

PROJECT REPORT

AI ENABLED CAR PARKING USING OPENCV

Team ID : NM2023TMID00106

GUIDED BY

FACULTY MENTOR : Dr. S Saravanakumar

INDUSTRY MENTOR : Hari , Sai Manish

TEAM MEMBERS

1. Srinivas M R
2. Jayachandran R
3. Karthickraja J
4. Niranjanan M

INDEX

1. INTRODUCTION

- 1.1 Project Overview
- 1.2 Purpose

2. IDEATION & PROPOSED SOLUTION

- 2.1 Problem Statement
- 2.2 Empathy Map Canvas
- 2.3 Brainstorming & Idea Prioritization
- 2.4 Proposed Solution

3. REQUIREMENT ANALYSIS

- 3.1 Functional requirement
- 3.2 Non-Functional requirements

4. PROJECT DESIGN

- 4.1 Data Flow Diagrams
- 4.2 Solution & Technical Architecture
- 4.3 User Stories

5. CODING & SOLUTIONING (Explain the features added in the project along with code)

- 5.1 Feature 1
- 5.2 Feature 2
- 5.3 Database Schema (if Applicable)

6. RESULTS

- 6.1 Performance Metrics

7. ADVANTAGES & DISADVANTAGES

8. CONCLUSION

9. FUTURE SCOPE

10. APPENDIX

Source Code

GitHub & Project Video Demo Link

INTRODUCTION

The advancement of Artificial Intelligence (AI) has revolutionized various industries, and one of its remarkable application is in the domain of car parking systems. In this project we aim to develop a web-application called “AI Enabled Car Parking System” utilizing OpenCV. By harnessing the power of AI and computer vision techniques .Our project aims to provide an efficient and automated solution for managing car parking spaces.

1.1 Project Overview

This project focusses on designing and implementing a web-application that incorporates AI algorithm and OpenCV. The application will utilize computer vision techniques to monitor and analyze parking spaces in real-time, allowing users to easily find available parking spots and optimize the overall parking experience.

The web-application will provide a user-friendly interface where drivers can access real-time parking information, view parking lot occupancy status, and to navigate to the nearest available parking spot. Additionally, the system will facilitate parking management by providing an administrative interface for monitoring and managing the parking spaces and ensures efficient utilization of parking resources.

Throughout this project, we aims to enhance the efficiency of car parking spaces and ensures efficient utilization of parking resources

1.2 Purpose

The purpose of the project is to develop a web-application that utilizes AI and computer vision techniques to enhance the efficiency and convenience of car parking system. The specific goals of the project are as follows

- **Improve User Experience**
- **Optimize Parking Resource Utilization**
- **Promote Sustainability**

The ultimate goal of our project is to revolutionize the traditional parking system by offering an automated, intelligent and convenient solution for the drivers and parking lot owners.




IDEATION & PROPOSED SOLUTION



2.1 Problem Statement

AI Enabled car parking is a modern technology that combines the power of AI(Artificial Intelligence) with the traditional car parking system by utilizing a popular open source library named “Open CV”. The AI Enabled car parking relies on advanced computer vision algorithms helps to analyze and detect the availability of the parking lots in real-time. With this Web Application one can know the status of the availability of the parking lots

remotely. This will drastically reduce the congestion and the wait time of the end user.

Table

Problem statements	I am (customer)	I'm trying to	But	Because	Which makes me feel
Increase in vehicle density makes the parking space congested	a Driver	Find the parking slots	I'm unable to find whether any parking slots available	The vehicle count increased drastically which makes parking area crowded	Helpless, increases pressure and anger
					
Unable to find parking lots in new places	An explorer	Explore different place in world and cultures	I'm unable to park my vehicle due to lack of knowledge about where the parking lots were	I'm not aware about where the parking lots were	Uncomfortable
					
Struggling to find parking lots in popular shows and concerts	A concert Lover	To participate in all live concerts	I'm unable to find where parking space since limited slots were available for events	Parking spaces were in high demand	Disappointed
					

Increases accidents due to the Increase in vehicles parked in no parking since people were unaware of the free space in parking lots	A Traffic police	Ensure the civilians safety	Accidents occurs	The civilians parks their vehicle in no parking which leads the path congested	Depressed and felt guilt
					
Decreases revenue	A parking lot owner	Earn	My business falls down	The parking allotment system is inefficient	Hopeless since I lost revenue
					

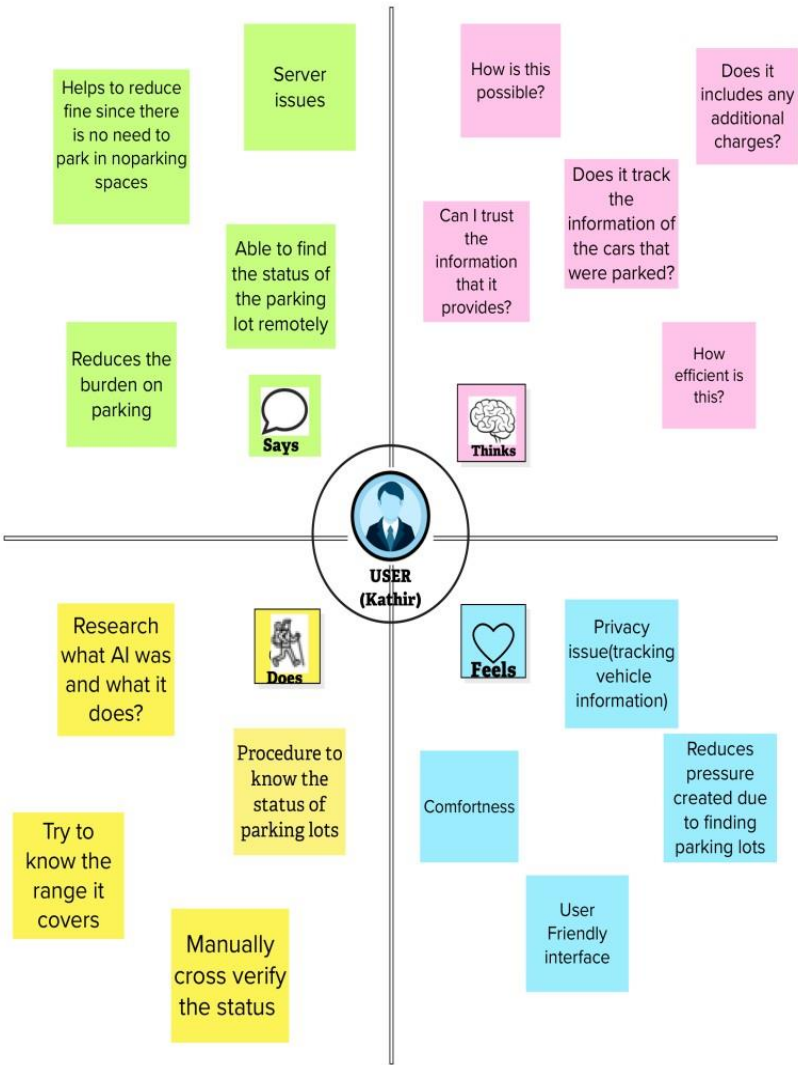
2.2 Empathy Map Canvas



Empathy map

It is a useful tool to helps teams better understand their users. Creating an effective solution requires understanding the true problem and the person who is experiencing it. The exercise of creating the map helps participants consider things from the user's perspective along with his or her goals and challenges.

Share template feedback



Need some inspiration?
See a finished version of this template to kickstart your work.
[Open example](#)



2.3 Brainstorming & Idea Prioritization



Before you collaborate

A little bit of preparation goes a long way with this session. Here's what you need to do to get going.

 10 minutes

A

Team gathering

Dear Team,I hope this message finds you well.I am excited to invite you to the Brainstorming session scheduled at 29 April 2023

Team member

Srinivas M R
Jayachandran R
Karthickraja J
Niranjanan M

B

Goal

We were focussing to optimize the car parking system in order to reduce congestion and to improve the efficiency of the traditional car parking system.

C

Learn how to use the facilitation tools

Use the Facilitation Superpowers to run a happy and productive session.

Open article



1

PROBLEM STATEMENT

PROBLEM

Car Parking is a common problem in cities and urban areas where there is limited space for vehicles. As the number of vehicles on the road increases, finding a parking spot becomes a challenging task for a drivers. Inefficient car parking systems not only cause inconvenience to drivers but also result in traffic congestion.



Key rules of brainstorming

To run an smooth and productive session

-  Stay in topic.  Encourage wild ideas.
-  Defer judgment.  Listen to others.
-  Go for volume.  If possible, be visual.

BRAINSTORM

Srinivas M R

Visual processing	Webpage based approach	
	Make use of Open cv	

Jayachandran R

Counting vehicles entering and leaving	Video Processing	
	acquiring Region of interest	need Real time monitoring

karthickraja J

Computer vision based approach		
	Make use of network to distribute across world	Integrating AI and Computer vision

Niranjanan M

	Display processed video from cctv on webpage	Open CV related approach
	AI Algorithm to detect spaces	

GROUPING IDEAS

Utilizing open source library Open CV

Make use of
Open cv

Computer
vision
based
approach

Open CV
related
approach

Integrating
AI and
Computer
vision

Using computer vision concepts to process raw footage

Visual
processing

Video
Processing

Using AI algorithms parking spaces were counted

AI Algorithm
to detect
spaces

AI Algorithm
to detect
spaces

Deploying it as web application

Make use of
network to
distribute
across world

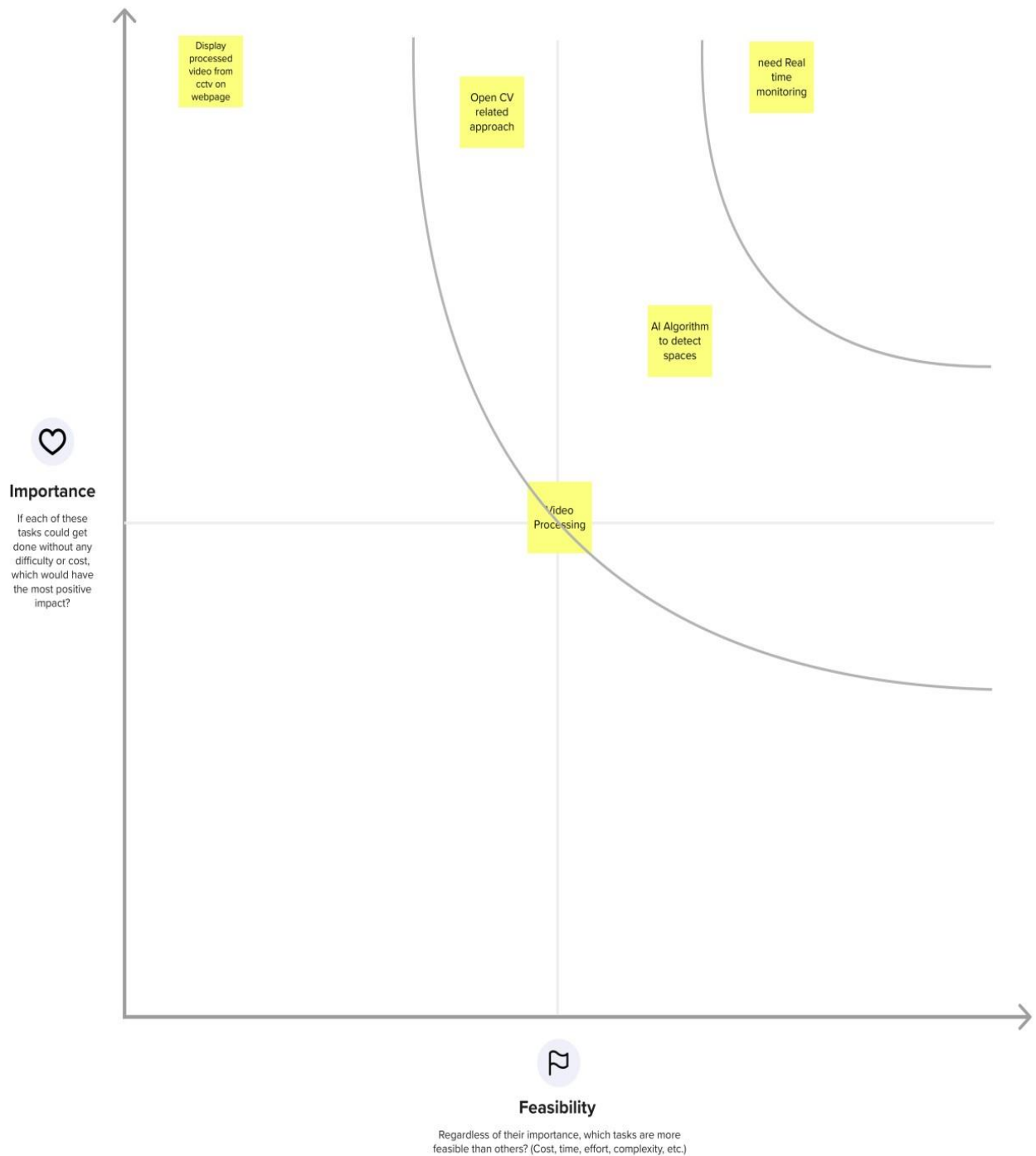
Display
processed
video from
cctv on
webpage

Webpage
based
approach

Acquiring footage from CCTVs placed at the Parking lots

need Real
time
monitoring

IDEA PRIORITIZATION



2.4 Proposed Solution

S.No.	Parameters	Description
1.	Problem Statement(Problem to be solved)	Car parking is a common problem in cities and urban areas where there is limited space for vehicles. As the number of vehicles on the road increases, finding a parking spot becomes a challenging task for a driver. Inefficient car parking systems not only cause inconvenience to drivers but also result in traffic congestion.
2.	Idea/Solution description	Smart parking system with cameras to monitor parking spots and provide real-time and to provide real-time information about the availability of the parking spaces. The obtained information is shared to the user through a web application.
3.	Novelty/Uniqueness	Enables users to know the status of the availability of the parking slots remotely. This system can reduce traffic congestion since there is no need to park the vehicles in “no parking”. This system can reduce fuel combustion and emission from vehicles while circling around to find parking spaces. This helps the parking lot owner to manage the parking spaces and make them adjust the price based on demand.
4.	Social impact/Customer Satisfaction	Reduces stress of users in finding the available parking spaces in parking lots. Reduces traffic congestion. Reduces environmental pollution.

5.	Business Model(Revenue Model)	Investment in necessary technologies like cameras, raspberry pi and a webserver. Need to make partnership with the parking lot owners/managers.Deploying the web application as a pay per service model.
6.	Scalability of the Solution	Highly scalable as it can be deployed in multiple regions. Can be integrated with other smart city technologies like smart traffic management system, public transportation and mobile applications.

REQUIREMENT ANALYSIS

3.1 Functional Requirements

Following are the functional requirements of the proposed solution.

FR No.	Functional Requirement (Epic)	Sub Requirement (Story / Sub-Task)
FR-1	User Registration	Registration through Gmail
FR-2	User Login	Login via Username or Gmail and Password
FR-3	Parking space Availability	Display real-time parking availability
FR-4	Parking space occupancy	Highlight occupied spots on the parking map
FR-5	Need to update the real-time availability	Update parking availability in the database
FR-6	Able to select a parking spot	Display parking spot selection interface
FR-7	Able to reserve a parking spot	Enable reservation of selected parking spot

FR-8	Able to cancel parking reservation	Provide an option for users to cancel their parking reservation
------	------------------------------------	---

3.2 Non-Functional Requirements

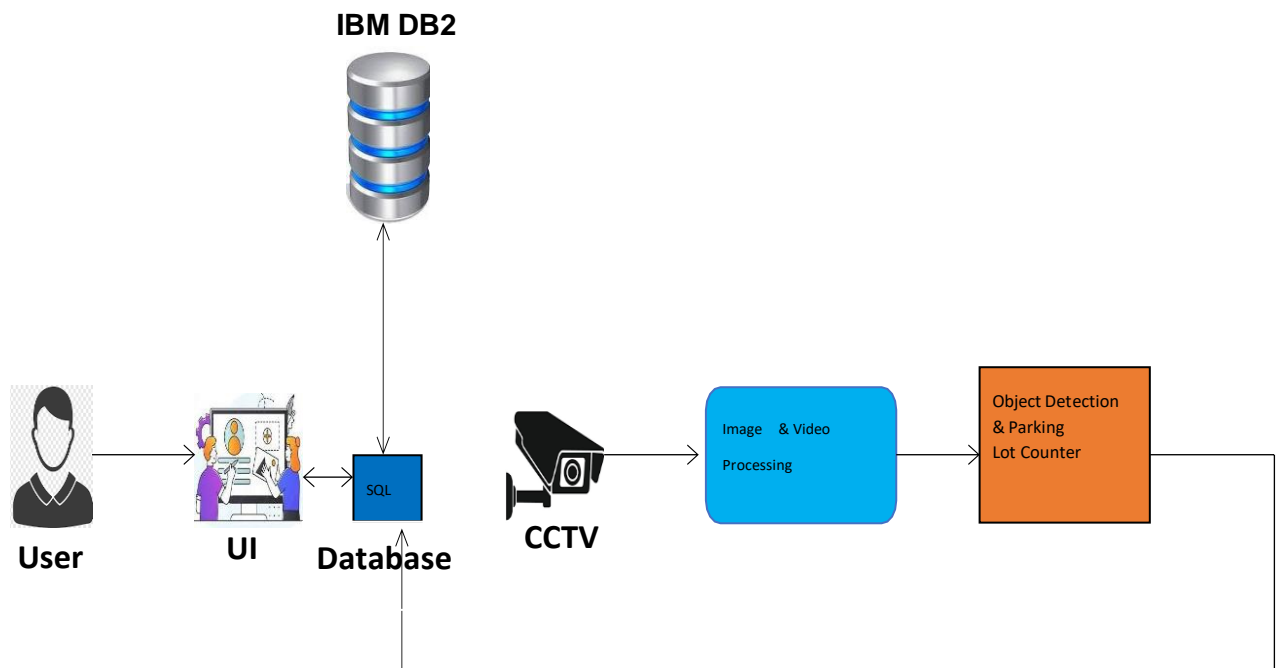
Following are the non-functional requirements of the proposed solution.

FR No.	Non-Functional Requirement	Description
NFR-1	Usability	The application should have a user-friendly interface that is easy to navigate and understand. It should provide clear instructions and intuitive controls to enhance user experience.
NFR-2	Security	The application should ensure the security of user data and prevent unauthorized access. It should implement authentication mechanisms, encrypt sensitive information, and follow best practices for secure coding.
NFR-3	Reliability	The system should be highly reliable and perform consistently without failures or crashes. It should handle errors gracefully and recover from any unexpected issues to ensure uninterrupted service.
NFR-4	Performance	The application should have fast response times and low latency to provide a seamless user experience. It should optimize algorithms and code execution to efficiently process car detection, parking updates, and other functionalities.
NFR-5	Availability	The system should be always available and accessible to users, minimizing downtime and disruptions. It should have robust infrastructure, fault-tolerant design, and appropriate backup mechanisms to ensure high availability.

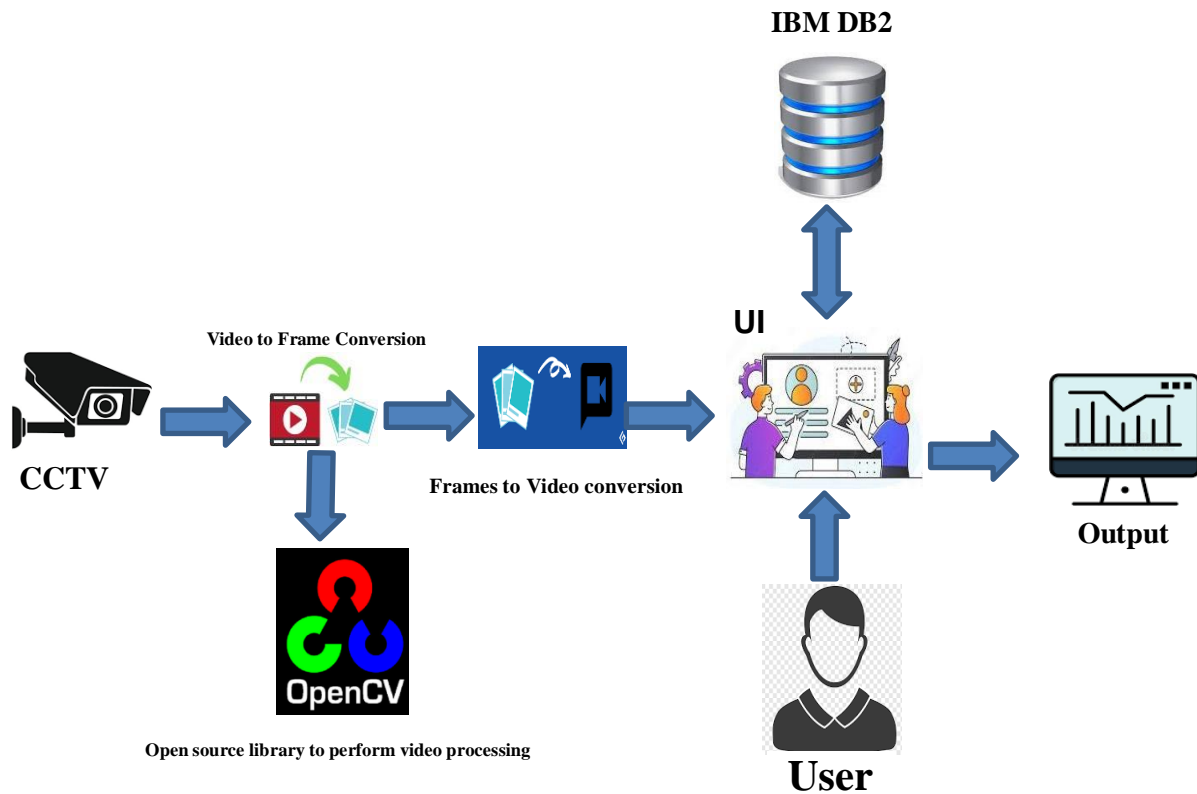
NFR-6	Scalability	The application should be designed to handle increasing numbers of users, parking lots, and concurrent requests. It should scale horizontally or vertically, allowing for future expansion and accommodating higher loads efficiently.
-------	--------------------	--

PROJECT DESIGN

4.1 Data Flow Diagram



4.2 Solution & Technical Architecture



Acquiring Data:

The first step in AI Enabled Car Parking system is to acquire the data from the sensors. The sensors can be a combination of cameras, ultrasonic or IR sensors. Here the data of parking lot can be acquired via “CCTV” cameras placed over the parking lot.

Data Preprocessing:

The data acquired from the CCTV camera is sent to the end device which performs some processing. The end device may be a computer or a raspberry pi which process the video using some video processing and machine learning techniques to acquire the information from the parking lot.

The tool that we used here for processing the input is an open-source library named “Open CV” which acquires the frames from the video.

Data Processing:

The end device processes the information acquired from the CCTV camera by making use of “Open cv”. Using Open CV, we acquire the region that we were interested in. Then using “Image thresholding” technique and

some further calculation the information about the availability of the parking slot was obtained.

Data Postprocessing:

The data obtained on processing is further postprocessed to convert the frames to video.

User Interface:

A user-friendly interface is needed to allow the user to find the parking spots in the lot. The user interface can be a mobile application or a web application that displays the status about the availability of the parking spots in the lot.

Database:

The web application that we built is a login authentication-based system to enhance security. Here Database is used to store the user information. The Database that we used here is "IBM DB2". The web application authenticates the user via user id and password.

Report:

On successful authentication the status of the availability of the parking spots in the lot was displayed.

4.3 User Stories

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Team Member
Customer (Mobile user)	Registration	USN-1	As a user, I can register for the application by entering my email, first-name, last-name, password, and confirming my password.	I can access my account / dashboard	High	Jayachandran & Karthickraja
		USN-2	As a user, I will receive confirmation email once I have registered for the application	I can receive confirmation email & click confirm	High	Jayachandran
		USN-3	As a user, I can register for the application through Gmail	I can register & access the dashboard with Facebook Login	Low	Jayachandran
	Login	USN-4	As a user, I can log into the application by entering email & password	-	High	Jayachandran, Niranjana, Karthickraja, Srinivas
Administrator	Manage parking database	USN-6	As an administrator, I want to add a new parking spot	The system allows me to enter the spot details: location, capacity, and type.	High	Niranjana

		USN-7	As an administrator, I want to edit a parking spot	The system allows me to modify the spot details: location, capacity, and type.	Low	Srinivas
	Generate parking usage reports	USN-8	As an administrator, I want to generate reports on parking spot usage	The system provides an option to select the desired date range. The system generates a usage report, including spot occupancy and revenue.	Medium	Karthickraja
	Manage registered vehicles	USN-9	As an administrator, I want to manage the list of registered users	The system allows me to add, edit, and remove registered users	High	Jayachandran
User	View available parking spots	USN-10	As a user, I want to view the number of available parking spots in a specific lot	The system displays the count of available parking spots in the selected lot	High	Jayachandran & Niranjana
	Get parking lot occupancy updates	USN-11	As a user, I want to receive real-time updates on the occupancy of a specific parking lot	The system provides live updates on the occupancy status of the selected parking lot	High	Karthickraja & Niranjana
	Reserve a parking spot	USN-12	As a user, I want to reserve a parking spot in a specific lot	The system displays available spots in the selected lot. I can select a spot and provide my vehicle details.	Medium	Jayachandran & karthickraja
	View parking lot information	USN-13	As a user, I want to view information about a specific parking lot	The system displays details of the selected parking lot, including location, total capacity, and current occupancy	Medium	Srinivas
Maintenance	Report parking lot issues	USN-14	As a maintenance team member, I want to report issues or maintenance requirements for a parking lot	The system allows me to submit a report including the parking lot details and a description of the issue	Medium	Srinivas & Niranjana
	View and manage parking lot maintenance requests	USN-15	As a maintenance team member, I want to view and manage maintenance requests for parking lots	The system displays a list of maintenance requests for parking lots	Medium	Karthickraja & Jayachandran

CODING AND SOLUTIONING

5.1 Feature 1

Real-time Parking Availability

The web application can utilize AI algorithms to analyze data from sensors or cameras placed in parking lots. This information can be displayed in real-time, indicating the availability of parking spaces to users.

1. Set up the Web Application Framework

Here we had used a flask as a web-application framework. The below code shows the initialization of flask framework

Step 1: Install Flask

Install Flask using package manager pip, which bundled inside python

Command – “pip install flask” for python 2.7 and “pip3 install flask” for python3

Step 2: Create a python file named app.py

```
from flask import Flask
app = Flask(__name__)
@app.route('/')
def hello():
    return 'Hello, World!'
if __name__ == '__main__':
    app.run()
```

1. User Authentication and Sign-In Page:

Sign-in and sign-up page with a form that includes the field email and password for the sign-in page and first name, last name, email, password, confirm password.

HTML CODE

```
1. <!DOCTYPE html>
2. <html lang="en"><head>
3.     <!-- Basic -->
4.     <meta charset="utf-8">
5.     <meta http-equiv="X-UA-Compatible" content="IE=edge">
6.
7.     <!-- Mobile Metas -->
8.     <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-
9. fit=no" />
10.    <meta http-equiv="content-type" content="text/html; charset=UTF-8">
11.    <title>signin</title>
12.
13.    <link rel="stylesheet" href="../static/css/style1.css">
14.    <!--we had linked our css file----->
15.</head>
16.<body>
17.    <div class="full-page">
18.        <div class="navbar">
19.            <div>
20.                <a href="{{url_for('index')}}" style="margin-left: 100px;">AI
Enabled Car Parking</a>
21.            </div>
22.            <nav>
```

```

23.         <ul id="MenuItems">
24.             <li style="margin-right: 10px;"><a
href="{{url_for('index')}}">Home</a></li>
25.             <li style="margin-right: 10px;"><a
href="{{url_for('about')}}">About</a></li>
26.             <li style="margin-right: 10px;"><a href="{{url_for('why')}}">why
us</a></li>
27.             <li style="margin-right: 10px;"><a
href="{{url_for('testimonial')}}">Review</a></li>
28.             <li style="margin-right: 10px;"><button class="loginbtn"
onclick="document.getElementById('login-form').style.display='block'" style="height:
50px;width: 100px;">Login</button></li>
29.         </ul>
30.     </nav>
31. </div>
32.     <div id="login-form" class="login-page" style="display: block;">
33.         <div class="form-box">
34.             <div class="button-box">
35.                 <div id="btn" style="left: 0px;"></div>
36.                 <button type="button" onclick="login()" class="toggle-btn">Sign
in</button>
37.                 <button type="button" onclick="register()" class="toggle-
btn">Sign up</button>
38.             </div>
39.             <form id="login" class="input-group-login" style="left:
50px;"action=""method="post">
40.                 <input type="text" id="mail" name="mail"class="input-field"
placeholder="Email Id" required="">
41.                 <input type="password" id="password" name="password"
class="input-field" placeholder="Enter Password" required="">
42.                 <input type="checkbox" class="check-box"><span>Remember
Password</span>
43.
44.                 <button type="submit" class="submit-btn">Sign in</button>
45.             </form>
46.             <form id="register" class="input-group-register" style="left: 450px;"
action="" method="post">
47.                 <input type="text" class="input-field" name="firstname"
placeholder="First Name" required="">
48.                 <input type="text" class="input-field" name="lastname"
placeholder="Last Name " required="">
49.                 <input type="email" class="input-field" name="email" placeholder="Email
Id" required="">
50.                 <input type="password" class="input-field" id="password"
name="password" placeholder="Enter Password" required=""value="">
51.
52.                 <input type="password" class="input-field" id="passwordconfirmation"
name="passwordconfirmation" placeholder="Confirm password" required="">

```

```

53.
54.         <input type="checkbox" class="check-box" style="margin-top:
12px;margin-bottom: 32px;"><span>I agree to the terms
and                                     conditions</span>
55.
56.         <button type="submit" name="signup" value="signup"
class="submit-btn">Sign up</button>
57.
58.
59.         </form>
60.     </div>
61. </div>
62. </div>
63. <script>
64.     var x=document.getElementById('login');
65.     var y=document.getElementById('register');
66.     var z=document.getElementById('btn');
67.     function register()
68.     {
69.         x.style.left='-400px';
70.         y.style.left='50px';
71.         z.style.left='110px';
72.     }
73.     function login()
74.     {
75.         x.style.left='50px';
76.         y.style.left='450px';
77.         z.style.left='0px';
78.     }
79. </script>
80. <script>
81.     var modal = document.getElementById('login-form');
82.     window.onclick = function(event)
83.     {
84.         if (event.target == modal)
85.         {
86.             modal.style.display = "none";
87.         }
88.     }
89. </script>
90. </body></html>
91.
92.

```

```
1. *
2. {
3.     margin: 0;
4.     padding: 0;
5.     box-sizing: border-box;
6. }
7. .full-page
8. {
9.     height: 100%;
10.    width: 100%;
11.    background-image: linear-
    gradient(rgba(0,0,0,0.4),rgba(0,0,0,0.4)),url(../img/pricing-bg.jpg);
12.    background-position: center;
13.    background-size: cover;
14.    position: absolute;
15.}
16.
17. .navbar
18. {
19.     display: flex;
20.     align-items: center;
21.     padding: 50px;
22.     padding-left: 30px;
23.     padding-right: 150px;
24.     padding-top: 10px;
25. }
26.
27. .nav
28. {
29.     flex: 1;
30.     text-align: right;
31. }
32. .nav ul
33. {
34.     display: inline-block;
35.     list-style: none;
36. }
37. .nav ul li
38. {
39.     display: inline-block;
40.     margin-right: 30px;
41. }
42. .nav ul li a
43. {
44.     text-decoration: none;
45.     font-size: 17px;
```

```

46.     color: white;
47.     font-family: "Roboto",sans-serif;
48.     font-weight: normal;
49.}
50.nav ul li button
51.{
52.
53.     font-size: 17px;
54.
55.     color: white;
56.     outline: none;
57.     border: none;
58.     background-color: #ff6f3c;
59.     cursor: pointer;
60.     font-family: "Roboto",sans-serif;
61.     text-transform: uppercase;
62.}
63.nav ul li button:hover
64.{
65.
66.     background-color: #ff6f3c;
67.}
68.nav ul li a:hover
69.{
70.     color: white;
71.
72.}
73.a
74.{
75.     text-decoration: none;
76.     color: #ffffff;
77.     font-size: 20px;
78.     text-transform: uppercase;
79.     font-weight: bold;
80.}
81.#login-form
82.{
83.     display: none;
84.}
85..form-box
86.{
87.     width:380px;
88.     height:480px;
89.     position:relative;
90.     margin:2% auto;
91.     background:rgba(0,0,0,0.8);
92.     padding:10px;
93.     overflow: hidden;

```

```

94.}
95..button-box
96.{
97.    width:220px;
98.    margin:35px auto;
99.    position:relative;
100.        box-shadow: white;
101.        border-radius: 30px;
102.    }
103.    .toggle-btn
104.    {
105.        padding:10px 25px;
106.        cursor:pointer;
107.        background:transparent;
108.        border:0;
109.        outline: none;
110.        position: relative;
111.        text-transform: uppercase;
112.        color: white;
113.    }
114.    #btn
115.    {
116.        top: 0;
117.        left:0;
118.        position: absolute;
119.        width: 110px;
120.        height: 100%;
121.        background: #433DF4;
122.        border-radius: 30px;
123.        transition: .5s;
124.    }
125.    .input-group-login
126.    {
127.        top: 150px;
128.        position:absolute;
129.        width:280px;
130.        transition:.5s;
131.    }
132.    .input-group-register
133.    {
134.        top: 120px;
135.        position:absolute;
136.        width:280px;
137.        transition:.5s;
138.    }
139.    .input-field
140.    {
141.        width: 100%;

```



```
142.         padding:10px 0;
143.         margin:5px 0;
144.         border-left:0;
145.         border-top:0;
146.         border-right:0;
147.         border-bottom: 1px solid #999;
148.         outline:none;
149.         background: transparent;
150.     }
151.     .submit-btn
152.     {
153.         width: 85%;
154.         padding: 10px 30px;
155.         cursor: pointer;
156.         display: block;
157.         margin: auto;
158.         background: #433DF4;
159.         border: 0;
160.         outline: none;
161.         border-radius: 30px;
162.     }
163.     .check-box
164.     {
165.         margin: 30px 10px 34px 0;
166.     }
167.     span
168.     {
169.         color:#777;
170.         font-size:12px;
171.         bottom:68px;
172.         position:absolute;
173.     }
174.     #login
175.     {
176.         left:50px;
177.     }
178.     #login input
179.     {
180.         color:white;
181.         font-size:15;
182.     }
183.     #register
184.     {
185.         left:450px;
186.     }
187.     #register input
188.     {
189.         color:white;
```

```
190.         font-size: 15;
191.     }
```

FLASK CODE TO HANDLE CLIENT AND SERVER SIDE VALIDATION:

Defining function to handle user-sign-in:

```
192.     @app.route('/signin', methods=['POST', 'GET'])
193.     def signin():
194.         if request.method == 'POST':
195.             #Get recquired field inputs from html(webpage)
196.             email = request.form["mail"]
197.             password = request.form["password"]
198.             # Check if user exists in IBM database
199.             sql = "SELECT firstname FROM AI_Enabled_car_parking_signup WHERE
                email = ? AND password = ?"
200.             stmt = ibm_db.prepare(conn, sql)
201.             ibm_db.bind_param(stmt, 1, email)
202.             ibm_db.bind_param(stmt, 2, password)
203.
204.             ibm_db.execute(stmt)
205.             result = ibm_db.fetch_tuple(stmt)
206.             #If the user name exists in the database redirect the user to home
                page else throughs an error message and redirect to signup page
207.             if result :
208.
209.                 username=result[0]
210.                 session['firstname']=username
211.                 return redirect(url_for('userhome'))
212.
213.             else:
214.
215.                 error = "Invalid UserId or password"
216.                 return redirect(url_for('signup', error=error))
217.             ibm_db.free_stmt(conn)
218.             print("failed conditions")
219.
220.         return render_template('signin.html')
```

2. Defining function to handle user-sign-up:

```
3. @app.route('/signup', methods=['POST', 'GET'])
4. def signup():
```

```

5.     error = request.args.get('error')
6.     if request.method == 'POST':
7.         #Get the required arguments from the webpage
8.         firstname = request.form["firstname"]
9.         lastname = request.form["lastname"]
10.        email = request.form["email"]
11.        password = request.form["password"]
12.        passwordconfirmation = request.form["passwordconfirmation"]
13.
14.        # Insert new user data into IBM database
15.        sql = "INSERT INTO AI_Enabled_car_parking_signup(firstname, lastname,
email, password, confirmpassword) VALUES (?, ?, ?, ?, ?)"
16.        stmt = ibm_db.prepare(conn, sql)
17.
18.        ibm_db.bind_param(stmt, 1, firstname)
19.        ibm_db.bind_param(stmt, 2, lastname)
20.        ibm_db.bind_param(stmt, 3, email)
21.        ibm_db.bind_param(stmt, 4, password)
22.        ibm_db.bind_param(stmt, 5, passwordconfirmation)
23.        ibm_db.execute(stmt)
24.        # Flash success message and redirect user to sign in page
25.        flash('Successfully signed up!')
26.        # send the mail to the user on successful registration with the signin link
27.        sign_in_link=request.host_url+url_for('signin')
28.        send_email(email,'Welcome to our Website',f'Dear {firstname} {lastname},
you have successfully registered with our platform.You can now login and utilize
our service by clicking the link below:\n\n\n{sign_in_link}')
29.        return redirect(url_for('signin'))
30.
31.    return render_template('signup.html',error=error)
32.

```

3. Video Feed Integration

1. Set up a video feed source, such as camera or pre-recorded video file.

```
2. Video path='/path for the video file'
```

3. Capture frames from the video feed using a video processing library.

```

4. #Import the computer vision library
5. import cv2
6. cap = cv2.VideoCapture(video_source)
7. while True:
8.     success,img = cap.read()

```

```

9.     cv2.imshow("Title",img)
10.    cv2.waitKey(0)

```

4. Preprocessing the frames as necessary.

```

1.  imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
2.  imgBlur = cv2.GaussianBlur(imgGray, (3, 3), 1)
3.
4.  if val1 % 2 == 0: val1 += 1
5.  if val3 % 2 == 0: val3 += 1
6.  imgThres = cv2.adaptiveThreshold(imgBlur, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
7.                                  cv2.THRESH_BINARY_INV, val1, val2)
8.  imgThres = cv2.medianBlur(imgThres, val3)
9.  kernel = np.ones((3, 3), np.uint8)
10. imgThres = cv2.dilate(imgThres, kernel, iterations=1)

```

4. Parking Lot Analysis

1. Using Image thresholding technique to detect the availability of the parking lots.

```

2. while (cap.isOpened()):
3.     # Get image frame
4.     success, img = cap.read()
5.     if success:
6.         if cap.get(cv2.CAP_PROP_POS_FRAMES) ==
cap.get(cv2.CAP_PROP_FRAME_COUNT):
7.             cap.set(cv2.CAP_PROP_POS_FRAMES, 0)
8.             # img = cv2.imread('img.png')
9.             imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
10.            imgBlur = cv2.GaussianBlur(imgGray, (3, 3), 1)
11.            # ret, imgThres = cv2.threshold(imgBlur, 150, 255, cv2.THRESH_BINARY)
12.            if val1 % 2 == 0: val1 += 1
13.            if val3 % 2 == 0: val3 += 1
14.            imgThres = cv2.adaptiveThreshold(imgBlur, 255,
cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
15.                                            cv2.THRESH_BINARY_INV, val1, val2)
16.            imgThres = cv2.medianBlur(imgThres, val3)
17.            kernel = np.ones((3, 3), np.uint8)
18.            imgThres = cv2.dilate(imgThres, kernel, iterations=1)
19.            checkSpaces()
20.            # Display Output
21.            out.write(img)
22.            cv2.imshow("Image",img)
23.            key = cv2.waitKey(1)
24.            if (cv2.waitKey(1)==ord('q')):

```

```

25.             break
26.         else:
27.             break
28.

```

2. Appropriate algorithm to mark the parking spaces and to count the number of available parking spaces.

```

1. #Setting the size of rectangle according to the parking slot length and width
2. width,height=45,21
3. #load the parking space information from a file named parking slot position
4. with open('ParkingSlotPosition', 'rb') as f:
5.     posList = pickle.load(f)
6.
7. #Setting the Adaptive thresholding value
8.
9. val1=26
10.val2=25
11.val3=2
12.def checkSpaces():
13.     b=0
14.     spaces = 0
15.     for pos in posList:
16.         x, y = pos
17.         if c==0:
18.             if (b<=7):
19.                 w, h = width, height
20.                 b=b+1
21.             else:
22.                 w,h=width1,height1
23.         elif c==1:
24.             w,h=width,height
25.         elif c==3:
26.             w,h=width,height
27.
28.         imgCrop = imgThres[y:y + h, x:x + w]
29.         count = cv2.countNonZero(imgCrop)
30.         if count < 980:
31.             color = (0, 255, 0)
32.             thic = 5
33.             spaces += 1
34.
35.         else:
36.             color = (0, 0, 255)

```

```

37.         thic = 2
38.         cv2.rectangle(img, (x, y), (x + w, y + h), color, thic)
39.         cv2.putText(img, str(cv2.countNonZero(imgCrop)), (x, y + h - 6),
40.                     cv2.FONT_HERSHEY_PLAIN,1,color,2)
41.         cvzone.putTextRect(img, f'Free:{spaces}/{len(posList)}', (50, 60),
42.                             thickness=3, offset=20, colorR=(0, 200, 0))

```

3. Update the parking lot availability status in real-time

```

1. def process_video(frame,c):
2.     if c == '2':
3.         width,height=45,21
4.         with open('ParkingSlotPosition3', 'rb') as f:
5.             posList = pickle.load(f)
6.     if c == '1':
7.         width,height=107,48
8.         with open('ParkingSlotPosition', 'rb') as f:
9.             posList = pickle.load(f)
10.    if c=='3':
11.        width, height = 140,72
12.        width1,height1=65,155
13.        with open('ParkingSlotPosition2', 'rb') as f:
14.            posList = pickle.load(f)
15.    def empty(c):
16.        pass
17.    if c=='3':
18.        val1=26
19.        val2=25
20.        val3=2
21.    if c=='1':
22.        val1=25
23.        val2=14
24.        val3=4
25.    if c=='2':
26.        val1=26
27.        val2=14
28.        val3=4
29.
30.    def checkSpaces():
31.        b = 0
32.        spaces = 0
33.
34.        for pos in posList:
35.            x, y = pos
36.            if c == '3':

```

```

37.         if b <= 7:
38.             w, h = width, height
39.             b = b + 1
40.         else:
41.             w, h = width1, height1
42.     elif c == '1':
43.         w, h = width, height
44.     elif c == '2':
45.         w, h = width, height
46.
47.     imgCrop = imgThres[y:y + h, x:x + w]
48.     count = cv2.countNonZero(imgCrop)
49.     if c == '1':
50.         if count < 980:
51.             color = (0, 255, 0)
52.             thic = 5
53.             spaces += 1
54.         else:
55.             color = (0, 0, 255)
56.             thic = 2
57.     if c == '3':
58.         if count < 710:
59.             color = (0, 255, 0)
60.             thic = 5
61.             spaces += 1
62.         else:
63.             color = (0, 0, 255)
64.             thic = 2
65.     if c == '2':
66.         if count < 250:
67.             color = (0, 255, 0)
68.             thic = 5
69.             spaces += 1
70.         else:
71.             color = (0, 0, 255)
72.             thic = 2
73.
74.     cv2.rectangle(frame, (x, y), (x + w, y + h), color, thic)
75.     cv2.putText(frame, str(cv2.countNonZero(imgCrop)), (x, y + h -
76.                                     6), cv2.FONT_HERSHEY_PLAIN, 1, color, 2)
77.     cvzone.putTextRect(frame, f'Free:{spaces}/{len(posList)}', (50,
78.                                     60), thickness=3, offset=20, colorR=(0, 200,
79.                                     0))
80.     if val1 % 2 == 0: val1 += 1
81.     if val3 % 2 == 0: val3 += 1
82.     imgGray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
83.
84.     imgBlur = cv2.GaussianBlur(imgGray, (3, 3), 1)

```

```

85.     imgThres = cv2.adaptiveThreshold(imgBlur, 255,
86.                                     cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
87.                                     cv2.THRESH_BINARY_INV, val1, val2)
88.     imgThres = cv2.medianBlur(imgThres, val3)
89.     kernel = np.ones((3, 3), np.uint8)
90.     imgThres = cv2.dilate(imgThres, kernel, iterations=1)
91.     checkSpaces()
92.     return frame

```

FLASK CODE FOR HANDLING AND DISPLAYING REALTIME STATUS

```

1. # make a request to video source and return the response on the output
2. @app.route('/video_feed')
3. def video_feed_endpoint(c):
4.     c = session['c']
5.     return Response(gen(c), mimetype='multipart/x-mixed-replace;boundary=frame')
6.
7. def gen(c):
8.     if c == '1':
9.         cap = cv2.VideoCapture('/home/arj/IBM_project/Data/carParkingInput.mp4')
10.    elif c == '2':
11.        cap = cv2.VideoCapture('/home/arj/IBM_project/Data/carparking3.mp4')
12.    elif c == '3':
13.        cap = cv2.VideoCapture('/home/arj/IBM_project/Data/carparking2.mp4')
14.    else:
15.        return
16.
17.    while True:
18.        ret, frame = cap.read()
19.        if not ret:
20.            break
21.        start_time=cv2.getTickCount()
22.        processed_frame = process_video(frame, c)
23.
24.        end_time=cv2.getTickCount()
25.        processing_time=(end_time-start_time)/cv2.getTickFrequency()
26.
27.        print("Processing time: %.3f sec" % processing_time)
28.        _, jpeg = cv2.imencode('.jpg', processed_frame)
29.        frame_bytes = jpeg.tobytes()
30.        yield (b'--frame\r\n'
31.              b'Content-Type: image/jpeg\r\n\r\n' + frame_bytes + b'\r\n')
32.
33.    cap.release()
34. @app.route('/userhome', methods=['POST', 'GET'])
35. def userhome():

```



```

36.     firstname = session.get('firstname')
37.     if firstname is None:
38.         return "No 'firstname' value found in the session"
39.
40.     if request.method == "POST":
41.         parkingslot = request.form.get("parkingslot")
42.         if parkingslot == "parking slot 1":
43.             c = '1'
44.             session['c'] = c
45.             return redirect(url_for('usermodel'))
46.         elif parkingslot == "parking slot 2":
47.             c = '2'
48.             session['c'] = c
49.             return redirect(url_for('usermodel'))
50.         elif parkingslot == "parking slot 3":
51.             c = '3'
52.             session['c'] = c
53.             return redirect(url_for('usermodel'))
54.
55.     return render_template('home.html', firstname=firstname)

```

FRONTEND DEVELOPMENT

1. Design and development of the frontend interface for the web-application.

We had used the Bootstrap framework to develop the web pages.

2. Dashboard page that displays the real-time parking availability information.

HTML CODE

```

1. <!DOCTYPE html>
2. <html>
3.     <head>
4.         <meta http-equiv="content-type" content="text/html; charset=UTF-8">
5.         <title>Car Parking Lot</title>
6.         <link href="../static/css/model.css" rel="stylesheet">
7.         <style>
8.             .video-container {
9.                 display: flex;
10.                justify-content: center;
11.                align-items: center;
12.                height: calc(100% - 90px);
13.            }
14.

```

```

15.     .hidden {
16.         display: none;
17.     }
18.
19.     .video-btn {
20.         padding: 10px 20px;
21.         font-size: 16px;
22.         background-color: #4CAF50;
23.         color: white;
24.         border: none;
25.         cursor: pointer;
26.         border-radius: 4px;
27.         transition: background-color 0.3s ease;
28.         box-shadow: 0px 2px 4px rgba(0, 0, 0, 0.2);
29.     }
30.
31.     .video-btn:hover {
32.         background-color: #45a049;
33.     }
34.
35.     .video-btn:active {
36.         background-color: #3e8e41;
37.     }
38. </style>
39. </head>
40. <body>
41.     <div class="full-page" method="post">
42.         <div class="navbar" style="padding-top: 15px;padding-bottom: 25px;">
43.             <h1 style="margin-left: 100px;">Welcome {{firstname}} !!</h1>
44.             <nav style="margin-right: 100px;">
45.                 <ul>
46.                     <li><a href="{{url_for('userhome')}}">Home</a></li>
47.                     <li><a href="{{url_for('userabout')}}">About</a></li>
48.                     <li><a href="{{url_for('index')}}">Logout</a> </li>
49.                 </ul>
50.             </nav>
51.         </div>
52.         <div class="video-container">
53.             <br>
54.             <button class="video-btn" id="statusButton" onclick="showOutput()">Check
55.             the Status</button>
56.             <div id="output" class="hidden">
57.                 
59.             </div>
60.         </div>
61.     </div>
62.

```

```

63.     <script>
64.         function showOutput() {
65.             var button = document.getElementById("statusButton");
66.             var output = document.getElementById("output");
67.             button.style.display = "none";
68.             output.classList.remove("hidden");
69.         }
70.     </script>
71. </body>
72. </html>
73.
74.

```

CSS

```

1. * {
2.     margin: 0;
3.     padding: 0;
4.     box-sizing: border-box;
5. }
6. .full-page {
7.     height: 100%;
8.     width: 100%;
9.     background-image: linear-
gradient(rgba(0,0,0,0.4),rgba(0,0,0,0.4)),url(images/pricing-bg.jpg);
10.    background-position: center;
11.    background-size: cover;
12.    position: absolute;
13. }
14. .navbar {
15.     display: flex;
16.     align-items: center;
17.     justify-content: space-between;
18.     padding: 30px;
19. }
20. .navbar h1 {
21.     font-size: 28px;
22.     font-weight: bold;
23.     color: white;
24. }
25. nav {
26.     flex: 1;
27.     text-align: right;
28. }
29. nav ul {
30.     display: inline-block;
31.     list-style: none;
32. }

```

```
33.   nav ul li {
34.     display: inline-block;
35.     margin-right: 30px;
36.   }
37.   nav ul li a {
38.     text-decoration: none;
39.     font-size: 17px;
40.     color: white;
41.     font-family: "Roboto",sans-serif;
42.     font-weight: normal;
43.   }
44.   nav ul li button {
45.     font-size: 17px;
46.     color: white;
47.     outline: none;
48.     border: none;
49.     background-color: #ff6f3c;
50.     cursor: pointer;
51.     font-family: "Roboto",sans-serif;
52.     text-transform: uppercase;
53.     padding: 10px 20px;
54.     border-radius: 30px;
55.   }
56.   nav ul li button:hover {
57.     background-color: #ff5e1c;
58.   }
59.   nav ul li a:hover {
60.     color: white;
61.   }
62.   .video-btn {
63.     position: absolute;
64.     top: 50%;
65.     left: 50%;
66.     transform: translate(-50%, -50%);
67.     z-index: 1;
68.   }
69.   .video-btn button {
70.     font-size: 24px;
71.     color: white;
72.     outline: none;
73.     border: none;
74.     background-color: rgba(0, 0, 0, 0.5);
75.     cursor: pointer;
76.     font-family: "Roboto",sans-serif;
77.     text-transform: uppercase;
78.     padding: 20px 30px;
79.     border-radius: 50%;
80.     transition: background-color 0.2s ease-in-out;
```

```

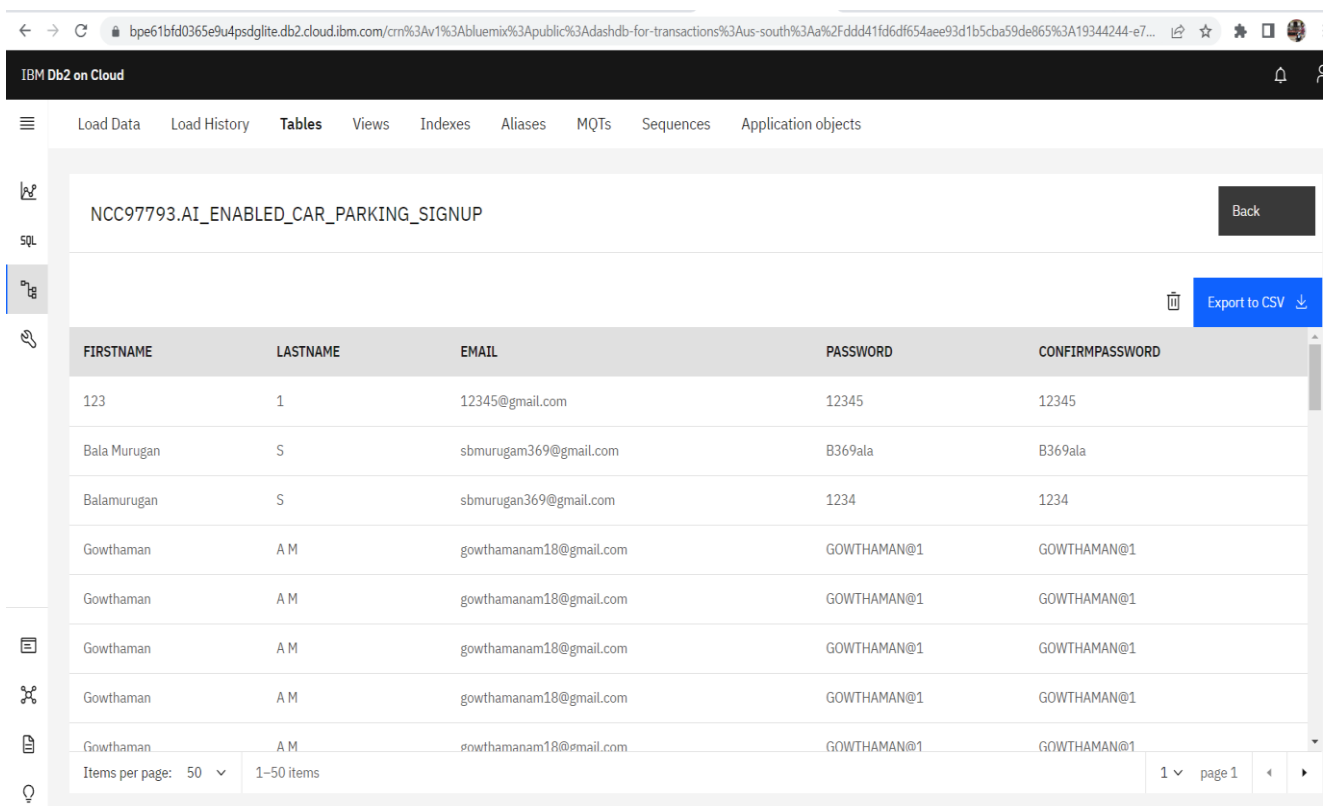
81.  }
82.  .video-btn button:hover {
83.    background-color: rgba(0, 0, 0, 0.8);
84.  }

```

DATABASE INTEGRATION

1. Set up the database to store user information, session data and any relevant parking lot data

- IBM DB2 is used here.
- The below picture depicts the database for the project “AI Enabled Car Parking Using Open CV”



IBM Db2 on Cloud

Load Data Load History **Tables** Views Indexes Aliases MQTs Sequences Application objects

NCC97793.AI_ENABLED_CAR_PARKING_SIGNUP [Back](#)

[Export to CSV](#)

FIRSTNAME	LASTNAME	EMAIL	PASSWORD	CONFIRMPASSWORD
123	1	12345@gmail.com	12345	12345
Bala Murugan	S	sbmurugam369@gmail.com	B369ala	B369ala
Balamurugan	S	sbmurugan369@gmail.com	1234	1234
Gowthaman	A M	gowthamanam18@gmail.com	GOWTHAMAN@1	GOWTHAMAN@1
Gowthaman	A M	gowthamanam18@gmail.com	GOWTHAMAN@1	GOWTHAMAN@1
Gowthaman	A M	gowthamanam18@gmail.com	GOWTHAMAN@1	GOWTHAMAN@1
Gowthaman	A M	gowthamanam18@gmail.com	GOWTHAMAN@1	GOWTHAMAN@1
Gowthaman	A M	gowthamanam18@gmail.com	GOWTHAMAN@1	GOWTHAMAN@1
Gowthaman	A M	gowthamanam18@gmail.com	GOWTHAMAN@1	GOWTHAMAN@1

Items per page: 50 1–50 items 1 page 1

3. Database operation to store and retrieve the data.

- Need to setup the administration to access the database
- Download and configure the “SSL” certificate

Connections

Connect your apps and clients to IBM Db2 on Cloud

Linux PowerLinux Mac **Windows**

Instructions

- 1. Download Windows driver package**
Download Windows driver package from [driver list](#)

File name: ibm_data_server_driver_package_win64_v11.5.exe (104 MB)
- 2. Install the drivers by running the ibm_data_server_driver_package_win64_v11.5.exe file as an administrator.**
- 3. In The Connection configuration resources section, select whether or not you want to secure your connections by using SSL.**
If your application uses its own driver and you want to connect with SSL, download the SSL certificate (DigiCertGlobalRootCA.crt).
For Java apps, use the JDBC string as the database URL in your call to the JDBC getConnection method.
For ODBC apps, add new entries to the db2dsdriver.cfg driver configuration file by running the following commands:

For connections with SSL

Connection configuration resources

Host name: b70af05b-76e4-4bca-a1f5-23dbb4c6a74e.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud

With SSL: Yes

Port number: 32716

Database name: bludb

User ID: <user name>

Password: *****

Version: Compatible with Db2, Version 11.5.0 or later

[Download SSL Certificate](#)

JDBC string

```
jdbc:db2://b70af05b-76e4-4bca-a1f5-23dbb4c6a74e.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud:32716/bludb:user=<user_name>;password=<your_password>;sslco
```

FLASK CODE TO ACCESS,STORE AND RETRIEVE DATA:

```
1. from flask import Flask, render_template, request, session, redirect,
2.                                     url_for, flash, Response
3. import ibm_db
4. app = Flask(__name__)
5. app.secret_key = 'ARJ23400'
6. # Establish connection with IBM database
7. conn = ibm_db.connect("DATABASE=bludb;HOSTNAME=b70af05b-76e4-4bca-a1f5-
    23dbb4c6a74e.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud;PORT=32716;SECURITY=s
    sl;SSLServerCertificate=DigiCertGlobalRootCA.crt;UID=ncc97793;PWD=TVIsFuhT1vXc7Zi
    f;", "", "")
8.
9. # example flask code to retrieve data from database and to authenticate the user
10. if request.method == 'POST':
11.     email = request.form["mail"]
12.     password = request.form["password"]
13.     # Check if user exists in IBM database
14.     sql = "SELECT firstname FROM AI_Enabled_car_parking_signup WHERE email =
        ? AND password = ?"
15.     stmt = ibm_db.prepare(conn, sql)
16.     ibm_db.bind_param(stmt, 1, email)
17.     ibm_db.bind_param(stmt, 2, password)
18.
19.     ibm_db.execute(stmt)
20.     result = ibm_db.fetch_tuple(stmt)
```

```

21. if result :
22.     print("Access Granted")
23. else:
24.     print("No user data found sign-up to continue")
25.
26. # example flask code to store the data to database
27. if request.method == 'POST':
28.     firstname = request.form["firstname"]
29.     lastname = request.form["lastname"]
30.     email = request.form["email"]
31.     password = request.form["password"]
32.     passwordconfirmation = request.form["passwordconfirmation"]
33.
34.     # Insert new user data into IBM database
35.     sql = "INSERT INTO AI_Enabled_car_parking_signup(firstname, lastname,
36.         email, password, confirmpassword) VALUES (?, ?, ?, ?, ?)"
37.     stmt = ibm_db.prepare(conn, sql)
38.
39.     ibm_db.bind_param(stmt, 1, firstname)
40.     ibm_db.bind_param(stmt, 2, lastname)
41.     ibm_db.bind_param(stmt, 3, email)
42.     ibm_db.bind_param(stmt, 4, password)
43.     ibm_db.bind_param(stmt, 5, passwordconfirmation)
44.     ibm_db.execute(stmt)
45.     print("Data Stored Successfully")

```

5.2 Feature 2

Mailing System For User Signup

Implementing a mailing system for user signup for our web application. Providing the user a confirmation on successful registration with our web application.

FLASK CODE:

```

1. from flask_mail import Mail, Message
2. def send_email(to, subject, body):
3.     msg=Message(subject=subject, recipients=[to], body=body)
4.     mail.send(msg)
5.
6.
7. app.config['MAIL_SERVER']='smtp.gmail.com'
8. app.config['MAIL_PORT']=587
9. app.config['MAIL_USE_TLS']=True
10. app.config['MAIL_USERNAME']='arj0654@gmail.com'

```

```

11.app.config['MAIL_PASSWORD']='enhlidejnzplqumx' # APP password to access mail
12.                                     delivery system
13.app.config['MAIL_DEFAULT_SENDER']='arj0654@gmail.com'
14.
15.if __name__ == '__main__':
16.    send_email(email,'Welcome to our Website',f'Dear user you have successfully
    registered with our platform')

```

5.2 Database Schema

User Table

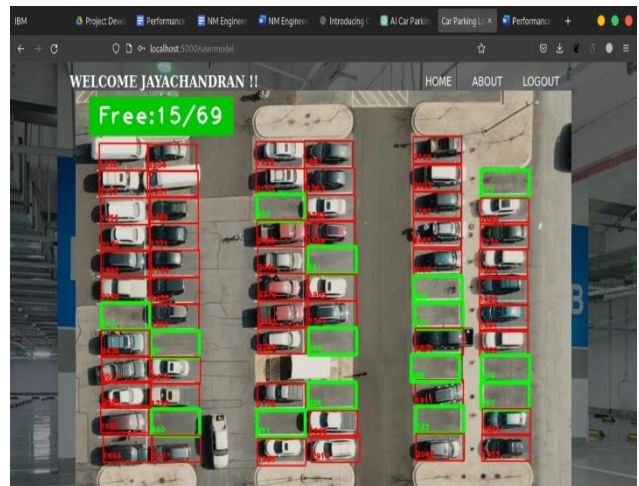
1. First-name
2. Last-name
3. Email
4. Password
5. Confirm password

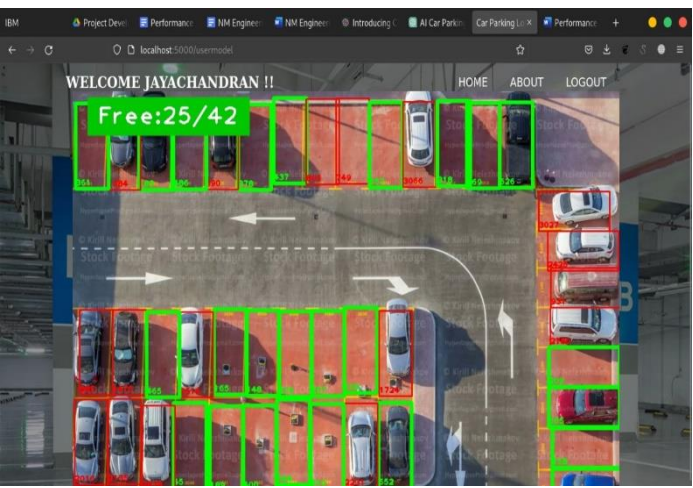
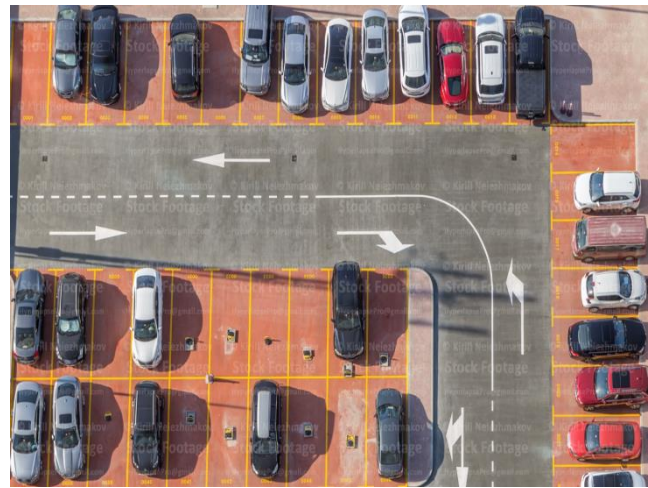
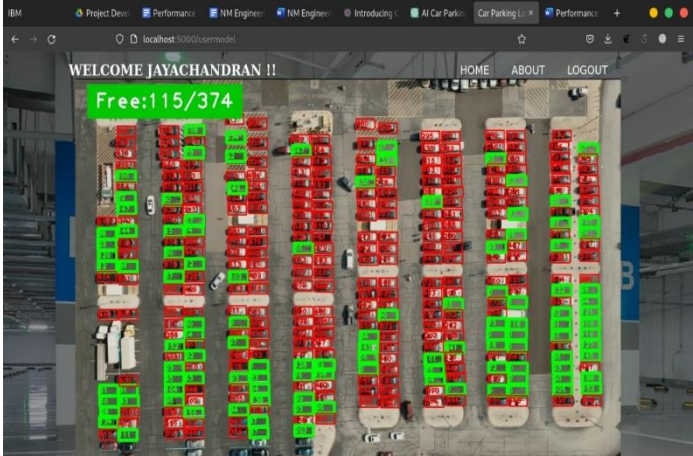
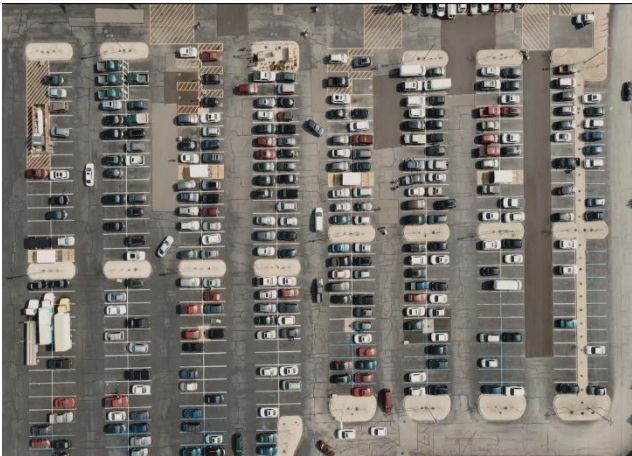
RESULT

Input

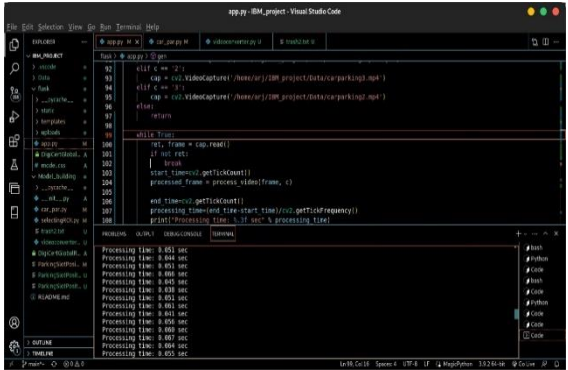
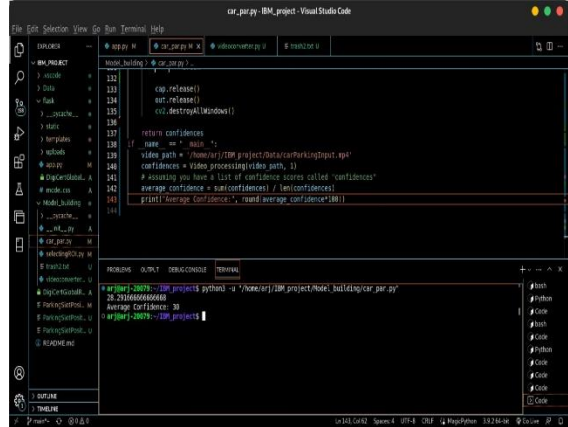
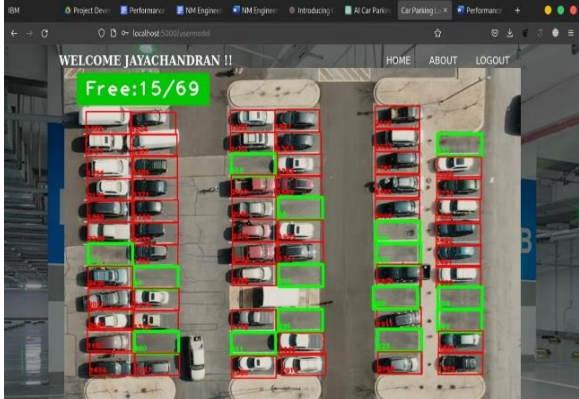


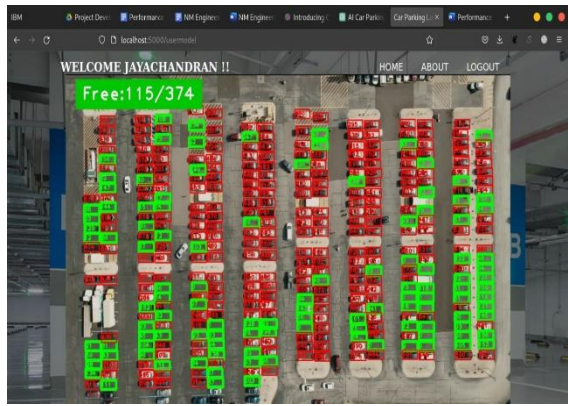
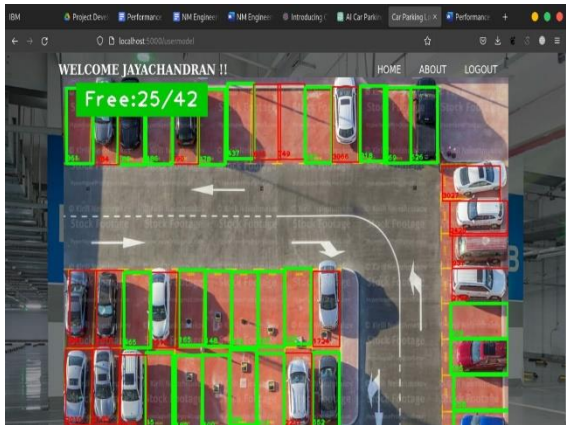
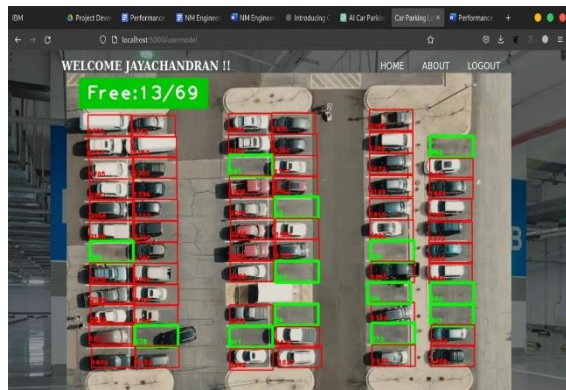
Output

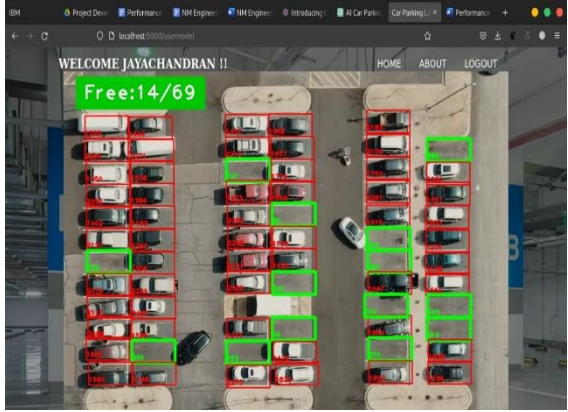




Performance Metrics

S.No.	Parameter	Values	Screenshot
1.	Processing Speed	60ms/frames	
2.	Confidence Score	30	
3.	Total Cars Parked and available slots in parking space 1	54 & 15	

4.	Total Cars Parked and available slots in sot parking space 2	259 & 115	
5.	Total Cars Parked and available slots in sot parking space 3	25 & 17	
6.	Average Parking Time	2-3 minutes	

7.	Miss Calculation	
----	------------------	--

ADVANTAGES

- 1. Real-time Parking Availability** – The application provides real-time information about parking space availability, allowing users to quickly find vacant spot. This helps to reduce the time spent searching for parking and improves the overall parking convenience.
- 2. Improved Efficiency** – This application automates the process of counting and monitoring parking spaces. This eliminates the need for manual counting or physical inspections and thus it save the time of parking lot operators.
- 3. Enhanced User Experience** – With real-time availability and a user-friendly interface, the web-application provides a seamless experience for users. They can easily check parking space availability.
- 4. Optimal Space Utilization** – By tracking the occupancy status of the parking spaces, this application helps to optimize the utilization of available parking resources. Parking lot operators can gain insights into peak hours, parking patterns and occupancy rates and it enables them to make informed decisions about capacity planning and resource allocation.
- 5. Remote monitoring** – As the application is web-based, parking lot operators can remotely monitor the parking lot status and access the real-time information from any location. This enables efficient management and quick response to any issues or irregularities.
- 6. Cost Savings** – This application helps to reduce the labor costs associated with manual monitoring and increases operational efficiency. It also minimizes revenue loss due to improper space utilization or inadequate parking guidance.

DISADVANTAGES

- 1. Environmental Limitations** – The accuracy of the AI algorithms used for vehicle detection and parking space counting can be influenced by environmental conditions, such as lighting, weather conditions, camera angles. Variations in these conditions can impact the accuracy and reliability of the system.

2. **Limited Coverage and Scalability** – The application’s effectiveness may be limited to the areas covered by the installed cameras or sensors. Expanding the coverage to larger parking lots or multiple locations may require additional hardware and resources, increasing the complexity and cost of the system.
3. **Dependency of the Video Feed Quality** – The accuracy of the parking space detection and occupancy tracking heavily relies on the quality and consistency of the video feed. Any disruptions or issues with the feed, such as low-resolution or poor visibility, can affect the system’s performance and reliability.
4. **Data Privacy and Security** – The web-application collects and stores the user data, including the personal information. Ensuring the robust security measures, including data encryption and secure access controls, is crucial to protect the user privacy and prevent unauthorized access or data breaches.
5. **System Reliance & Downtime** – The web-application relies on stable internet connectivity and server infrastructure for real-time updates and user access. Any downtime or technical issues can impact the availability and functionality of the application, potentially causing inconvenience to users.
6. **Regulatory and Legal Compliance** – Depending on the jurisdiction, there may be specific regulations and privacy laws that govern the collection, storage and usage of data obtained by the web-application. Compliance with these regulations is essential to avoid legal compliances and maintain trust with users.

CONCLUSION

The web-application on AI Enabled Car Parking using OpenCV offers significant advantages in improving parking management efficiency, enhancing user experience, and optimizing parking space utilization. By leveraging AI algorithms and computer vision techniques, the application provides real-time parking availability information, accurate vehicle detection and a seamless user friendly user interface. This project enables users to easily find the vacant parking spaces, and navigate to the parking lots.

Overall, this project aims to revolutionize the traditional parking management approach by leveraging AI and Computer vision technology. By automating parking space monitoring, providing real-time information and optimizing resource allocation, the web-application offers a more efficient and convenient parking experience for users while helping parking lot operators maximize their operational efficiency.

With careful consideration of the advantages, disadvantages and ongoing improvement, the web-application on AI enabled Car Parking Using OpenCV can contribute to transforming the web-application and making the parking management more streamlined, convenient and optimized.

FUTURE SCOPE

The web-application on AI Enabled Car Parking Using OpenCV holds significant potential for future development and enhancement. Here are some future scope considerations for the project:

1. **Integration with Smart City Initiatives** – Explore opportunities to integrate the parking application with broader smart city initiatives. This can involve integrating with smart traffic management systems, intelligent transportation systems.
2. **Mobile Application Development** – Develop a mobile application companion for the web application, allowing users to access real-time parking information and receive notifications on their smartphones. This enhances the user convenience and accessibility.
3. **Geolocation and Navigation Integration** – Incorporate the geolocation services and navigation functionalities within the application to assist users in finding parking lot, navigating to available spaces, and providing directions to their destinations.
4. **Expansion to Multiple Locations** – Extend the application's scope to cover multiple parking locations, including multi-level parking facilities or parking lots in different geographical areas.
5. **Integration with Internet of Things(IOT)** – Explore the integration of IOT devices and platforms to enhance real-time monitoring and control of parking lots. This can include the integration of smart camera, sensors or dynamic signage systems for improved management and user experience

APPENDIX

Github link:

<https://github.com/naanmudhalvan-SI/PBL-NT-GP--1602-1680518110>

Demo Video link:

[https://drive.google.com/file/d/132EynyVLlkvMof5Cb58nqzvF4g-1leN/view?usp=share link](https://drive.google.com/file/d/132EynyVLlkvMof5Cb58nqzvF4g-1leN/view?usp=share_link)