CAR PARKING MANAGEMENT USING YOLO

A project submitted in partial fulfilment of the requirements for the degree of

B.Sc. INFORMATION TECHNOLOGY

SUBMITTED BY

U. GNANAMBIKA-22ITA12

UNDER THE GUIDANCE OF

Mrs. S. SUMATHI, M.S(IT&M), M.Sc.(CS), M.Phil., DGT

DEPARTMENT OF INFORMATION TECHNOLOGY

E.M.G. YADAVA WOMEN'S COLLEGE MADURAI-625014

(An Autonomous Institution-Affiliated to Madurai Kamaraj University)

(Re-accredited (3rd Cycle) with Grade A+ and CGPA3.51 by NAAC)

2022-2025

E.M.G. YADAVA WOMEN'S COLLEGE MADURAI-625014

(An Autonomous Institution-Affiliated to Madurai Kamaraj University)

(Re-accredited ((3rd Cycle) with Grade A+ and CGPA3.51 by NAAC)

DEPARTMENT OF INFORMATION TECHNOLOGY

BONAFIDE CERTIFICATE

This is to certify that the project titled **"CAR PARKING MANAGEMENT USING YOLO"** using Python has been carried by **U. GNANAMBIKA (22ITA12),** and submitted to the department of B.Sc. (Information Technology), E.M.G. Yadava Women's college, Madurai, in partial fulfilment of the requirements for the award of Bachelor of Information Technology during the period of 2022-2025.

Submitted for the Viva-Voice exam held at E.M.G Yadava Women's College, Madurai-14 on…….

HEAD OF THE DEPARTMENT

INTERNAL EXAMINER **EXTERNAL EXAMINER**

DECLARATION

We hereby declare that the project work entitled **"CAR PARKING MANAGEMENT USING YOLO"** submitted to the department of **B.Sc.(Information Technology), E.M.G. Yadava Women's college** for the award of Bachelor of Information Technology is a record of an original work done by us under the guidance of **Mrs. S. SUMATHI, M.S(IT&M), M.Sc.(CS), M.Phil., DGT, Assistant Professor,** E.M.G.YADAVA WOMEN'S COLLEGE, Madurai and this project work have not performed the basic for the award of any Degree or diploma/associateship/fellowship and similar project if any.

Signature of the candidate 1

Place:

Date:

ACKNOWLEDGEMENT

I heartily express our sincere thanks to **Dr.(Mrs.)V.PUSHPALATHA, M.Com., H.S.M., M.Phil., Ph.D.,** The Principal i/c & Head, Department of Commerce, E.M.G. Yadava Women's College, and **Mrs. R. BOOMADEVI, M.Com., M.Phil., DGT,** Head, Department of Information Technology, for providing us such a nice opportunity.

I would like to acknowledge our hearty thanks to **Mrs. S. SUMATHI, M.S(IT&M), M.Sc.(CS), M.Phil., DGT,** who was the guide for doing our project.

I express our deep Gratitude to project guide **ELYSIUM TECHNOLOGIES Pvt. Ltd., Madurai.**

I heartily express our sincere thanks to our college for providing us with the facilities to undergo this project work.

Above all, I Thank God. The **ALMIGHTY** and my beloved parents, without their abundant and grace, we would not have completed this project work.

CAR PARKING MANAGEMENT USING YOLO

CONTENTS

CONTENTS

ABSTRACT

ABSTRACT

Efficient management of parking spaces is a critical challenge in urban areas, often leading to significant traffic congestion and wasted time as drivers search for available parking spots. The Car Parking Management System leverages YOLO (You Only Look Once), a state-of-the-art object detection algorithm, to provide a real-time solution for monitoring and managing parking spaces. This system incorporates YOLO to detect parking slots on the admin side and OCR (Optical Character Recognition) to extract vehicle number plates on the user side. The proposed system begins with the collection and preprocessing of a comprehensive dataset of parking lot images, labeled to identify individual parking spaces and their status (occupied or available).

Once trained, the YOLO model is deployed to analyze live video feeds from cameras installed in the parking lot on the admin side. The system continuously processes these video streams, accurately identifying and updating the status of each parking space. To ensure detection accuracy, advanced techniques like non-maximal suppression are employed to eliminate redundant detections. For the user side, an OCR system is integrated to extract vehicle number plates from the camera feeds. This helps in tracking vehicles and managing parking reservations. A user-friendly interface is developed to provide real-time information on parking availability, accessible via web and mobile applications. This interface offers users a convenient way to check for available parking spaces, includes features such as navigation assistance, and allows for number plate recognition to enhance security and efficiency.

The performance of the system is evaluated using metrics like mean average precision (mAP) and intersection over union (IoU). Continuous monitoring and optimization are implemented to identify areas for improvement and ensure the system's accuracy and efficiency. Real-world testing and validation are conducted to guarantee the system's reliability in diverse scenarios. The Car Parking Management System using YOLO and OCR represents a significant advancement in urban infrastructure management. By providing real-time information on parking availability and integrating number plate recognition, the system reduces the time drivers spend searching for parking, enhances security, alleviates traffic congestion, and improves the overall user experience. The integration of advanced computer vision techniques and user-friendly interfaces makes this system a valuable tool for smart city initiatives and sustainable urban development.

INTRODUCTION

1.INTRODUCTION

1.1 Project Introduction

The rapid urbanization and increasing number of vehicles have led to a significant rise in the demand for efficient parking management solutions. Traditional methods of parking management often rely on manual inspections or outdated technologies, resulting in inefficient utilization of parking spaces and increased frustration for drivers. The Car Parking Management System utilizing YOLO (You Only Look Once) and OCR (Optical Character Recognition) aims to address these challenges by providing a real-time, automated solution for monitoring and managing parking spaces.

This project leverages YOLO, a state-of-the-art object detection algorithm, to detect parking slots on the admin side. The YOLO model is trained to accurately identify and monitor the status of each parking space in real-time, using live video feeds from cameras installed in the parking lot. By continuously updating the status of parking spaces, the system ensures optimal utilization and reduces the time drivers spend searching for available spots.

On the user side, the system integrates OCR technology to extract vehicle number plates from camera feeds. This feature enhances security by tracking vehicles entering and exiting the parking lot, and facilitates parking reservations. The user-friendly interface provides real-time information on parking availability and offers additional features such as navigation assistance and number plate recognition, making the parking experience more convenient and efficient.

The Car Parking Management System using YOLO and OCR represents a significant advancement in urban infrastructure management. By combining advanced computer vision techniques with user-friendly interfaces, the system aims to streamline parking operations, alleviate traffic congestion, and contribute to the development of smarter, more sustainable urban environments.

1.2 Objective

The primary goal of this project, "Car Parking Management Using YOLO," is to develop an automated, intelligent, and efficient parking management system using deep learning and computer vision. The key objectives of the system are:

- **Automated Parking Slot Detection**:
  - Implement YOLO (You Only Look Once) to detect occupied and empty parking slots in real-time using camera feeds or uploaded images/videos.
- **Real-Time Monitoring and Visualization:**
  - Develop a Streamlit-based web application for users to view real-time parking availability and slot status through an interactive dashboard.
- **Automatic License Plate Recognition (ALPR):**
  - Integrate YOLO-based license plate detection and OCR (Optical Character Recognition) to extract and store vehicle registration numbers for security and logging purposes.
- **Parking Slot Reservation System:**
  - Enable users to book parking slots online through the web interface, preventing conflicts and optimizing space utilization.
- **Database Management and Logging:**
  - Use SQLite to maintain records of vehicle entries, exits, timestamps, and parking history for efficient data tracking and management.
- **Manual Slot Marking for Enhanced Accuracy:**
  - Allow users to manually define and adjust parking slot boundaries, ensuring flexibility in detection and reducing false positives.
- **Improved Traffic Flow and Reduced Congestion:**
  - Minimize parking-related traffic congestion by providing accurate and real-time information on slot availability, reducing time spent searching for parking.

- **Cost-Effective and Scalable Solution:**
    - Replace expensive hardware-based systems (e.g., RFID, sensors) with a vision-based approach that works with existing CCTV infrastructure.
- **Security and Anti-Fraud Measures:**
    - Prevent unauthorized parking by cross-verifying license plates against registered bookings and flagging discrepancies.
- **User-Friendly and Scalable Implementation:**
    - Ensure that the system is easy to use, deployable in multiple locations, and adaptable for various parking environments such as malls, offices, hospitals, and public parking areas.
    - By achieving these objectives, the project aims to enhance parking efficiency, reduce human intervention, and improve overall urban mobility.

1.3 Module Description

- Data Collection and Preprocessing Module
- YOLO Model Training Module
- Real-time Detection Module
- OCR Integration Module
- User Interface Module
- Performance Evaluation and Optimization Module

1. Data Collection and Preprocessing Module

Description:

This module is foundational to the system as it involves gathering and preparing the dataset used for training the YOLO model. The dataset consists of images of various parking lots, captured from different angles and under different lighting conditions.

Key Features:

- **Image Collection:** High-resolution images of parking lots are captured using cameras placed at strategic locations. The images should cover a diverse range of conditions to ensure the robustness of the model.
- **Labeling:** Each image is annotated to mark the parking spaces and their status (occupied or available). This involves drawing bounding boxes around each parking space and assigning a label.
- **Preprocessing:** Techniques such as image normalization, resizing, and augmentation (e.g., rotation, flipping, brightness adjustments) are applied to enhance the dataset and improve the model's ability to generalize.

Technical Details:

- **Tools and Libraries:** OpenCV for image processing, LabelImg for annotation, and custom scripts for augmentation.
- **Storage:** The dataset is stored in a structured format, with images and corresponding annotation files.

2. YOLO Model Training Module

Description:

This module focuses on training the YOLO object detection model using the preprocessed dataset. YOLO is chosen for its high speed and accuracy in real-time object detection.

Key Features:

- **Training Process:** The labeled dataset is used to train the YOLO model. The training involves feeding the images and annotations into the model, allowing it to learn the features that distinguish occupied and available parking spaces.
- **Hyperparameter Tuning:** Key hyperparameters such as learning rate, batch size, and the number of epochs are tuned to optimize the model's performance. Techniques like early stopping and learning rate scheduling are used to prevent overfitting.

- **Evaluation:** The model's performance is evaluated using metrics such as mean average precision (mAP) and intersection over union (IoU). These metrics provide insights into the model's accuracy and precision.

Technical Details:

- **Framework:** TensorFlow or Darknet.
- **Hardware:** High-performance GPUs for faster training.
- **Data Split:** The dataset is split into training, validation, and test sets to assess the model's performance.

3. Real-time Detection Module

Description:

This module deploys the trained YOLO model for real-time detection of parking spaces. It continuously processes live video feeds from cameras installed in the parking lot to update the status of each parking space.

Key Features:

- **Video Feed Processing:** Live video feeds from multiple cameras are captured and processed in real-time.
- **Object Detection:** The YOLO model detects and classifies parking spaces as occupied or available. Bounding boxes are drawn around each detected space, and the status is updated.
- **Non-Maximal Suppression:** This technique is used to eliminate redundant detections, ensuring that each parking space is detected only once.

Technical Details:

- **Framework:** TensorFlow or Darknet for model inference.
- **Integration:** OpenCV for video capture and processing.
- **Latency:** Optimization techniques are employed to minimize detection latency and ensure real-time performance.

4. OCR Integration Module

Description:

This module integrates OCR technology to extract and recognize vehicle number plates from camera feeds. It enhances security by tracking vehicles and facilitates parking reservations and automated billing.

Key Features:

- **Number Plate Detection:** Using the live video feeds, the system detects the number plates of vehicles entering and exiting the parking lot.
- **Text Extraction:** OCR algorithms extract the alphanumeric characters from the detected number plates.
- **Database Integration:** The extracted number plate information is stored in a database for tracking and management.

Technical Details:

- **Tools and Libraries:** Tesseract OCR or OpenCV for number plate detection and text extraction.
- **Accuracy:** Preprocessing techniques like image binarization and noise reduction are applied to improve OCR accuracy.
- **Storage:** A relational database (e.g., MySQL or PostgreSQL) to store number plate information.

5. User Interface Module

Description:

This module develops a user-friendly interface accessible via web and mobile applications. It provides real-time information on parking availability, navigation assistance, and reservation options.

Key Features:

- **Real-time Updates:** The interface displays the current status of parking spaces, showing available and occupied spots.

- **Navigation Assistance:** Users can receive directions to available parking spaces within the lot.
- **Reservation System:** Users can reserve parking spots in advance, ensuring availability upon arrival.

Technical Details:

- **Frontend Development:** React, Angular, or Vue.js for web applications; Swift or Kotlin for mobile applications.
- **Backend Integration:** RESTful APIs to connect the frontend with the backend services and the database.
- **User Authentication:** Secure user authentication and authorization mechanisms.

6. Performance Evaluation and Optimization Module

Description:

This module evaluates the system's performance and implements continuous optimization to ensure high accuracy and efficiency.

Key Features:

- **Performance Metrics:** The system is evaluated using metrics like mean average precision (mAP), intersection over union (IoU), and response time.
- **Monitoring:** Continuous monitoring of the system to identify areas for improvement and to detect anomalies.
- **Optimization:** Regular updates and optimization of the model and system components to enhance performance.

Technical Details:

- **Logging and Monitoring:** Tools like Prometheus and Grafana for performance monitoring and logging.
- **Feedback Loop:** User feedback is collected to identify and address usability issues.

SYSTEM ANALYSIS

SYSTEM ANALYSIS

2.1 Existing System

Currently, traditional parking management systems rely on manual supervision or sensor-based approaches to monitor parking slot availability. These existing systems have several limitations, making them inefficient in handling modern-day parking challenges. The main types of existing parking systems are:

1. Manual Parking Management

- Parking attendants manually check and guide vehicles to available slots.
- Entry and exit logs are recorded on paper or basic software.
- **Limitations**:
    - Time-consuming and prone to human errors.
    - Inefficient in large parking areas.
    - No real-time monitoring or automated tracking.

2. Sensor-Based Parking Systems

- Uses RFID, infrared sensors, or ultrasonic sensors to detect vehicle presence.
- Data is transmitted to a central system, displaying available slots.
- **Limitations:**
    - Expensive to install and maintain due to hardware costs.
    - Sensors can malfunction or require frequent calibration.
    - Limited scalability for large-scale deployment.

3. Ticket-Based and Boom Barrier Systems

- Vehicles enter and exit through automated boom barriers with ticket generation or RFID cards.
- Payment is based on the time spent in the parking lot.

- **Limitations:**
    - Not efficient for real-time slot detection.
    - Ticket loss or RFID malfunction can cause delays.
    - High maintenance costs and dependency on additional hardware.

4. Mobile App-Based Reservation Systems

- Users book parking slots through a mobile app.
- Uses GPS or manual input to find available parking.
- **Limitations:**
    - Relies on user input rather than real-time vision-based monitoring.
    - Inaccurate due to delayed updates and human errors.

2.1.1 Disadvantages of the Existing System

The current parking management systems suffer from multiple drawbacks, making them inefficient for large-scale and real-time operations. Some of the key disadvantages include:

1. Lack of Real-Time Monitoring

- Many existing systems, especially manual and ticket-based ones, do not provide live updates on parking slot availability.
- This leads to congestion and frustration for drivers searching for empty slots.

2. High Installation and Maintenance Costs

- Sensor-based parking systems require expensive hardware such as RFID, infrared, or ultrasonic sensors.
- Maintenance and calibration of these sensors increase operational costs.
- Malfunctioning sensors cause incorrect slot detection, leading to mismanagement.

3. Inefficient and Time-Consuming

- Manual systems depend on human attendants, making them slow and prone to errors.
- Ticket-based and RFID systems require scanning at entry and exit points, causing delays during peak hours.
- Drivers often waste time searching for available slots due to inaccurate data updates.

4. Poor Scalability

- Many parking management systems are not easily scalable to handle large parking areas.
- Sensor-based solutions require individual installation for each slot, which is impractical in large parking spaces like malls, airports, and stadiums.
- Some mobile app-based systems rely on manual input rather than automated detection, leading to inaccurate slot availability.

5. Limited Security and Fraud Prevention

- Unauthorized parking is a major issue in manual and sensor-based systems.
- Ticket-based systems can be bypassed or manipulated.
- Many existing solutions do not integrate license plate recognition, making it difficult to track vehicles and prevent misuse.

6. Dependence on Additional Infrastructure

- RFID-based parking requires specialized cards and scanners, making it inconvenient for visitors.
- Boom barrier systems require additional physical infrastructure, which is costly and needs frequent maintenance.

7. Environmental Impact

- Increased time spent searching for parking leads to higher fuel consumption and carbon emissions.
- Systems that require paper tickets contribute to waste and environmental damage.

2.2 Proposed System

To overcome the limitations of existing parking management systems, this project proposes an AI-powered Car Parking Management System using YOLO. The proposed system leverages deep learning and computer vision to detect parking slot occupancy in real-time, eliminating the need for manual supervision or expensive sensor-based infrastructure.

Key Features of the Proposed System

- **Real-Time Parking Slot Detection:**
  - Uses YOLO (You Only Look Once) to detect occupied and empty parking slots from live CCTV feeds or uploaded images/videos.
  - Eliminates the need for RFID, ultrasonic sensors, or manual supervision.
- **Automatic License Plate Recognition (ALPR):**
  - YOLO-based license plate detection integrated with OCR (Optical Character Recognition) to extract vehicle registration numbers.
  - Enhances security by verifying authorized vehicles and preventing unauthorized parking.
- **Web-Based Interface (Streamlit):**
  - A user-friendly dashboard for real-time parking monitoring, slot booking, and vehicle tracking.
  - Provides interactive visualizations of parking slot availability.
- **Manual Slot Marking for Accuracy Enhancement:**
  - Users can manually define and adjust parking slot positions for improved detection accuracy.
- **Parking Slot Reservation System:**
  - Enables users to book parking slots online in advance.
  - Reduces search time and improves space utilization.
- **Database Management (SQLite):**
  - Stores vehicle entry/exit records, parking history, and booking details.
  - Helps in generating reports and tracking parking usage patterns.
- **Scalable and Cost-Effective:**
  - Works with existing CCTV infrastructure, making it affordable and easy to deploy.
  - Suitable for malls, offices, hospitals, airports, and public parking areas.

2.2.1 Advantages:

The Car Parking Management System using YOLO offers several benefits over traditional parking management methods. By leveraging deep learning and computer vision, the system ensures real-time monitoring, security, and efficiency while minimizing costs.

1. Real-Time and Automated Parking Detection

- Uses YOLO (You Only Look Once) to accurately detect occupied and empty parking slots in real-time.
- Eliminates the need for manual monitoring or sensor-based systems.
- Works with live CCTV feeds, images, or videos, making it fast and efficient.

2. No Need for Additional Sensors or Hardware

- Unlike RFID, infrared, or ultrasonic sensors, this system only requires a camera, reducing installation and maintenance costs.
- Avoids hardware failures and calibration issues, making it more reliable.

3. Cost-Effective and Scalable

- Uses open-source technologies (YOLO, OpenCV, SQLite, Streamlit), reducing software expenses.
- Works with existing CCTV infrastructure, making it affordable.
- Easily scalable for malls, offices, hospitals, airports, and public parking areas.

4. Automatic License Plate Recognition (ALPR) for Security

- Uses YOLO + OCR (Optical Character Recognition) to extract license plate numbers.
- Prevents unauthorized parking and improves vehicle tracking.
- Stores vehicle data in SQLite database for future reference.

5. Web-Based Interactive Dashboard (Streamlit)

- User-friendly interface for monitoring parking availability, booking slots, and checking logs.
- Works on any device (PC, mobile, tablet) with an internet connection.
- Allows manual slot marking for improved accuracy.

6. Reduces Traffic Congestion and Parking Search Time

- Drivers can check slot availability online before reaching the parking area.
- Reduces unnecessary movement and waiting time, improving traffic flow.

7. Parking Slot Reservation System

- Users can book parking slots in advance, preventing conflicts.
- Reduces time spent searching for available spaces.

8. Secure and Reliable Data Storage

- All vehicle entries, exits, and booking details are stored in an SQLite database.
- Ensures efficient tracking of parking history for future analysis.

9. Environmentally Friendly

- Reduces fuel consumption and carbon emissions by minimizing the time spent searching for parking.
- No paper-based tickets required, making it eco-friendly.

2.3 Feasibility Study

A feasibility study evaluates the practicality and effectiveness of implementing the Car Parking Management System using YOLO in real-world scenarios. It assesses the project from different aspects, including technical, economic, operational, legal, and scheduling feasibility.

1. Technical Feasibility

This project is technically feasible as it utilizes advanced deep learning and computer vision techniques, which are already well-established.

- Use of YOLO (You Only Look Once) → A fast and accurate object detection algorithm for real-time parking slot monitoring.
- Integration with ALPR (Automatic License Plate Recognition) for vehicle tracking and security verification.
- Implementation in Streamlit → Provides a lightweight and user-friendly web interface.
- No dependency on additional sensors → Eliminates the need for costly hardware-based parking systems.
- Scalability → Works with existing CCTV cameras or uploaded images/videos.

Thus, the system can be efficiently deployed and integrated with existing infrastructure.

2. Economic Feasibility

This project is cost-effective compared to traditional parking management solutions.

- Low hardware costs → Does not require RFID, boom barriers, or sensors.
- Uses open-source technologies → YOLO, OpenCV, SQLite, and Streamlit reduce software expenses.
- Scalable with minimal investment → Can be implemented in various parking lots, malls, and offices without major modifications.
- Reduces operational costs → Eliminates the need for manual supervision and improves efficiency.

Since it utilizes existing surveillance cameras, the initial investment is minimal, making the project financially viable.

3. Operational Feasibility

The system is designed to be user-friendly, efficient, and easy to maintain.

- Automated slot detection reduces manual errors and improves parking efficiency.
- Live monitoring and booking system ensure accurate real-time updates.
- License plate recognition enhances security and prevents unauthorized parking.
- Easy to integrate with existing parking infrastructures and mobile/web platforms.
- Minimal training required for operators and administrators.

Thus, the system can be efficiently managed with minimal human intervention.

4. Legal Feasibility

Legal compliance is essential when dealing with vehicle monitoring and license plate recognition.

- Data Privacy & Security: The system must comply with data protection laws to ensure that vehicle data is not misused.
- Camera Regulations: Public parking areas must have permissions to use CCTV footage for monitoring.
- Fair Usage Policy: Vehicle owners should be informed about the data collection to ensure transparency.

By following privacy regulations and ethical AI practices, the project remains legally viable.

5. Scheduling Feasibility

The system can be developed and deployed within a reasonable timeframe.

Project Timeline:

- Phase 1: Research and dataset preparation (1-2 weeks).
- Phase 2: YOLO model training and testing (2-3 weeks).
- Phase 3: Web interface development using Streamlit (2 weeks).
- Phase 4: Integration with ALPR and database setup (2-3 weeks).
- Phase 5: Testing and deployment (2 weeks).

The system can be fully functional within 2-3 months, making it time-efficient and practical.

SYSTEM REQUIREMETNS

3. SYSTEM REQUIREMENTS

3.1 Hardware Requirements

The hardware requirements for using YOLO models depend on the specific architecture and size of the model.

- **CPU:** At least 2 cores.
- **GPU:** At least 4GB memory for small YOLO models.
  At least 8GB for larger YOLO models.
- **RAM:** Minimum 8 GB.
- **Hard Disk:** 500 GB.
- **Monitor:** 15" LED.
- **Input Devices:** Keyboard, Mouse.

3.2 Software Requirements

The software requirements for using YOLO models include operating systems, necessary platforms and frameworks on which the software requires support to run.

- **Operating System:** Windows 10 or Linux.
- **Programming Language:** Python 3.5 or higher.
- **IDE's:** Visual Studio, PyCharm, Anaconda.
- **Tools:** Spyder, OpenCV, NumPy, Keras.
- **Deep Learning Frameworks:** TensorFlow.

3.3 Tools & Techniques:

3.3.1 Tools:

These are the software and frameworks used in the project:

- **Programming Language:**
  - Python 3.8+ – Used for model development, data handling, and application logic.
- **Deep Learning & Computer Vision Frameworks:**
  - YOLOv8 (Ultralytics) – Used for detecting vehicles and empty parking slots.
  - OpenCV – Used for image processing and drawing bounding boxes.

- **Database & Web Development:**
  - SQLite – Stores user data, booked slots, and vehicle entries.
  - Streamlit – Creates a web-based interface for users and administrators.

3.3.2. Techniques:

These are the key techniques used in the system:

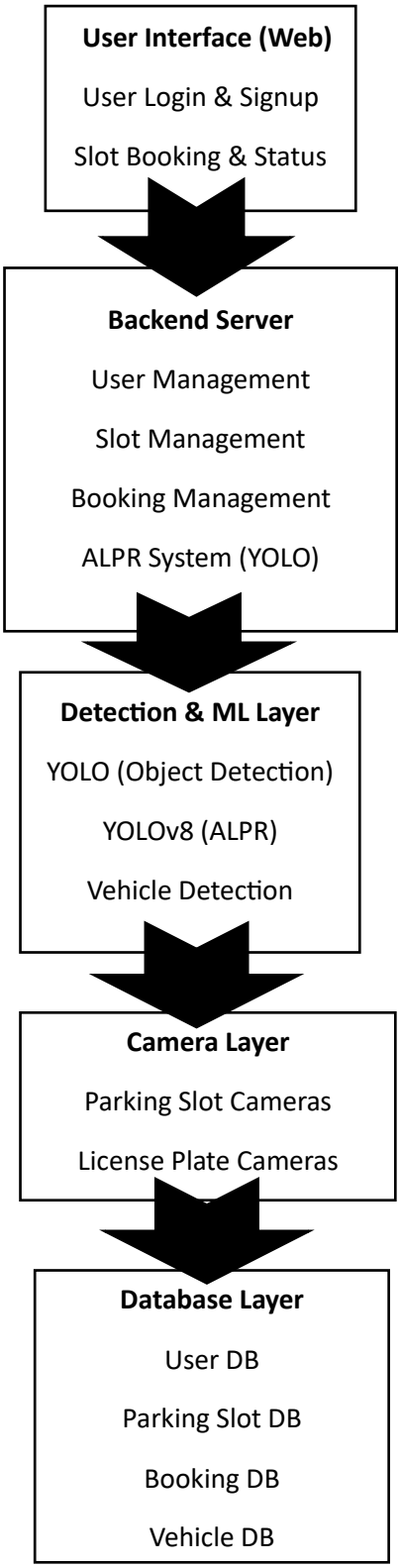
- **Object Detection using YOLO:**
  - YOLO (You Only Look Once) is used to detect vehicles and classify parking slots as occupied or empty in real time.
- **Bounding Box Detection:**
  - YOLO outputs bounding boxes around detected vehicles, which helps in slot identification.
- **Database Management:**
  - SQLite stores user registrations, parking logs, and booking details to ensure secure and structured data storage.
- **Image Processing:**
  - OpenCV filters and enhances images before passing them to the YOLO model for better accuracy.
- **Streamlit UI for User Interaction:**
  - Users can upload images/videos and view detected parking slot statuses using a simple web interface.

SYSTEM DESIGN

4. SYSTEM DESIGN

A computer vision-based car parking management system that utilizes YOLO (You Only Look Once) for detecting occupied and empty parking slots. The system also integrates license plate recognition (ALPR) for automated booking and verification.

System Architecture Diagram:



1. User Interaction Layer (Frontend):

- User Registration & Login: Users sign up and log in to book parking slots.
- Parking Slot Booking: Interface for users to choose available parking slots.
- Parking Slot Availability: Displays parking slots' real-time occupancy status.

2. Backend Layer (Server-side):

- User Management System: Manages user accounts, registrations, and login sessions.
- Slot Management: Manages slot availability, user bookings, and updates slot status.
- ALPR System: Uses YOLO for automatic license plate recognition to validate parking bookings.
- Parking Slot Detection using YOLO: Real-time parking slot detection using a camera feed, detecting whether the slot is occupied or free.

3. Detection Layer (Computer Vision & ML Models):

- YOLO for Object Detection:
  - Detects vehicles and identifies occupied/empty parking slots using cameras.
- License Plate Recognition (YOLOv8 for ALPR):
  - Extracts the vehicle’s license plate and matches it against registered users.

4. Database Layer (Storage):

- User Database: Stores user credentials and information.
- Parking Slot Database: Stores the parking slot status (available/occupied).
- Booking Database: Stores the booking details and tracks the booked slots with timestamps.
- Vehicle Database: Stores license plate data for validation during booking.

5. Camera/Device Layer:

- Cameras (for Slot Monitoring): Monitors parking slots, feeds images or videos to the detection system.

- Camera (for License Plate Recognition): Captures vehicle license plates for validation during booking.

Description of Key Components:

- **User Interface (Web):**
  - Allows users to log in, sign up, and book parking slots. Displays available slots, their status, and parking history.
- **Backend Server:**
  - Handles user requests, slot bookings, and slot status updates. It communicates with the detection and machine learning layers for real-time parking slot occupancy updates.
- **Detection & ML Layer:**
  - YOLO Object Detection: Detects vehicles in parking slots. If a vehicle is detected, the slot is marked as occupied.
  - ALPR: Extracts and validates the vehicle's license plate number, ensuring that only registered vehicles can book or occupy a parking slot.
- **Camera Layer:**
  - Camera feeds are used to monitor the parking slots and license plates. The cameras provide real-time data to the detection system.
- **Database Layer:**
  - Contains all the system data, including user details, parking slot statuses, bookings, and vehicle registration information.

5.1 Input Design

1. Input Design Overview:

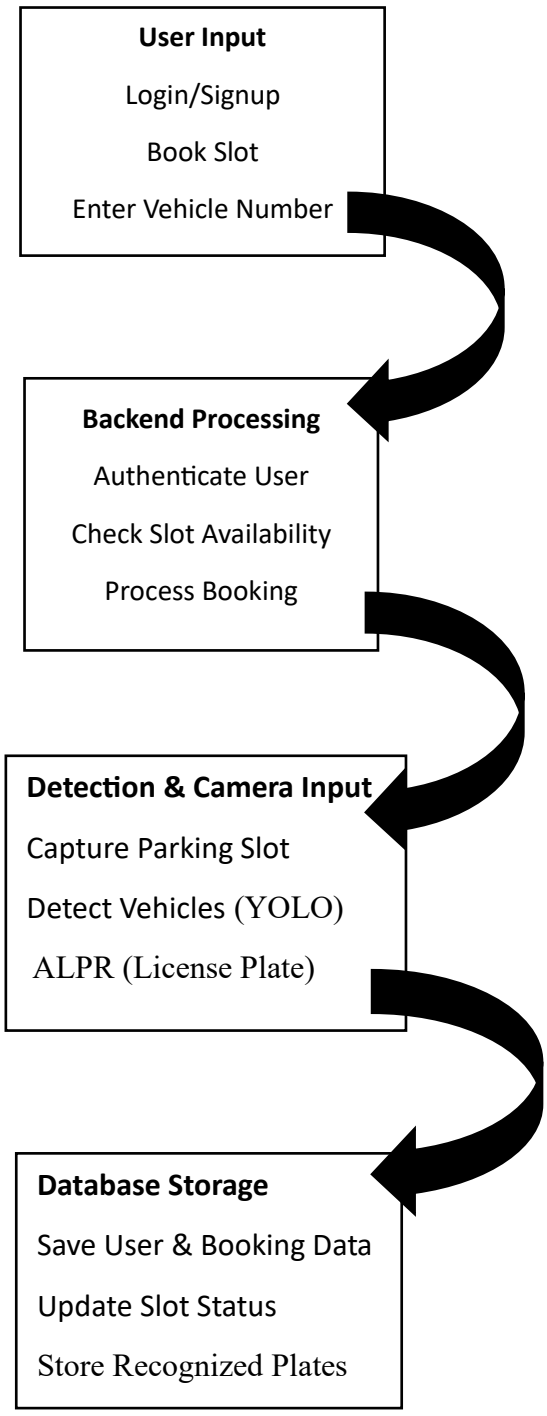
The input design ensures smooth data entry and processing for parking slot detection, license plate recognition, and user interaction. It involves:

- **User Inputs**: Login, registration, booking, and manual slot marking.

- **Camera Inputs**: Real-time video/image feeds for parking slot detection and license plate recognition.
- **Database Inputs**: Storing user details, vehicle details, and slot booking records.

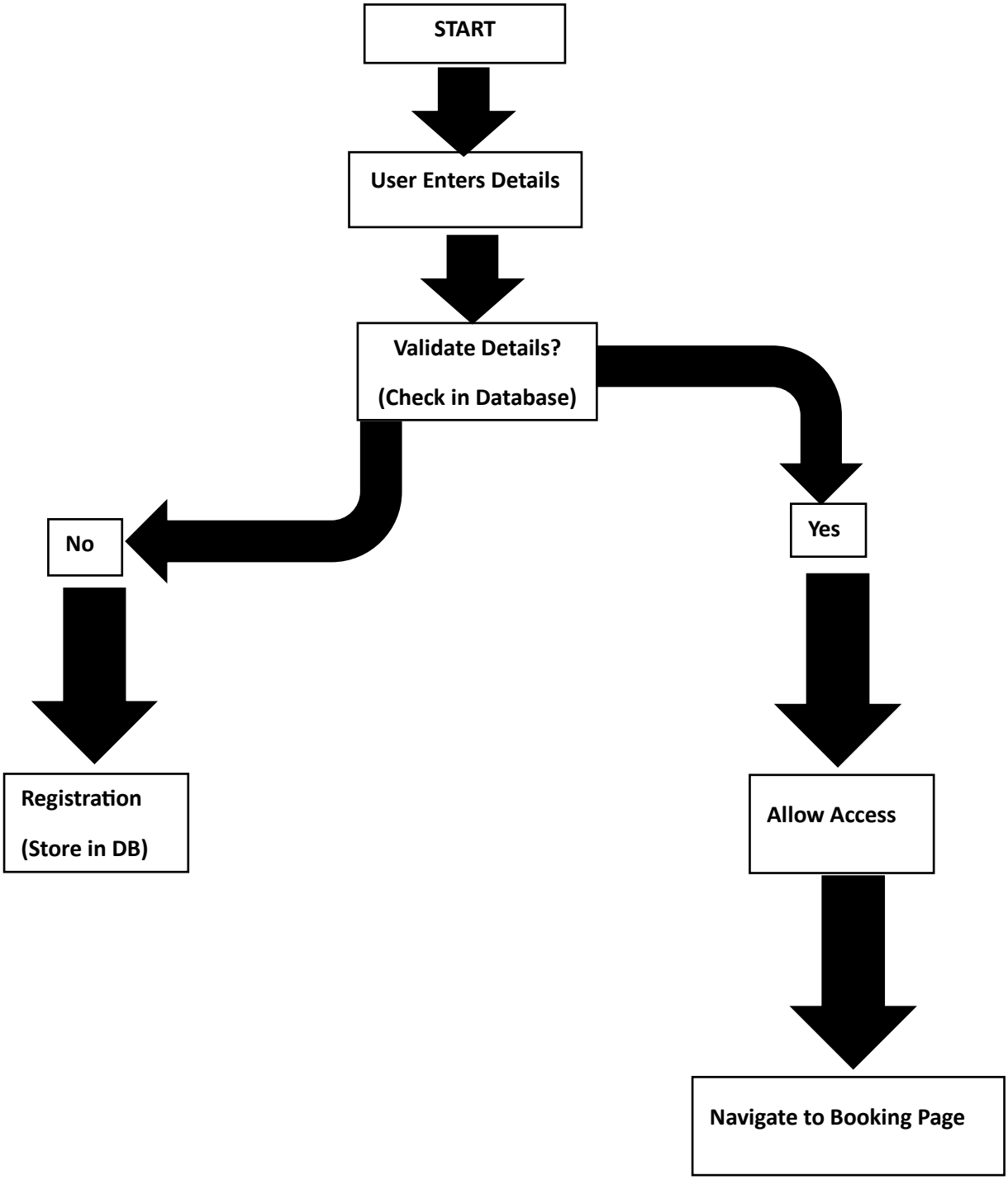
2. Input Flow Diagram

Flow of Data through the System

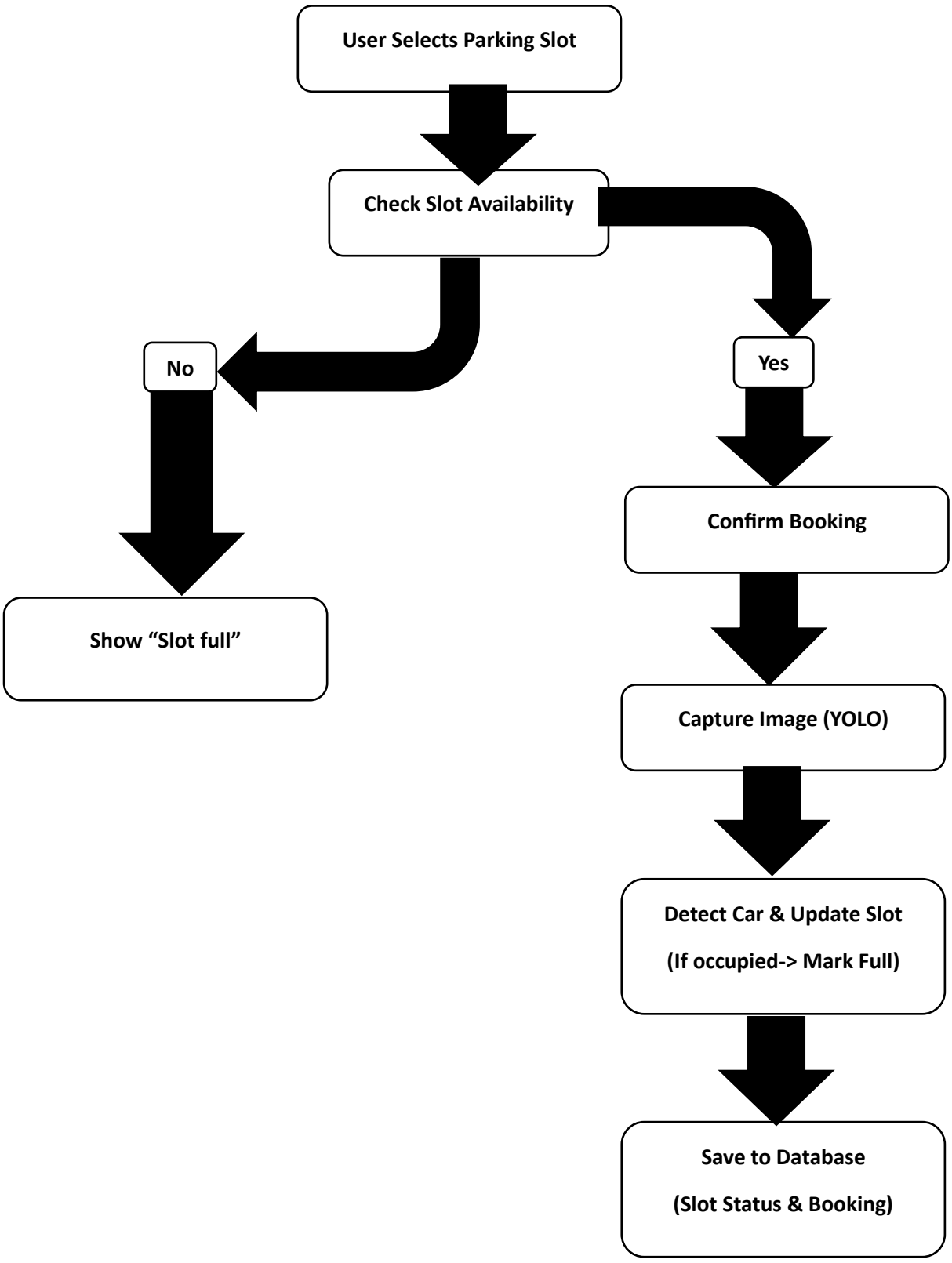


3. Flowchart for the System Input Process

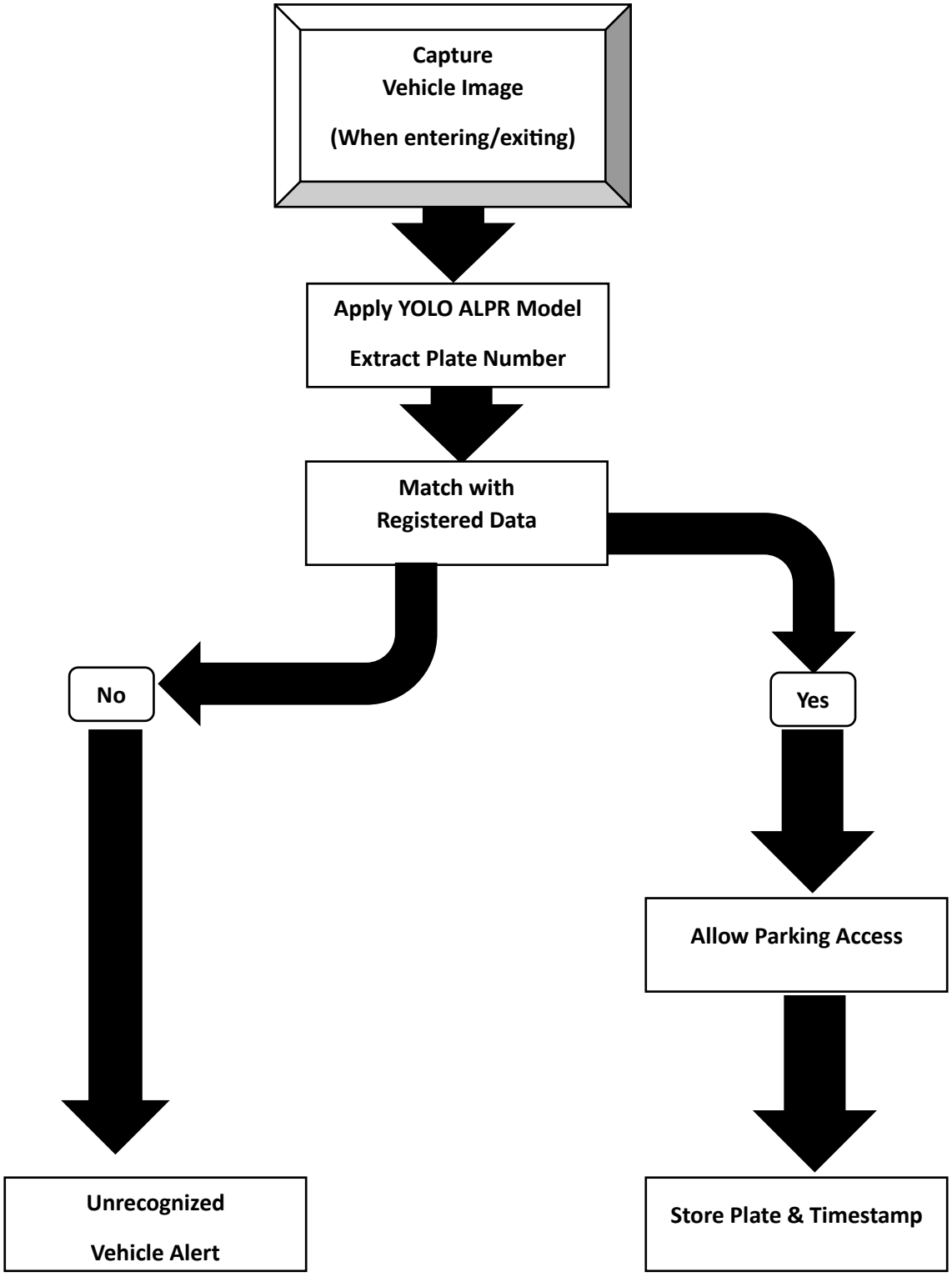
A. User Login & Registration



B. Parking Slot Booking & Detection



C. Automatic License Plate Recognition (ALPR) Using YOLO



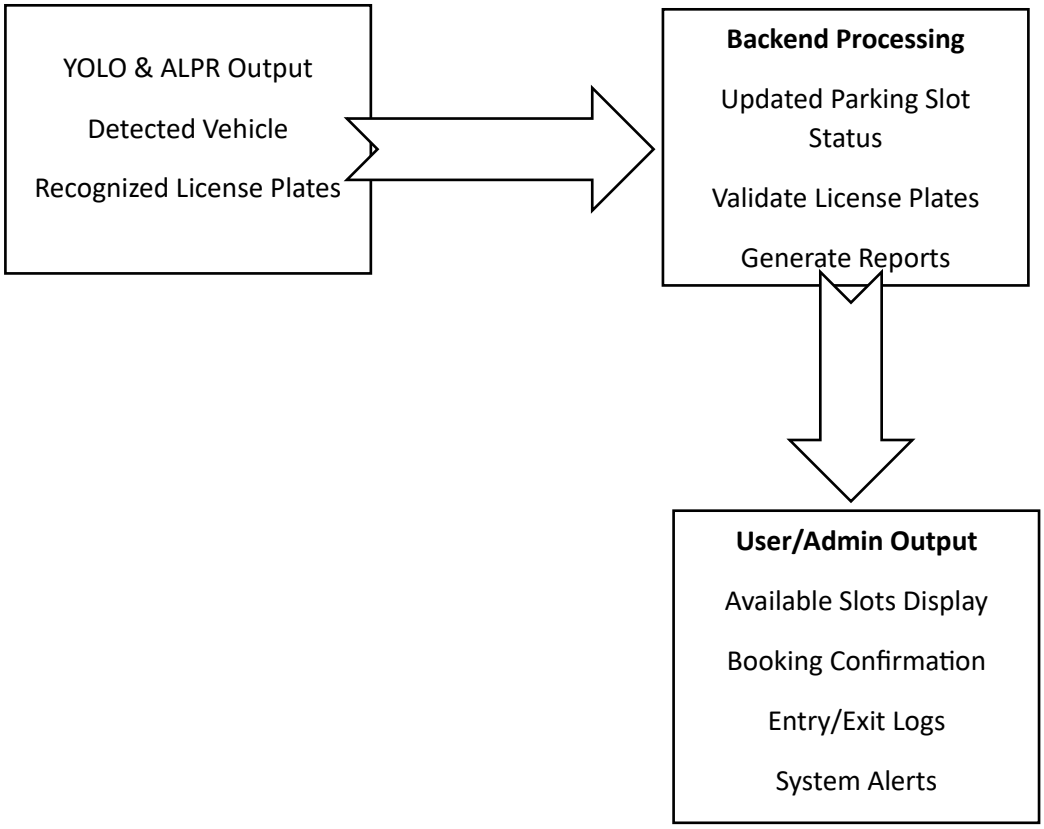
5.2 Output Design

1. Output Design Overview

The system generates different types of outputs:

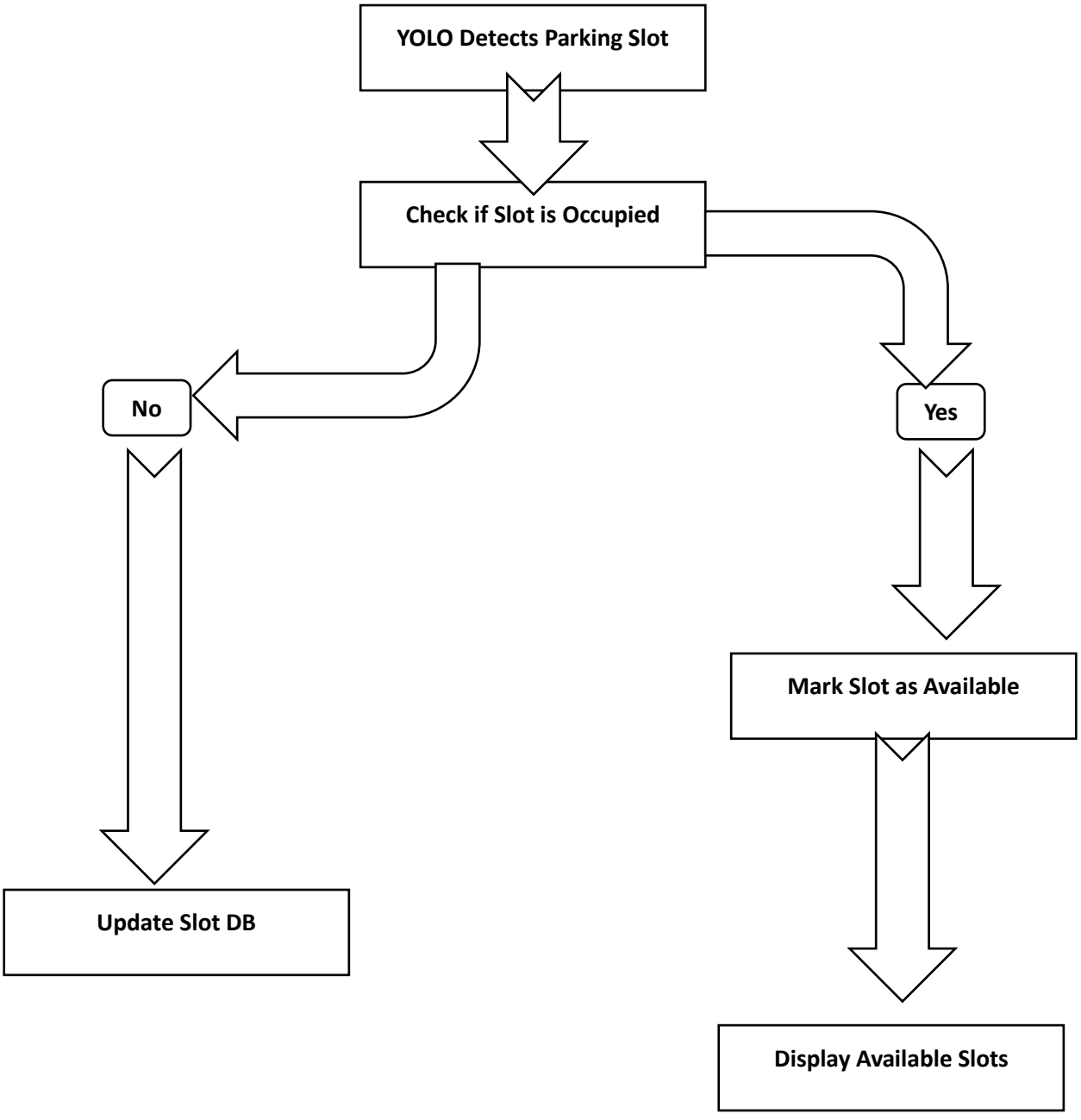
- **Real-time Parking Slot Status**: Shows available and occupied slots using YOLO detection.
- **User Notifications**: Displays booking confirmations, slot availability, and alerts for unrecognized vehicles.
- **License Plate Recognition Output**: Shows recognized plate numbers and matches them with registered users.
- **Admin Dashboard Reports**: Includes statistics on parking usage, bookings, and entry/exit logs.

2. Output Flow Diagram

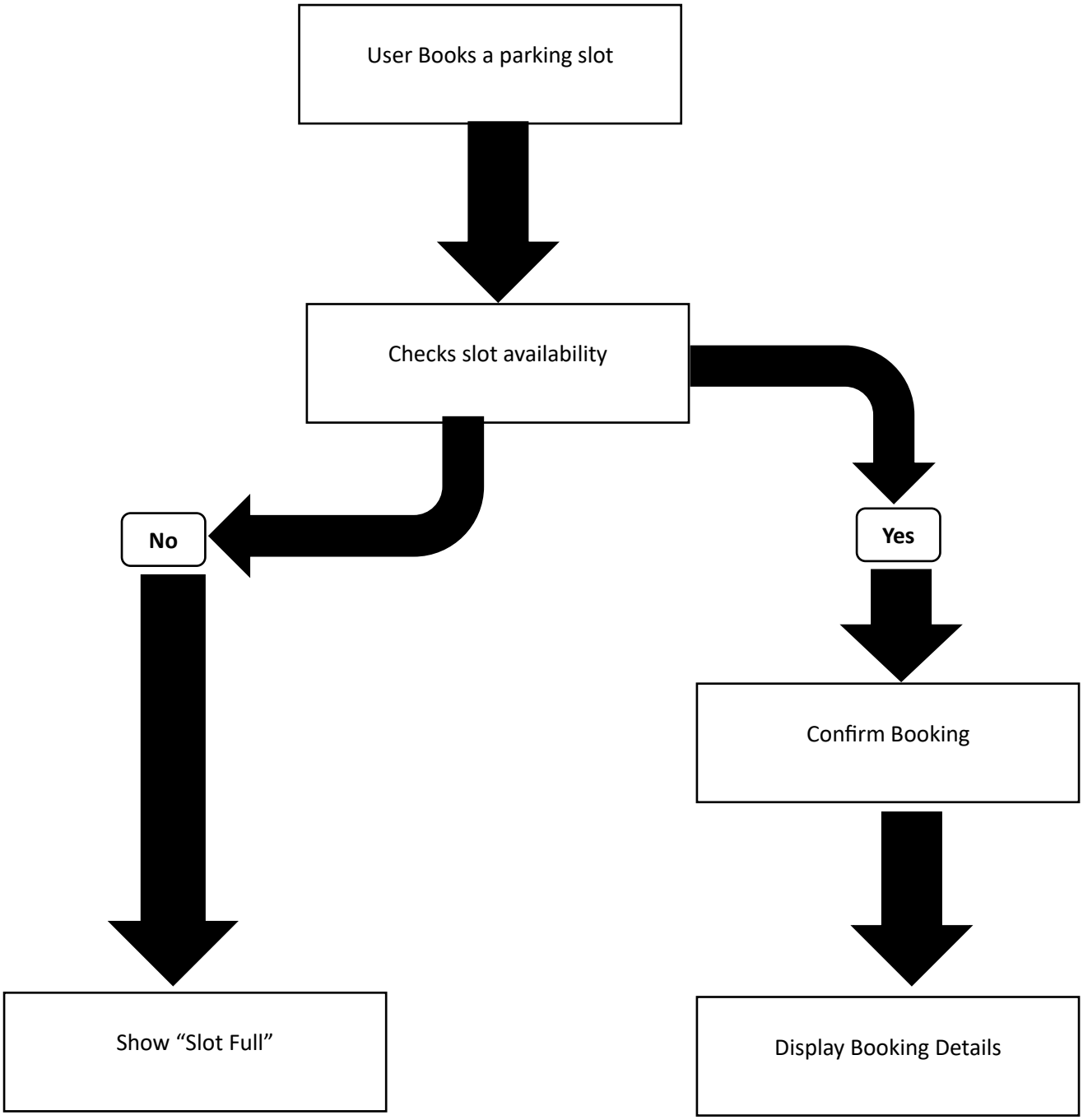


3. Flowchart for System Output Process

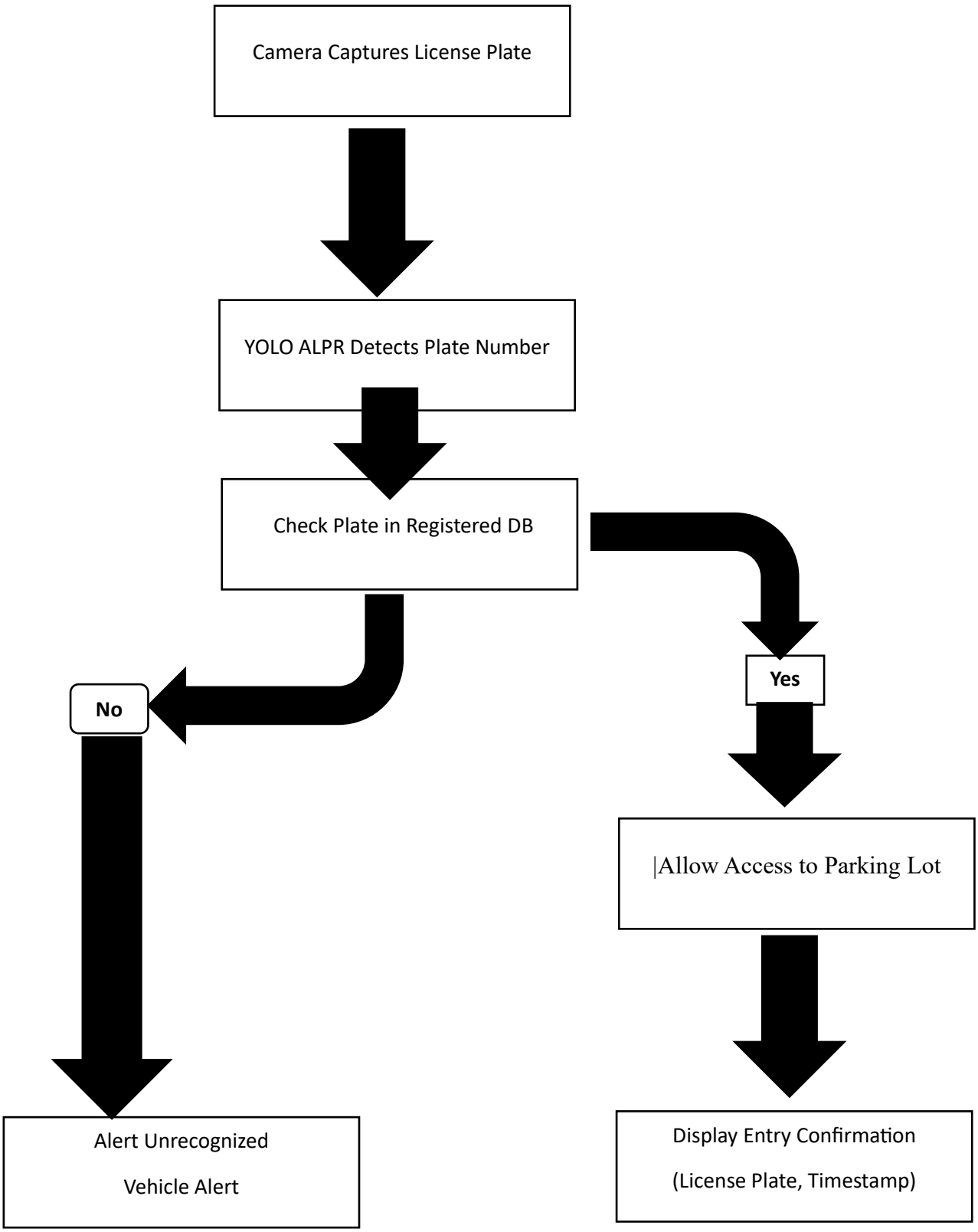
A. Parking Slot Status Display



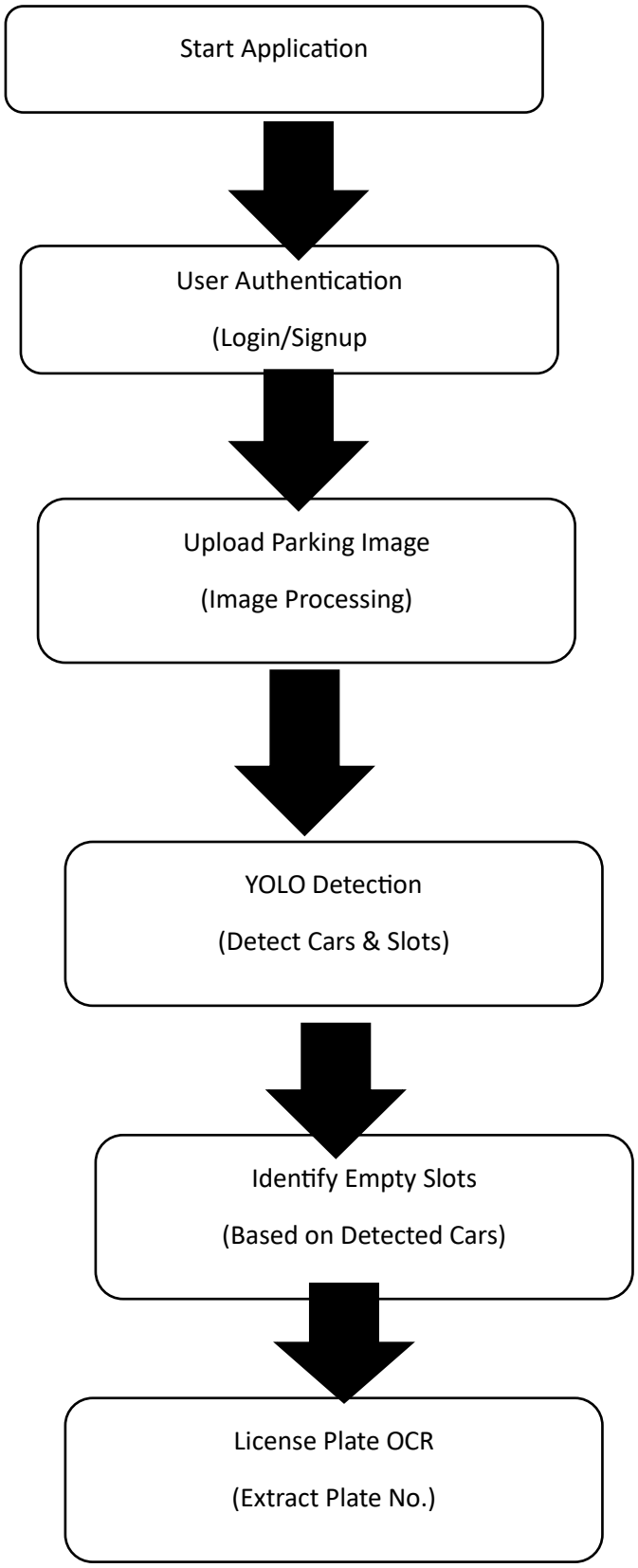
B. Booking Confirmation & User Output

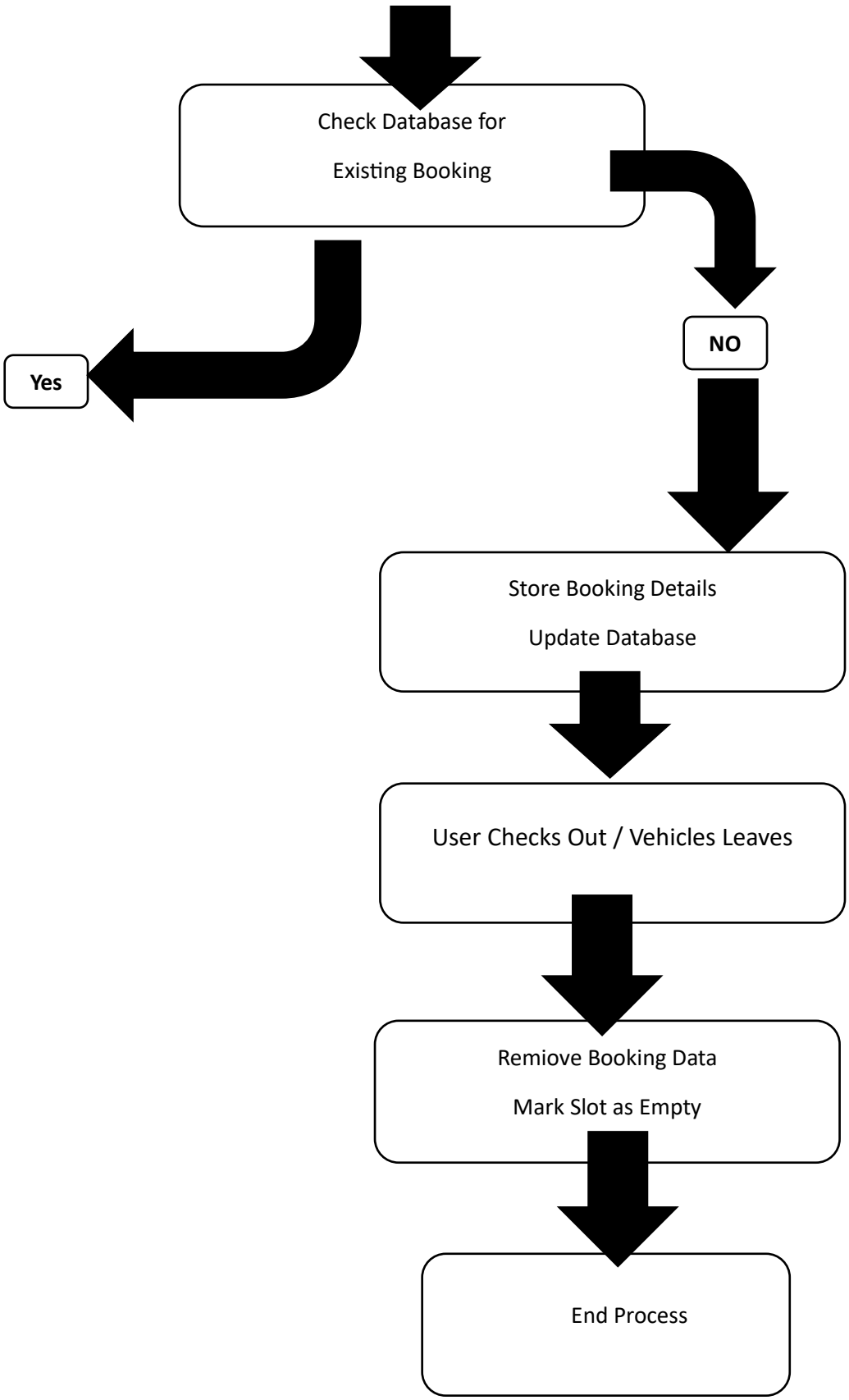


C. License Plate Recognition Output (ALPR)



5.3 Flow Diagram




Check Database for
Existing Booking
Yes
NO
Store Booking Details
Update Database
User Checks Out / Vehicles Leaves
Remiove Booking Data
Mark Slot as Empty
End Process

SYSTEM IMPLEMENTATION

6. SYSTEM IMPLEMENTATION

6.1 Sample code

Main.py

```
import streamlit as st

import subprocess

import base64

# Set Streamlit page configuration

st.set_page_config(page_title="Home", page_icon=" 🏠 ")


def add_bg_from_local(image_file):

    with open(image_file, "rb") as image_file:

        encoded_string = base64.b64encode(image_file.read()).decode()

    st.markdown(

        f"""

        <style>

        .stApp {{

            background-image: url({encoded_string});

            background-size: cover;

        }}

        .stTextInput label, .stSelectbox label, .stButton button, .stMarkdown, .stTitle,
.stSuccess, .stError, .stWarning {{

            color: black !important;

        }}
```

```
        </style>
        """,
        unsafe_allow_html=True
    )

add_bg_from_local(r"D:\project_finalize\assets\images.jpeg")
# Initialize session state variables
if "authenticated" not in st.session_state:
    st.session_state.authenticated = False

# Home Page Title
st.title("🏠 Welcome to the Parking Management System 🚗")

st.write("Please log in as a User or Admin to continue.")

# User/Admin Selection
user_type = st.radio("Select Role:", ["User", "Admin"])

# Navigation Buttons
if st.button("Proceed"):
    if user_type == "User":
        subprocess.Popen(["streamlit", "run", "D:/project_finalize/user/main1.py"])  # Redirect User to main1.py
    elif user_type == "Admin":
```

subprocess.Popen(["streamlit", "run", "D:/project_finalize/admin/stream_log.py"]) # Redirect Admin to stream_signup.py

Redirect if authenticated

if st.session_state.authenticated:

st.success("✅ Authentication successful! Redirecting to upload page...")

subprocess.Popen(["streamlit", "run", "D:/project_finalize/adminadmin/stream_upload.py"])

Admin:

stream_log.py:

import streamlit as st

import sqlite3

import hashlib

import subprocess

import base64

✅ Set Page Configuration FIRST

st.set_page_config(page_title="Admin Login", page_icon="🔑")

✅ Database File Path

DB_FILE = "D:/project_finalize/database/admin.db"

✅ Function to Hash Passwords

def make_hashes(password):

```
    return hashlib.sha256(password.encode()).hexdigest()


# ✅ Function to Verify Admin Login
def login_admin(username, password):
    conn = sqlite3.connect(DB_FILE, check_same_thread=False)
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM admins WHERE username = ? AND password = ?",
                   (username, make_hashes(password)))
    result = cursor.fetchone()
    conn.close()
    return result


# ✅ Function to Set Background Image
def add_bg_from_local(image_file):
    with open(image_file, "rb") as image_file:
        encoded_string = base64.b64encode(image_file.read()).decode()
    st.markdown(
        f"""
        <style>
        .stApp {{
            background-image: url({encoded_string});
            background-size: cover;
        }}
        .stTextInput label, .stButton button, .stMarkdown, .stTitle, .stSuccess,
```

```
        .stError, .stWarning {{
            color: black !important;
        }}
        </style>
        """,
        unsafe_allow_html=True
    )

# ✅ Set Background Image
add_bg_from_local("D:/project_finalize/assets/images.jpeg")

# ✅ Title
st.title("🚪 Admin Login - Car Parking System")

# ✅ Input Fields
username = st.text_input("👤 Admin Username")
password = st.text_input("🔑 Password", type="password")

# ✅ Login Button
if st.button("Login"):
    if login_admin(username, password):
        st.session_state.logged_in = True
        st.session_state.username = username
```

```
        st.success(f"✅ Logged in as Admin: {username}")
        st.write("Opening Admin Dashboard...")

        # ✅ Open Admin Dashboard
        subprocess.Popen(["streamlit", "run", "D:/project_finalize/admin/stream_upload.py"])
        st.stop()
    else:
        st.error("❌ Invalid Admin Credentials! Please try again.")

# ✅ Logout Button
if st.button("Logout"):
    st.session_state.clear()
    st.success("✅ Successfully logged out!")
    subprocess.Popen(["streamlit", "run", "D:/project_finalize/user/userin/login.py"])
    st.stop()
```

stream_upload.py:

```
import streamlit as st
import os
import subprocess
import base64
import sqlite3
import pandas as pd

# ✅ Set Page Configuration
```

```
st.set_page_config(page_title="Dashboard", page_icon=" 📋 ", layout="wide")


# ✅ Function to set background image
def add_bg_from_local(image_file):
    with open(image_file, "rb") as image_file:
        encoded_string = base64.b64encode(image_file.read()).decode()
    st.markdown(
        f"""
        <style>
        .stApp {{
            background-image: url({encoded_string});
            background-size: cover;
        }}
        .stButton > button {{
            width: 300px !important;
            height: 50px !important;
            font-size: 18px !important;
            font-weight: bold !important;
        }}
        </style>
        """,
        unsafe_allow_html=True
    )
```

```
add_bg_from_local(r"D:\project_finalize\assets\images.jpeg")


# ✅ Database Connection Functions
def get_user_db_connection():
    return sqlite3.connect("D:/project_finalize/database/my.db", check_same_thread=False)


def get_admin_db_connection():
    return sqlite3.connect("D:/project_finalize/database/admin.db", check_same_thread=False)


# ✅ Create Tables (If Not Exists)
def create_user_table():
    conn = get_user_db_connection()
    cursor = conn.cursor()
    cursor.execute("""
        CREATE TABLE IF NOT EXISTS users (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            username TEXT UNIQUE NOT NULL,
            password TEXT NOT NULL,
            login_time TIMESTAMP
        )
    """)
    conn.commit()
    conn.close()
```

```
def create_admin_table():
    conn = get_admin_db_connection()
    cursor = conn.cursor()
    cursor.execute("""
        CREATE TABLE IF NOT EXISTS admins (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            username TEXT UNIQUE NOT NULL,
            password TEXT NOT NULL,
            login_time TIMESTAMP
        )
    """)
    conn.commit()
    conn.close()


# ✅ Fetch Users from user.db
def get_users():
    conn = get_user_db_connection()
    cursor = conn.cursor()
    cursor.execute("SELECT id, username, login_time FROM users ORDER BY login_time DESC")
    users = cursor.fetchall()
    conn.close()
    return users
```

```
# ✅ Fetch Admins from admin.db
def get_admins():
    conn = get_admin_db_connection()
    cursor = conn.cursor()
    cursor.execute("SELECT id, username, login_time FROM admins ORDER BY login_time DESC")
    admins = cursor.fetchall()
    conn.close()
    return admins

# ✅ Ensure tables exist before querying
create_user_table()
create_admin_table()

# ✅ Main Dashboard Title
st.title("📊 Parking Management System Dashboard")

# ✅ Create Three Buttons
col1, col2, col3 = st.columns(3)

with col1:
    if st.button("📋 View User Details"):
        st.session_state.show_users = True
        st.session_state.show_admins = False  # Hide admin table
```

```
with col2:
    if st.button("🔐 View Admin Details"):
        st.session_state.show_users = False  # Hide user table
        st.session_state.show_admins = True

with col3:
    if st.button("🚪 Logout"):
        subprocess.Popen(["streamlit", "run", "D:/project_finalize/admin/stream_log.py"])
        st.stop()

# ✅ Show User Details
if st.session_state.get("show_users", False):
    st.subheader("👤 User Details")
    users_data = get_users()
    if users_data:
        df_users = pd.DataFrame(users_data, columns=["ID", "Username", "Last Login Time"])
        st.dataframe(df_users)
    else:
        st.info("No users found.")

# ✅ Show Admin Details
if st.session_state.get("show_admins", False):
    st.subheader("🔐 Admin Details")
```

```
    admins_data = get_admins()
    if admins_data:
        df_admins = pd.DataFrame(admins_data, columns=["ID", "Username", "Last Login Time"])
        st.dataframe(df_admins)
    else:
        st.info("No admins found.")

# ✅ Upload & Processing Section
st.write("---")
st.subheader(" 📤 Upload & Process Files")

# Ensure directories exist
os.makedirs("uploads_img", exist_ok=True)
os.makedirs("uploads_video", exist_ok=True)

# Upload Section (Only Shows if Button is Clicked)
if st.button(" 🚀 Upload & Process Files"):
    st.session_state.show_upload = True

if st.session_state.get("show_upload", False):
    st.subheader(" 📤 Upload Page")

    # Upload Image
```

```
    if st.button("Process Image 📷"):
        subprocess.Popen(["streamlit", "run", "D:/project_finalize/admin/stream_img.py"])

    # Upload Video
    if st.button("Process Video 🎥"):
        subprocess.Popen(["streamlit", "run", "D:/project_finalize/admin/stream_video.py"])
```

stream_img.py:

```
import streamlit as st
import os
import cv2
import numpy as np
import sqlite3
from ultralytics import YOLO
from PIL import Image, ImageDraw
import subprocess
import base64

st.set_page_config(page_title="Image Processing", page_icon="📤")

# ✅ Function to set background image
def add_bg_from_local(image_file):
    with open(image_file, "rb") as image_file:
        encoded_string = base64.b64encode(image_file.read()).decode()
    st.markdown(
```

```
        f"""
        <style>
        .stApp {{
            background-image: url({encoded_string});
            background-size: cover;
        }}
        </style>
        """,
        unsafe_allow_html=True
    )

add_bg_from_local(r"D:\project_finalize\assets\images.jpeg")

# ✅ Load YOLO model
model = YOLO("yolov8n.pt")

# ✅ Function to detect red areas as empty slots
def detect_red_as_empty(image):
    hsv_img = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    lower_red1 = np.array([0, 120, 70])
    upper_red1 = np.array([10, 255, 255])
    lower_red2 = np.array([170, 120, 70])
    upper_red2 = np.array([180, 255, 255])
```

```
    mask1 = cv2.inRange(hsv_img, lower_red1, upper_red1)
    mask2 = cv2.inRange(hsv_img, lower_red2, upper_red2)
    mask = mask1 + mask2

    contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    empty_slots = [(cv2.boundingRect(cnt)) for cnt in contours]
    return empty_slots

# ✅ Function to process the uploaded image
def process_uploaded_image(image_path):
    img = cv2.imread(image_path)
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    # YOLO Detection for occupied slots
    results = model(image_path)
    detections = results[0].boxes.xyxy.cpu().numpy()
    occupied_slots = [(int(x1), int(y1), int(x2), int(y2)) for x1, y1, x2, y2 in detections]

    # Detect Red Regions as Empty Slots
    empty_slots = detect_red_as_empty(img)

    # Convert OpenCV image to PIL for annotation
    annotated_img = Image.fromarray(img_rgb)
    draw = ImageDraw.Draw(annotated_img)
```

```
    # Draw occupied slots (YOLO)
    for x1, y1, x2, y2 in occupied_slots:
        draw.rectangle([x1, y1, x2, y2], outline="green", width=3)
        draw.text((x1, y1 - 10), "Occupied", fill="green")

    # Draw empty slots (Red Regions)
    for x, y, w, h in empty_slots:
        draw.rectangle([x, y, x + w, y + h], outline="blue", width=3)
        draw.text((x, y - 10), "Empty", fill="blue")

    img_rgb = np.array(annotated_img)
    st.image(img_rgb, caption="Processed Image with Detections",
use_container_width=True)

    return len(occupied_slots), len(empty_slots)

# ========== MAIN STREAMLIT APP ==========
st.title("🚗 Parking Lot Detection - Image Processing")
st.subheader("📁 Upload Parking Lot Image 📷")

uploaded_file = st.file_uploader("Choose an image", type=["jpg", "png", "jpeg"])

if st.button("To Process Video 🎥"):
```

```
    subprocess.Popen(["streamlit", "run", "D:/project_finalize/admin/stream_video.py"])


if uploaded_file is not None:
    save_path = os.path.join("uploads", uploaded_file.name)
    os.makedirs("uploads", exist_ok=True)

    with open(save_path, "wb") as f:
        f.write(uploaded_file.getbuffer())

    st.success(f"✅ File saved: {save_path}")

    # Display original image
    st.image(save_path, caption="Uploaded Image", use_container_width=True)

    # Process Image
    auto_occupied, auto_empty = process_uploaded_image(save_path)

    # Display slot counts
    st.write(f"🔹 **Total Slots:** {auto_occupied + auto_empty}")
    st.write(f"🔴 **Occupied Slots (YOLO-detected):** {auto_occupied}")
    st.write(f"🟢 **Empty Slots (Red Regions):** {auto_empty}")

# ✅ Logout Button
if st.button("Logout"):
```

```
    subprocess.Popen(["streamlit", "run", "D:/project_finalize/main2.py"])
```

stream_video1.py:

```
import streamlit as st
import os
import cv2
import numpy as np
import sqlite3
import pickle
from ultralytics import YOLO
import subprocess
import base64


# Set Page Configurations
st.set_page_config(page_title="Video", page_icon=" 📤 ")


def add_bg_from_local(image_file):
    with open(image_file, "rb") as image_file:
        encoded_string = base64.b64encode(image_file.read()).decode()
    st.markdown(
        f"""
        <style>
        .stApp {{
            background-image: url({encoded_string});
            background-size: cover;
```

```
        }}
        .stTextInput label, .stSelectbox label, .stButton button, .stMarkdown, .stTitle, .stSuccess, .stError, .stWarning {{
            color: black !important;
        }}
        </style>
        """,
        unsafe_allow_html=True
    )

add_bg_from_local(r"D:\project_finalize\assets\images.jpeg")

# ========== DATABASE CONNECTION ==========
def get_db_connection():
    return sqlite3.connect("D:/project_finalize/database/admin.db", check_same_thread=False)

# Create Table to Store Parking Slots in admin.db
def create_parking_table():
    conn = get_db_connection()
    cursor = conn.cursor()
    cursor.execute("""
        CREATE TABLE IF NOT EXISTS parking_slots (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            x1 INTEGER NOT NULL,
            y1 INTEGER NOT NULL,
```

```
            x2 INTEGER NOT NULL,
            y2 INTEGER NOT NULL
        )
    """)
    conn.commit()
    conn.close()

create_parking_table()

# Function to fetch stored parking slots
def get_parking_slots():
    conn = get_db_connection()
    cursor = conn.cursor()
    cursor.execute("SELECT x1, y1, x2, y2 FROM parking_slots")
    slots = cursor.fetchall()
    conn.close()
    return slots

# Load YOLO model
model = YOLO("yolov8n.pt")

# Function to process video and detect occupied/empty slots
def process_video(video_path):
    cap = cv2.VideoCapture(video_path)
```

```
stframe = st.empty()  # Streamlit placeholder for displaying frames

frame_rate = cap.get(cv2.CAP_PROP_FPS)  # Get frame rate
frame_interval = int(frame_rate * 10)  # Frames per 10 seconds

total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
video_duration = total_frames / frame_rate

st.sidebar.write(f"📌 **Video Duration: {video_duration:.2f} seconds**")

final_occupied = 0
final_empty = 0

parking_slots = get_parking_slots()  # Fetch slots from admin.db

for frame_num in range(total_frames):
    ret, frame = cap.read()
    if not ret:
        break  # Stop if video ends

    img_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

    # Run YOLO detection
    results = model(frame)
```

```
    detections = results[0].boxes.xyxy.cpu().numpy()


    occupied_boxes = []
    for box in detections:
        if len(box) >= 4:
            x1, y1, x2, y2 = map(int, box[:4])
            occupied_boxes.append((x1, y1, x2, y2))
            cv2.rectangle(img_rgb, (x1, y1), (x2, y2), (0, 0, 255), 2)
            cv2.putText(img_rgb, "Occupied", (x1, y1 - 5), cv2.FONT_HERSHEY_SIMPLEX,
0.6, (0, 0, 255), 2)


    occupied_slots = len(occupied_boxes)


    # Identify empty slots
    empty_slots = []
    if parking_slots:
        for slot in parking_slots:
            if len(slot) == 4:
                x1, y1, x2, y2 = slot
                is_occupied = any(
                    ox1 < x2 and ox2 > x1 and oy1 < y2 and oy2 > y1
                    for ox1, oy1, ox2, oy2 in occupied_boxes
                )
                if not is_occupied:
                    empty_slots.append(slot)
```

```
                cv2.rectangle(img_rgb, (x1, y1), (x2, y2), (0, 255, 0), 2)
                cv2.putText(img_rgb, "Empty", (x1, y1 - 5),
cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 0), 2)

        total_slots = len(parking_slots)
        empty_count = len(empty_slots)

        # Show parking stats every 10 seconds
        if frame_num % frame_interval == 0:
            timestamp = frame_num / frame_rate
            st.write(f"📌 **Parking Status at {timestamp:.0f} Seconds**")
            st.write(f"🔹 **Total Slots:** {total_slots}")
            st.write(f"🔴 **Occupied Slots:** {occupied_slots}")
            st.write(f"🟢 **Empty Slots:** {empty_count}")
            st.image(img_rgb, caption=f"Status at {timestamp:.0f} Seconds", channels="RGB",
use_container_width=True)

        # Update final counts
        final_occupied = occupied_slots
        final_empty = empty_count

        # Show real-time video frame in Streamlit
        stframe.image(img_rgb, channels="RGB", use_container_width=True)
```

```
    cap.release()

    # Show final results after the video ends
    st.write("✅ **Final Parking Status After Video**")
    st.write(f"🔹 **Total Slots:** {total_slots}")
    st.write(f"🔴 **Final Occupied Slots:** {final_occupied}")
    st.write(f"🟢 **Final Empty Slots:** {final_empty}")

# ========== MAIN STREAMLIT APP ==========
def main():
    st.title("🚗 Parking Lot Detection - Video Processing 🚗")
    st.subheader("Upload Parking Lot Video 🎥")
    uploaded_file = st.file_uploader("📂 Choose a video", type=["mp4", "avi", "mov"])

    if st.button("To Process Image 📷"):
        subprocess.Popen(["streamlit", "run", "D:/project_finalize/admin/stream_img.py"])

    if uploaded_file is not None:
        save_path = os.path.join("uploads", uploaded_file.name)
        os.makedirs("uploads", exist_ok=True)
        with open(save_path, "wb") as f:
            f.write(uploaded_file.getbuffer())
        st.success(f"✅ File saved: {save_path}")
```

```
        # Process video and detect parking status dynamically
        process_video(save_path)

    if st.button("Logout"):
        subprocess.Popen(["streamlit", "run", "D:/project_finalize/main2.py"])

if __name__ == '__main__':
    main()
```

User:

Main.py:

```
import streamlit as st
import sqlite3
import hashlib
import subprocess
import base64

# ✅ Database Connection (Using my.db)
def get_db_connection():
    conn = sqlite3.connect("D:/project_finalize/database/my.db", check_same_thread=False)
    conn.execute("""
        CREATE TABLE IF NOT EXISTS users (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            username TEXT UNIQUE NOT NULL,
```

```
        password TEXT NOT NULL
    )
    """)
    conn.commit()  # Ensure table is created before inserting users
    return conn

# ✅ Hashing passwords for security
def hash_password(password):
    return hashlib.sha256(password.encode()).hexdigest()

# ✅ Set background image
def add_bg_from_local(image_file):
    with open(image_file, "rb") as image_file:
        encoded_string = base64.b64encode(image_file.read()).decode()
    st.markdown(
        f"""
        <style>
        .stApp {{
            background-image: url({encoded_string});
            background-size: cover;
        }}
        .stTextInput label, .stButton > button {{
            font-size: 18px !important;
            font-weight: bold !important;
```

```
            width: 200px !important;
            height: 50px !important;
            border-radius: 10px !important;
            color: white !important;
            background-color: #FF5733 !important;
        }}
        </style>
        """,
        unsafe_allow_html=True
    )

# ✅ Apply background image
add_bg_from_local(r"D:\project_finalize\assets\images2.jpg")

# ✅ Page Title
st.title("🚗 Car Parking Management")

# ✅ Centered Layout for Login & Signup Buttons
col1, col2 = st.columns(2)

with col1:
    if st.button("🔐 Login"):
        subprocess.Popen(["streamlit", "run", "D:/project_finalize/user/userin/login.py"])
```

```
with col2:
    if st.button(" 📝 Register"):
        subprocess.Popen(["streamlit", "run", "D:/project_finalize/user/userin/register.py"])
```

register.py:

```
import streamlit as st
import sqlite3
import hashlib
import subprocess
import base64


# ✅ Database Connection
def get_db_connection():
    conn = sqlite3.connect("D:/project_finalize/database/my.db", check_same_thread=False)
    conn.execute("""
        CREATE TABLE IF NOT EXISTS users (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            username TEXT UNIQUE NOT NULL,
            password TEXT NOT NULL,
            role TEXT NOT NULL DEFAULT 'user'  -- ✅ Ensures role is always set
        )
    """)
    conn.commit()
    return conn
```

```
# ✅ Hashing Function
def hash_password(password):
    return hashlib.sha256(password.encode()).hexdigest()

# ✅ Background Image
def add_bg_from_local(image_file):
    with open(image_file, "rb") as image_file:
        encoded_string = base64.b64encode(image_file.read()).decode()
    st.markdown(
        f"""
        <style>
        .stApp {{
            background-image: url({encoded_string});
            background-size: cover;
        }}
        </style>
        """,
        unsafe_allow_html=True
    )

add_bg_from_local(r"D:\project_finalize\assets\images2.jpg")

# ✅ Signup Form
st.title("📝 Register - Car Parking Management")
```

```
username = st.text_input("👤 Username", placeholder="Enter a unique username").strip().lower()
password = st.text_input("🔑 Password", type="password", placeholder="Enter a strong password")

if st.button("Sign Up"):
    if username and password:
        conn = get_db_connection()
        cursor = conn.cursor()

        # ✅ Check if username exists
        cursor.execute("SELECT username FROM users WHERE username = ?", (username,))
        existing_user = cursor.fetchone()

        if existing_user:
            st.error("❌ Username already exists. Try another one.")
        else:
            cursor.execute("INSERT INTO users (username, password, role) VALUES (?, ?, ?)",
                           (username, hash_password(password), "user")) # ✅ Includes role
            conn.commit()
            st.success("✅ Signup Successful! Proceed to Login.")
            subprocess.Popen(["streamlit", "run", "D:/project_finalize/user/userin/login.py"])

        conn.close()
```

```
    else:
        st.warning("⚠️ Please enter both username and password.")


# ✅ Login Redirect Button
if st.button("🔐 Go to Login"):
    subprocess.Popen(["streamlit", "run", "D:/project_finalize/user/userin/login.py"])
```

login.py:

```
import streamlit as st
import sqlite3
import hashlib
import subprocess
import base64
# ✅ Database Connection (Using my.db)
def get_db_connection():
    conn = sqlite3.connect("D:/project_finalize/database/my.db", check_same_thread=False)
    return conn


# ✅ Hashing passwords for security
def hash_password(password):
    return hashlib.sha256(password.encode()).hexdigest()


st.title("🚗 Login - Car Parking Management")
def add_bg_from_local(image_file):
    with open(image_file, "rb") as image_file:
```

```
        encoded_string = base64.b64encode(image_file.read()).decode()
    st.markdown(
        f"""
        <style>
        .stApp {{
            background-image: url({encoded_string});
            background-size: cover;
         }}
         .stTextInput label, .stSelectbox label, .stButton button, .stMarkdown, .stTitle, .stSuccess, .stError, .stWarning {{
              color: black !important;
         }}
        </style>
        """,
        unsafe_allow_html=True
    )

add_bg_from_local(r"D:\project_finalize\assets\images2.jpg")
username = st.text_input("👤 Username", placeholder="Enter your username")
password = st.text_input("🔑 Password", type="password", placeholder="Enter your password")

# ✅ Login Button
if st.button("Login"):
    if username and password:
```

```
        conn = get_db_connection()
        cursor = conn.cursor()
        cursor.execute("SELECT * FROM users WHERE username = ? AND password = ?",
                       (username, hash_password(password)))
        user = cursor.fetchone()
        conn.close()

        if user:
            st.success("✅ Login Successful! Redirecting to Booking Page...")
            subprocess.Popen(["streamlit", "run", "D:/project_finalize/user/userin/booking.py"])
# Redirect to Booking Page
        else:
            st.error("❌ Invalid Username or Password!")
    else:
        st.warning("⚠️ Please enter both username and password.")

# ✅ Signup Redirect Button
if st.button("Go to Signup"):
    subprocess.Popen(["streamlit", "run", "D:/project_finalize/user/userin/register.py"])  # Redirect to Signup Page
```

booking.py:

```
import streamlit as st
import sqlite3
import cv2
```

```
import numpy as np

import easyocr

import os

import base64

import subprocess


# ✅ Predefined GPS coordinates & images for each parking slot

PARKING_SLOTS = {

    "Meenakshi Amman Temple - Multi-level Car and Bike Parking": {

        "map":
"https://www.google.com/maps/embed?pb=!1m18!1m12!1m3!1d3945.3745!2d78.119776!3d
9.919592",

        "image": "D:/project_finalize/assets/meenakshi_parking.jpg",

        "fee": "₹30/hour"

    },

    "ELCOT IT Park, Madurai": {

        "map":
"https://www.google.com/maps/embed?pb=!1m18!1m12!1m3!1d3945.64523!2d78.119876!3
d9.920987",

        "image": "D:/project_finalize/assets/elcot_parking.jpg",

        "fee": "₹40/hour"

    },

    "AK AHAMED CAR PARKING": {

        "map":
"https://www.google.com/maps/embed?pb=!1m18!1m12!1m3!1d3945.6745!2d78.120876!3d
9.922345",
```

```
        "image": "D:/project_finalize/assets/ahamed_parking.jpg",
        "fee": "₹50/hour"
    }
}

# ✅ Database setup (Using my.db)
def get_db_connection():
    conn = sqlite3.connect("D:/project_finalize/database/my.db", check_same_thread=False)
    cursor = conn.cursor()

    # ✅ Ensure the bookings table exists
    cursor.execute("""
        CREATE TABLE IF NOT EXISTS bookings (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            plate_number TEXT UNIQUE NOT NULL,
            slot_number TEXT NOT NULL,
            fee TEXT NOT NULL DEFAULT '₹0',
            status TEXT DEFAULT 'Pending'
        )
    """)

    # ✅ Ensure the admin_notifications table exists
    cursor.execute("""
        CREATE TABLE IF NOT EXISTS admin_notifications (
```

```
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            plate_number TEXT NOT NULL,
            slot_number TEXT NOT NULL,
            status TEXT DEFAULT 'Pending'
        )
    """)

    conn.commit()
    return conn



st.title("🚗 Book Parking Slot")


# ✅ Function to set background image
def add_bg_image(main_bg):
    with open(main_bg, "rb") as img_file:
        main_bg_encoded = base64.b64encode(img_file.read()).decode()
    st.markdown(
        f"""
        <style>
        .stApp {{
            background-image: url({main_bg_encoded});
            background-size: cover;
        }}
```

```
        </style>
        """,
        unsafe_allow_html=True
    )

# ✅ Set main background only
add_bg_image("D:/project_finalize/assets/images2.jpg")

# ✅ Initialize detected_plate to prevent errors
detected_plate = None

uploaded_file = st.file_uploader("📂 Upload Car Image for OCR", type=["jpg", "png", "jpeg"])

if uploaded_file:
    # ✅ Save uploaded image
    img_path = os.path.join("uploads", uploaded_file.name)
    os.makedirs("uploads", exist_ok=True)
    with open(img_path, "wb") as f:
        f.write(uploaded_file.getbuffer())

    # ✅ Read Image and Perform OCR
    image = cv2.imread(img_path)
    reader = easyocr.Reader(['en'])
```

```
result = reader.readtext(image)

# ✅ Extract plate number (if detected)
if result:
    detected_plate = result[0][-2]  # Ensures a valid plate is detected
else:
    detected_plate = "UNKNOWN"

st.image(image, caption=f"🔍 Detected Plate: {detected_plate}")

# ✅ User must enter the plate number manually for validation
user_plate = st.text_input("📝 Enter Your Vehicle Plate Number (For Confirmation)")

# ✅ Compare entered plate with detected plate
if st.button("✅ Confirm Plate Number"):
    if user_plate.strip().upper() == detected_plate.upper():
        st.success("✅ Plate Number Verified! You may proceed with booking.")
    else:
        st.error("❌ Plate Number Mismatch! Please check and try again.")

# ✅ Show available slots if plate is verified
if user_plate.strip().upper() == detected_plate.upper():
    st.write("### 🅿️ Available Parking Slots")
```

```
    # ✅ Display slot images as clickable buttons
    selected_slot = None
    for slot_name, slot_data in PARKING_SLOTS.items():
        st.image(slot_data["image"], caption=f"📍 {slot_name}", use_column_width=True)
        col1, col2 = st.columns(2)

        with col1:
            if st.button(f"📍 View Map: {slot_name}"):
                st.markdown(
                    f'<iframe width="700" height="500" src="{slot_data["map"]}"
frameborder="0" allowfullscreen></iframe>',
                    unsafe_allow_html=True
                )

        with col2:
            if st.button(f"🚗 Book {slot_name}"):
                selected_slot = slot_name

    # ✅ If a slot is selected, show details
    if selected_slot:
        slot_data = PARKING_SLOTS[selected_slot]
        st.image(slot_data["image"], caption=f"📍 {selected_slot}",
use_column_width=True)
```

```
        st.write(f"💰 **Fee Structure:** {slot_data['fee']}")

        if st.button("✅ Confirm Booking"):
            conn = get_db_connection()
            cursor = conn.cursor()

            # ✅ Check if plate is already booked
            cursor.execute("SELECT * FROM bookings WHERE plate_number = ?", (detected_plate,))
            existing_booking = cursor.fetchone()

            if existing_booking:
                st.error("❌ This car is already booked in a parking slot.")
            else:
                # ✅ Insert booking as 'Pending' for admin approval
                cursor.execute("INSERT INTO bookings (plate_number, slot_number, fee, status) VALUES (?, ?, ?, 'Pending')",
                               (detected_plate, selected_slot, slot_data["fee"]))
                cursor.execute("INSERT INTO admin_notifications (plate_number, slot_number, status) VALUES (?, ?, 'Pending')",
                               (detected_plate, selected_slot))
                conn.commit()
                st.success(f"✅ Booking Request Sent for {detected_plate}!")
```

```
                # ✅ Show clickable Google Maps link
                st.markdown(f"📍 **[View Directions on Google Maps](https://www.google.com/maps?q={selected_slot})**", unsafe_allow_html=True)

                # ✅ Embed Google Map in Streamlit
                st.markdown(
                    f'<iframe width="700" height="500" src="{slot_data["map"]}" frameborder="0" allowfullscreen></iframe>',
                    unsafe_allow_html=True
                )

            conn.close()

# ✅ Logout Button
st.write("---")
if st.button("🚪 Logout"):
    st.session_state.clear()  # Clear session state
    st.success("✅ Logged out successfully! Redirecting to login page...")
    subprocess.Popen(["streamlit", "run", "D:/project_finalize/main2.py"])
    st.stop()
```

Database Codes:

Admin.py:

```
import sqlite3
import hashlib
```

```
# ✅ Define Database File
DB_FILE = "D:/project_finalize/database/admin.db"

# ✅ Connect to the Database (Creates it if it doesn’t exist)
conn = sqlite3.connect(DB_FILE)
cursor = conn.cursor()

# ✅ Create `admins` Table (Stores Admin Logins)
cursor.execute("""
    CREATE TABLE IF NOT EXISTS admins (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        username TEXT UNIQUE NOT NULL,
        password TEXT NOT NULL
    )
""")

# ✅ Predefined Admin Users (Modify if Needed)
admin_users = [
    ("admin1", "adminpass1"),
    ("superadmin", "securepass"),
    ("manager", "manager123"),
]
```

✅ Hash Passwords Before Storing in Database

hashed_admins = [(u, hashlib.sha256(p.encode()).hexdigest()) for u, p in admin_users]

✅ Insert Admin Users Only If Table is Empty

cursor.execute("SELECT COUNT(*) FROM admins")

if cursor.fetchone()[0] == 0:

    cursor.executemany("INSERT INTO admins (username, password) VALUES (?, ?)", hashed_admins)

    conn.commit()

    print("✅ Admin users added successfully!")

else:

    print("⚠️ Admin users already exist.")

✅ Close Database Connection

conn.close()

print("✅ Admin Database (`admin.db`) Setup Completed!")

user.db.py:

import sqlite3

import hashlib

✅ Define Database File

DB_FILE = "D:/project_finalize/database/my.db"

✅ Connect to the Database (Creates it if it doesn't exist)

```
conn = sqlite3.connect(DB_FILE)
cursor = conn.cursor()

# ✅ Create `users` Table (Stores User Logins)
cursor.execute("""
    CREATE TABLE IF NOT EXISTS users (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        username TEXT UNIQUE NOT NULL,
        password TEXT NOT NULL,
        email TEXT UNIQUE NOT NULL
    )
""")

conn.commit()
conn.close()
print("✅ User Database (`my.db`) Setup Completed!")
```

7. SYSTEM TESTING

7.1 Testing

Test Type	Purpose
Unit Testing →	Check individual functions (e.g., image upload, YOLO detection, database operations).
Integration Testing →	Ensure that different modules work together (e.g., YOLO detection with manual slot marking).
Functional Testing →	Verify that the system meets all functional requirements (e.g., detecting empty & occupied slots).
Performance Testing→	Check response time, processing speed, and system load capacity.
Security Testing →	Ensure secure login, database access control, and data protection.
User Acceptance Testing (UAT) →	Test with end users for usability, accuracy, and ease of use.

7.2 Unit Testing

Unit testing is the testing of each module and the integration of the overall system is done. Unit testing becomes verification efforts on the smallest unit of software design in the module. This is also known as 'module testing'. The modules of the system are tested separately. This testing is carried out during the programming itself. In this testing step, each model is found to be working satisfactorily as regard to the expected output from the module. There are some validation checks for the fields. For example, the validation check is done for verifying the data given by the user where both format and validity of the data entered is included. It is very easy to find error and debug the system.

7.3 Integrating Testing

Data can be lost across an interface, one module can have an adverse effect on the other sub function, when combined, may not produce the desired major function. Integrated testing is systematic testing that can be done with sample data. The need for the integrated test is to find the overall system performance. There are two types of integration testing. They are:

i) Top-down integration testing.

ii) Bottom-up integration testing.

FUTURE ENHANCEMENT

- **Multi-camera support:**

Currently, the proposed system is designed to detect parking spaces in a single camera feed. In future work, the system can be enhanced to support multiple camera feeds, enabling more extensive coverage of a parking lot. This would require integrating a camera network and developing algorithms for camera calibration and object tracking across multiple feeds. This can increase the coverage area of the system and improve its accuracy.

- **Integration with parking guidance systems:**

The proposed system could be integrated with parking guidance systems to direct drivers to available parking spaces in real-time. By providing drivers with real-time parking availability information, the system could reduce traffic congestion and help drivers save time and fuel. This integration would require the development of an interface that can communicate with parking guidance systems.

- **Improved object detection accuracy:**

While the YOLO algorithm is highly accurate, there is always room for improvement. In future work, the system's accuracy could be further enhanced by incorporating more advanced object detection techniques, such as deep learning and machine learning algorithms. This would

require a larger dataset and more computing resources, but it could result in even higher accuracy for car parking space detection.

- **Edge computing:**

Implementing edge computing can improve the speed and efficiency of car parking space detection. This can be achieved by deploying the object detection model on edge devices, such as Raspberry Pi or NVIDIA Jetson, to perform real-time object detection at the edge of the network, without relying on cloud computing. This can also reduce network latency and save cloud storage space.

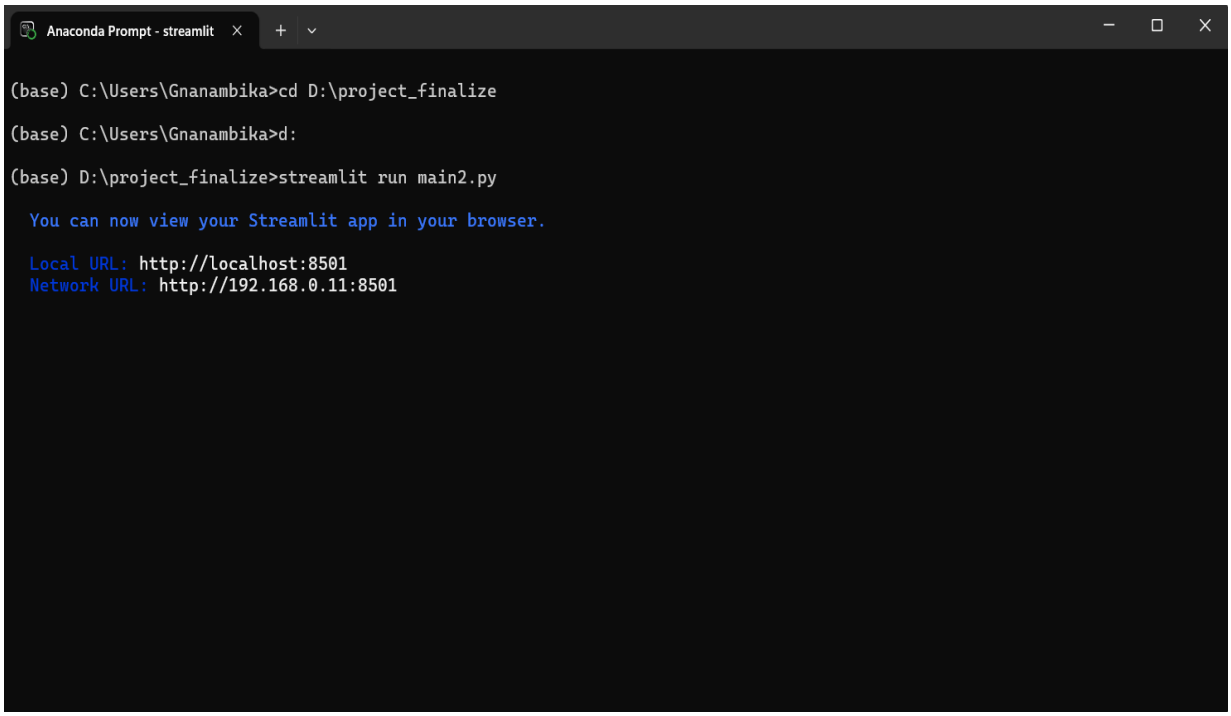
CONCLUSION

In this study, a car parking space detection system is developed using the YOLO algorithm. The system is trained and tested on a dataset of images taken from a multi-level car park. The results of the experiments showed that the YOLO algorithm is able to accurately detect the parking spaces and vehicles in the images, with an AP. The results were compared with those of other object detection algorithms and the results showed that YOLO outperformed the other algorithms in terms of speed and accuracy. The results of this study have important implications for the development of intelligent parking systems. The use of YOLO in this study has demonstrated that it is possible to develop a fast, accurate and reliable car parking space detection system that can be used in real-world applications. These results have the potential to improve the efficiency and convenience of car parking. In conclusion, the car parking space detection system developed in this study provides a promising approach for detecting parking spaces and vehicles in real-world images. The use of YOLO has shown that it is possible to develop an accurate and efficient system that can be used in real-world applications. Future work could focus on improving the performance of the system by using more advanced deep learning techniques and incorporating additional information.

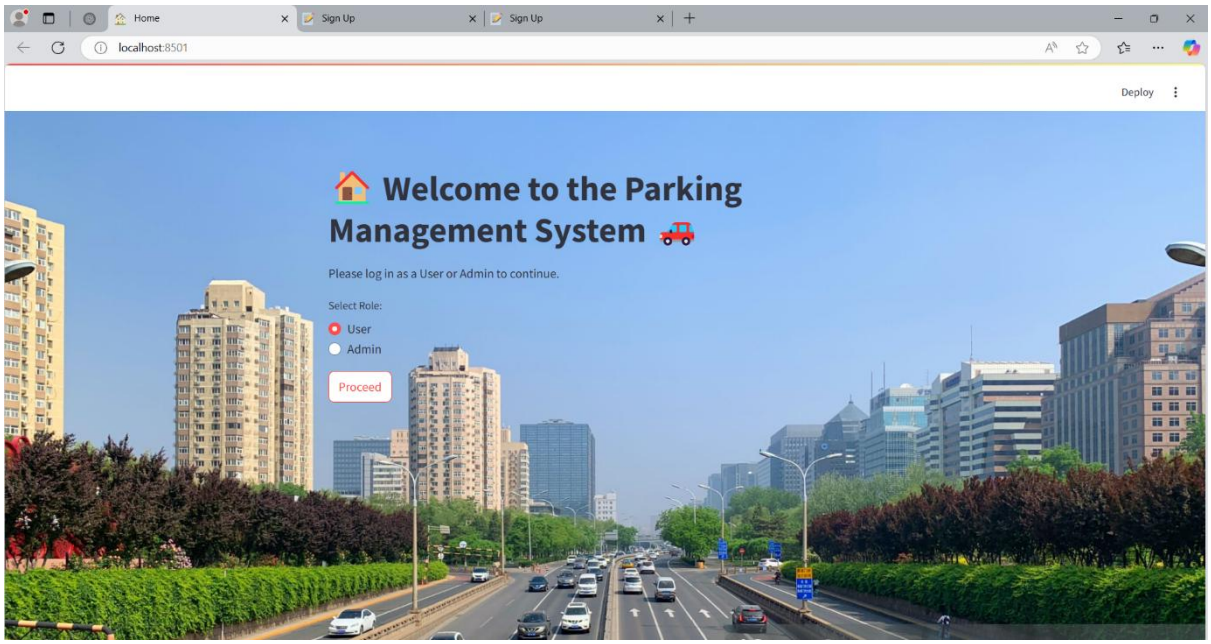
ANNEXURE

9.1 Sample Output:

Run in prompt:

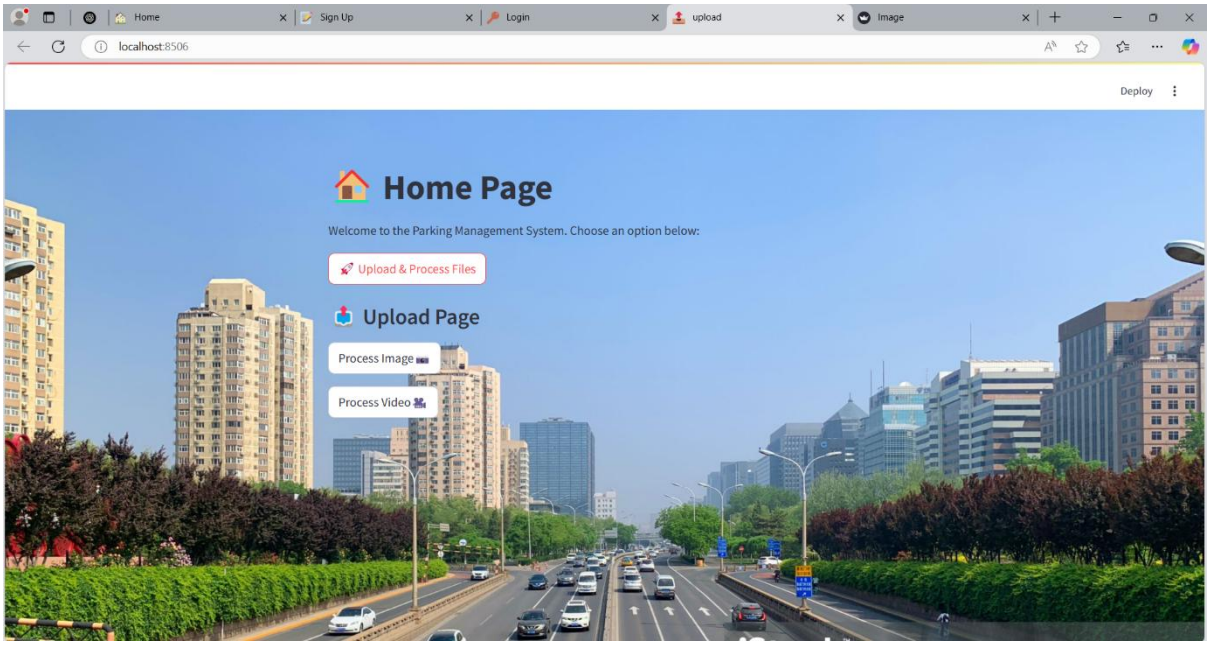


Welcome page:



ADMIN SIDE :

Home Page:



Upload Page:

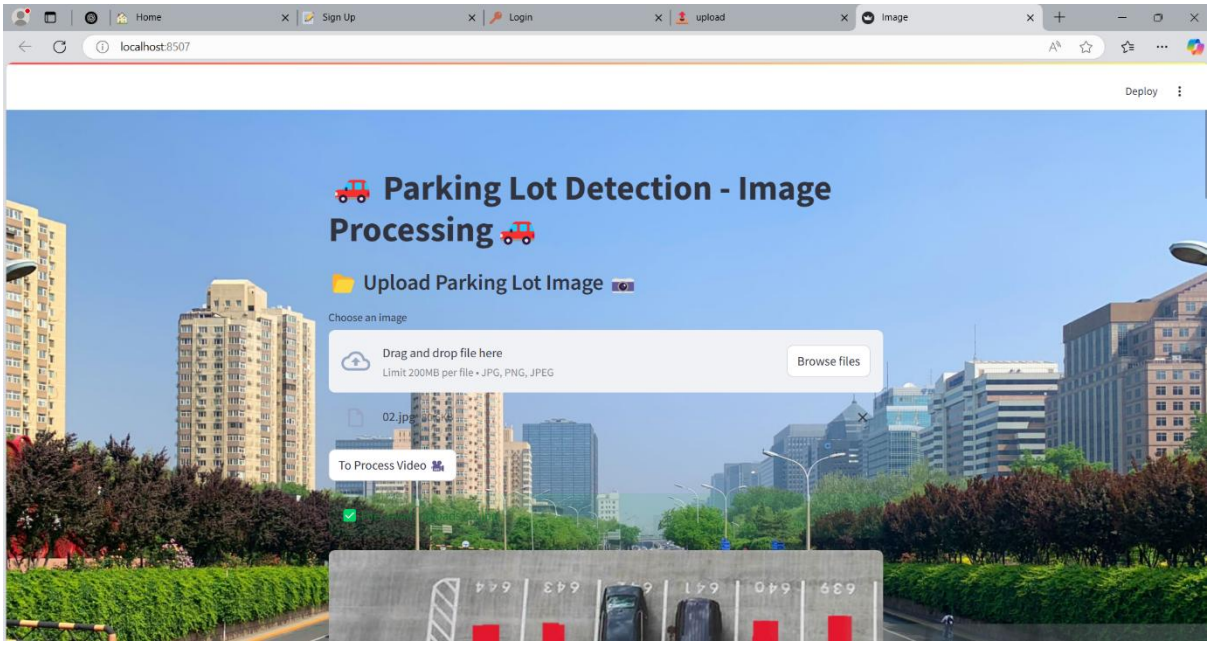
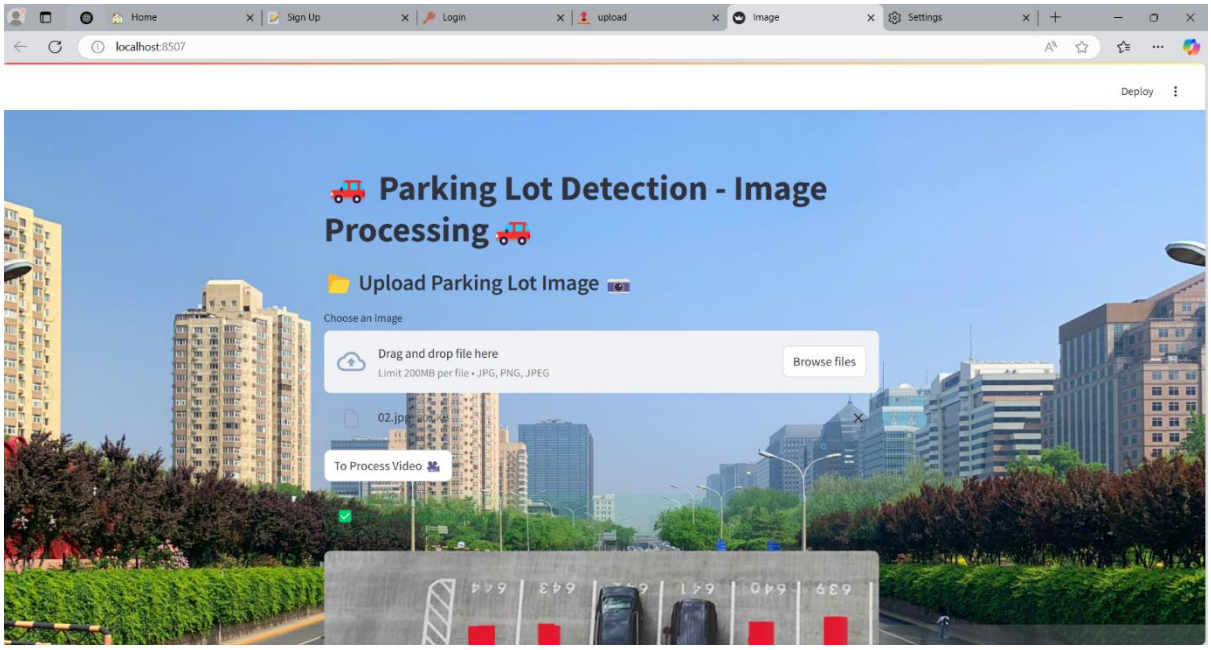
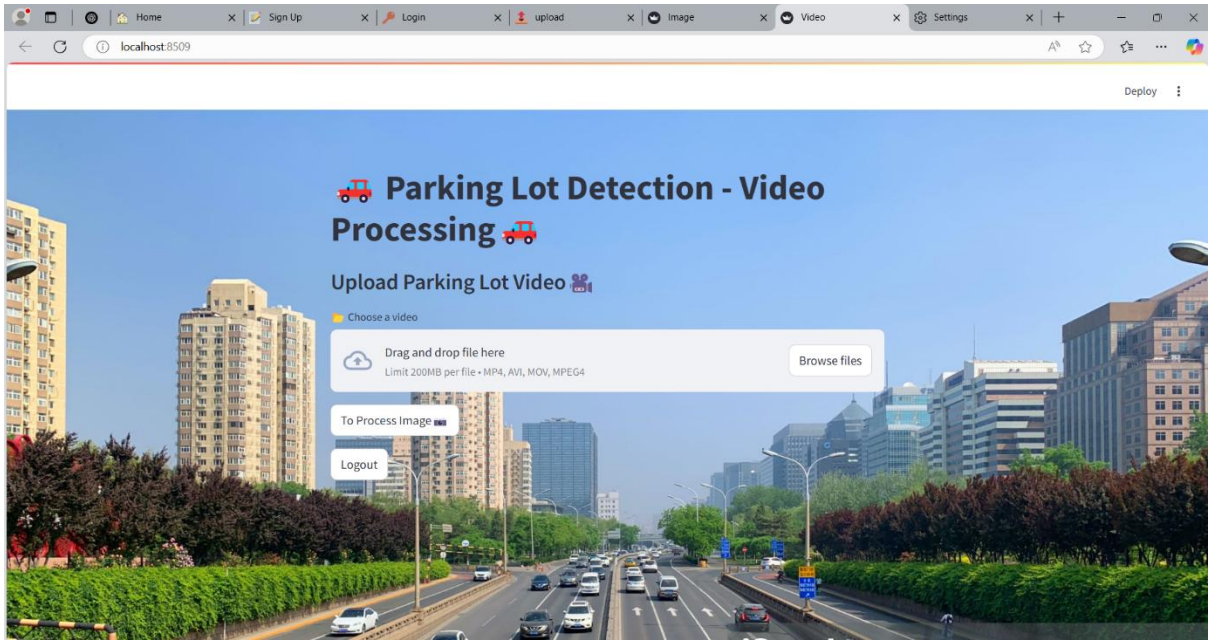


Image Uploading:

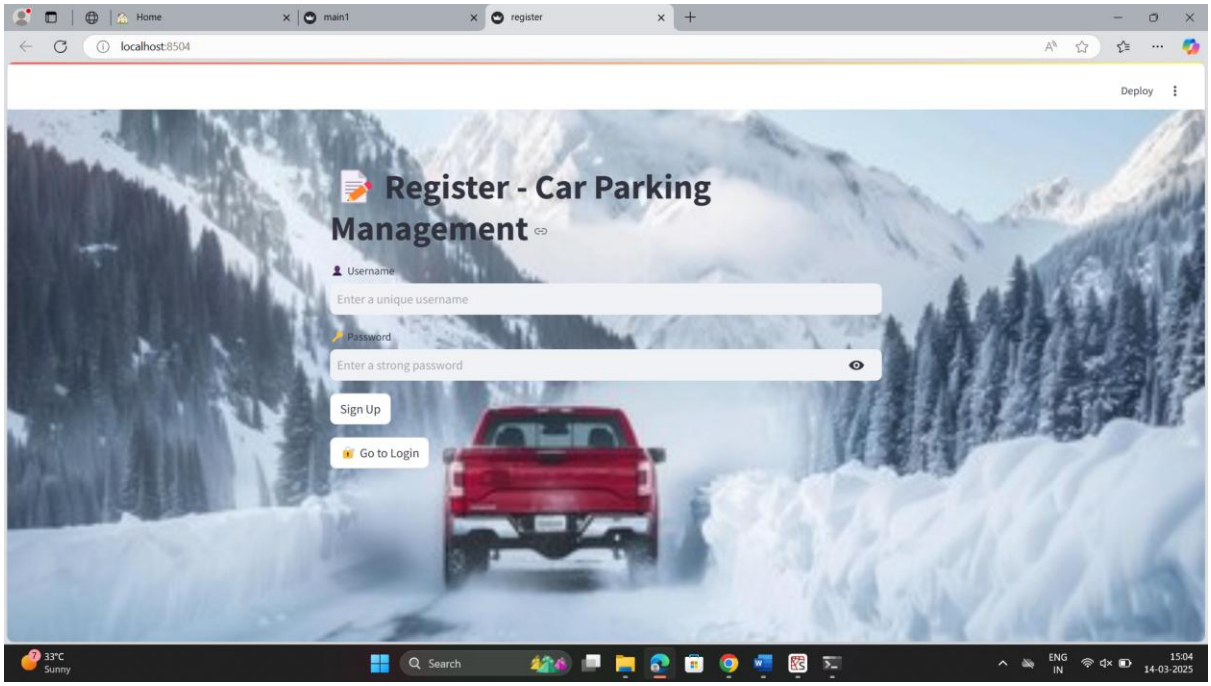


Video Processing Page:

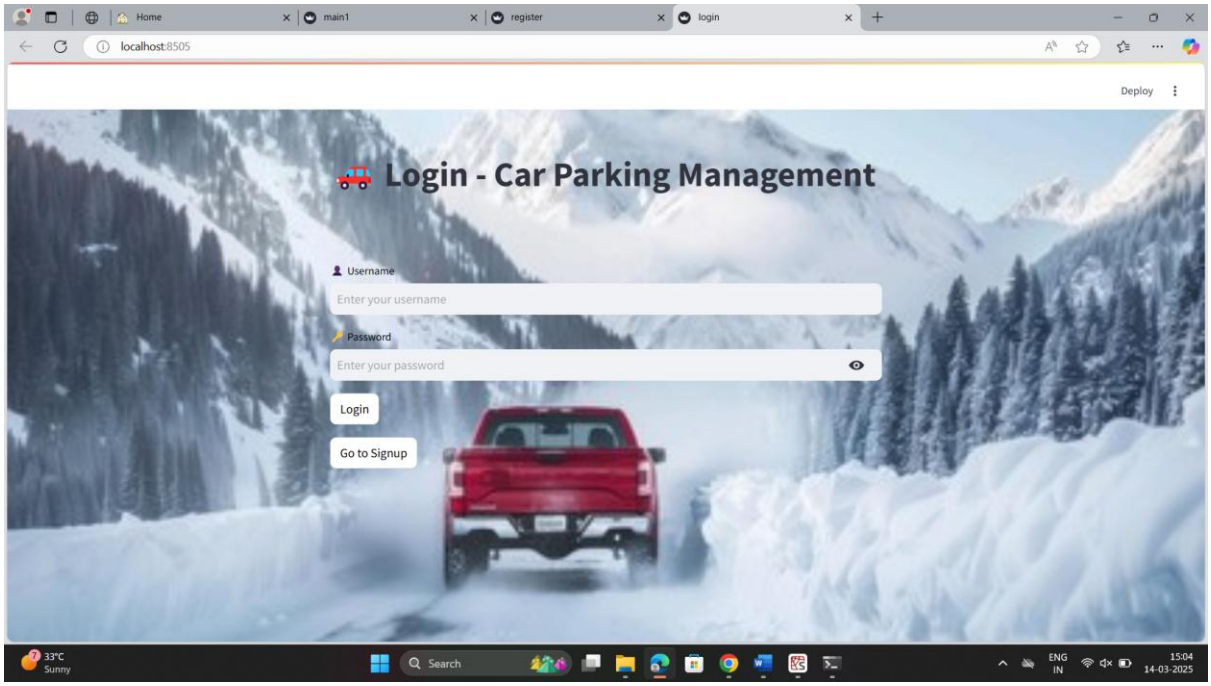


USER SIDE:

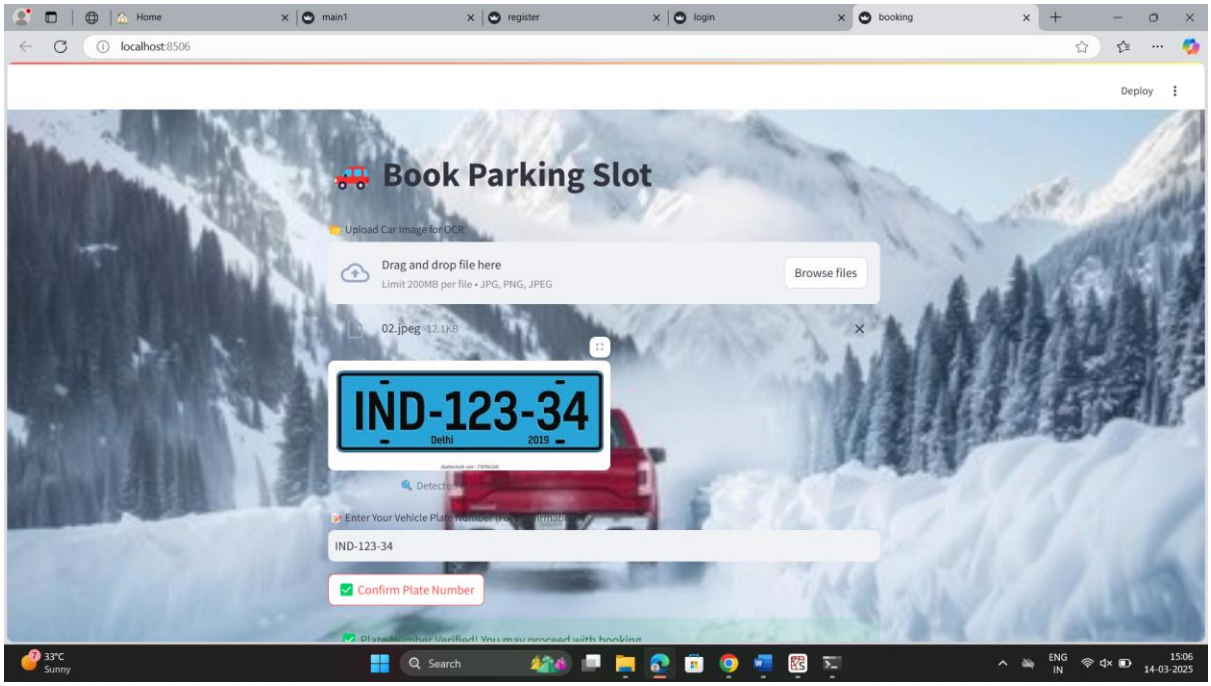
Register Page:

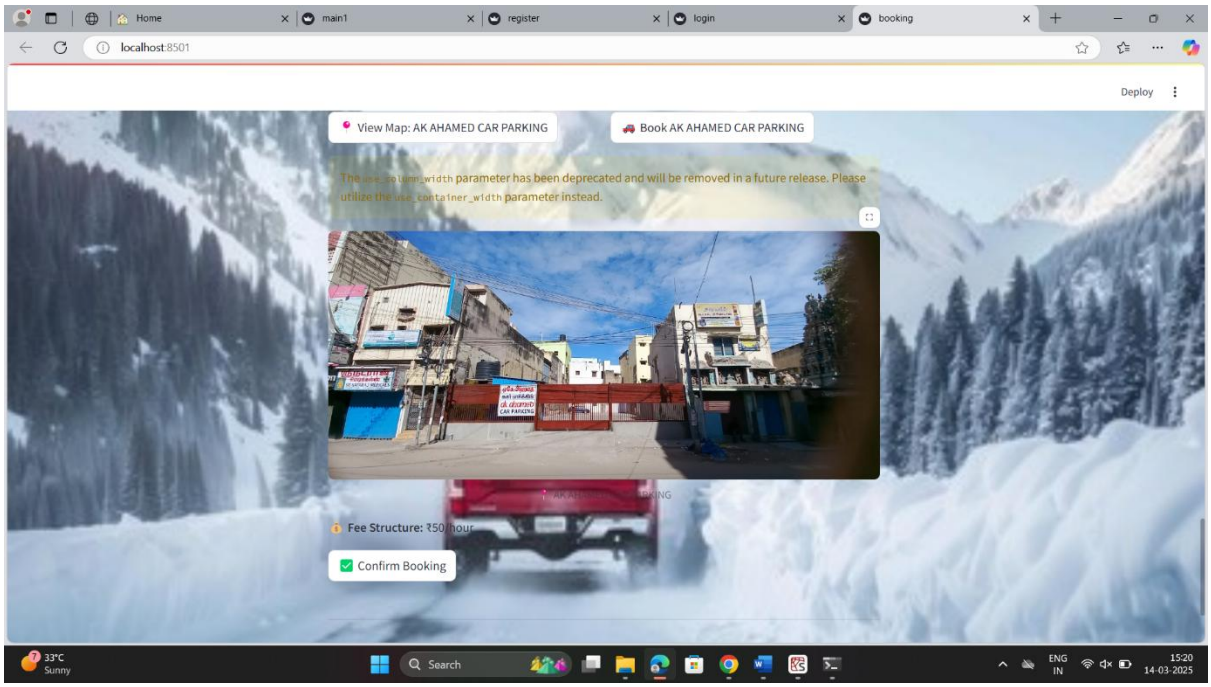


Login Page:



Booking Page:





BIBILOGRAPY

Here are some references and sources that were useful in developing the Car Parking Management System:

- Research Papers & Articles

1. Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016).
   You Only Look Once: Unified, Real-Time Object Detection. IEEE Conference on Computer Vision and Pattern Recognition (CVPR).
   Available at: https://arxiv.org/abs/1506.02640
2. Bochkovskiy, A., Wang, C. Y., & Liao, H. Y. M. (2020).
   YOLOv4: Optimal Speed and Accuracy of Object Detection.
   Available at: https://arxiv.org/abs/2004.10934
3. Gonzalez, R., & Woods, R. (2018).
   Digital Image Processing (4th Edition). Pearson Education.
4. Dey, A., & Shrivastava, A. (2018).
   Machine Learning Approaches for Smart Parking Detection Using Computer Vision.

- Online Documentation & Tutorials

5. Ultralytics YOLO Documentation
   Available at: https://docs.ultralytics.com
6. OpenCV Python Documentation
   Available at: https://docs.opencv.org
7. Streamlit Documentation – Web App Development
   Available at: https://docs.streamlit.io
8. SQLite Database Documentation
   Available at: https://www.sqlite.org/docs.html

- Related GitHub Repositories

9. YOLO Object Detection Models (Ultralytics)

📌 Available at: https://github.com/ultralytics/yolov5

10. Smart Parking Systems Using Computer Vision

📌 Available at: https://github.com/topics/smart-parking