UNIVERSITY OF PETROLEUM AND ENERGY STUDIES, DEHRADUN

Optimal Parking Spot Allocation

Software Requirement Specification (Minor Project-1, V Sem)

Submitted by Team No:

S.No.	Student Name	Roll No.	Sap ID
1	Akshay Mohpal	R2142210078	500088177
2	Diya Khandelwal	R2142210293	500090939
3	Gaurav Bhandari	R2142210311	500090993

BACHELOR OF TECHNOLOGY, COMPUTER SCIENCE ENGINEERING

With specialization in DevOps

Under the guidance of

Mr. Sandeep Pratap Singh

School of Computer Science (SOCS)

Department of CSO, UPES

Bidholi Campus, Energy Acres, Dehradun – 248007

Software Requirements Specification (SRS)

The introduction of the Software Requirements Specification (SRS) provides an overview of the entire
SRS with purpose, scope, definitions, acronyms, abbreviations, references and overview of the SRS. The aim of this document is to gather, analyze, and give an in-depth insight of the complete **Project** by defining the problem statement in detail. Nevertheless, it also concentrates on the capabilities required by stakeholders and their needs while defining high-level product features. The detailed requirements of the **Project** are provided in this document.

INDEX

1. **Introduction:**

Due to a lack of available space and strong demand, parking in cities has always been difficult. An optimal parking spot allocation system is suggested as a solution to this problem. According to a number of factors, this system seeks to effectively distribute parking spaces in order to maximize space usage and improve the overall parking experience.

Purpose: This document's aim is to give a comprehensive overview of the Optimal Parking Spot Allocation System, outlining its features, needs, and components.

Scope: When allocating parking in a defined area, the system will take into account criteria including availability, user preferences, and special considerations. Additionally, it will give users a way to request parking spaces and administrators a way to control the allocation procedure.

Definitions, Acronyms, and Abbreviations: Optimal Parking Spot Allocation: Enhancing Parking Efficiency.

Objectives

1. **Size-based Allocation:** Create an allocation system that considers the size of the user's vehicle and suggests spots that match the vehicle's dimensions

2. **Real-time Updates**: Design the system to handle real-time updates to the parking lot's occupancy status and dynamically adjust path-finding and allocation recommendations accordingly to calculate the most efficient routes from entry points to vacant parking spots within the parking lot.

3. **User-friendly Interface:** The interface should provide clear instructions, visual representations of parking spot availability, and an easy way for users to input their vehicle size and receive recommendations for the nearest available parking spot. The objective is to enhance the overall user experience and make the system accessible.

4. **Analytical View:** Provide an analytical view of the parking lot management system's performance by implementing and comparing different path-finding algorithms, such as Dijkstra's, to assess their efficiency in finding the most suitable routes from entry points to available parking spots within the parking lot.

2. **Over all descriptions**: A complex answer to the problem of parking in cities is the Optimal Parking Spot Allocation System. Through sophisticated algorithms and user preferences, space assignments are made more efficiently. It consists of an Administrative Interface for managing allocations and a User Interface for user interactions. Security safeguards protect user data, and performance is optimized even under heavy load. The system is built to be expandable.

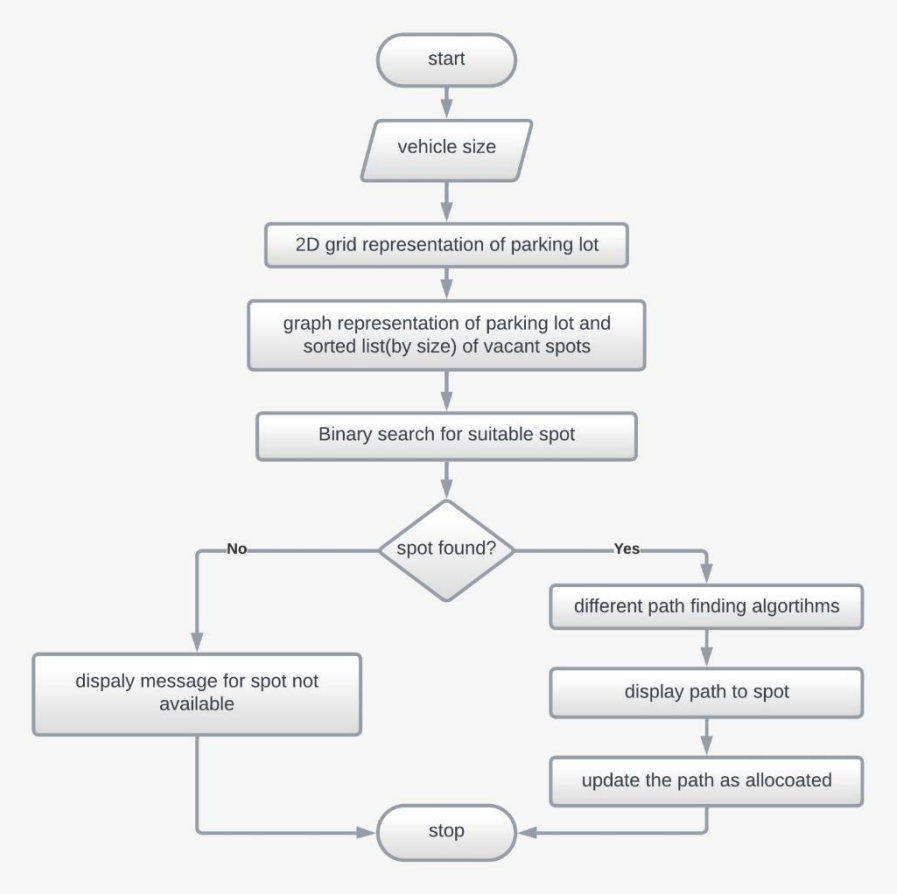


Fig: Proposed Flowchart

How the project works?

Initialization:

The program starts by opening a file ("occupancy.txt") to read the occupancy status of the parking lot.
It reads the occupancy status from the file and initializes a 2D matrix (array of Parking Spot objects) representing the parking lot.
The size of the parking lot (number of rows and columns) is set based on the data in the file.
Matrix Representation:

The parking lot is represented as a 2D matrix (ParkingSpotlot) of Parking Spot objects.

Each element of the matrix represents a parking spot with attributes like its size (S for small, M for medium, L for large), occupancy status (1 for occupied, 0 for unoccupied), and priority.
User Interaction:

The program interacts with the user by asking them to input their vehicle size (S, M, or L) in the main function.
It also gives the user the option to choose a combination of algorithms and approaches for finding the nearest parking spot.
Algorithm Selection:

Based on the user's choice of algorithm and approach, the program uses one of the following methods to find the nearest empty parking spot:
findNearestSpotUsingDijkastras with Dijkstra's algorithm and either linear or binary search.
FindNearestSpotBellmanFord with Bellman-Ford algorithm and either linear or binary search.
Finding the Nearest Spot:

The selected algorithm is applied to calculate the nearest parking spot based on the chosen approach.
The algorithm calculates distances and finds the parking spot with the minimum distance from the entrance (first row and first column in the matrix).
Display Results:

The program displays the results, including the chosen algorithm, approach, and the time taken to find the nearest spot.
It also displays the coordinates (row and column) of the nearest parking spot.
Matrix Update:

After finding and displaying the nearest spot, the program updates the occupancy status in the matrix to mark the spot as occupied (sets occupied to 1).
File Update:

The program saves the updated occupancy status to the "occupancy.txt" file, which reflects the current state of the parking lot.
Display Parking Lot Status:

It prints the current occupancy status of the parking lot, displaying each parking spot as a cell in a grid.
Unoccupied spots are shown in green, and the nearest spot is highlighted in a different color.
Error Handling:

The program handles errors, such as when there are no available parking spots for the given vehicle size or when the user provides an invalid option.
Memory Cleanup:

The program cleans up memory through the destructor when it is done. This involves deallocating memory for the 2D matrix of Parking Spot objects.

3. **Specific Requirements:**

Usability:

- The user interface must be simple to use and navigate.
- All user interactions must be accompanied by clear instructions and prompts.

- Users should require little effort to request parking spaces.

Reliability & Availability:

- The uptime of the system must be at least 99.9%.
- It must be built with little downtime for upgrades and maintenance.
- To avoid information loss, regular data backups must be carried out.

Performance:

- Under normal load levels, the system must reply to user requests in under seconds.
- It must be able to handle several users making requests simultaneously without significantly degrading its performance.

Security:

- Security must be maintained throughout user authentication and authorization.
- Modern encryption techniques must be used to save passwords.
- To prevent common internet vulnerabilities, there must be security measures in place.
- Access to sensitive information is only permitted for authorized individuals.

Supportability:

- A support team for the system must be accessible for help and problem solving.
- It must have a way for users to give comments and report issues or make ideas.

Design constraints:

- The solution must work with popular web browsers including Chrome, Firefox, and Safari.
- It must be created with mobile access on multiple devices in mind.

On-line User Documentation and Help System Requirements:

- Within the programmer, there should be quick access to FAQs and tutorials as well as other help materials.

Purchased components:

- For some functions (such as mapping services for location-based features), the system may use third-party components.

Interfaces:

- The system must offer interfaces for administrators and end users.
- In order to provide services like payment processing or mapping, it may integrate with external systems.

Licensing requirements:

- All applicable licensing agreements for any third-party components or libraries utilized by the system must be complied with.

Functional Requirements:

Legal copyright and other notices:

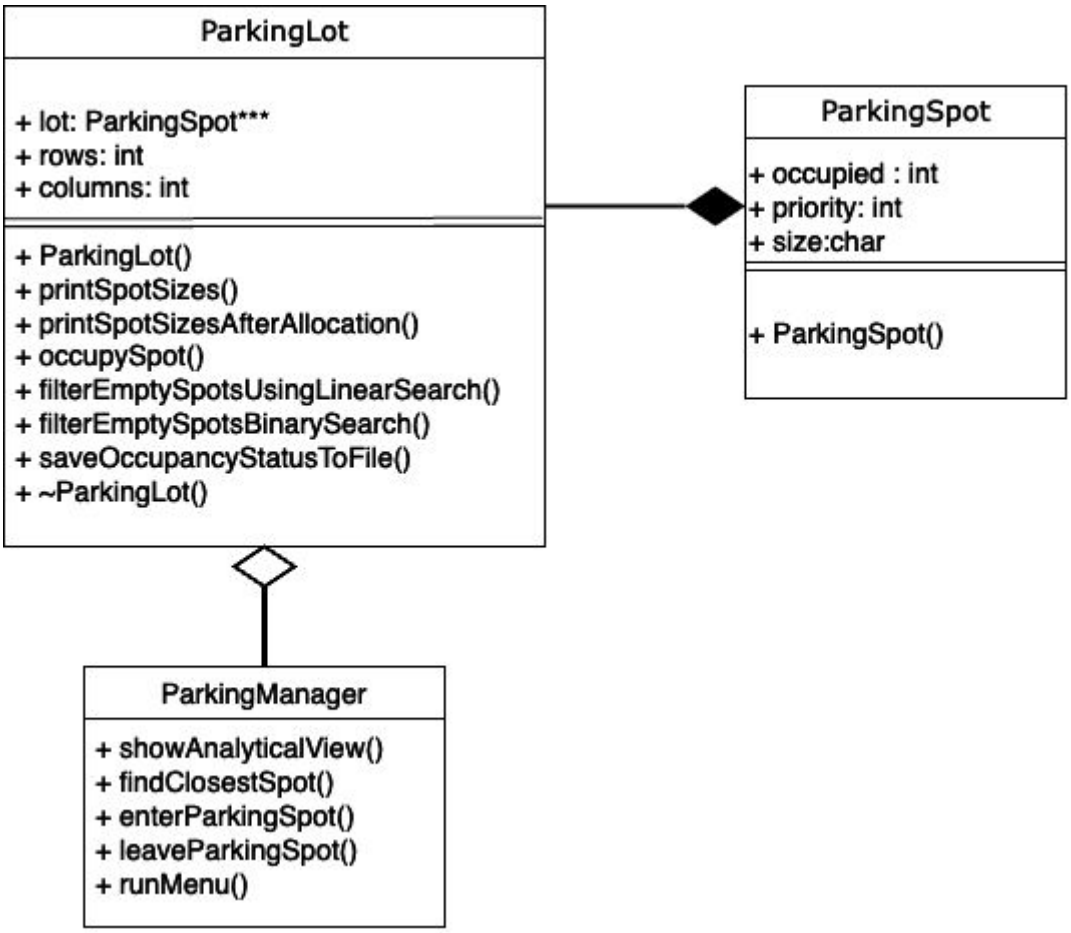
- It is necessary to be aware of and to respect all copyright and intellectual property rights.

Applicable Standards:

- The system must follow industry standards and best practices for user experience, security, and web application development.\

This section contains the functional requirements required of the Optimal Parking Spot Allocation system arranged by subsystem. The requirements in this section specify the functions that each subsystem must be capable of performing.

Class diagram:



1. **ParkingSpot Type:**

- A single parking space is represented.
- Size ('S', 'M', 'L'), occupancy status, and priority based on size are all attributes.
- There are no direct connections with other classes.

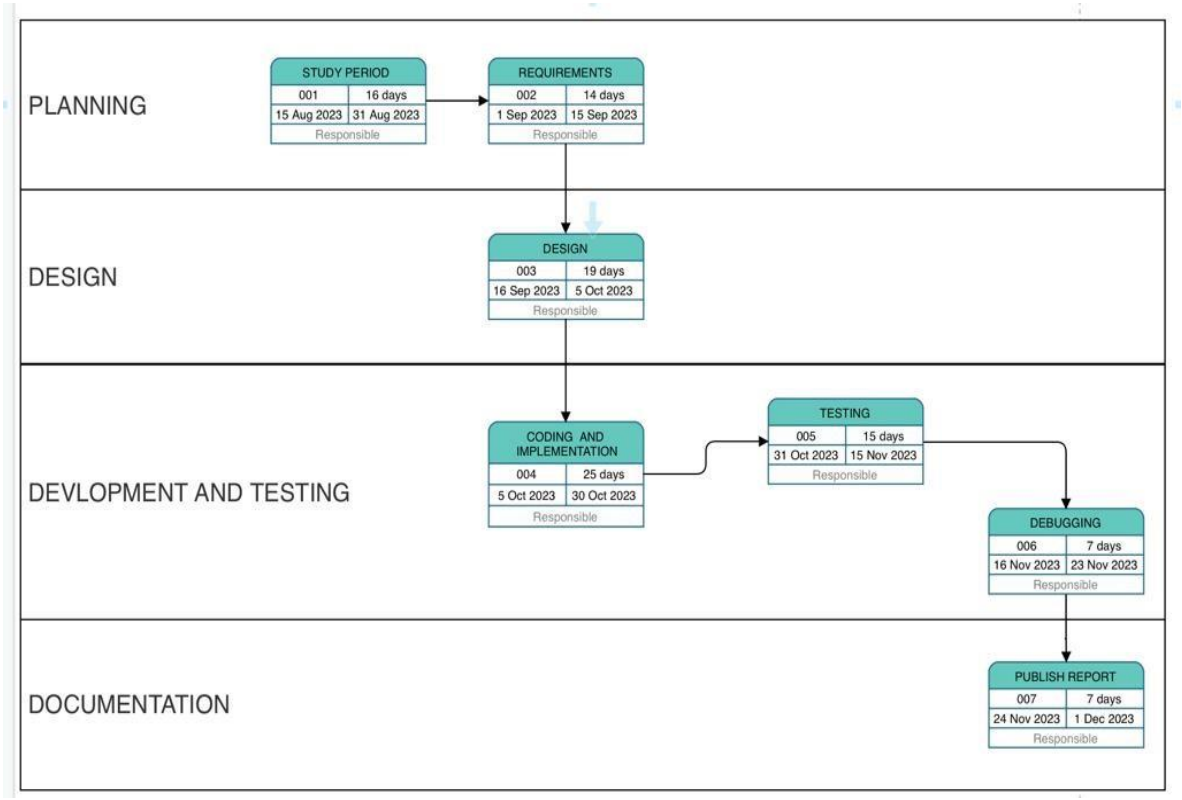
2. **ParkingLot Type:**

- A 2D array of ParkingSpot objects is used to manage the parking lot.
- Reads and initialises the parking lot from a file based on occupancy status.
- Filtering and detecting vacant places using linear or binary search algorithms are provided.
- Handles tasks such as occupying a parking space and saving the occupancy status to a file.
- There is a composition link between this class and the ParkingSpot class.

3. **ParkingLotManager Type:**

- The primary logic of the parking system is orchestrated.
- Uses a menu-driven interface to interact with the user.
- To discover the nearest open parking spot, it employs a variety of algorithms (Dijkstra's, Bellman-Ford, and A*).
- Controls user input for entering and exiting parking.

Pert Chart:



4. **Supporting Information:**

1. Use case analysis:

Use case analysis is a technique used in software development and requirements engineering to define and document how users or actors interact with a system or software application. It helps capture functional requirements and understand the system's behavior from the user's perspective. Below, some key use cases for our Optimal Parking Spot Allocation are outlined below:

1) Use Case: Find Nearest Parking Spot

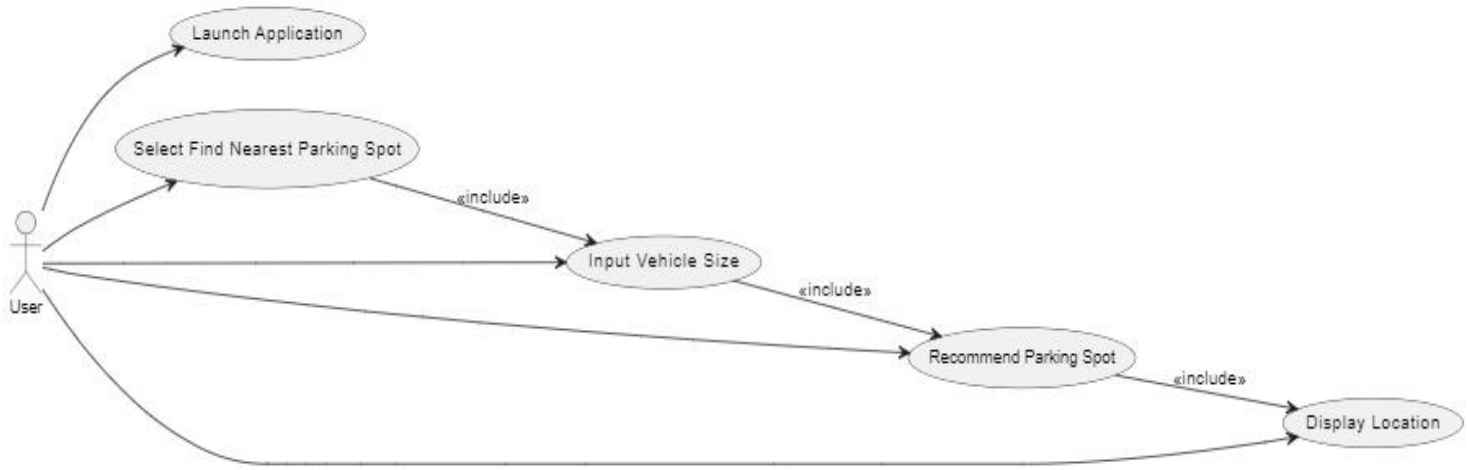
- Actors

- User (Customer)

Description:
The primary use case allows a user to find the nearest available parking spot based on their vehicle size.

Main Flow:
1. The user launches the Parking Lot Management System application.
2. The system displays a graphical user interface (GUI) with options.
3. The user selects the "Find Nearest Parking Spot" option.
4. The system prompts the user to input their vehicle size (Small, Medium, and Large).
5. The user provides the vehicle size.
6. The system uses the selected vehicle size to recommend the nearest available parking spot using the selected pathfinding algorithm and searching technique.
7. The system displays the recommended parking spot's location (Row and Column) to the user.
8. The use case ends.

Iterate Flow: Invalid Input
- If the user provides an invalid or unsupported vehicle size (e.g., "Extra Large"), the system displays an error message and returns to step 4 of the main flow.



2) Use Case: View Parking Lot Information

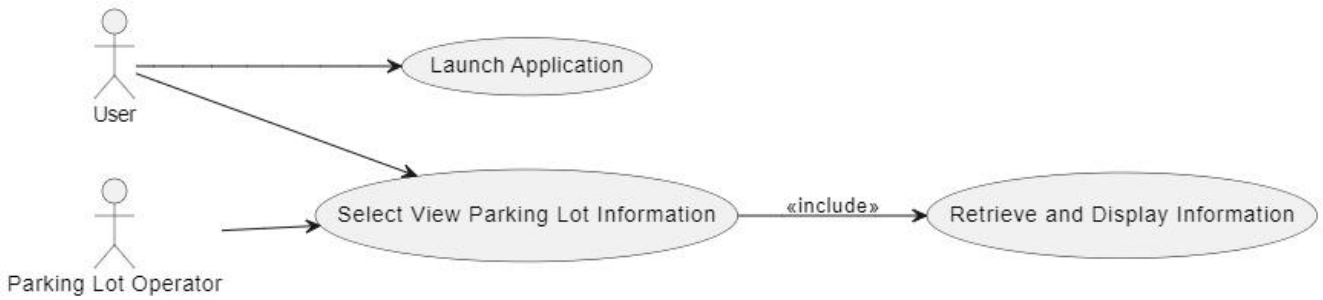
- Actors

- User (Customer)
- Parking Lot Operator

Description:
This use case allows users and parking lot operators to view parking lot information, including spot sizes and occupancy status.

Main Flow (For User):
1. The user launches the Parking Lot Management System application.
2. The system displays a graphical user interface (GUI) with options.
3. The user selects the "View Parking Lot Information" option.
4. The system retrieves and displays the parking lot grid with spot sizes and occupancy status.
5. The user can examine the parking lot information.
6. The use case ends.

Main Flow (For Operator):
1. The parking lot operator launches the Parking Lot Management System application.
2. The system displays a graphical user interface (GUI) with options.
3. The operator selects the "View Parking Lot Information" option.
4. The system retrieves and displays the parking lot grid with spot sizes and occupancy status.
5. The operator can monitor the parking lot's occupancy and status.
6. The use case ends.



3) Use Case: Generate Comparative Analysis Report

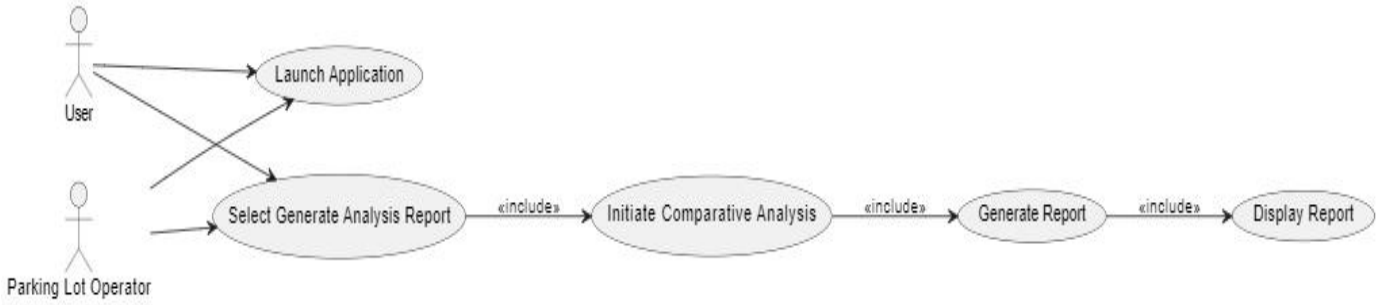
- Actors

- User (Customer)
- Parking Lot Operator

Description:
This use case allows users and parking lot operators to generate comparative analysis reports of pathfinding algorithm and searching technique performance.

Main Flow:
1. The user or parking lot operator launches the Parking Lot Management System application.

2. The system displays a graphical user interface (GUI) with options.
3. The user or operator selects the "Generate Comparative Analysis Report" option.
4. The system initiates the comparative analysis process.
5. The system generates a report that includes data on the performance of different algorithms and techniques.
6. The system displays the report to the user or operator.
7. The user or operator can review and analyze the report.
8. The use case ends.



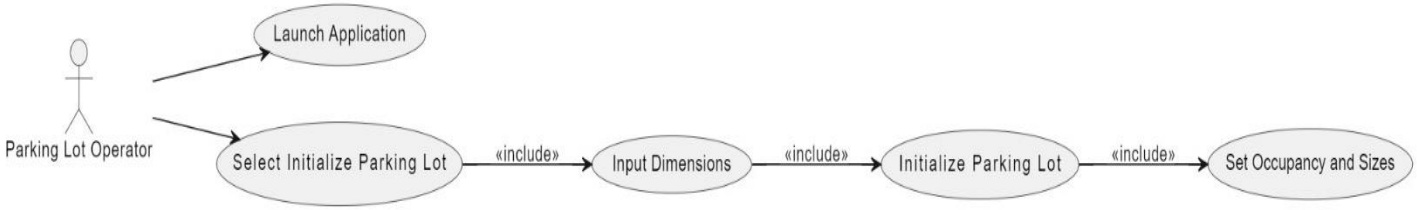
4) Use Case: Initialize Parking Lot

- Actors

- Parking Lot Operator

Description
This use case allows parking lot operators to initialize the parking lot with spot sizes and occupancy status.

Main Flow
1. The parking lot operator launches the Parking Lot Management System application.
2. The system displays a graphical user interface (GUI) with options.
3. The operator selects the "Initialize Parking Lot" option.
4. The system prompts the operator to input parking lot dimensions (rows and columns).
5. The operator provides the dimensions.
6. The system initializes the parking lot grid based on the provided dimensions.
7. The operator can set the occupancy status and spot sizes for each parking spot.
8. The operator saves the occupancy and spot size data.
9. The use case ends.



These are just a few use cases to get started. It can be further expanded and refined these use cases based on your project's specific requirements and features. Use case analysis is a valuable tool for capturing and documenting how users interact with your system and what functionality is expected.

2. **Structural models.**

   The process of building a system to optimally distribute available parking spaces to automobiles is known as optimal parking spot allocation. This process of allocation is critical in urban planning and transportation management. It tries to optimize aspects such as proximity to locations and walking distance for vehicle owners. The complexity and approach of the models employed for this purpose can vary. To reach the greatest possible allocation result, they frequently employ algorithms and optimization approaches. These models play an important role in improving the overall efficiency of parking facilities in metropolitan areas.

3. **Behavioral models.**

   Behavioral models for optical parking place distribution are concerned with understanding how drivers see and choose parking spaces visually. They take into account elements such as visibility, distance estimation, and ease. These models aid in the optimization of parking layouts based on human behavior.

4. **Nonfunctional requirements model.**

   i. **Correctness-**

   Syntax, logic and all the programming code methods must be applied precisely to avoid errors.

   ii. **Flexibility-**

   It is able to run various algorithms and provide the free spaces, as we require.

5. **Project Plan**

   The project Optimal Parking Spot Allocation intends to transform parking operations in high-traffic places such as retail malls, airports, and event venues. This strategic project aims to maximize space use, improve customer experience, reduce congestion, and improve accessibility.

   a. **Planning and preparation:**

- Examine the present parking facility thoroughly.
- Gather information on size, traffic patterns, and unique requirements.
- Define vehicle size classifications and accessibility norms.

   b. Design of a Traceability Matrix:

- Based on the facts gathered, create a customized Traceability Matrix.
- Assign variables such as available space and vehicle size.
- Set the allocation algorithm's parameters.

c. **Algorithm Development for Allocation:**
   - Create a complex algorithm using the Traceability Matrix.
   - To achieve correct parking place assignments, rigorously test the algorithm with sample data.

d. **Real-time Updates:**
   - Set up a monitoring system to track and manage parking allocations on a real-time basis.

Code Summary:

1. **Classes:**

- **ParkingSpot:** A single parking place with features such size ('S' for small, 'M' for medium, and 'L' for large), occupancy status (1 for occupied, 0 for vacant), and priority based on size.

- **ParkingLot:** Controls the parking lot, which is a two-dimensional array of ParkingSpot objects. It reads and initialises the parking lot from a file depending on occupancy state, provides ways for filtering and finding empty spaces using linear or binary search, and handles tasks such as occupying a place and writing occupancy status to a file.
- **ParkingLotManager:** Orchestrates the parking system's basic logic. It communicates with the user, locates the nearest available parking spot using various algorithms, and provides an analytical perspective of the parking lot. It also has functions for entering and leaving parking spots.

2. **Algorithms:**
   To discover the nearest open parking spot, the code applies several techniques, including Dijkstra's algorithm (both linear and binary search versions), the Bellman-Ford algorithm, and the A* algorithm. These algorithms take distance from the entrance into account and prioritise slots based on size and occupancy status.

3. **User Interaction:**
   The ParkingLotManager class provides the user with a menu-driven interface. Users have the option of entering a parking spot (specifying vehicle size), leaving a parking spot, or exiting the programme.

4. **File Handling:**
   The code reads and writes the status of occupancy to a file (occupancy.txt). After a parking spot is occupied, the file is used to initialise the parking lot and preserve the occupancy state.

5. **Visualisation:**
   The code provides routines for printing the current status of the parking lot as well as a visual representation of the parking lot when a vehicle has been parked, emphasising the closest spot.

6. **Handling Exceptions:**
   Exception handling is included in the code to solve potential issues such as invalid user input or when no parking spaces are available.

7. **Optimisation:**
   The optimisation part involves efficiently locating the nearest available parking spot using various search algorithms and minimising the search space based on the vehicle's size.

In summary, the code represents a programme for an optimised parking system including user interaction, algorithmic approaches to parking spot detection, and efficient data management for parking lot status.

The C++ code provided implements an optimized parking system. A ParkingSpot class represents individual parking spaces, a ParkingLot class manages the parking lot layout and occupancy status, and a ParkingLotManager class orchestrates the core logic. Based on multiple search techniques, the system applies several algorithms, such as Dijkstra's, Bellman-Ford, and A*, to discover the nearest accessible parking spot. A menu-driven interface facilitates user interaction, allowing users to enter and exit parking slots. The programming prioritizes efficiency through algorithms and displays visual representations of the parking lot's status. Furthermore, occupancy status is read from and saved to a file, which contributes to system durability across executions.

Conclusion:

This Software Requirements Specification outlines the requirements for the Parking Spot Management System, including the implementation of pathfinding algorithms and searching techniques for finding the nearest available parking spot. The system aims to conduct a comparative study to evaluate the performance of these algorithms and techniques.

The "Optimal Parking Spot Allocation System" project uses this approach to compare and assess pathfinding algorithms, allowing for the informed selection of the most efficient algorithm for real-time parking spot distribution. This improves the overall user experience by optimizing the allocation process.

References:

[1] A Balanced Algorithm for In-City Parking Allocation: A Case Study of Al Madinah City Mohammad A. R. Abdeen,* Ibrahim A. Nemer,and Tarek R. Sheltami
Giovanni Pau, Academic Editor

[2] An Improved Vehicle Parking Mechanism to reduce Parking Space Searching Time using Firefly Algorithm and Feed Forward Back Propagation Method by Ruby Singh1*, Chiranjit Dutta, Niraj Singhal, Tanupriya Choudhury

[3] Smart-parking management algorithms in smart city Mahdi Jemmali, Loai Kayed B. Melhim1*, MafawezT.Alharbi, Abdullah Bajahzar & Mohamed Nazih Omri

[4] Automated parking system using graph algorithm by Ashim Md. Ahsan Fahim and Shahrin Chowdhury