

**Abbottabad university of science & technology**

**SOFTWARE REQUIREMENTS SPECIFICATION   
 (SRS DOCUMENT)**

**For  
 <Parking lot management>**

**Submitted by : Fatima bibi**

**Submitted to : Sir Jamal Abdul Ahad**

**Roll No : 14646**

**Course : Data structure & Algorithm**

**Supervisor**

**(Sir Jamal Abdul Ahad)**

Table of content:

*[Parking Lot Management System:](#_Toc14195)* [2](#_Toc14195)

*[1. Introduction:](#_Toc8702)* [2](#_Toc8702)

[1.1 Purpose 3](#_Toc25625)

[1.2 Scope 3](#_Toc12926)

[1.3 Definitions, Acronyms, and Abbreviations 3](#_Toc31579)

[1.4 References 3](#_Toc28178)

*[2. Overall Description](#_Toc12726)* [4](#_Toc12726)

[2.1 Product Perspective 4](#_Toc18530)

[2.2 Product Features 4](#_Toc202)

[2.3 User Characteristics 4](#_Toc16061)

[2.4 Constraints 4](#_Toc31252)

[2.5 Assumptions and Dependencies 5](#_Toc10352)

*[3. Features and Functional Overview](#_Toc31844)* [5](#_Toc31844)

[3.1 Core Features 5](#_Toc8926)

[3.2 Optional Features 5](#_Toc18976)

*[4. Specific Requirements](#_Toc16180)* [5](#_Toc16180)

[4.1 Functional Requirements 5](#_Toc11930)

[4.2 Non-Functional Requirements 6](#_Toc1780)

[4.2.1 Performance 6](#_Toc4854)

[4.2.2 Usability 6](#_Toc21934)

[4.2.3 Reliability 6](#_Toc10110)

[4.2.4 Maintainability 6](#_Toc14745)

[4.3 Interface Requirements 6](#_Toc11346)

*[5. System Design](#_Toc23029)* [6](#_Toc23029)

[5.1 Modules 6](#_Toc31957)

[5.2 Data Storage 7](#_Toc6477)

*[6. Future Enhancements:](#_Toc23368)* [7](#_Toc23368)

*[7. Appendices:](#_Toc6943)* [7](#_Toc6943)

*[8 System Architecture:](#_Toc9838)* [7](#_Toc9838)

[8.1 Modular Design 7](#_Toc20341)

[8.1 System Components 8](#_Toc12598)

[8.2 Data Flow Diagram 8](#_Toc31845)

*[9. User Stories:](#_Toc29875)* [8](#_Toc29875)

*[10. Future Enhancements:](#_Toc15523)* [8](#_Toc15523)

[10.1 Planned Features 8](#_Toc22262)

[10.2 Scalability 8](#_Toc25475)

*[11. Security Considerations:](#_Toc21886)* [9](#_Toc21886)

*[12. Performance Metrics:](#_Toc31256)* [9](#_Toc31256)

*[13. Technical Specifications:](#_Toc6232)* [9](#_Toc6232)

*[14. Usability Scenarios:](#_Toc26206)* [9](#_Toc26206)

*[15. Limitations:](#_Toc23342)* [9](#_Toc23342)

*****Parking Lot Management System:*****

#### *****1. Introduction:*****

##### ****1.1 Purpose****

The Parking Lot Management System aims to streamline the process of allocating and managing parking spaces for bicycles, bikes, and cars. The system allows users to register vehicles, assign parking spaces dynamically, and manage vehicle entries and exits efficiently.

##### ****1.2 Scope****

This system will:

* Manage parking spaces for different types of vehicles (bicycles, bikes, cars).
* Dynamically allocate the nearest available parking space based on the vehicle type.
* Store and display information about parked vehicles.
* Support removal of vehicles and reallocation of parking spaces.
* Display available parking spaces.

The system is ideal for small to medium-sized parking lots where space management and record-keeping are critical.

PLMS is suitable for small to medium-sized parking lots. It dynamically assigns parking spaces using efficient algorithms, ensures that no two vehicles share the same space, and provides real-time data on space availability. The system can also generate reports for operational insights and is expandable to include additional features like automated payment systems.

##### ****1.3 Definitions, Acronyms, and Abbreviations****

* **SRS**: Software Requirements Specification

· **PLMS**: Parking Lot Management System

· **CLI**: Command-Line Interface

· **Heap**: A data structure used for efficient space allocation.

* **System**: Parking Lot Management System

##### ****1.4 References****

* Python Standard Library (heapq, time)

#### *****2. Overall Description*****

##### ****2.1 Product Perspective****

The system is a standalone application written in Python. It uses heap structures to efficiently manage available parking spaces, ensuring the nearest available space is allocated for each vehicle type.

PLMS is an independent application that does not rely on third-party systems. It operates using Python and leverages the heapq library for managing parking spaces. Future enhancements will include database integration and web-based interfaces for better accessibility.

##### ****2.2 Product Features****

* **Vehicle Registration**: Accepts input for vehicle number, type, name, owner, date, and time.
* **Dynamic Space Allocation**: Assigns parking spaces using a priority queue (heap).
* **Vehicle Removal**: Allows removing parked vehicles and reallocates the space.
* **View Parked Vehicles**: Displays a detailed list of all parked vehicles.
* **Available Spaces Overview**: Shows available parking spaces for each vehicle type.

##### ****2.3 User Characteristics****

The system is designed for parking lot attendants and administrators who need basic computer knowledge to operate the system.

##### ****2.4 Constraints****

* Input validation for date, time, and vehicle number format is implemented, but human errors in entry are still possible.
* Limited to managing a fixed number of spaces (78 for bicycles, 100 for bikes, 250 for cars).

##### ****2.5 Assumptions and Dependencies****

* The system assumes that vehicle numbers are unique.
* Python 3.x is required to run the application.

#### *****3. Features and Functional Overview*****

##### ****3.1 Core Features****

* **Add Vehicle**: Validate and record vehicle details, assign nearest parking space.
* **Remove Vehicle**: Allow removal and reallocation of freed parking space.
* **View Parked Vehicles**: List all parked vehicles with details.
* **View Available Spaces**: Display count of available parking slots.

##### ****3.2 Optional Features****

* **Error Notifications**: Alerts for invalid inputs or full capacity.
* **Real-Time Updates**: Space availability updated dynamically as vehicles enter/exit.

#### *****4. Specific Requirements*****

##### ****4.1 Functional Requirements****

**Add Vehicle**:

* + Validate and record vehicle information.
  + Assign the nearest available space based on vehicle type.
  + Save all entries persistently (future versions may add database support).

**Remove Vehicle**:

* + Search and remove a vehicle using the vehicle number.
  + Reallocate the freed parking space back to the heap.

**View Parked Vehicles:**

* + Display vehicle details including type, owner, and time of parking.

**View Available Spaces**:

* + Display the count of available spaces for each type of vehicle.

##### ****4.2 Non-Functional Requirements****

#### 4.2.1 Performance

The system must respond to user inputs within 2 seconds.

#### 4.2.2 Usability

The interface must be simple and text-based.

Error messages must be clear and user-friendly.

#### 4.2.3 Reliability

The system must handle invalid inputs gracefully without crashing.

#### 4.2.4 Maintainability

The code should be modular to allow for easy updates or enhancements.

##### ****4.3 Interface Requirements****

* **User Interface**: Command-line interface (CLI).
* **Hardware Interface**: Standard keyboard and display for input/output operations.

#### *****5. System Design*****

##### ****5.1 Modules****

* **Heap Management Module**: Uses heapq to maintain the nearest available parking space for each vehicle type.
* **Vehicle Information Module**: Handles vehicle registration and removal.
* **Display Module**: Manages views for parked vehicles and available spaces.

##### ****5.2 Data Storage****

Data is stored temporarily in lists during runtime. Persistent storage (e.g., files or databases) is not implemented but may be included in future versions.

#### *****6. Future Enhancements:*****

* Integrate a database for persistent storage of vehicle records.
* Add a graphical user interface (GUI) for better usability.
* Support for additional vehicle types and flexible space allocation.
* Implement analytics and reporting features.

#### *****7. Appendices:*****

**7.1 Sample Input/Output**

**Adding a Vehicle**:

* + Input: Vehicle number, type, owner, date, and time.
  + Output: Confirmation of parking space allocation.

**Removing a Vehicle**:

* + Input: Vehicle number.
  + Output: Confirmation of space reallocation.

**Viewing Available Spaces**:

* + Input: Menu option.
  + Output: Counts of available spaces by type.

#### *****8 System Architecture:*****

#### ****8.1 Modular Design****

* **Input Validation Module**: Ensures correctness of all user inputs.
* **Heap Management Module**: Manages parking space allocation dynamically.
* **Display Module**: Handles output for viewing parked vehicles and available spaces.

##### ****8.1 System Components****

* **Input Validation Module**: Ensures correct formats for vehicle details.
* **Heap Management Module**: Handles dynamic parking space allocation.
* **Reporting Module**: Generates real-time views of parked vehicles and spaces.

##### ****8.2 Data Flow Diagram****

* **Step 1**: User inputs vehicle details.
* **Step 2**: System validates input and checks space availability.
* **Step 3**: Allocates space and stores entry details.
* **Step 4**: Updates the parking space heap.

#### *****9. User Stories:*****

1. **As a Parking Attendant**, I want to quickly find available parking spaces for different vehicle types.
2. **As a Parking Manager**, I want to view all parked vehicles at a glance.
3. **As a Parking Lot Owner**, I want to ensure efficient space utilization to maximize revenue.

#### *****10. Future Enhancements:*****

##### ****10.1 Planned Features****

1. **Database Integration**: Persistent storage of parking data.
2. **Mobile App Support**: Access system features on mobile devices.
3. **Analytics Dashboard**: Insights into parking lot performance and usage trends.

##### ****10.2 Scalability****

The system will be scalable to manage larger parking lots with thousands of spaces and multiple vehicle types.

#### *****11. Security Considerations:*****

1. **Access Control**: Add user authentication to limit system access to authorized personnel.
2. **Data Validation**: Ensure robust input validation to prevent incorrect or malicious entries.
3. **Backup Mechanism**: Future versions will include automated data backup.

#### *****12. Performance Metrics:*****

1. **Operation Speed**: All core operations (e.g., add/remove vehicle) should execute within 1 second.
2. **System Uptime**: The system should be operational 99.9% of the time.
3. **Error Rate**: Less than 1% errors in allocation or data handling.

#### *****13. Technical Specifications:*****

* **Programming Language**: Python 3.x
* **Libraries**: heapq, time
* **Hardware Requirements**: Any standard PC or laptop with Python installed.

#### *****14. Usability Scenarios:*****

1. A parking attendant logs a new vehicle and assigns the nearest available space in seconds.
2. A manager views real-time parking availability before peak hours.
3. A vehicle exits, and its space is immediately marked as available for new entries.

#### *****15. Limitations:*****

1. The current version does not support concurrent multi-user operations.
2. Manual data entry may lead to occasional errors.
3. Space allocation is limited to predefined capacities.