AI Enabled Car Parking Using OpenCV

Tarun Satheesh 20BCT0203

Gurik Sachar 20BEC0130

Jaskaran Singh 20BCT0306

Dev Kolsawala 20BDS0278

Smart-Bridge

Project-Title: AI Enabled Car Parking Using OpenCV

INTRODUCTION:

Overview:

The goal of this project is to create a website/app which enables users to check for free parking spaces without having to actually be present at the parking lot. This is possible through the use of OpenCV with the help of an algorithm to check for free parking spots and tally the amount of free parking spots in total. The user will also be able to see the empty and occupied spots in real time via green and red bounding boxes respectively.

Purpose:

The purpose of this project is to simplify the parking space finding task for drivers. It is highly inconvenient to wait for a parking spot to become empty. Instead of wasting time waiting, if the driver could get an idea before-hand through the application then they can plan some other tasks before the parking spot becomes empty. As soon as a free parking spot is available as shown on the application, then the driver can proceed to occupy the free parking spot. This will save a lot of time and in turn, the productivity of the driver.

LITERATURE SURVEY:

Existing Problem:

As we know, it is highly inconvenient to keep looking for a free parking spot. The driver can end up waiting for at least 10-15 minutes before finally getting a free spot.

According to USA Today, motorists spend an average of 17 hours a year searching for spots on streets, in lots, or in garages, according to a report issued Wednesday. The hunt adds up to an estimated $345 per driver in wasted time, fuel, and emissions, according to the analysis by INRIX, a leading specialist in connected car services and transportation analytics. Overpaying, caused by drivers' inability to estimate how long they need to park or forking over extra at a garage to avoid the risk of getting a parking ticket costs Americans more than $20 billion a year or $97 per driver, the report estimated.

Existing Solution:

The solution to the current problem is that, complexes try to keep multiple floors of parking to avoid overcrowding as much as possible. It does help to an extent. But there is a new disadvantage here, the drivers might have to go up many floors before they finally find a free parking spot, because the lower floors might already be occupied. Again, this leads to a waste of time, fuel energy and money.

Proposed Solution:

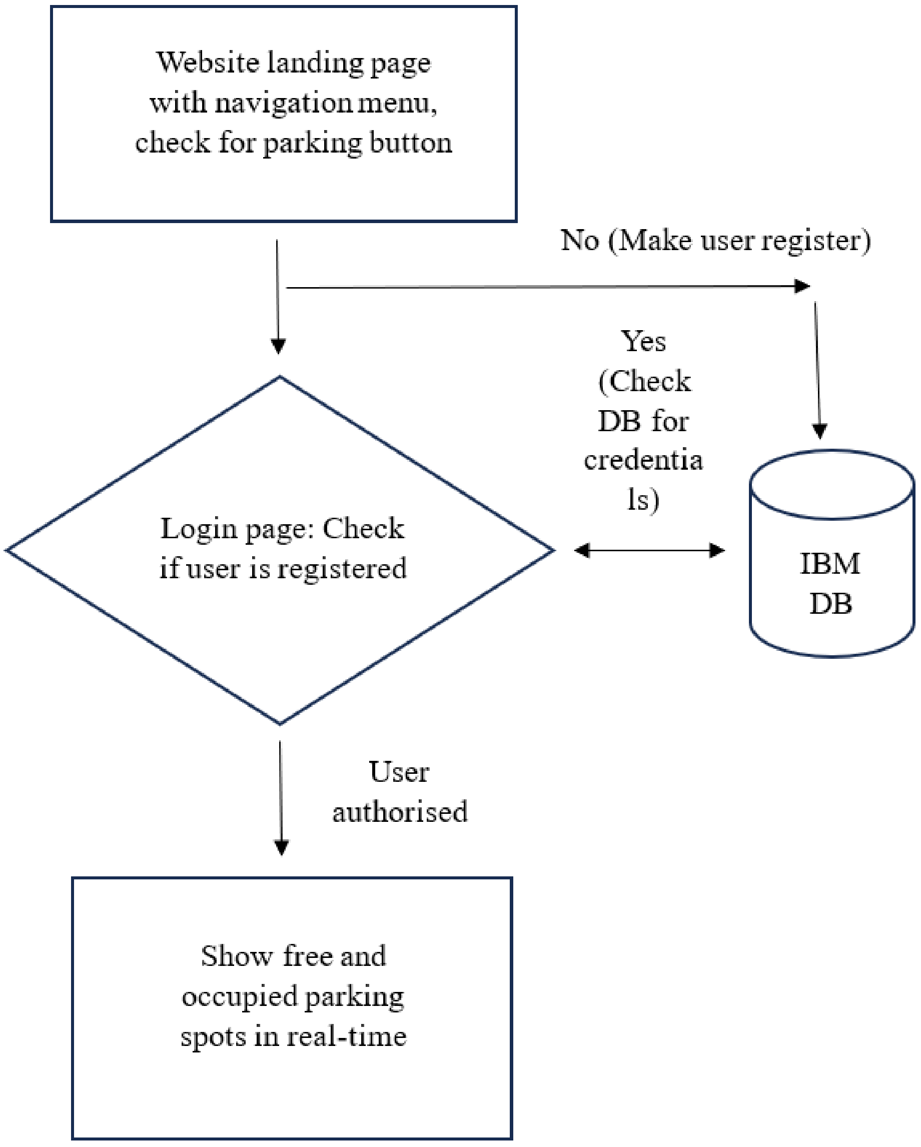
Through AI enabled car parking using OpenCV, the users will be able to use an app which uses OpenCV along with an algorithm to check for free parking spots and tally the amount of free parking spots in total. The user will also be able to see the empty and occupied spots in real time via green and red bounding boxes respectively.

THEORETICAL ANALYSIS:

Software Designing:

1. **Webpages**: The application consists of three major components:
   a. **Index.html:** Consist of a landing page which has a navigation menu, a background of a car parking, and a check for parking button which can be used to move to the next web page which is login.html.
   b. **Login.html:** This page asks the user for their email and password. If they do not have an account, then they are required to sign up via a basic html form. After logging in, the user is redirected to model.html.
   c. **Model.html:** This page shows a live feed of the free and occupied parking spaces with the use of OpenCV with an AI algorithm running in the background.
2. **Python source code:** This includes the OpenCV code along with an AI algorithm to draw bounding boxes around the occupied and the free parking spots in the parking lot and to take a tally of the number of free spots available.
3. **Flask framework:** Flask is a Python framework which is used for the deployment of static webpages. With Flask, we have a template for the website ready, which consists of HTML, CSS and JS files. Flask renders these files by routing the user in between them. Using Flask, we have also embedded the actual AI algorithm with the help of OpenCV to check for free parking spaces.

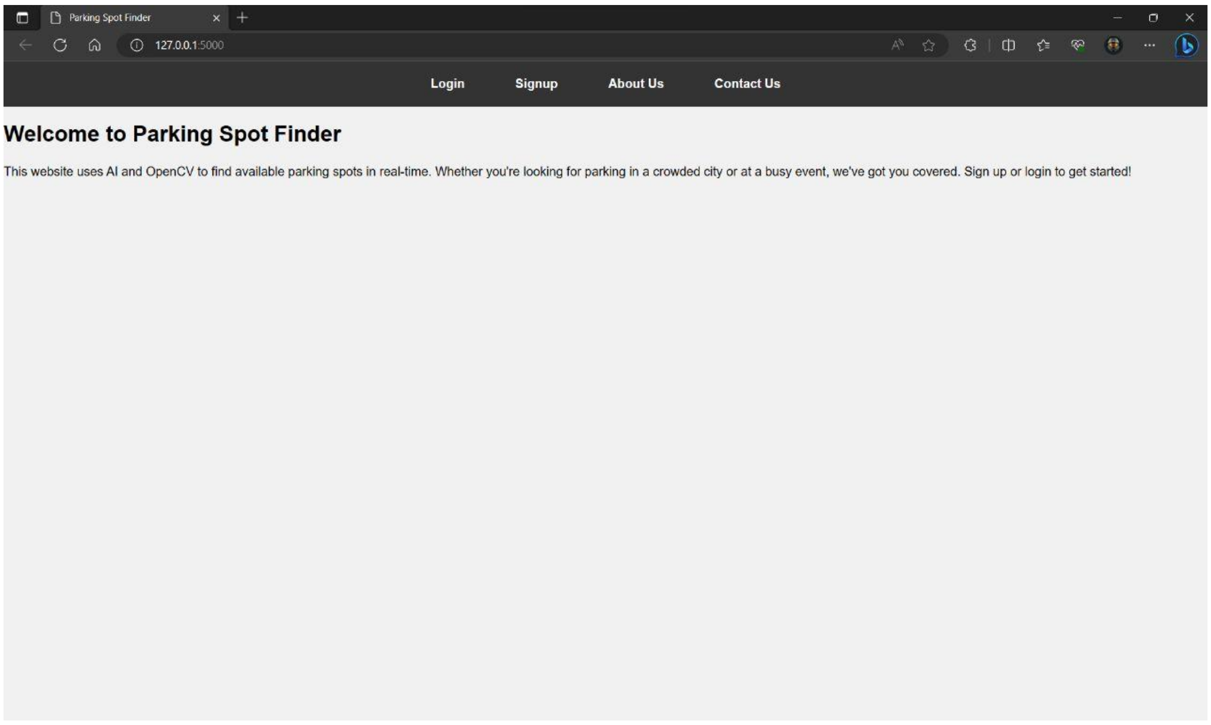
FLOWCHART:


Website landing page with navigation menu, check for parking button
No (Make user register)
Yes (Check DB for credentials)
Login page: Check if user is registered
IBM DB
User authorised
Show free and occupied parking spots in real-time

RESULT:

Once the project was completed and all the code was put together, this is what it finally looked like:

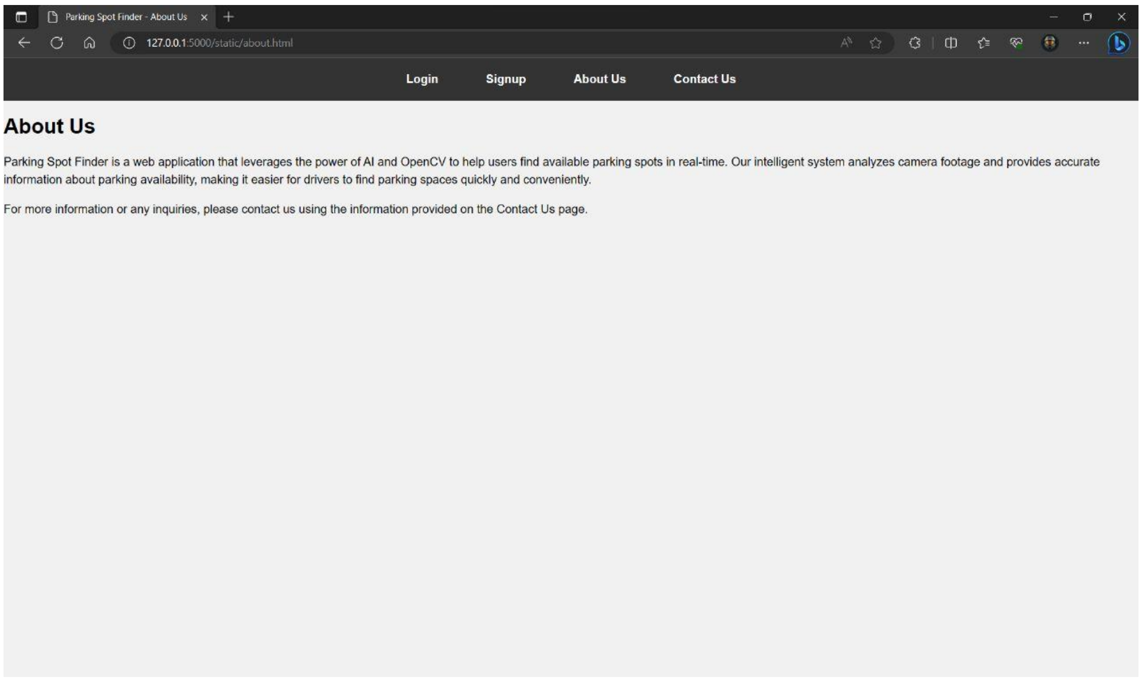
Landing page:



Welcome to Parking Spot Finder

This website uses AI and OpenCV to find available parking spots in real-time. Whether you're looking for parking in a crowded city or at a busy event, we've got you covered. Sign up or login to get started!

About Us page:

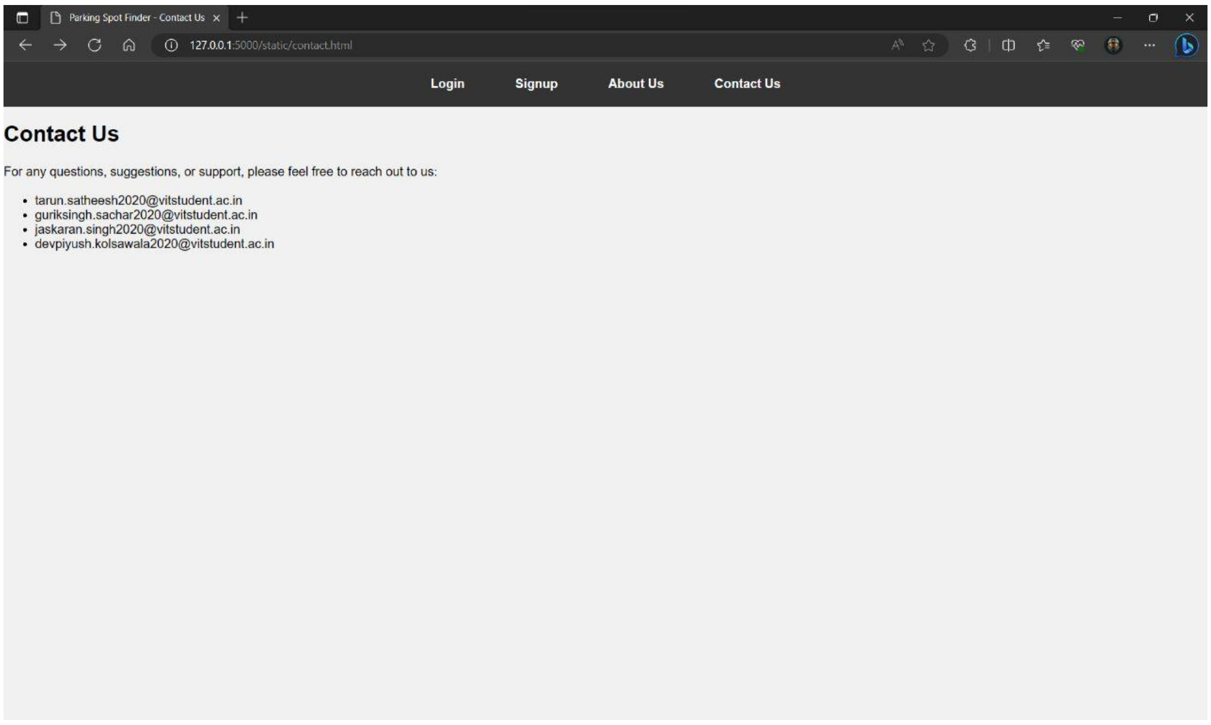

Parking Spot Finder - About Us
127.0.0.1:5000/static/about.html
Login
Signup
About Us
Contact Us

About Us

Parking Spot Finder is a web application that leverages the power of AI and OpenCV to help users find available parking spots in real-time. Our intelligent system analyzes camera footage and provides accurate information about parking availability, making it easier for drivers to find parking spaces quickly and conveniently.

For more information or any inquiries, please contact us using the information provided on the Contact Us page.

Contact page:



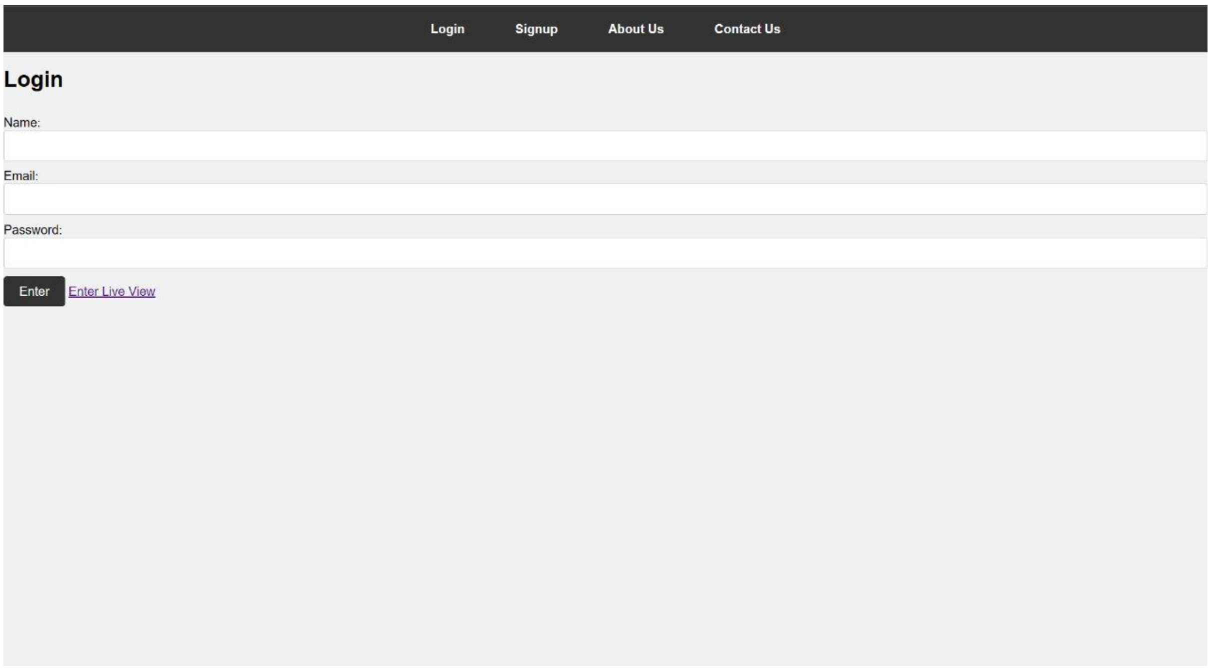
Parking Spot Finder - Contact Us
127.0.0.1:5000/static/contact.html
Login
Signup
About Us
Contact Us

Contact Us

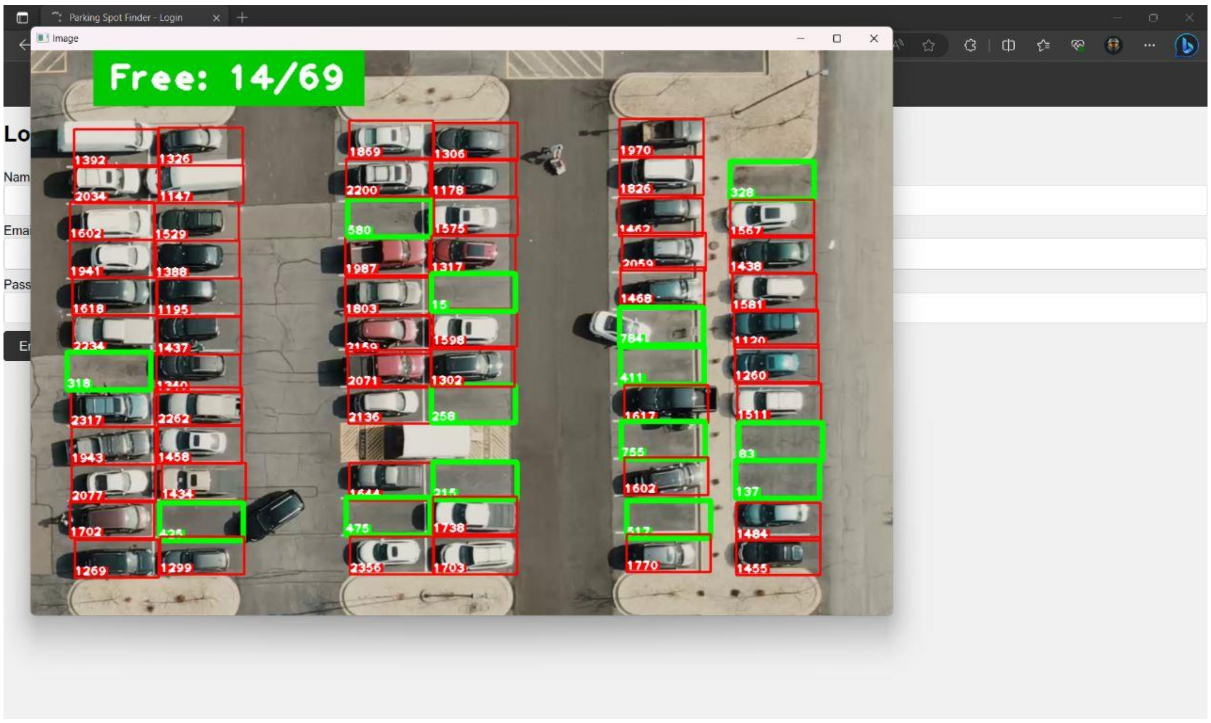
For any questions, suggestions, or support, please feel free to reach out to us:

- tarun.satheesh2020@vitstudent.ac.in
- guriksingh.sachar2020@vitstudent.ac.in
- jaskaran.singh2020@vitstudent.ac.in
- devpiyush.kolsawala2020@vitstudent.ac.in

Login page:



Live feed page:



ADVANTAGES AND DISADVANTAGES:

Advantages:

1. Enhanced Efficiency: Automates parking process, reduces time to find spots, and optimizes space utilization.
2. Improved Accuracy: Accurately detects and tracks vehicles in real-time, minimizing errors.
3. Cost-effective: Reduces labor costs associated with manual parking management.

Disadvantages:

1. Initial Setup and Cost: Requires significant investment in hardware, software, and infrastructure.
2. Technical Complexity: Requires expertise in computer vision and machine learning for development and maintenance.
3. Environmental Limitations: Adverse weather conditions can affect accuracy.

APPLICATIONS:

1. Automated Parking Systems: Real-time detection of available parking spaces to reduce search time.
2. Parking Lot Management: Analyzing occupancy, optimizing layout, and improving management.
3. Traffic Flow Optimization: Directing vehicles to less crowded areas based on parking availability.
4. Enhanced Security: License plate recognition for automated entry control and alerting security.

FUTURE SCOPE:

1. Efficient parking space management: AI algorithms can analyze real-time video feeds, detect available parking spaces, and optimize their allocation.
2. Automated parking guidance: OpenCV can track and identify vehicles, helping drivers find available parking spots through interactive signage or mobile apps.
3. Enhanced security and surveillance: AI and OpenCV can detect suspicious activities in parking lots, improving overall safety and security.
4. Smart payment and billing systems: OpenCV can capture license plate information for automatic entry and exit recognition, enabling seamless payment and transactions.

In summary, the future of AI-enabled car parking using OpenCV lies in efficient space management, automated guidance, improved security, seamless payment systems, integration with autonomous vehicles, and environmental sustainability.

CONCLUSION:

Successful creation of landing page along with login page for the website. Live feed was functional along with the code running in the background to check if free parking spaces are available. Expected storage of credentials in the IBM Database.

APPENDIX:

Source Code:

Main.py:

```
def print_hi(name):
    # Use a breakpoint in the code line below to debug your script.
    print(f'Hi, {name}')  # Press Ctrl+F8 to toggle the breakpoint.
# Press the green button in the gutter to run the script.
if __name__ == '__main__':
    print_hi('PyCharm')
```

App.py:

```
from flask import Flask, render_template, request, session
import cv2
import pickle
import cvzone
import numpy as np
import ibm_db
import re

app = Flask(__name__)
app.secret_key = 'a'
```

```
conn = ibm_db.connect("DATABASE=bludb;HOSTNAME=54a2f15b-5c0f-
46df-8954-
7e38e612c2bd.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud;PORT=32733
;SECURITY=SSL;SSLServerCertificate=DigiCertGlobalRootCA.crt;UID=qxx
23161;PWD=Ssj7NmCCKKboPUvd;", "", "")
print('connected')

@app.route('/')
def project():
    return render_template('index.html')

@app.route('/hero')
def home():
    return render_template('index.html')

@app.route('/model')
def model():
    return render_template('/model.html')

@app.route('/login')
def login():
    return render_template('login.html')

@app.route("/reg", methods=['POST', 'GET'])
def signup():
    msg = ''
    if request.method == 'POST':
        name = request.form["name"]
        email = request.form["email"]
        password = request.form["password"]
        sql = "SELECT * FROM REGISTER WHERE name= ?"
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt, 1, name)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)
        print(account)
        if account:
            return render_template('login.html', error=True)
        elif not re.match(r'[^@]+@[^@]+\.[^@]+', email):
            msg = "Invalid Email Address"
        else:
            insert_sql = "INSERT INTO REGISTER VALUES (?,?,?)"
            prep_stmt = ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(prep_stmt, 1, name)
            ibm_db.bind_param(prep_stmt, 2, email)
            ibm_db.bind_param(prep_stmt, 3, password)
            ibm_db.execute(prep_stmt)
            msg = "You have successfully registered!"

    return render_template('login.html', msg=msg)

@app.route("/log", methods=['POST', 'GET'])
def login1():
    if request.method == "POST":
        email = request.form["email"]
        password = request.form["password"]
        sql = "SELECT * FROM REGISTER WHERE EMAIL=? AND
PASSWORD=?"
```

```
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt, 1, email)
        ibm_db.bind_param(stmt, 2, password)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)
        print(account)
        if account:
            session['Loggedin'] = True
            session['id'] = account['EMAIL']
            session['email'] = account['EMAIL']
            return render_template('model.html')
        else:
            msg = "Incorrect Email/password"
            return render_template('login.html', msg=msg)
    else:
        return render_template('login.html')

@app.route('/predict_live')
def liv_pred():
    # Video feed
    cap = cv2.VideoCapture('carPark.mp4')

    with open('CarParkPos', 'rb') as f:
        posList = pickle.load(f)

    width, height = 107, 48

    def checkParkingSpace(imgPro):
        spaceCounter = 0
        for pos in posList:
            x, y = pos

            imgCrop = imgPro[y:y + height, x:x + width]
            # cv2.imshow(str(x * y), imgCrop)
            count = cv2.countNonZero(imgCrop)

            if count < 900:
                color = (0, 255, 0)
                thickness = 5
                spaceCounter += 1
            else:
                color = (0, 0, 255)
                thickness = 2

            cv2.rectangle(img, pos, (pos[0] + width, pos[1] + height), color,
thickness)
            cvzone.putTextRect(img, str(count), (x, y + height - 3), scale=1,
                               thickness=2, offset=0, colorR=color)

        cvzone.putTextRect(img, f'Free: {spaceCounter}/{len(posList)}', (100, 50),
scale=3,
                           thickness=5, offset=20, colorR=(0, 200, 0))

    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()
        imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        imgBlur = cv2.GaussianBlur(imgGray, (3, 3), 1)
        imgThreshold = cv2.adaptiveThreshold(imgBlur, 255,
cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
                                             cv2.THRESH_BINARY_INV, 25, 16)
        imgMedian = cv2.medianBlur(imgThreshold, 5)
        kernel = np.ones((3, 3), np.uint8)
        imgDilate = cv2.dilate(imgMedian, kernel, iterations=1)

        checkParkingSpace(imgDilate)
        cv2.imshow("Image", img)
        # cv2.imshow("ImageBlur", imgBlur)
        # cv2.imshow("ImageThres", imgMedian)
        cv2.waitKey(10)

if __name__ == "__main__":
    app.run(debug=True)
```

Index.html:

```
<!DOCTYPE html>
<html>
<head>
    <title>Parking Spot Finder</title>
     <link rel="stylesheet" type="text/css" href="{{
url_for('static',filename='styles/styles.css') }}">
</head>
<body>
```

```
    <nav>
        <ul>
            <li><a href="/login">Login</a></li>
             <li><a href="/reg">Signup</a></li>
             <li><a href="{{  url_for('static', filename='about.html') }}">About
Us</a></li>
             <li><a href="{{  url_for('static', filename='contact.html') }}">Contact
Us</a></li>
        </ul>
    </nav>

    <h1>Welcome to Parking Spot Finder</h1>

    <p>This website uses AI and OpenCV to find available parking spots in real-
time. Whether you're looking for parking in a crowded city or at a busy event,
we've got you covered. Sign up or login to get started!</p>

</body>
</html>
```

Login.html:

```
<!DOCTYPE html>
<html>
<head>
    <title>Parking Spot Finder - Login</title>
    <link rel="stylesheet" type="text/css" href="{{
url_for('static',filename='styles/styles.css') }}">
</head>
<body>
```

```
    <nav>
      <ul>

        <li><a href="/login">Login</a></li>
        <li><a href="/reg">Signup</a></li>
        <li><a href="{{  url_for('static', filename='about.html') }}">About
Us</a></li>
        <li><a href="{{  url_for('static', filename='contact.html') }}">Contact
Us</a></li>
      </ul>
    </nav>

    <h1>Login</h1>

     <form>
      <div class="form-group">
        <label for="name">Name:</label>
        <input type="text" id="name" name="name" required>
      </div>
      <div class="form-group">
        <label for="email">Email:</label>
        <input type="email" id="email" name="email" required>
      </div>
      <div class="form-group">
        <label for="password">Password:</label>
        <input type="password" id="password" name="password" required>
      </div>
      <button type="submit" class="btn">Enter</button>
      <a href="/predict_live">Enter Live View</a>
    </form>

</body>
</html>
```

Signup.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Parking Spot Finder - Signup</title>
     <link rel="stylesheet" type="text/css" href="{{
url_for('static',filename='styles/styles.css') }}">
</head>
<body>
    <nav>
        <ul>
            <li><a href="/login">Login</a></li>
            <li><a href="/reg">Signup</a></li>
            <li><a href="{{  url_for('static', filename='about.html') }}">About
Us</a></li>
            <li><a href="{{  url_for('static', filename='contact.html') }}">Contact
Us</a></li>
        </ul>
    </nav>

    <h1>Signup</h1>

   <form>
```

```
        <div class="form-group">
          <label for="name">Name:</label>
          <input type="text" id="name" name="name" required>
        </div>
        <div class="form-group">
          <label for="email">Email:</label>
          <input type="email" id="email" name="email" required>
        </div>
        <div class="form-group">
          <label for="password">Password:</label>
          <input type="password" id="password" name="password" required>
        </div>
        <button type="submit" class="btn">Enter</button>
        <a href="/predict_live">Enter Live View</a>
      </form>
  </body>
  </html>
```