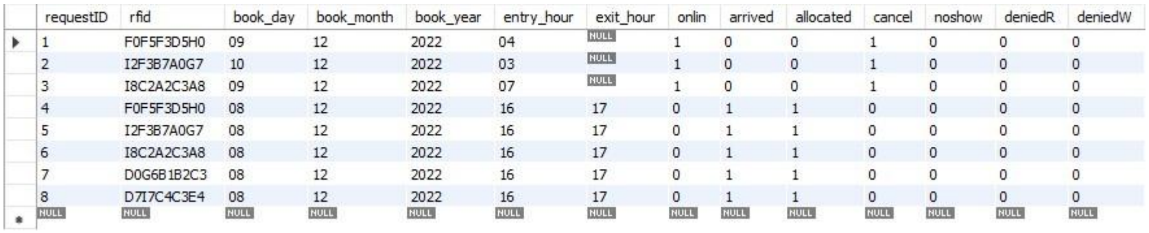
Screenshots:

1) **User Table**



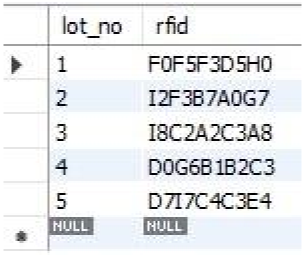
registration	first_name	last_name	car_make	car_model	card_number	cvv	expiry_date_mm	expiry_date_yy	rfid
GFSHGF5435	Amarendra	Bahubali	Toyota	Corolla	9886324632598457	534	04	25	F0F5F3D5H0
HJGHJG2342	Vijay	Joseph	GMC	Dengali	9438462374627846	243	03	25	I2F3B7A0G7
OIPYFHH787	Brathyush	Vuppala	Lambhorgini	Urus	9473289473279478	324	11	24	I8C2A2C3A8
UYIYYU9999	Taraka Rama Rao	Nandamuri	BMW	i7	9798347239747324	324	08	25	D0G6B1B2C3
YIUYUFGF79	Shiva	Kantara	Honda	Pilot	9789724598274892	435	01	26	D7I7C4C3E4
YUYUIY4234	Balakrishna	Nandamuri	Tesla	Y	9324782347238472	432	04	25	B7F2B0B0I4
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

2) **Reservation Table**



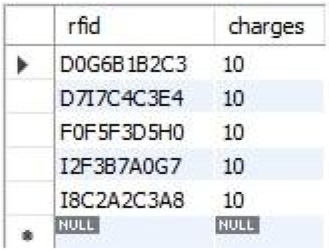
requestID	rfid	book_day	book_month	book_year	entry_hour	exit_hour	onlin	arrived	allocated	cancel	noshow	deniedR	deniedW
1	F0F5F3D5H0	09	12	2022	04	NULL	1	0	0	1	0	0	0
2	I2F3B7A0G7	10	12	2022	03	NULL	1	0	0	1	0	0	0
3	I8C2A2C3A8	09	12	2022	07	NULL	1	0	0	1	0	0	0
4	F0F5F3D5H0	08	12	2022	16	17	0	1	1	0	0	0	0
5	I2F3B7A0G7	08	12	2022	16	17	0	1	1	0	0	0	0
6	I8C2A2C3A8	08	12	2022	16	17	0	1	1	0	0	0	0
7	D0G6B1B2C3	08	12	2022	16	17	0	1	1	0	0	0	0
8	D7I7C4C3E4	08	12	2022	16	17	0	1	1	0	0	0	0
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

3) **Free Spots Table**



lot_no	rfid
1	F0F5F3D5H0
2	I2F3B7A0G7
3	I8C2A2C3A8
4	D0G6B1B2C3
5	D7I7C4C3E4
NULL	NULL

4) **Payments Table**



	rfid	charges
▶	D0G6B1B2C3	10
	D7I7C4C3E4	10
	F0F5F3D5H0	10
	I2F3B7A0G7	10
	I8C2A2C3A8	10
●	NULL	NULL

Appendix C: Web API Design

SignUp WS: This web service gets the user information from where customers enter their information in the SignUp page and then stores it into the user table.

Reservation WS: A web service, where people can enter their VRN (vehicle registration number) to reserve a parking spot.

Cancel WS: A web service, where people can cancel their reservation made before.

BookSpotOnline WS: A web service, in which people can reserve a spot on the go by accessing the URL and entering their details.

BookSpotWalkin WS: A web service, calls when people reserve a spot when they are at the entry gate of the parking area.

EntryGate WS: This service is performed at the entry gate of the parking spot where this web service checks if the VRN number exists in the user table whether the customer is registered or not.

- **Case 1 Online:** If the person is already registered and reserved a spot, entry gate will scan his RFID to enter the parking spot. There is other case where when they try to reserve using their VRN number. So, when he enters the entry gate, it will check scan the RFID is available in the reservation lists. If it is there, it will allow the vehicle into the parking spot. Another case where if he/she enters the entry gate after reservation, when the entry gate scans RFID and if there are no free spots available to park, the entry gate will deny the person to park his/her vehicle.
- **Case 2 Walkin:** If the person is already registered but not reserved a spot, the entry gate will scan the VRN to get the RFID and then it will add him to the reservation list. Then if there are free spots, it will allow the person inside to park the vehicle. If not, to enter the parking spot. There is another case where when they try to reserve using their VRN number, it may say parking spots are full. So, he can't enter the entry gate to park his vehicle.
- **Case 3 Sign Up:** If a person directly enters at the entry gate without signing up for the parking spot, the entry gate will ask to enter the details in the SignUp page and then it will call the BookSpotWalkin web service.

CheckRFID WS: It is a backend SOAP service which is called by backend web services to check the RFID whether it exists in the database.

ExitGate WS: A web service that implements the exit gate operations where it takes the exit time and RFID from the exit vehicle and calls the payment webservice to keep track of the revenue.

FetchRFID WS: It is called by backend services to get the RFID using VRN.

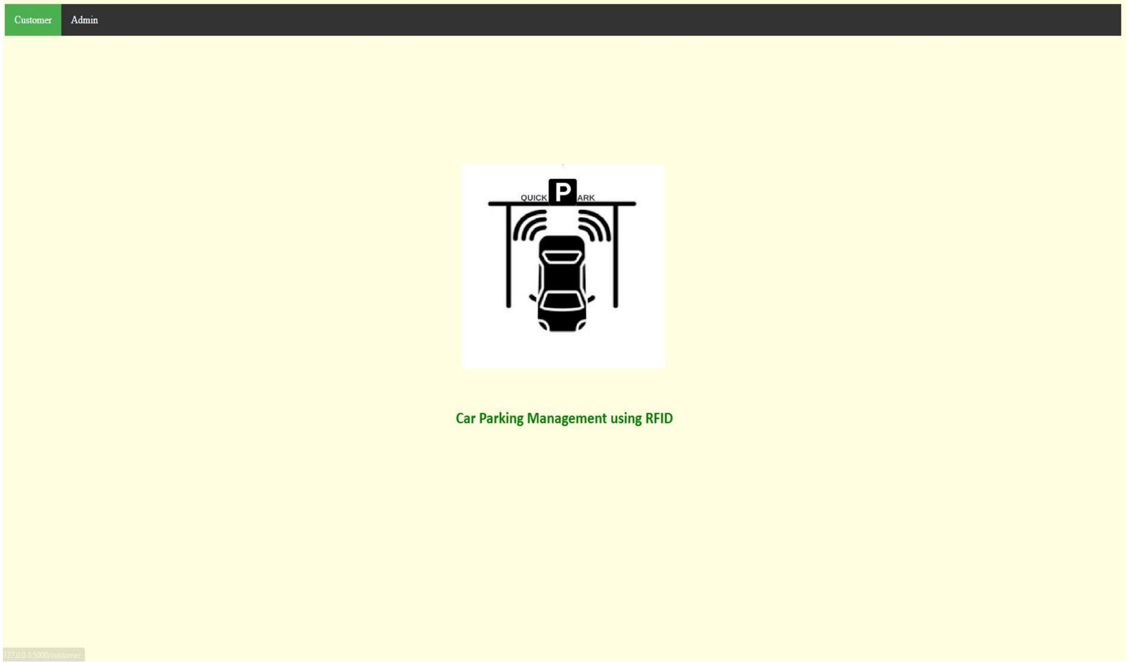
FreeSpots WS: The web service that checks if a free spot is available in the parking lot. Return 1 if a spot is available, otherwise 0.

NoShow WS: The web service that free up spots by removing de-allocating RFIDs of vehicles that reserved spots online but didn't show up for more than 2 hours.

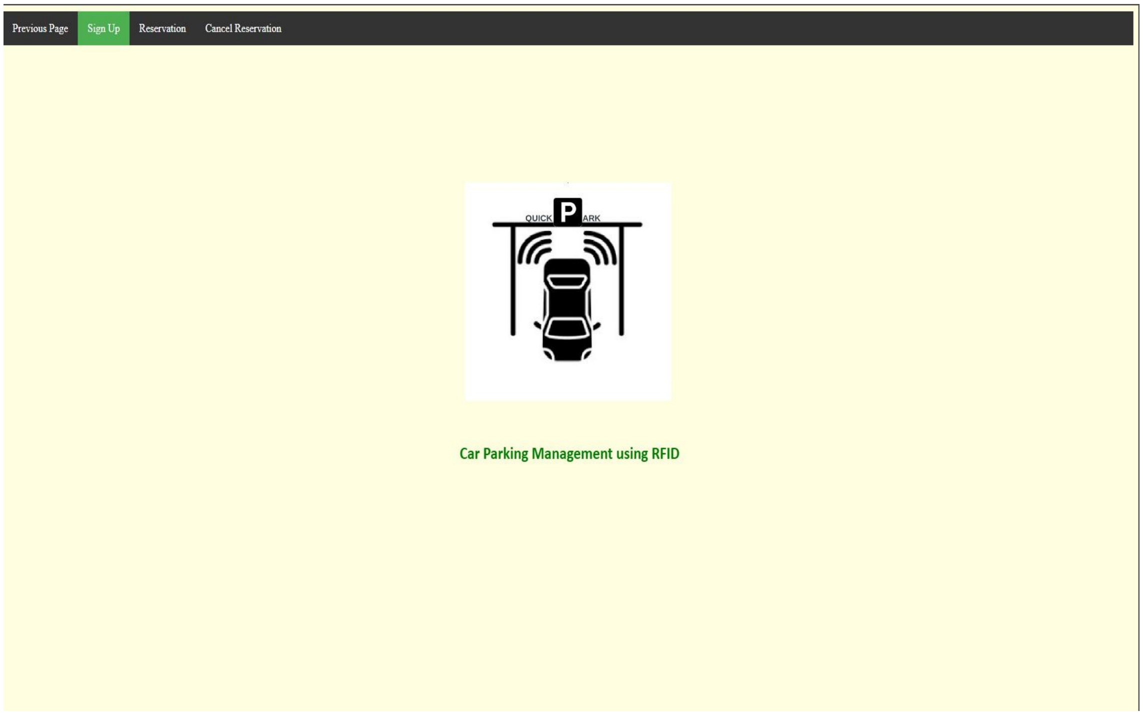
Payment WS: The web service that takes care of payments. Online and Walk-in charges are different.

Insights: Insights is not a web service instead it runs on a python flask framework where it has few methods in the backend, the role of insights is to represent the Quick Park data in various visualizations in the front-end for the admins to track for future purposes.

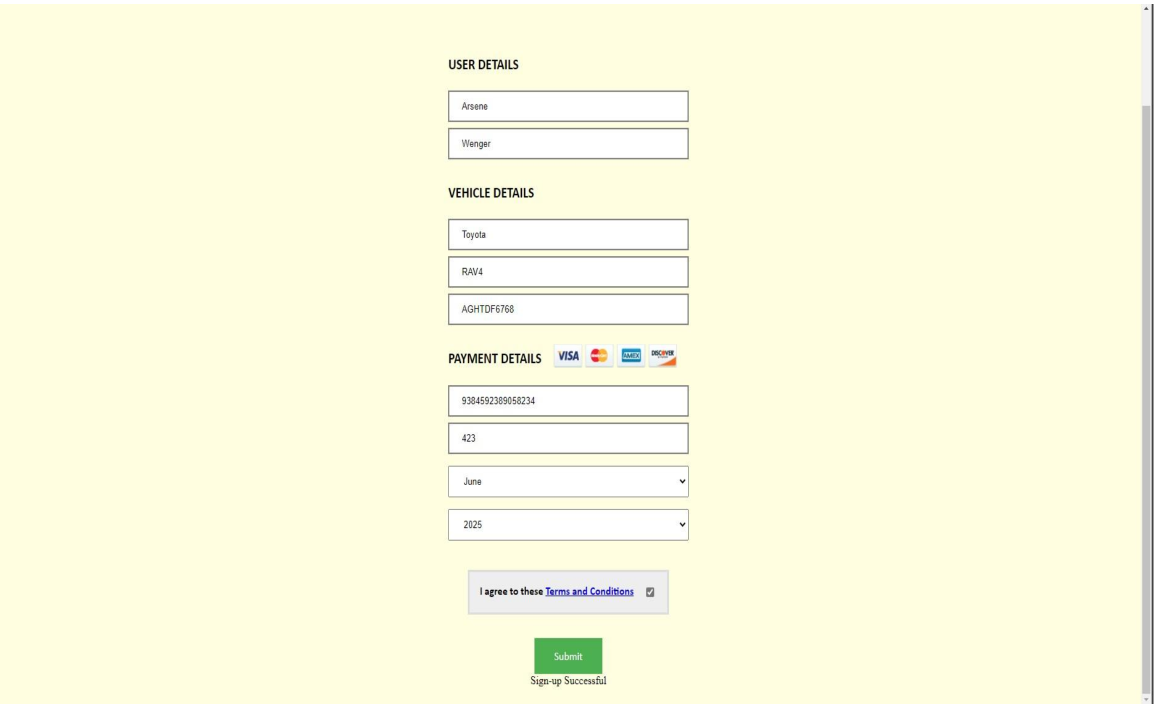
Appendix D: Front End screenshots



Car Parking Management using RFID



Car Parking Management using RFID



USER DETAILS

Arsene

Wenger

VEHICLE DETAILS

Toyota

RAV4

AGHTDF6768

PAYMENT DETAILS

9384592389058234

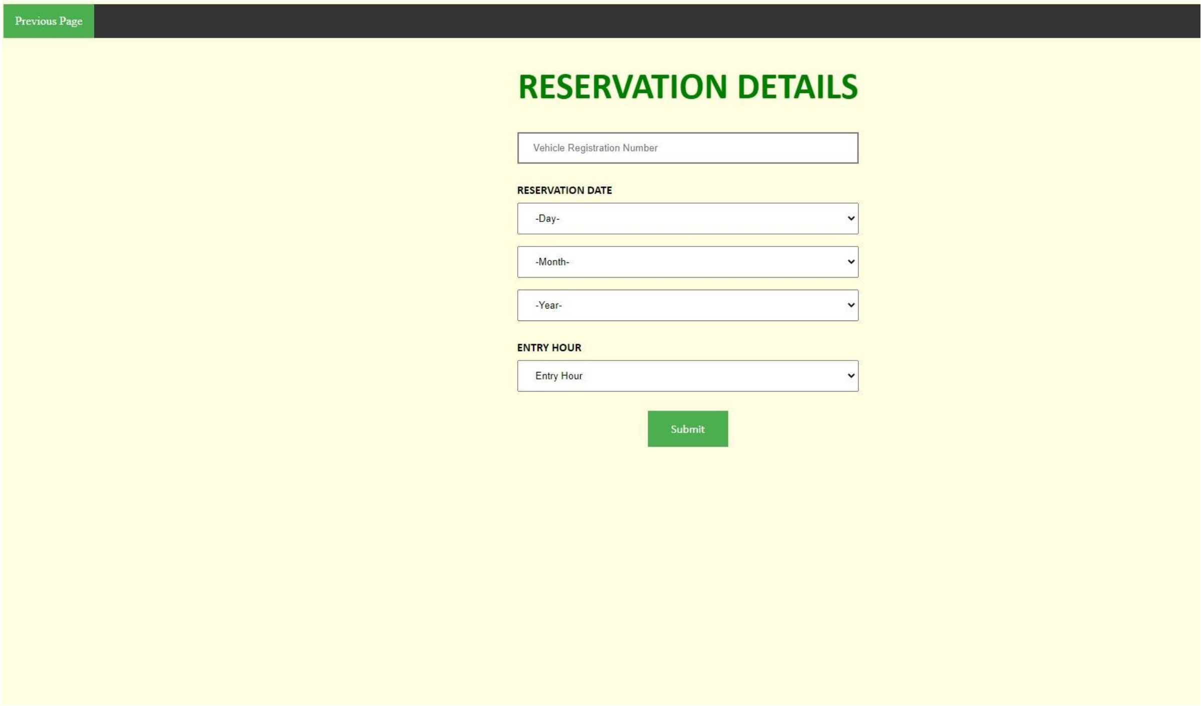
423

June

2025

I agree to these Terms and Conditions ☑

Sign-up Successful



RESERVATION DETAILS

Vehicle Registration Number

RESERVATION DATE

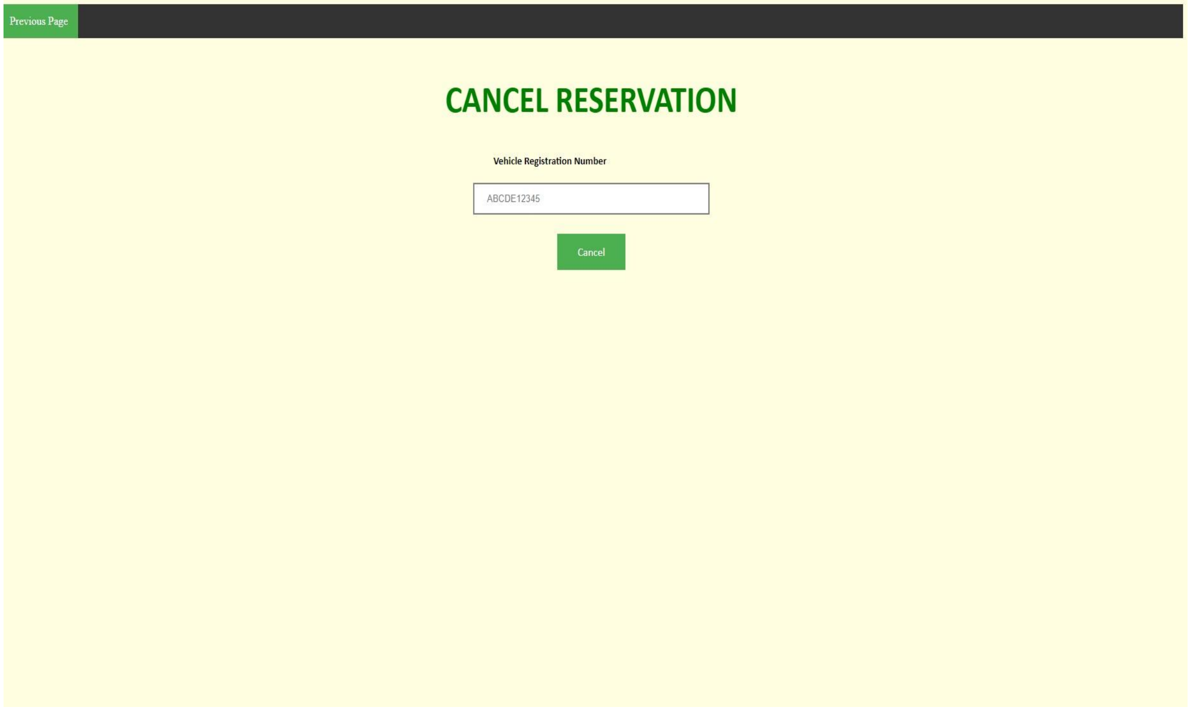
-Day-

-Month-

-Year-

ENTRY HOUR

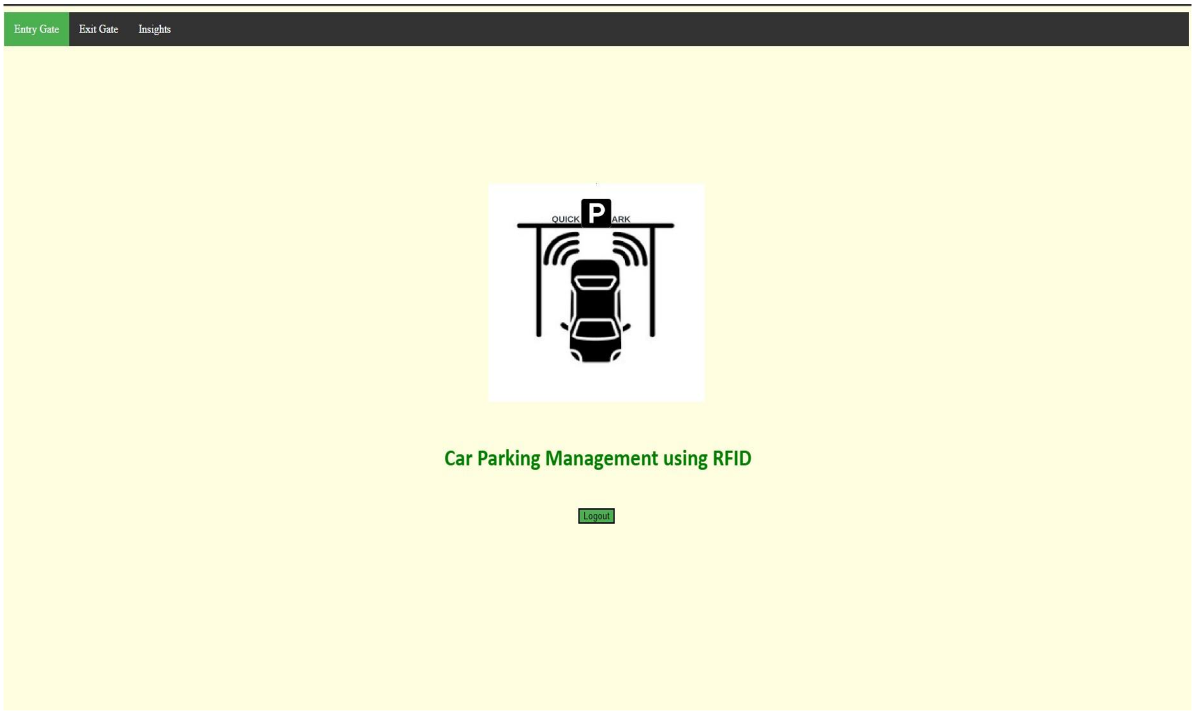
Entry Hour



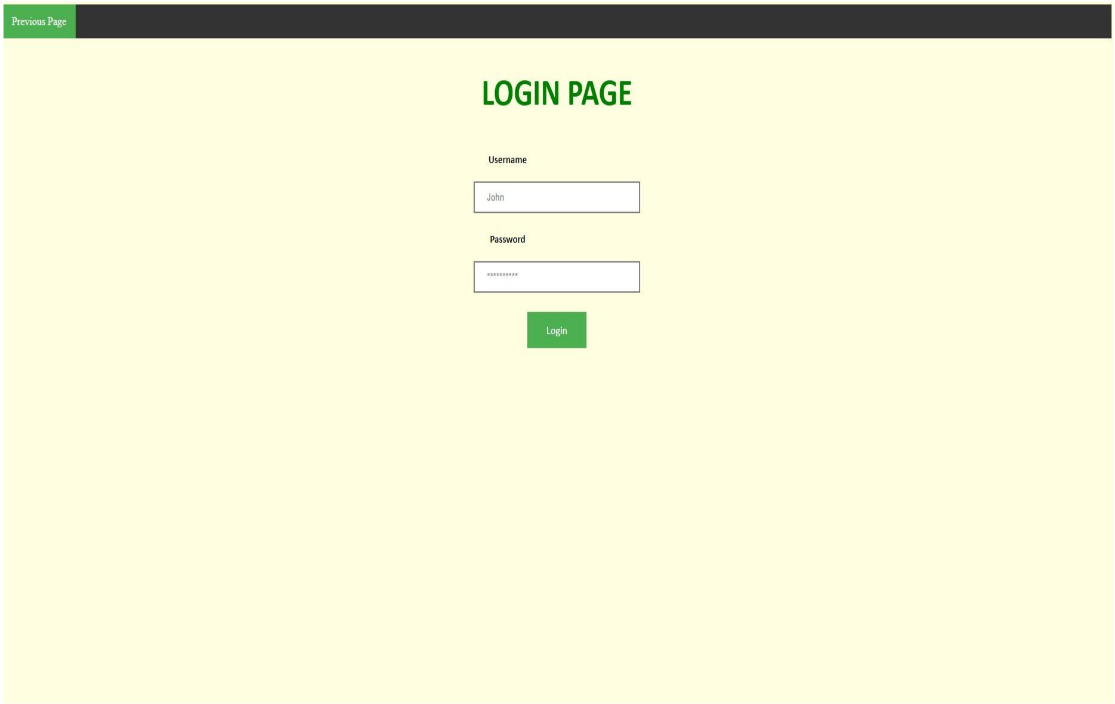
CANCEL RESERVATION

Vehicle Registration Number

ABCDE12345



Car Parking Management using RFID



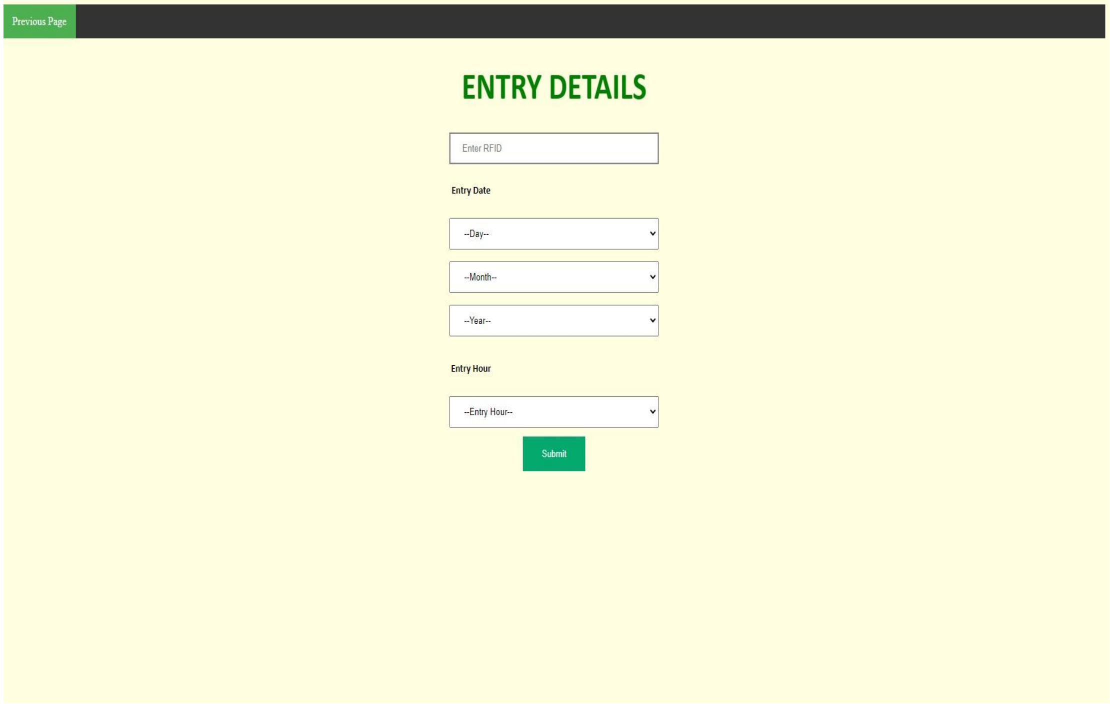
LOGIN PAGE

Username

John

Password

****



ENTRY DETAILS

Enter RFID

Entry Date

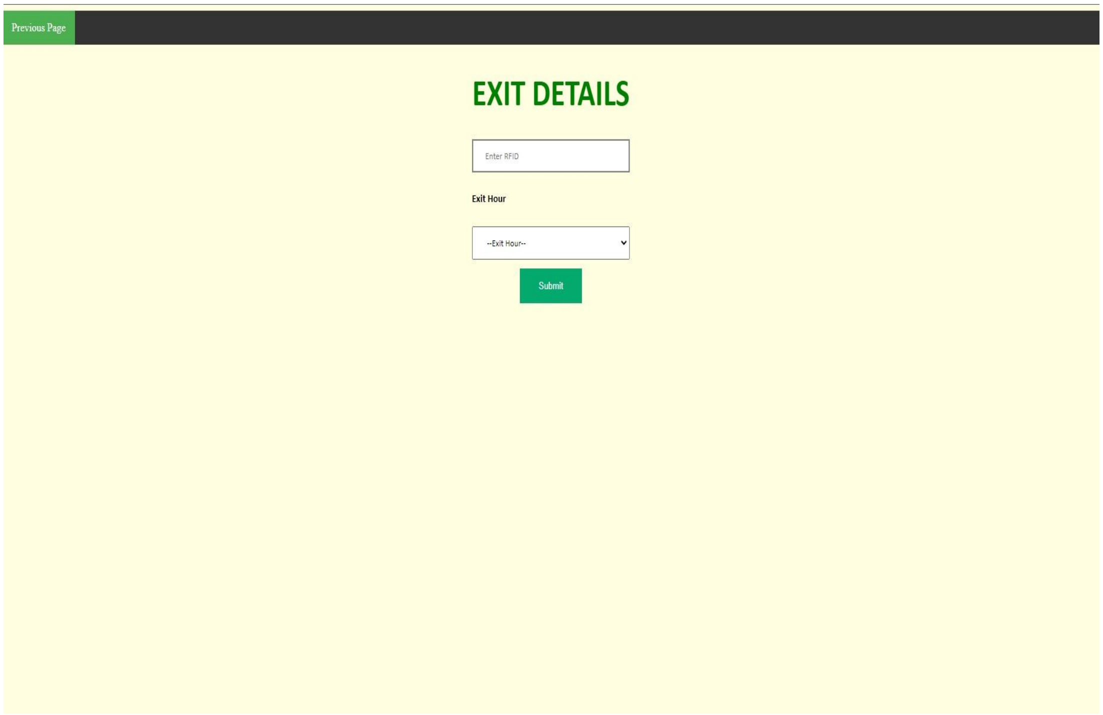
--Day--

--Month--

--Year--

Entry Hour

--Entry Hour--

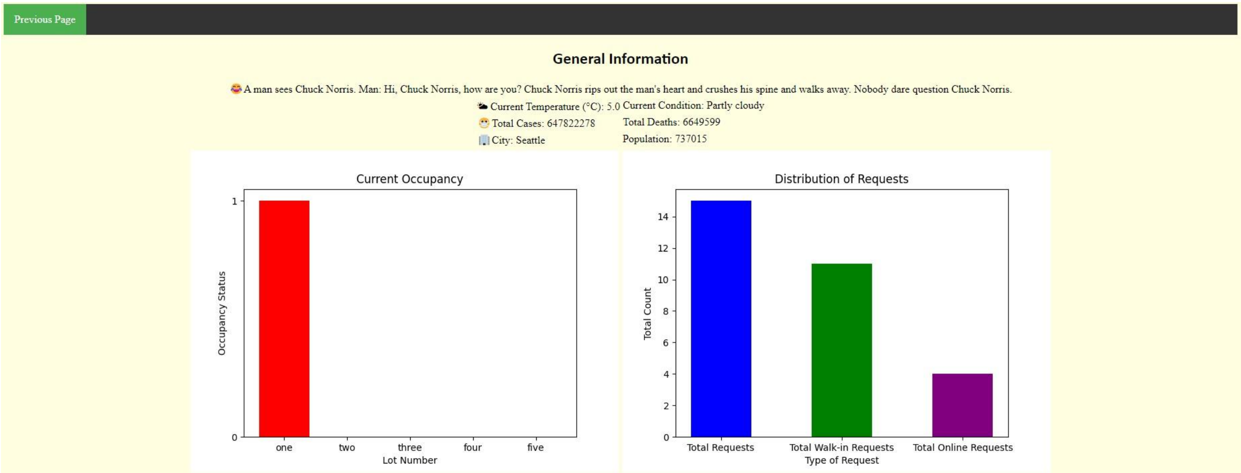


EXIT DETAILS

Enter RFID

Exit Hour

--Exit Hour--



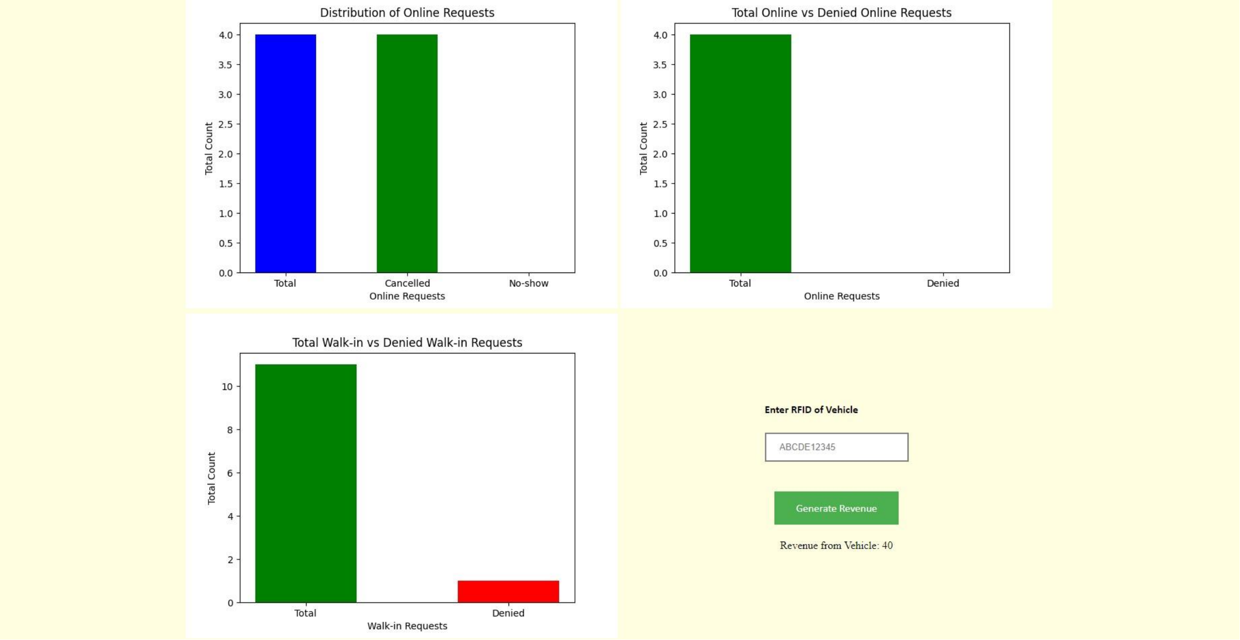
General Information

😂A man sees Chuck Norris. Man: Hi, Chuck Norris, how are you? Chuck Norris rips out the man's heart and crushes his spine and walks away. Nobody dare question Chuck Norris.

🌥 Current Temperature (°C): 5.0 Current Condition: Partly cloudy

😷 Total Cases: 647822278 Total Deaths: 6649599

🏢 City: Seattle Population: 737015



Enter RFID of Vehicle

ABCDE12345

Revenue from Vehicle: 40