

# Visvesvaraya Technological University

Jnana Sangama, Belagavi – 590018



**A Mini Project(BAI586)  
Report on**

## **“PARKING MANAGEMENT SYSTEM”**

*Mini Project Report submitted in partial fulfilment of the requirement for the  
award of the degree of*

**BACHELOR OF ENGINEERING**

**IN**

**ARTIFICIAL INTELLIGENCE & MACHINE LEARNING**

**Submitted by**

**AFRA FURKHEEN  
SANDEEP NAIDU  
SANNIDHI JOSHI  
VANISHA M**

**1KS22AI003  
1KS22AI021  
1KS22AI044  
1KS22AI058**

Under the guidance of  
**Prof. Sudha M, Assistant Professor**

**Department of Artificial Intelligence & Machine Learning K.S.I.T,  
Bengaluru-560109**



**DEPARTMENT OF ARTIFICIAL INTELLIGENCE & MACHINE LEARNING**

**K. S. Institute of Technology**

**#14, Raghuvanahalli, Kanakapura Road, Bengaluru - 560109**

**2024 – 2025**

# K. S. Institute of Technology

#14, Raghuvanahalli, Kanakapura Road, Bengaluru - 560109

## Department of Artificial Intelligence & Machine Learning



### CERTIFICATE

Certified that the Mini Project (**BAI586**) entitled “**PARKING MANAGEMENT SYSTEM**” is a bonafide work carried out by:

**AFRA FURKHEEN**

**1KS22AI003**

**SANDEEP NAIDU**

**1KS22AI021**

**SANNIDHI JOSHI**

**1KS22AI044**

**VANISHA M**

**1KS22AI058**

in partial fulfilment for V semester B.E., Mini Project Work in the branch of Artificial Intelligence & Machine Learning prescribed by **Visvesvaraya Technological University**, Belagavi during the period of September 2024 to February 2025. It is certified that all the corrections and suggestions indicated for internal assessment have been incorporated in the report deposited in the department library. The Mini Project Report has been approved as it satisfies the academic requirements in report of project work prescribed for the Bachelor of Engineering degree.

.....

**Signature of the Guide**

[Prof. Sudha M]

.....

**Signature of the HOD**

[Dr. Suresh M B]

.....

**Signature of Principal**

[Dr. Dilip Kumar K]

## DECLARATION

We, the undersigned students of 5th semester, department of Artificial Intelligence & Machine Learning, KSIT, declare that our Mini Project Work “**PARKING MANAGEMENT SYSTEM**”, is a bonafide work of ours. Our project is neither a copy nor by means a modification of any other engineering project.

We also declare that this project was not entitled for submission to any other university in the past and shall remain the only submission made and will not be submitted by us to any other university in the future.

Place: Bengaluru

Date: 09-12-2024

**Name and USN**

**Signature**

**AFRA FURKHEEN**

.....

**SANDEEP NAIDU**

.....

**SANNIDHI JOSHI**

.....

**VANISHA M**

.....

## ACKNOWLEDGEMENT

The satisfaction and euphoria that accompany the successful completion of any task will be incomplete without the mention of the individuals, we are greatly indebted to, who through guidance and providing facilities have served as a beacon of light and crowned our efforts with success.

First and foremost, our sincere prayer goes to almighty, whose grace made us realize our objective and conceive this project. We take pleasure in expressing our profound sense of gratitude to our parents for helping us complete our Mini Project Work successfully.

We take this opportunity to express our sincere gratitude to our college **K.S. Institute of Technology**, Bengaluru for providing the environment to work on our project.

We would like to express our gratitude to our **MANAGEMENT**, K.S. Institute of Technology, Bengaluru, for providing a very good infrastructure and all the kindness forwarded to us in carrying out this project work in college.

We would like to express our gratitude to **Dr. K.V.A Balaji**, CEO, K.S. Group of Institutions, Bengaluru, for his valuable guidance.

We would like to express our gratitude to **Dr. Dilip Kumar K**, Principal/Director, K.S. Institute of Technology, Bengaluru, for his continuous support.

We like to extend our gratitude to **Dr. Suresh M B**, Professor and Head, Department of Artificial Intelligence & Machine Learning, for providing very good facilities and all the support forwarded to us in carrying out this Project Work Phase-II successfully.

Also, we are thankful to **Prof. Sudha M**, Assistant Professor, for being our Project Guides, under whose able guidance this project work has been carried out and completed successfully.

We are also thankful to the teaching and non-teaching staff of department of Artificial Intelligence & Machine Learning, KSIT for helping us in completing the Project Work Phase II work.

**AFRA FURKHEEN**  
**SANDEEP NAIDU**  
**SANNIDHI JOSHI**  
**VANISHA M**



## **ABSTRACT**

The objective of this project is to enable customers/drivers to reserve a parking space. It is a system that assists individuals, businesses and organisations in managing their parking spaces. It allows the customers/drivers to view the parking status. This python project on Vehicle Parking Management System is mostly concerned with dealing with client parking details such as number and slot. The system also allows vehicle owners to enter information such as their contact information, vehicle number and vehicle category. However, after entering vehicle information, the system creates a reserved slot that lasts until the vehicle leaves. Therefore, the project aimed at solving such problems by designing a desktop-based system that will enable the customers/drivers to make a reservation. . In places where more than 100 vehicles can be parked, this system proves to be useful in reducing wastage of space. This system enables parking of vehicles one after the other thus reducing the space used.

## TABLE OF CONTENTS

<b>Chapter No.</b>	<b>TITLE</b>	<b>Page No.</b>
<b>1</b>	<b>INTRODUCTION</b>	<b>1-2</b>
<b>1.1</b>	Overview of parking	<b>1</b>
<b>1.2</b>	Overview of parking management system	<b>1-2</b>
<b>1.3</b>	Importance and Applications	<b>2</b>
<b>1.4</b>	Objectives	<b>2</b>
<b>2</b>	<b>LITERATURE SURVEY</b>	<b>3-7</b>
<b>2.1</b>	Related Work	<b>3-5</b>
<b>2.2</b>	Existing System	<b>5</b>
<b>2.3</b>	Proposed Work	<b>5-6</b>
<b>2.4</b>	Application and Domain	<b>6</b>
<b>2.5</b>	Challenges and Limitations	<b>7</b>
<b>2.6</b>	Emerging Trends And FutureDirections	<b>7</b>
<b>3</b>	<b>PROBLEM IDENTIFICATION</b>	<b>8-9</b>
<b>3.1</b>	Problem Identification	<b>8</b>
<b>3.2</b>	Goals and objectives	<b>8-9</b>
<b>4</b>	<b>SYSTEM REQUIREMENT SPECIFICATION</b>	<b>10</b>
<b>4.1</b>	Hardware Requirements	<b>10</b>

<b>4.2</b>	Software Requirements	<b>10</b>
<b>5</b>	<b>DESIGN AND IMPLEMENTATION</b>	<b>11</b>
<b>5.1</b>	Design	<b>11</b>
<b>5.2</b>	Methodology	<b>11</b>
<b>6</b>	<b>IMPLEMENTATION</b>	<b>11-17</b>
<b>6.1</b>	Block Diagram	<b>14</b>
<b>6.2</b>	Algorithms	<b>15-17</b>
<b>7</b>	<b>TESTING AND RESULTS</b>	<b>18-37</b>
<b>7.1</b>	Testing and Results	<b>18-35</b>
<b>7.2</b>	Snapshots	<b>36</b>
<b>7.3</b>	Applications	<b>37</b>
<b>7.4</b>	Contribution to Society and Environment	<b>37</b>
<b>8</b>	<b>CONCLUSION</b>	<b>38</b>
<b>9</b>	<b>FUTURE ENHANCEMENTS</b>	<b>39</b>
	<b>REFERENCES</b>	<b>40</b>





### **LIST OF FIGURE**

<b>Fig.No.</b>	<b>FIGURE NAME</b>	<b>Page No.</b>
<b>6.1</b>	<b>Block diagram</b>	<b>12</b>
<b>7.1</b>	<b>Admin Login</b>	<b>36</b>
<b>7.2</b>	<b>Home Page</b>	<b>36</b>
<b>7.3</b>	<b>Adding Vehicle</b>	<b>36</b>
<b>7.4</b>	<b>Managing Vehicle</b>	<b>36</b>



## Chapter 1

# INTRODUCTION

### 1.1 OVERVIEW OF PARKING

College parking is a critical aspect of campus infrastructure, influencing the daily lives of students, faculty, and staff. As college campuses grow in size and enrollment, effective parking management becomes increasingly important. This report will explore the current state of parking on campus, the challenges faced by users, and potential solutions that could enhance the parking experience for everyone involved.

Most college campuses provide a range of parking options, including designated lots for students, faculty, and staff. Parking availability varies by time of day and season, with some lots becoming overcrowded during peak hours, especially in the mornings. The institution typically manages the use of these spaces through a permit system, where individuals must purchase permits to park in specific areas.

### 1.2 OVERVIEW OF PARKING MANAGEMENT SYSTEM

Parking management is a critical concern for many colleges, as campuses often face high demand for limited parking spaces. A well-structured Parking Management System is essential for organizing parking, improving efficiency, and addressing congestion issues. The system ensures that students, faculty, staff, and visitors have access to convenient and reliable parking spaces, thereby enhancing the overall campus experience. This report explores the components, functionalities, challenges, and benefits of implementing an effective parking management system in a college environment.

The primary goal of a College Parking Management System is to optimize the use of parking spaces while minimizing congestion and traffic-related issues on campus. It aims to ensure that parking resources are efficiently allocated, allowing for better traffic flow, reduced frustration for users, and increased accessibility.

### 1.3 IMPORTANCE AND APPLICATIONS

With the growing number of vehicles on college campuses, parking management has become an essential issue for many institutions. Efficiently managing parking resources not only ensures that space is used optimally but also improves overall campus traffic flow. A Parking Management System is an automated system that helps manage parking spaces, ensuring users can quickly find available spots and park without hassle.

A Parking Management System can efficiently allocate parking spaces based on real-time availability and demand, ensuring that no space is left underutilized. This optimization reduces the time users spend searching for a parking spot and ensures that the campus parking resources are used to their fullest potential.

A Parking Management System enhances the overall user experience by simplifying the parking process. Students, faculty, and visitors can easily check parking availability in real-time, either through a mobile app or digital signs placed around the campus. The system can even allow users to reserve parking spaces in advance, ensuring a guaranteed spot when they arrive. This eliminates the uncertainty and stress of finding parking, which is particularly beneficial during busy times.

### 1.4 OBJECTIVES OF REPORT

The primary objective of the College Parking Management System is to automate the management of parking spaces. This includes the allocation, monitoring, and reporting of parking spots for students, faculty, and visitors. By automating the entire process, the system eliminates the inefficiencies and errors associated with manual tracking, ensuring a more accurate and timely approach to managing parking resources.

The system also seeks to provide real-time information on the availability of parking spots. This feature helps users—whether students, faculty, or visitors—know which parking spots are open, reducing time spent looking for available spaces. Real-time updates improve the overall parking experience and help in managing traffic flow by guiding users directly to open spots, thereby preventing unnecessary congestion in parking areas.

## Chapter 2

# LITERATURE SURVEY

## 2.1 RELATED WORK

### “An Enhanced Vehicle Parking Management using Artificial Intelligence”

Managing vehicle parking is a significant challenge, especially in crowded areas. Currently, parking is usually handled manually, which requires a lot of time and effort. This leads to problems like wasted time and extra fuel consumption as drivers struggle to find a parking spot. This paper proposes a solution using Artificial Intelligence (AI) and Wireless Sensor Networks (WSN) to improve parking management and make it more efficient. The paper suggests using **Wireless Sensor Networks** to detect available parking spots. These sensors are placed in parking areas and can sense whether a parking space is occupied or free. The data collected by these sensors is then sent to a central system, which uses **Artificial Intelligence** to analyse the information and make intelligent decisions about parking management. AI can optimize the distribution of vehicles, predict which parking spaces are likely to be free soon, and guide drivers to the best available spots. By using AI to direct drivers to available parking spaces, the system reduces the time drivers spend looking for a spot. This, in turn, reduces the amount of fuel wasted, which is both cost-effective for drivers and environmentally friendly. With AI's ability to predict parking availability, drivers will face less frustration and will spend less time circling around looking for a spot. This leads to a smoother, faster parking experience.

### “Vehicle Parking Management System”

This paper proposes a **Smart Parking System** that integrates **Wireless Sensor Technology** with an **Android Application** to make parking more efficient and convenient for vehicle owners. The proposed parking system combines **Wireless Sensor Technology** and an **Android Application**. Sensors are placed in parking spaces to detect whether a spot is occupied or available. This information is then sent to the system, which is connected to a mobile application. Through the Android app, users can check the availability of parking spaces in real-time and can reserve a parking spot in advance. Once a reservation is made, the vehicle owner receives a **QR code**, which can be scanned upon entering the parking area to confirm their reservation. If the parking area is full, the system also provides information about nearby parking areas, allowing the user to quickly find alternative spaces. By integrating wireless sensors, the system ensures that parking spaces are utilized effectively, reducing the number of empty or underused

spots. If the parking area is full, the system provides information about nearby available parking spaces, helping users quickly find alternatives without wasting time.

### “An Efficient IoT based Smart Vehicle Parking Management System”

The proposed smart parking system uses **IoT (Internet of Things)** technology to monitor parking spaces. Sensors are placed in parking spots to detect whether they are occupied or available. The data collected by these sensors is then processed and sent to the **cloud**, a central online platform that stores and processes this information. Users can access this data via a **mobile application**, which shows them the real-time availability of parking spaces. Additionally, if a parking lot is full, the system can control access to the lot, preventing users from entering and wasting time searching for a spot. This helps manage parking spaces efficiently and reduces unnecessary congestion. The system helps drivers find available parking spots quickly by providing real-time data. This significantly reduces the time spent looking for a parking space. The system uses sensors to detect the availability of parking spots and ensures that spaces are used efficiently. When a parking lot is full, the access is automatically controlled, preventing drivers from entering unnecessarily. Storing data on the cloud allows users to access parking information from anywhere, at any time, making the system more convenient and reliable. By integrating sensors, cloud technology, and a mobile app, the system helps users find parking spots quickly and easily while managing the parking lot more effectively.

### “ParkingKS: Parking Management System Using Open Automatic License Plate Recognition”

The field of **Computer Vision** is a significant area of research in data science and has various applications, such as in **autonomous vehicles**, **machine vision**, and **military technology**. One of its most practical uses is for **license plate recognition**, which can be integrated into systems for toll collection, border control, speed limit monitoring, and **car parking management**. This paper introduces a **web-based car parking management system** that uses computer vision techniques for license plate recognition to manage parking lots efficiently. The system is built using the **Python programming language** and the **Flask web framework**, which allows it to run on a web-based platform. It uses **OpenCV**, a powerful computer vision library, to process images received from cameras. The key function of the system is to recognize **license plates** using **OpenALPR (Automatic License Plate Recognition)** technology. The system is designed to run on a **Linux** operating system with the **Raspbian** platform, making it compatible with low-cost devices like the **Raspberry Pi**. When a vehicle enters the parking lot, the camera captures an image, and the system analyzes the license plate to identify

the vehicle. The system is capable of recognizing **85%** of the images captured by the camera during the experiments.

## 2.2 EXISTING SYSTEM

The existing work in this paper focuses on the challenges of managing vehicle parking, which is traditionally done manually and is time-consuming. Previous approaches have relied on basic methods to track parking spaces, often leading to inefficiency, wasted time, and increased fuel consumption. The paper builds upon existing ideas by integrating *\*Wireless Sensor Networks (WSN)\** and *\*Artificial Intelligence (AI)\** to create a smarter, more automated parking management system. Some systems have used *\*Wireless Sensor Networks (WSN)\** to detect available parking spaces, while others have explored the idea of providing real-time information about parking availability through mobile apps. However, these systems often lacked features like reservation capabilities or seamless integration with user-friendly interfaces. The proposed system in this paper builds on these existing solutions by combining *\*Wireless Sensor Technology\** with an *\*Android Application\**, enabling vehicle owners to *\*\*reserve parking spots remotely, receive \*\*QR codes for entry, and access \*\*alternative parking options\** if the current parking area is full. This integration enhances the efficiency and convenience of parking management compared to earlier methods. Previous solutions have used basic sensor-based systems or manual methods to detect parking space availability, but they often lacked real-time data integration or efficient management of full parking lots. Several *\*IoT-based smart parking systems\** have been developed, which use sensors to monitor parking spaces and provide real-time information to users. Some earlier license plate recognition (LPR) systems use *\*OpenCV\** for image processing but often struggle with accuracy and scalability. These systems typically require expensive hardware and are not always integrated with web-based platforms for easy remote monitoring.

## 2.3 PROPOSED WORK

The proposed **Parking Management System** for a college aims to automate the process of managing parking spaces by using **Python** for backend development and a **Database Management System (DBMS)** for efficient data storage. The system will allow students, faculty, and staff to view real-time availability of parking spaces, reserve spots in advance, and track their parking history. The backend will handle key operations such as updating parking space availability, managing reservations, and storing



user and transaction data through SQL queries in the database. The system will feature user authentication and role-based access for security and ensure real-time updates on parking space occupancy. By integrating **Python** with a **web interface** and a **DBMS**, the system will provide an efficient, cost-effective solution to optimize parking space utilization and reduce congestion on campus.

## 2.4 APPLICATIONS AND DOMAIN

Some key applications and domains of parking management system are given below:

### 1. Vehicle Parking Management in Wireless Sensor Networks (WSN):

- **Application:** Management of parking space in urban areas using AI and WSN.
- **Domain:** Smart Cities, IoT, Traffic Management, Urban Planning.
- **Purpose:** Automates the allocation and optimization of parking spaces based on Space Utilization Factor and Delay for parking.

### 2. Smart Parking System with Wireless Sensor Technology and Android Application:

- **Application:** Reservation and booking of parking slots via mobile app with real-time updates.
- **Domain:** Smart Cities, IoT, Mobile Applications, Traffic Management.
- **Purpose:** Minimizes time and fuel consumption for vehicle owners by enabling remote parking slot booking and providing nearby parking availability.

### 3. IoT-Based Smart Parking System:

- **Application:** Reducing time to find parking and fuel consumption by integrating IoT technology with mobile applications for smart parking.
- **Domain:** IoT, Smart Cities, Traffic Management, Mobile Computing.
- **Purpose:** Provides a solution for efficient parking in urban areas by monitoring parking spaces and directing vehicles to available spots.

### 4. Computer Vision for License Plate Recognition in Parking Management:

- **Application:** License plate recognition for automated vehicle entry/exit in parking lots.
- **Domain:** Computer Vision, AI, Machine Learning, Traffic Management, Security Systems.

- **Purpose:** Uses OpenCV and OpenALPR for vehicle identification and access control in parking lots, toll points, and other locations.

## 2.5 CHALLENGES AND LIMITATIONS

### 1. Vehicle Parking Management in Wireless Sensor Networks (WSN):

- **Challenges:** Scaling up the number of sensors and ensuring their reliability in large areas can be difficult. Wireless interference and sensor malfunction may lead to inaccurate data and disruptions in parking management.
- **Limitations:** High infrastructure costs for setting up and maintaining sensor networks. Limited coverage in certain areas may affect the system's efficiency, especially in densely built environments.

### 2. Smart Parking System with Wireless Sensor Technology and Android Application:

- **Challenges:** Real-time data accuracy from sensors can be affected by signal loss or misreadings, leading to incorrect information. Integrating wireless sensors and mobile applications in a seamless way can be technically challenging.
- **Limitations:** Network coverage issues may impact the reliability of the mobile app, especially in areas with poor signal. The system's effectiveness relies heavily on users having smartphones and data connectivity.

## 2.6 EMERGING TRENDS AND FUTURE DIRECTIONS

- **Integration with AI & Machine Learning:** Leveraging AI and machine learning algorithms to predict parking space availability and optimize parking management dynamically, reducing congestion and increasing efficiency.
- **5G and IoT Integration:** The emergence of 5G technology can significantly enhance the reliability and speed of data transmission for WSN-based parking systems, enabling real-time data collection and faster response times.
- **Mobile App Enhancements:** Increasing reliance on mobile applications to not only book and manage parking slots but also to provide real-time data on parking availability, prices, and dynamic space management.

## Chapter 3

# PROBLEM IDENTIFICATION

### 3.1 PROBLEM IDENTIFICATION

#### 1. Overview

- **Increasing Demand and Limited Space:** The growing number of students, faculty, and visitors at colleges has led to a significant increase in the demand for parking spaces.
- **Inefficient Space Utilization:** Many colleges rely on manual or outdated parking systems, which often result in inefficient use of available parking spaces.
- **Traffic Congestion and Environmental Impact:** Drivers often spend a lot of time circling around the parking area in search of a spot, causing traffic congestion within the campus..

#### 2. Motivation and Importance

- **Efficiency and Space Optimization:** A parking management system helps maximize the utilization of available parking spaces, guiding drivers to open spots, and reducing congestion caused by searching for parking
- **Environmental and Traffic Benefits:** By reducing the time spent circling for a parking space, the system helps cut down on fuel consumption and vehicle emissions, contributing to a greener campus.
- **Enhanced Security and Convenience:** The integration of smart technologies, such as surveillance cameras and automated access control, enhances the security of the parking area, protecting vehicles from theft or damage.

### 3.2 GOALS AND OBJECTIVES

**Goal:** To optimize the use of available parking spaces, reduce congestion, and provide a more efficient, convenient, and sustainable parking experience for students, faculty, and staff.

### Objectives:

- **Real-time Parking Availability:** To provide real-time data on parking space availability, enabling users to quickly find vacant spots.
- **Reservation and Payment Integration:** To allow users to reserve and pay for parking spaces in advance, enhancing convenience and streamlining the process.
- **Security and Monitoring:** To integrate security features such as surveillance cameras and access controls, ensuring the safety of parked vehicles .

## Chapter 4

# SYSTEM REQUIREMENT SPECIFICATION

### 4.1 HARDWARE REQUIREMENTS

- **Processor (CPU):** A multi-core processor is required to handle database queries, parking space management algorithms, and possibly real-time updates.
- **Memory (RAM):** The system needs sufficient RAM to handle the operations of the DBMS and other software.
- **Storage (Hard Disk / SSD):** If the parking management system involves remote communication between vehicles, parking gates, and the server, a stable and fast network connection is essential for real-time updates.

### 4.2 SOFTWARE REQUIREMENTS

- **Operating System (OS):** The operating system provides the foundation for running all software components, supporting the hardware, and offering the necessary stability and security for a college parking management system.
- **Database Management System (DBMS):** The DBMS handles the storage, retrieval, and management of parking data, including user records, parking space availability, and more.
- **Programming Languages and Frameworks:** These technologies are used to build the frontend (client-side) of the PMS, allowing students, staff, and administrators to interact with the system (e.g., through web browsers).
- **Web Server:** A web server is necessary to host the web application, handle requests, and serve web pages to users via browsers.

## Chapter 5

---

## DESIGN AND METHODOLOGY

### 5.1 DESIGN

- **System Architecture:** The PMS follows a client-server architecture with a web-based interface for users and a server-side backend for data processing and management.
- **Frontend Design:** The frontend is built using CSS to provide a dynamic, user-friendly interface for accessing parking details and making reservations.
- **Backend Design:** The backend uses Python (Django) to handle user requests, manage parking slot data, and process reservations.
- **Database Design:** The database stores essential data, including user details, parking slot status, reservations using relational DBMS like MySQL.
- **Entity-Relationship Diagram (ERD):** An ERD models the relationships between entities like Users, Parking Slots, Reservations to ensure proper data management.

### 5.2 METHODOLOGY

- **Data collection:** admin enters the data that gets stored in the database
- **Database design:** SQL to create tables and define relationships
- **SplashScreen method:** graphical representation that appears during the loading of the application
- **Modules:** various modules like Qwidget,QMainWindow,etc which are part of PyQt5 library that provides functions for creating graphical user interfaces



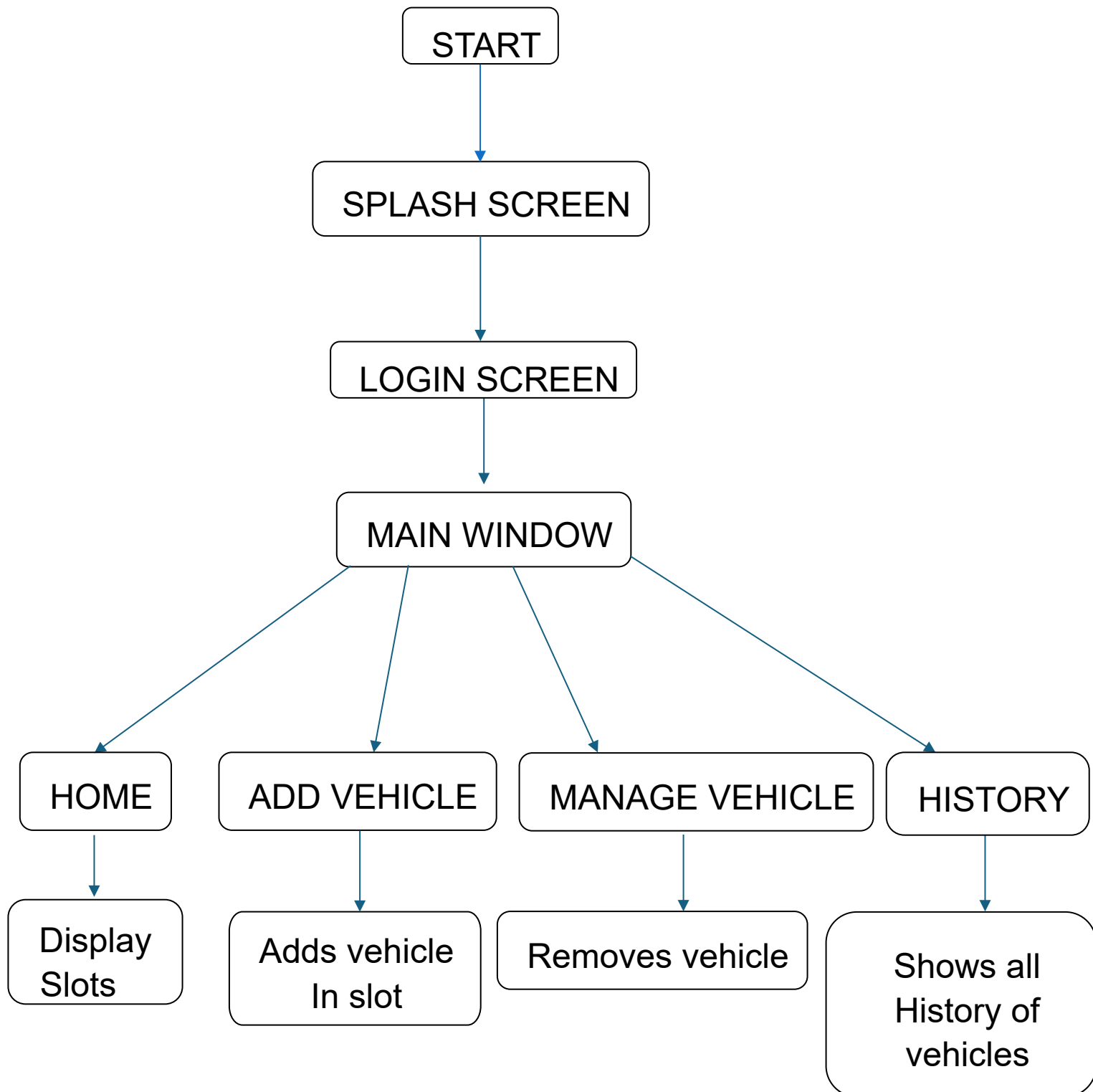
## Chapter 6

---



## IMPLEMENTATION

### 6.1 BLOCK DIAGRAM



### 6.2 Algorithm

BEGIN

// **Step 1:** Start the Application

Initialize the QApplication (app)

Initialize MainScreen, LoginScreen, InstallWindow

// **Step 2:** Show Splash Screen

Show SplashScreen with image "slash\_img.jpg"

Set the splash screen to stay on top of other windows

// **Step 3:** Check for config.json file

IF config.json exists THEN

// **Step 4:** Show Login Screen

WAIT for 3 seconds (use QTimer)

Close SplashScreen

Show Login Screen for Admin login

ELSE

// **Step 5:** Show Installation Screen

WAIT for 3 seconds (use QTimer)

Close SplashScreen

Show InstallWindow for configuration (set up DB, Admin, Parking)

END IF

END

// **Step 6:** Install Window Functionality FUNCTION

showInstallWindow:

// Display all required fields for installation

Display input fields for Database Name, Username, Password, Admin Username, Admin

Password, Two-Wheeler Space, Four-Wheeler Space

```
// Validate Inputs
IF any input is empty THEN

    Show error message: "Please fill all the fields"

    RETURN

// Save configuration to config.json
data = {

    "username": input_db_username.text(),

    "database": input_db_name.text(),

    "password": input_db_password.text()

}

Save data to config.json

// Initialize Database Tables and Insert Data
dbOperation = DBOperation()
dbOperation.CreateTables()

dbOperation.InsertAdmin(input_admin_username.text(), input_admin_password.text())
dbOperation.InsertOneTimeData(int(input_two_wheeler.text()), int(input_four_wheeler.text())) //
Close Install Window and Show Login Screen

    Close InstallWindow

    Show LoginScreen

END

// Step 7: Login Window Functionality FUNCTION
showLoginWindow:

    // Display fields for Username and Password
    Display input fields for Username and Password

    // Validate Login Inputs
    IF Username or Password is empty THEN

        Show error message: "Please Enter Username/Password"

        RETURN
```

```
// Attempt to Log in using provided credentials
dbOperation = DBOperation()
result = dbOperation.doAdminLogin(input_username.text(), input_password.text())
IF result is TRUE THEN
    Show success message: "Login Successful"
    Close Login Screen
    Show HomeScreen (Admin Dashboard)
ELSE
    Show error message: "Invalid Login Details"
END IF
END
```

**// Step 8: Home Screen Functionality (Admin Dashboard) FUNCTION**

```
showHomeScreen:
    // Admin dashboard for managing the parking system
    Display options to view and manage parking spaces, users, or logs
    // Application continues until user decides to exit
END
```

**// Step 9: Exit the Application FUNCTION**

```
exitApplication:
    // Close the application cleanly when the user exits
    Close HomeScreen or any other open window
    EXIT application
END
```

---

## Chapter 7

## TESTING AND RESULTS

### 7.1 TESTING AND RESULTS

```
Dataoperation.py import
mysql.connector import json
from datetime import datetime
class DBOperation():
    def __init__(self):
        file=open("./config.json","r")
        datadic=json.loads(file.read())
        file.close()
        self.mydb=mysql.connector.connect(host="localhost",user="vehicle",passwd="vehicle_password",database="vehicle_parking")
    def CreateTables(self):
        cursor=self.mydb.cursor()
        cursor.execute("DROP TABLE if exists admin")
        cursor.execute("DROP TABLE if exists slots")
        cursor.execute("DROP TABLE if exists vehicles")
        cursor.execute("CREATE TABLE admin (id int(255) AUTO_INCREMENT PRIMARY KEY,username varchar(30),password varchar(30),created_at varchar(30))")
        cursor.execute("CREATE TABLE slots (id int(255) AUTO_INCREMENT PRIMARY KEY,vehicle_id varchar(30),space_for int(25),is_empty int(25))")
        cursor.execute("CREATE TABLE vehicles (id int(255) AUTO_INCREMENT PRIMARY KEY,name varchar(30),mobile varchar(30),entry_time varchar(30),exit_time varchar(30),is_exit varchar(30),vehicle_no varchar(30),vehicle_type varchar(30),created_at varchar(30),updated_at varchar(30))")
        cursor.close()
    def InsertOneTimeData(self,space_for_two,space_for_four):
        cursor=self.mydb.cursor()
        for x in range(space_for_two):
            cursor.execute("INSERT into slots (space_for,is_empty) values ('2','1')")
            self.mydb.commit()
        for x in range(space_for_four):
            cursor.execute("INSERT into slots (space_for,is_empty) values ('4','1')")
            self.mydb.commit()
        cursor.close()
    def InsertAdmin(self,username,password):
```

---

```
        cursor=self.mydb.cursor()        val=(username,password)
cursor.execute("INSERT into admin (username,password) values (%s,%s)",val)
self.mydb.commit()        cursor.close()    def
doAdminLogin(self,username,pasword):
    cursor=self.mydb.cursor()

                                cursor.execute("select          *          from  admin where
username='"+username+"'"    and password='"+pasword+"'")    data=cursor.fetchall()
cursor.close()    if len(data)>0:        return True    else:
    return False
def getSlotSpace(self):
    cursor=self.mydb.cursor()    cursor.execute("select *
from slots")    data=cursor.fetchall()    cursor.close()
return data    def getCurrentVehicle(self):
cursor=self.mydb.cursor() cursor.execute("select * from
vehicles where is_exit='0'") data=cursor.fetchall()
cursor.close()
return data
def getAllVehicle(self):        cursor=self.mydb.cursor()
cursor.execute("select * from vehicles where is_exit='1'")
data=cursor.fetchall()    cursor.close()    return data    def
AddVehicles(self,name,vehiclno,mobile,vehicle_type):
    spacid=self.spaceAvailable(vehicle_type)
if spacid:
    currentdata=datetime.now().strftime("%Y-%m-%d %H:%M:%S")

    data=(name,mobile,str(currentdata),'0',vehiclno,str(currentdata),str(currentdata),vehicle_ty
pe)
    cursor=self.mydb.cursor()

cursor.execute("INSERT                                into                                vehicles
(name,mobile,entry_time,exit_time,is_exit,vehicle_no,created_at,updated_at,vehicle_type)    values
(%s,%s,%s,%s,%s,%s,%s,%s,%s,%s)",data)
```

---

```
        self.mydb.commit()
lastid=cursor.lastrowid
        cursor.execute("UPDATE slots set vehicle_id='"+str(lastid)+"',is_empty='0' where
id='"+str(spacid)+"'")        self.mydb.commit()        cursor.close()        return True        else:
        return "No Space Available for Parking"
def spaceAvailable(self,v_type):
    cursor=self.mydb.cursor()

    cursor.execute("select * from slots where is_empty='1' and space_for='"+str(v_type)+"'")
    data=cursor.fetchall() cursor.close() if len(data)>0:
        return data[0][0]
    else:    return
    False
def exitVehicle(self,id):
    cursor=self.mydb.cursor()

    currentdata = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

    cursor.execute("UPDATE slots set is_empty='1',vehicle_id=" where vehicle_id='"+id+"'")
self.mydb.commit()    cursor.execute("UPDATE vehicles set
is_exit='1',exit_time='"+currentdata+" where id=" + id +
'")
self.mydb.commit()
```

### HomeWindow.py:

```
from PyQt5.QtWidgets import
QWidget,QMainWindow,QPushButton,QLineEdit,QLabel,QVBoxLayout,QHBoxLayout,QFrame,Q
GridLayout,QComboBox,QTableWidget,QTableWidgetItem from DataBaseOperation import
DBOperation from PyQt5.QtWidgets import QHeaderView,qApp import PyQt5.QtGui class
HomeScreen(QMainWindow):    def __init__(self):        super().__init__()
self.setWindowTitle("Home")        self.dbOperation=DBOperation()        widget=QWidget()
widget.setStyleSheet("background:#000")        layout_horizontal=QHBoxLayout()
menu_vertical_layout=QVBoxLayout()        self.btn_home=QPushButton("Home")
```

```
self.btn_add = QPushButton("Add Vehicle") self.btn_manage = QPushButton("Manage Vehicle")
self.btn_history = QPushButton("History")

menu_vertical_layout.setContentsMargins(0,0,0,0)
menu_vertical_layout.setSpacing(0)

self.btn_home.setStyleSheet("width:200px;height:160px;font-
size:20px;background:#1A3668;color:#fff;font-weight:bold;border:1px solid white")

self.btn_add.setStyleSheet("width:200px;height:160px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")

self.btn_manage.setStyleSheet("width:200px;height:160px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")

self.btn_history.setStyleSheet("width:200px;height:160px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")

self.btn_home.clicked.connect(self.showHome)
self.btn_add.clicked.connect(self.showAdd)
self.btn_manage.clicked.connect(self.showManage)
self.btn_history.clicked.connect(self.showHistory)    menu_frame=QFrame()
menu_vertical_layout.addWidget(self.btn_home)
menu_vertical_layout.addWidget(self.btn_add)
menu_vertical_layout.addWidget(self.btn_manage)
menu_vertical_layout.addWidget(self.btn_history)
menu_vertical_layout.addStretch()
menu_frame.setLayout(menu_vertical_layout)
#menu_frame.setMinimumWidth(200)
#menu_frame.setMaximumHeight(200)
parent_vertical=QVBoxLayout()
parent_vertical.setContentsMargins(0,0,0,0)
self.vertical_1=QVBoxLayout()    self.addHomePageData()
self.vertical_2=QVBoxLayout()
```



```
self.vertical_2.setContentsMargins(0,0,0,0)
self.addAddStudentPage()
    self.vertical_3=QVBoxLayout()
self.vertical_3.setContentsMargins(0,0,0,0)
self.addManagePage() self.vertical_4=QVBoxLayout()
self.addHistoryPage()

    self.frame_1=QFrame()
    self.frame_1.setMinimumWidth(self.width())
self.frame_1.setMaximumWidth(self.width())
self.frame_1.setMaximumHeight(self.width())
self.frame_1.setMaximumHeight(self.width())
self.frame_1.setLayout(self.vertical_1)
self.frame_2=QFrame()
self.frame_2.setLayout(self.vertical_2)
self.frame_3=QFrame()
self.frame_3.setLayout(self.vertical_3)
self.frame_4=QFrame()
self.frame_4.setLayout(self.vertical_4)
parent_vertical.addWidget(self.frame_1)
parent_vertical.addWidget(self.frame_2)
parent_vertical.addWidget(self.frame_3)
parent_vertical.addWidget(self.frame_4)
layout_horizontal.addWidget(menu_frame)
layout_horizontal.addLayout(parent_vertical)
layout_horizontal.setContentsMargins(0,0,0,0)
parent_vertical.setContentsMargins(0,0,0,0)
parent_vertical.addStretch()
#menu_vertical_layout.addStretch()
layout_horizontal.addStretch()
```

```
widget.setLayout(layout_horizontal)

self.frame_1.show()      self.frame_2.hide()

self.frame_3.hide()      self.frame_4.hide()

self.setCentralWidget(widget)  def

showHistory(self):

        self.btn_home.setStyleSheet("width:200px;height:160px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
self.btn_add.setStyleSheet("width:200px;height:160px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")

        self.btn_manage.setStyleSheet("width:200px;height:160px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")

        self.btn_history.setStyleSheet("width:200px;height:160px;font-
size:20px;background:#1A3668;color:#fff;font-weight:bold;border:1px solid white")

self.frame_1.hide()      self.frame_2.hide()      self.frame_3.hide()

self.frame_4.show()  def showManage(self):

        self.btn_home.setStyleSheet("width:200px;height:160px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")

        self.btn_add.setStyleSheet("width:200px;height:160px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")

        self.btn_manage.setStyleSheet("width:200px;height:160px;font-
size:20px;background:#1A3668;color:#fff;font-weight:bold;border:1px solid white")

        self.btn_history.setStyleSheet("width:200px;height:160px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")

self.frame_1.hide()      self.frame_2.hide()      self.frame_4.hide()

self.frame_3.show()  def showAdd(self):

        self.btn_home.setStyleSheet("width:200px;height:160px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")

        self.btn_add.setStyleSheet("width:200px;height:160px;font-
size:20px;background:#1A3668;color:#fff;font-weight:bold;border:1px solid white")

        self.btn_manage.setStyleSheet("width:200px;height:160px;font-
```

```
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_history.setStyleSheet("width:200px;height:160px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
self.frame_1.hide()
    self.frame_3.hide()
self.frame_4.hide()
self.frame_2.show()    def
showHome(self):
        self.btn_home.setStyleSheet("width:200px;height:160px;font-
size:20px;background:#1A3668;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_add.setStyleSheet("width:200px;height:160px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_manage.setStyleSheet("width:200px;height:160px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
        self.btn_history.setStyleSheet("width:200px;height:160px;font-
size:20px;background:#A4866F;color:#fff;font-weight:bold;border:1px solid white")
self.frame_2.hide()    self.frame_3.hide()    self.frame_4.hide()
self.frame_1.show()    def refreshHome(self):    while self.gridLayout.count():
    child=self.gridLayout.takeAt(0)
if child.widget():
    child.widget().deleteLater()
row=0    i=0
    alldata=self.dbOperation.getSlotSpace()
for data in alldata:
    label=QPushButton("Slot "+str(data[0])+" \n "+str(data[1]))
if data[3]==1:
        label.setStyleSheet("background-
color:green;color:white;padding:5px;width:100px;height:100px;border:1px    solid
white;textalign:center;font-weight:bold")    else:
```

---

```

label.setStyleSheet("background-
color:red;color:white;padding:5px;width:100px;height:100px;border:1px
solid
white;textalign:center;font-weight:bold")    if i%5==0:
        i=0        row=row+1
self.gridLayout.addWidget(label,row,i)
i=i+1    def addHomePageData(self):
        self.vertical_1.setContentsMargins(0,0,0,0)
button=QPushButton("Refresh Slot")
        button.setStyleSheet("color:#fff;padding:8px
0px;fontsize:20px;background:#696969;border:1px solid white")
button.clicked.connect(self.refreshHome)    vertical_layout=QVBoxLayout()
vertical_layout.setContentsMargins(0,0,0,0)    frame=QFrame()    horizontal=QHBoxLayout()
horizontal.setContentsMargins(0,0,0,0)    vertical_layout.addLayout(horizontal)
alldata=self.dbOperation.getSlotSpace()    self.gridLayout=QGridLayout()
self.gridLayout.setContentsMargins(0,0,0,0)    self.gridLayout.setHorizontalSpacing(0)
self.gridLayout.setVerticalSpacing(0)    vertical_layout.addWidget(button)
vertical_layout.addLayout(self.gridLayout)    row=0    i=0 for data in alldata:
        label=QPushButton("Slot "+str(data[0])+" \n "+str(data[1]))
        if data[3]==1:
                label.setStyleSheet("background-
color:green;color:white;padding:5px;width:100px;height:100px;border:1px
solid
white;textalign:center;font-weight:bold")    else:
                label.setStyleSheet("background-
color:red;color:white;padding:5px;width:100px;height:100px;border:1px
solid
white;textalign:center;font-weight:bold")    if i%5==0:
        i=0        row=row+1
self.gridLayout.addWidget(label,row,i)
i=i+1    frame.setLayout(vertical_layout)
self.vertical_1.addWidget(frame)
self.vertical_1.addStretch()    def

```

```
addAddStudentPage(self):
    layout=QVBoxLayout()
    frame=QFrame()
        name_label=QLabel("Name : ")
        name_label.setStyleSheet("color:#fff;padding:8px 0px;font-size:20px")
    mobile_label=QLabel("Mobile : ")
    mobile_label.setStyleSheet("color:#fff;padding:8px 0px;font-size:20px")
    vechicle_label=QLabel("Vehicle No : ")
    vechicle_label.setStyleSheet("color:#fff;padding:8px 0px;font-size:20px")
    vechicle_type=QLabel("Vehicle Type : ")
    vechicle_type.setStyleSheet("color:#fff;padding:8px 0px;font-size:20px")
    error_label=QLabel("")    error_label.setStyleSheet("color:red;padding:8px
0px;font-size:20px")    name_input=QLineEdit()
    name_input.setStyleSheet("color:#fff;padding:8px 0px;font-size:20px")
    mobile_input=QLineEdit()
    mobile_input.setStyleSheet("color:#fff;padding:8px 0px;font-size:20px")
    vehicle_input=QLineEdit()
    vehicle_input.setStyleSheet("color:#fff;padding:8px 0px;font-size:20px")

        vtype=QComboBox()                vtype.setStyleSheet("color:#fff;padding:8px 0px;font-
size:20px;border:1px solid white")    vtype.addItem("2 Wheeler")    vtype.addItem("4 Wheeler")
    button=QPushButton("Add Vehicle")    button.setStyleSheet("color:#fff;padding:8px 0px;font-
size:20px;background:green;border:1px solid white")    layout.addWidget(name_label)
    layout.addWidget(name_input)                layout.addWidget(mobile_label)
    layout.addWidget(mobile_input)                layout.addWidget(vechicle_label)
    layout.addWidget(vehicle_input)    layout.addWidget(vechicle_type)    layout.addWidget(vtype)
    layout.addWidget(button)    layout.addWidget(error_label)    layout.setContentsMargins(0,0,0,0)
    frame.setMinimumHeight(self.height())                frame.setMinimumWidth(self.width())
    frame.setMaximumHeight(self.width())                frame.setMaximumWidth(self.width())
    layout.addStretch()                frame.setLayout(layout)
    button.clicked.connect(lambda:self.addVehicles(name_input.text(),vehicle_input.text(),mobile_i
```

```

nput.text(),vtype.currentIndex(),error_label))                self.vertical_2.addWidget(frame)                def
addVehicles(self,name,vehiclenu,mobile,index,error_label):
    vtp=2 if
    index==0:
    vtp=2 else:
        vtp=4
    data=self.dbOperation.AddVehicles(name,vehiclenu,mobile,str(vtp))
    if data==True:
        error_label.setText("Added Successfully")
    elif data==False:
        error_label.setText("Failed to Add Vehicle")
else:
    error_label.setText(str(data))                                def    addManagePage(self):
data=self.dbOperation.getCurrentVehicle()                                self.table=QTableWidget()
self.table.setStyleSheet("background:#fff")                self.table.resize(self.width(),self.height())
self.table.setRowCount(len(data))                                self.table.setColumnCount(7)
self.table.horizontalHeader().setSectionResizeMode(QHeaderView.ResizeToContents)
self.table.setHorizontalHeaderItem(0,QTableWidgetItem("ID"))
self.table.setHorizontalHeaderItem(1,QTableWidgetItem("Name"))
self.table.setHorizontalHeaderItem(2,QTableWidgetItem("VEHICLE                No"))
self.table.setHorizontalHeaderItem(3,QTableWidgetItem("MOBILE"))
self.table.setHorizontalHeaderItem(4,QTableWidgetItem("VEHICLE                TYPE"))
self.table.setHorizontalHeaderItem(5,QTableWidgetItem("ENTRY                TIME"))
self.table.setHorizontalHeaderItem(6,QTableWidgetItem("ACTION"))                loop=0                for
smalldata in data:
    self.table.setItem(loop,0,QTableWidgetItem(str(smalldata[0])))
    self.table.setItem(loop,1,QTableWidgetItem(str(smalldata[1])))
    self.table.setItem(loop,2,QTableWidgetItem(str(smalldata[6])))
    self.table.setItem(loop,3,QTableWidgetItem(str(smalldata[2])))
    self.table.setItem(loop,4,QTableWidgetItem(str(smalldata[7])))

```

---

```
self.table.setItem(loop,5,QTableWidgetItem(str(smalldata[3])))
self.button_exit=QPushButton("Exit")
self.button_exit.setStyleSheet("color:#fff;padding:8px      0px;font-
size:20px;background:green;border:1px      solid      white")
self.table.setCellWidget(loop,6,self.button_exit)      self.button_exit.clicked.connect(self.exitCall)
loop=loop+1      frame=QFrame()      layout=QVBoxLayout()      button=QPushButton("Refresh")
button.setStyleSheet("color:#fff;padding:8px 0px;font-size:20px;background:green;border:1px solid
white")      button.clicked.connect(self.refreshManage)      layout.setContentsMargins(0,0,0,0)
layout.setSpacing(0)      layout.addWidget(button)      layout.addWidget(self.table)
frame.setLayout(layout)      frame.setContentsMargins(0,0,0,0)
frame.setMaximumWidth(self.width())      frame.setMinimumWidth(self.width())
frame.setMaximumHeight(self.height())      frame.setMinimumHeight(self.height())
self.vertical_3.addWidget(frame)      self.vertical_3.addStretch()      def refreshManage(self):
data=self.dbOperation.getCurrentVehicle()
self.table.setRowCount(len(data))
self.table.setColumnCount(7)      loop=0
for smalldata in data:
    self.table.setItem(loop,0,QTableWidgetItem(str(smalldata[0])))
    self.table.setItem(loop,1,QTableWidgetItem(str(smalldata[1])))
    self.table.setItem(loop,2,QTableWidgetItem(str(smalldata[6])))
    self.table.setItem(loop,3,QTableWidgetItem(str(smalldata[2])))
    self.table.setItem(loop,4,QTableWidgetItem(str(smalldata[7])))
    self.table.setItem(loop,5,QTableWidgetItem(str(smalldata[3])))
    self.button_exit=QPushButton("Exit")      self.table.setCellWidget(loop,6,self.button_exit)
    self.button_exit.clicked.connect(self.exitCall)
loop=loop+1      def refreshHistory(self):
    self.table1.clearContents()
data=self.dbOperation.getAllVehicle()
loop=0
self.table1.setRowCount(len(data))
```

```

self.table1.setColumnCount(7)    for
smalldata in data:

    self.table1.setItem(loop,0,QTableWidgetItem(str(smalldata[0])))
self.table1.setItem(loop,1,QTableWidgetItem(str(smalldata[1])))
self.table1.setItem(loop,2,QTableWidgetItem(str(smalldata[6])))
self.table1.setItem(loop,3,QTableWidgetItem(str(smalldata[2])))
self.table1.setItem(loop,4,QTableWidgetItem(str(smalldata[7])))
self.table1.setItem(loop,5,QTableWidgetItem(str(smalldata[3])))
self.table1.setItem(loop,6,QTableWidgetItem(str(smalldata[4])))
loop=loop+1    def addHistoryPage(self):

    data=self.dbOperation.getAllVehicle()                self.table1=QTableWidget()
self.table1.resize(self.width(),self.height())           self.table1.setRowCount(len(data))
self.table1.setStyleSheet("background:#fff")             self.table1.setColumnCount(7)
button=QPushButton("Refresh")        button.setStyleSheet("color:#fff;padding:8px 0px;font-
size:20px;background:green;border:1px solid white")    button.clicked.connect(self.refreshHistory)
self.table1.horizontalHeader().setSectionResizeMode(QHeaderView.ResizeToContents)
self.table1.setHorizontalHeaderItem(0,QTableWidgetItem("ID"))

    self.table1.setHorizontalHeaderItem(1,QTableWidgetItem("Name"))
    self.table1.setHorizontalHeaderItem(2,QTableWidgetItem("VEHICLE No"))
    self.table1.setHorizontalHeaderItem(3,QTableWidgetItem("MOBILE"))
    self.table1.setHorizontalHeaderItem(4,QTableWidgetItem("VEHICLE TYPE"))
self.table1.setHorizontalHeaderItem(5,QTableWidgetItem("ENTRY        TIME"))
self.table1.setHorizontalHeaderItem(6,QTableWidgetItem("EXIT        TIME"))
loop=0    for smalldata in data:

    self.table1.setItem(loop,0,QTableWidgetItem(str(smalldata[0])))
self.table1.setItem(loop,1,QTableWidgetItem(str(smalldata[1])))
self.table1.setItem(loop,2,QTableWidgetItem(str(smalldata[6])))
self.table1.setItem(loop,3,QTableWidgetItem(str(smalldata[2])))
self.table1.setItem(loop,4,QTableWidgetItem(str(smalldata[7])))
self.table1.setItem(loop,5,QTableWidgetItem(str(smalldata[3])))

```

---



```
self.table1.setItem(loop,6,QTableWidgetItem(str(smalldata[4])))
loop=loop+1      self.frame5=QFrame()
self.layout1=QVBoxLayout()
self.layout1.setContentsMargins(0,0,0,0)
self.layout1.setSpacing(0)      self.layout1.addWidget(button)
self.layout1.addWidget(self.table1)
self.frame5.setLayout(self.layout1)
self.frame5.setContentsMargins(0,0,0,0)
self.frame5.setMaximumWidth(self.width())
self.frame5.setMinimumWidth(self.width())
self.frame5.setMaximumHeight(self.height())
self.frame5.setMinimumHeight(self.height())
self.vertical_4.addWidget(self.frame5)  self.vertical_4.addStretch()
def exitCall(self): btton=self.sender()
    if btton:
        row=self.table.indexAt(btton.pos()).row()
        id =str(self.table.item(row,0).text())
        self.dbOperation.exitVehicle(id)
        self.table.removeRow(row)
```

**InstallWindow.py:** from PyQt5.QtWidgets import

QWidget,QPushButton,QVBoxLayout,QLabel,QLineEdit from LoginWindow import

LoginScreen import json from DataBaseOperation import DBOperation class

InstallWindow(QWidget): def \_\_init\_\_(self): super().\_\_init\_\_()

self.setWindowTitle("Install Vehical Parking System") self.resize(400,200)

layout=QVBoxLayout()

label\_db\_name=QLabel("Database Name : ")

label\_db\_name.setStyleSheet("color:#000;padding:8px 0px;font-size:18px;")

label\_db\_username=QLabel("Database Username : ")

```
label_db_username.setStyleSheet("color:#000;padding:8px 0px;font-size:18px;")
label_db_password=QLabel("Database Password : ")
label_db_password.setStyleSheet("color:#000;padding:8px 0px;font-size:18px;")
label_admin_username=QLabel("Admin Username : ")
label_admin_username.setStyleSheet("color:#000;padding:8px 0px;font-size:18px;")
label_admin_password=QLabel("Admin Password : ")
label_admin_password.setStyleSheet("color:#000;padding:8px 0px;font-size:18px;")
label_no_of_two_seater=QLabel("No of Two Wheeler Space : ")
label_no_of_two_seater.setStyleSheet("color:#000;padding:8px 0px;font-size:18px;")
label_no_of_four_seater=QLabel("No. of Four Wheeler Space : ")
label_no_of_four_seater.setStyleSheet("color:#000;padding:8px 0px;font-size:18px;")
self.input_db_name=QLineEdit() self.input_db_name.setText("vehicle_parking")
self.input_db_name.setStyleSheet("padding:5px;font-size:17px")
self.input_db_username=QLineEdit() self.input_db_username.setText("vehicle")
self.input_db_username.setStyleSheet("padding:5px;font-size:17px")
self.input_db_password=QLineEdit()
self.input_db_password.setText("vehicle_password")
self.input_db_password.setStyleSheet("padding:5px;font-size:17px")
self.input_admin_username=QLineEdit()
self.input_admin_username.setStyleSheet("padding:5px;font-size:17px")
self.input_admin_password=QLineEdit()
self.input_admin_password.setStyleSheet("padding:5px;font-size:17px")
self.input_two_wheeler=QLineEdit() self.input_two_wheeler.setStyleSheet("padding:5px;font-
size:17px") self.input_four_wheeler=QLineEdit()
self.input_four_wheeler.setStyleSheet("padding:5px;font-size:17px")
buttonsave=QPushButton("save config")
buttonsave.setStyleSheet("padding:5px;font-size:17px;background:green;color:#fff")
self.error_label=QLabel() self.error_label.setStyleSheet("color:red")
layout.addWidget(label_db_name) layout.addWidget(self.input_db_name)
```

---

```
layout.addWidget(label_db_username)    layout.addWidget(self.input_db_username)
layout.addWidget(label_db_password)    layout.addWidget(self.input_db_password)
layout.addWidget(label_admin_username)
layout.addWidget(self.input_admin_username)
layout.addWidget(label_admin_password)
layout.addWidget(self.input_admin_password)
layout.addWidget(label_no_of_two_seater) layout.addWidget(self.input_two_wheeler)
layout.addWidget(label_no_of_four_seater) layout.addWidget(self.input_four_wheeler)
    layout.addWidget(buttonsave)
layout.addWidget(self.error_label)
buttonsave.clicked.connect(self.showStepInfo)
self.setLayout(layout)  def showStepInfo(self):
if self.input_db_name.text()=="":
    self.error_label.setText("Please Enter DB Name")
return    if self.input_db_username.text()=="":
    self.error_label.setText("Please Enter DB Username")
return    if self.input_db_password.text()=="":
    self.error_label.setText("Please Enter DB Password")
return    if self.input_admin_username.text()=="":
    self.error_label.setText("Please Enter Admin Username")
return    if self.input_admin_password.text()=="":
    self.error_label.setText("Please Enter Admin Password")
return    if self.input_two_wheeler.text()=="":
    self.error_label.setText("Please Enter Two Wheeler Space")
return    if self.input_four_wheeler.text()=="":
    self.error_label.setText("Please Enter Four Wheeler Space")
return
    data={"username":self.input_db_username.text(),"database":self.input_db_name.text(),"password":self.input_db_password.text()}    file=open("./config.json","w") file.write(json.dumps(data))
file.close()
```

---

```
dbOperation=DBOperation()
dbOperation.CreateTables()
dbOperation.InsertAdmin(self.input_admin_username.te
xt(),self.input_admin_password.text())
dbOperation.InsertOneTimeData(int(self.input_two_whe
eler.text()),int(self.input_four_wheeler.
text()))    self.close()
self.login=LoginScreen()
self.login.showLoginScreen()
print("Save")
```

```
LoginWindow.py: from PyQt5.QtWidgets import
QWidget,QVBoxLayout,QPushButton,QLabel,QLineEdit,QApplication
import sys from DataBaseOperation import DBOperation from
HomeWindow import HomeScreen class LoginScreen(QWidget):    def
__init__(self):    super().__init__()
self.setWindowTitle("Admin Login")    self.resize(300,100)
layout=QVBoxLayout()
    label_username=QLabel("Username : ")
label_username.setStyleSheet("color:#000;padding:8px 0px;font-size:18px;")
self.input_username=QLineEdit()    self.input_username.setStyleSheet("padding:5px;font-
size:17px")    label_password=QLabel("Password : ")
label_password.setStyleSheet("color:#000;padding:8px 0px;font-size:18px;")
self.error_msg=QLabel()    self.error_msg.setStyleSheet("color:red;padding:8px 0px;font-
size:18px;text-align:center")    self.input_password=QLineEdit()
self.input_password.setStyleSheet("padding:5px;font-size:17px")
btn_login=QPushButton("Login")
    btn_login.setStyleSheet("padding:5px;font-size:20px;background:green;color:#fff")
    layout.addWidget(label_username)
```

```
        layout.addWidget(self.input_username)
layout.addWidget(label_password)
layout.addWidget(self.input_password)
layout.addWidget(btn_login)
layout.addWidget(self.error_msg)
layout.addStretch()
btn_login.clicked.connect(self.showHome)
self.setLayout(layout)    def
showLoginScreen(self):
    self.show()    def
showHome(self):
    if self.input_username.text()=="":
        self.error_msg.setText("Please Enter Username")
return    if self.input_password.text()=="":
        self.error_msg.setText("Please Enter Password")
return
        dboperation=DBOperation()
result=dboperation.doAdminLogin(self.input_username.text(),self.input_password.text())    if
result:
        self.error_msg.setText("Login Successful")
self.close()
        self.home = HomeScreen()
self.home.show()    else:
        self.error_msg.setText("Invalid Login Details")
```

### **MainProgram.py:**

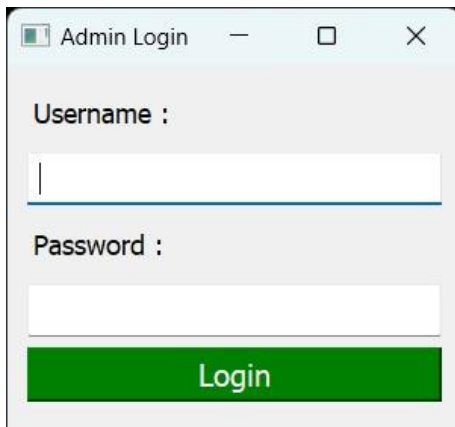
```
import sys
import
os
from InstallWindow import InstallWindow
from
LoginWindow import LoginScreen
```

```
from PyQt5.QtWidgets import QApplication,QSplashScreen,QLabel
from PyQt5.QtGui import QPixmap

from PyQt5.QtCore import Qt,QTimer

class MainScreen():
    def showSplashScreen(self):
        self.pix=QPixmap("slash_img.jpg")
        self.splash=QSplashScreen(self.pix,Qt.WindowStaysOnTopHint)
        self.splash.show()
    def showSetupWindow():
        mainScreen.splash.close()
        installWindow.show()
    def showLoginWindow():
        mainScreen.splash.close()
        login.showLoginScreen()
        app=QApplication(sys.argv)
        login=LoginScreen()
        mainScreen=MainScreen()
        mainScreen.showSplashScreen()
        installWindow=InstallWindow()
        if os.path.exists("./config.json"):
            QTimer.singleShot(3000,showLoginWindow)
        else:
            QTimer.singleShot(3000,showSetupWindow)
        sys.exit(app.exec_())
```

## 7.2 SNAPSHOTS



Admin Login

Username :

Password :

Login

Fig no:7.1 ADMIN LOGIN



Home

Refresh Slot

Slot 1 1	Slot 2 None	Slot 3 None	Slot 4 None	Slot 5 None
Slot 6 None	Slot 7 None	Slot 8 None	Slot 9 None	Slot 10 None
Slot 11 None	Slot 12 None	Slot 13 None	Slot 14 None	Slot 15 None
Slot 16 None	Slot 17 None	Slot 18 None	Slot 19 None	Slot 20 None
Slot 21 None	Slot 22 None	Slot 23 None	Slot 24 None	Slot 25 None
Slot 26 2	Slot 27	Slot 28 None	Slot 29 None	Slot 30 None

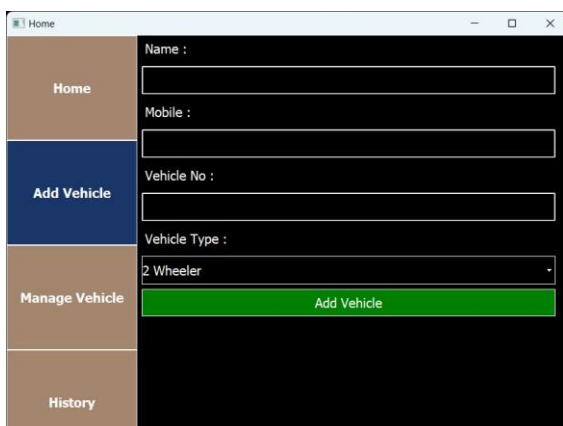
Home

Add Vehicle

Manage Vehicle

History

Fig no:7.2 HOME PAGE



Home

Name :

Mobile :

Vehicle No :

Vehicle Type :

2 Wheeler

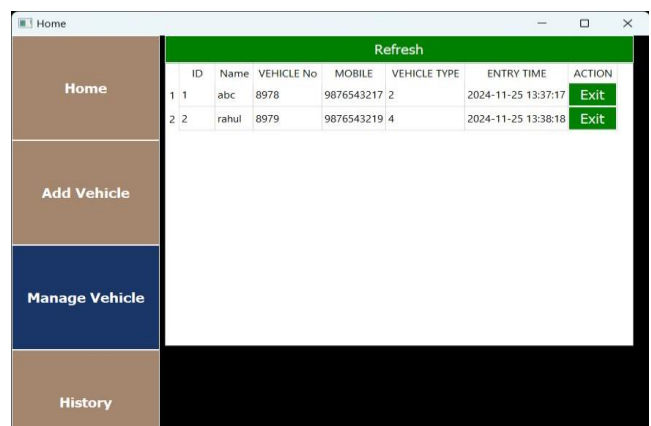
Add Vehicle

Add Vehicle

Manage Vehicle

History

Fig no:7.3 ADDING VEHICLE  
VEHICLE



Home

Refresh

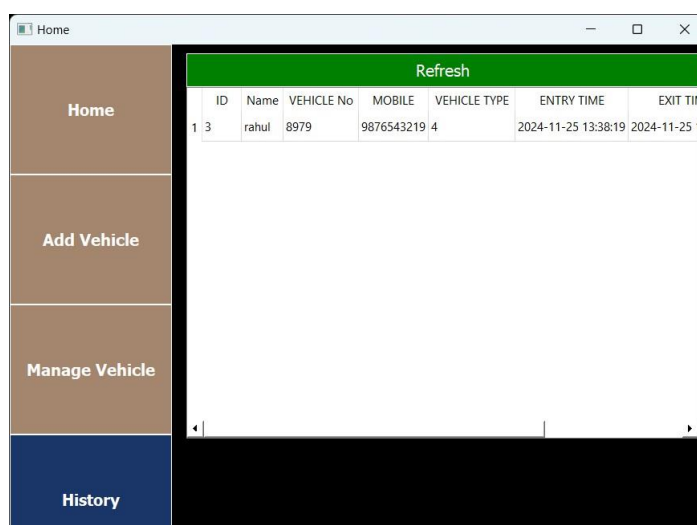
ID	Name	VEHICLE No	MOBILE	VEHICLE TYPE	ENTRY TIME	ACTION	
1	1	abc	8978	9876543217	2	2024-11-25 13:37:17	Exit
2	2	rahul	8979	9876543219	4	2024-11-25 13:38:18	Exit

Add Vehicle

Manage Vehicle

History

Fig no:7.4 MANAGING



Home

Refresh

ID	Name	VEHICLE No	MOBILE	VEHICLE TYPE	ENTRY TIME	EXIT TIME	
1	3	rahul	8979	9876543219	4	2024-11-25 13:38:19	2024-11-25 13:38:19

Add Vehicle

Manage Vehicle

History

Fig no:7.5 HISTORY

## 7.3 APPLICATIONS

1. **Real-Time Parking Availability:** Provides real-time information on available parking spots, reducing search time and traffic congestion.
2. **Student and Faculty Parking Registration:** The system stores and manages parking permits for students and staff, linking parking access with their college database records.
3. **Usage Analytics:** The DBMS stores parking data that can be analyzed for trends, such as peak hours, space usage, and demand patterns.
4. **Maintenance Records:** The DBMS stores records of parking area maintenance, including repair requests, scheduled maintenance, and facility status.

## 7.4 CONTRIBUTION TO SOCIETY AND ENVIRONMENT

1. **Improved Traffic Flow:** Reduces congestion, ensuring smoother movement within and around the campus.
2. **Enhanced Campus Safety:** Increases security by regulating access and ensuring proper parking, reducing accidents and unsafe practices.
3. **Time and Stress Reduction:** Helps users find parking more efficiently, reducing the time spent looking for spaces and lowering stress levels.
4. **Environmental Impact Reduction:** Minimizes fuel consumption and carbon emissions by reducing the time vehicles spend circling for parking.
5. **Encouragement of Sustainable Transport:** Supports electric vehicles and carpooling by providing dedicated spaces, fostering eco-friendly commuting options.
6. **Space Optimization:** Increases the utilization of existing parking areas, reducing the need for expanding parking infrastructure, which preserves green spaces.
7. **Waste Reduction:** Reduces the need for paper-based parking permits or manual management systems, contributing to less paper waste.
8. **Support for Alternative Transport:** Encourages the use of public transport, cycling, or walking by managing parking and reducing dependency on personal vehicles.



## Chapter 8

# CONCLUSION

The **Parking Management System** developed using **DBMS** (Database Management System) and **Python** provides a robust solution for managing and optimizing parking spaces within a college campus. The system effectively addresses key issues such as parking congestion, inefficient space usage, and manual handling of parking operations.

1. **Efficiency and Convenience:** The system automates the entire parking process, from space reservation to fee calculation, making it highly convenient for students, staff, and visitors. By integrating real-time updates and automated access control, it reduces waiting times and ensures smoother traffic flow, contributing to a better user experience.
2. **Data Management and Reporting:** The use of a DBMS ensures centralized storage and management of all parking data, such as user registrations, parking reservations, payment records, and violations. Python's powerful libraries facilitate generating detailed reports, allowing administrators to analyze parking trends, peak usage times, payment summaries, and violation statistics. This data-driven approach helps in making informed decisions for future parking space management.
3. **Scalability and Flexibility:** The system's structure, based on DBMS and Python, ensures scalability, making it adaptable to a growing number of users and parking areas. It can be easily extended with additional features such as integration with mobile apps, electric vehicle charging stations, or enhanced security measures like real-time monitoring.
4. **Cost and Resource Optimization:** By efficiently managing parking spaces and automating various tasks, the system reduces the need for physical infrastructure, minimizes administrative workload, and optimizes space utilization. This leads to significant cost savings for the college while ensuring equitable access to parking for all users.

## Chapter 9

### FUTURE ENHANCEMENTS

- **Mobile App Integration:** Develop a mobile application to allow users to check real-time parking availability, make reservations, and complete payments.
- **Dynamic Pricing Based on Demand:** Implement dynamic pricing where parking rates adjust based on demand, such as during peak hours or special campus events.
- **Integration with Navigation Systems:** Integrate the parking system with GPS or campus navigation apps to guide users to the nearest available parking spot.
- **Electric Vehicle (EV) Charging Stations:** Add dedicated electric vehicle parking spots with integrated charging stations and booking options.
- **AI and Machine Learning for Predictive Analytics:** Incorporate AI/ML to predict parking demand based on data such as historical usage patterns, weather, or events.
- **Automated License Plate Recognition (LPR):** Integrate LPR for automated entry and exit, reducing manual entry and improving access control.
- **Integration with College Management System:** Connect the parking system with the college's student, faculty, and staff databases for automatic permit allocation and access control.
- **Parking Space Sharing and Carpooling:** Implement a feature to allow users to share parking spaces or form carpooling groups, with the system assigning spots accordingly.
- **Advanced Reporting with Data Visualization:** Enhance the reporting system with interactive data visualizations such as heatmaps, graphs, and charts to analyze parking trends.
- **Integration with External Payment Gateways:** Include additional payment methods such as digital wallets, cryptocurrency, or contactless payment systems.

## REFERENCES

1. C. Iswahyudi, B. Firman and U. Lestari, "A magnetometer smart parking sensor: A new model design", *In AIP Conference Proceedings*, vol. 2706, no. 1, 2024, May.
2. A. Balaji, V. P. Srinivasan, J. Rangarajan, V. Nagaraju, B. Bharathi and S. Murugan, "Smart Technique to Prevent Flood Disaster due to High Rainfall in Coastal Areas using IoT", *International Conference on Intelligent and Innovative Technologies in Computing Electrical and Electronics*, pp. 1252-1256, 2023.
3. A. Farid, F. Hussain, K. Khan, M. Shahzad, U. Khan and Z. Mahmood, "A fast and accurate real-time vehicle detection method using deep learning for unconstrained environments", *Applied Sciences*, vol. 13, no. 5, pp. 3059, 2023.
4. H. Zulficar, H. M. Ul Haque, F. Tariq and R. M. Khan, "A survey on smart parking systems in urban cities", *Concurrency Comput. Pract. Exper.*, vol. 35, no. 15, Jul. 2023.