PARKING MANAGEMENT SYSTEM

Dinesh Bahadur Kunwar, Aman Chaudhary, Aakash Shahi, Aayush Shrestha

B.Sc. (Hons.) Computer Softwarica College of IT and E-commerce, Coventry University

ST4008CEM: Computing Activity Led Learning Project 1

Giriraj Rawat

Aug 10, 2023

Table of Contents

TABLE OF FIGURES

PARKING MANAGEMENT SYSTEM

Introduction

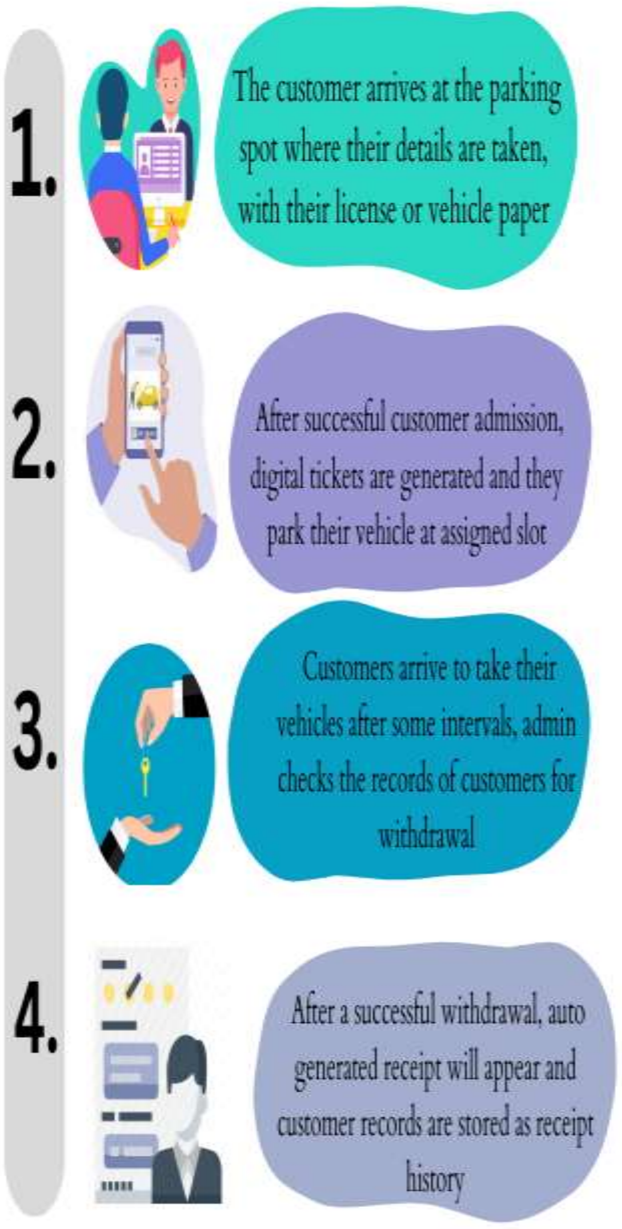
A parking management system is an innovative software solution made to automate and improve parking operations, whether they are for public lots or private businesses. The latest technology is used by this system to efficiently handle parking operations, offering both park operators and customers an effortless and user-friendly experience. Several advantages provided by this system completely change the parking experience. First, by simplifying processes like obtaining tickets, transactions, and area allocation, it boosts operational effectiveness while lowering the need for human intervention and boosting the use of space. Additionally, the structure's data-driven conclusions help informing decision-making, which leads to better resource allocation and planning.

Parking management systems frequently provide clients with practical features like security, fast payments, and many more. A system for handling parking paves the way for an improved and customer-focused parked environment because of its scalability and adaptation to different automotive situations. Parking management systems assist to reduce fuel consumption and carbon emissions through bettering the movement of traffic and decreasing the sum of time wasted attempting to find parking. This promotes environmental sustainability***(Byrne, 2023).***

This system uses the CRUD method (Create-Read-Update-Delete) on the databases. This helps to save the data of the admin and users which is needed for the records.

Figure 1

Work Flow


1.
The customer arrives at the parking spot where their details are taken, with their license or vehicle paper
2.
After successful customer admission, digital tickets are generated and they park their vehicle at assigned slot
3.
Customers arrive to take their vehicles after some intervals, admin checks the records of customers for withdrawal
4.
After a successful withdrawal, auto generated receipt will appear and customer records are stored as receipt history

Aim

Our application aims to solve parking problems by using modern and scientific methods. Make less use of paper throughout giving the facility as tickets and receipt are digital based rather than paper based.

Objectives

- Maximum use of the space for parking.
- Reduce traffic on the road.
- Supports for the parking of both small and large vehicles.
- User-friendly with contains a lot of services.
- No hectic to customers for use of our system.
- The database uses to secure data.
- All payment histories are saved which helps to contribute tax on the country.
- Remove old technology parking management system.
- Better service for both consumers and customers.
- Uses the latest technology to enhance user experiences.
- Reduce carbon emissions.

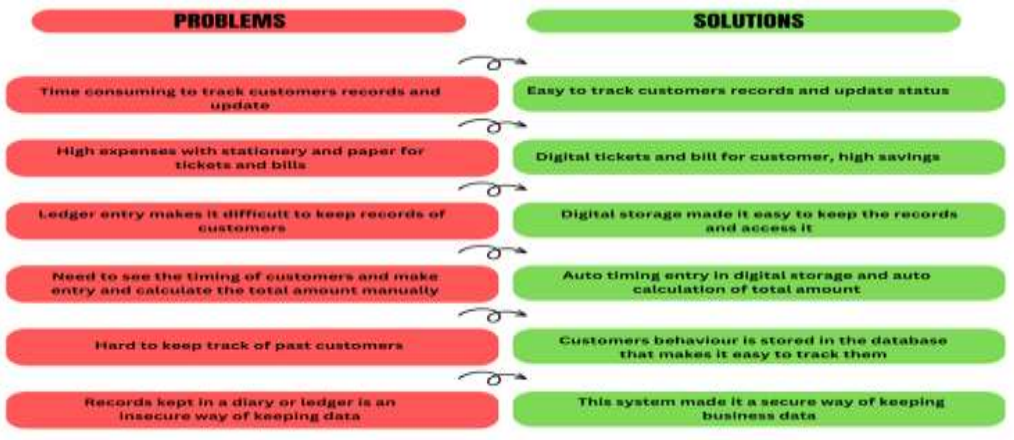
Problem Statement

The old parking management approaches contain plenty of problems that disadvantages that the newly developed garage administration system attempts to remedy. The drawbacks of manual parking management, which cause delays, loss of revenue, and customer and garage operator unhappiness, are at the center of the issue statement. The inadequate use of spots for parking represents one of the key problems. Drivers waste a lot of time looking for parking through outdated systems since there doesn't exist real-time information regarding available spaces, which causes traffic congestion and annoyance. Furthermore, improper space distribution may result in inefficient use, with some sections going unused while others become congested.

The income leakage brought on by manual payment processing is an additional issue. Payment in cash and automated flagging may additionally end in mistakes and abuse, which might cost parking owners money. Additionally, because parking charges remain constant regardless of demand due the absence of dynamic pricing schemes, potential for revenue optimization during peak hours or special events are lost.

Figure 3

Problems vs Solution



PROBLEMS	SOLUTIONS
Time consuming to track customers records and update	Easy to track customers records and update status
High expenses with stationary and paper for tickets and bills	Digital tickets and bill for customer, high savings
Ledger entry makes it difficult to keep records of customers	Digital storage made it easy to keep the records and access it
Need to see the timing of customers and make entry and calculate the total amount manually	Auto timing entry in digital storage and auto calculation of total amount
Hard to keep track of past customers	Customers behaviour is stored in the database that makes it easy to track them
Records kept in a diary or ledger is an insecure way of keeping data	This system made it a secure way of keeping business data

Features

- User-friendly UI
- Update the data of the consumers easily
- Privacy of users
- Easy bills and payment
- Handle the data without error
- Delete of user’s details by admin
- Use latest tools and technology
- No hectic to use
- Easy to use

Functional Requirements

Functional Requirements means what the system should do, it includes the features and functions:

- Easy data modification of the users
- generate tickets after the booking of slots
- CRUD database used
- Payment is done at the use of the services
- categorize the vehicle types
- Delete the data of the users by admin
- store all the data of the users in the system
- shows the available spaces for parking

Non-functional requirements

Non-functional Requirements means how the system performs a certain function are as follows:

- ❖ Usability: Easy to use and eye-catching UI.
- ❖ Performance: it takes only 1-2 second to go from one UI to another and login response time is 2 seconds.
- ❖ Security: Login with the correct password and username. Otherwise, the system doesn't give you enter the permissions of the use of the system.
- ❖ Maintainability: Easy to maintain the system.
- ❖ Changeability: Easy to modification and delete of the data.
- ❖ Capacity: It holds a lot of the data at a time approximately 300.
- ❖ Scalability: Not too many issues in upgrading the system.
- ❖ Portability: Only valid in the Operating system.
- ❖ Recoverability: system should handle the failure data without any loss of data.

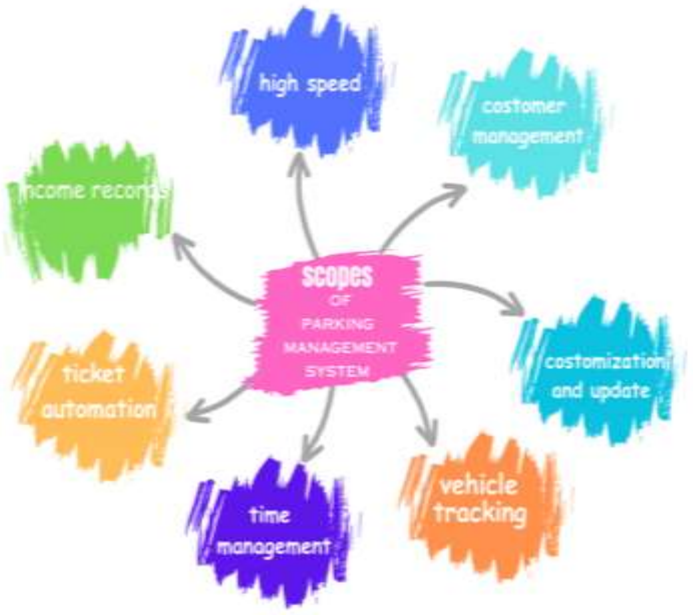
Scope

The parking management system determines its scope by its functions, services, features, aim and objectives. The main scope of this system is given below:

- Clients use our system without any risks
- Better management of the parking system by using the latest tools and technology
- Designed to hold the large numbers of vehicles at a time
- No chance to loss the data of users
- Easy modification of the data
- Use in the city area where large numbers of vehicles are parked
- Maintain security of the users
- Using analysis of data to arrive at better conclusions.
- Full time supports for users
- environmental factors for environmentally friendly actions.

Figure 5

Scope of System



Development Methodology

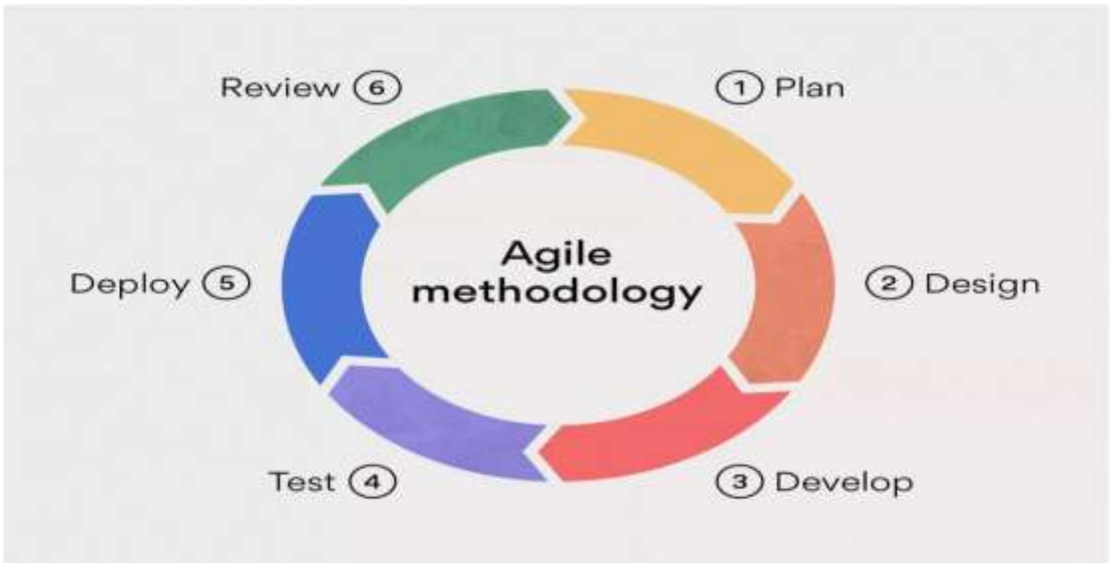
Development of software requires a Software Development Life cycle(SDLC) which gives the full information about the development of software step by step. It is crucial to make software effective and efficient.

Methodology

In projects, we employ the rapid development process, which produces flexible software quickly. This is the most recent methodology that has gained popularity recently; initially, the team members create a strategy for finishing the project. The system's front end is designed during the subsequent cycle generally employing a canvas, Figma, and Balsamiq to interact with users. Developed the strategy and regularly assessed how it performed in a real-world setting. following the discovery of systemic faults. We fix the system's mistakes.

Figure 7

Agile methodology



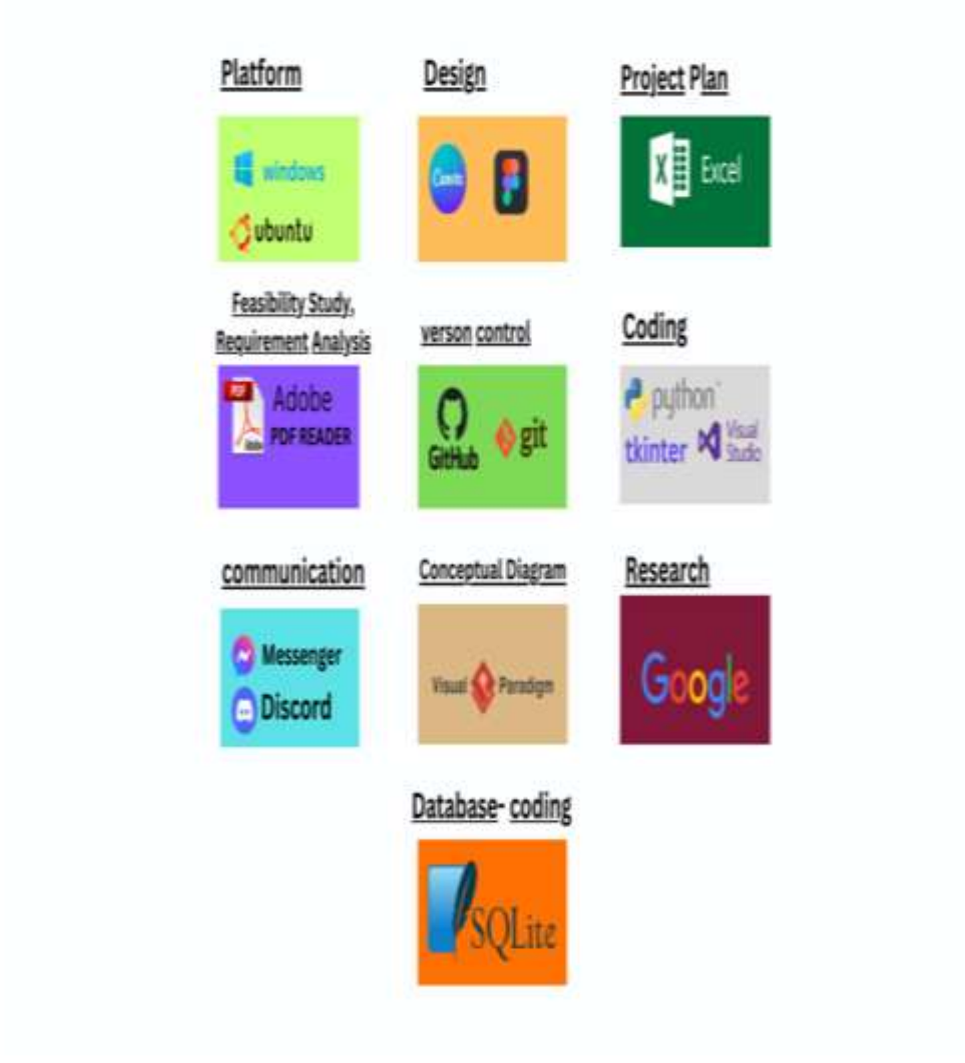
Tools and Technologies

There are several tools and technologies used to build this software. They are given below:

- **windows and Ubuntu** as working platforms
- **Adobe pdf Reader** for feasibility study and requirements analysis
- **Canva and figma** used for the design phase
- **Python** with tkinter library and **Visual Studio Code** for coding phase
- **Sqlite3** used for database in Development phase
- **Visual paradigm** used for conceptional diagram
- **Git** and **GitHub** uses for version control
- **Discord and messenger** used for feedback and discussions
- **Google** used for the search
- **MS Excel** used for creating a project plan

Figure 9

Tools and Technology used

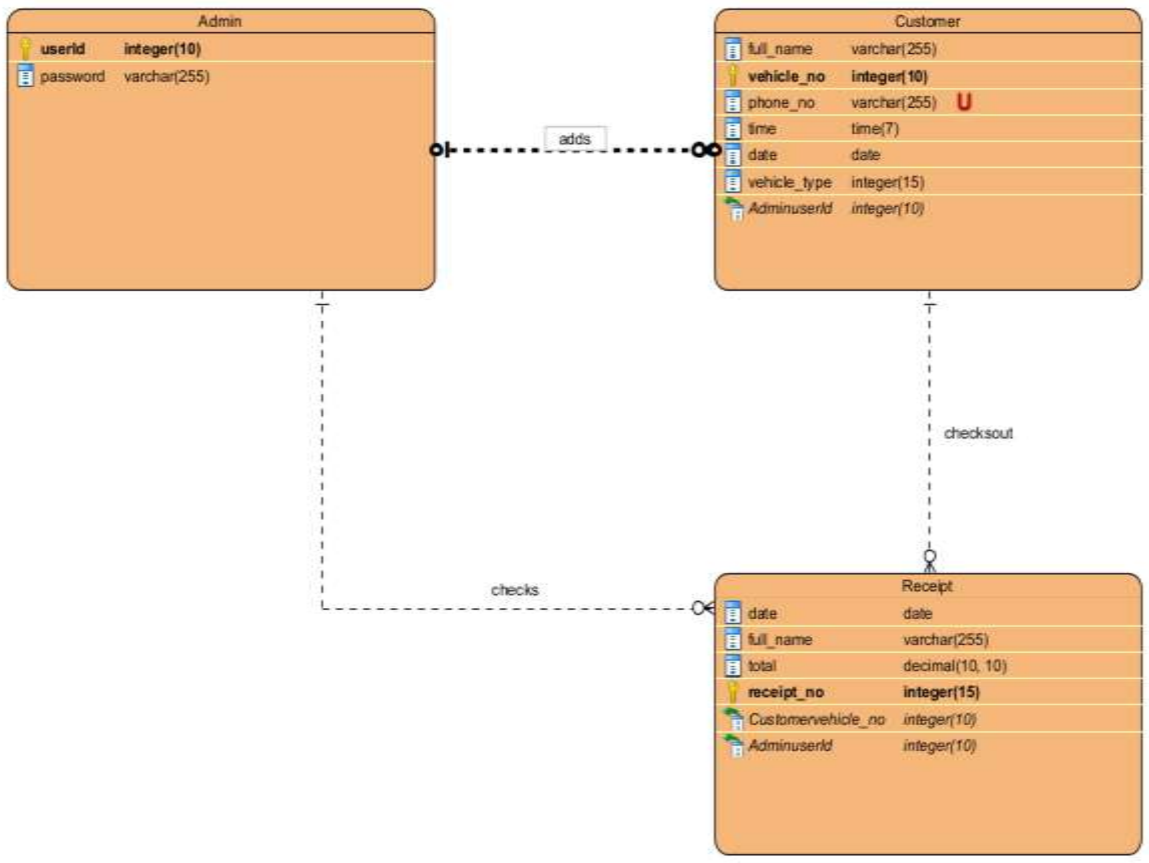


Conceptual Diagram

An Entity-relationship diagram shows the relationship between entities in the system *(Free Visual Paradigm Online, n.d.).*

Figure 11

ER diagram



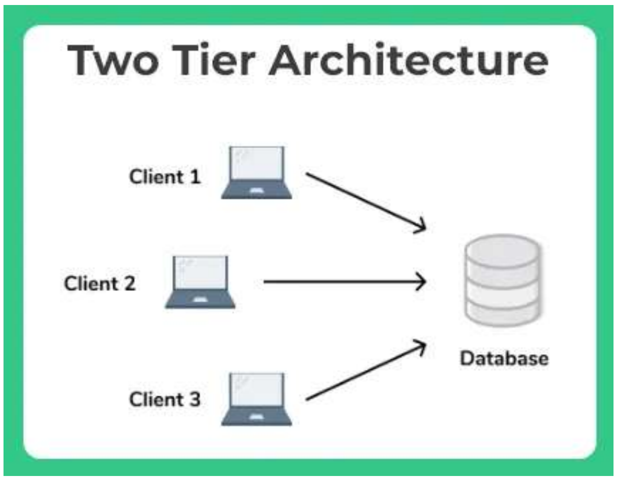
System Architecture

A software architecture having two tiers, the client tier, and the server tier, is referred to as a two-tier system in the context of parking management. The top end's client layer, which makes up the user interface used by administrators and consumers, includes this component. This layer facilitates handles like booking, examining carpark details, and gaining knowledge of real-time changes in the overall picture of parking management. For the purposes of data retrieval and task execution, the client's level interfaces alongside the server's resources tier.

Processing of data, logic, and memory is the responsibility of the server's layer, sometimes known as the back end. It manages duties including bookings, occupancy status, and transaction processing while keeping an eye on databases containing vital customer and parking space information. The user interface and the underlying data and company logic are effectively connected therewith through the server layer, and that effectively organizes customer inquiries by analyzing and replying to them. Even while the two-tier architecture makes system design simpler, it may have scaling and load distribution issues, which forces increasingly complex systems to use multi-tier or services designs for better performance and flexibility ***(Rajkumar, 2023).***

Figure 13

Architecture



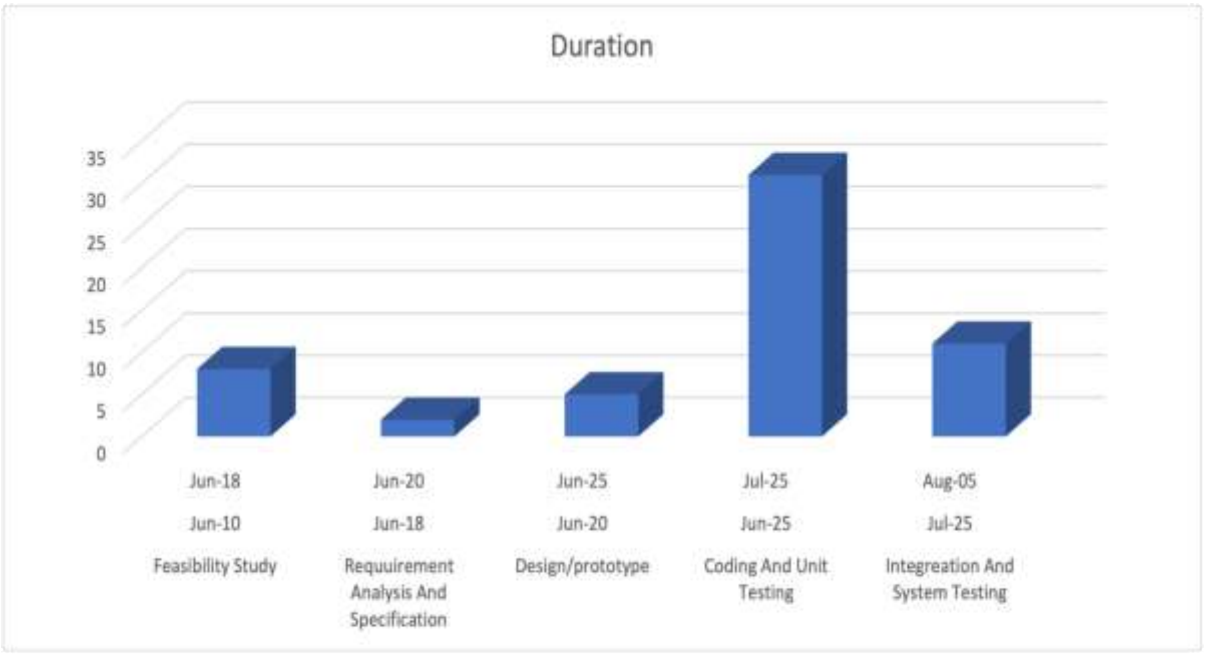
Project Plan

A project plan is necessary to track our efforts, progress, and work remaining and give all information about the project. Due to this chart, we evaluate the pace of the project and how many days require to fulfil complete the work**.**

Task	**Start Date**	**End Date**	**Duration**
Feasibility Study	**Jun-10**	**Jun-18**	**8**
Requuirement Analysis And Specification	**Jun-18**	**Jun-20**	**2**
Design/prototype	**Jun-20**	**Jun-25**	**5**
Coding And Unit Testing	**Jun-25**	**Jul-25**	**31**
Integreation And System Testing	**Jul-25**	**Aug-05**	**11**

Figure 15

Development Chart

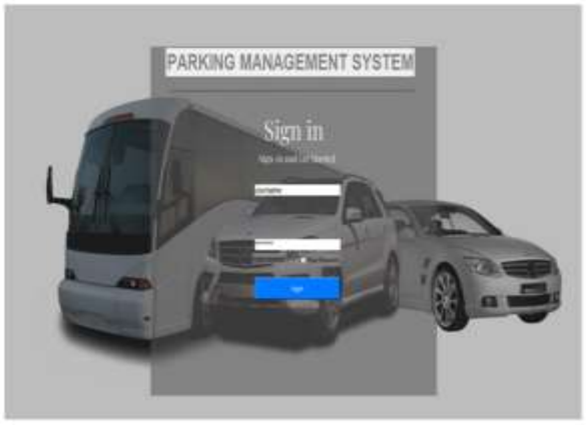


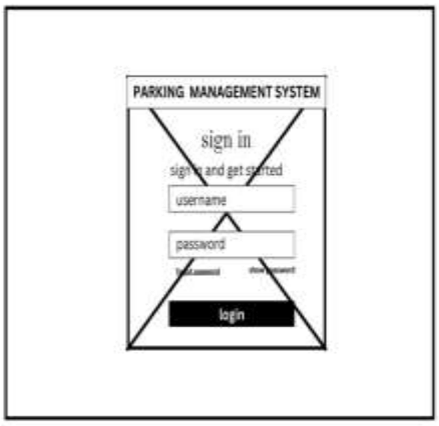
Prototypes

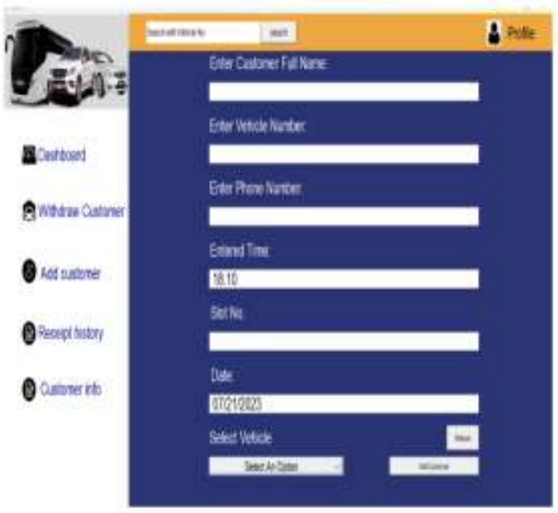
Prototypes are the preliminary version of the product. Throughout the early phases if growth and development, prototypes are primitive sketches or sketches from an architecture, good, any computer program that are used to showcase and evaluate the features, functionality, and design. These models, which can be interactive, functional, or both, give stakeholders, developers, and consumers a concrete grasp of the planned result. Before committing to full-scale development, prototyping is a useful technique for modifying requirements, obtaining feedback, and spotting possible problems. It helps teams interact and make well-informed decisions by helping them understand design ideas, graphical interfaces, and user interactions. Depending on the requirements and complexity of the project, prototypes can take a variety of shapes, ranging from low-fidelity drawings to high-fidelity interactive simulations.

Figure 17

High fidelity and low-fidelity designs

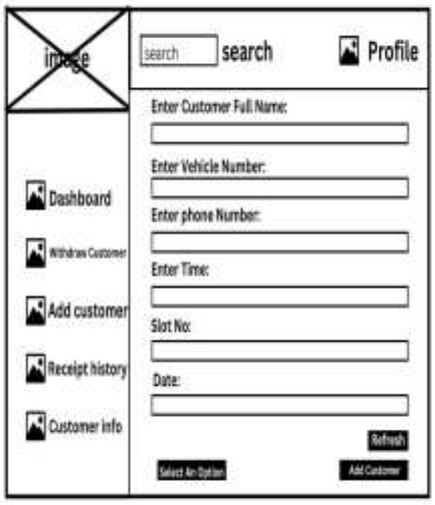


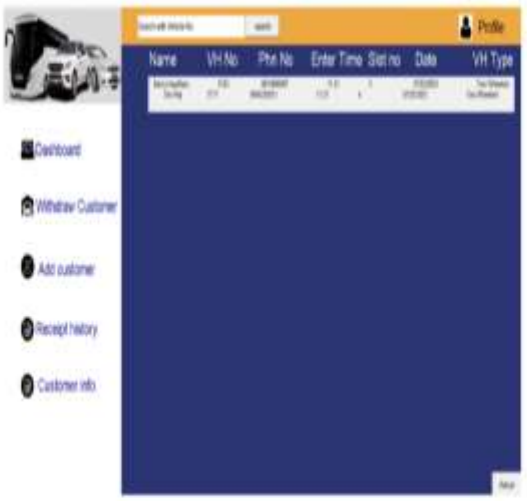




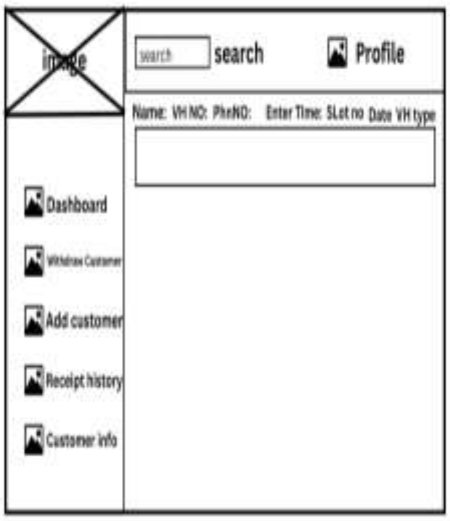
Entered Time: 18.10

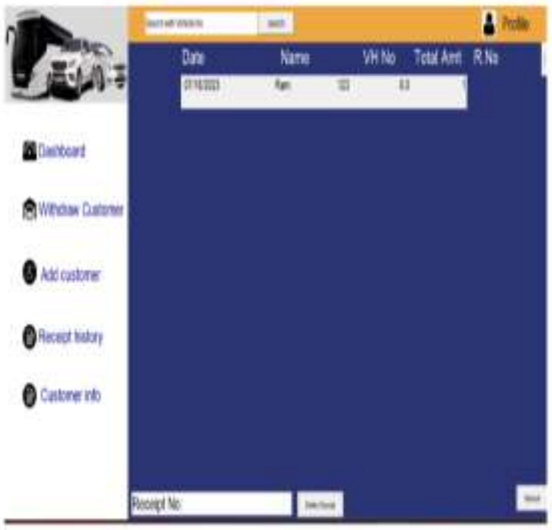
Date: 07/21/2023



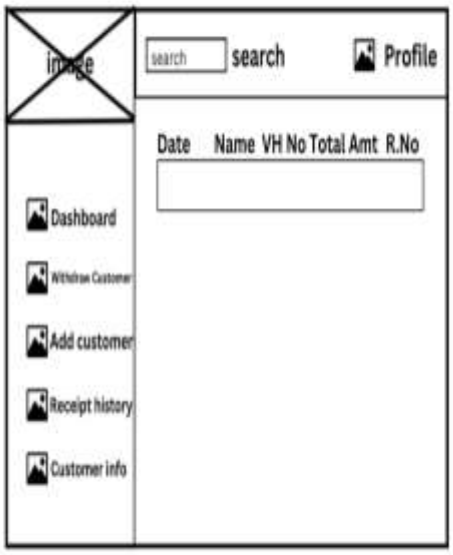


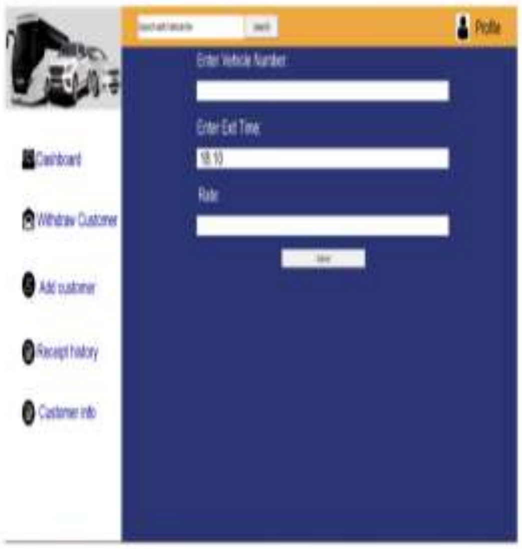
Name	VH No	Phn No	Enter Time	Slot no	Date	VH Type
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]





Date	Name	VH No	Total Amt	R.No
07/16/2023	Ram	123	63	1



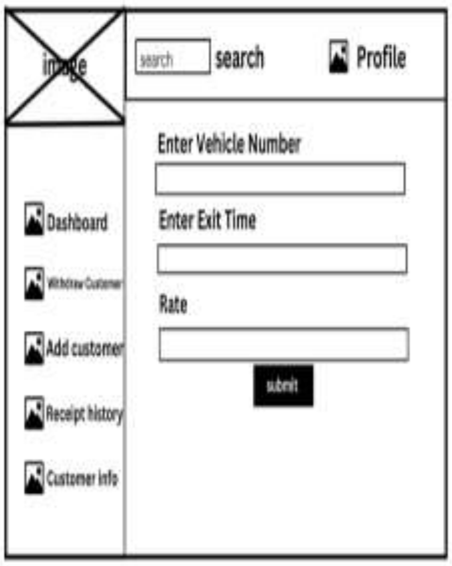


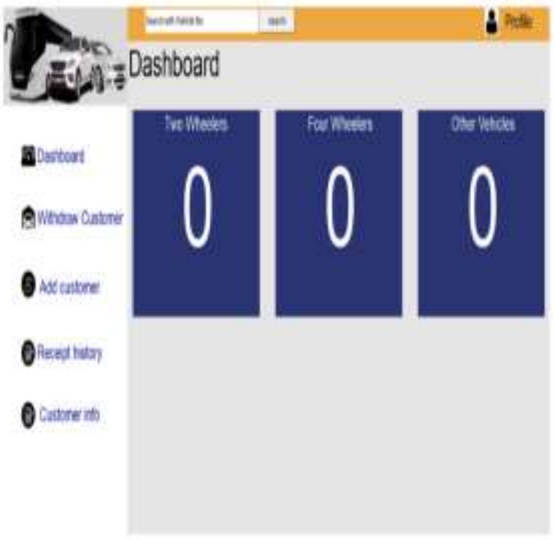
Enter Vehicle Number

Enter Exit Time

18:10

Rate




Search with Vehicle No
search
Profile
Dashboard
Dashboard
Withdraw Customer
Add customer
Receipt history
Customer info
Two Wheelers
0
Four Wheelers
0
Other Vehicles
0

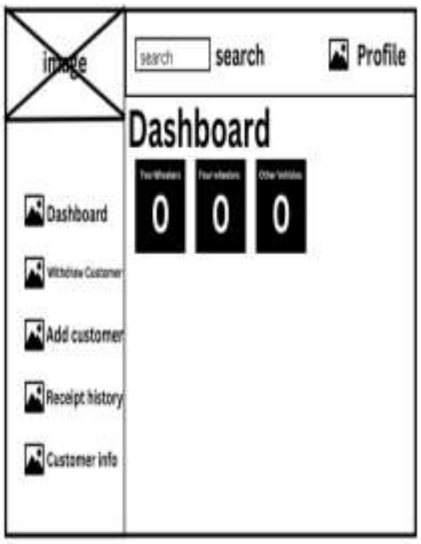


image
search
search
Profile
Dashboard
Dashboard
Withdraw Customer
Add customer
Receipt history
Customer info
Two Wheelers
0
Four wheelers
0
Other Vehicles
0

Developed System

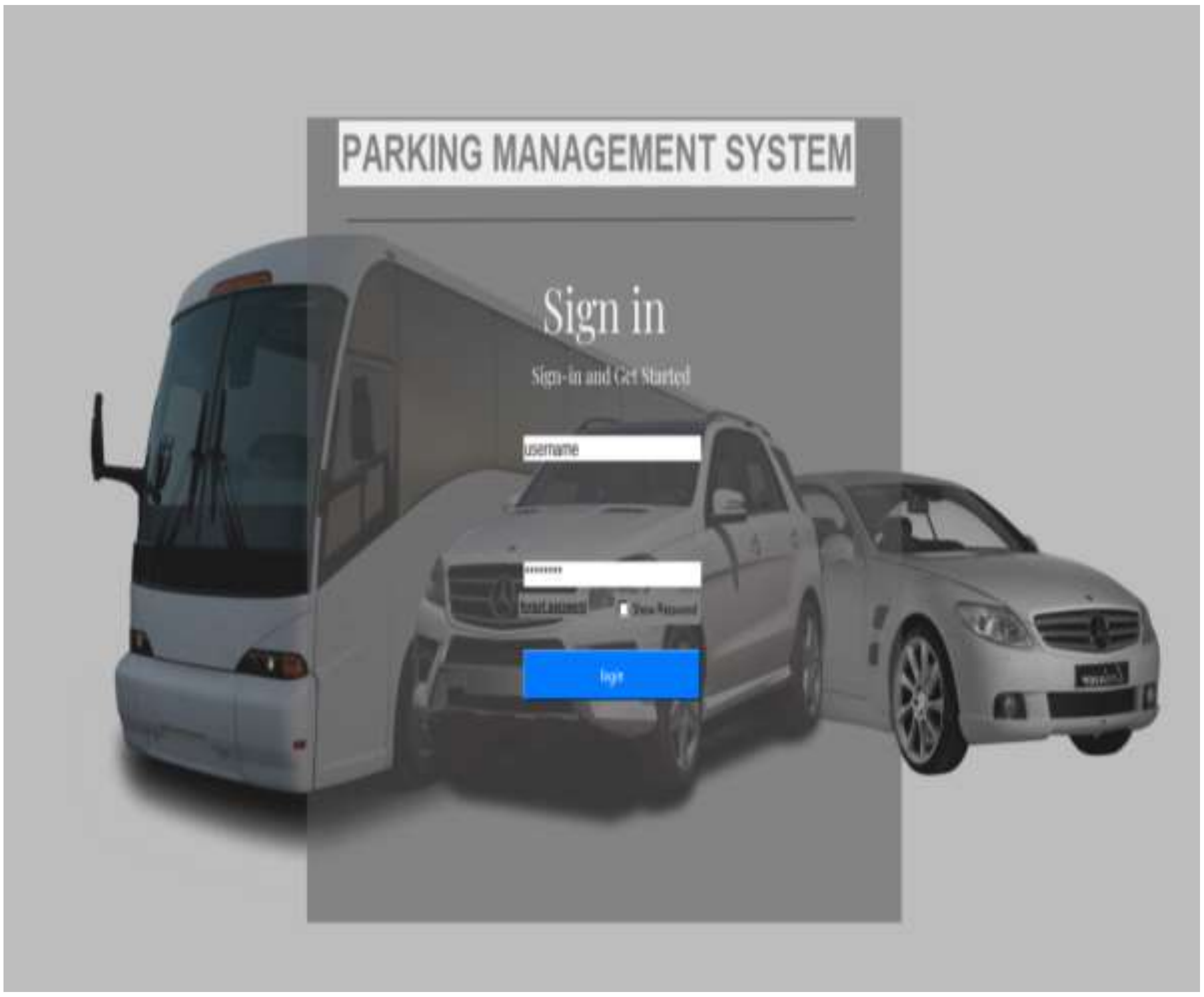
The parking management system displays a user frontend for driving monitoring and was created employing Python's Tkinter, who framework and SQLite3. It shows car parks that provides number identification update for registration. The graphical interface components are built by Tkinter, while database handling is made easier with SQLite3. The system captures an identification information and keeps it in an SQLite database. A button that is press produces a listing in cars currently parked. To address needs, such as error handling, validations, and a larger feature set for an effective parking management platform, additional adjustments and adaptation are advised.

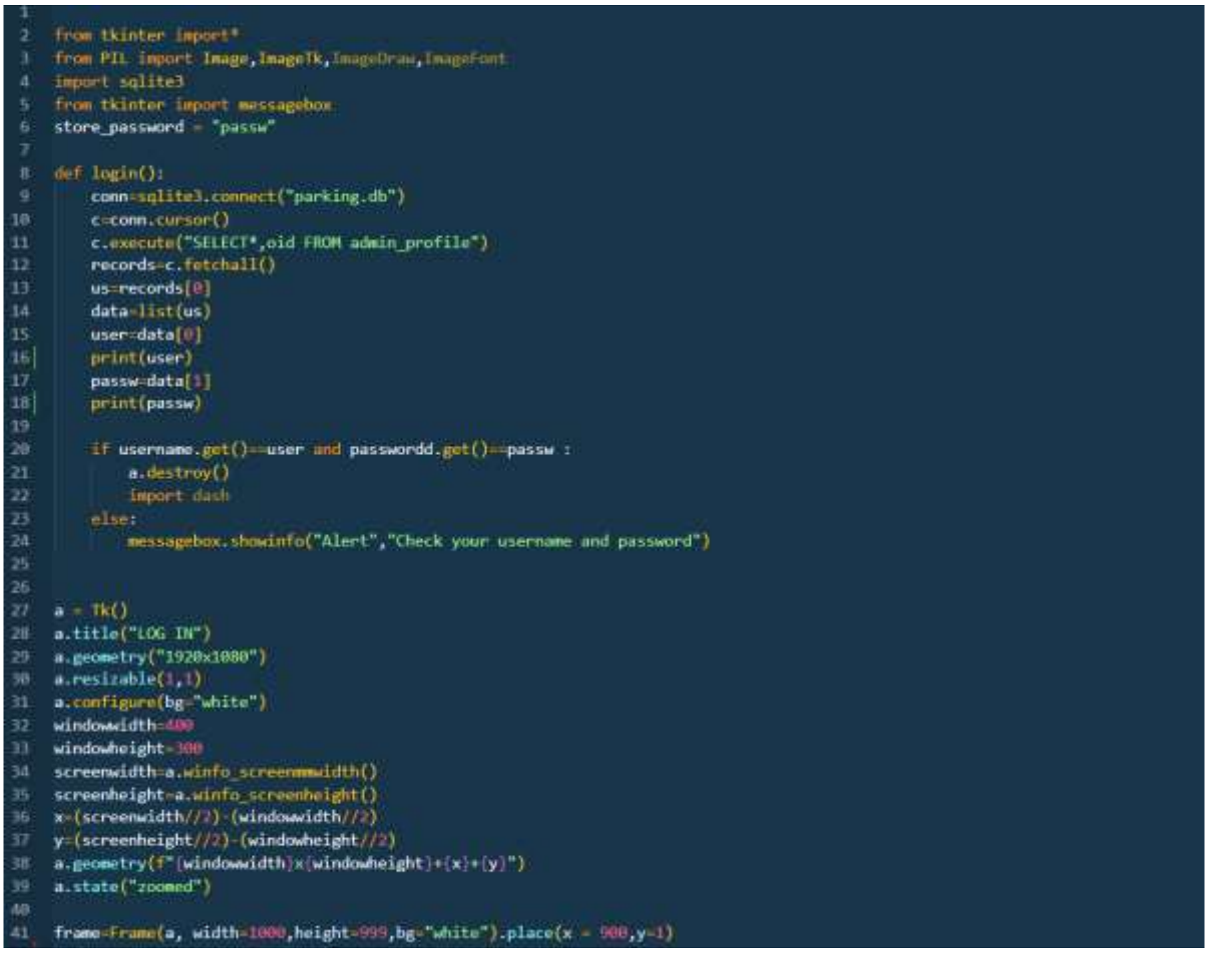
Login UI

The lot administration method's admin login page has a clean, well-organized layout. The words "Sign-in" are shown at the top, along with the identifiable system emblem. A labeled input area for entering the username is immediately below, and a similar layout is used for the password entry. The password field has a "Show Password" option next to it that users may utilize to control password visibility. If administrators lose their password, a handy "Forgot Password?" link is provided in the login name and password draw making recovery of the password simple. Administrators can start the login process through clicking the "Login" button, which appears prominently at the bottom. Any input problems are immediately reported via error warnings that appear beneath the relevant input forms.
If the administrator enters the correct login and password, they are sent immediately to the dashboard. If not, the computer displays the incorrect username and password.

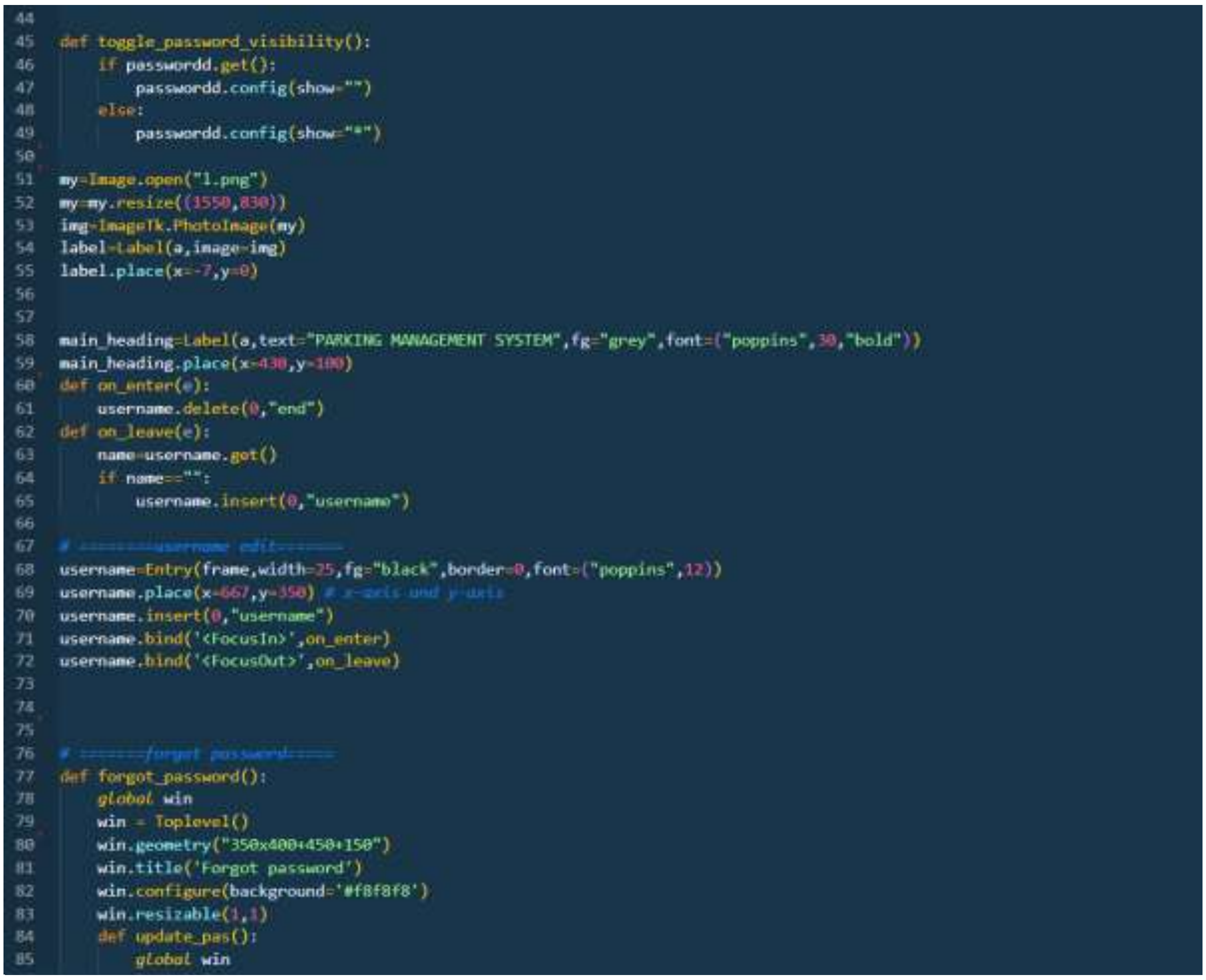
Figure 19

Login page and code

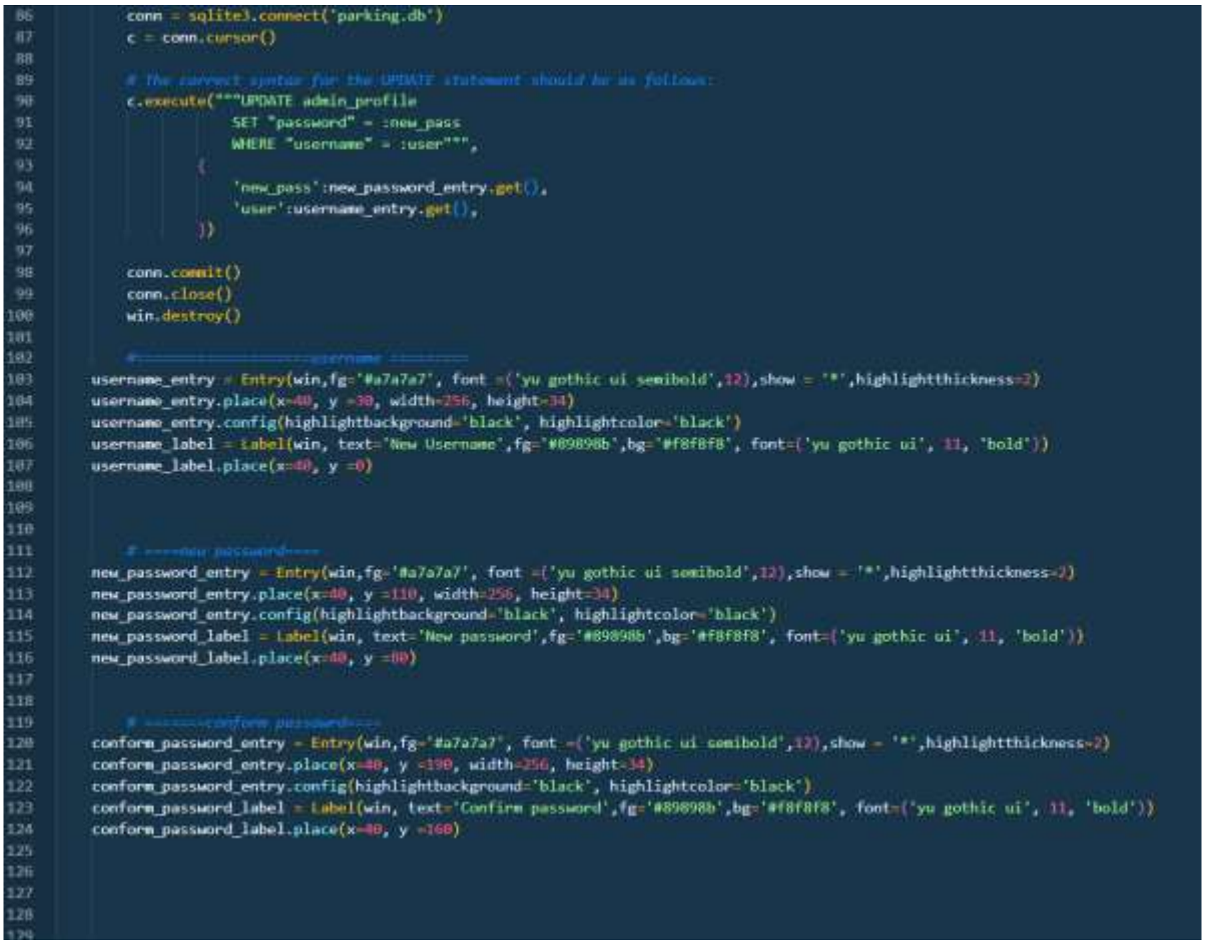




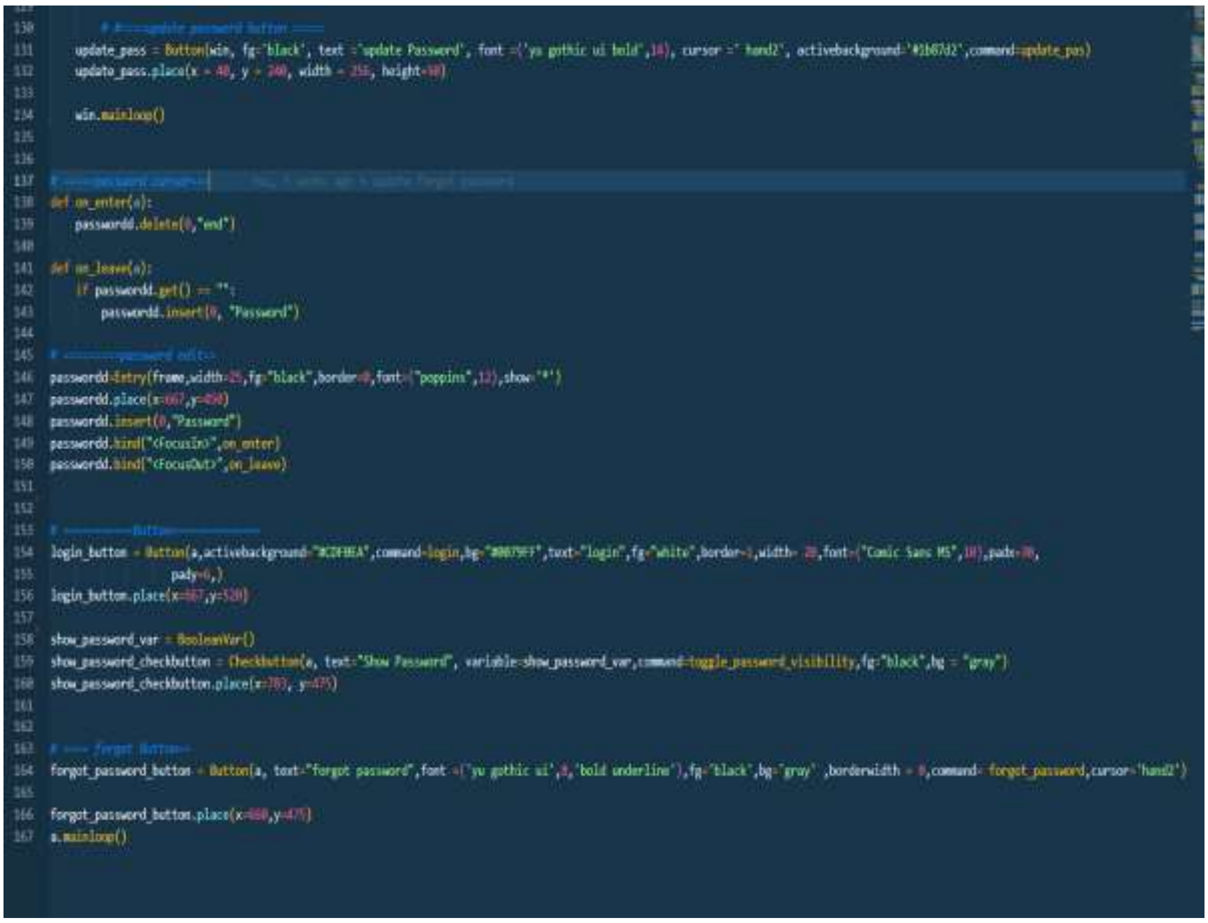
```

from tkinter import*
from PIL import Image,ImageTk,ImageDraw,ImageFont
import sqlite3
from tkinter import messagebox
store_password = "passw"

def login():
    conn=sqlite3.connect("parking.db")
    c=conn.cursor()
    c.execute("SELECT*,oid FROM admin_profile")
    records=c.fetchall()
    us=records[0]
    data=list(us)
    user=data[0]
    print(user)
    passw=data[1]
    print(passw)

    if username.get()==user and passwordd.get()==passw :
        a.destroy()
        import dash
    else:
        messagebox.showinfo("Alert","Check your username and password")


a = Tk()
a.title("LOG IN")
a.geometry("1920x1080")
a.resizable(1,1)
a.configure(bg="white")
windowwidth=400
windowheight=300
screenwidth=a.winfo_screenmmwidth()
screenheight=a.winfo_screenheight()
x=(screenwidth//2)-(windowwidth//2)
y=(screenheight//2)-(windowheight//2)
a.geometry(f"{windowwidth}x{windowheight}+{x}+{y}")
a.state("zoomed")

frame=Frame(a, width=1000,height=999,bg="white").place(x = 900,y=1)
```



```python

def toggle_password_visibility():
    if passwordd.get():
        passwordd.config(show="")
    else:
        passwordd.config(show="*")

my=Image.open("1.png")
my=my.resize((1550,830))
img=ImageTk.PhotoImage(my)
label=Label(a,image=img)
label.place(x=-7,y=0)


main_heading=Label(a,text="PARKING MANAGEMENT SYSTEM",fg="grey",font=("poppins",30,"bold"))
main_heading.place(x=430,y=100)
def on_enter(e):
    username.delete(0,"end")
def on_leave(e):
    name=username.get()
    if name=="":
        username.insert(0,"username")

# ==========username edit========
username=Entry(frame,width=25,fg="black",border=0,font=("poppins",12))
username.place(x=667,y=350) # x-axis and y-axis
username.insert(0,"username")
username.bind('<FocusIn>',on_enter)
username.bind('<FocusOut>',on_leave)



# ========forgot password=====
def forgot_password():
    global win
    win = Toplevel()
    win.geometry("350x400+450+150")
    win.title('Forgot password')
    win.configure(background='#f8f8f8')
    win.resizable(1,1)
    def update_pas():
        global win
```



```
        conn = sqlite3.connect('parking.db')
        c = conn.cursor()

        # The correct syntax for the UPDATE statement should be as follows:
        c.execute("""UPDATE admin_profile
                    SET "password" = :new_pass
                    WHERE "username" = :user""",
                {
                    'new_pass':new_password_entry.get(),
                    'user':username_entry.get(),
                })

        conn.commit()
        conn.close()
        win.destroy()

        #======================username =========
    username_entry = Entry(win,fg='#a7a7a7', font =('yu gothic ui semibold',12),show = '*',highlightthickness=2)
    username_entry.place(x=40, y =30, width=256, height=34)
    username_entry.config(highlightbackground='black', highlightcolor='black')
    username_label = Label(win, text='New Username',fg='#89898b',bg='#f8f8f8', font=('yu gothic ui', 11, 'bold'))
    username_label.place(x=40, y =0)



        # ======new password====
    new_password_entry = Entry(win,fg='#a7a7a7', font =('yu gothic ui semibold',12),show = '*',highlightthickness=2)
    new_password_entry.place(x=40, y =110, width=256, height=34)
    new_password_entry.config(highlightbackground='black', highlightcolor='black')
    new_password_label = Label(win, text='New password',fg='#89898b',bg='#f8f8f8', font=('yu gothic ui', 11, 'bold'))
    new_password_label.place(x=40, y =80)


        # =======conform password====
    conform_password_entry = Entry(win,fg='#a7a7a7', font =('yu gothic ui semibold',12),show = '*',highlightthickness=2)
    conform_password_entry.place(x=40, y =190, width=256, height=34)
    conform_password_entry.config(highlightbackground='black', highlightcolor='black')
    conform_password_label = Label(win, text='Confirm password',fg='#89898b',bg='#f8f8f8', font=('yu gothic ui', 11, 'bold'))
    conform_password_label.place(x=40, y =160)
```



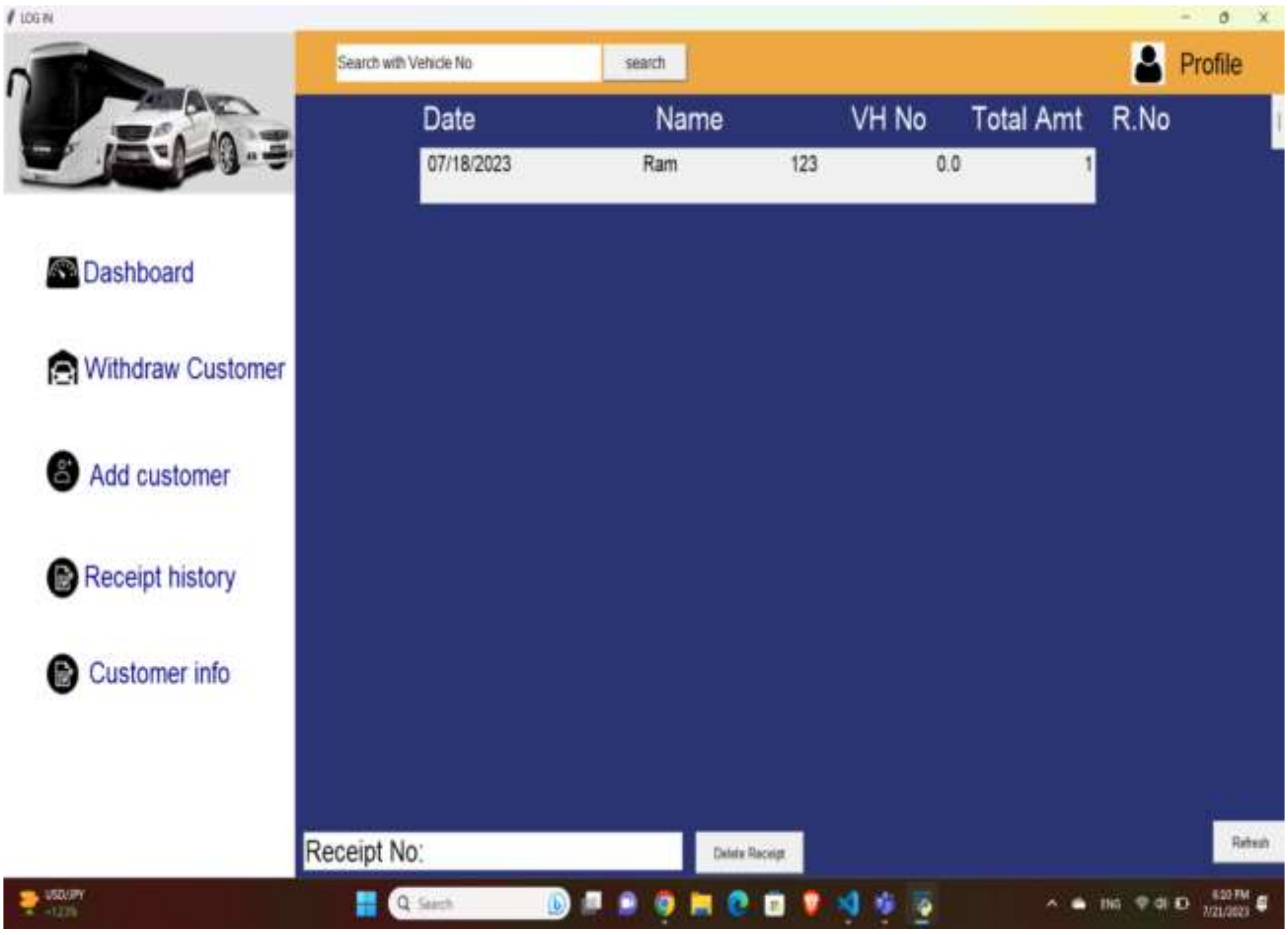
```
        # #====update password button ====
    update_pass = Button(win, fg='black', text ='update Password', font =('yu gothic ui bold',14), cursor =' hand2', activebackground='#1b87d2',command=update_pas)
    update_pass.place(x = 40, y = 240, width = 256, height=40)

    win.mainloop()


# ====password cursor===
def on_enter(e):
    passwordd.delete(0,"end")

def on_leave(e):
    if passwordd.get() == "":
        passwordd.insert(0, "Password")

# ========password edit==
passwordd=Entry(frame,width=25,fg="black",border=0,font=("poppins",12),show="*")
passwordd.place(x=667,y=450)
passwordd.insert(0,"Password")
passwordd.bind("<FocusIn>",on_enter)
passwordd.bind("<FocusOut>",on_leave)


# ----------Button-------------
login_button = Button(a,activebackground="#CDF0EA",command=login,bg="#0079FF",text="login",fg="white",border=1,width= 20,font=("Comic Sans MS",10),padx=30,
                pady=6,)
login_button.place(x=667,y=520)

show_password_var = BooleanVar()
show_password_checkbutton = Checkbutton(a, text="Show Password", variable=show_password_var,command=toggle_password_visibility,fg="black",bg = "gray")
show_password_checkbutton.place(x=703, y=475)


# ---- forgot Button--
forgot_password_button = Button(a, text="forgot password",font =('yu gothic ui',8,'bold underline'),fg='black',bg='gray' ,borderwidth = 0,command= forgot_password,cursor='hand2')

forgot_password_button.place(x=660,y=475)
a.mainloop()
```

Customer Details UI

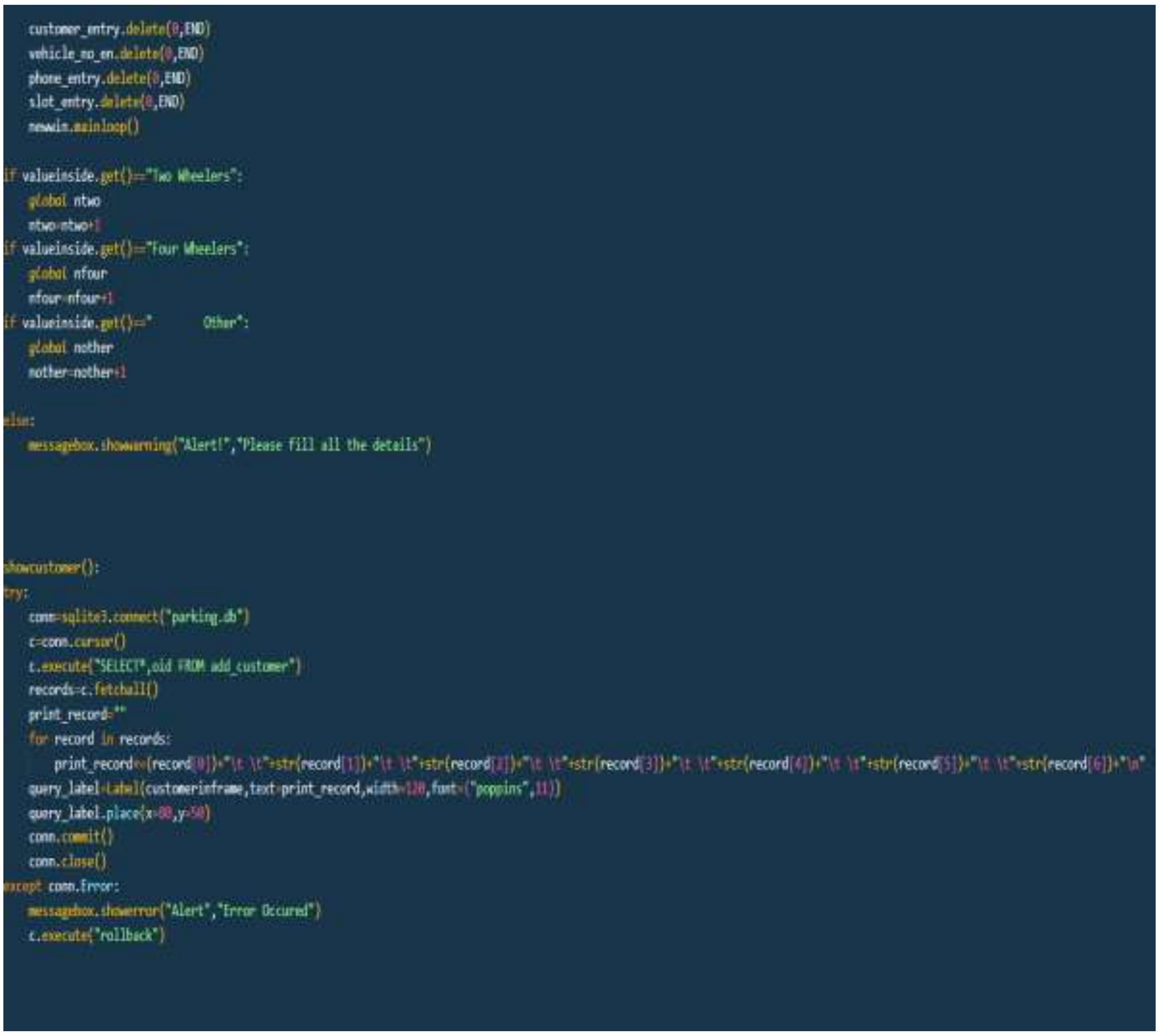
This UI shows the details of the customers, admin have only access to the see data, deletes and updates. It is used CRUD method.

Figure 21

System UI and code



Date	Name	VH No	Total Amt	R.No
07/18/2023	Ram	123	0.0	1



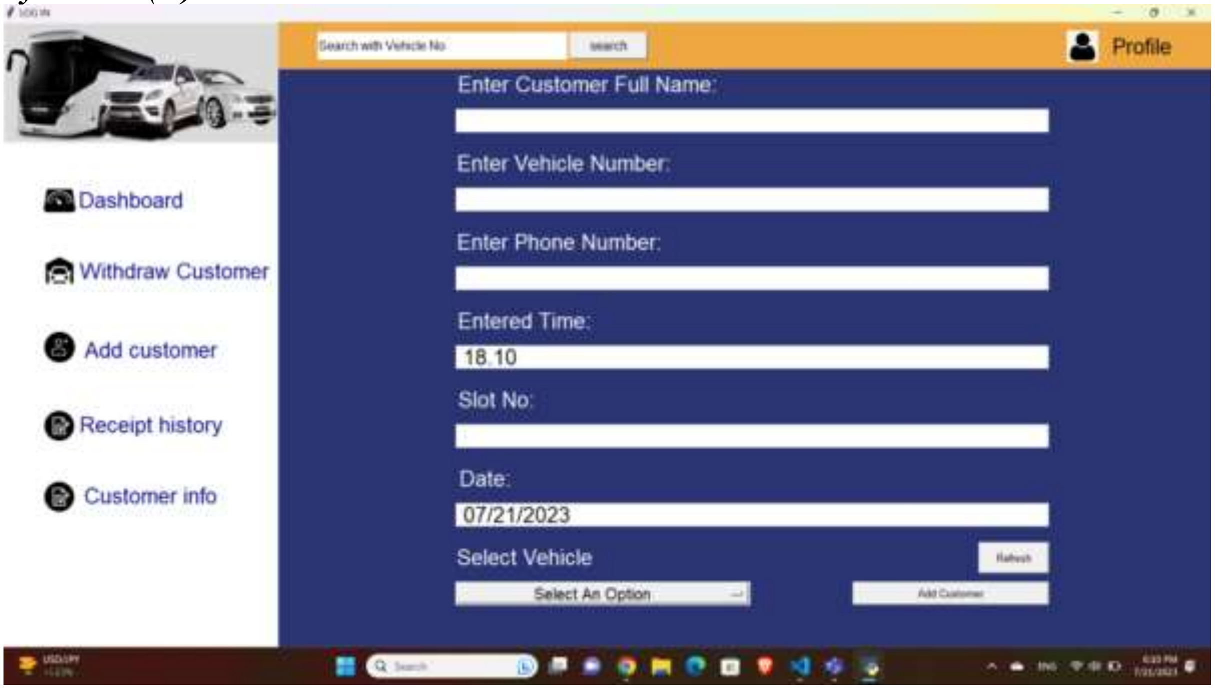
```
    customer_entry.delete(0,END)
    vehicle_no_en.delete(0,END)
    phone_entry.delete(0,END)
    slot_entry.delete(0,END)
    newwin.mainloop()

if valueinside.get()=="Two Wheelers":
    global ntwo
    ntwo=ntwo+1
if valueinside.get()=="Four Wheelers":
    global nfour
    nfour=nfour+1
if valueinside.get()=="        Other":
    global nother
    nother=nother+1

else:
    messagebox.showwarning("Alert!","Please fill all the details")




showcustomer():
try:
    conn=sqlite3.connect("parking.db")
    c=conn.cursor()
    c.execute("SELECT*,oid FROM add_customer")
    records=c.fetchall()
    print_record=""
    for record in records:
        print_record+=(record[0])+"\t \t"+str(record[1])+"\t \t"+str(record[2])+"\t \t"+str(record[3])+"\t \t"+str(record[4])+"\t \t"+str(record[5])+"\t \t"+str(record[6])+"\n"
    query_label=Label(customerinframe,text=print_record,width=120,font=("poppins",11))
    query_label.place(x=80,y=50)
    conn.commit()
    conn.close()
except conn.Error:
    messagebox.showerror("Alert","Error Occured")
    c.execute("rollback")
```

Update / Delete UI

The Update UI is a user interface where admins may update data. The UI now shows a detailed list of all users who have registered, along with their information. An editable form that dynamically populates with the customer's current data after selection makes modifications easier. Administrators may easily alter fields like Full Name, Vehicle Number, Phone Number, Entry Time, Slot Number, Date, and Vehicle Type by following the instructions on the labels. Easy-to-use controls make data alterations simple, and immediate validation guarantees correctness. The updated user interface (UI) simplifies client data changes, improving the lot management system's operational effectiveness and precision.

Figure 23

System UI (II) and Code



Enter Customer Full Name:

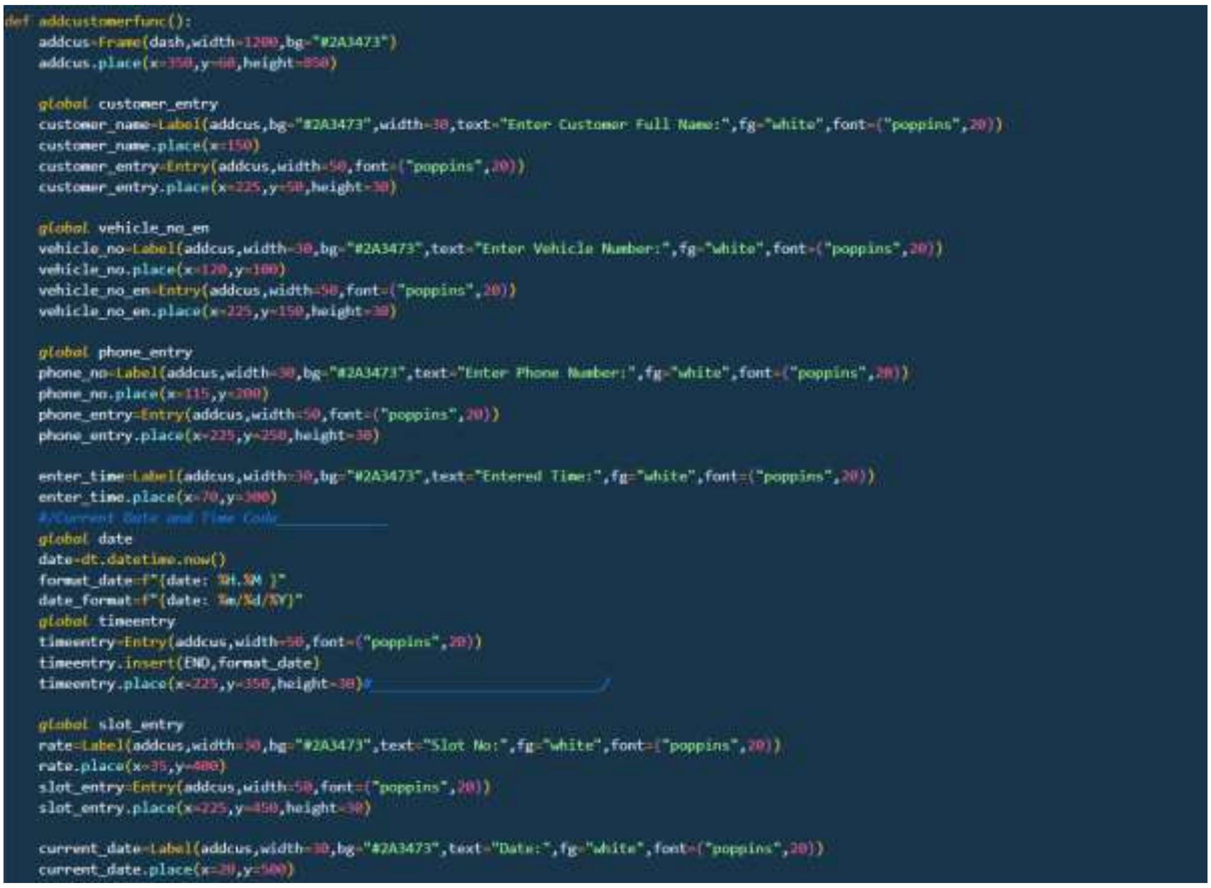
Enter Vehicle Number:

Enter Phone Number:

Entered Time:
18.10

Slot No:

Date:
07/21/2023



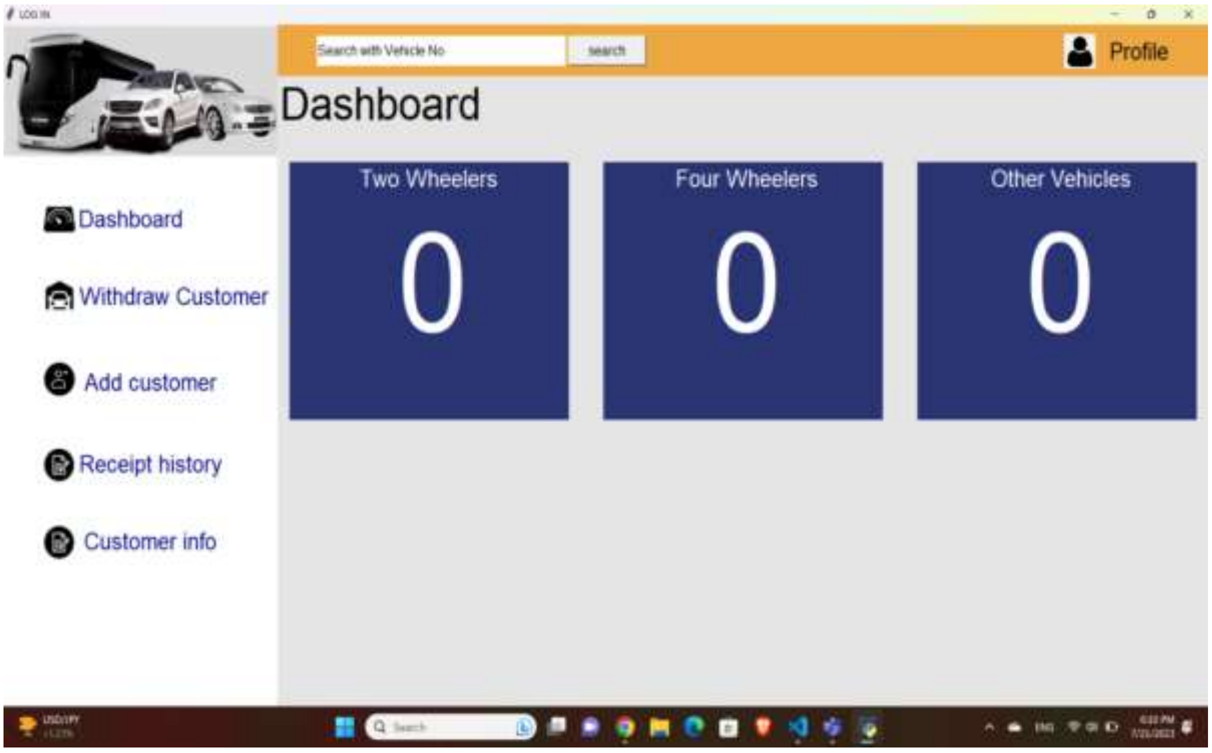
```
def addcustomerfunc():
    addcus=Frame(dash,width=1200,bg="#2A3473")
    addcus.place(x=350,y=60,height=850)

    global customer_entry
    customer_name=Label(addcus,bg="#2A3473",width=30,text="Enter Customer Full Name:",fg="white",font=("poppins",20))
    customer_name.place(x=150)
    customer_entry=Entry(addcus,width=50,font=("poppins",20))
    customer_entry.place(x=225,y=50,height=30)

    global vehicle_no_en
    vehicle_no=Label(addcus,width=30,bg="#2A3473",text="Enter Vehicle Number:",fg="white",font=("poppins",20))
    vehicle_no.place(x=120,y=100)
    vehicle_no_en=Entry(addcus,width=50,font=("poppins",20))
    vehicle_no_en.place(x=225,y=150,height=30)

    global phone_entry
    phone_no=Label(addcus,width=30,bg="#2A3473",text="Enter Phone Number:",fg="white",font=("poppins",20))
    phone_no.place(x=115,y=200)
    phone_entry=Entry(addcus,width=50,font=("poppins",20))
    phone_entry.place(x=225,y=250,height=30)

    enter_time=Label(addcus,width=30,bg="#2A3473",text="Entered Time:",fg="white",font=("poppins",20))
    enter_time.place(x=70,y=300)
    #/Current Date and Time Code_____________
    global date
    date=dt.datetime.now()
    format_date=f"{date: %H.%M }"
    date_format=f"{date: %m/%d/%Y}"
    global timeentry
    timeentry=Entry(addcus,width=50,font=("poppins",20))
    timeentry.insert(END,format_date)
    timeentry.place(x=225,y=350,height=30)#_______________________/

    global slot_entry
    rate=Label(addcus,width=30,bg="#2A3473",text="Slot No:",fg="white",font=("poppins",20))
    rate.place(x=35,y=400)
    slot_entry=Entry(addcus,width=50,font=("poppins",20))
    slot_entry.place(x=225,y=450,height=30)

    current_date=Label(addcus,width=30,bg="#2A3473",text="Date:",fg="white",font=("poppins",20))
    current_date.place(x=20,y=500)
```

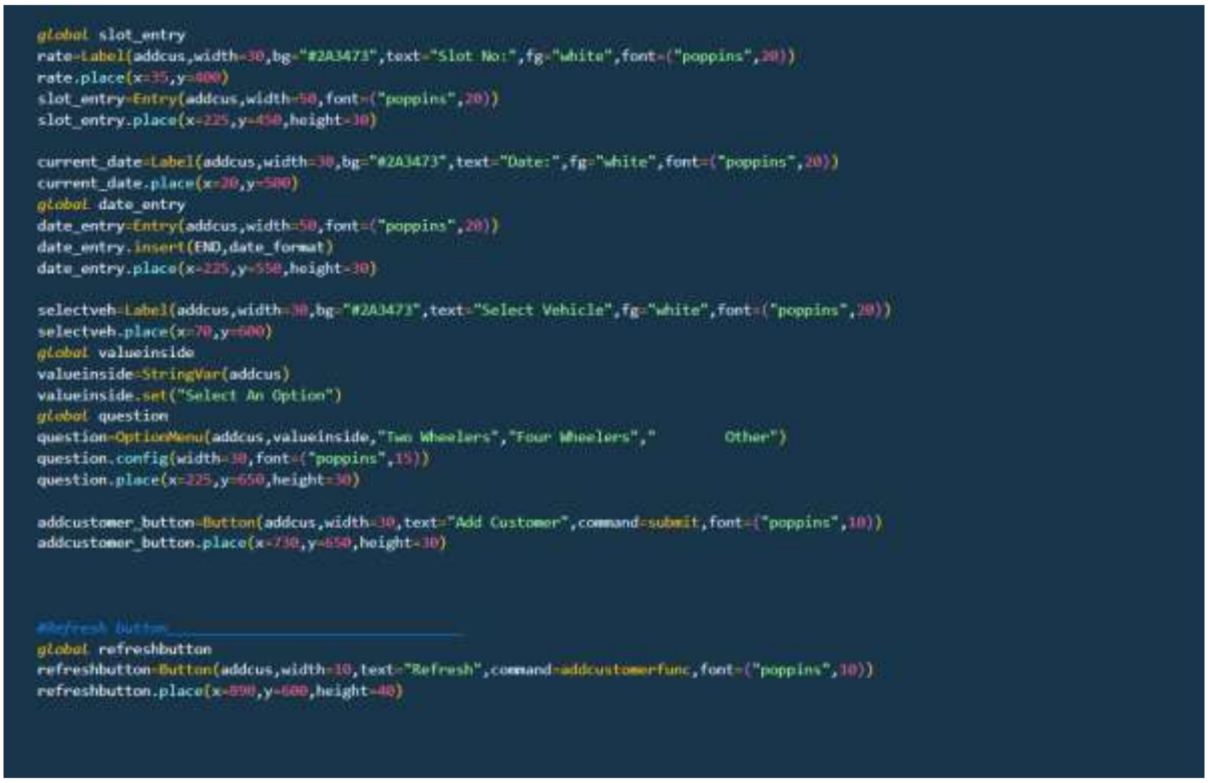
Dashboard UI

There are separate choices within the parking management dashboard for "Two Wheelers," "Four Wheelers," and maybe "Other Vehicles." To allocate slots effectively and provide excellent service to clients, the "Two Wheelers" update information. The "Four Wheelers" portion breaks down parking for cars, making the best use of available space and making it possible to acquire specific information about each vehicle. The addition of an “Other Vehicles” category attend to specific vehicle requirements and provides truck or vans motorists additional information on available parking. With the help of this menu structure, complete parking management for all vehicle kinds is made more operationally efficient.

Figure 25

Dashboard UI and Code





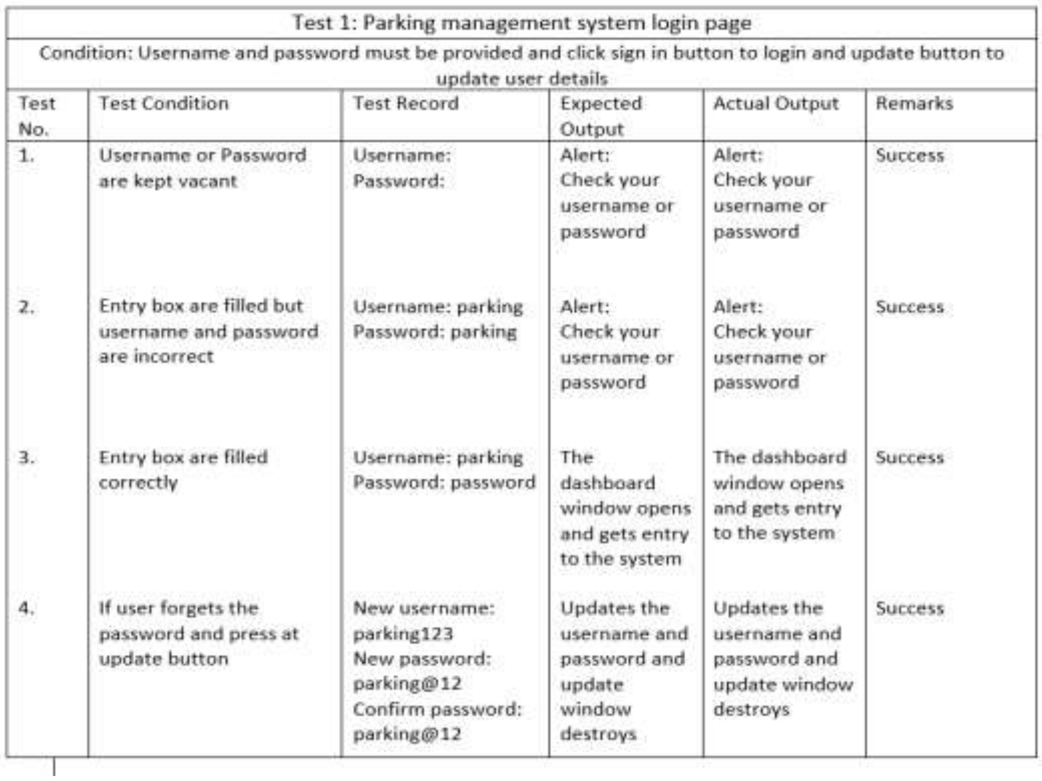
```
global slot_entry
rate=Label(addcus,width=30,bg="#2A3473",text="Slot No:",fg="white",font=("poppins",20))
rate.place(x=35,y=400)
slot_entry=Entry(addcus,width=50,font=("poppins",20))
slot_entry.place(x=225,y=450,height=30)

current_date=Label(addcus,width=30,bg="#2A3473",text="Date:",fg="white",font=("poppins",20))
current_date.place(x=20,y=500)
global date_entry
date_entry=Entry(addcus,width=50,font=("poppins",20))
date_entry.insert(END,date_format)
date_entry.place(x=225,y=550,height=30)

selectveh=Label(addcus,width=30,bg="#2A3473",text="Select Vehicle",fg="white",font=("poppins",20))
selectveh.place(x=70,y=600)
global valueinside
valueinside=StringVar(addcus)
valueinside.set("Select An Option")
global question
question=OptionMenu(addcus,valueinside,"Two Wheelers","Four Wheelers","        Other")
question.config(width=30,font=("poppins",15))
question.place(x=225,y=650,height=30)

addcustomer_button=Button(addcus,width=30,text="Add Customer",command=submit,font=("poppins",10))
addcustomer_button.place(x=730,y=650,height=30)



#Refresh button
global refreshbutton
refreshbutton=Button(addcus,width=10,text="Refresh",command=addcustomerfunc,font=("poppins",10))
refreshbutton.place(x=890,y=600,height=40)
```

System Testing

Black box testing is a form testing of software concentrates on assessing a software application's functioning without considering its underlying code structure or implementation specifics. With this method, testers consider the program as a "black box," interacting with the input and watching the output while examining how the system reacts. The primary objective is to guarantee that the program complies with all necessary specifications and operates properly from the user's perspective. Without any prior understanding of the internal functioning of the software program, testers construct instances of testing according to inputs, expected results, and the system's requirements. This technique aids in finding errors, exposing problems affecting user experience, and verifying if the program corresponds to the functionality planned and user expectations ***(Educative, n.d.).***

Test 4: Withdraw Customers					
Condition: Customer should be withdrawn with receipt after pressing button					
Test No.	Test Condition	Test Record	Expected Output	Actual Output	Remarks
1.	Customer should be withdrawn from the system after filling all the required entry box and clicking the button	Vehicle no: 1234 Exit Time: Auto Rate: 25	Customer will be deleted from database; Receipt will be created and records will remain in receipt history	Customer will be deleted from database; Receipt will be created and records will remain in receipt history	Success
2.	Entry boxes are kept empty partially or fully	Vehicle no: Exit Time: Auto Rate:	Alert: Please fill all the details	Alert: Please fill all the details	Success

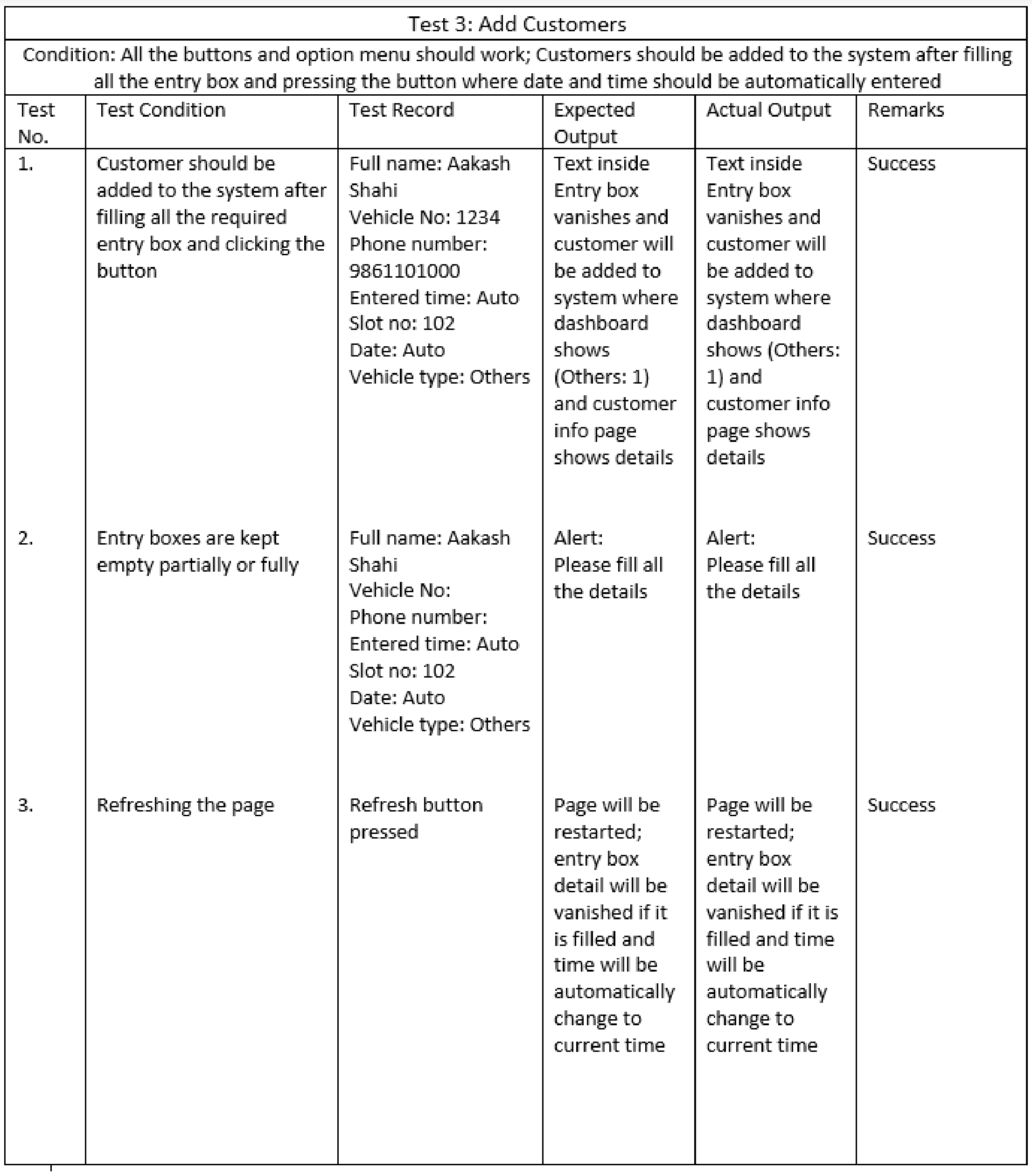
Test 2: Dashboard and sidebar					
Condition: All the buttons on the sidebar should be responsive after being pressed, dashboard interface should be able to show the live overview of different types of vehicles entered in the database					
Test No.	Test Condition	Test Record	Expected Output	Actual Output	Remarks
1.	All buttons on the sidebar must respond every time user clicks	Dashboard, withdraw customer, Add customer, Receipt History, Customer info	Dashboard, withdraw customer, add customer, Receipt History, and Customer info page should be accessible	Dashboard, withdraw customer, add customer, Receipt History, and Customer info page should be accessible	Success
2.	After the user login success, the land page should always be the dashboard page	Successful login from the login window	Dashboard interface opens	Dashboard interface opens	Success
3.	The dashboard page must show the counting of vehicles accordingly after customer is added to the system	Added customer from add customer section, vehicle type: (two wheelers)	Two-wheelers: 1 Four Wheelers: 0 Others: 0	Two-wheelers: 1 Four Wheelers: 0 Others: 0	Success



Test 1: Parking management system login page

Condition: Username and password must be provided and click sign in button to login and update button to update user details

Test No.	Test Condition	Test Record	Expected Output	Actual Output	Remarks
1.	Username or Password are kept vacant	Username: Password:	Alert: Check your username or password	Alert: Check your username or password	Success
2.	Entry box are filled but username and password are incorrect	Username: parking Password: parking	Alert: Check your username or password	Alert: Check your username or password	Success
3.	Entry box are filled correctly	Username: parking Password: password	The dashboard window opens and gets entry to the system	The dashboard window opens and gets entry to the system	Success
4.	If user forgets the password and press at update button	New username: parking123 New password: parking@12 Confirm password: parking@12	Updates the username and password and update window destroys	Updates the username and password and update window destroys	Success



Test 3: Add Customers

Condition: All the buttons and option menu should work; Customers should be added to the system after filling all the entry box and pressing the button where date and time should be automatically entered

Test No.	Test Condition	Test Record	Expected Output	Actual Output	Remarks
1.	Customer should be added to the system after filling all the required entry box and clicking the button	Full name: Aakash Shahi Vehicle No: 1234 Phone number: 9861101000 Entered time: Auto Slot no: 102 Date: Auto Vehicle type: Others	Text inside Entry box vanishes and customer will be added to system where dashboard shows (Others: 1) and customer info page shows details	Text inside Entry box vanishes and customer will be added to system where dashboard shows (Others: 1) and customer info page shows details	Success
2.	Entry boxes are kept empty partially or fully	Full name: Aakash Shahi Vehicle No: Phone number: Entered time: Auto Slot no: 102 Date: Auto Vehicle type: Others	Alert: Please fill all the details	Alert: Please fill all the details	Success
3.	Refreshing the page	Refresh button pressed	Page will be restarted; entry box detail will be vanished if it is filled and time will be automatically change to current time	Page will be restarted; entry box detail will be vanished if it is filled and time will be automatically change to current time	Success

Test 5: Receipt history and Customer info					
Condition: Customer info displayed after being added and buttons should respond in receipt history window					
Test No.	Test Condition	Test Record	Expected Output	Actual Output	Remarks
1.	All the customer added in the system should be displayed in Customer info page	Details added from add customer page	All the details of particular customer should be displayed in this page	All the details of particular customer should be displayed in this page	Success
2.	All the transaction or receipt history should be displayed on receipt history page	Successful withdraw customer from withdraw customer page	Receipt details should be displayed of all the customers in this page	Receipt details should be displayed of all the customers in this page	Success
3.	Deleting the receipt history with receipt no after clicking button	Enter Receipt No: 1	Receipt matching the receipt no should be deleted	Receipt matching the receipt no should be deleted	Success

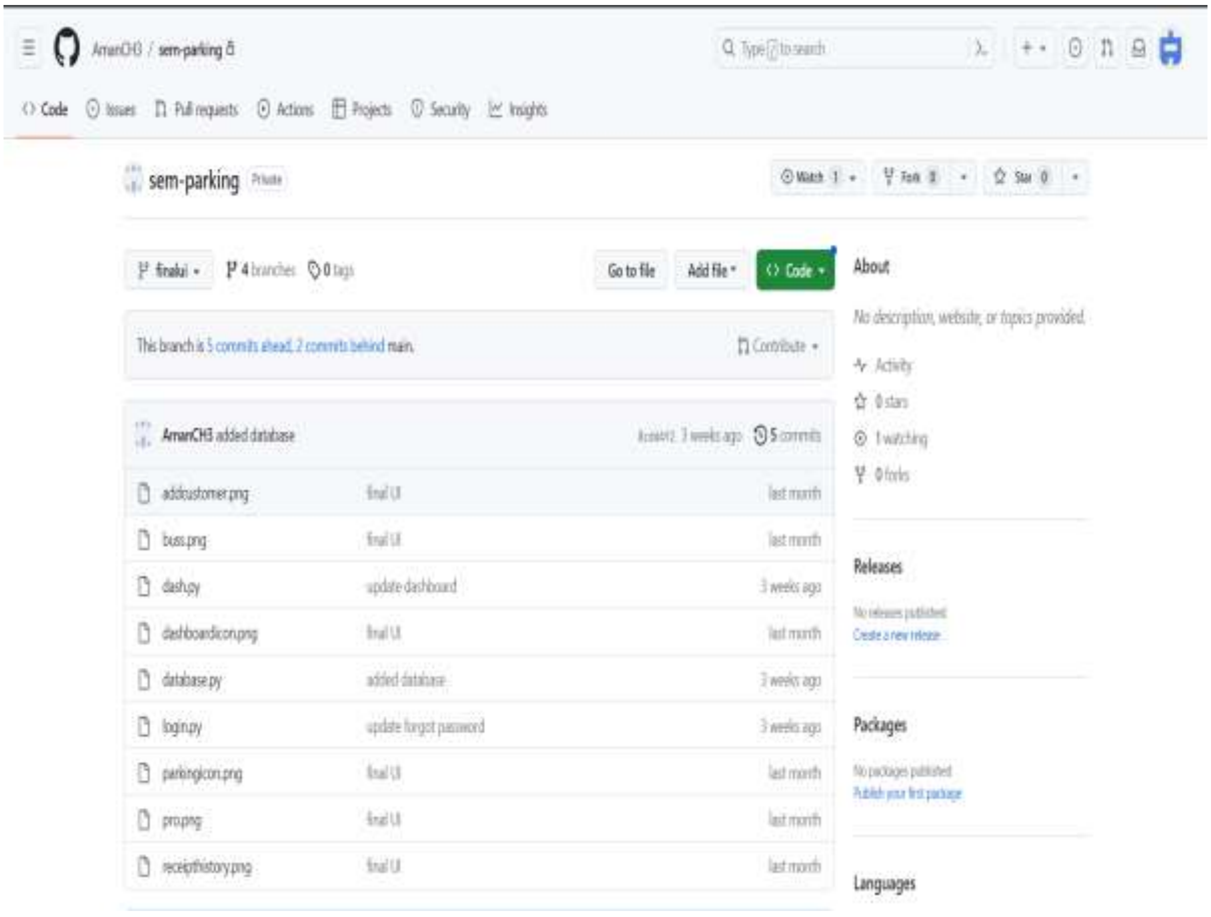
Test 6: Search Bar					
Condition: Customer details should be displayed in new window and feature to update their details					
Test No.	Test Condition	Test Record	Expected Output	Actual Output	Remarks
1.	Customer details will be searched through the entry box and clicking the button	<u>Input:</u> Vehicle no: 1234	<u>Output:</u> New window pops Shows customer details inside entry box	<u>Output:</u> New window pops Shows customer details inside entry box	Success
2.	Update customer details After successful search	Full Name: Aayush Shrestha Vehicle no: 1222 Slot no: 123 Phone No: 9861424363	Updated customer detail successfully	Updated customer detail successfully	Success

Version Control

- **YouTube:** https://youtu.be/Z7grfe7DUCU
- **GitHub:** githublink:https://github.com/AmanCH3/sem-parking/tree/finalui

Figure 27

Git hub



This branch is 5 commits ahead, 2 commits behind main.

AmanCH3 added database · 3 weeks ago · 5 commits

addcustomer.png	final UI	last month
buss.png	final UI	last month
dash.py	update dashboard	3 weeks ago
dashboardicon.png	final UI	last month
database.py	added database	3 weeks ago
login.py	update forgot password	3 weeks ago
parkingicon.png	final UI	last month
pro.png	final UI	last month
receipthistory.png	final UI	last month

About

No description, website, or topics provided.

Activity

0 stars

1 watching

0 forks

Releases

No releases published

Create a new release

Packages

No packages published

Publish your first package

Languages

Conclusion

The team made software that solves the problems of parking by using the latest tools and technologies gaining knowledge from the classroom in the topic of mathematics, Programming and Algorithm, and Software Design. Try to use the full concepts and ideas about the programming and effective tools.

The team project solves the old base pen-paper-based parking management system which is mainly focused on a person's name and contact details. But in this software, we introduced a lot of tools for making software. The system has passed both functional and non-functional requirements. All the errors are correct from the beginning of the project. All testing has successfully passed this system. Data are secure for the customer that is not violent by others. The GUI is also a main part of the system that is so much attractive, effective, and impactful to the customers.

The team make software for the first time and gained a lot of experience from it, making software from scratch is challenging but a chance to learn new things also. All the efforts will lead to a sweet result in the end, good coordination is to solve the projects easily. A tutor is supportive and friendly to students throughout the process. Expanding knowledge about making software and handling challenges smartly is important in the upcoming days for one good IT Cian.

References

Agile Alliance. (2023a, July 29). ***What is Agile? | Agile 101 | Agile Alliance*. Agile Alliance |.** Retrieved August 9, 2023, from https://www.agilealliance.org/agile101/

Free Visual Paradigm online. (n.d.). ***Free Visual Paradigm Online***. Retrieved August 9, 2023, from https://online.visual-paradigm.com/diagrams/solutions/free-visual-paradigm-online/

Rajkumar. (2023, January 1). ***Software Architecture: One-Tier, Two-Tier, Three tier, N tier***. ***Software Testing Material***. Retrieved August 9, 2023, from https://www.softwaretestingmaterial.com/software-architecture/

Byrne, L. (2023, July 4). ***Parking Management System Comprehensive Guide | Workero - Managing and operating workspace. Workero - Managing and operating workspace.*** Retrieved August 9, 2023, from https://www.workero.com/parking-management-system-guide/

Educative. (n.d.). ***What is black-box testing in software testing? Educative: Interactive Courses for Software Developers***. Retrieved August 9, 2023, from https://www.educative.io/answers/what-is-black-box-testing-in-software-testing