SOFTWARE REQUIREMENTS SPECIFICATION

PROJECT TITLE:- Parking Management System

1. INTRODUCTION:-

1.1 Purpose

The purpose of the Parking Management System is to design and implement an efficient computerized system to manage the entry and exit of vehicles in a parking lot.
The system automates slot allocation, fee calculation, and record maintenance using fundamental data structures such as queues, stacks, array and linked lists etc.

1.2 Scope

The system maintains real-time parking status, including:

- Vehicle entry and exit operations
- Slot availability management
- Parking fee calculation based on duration

1.3 Defination

PMS- Parking management System

SRS- Software requirement specification

GUI- Graphical user interface

CLI- Command line interface

FIFO- First in first out

OOP- Object-oriented programming

1.4 Objectives

- To automate vehicle parking management.
- To minimize manual tracking of vehicles.
- To ensure efficient slot allocation and retrieval.
- To implement and demonstrate core DSA concepts.

1.5 System Overview

The Parking Management System simulates a parking slot with limited slots. When a vehicle enters, it is assigned an available slot. Upon exit, the system calculates the parking fee and releases the slot for new vehicles.

2. OVERALL DESCRPITION

2.1 Product Perspective

This is a standalone, console-based application (or optional GUI).
It uses in-memory data structures and optionally stores logs in files for persistence.

2.2 Product Functions

- **Add Vehicle:** Insert vehicle data into a linked list or queue.
- **Remove Vehicle:** Remove vehicle on exit and calculate charges.
- **Display Vehicles:** Show list of currently parked vehicles.
- **Search Vehicle:** Retrieve details by vehicle number (using hash map).
- **Manage Slots:** Track available and occupied slots.

2.3 User Characteristics

- Basic computer literacy.
- Understanding of vehicle types and time-based parking fees.

2.4 Constraints

- Limited number of slots (defined by the user).
- Console-based user interface.
- Requires accurate system time for fee calculation.
- Data loss on restart (unless file handling is implemented).

2.5 Assumptions and Dependencies

- The parking lot capacity is fixed.
- Each vehicle has a unique registration number.
- The system clock provides accurate time for entry/exit.

3. SPECIFIC REQIUREMENTS

3.1 Functional Requirements

ID	Requirement Description	Input	Output
FR-1	Vehicle Entry	Vehicle details (number, owner, type)	Slot assigned message
FR-2	Vehicle Exit	Vehicle number	Fee calculated, slot freed
FR-3	Display Records	Command	List of parked vehicles
FR-4	Search Vehicle	Vehicle number	Vehicle details
FR-5	Slot Management	Command	Number of available slots
FR-6	File Handling	Save/Load command	Persistent record storage

3.2 Non-functional Requirements

Category	Requirement
Performance	Entry and exit operations must execute in O(1) or O(n) depending on data structure used.
Reliability	Must handle invalid input gracefully.
Usability	Console menu-driven interface.
Scalability	Should support multiple vehicle types.
Maintainability	Modular code with separate functions for entry, exit, display, etc.
Portability	Should run on any system with a C++ compiler.

4. System Design

4.1 Data Structures Used

Feature	Data Structure	Purpose
Vehicle Queue	Queue / Linked List	Manage vehicle entry and exit
Vehicle Lookup	Hash Table	Fast search by vehicle number
Slot Tracking	Array	Keep track of free/occupied slots

Feature	Data Structure	Purpose
Fee Calculation	Stack or Time API	Compute total fee dynamically
Record Storage	File Handling	Store past parking records

4.2 Major Functions

- addVehicle()
- removeVehicle()
- displayAll()
- calculateCharges()
- saveToFile() / loadFromFile()

5. User Interface

Console Menu Example:

```
========== Parking Management System ==========
1. Vehicle Entry
2. Vehicle Exit
3. Display Parked Vehicles
4. Check Available Slots
5. Search Vehicle
6. Exit
==============================================
Enter your choice:
```

6. Performance Requirements

- Vehicle entry/exit operations must complete in under 2 second.
- System should handle up to 100–200 parking slots efficiently.

7. Future Enhancements

- GUI-based interface using C++ (Qt/SFML).
- Use database (MySQL / SQLite) instead of text files.

- Add admin login authentication using hashing.
- Implement sorting by entry time or vehicle type using trees.

8. References

- DSA textbook and class materials.
- C++ documentation for STL containers.
- Parking management workflow examples from real-life systems.