PARKIFY - PARKING MANAGEMENT SOLUTION BUILT USING MYSQL, PYTHON, AND TKINTER

COMPUTER SCIENCE PROJECT

ANADYA NAIR

ACKNOWLEDGEMENT

I would like to thank my computer science teacher,
Ms. Shruti Mehta, for giving me the opportunity to work on this parking management project and for all the support and guidance provided throughout the process. Your insights, encouragment and feedback were invaluable in inspiring me to explore and helping me understand the concepts and execute the project effectively.
I would also like to thank my family and friends for their constant encouragement and support during this project. Their belief in my abilities provided me with the confidence to push forward.
Additionally, I am grateful to the resources and communities online, including documentation, tutorials, and forums, which provided the necessary knowledge and solutions to technical challenges encountered during the development process.
Thank you all for your support and contributions.
Sincerely,
Anadya Nair

INDEX

INTRODUCTION

In today's fast-paced world, efficient parking management is a necessity for facilities of all sizes. From large office complexes to smaller residential buildings, the need to streamline parking operations has never been more critical. Parking in India, especially in urban areas, has become one of the most pressing infrastructure issues, directly impacting daily life, the economy, and the environment. As cities grow rapidly and the number of vehicles increases exponentially, the lack of adequate parking solutions is both a symptom and a contributor to broader urban challenges. Cities like Delhi, Mumbai, and Bangalore face severe overcrowding. Roads are packed not only with moving vehicles but also with parked ones, as there simply isn't enough designated parking space for the ever-growing number of cars and motorcycles.

This situation worsens in high-density areas, like market streets, office hubs, and residential zones, where roadsides are often used as makeshift parking spots, resulting in reduced road capacity and frequent traffic jams. Many areas lack designated parking zones, and where facilities do exist, they are often poorly planned and maintained. The absence of integrated parking management systems and limited enforcement of parking regulations exacerbate the problem. Time and fuel are wasted as people circle around looking for a parking spot, often increasing travel times significantly. For businesses, the lack of accessible parking can drive away customers, impacting sales and growth. Additionally, the informal "parking economy" has mushroomed, with unauthorized attendants charging fees for makeshift parking spaces on roadsides or private plots. These unregulated setups often lead to conflicts, inefficiency, and even exploitation.

Illegal parking is rampant in Indian cities. People often park on sidewalks, along narrow streets, or in no-parking zones because the alternatives are either far from their destinations or too costly. This leads to conflicts between vehicle owners and pedestrians, as sidewalks meant for walking are occupied by vehicles, endangering pedestrians and making cities less walkable. Law enforcement typically lacks the manpower or resources to monitor every street, which allows this problem to persist. Parking policies and regulations in India are inconsistent, outdated, and poorly enforced. Different areas have different rules, and the lack of a unified policy makes it hard for both drivers and enforcement officers to follow or apply the law effectively. Fines for illegal parking are often too low to deter offenders, and with inadequate digital or automated enforcement systems, monitoring violations is difficult.

Although parking management technologies—like automated parking systems, real-time tracking, and online parking reservation platforms—are common in many countries, they have not yet seen widespread adoption in India. Cost, lack of infrastructure, and general resistance to change are major obstacles. Furthermore, public awareness about digital parking solutions remains limited, and many drivers prefer the traditional "search and park" approach, which is inefficient and adds to congestion.

Parkify is a comprehensive parking management solution designed to streamline the complexities of parking operations across facilities, whether large or small. Built with a focus on user experience, it combines a robust MySQL database for secure, organized data storage with a Python and Tkinter interface that prioritizes intuitive navigation and ease of use. With Parkify, administrators can efficiently manage parking spaces, track vehicle entries and exits, monitor occupancy levels in real time, and generate reports effortlessly. The system's interactive interface is carefully crafted to minimize the learning curve for users, ensuring that they can quickly adapt and maximize efficiency from the start. The platform is highly adaptable, making it suitable for varied settings—be it office complexes, shopping malls, or private garages. Security and reliability are key to Parkify's design. The MySQL database not only offers secure data handling but also supports extensive scalability, making it easy to expand as parking facilities grow. In combination with Python's powerful functionality and Tkinter's responsive GUI framework, Parkify delivers a seamless experience for both administrators and users.

SMART PARKING SOLUTIONS

Smart parking solutions use technology to improve the efficiency, convenience, and management of parking spaces. Smart parking makes it easier for drivers to find available parking spots, reduces traffic congestion, optimizes space usage, and provides real-time data to operators.

Benefits of Smart Parking

- *Improved Convenience:* Users can quickly find parking spots, reducing the time spent searching.
- *Reduced Traffic and Pollution:* Efficient parking reduces the number of vehicles circling in search of a spot, which can lower congestion and emissions.
- *Optimized Space Usage:* Operators can use data to allocate parking spaces effectively, which is especially beneficial in busy areas.

- *Enhanced Security:* With surveillance and automated license plate recognition, smart parking can increase security, tracking vehicle movements accurately.
- *Real-time Monitoring and Management:* Operators can monitor parking occupancy in real time, which helps in managing peak times or special events more efficiently.

Example Use Case

In a shopping mall parking area, the smart parking system is designed to provide an efficient and convenient parking experience. Sensors are installed at each parking spot to detect whether a space is occupied or vacant. This information is transmitted to a central server, where it is processed and updated in real time. Drivers can access the updated parking information through a dedicated mobile application. The app enables users to check the availability of parking spaces, reserve a spot in advance, or navigate directly to an open space. The navigation system provides turn-by-turn directions, helping users save time and effort in locating a parking spot. This system not only reduces the frustration of searching for parking but also optimizes the utilisation of available space. It helps in minimizing traffic congestion within the parking area and enhances the overall parking experience. Furthermore, by reducing unnecessary vehicle movement, the system contributes to lower fuel consumption and reduced environmental pollution.

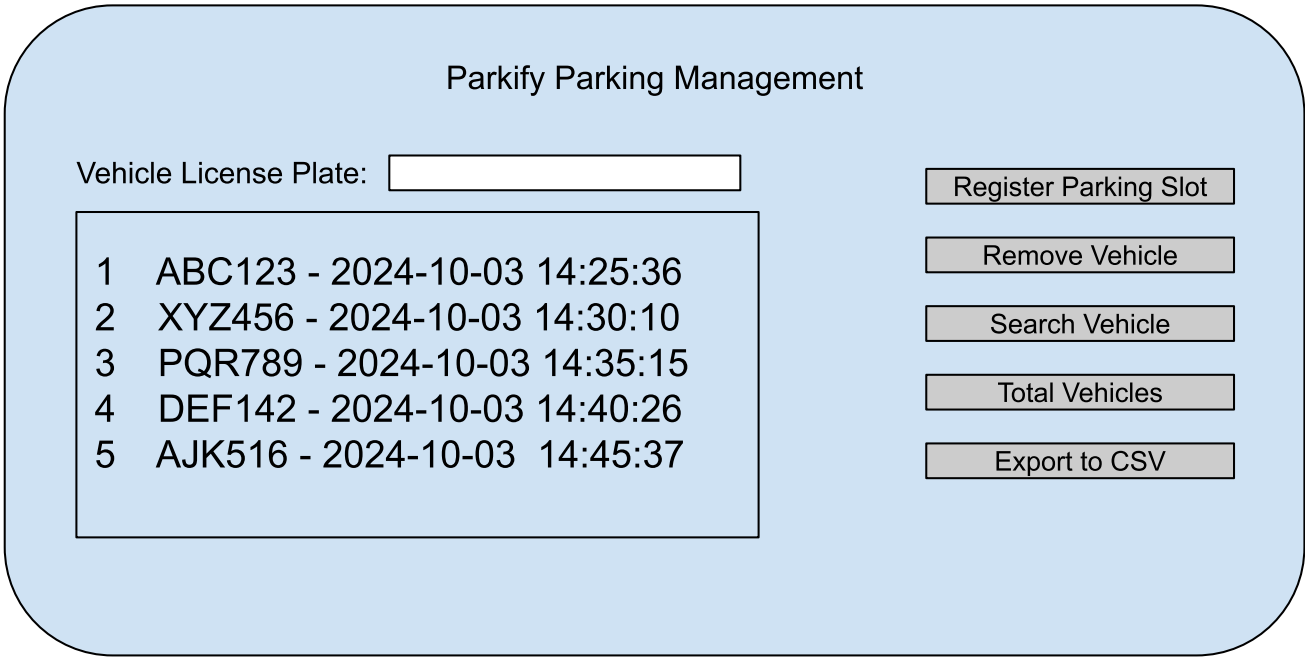
OBJECTIVES

1. Automate parking slot allocation.
2. Track vehicle entry and exit by license plate.
3. Display real-time parking availability.
4. Provide a search feature to locate specific vehicles.
5. Calculate parking duration and fees.
6. Export parking data to a CSV file for record-keeping.
7. Maintain an operation log.

The smart parking system is designed to automate the allocation of parking slots, eliminating the need for manual management. Upon a vehicle's entry, the system reads and records its license plate, instantly identifying and assigning an available parking slot. This automated allocation process continues seamlessly, allowing the system to track each vehicle's entry and exit. License plate recognition ensures that the system accurately logs vehicle movements, providing a reliable record of slots at any given time. Through this data, the system displays real-time parking availability, enabling drivers and facility managers alike to monitor and access up-to-the-minute information. This automation greatly improves parking efficiency, reduces wait times, and minimizes the operational burden on parking staff. With features like quick vehicle search, CSV export for data records, and an operation log for tracking activities, the system enhances parking efficiency, reduces wait times, and simplifies overall management.

GUI BLUEPRINT

In preparation for Parkify project, a GUI design blueprint was first made to enhance constructive processes. This blueprint is essential as it provides a clear visual structure, ensuring a user-friendly and intuitive experience. By pre-planning the interface layout, I aim to make the application more efficient and accessible, aligning with the project's goal to simplify parking management.



SYSTEM REQUIREMENTS

1. Python

Python is the core programming language used to develop the Parkify Parking Management software. User needs to install Python, ideally version 3.7 or above, to ensure compatibility with Tkinter and MySQL libraries.

Minimum Requirements:

- Operating System: Windows 7, macOS 10.9 (Mavericks), Linux (any distribution that supports Python 3)
- Python Version: 3.7 or above
- Disk Space: Approximately 100 MB for installation files and libraries
- RAM: 512 MB or more (1 GB recommended)

Recommended Configuration:

- Python Version: 3.9 or higher (for compatibility with the latest libraries and security updates)
- IDE (Optional): It is recommended to use an Integrated Development Environment (IDE) like PyCharm, VS Code, or IDLE for a better coding experience.

Additional Python Packages:

- MySQL Connector: To interact with the MySQL database. Install via pip install mysql-connector-python.
- Tkinter: This package is included by default in Python installations on most platforms.

2. MySQL

MySQL is the database management system used to store vehicle information, including license plates and timestamps. A MySQL server must be installed and configured to manage the database for this application.

Minimum Requirements:

- *Operating System:* Windows 7 or later, macOS 10.10 (Yosemite) or later, most Linux distributions
- *MySQL Version:* 5.7 or higher (MySQL 8.0 is recommended for enhanced performance and security features)
- *Disk Space:* 1 GB (minimum), additional space required depending on the database size and logs
- *RAM:* 1 GB (2 GB recommended for optimal performance)

Configuration:

- *Database Setup:* Set up a database with a table named `vehicles` to store records.
- *User Permissions:* The MySQL user must have the following permissions: `CREATE, INSERT, DELETE, SELECT`.
- *Connection:* Ensure the MySQL server is configured to accept connections from the application.

MySQL Workbench (Optional):

- *Workbench Version:* 8.0 or later
- MySQL Workbench can be used for database management and visualization, making it easier to manage tables and records manually.

3. Tkinter

Tkinter is the Python library used to create the graphical user interface (GUI) for the Parkify Parking Management software. Tkinter is typically bundled with Python, but you should confirm its availability and compatibility.

Minimum Requirements:

- *Python Compatibility:* Ensure Tkinter is compatible with your installed Python version (Python 3.x)
- *Operating System:* Tkinter is cross-platform and works on Windows, macOS, and Linux

Installation:

- Tkinter is usually included with Python on Windows and macOS.
- On Linux, you may need to install Tkinter manually:
  - *Ubuntu/Debian:* `sudo apt-get install python3-tk`
  - *Fedora:* `sudo dnf install python3-tkinter`

Summary of System Requirements

Component	*Minimum Requirements*	*Recommended Configuration*
Python	Version 3.7+, 512 MB RAM	Version 3.9+, 1 GB RAM, IDE (optional)
MySQL	Version 5.7+, 1 GB Disk Space, 1 GB RAM	Version 8.0, 2 GB Disk Space, 2 GB RAM
Tkinter	Included with Python (cross-platform)	Python 3.x compatible, install as required

METHODOLOGY

Technology Stack

Python was chosen for its versatility and ease of use, particularly for rapid application development and integration with other tools. Its extensive libraries and modules support a wide range of functionalities, which simplifies tasks such as GUI design, database connectivity, and time manipulation. The use of Python's Tkinter library also allows for a straightforward GUI development experience, which is essential for creating a user-friendly interface in this project. MySQL was selected as the database system for its reliability, speed, and ability to handle multiple queries efficiently. It provides a solid backend for storing data on parked vehicles, including unique license plate numbers and timestamps. MySQL's relational structure supports structured data management, ensuring that vehicle records are well-organized and easily retrievable. Tkinter, a built-in library in Python for GUI development, was used to create a simple and effective interface that allows users to interact with the parking management system. Tkinter is a lightweight solution that is easy to implement, and its compatibility with Python makes it a convenient choice for projects that require graphical interfaces.

Software Functionality

The program tracks each vehicle by its license plate number, storing this data along with a timestamp in a MySQL database. It includes a registration function (add_vehicle()) that allows users to enter a vehicle's license plate number. When a vehicle is

registered, its license plate and current timestamp are stored in the MySQL database as a new record in the parking table. Removal of a vehicle is facilitated by the delete_vehicle() function, which identifies the selected vehicle (based on license plate) and removes its entry from the database. These functions update the Tkinter listbox display, ensuring real-time reflection of changes in the parked vehicles list.

Additional functionalities include a search_vehicle() feature for locating specific vehicles by their license plate, and an export_to_csv() function, which generates a CSV file of the current parking data for record-keeping. Furthermore, the system maintains an operation log, recording each vehicle's addition or removal, ensuring accountability and accurate tracking of all parking activities.

The parking_db database is created in MySQL, with a parking table to store vehicle entries. This table uses fields for a unique identifier (id), the vehicle number (vehicle_no), and the timestamp of registration (registration_time). Upon each addition or removal of a vehicle, a refresh_list() function retrieves updated data from MySQL and refreshes the Tkinter listbox, providing a seamless experience by displaying all currently parked vehicles with their entry times.

FEATURES IMPLEMENTED

1. Vehicle Registration

Registers a vehicle into the system by storing its license plate number and the current timestamp. This action is performed through the GUI when a user enters a vehicle number and clicks "Register parking slot." The function takes the entered license plate, captures the current time, and stores these details in the MySQL database.

2. Parking Slot Display and Assignment

Assigns a parking slot and displays all currently parked vehicles along with their registration times in the Tkinter listbox. Each time a vehicle is added or removed, the listbox refreshes to show the updated list. The function queries all entries from the parking table, retrieves each vehicle's information, and displays it in a formatted way in the listbox. This visual representation allows users to see the currently assigned parking slots and their details.

3. Vehicle Removal with Tarriff

Allows users to remove a vehicle by selecting its license plate from the listbox and clicking the "Remove vehicle" button. This action deletes the entry from the database, and generates total parking fee for the duration that the vehicle was parked. The tariff system considers 20 rupees as the fee per hour, and if the vehicle is removed within an hour, it considers a minimum fee of 10 rupees. This function identifies the vehicle by its license

plate, removes it from the MySQL database, and refreshes the list to reflect the updated parking status.

4. Real-Time List Display

Continuously displays all parked vehicles in the listbox, updating automatically with each new addition or removal. The refresh_list() function is called after every change, ensuring that the displayed data is always current. The listbox format makes it easy for users to see all active records at once.

5. Search Functionality:

The search_vehicle() feature allows users to locate specific vehicles by their license plates. This is especially useful in busy parking lots, as it retrieves and displays precise vehicle information instantly.

6. Total Vehicles Display:

The system provides a display_total_vehicles() function that calculates and displays the total number of vehicles currently parked. This feature offers administrators and users a quick overview of parking occupancy, helping them monitor and manage the facility's capacity efficiently. The total count is displayed in a dialog box, ensuring easy access to this critical information.

7. Data Export:

Through the export_to_csv() function, the system enables the export of all parking data into a CSV file. This feature is beneficial for maintaining records, analyzing trends, or sharing data with stakeholders.

8. Operation Log Maintenance:

The system automatically maintains an operation log that records every addition and removal of vehicles, including timestamps and license plate numbers. This log enhances accountability, provides an audit trail, and supports efficient dispute resolution.

The combination of these features allows the software to manage vehicle entries and exits efficiently while providing real-time feedback to users.

CODE FOR PROGRAM

MySQL code

```
CREATE DATABASE parking_db;
USE parking_db;

CREATE TABLE parking (
    id INT AUTO_INCREMENT PRIMARY KEY,
    vehicle_no VARCHAR(10) UNIQUE,
    registration_time DATETIME
);
```

Python code along with Tkinter integration

```
from tkinter import *
from tkinter import messagebox
from datetime import datetime
import mysql.connector as sqltor
import csv

# Connect to MySQL database
mycon = sqltor.connect(
    host="localhost",
    user="root",
    password="your_password",  # Replace with your MySQL password
    database="parking_db"      # Ensure the database exists
)
cursor = mycon.cursor()

# Functions for database operations
def add_vehicle():
    """
    Adds a vehicle to the parking database.
    Validates the license plate format and prevents duplicate entries.
    Logs the operation in a text file.
    """
    vehicle_no = vehicle_entry.get().upper()  # Convert input to uppercase
    if not vehicle_no:
        # Display error for empty input
```

```
        messagebox.showerror("Input Error", "License plate cannot be empty!")
        return
    if len(vehicle_no) > 10:
        # Display error for invalid format
        messagebox.showerror("Format Error", "License plate format is invalid!")
        return
    # Check for duplicate entry
    cursor.execute("SELECT vehicle_no FROM parking WHERE vehicle_no = %s",
(vehicle_no,))
    if cursor.fetchone():
        messagebox.showerror("Duplicate Entry", "Vehicle already parked!")
        return

    registration_time = datetime.now()
    # Insert vehicle into the database
    cursor.execute("INSERT INTO parking (vehicle_no, registration_time) VALUES (%s,
%s)", (vehicle_no, registration_time))
    mycon.commit()
    log_operation("ADD", vehicle_no)  # Log the operation
    vehicle_entry.delete(0, END)  # Clear the input field
    refresh_list()

def delete_vehicle():
    """
    Removes a vehicle from the parking database.
    Calculates the tariff based on parking duration and logs the operation.
    """
    selected = vehicle_listbox.get(ACTIVE)  # Get the selected vehicle from the listbox
    if not selected:
        messagebox.showerror("Selection Error", "No vehicle selected!")
        return
    vehicle_id, vehicle_no = selected.split(" ")[0], selected.split(" ")[1]  # Extract vehicle ID
and number
    cursor.execute("SELECT registration_time FROM parking WHERE id = %s",
(vehicle_id,))
    entry_time = cursor.fetchone()[0]
    exit_time = datetime.now()
    # Calculate duration and tariff
    duration = (exit_time - entry_time).total_seconds() / 3600  # Duration in hours
    tariff = max(10, round(duration * 20))  # ₹20 per hour, minimum ₹10
    cursor.execute("DELETE FROM parking WHERE id = %s", (vehicle_id,))
    mycon.commit()
    log_operation("REMOVE", vehicle_no)  # Log the operation
    messagebox.showinfo("Tariff", f"Vehicle removed. Total parking fee: ₹{tariff}")
    refresh_list()
```

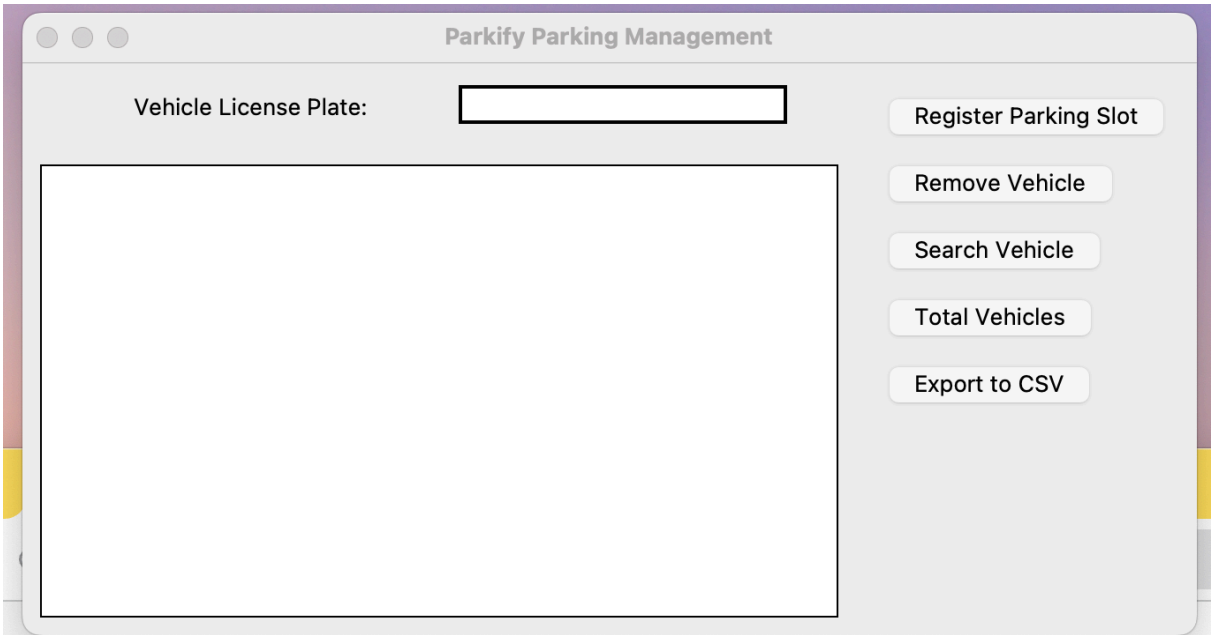
```
def refresh_list():
    """
    Refreshes the list of parked vehicles displayed in the GUI.
    """
    cursor.execute("SELECT id, vehicle_no, registration_time FROM parking")
    vehicles = cursor.fetchall()
    vehicle_listbox.delete(0, END)  # Clear the listbox
    # Add each vehicle to the listbox
    for vehicle in vehicles:
        vehicle_listbox.insert(END, f"{vehicle[0]} {vehicle[1]} -
{vehicle[2].strftime('%Y-%m-%d %H:%M:%S')}")

def search_vehicle():
    """
    Searches for a vehicle by its license plate.
    Displays the search result in a dialog box.
    """
    search_query = vehicle_entry.get().upper()  # Get search input
    if not search_query:
        messagebox.showerror("Search Error", "Enter a license plate to search!")
        return
    cursor.execute("SELECT * FROM parking WHERE vehicle_no = %s", (search_query,))
    result = cursor.fetchone()
    if result:
        # Display vehicle details
        messagebox.showinfo("Search Result", f"Vehicle Found: {result[1]} - {result[2]}")
    else:
        messagebox.showerror("Search Result", "Vehicle not found!")

def display_total_vehicles():
    """
    Displays the total number of vehicles currently parked.
    """
    cursor.execute("SELECT COUNT(*) FROM parking")
    count = cursor.fetchone()[0]
    messagebox.showinfo("Total Vehicles", f"Total vehicles parked: {count}")

def export_to_csv():
    """
    Exports the current parking data to a CSV file.
    """
    cursor.execute("SELECT * FROM parking")
    vehicles = cursor.fetchall()
    with open("parking_data.csv", "w", newline="") as file:
```

```
        writer = csv.writer(file)
        writer.writerow(["ID", "Vehicle No", "Registration Time"])  # Write headers
        writer.writerows(vehicles)  # Write data rows
    messagebox.showinfo("Export", "Parking data exported to parking_data.csv")

def log_operation(operation, vehicle_no):
    """
    Logs an operation (ADD/REMOVE) with details into a text file.
    """
    with open("parking_log.txt", "a") as log_file:
        log_file.write(f"{operation} - {vehicle_no} at {datetime.now()}\n")

# Tkinter GUI setup
root = Tk()
root.title("Parkify Parking Management")

# Vehicle License Plate Label and Entry
Label(root, text="Vehicle License Plate:").grid(row=0, column=0, padx=10, pady=10,
sticky="e")
vehicle_entry = Entry(root, width=20)
vehicle_entry.grid(row=0, column=1, padx=10, pady=10)

# Frame for the Floating Panel (Buttons)
button_frame = Frame(root)
button_frame.grid(row=0, column=2, rowspan=2, padx=10, pady=10, sticky="n")

# Buttons in the Floating Panel
Button(button_frame, text="Register Parking Slot", command=add_vehicle).grid(row=0,
column=0, padx=5, pady=5, sticky="w")
Button(button_frame, text="Remove Vehicle", command=delete_vehicle).grid(row=1,
column=0, padx=5, pady=5, sticky="w")
Button(button_frame, text="Search Vehicle", command=search_vehicle).grid(row=2,
column=0, padx=5, pady=5, sticky="w")
Button(button_frame, text="Total Vehicles",
command=display_total_vehicles).grid(row=3, column=0, padx=5, pady=5, sticky="w")
Button(button_frame, text="Export to CSV", command=export_to_csv).grid(row=4,
column=0, padx=5, pady=5, sticky="w")

# Listbox to display parked vehicles
vehicle_listbox = Listbox(root, width=50, height=15)
vehicle_listbox.grid(row=1, column=0, columnspan=2, padx=10, pady=10)

# Load the vehicle list initially
refresh_list()
```

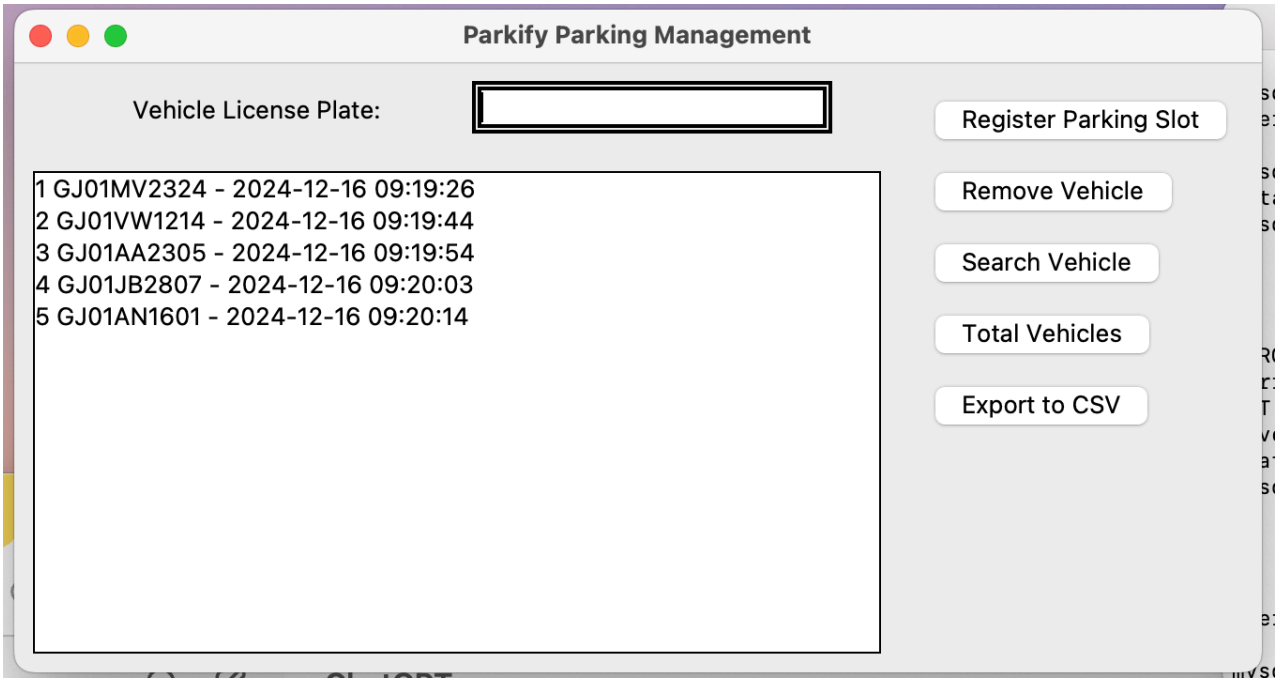
```
# Run the Tkinter event loop
root.mainloop()
```

OUTPUT

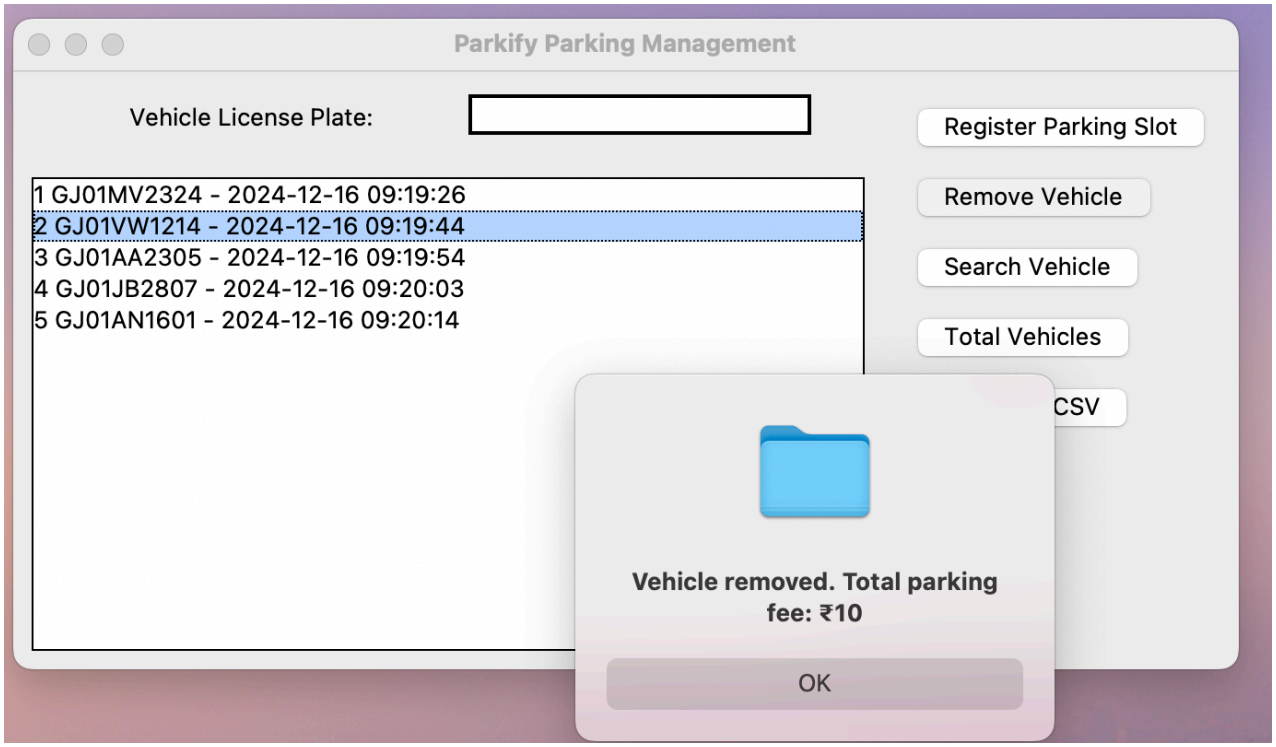
Parkify Main Window



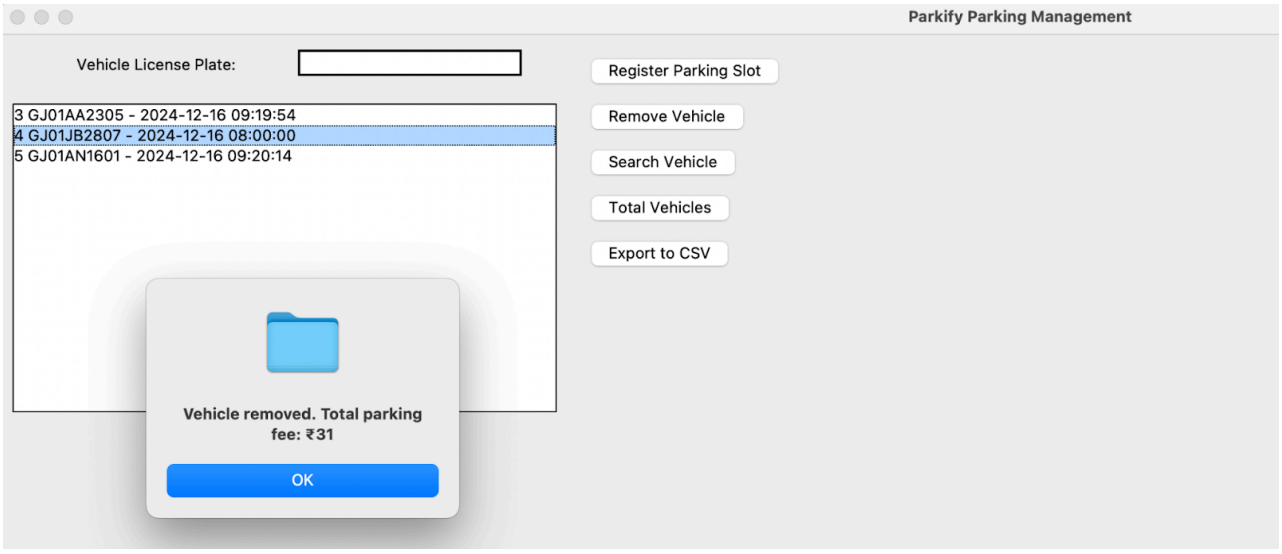
Registered parking slots



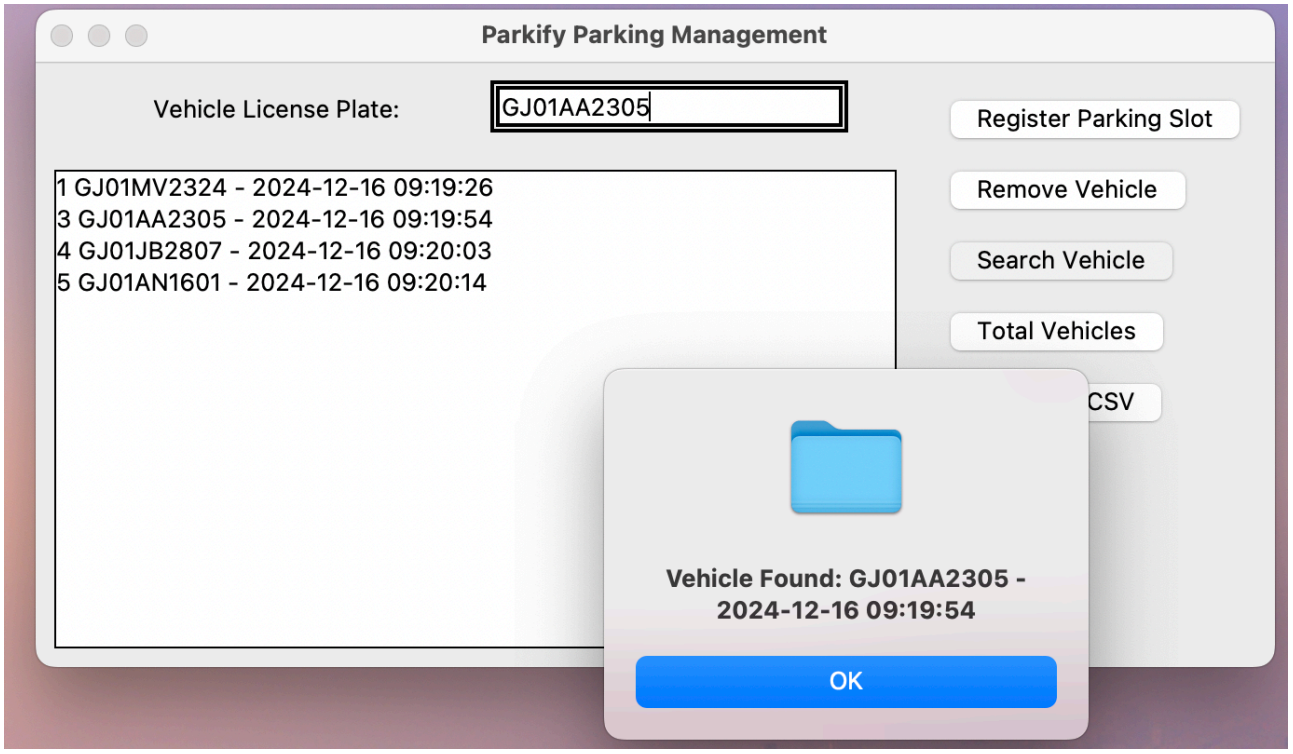
Vehicle removed within some time



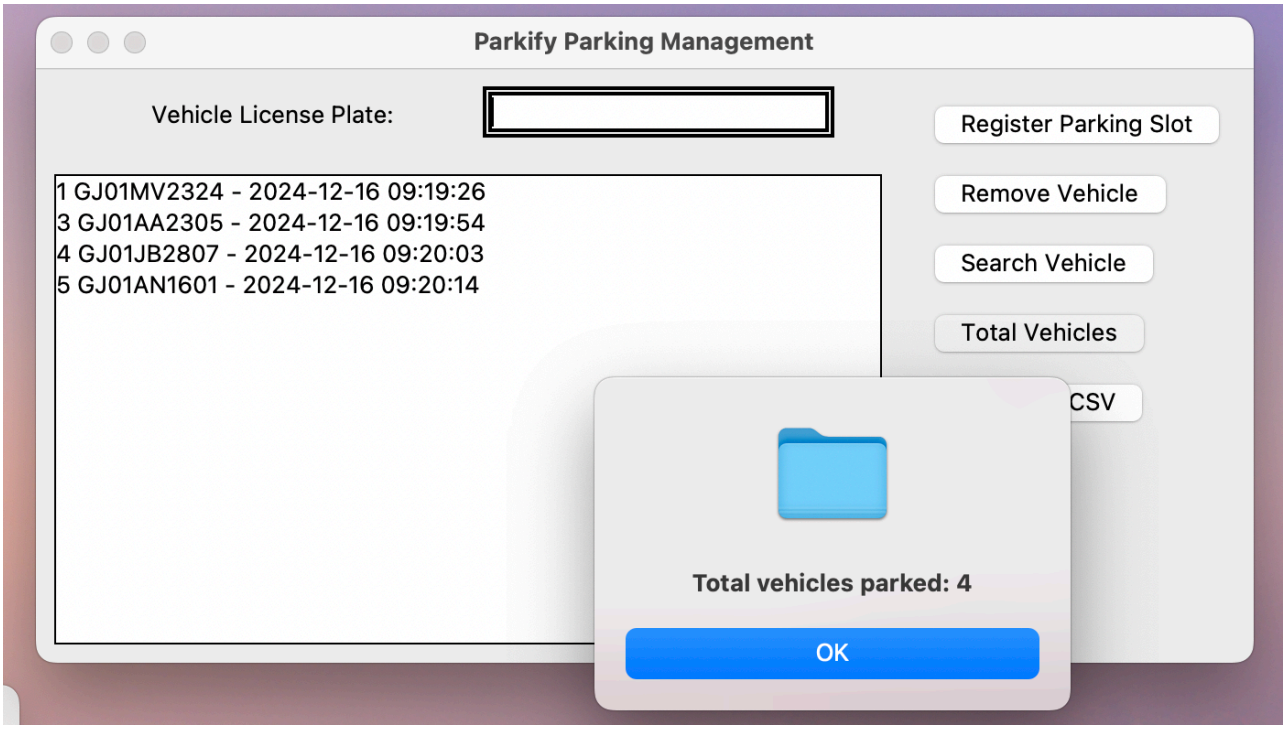
Vehicle removed after a long duration with the parking fee updated



Search Vehicle



Total Vehicles



MySQL Window

```
mysql>  CREATE TABLE parking (
    ->  id INT AUTO_INCREMENT PRIMARY KEY,
    ->  vehicle_no VARCHAR(10) UNIQUE,
    ->  registration_time DATETIME
    ->  );
Query OK, 0 rows affected (0.01 sec)
```

```
mysql> select * from parking;
+----+------------+---------------------+
| id | vehicle_no | registration_time   |
+----+------------+---------------------+
|  3 | GJ01AA2305 | 2024-12-16 09:19:54 |
|  5 | GJ01AN1601 | 2024-12-16 09:20:14 |
|  6 | GJ01JB2807 | 2024-12-17 10:06:30 |
|  7 | GJ01VN2107 | 2024-12-17 10:06:48 |
+----+------------+---------------------+
4 rows in set (0.00 sec)
```

```
mysql> desc parking;
+------------------+-------------+------+-----+---------+----------------+
| Field            | Type        | Null | Key | Default | Extra          |
+------------------+-------------+------+-----+---------+----------------+
| id               | int         | NO   | PRI | NULL    | auto_increment |
| vehicle_no       | varchar(10) | YES  | UNI | NULL    |                |
| registration_time | datetime    | YES  |     | NULL    |                |
+------------------+-------------+------+-----+---------+----------------+
3 rows in set (0.01 sec)
```

Exported CSV Data

parking_data

ID	Vehicle No	Registration Time
1	GJ01MV2324	2024-12-16 09:19:26
3	GJ01AA2305	2024-12-16 09:19:54
4	GJ01JB2807	2024-12-16 09:20:03
5	GJ01AN1601	2024-12-16 09:20:14

Operation Log

```
ADD - GJ01MV2324 at 2024-12-16 09:19:26.204501
ADD - GJ01VW1214 at 2024-12-16 09:19:43.705381
ADD - GJ01AA2305 at 2024-12-16 09:19:54.412010
ADD - GJ01JB2807 at 2024-12-16 09:20:03.020342
ADD - GJ01AN1601 at 2024-12-16 09:20:14.470302
REMOVE - GJ01VW1214 at 2024-12-16 09:20:26.638492
```

RESULTS

The smart parking management system achieved its intended functionalities, providing a reliable method for managing parking space allocations through vehicle registration and removal based on license plates. Testing verified that each feature works as expected. The program adds vehicles with unique license plate numbers, storing both the plate and timestamp in the MySQL database. Duplicate entries are prevented, ensuring data accuracy. The Tkinter GUI listbox accurately displays the current status of all parked vehicles, refreshing instantly after each registration or removal. Users can seamlessly remove selected vehicles from the database, with the list reflecting the changes immediately. Error handling was tested, confirming no system crashes or unexpected behavior. These results demonstrate that the system performs well under various conditions, providing a user-friendly interface that updates in real time to reflect the current parking situation.

CONCLUSION

The project successfully addresses the need for an efficient, digital approach to parking management. By integrating Python, MySQL, and Tkinter, the system simplifies vehicle tracking within a parking space and ensures data reliability. This project showcases the potential of automated parking management systems to reduce human error, streamline parking processes, and provide a clear view of parking availability at any given moment. The implementation not only fulfills the basic parking management needs but also lays a foundation for future enhancements, such as automated billing and slot reservations. This project also highlights key learnings in database management, GUI development, and error handling, making it a comprehensive demonstration of software development and system design principles.

PRECAUTIONS

To ensure consistent performance and data integrity in real-world scenarios, the following precautions should be taken:

1. *Data Validation:* Ensure all inputs are validated to prevent invalid or malicious entries, such as overly long or improperly formatted license plates.
2. *Database Backups:* Regularly back up the MySQL database to prevent data loss, especially if the software is used in a live environment with critical data.
3. *Concurrent Access Management:* If multiple users are expected to interact with the system simultaneously, consider implementing concurrency controls to prevent data conflicts.
4. *Error Handling:* Add thorough error handling to manage unexpected events, such as database disconnections or unrecognized inputs, to prevent application crashes.
5. *System Security:* Implement security measures to restrict unauthorized access, especially for systems dealing with sensitive vehicle information.

By adhering to these precautions, the project's effectiveness and reliability can be maintained, ensuring safe and smooth operation in a real-world setting.

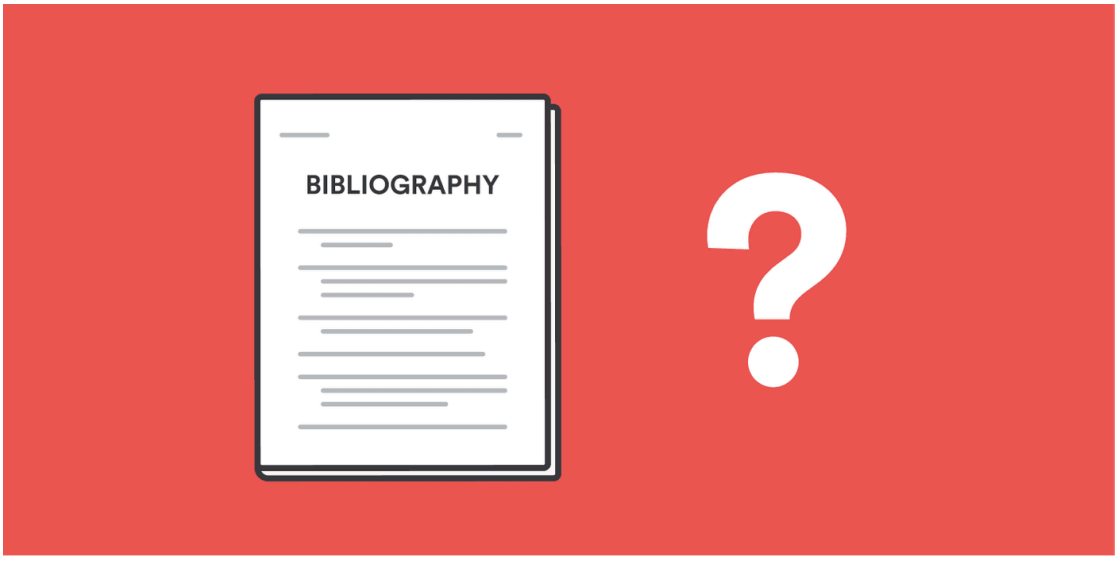
SCOPE OF PROJECT

Parkify is a smart parking system that helps to manage parking slots in an efficient, simple, and automated way. It makes the process of vehicle entry, slot allocation, and exit easier, while also ensuring proper use of parking space. This system reduces the need for manual work and provides a smooth experience for users. It is best suited for small to medium-sized parking areas, such as those in shopping malls, office buildings, and residential complexes. However, this project can be upscaled with the following:

1. *Sensors and IoT Devices:* Installed in parking spots, these sensors detect whether a spot is occupied or available. They send data to a central server that updates in real time.
2. *License Plate Recognition (LPR):* Cameras and software use image processing to identify vehicles by their license plates. This is particularly useful for automatic entry and exit, as well as for tracking vehicle occupancy.
3. *Mobile Applications:* Many smart parking systems come with apps that allow users to view available spots, reserve parking, pay online, and even navigate to their designated space.
4. *Automated Payment Systems:* These systems handle payments digitally, reducing the need for cash or physical tickets. Users can pay via apps, contactless methods, or even prepay for a reserved spot.
5. *Data Analytics:* Collecting data over time helps operators analyze usage patterns, peak hours, and other trends. This

can lead to better space allocation, pricing strategies, and operational adjustments.

BIBLIOGRAPHY



Author: Anadya Nair

Title of Work: Parkify - Parking management solution built using MySQL, Python, and Tkinter

Resources used:

1. *Python Documentation*

Python Software Foundation. Python 3 Documentation. Retrieved from https://docs.python.org/3/ Used for reference on Python syntax, libraries, and datetime management.

2. *MySQL Documentation*

Oracle Corporation. MySQL 8.0 Reference Manual. Retrieved from https://dev.mysql.com/doc/ Provided information on MySQL database management, table creation, and SQL commands.

3. *Tkinter Documentation*

Python Software Foundation. Tkinter – Python Interface to Tcl/Tk. Retrieved from https://docs.python.org/3/library/tkinter.html Used to understand Tkinter functions for creating and managing GUI components like labels, buttons, and listboxes.

4. Stack Overflow

Used for troubleshooting common issues related to Tkinter GUI refresh, database connections, and SQL query syntax.

5. GeeksforGeeks

GeeksforGeeks. Introduction to Tkinter. Retrieved from https://www.geeksforgeeks.org/python-gui-tkinter/ This resource provided examples and explanations on using Tkinter for GUI development.