

Smart Cities Smarter Parking: Developing and Implementing Vehicle Parking Management System

Samiksha Shrivastava

Student, Kalinga University, Raipur (C.G.)

ABSTRACT

This research paper explores the design, implementation, and impact of a state-of-the-art Vehicle Parking Management System (VPMS) as a pivotal solution to the escalating challenges associated with urban mobility. The rapid growth of urbanization has led to an increased demand for efficient parking solutions to alleviate congestion, reduce environmental impact, and enhance overall urban living.

KEYWORDS: Introduction, VPMS Architecture, Development of VPMS using Python and PHP XAMPP, Program Evaluation and Output, Future Directions, Conclusion, References

I. INTRODUCTION

The proposed VPMS integrates cutting-edge technologies such as Internet of Things (IoT), data analytics, and artificial intelligence to revolutionize traditional parking management strategies. By employing real-time data collection, the system optimizes the allocation of parking spaces, reduces search time for drivers, and contributes to a more sustainable and resilient urban infrastructure.

The research delves into the technical aspects of the VPMS, emphasizing its ability to provide dynamic parking availability information through a user-friendly interface. The system's adaptability to diverse urban landscapes and scalability for future growth are also explored, making it a robust solution for a wide range of urban environments.

Furthermore, the paper investigates the economic and environmental impacts of the proposed VPMS. Through case studies and simulations, the research demonstrates the potential reduction in traffic congestion, fuel consumption, and carbon emissions. Additionally, the economic benefits, including increased revenue from optimized parking utilization, are highlighted.

In conclusion, this research contributes to the evolving field of smart urban systems by presenting a comprehensive and innovative approach to vehicle parking management. The proposed VPMS not only addresses the immediate challenges of urban mobility but also sets the stage for sustainable and intelligent urban development in the face of growing urbanization.

II. Vehicle Parking Management System Architecture

The Vehicle Parking Management System (VPMS) is designed as a comprehensive solution to efficiently manage parking spaces in urban environments. The architecture is divided into four key sections: Slots Management, Add Vehicle, Manage Vehicle, and History.

1. Slots Management:

- **Sensor Network:** Utilizes IoT-enabled sensors installed in each parking slot to detect the presence or absence of vehicles.
- **Data Processing Module:** Gathers real-time data from sensors, processes it, and updates the availability status of each parking slot.
- **Parking Slot Database:** Stores information about each parking slot, including its location, status, and any relevant attributes.

2. Add Vehicle:

- **User Interface:** Provides a user-friendly interface accessible via mobile app or web portal.
- **User Authentication:** Ensures secure access through user authentication mechanisms.
- **Vehicle Registration:** Allows users to register their vehicles by entering details such as license plate number, vehicle type, and user information.

3. Manage Vehicle:

- **User Profile Management:** Allows users to view and update their profile information.
- **Parking Reservation:** Enables users to reserve parking spaces in advance, specifying the desired time and location.
- **Notifications:** Sends notifications to users regarding parking reservations, expiration alerts, and other relevant information.

4. History:

- **Parking Activity Log:** Stores a historical record of all parking activities, including entry and exit times, duration of stay, and associated fees.
- **Search and Filter Options:** Allows users to search and filter their parking history based on various parameters such as date, location, or vehicle.
- **Reporting Module:** Generates reports and analytics for parking trends, peak hours, and revenue generation.

Integration Points:

- **APIs:** Facilitates communication between different modules and external systems.
- **Mobile/Web Application Interface:** Connects the system with user interfaces for a smooth user experience.
- **External Database Integration:** Integrates with external databases for backup, data analysis, and reporting purposes.

This architecture ensures a scalable, flexible, and efficient Vehicle Parking Management System that optimizes parking space utilization, enhances user experience, and provides valuable insights into parking dynamics.

III. DEVELOPMENT OF VEHICLE PARKING MANAGEMENT SYSTEM USING PYTHON AND PHP XAMPP

MODULE 1: HomeWindow.py

Imports: The script begins by importing necessary modules from PyQt5 and a custom module **DataBaseOperation** (assuming it handles database operations).

Class Definition (HomeScreen): The class inherits from **QMainWindow** and serves as the main window for the application. The class initializes the window with a title, sets up a database operation instance (**self.dbOperation**), and creates a main widget.

Menu Buttons and Styling: Four buttons ("Home," "Add Vehicle," "Manage Vehicle," "History") are created and styled with specific colors, sizes, and borders. Button signals are connected to corresponding methods (**self.showHome**, **self.showAdd**, **self.showManage**, **self.showHistory**).

Menu Frame: The buttons are added to a vertical layout, and the layout is applied to a frame (**menu_frame**).

Content Layout (parent_vertical): Four vertical layouts (**self.vertical_1**, **self.vertical_2**, **self.vertical_3**, **self.vertical_4**) are created for different sections of the application (Home, Add Vehicle, Manage Vehicle, History).

Frames (self.frame_1 to self.frame_4): Four frames are created to hold the content of each section, and the corresponding vertical layout is set for each frame.

Widgets and Layouts: The frames are added to the main layout (**parent_vertical**), which is a part of the horizontal layout (**layout_horizontal**). The horizontal layout includes both the menu frame and the parent vertical layout.

Styling and Display: Styling, margins, and stretch factors are applied to achieve the desired layout appearance. Initial visibility settings (**self.frame_1.show()**, others hidden) are set.

Methods (showHome, showAdd, showManage, showHistory): These methods handle the button clicks, adjust button styles, and show/hide the corresponding frames.

Database Interaction Methods (refreshHome, addHomePageData, addVehicles, addManagePage, refreshManage, refreshHistory, addHistoryPage, exitCall): These methods interact with the database (**self.dbOperation**) to fetch, add, or update data. They create and update widgets (e.g., buttons, labels, tables) based on the retrieved data.

Event Handling (exitCall): The **exitCall** method is connected to the "Exit" button in the Manage Vehicle section, allowing users to mark a vehicle as exited and update the table accordingly

MODULE 2: InstallWindow.py

Import Statements: The code starts by importing necessary PyQt5 widgets (**QWidget**, **QPushButton**, **QVBoxLayout**, **QLabel**, **QLineEdit**) and other dependencies (**json** for JSON operations and custom classes **LoginScreen** and **DBOperation**).

Class Initialization: The **InstallWindow** class is defined, inheriting from **QWidget**. The constructor (**__init__**) sets up the initial window properties such as title and size.

UI Layout Setup: A vertical layout (**QVBoxLayout**) is created to organize the widgets vertically. Labels and input fields for various configuration parameters (database name, username, password, etc.) are created and styled using CSS. Default values are set for some input fields. A "Save Config" button (**buttonsave**) and an error label (**error_label**) are created and styled.

Signal Connection: The **buttonsave.clicked** signal is connected to the **showStepInfo** method. This means that when the "Save Config" button is clicked, the **showStepInfo** method will be executed.

showStepInfo Method: This method is responsible for validating user inputs. If any input field is empty, it sets an error message in the **error_label** and returns, preventing further execution. If all inputs are provided, it creates a dictionary (**data**) with the entered database information and writes it to a JSON file (**config.json**). An instance of the **DBOperation** class is created (**dbOperation**), and methods are called to create tables, insert admin information, and insert one-time data into the database. The window is closed, and a new instance of the **LoginScreen** class is created and displayed.

Execution: If the user successfully enters all the required information and clicks "Save Config," the information is saved to a configuration file, database tables are created, and admin and parking space information is inserted into the database. Finally, the current window is closed, and the login screen is displayed.

Print Statement: A print statement at the end ("Save") is likely for debugging purposes and indicates successful execution of the **showStepInfo** method

MODULE 3: LoginWindow.py

Import necessary modules and classes from PyQt5, sys, DataBaseOperation, and HomeWindow.

Create a class **LoginScreen** that inherits from **QWidget** (a basic PyQt5 widget). This class represents the GUI for the admin login screen.

In the **__init__** method: Set the window title to "Admin Login" and resize the window. Create layout using **QVBoxLayout** to arrange the widgets vertically. Create **QLabel** instances for "Username" and "Password" labels. Create **QLineEdit** instances for inputting the username and password. Create a **QLabel** for displaying error messages. Create a **QPushButton** for the "Login" action. Set styles for labels, input fields, button, and error message. Add widgets to the layout. Connect the "Login" button click event to the **showHome** method.

Define the **showLoginScreen** method, which shows the login screen.

Define the **showHome** method: Check if the username or password fields are empty. If so, display an error message and return. Create an instance of the **DBOperation** class. Call the **doAdminLogin** method from the **DBOperation** class with the entered username and password. If login is successful, close the current login screen, create an instance of the **HomeScreen** class, and show it. If login fails, display an "Invalid Login Details" error message.

MODULE 4: MainProgram.py

Importing Modules: **import sys:** Imports the sys module, which provides access to some variables used or maintained by the Python interpreter. **import os:** Imports the os module, allowing interaction with the operating system. **from InstallWindow import InstallWindow:** Imports the InstallWindow class from the InstallWindow module. **from LoginWindow import LoginScreen:** Imports the LoginScreen class from the LoginWindow module. **from PyQt5.QtWidgets import QApplication, QSplashScreen, QLabel:** Imports necessary classes for creating a PyQt5 GUI application. **from PyQt5.QtGui import QPixmap:** Imports the QPixmap class for handling images. **from PyQt5.QtCore import Qt, QTimer:** Imports classes related to Qt and QTimer for timing operations.

MainScreen Class: Defines a class named **MainScreen**. Contains a method **showSplashScreen()** that initializes a splash screen with an image and displays it.

Functions: **showSetupWindow():** Closes the splash screen and shows the installation window.

showLoginWindow(): Closes the splash screen and shows the login window.

Application Setup: Creates a **QApplication** instance named **app** with command-line arguments **sys.argv**. Initializes instances of the **LoginScreen**, **MainScreen**, and **InstallWindow** classes. Calls the **showSplashScreen()** method of the **MainScreen** class, displaying a splash screen with an image.

Conditional Check: Checks if a file named "config.json" exists in the current directory. If the file exists, a **QTimer** is set to call **showLoginWindow()** after a delay of 3000 milliseconds (3 seconds). If the file doesn't exist, a **QTimer** is set to call **showSetupWindow()** after the same delay.

Execution: `sys.exit(app.exec_())`: Starts the application's event loop, allowing it to respond to user interactions and system events. The application exits when the event loop is terminated

MODULE 5: DataBaseOperation.py

Initialization: The class constructor (`__init__`) reads the database configuration from a JSON file (`config.json`). It establishes a connection to the MySQL database (`vpms_py`).

Table Creation: The `CreateTables` method drops existing tables (`admin`, `slots`, `vehicles`) and creates new ones with specified columns.

Insert One-Time Data: The `InsertOneTimeData` method populates the `slots` table with data based on the provided number of parking spaces for two-wheelers (`space_for_two`) and four-wheelers (`space_for_four`).

Insert Admin: The `InsertAdmin` method inserts a new admin user into the `admin` table with the provided username and password.

Admin Login: The `doAdminLogin` method checks the entered username and password against the `admin` table for authentication.

Get Slot Space: The `getSlotSpace` method retrieves information about all parking slots from the `slots` table.

Get Current Vehicle: The `getCurrentVehicle` method retrieves information about vehicles currently parked (not yet exited) from the `vehicles` table.

Get All Vehicles: The `getAllVehicle` method retrieves information about vehicles that have exited from the parking area (marked with `is_exit='1'`).

Add Vehicles: The `AddVehicles` method adds a new vehicle entry to the `vehicles` table and updates the corresponding parking slot information. It checks for available parking space based on the vehicle type.

Space Available: The `spaceAvailable` method checks for an available parking slot based on the vehicle type.

Exit Vehicle: The `exitVehicle` method updates the parking slot as available and marks the vehicle as exited in the `slots` and `vehicles` tables, respectively.

IV. PROGRAM EVALUATION AND OUTPUT



Admin Login

Username :

Password :

Login

Home

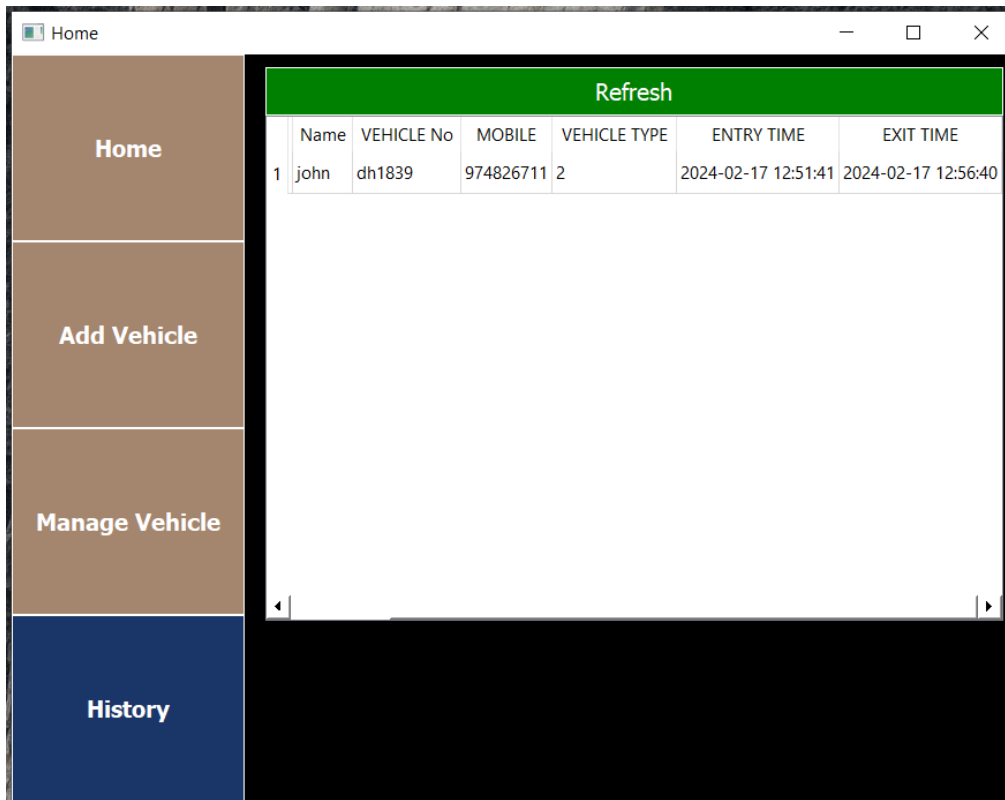
Refresh Slot				
Slot 1	Slot 2	Slot 3	Slot 4	Slot 5
Slot 6 None	Slot 7 None	Slot 8 None	Slot 9 None	Slot 10 None
Slot 11 None	Slot 12 None	Slot 13 None	Slot 14 None	Slot 15 None
Slot 16 None	Slot 17 None	Slot 18 None	Slot 19 None	Slot 20 None
Slot 21 None	Slot 22 None	Slot 23 None	Slot 24 None	Slot 25 None
Slot 26 None	Slot 27 None	Slot 28 None	Slot 29 None	Slot 30 None
Slot 31	Slot 32	Slot 33	Slot 34 None	Slot 35 None
Slot 36 None	Slot 37 None	Slot 38 None	Slot 39 None	Slot 40 None
Slot 41 None	Slot 42 None	Slot 43 None	Slot 44 None	Slot 45 None
Slot 46 None	Slot 47 None	Slot 48 None	Slot 49 None	Slot 50 None
Slot 51 None	Slot 52 None	Slot 53 None	Slot 54 None	Slot 55 None
Slot 56 None	Slot 57 None	Slot 58 None	Slot 59 None	Slot 60 None

Home

Home	Name :	<input type="text" value="john"/>
Add Vehicle	Mobile :	<input type="text" value="974826711"/>
Manage Vehicle	Vehicle No :	<input type="text" value="dh1839"/>
History	Vehicle Type :	<input type="text" value="2 Wheeler"/>
Add Vehicle		

Home		Refresh Slot				
		Slot 1 25	Slot 2	Slot 3	Slot 4	Slot 5
Add Vehicle	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 None	Slot 27 None	Slot 28 None	Slot 29 None	Slot 30 None	
Manage Vehicle	Slot 31	Slot 32	Slot 33	Slot 34 None	Slot 35 None	
	Slot 36 None	Slot 37 None	Slot 38 None	Slot 39 None	Slot 40 None	
	Slot 41 None	Slot 42 None	Slot 43 None	Slot 44 None	Slot 45 None	
	Slot 46 None	Slot 47 None	Slot 48 None	Slot 49 None	Slot 50 None	
	Slot 51 None	Slot 52 None	Slot 53 None	Slot 54 None	Slot 55 None	
History	Slot 56 None	Slot 57 None	Slot 58 None	Slot 59 None	Slot 60 None	

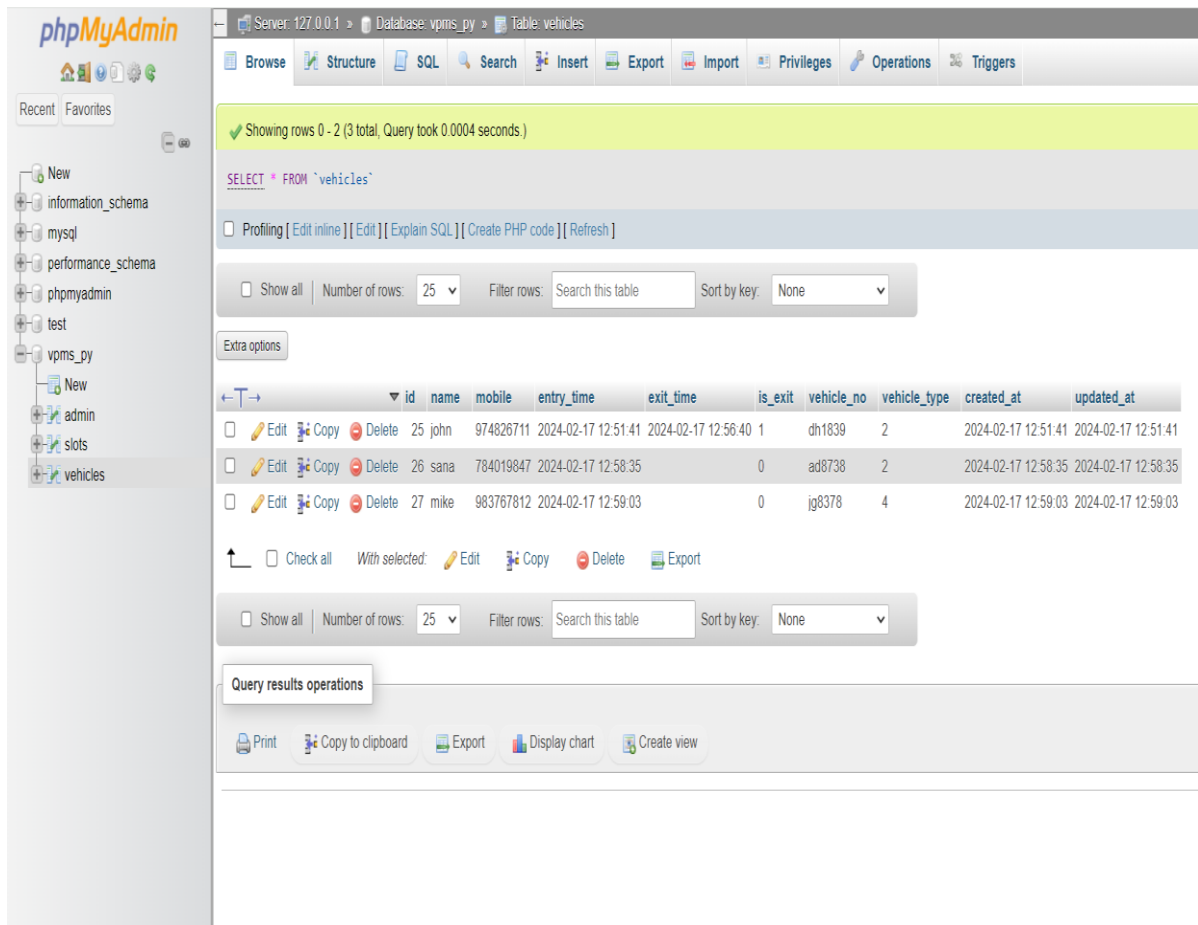
Home		Refresh							
		ID	Name	VEHICLE No	MOBILE	VEHICLE TYPE	ENTRY TIME	ACTION	
Add Vehicle	Manage Vehicle	1	25	john	dh1839	974826711	2	2024-02-17 12:51:41	Exit
History									



Refresh						
	Name	VEHICLE No	MOBILE	VEHICLE TYPE	ENTRY TIME	EXIT TIME
1	john	dh1839	974826711	2	2024-02-17 12:51:41	2024-02-17 12:56:40



Refresh Slot					
Home	Slot 1 26	Slot 2	Slot 3	Slot 4	Slot 5
	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
Add Vehicle	Slot 21 None	Slot 22 None	Slot 23 None	Slot 24 None	Slot 25 None
	Slot 26 None	Slot 27 None	Slot 28 None	Slot 29 None	Slot 30 None
	Slot 31 27	Slot 32	Slot 33	Slot 34 None	Slot 35 None
	Slot 36 None	Slot 37 None	Slot 38 None	Slot 39 None	Slot 40 None
Manage Vehicle	Slot 41 None	Slot 42 None	Slot 43 None	Slot 44 None	Slot 45 None
	Slot 46 None	Slot 47 None	Slot 48 None	Slot 49 None	Slot 50 None
	Slot 51 None	Slot 52 None	Slot 53 None	Slot 54 None	Slot 55 None
	Slot 56 None	Slot 57 None	Slot 58 None	Slot 59 None	Slot 60 None
History					



Server: 127.0.0.1 • Database: vpms_py • Table: vehicles

Showing rows 0 - 2 (3 total, Query took 0.0004 seconds)

```
SELECT * FROM `vehicles`
```

Number of rows: 25 | Filter rows: Search this table | Sort by key: None

	id	name	mobile	entry_time	exit_time	is_exit	vehicle_no	vehicle_type	created_at	updated_at
<input type="checkbox"/>	25	john	974826711	2024-02-17 12:51:41	2024-02-17 12:56:40	1	dh1839	2	2024-02-17 12:51:41	2024-02-17 12:51:41
<input type="checkbox"/>	26	sana	784019847	2024-02-17 12:58:35		0	ad8738	2	2024-02-17 12:58:35	2024-02-17 12:58:35
<input type="checkbox"/>	27	mike	983767812	2024-02-17 12:59:03		0	jj8378	4	2024-02-17 12:59:03	2024-02-17 12:59:03

Query results operations: Print, Copy to clipboard, Export, Display chart, Create view

V. FUURE DIRECTIONS

- Integration with Smart City Initiatives:** Explore deeper integration with broader smart city frameworks. This involves collaborating with other urban management systems such as traffic control, public transportation, and environmental monitoring for a holistic urban experience.
- AI and Predictive Analytics:** Implement advanced artificial intelligence (AI) algorithms and predictive analytics to anticipate parking demand, optimize space allocation dynamically, and provide real-time recommendations for drivers.
- Automated Valet Parking:** Investigate the feasibility of automated valet parking systems where vehicles can autonomously find and park in designated spaces without human intervention.
- IoT Sensors and Edge Computing:** Utilize a more extensive network of Internet of Things (IoT) sensors for precise monitoring and leverage edge computing to process data locally, reducing latency and enhancing real-time decision-making.
- Mobile App Enhancements:** Enhance mobile applications for VPMS, providing users with features such as augmented reality navigation to available parking spaces, real-time traffic updates, and seamless payment options.
- Green Parking Solutions:** Integrate eco-friendly practices by promoting electric vehicle charging stations, incentivizing green vehicle parking, and incorporating sustainability metrics into the parking management system.

7. **Blockchain for Security and Transparency:** Investigate the use of blockchain technology to enhance security and transparency in transactions, ensuring secure and tamper-proof records of parking activities, payments, and reservations.
8. **Dynamic Pricing Models:** Implement dynamic pricing models that adjust parking fees based on demand, peak hours, or environmental considerations. This can help optimize revenue and encourage off-peak usage.
9. **Gesture Recognition and Biometrics:** Explore innovative user authentication methods such as gesture recognition or biometrics for a seamless and secure entry and exit process.
10. **Collaboration with Autonomous Vehicles:** Consider partnerships with autonomous vehicle manufacturers to develop parking solutions that cater specifically to self-driving cars, including designated parking areas and integration with autonomous vehicle navigation systems.
11. **Community Engagement and Feedback:** Establish mechanisms for community engagement, allowing users to provide feedback on the system, report issues, and suggest improvements to enhance user satisfaction.

VI. CONCLUSION

In conclusion, the research signifies the transformative impact of the Vehicle Parking Management System on urban mobility. By leveraging technology to streamline parking processes, enhance user experiences, and contribute to sustainable practices, the VPMS emerges as a key player in shaping the future of urban transportation. As cities continue to evolve, embracing innovative solutions like the VPMS becomes imperative for creating efficient, livable, and environmentally conscious urban spaces.

VII. REFERNCES

1. "The Death and Life of Great American Cities" by Jane Jacobs (1961) and "Traffic: Why We Drive the Way We Do (and What It Says About Us)" by Tom Vanderbilt (2008).
2. "A Survey on Internet of Things: Architecture, Enabling Technologies, Security, and Privacy" by Jayavardhana Gubbi et al. (2013).
3. "Big Data Analytics in the Smart City: Best Practices and Recommendations" by Jonathan Reichental (2017).