

Project Title

Automated Parking Space Detection System

Team Members:

Mohammed Sami 23BBS0006

Kartik Batla 23BBS0017

Aryan Arya 23BBS0142



The process of making ParkingBuddy

Topic	Page No.
1. Introduction	3
◦ Why did we create ParkingBuddy?	
◦ Why should you use ParkingBuddy?	
2. Defining the Scope of the project	7
◦ What does ParkingBuddy aim for?	
◦ The 4 main objectives	
◦ Project Scheduling	
3. SRS document	10
◦ Introduction	
◦ Defining Functional requirements	
◦ Defining Non functional requirements	
4. Software Design	14
5. Codes	17

- File structure of the application
- The main model's code base

6. Software Testing

- Functional Testing
- Non-Functional Testing

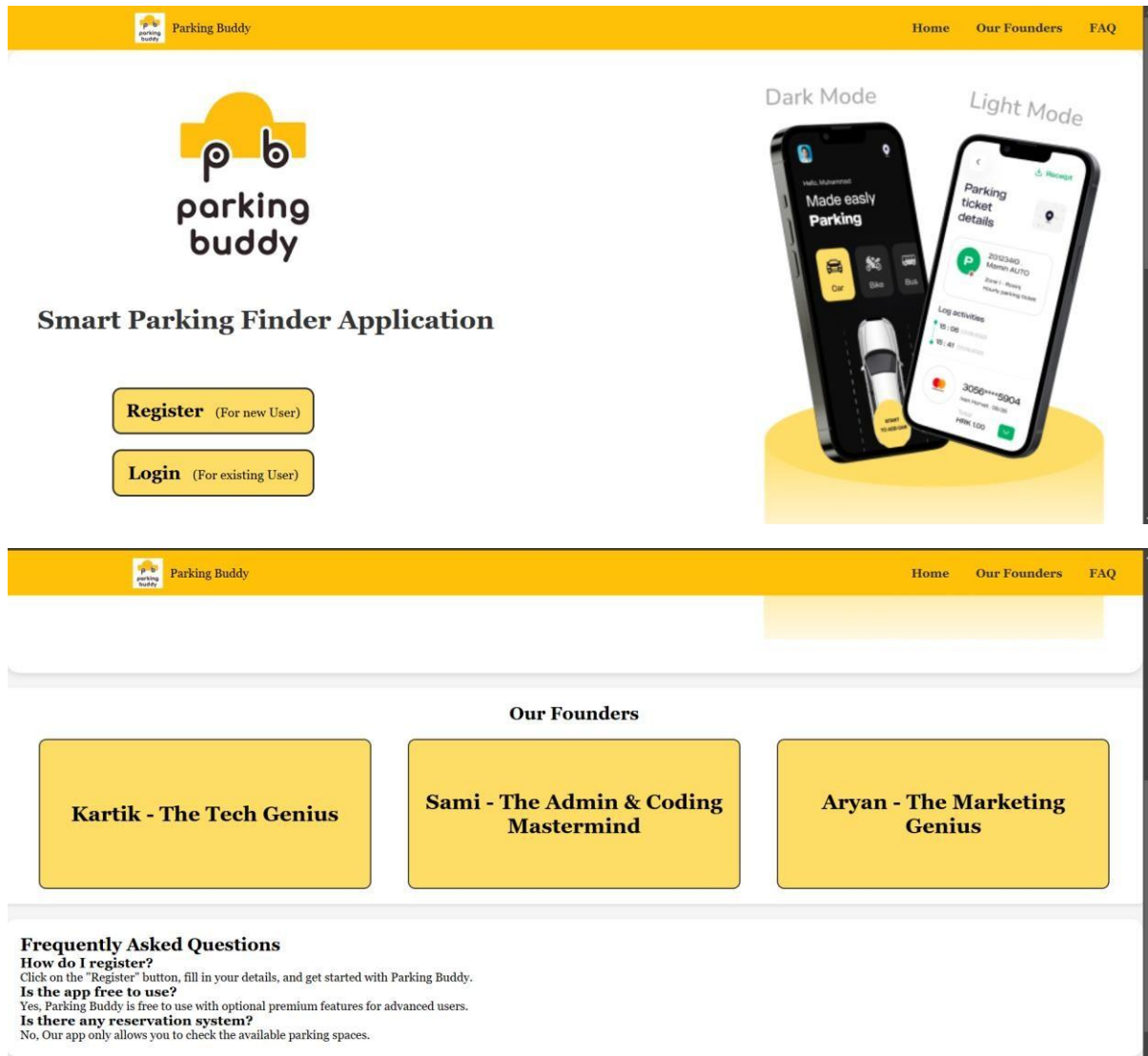
Why ParkingBuddy?



The idea for ParkingBuddy originated from the need to address the challenges faced by drivers in **finding available parking spaces**, especially in congested urban areas. The request for this software was triggered by an **increasing demand for a more efficient and automated solution to parking management**. The goal of the proposed system is to provide **real-time parking space detection**, allowing users to easily locate and navigate to available spots. The system will include functionalities such as space detection, user interface for viewing parking availability, and administrative tools for managing parking lots. The overall objective is to enhance user convenience, reduce parking-related stress, and optimize parking space utilization.

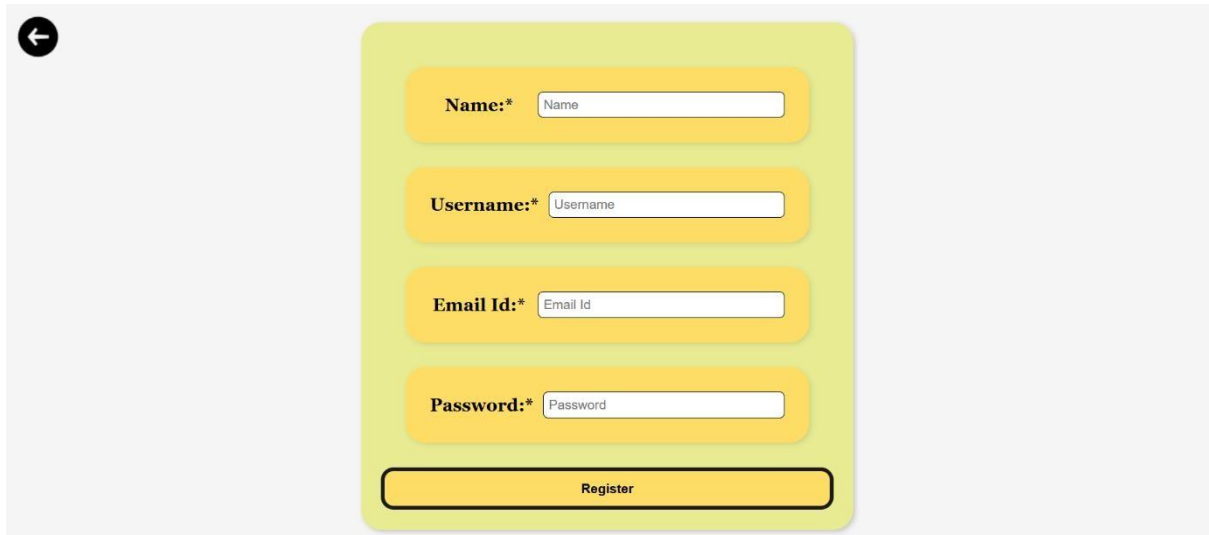
How does ParkingBuddy help you?

ParkingBuddy – Frontend Design/ User Interface:



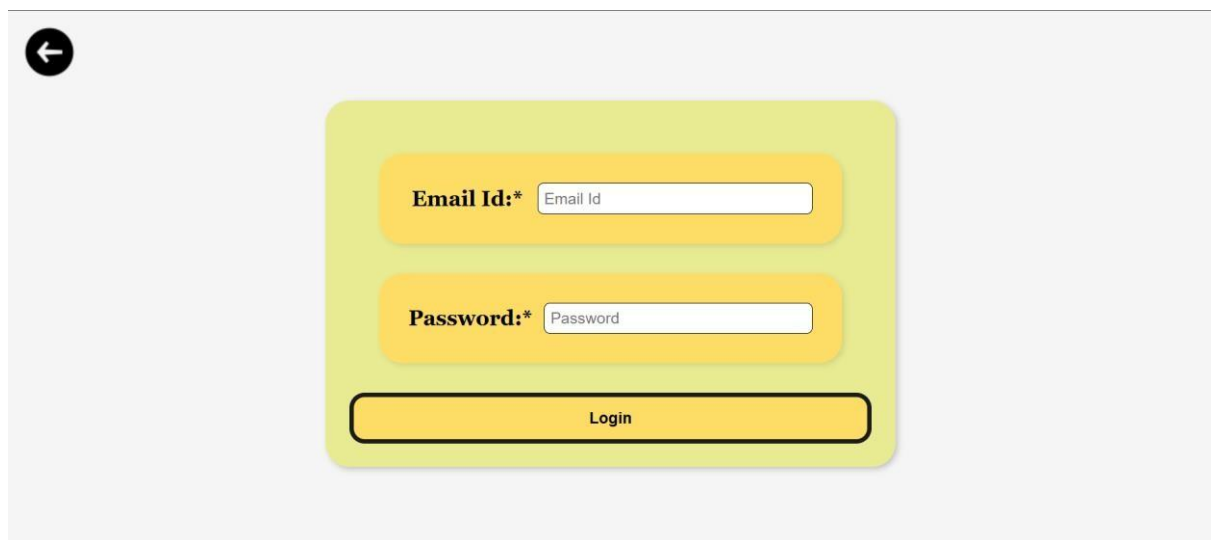
Landing Page:

Register option (for new users):



A registration form for new users. It features a back arrow icon in the top left corner. The form is contained within a yellow rounded rectangle and includes four input fields: 'Name:*' with a placeholder 'Name', 'Username:*' with a placeholder 'Username', 'Email Id:*' with a placeholder 'Email Id', and 'Password:*' with a placeholder 'Password'. Below these fields is a yellow 'Register' button.

Login (for registered users):



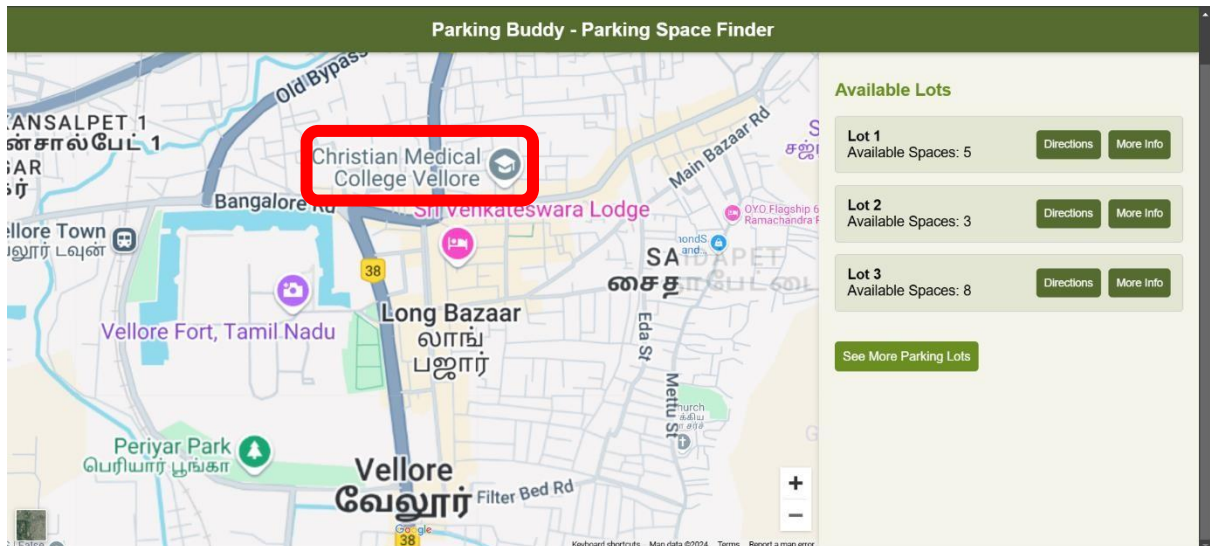
A login form for registered users. It features a back arrow icon in the top left corner. The form is contained within a yellow rounded rectangle and includes two input fields: 'Email Id:*' with a placeholder 'Email Id' and 'Password:*' with a placeholder 'Password'. Below these fields is a yellow 'Login' button.

After successful Registration/Login:

Enter the location you want to park in:

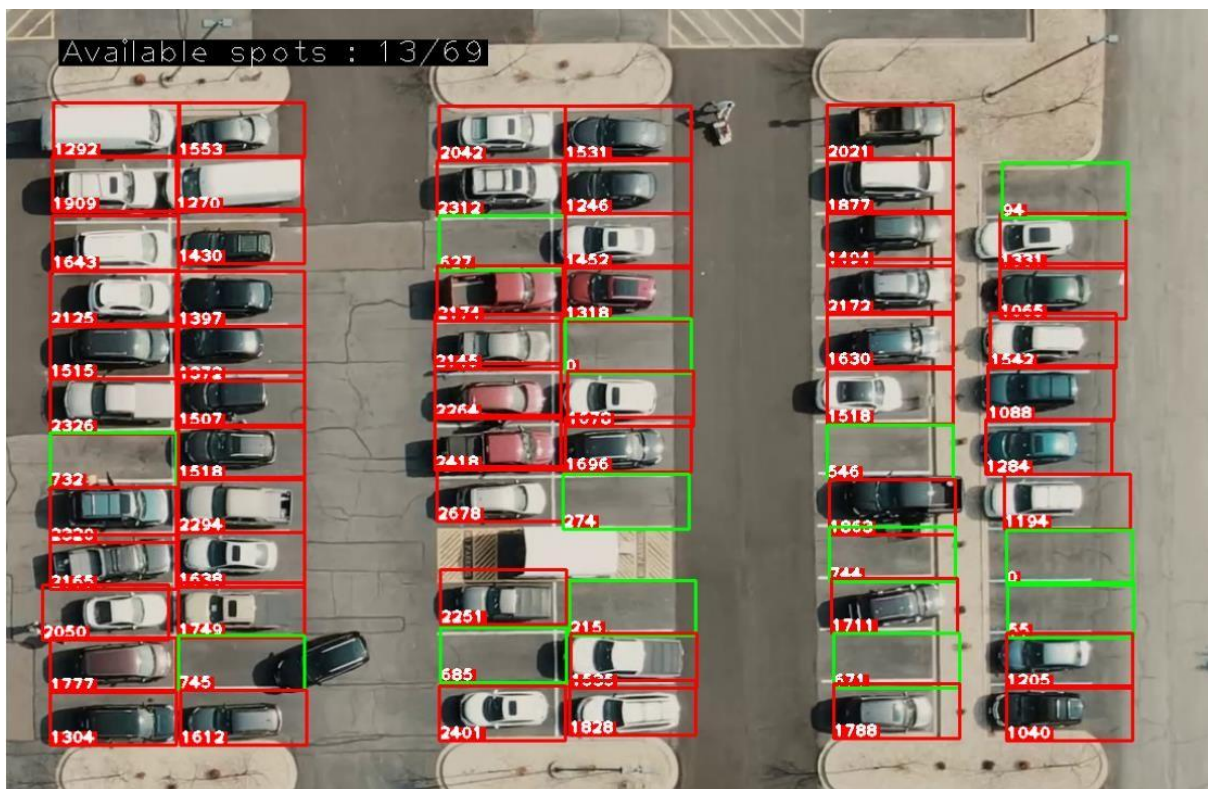


Then our page will lead you to the **nearest parking lots: (integrated Google Maps)**



Finally, after selecting a parking lot:

ParkingBuddy will show you the **real-time footage of the exact parking lot:**



Voila! You have successfully saved your precious time
by using ParkingBuddy!

SCOPE :

The Parking Buddy project focuses on developing a sophisticated system to detect and monitor parking spaces in real time using video feeds. The system is designed to analyze live video to identify available parking spots and provide updates on their status. In addition to automated detection, Parking Buddy includes a manual configuration tool that allows users to define and adjust the positions of parking spaces on a static image of the parking lot. This feature enables customization for various parking lot. This feature enables customization for various parking lot layouts.

The project does not encompass the integration with physical parking management systems or advanced functionalities such as dynamic pricing, pricing and automated reservation system.

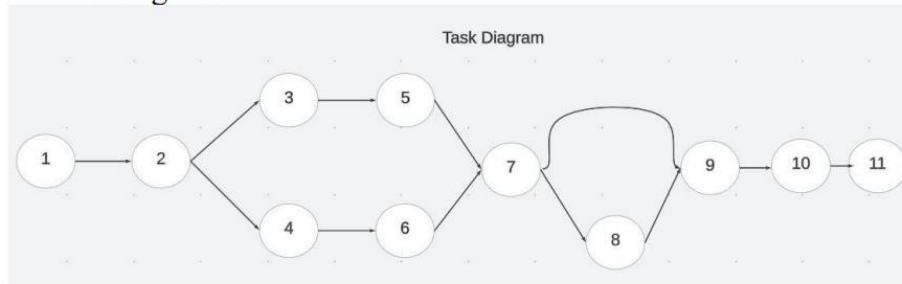
OBJECTIVES :

The primary objectives of the Parking Buddy Project are:

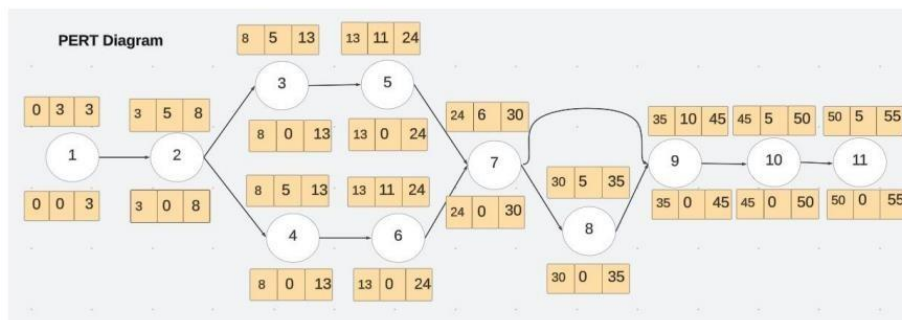
1. Develop a Real Time Parking Space Detection system:
Create a solution that accurately identifies and counts available parking spaces from video feeds.
2. Implement a user-friendly Manual selection tool:
Design an intuitive interface that allows users to define and adjust parking space positions on a parking lot image.
3. Ensure System Reliability and Performance:
Optimize the system for various lighting conditions and parking lot configurations to provide consistent and reliable performance.
4. Deliver clear and Informative Visual Feedback:
Develop features that offer real time updates and clear visual indicators of parking space availability to users.

Scheduling diagrams –

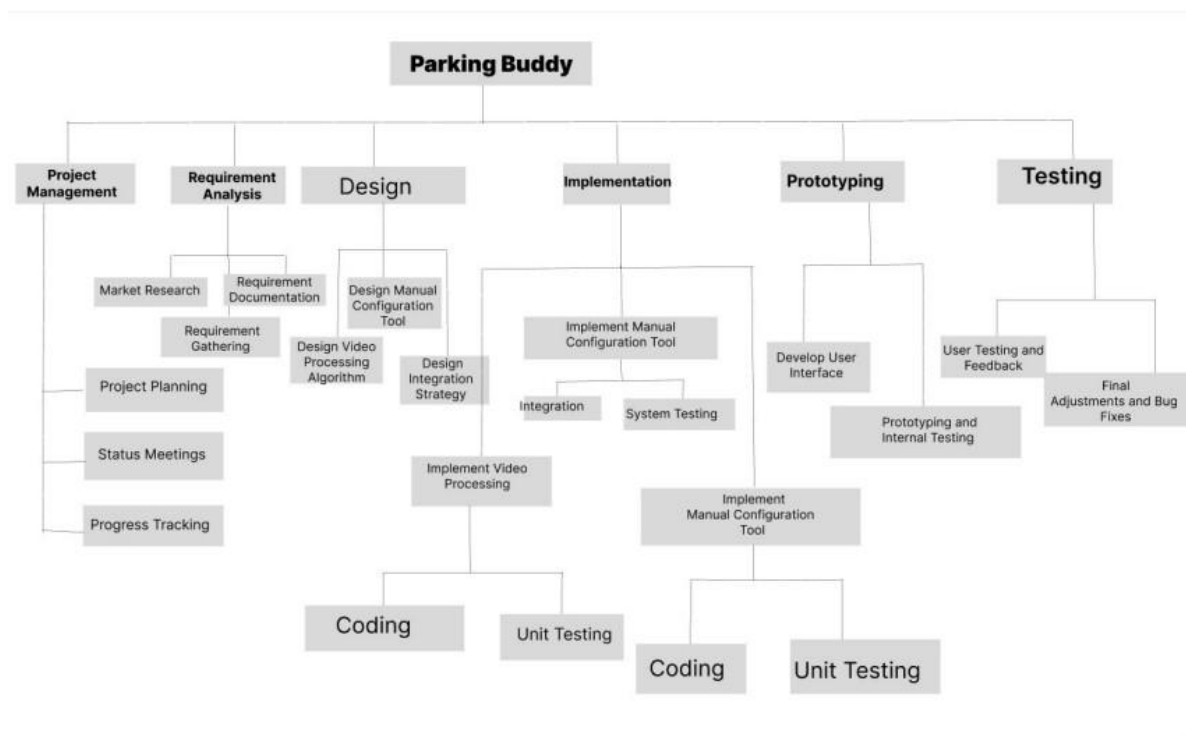
1.Task Diagram



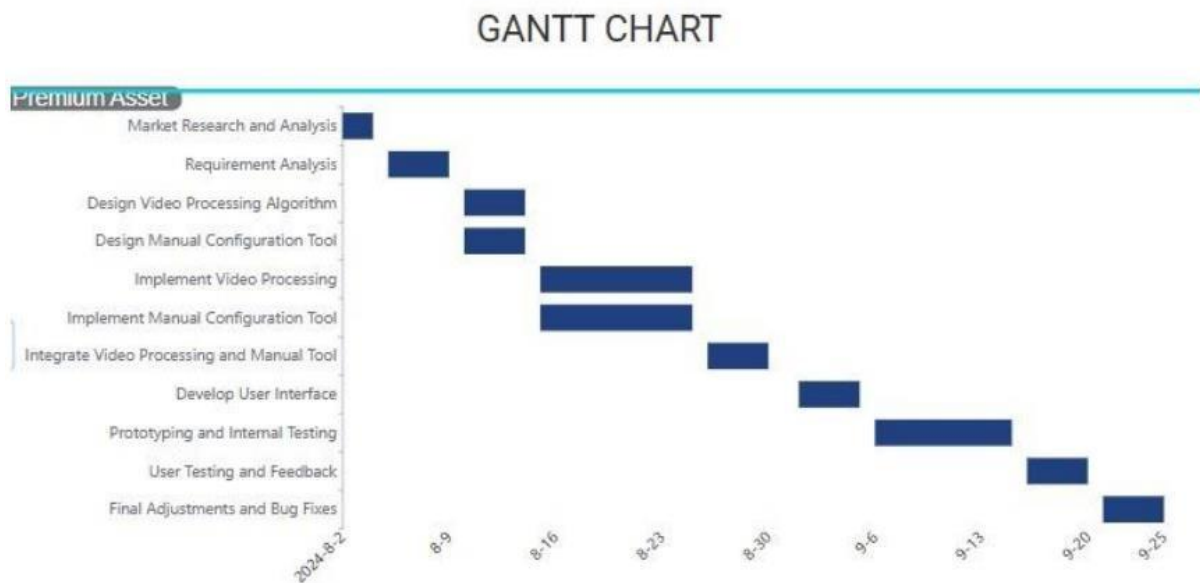
2. Pert Diagram –



3.Work Breakdown Structure - WBS



4.GANTT chart-



The SRS document:

Software Requirements Specification (SRS) Document for Parking Space Finder

1. Introduction

A. Purpose

The purpose of this Software Requirements Specification (SRS) document is to outline the functional and non-functional requirements for the Parking Space Finder project. This project utilizes OpenCV and computer vision techniques to process images and provide real-time updates on parking space availability. The system is designed to assist users, including drivers, passengers, and administrators, in efficiently managing and locating parking spaces through a web-based interface. The document serves as a guide for developers, stakeholders, and project managers to ensure that the project aligns with the intended goals and meets all required specifications.

B. Scope

The Parking Space Finder project originated from the need to address the challenges faced by drivers in finding available parking spaces, especially in congested urban areas. The request for this software was triggered by an increasing demand for a more efficient and automated solution to parking management. The goal of the proposed system is to provide real-time parking space detection, allowing users to easily locate and navigate to available spots. The system will include functionalities such as space detection, user interface for viewing parking availability, and administrative tools for managing parking lots. The overall objective is to enhance user convenience, reduce parking-related stress, and optimize parking space utilization.

C. Definitions, Abbreviations, and Acronyms:

Term	Definition
OpenCV	Open-Source Computer Vision Library used for image processing
GUI	Graphical User Interface, the visual component of the system
Admin	System administrator responsible for managing the website.
Driver	User seeking parking space using the web interface.
Passenger	User accompanying the driver, may also use the web interface
Parking Lot	A designated area where vehicles are parked.
SRS	Software Requirements Specification.

D. References

- OpenCV Documentation: <https://docs.opencv.org/>
- Sommerville, I. (2016). *Software Engineering* (10th ed.).
- Web Development Resources: <https://developer.mozilla.org/>
- IEEE Standard for SRS: <https://ieeexplore.ieee.org/document/7294755>

E. Overview

This document details the requirements for the Parking Space Finder project, including the general description, functional and non-functional requirements, system architecture, and system model. It is structured to provide a comprehensive guide for the development team, ensuring that the system meets the needs of all stakeholders and achieves the desired functionality.

2. General Description

A. Product Perspective

The Parking Space Finder is designed as a standalone web application that processes live video feeds from parking lots using OpenCV to detect available parking spaces. The system is primarily intended for drivers and passengers seeking parking spots, as well as parking lot administrators who manage parking availability. From the perspective of a driver, the system allows for real-time parking space updates and easy navigation to the nearest available spot. Passengers can use the system to assist drivers, while parking lot administrators use the administrative interface to manage parking space data and monitor parking lot usage. System administrators are responsible for maintaining the overall system, including software updates and data management.

B. Product Functions

The main functions of the Parking Space Finder include:

1. **Real-Time Parking Space Detection:** The system processes video feeds to detect and update parking space availability in real time.

2. **User Interface:** Users can view available parking spaces on a map or in a list view.
3. **Navigation Assistance:** The system provides directions to available parking spaces.
4. **Administrative Management:** Administrators can update parking space information, manage user accounts, and generate reports.

C. User Characteristics

- Drivers: Can view available parking spaces and receive navigation assistance but cannot modify system settings.
- Passengers: Can view parking space availability to assist the driver.
- Parking Lot Administrators: Have access to tools for managing parking space data and monitoring parking lot usage.
- System Administrators: Have full access to all system functions, including user management, system maintenance, and data processing.

D. General Constraints

- Software: The system is developed using web technologies (HTML, CSS, JavaScript) and OpenCV for image processing.
- Hardware: The system requires servers capable of processing video feeds in real-time and handling multiple user requests.
- Network: A reliable internet connection is necessary for both the server and end users.
- Budget: The project is limited by a budget that covers development, hosting, and maintenance costs.
- Time: The project follows a timeline starting from August 1, 2024, and ending on November 10, 2024.
- Space: The system requires both physical space for server setup and logical space for data storage and processing.
-

E. Assumptions

- The system will be accessed exclusively through a web-based interface; no mobile application will be developed.
- Users are expected to access the system using modern browsers like Chrome or Firefox.
- The system assumes the availability of high-quality video feeds from parking lots.

3. Functional Requirements

A. Business Functions

The software under development will include the following business functions:

1. **Parking Space Detection:** The system must continuously process video feeds to detect available parking spaces.
2. **User Interface:** The system must display available parking spaces in a user-friendly manner.
3. **Administrative Tools:** Administrators must have access to tools for managing parking spaces, updating system settings, and generating reports.
4. **Navigation Assistance:** The system should provide users with directions to the nearest available parking space.

B. Detailed Scenario

1. **Operational Requirements:** The system must operate 24/7, providing real-time updates on parking space availability. It must be capable of handling high traffic, especially during peak hours.

2. Detailed Scenarios:

- **Scenario 1:** A driver accesses the website, views available parking spaces, and receives navigation directions to a chosen spot.
- **Scenario 2:** A parking lot administrator updates the status of a parking spot after maintenance work.
- **Scenario 3:** The system administrator performs routine maintenance, ensuring minimal downtime and optimal performance.

<h2>4. Non-Functional Requirements</h2>
--

A. Response Time

The system should have a fast response time, with the user interface loading within 3 seconds and parking space updates occurring within 1 second of detection.

B. Security Requirements

- **User Authentication:** All users must authenticate using secure login credentials.
- **Data Encryption:** All sensitive data must be encrypted in transit and at rest.
- **Access Control:** Access to the administrative functions must be restricted to authorized personnel only.

C. Safety Requirements

- **Data Integrity:** The system must ensure that parking space data is accurate and not corrupted due to system failures.

- **Access Control:** Only authorized users should have access to sensitive system functionalities.

D. Reliability: • Availability: The system should have an uptime of at

least 99.9%.

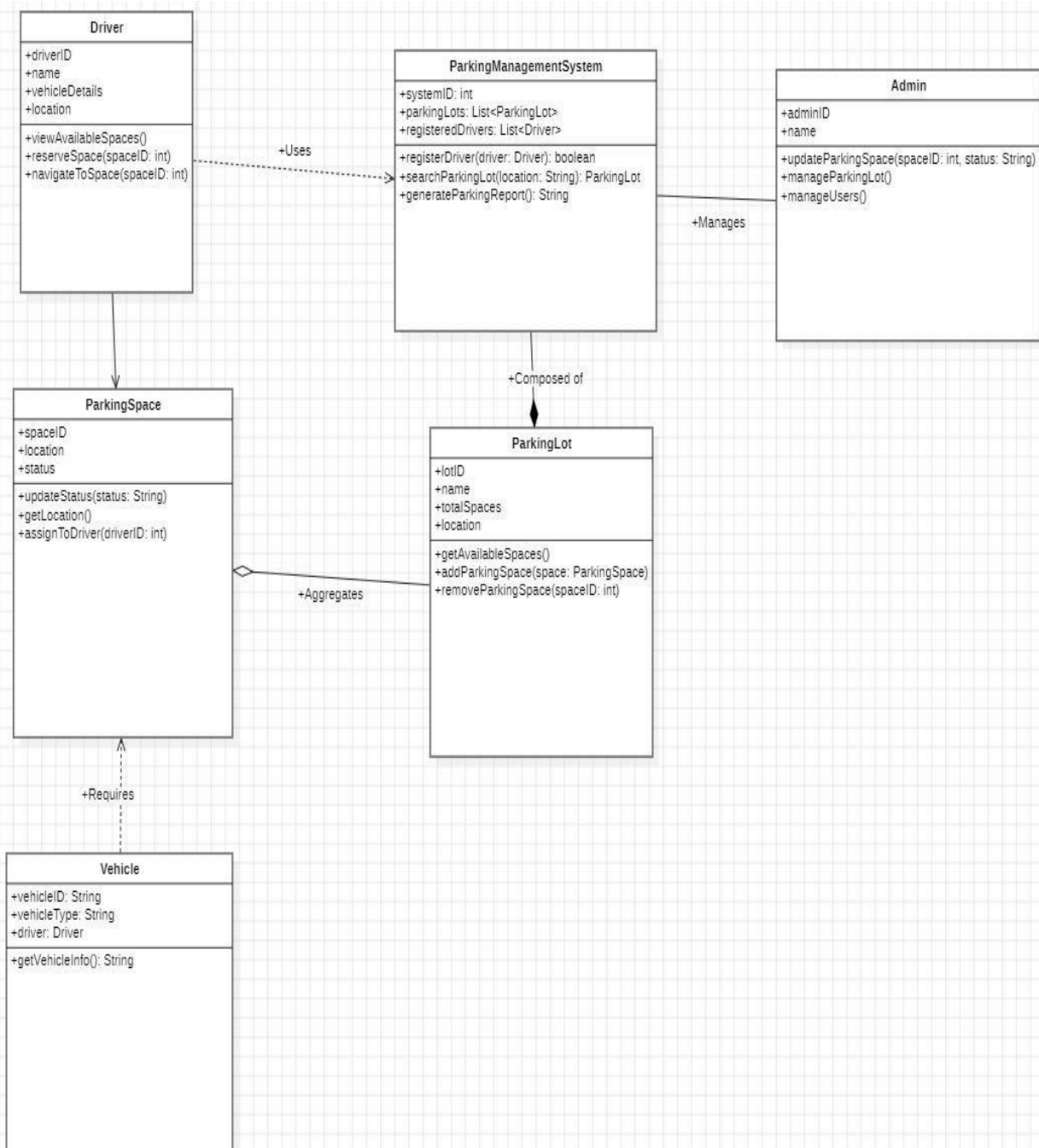
- Maintainability: The system should be designed for easy maintenance, allowing updates with minimal downtime.
- Reliability Testing: Regular testing must be conducted to ensure the system's reliability.
- Failure Rate: The system should have a failure rate of less than 0.1% during normal operations.

Conclusion

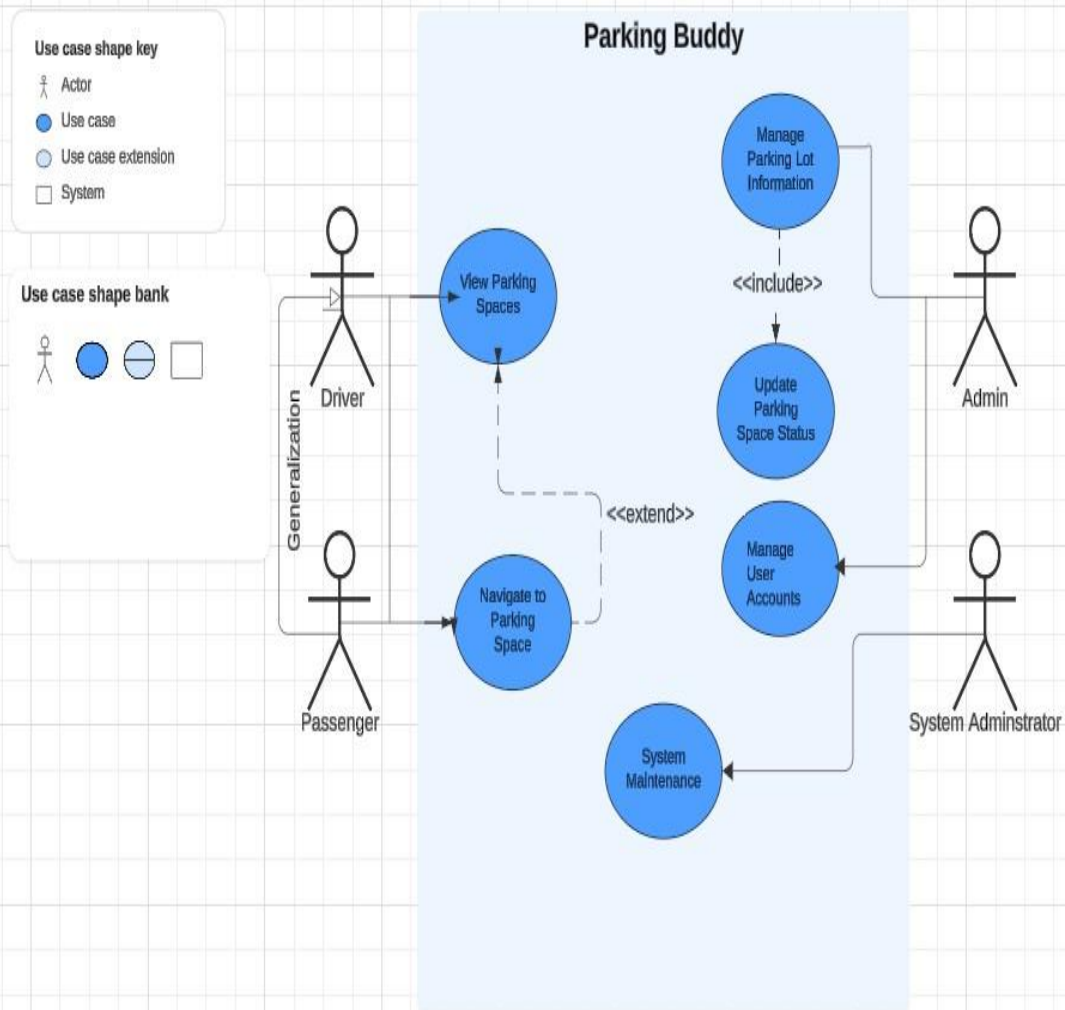
This Software Requirements Specification (SRS) document provides a comprehensive outline of the requirements for the Parking Space Finder project. It includes detailed descriptions of the system's functional requirements, non-functional requirements, system architecture, and use case model. The document serves as a crucial guide for the development team to ensure that the project meets its goals and delivers a reliable, user-friendly solution for parking space management.

Software Design:Essential UML Diagrams

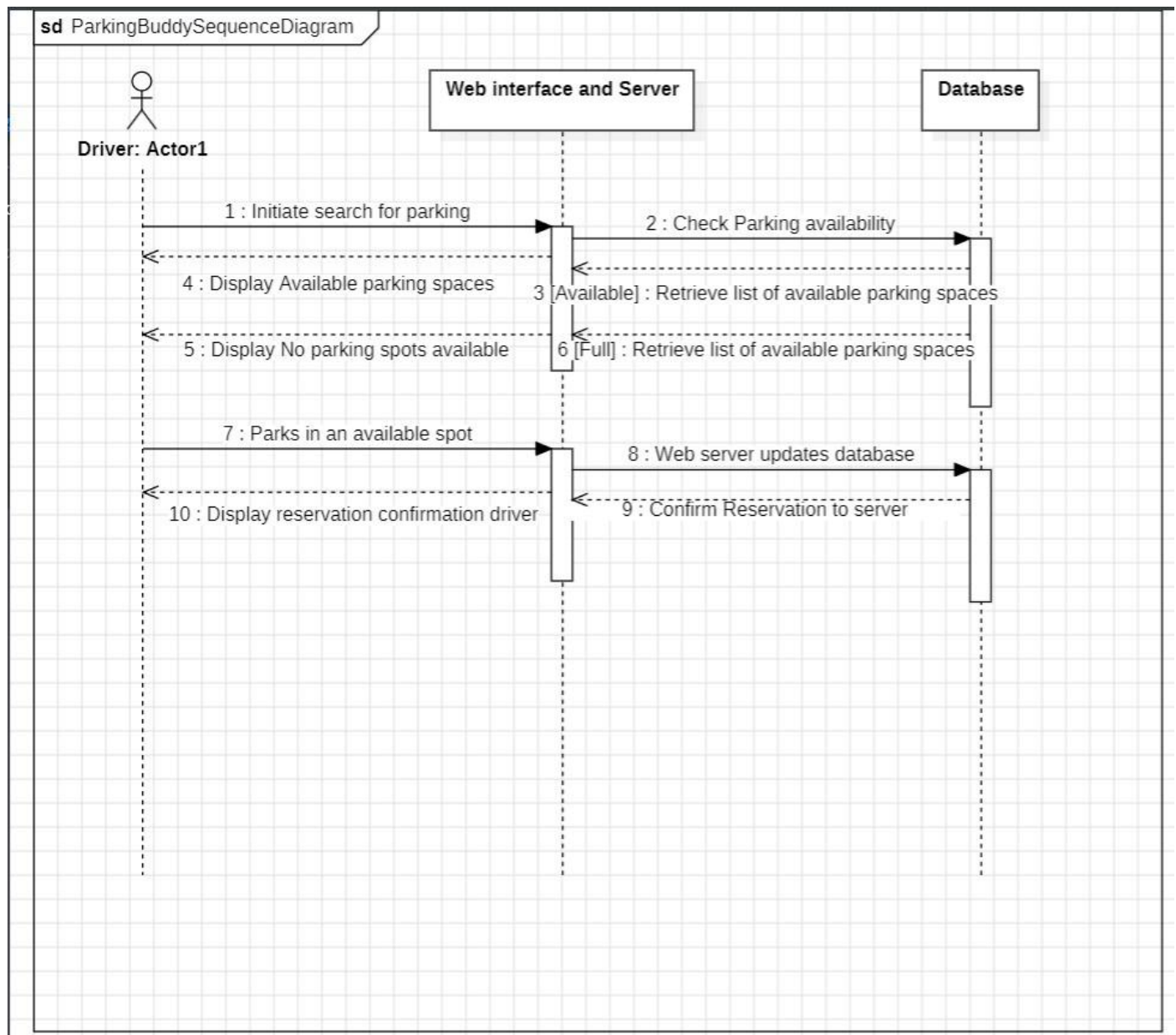
Class Diagram:



Use case diagram:

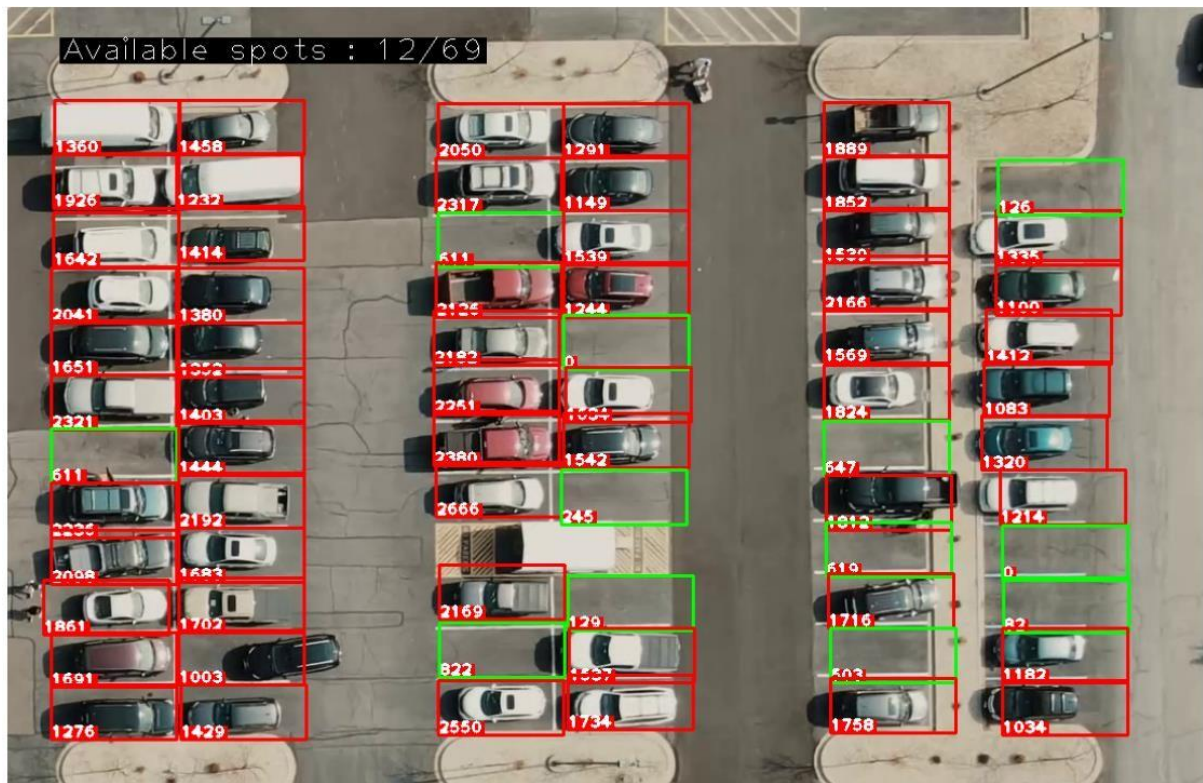


Sequence Diagram:



Now, the source codes for the above design:

Code Base for the Model:



File Structure

	CarParkPos	11-09-2024 17:26	File	1 KB
	carParkVideo	27-07-2024 18:25	MP4 File	10,360 KB
	main	10-11-2024 22:13	Python Source File	3 KB
	parkingimg	28-07-2024 00:02	PNG File	988 KB
	ParkingSpacePicker	28-07-2024 22:10	Python Source File	3 KB

Main.py

```
import cv2
import pickle
import cvzone
import matplotlib.pyplot as plt
import numpy as np
width, height = 115, 50 # width and height of a single parking
space
# Video feed
```



```

cap =
cv2.VideoCapture('C:/Users/baber/Desktop/VIT/ParkingBuddy/carParkVideo.mp4')
with open('C:/Users/baber/Desktop/VIT/ParkingBuddy/CarParkPos', 'rb') as f:
    posList = pickle.load(f)
    def checkParkingSpace(imgPro,
img):
        spaceCounter = 0
    for pos in poslist:
        x, y = pos          imgCrop =
imgPro[y:y+height, x:x+width]

        # counting the number of pixels in the rectangle (if more car
is present)          count = cv2.countNonZero(imgCrop)
cvzone.putTextRect(img, str(count), (x, y+height-2), scale=1,
thickness=2, offset=0, colorT=(255, 255, 255), colorR=(0, 0, 255))
        if count < 900:
color = (0, 255, 0)
thickness = 2
spaceCounter += 1
        else:
            color = (0, 0, 255)
thickness = 2
            cv2.rectangle(img, pos, (pos[0] + width, pos[1] + height),
color, thickness)
            cvzone.putTextRect(img, f'Available spots
:
{spaceCounter}/{len(posList)}', (50, 50), scale=2, thickness=1, offset=2,
colorR=(0, 0, 0))

#Main function
while
True:
    if cap.get(cv2.CAP_PROP_POS_FRAMES) == cap.get(cv2.CAP_PROP_FRAME_COUNT):
        cap.set(cv2.CAP_PROP_POS_FRAMES, 0)
    success, img = cap.read()

    # converting image to grayscale
    imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)          imgBlur =
cv2.GaussianBlur(imgGray, (3, 3), 1) # adding blur to the image
    # converting to binary image using adaptive thresholding
imgThreshold = cv2.adaptiveThreshold(imgBlur, 255,
cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY_INV, 25, 16)
    # to remove noise using median blur
    imgMedian = cv2.medianBlur(imgThreshold, 5)

```



```

        # using dilation to differentiate between empty space and a car making
        the boundaries more thicker        kernel = np.ones((3, 3), np.uint8)
imgDilate = cv2.dilate(imgMedian, kernel, iterations=1)
        checkParkingSpace(imgDilate,
img)

        # Display the frame using Matplotlib
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
plt.imshow(img_rgb)        plt.axis('off')
plt.pause(0.001)        plt.clf()

plt.show()

```

ParkingSpacePicker.py

```

import cv2 import pickle
import matplotlib.pyplot as
plt import numpy as np import
os

width, height = 115, 50 # width and height of a single parking
space
# Path to the file
file_path = 'CarParkPos'

# Load previously saved parking positions if available
if os.path.exists(file_path):
    with open(file_path, 'rb') as f:
        posList = pickle.load(f)
else:
    posList = []

# Mouse click event handler def
mouseClick(event, x, y, flags, param):
    if event == cv2.EVENT_LBUTTONDOWN:        #left click to add parking space
posList.append((x, y))
        print(f"Added position: ({x}, {y})")        if event ==
cv2.EVENT_RBUTTONDOWN:        #right click to remove parking space
for i, pos in enumerate(posList):
    x1, y1 = pos        if x1 < x < x1 + width
and y1 < y < y1 + height:
        posList.pop(i)
print(f"Removed position: ({x1}, {y1})")
        # Save the updated positions

```



```

        with open(file_path, 'wb') as f:
            pickle.dump(posList, f)
    print(f"Positions saved to {file_path}")
    # Function to handle matplotlib events
    def onclick(event):
        if event.button == 1: # Left click
            mouseClick(cv2.EVENT_LBUTTONDOWN, int(event.xdata), int(event.ydata),
None, None)
        elif event.button == 3: # Right click
            mouseClick(cv2.EVENT_RBUTTONDOWN, int(event.xdata), int(event.ydata),
None, None)
        update_display()

    # Function to update the display
    def update_display(): # Read the image
        img = cv2.imread('C:/Users/baber/Desktop/VIT/ParkingBuddy/parkingimg.png')
        # Draw rectangles on the image based on the positions in posList
        for pos in poslist:
            cv2.rectangle(img, pos, (pos[0] + width, pos[1] + height), (255, 0,
0), 2)

        # Convert BGR (OpenCV format) to RGB (matplotlib format)
        img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

        # Clear the current plot
        plt.clf()

        # Display the image using matplotlib
        plt.imshow(img_rgb)
        plt.title("Image")
        plt.axis('off') # Hide the axis
        plt.draw()

    # Initial display
    fig, ax = plt.subplots()
    fig.canvas.mpl_connect('button_press_event',
onclick)
    update_display()
    plt.show()

```


Software Testing

Test Planning

•**Testing Strategy:** A combination of manual and automated testing approaches was adopted. Manual testing focused on UI and functional aspects, while automated testing assessed backend functions and real-time performance.

•**Test Environment Setup:** Testing was conducted on a range of devices (desktop and mobile) using different browsers (Chrome, Firefox, Edge) to evaluate compatibility.

Scope of Testing

The testing encompasses the following areas:

- **Functionality Testing:** Ensures each feature performs as expected.
- **Security Testing:** Identifies vulnerabilities and data protection mechanisms.
- **Performance Testing:** Measures application speed, responsiveness, and reliability.
- **Usability Testing:** Tests the application's user-friendliness.

Objectives

The primary objectives of this testing report are to:

- Verify the accuracy of parking detection.
- Validate the integrity and security of user data.
- Ensure responsiveness and performance under different usage loads.
- Confirm that the UI is accessible and easy to navigate.

Types of Testing Conducted

- **Functional Testing:**

•**Map Loading:** Tested the Google Maps integration to ensure that location coordinates result in the accurate display of the corresponding map.

•**Parking Spot Detection:** Verified the OpenCV model's ability to detect parking spaces based on uploaded or live feed images. Adjusted model sensitivity to balance detection accuracy.

- **Integration Testing:**

•**OpenCV & Google Maps Integration:** Tested interactions between the OpenCV detection module and the Google Maps display to ensure seamless data flow and real-time updates.

•**Location Search & Map Update:** Ensured that selecting or entering a new location automatically updates the map display.

- **User Interface (UI) Testing:**

•**Map Interactivity:** Verified that users could pan, zoom, and switch between locations smoothly.

•**Marker Display:** Tested the accuracy and visibility of parking markers displayed on the map.

- **Usability Testing:**

•Collected feedback from users who tested the app’s functionality to assess ease of use, particularly for location input, parking spot visibility, and overall navigation.

- **Compatibility Testing:**

•**Cross-Browser:** Tested the application on Chrome, Firefox, and Edge for consistent behavior and display.

•**Device Compatibility:** Assessed the UI on both desktop and mobile browsers to ensure responsiveness and usability across devices.

- **Performance Testing:**

•**Load Testing:** Simulated multiple users accessing the application simultaneously to evaluate how well the map loads and detects parking spaces under load.

•**Response Time:** Measured the time taken for map loading and parking spot detection, with a goal of maintaining a load time under 3 seconds.

•**Stress Testing:** Assessed how the application behaves when handling an unusually high number of requests, especially in urban areas with a high density of parking spots.

- **Security Testing:**

•**Input Validation:** Ensured that location inputs and other fields handle invalid or unexpected inputs without causing system crashes.

Functional Testing

Black Box Testing

Test Case	Description	Steps	Expected Result	Status ID	Actual Result
-----------	-------------	-------	-----------------	-----------	---------------

1. Fill user

TC_BB_01 User Registration details in form Data is stored and As Pass confirmation displayed expected

2. Submit

Parking space

TC_BB_02 Check Parking
Button button

Click on
expected

availability video

As

Pass Availability

appears

The image shows a registration form with a green background and yellow input fields. The fields are labeled 'Name:*', 'Username:*', 'Email Id:*', and 'Password:*'. The 'Name' field contains 'jy7g', 'Username' contains 'dfb', and 'Email Id' contains 'awrr@gmail.com'. The 'Password' field contains three dots and has an eye icon for toggling visibility. Below the fields is a yellow 'Register' button.

White Box Testing

This testing checks logical flows, algorithms, and functions within the application's codebase.

Test Case ID	Functionality	Expected Outcome	Path Tested	Actual Result Status
--------------	---------------	------------------	-------------	----------------------

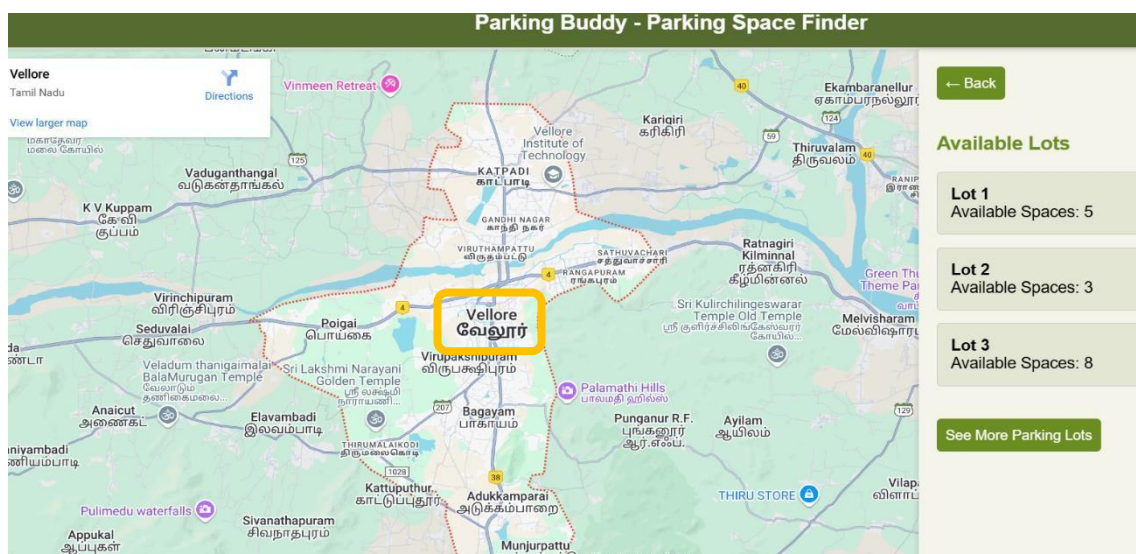
TC_WB_01	Parking Counter Accuracy	Returns accurate count of available spaces	Test OpenCV algorithm accuracy	Works as expected	Pass
TC_WB_02	Database Connectivity errors	Database user info without errors	saves Verify SQL query for data insertion	Not implemented yet	On hold

Non-Functional Testing

Usability Testing

Usability Testing assesses the application's accessibility, intuitiveness, and overall user experience.

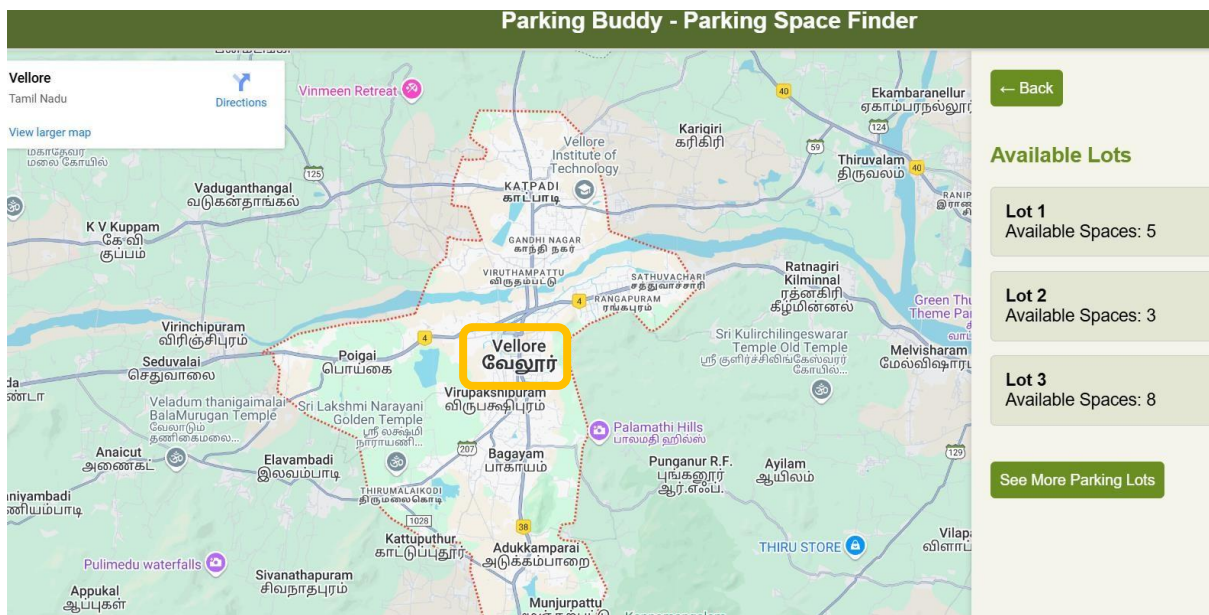
Test Case	Aspect	Steps	Expected Result	Status ID	Actual Result
TC_US_01	Navigation Ease	1. Click 'Check Parking' 2. Click 'Map'	Navigation is seamless without errors	As expected	Pass
TC_US_02	Form accessible and easy to	1. Enter data 2. Submit form	Form fields are accessible and easy to fill	As expected	Pass



Integration Testing

Integration Testing ensures each module works cohesively with others.

Test Case ID	Modules	Interaction	Expected Outcome	Actual Outcome	Status
TC_INT_01	Frontend (Flask) - OpenCV	UI triggers video feed	Video loads without errors	As expected	Pass
TC_INT_02	Database - Location Module	Saves location and centers on saved location	Map loads and As expected	Pass	Pass displays on map



Performance Testing

Performance Testing includes response times, system stability under load, and resource usage.

Test Case	Load	Expected	Actual	Status
Scenario				
ID	Condition	Performance	Performance	
TC_PER_01	Multi-User concurrent users	Simulate 50 time per action	< 2 seconds load pending]	[Testing Load On hold
TC_PER_02	Video Processing Speed	Load high-resolution video delay > 1 sec/frame	Processes without delay pending]	[Testing On hold

Security Testing

Security Testing examines data handling, encryption, and user protection.

Test Case ID	Vulnerability	Description	Mitigation Strategy	Status
TC_SEC_01	SQL fields	Input "1' OR '1'='1" in login prevent SQL injection	Sanitize inputs to Injection	On hold
TC_SEC_02	XSS Attack	Input <script>alert(1)</script> characters in inputs	Escape special in form	On hold