Kathmandu University

Department of Computer Science and Engineering

Dhulikhel, Kavre

A Project Proposal

On

"Parking Management System"

[Code No:]

(For partial fulfillment of 1st Year/2nd Semester in Computer Science/Engineering)

Submitted by:

Bikram Thapa (Roll No. 32) CS2

Aashutosh Sitaula (Roll No. 29) CS2

Kaustauv Bhandari (Roll No. 7) CS1

Hiten Mulmi (Roll No. 45) CS1

Yajju Dangol (Roll No. 18) CS1

Submitted to

Suman Shrestha

Department of Computer Science and Engineering

Submission Date: 24th April 2025

Abstract

The Parking Management System (PMS) is a project aimed at addressing the common issue of disorganized parking and the stress commuters face in locating available parking spaces. The problem is especially prominent in urban areas where the demand for parking exceeds supply. The main purpose of this project is to develop an efficient and user-friendly system that simplifies parking space management for both administrators and everyday users. The application will be developed using C/C++ for the backend logic and the Qt framework for designing a responsive graphical user interface. It will include user authentication, a parking space tracker, and a real-time statistics viewer to monitor usage and availability. The expected outcome is a powerful desktop-based application that can manage user data and dynamically display real-time updates about parking space status. In conclusion, this project aims to provide a convenient solution for managing parking facilities and reducing commuter frustration. It is recommended to further extend this system by integrating it with IoT sensors or GPS modules to make it adaptable for real-world, large-scale deployments.

Keywords:

Parking Management

User authentication

Real-time statistics

Table of Contents

Abstract i

List of Figures iii

Acronyms/Abbreviations iv

Chapter 1: Introduction 1

1.1 Background 1

1.2 Objectives 2

1.3 Motivation and Significance 2

1.4 Expected Outcomes 3

Chapter 2: Related Works 4

Chapter 3: Procedure and Methods 5

Requirement Analysis & Planning 5

System Design 5

Frontend Development (Qt GUI) 5

Backend Development (C++ + Database) 6

API Integration 6

Real-Time Statistics Viewer 6

Testing & Debugging 6

Deployment & Documentation 6

Flowchart of PMS..... 7

Chapter 4: System Requirement Specifications 8

4.1 Software Requirements 8

4.2 Hardware Specifications 8

Chapter 5: Project Planning and Scheduling 9

Task Breakdown and Duration 9

Gantt Chart 10

Appendix 11

References 12

List of Figures

Figure 1: Flowchart for PMS 7
Figure 2: Gannt Chart 10

Acronyms/Abbreviations

The list of all abbreviations used in this document is as follows:

Abbreviation	Full-Form
PMS	Parking Management System
IoT	Internet of Things
API	Application Programming Interface
UML	Unified Modeling Language
GUI	Graphical User Interface
HTTP	HypterText Transfer Protocol
UI	User Interface

Chapter 1. Introduction

1.1 Background

Parking management systems have become increasingly vital in urban areas due to rising vehicle ownership and limited parking spaces. The complexity of managing parking spots is growing, particularly in areas with high foot traffic, such as cities, universities, and commercial zones. Over the years, smart parking systems have emerged, leveraging IoT (Internet of Things) sensors, real-time data analytics, and mobile apps to help users locate available parking spots more efficiently. These systems are designed to reduce the time spent looking for parking, alleviate congestion, and improve the overall efficiency of urban spaces. Recent developments in this field have included the integration of cloud computing and machine learning algorithms to predict parking demand and suggest optimal parking spots based on real-time data. However, many existing systems are costly and require complex infrastructure, making them impractical for smaller institutions or local businesses. Moreover, many systems still face limitations related to scalability, data security, and user experience. The significance of a Parking Management System (PMS) lies in its ability to provide real-time information to users and administrators, enhancing the experience for both parties. While large-scale systems have made significant strides, simpler and more cost-effective solutions for local businesses, universities, and residential areas remain an underserved market.

1.2 Objectives

Develop a Parking Management System that efficiently tracks parking space availability. Create an intuitive user interface using Qt to ensure easy navigation for both administrators and users. Implement real-time data updates to allow users to view current parking space statuses. Enhance the system with a database for user management and parking history tracking.

1.3 Motivation and Significance

The motivation for choosing this project comes from the growing issue of parking congestion in urban areas, where finding a parking spot can often be a stressful and time-consuming task. The problem is especially prevalent in university campuses, commercial districts, and public spaces. Through this project, we aim to create a practical and effective solution that can be easily deployed in such environments. This Parking Management System addresses the shortcomings of existing solutions by offering a simplified approach that is easier to implement and maintain. Unlike many complex, expensive systems, this PMS will focus on offering real-time updates, user authentication, and an intuitive interface that can easily be adapted to local businesses or educational institutions. The project will also offer scalability for future integration with more advanced technologies like IoT sensors or GPS.

1.4 Expected Outcomes

The expected outcome of this project is a fully functional Parking Management System that can:

- Help users find available parking spaces in real-time.
- Allow administrators to manage parking spaces, user data, and parking history effectively.
- Provide a user-friendly interface that can be easily navigated by both users and administrators.
- Serve as a foundation for further development, including the integration of smart parking technologies like IoT sensors and mobile applications.

Chapter 2 Related Works

The increasing demand for efficient parking solutions in urban areas has led to the development of various smart parking systems. These systems aim to optimize parking space utilization, reduce traffic congestion, and enhance user convenience. One notable approach involves the integration of Internet of Things (IoT) technologies. For instance, a study by Singh et al. (2021) proposed an IoT-based smart parking system that utilizes sensors to detect vehicle presence and communicates this information to users via a mobile application. This system effectively reduces the time spent searching for parking spots and minimizes traffic obstruction. Another significant development is the implementation of cloud-based platforms for parking management. According to Zhang and Lee (2020), a cloud-integrated smart parking system allows for real-time data processing and scalability, enabling efficient management of parking resources across multiple locations. This approach also facilitates data analytics for predictive parking space availability. Despite these advancements, certain limitations persist. High implementation costs and the need for extensive infrastructure can hinder the adoption of such systems, especially in developing regions. Moreover, issues related to data security and user privacy remain concerns that need to be addressed. Our proposed Parking Management System (PMS) aims to overcome these challenges by offering a cost-effective and scalable solution. By leveraging C++ and the Qt framework, the PMS provides a user-friendly interface and real-time parking space monitoring without the need for extensive infrastructure. This approach ensures broader accessibility and ease of deployment, particularly in resource-constrained environments.

Chapter 3 Procedure and Methods

The development of the Parking Management System (PMS) will follow a structured and iterative approach that ensures each component of the system is carefully designed, implemented, and tested. The project will be developed using the C++ programming language and the Qt framework for the graphical user interface. The backend will include a database system to store user information, parking logs, and real-time slot statuses.

1. Requirement Analysis & Planning: The project begins with identifying functional and non-functional requirements. Research will be conducted on existing systems, user needs, and hardware constraints. A project timeline will be created to divide tasks into achievable milestones.
2. System Design: The overall architecture of the system will be outlined. This includes creating UML diagrams, flowcharts, and entity-relationship diagrams to model how users, administrators, and data will interact within the application.
3. Frontend Development (Qt GUI): The interface will be designed using Qt Designer. The GUI will allow users to log in, view parking availability in real-time, and book or release parking slots. Admins will be able to manage users and view system analytics.
4. Backend Development (C++ + Database): C++ will handle business logic, user authentication, and real-time processing. A lightweight database (like SQLite or MySQL) will be used to store persistent data such as user credentials, parking slot data, and activity logs.

5. API Integration: The API for the Parking Management System is developed using C++ with the help of the Crow framework, a lightweight web library ideal for building simple RESTful services. This API allows the system to handle core functionalities like user login, checking parking slot status, and booking a parking space through defined endpoints. These endpoints can be accessed from the main Qt application using the Qt Network module, which sends HTTP requests and handles the responses. By integrating this API, the system becomes more modular and enables real-time communication between the user interface and the backend logic, ensuring a smooth and efficient user experience.
6. Real-Time Statistics Viewer: A dynamic dashboard using Qt will be implemented to visualize current parking usage, availability, and history. This may involve custom charts or third-party Qt modules.
7. Testing & Debugging: Unit tests and integration tests will be conducted to ensure the software is reliable, secure, and user-friendly.
8. Deployment & Documentation: Once development is complete, the system will be packaged and deployed on target machines. User manuals and technical documentation will be prepared.

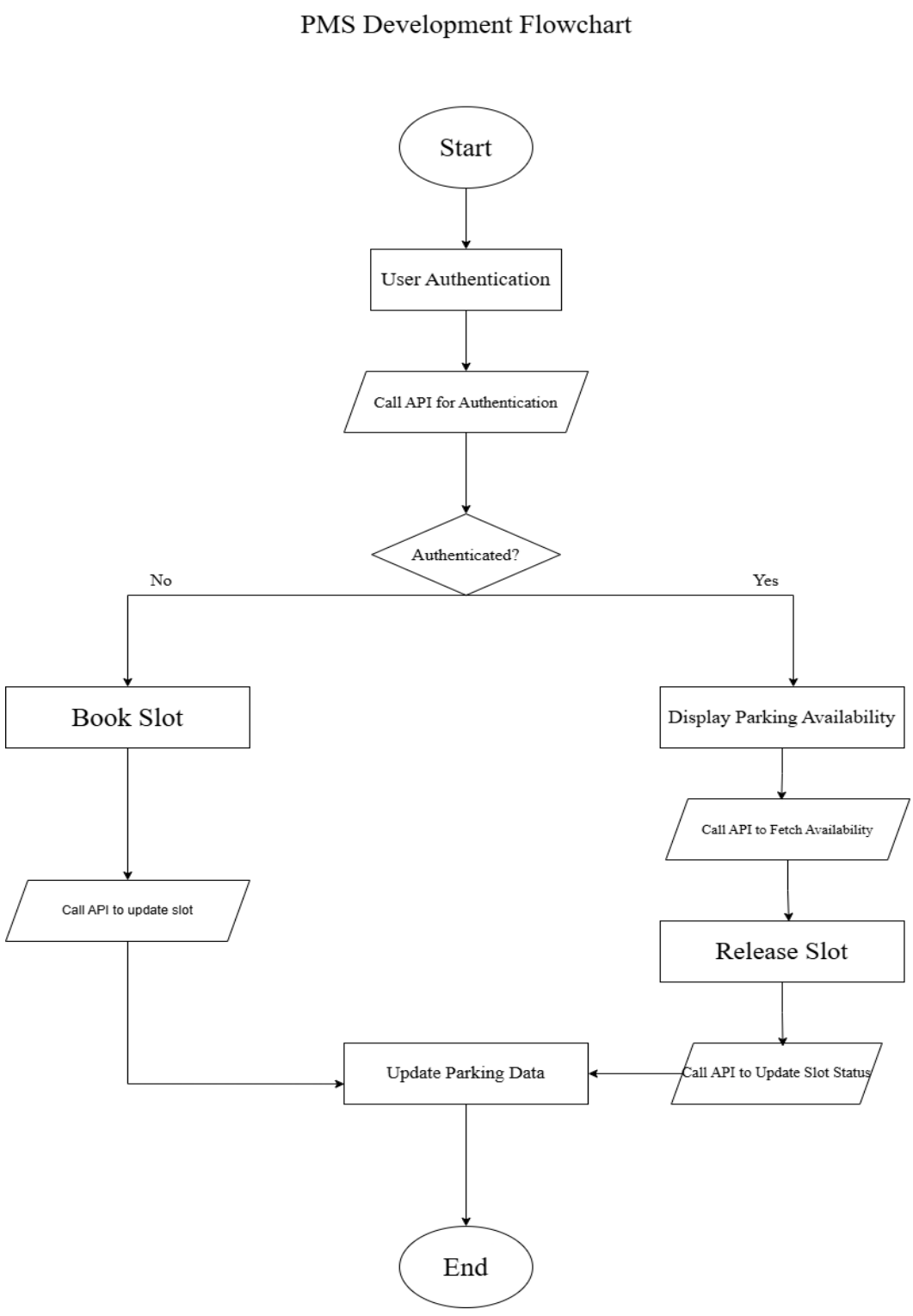

Figure 1: Flowchart for PMS

Chapter 4 System Requirement Specifications

4.1 Software Requirements

The Parking Management System (PMS) will require the following software components:

1. Programming Language: C++
2. UI Framework: Qt 6.2
3. Database: MySQL
4. Web Server (if needed): Apache (for hosting API services) Version
5. Git (for code management)

4.2 Hardware Specifications

Following are the hardware specifications required for the PMS:

1. Processor: Intel i3 or equivalent
2. RAM: 4 GB or more
3. Storage: 100 GB or more based on the user database (SSD recommended)
4. Operating System: Windows/Linux (Ubuntu or similar for development)
5. Graphics: Integrated graphics for UI
6. Network: Reliable internet connection for real time data synchronization

(Note: These specifications are arbitrary for the smooth operation of the system the actual system requirements will be confirmed once the system is developed.)

Chapter 5 Project Planning and Scheduling

Below are the key tasks involved in the development of the Parking Management System (PMS). These tasks are to be performed sequentially but may also overlap in some cases depending on project needs.

1. Problem Identification and Requirement Gathering:
   Objective: Understand the core problem (parking management) and gather detailed requirements.
   Duration: 4 days
2. Front-End Development (UI using Qt) :
   Objective: Develop the interface for the user as well as the admin based on the design mockup.
   Duration: 1 week
3. Back-End Development (C++ and Database integration):
   Objective: Development of the database and back-end logic as well as its integration with the front-end.
   Duration: 2 weeks
4. Database Design and API Development:
   Objective: Set up the database schema, tables, and relationships.
   Duration: 2 weeks
5. API Development:
   Objective: Develop API endpoints.
   Duration: 2 weeks
6. Validation:
   Objective
   Duration: 1 week

7. Real Time Statistics (Using C++ and Qt):

   Objective: Integrate a live data feed that updates parking availability in real time.

   Duration: 1 week

8. Testing and Debugging:

   Objective: Test the system for bugs and performance issues, and ensure smooth operation.

   Duration: 2 weeks

9. Final Review and Presentation:

   Objective: Final review of the project.

   Duration: 1 day

Gannt Chart:

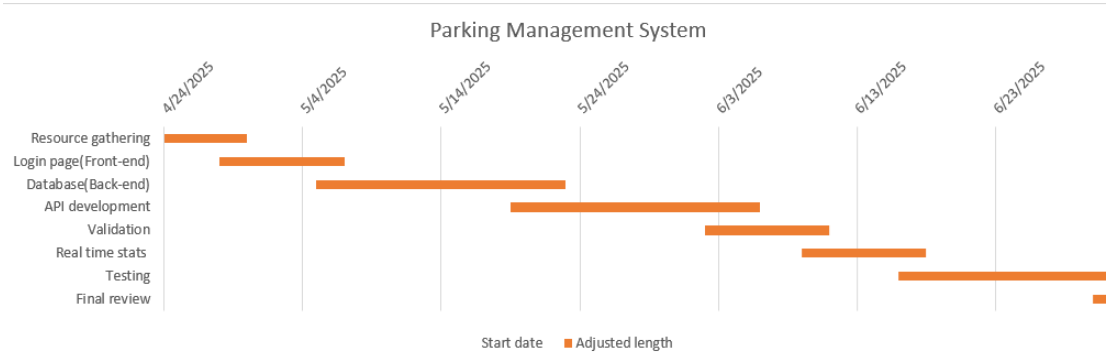

Figure 2: Gannt Chart

Appendix

The appendix of this document provides supplementary materials that support the development, design, and implementation of the Parking Management System. It includes detailed technical information, diagrams, and examples that are referenced throughout the proposal. These materials are intended to offer further insight into the system's architecture, functionality, and the methodologies employed in its development. Additionally, this section contains relevant data, such as API specifications, database schemas, and user interface designs, which contribute to a deeper understanding of the project's technical aspects and overall workflow.

Data Usage in parking management system:

Data Field	Description
Vehicle Number	Unique alphanumeric identifier for each vehicle
Entry Time	Timestamp of when the vehicle enters the parking lot
Exit Time	Timestamp of when the vehicle leaves the parking lot
Slot Number	Identifier for the assigned parking slot
Vehicle Type	Type/category of vehicle (e.g., car, bike, truck)
Duration	Time spent in the parking lot
Billing Amount	Calculated fee based on duration and vehicle type

References

- Qt Company. (2022). Qt for Application Development – Qt 6.2 Documentation. https://doc.qt.io/qt-6/
- Microsoft. (n.d.). C++ documentation. https://learn.microsoft.com/en-us/cpp/
- SQLite Consortium. (n.d.). SQLite: Official documentation. https://www.sqlite.org/docs.html
- CrowCpp Organization. (n.d.). Crow - A C++ microframework for web services. https://crowcpp.org/
- draw.io for flowchart design