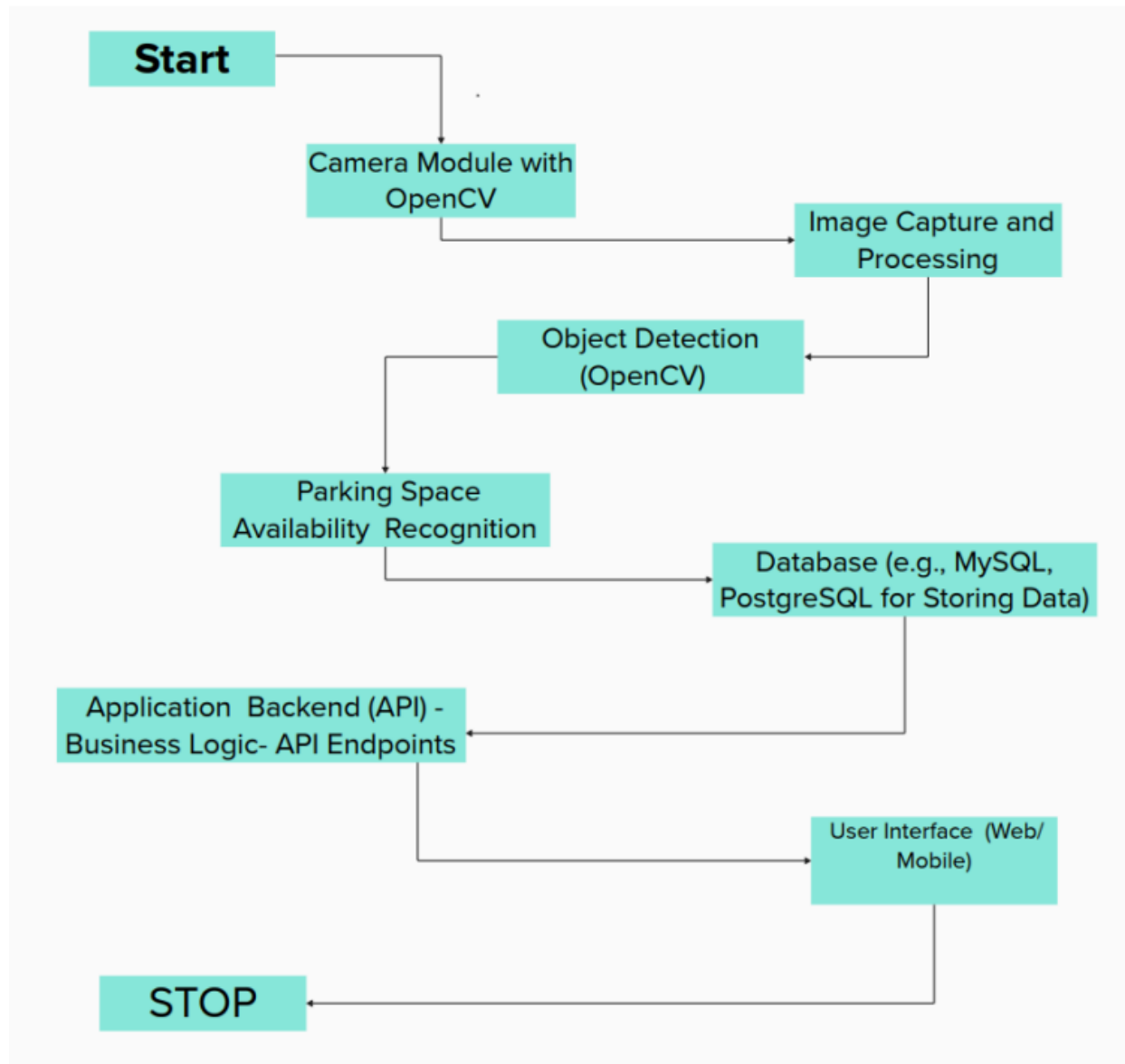


Team Id:592031

## AI Enable car parking using OpenCV

In crowded cities, parking is a regular issue for vehicles. Consider the following scenario: you are traveling during rush hour to a retail center. You see that the parking lot is filled as you get closer to the mall and that additional cars are circling the area in search of spaces. In the hopes of finding a place quickly, you join the queue of automobiles. Still, as time goes on, you discover that there are too many people in the parking lot and that it's getting harder and harder to locate a space. Knowing that you could miss a fantastic shopping opportunity or be late for your appointment causes you to get agitated and nervous. An initiative based on computer vision called AI-enabled automobile parking using OpenCV seeks to automate the parking procedure. The project entails creating an intelligent system that can count the number of available parking places and find unoccupied ones. The OpenCV (Open Source Computer Vision) framework and a camera are used by the system to record live video of the parking lot.

## Architecture:



## Pre-requisites:

1. To complete this project you must have the following software versions and packages.

- PyCharm ( Download: <https://www.jetbrains.com/pycharm/> )
- Python 3.7.0 (Download: <https://www.python.org/downloads/release/python-370/> )

- Numpy
- cvzone

2. To make a responsive python script you must require the following packages.

Flask:

- Web framework used for building web applications.
- Flask Basics: [Click here](#)

If you are using anaconda navigator, follow below steps to download required packages:

- Open anaconda prompt.
- Type "pip install opencv-python" and click enter.
- Type "pip install cvzone" and click enter.
- Type "pip install Flask" and click enter.

If you are using Pycharm IDE, you can install the packages through the command prompt and follow the same syntax as above.

### **Project objectives:**

By the end of this project you will:

- Know fundamental concepts and techniques of computer vision (openCV).
- Gain a broad understanding of image thresholding.

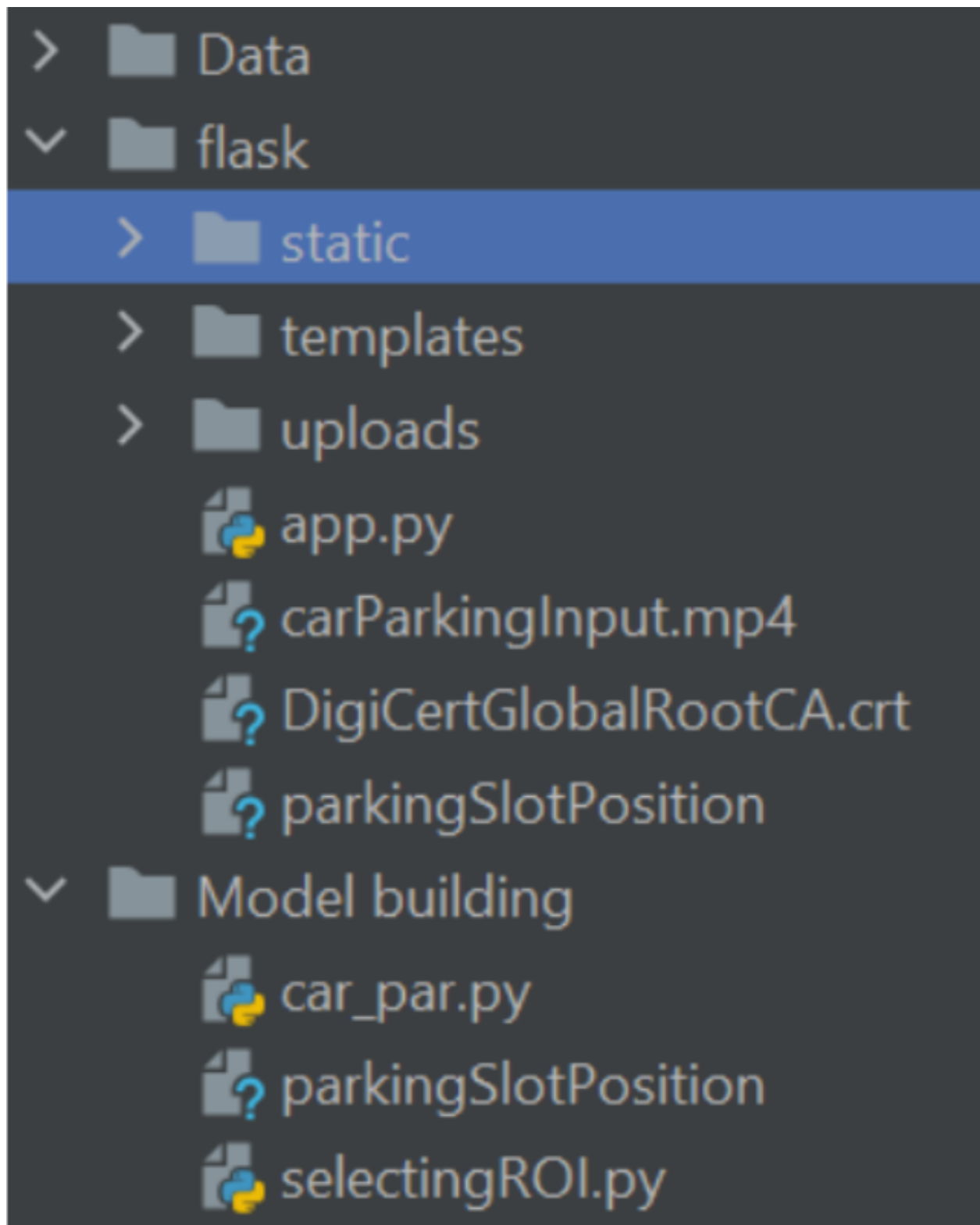
### **Project Flow:**

- Data Collection
  - o Download the dataset
- ROI (Region of interest)
  - o Create python file
  - o Import required libraries
  - o Define ROI width and height

- o Select and deselect ROI
- o Denote ROI with BBOX
- Video Processing and object detection
- o Import required libraries
- o Reading input and loading ROI file
- o Checking for parking space
- o Looping the video
- o Frame processing and empty parking slot counters
- IBM Database Connection
- o Create IBM DB2 service and table
- Application building
- o Build HTML
- o Build python script for Flask

### **Project Structure:**

Create a project folder which contains files as shown below:



- The Dataset folder contains the video and image file

- For building a Flask Application we need HTML pages stored in the templates folder, CSS for styling the pages stored in the static folder and a python script app.py for server side scripting
- Model Building folder consists of car.py & selectingROI.py python script

## **Milestone – 1: Data Collection**

Activity 1: Download the dataset

Click the link below to download the dataset for AI Enabled Car Parking using OpenCV. [Clickhere](#)

.

## **Milestone – 2: ROI (Region of interest)**

ROI stands for Region of Interest, which refers to a specific rectangular portion of an image or video frame that is used for processing or analysis.

Activity 1: Create a python file

Create a python file in the directory and name it as 'selectingROI.py'. This python file is used for creating ROI and deleting ROI.

Activity 2: Importing the required packages

- Opencv: OpenCV is a great tool for image processing and performing computer vision tasks. It is an open-source library that can be used to perform tasks like face detection, object tracking, landmark detection, and much more. It supports multiple languages including python, java C++.

- Pickle: The pickle library in Python is used for serialization and deserialization of Python objects. Serialization is the process of converting a Python object into a stream of bytes that can be stored in a file or sent over a network. De-serialization is the process of converting the serialized data back into a Python object.

```
import cv2
import pickle
```

### Activity 3: Define ROI width and height

- Calculating the ROI width and height (manually width and height are calculated and given as 107 & 48).
- An empty file parkingSlotPosition is created to save all the ROI values. Try and except combo is used.
- In Python, try and except are used for error handling, to catch and handle exceptions that may occur during program execution. The try block is used to enclose the code that may raise an exception, and the except block is used to define what should happen if an exception is raised.

```
#Define the width and height of ROI
width, height = 107, 48
# Creating an empty file and Loading to a variable & Creating an empty list
try:
    with open('parkingSlotPosition', 'rb') as f:
        posList = pickle.load(f)
except:
    posList = []
```

#### Activity 4: Select and deselect ROI

- A function is defined as `mouseClick`. As parameters we are passing events (`mouseaction`), `x` (ROI starting point), `y` (ROI ending point), `flags` (Boolean flag) and `params`(other parameters).
- In 1st if condition: After left click from the mouse, the starting and ending points will be added to `posList` by `append` method.
- In 2nd if condition: If ROI is selected in an unwanted region. Then we can remove that unwanted ROI by right clicking from the mouse.
- Python object are converted into a stream of bytes and stored in the `parkingSlotPosition` file.

```
def mouseClick(events, x, y, flags, params):  
    if events == cv2.EVENT_LBUTTONDOWN:  
        posList.append((x, y))  
    # Removing unwanted ROI from posList  
    if events == cv2.EVENT_RBUTTONDOWN:  
        for i, pos in enumerate (posList):  
            x1, y1 = pos  
            if x1 < x < x1 + width and y1 < y < y1 + height:  
                posList.pop(i)  
    # Saving the postist values to parkingSlotPosition file  
    with open('parkingSlotPosition', 'wb') as f:  
        pickle.dump(posList, f)
```

#### Activity 5: Denote ROI with BBOX

- Reading the image with `imread()` method from `cv2`.
- All ROIs are saved in `posList` (refer activity 4).
- The rectangle is created as BBOX with the starting value (`x`) and ending value (`y`) from `posList` by `rectangle()` method. As the parameters give image source, starting value, ending value, (starting value + width, ending value + height), (color) and thickness.
- For displaying the image `imshow()` method is used.



- setMouseCallback() function is used to perform some action on listening to the event which we have defined in activity 4.

```
while True:
    img = cv2.imread("D:\programs\Python programs\smart internz\carParkImg.png")
    for pos in posList:
        cv2.rectangle(img, pos, (pos[0] + width, pos [1] + height), (255, 255, 255), 2)
    cv2.imshow("Image", img)
    cv2.setMouseCallback("Image", mouseClicked)
    cv2.waitKey(1)
```

### Milestone - 3: Video processing and Object detection

Create a new python file to perform video processing, object detection and counters.

Activity 1: Import the required packages

Importing the required libraries to new python file.

```
import cv2
import pickle
import cvzone
import numpy as np
```

Activity 2: Reading input and loading ROI file

- The VideoCapture() method from cv2 is used to capture the video input. Download the input video from milestone 1.
- Load the parkingSlotPosition file by load() method from pickle library. The parkingSlotPosition file is created from milestone 2. The ROI values are presented in the parkingSlotPosition file.
- Define width and height which we have used in milestone 2.

```

#Video feed
cap= cv2.VideoCapture("D:\programs\Python programs\smart internz\carParkingInput.mp4")
#Loading the ROI from parkingSlotPosition file
with open('parkingSlotPosition', 'rb') as f:
    posList = pickle.load(f)
#Define width and height
width, height = 107, 48

```

### Activity 3: Checking for parking space

- The function is defined with the name as carParkingSpace. As a parameter image has to be passed.
- Taking the position from the position list and saved to variable x and y.
- Cropping the image based on ROI (x and y value).
- To count the pixels from ROI, countNonZero() method is used from cv2. The BBOX (rectangle) is drawn in green color if the pixel count is lesser than 900 else it is drawn in red color.
- The count of green color is displayed in the frame by the putTextRect() method from cvzone library.

```

def checkParkingSpace (imgPro):
    spaceCounter = 0
    for pos in posList:
        x, y = pos
        # Crop the image based on ROI
        imgCrop= imgPro[y:y + height, x:x + width]
        # Counting the pixel values from cropped image
        count = cv2.countNonZero (imgCrop)
        if count < 900:
            color = (0, 255, 0)
            thickness = 5
            spaceCounter += 1
        else:
            color= (0, 0, 255)
            thickness = 2
        # Draw the rectangle based on the condition defined above
        cv2.rectangle(img, pos, (pos[0]+ width, pos[1]+ height), color, thickness)
    # Display the available parking slot count / total parking slot count
    cvzone.putTextRect(img, f'Free: {spaceCounter}/{len(posList)}', (100, 50), scale=3, thickness= 5, offset=20, color=(0,200,0))

```

#### Activity 4: Looping the video

- Current frame position is compared with total number of frames. If it reaches the maximum frame, again the frame is set to zero. So, continuously it'll loop the frame.
- `cv2.CAP_PROP_POS_FRAMES` = Current frame
- `cv2.CAP_PROP_FRAME_COUNT` = Total no. of frame

```
while True:  
    # Looping the video  
    if cap.get(cv2.CAP_PROP_POS_FRAMES) == cap.get(cv2.CAP_PROP_FRAME_COUNT):  
        cap.set(cv2.CAP_PROP_POS_FRAMES, 0)
```

#### Activity 5: Frame Processing and empty parking slot counters

- The captured video has to be read frame by frame. The `read()` method is used to read the frames.
- OpenCV reads the frame in BGR (Blue Green Red).
- Converting BGR frame to gray scale image. And the gray scale image is blurred with `GaussianBlur()` method from `cv2`.
- The `adaptiveThreshold()` method is used to apply threshold to the blurred image. And again blur is applied to the image. [Click here](#) to understand about `adaptiveThreshold()`.
- The `dilate()` method is used to computing the minimum pixel value by overlapping the kernel over the input image. The blurred image after thresholding and kernel are passed as the parameters.
- The `checkParkingSpace` function is called with dilated image. As per the activity 3 we will get the count of empty parking slot.

- Display the frames in form of video. The frame will wait for 10 seconds and it'll go to next frame.

```
while True:
    # Looping the video
    if cap.get(cv2.CAP_PROP_POS_FRAMES) == cap.get(cv2.CAP_PROP_FRAME_COUNT):
        cap.set(cv2.CAP_PROP_POS_FRAMES, 0)
    # Reading frame by frame from video
    success, img = cap.read()
    # Converting to gray scale image
    imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    imgBlur = cv2.GaussianBlur(imgGray, (3, 3), 1) # Applying blur to image
    # Applying threshold to the image
    imgThreshold = cv2.adaptiveThreshold(imgBlur, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY_INV, 25, 16)
    imgMedian = cv2.medianBlur(imgThreshold, 5) # Applying blur to image
    kernel = np.ones((3, 3), np.uint8)
    imgDilate = cv2.dilate(imgMedian, kernel, iterations=1)
    # Passing dilate image to the function
    checkParkingSpace(imgDilate)
    cv2.imshow("Image", img)
    cv2.waitKey(10)
```

## Milestone - 4: IBM database connection

Activity1: Create IBM DB2 service & table:

- For creating DB2 service and table, kindly refer the below video.
- Link: [https://drive.google.com/file/d/16eOJGfydVN3oWkmyxF559d3oZDptlsm7/view?usp=share\\_link](https://drive.google.com/file/d/16eOJGfydVN3oWkmyxF559d3oZDptlsm7/view?usp=share_link)

## Milestone - 5: Application building

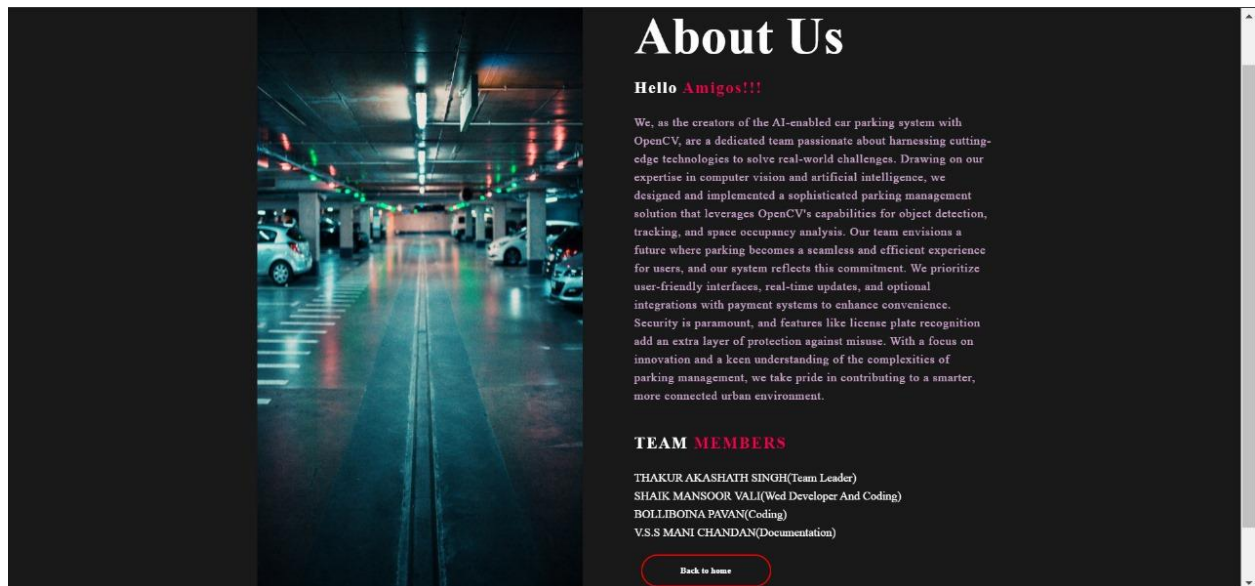
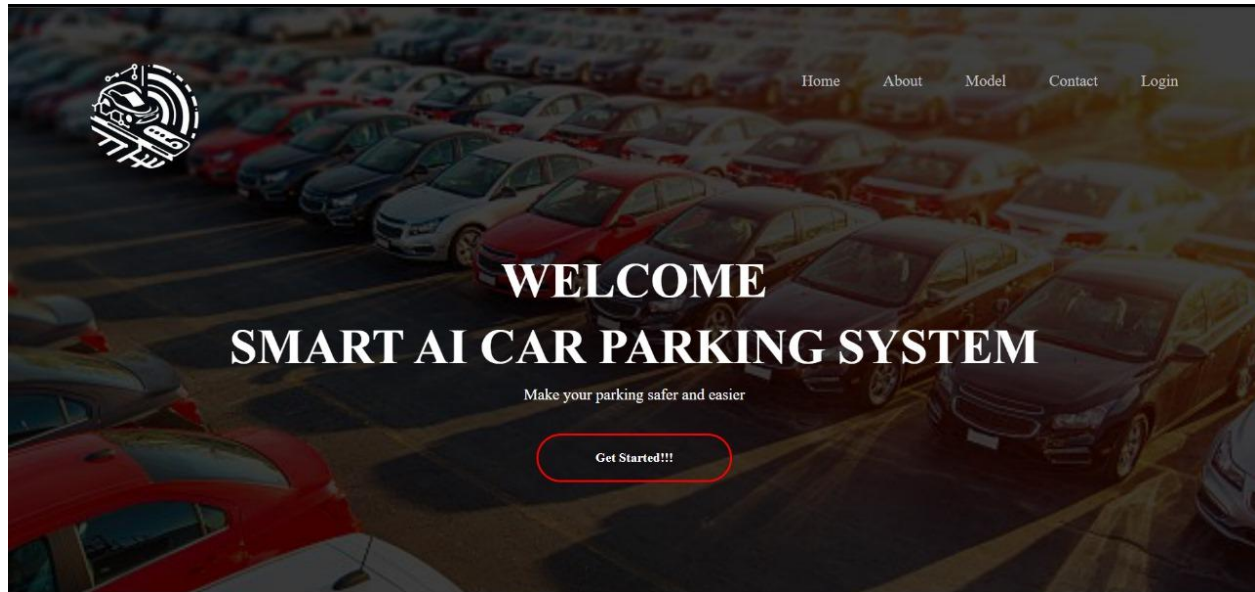
In this section, we will be building a web application that is integrated into the model we built. A UI is provided for the users where he/she has to navigate to detect button. Then the video will be showcased on the UI.

This section has the following tasks

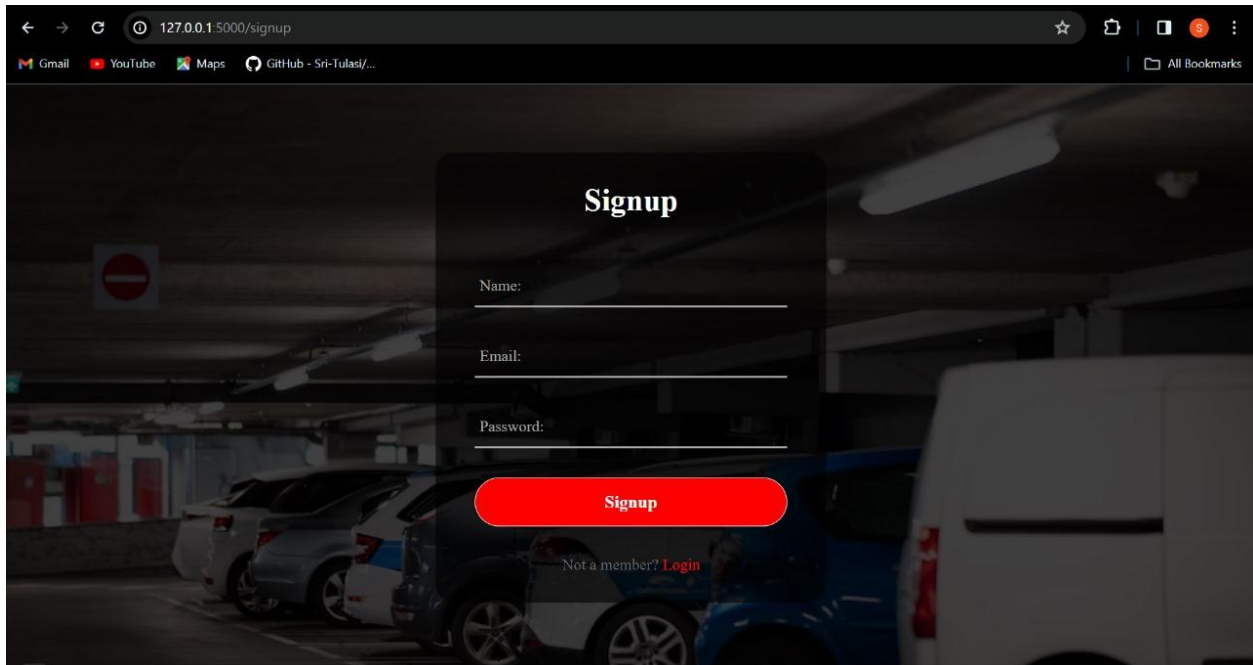
- Building HTML Pages
- Building server-side script

## Activity1: Building Html Pages:

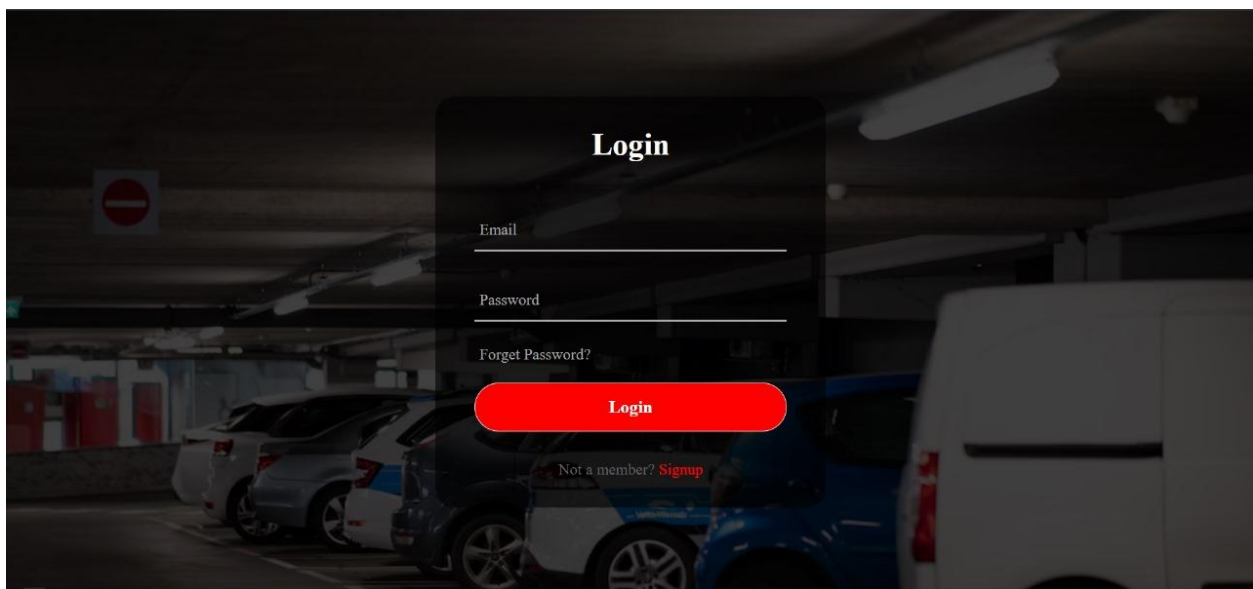
For this project we have created 2 HTML files and saved them in the templates folder. Let's see how those html pages looks like:







A screenshot of a web browser window. The address bar shows the URL "127.0.0.1:5000/signup". The browser's bookmark bar includes links to Gmail, YouTube, Maps, and GitHub. The main content area features a dark, semi-transparent overlay with the title "Signup" in white. Below the title are three input fields labeled "Name:", "Email:", and "Password:". A prominent red button with the text "Signup" is positioned below the fields. At the bottom of the overlay, the text "Not a member? [Login](#)" is displayed, with "Login" in red.



A screenshot of a web browser window showing a "Login" form overlay. The background is a dimly lit parking garage with several cars parked. The overlay is dark and semi-transparent, with the title "Login" in white. It contains two input fields labeled "Email" and "Password". Below these fields is a link that says "Forgot Password?". A red button with the text "Login" is located at the bottom of the form. At the very bottom of the overlay, the text "Not a member? [Signup](#)" is shown, with "Signup" in red.

Activity 2: Build Python code:

- Import the libraries

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

- Importing the flask module into the project is mandatory. An object of the Flask class is our WSGI application. Flask constructor takes the name of the current module (`__name__`) as an argument.

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

- From `ibm_db` package, `connect()` method is used to connect DB service from IBM to Flask framework.
- To understand about IBM DB, refer the link <https://www.ibm.com/docs/en/db2/11.5?topic=framework-application-development-db>

```
conn= ibm_db.connect("DATABASE=vali3;HOST=localhost;PORT=25000;PROTOCOL=TCPIP;UID=
print("connected")
```

Render HTML page:

- Here we will be using the declared constructor to route to the HTML page that we have created earlier. In the above example, the `/` URL is bound with the `index.html` function. Hence, when the home page of the web server is opened in the browser, the HTML page will be rendered.

```
@app.route('/')
def project():
    return render_template('Login.html')
@app.route('/login')
def login():
    return render_template('Login.html')
@app.route('/signup')
def signup1():
    return render_template('Signup.html')
@app.route('/index')
def index():
    return render_template('Index.html')
@app.route('/home')
def Home():
    return render_template('Index.html')
@app.route('/model')
def model():
    return render_template('Model.html')
@app.route('/about')
def about():
    return render_template('About.html')
@app.route('/contact')
def contact():
    return render_template('Contact.html')
```

- Fetching user name, email and password from web page and passing those values to the register table which we have created in IBM DB with SQL query.



```

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'^[^\s@]+@[^\s@]+\.[^\s@]+', email):
            msg = "Invalid Email Address!"
        else:
            insert_sql = "INSERT INTO REGISTER VALUES (?, ?, ?)"
            prep_stmt = ibm_db.prepare(conn, insert_sql)
            # this username & password should be same as db-2 details & order also
            ibm_db.bind_param(prepare_stmt, 1, name)
            ibm_db.bind_param(prepare_stmt, 2, email)
            ibm_db.bind_param(prepare_stmt, 3, password)
            ibm_db.execute(prepare_stmt)
            msg = "You have successfully registered !"
            return render_template('login.html', msg=msg)

```

- Fetching email and password from login web page and matching with our register table. If email and password are matched, it'll render to next template. If its not matched we'll get an alert in login web page as Incorrect email/password.

```

@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=?" # from db2 sql table
        stmt = ibm_db.prepare(conn, sql)
        # this username & password should be same as db-2 details & order also
        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')

```

- Create a decorator to route to '/predict\_live'. Go to the milestone 3 and copy the entire code and paste it inside the function liv\_pred.

```

@app.route('/predict_live')
def liv_pred():
    # Video feed
    cap = cv2.VideoCapture('carParkingInput.mp4')
    with open('parkingSlotPosition', 'rb') as f:
        posList = pickle.load(f)
    width, height = 107, 48
    def checkParkingSpace (imgPro):
        spaceCounter = 0
        for pos in posList:
            x, y = pos
            imgCrop = img Pro[y:y + height, x:x + width]
            # cv2.imshow(str(x * y), imgCrop)
            count = cv2.count NonZero (imgCrop)
            if count < 900:
                color = (0, 255, 0)
                thickness = 5
                spaceCounter += 1
            else:
                color = (0, 255, 0)
                thickness = 2
            cv2.rectangle(img, pos, (pos[0] + width, pos[1] + height), color, thickness)
        cvzone.putTextRect(img, f'Free: {spaceCounter}/{len(posList)}' (100, 50), scale=3, thickness=5, offset=20, colorR=(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)
            if cv2.waitKey(1) & 0xFF== ord('q'):
                break

```

- Main Function: Used to run the current module.

```

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

```

### Activity 3: Run the application

- Open the anaconda prompt from the start menu
- Navigate to the folder where your python script is.
- Now type the “python app.py” command
- Navigate to the localhost where you can view your web page.
- Click on the predict button from the top right corner, enter the inputs, click on the submit button, and see the result/prediction on the web.

```
AI car Parking  Version control  app  Run  app
liv_pred() > while True
Run  app
"D:\programs\Python programs\smart internz\AI car Parking\venv\Scripts\python.exe" "D:\programs\Python programs\smart internz\AI car Parking\fla:
connected
* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
connected
* Debugger is active!
* Debugger PIN: 843-634-263
127.0.0.1 - - [22/Nov/2023 16:14:05] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [22/Nov/2023 16:14:05] "GET /static/Style.css HTTP/1.1" 304 -
127.0.0.1 - - [22/Nov/2023 16:14:05] "GET /static/car2.jpg HTTP/1.1" 304 -
127.0.0.1 - - [22/Nov/2023 16:14:21] "POST /log HTTP/1.1" 200 -
{'NAME': 'vali', 'EMAIL': 'vali@gmail.com', 'PASSWORD': 'vali'}
127.0.0.1 - - [22/Nov/2023 16:14:21] "GET /static/Style.css HTTP/1.1" 304 -
127.0.0.1 - - [22/Nov/2023 16:14:21] "GET /static/Logo.png HTTP/1.1" 304 -
127.0.0.1 - - [22/Nov/2023 16:14:21] "GET /static/car.jpg HTTP/1.1" 304 -
AI car Parking  flask  app.py  8:32  LF  UTF-8  4 spaces  Python 3.11 (AI car Parking)
```

Output:

