

PARKING MANAGEMENT SYSTEM

A MINI PROJECT REPORT

Submitted by

MOHAMMED 220701515
REHAN SHARIEF
MT

SATHISH S 220701526

In partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING
IN
COMPUTER SCIENCE AND ENGINEERING

RAJALAKSHMI ENGINEERING COLLEGE (AUTONOMOUS)

THANDALAM

CHENNAI-602105

2023 - 24

BONAFIDE CERTIFICATE

Certified that this project report “**PARKING MANAGEMENT SYSTEM**” is the bonafide work of “**MOHAMMED REHAN SHARIEF MT (220701515), SATHISH S (220701526)**”

who carried out the project work under my supervision.

Submitted for the Practical Examination held on _____

SIGNATURE

Mrs.K. MAHESMEENA
Assistant Professor (SG),
Computer Science and Engineering
Rajalakshmi Engineering College,
(Autonomous),
Thandalam, Chennai - 602 105

INTERNAL EXAMINER

EXTERNAL EXAMINER

ABSTRACT

The Parking Management System (PMS) is developed to streamline the management of parking spaces in various facilities, such as shopping malls, office complexes, and public parking areas. This system provides an efficient way to track parking space availability, record vehicle details, and handle payment processes, ensuring a smooth and organized parking experience for users.

The primary objective of the PMS is to offer an easy-to-use, reliable, and scalable solution for parking facility management. It focuses on automating data entry, enhancing data retrieval efficiency, and improving overall parking operations through a centralized database and a user-friendly interface.

The system features a secure login mechanism to ensure that only authorized personnel can manage and view the parking data. The integration with a MySQL database allows for the storage and management of parking details and vehicle information in a structured manner.

Administrators can add new parking records, including details such as parking ID, name, level, free space availability, vehicle number, duration of parking, and payment. The system also provides a treeview widget to display all parking records in a structured format, facilitating easy browsing and inspection of data. Additionally, the PMS captures vehicle-specific details, such as vehicle ID, model name, and date of purchase, and allows for the removal of vehicle records from the database.

TABLE OF CONTENTS

S.NO	TITLE	PAGE. NO
1.	INTRODUCTION	2
	1.1. OBJECTIVES	2
	1.2. MODULE	2
2.	SURVEY OF TECHNOLOGIES	3
	2.1.SOFTWARE DESCRIPTION	3
	2.2. LANGUAGE	3
	2.2.1. SQL	3
	2.2.2. PYTHON	4
3.	REQUIREMENTS AND ANALYSIS	4
	3.1 REQUIREMENT SPECIFICATIONS ⁵	5
	3.2 HARDWARE AND SOFTWARE REQUIREMENTS ⁶	5
	3.3 DATA FLOW DIAGRAM ⁶	6
	3.4 DATA DICTIONARY ⁶	7
	3.5 ER-DIAGRAM ⁶	8
	3.6 NORMALIZATION ⁶	8
4.	PROGRAM CODE	11
5.	RESULT AND DISCUSSION	24
6.	TESTING	29
7.	CONCLUSION	30
8.	FUTURE ENHANCEMENTS	30

1. INTRODUCTION

The Parking Management System (PMS) is an innovative solution designed to address the growing challenges associated with managing parking facilities in urban areas, commercial complexes, and public spaces. With the increasing number of vehicles on the road, the demand for efficient parking management has become more critical than ever. Traditional methods of parking management, which often rely on manual record-keeping and monitoring, are not only inefficient but also prone to errors and mismanagement. PMS aims to modernize this aspect of urban infrastructure by introducing a digital, automated approach to parking management.

1.1. OBJECTIVES

The primary objective of the Parking Management System (PMS) is to provide a robust, efficient, and user-friendly solution for managing parking facilities. The system aims to automate the recording and retrieval of parking data, reducing reliance on manual processes and ensuring accurate maintenance of information. By offering real-time tracking of parking space availability and a straightforward data entry interface, PMS enhances the user experience for both administrators and end-users, facilitating quick and easy management of parking records. Additionally, PMS incorporates a secure login mechanism to restrict access to authorized personnel only, thereby protecting sensitive data from unauthorized access and maintaining data accuracy. The system enables administrators to monitor and analyze parking space usage patterns, optimizing space allocation and utilization, thus reducing congestion and improving overall efficiency.

1.2. MODULES

- Database Module
- Parking Management Module
- New vehicle Management Module
- Add vehicle details Management Module

2. SURVEY OF TECHNOLOGIES

2.1. SOFTWARE DESCRIPTION

PyCharm is an Integrated Development Environment (IDE) specifically designed for Python development. PyCharm provides a comprehensive set of tools for coding, debugging, testing, and deploying Python applications. It offers a user-friendly interface and a wide range of features. One of the key features of PyCharm is its powerful code editor, which supports syntax highlighting, code completion, and code analysis, helping developers write clean and error-free code more efficiently. It also comes with built-in support for version control systems like Git, making it easy to manage code repositories directly from the IDE.

2.2. LANGUAGE

2.2.1. SQL

SQL, or Structured Query Language, is a powerful and widely-used programming language designed for managing and manipulating relational databases. It provides a standardized way to interact with databases, allowing users to perform various tasks such as querying data, updating records, and defining database structures. SQL is essential for creating, retrieving, updating, and deleting data from databases. One of the key features of SQL is its ability to perform complex queries on large datasets efficiently. Using SQL, users can write queries to retrieve specific information from databases based on specified criteria, such as selecting all students enrolled in a particular course or calculating the average grade for a group of students. SQL also provides mechanisms for data manipulation, including inserting new records into a database, updating existing records, and deleting unwanted data. This makes it a versatile tool for managing data throughout its lifecycle. Additionally, SQL allows for the creation and management of database structures such as tables, indexes, views, and stored procedures. These structures help organize and optimize data storage and retrieval, ensuring efficient operation of the database system.

2.2.2. PYTHON

Python is a high-level, interpreted programming language known for its simplicity and readability. Guido van Rossum released Python in 1991, and since then, it has gained immense popularity in various fields, including web development, data analysis, artificial intelligence, and scientific computing. One of Python's key strengths lies in its clean and concise syntax, which allows developers to write code that is easy to understand and maintain. Python supports multiple programming paradigms, including procedural, object-oriented, and functional programming, making it versatile for a wide range of applications. It comes with a comprehensive standard library that provides ready-to-use modules and functions for tasks such as file I/O, networking, and data processing, reducing the need for external dependencies. Python's dynamic typing and automatic memory management simplify development, as programmers do not need to declare variable types explicitly, and memory allocation and deallocation are handled by the interpreter. This feature contributes to Python's flexibility and ease of use, especially for beginners.

3. REQUIREMENTS AND ANALYSIS

3.1. REQUIREMENT SPECIFICATION

The Parking Management System (PMS) requires a secure user authentication interface, efficient database management for storing and retrieving parking and vehicle data, and a user-friendly GUI for administrators to manage records :

- **USER AUTHENTICATION :**

The system must provide a secure login interface to authenticate users. Only authorized personnel should have access to the system's functionalities.

- **DATABASE MANEAGEMENT :**

The Parking Management System (PMS) must connect to a MySQL database to store and retrieve data efficiently. It should create and manage two primary tables: parkmaster12 for parking details and vehicle for vehicle information.

- **PARKING DATA MANAGEMENT :**

The Parking Management System (PMS) should enable administrators to add new parking records, including details such as parking ID, name, level, free space availability, vehicle number, duration of parking, and payment.

- **REPORTING :**

The system should generate detailed reports on parking usage and revenue, providing valuable insights for administrators.

3.2. HARDWARE AND SOFTWARE REQUIREMENTS

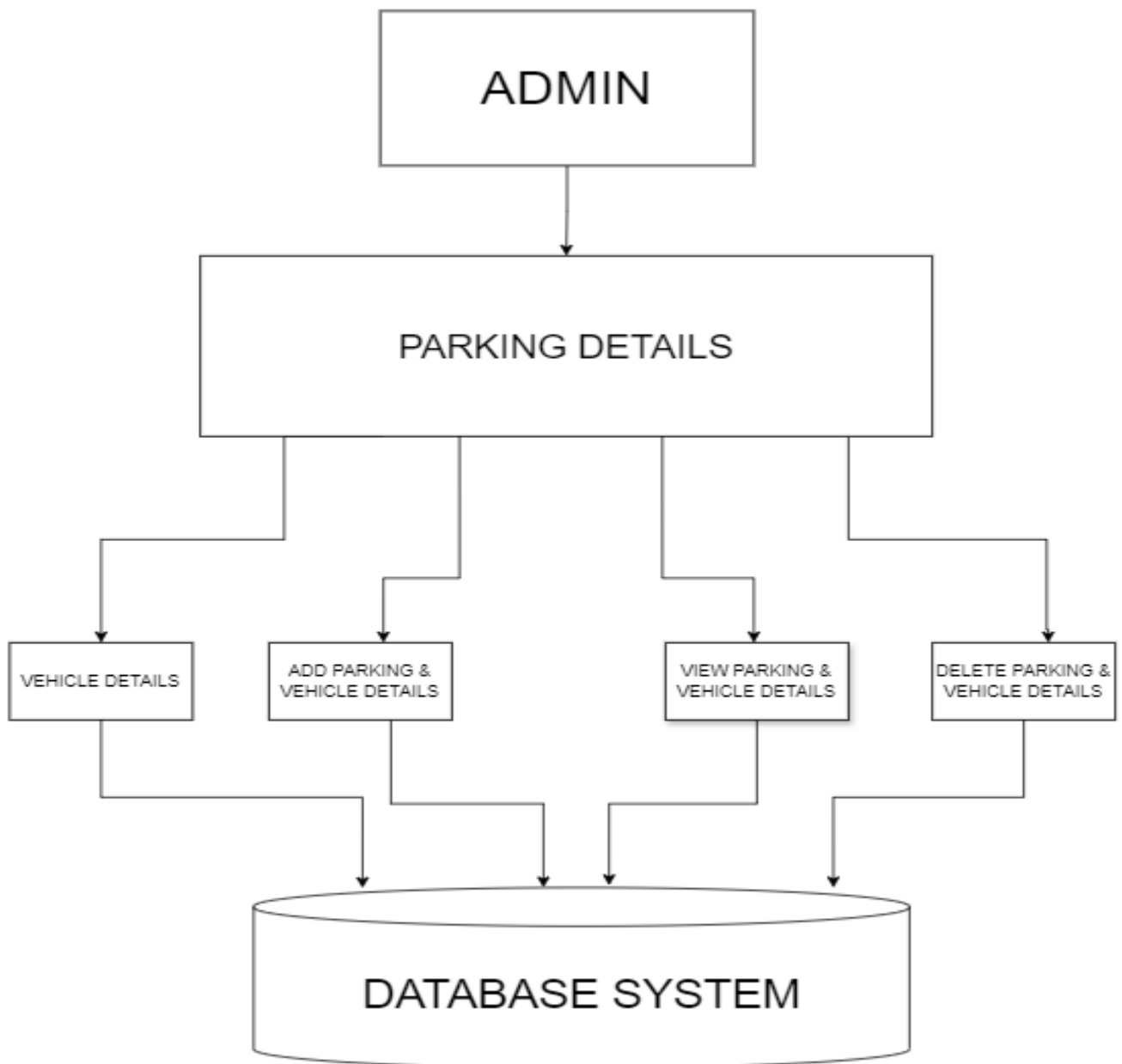
Software Requirements

- Operating System Windows 11
- Front End: Python
- Back End: MySQL

Hardware Requirements

- Desktop PC or a Laptop
- Operating System – Windows 10 , 64-bit operating system
- Intel® Core™ i3-6006U CPU @ 2.00GHz
- 4.00 GB RAM
- Keyboard and Mouse

3.3. DATA FLOW DIAGRAM



3.4. DATA DICTIONARY

- **ADMIN**

Column Name	Data Type	Description
Admin_id	INT	Unique admin ID
username	VARCHAR(30)	Unique,Not Null
Password	VARCHAR(64)	Unique,Not Null
Created_at	DATETIME	Not Null,Default : CURRENT_TIMESTAMP

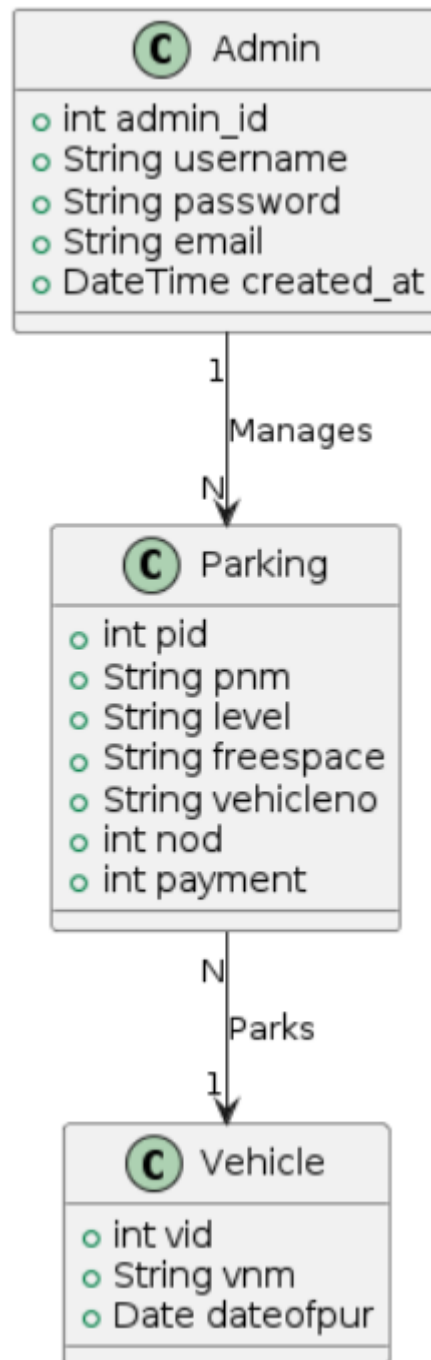
- **PARKMASTER12**

Column Name	Data Type	Description
Pid	INT	Unique parking ID
Pnm	VARCHAR(30)	Parking name
Level	VARCHAR(30)	Parking level
Freespace	VARCHAR(30)	Indicates if space is free
Vehicle no	VARCHAR(30)	Vehicle number
nod	INT	Number of days parked
Payment	INT	Payment amount

- **VECHICLE**

Column Name	Data Type	Description
Vid	INT	Unique vehicle ID
vnm	VARCHAR(30)	Vehicle name
dateofpur	DATE	Date of purchase

3.5. ER DIAGRAM



3.6. NORMALIZATION

First Normal Form (1NF)

ADMIN TABLE :

Column Name	Data Type	Constraints
admin_id	INT,Primary key	PRIMARY KEY, AUTO_INCREMENT
username	VARCHAR(30)	UNIQUE,NOT NULL
Password	VARCHAR(64)	NOT NULL
email	VARCHAR(100)	UNIQUE,NOT NULL
Created_at	DATETIME	NOT NULL,DEFAULT CURRENT_TIMESTAMP

PARKING :

Column Name	Data Type	Constraints
pid	INT	PRIMARY KEY, AUTO_INCREMENT
pnm	VARCHAR(100)	
level	TEXT	
freespace	INT	

VEHICLE :

Column Name	Data Type	Constraints
vid	INT	PRIMARY KEY, AUTO_INCREMENT
vnm	INT	
dateofpur	INT	

Second Normal Form (2NF)

A table is in 2NF if:

- It is in 1NF.
- All non-key attributes are fully functionally dependent on the primary key.

Our tables already comply with 2NF:

- In Admin, all attributes depend on admin_id.
- In Parking, all attributes depend on pid.

- In Vehicle, all attributes depend on vid.

` Third Normal Form (3NF) :

A table is in 3NF if:

- It is in 2NF.
- There are no transitive dependencies, meaning non-key attributes do not depend on other non-key attributes.

Your tables already comply with 3NF:

- There are no transitive dependencies in Admin.
- There are no transitive dependencies in Parking.
- There are no transitive dependencies in Vehicle.

4. PROGRAM CODE

PYTHON CODE:

```
import os

import platform

import mysql.connector

from mysql.connector

import Error

import tkinter as tk

from tkinter import messagebox, ttk

def create_connection():

    try:

        connection = mysql.connector.connect(

            host="localhost",

            user="rehan",

            password="rehan@sql2005",

            database="PARKINGSYSTEM"

        )

        if connection.is_connected():

            print("Connected to MySQL database")

            return connection
```

```

except Error as e:

    print(f"Error: {e}")

    return None

mydb = create_connection()

if not mydb:

    exit(1)

mycursor = mydb.cursor()

def create_tables():

    table_creation_query_parkmaster12 = """

    CREATE TABLE IF NOT EXISTS parkmaster12 (

        pid INT PRIMARY KEY,

        pnm VARCHAR(30) NOT NULL,

        level VARCHAR(30) NOT NULL,

        freespace VARCHAR(30) NOT NULL,

        vehicleno VARCHAR(30) NOT NULL,

        nod INT NOT NULL, payment INT NOT NULL

    );

    """ table_creation_query_vehicle = """

    CREATE TABLE IF NOT EXISTS vehicle (

        vid INT PRIMARY KEY,

```



```

vnm VARCHAR(30) NOT NULL,

dateofpur DATE NOT NULL

); """

try:

    mycursor.execute(table_creation_query_parkmaster12)
    mycursor.execute(table_creation_query_vehicle) mydb.commit()

except Error as e:

    print(f"Error creating tables: {e}")

create_tables()

def show_data_window():

    def refresh_data():

        try:

            mycursor.execute("SELECT * FROM parkmaster12")

            rows = mycursor.fetchall()

            for row in tree.get_children():

                tree.delete(row)

            for row in rows:

                tree.insert("", "end", values=row)

        except Error as e:

            messagebox.showerror("Error", f"Error: {e}")

    def open_add_record_window():

```

```

def add_parking_record():

    try:

        pid = int(pid_entry.get())

        pnm = pnm_entry.get()

        level = level_entry.get()

        freespace = freespace_entry.get()

        vehiclenu = vehiclenu_entry.get()

        nod = int(nod_entry.get())

        payment = min(max(nod * 20, 20), 120)

        record = (pid, pnm, level, freespace, vehiclenu, nod, payment)

        sql = ('INSERT INTO parkmaster12 (pid, pnm, level, freespace, vehiclenu, nod, payment)
        ' 'VALUES (%s, %s, %s, %s, %s, %s, %s)')

        mycursor.execute(sql, record)

        mydb.commit()

        messagebox.showinfo("Success", "Record added successfully.")

        add_window.destroy()

        refresh_data()

    except Error as e:

        messagebox.showerror("Error", f"Error: {e}")

    add_window = tk.Toplevel(data_window)

    add_window.title("Add New Parking Record") add_window.configure(bg='black')

```

```

window_width, window_height = 400, 300

screen_width = add_window.winfo_screenwidth()

screen_height = add_window.winfo_screenheight()

position_top = int(screen_height / 2 - window_height / 2)

position_right = int(screen_width / 2 - window_width / 2)
add_window.geometry(f'{window_width}x{window_height}+{position_right}+{position_top}')

form_frame = tk.Frame(add_window, bg='black')

form_frame.pack(fill=tk.X, padx=10, pady=10)

tk.Label(form_frame, text="Parking ID:", bg='black', fg='white').grid(row=0, column=0,
padx=5, pady=5)

pid_entry = tk.Entry(form_frame)

pid_entry.grid(row=0, column=1, padx=5, pady=5)

tk.Label(form_frame, text="Parking Name:", bg='black', fg='white').grid(row=1,
column=0, padx=5, pady=5)

pnm_entry = tk.Entry(form_frame)

pnm_entry.grid(row=1, column=1, padx=5, pady=5)

tk.Label(form_frame, text="Level:", bg='black', fg='white').grid(row=2, column=0,
padx=5, pady=5)

level_entry = tk.Entry(form_frame)

level_entry.grid(row=2, column=1, padx=5, pady=5)

tk.Label(form_frame, text="Free Space (Y/N):", bg='black', fg='white').grid(row=3,

```

```
column=0, padx=5, pady=5)
```

```
freespace_entry = tk.Entry(form_frame)
```

```
freespace_entry.grid(row=3, column=1, padx=5, pady=5)
```

```
tk.Label(form_frame, text="Vehicle No:", bg='black', fg='white').grid(row=4, column=0,  
padx=5, pady=5)
```

```
vehicleno_entry = tk.Entry(form_frame)
```

```
vehicleno_entry.grid(row=4, column=1, padx=5, pady=5)
```

```
tk.Label(form_frame, text="No of Days:", bg='black', fg='white').grid(row=5, column=0,  
padx=5, pady=5) nod_entry = tk.Entry(form_frame) nod_entry.grid(row=5, column=1,  
padx=5, pady=5)
```

```
tk.Button(form_frame, text="Add Record", command=add_parking_record, bg='white',  
fg='black').grid(row=6, column=0, columnspan=2, pady=10)
```

```
data_window = tk.Tk()
```

```
data_window.title("Parking Management System")
```

```
data_window.configure(bg='black')
```

```
window_width, window_height = 600, 400
```

```
screen_width = data_window.winfo_screenwidth()
```

```
screen_height = data_window.winfo_screenheight()
```

```
position_top = int(screen_height / 2 - window_height / 2) position_right =  
int(screen_width / 2 - window_width / 2)
```

```
data_window.geometry(f'{window_width}x{window_height}+{position_right}+{positio  
n_top}')
```

```

columns = ("pid", "pnm", "level", "freespace", "vehicleno", "nod", "payment")

tree = ttk.Treeview(data_window, columns=columns, show='headings')
tree.heading("pid", text="Parking ID")

tree.heading("pnm", text="Parking Name")

tree.heading("level", text="Level")

tree.heading("freespace", text="Free Space")

tree.heading("vehicleno", text="Vehicle No")

tree.heading("nod", text="No of Days")

tree.heading("payment", text="Payment")

tree.pack(fill=tk.BOTH, expand=True)

refresh_data()

tk.Button(data_window, text="Add New Record", command=open_add_record_window,
bg='white', fg='black').pack(pady=10)

data_window.mainloop()

def login_page():

def on_enter(event):

    event.widget['background'] = 'white'

def on_leave(event):

    event.widget['background'] = 'SystemButtonFace'

def authenticate():

    username = username_entry.get()

```

```

password = password_entry.get()

if username == "admin" and password == "admin":

    root.destroy()

    show_data_window()

else: messagebox.showerror("Login Failed", "Invalid username or password")

root = tk.Tk()

root.title("PARKMASTER")

root.configure(bg='black')

window_width, window_height = 300, 200

screen_width = root.winfo_screenwidth()

screen_height = root.winfo_screenheight()

position_top = int(screen_height / 2 - window_height / 2) position_right =
int(screen_width / 2 - window_width / 2)

root.geometry(f'{ window_width }x{ window_height }+{ position_right }+{ position_top }')

tk.Label(root, text="Username", bg='black', fg='white').grid(row=0, column=0, padx=10,
pady=10)

username_entry = tk.Entry(root)

username_entry.grid(row=0, column=1, padx=10, pady=10)

tk.Label(root, text="Password", bg='black', fg='white').grid(row=1, column=0, padx=10,
pady=10)

password_entry = tk.Entry(root, show="*")

```

```
password_entry.grid(row=1, column=1, padx=10, pady=10)

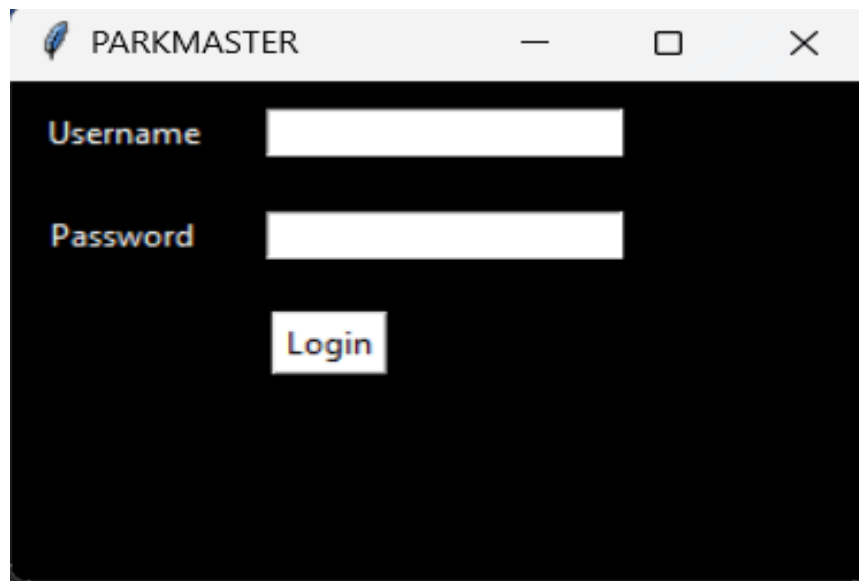
login_btn = tk.Button(root, text="Login",

command=authenticate, bg='white', fg='black')

login_btn.grid(row=2, column=0, columnspan=2, pady=10)
username_entry.bind("<Enter>", on_enter) username_entry.bind("<Leave>", on_leave)
password_entry.bind("<Enter>", on_enter) password_entry.bind("<Leave>", on_leave)
root.mainloop()

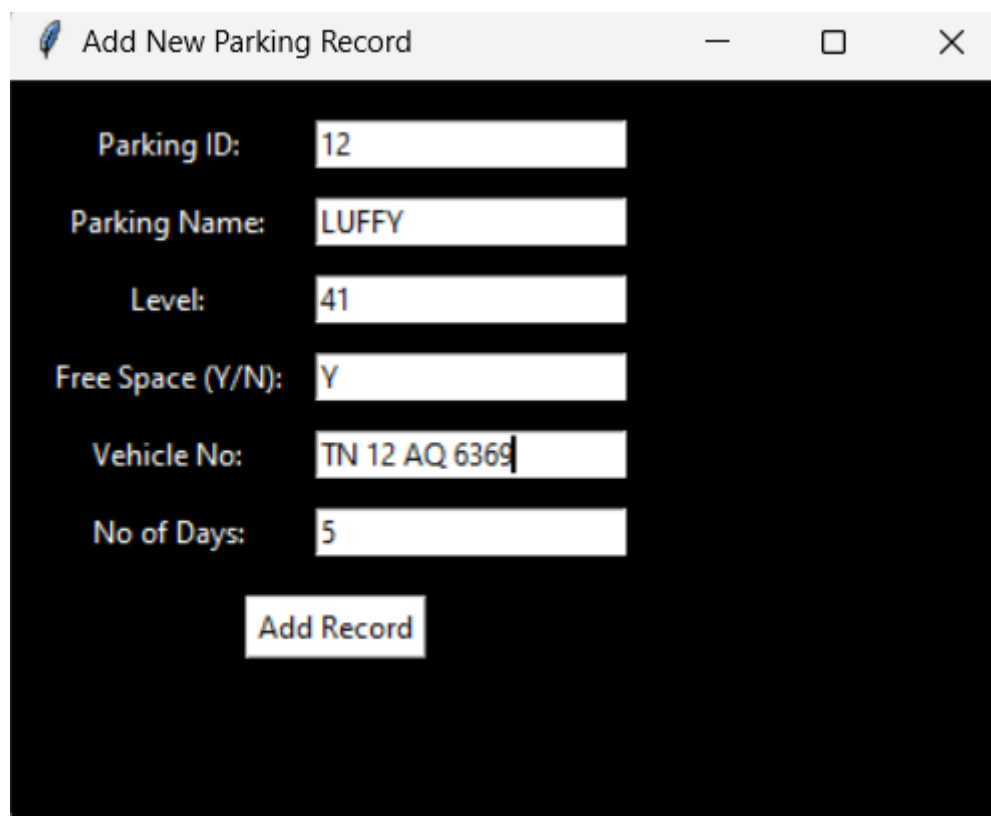
login_page()
```

5. RESULT AND DISCUSSION



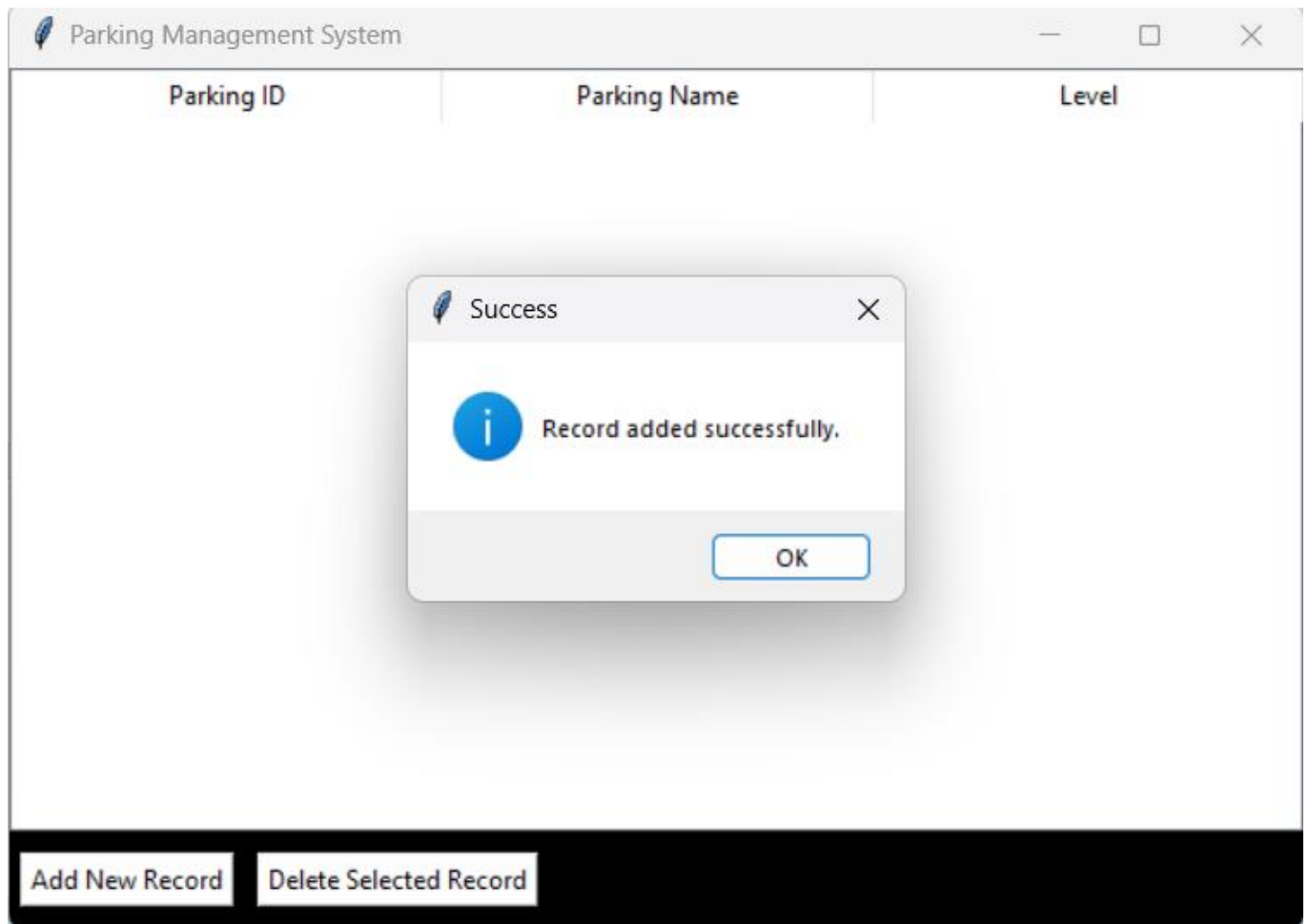
A screenshot of a web application window titled "PARKMASTER". The window has a dark background. It contains two text input fields: "Username" and "Password". Below the "Password" field is a "Login" button.

Username	<input type="text"/>
Password	<input type="password"/>
<input type="button" value="Login"/>	



A screenshot of a web application window titled "Add New Parking Record". The window has a dark background. It contains six text input fields: "Parking ID:", "Parking Name:", "Level:", "Free Space (Y/N):", "Vehicle No:", and "No of Days:". Below these fields is an "Add Record" button.

Parking ID:	<input type="text" value="12"/>
Parking Name:	<input type="text" value="LUFFY"/>
Level:	<input type="text" value="41"/>
Free Space (Y/N):	<input type="text" value="Y"/>
Vehicle No:	<input type="text" value="TN 12 AQ 6369"/>
No of Days:	<input type="text" value="5"/>
<input type="button" value="Add Record"/>	



Parking Management System			
Parking ID	Parking Name	Level	
12	LUFFY	41	

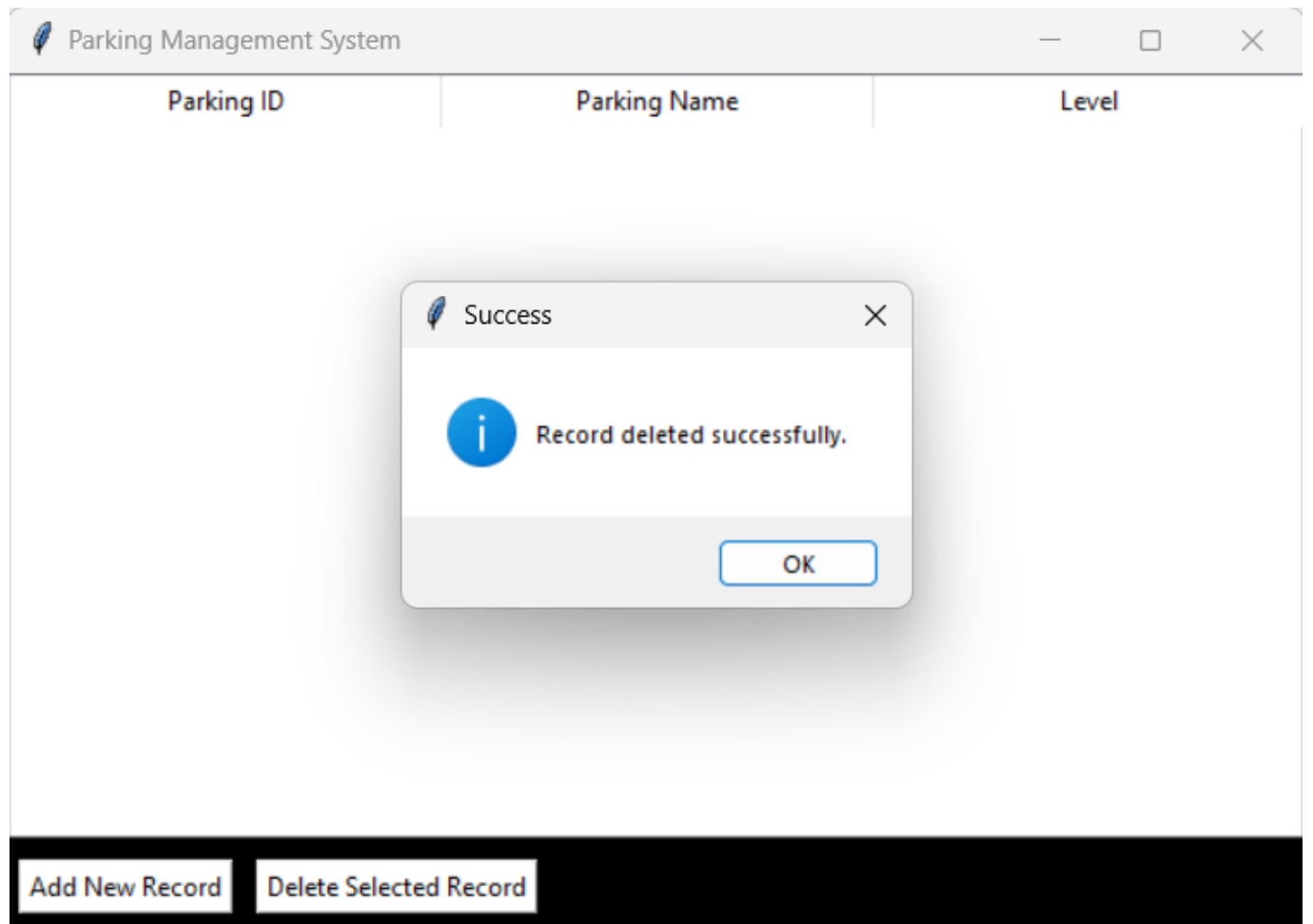
Add New Record

Delete Selected Record

Parking Management System		
Parking ID	Parking Name	Level
12	LUFFY	41

Add New Record

Delete Selected Record



6. TESTING

6.1. Unit Testing

Unit testing is a testing technique in which modules are tested individually. Small individual units of source code are tested to determine whether it is fit to use or not. Different modules of games are put to test while the modules are being developed. Here modules refer to individual levels, players, scenes.

6.2. Integration Testing

Integration testing is the technique in which individual components or modules are grouped together and tested. It occurs after testing. The inputs for the integrated testing are the modules that have already been unit tested.

6.3. System Testing

System testing is conducted on the entire system as a whole to check whether the system meets its requirements or not. software was installed on different systems and any errors or bugs that occurred were fixed.

6.4. Acceptance Testing

User Acceptance is defined as a type of testing performed by the Client to certify the system with respect to the requirements that was agreed upon. This testing happens in the final phase of testing before moving the software application to the Market or Production environment.

7. CONCLUSION

The Parking Management System (PMS) is designed to streamline and enhance the efficiency of parking operations through a user-friendly interface and robust database management. By integrating Python's Tkinter for the graphical user interface and MySQL for backend database operations, PMS offers a comprehensive solution for managing parking records, ensuring seamless data entry, retrieval, and management. Key features of the system include secure user authentication, allowing only authorized personnel to access the system; efficient data management capabilities for adding, viewing, and deleting parking records with automatic payment calculations based on the number of days; and enhanced error handling and input validation to prevent invalid data entry and ensure data integrity. The intuitive and responsive user interface facilitates easy navigation and operation, providing real-time data updates and feedback. Implementing PMS can significantly improve operational efficiency, reduce manual errors, and enhance the overall experience for both staff and users. Furthermore, the system's scalability and flexibility allow for future enhancements and customization to meet evolving needs.

8. FUTURE ENHANCEMENTS:

- **Mobile Application Integration:** Develop a mobile application that allows users to reserve parking spots, check availability in real-time, and make payments through their smartphones. This would greatly enhance user convenience and accessibility.
- **Automated Payment Systems:** Integrate with various digital payment platforms such as credit/debit cards, mobile wallets, and online banking to facilitate seamless and secure transactions. Implementing automated payment kiosks at parking locations could further enhance the user experience.
- **License Plate Recognition (LPR):** Implement a license plate recognition system to automate the process of vehicle entry and exit, reducing the need for manual checks and improving security. This can also be integrated with the payment system for automatic billing.
- **Real-Time Analytics and Reporting:** Add advanced analytics features to monitor parking usage patterns, peak hours, and revenue trends. This can help in better space management and informed decision-making.
- **Dynamic Pricing:** Introduce dynamic pricing models based on demand, time of day, or special events. This can help in optimizing space utilization and maximizing revenue.