Alexandria University

Faculty of Computers and Data Science

Department of Data Science

Smart Parking System

Omar Mohamed Mostafa	2022446471
Abdulrahman Salah Anwar	20221458503
Mahmoud Reda Hassan	20221469438
Mohamed Yousri Ibrahim	20221509866
Abu-Bakr Mohamed Mahmoud	20221458962
Fares Mohamed Fathy	20221461330

Project Supervisor

Dr. Mahmoud Gamal

Table of Contents

Acknowledgment

We extend our heartfelt gratitude to our esteemed supervisor, **Dr. Mahmoud Gamal**, for his invaluable guidance, support, and encouragement throughout the course of this project. His expertise and insights have been instrumental in shaping the "El-Sayes" Smart Parking System, from its initial concept to its successful realization.

Dr. Mahmoud Gamal's mentorship not only provided us with technical direction but also inspired us to approach challenges with creativity and determination. His constructive feedback and unwavering belief in our abilities have motivated us to push boundaries and achieve our goals. We are truly grateful for his dedication to our academic and professional growth.

We also extend our sincere thanks to the **Faculty of Computers and Data Science** for the comprehensive academic content and practical training provided during our studies. The knowledge and skills imparted by the college have been fundamental in helping us understand critical concepts and features, enabling us to develop this innovative idea. The resources and support offered by the faculty played a vital role in the successful completion of this project.

This project stands as a testament to the collective knowledge, guidance, and inspiration we have gained, and we are deeply appreciative of the contributions that made it possible.

Abstract

The "El-Sayes" Smart Parking System project addresses the growing challenges of urban parking by integrating cutting-edge technologies such as Internet of Things (IoT), machine learning, and real-time data analytics. This innovative solution aims to mitigate issues like traffic congestion, inefficient space utilization, security vulnerabilities, and environmental damage caused by excessive fuel consumption during parking searches. Key features include real-time parking slot monitoring, automated gate operations with dual-layer authentication, and a centralized database powered by Supabase for seamless synchronization and management.

User-centric insights were gathered through extensive surveys with car owners and garage operators, revealing critical pain points such as parking availability (a problem for over 91.6% of respondents), delays at entry and exit points, and security concerns. These insights informed the design of a comprehensive system integrating hardware—like IR sensors and ESP32 controllers—with intuitive mobile and web applications, ensuring convenience and reliability for both users and administrators.

The project's innovative business model offers multiple revenue streams, including subscription plans, pay-per-use fees, and data analytics services. Leveraging YOLOv8 for real-time license plate recognition and advanced workflow integration, the system ensures accuracy, scalability, and operational efficiency. Additionally, future enhancements include multi-garage integration, EV charging infrastructure, and sustainability initiatives like solar-powered systems, aligning with smart city goals and global environmental standards.

By blending technological innovation with a deep understanding of urban mobility challenges, the "El-Sayes" Smart Parking System redefines the parking experience, paving the way for smarter, safer, and more sustainable cities.

Chapter One

Introduction to the Project

1.Chapter One (Introduction)

1.1 Introduction

In today's fast-paced urban environments, the challenges of traffic congestion and inefficient parking management have become pressing issues that impact the daily lives of millions of people. Parking-related problems, such as limited availability, lengthy searches for open spots, and lack of security, contribute to increased frustration, wasted time, and environmental harm from unnecessary fuel consumption. Recognizing the critical need for innovative solutions, the **Smart Parking System Project "El-Sayes"** was conceived to revolutionize the parking experience.

El-Sayes integrates advanced technologies like IoT, machine learning, and real-time data analytics to provide a seamless and secure parking management solution. Designed to cater to both car owners and garage business operators, the system aims to streamline parking operations, enhance user convenience, and optimize resource utilization. By addressing key pain points in current parking practices, El-Sayes represents a transformative approach to solving one of the most persistent urban challenges.

1.2 Overview

The **Smart Parking System Project "El-Sayes"** is a groundbreaking solution aimed at addressing the inefficiencies and frustrations associated with traditional parking systems. By leveraging state-of-the-art technologies, El-Sayes creates an interconnected ecosystem that transforms how parking facilities operate and how users interact with them. The system integrates IoT-enabled sensors to monitor parking space availability in real time, machine learning algorithms to optimize resource usage, and user-friendly mobile and web applications to provide a seamless experience for car owners and garage operators. With a focus on convenience, security, and efficiency, El-Sayes redefines the urban parking landscape, making it smarter, safer, and more sustainable.

1.3 Problem Statement

The traditional parking ecosystem is plagued by numerous challenges, including:

- **Traffic congestion:**
  - caused by vehicles searching for available spaces.
- **Inefficient space utilization:**
  - leading to underperforming facilities.

- **Lack of security:**
  - with unauthorized access and theft concerns.
- **Manual operations:**
  - resulting in errors, delays, and customer dissatisfaction.
- **Environmental impact:**
  - due to increased emissions from vehicles idling while searching for parking.

These challenges highlight the urgent need for a more efficient and user-centric approach to parking management, paving the way for innovative solutions like El_Sayes.

1.4 Key Problems

1. Limited real-time visibility of parking space availability.
2. Lengthy and frustrating parking search times.
3. Lack of security measures to prevent unauthorized access.
4. Manual, error-prone payment and entry processes.
5. Insufficient data-driven insights for garage operators to improve efficiency and profitability.

1.5 Solutions

The Smart Parking System provides innovative solutions to these problems:

- **Real-Time Slot Availability:**
  - Users can check and reserve parking spots via a mobile app, reducing search time and frustration.
- **Automated Entry and Exit:**
  - Dual **ESP32-CAM** modules are installed at the entrance and exit gates to capture vehicle license plates. These images are automatically uploaded to the backend and processed using an OCR model, eliminating the need for RFID cards or manual logging.
- **Enhanced Security:**
  - License plate recognition ensures that only authorized or registered vehicles can access reserved slots. All vehicle movements are logged with timestamps and photographic evidence, improving traceability and security.
- **Flexible Payments:**
  - Users can choose from various payment methods, including online and cashless options.

- **Mobile & Web Integration**
  - A dedicated **mobile application** allows car owners to reserve slots in advance, while a **web-based dashboard** provides garage administrators with real-time control over parking operations, reservations, occupancy data, and vehicle entry/exit history.
- **Comprehensive Analytics:**
  - The **admin dashboard**, powered by Supabase, offers insights into parking trends, occupancy rates, reservation statistics, and entry/exit logs. This data helps optimize garage management and enhances service quality for users.

1.6 Vision

We envision a world where finding parking is no longer a hassle. Our vision is to create a stress-free, technologically advanced parking experience that saves time, reduces frustration, and contributes to a cleaner, more efficient urban environment. By seamlessly integrating cutting-edge technologies and user-friendly interfaces, El-Sayes aims to be the benchmark for modern parking solutions, driving progress and sustainability in urban mobility.

1.7 Mission

Our mission is to revolutionize parking by delivering innovative, reliable, and user-focused solutions that address the everyday challenges faced by car owners and garage operators. Through a commitment to convenience, security, and efficiency, we strive to make parking simpler, faster, and more secure, enhancing the quality of life for users and enabling smarter urban development.

1.8 Value

El-Sayes delivers unparalleled value through:

- **Convenience:**
  - Real-time updates on parking availability and seamless slot reservation via mobile and web applications.
- **Efficiency:**
  - Streamlined, fully automated entry and exit processes powered by license plate recognition, reducing congestion and eliminating the need for manual checks or RFID cards.

- **Security:**
  - Image-based vehicle tracking with timestamped records ensures reliable monitoring and prevents unauthorized access to reserved or occupied slots.
- **Cost Savings:**
  - Optimized parking space usage minimizes operational costs for garage operators while offering competitive pricing for users.
- **Innovation:**
  - Integration of ESP32-CAM, OCR models, and cloud-based backend (Supabase) demonstrates a cutting-edge approach to solving modern urban parking challenges.
- **Enhanced User Experience:**
  - Contactless, fast, and secure parking solutions that cater to modern lifestyles and expectations.

1.9 Define users and services

Users:

1. **Car Owners:**

   Individuals seeking a stress-free, secure, and convenient parking experience. Whether they are daily commuters, shoppers, or visitors, El-Sayes ensures a smooth journey from entry to exit.

2. **Garage Operators:**

   Business owners and managers looking to optimize parking space utilization, enhance customer satisfaction, and maximize revenue through advanced analytics and efficient operations.

Services:

1. **Mobile Application:**
   - A user-friendly app offering real-time parking updates, slot reservations, and flexible payment options, ensuring convenience for car owners.

1. **Web Application:**
    - A comprehensive platform for garage operators featuring analytics, slot management, and reporting tools to enhance operational efficiency.
2. **IoT Integration:**
    - Advanced technologies such as ESP32-CAM modules and OCR-based license plate recognition automate vehicle entry and exit, enhancing speed, accuracy, and reducing the need for physical identification systems.
3. **Analytics and Reporting:**
    - In-depth insights into parking trends, revenue generation, and user behavior, enabling data-driven decision-making and strategic planning.

1.10 System Development Life Cycle (SDLC)

The System Development Life Cycle (SDLC) is a structured approach that outlines the stages involved in developing the El-Sayes Smart Parking System. By adopting the **Agile methodology**, this SDLC ensures flexibility, collaboration, and iterative improvement.

Below, we present the SDLC divided into four comprehensive stages:

1.10.1 Planning

- **Objective:**
    - Establish the project's goals, scope, and roadmap.
- **Activities:**
    - Conduct a feasibility study to ensure technical, economic, and operational viability.
    - Identify stakeholder needs, including car owners and garage operators.
    - Develop a project timeline with key milestones.
    - Allocate resources, including human expertise, hardware, and software tools.
- **Deliverables:**
    - A detailed project charter and high-level plan.

1.10.2 Analysis

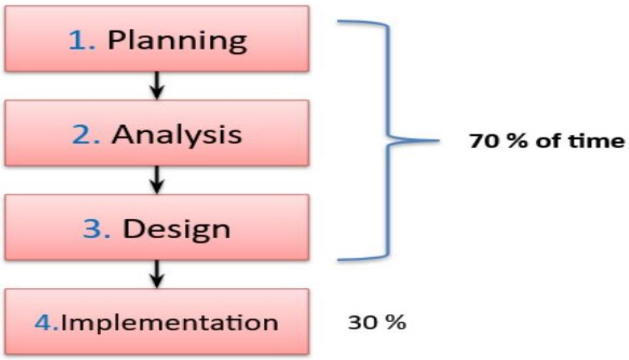
- **Objective:**
  - Define user requirements and understand the system's needs.
- **Activities:**
  - Collect user feedback through surveys and interviews.
  - Document functional requirements (e.g., slot reservation, flexible payments) and non-functional requirements (e.g., scalability, security).
  - Analyze competitors and market trends to identify differentiators.
- **Deliverables:**
  - A comprehensive requirement specification document.

1.10.3 Design

- **Objective:**
  - Create a blueprint for the system architecture and user interfaces.
- **Activities:**
  - Design system components, including IoT sensors, databases, and APIs.
  - Develop wireframes and prototypes for mobile and web applications.
  - Create detailed workflows for entry, exit, and payment processes.
- **Deliverables:**
  - Prototypes, system architecture diagrams, and interface designs.

1.10.4 Implementation

- **Objective:**
  - Build and deploy the system components.
- **Activities:**
  - Develop IoT-based features, such as real-time parking availability.
  - Build mobile and web applications using frameworks like Flutter and React.
  - Test and integrate license plate recognition and ESP32 CAM recognition.
  - Deploy the system in real-world environments.
- **Deliverables:**
  - A fully functional system ready for operational use.



1.11 Why Agile?

Agile is recommended for this project because it allows:

1. **User-Centric Development:**
    - Frequent feedback ensures the system meets user needs.
2. **Flexibility:**
    - Iterative processes adapt to changing requirements or challenges.
3. **Collaboration:**
    - Encourages teamwork and open communication among stakeholders.
4. **Faster Delivery:**
    - Delivers functional components quickly, enhancing time-to-market.
5. **Continuous Improvement:**
    - Regular iterations help refine the system and maintain alignment with user expectations.

By adhering to this SDLC and leveraging the Agile methodology, the El-Sayes project ensures a high-quality, user-friendly solution that effectively addresses modern parking challenges.

Chapter Two

The System Development Life Cycle

“Planning”

2.Chapter Two (Planning)

2.1 Project Methodology

The planning stage of the El-Sayes Smart Parking System focuses on defining the core components and how they integrate to provide a seamless, efficient, and user-centric parking experience. This methodology emphasizes the use of IoT, machine learning, Firebase integration, and mobile and web applications to create a robust and scalable system. Below is a detailed breakdown of each component, its functionality, and the reasons for its inclusion in the project.

2.1.1 IoT Integration

1. Slot Management System

- **Components:** IR sensors, ESP32 modules, and addressable LEDs.
- **Functionality:**
  - Each parking slot is equipped with an IR sensor to detect the presence of a vehicle.
  - When a car occupies a slot, the sensor signals the ESP32 module to turn the LED red, indicating the slot is occupied. When empty, the LED remains green. If the slot is reserved but not yet occupied, the LED turns yellow, signaling the reservation status to other users.
- **Reason for Use:**
  - Real-time monitoring of slot availability provides instant updates to users and administrators.
  - Simple and cost-effective hardware ensures scalability.

1.2 Gate Management System

- **Components:** IR sensors, ESP32 modules (for gate control), ESP32-CAM modules (for license plate capture), servo motors, and LCD display.
- **Functionality:**
  - **At entry:**
    - IR sensors detect the car and trigger the ESP32 to record the start time and activate the camera module.
    - The ESP32 sends a signal via ESP-NOW to the entrance ESP32-CAM to capture the car's license plate.
    - The captured image is uploaded to **Supabase Storage**, and a timestamp is recorded

        - A backend process (Python OCR model) extracts the plate number and logs it into the Supabase database.
        - If slots are available, the gate opens automatically.
    - **At exit:**
        - IR sensors detect the car
        - The exit ESP32 sends a signal to the exit ESP32-CAM to capture the car plate.
        - Image is uploaded and processed similarly via OCR to identify the plate number.
        - The system calculates total parking duration based on entry time, logs exit time, and stores the record in Supabase.
        - The gate opens once the car is verified to exit.
- **Reason for Use:**
    - Fully automates the entry and exit process using camera-based recognition instead of RFID.
    - Improves efficiency by reducing manual checks and enhances security through license plate logging and timestamped records.
    - Provides accurate vehicle tracking and seamless gate control.

1.3 IoT Integration with Supabase Database

- **Functionality:**
    - The system is designed to leverage IoT devices (ESP32 and ESP32-CAM modules) to collect real-time data
    - This data will be transmitted to a centralized cloud database (**Supabase**) to ensure synchronized and consistent system behavior.
    - The architecture will support communication between the gate control units and camera modules, while also enabling backend services to process and store relevant parking records and status updates.
    - Supabase will serve as the main backend platform for structured data storage, media uploads, user access, and real-time synchronization across the system.
- **Reason for Use:**
    - Supabase offers a robust, scalable, and developer-friendly backend that integrates seamlessly with modern IoT systems.
    - Its real-time database updates, RESTful APIs, and media storage capabilities make it ideal for managing dynamic parking data and integrating mobile and web applications.

2.1.2 Machine Learning Integration

Car Plate Recognition

- **Functionality:**
  - A trained machine learning model extracts text from images captured by the camera module.
  - The extracted text (plate number) is sent to Firebase or the web application database.
- **Reason for Use:**
  - Automates vehicle identification, reducing errors and speeding up entry/exit processes.
  - Enhances security by cross-referencing plate numbers with user accounts.

2.1.3 Web Application

Features for Administrators

- **Dashboard:** Displays key metrics such as the total number of cars, revenue collected, and slot availability.
- **Queue Management:** Tracks cars currently in the garage and those with active reservations.
- **Car List:** Maintains a database of all cars that have entered the garage, including details like plate number, entry/exit times, and payment history.
- **Analysis and Reporting:**
  - Provides insights into garage usage, peak times, and revenue trends.
  - Generates daily and monthly reports in downloadable formats.
- **Reason for Use:**
  - Empowers garage operators with tools to optimize operations.
  - Data-driven insights enable better decision-making and resource allocation.

2.1.4 Mobile Application

Features for Users

- **Garage Details:** Shows real-time availability of slots with visual indicators for empty and occupied spaces.
- **Slot Reservation:**
    - Allows users to book a slot by selecting a payment method and specifying the expected arrival time.
    - Reservations are synchronized with Firebase for real-time updates.
- **Payments:** Supports multiple methods, including online and prepaid card options.
- **Notifications:** Sends reminders for reservations, payment confirmations, and alerts about expiring bookings.
- **Reason for Use:**
    - Enhances user convenience by reducing the time spent searching for parking.
    - Streamlines the booking and payment process, improving overall user satisfaction.

2.1.5 Supabase Integration

Role of Supabase

- **PostgreSQL Database:** Provides a structured and scalable relational database to store essential data, including vehicle logs, parking slot status, reservations, and user accounts.
- **Real-Time Updates:** Supports instant synchronization of parking slot availability, vehicle entry and exit records, and reservation changes between IoT devices, mobile apps, and the admin dashboard.
- **Authentication:** Manages secure user login and implements role-based access control for different system users, such as car owners and garage administrators.
- **Storage:** Hosts and organizes images captured by ESP32-CAM modules, primarily for license plate recognition and record-keeping.
- **APIs and Backend Integration:** Offers RESTful APIs and seamless connectivity with custom backend services (e.g., OCR processing), enabling dynamic and automated data workflows.

Reason for Use:

- Provides a flexible, developer-friendly backend with real-time capabilities tailored to IoT systems.
- Ensures efficient data management and synchronization across all system components.
- Offers full SQL-level control over the data model, supporting future scalability and integration with additional features.

2.1.6 Integration and Simulation

Integration Workflow:

At Entry:

- IR sensors detect the approaching vehicle and initiate data capture.
- The ESP32-CAM captures an image of the car's license plate.
- The image is uploaded to Supabase Storage and processed by a backend OCR model to extract the plate number.
- The system checks the extracted plate against existing records or reservations in the Supabase database.
- If a valid reservation or existing vehicle record is found, the entry gate opens automatically. If not, a new entry record is created.

During Parking:

- Slot monitoring ESP32 devices continuously update the is_occupied status of each slot in the Supabase slots table.
- Changes are reflected in real time on the mobile application and web dashboard.

At Exit:

- IR sensors detect the car at the exit gate and trigger a new plate capture using the exit ESP32-CAM.
- The image is processed similarly via OCR, and the system matches the plate to the entry record.
- Parking duration is calculated, and a log entry is recorded in Supabase.
- The gate opens automatically once the car is verified to exit.

Simulation:

IoT Devices:
Simulations are performed for slot status changes and gate behavior to verify hardware-software synchronization.

Database Updates:
Tests ensure real-time synchronization of slot occupancy, car logs, and reservation statuses within the Supabase database.

Applications:
Mobile and web platforms are tested for end-to-end functionality, including reservation flow, gate events, status updates, and administrative analytics.

2.2 Schedule

2.2.1 Phase 1: Planning

- **Objective:** Define the project's goals, scope, and feasibility.
- **Key Activities:**
  - Stakeholder meetings with car owners and garage operators.
  - Requirement gathering for functional and non-functional needs.
  - Feasibility study to assess technical, operational, and financial viability.
  - Resource allocation for team roles, hardware, and tools.
  - Development of a project roadmap.
- **Deliverables:** Project charter, high-level plan, and feasibility report.

2.2.2 Phase 2: Design

- **Objective:** Create a detailed blueprint for the system architecture and user interfaces.
- **Key Activities:**
  - Design of IoT integrations, databases, and backend services.
  - Workflow mapping for slot reservation, entry/exit, and payment processes.
  - Development of wireframes and prototypes for mobile and web applications.
  - Security framework design for encryption, authentication, and access control.
- **Deliverables:** System architecture diagrams, process workflows, and prototypes.

2.2.3 Phase 3: Development

Objective: Build and integrate system components based on the design specifications.

Week 1–2: IoT Integration Development

- Program IR sensors and ESP32 modules for parking slot detection.
- Configure ESP32-CAM modules to capture license plate images at entry and exit points.
- Test WS2811 LED indicators for reserved, occupied, and empty slot statuses.
- Set up real-time communication between ESP32 devices and Supabase for live data updates.

Week 3–4: Mobile Application Development

- Develop key user features such as viewing available slots, making reservations, and receiving notifications.
- Integrate Supabase for user authentication and real-time data access.
- Ensure cross-platform functionality for both Android and iOS devices.

Week 5–6: Web Application Development

- Build a web-based dashboard for administrators to monitor garage status and access reports.
- Implement user and slot management features.
- Add analytics tools to display parking trends and usage statistics.

Week 7–8: Backend Integration with Supabase

- Design and deploy the Supabase database schema to support all entities (vehicles, users, slots, logs, reservations).
- Connect OCR backend services for automatic plate number extraction and record insertion.
- Implement authentication, media storage, and API endpoints using Supabase features.

Week 9: Component Integration

- Ensure seamless interaction among ESP32 devices, mobile and web applications, and the Supabase backend.
- Conduct integration testing to validate end-to-end workflows, including reservation, entry, exit, and logging.

Week 10: Unit Testing

- Independently test each component: IoT modules, mobile app screens, and web features.
- Debug and optimize isolated modules for accuracy and performance.

Week 11: Integration Testing

- Test the full system for real-time synchronization, workflow coordination, and reliability.
- Simulate edge cases to evaluate system response and data consistency.

Week 12: User Acceptance Testing (UAT)

- Conduct testing sessions with end users (garage admins and car owners).
- Gather feedback on usability, functionality, and performance.
- Make adjustments to enhance user experience and system stability.

2.2.4 Phase 4: Testing and Deployment

- **Objective:** Validate and launch the fully developed system.

Testing Phase:

- Performance testing to ensure scalability and reliability.
- Security testing for data protection and access control.

Deployment Phase:

- Hardware installation in garages.
- Launch of mobile and web applications.
- Training for garage operators and support setup.

2.2.5 Phase 5: Maintenance and Updates (Future)

- **Objective:** Ensure system reliability and implement continuous improvements.
- **Key Activities:**
  - Monitor system performance and user activity.
  - Collect feedback for enhancements.
  - Roll out periodic updates and introduce new features as needed.

2.3 Time Estimation

Planning Phase:

Tasks	Duration	Start Date	End Date	Status
• Stakeholder meetings • requirements gathering • feasibility study • project roadmap creation	2 Weeks	29 Oct 2024	12 Nov 2024	Completed

Design Phase:

Tasks	Duration	Start Date	End Date	Status
• System architecture design • workflow mapping • wireframe development • prototype creation	3 Weeks	13 Nov 2024	3 Dec 2024	Completed

Development Phase:

Tasks	Duration	Start Date	End Date	Status
• IoT Integration Development: ○ Program IoT devices ○ establish real-time sync	2 Weeks	4 Dec 2024	17 Dec 2024	Completed
• Mobile Application Development: ○ Build features ○ Firebase integration ○ testing	2 Weeks	18 Dec 2024	1 Jan 2025	Completed
• Web Application Development: ○ Dashboard ○ user management ○ analytics	2 Weeks	2 Jan 2025	15 Jan 2025	Ongoing
• Backend Integration with Supabase: ○ Set up database ○ cloud functions ○ authentication	2 Weeks	16 Jan 2025	29 Jan 2025	Ongoing
• Component Integration: ○ Seamless interaction of subsystems	1 Week	30 Jan 2025	5 Feb 2025	Upcoming
• Unit Testing: ○ Test individual components	1 Week	6 Feb 2025	12 Feb 2025	Upcoming
• Integration Testing: ○ Verify subsystem interaction	1 Week	13 Feb 2025	19 Feb 2025	Upcoming
• User Acceptance Testing (UAT): ○ Feedback-based refinements	1 Week	20 Feb 2025	26 Feb 2025	Upcoming

Testing and Deployment Phase:

Tasks	Duration	Start Date	End Date	Status
• Performance and Security Testing: ○ Scalability ○ security validation	2 Weeks	27 Feb 2025	12 Mar 2025	Upcoming
• Hardware Installation: ○ Deploy IoT devices ○ set up infrastructure	1 Week	13 Mar 2025	19 Mar 2025	Upcoming
• Application Launch: ○ Release mobile ○ web apps	1 Week	20 Mar 2025	26 Mar 2025	Upcoming
• Maintenance and Updates: ○ Monitor ○ collect feedback ○ implement updates	Ongoing	27 Mar 2025	Continuous	Post-deployment

2.4 Task Identification

Planning Phase:

Tasks	Assigned to	Status
• Stakeholder meetings • requirements gathering • feasibility study • project roadmap creation	All team	Completed

Design Phase:

Tasks	Assigned to	Status
• System architecture design • workflow mapping • wireframe development • prototype creation	All team	Completed
• Database Schema Design	Abdulrahman Salah	Completed
• UI/UX Design	Fares Mohamed Adu-Bakr Mohamed	Completed

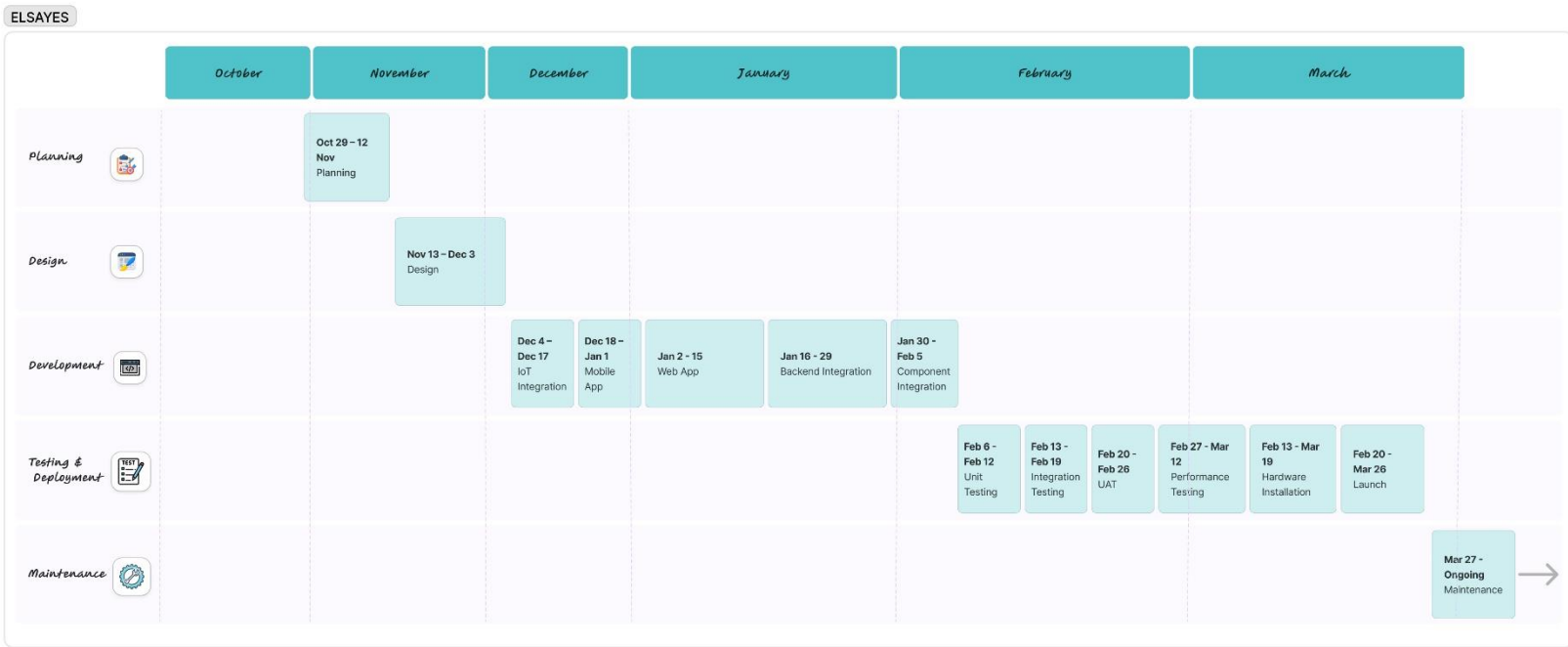
Development Phase:

Tasks	Assigned to	Status
• IoT Integration Development	Omar Mohamed Mahmoud Reda	Completed
• IoT Supabase integration	Omar Mohamed Mahmoud Reda	Completed
• Mobile Application Development	Fares Mohamed	Completed
• Web Application Development	Abu-Bakr Mohamed	Completed
• Backend Integration with Firebase	Abdulrahman Salah Abu-Bakr Mohamed	Completed
• Car Plate Recognition	Mohamed Yousri	Completed
• Component Integration	All team	Completed

Testing and Deployment Phase:

Tasks	Assigned to	Status
• Performance and Security	Mohamed Yousrii Abdulrahman Salah	Upcoming
• Hardware Testing	Omar Mohamed Mahmoud Reda	Upcoming
• Application Launch	Abu-Bakr Mohamed Fares Mohamed	Upcoming
• Maintenance and Updates	All team	Post-deployment

2.4 Gantt Chart



Chapter Three

The System Development Life Cycle

“Analysis”

3.Chapter Three (Analysis)

3.1 Car owners Survey

3.1.1 Introduction

Parking is a critical component of modern urban infrastructure, but it often comes with challenges that disrupt daily life for millions. Issues such as limited availability, time-consuming searches for spots, and lack of security lead to frustration, wasted time, and environmental impacts from unnecessary fuel consumption.

To better understand these challenges and tailor solutions to meet real-world needs, we conducted a survey targeting a diverse group of **190 respondents**. The survey aimed to capture insights into user behaviors, preferences, and pain points related to parking facilities.

This report analyzes the survey data to highlight current parking challenges and opportunities for improvement. It also serves as a foundation for the development of the Smart Parking System **("El-Sayes"),** which integrates IoT, machine learning, and real-time analytics to create a seamless and efficient parking experience.

3.1.2 Methodology

To ensure a comprehensive understanding of parking challenges and user preferences, a structured survey was conducted as part of the research for the Smart Parking System ("El-Sayes").

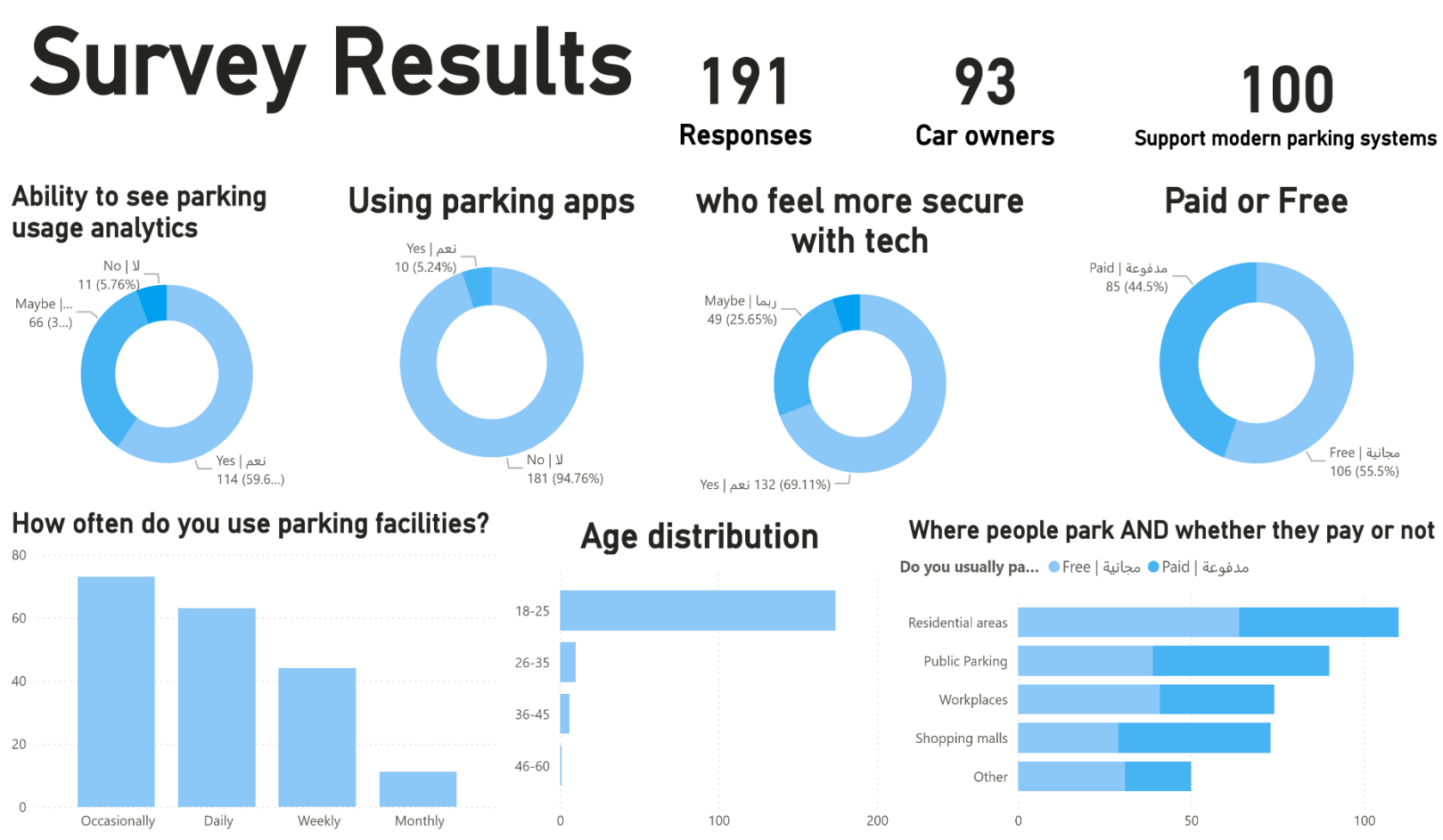
Below are the details of the methodology used:

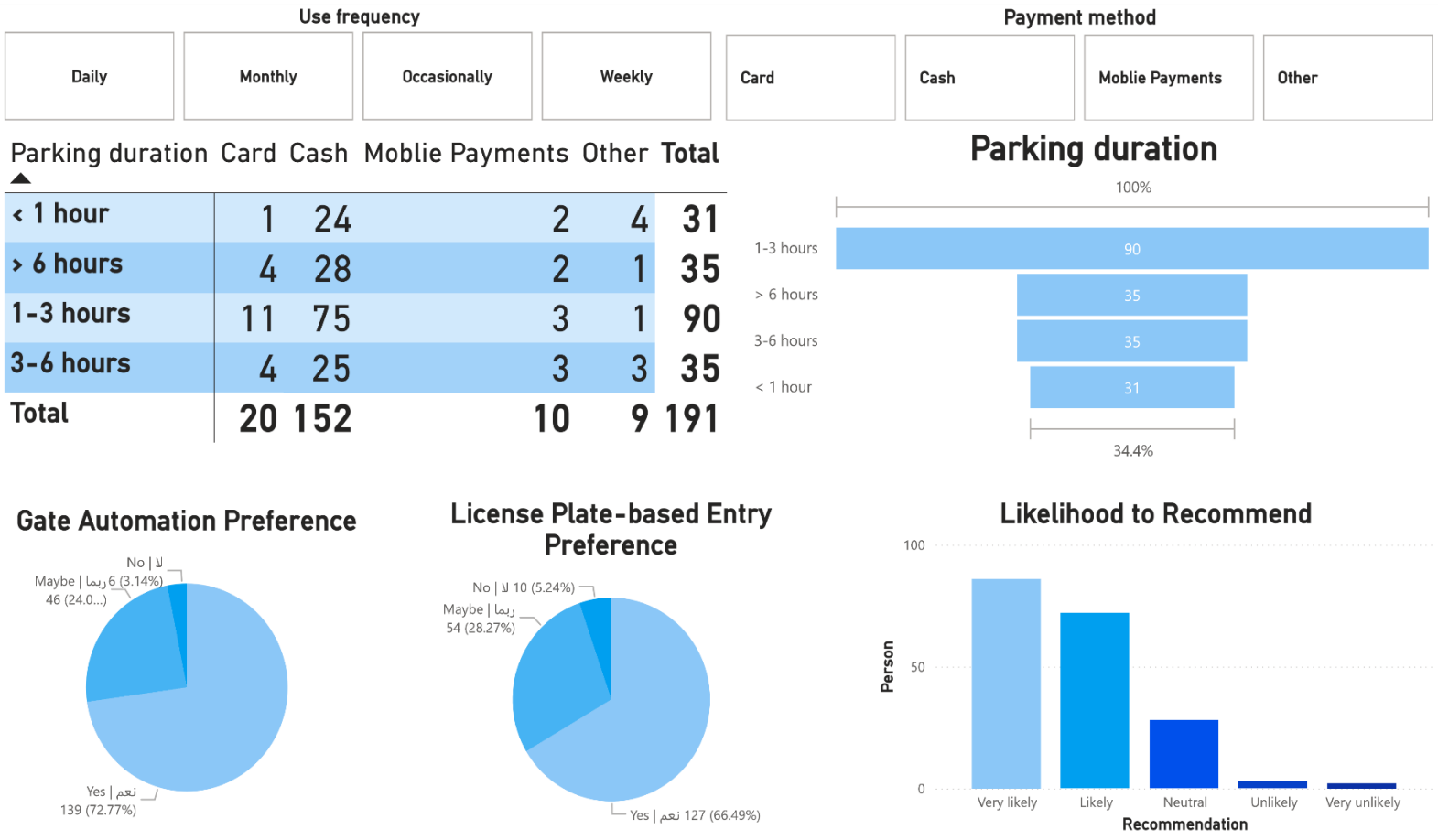
1. **Survey Design**:
    - The survey consisted of 24 structured questions, covering areas such as parking frequency, preferred payment methods, major frustrations, and interest in modern parking solutions.
    - Both multiple-choice and 2 open-ended questions were included to gather quantitative and qualitative insights.
2. **Target Audience**:
    - The survey targeted individuals aged 18 and above, with a focus on car owners and frequent parking facility users.
    - A total of 190 responses were collected from a diverse group, ensuring representation across age groups and parking usage patterns.
3. **Data Collection**:
    - The survey was distributed online through social media platforms, messaging apps, and community groups to reach a wide audience efficiently.
    - Participants were encouraged to provide honest and detailed responses to capture authentic user experiences and preferences.
4. **Data Analysis Tools**:
    - The responses were collated and analyzed using spreadsheet tools and data visualization software.
    - Metrics such as response frequency, percentages, and user feedback themes were used to interpret the results.
5. **Limitations**:
    - The sample size of 190 respondents, while significant, may not represent the entire population.
    - The online nature of the survey might have excluded individuals without internet access.

This methodology ensures that the survey results provide valuable insights into the parking challenges and expectations of users, forming a strong foundation for designing a user-centric Smart Parking System.

3.1.3 Survey Results and Analysis

The survey collected responses from 190 participants, providing valuable insights into user behaviors, preferences, and challenges related to parking facilities. Below is a detailed analysis of the results:





3.1.4 Detailed Analysis of Survey Data

- **Overview of Respondent Demographics and Behavior**
  - **Car Ownership**: Approximately 51.6% of respondents own a vehicle, while 48.4% do not. This balance ensures the system is designed to serve both current drivers and potential future users.
  - **Age Group**: Over 90% of participants fall within the 18–25 age range. This youth dominance indicates a strong preference for tech-integrated, mobile-friendly, and user-centric designs.
  - **Parking Frequency**: About 38% of users park daily and 38% occasionally, showing a significant dependence on accessible and reliable parking facilities.
  - **Parking Locations**: Residential areas (57.4%) and public lots (47.4%) are the most common, which guides the initial deployment strategy for the system.

- **Parking Duration**: More than 46% park for over 6 hours, with a notable segment (18.4%) parking between 1–3 hours. These insights can shape time-based pricing models or subscription tiers.

- **Pain Points and User Preferences**
  - **Availability Issues**: A striking 91.6% of respondents experience difficulty finding available parking, reinforcing the need for real-time slot monitoring and availability alerts.
  - **Payment Preferences**: Despite increasing digitalization, 79.5% still prefer to pay by cash. This signals the need for offering flexible payment methods—including both traditional and modern options.
  - **Security Concerns**: Over 36% feel insecure leaving their cars in public areas. Features like camera surveillance, smart gate control, and license plate recognition could greatly enhance trust.
  - **Delays at Entry/Exit**: 63.7% report experiencing delays, indicating a need for automated gate systems to improve flow and reduce congestion.
- **Interest in Smart Features**
  - **Smart Entry**: 95% of respondents believe that automated gates and license plate recognition would save time, confirming strong support for advanced access systems.
  - **Parking Analytics**: Over 60% of users expressed interest in receiving usage analytics via a mobile app. The Power BI dashboard developed during the project visually demonstrates how this can be implemented, offering reports on session duration, usage frequency, and location-specific trends.
  - **Current Tech Usage**: Despite high support for technology, 94.7% of users do not currently use parking apps—highlighting a gap in adoption and a major opportunity for a well-designed, intuitive application.

- **Strategic Implications**

  Based on the combined survey results and Power BI analysis, the following conclusions guide the Smart Parking System's development roadmap:

  - **Focus on Young, Mobile-First Users**: The system should prioritize mobile compatibility, fast interfaces, and visual feedback tailored for younger users who are already comfortable with digital tools.
  - **Introduce Real-Time Slot Tracking and Availability Alerts**: Since availability is the top challenge, accurate and live feedback through LED indicators and mobile updates is essential.
  - **Deploy Smart Entry/Exit Systems with License Plate Recognition**: This reduces human dependency, enhances security, and aligns with the 95% support for automation.
  - **Incorporate Parking Analytics and Reports**: Enable users to review their own behavior and optimize usage while giving administrators powerful tools for planning and policy.
  - **Leverage Free or Freemium Models**: Since users lean toward free parking, introducing low-cost options with premium features can ease adoption and increase engagement.
  - **Educate and Onboard**: With low current usage of parking apps, onboarding tutorials, incentives, and seamless UI design will be key to user retention and growth.

3.1.5 Conclusion

The survey data identifies critical pain points and preferences among car owners:

1. **Key Challenges:** Parking availability, security, and delays at entry/exit points.
2. **Preferred Features:** Automated gates, real-time updates, and parking usage analytics.
3. **Opportunities:** Address the gap in digital parking solutions and promote alternatives to cash payments.

3.1.6 Recommendation

1. Focus on **automated systems** to address entry/exit delays and enhance the user experience.
2. Incorporate **advanced security features** to build trust among users.
3. Design a **user-friendly app** with real-time parking updates, reservation options, and multiple payment methods.
4. Target **residential and public parking areas** as primary implementation zones for maximum adoption.
5. Offer **subscription plans** for long-term users, aligning with their parking duration preferences.

3.2 Logo and Design

3.3 Functional Requirements

1. **Real-Time Parking Slot Availability:**
   - The system must display live updates on parking slot availability via the mobile and web applications. This feature allows users to view available parking spots instantly, reducing time spent searching for spaces.
2. **Slot Reservation:**
   - Users must be able to reserve parking slots in advance through the app. This reservation system ensures a hassle-free parking experience by securing a spot before arrival.
3. **Automated Entry and Exit:**
   - Integration with license plate recognition to automate gate operations, minimizing manual intervention and speeding up the process.
4. **Flexible Payment Options:**
   - The system should support multiple payment methods, including online payments, credit cards, and cash, catering to diverse user preferences.

5. **User Notifications:**
    - Notifications for reservation confirmations, expiration alerts, and payment receipts to keep users informed and engaged.
6. **Security Features:**
    - Dual-layer authentication using license plate recognition to prevent unauthorized access and ensure the safety of vehicles.
7. **Analytics Dashboard:**
    - For garage operators, the system must provide real-time analytics, including parking trends, revenue statistics, and occupancy rates.
8. **Customer Support Integration:**
    - A support system within the app for users to address issues or report problems, enhancing user satisfaction.

3.4 Non-Functional Requirements

1. **Scalability & Reliability:**
   The system must support a growing number of users and parking slots while maintaining high performance and ensuring 99.9% uptime.
2. **Performance & Efficiency:**
   User actions (e.g., slot reservations, gate access) should be processed within 2 seconds, with optimized server and network resource usage.
3. **Security & Compliance:**
   All user data and payment information must be encrypted and comply with relevant data protection and vehicle identification regulations.
4. **Usability & Compatibility:**
   Interfaces should be intuitive for all user levels and fully compatible with major platforms, including Android, iOS, Chrome, and Safari.
5. **Maintainability & Localization:**
   The system architecture should allow for easy updates and debugging, while supporting multiple languages and regional formats for broader accessibility.

By addressing these functional and non-functional requirements, the El_Sayes Smart Parking System ensures a comprehensive, efficient, and user-friendly solution for modern parking challenges.

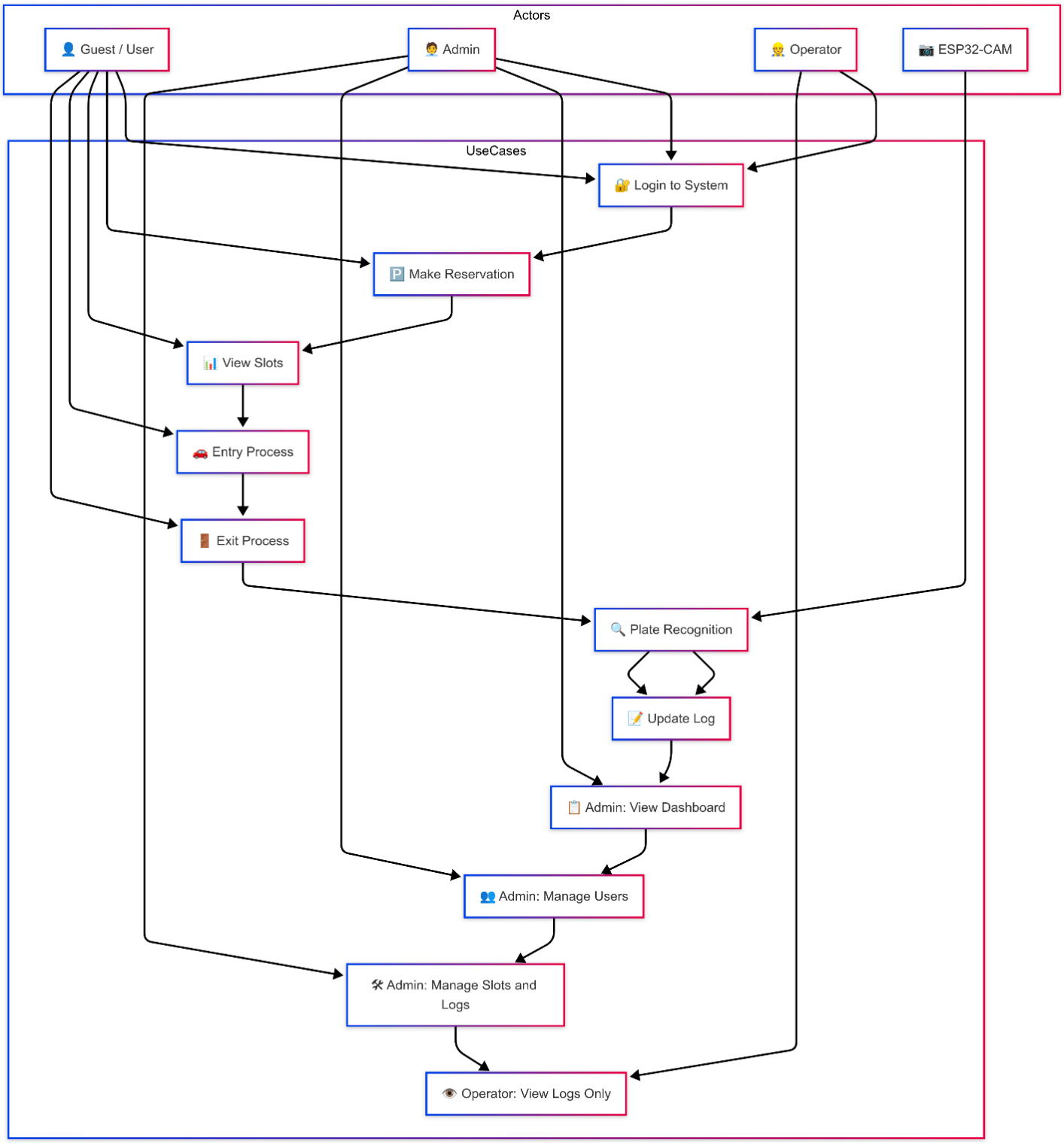
Chapter Four

The System Development Life Cycle

“Design”

4. Chapter Four (Design)

4.1 Use case



4.1.1 Login to System

- **Actors**: Guest, Admin, Operator
- **Description**: All users authenticate into the system via their assigned roles.
- **Flow**:
    1. User enters email and password.
    2. System checks credentials against users or owners tables.
    3. If valid → redirects based on role:
        - **Guest** → Reservation screen
        - **Admin** → Dashboard
        - **Operator** → Logs-only view

4.1.2 Make a Reservation

- **Actor**: Guest (mobile user)
- **Description**: User books a slot for a selected time period.
- **Flow**:
    1. Enters plate number.
    2. Selects from_time and to_time.
    3. Chooses from available slots (slots.is_occupied = false).
    4. System inserts new reservation with status = active.
    5. Slot status is updated to reserved.
- **Database Tables**:
    - cars, reservations, slots

4.1.3 View Available Slots

- **Actor**: Guest
- **Description**: Guest sees the real-time status of all parking slots.
- **Logic**:
    - Green = Available (is_occupied = false)
    - Red = Occupied (is_occupied = true)
    - Yellow = Reserved (exists active reservation)
- **Data Source**: slots + optional filter from reservations

4.1.4 Entry Process

- **Actors**: Guest + IoT camera
- **Flow**:
    1. ESP32 detects motion.
    2. Camera captures image → uploads to Supabase bucket.
    3. Python script downloads → runs YOLO model.
    4. Plate cropped → sent to Roboflow OCR.
    5. Plate text is matched with cars table.
- New parking_log inserted with entry_time.

4.1.5 Exit Process

- **Actors**: Guest + Camera
- **Flow**:
    - ESP32 detects exit motion.
    - Photo uploaded to "exit" bucket.
    - Plate detected and matched to existing car.
    - System finds latest log where exit_time is null.
    - Updates log with current time as exit_time.
- **Triggers**:
    - calculate_duration()
    - calculate_amount()

4.1.6 Plate Recognition

- **Actor**: Camera (System)
- **Description**: Automatically identifies car plates using AI.
- **Steps**:
    - YOLO detects bounding box
    - Plate cropped
    - Roboflow OCR predicts characters
    - System converts output to Arabic plate

4.1.7 Update Parking Log

- **Actor**: System
- **Tables**: parking_logs
- **Logic**:
    - On **entry**: Insert new log with entry_time
    - On **exit**: Update existing row (where exit_time IS NULL)
    - Triggers calculate duration and amount

4.1.8 Admin: View Dashboard

- **Actor**: Admin
- **Description**: Admin sees all key stats
    - Active reservations
    - Parking usage
    - Slot status
    - User activity

4.1.9 Admin: Manage Users

- **Actor**: Admin
- **Tables**: users, owners
- **Permissions**:
    - Add/remove operators
    - Reset passwords
    - Control access roles

4.1.10Admin: Manage Slots and Logs

- **Actor**: Admin
- **Actions**:
    - Change slot status manually
    - Delete logs if needed
    - Override reservations
    - Add slots or deactivate broken ones

4.1.11 Operator: View Logs Only

- **Actor**: Operator
- **Access**:
  - View parking_logs table
  - Cannot modify cars, users, or reservations
  - Can assist in real-time monitoring only

4.2 Sequence Design (Flow Chart)

4.2.1 Introduction

This sequence diagram illustrates the complete interaction between the **user**, the **mobile application**, the **IoT devices**, and the **backend infrastructure** of the El-Sayes Smart Parking System. It showcases how a simple user action—reserving a parking slot—triggers a series of automated, real-time operations across the system, involving:

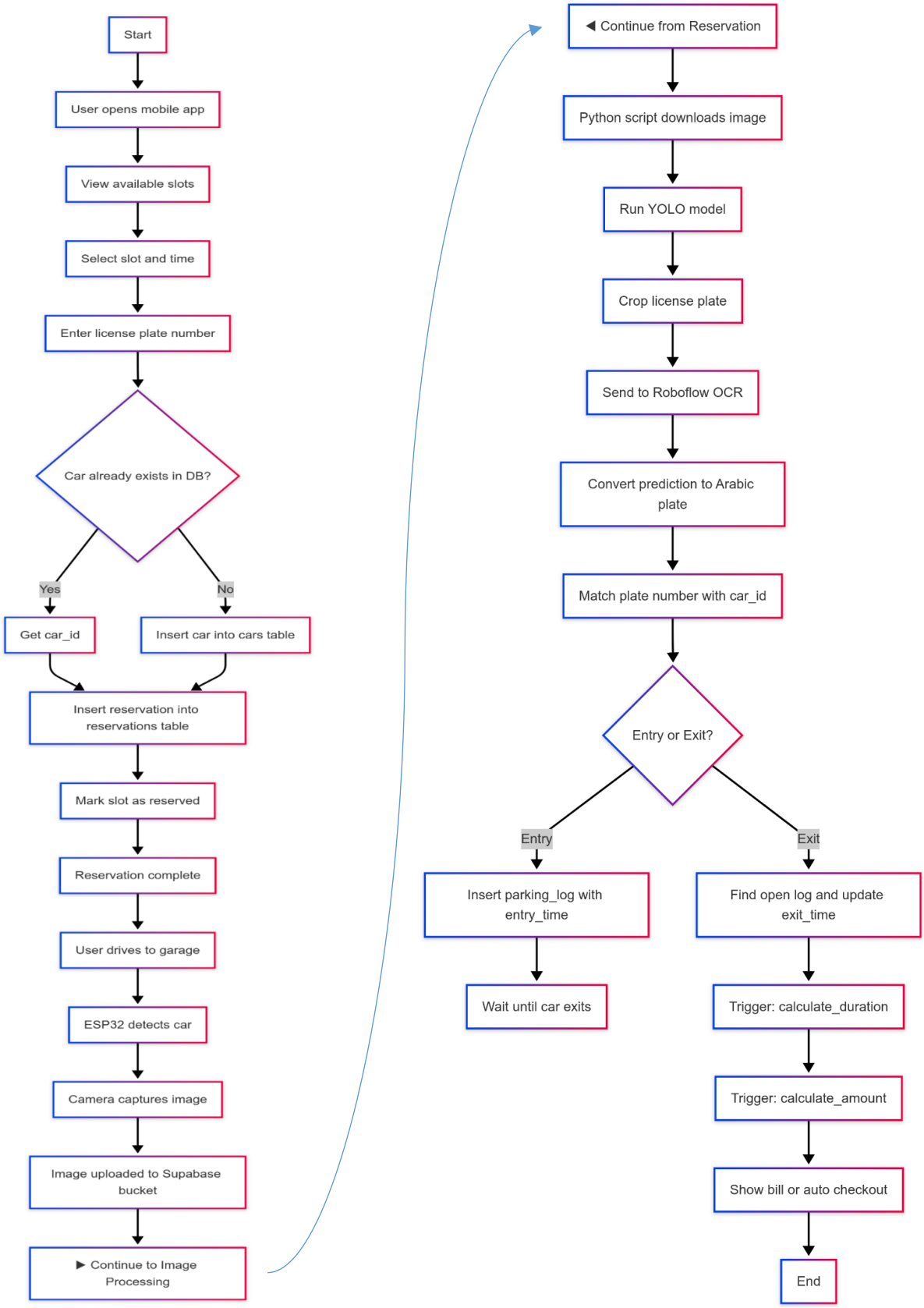
- **Frontend** (User interface and mobile app)
- **Database logic** (PostgreSQL tables, triggers, and stored procedures)
- **IoT edge devices** (ESP32 boards, IR sensors, and ESP32-CAMs)
- **Machine learning and OCR pipelines** (YOLO and Roboflow integration)

The sequence is divided into three major phases:

1. **User-Side Reservation Flow** – Begins when the user browses and reserves a slot using the mobile application.
2. **Entry/Exit Recognition Process** – Managed by IoT hardware and AI-based license plate detection.
3. **Automated Billing Logic** – Executed through Supabase triggers to calculate parking duration and fees without manual input.

This end-to-end automated cycle ensures a **fully contactless, intelligent, and scalable** parking experience—delivering efficiency for users and garage operators alike.

4.2.2 Flow Diagram



4.2.3 Explanation of Flow Diagram Steps

1. User Side (Mobile App): Reservation Workflow

This phase starts when the user interacts with the mobile application to reserve a parking slot. The flow is as follows:

- The user **browses and views** all available parking slots (fetched from the slots table).
- They **select a specific slot** and define a reservation time window (reserved_from, reserved_to).
- They input their **vehicle license plate number**.
- The system checks if the car already exists in the cars table:
  - If **not found**, a new record is inserted with the user's plate number and default owner ID.
- The reservation is stored in the reservations table with status active.
- The selected slot is immediately marked as **reserved** by setting is_occupied = true in the slots table.

2. Entry & Exit Process (AI + IoT + Image Recognition)

Once the vehicle physically arrives at the parking entrance or exit:

- The **ESP32 microcontroller** detects motion through its sensor.
- A camera connected to the ESP32 captures a **photo** and uploads it to the appropriate Supabase bucket (entrance or exit).
- A **Python background script** automatically processes the new image:
  - It runs the **YOLO model** to detect the license plate region.
  - The plate is **cropped** from the image.
  - The cropped image is sent to **Roboflow OCR** to extract characters.
  - The result is **converted to Arabic plate format**.
- The plate number is **matched with existing cars** in the database.
- If matched:
  - On **entry**: a new row is inserted in parking_logs with the current entry_time.
  - On **exit**: the system looks up the most recent log where exit_time IS NULL, and updates it with the current exit_time.

3. Billing Process (Database Trigger Logic)

Once the exit is logged, two triggers are automatically activated:

- calculate_duration():
  - Calculates the total time the car spent inside the garage, based on the difference between entry_time and exit_time.
- calculate_amount():
  - Computes the final amount to be paid by multiplying the duration by the predefined rate per hour (e.g. 10 EGP/hour).

The updated record in the parking_logs table includes:

- Entry time
- Exit time
- Duration (in minutes or hours)
- Final billed amount

4.3 Data Flow Diagram

4.3.1 Introduction

A Data Flow Diagram (DFD) is a graphical representation used to depict the flow of data within a system. It illustrates how data is processed, stored, and transferred between various components. DFDs are divided into levels to provide a step-by-step understanding of the system's functionality, starting from a high-level overview (DFD0) and progressing into detailed views (DFD1, DFD2, etc.).

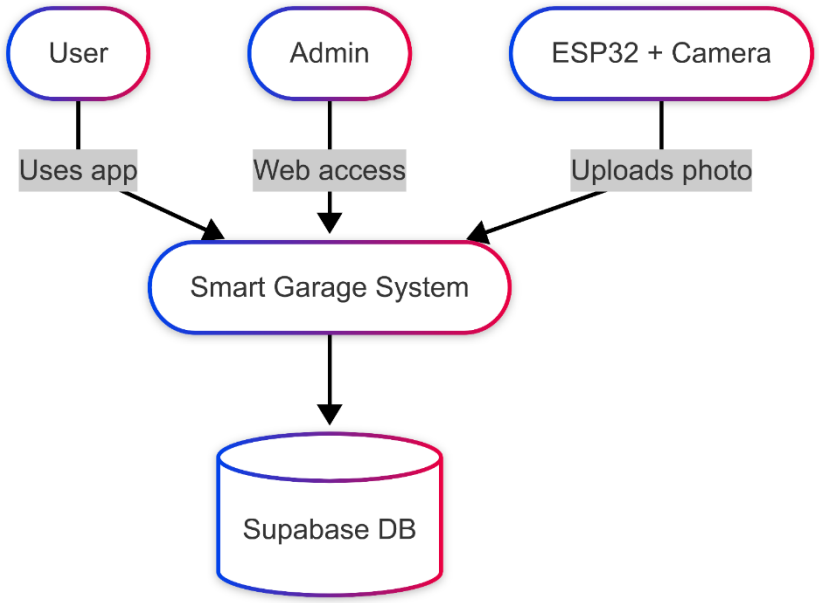
For the **Smart Garage System**, the DFDs are structured to show how data flows between the IoT components, machine learning model, Firebase database, web application, and mobile application. The DFD levels provide insight into system functionality, data interactions, and integration points.

4.3.2 DFD Levels

4.3.2.1 DFD Level 0: Context Diagram (System Overview)

Description:

DFD 0 provides a high-level overview of the system, highlighting the main entities, external users, and data flows. It shows the interactions between the key components: users, IoT devices, Firebase, the web application, the machine learning model, and the mobile application. This level focuses on the system as a whole without diving into details.



External Entities:

- User (Driver)
- Admin (Web App)
- IoT Hardware (ESP32 + Sensors + Cameras)
- ML Model (OCR + YOLO)

4.3.2.2 DFD Level 1: Subsystems Overview

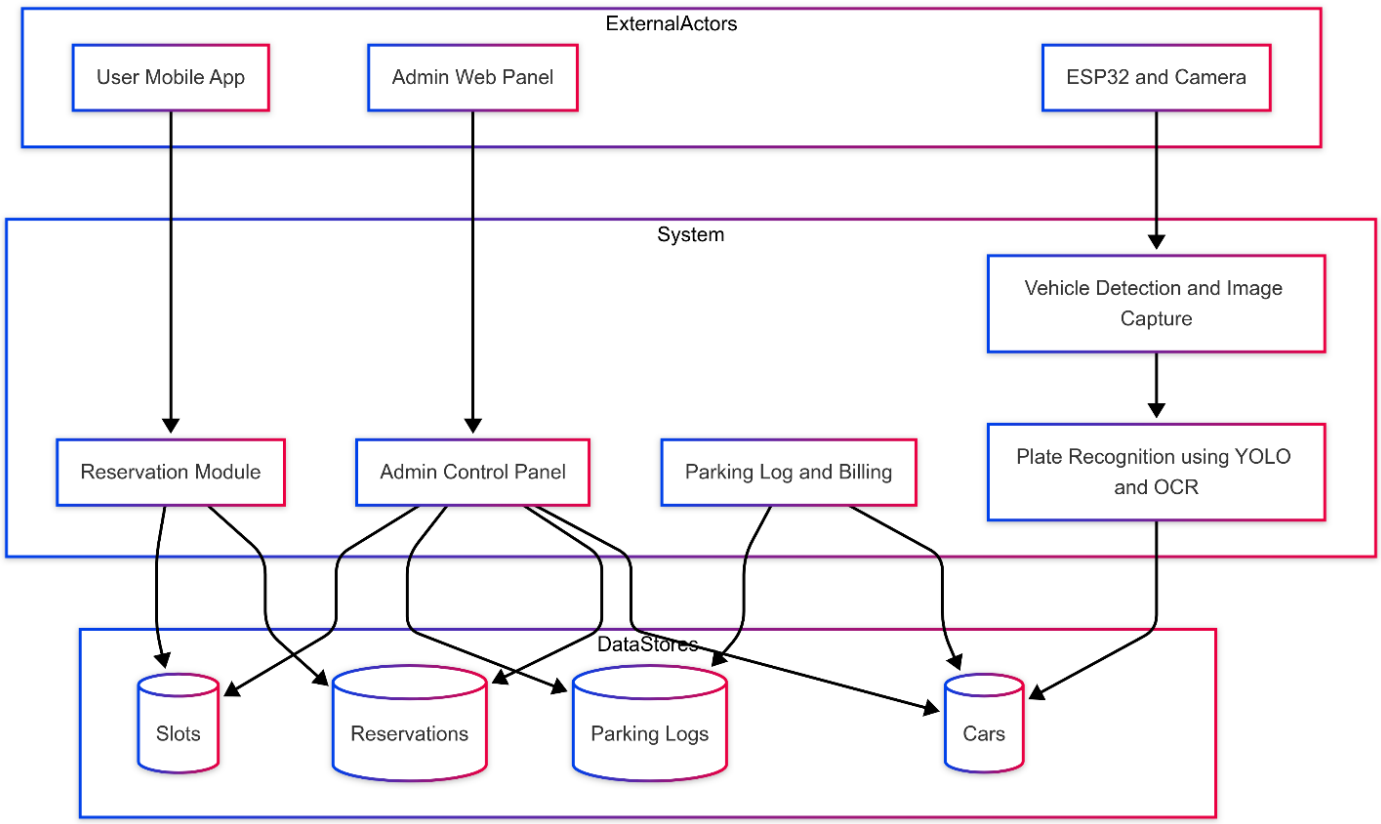
Here we decompose Smart Garage System into core logical processes:

Processes:

- **P1.0**: Manage Vehicle Detection (ESP32, camera triggers)
- **P2.0**: AI Plate Recognition (YOLO + Roboflow OCR)
- **P3.0**: Reservation Management (app/web)
- **P4.0**: Log Entry/Exit and Payment Calculation
- **P5.0**: Admin Management Interface

Data Stores:

- D1: cars
- D2: reservations
- D3: parking_logs
- D4: slots

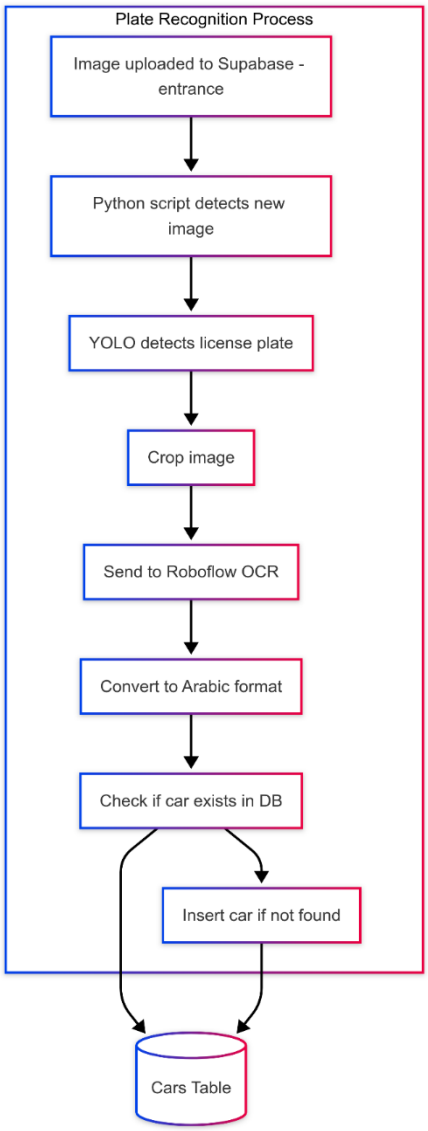


4.3.2.3 DFD Level 2: Subsystem Details

The **Level 2 Data Flow Diagram** (DFD) offers a detailed breakdown of the key internal processes within the *El-Sayes Smart Parking System*. This level focuses on the most critical operational layers: **AI-powered license plate recognition**, **reservation management**, and **automated parking log + billing operations**. Each process is decomposed to show how data flows between components, systems, and data stores, ensuring clarity and traceability

1. AI Plate Recognition

This process is responsible for handling the automation of vehicle identification through image recognition and OCR.



Flow Overview:

- Triggered when a car is detected by the gate ESP32's IR sensor.
- The ESP32 sends a flag to Supabase indicating a new capture request.
- The ESP32-CAM takes a real-time image of the vehicle and uploads it to Supabase Storage.
- A Python-based backend service:
    - Applies the YOLOv9 model to locate the license plate.
    - Crops the plate and sends it to Roboflow OCR for text extraction.
    - Converts the extracted plate into the correct Arabic format.
- The result is matched with the cars table to verify identity and status.

Data Stores Involved:

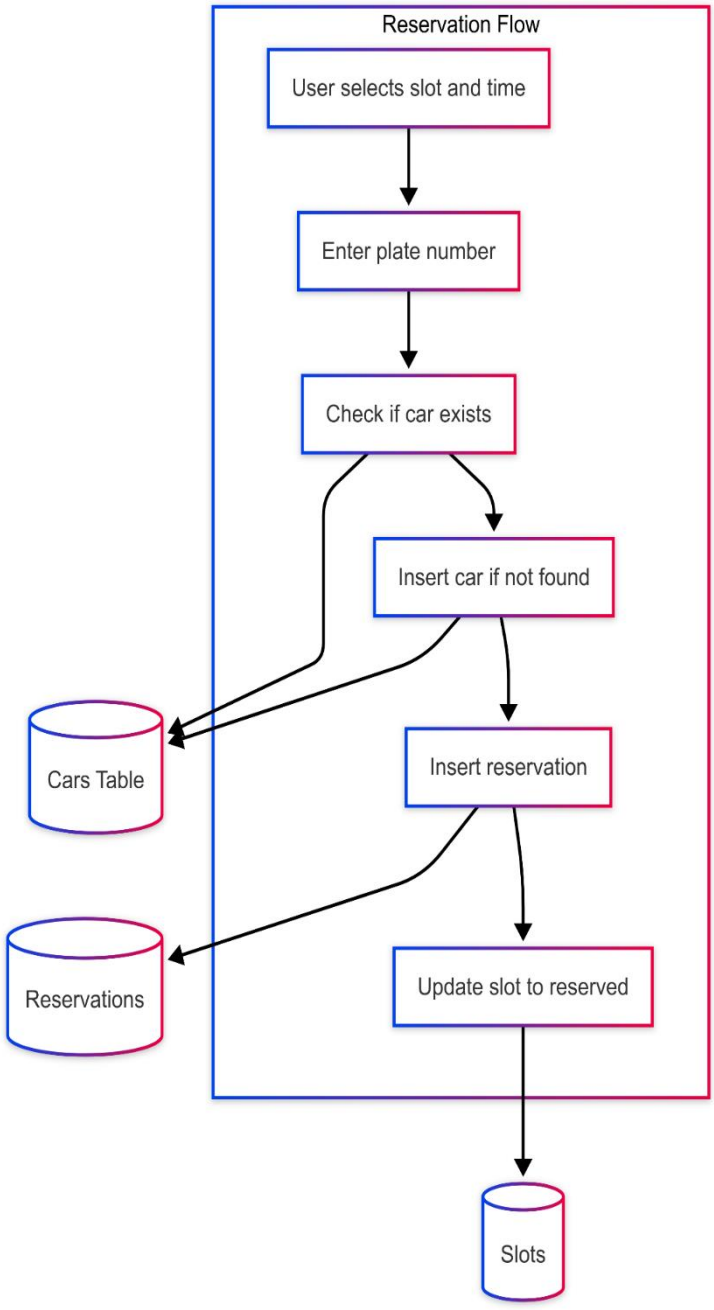
- D1: cars
- Supabase Storage (external media store for images)

Reservation Management

This process handles all user-side interactions related to booking a parking slot.

Flow Overview:

- Users view live slot status via the mobile app.
- Upon selecting a slot and entering their license plate:
    - The system checks if the plate exists in the cars table.
        - If not found, it creates a new record.
    - A reservation is then created in the reservations table with status set to "active."
    - The chosen slot is marked as is_reserved = true in the slots table.



Data Stores Involved:

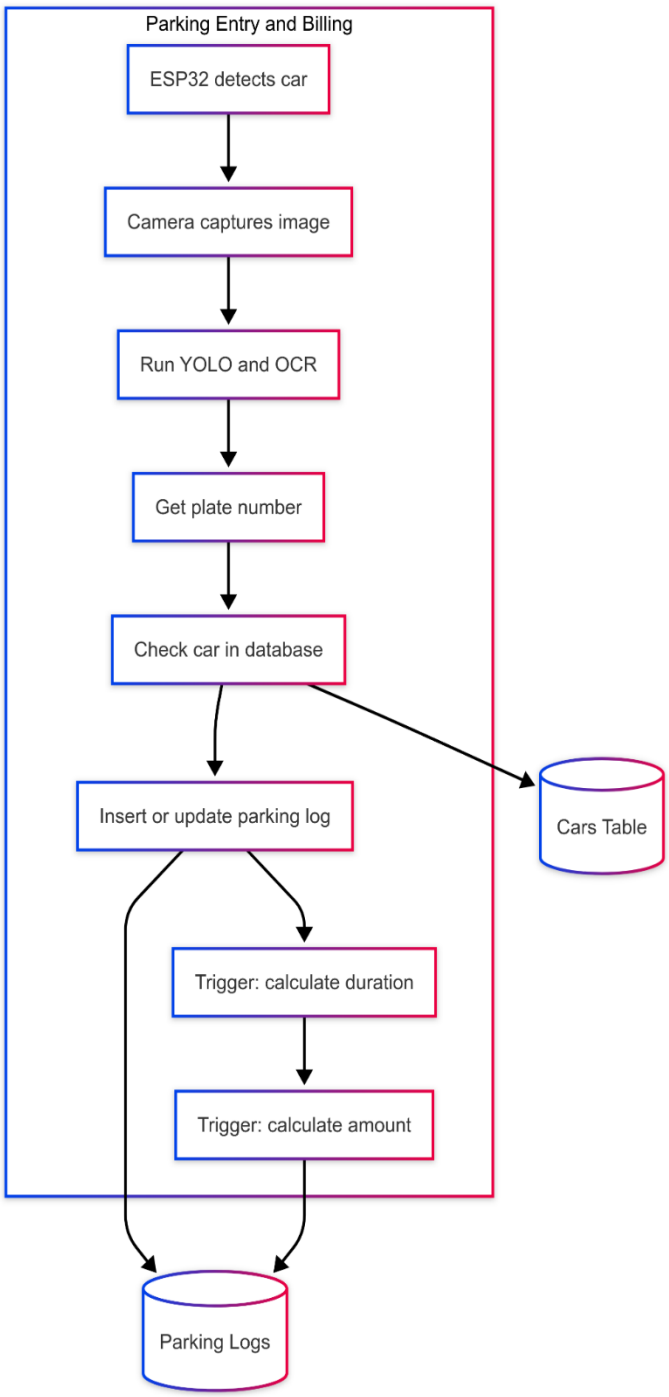
- D1: cars
- D2: reservations
- D4: slots

Parking Log and Billing

This process manages the full lifecycle of a vehicle's parking session—from entry to billing.

Flow Overview:

- When a license plate is matched upon entry:
    - A new record is added to parking_logs with an entry_time and no exit yet.
- Upon exit detection and license match:
    - The latest log where exit_time IS NULL is updated with the current exit_time.
- **Database triggers** are automatically executed:
    - calculate_duration(): Computes parking duration.
    - calculate_amount(): Multiplies duration by hourly rate (e.g., 10 EGP/hour).
- The log is finalized with:
    - Entry & exit timestamps
    - Duration
    - Final billed amount



Data Stores Involved:

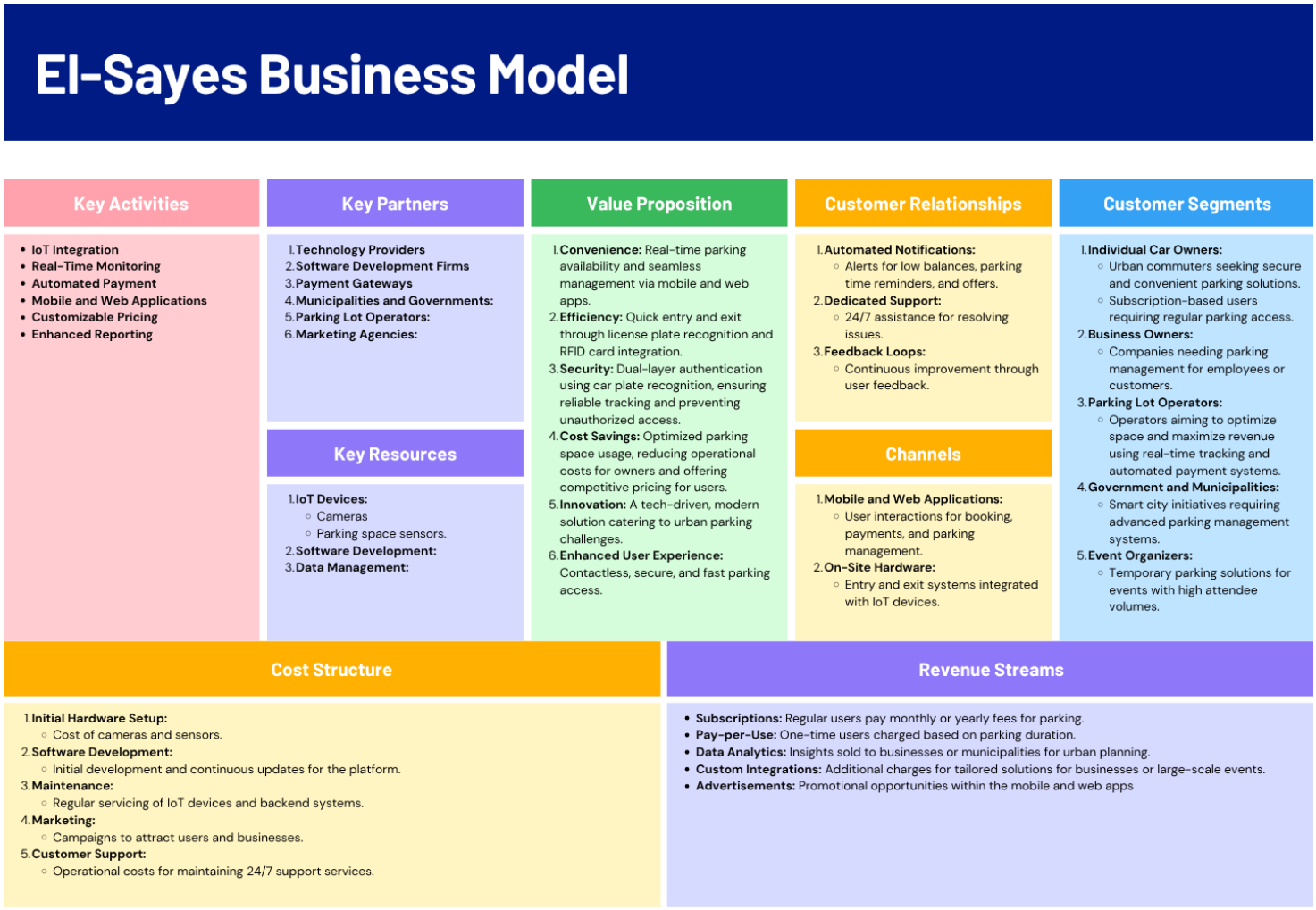
- D3: parking_logs

Chapter Five

Business Model and SWOT analysis

5. Chapter Five

5.1 Business Model



Value Proposition

1. **Convenience**: Real-time parking availability and seamless management via mobile and web apps.
2. **Efficiency**: Quick entry and exit through license plate recognition.
3. **Security**: Dual-layer authentication using car plate recognition, ensuring reliable tracking and preventing unauthorized access.
4. **Cost Savings**: Optimized parking space usage, reducing operational costs for owners and offering competitive pricing for users.
5. **Innovation**: A tech-driven, modern solution catering to urban parking challenges.
6. **Enhanced User Experience**: Contactless, secure, and fast parking access.

Customer Segmentation

1. **Individual Car Owners**:
    - Urban commuters seeking secure and convenient parking solutions.
    - Subscription-based users requiring regular parking access.
2. **Business Owners**:
    - Companies needing parking management for employees or customers.
3. **Parking Lot Operators**:
    - Operators aiming to optimize space and maximize revenue using real-time tracking and automated payment systems.
4. **Government and Municipalities**:
    - Smart city initiatives requiring advanced parking management systems.
5. **Event Organizers**:
    - Temporary parking solutions for events with high attendee volumes.

Key Features

1. **IoT Integration**:

    - **License Plate Recognition:** Automated detection of vehicle license plates at both entry and exit points using ESP32-CAM modules. This eliminates the need for physical cards or manual checks, enhancing security and convenience through contactless access.
    - **Real-Time Synchronization:**Captured images are processed by an OCR model and linked to user data in Supabase for accurate logging and seamless gate control.

2. **Real-Time Monitoring**:
    - Sensors track parking space availability and sync with the mobile app.
3. **Automated Payment**:
    - Payment calculated based on parking duration.
4. **Mobile and Web Applications**:
    - User-friendly interface for booking, monitoring, and managing parking.
5. **Customizable Pricing**:
    - Dynamic pricing based on peak/off-peak hours or user subscriptions.
6. **Enhanced Reporting**:
    - Detailed analytics for operators and users, including usage patterns and revenue.

Revenue Streams

1. **Subscriptions**: Regular users pay monthly or yearly fees for parking.
2. **Pay-per-Use**: One-time users charged based on parking duration.
3. **Data Analytics**: Insights sold to businesses or municipalities for urban planning.
4. **Custom Integrations**: Additional charges for tailored solutions for businesses or large-scale events.
5. **Advertisements**: Promotional opportunities within the mobile and web apps.

Key Resources

1. **IoT Devices**:
    - Cameras for license plate recognition.
    - Parking space sensors.
2. **Software Development**:
    - Mobile and web app platforms.
    - Backend systems integrated with Supabase and machine learning models.
3. **Data Management**:
    - Secure storage and processing of user and transaction data.

Key Partners

1. **Technology Providers**:
    - Suppliers for cameras and IoT devices.
2. **Software Development Firms**:
    - Partners for creating and maintaining mobile and web applications.
3. **Payment Gateways**:
    - Integration with online payment systems for seamless transactions.
4. **Municipalities and Governments**:
    - Collaboration for urban infrastructure integration and smart city initiatives.
5. **Parking Lot Operators**:
    - Partnerships for deploying and managing the system at scale.
6. **Marketing Agencies**:
    - Support for promotional campaigns and user acquisition.

Channels

1. **Mobile and Web Applications**:
    - User interactions for booking, payments, and parking management.
2. **On-Site Hardware**:
    - Entry and exit systems integrated with IoT devices.
3. **Customer Support**:
    - Chat and call support for user assistance.

Customer Relationships

1. **Automated Notifications**:
    - Alerts for low balances, parking time reminders, and offers.
2. **Dedicated Support**:
    - 24/7 assistance for resolving issues.
3. **Feedback Loops**:
    - Continuous improvement through user feedback.

Cost Structure

1. **Initial Hardware Setup**:
    - Cost of cameras and sensors.
2. **Software Development**:
    - Initial development and continuous updates for the platform.
3. **Maintenance**:
    - Regular servicing of IoT devices and backend systems.
4. **Marketing**:
    - Campaigns to attract users and businesses.
5. **Customer Support**:
    - Operational costs for maintaining 24/7 support services.

5.2 SWOT Analysis

Strengths

1. **Innovative Technology**: Integration of RFID, IoT, and machine learning provides a cutting-edge, secure, and efficient parking solution.
2. **User Convenience**: Real-time availability and automated payment systems enhance user experience.
3. **Scalability**: Easily adaptable to different locations and user needs, from small parking lots to large urban spaces.
4. **Dual Authentication**: The combination of RFID cards and license plate recognition ensures high security.
5. **Customizable Pricing**: Dynamic pricing models cater to diverse customer segments.
6. **Data-Driven Insights**: Analytics provide value to parking operators and urban planners.
7. **Diverse Revenue Streams**: Subscriptions, pay-per-use, and advertisements offer multiple income channels.

Weaknesses

1. **High Initial Costs**: Significant investment in IoT devices, Camera System, and software development.
2. **Complex Implementation**: Integration of multiple technologies requires careful planning and skilled teams.
3. **Dependence on Technology**: System downtime or technical failures could disrupt operations and affect user trust.
4. **User Adoption**: Educating users about the new system and encouraging adoption may require time and resources.

Opportunities

1. **Smart City Initiatives**: Aligning with government projects to promote sustainable and efficient urban infrastructure.
2. **Expansion to New Markets**: Introducing the system in underdeveloped areas with high parking demand.
3. **Partnerships with Businesses**: Collaborating with malls, airports, and event organizers to deploy tailored solutions.
4. **Additional Features**: Gamification, loyalty programs, and EV charging integration could attract more users.
5. **Regulatory Compliance**: Systems like this could align with environmental and traffic regulations, making it attractive to policymakers.
6. **Data Monetization**: Offering anonymized parking and traffic data to third parties for urban planning and marketing.

Threats

1. **Competition**: Emerging competitors with similar or more advanced technologies.
2. **Regulatory Changes**: Legal and compliance issues related to data privacy and IoT device use.
3. **Economic Factors**: Changes in economic conditions might impact users' ability to pay for premium parking services.
4. **Technological Obsolescence**: Rapid advancements in technology could render current systems outdated.

Cybersecurity Risks: Potential vulnerabilities in IoT devices and data storage systems could lead to breaches.

5.3 Segmentation

Segmentation involves dividing your target market into distinct groups of customers based on various criteria. For your Smart Garage System, you could segment your customers based on factors like demographics, usage patterns, location, and needs. Here's a potential breakdown:

- **Demographic Segmentation**:
  - **Age**: Primarily targeting adults (18-50 years) who are working professionals or homeowners, as they are more likely to use parking services regularly.
  - **Income**: Middle to high-income groups, as they are more likely to own cars and be willing to pay for secure parking services.
  - **Family Status**: Families with multiple cars or professionals with dedicated parking spaces may be more inclined to use such systems.
- **Geographic Segmentation**:
  - **Urban Areas**: Major cities with high traffic congestion and a shortage of parking spaces.
  - **Suburban Areas**: Areas where people have larger parking spaces but still face issues with parking management and security.
- **Behavioral Segmentation**:
  - **Frequent Users**: Car owners who commute daily and need reliable parking.
  - **Occasional Users**: People who might only use parking services during weekends or special events.
- **Psychographic Segmentation**:
  - **Tech-Savvy Users**: Individuals who prefer smart systems for convenience and security.
  - **Security-Conscious Users**: Customers who prioritize high-level security features, like RFID and license plate recognition.

These segments allow you to tailor your marketing and product offerings for each specific group, ensuring better engagement and sales conversion.

5.4 Future Work

As the Smart Garage System evolves, our primary focus is on broadening its applicability and enhancing the user experience through cutting-edge features, extensive integrations, and new service offerings. Below are the key directions for future work:

1. Multi-Garage Integration

- **Centralized Management**: Expand the system to manage and monitor multiple garages simultaneously through a unified platform. This will enable users to select and reserve slots in any participating garage from a single application.
- **Inter-Garage Optimization**: Implement algorithms to dynamically allocate parking spaces across different garages within the same area to optimize utilization and reduce overcrowding.

2. Google Maps Integration

- **Real-Time Availability**: Collaborate with Google Maps to display live parking slot availability directly on the map, making it easier for users to locate nearby garages with free spaces.
- **Dynamic Route Suggestions**: Provide navigation routes optimized for parking availability, taking into account real-time traffic conditions and slot occupancy rates.

3. Advanced Services within Garages

- **Mechanic Services**: Offer on-demand mechanic services for minor repairs, tire changes, and routine maintenance.
- **EV Charging Stations**: Install electric vehicle charging points in garages, catering to the growing EV market.
- **Car Wash and Detailing**: Provide automated or manual car wash facilities, as well as detailing services for vehicle upkeep.
- **Valet Parking**: Introduce valet services for premium customers seeking added convenience.

4. Extended Application Availability

- **Diverse Locations**: Extend the application's presence to include not just garages but also cafes, picnic spots, malls, and other high-traffic locations. Users can locate parking spaces and associated services wherever they go.
- **Event-Based Parking**: Partner with event organizers to provide temporary parking solutions during concerts, sports matches, and festivals.

5. On-Road Parking Assistance

- **Free Slot Detection**: Use IoT and camera-based systems to detect and display free parking slots along roads, reducing the time users spend searching for parking.
- **Dynamic Pricing**: Implement flexible pricing for on-road parking, encouraging fair usage and turnover during peak times.

6. Enhanced User Experience

- **Subscription Plans**: Introduce personalized subscription models, such as pay-per-use, monthly passes, or premium access to exclusive parking spaces.
- **Loyalty Programs**: Reward frequent users with discounts, priority access, or complimentary services.
- **Custom Notifications**: Send personalized alerts about slot availability, reservation reminders, and promotions.

7. Sustainability and Eco-Friendly Solutions

- **Green Energy Integration**: Equip garages with solar panels to power the system, reducing the environmental footprint.
- **Electric Vehicle Focus**: Prioritize EV charging infrastructure in all locations, with support for fast-charging technologies.
- **Paperless Operations**: Use QR codes and digital receipts to eliminate paper use across all operations.

8. Business Expansion

- **Partnerships with Malls and Cafes**: Collaborate with major retail chains, cafes, and shopping malls to embed parking services into their offerings.
- **Smart City Integration**: Work with municipal authorities to integrate the system into smart city initiatives, enhancing urban mobility and reducing congestion.
- **Global Reach**: Expand into international markets, focusing on cities with high vehicle density and limited parking facilities.

9. Data and AI Integration

- **Predictive Analytics**: Use AI to forecast peak parking times and recommend optimal parking slots to users.
- **Behavioral Insights**: Analyze user behavior to provide tailored suggestions and improve service efficiency.
- **Demand-Driven Expansion**: Utilize collected data to identify high-demand areas and strategically open new garages or add services.

Chapter Six

The System Development Life Cycle

“Implementation”

6. Chapter Six (Implementation)

6.1 Implementation

The implementation phase focuses on the realization of the system's design, ensuring that all components function cohesively. This phase involves hardware installation, software deployment, and system integration to deliver a fully operational smart parking solution.

The core objectives of this phase are:

- Deploy IoT devices for real-time parking slot monitoring and gate automation.
- Integrate the mobile and web applications with the backend system.
- Implement machine learning for car plate recognition.
- Ensure seamless communication and data synchronization through Firebase

The El-Sayes Smart Parking System operates as an interconnected ecosystem where each component contributes to a seamless parking experience. The simulation of the system highlights the interaction between IoT devices, applications, and backend services. Below is a detailed overview:

6.2 IoT Integration

6.2.1 Introduction

Objective

This report outlines the purpose and significance of integrating Internet of Things (IoT) technology into the El-Sayes Smart Parking System, focusing exclusively on the components utilized in the system's operation. The report emphasizes their roles in achieving seamless functionality and enhanced efficiency.

Scope

The discussion centers on the hardware components, their configuration, and the data flow mechanisms that underpin the IoT system's operation.

Significance

IoT integration using these components is essential for automating parking operations, ensuring real-time monitoring, and providing a robust infrastructure for future scalability.

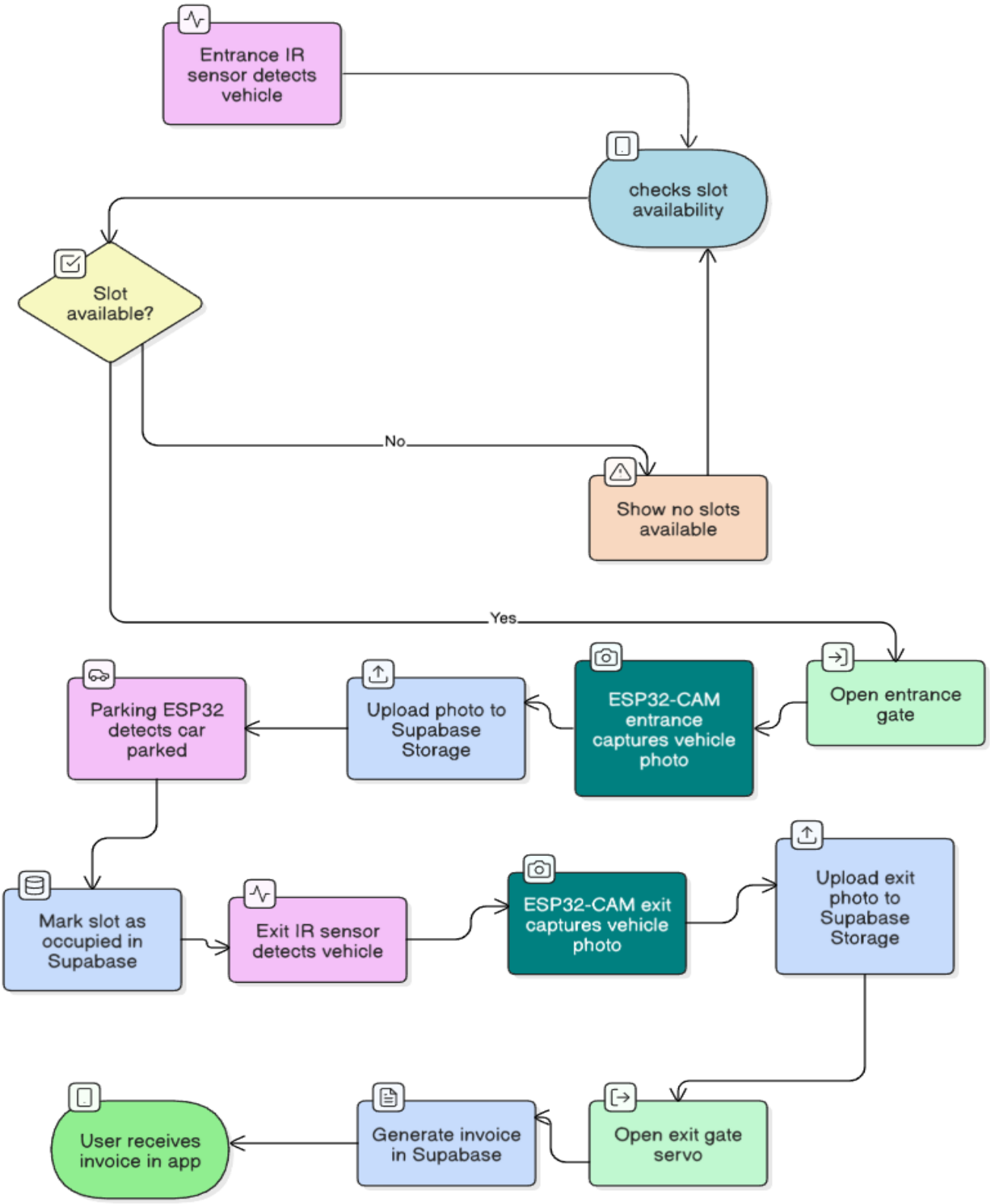
6.2.2. System Architecture

6.2.2.1 Full IoT System Cycle Overview

The smart garage system is an integrated IoT solution designed to manage vehicle parking through real-time coordination between multiple ESP32 devices and a Supabase backend. It includes:

- **5 parking slots** monitored by an ESP32 with IR sensors and LED indicators.
- **Gate controller ESP32** managing **2 gates** (entrance and exit) using **2 IR sensors** and **2 servo motors**.
- **2 ESP32-CAMs** at entrance and exit to capture vehicle photos.
- **Supabase** as the central cloud backend for device communication, user authentication, and database updates.
- A **Flutter mobile app** for users to check availability and reserve slots.
- A **web dashboard** for admin management and monitoring.

6.2.2.2 IOT FLOW CHART



6.2.2.3 Flow of Operation

Entry Process

- The IR sensor at the entrance detects the approaching vehicle.
- The ESP32 gate controller sends a signal to Supabase to trigger the entrance ESP32-CAM.
- The ESP32-CAM captures a photo of the car and uploads it to Supabase Storage.
- A backend OCR model processes the image to extract the license plate number.
- The system checks Supabase for a matching reservation or existing vehicle record.
- If valid, the entrance gate servo opens automatically, and entry is logged with a timestamp and image URL.

Parking Slot Monitoring

- IR sensors in each parking slot detect vehicle presence continuously.
- The ESP32 reads each sensor's status in real time.
- LED indicators display slot status as:
    - Green: Available
    - Red: Occupied
    - Yellow: Reserved
- If a slot is reserved and becomes occupied, the system marks the reservation as completed in Supabase.
- All status changes are synced instantly with Supabase to update the mobile app and admin dashboard.

Exit Process

- The IR sensor at the exit gate detects the vehicle.
- The ESP32 triggers the exit ESP32-CAM to capture an image and upload it to Supabase.
- The system identifies the plate number and links it to the corresponding entry record.
- Exit is logged with a timestamp, and parking duration can be calculated for billing or analytics.
- The exit gate servo motor opens automatically, and the slot status is updated to available once the car leaves.

6.2.3. Slot Management System

6.2.3.1 Overview of Slot System Design

The IoT architecture is designed with interconnected ESP32 modules managing parking slots and gate operations. Components such as IR sensors, addressable LEDs, level shifters, and step-down converters work in tandem to provide real-time updates and seamless operation. A schematic diagram (to be included) will illustrate the architecture.

6.2.3.2 Components

Hardware

1. **IR Sensors:** Detect vehicle presence at parking slots and gates.
2. **WS2811 Addressable LEDs:** Provide visual feedback on slot statuses.
3. **ESP32 Microcontroller:** Serves as the processing hub for sensor data and LED control.
4. **Level Shifter:** Converts 3.3V signals from ESP32 to 5V for LED operation.
5. **Step-Down Converter:** Reduces 12V input to 5V to power the ESP32.

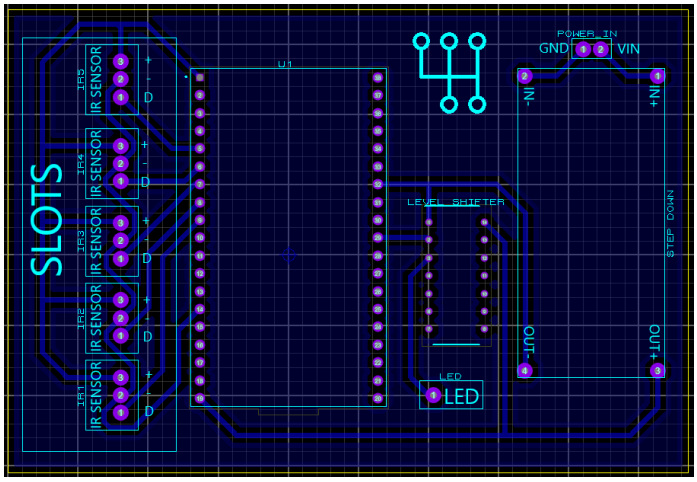
Communication Protocols

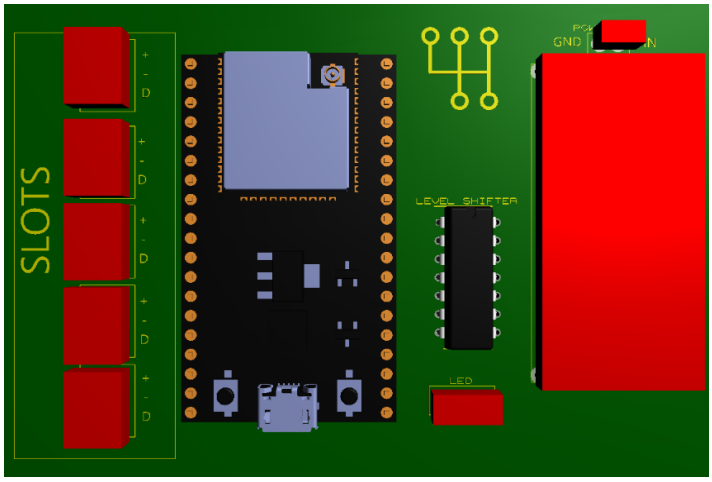
- **Wi-Fi:** Facilitates real-time data exchange with **Supabase**.
- **HTTP (REST**): Used to send real-time PATCH requests for updating the is_occupied status.

Backend

- **Supabase**: Acts as the central database to store and update parking slot statuses, reservation states, and timestamps.

PCB Design





Component-Level Details

IR Sensors

- **Purpose:** Detect vehicle presence.
- **Placement:** At parking slots and gates.
- **Specifications:** High precision with minimal false detection.
- **Communication:** Sends signals to ESP32 to update Supabase.

Addressable LEDs (WS2811)

- **Purpose:** Provide visual feedback for slot statuses.
- **Design:** Fifteen LEDs per slot, arranged in groups of five LEDs.
- **Control:** Managed via a level shifter converting ESP32's 3.3V signals to 5V.
- **Indications:**
    - Green: Slot available.
    - Red: Slot occupied.
    - Yellow: Slot reserved.

Level Shifter

- **Role:** Converts 3.3V signals from ESP32 to 5V required by WS2811 LEDs.
- **Significance:** Ensures reliable operation of addressable LEDs by matching voltage levels.

Step-Down Converter

- **Purpose:** Reduces 12V adapter power to 5V to supply power to the ESP32 module.
- **Integration:** Provides a stable and efficient power source for the system.

ESP32 Microcontroller

- **Role:** Serves as the processing hub for:
    - Receiving inputs from IR sensors.
    - Controlling the addressable LEDs.
    - Communicating data to Supabase in real time.
- **Power Source:** Powered via the step-down converter.

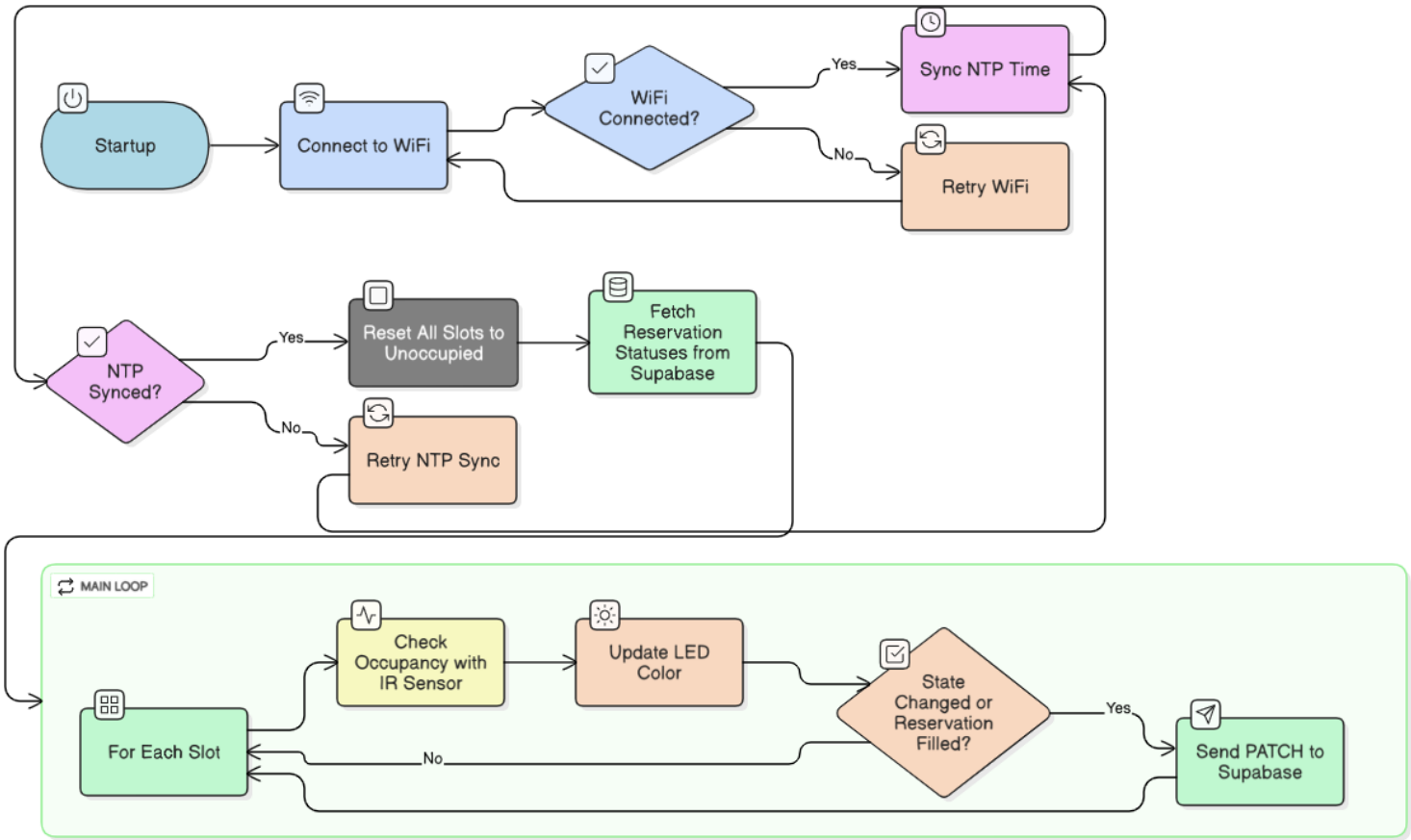
6.2.3.3 Data Flow and Communication

Real-Time Data Flow

- IR sensors detect slot occupancy and send signals to the ESP32.
- ESP32 updates Supabase with real-time slot statuses and operational data.
- Supabase synchronizes data with user-facing applications.

Error Handling

- **Voltage Mismatches:** Addressed using the level shifter.
- **Connectivity Issues:** Monitored with fallback protocols.
- **Power Stability:** Ensured by the step-down converter.



6.2.3.4 Code Logic

Libraries used :

Library	Purpose
WiFi.h	Establishes Wi-Fi connection
HTTPClient.h	Sends HTTP PATCH requests to Supabase
FastLED.h	Controls the RGB LED strip

Setup Stage

- Connects to Wi-Fi.
- Initializes 5 IR sensor input pins.
- Configures the LED strip with 5 addressable LEDs.

Main Loop Logic

- Reads the digital value from each IR sensor.
- Calls updateSlot(index, isOccupied):
    - If IR sensor is triggered (car present), turns corresponding LED red.
    - If not triggered (vacant), turns LED green.
    - If the state changes from last check, updates Supabase via sendToSupabase().

Function Summaries:

- `void sendToSupabase(int slotId, bool isOccupied)`
    - Sends a PATCH request to update the is_occupied status of a specific slot in Supabase.
- `void updateSlot(int slotIndex, bool isOccupied)`
    - Sets LED color (green or red) for the slot.
    - If state changed, sends update to Supabase and stores the new state.

- `void setup()`
    - Initializes serial, Wi-Fi, LED strip, and IR sensor pins.
- `int loop()`
    - Continuously monitors IR sensors, updates LEDs and Supabase every 500 ms.

6.2.3.5 Advantages of Integration

- **Automation:** Reduces manual intervention.
- **Real-Time Monitoring:** Updates slot statuses instantly.
- **Scalability:** Modular design allows easy expansion.
- **Cost-Effectiveness:** Efficient hardware utilization minimizes expenses.

6.2.3.6 Challenges and Mitigation

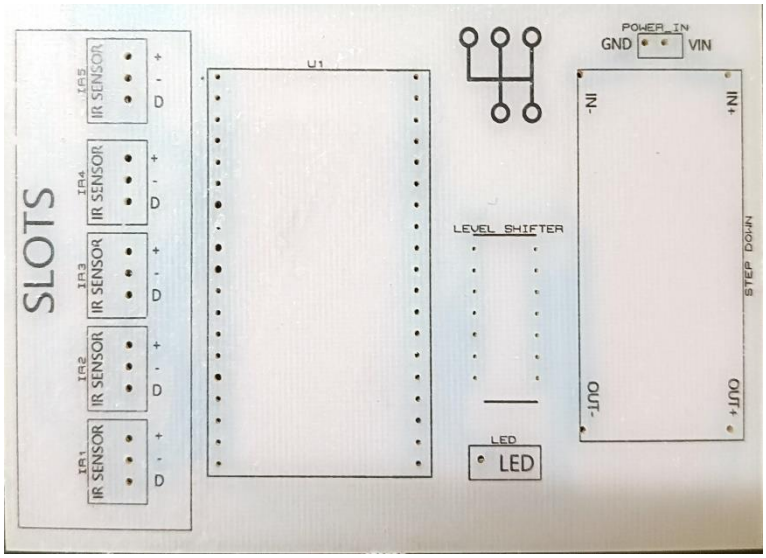
Challenges

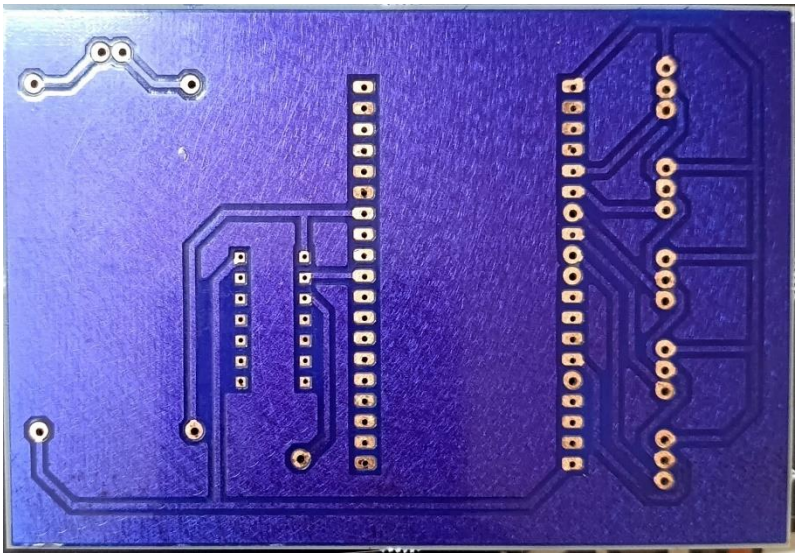
1. **Signal Interference:** May affect sensor communication.
2. **Voltage Regulation:** Critical for stable LED operation.
3. **Hardware Costs:** Need for cost-effective sourcing.

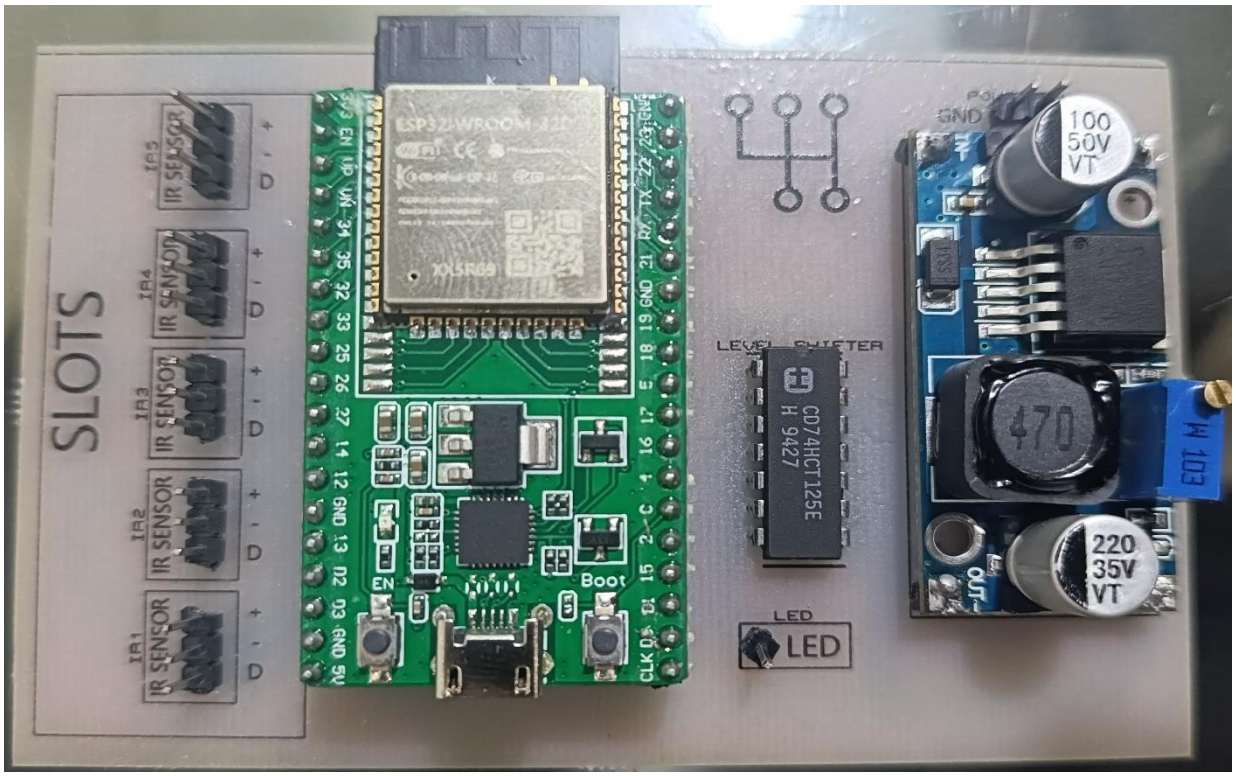
Mitigation Strategies

- **Shielding:** Minimize interference for IR sensors.
- **Quality Components:** Use reliable level shifters and step-down converters.
- **Bulk Sourcing:** Reduce hardware costs through strategic procurement.

6.2.3.7 Final Component







6.2.4 Gate Automation

6.2.4.1 Overview of Gate System Design

The gate automation architecture is built on interconnected ESP32 modules that coordinate entrance and exit gate operations. Core components include IR sensors for vehicle detection, servo motors for gate actuation, I2C LCDs for real-time display, and ESP32-CAM modules for license plate recognition. All components work together to deliver a secure and automated flow, fully synchronized with the Supabase backend. A schematic diagram (to be included) illustrates the architecture.

6.2.4.2 Components

Hardware

1. **IR Sensors:** Detect vehicle presence at entrance and exit points.
2. **Servo Motors:** Control physical gate movement (open/close).
3. **ESP32-CAM Modules:** Capture vehicle images for license plate recognition.
4. **I2C LCD Display:** Presents system messages and status updates in real time.
5. **ESP32 Controller:** Orchestrates all gate operations, sensor readings, and Supabase communication.

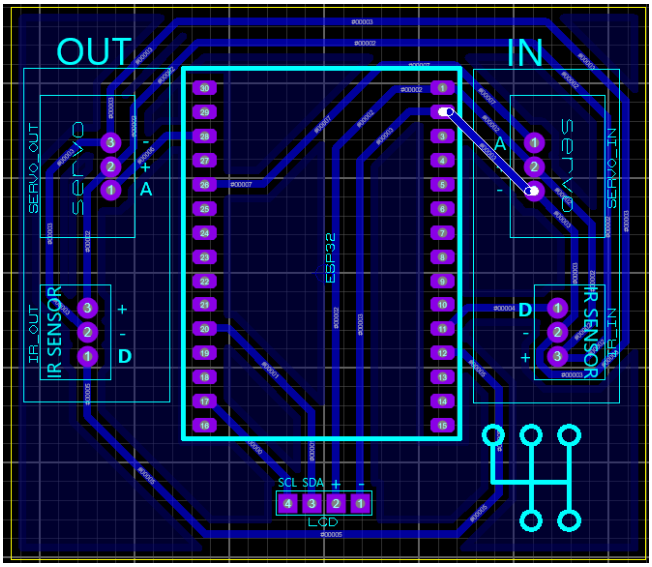
Communication Protocols

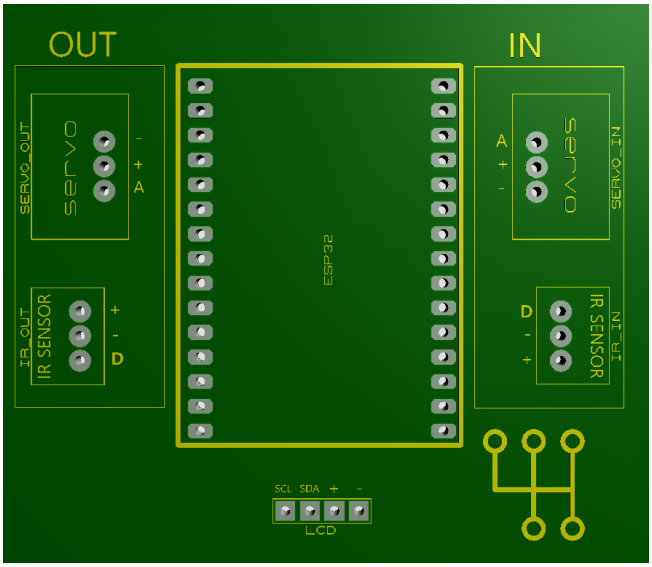
- **Wi-Fi:** Connects all ESP32 modules to the internet for backend interaction.
- **HTTP (REST):** Used to send/receive real-time data from Supabase (e.g., camera triggers, slot checks, entry logs).

Backend

- **Supabase:** Provides a real-time platform for data synchronization and storage.

PCB Design





Component-Level Details

IR Sensors

- **Purpose:** Detect cars at entrance and exit gates.
- **Placement:** Installed near the gate paths for early detection.
- **Integration:** Signals sent to ESP32 to initiate photo capture and gate logic.

Servo Motors

- **Purpose:** Control gate barrier movement.
- **Specifications:** Capable of smooth motion with adequate torque.
- **Operation Logic:**
    - Opens gate when entry/exit is validated
    - Closes gate after a preset time or vehicle departure confirmation

ESP32-CAM Modules

- **Purpose:** Capture real-time images of vehicles at entry and exit.
- **Placement:** Mounted at appropriate height for clear plate visibility.
- **Integration:** Triggered by Supabase signals from the gate ESP32 controller.

I2C LCD

- **Purpose:** Display parking system status, user prompts, and errors.
- **Integration:** Controlled by ESP32 and updated based on system events (e.g., access granted, garage full, exit logged).

ESP32 Controller

- **Role:**
    - Manages IR inputs and gate logic
    - Sends camera triggers to Supabase
    - Handles LCD feedback
    - Syncs with Supabase for vehicle validation and slot availability

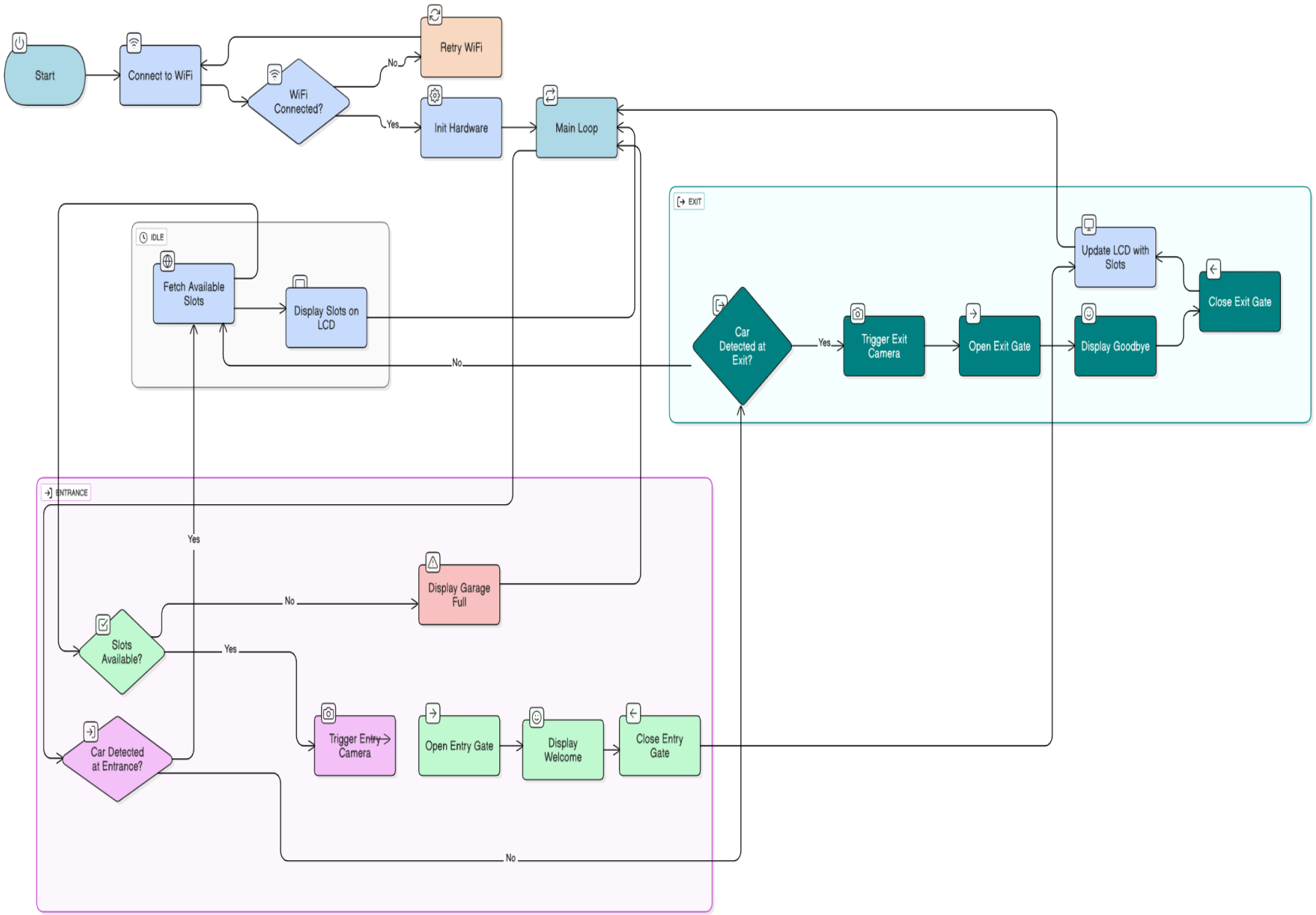
6.2.4.3 Data Flow and Communication

Real-Time Data Flow

- IR sensor detects a vehicle → ESP32 triggers camera → ESP32-CAM uploads image → OCR extracts plate → Plate is matched in Supabase → Gate opens if valid → LCD displays message → Entry or exit is logged.

Error Handling

- Camera Timeout: If ESP32-CAM does not respond, retry or show error on LCD.
- Invalid Plate: Gate remains closed, and a warning is displayed.
- Wi-Fi Disruption: ESP32 attempts reconnection and maintains last known state temporarily.


Start
Connect to WiFi
WiFi Connected?
No
Retry WiFi
Yes
Init Hardware
Main Loop
IDLE
Fetch Available Slots
Display Slots on LCD
No
EXIT
Car Detected at Exit?
Yes
Trigger Exit Camera
Open Exit Gate
Update LCD with Slots
Display Goodbye
Close Exit Gate
ENTRANCE
Yes
Slots Available?
No
Display Garage Full
Yes
Car Detected at Entrance?
Trigger Entry Camera
Open Entry Gate
Display Welcome
Close Entry Gate
No

6.2.4.4 Code Logic

Libraries Used:

Library	Purpose
WiFi.h	Connects to Wi-Fi network
HTTPClient.h	Makes HTTP GET and PATCH requests to Supabase
Wire.h	Required for I2C communication (used by LCD)
LiquidCrystal_I2C.h	Controls the 16x2 LCD via I2C
ESP32Servo.h	Controls the servo motors
ArduinoJson.h	Parses JSON responses from Supabase

Setup Stage

- Connects to Wi-Fi.
- Initializes and closes both servos (servo.write(90)).
- Sets LCD to “System Ready”.
- Sets IR sensor pins as input.

Main Loop Logic

1. **Entrance Detection:**
    - If a car triggers IR_IN and there's a change in signal:
        - Calls triggerCameraRequest() to notify the ESP32-CAM.
        - Calls getAvailableSlots() from Supabase.
        - If a slot is available, opens entrance gate via handleEntry(); otherwise, shows "Garage Full" on LCD.
2. **Exit Detection:**
    - When IR_OUT is triggered:
        - Calls triggerCameraRequest() to capture exit photo.
        - Calls handleExit() to open exit gate.

3. **Idle Display:**
    - When both IR sensors are inactive, refreshes LCD to show current available slots.

Function Summaries

- `void triggerCameraRequest()`
    - Sends PATCH request to Supabase camera_request table to trigger ESP32-CAM capture.
- `void handleEntry()`
    - Displays welcome message, opens entrance gate (servo to 180°), waits, then closes (servo to 90°), and updates LCD.
- `void handleExit()`
    - Displays goodbye message, opens exit gate (servo to 0°), waits, then closes (servo to 90°), and updates LCD.
- `int getAvailableSlots()`
    - Sends GET request to Supabase parking_slots table, counts how many slots have is_occupied = false, and returns the count.

6.2.4.5 Advantages of Integration

Automation: Enables fully automated gate entry and exit using IR sensors and servo motors, removing the need for human intervention.

Enhanced User Experience: Real-time system feedback is provided through the I2C LCD display, improving user trust and convenience.

Scalability: The system is modular and can support additional gates, cameras, or sensors as needed without significant redesign.

Security: Vehicle license plate recognition via ESP32-CAM and Supabase logging ensures traceability and prevents unauthorized access.

Integration with Cloud Services: Real-time synchronization with Supabase enables seamless data handling, logging, and access across mobile and web platforms.

6.2.4.6 Challenges and Mitigation

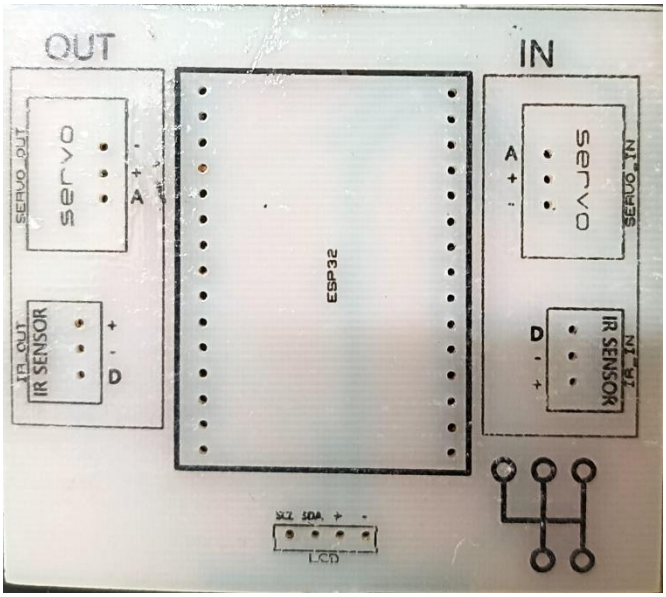
Challenges

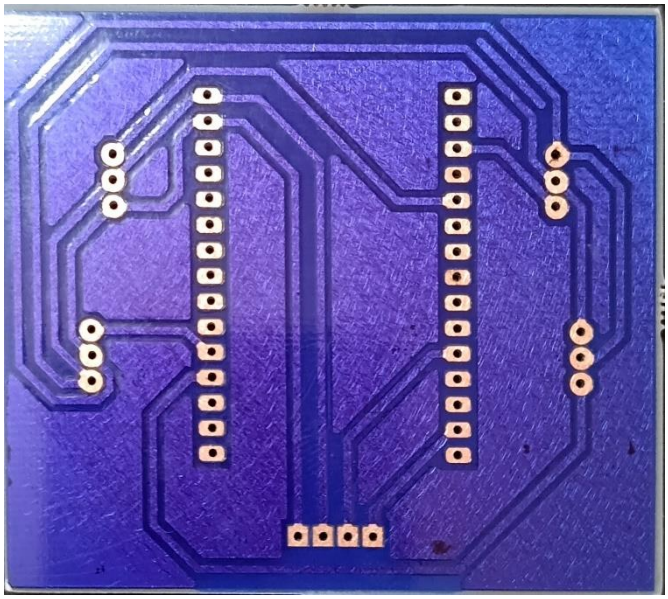
- **IR Sensor Accuracy:** Risk of false detection due to environmental factors (sunlight, shadows, etc.).
- **Servo Motor Durability:** Potential wear and tear from frequent gate operations.
- **Camera Reliability:** Lighting conditions may affect license plate image clarity.
- **Wi-Fi Stability:** Network disruptions may interrupt real-time operations.

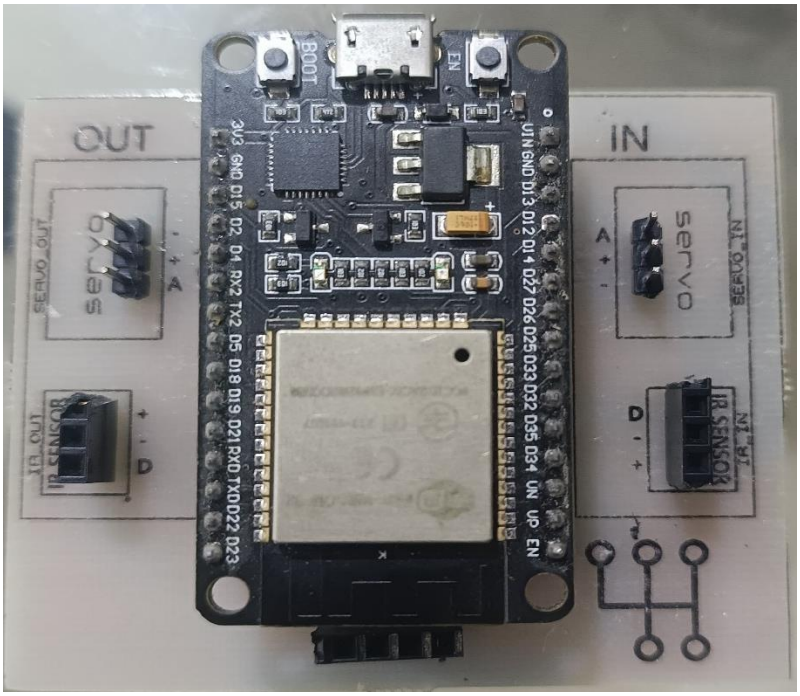
Mitigation Strategies

- **Sensor Calibration:** Use high-quality IR sensors and apply proper shielding to minimize false triggers.
- **Preventive Maintenance:** Implement a regular check-up schedule for servo motors to extend their operational lifespan.
- **Optimized Camera Placement:** Mount ESP32-CAMs at optimal angles and heights with lighting adjustments to improve image capture.
- **Connection Monitoring: Add retry logic and local caching to reduce dependency on immediate internet availability.**

6.2.4.7 Final Component







6.2.5 ESP32-CAM

6.2.5.1 Overview of Camera System Design

The ESP32-CAM subsystem plays a critical role in automating vehicle recognition and ensuring secure, contactless entry and exit in the El-Sayes Smart Parking System. It consists of two ESP32-CAM modules—one installed at the entrance gate and one at the exit.

Each ESP32-CAM captures a real-time image of a vehicle upon trigger from the gate ESP32 controller. The image is then uploaded to Supabase Storage, where it is processed by an OCR model to extract the license plate number. This system replaces traditional RFID authentication, enabling a modern, scalable, and secure entry/exit process.

Key Components

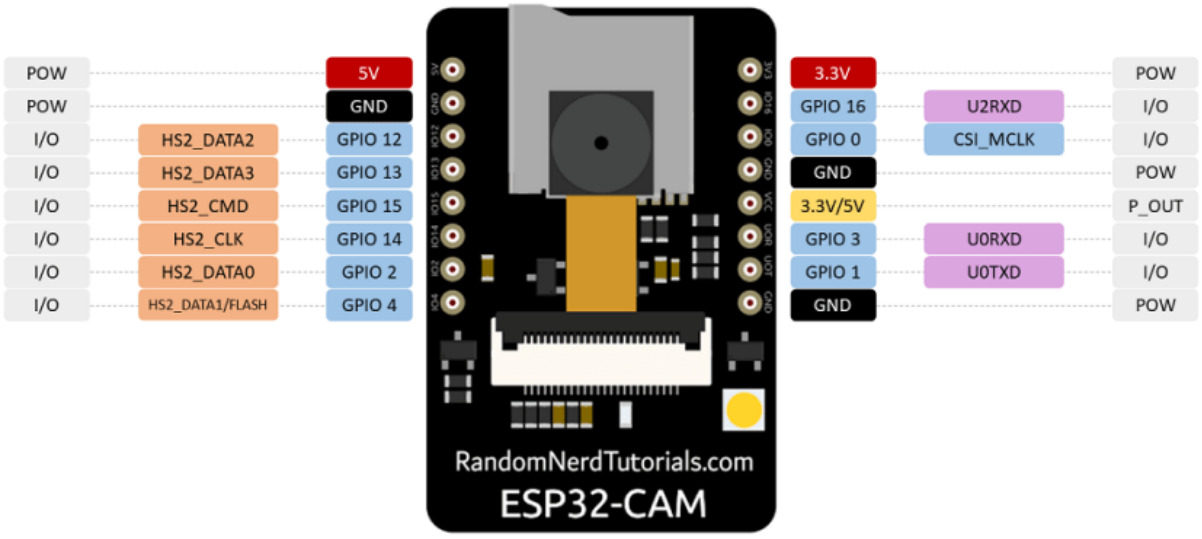
- **ESP32-CAM Module:** Equipped with an OV2640 camera sensor for capturing images in JPEG format.
- **Wi-Fi Connectivity:** Connects the ESP32-CAM to the internet for uploading captured images to Supabase.
- **Supabase Storage:** Stores uploaded photos with timestamped filenames for easy identification.
- **Backend OCR Processing:** A Python or cloud-based backend automatically fetches new images from Supabase, performs OCR to extract the plate number, and logs it into the Supabase database.

Placement and Orientation

- Cameras are mounted near the entrance and exit gates at approximately 2 meters height, angled for optimal plate visibility regardless of lighting conditions.

Trigger Mechanism

- ESP32-CAMs remain in standby mode, continuously checking a "camera_request" flag in the Supabase database.
- When the gate ESP32 detects a vehicle via IR sensor, it sets the camera request flag to true.
- The corresponding ESP32-CAM detects this change, captures the image, uploads it, and resets the flag to false.



6.2.4.3 Data Flow and Communication

The ESP32-CAM modules communicate with Supabase through HTTP requests and a simple database polling mechanism, enabling real-time, cloud-based image capture and recognition.

1. Trigger and Image Capture

- The gate ESP32 controller sends a PATCH request to Supabase, updating a flag in the camera_request table.
- The ESP32-CAM checks this flag every few seconds.
- When the flag is true, the camera:
    - Initializes the camera module.
    - Captures a fresh JPEG image.
    - Generates a timestamped filename.

2. Uploading to Supabase Storage

- The ESP32-CAM sends a POST request containing the image buffer to the appropriate Supabase Storage bucket (entrance or exit).
- The file is uploaded with a unique name such as photo_2025-06-13_10-45-30.jpg.

3. Post-Capture Process

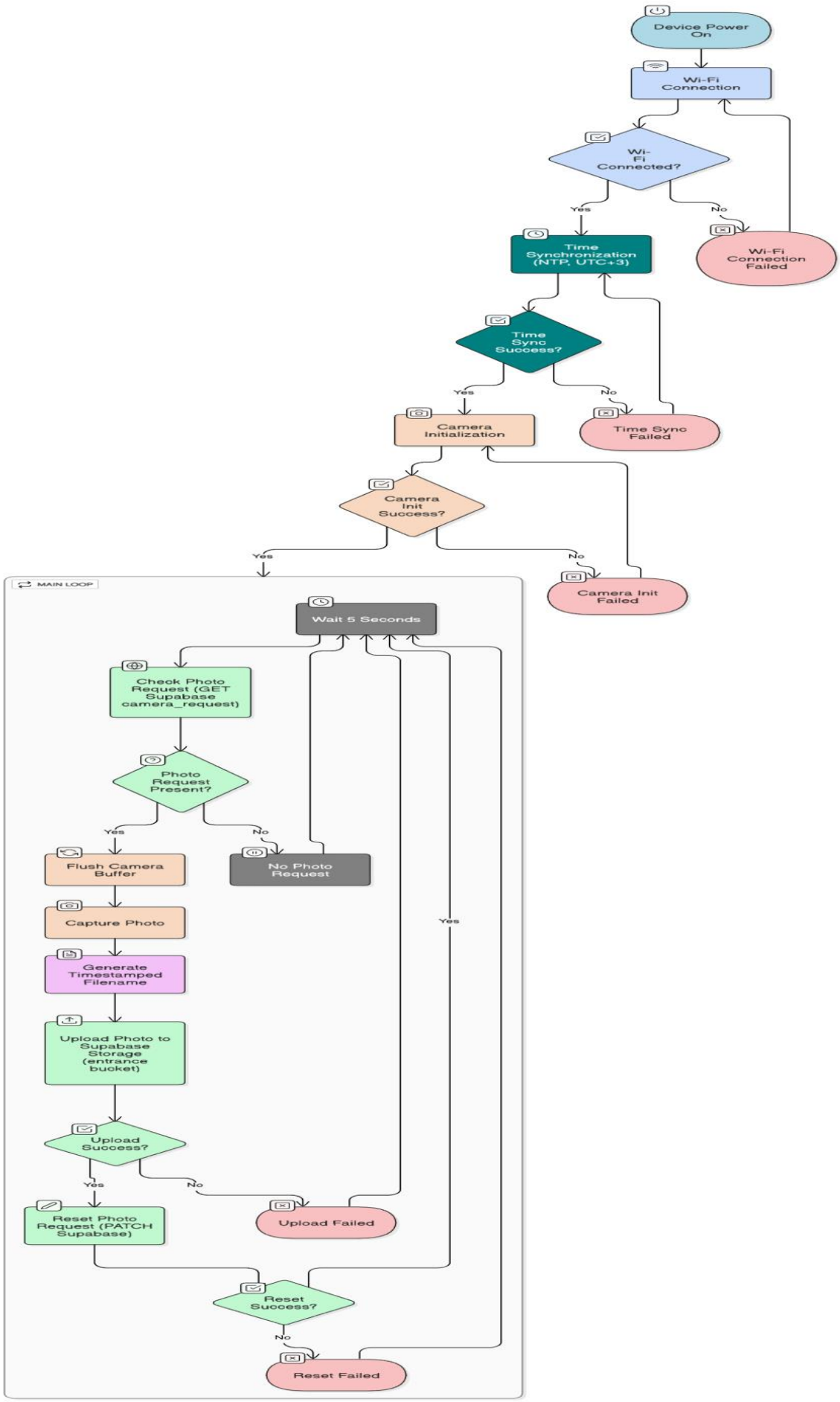
- After a successful upload, the ESP32-CAM resets the camera_request flag to false to avoid duplicate captures.
- Simultaneously, a backend OCR service monitors the Storage bucket:
    - When a new image is detected, the image is downloaded and processed.
    - The extracted plate number is inserted into the logs table in Supabase along with the photo URL, timestamp, and gate type (entry/exit).

4. Integration with System Logic

- The gate controller uses the updated logs and OCR results to validate access and manage gate operations.
- Admins can view uploaded images and their associated data via the web dashboard.

Error Handling & Reliability

- If image upload fails, the ESP32-CAM retries the request after a short delay.
- Camera initialization is checked during each capture to prevent failed images.
- Time synchronization via NTP ensures accurate and unique image filenames.


Device Power On
Wi-Fi Connection
Wi-Fi Connected?
Yes
No
Wi-Fi Connection Failed
Time Synchronization (NTP, UTC+3)
Time Sync Success?
Yes
No
Time Sync Failed
Camera Initialization
Camera Init Success?
Yes
No
Camera Init Failed
MAIN LOOP
Wait 5 Seconds
Check Photo Request (GET Supabase camera_request)
Photo Request Present?
Yes
No
No Photo Request
Flush Camera Buffer
Capture Photo
Generate Timestamped Filename
Upload Photo to Supabase Storage (entrance bucket)
Upload Success?
Yes
No
Upload Failed
Reset Photo Request (PATCH Supabase)
Reset Success?
No
Reset Failed
Yes

6.2.4.4 Code Logic

Used Libraries:

Library	Purpose
WiFi.h	Wi-Fi connection setup
HTTPClient.h	HTTP GET, PATCH, POST to/from Supabase
esp_camera.h	ESP32-CAM camera interface
time.h	NTP-based time synchronization

1.Wi-Fi Connection

```
connectWiFi();
```

- Connects ESP32-CAM to a predefined Wi-Fi network using SSID and password.

2.Time Synchronization via NTP

```
initTime();
```

- Uses NTP to get the current time.
- Required to generate timestamped filenames for image uploads (e.g., photo_2025-06-08_12-45-30.jpg).

3.Camera Initialization

```
initCamera();
```

- Sets all necessary GPIOs and parameters (resolution, JPEG quality) for the OV2640 camera.
- Restarts device on failure.

4. Polling Camera Request Flag

```
shouldTakePhoto();
```

- Sends HTTP GET to Supabase's camera_request table.
- If request = true, the ESP32-CAM proceeds to capture and upload an image.

5. Resetting the Flag

```
markRequestFalse();
```

- After capturing and uploading a photo, this function sets request = false in Supabase to prevent redundant captures.

6. Image Capture

```
captureFreshPhoto();
```

- Flushes previous camera frames (up to 3).
- Ensures a clean and up-to-date frame is captured for upload.

7. Image Upload to Supabase Storage

```
uploadImageToSupabase(fb->buf, fb->len);
```

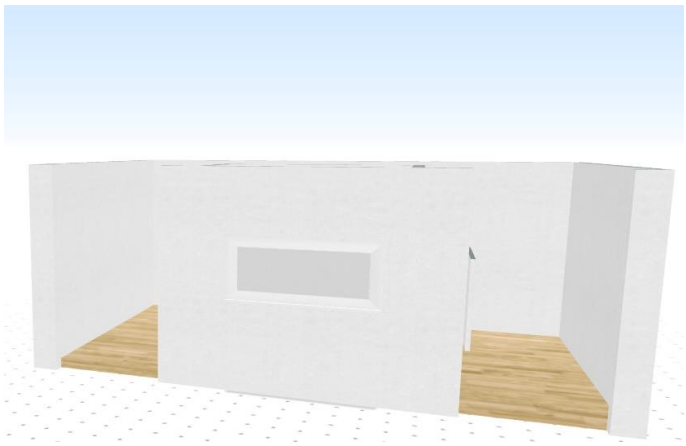
- Uploads the JPEG image to a predefined Supabase bucket (entrance or exit) using a RESTful POST request.
- File name includes a timestamp for easy tracking and retrieval.

Main Loop Behavior

Every 5 seconds:

1. Checks Supabase for a true value in camera_request.
2. If triggered:
    - Captures a fresh image.
    - Uploads to Supabase with a timestamped name.
    - Sets request = false.
3. If not triggered, prints "⏸ No photo requested."

6.2.6 Maquette (System Prototype)



- **Parking Slots:**
    - 5 slots with IR sensors mounted at ground level (detects vehicle presence).
    - LED indicators (red/green) visible above each slot.
- **Gates:**
    - Entrance/Exit Gates: Servo motors attached to barrier arms.
    - IR sensors placed 1 meter before gates to trigger detection.
- **ESP32-CAMs:**
    - Mounted at 2m height near gates for clear vehicle image capture.

6.3 Machine Learning

6.3.1 Abstract

This section details the Machine Learning (ML) phase of the **"El-Sayes" Smart Garage System**, focusing on the development and deployment of an automated **License Plate Recognition (LPR)** pipeline. The primary objective is to provide a robust, real-time solution for vehicle identification, which is critical for automating the garage's entry and exit procedures.

The core challenge lies in accurately processing Egyptian license plates, which have a unique structure combining Arabic letters and Eastern Arabic numerals with a right-to-left reading format. To address this, we engineered an improved, multi-stage workflow that deviates from our initial proposals to enhance modularity and performance.

Our final, end-to-end pipeline integrates several key technologies:

- A custom-trained **YOLOv8** model for precise license plate detection.
- Cloud-based **Optical Character Recognition (OCR)** using a specialized Roboflow model accessed via its API.
- A custom post-processing algorithm to correctly assemble the recognized characters into the valid Egyptian plate format.
- Automated data logging and integration with a **Supabase** cloud database.

The model was trained on the **Egyptian Cars Plates Dataset** from Kaggle, which was selected for its high relevance and diverse collection of real-world images. This report provides a detailed explanation of the system's architecture, the methodology behind our technology choices, and the implementation of the full LPR pipeline.

6.3.2 Introduction

The **Smart Garage System** aims to provide a seamless and efficient solution for managing parking spaces, monitoring vehicle entry/exit, and offering real-time updates to users. A critical component of this system is the Automated **License Plate Recognition (LPR)** module, which uses machine learning to automatically detect and recognize license plates from camera feeds. This module will be integrated with the IoT system to enable real-time monitoring and management of parking spaces.

In the ML phase, we focus on developing a **YOLOv8 model** for license plate detection and an **OCR pipeline** for character recognition. The model will be trained on a dataset of Egyptian license plates, preprocessed to ensure consistency and quality, and integrated with the IoT system for real-time inference. Our choice of tools, frameworks, and methodologies is guided by the need for **accuracy**, **real-time performance**, and **scalability**.

Automated LPR is the cornerstone of the "El-Sayes" system, enabling a seamless, secure, and efficient user experience by eliminating the need for manual checks, RFID cards, or other physical tokens. The primary challenge lies in the unique format of Egyptian license plates, which consist of both Arabic letters and "Eastern Arabic" numerals, read from right to left in a specific grouping (letters, then numbers).

The goals for this ML component were:

1. **High Accuracy:** Reliably detect license plates in various real-world conditions (e.g., different angles, lighting, and partial occlusions).
2. **Real-Time Performance:** Process images with minimal latency to ensure smooth entry and exit from the garage.
3. **Specific Domain Adaptation:** Correctly interpret and format the unique structure of Egyptian license plates.

To achieve this, we developed a sophisticated pipeline that leverages state-of-the-art deep learning models for detection and recognition, paired with domain-specific logic for formatting.

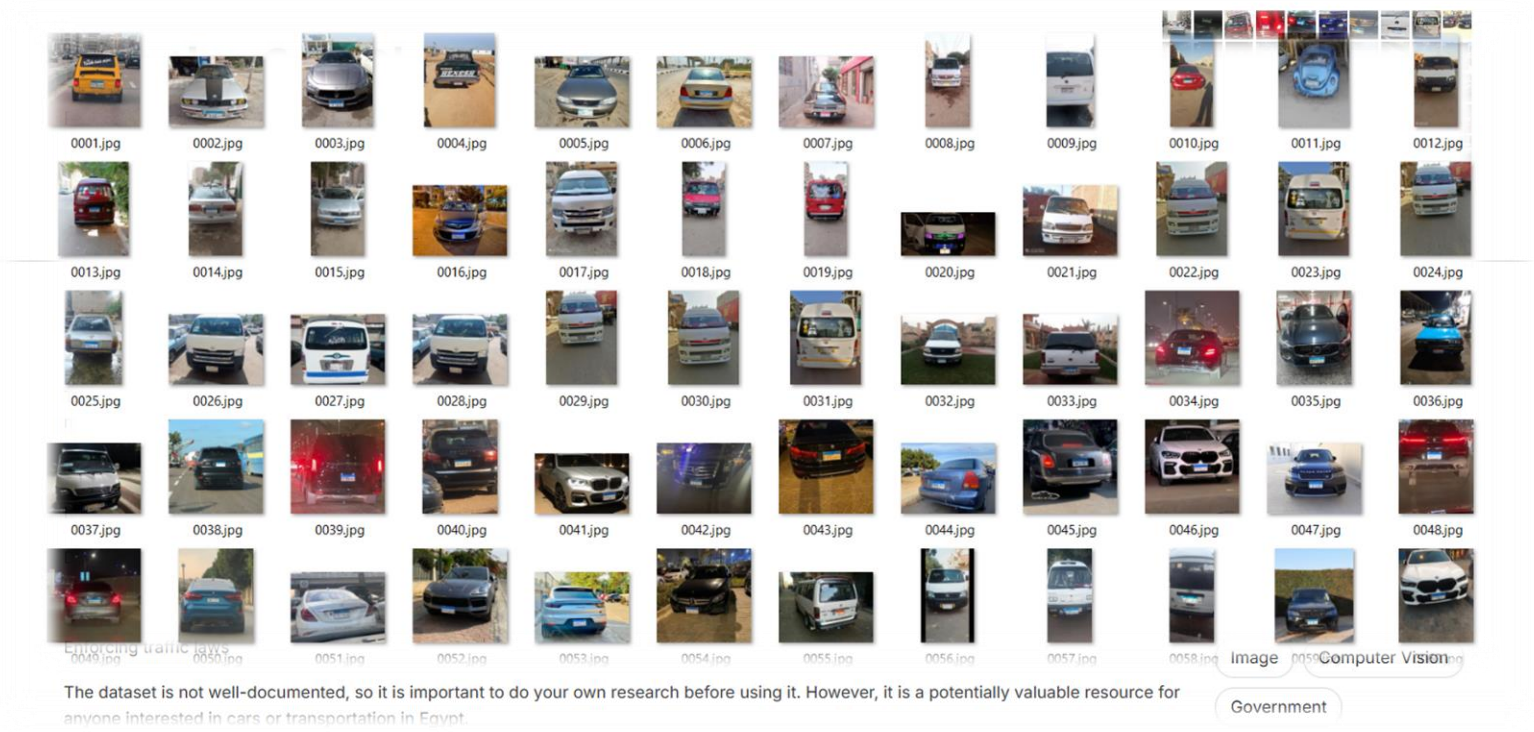
6.3.3 Dataset Selection

The performance of any deep learning model is fundamentally dependent on the quality of the training data.

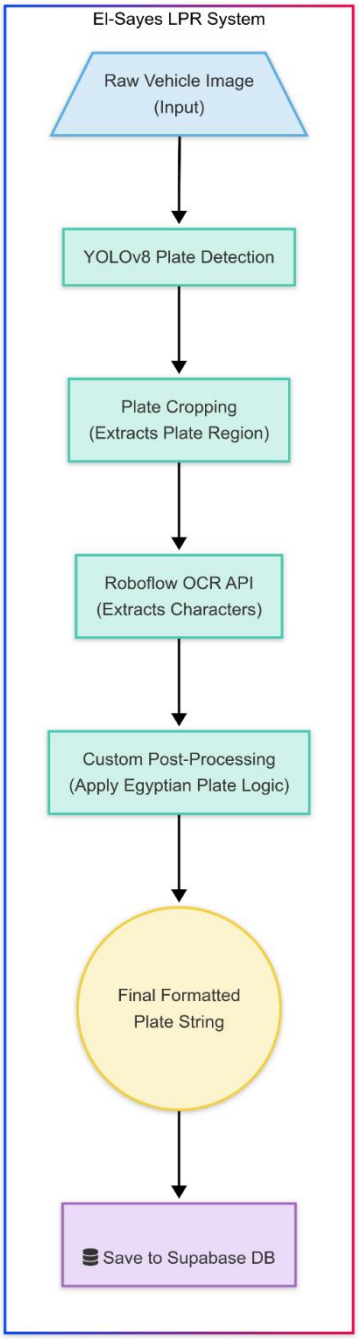
- **Dataset Selection:** We utilized the **Egyptian Cars Plates Dataset** available on Kaggle. This dataset was chosen for its relevance and diversity, containing images of Egyptian vehicles under various real-world conditions, which is crucial for training a robust and generalizable model.
- **Annotation and Formatting:** The raw dataset was processed to conform to the YOLOv8 format.

(https://www.kaggle.com/datasets/mahmoudeldebase/egyptian-cars-plates/data)

And here is a sample of how the data looks like:



1. **Relevance:** The dataset contains images of Egyptian license plates, making it highly suitable for our target application.
2. **Diversity:** The dataset includes license plates under various conditions (e.g., different lighting, angles, and backgrounds), which helps in training a robust model.
3. **Availability:** The dataset is publicly available and well-structured, reducing the time required for data collection and annotation.
4. **Size:** With a sufficient number of images, the dataset provides a good foundation for training a deep learning model.



6.3.4 Methodology and System Architecture

Our LPR system is architected as a sequential four-stage pipeline. This modular design allows for independent optimization of each component and ensures clarity in the workflow.

Overall Workflow

The end-to-end process, from image capture to final plate number extraction, is illustrated below. Each stage feeds its output to the next, creating a fully automated data flow.

The diagram showing: Raw Image -> YOLOv8 Plate Detection -> Plate Cropping -> Roboflow OCR -> Custom Formatting -> Final Plate String.

6.3.5 Why YOLOv8?

The selection of YOLOv8 (You Only Look Once version 8) as the core of our license plate detection system was based on a comprehensive evaluation of its inherent strengths and a comparative analysis against other leading architectures. The YOLO family of models is ideal for this project, and the v8 iteration, specifically, offers the best combination of performance, flexibility, and ease of use for our requirements.

We selected YOLOv8 as the model architecture for several compelling reasons:

1. Pre-trained Models for Object Detection

- YOLO is a state-of-the-art object detection framework that is pre-trained on large datasets like **COCO**, meaning it already has a robust understanding of detecting objects in images.
- We can **fine-tune** YOLO on our specific dataset (Egyptian license plates) instead of starting from scratch, saving both time and computational resources.

2. End-to-End Pipeline

- YOLO can directly detect license plates in images and output their locations as bounding boxes.
- With YOLO, we won't need a separate step for license plate localization. This simplifies our workflow compared to a custom CNN, where we might need one model for plate detection and another for character recognition.

3. Real-Time Performance

- YOLOv8 is optimized for real-time object detection, making it ideal for applications like license plate recognition in a smart garage system.
- It can process video feeds and make predictions in milliseconds per frame, ensuring smooth and efficient real-time performance.

4. Accuracy

- YOLOv8 achieves state-of-the-art accuracy in object detection tasks, ensuring reliable performance even under challenging conditions.
- This high accuracy is crucial for ensuring that the system correctly identifies license plates in various environments.

5. Ease of Use

- The Ultralytics implementation of YOLOv8 provides a user-friendly API for training, evaluation, and deployment.
- This ease of use allows us to focus on the project's goals rather than spending excessive time on setup and configuration.

6. Flexibility

- YOLOv8 supports various model sizes (e.g., YOLOv8n, YOLOv8s, YOLOv8m), allowing us to balance accuracy and inference speed based on system requirements.
- This flexibility is essential for optimizing the system's performance based on the available hardware resources.

7. Robustness and Versatility

- YOLO is highly accurate in detecting objects under various conditions (e.g., different lighting, angles, and partial occlusions). This robustness is hard to match with a custom-built CNN without a very large dataset.
- YOLO's object detection capabilities can be extended to other tasks, like detecting the car model or color, in the future.

8. Wide Community Support

- YOLO has extensive documentation, tutorials, and an active community. If we face issues, we'll find plenty of resources to help troubleshoot or optimize our implementation.

Comparative Analysis: YOLOv8 vs. Other Models

While the general advantages of YOLO are clear, selecting version 8 specifically was a deliberate engineering decision based on its position relative to other models at the time of project development.

- **Against Other Architectures (e.g., Faster R-CNN, SSD):** Two-stage detectors like Faster R-CNN, while accurate, are too slow for our real-time requirements. Other single-stage detectors like SSD have generally been surpassed by modern YOLO versions in both speed and accuracy, especially for detecting smaller objects.
- **Against Other YOLO Versions:** The choice of v8 over older and newer versions was based on finding the optimal point of performance, stability, and ecosystem maturity. YOLOv8's anchor-free design and improved network modules offered a clear performance benefit over older versions like YOLOv5. Simultaneously, it provided a stable, well-documented, and reliable platform, mitigating the risks associated with newer, "bleeding-edge" versions like YOLOv9 and beyond, which lacked the same level of community support and proven stability during our development window.

Summary of Model Comparison

The following table summarizes the comparative analysis that led to our decision.

Model	Key Feature	Advantages for This Project	Reason Not Chosen for This Project
Faster R-CNN	Two-stage detection	High accuracy on complex datasets.	Too slow for real-time gate automation.
SSD	Single-stage detector.	Fast inference speed.	Generally, less accurate than modern YOLO versions.
YOLOv5	Anchor-based, single-stage detector.	Mature and well-regarded.	Less efficient anchor-based design; superseded by YOLOv8's performance and usability.

YOLOv8 (Chosen)	Anchor-free, C2f module, unified Ultralytics API.	Optimal balance of state-of-the-art speed/accuracy and a user-friendly, stable framework that accelerated development.	-
YOLOv9 & Newer	Cutting-edge architectures and features.	Theoretically higher accuracy.	Too new at the time of development; lacked the mature ecosystem, less community support and stability of YOLOv8, posing a project risk.

6.3.6 Why Roboflow and why not using a Custom OCR Model?

Roboflow API vs. Custom OCR Model: A Strategic Trade-off

Feature	Roboflow API (Current Approach)	Custom OCR Model (Future Work)
Development Speed	**Extremely Fast.** Integration is a single API call.	**Very Slow.** Requires extensive effort in data collection, annotation, training, and tuning.
Initial Cost & Effort	**Low.** The main cost is the API usage fee, but the development effort is minimal.	**High.** Requires significant developer hours and potentially expensive GPU time for training.

Performance & Accuracy	**High Initial Accuracy.** Leverages a model trained on vast, diverse datasets.	**Variable.** Performance depends entirely on the quality and quantity of your training data. It can be difficult to match Roboflow's accuracy without a massive dataset.
Maintenance	**Zero.** The model is maintained and updated by Roboflow.	**High.** You are responsible for all maintenance, bug fixes, and retraining.
Offline Capability	**No.** Requires a constant internet connection.	**Yes.** Can be deployed on-device for fully offline inference.
Dataset Requirement	**None.** You only need the images you want to run inference on.	**Massive.** Requires thousands of labeled images of individual characters for robust training.

6.3.7 The LPR System's Data Workflow

1. **Data Input from IoT system:**
    - The process is triggered when a vehicle arrives at a gate, and an ESP32-CAM captures an image of its license plate.
    - This image is uploaded to **Supabase Storage**, serving as the direct input for the ML pipeline

2. **License Plate Detection (YOLOv8):**

    - A background Python service detects the new image upload in Supabase Storage.
    - The fine-tuned **YOLOv8 model** processes the image to detect the bounding box coordinates of the license plate region.

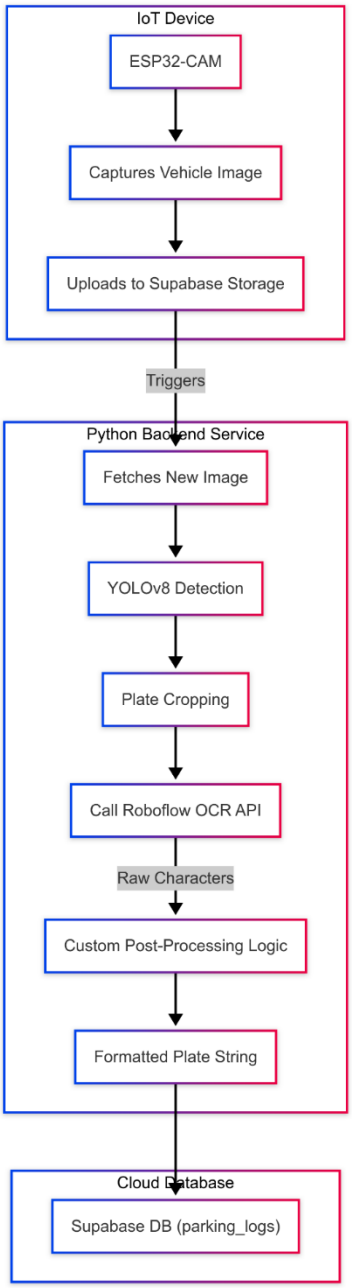
3. **Plate Cropping and Preprocessing:**

    - Using the coordinates from YOLOv8, the license plate is precisely cropped from the larger vehicle image.
    - This smaller, isolated image of the plate is then prepared for character recognition.

4. **Character Recognition (via Roboflow OCR API)**

    - Instead of a generic local OCR, the cropped plate image is sent to a specialized, cloud-hosted **Roboflow OCR model** via a REST API call.
    - The API returns a structured JSON response containing the recognized characters and their coordinates..

5. **Custom Post-Processing and Formatting:**



This is the most critical step, designed specifically for the unique format of Egyptian plates. A custom Python script takes the raw OCR output and performs the following:

- Validates and sorts the recognized characters based on their horizontal position to ensure the correct right-to-left order.
- Separates the characters into two distinct groups: letters and numbers.
- Re-orders the number group to match the correct visual sequence.
- Combines the groups to assemble the final, correctly formatted license plate string.

6. **Integration and Logging in Supabase**

    - The final, formatted plate number is logged in the **Supabase database**.
    - A new entry is created or updated in the cars and parking_logs tables, linking the vehicle to the entry/exit event with a timestamp. This makes the data immediately available to the larger "El-Sayes" system, including the admin dashboard and user applications.

6.3.8 Phases Breakdown

The development of the LPR system was structured into four distinct phases, from data preparation to final deployment and integration.

6.3.8.1 Phase 1: Dataset Preparation and Annotation

Objective: Prepare a high-quality, annotated dataset suitable for training a robust license plate detector.

Tasks:

1. **Data Collection:**

    - Sourced images from the *Egyptian Cars Plates Dataset* on Kaggle.
    - Ensured the dataset included a diverse range of lighting conditions, plate angles, and vehicle types to improve model generalization.

2. **Annotate Dataset:**

    - Used annotation tools like Roboflow to draw precise bounding boxes around each license plate.
    - Exported annotations in the required YOLO .txt format, with each file containing the class and normalized bounding box coordinates.

3. **Data Standardization and Augmentation:**

    - Applied preprocessing steps including resizing images to a consistent resolution.
    - Utilized data augmentation techniques (e.g., brightness adjustments, rotation) to artificially expand the dataset and make the model more robust.

Deliverables:

- A fully annotated and augmented dataset in YOLOv8 format.
- A data.yaml configuration file for the training process.

6.3.8.2 Phase 2: Training YOLO for License Plate Detection

Objective: Train a high-performance YOLOv8 model to accurately detect and localize Egyptian license plates.

Tasks:

1. **Framework Setup:**
    - Installed the Ultralytics framework to leverage the YOLOv8 architecture.
    - Configured the development environment on a local machine equipped with a GPU to accelerate training.
2. **Model Training:**
    - Employed **transfer learning** by starting with weights from a YOLOv8 model pre-trained on the **COCO** dataset.
    - **Fine-tuned** the model on our custom **Egyptian license plate dataset**.
    - Monitored key metrics during training, including precision, recall, and mean Average Precision (mAP), to assess performance.
3. **Model Evaluation:**
    - Tested the best-performing model on a **validation** set of images that **were not used** during training to ensure its real-world accuracy.
    - Ensure the model detects license plates accurately under various conditions.

Deliverables:

- A trained YOLOv8 model with saved weights (.pt file) for license plate detection.
- Evaluation report and graphs showing the model's training metrics (mAP, precision, recall).

6.3.8.3 Phase 3: OCR Integration and Character Recognition

Objective: Integrate a cloud-based OCR service for character recognition and develop a custom formatting pipeline for Egyptian plates.

Tasks:

1. **Integrate Roboflow OCR API:**
    - Developed a Python script to send cropped license plate images to a specialized, pre-trained **Roboflow OCR model** via its REST API.
    - This approach was chosen over building a local model (like Tesseract) to leverage a high-accuracy, managed service and accelerate development.
2. **Automate Plate Cropping:**
    - Created an automated pipeline that takes the bounding box output from the trained YOLOv8 model and uses it to crop the plate region from the source image.
3. **Develop Custom Post-Processing Logic:**
    - Engineered a crucial post-processing algorithm in Python to correctly interpret the raw JSON output from the OCR API.
    - This logic sorts recognized characters from right-to-left based on their coordinates and formats them according to the specific rules of Egyptian license plates (letters first, then numbers).

Deliverables:

- A functional Python script that orchestrates the full pipeline: plate detection, cropping, API call to OCR, and formatting.
- A verified post-processing function that accurately converts raw OCR data into valid Egyptian plate numbers.

6.3.8.4 Phase 4: Model Integration and Real-Time Testing

Objective: Deploy the full LPR pipeline as an automated backend service and integrate it with the Supabase cloud environment.

Tasks:

1. **Deploy the LPR Pipeline:**
   - Set up the Python pipeline to run as a backend service that continuously monitors the system.
2. **Implement Real-Time Trigger:**
   - Configured the service to be triggered automatically whenever a new vehicle image is uploaded to **Supabase Storage** by an IoT camera.
3. **Test Under Real-World Conditions:**
   - Tested the end-to-end system using a variety of images to simulate diverse environmental conditions (e.g., different lighting, moving vehicles).
4. **Integrate with Supabase Database:**
   - Saved the final, formatted plate numbers and corresponding metadata (timestamp, entry/exit) into the parking_logs table in the Supabase database, making the data accessible to the entire "El-Sayes" application.

Deliverables:

- A fully integrated LPR system that automatically processes images from Supabase Storage and logs the results back to the database.
- A final, deployment-ready LPR pipeline.

Tools Used in Each Step

Step	Tools
Annotation	LabelImg, Roboflow
YOLO Training	YOLOv8 (Ultralytics), PyTorch, Jupyter Notebook
OCR Framework	Roboflow OCR API
Preprocessing	OpenCV, Pillow (PIL)
Integration & Testing	Python, OpenCV, requests library
Database	Supabase (PostgreSQL)

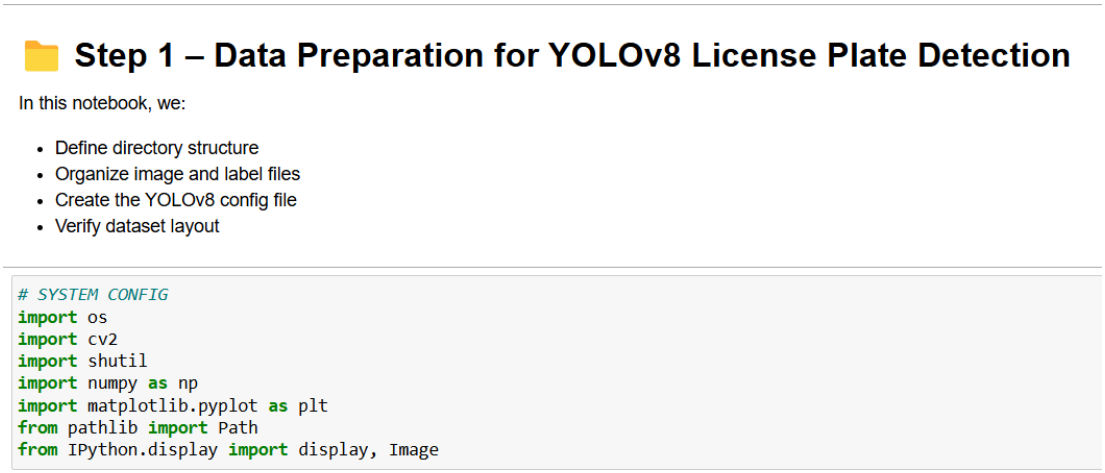
6.3.9 ML Implementation

The implementation of the License Plate Recognition (LPR) system was conducted in Python using Jupyter Notebooks, which allowed for modular development, experimentation, and clear visualization of results at each stage. The process was broken down into four primary stages: dataset preparation, model training, post-training processing, and the final end-to-end pipeline integration, followed by a thorough performance evaluation.

6.3.9.1 Stage 1: Dataset Preparation and Annotation (1_data_preparation.ipynb)

The foundation of any successful deep learning model is a high-quality, well-structured dataset. This initial phase focused on preparing the raw image data for training with the YOLOv8 framework.

1. Setup and Path Definition The first step involved importing the necessary Python libraries for file operations and image handling (pathlib, cv2, shutil) and defining the core directory paths. These paths point to the source folder containing the raw vehicle images, the folder with their corresponding annotation files, and the destination directory where the final YOLOv8-formatted dataset would be created.



📁 **Step 1 – Data Preparation for YOLOv8 License Plate Detection**

In this notebook, we:

- Define directory structure
- Organize image and label files
- Create the YOLOv8 config file
- Verify dataset layout

```
# SYSTEM CONFIG
import os
import cv2
import shutil
import numpy as np
import matplotlib.pyplot as plt
from pathlib import Path
from IPython.display import display, Image
```

```
# Define paths
base_path = Path(r"D:\College (FCDS)\Graduation Project\Graduation Project ML Phase\EALPR Vechicles dataset")
images_path = base_path / "Vehicles"
labels_path = base_path / "Vehicles Labeling"
yolo_dataset_path = base_path / "YOLOv8_dataset"
```

2. YOLOv8 Directory Structuring and Data Organization YOLOv8 requires a specific directory structure for training. The script programmatically creates the necessary subdirectories (/images/train and /labels/train). Following this, a loop iterates through every image in the source directory. For each image, it verifies that a corresponding .txt annotation file exists. Only these validated image-label pairs are copied into the newly created YOLOv8 directories. This critical step ensures that our training dataset is clean and free of unannotated images.

```
# Create YOLOv8 folder structure
(yolo_dataset_path / "images" / "train").mkdir(parents=True, exist_ok=True)
(yolo_dataset_path / "labels" / "train").mkdir(parents=True, exist_ok=True)

# Move valid images + labels
for img_file in images_path.glob("*.jpg"):
    label_file = labels_path / (img_file.stem + ".txt")
    if label_file.exists():
        shutil.copy(img_file, yolo_dataset_path / "images" / "train" / img_file.name)
        shutil.copy(label_file, yolo_dataset_path / "labels" / "train" / label_file.name)

print("✅ Dataset organized in YOLOv8 format.")
```

```
✅ Dataset organized in YOLOv8 format.
```

3. YOLO Configuration File (yolo_config.yaml) A YAML configuration file is required to tell the YOLOv8 trainer where to find the dataset and what the object classes are. The script automatically generates this yolo_config.yaml file. It dynamically inserts the path to the dataset and defines the single class name for our project: license_plate.

```
# Create YOLO config file
yaml_content = f"""
path: {yolo_dataset_path}
train: images/train
val: images/train
names: ['license_plate']
"""

with open(base_path / "yolo_config.yaml", "w") as f:
    f.write(yaml_content)

print("✅ YOLOv8 config file created.")
```

```
✅ YOLOv8 config file created.
```

4. Visualization and Verification To confirm that the dataset was prepared correctly, the final part of the notebook visualizes the data. It first displays a few raw images. Then, using a helper function (draw_yolo_box), it reads the annotation files and draws the bounding boxes directly onto the images. This provides immediate visual confirmation that the labels accurately correspond to the license plates in the images.

Before Annotation:

```
# 📷 Show raw vehicle images (individually, no annotations)
print("\n🔍 Sample of raw vehicle images (unannotated):")

# ✅ Get sample images
sample_raw_imgs = list((yolo_dataset_path / "images" / "train").glob("*.jpg"))[:5]

for img_path in sample_raw_imgs:
    img = cv2.imread(str(img_path))
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    plt.figure(figsize=(6, 4))
    plt.imshow(img)
    plt.axis("off")
    plt.title(f"Raw: {img_path.name}")
    plt.show()
```

🔍 Sample of raw vehicle images (unannotated):

Raw: 0001.jpg

Raw: 0002.jpg

After Annotation:

```python
# 🔧 Helper to draw YOLOv8-style bounding boxes (with green color)
def draw_yolo_box(image_path, label_path):
    img = cv2.imread(str(image_path))
    h, w = img.shape[:2]

    if not os.path.exists(label_path):
        return img

    with open(label_path, "r") as f:
        lines = f.readlines()

    for line in lines:
        class_id, x_center, y_center, width, height = map(float, line.strip().split())
        x1 = int((x_center - width / 2) * w)
        y1 = int((y_center - height / 2) * h)
        x2 = int((x_center + width / 2) * w)
        y2 = int((y_center + height / 2) * h)

        cv2.rectangle(img, (x1, y1), (x2, y2), color=(0, 255, 0), thickness=2)  # 💚 green
        cv2.putText(img, f"license_plate", (x1, y1 - 10),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)

    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    return img_rgb
```

```python
# 🗂️ Get sample annotated images
annotated_imgs = list((yolo_dataset_path / "images" / "train").glob("*.jpg"))[:5]

# 🎯 Show training images with bounding boxes (individually)
print("\n🎯 Sample of training images with bounding boxes:")

for img_path in annotated_imgs:
    label_path = yolo_dataset_path / "labels" / "train" / (img_path.stem + ".txt")
    img = draw_yolo_box(img_path, label_path)
    plt.figure(figsize=(6, 4))
    plt.imshow(img)
    plt.axis("off")
    plt.title(f"With BBox: {img_path.name}")
    plt.show()
```

```
🎯 Sample of training images with bounding boxes:
```

6.3.9.2 Stage 2: Training the YOLOv8 Detection Model (2_yolo_training.ipynb)

With the dataset prepared, this stage focused on training the object detection model.

1. Model Initialization and Transfer Learning The script begins by initializing a YOLOv8 model. We used yolov8n.pt, a lightweight version of YOLOv8 that has been pre-trained on the extensive COCO dataset. This technique, known as **transfer learning**, provides the model with a foundational knowledge of general object features, drastically reducing training time and improving final performance compared to training from scratch.

2. Model Training This is the core of the notebook, where the training process is launched with a single command: model.train(). We configured the training with key parameters:

- **data**: Path to the yolo_config.yaml file created in Stage 1.
- **epochs**: Set to 10, meaning the model trained on the entire dataset 10 times.
- **imgsz**: All images were resized to 416x416 pixels for consistent input.
- **batch**: A batch size of 16 was used to process images in parallel.
- **name**: The training results were saved in a directory named lp_detector.

```python
# SYSTEM CONFIG
from ultralytics import YOLO
from pathlib import Path

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

# Paths
base_path = Path(r"D:\College (FCDS)\Graduation Project\Graduation Project ML Phase\EALPR Vechicles dataset")

# Train
model.train(
    data=str(base_path / "yolo_config.yaml"),
    epochs=10,
    imgsz=416,
    batch=16,
    name="lp_detector",
    project=str(base_path),
    workers=4
)
```

Here is a sample from the Model Training epochs:

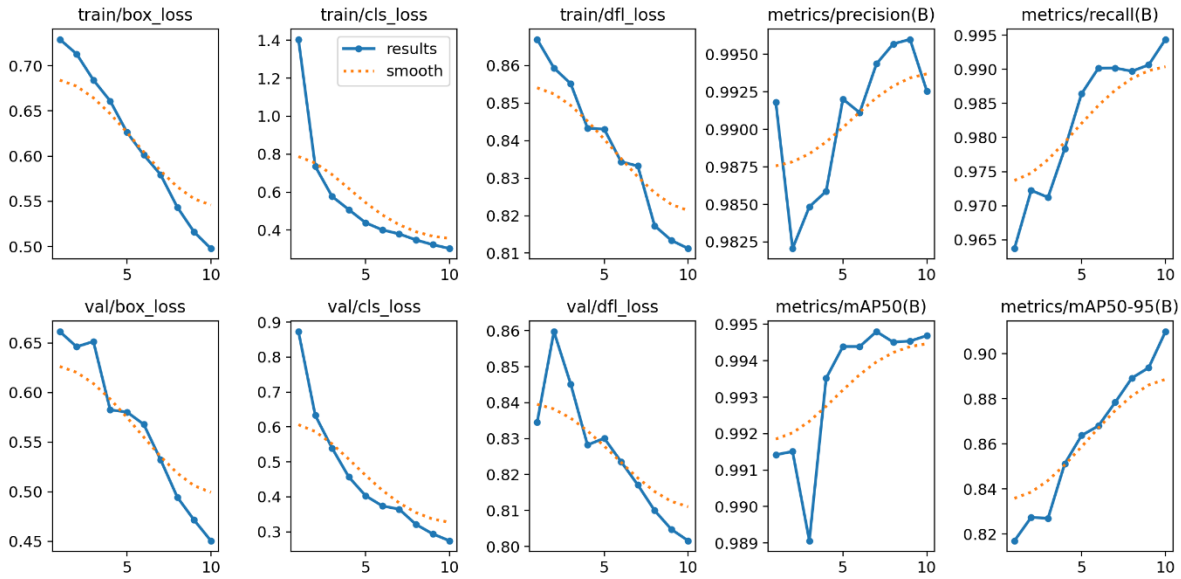
```
Plotting labels to D:\College (FCDS)\Graduation Project\Graduation Project ML Phase\EALPR Vechicles dataset\lp_detector2\label
s.jpg...
optimizer: 'optimizer=auto' found, ignoring 'lr0=0.01' and 'momentum=0.937' and determining best 'optimizer', 'lr0' and 'moment
um' automatically...
optimizer: AdamW(lr=0.002, momentum=0.9) with parameter groups 57 weight(decay=0.0), 64 weight(decay=0.0005), 63 bias(decay=0.
0)
Image sizes 416 train, 416 val
Using 0 dataloader workers
Logging results to D:\College (FCDS)\Graduation Project\Graduation Project ML Phase\EALPR Vechicles dataset\lp_detector2
Starting training for 10 epochs...
Closing dataloader mosaic
albumentations: Blur(p=0.01, blur_limit=(3, 7)), MedianBlur(p=0.01, blur_limit=(3, 7)), ToGray(p=0.01, num_output_channels=3, m
ethod='weighted_average'), CLAHE(p=0.01, clip_limit=(1.0, 4.0), tile_grid_size=(8, 8))

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       1/10         0G     0.7291      1.405     0.8669          5        416: 100%|██████████| 131/131 [03:34<00:00,
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 66/66 [01:27
                   all       2084       2140      0.991      0.964      0.991      0.817

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       2/10         0G     0.7129     0.7356     0.8593          4        416: 100%|██████████| 131/131 [03:31<00:00,
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 66/66 [01:25
                   all       2084       2140      0.982      0.972      0.992      0.827

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       3/10         0G     0.6841     0.5777     0.8551          4        416: 100%|██████████| 131/131 [03:31<00:00,
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 66/66 [01:24
                   all       2084       2140      0.985      0.971      0.989      0.827

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
```

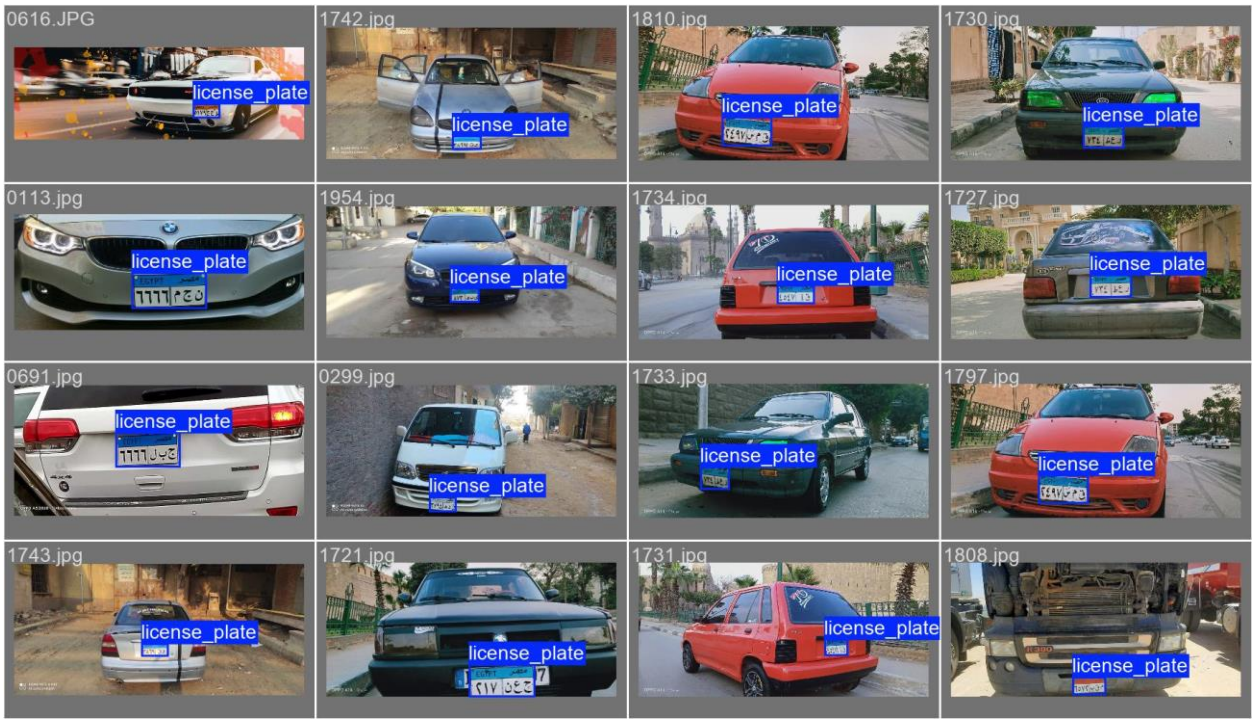
3. Verifying Training Results After the training concluded, the script loads the best-performing weights (best.pt) from the output directory. It then runs inference on a few sample images from the training set and displays the results. This step serves as a quick "sanity check" to visually confirm that the model learned to accurately detect and draw bounding boxes around license plates.

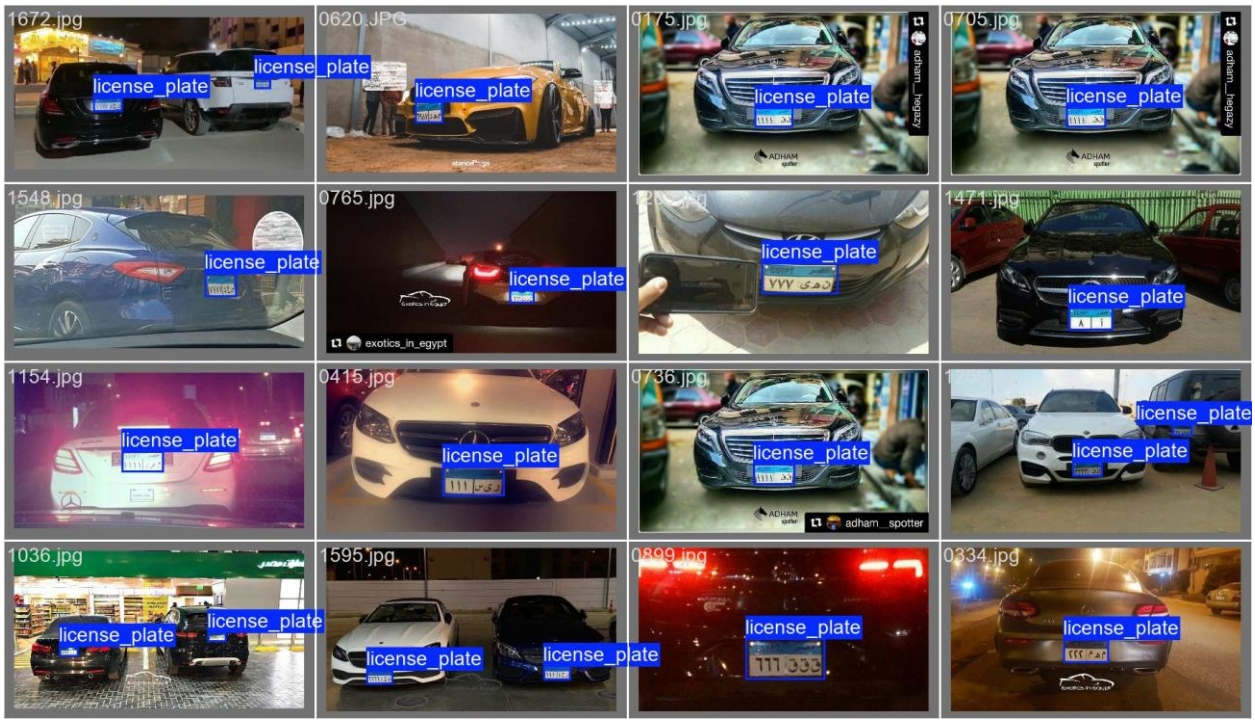
Visualizing Detection Results on Sample Images

After training, let's run inference on 5 images from the training set and plot their predicted license plates.

```
import matplotlib.pyplot as plt
import cv2

# Reload the trained model
model = YOLO(str(base_path / "lp_detector" / "weights" / "best.pt"))

# Load 5 sample images from the training set
train_imgs = list((base_path / "YOLOv8_dataset" / "images" / "train").glob("*.jpg"))[:5]
results = model(train_imgs)

# Show prediction results
for i, r in enumerate(results):
    im_array = r.plot()
    plt.imshow(im_array[..., ::-1])
    plt.title(f"Prediction: {train_imgs[i].name}")
    plt.axis("off")
    plt.show()
```




0616.JPG
1742.jpg
1810.jpg
1730.jpg
0113.jpg
1954.jpg
1734.jpg
1727.jpg
0691.jpg
0299.jpg
1733.jpg
1797.jpg
1743.jpg
1721.jpg
1731.jpg
1808.jpg
license_plate


1672.jpg
0620.JPG
0175.jpg
0705.jpg
1548.jpg
0765.jpg
1471.jpg
1154.jpg
0415.jpg
0736.jpg
1036.jpg
1595.jpg
0899.jpg
0334.jpg
license_plate
exotics_in_egypt
adham_spotter

6.3.9.3 Stage 3: Post-Training Processing (3_plate_cropping.ipynb)

Before building the final pipeline, this script was used to process the entire dataset with our trained model. The goal was to detect all license plates and save them as individual cropped images. This provided a dataset of isolated plates that was essential for developing and testing the subsequent OCR and formatting stages.

The script iterates through every image in our dataset, runs the trained YOLOv8 model to get bounding box coordinates, and then uses OpenCV to crop and save each detected plate region into a dedicated output folder (/cropped_plates).

```
# 📝 Optional log file for corrupted or unreadable images
log_file = open(base_path / "corrupt_files.txt", "a")

for img_path in image_dir.glob("*.jpg"):
    img = cv2.imread(str(img_path))

    # ❌ Skip unreadable images
    if img is None:
        print(f"⚠️ Skipping unreadable image: {img_path}")
        log_file.write(str(img_path) + "\n")
        continue

    # 🧠 Run YOLOv8 inference
    results = model(img_path)
    boxes = results[0].boxes.xyxy.cpu().numpy()

    # ✂️ Crop and save each detected plate
    for i, box in enumerate(boxes):
        x1, y1, x2, y2 = map(int, box[:4])
        crop = img[y1:y2, x1:x2]

        crop_name = f"{img_path.stem}_plate{i}.jpg"
        cv2.imwrite(str(output_crop_dir / crop_name), crop)

log_file.close()
print("✅ Cropping complete (corrupt images skipped).")
```

```
image 1/1 D:\College (FCDS)\Graduation Project\Graduation Project ML Phase\EALPR Vechicles dataset\YOLOv8_dataset\images\trai
n\0001.jpg: 416x416 2 license_plates, 226.9ms
Speed: 3.3ms preprocess, 226.9ms inference, 1.3ms postprocess per image at shape (1, 3, 416, 416)

image 1/1 D:\College (FCDS)\Graduation Project\Graduation Project ML Phase\EALPR Vechicles dataset\YOLOv8_dataset\images\trai
n\0002.jpg: 320x416 1 license_plate, 49.4ms
Speed: 1.4ms preprocess, 49.4ms inference, 0.8ms postprocess per image at shape (1, 3, 320, 416)

image 1/1 D:\College (FCDS)\Graduation Project\Graduation Project ML Phase\EALPR Vechicles dataset\YOLOv8_dataset\images\trai
n\0003.jpg: 416x320 1 license_plate, 48.8ms
Speed: 1.4ms preprocess, 48.8ms inference, 0.9ms postprocess per image at shape (1, 3, 416, 320)

image 1/1 D:\College (FCDS)\Graduation Project\Graduation Project ML Phase\EALPR Vechicles dataset\YOLOv8_dataset\images\trai
n\0004.jpg: 416x320 1 license_plate, 41.6ms
Speed: 1.5ms preprocess, 41.6ms inference, 0.9ms postprocess per image at shape (1, 3, 416, 320)
```

Sample from the cropped images:

And here is a sample from the cropped_plates file that is made through this script:

6.3.9.4 Stage 4: Final End-to-End LPR Pipeline Implementation (Final Version CV.ipynb)

This notebook integrates all the previous components into a single, functional pipeline that replicates the final automated workflow of the LPR system.

1. Helper Functions and Configuration The script begins by defining several key components:

- **Helper Functions:** load_yolo_model() and detect_and_crop() encapsulate the logic for loading our trained model and performing detection on an input image.
- **API Configuration:** The OCR_URL variable is defined with the specific endpoint for our trained Roboflow OCR model.
- **Character Map (label_map):** A crucial Python dictionary is defined to map the English class names returned by the Roboflow API

  (e.g., '3', '6', '2', '5', 'ain', 'baa', 'seen')

to their corresponding Arabic characters (e.g., ٣٦٢٥سبع).

```
# English-to-Arabic character map "Same As each class is named in Roboflow"
label_map = {
    "0": "٠", "1": "١", "2": "٢", "3": "٣", "4": "٤",
    "5": "٥", "6": "٦", "7": "٧", "8": "٨", "9": "٩",
    "alif": "أ", "baa": "ب", "taa": "ت", "thaa": "ث",
    "jeem": "7" ,"gaa": "ح", "khaa": "خ", "daal": "د",
    "zaal": "ذ", "raa": "ر", "zay": "ز", "seen": "س",
    "sheen": "ش", "saad": "ص", "daad": "ض", "Taa": "ط",
    "zaa": "ظ", "ain": "ع", "ghayn": "غ", "faa": "ف",
    "qaaf": "ق", "kaaf": "ك", "laam": "ل", "meem": "م",
    "noon": "ن", "haa": "ه", "waw": "و", "yaa": "ي"
}
```

2. Custom Formatting Algorithm The most innovative part of the project is the convert_to_arabic_with_digit_fix() function. Standard OCR reads from **left to right**, which is **incorrect** for Egyptian plates. This function solves that problem with the following logic:

1. It receives the list of character predictions from the **Roboflow API.**
2. It sorts all characters based on their horizontal (x-coordinate) position in **reverse order** to establish the correct right-to-left sequence.
3. It intelligently separates the sorted characters into two groups: **letters** and **digits**.
4. It reverses *only* the digits group to match the correct numerical reading order.
5. Finally, it **concatenates** the letters and the **correctly ordered** digits to produce the final, accurate license plate string.

Sample

```python
# File paths
INPUT_IMAGE_PATH = r"D:\College (FCDS)\Graduation Project\Graduation Project ML Phase\EALPR Vechicles dataset\Vehicles\1913.jpg"
YOLO_MODEL_PATH = r"D:\College (FCDS)\Graduation Project\Graduation Project ML Phase\EALPR Vechicles dataset\lp_detector\weights\

# Load YOLO model
print("📦 Loading YOLO model...")
yolo_model = load_yolo_model(YOLO_MODEL_PATH)

# Detect license plates
print("🔍 Detecting license plates...")
cropped_plates = detect_and_crop(INPUT_IMAGE_PATH, yolo_model)
print(f"✅ {len(cropped_plates)} plate(s) cropped.")

# OCR and final output
for idx, plate_img in enumerate(cropped_plates):
    print(f"\n📸 Plate #{idx + 1}")
    display(plate_img)

    # Send image to Roboflow
    buffered = io.BytesIO()
    plate_img.save(buffered, format="JPEG")
    img_base64 = base64.b64encode(buffered.getvalue()).decode("utf-8")

    response = requests.post(
        OCR_URL,
        headers={"Content-Type": "application/x-www-form-urlencoded"},
        data=img_base64,
    )

    if response.ok:
        data = response.json()
        predictions = data.get("predictions", [])

        raw_classes = [p["class"] for p in sorted(predictions, key=lambda x: x["x"])]
        print("🔠 OCR Predictions:", raw_classes)

        arabic_plate_list = convert_to_arabic_with_digit_fix(predictions)
        print("📄 Arabic Plate:", ''.join(arabic_plate_list))
    else:
        print("❌ OCR request failed.")
```

Outputs for the OCR Predictions:

```
📦 Loading YOLO model...
🔍 Detecting license plates...

image 1/1 D:\College (FCDS)\Graduation Project\Graduation Project ML Phase\EALPR Vechicles dataset\Vehicles\2016.jpg: 416x352 1
license_plate, 36.1ms
Speed: 1.8ms preprocess, 36.1ms inference, 0.8ms postprocess per image at shape (1, 3, 416, 352)
✅ 1 plate(s) cropped.

📸 Plate #1
```

```
🔠 OCR Predictions: ['1', '1', '1', '1', 'daal', 'yaa', 'seen']
📄 Arabic Plate: ١١١١سيد


📦 Loading YOLO model...
🔍 Detecting license plates...

image 1/1 D:\College (FCDS)\Graduation Project\Graduation Project ML Phase\EALPR Vechicles dataset\Vehicles\1913.jpg: 416x224 1
license_plate, 40.6ms
Speed: 1.2ms preprocess, 40.6ms inference, 1.2ms postprocess per image at shape (1, 3, 416, 224)
✅ 1 plate(s) cropped.

📸 Plate #1
```

```
🔠 OCR Predictions: ['3', '6', '2', '5', 'ain', 'baa', 'seen']
📄 Arabic Plate: سبع٣٦٢٥
```

3. Main Execution and Final Output the Roboflow API processes the cropped image and returns a JSON response containing the raw recognized characters and their coordinates. This provides the unordered data that our custom logic will then process, as illustrated below.

The main block of the script then simulates the full, real-world process by loading the model, running detection, calling the API, and applying the formatting logic. The screenshot below demonstrates the successful execution of the entire pipeline on a sample image, showcasing the speed and accuracy of the system from start to finish.

Plate #1

OCR Predictions: ['1', '2', '5', '2', 'alif', 'seen', 'qaaf']
Arabic Plate: ١٢٥٢قسأ

And here is a full image showing all the details and the time taken by the model to detect the car plate characters.

Loading YOLO model...
Detecting license plates...

image 1/1 D:\College (FCDS)\Graduation Project\Graduation Project ML Phase\EALPR Vechicles dataset\Vehicles\1628.jpg: 416x320 1 license_plate, 32.1ms
Speed: 1.1ms preprocess, 32.1ms inference, 0.6ms postprocess per image at shape (1, 3, 416, 320)
1 plate(s) cropped.

Plate #1

OCR Predictions: ['1', '2', '5', '2', 'alif', 'seen', 'qaaf']
Arabic Plate: ١٢٥٢قسأ

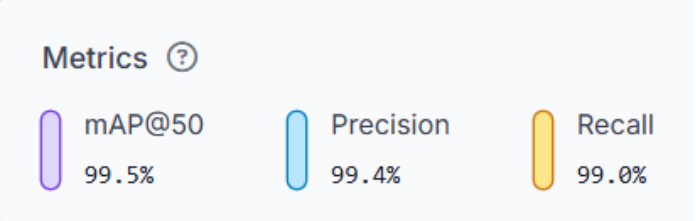
6.3.9.5 Model Performance and Evaluation

After the YOLOv8 model was successfully trained, its performance was rigorously evaluated to ensure it met the high-accuracy requirements of the "El-Sayes" system. The evaluation was based on standard object detection metrics calculated on the validation dataset.

1. Key Performance Metrics The model's effectiveness was measured using the following key metrics:

- **Precision:** Measures the accuracy of the model's predictions. A high precision indicates that when the model detects a license plate, it is highly likely to be correct.
- **Recall:** Measures the model's ability to find all relevant objects. A high recall indicates that the model successfully identified most of the actual license plates present in the images.
- **mAP@50 (mean Average Precision at IoU=0.50):** This is the primary metric for object detection. It provides a comprehensive score of the model's accuracy by combining precision and recall. A score of **99.5%** indicates that the model is exceptionally accurate at an Intersection over Union (IoU) threshold of 50%.

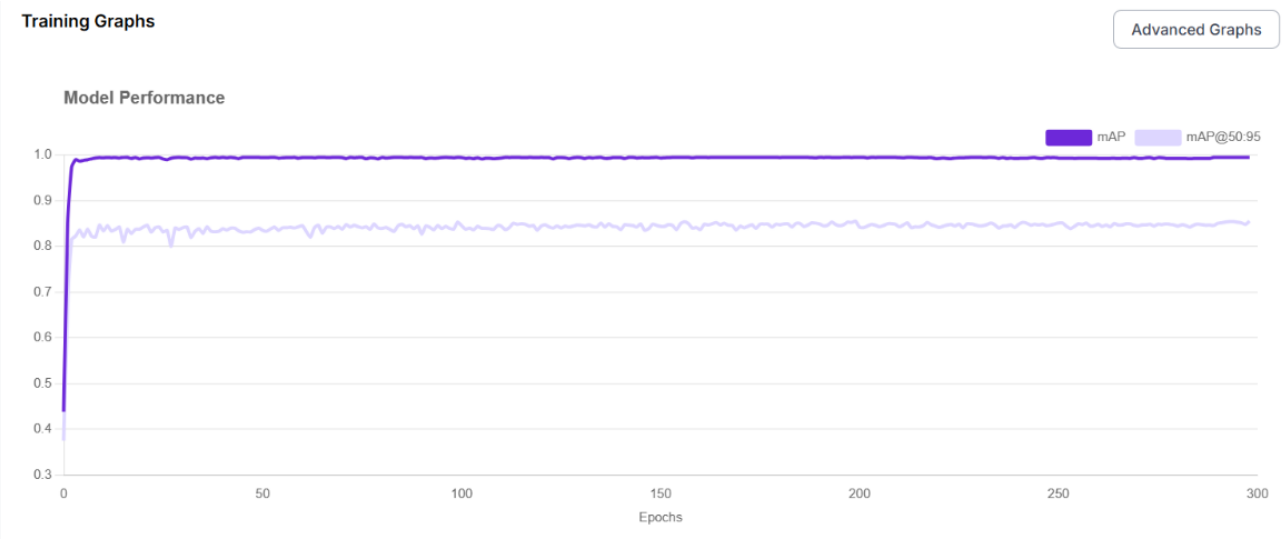
Our trained model achieved the following outstanding results on the validation set:

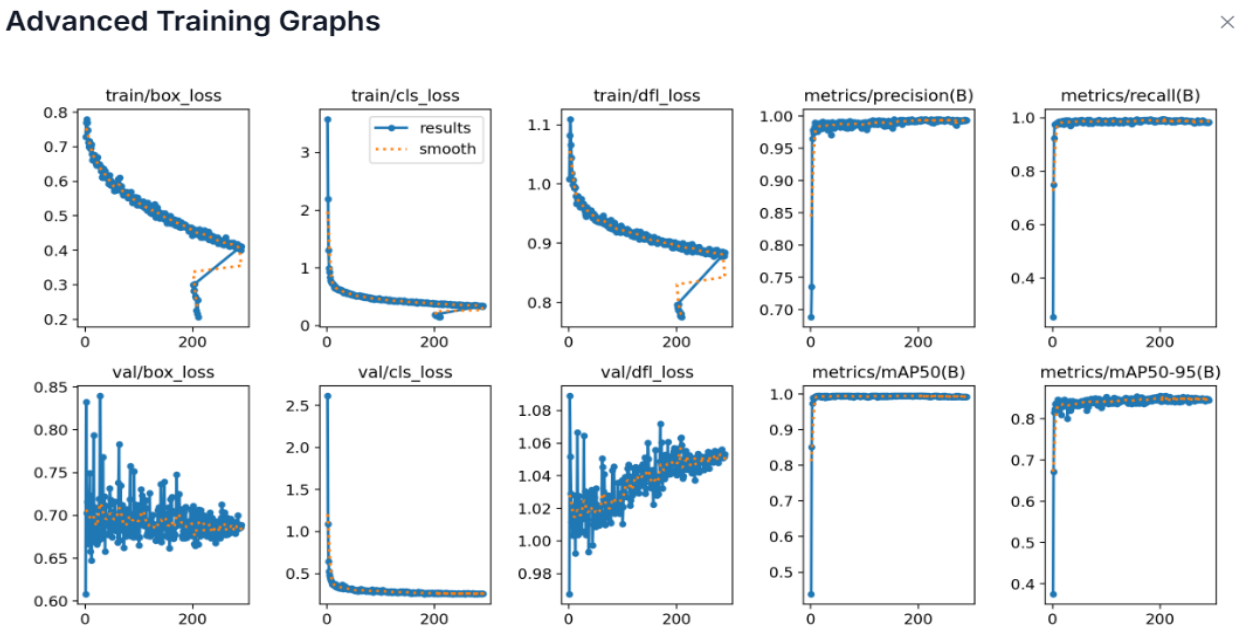


These top-tier metrics confirm that the detector is highly reliable and forms a solid foundation for the rest of the LPR pipeline.

2. Analysis of Training Graphs The training process was monitored through a series of graphs that visualize the model's learning over time. These graphs are essential for diagnosing the health of the training process and confirming that the model learned effectively without significant issues like overfitting.

- **Loss Curves:** The loss functions (Box Loss, Class Loss) measure the model's prediction error. As seen in the graphs, all loss values for both the training and validation sets decreased sharply in the initial epochs and then stabilized at a low value. This is the ideal behavior, indicating that the model successfully learned to minimize its errors. The validation loss (val/box_loss, val/cls_loss) closely tracks the training loss without diverging, which suggests the model generalized well and did not overfit.
- **Metric Curves:** The metric graphs (Precision, Recall, mAP) show that the model's performance on the validation set rapidly increased to a near-perfect level and then plateaued. This demonstrates that the model learned the task efficiently and maintained a high level of accuracy throughout the training process.





6.3.10 Challenges and Future Work

While the implemented LPR system is highly effective, its development was not without challenges. This section details the obstacles encountered and outlines potential enhancements that could make the system even more robust in future iterations.

6.3.10.1 Limitations and Challenges Encountered

1. **Domain-Specific Character Rules and Ambiguity:** A significant challenge arose from the specific rules governing Egyptian license plates. Officially, a set of 11 Arabic letters is excluded to prevent visual confusion with other characters. This set of unused letters is **(ت, ث, ح, خ, ذ, ز, ش, ض, ظ, غ, ك)**. However, in the real world, we encountered plates where owners had made modifications, such as adding a dot to the letter 'ر' to make it resemble 'ز' (which is an officially unused letter) to form a name like "زين" (Zayn). Our model, trained on official data, had to be robust enough to correctly classify the base letter ('ر') despite such "noise." This highlighted the need for a model that doesn't just recognize characters but also implicitly learns the underlying rules of valid plates.

Examples for cars that we have found in our data set were this car that had two images which they were 1974 and 1991 respectively:

And our model showed solid learning also not just for the character recognition but also implicitly learning the underlying rules of valid plates:

First image is 1974:

```
Loading YOLO model...
Detecting license plates...

image 1/1 D:\College (FCDS)\Graduation Project\Graduation Project ML Phase\EALPR Vechicles dataset\Vehicles\1974.jpg: 256x416 1
license_plate, 41.8ms
Speed: 1.0ms preprocess, 41.8ms inference, 1.2ms postprocess per image at shape (1, 3, 256, 416)
1 plate(s) cropped.

Plate #1
```


```
OCR Predictions: ['5', '5', '5', '5', 'noon', 'yaa', 'raa']
Arabic Plate: ٥٥٥٥رين
```

Second image is 1991:

```
Loading YOLO model...
Detecting license plates...

image 1/1 D:\College (FCDS)\Graduation Project\Graduation Project ML Phase\EALPR Vechicles dataset\Vehicles\1991.jpg: 288x416 1
license_plate, 42.5ms
Speed: 1.7ms preprocess, 42.5ms inference, 0.9ms postprocess per image at shape (1, 3, 288, 416)
1 plate(s) cropped.

Plate #1
```


```
OCR Predictions: ['5', '5', '5', '5', 'noon', 'yaa', 'raa']
Arabic Plate: ٥٥٥٥رين
```

2. **Image Quality from IoT Cameras:** The quality of images captured by the low-cost ESP32-CAM modules presented a major hurdle for the OCR model. Initial tests with very low-quality or poorly lit images led to inconsistent results, where the OCR model would sometimes confuse visually similar Arabic numerals (like '٢', '٣', and '٦') or even begin to overfit on noise. This necessitated an iterative development process, where we had to carefully curate the training data and test multiple versions of the OCR model to find one that balanced performance on both high-quality and low-quality images.
3. **API Dependency and Latency:** The strategic decision to use the cloud-based Roboflow OCR API, while accelerating development and working on the cloud, introduced an external dependency. The system requires a stable internet connection to function, and the round-trip API call adds a small amount of latency to the recognition process. This makes the current implementation best suited for environments with reliable network access.
4. **General Computer Vision Challenges:** Like any real-world vision system, performance can be affected by environmental factors that were difficult to fully represent in the training data. These include:
    - **Motion Blur:** From fast-moving vehicles.
    - **Extreme Angles:** Cars entering or exiting the frame at sharp angles.
    - **Poor Lighting:** Glare, shadows, or insufficient light at night.
    - **Occlusion:** Plates that are partially dirty, damaged, or obscured.

6.3.10.2 Future Enhancements

Based on the challenges identified, we propose the following directions for future work:

1. **Implement Advanced Data Augmentation:** To make the system more resilient to image quality issues and character ambiguity, we can implement more advanced data augmentation techniques during training. This would involve programmatically creating a more diverse dataset by adding realistic motion blur, varied lighting effects, and even synthetically generating character modifications (like adding dots or altering strokes) to teach the model to handle real-world variations.

2. **Optimize Models for Edge Deployment:** The YOLOv8 detection model could be converted to a more efficient format like ONNX or TensorFlow Lite. This would allow the entire detection pipeline to run on more powerful edge devices (like a Raspberry Pi or NVIDIA Jetson) located directly at the garage gate, reducing the reliance on a central server and minimizing latency.

3. **Expand Plate Type and Character Support:** The system could be extended to recognize different types of Egyptian plates, such as commercial, diplomatic, or temporary plates, which often have different color schemes and layouts. This would require expanding the dataset and potentially adding a classification step to the pipeline.

6.3.11 Conclusion

The machine learning component of the "El-Sayes" project has successfully delivered on its primary objective: to create a fast, accurate, and reliable License Plate Recognition system specifically tailored to the complexities of Egyptian license plates. Through the detailed implementation phases—from meticulous dataset preparation to the final pipeline integration—we have constructed a system that forms the technological backbone of the smart garage's automation features.

By strategically integrating a **fine-tuned YOLOv8 detector** for localization with a cloud-based **Roboflow OCR service** for character extraction, we achieved high performance while maintaining development efficiency. The key innovation remains our **custom post-processing algorithm**, which correctly interprets the right-to-left structure of the plates and translates raw data into meaningful information.

This LPR module stands as a working testament to how modern AI techniques, when combined with domain-specific knowledge, can effectively solve complex, real-world problems. It provides the "El-Sayes" system with the robust vehicle identification capabilities necessary to create a truly seamless and intelligent parking experience.

6.4 Supabase

Technology Used: Supabase (PostgreSQL), Supabase Storage

6.4.1 Purpose and Role of the Database

Purpose: Store, manage, and coordinate all operational data between the IoT hardware layer, reservation system, and admin interface.

The database plays a central role in enabling all components of the Smart Garage System to work cohesively. It serves as the unified data source between:

- **The IoT layer**: Devices like ESP32, IR sensors, ultrasonic sensors, servo motors, and ESP32-CAM
- **The frontend application**: Where users reserve parking slots and view availability
- **The admin dashboard**: Used for monitoring all system activities

Every action in the system — car entry, exit, reservation, payment, slot availability — is logged, updated, and retrieved via the database in real time using Supabase APIs and triggers.

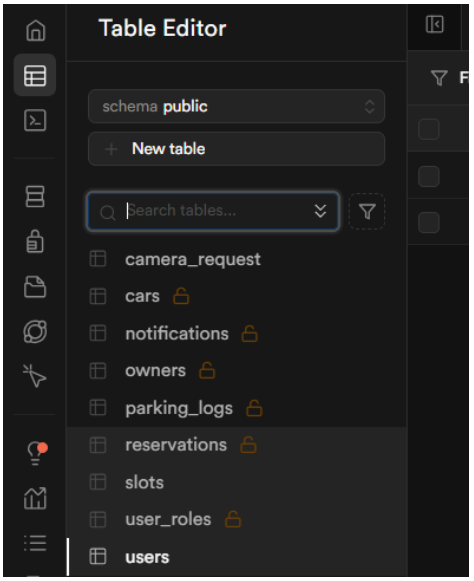
6.4.2 Why Supabase?

Supabase was chosen because:

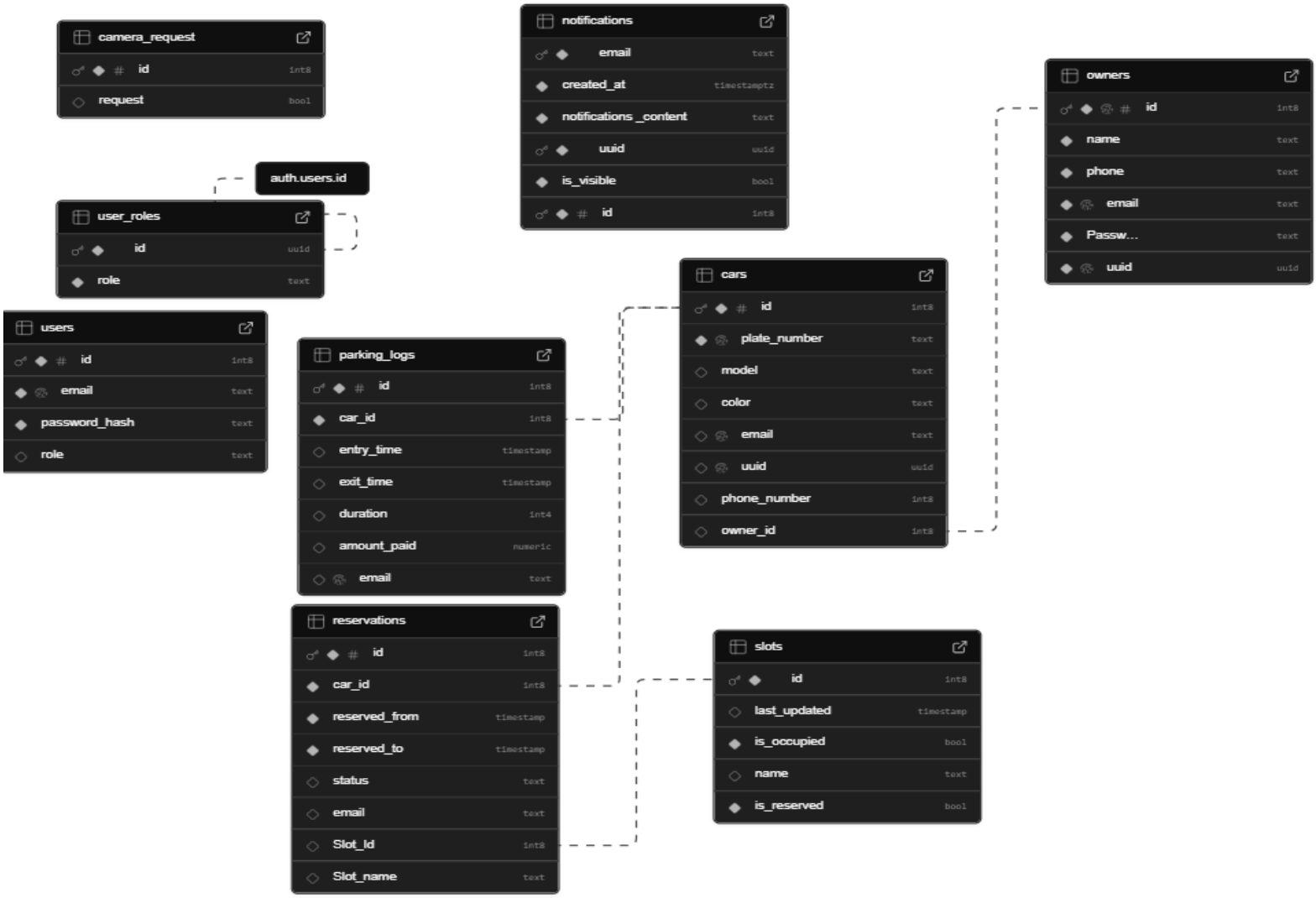
- It is built on **PostgreSQL**, which is powerful, scalable, and ACID-compliant.
- It provides **RESTful APIs** and **Realtime** features out-of-the-box.
- It integrates easily with IoT devices (via HTTP requests) and frontend apps (via SDKs).
- Offers **Supabase Storage** to manage images from the ESP32-CAM.
- Easy to use with triggers, foreign key constraints, and RLS (Row-Level Security) if needed.

6.4.3 Database Schema (Main Tables)

The ERD:



- Each car is linked to an owner through a foreign key (owner_id).
- A reservation connects a car to a specific slot, with booking times tracked.
- A parking_log is created for each actual car entry/exit and stores entry/exit timestamps.
- camera_request and notifications tables are used for IoT coordination and user communication.
- The slots table tracks both live occupancy and reservation status using boolean flags.



6.4.4 Key Tables

a. owners

Stores personal info for registered garage users (car owners).

```
id (PK) | name | phone | email | password | uuid
```

b. cars

Each car is associated with an owner.

```
id (PK) | plate_number (unique) | model | color | owner_id (FK → owners.id)
```

c. slots

Represents each parking slot in the garage.

```
id (PK) | name | is_occupied (bool) | is_reserved (bool) | last_updated (timestamp)
```

d. reservations

Handles future bookings for parking slots.

```
id (PK) | car_id (FK) | Slot_Id (FK → slots.id) | reserved_from | reserved_to |
status | email
```

e. parking_logs

Logs each parking session: entry, exit, duration, payment.

```
id (PK) | car_id (FK) | entry_time | exit_time | duration | amount_paid
```

f. camera_request

Acts as a flag. When true, the ESP32-CAM captures a photo.

```
id (PK) | request (boolean)
```

g. notifications

Holds system messages and alerts.

```
id (PK) | email | message | is_visible | created_at
```

h. users

Stores admin/operator accounts for system management.

```
id (PK) | email | password_hash | role (admin/operator)
```

Relationships Between Tables

Table	Relationship	Description
cars	FK → owners.id	Each car belongs to an owner
reservations	FK → cars.id, slots.id	Reservation ties car with a specific slot
parking_logs	FK → cars.id	Each parking log is for a single car
slots	1:N with reservations	A slot can be reserved multiple times (non-overlapping)
users	Authentication layer	Access control via Supabase Auth

6.4.5 Database Triggers

Two PostgreSQL triggers were implemented to automate calculations:

a. calculate_parking_duration

Automatically calculates the duration between entry_time and exit_time.

```
duration := EXTRACT(EPOCH FROM (NEW.exit_time - NEW.entry_time)) / 3600;
```

b. calculate_parking_amount

Automatically computes amount_paid based on the duration and hourly rate.

```
amount_paid := ROUND(NEW.duration * 10.0, 2); -- 10 units per hour
```

These are applied via:

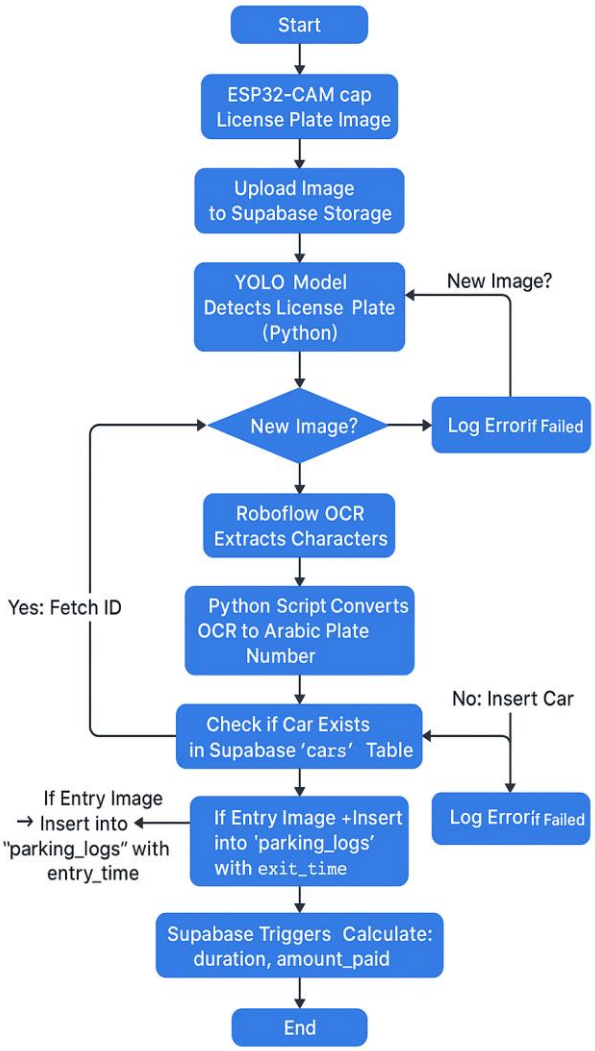
```
CREATE TRIGGER trg_calculate_duration
BEFORE UPDATE ON parking_logs
FOR EACH ROW
EXECUTE FUNCTION calculate_parking_duration();

CREATE TRIGGER trg_calculate_amount
BEFORE UPDATE ON parking_logs
FOR EACH ROW
EXECUTE FUNCTION calculate_parking_amount();
```

Integration with IoT and Applications

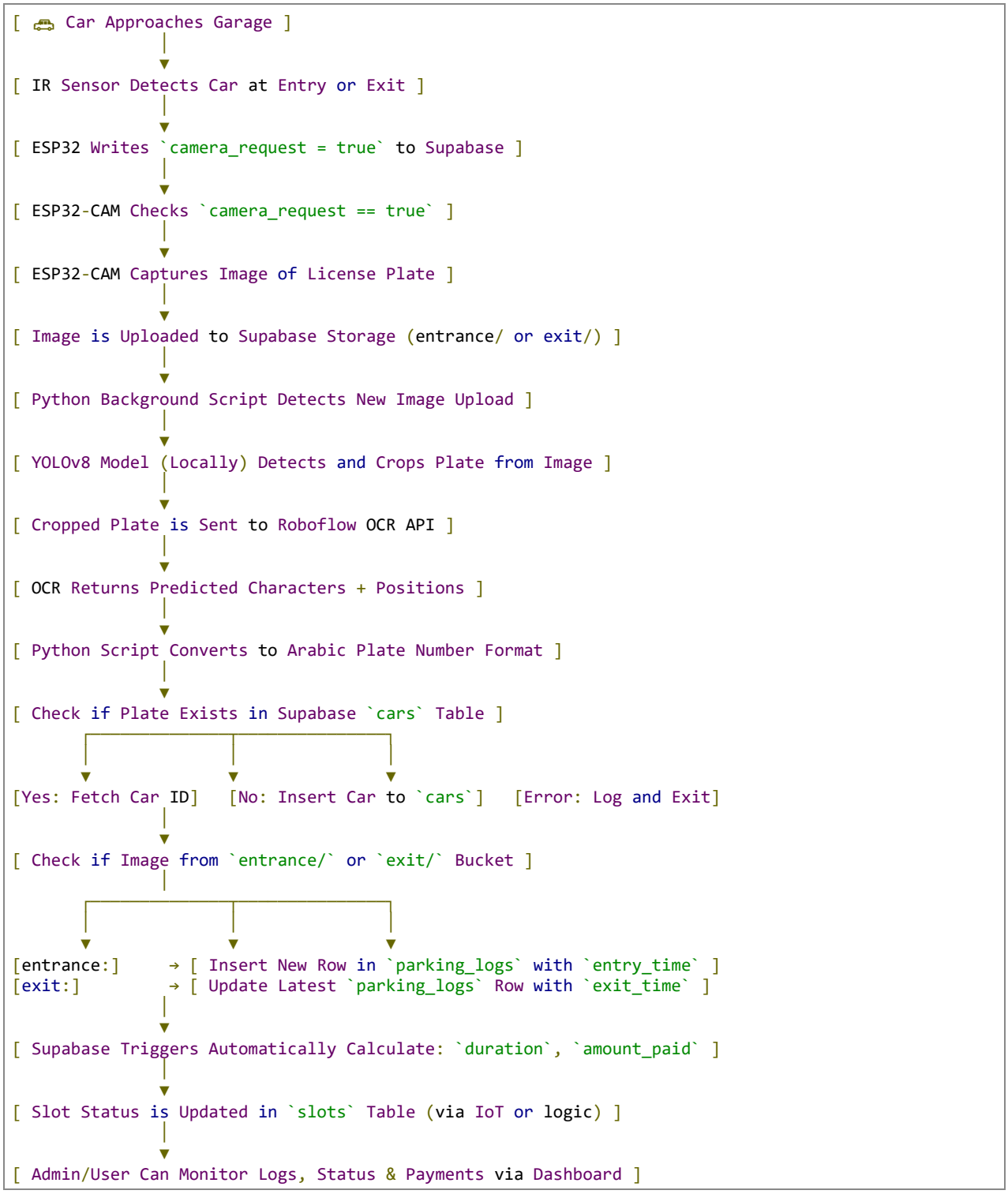
Device/Module	How it interacts with DB
ESP32	Updates slots.is_occupied, sends camera_request
ESP32-CAM	Monitors camera_request → uploads image to Supabase Storage
Python/Streamlit	Sends reservation data, displays slot status
Supabase Functions	Used for Edge Functions (image OCR integration)

6.4.6 System Workflow Steps

1. **Car arrives at the garage**
    1. IR sensor detects presence
    2. ESP32 is triggered
2. **ESP32 sends a signal by setting camera_request = true**
    - This flag is written to Supabase (camera_request table)
3. **A background service (Python script)** constantly checks for new images in entrance/ or exit/ buckets in Supabase Storage.
4. **Once a new image is uploaded by the ESP32-CAM**, the script:
    1. Downloads the image
    2. Uses a **YOLO model (run locally)** to detect and crop license plates
    3. Sends cropped plates to **Roboflow OCR API**
    4. Converts OCR result into **Arabic license number**



5. **Script interacts with Supabase**:
    1. If the car is new: insert into cars
    2. If a reservation exists: create entry in parking_logs with entry_time
    3. On exit: update exit_time for that car's latest log
6. **Triggers in Supabase** automatically calculate:
    1. duration
    2. amount_paid


[🚗 Car Approaches Garage]
[IR Sensor Detects Car at Entry or Exit]
[ESP32 Writes `camera_request = true` to Supabase]
[ESP32-CAM Checks `camera_request == true`]
[ESP32-CAM Captures Image of License Plate]
[Image is Uploaded to Supabase Storage (entrance/ or exit/)]
[Python Background Script Detects New Image Upload]
[YOLOv8 Model (Locally) Detects and Crops Plate from Image]
[Cropped Plate is Sent to Roboflow OCR API]
[OCR Returns Predicted Characters + Positions]
[Python Script Converts to Arabic Plate Number Format]
[Check if Plate Exists in Supabase `cars` Table]
[Yes: Fetch Car ID]
[No: Insert Car to `cars`]
[Error: Log and Exit]
[Check if Image from `entrance/` or `exit/` Bucket]
[entrance:] → [Insert New Row in `parking_logs` with `entry_time`]
[exit:] → [Update Latest `parking_logs` Row with `exit_time`]
[Supabase Triggers Automatically Calculate: `duration`, `amount_paid`]
[Slot Status is Updated in `slots` Table (via IoT or logic)]
[Admin/User Can Monitor Logs, Status & Payments via Dashboard]

Full Workflow

1. **Vehicle Detection at Garage Entry/Exit**
    - The system uses **IR sensors** installed at the entrance and exit gates to detect the presence of a vehicle.
2. **Trigger Camera Capture**
    - Upon detection, the **ESP32 microcontroller** sets a flag camera_request = true in the Supabase camera_request table.
    - This acts as a signal for the **ESP32-CAM** to capture a photo of the vehicle's license plate.
3. **License Plate Image Upload**
    - The captured image is uploaded to **Supabase Storage**, specifically into either the entrance/ or exit/ bucket based on the gate.
4. **Python Background Service (Local Model Integration)**
    - A locally-running **Python script** continuously monitors the storage buckets.
    - When a new image is detected, the following steps are performed:
        - The image is downloaded locally.
        - A **YOLOv8 model** detects and crops the license plate region.
        - Cropped plates are passed to the **Roboflow OCR API** for character recognition.
5. **Arabic Plate Extraction**
    - The OCR predictions are parsed and converted into the Arabic format using a custom mapping.
    - Example: ['3', '3', 'alif', 'meem'] → ٣٣أم

6. **Car Record Handling in Supabase**
    - The system checks if the plate number exists in the cars table.
        - If found: retrieves the car_id.
        - If not: inserts a new record with an owner_id = 1 (default for unknown users).
7. **Insert or Update Parking Log**
    - If the image was from the entrance/ bucket:
        - A new row is inserted into the parking_logs table with entry_time = NOW().
    - If from the exit/ bucket:
        - The latest parking log with exit_time = NULL for this car is updated with the current exit_time.
8. **Automatic Duration and Payment Calculation**
    - Two PostgreSQL triggers are invoked:
        - calculate_parking_duration: Computes the number of hours parked.
        - calculate_parking_amount: Multiplies duration by a rate (e.g., 10 EGP/hour) to calculate amount_paid.
9. **Slot Status Synchronization (Optional via IoT)**
    - Although reservation logic tracks reserved slots, **IR sensors on each slot** also update the is_occupied status in real-time.
    - The user interface (e.g., Streamlit or Flutter app) reflects both **availability** and **reservation state**.

6.5 UI / UX Design

6.5.1 Introduction to UI/UX Design

- UI/UX design plays a critical role in bridging the gap between users and technology. While UI focuses on creating visually appealing and functional interfaces, UX ensures a seamless and intuitive user journey. In our graduation project, the goal was to design an application that catered to user needs while maintaining a clean and engaging interface. This section outlines the principles of UI/UX design and its significance in achieving the project's objectives.

6.5.2 Importance of UI/UX in Modern Applications

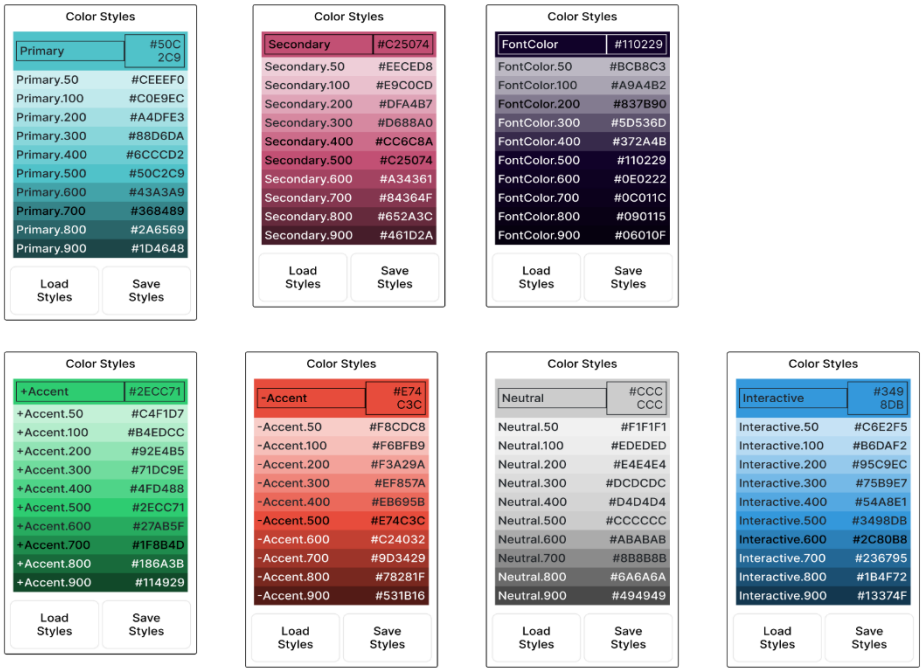
- The importance of UI/UX design cannot be overstated in the development of modern applications. As technology evolves, users expect intuitive and visually engaging experiences. A well-designed interface not only enhances usability but also fosters user satisfaction and retention. In our project, UI/UX design played a pivotal role in addressing user needs, ensuring that the application was accessible, functional, and aesthetically pleasing. By prioritizing UI/UX, we aimed to deliver a product that meets both user expectations and industry standards.

6.5.3 Tools and Technologies Used

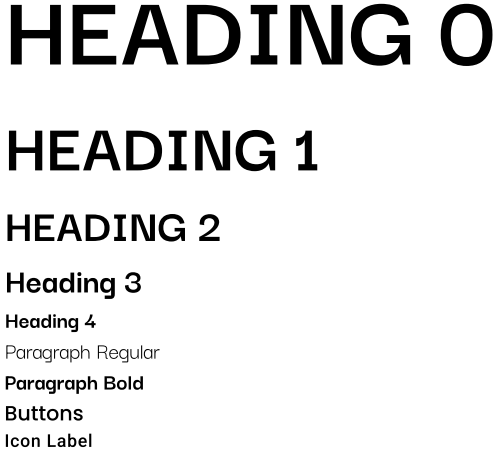
- For the design phase of our graduation project, Figma was the primary tool used due to its versatility and collaborative capabilities. Figma allowed me to design wireframes, create high-fidelity mockups, and prototype user interactions efficiently. Its real-time collaboration features made it easier to gather feedback and make quick iterations. Additionally, Figma's library of plugins streamlined the process of adding icons and illustrations, ensuring a polished final design. The tool's ability to hand off designs to developers with accurate specifications further facilitated the transition from design to development.

6.5.4 Establishing Application Identity

- Before starting designing the mobile and web app, we first established a unique identity for our application including:
  - Color Palette: The chosen color palette reflects a modern and professional tone, enhancing the application's visual appeal while ensuring readability. The palette maintains consistency across all pages for a cohesive user experience.



  - Typography, insuring a simple and professional looking fonts, The fonts used in the design were carefully selected for readability and elegance.

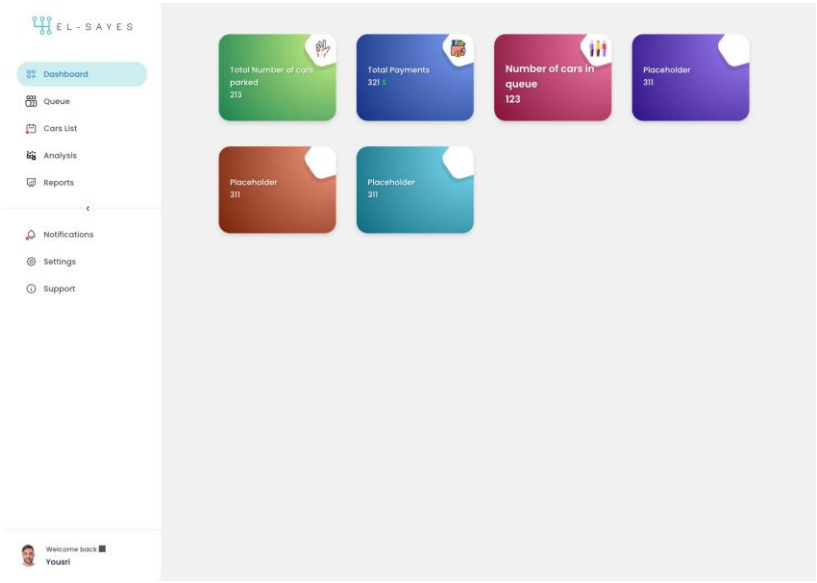


- Logo (Brand): The logo combines simplicity and symbolism, reflecting the purpose of the application. The gear-like element visually represents the garage and operations, while the modern typography of "EL-SAYES" emphasizes clarity and professionalism. The logo aligns with the overall design theme and serves as a recognizable element of the brand.

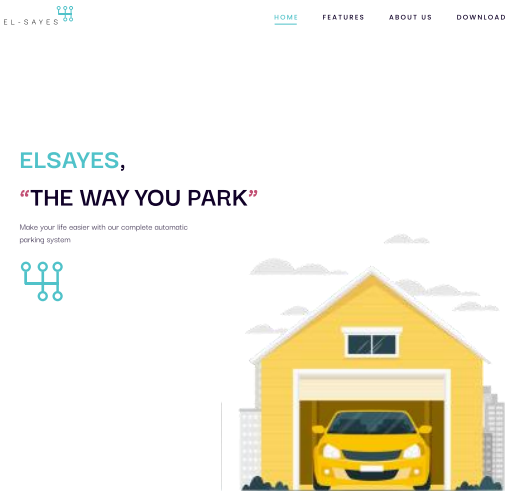
EL-SAYES EL-SAYES

6.5.5 Design Process for Web Application

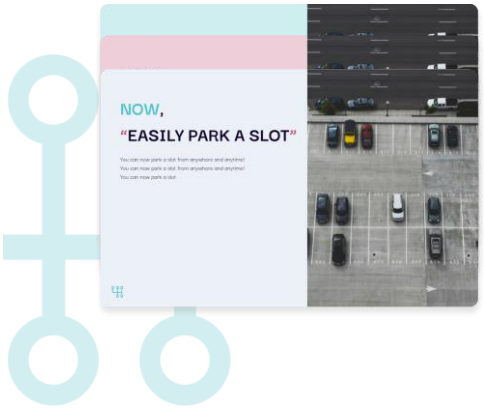
- After establishing the identity for our application, we designed an intuitive dashboard tailored for the admin to effectively monitor and manage all operations within the garage. The dashboard provides a comprehensive overview, including key analytics and management tools. Admins can access a dedicated dashboard tab to view critical metrics such as the total number of cars currently parked, a detailed history of parking activity, total payments received, the number of cars in the queue, and more. This centralized hub ensures streamlined management and real-time insights, empowering the admin to make informed decisions efficiently.



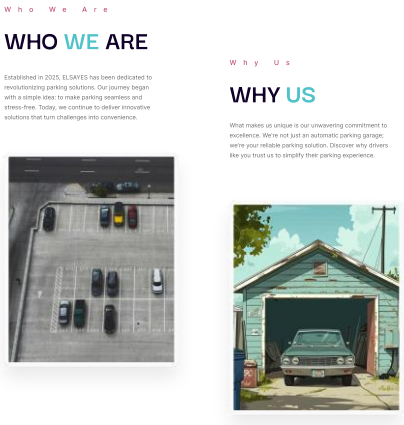
- For users, we designed an engaging landing page to provide a welcoming experience and encourage them to download our application. The landing page features a prominent header with intuitive navigation, allowing users to explore the site effortlessly. This design ensures a seamless introduction to our application and its features while guiding users toward the download process with clear calls-to-action.



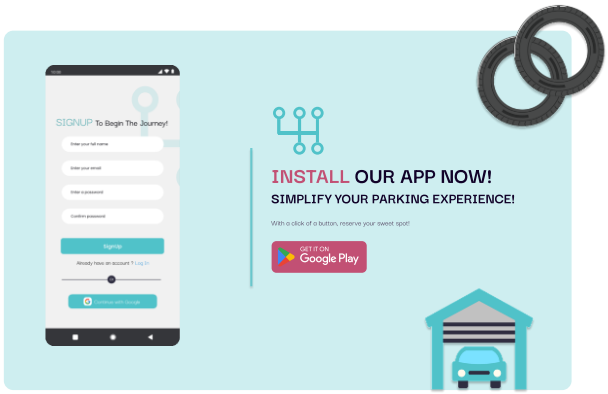
- We also designed a features section to showcase the key offerings of our application. This section provides users with a clear and concise overview of the benefits and functionalities available, highlighting what sets our solution apart. By presenting these features in an organized and visually appealing manner, we aim to engage users and build their interest in exploring the application further.



- An About Us section to introduce ourselves to the users and provide insight into the purpose and vision behind our application. This section highlights our mission, the challenges we aim to solve, and the dedication of our team in creating a solution tailored to meet user needs. By sharing our story and values, we aim to build trust and establish a meaningful connection with our audience.



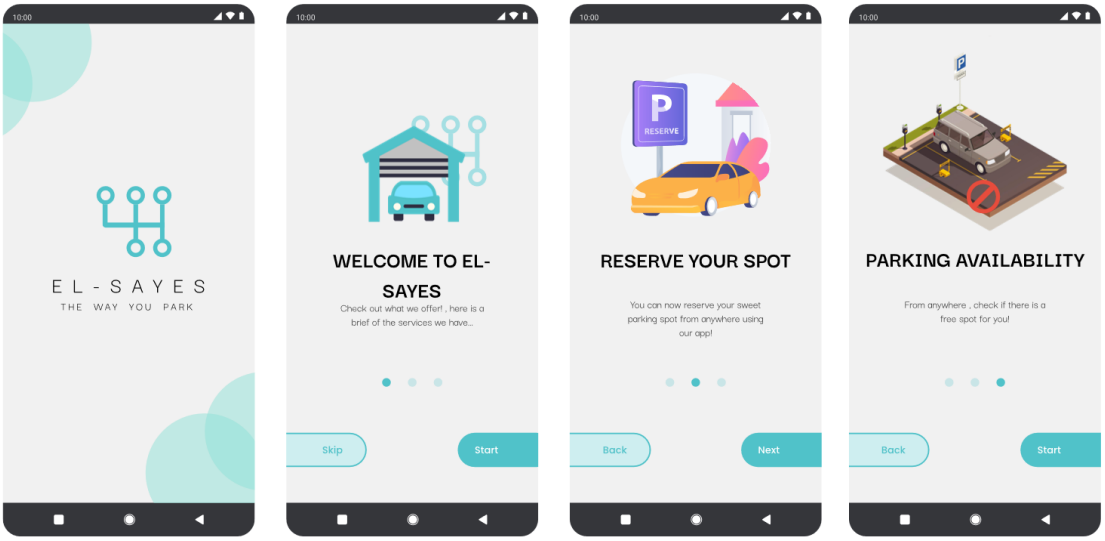
- We also designed a Download Section to guide users seamlessly when they want to download our application. This section features clear and prominent call-to-action buttons, allowing users to download the app for their preferred platform, such as iOS or Android. It includes a brief description of the app's key benefits and functionalities, encouraging users to take action. Additionally, we integrated visually appealing icons and direct links to the respective app stores, ensuring a straightforward and hassle-free experience for the user.



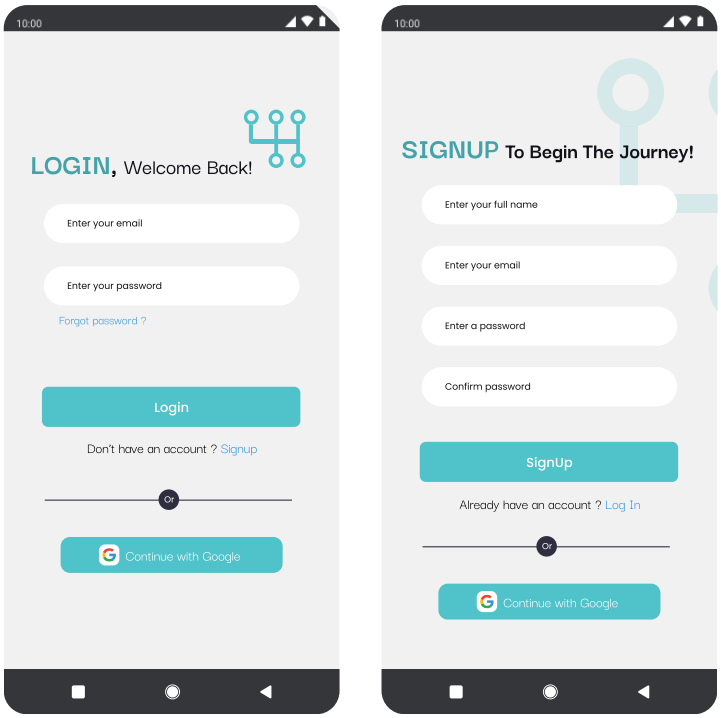
- The footer features a minimalist design with the "EL-SAYES" logo and name on the left, reinforcing branding. On the right, it provides links to Instagram, Facebook, and Twitter, encouraging users to connect with us for updates and engagement. Its clean layout ensures simplicity and usability.

6.5.6 Design Process for Mobile Application

- Onboarding: We designed a series of onboarding screens to guide users as they open our app for the first time. These screens introduce the app's key features and benefits, ensuring users understand its functionality and value. By providing a visually engaging and user-friendly onboarding experience, we aim to familiarize users quickly and make their transition into the app seamless.



- Registration: We designed intuitive Login and Signup screens for a seamless user experience. The Login screen allows users to sign in with email/password or Google, with a "Forgot password?" option for recovery. The Signup screen enables new users to create an account by entering their details or signing up via Google. Both screens feature a clean, modern design, ensuring simplicity and accessibility.



6.6 Flutter Development

6.6.1 Introduction

The Smart Garage System is a mobile application designed to provide users with a seamless and efficient parking experience. It allows users to find, reserve, and manage parking spots using real-time data. The app is built using FlutterFlow, a powerful visual development tool for building cross-platform applications.

6.6.2 Why Flutter?

The development of the Smart Garage System application initially began using Flutter because of its powerful features for cross-platform mobile app development:

- **Single Codebase:** Flutter enables developers to write a single codebase that works seamlessly on both Android and iOS, significantly reducing development time and effort.
- **Beautiful and Responsive UI:** Flutter's rich widget library allows the creation of visually appealing and highly responsive UIs.
- **High Performance:** Flutter apps are compiled to native ARM code, ensuring high performance and smooth animations.
- **Open Source and Strong Community:** As an open-source framework backed by Google, Flutter offers extensive community support and a wealth of resources for developers.
- **Integration with Backend Services:** Flutter can easily integrate with backend solutions like Firebase for real-time databases, authentication, and cloud functions.

6.6.3 Why FlutterFlow?

Initially, the app development began using Flutter. However, during the development process, the team discovered FlutterFlow as a potential tool to accelerate and simplify the project. After evaluating its capabilities, the decision was made to transition to FlutterFlow. Below are the reasons for adopting FlutterFlow and the features it provides:

We chose FlutterFlow for this project due to its unique features and benefits, particularly for simplifying and accelerating mobile application development:

1. Visual Development Platform
FlutterFlow provides a drag-and-drop interface, allowing for rapid prototyping and development without requiring extensive coding knowledge. This makes it ideal for building complex UI designs quickly.

2. Based on Flutter
FlutterFlow leverages the Flutter framework, enabling the creation of high-performance applications with beautiful and responsive UIs. It combines the power of Flutter with an intuitive visual editor.

3. Cross-Platform Development
Just like Flutter, FlutterFlow allows developers to create a single codebase that works seamlessly on both Android and iOS, reducing development time and costs.

4. Integration with Backend Services
FlutterFlow offers built-in support for integrating with backend services such as Firebase, making it easy to set up real-time databases, authentication, and cloud functions.

5. Custom Code Flexibility
While FlutterFlow simplifies development with its visual tools, it also allows developers to add custom code for advanced functionalities, providing the best of both worlds.

6. Faster Iterations
With FlutterFlow, changes to the application can be previewed instantly, speeding up the design and testing process.

7. Exportable Code
FlutterFlow generates clean Flutter code, which can be exported and further customized if needed, ensuring flexibility and scalability.

6.6.4 Application Features

User-Centric Features

- **User Authentication**
  - Login/Sign up using email, phone number, or social media.
  - Secure authentication with optional two-factor authentication.

- **Real-Time Parking Availability**
  - Interactive map view of available parking spots.
  - Filters for location, price, or type of spot (e.g., covered, EV charging).

- **Parking Spot Reservation**
  - Reserve a spot for a specific time and date.
  - Modify or cancel reservations.

- **Payment Integration**
  - Secure payment gateways for credit cards, e-wallets, and subscriptions.
  - Transaction history for easy reference.

- **Navigation and Directions**
  - GPS navigation to the parking garage.
  - Indoor navigation for multi-level garages.

- **Notifications and Alerts**
  - Booking confirmations and reminders.
  - Alerts for expiring parking time or special offers.

- **License Plate Recognition**
  - Seamless entry and exit via automated license plate recognition.

6.6.5 Admin Features

- **Dashboard Overview**
  - Real-time monitoring of parking spots.
  - Insights into occupancy and revenue.

- **User and Spot Management**
  - Manage user accounts and vehicle registrations.
  - Add, update, or deactivate parking spots.

- **Reports and Analytics**
    - Generate reports on usage patterns and revenue.
    - Predict peak times using machine learning insights.

Technology Stack

Frontend: FlutterFlow (Dart Language via Flutter)
Backend: Supabase for real-time database and authentication
Payment Integration: Stripe
Navigation: Google Maps API for GPS and indoor navigation

6.6.6 Development Roadmap

6.6.6.1. Design Phase

- **Overview**: Focuses on creating the visual structure and layout of the application. The primary goal is to ensure an intuitive user interface (UI) that is both visually appealing and user-friendly.
- **Key Tasks**:
    - Wireframing and prototyping the app's layout using FlutterFlow's drag-and-drop interface.
    - Selecting the appropriate color scheme, typography, and design elements that align with the app's branding.
    - Adding static UI components, such as buttons, navigation bars, and placeholders for dynamic content.
- **Outcome**: By the end of this phase, the app's layout and user interface will be complete, but the buttons and other interactive elements will not yet have functionality.

6.6.6.2. Adding Actions to Buttons

- **Overview**: This phase involves making the static UI interactive by assigning actions and behaviors to buttons and other UI elements.

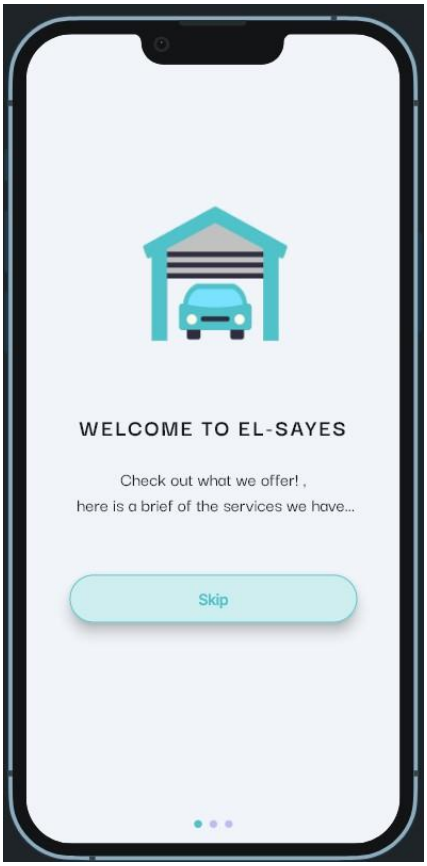
- **Key Tasks**:
    - Linking buttons to their corresponding pages (e.g., navigating to the reservation page when the "Reserve" button is clicked).
    - Adding functionalities like form submission, user authentication, and real-time data fetching using Firebase.
    - Implementing error handling for invalid user inputs or system errors.
    - Testing the interactions to ensure they work as intended and provide a smooth user experience.
- **Outcome**: By the end of this phase, all buttons and interactive elements will perform their intended actions, transforming the app from a static prototype to a functional application.

6.6.6.3. Splash Screen

- **Purpose**: Provides a visual entry point and gives the user a welcoming experience while the app loads essential resources.
- **Key Features**:
    - **Design Elements**:
        - A centered, high-quality logo of the app.
        - The app's name displayed beneath the logo.
        - Optional tagline or slogan (e.g., "EL-SAYES") to communicate the app's purpose.
        - A clean and minimalistic background (e.g., solid color or gradient).
    - **Functionality**:
        - Displayed for 2–3 seconds while the app preloads resources.
        - Includes a timer or event-based transition to navigate to the Welcome Screen.
- **Implementation in FlutterFlow**:
    - Configured using the initial screen feature.
    - Timer Action: Set a delay of 2–3 seconds.
    - Redirect Logic: Conditional navigation based on user authentication status.

6.6.6.4. Welcome Screen

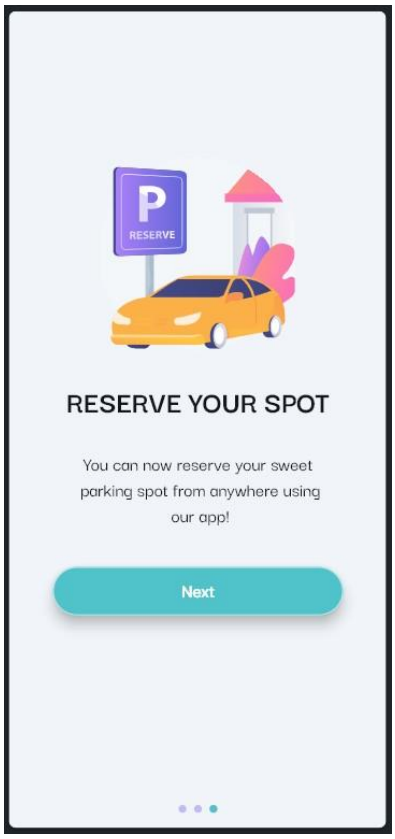
- **Purpose**: Introduces users to the app with a user-friendly onboarding experience. This screen serves as the initial entry point into the feature tour, setting a welcoming tone.



- **Key Features**:
  - An illustrative icon of a car within a garage, visually representing the app's domain.
  - A prominent welcome message and a short description of the app's offerings.
  - Navigation buttons, including a "Skip" button, to allow users to bypass the onboarding process.
  - A progress indicator to show the user's position in the onboarding process.
- **Design Elements**:
  - **Background**: The screen utilizes a clean, light-themed background for a modern and inviting aesthetic.
  - **Typography**: The text is presented in a simple, dark font for maximum readability.
  - **Buttons**: A prominent "Skip" button is styled with a soft teal color and rounded corners to provide a clear call to action.
- **Functionality**:
  - The navigation action for the "Skip" button is configured to take the user directly to the authentication flow (Login/Sign-up screen).
  - Users can swipe left to proceed to the next step in the onboarding tour.
  - The progress indicator updates dynamically for each onboarding step.
- **Implementation in FlutterFlow:**
  - This screen is implemented as the first page within a `PageView` widget.
  - Navigation actions are configured for the "Skip" button.
  - The progress indicator is dynamically linked to the current onboarding step to provide visual feedback to the user.

6.6.6.5. Reserve Your Spot Screen

Purpose: The screen introduces users to one of the app's core features: the ability to reserve parking spots remotely. It is part of the onboarding process designed to familiarize users with the app's functionality.



Key Features

- **Feature Description**: Highlights the app's key feature: parking spot reservation.
  - Includes a short description: *"You can now reserve your sweet parking spot from anywhere using our app."*
- **Visual Representation**:
  - Contains a vibrant illustration of a car and a reservation sign to visually communicate the feature.
- **Navigation Buttons**:
  - A "Next" button proceeds to the next onboarding screen.
- **Progress Indicator**:

  Displays the user's progress in the onboarding flow

Design Elements

- **Background**:
  - Maintains a light theme for visual continuity with the subsequent onboarding screen.
- **Font Style**:
  - Features easy-to-read white typography that aligns with the app's overall design language.
- **Buttons**:
  - A clearly labeled "Next" button for intuitive navigation.

Functionality

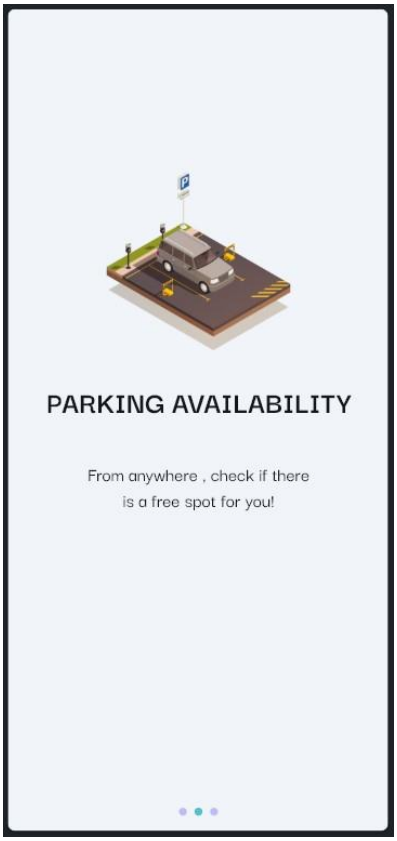
- The progress indicator updates dynamically to reflect the user’s current position in the onboarding sequence.

Implementation in FlutterFlow

1. **Progress Indicator**:
    - The progress indicator is dynamically linked to the current onboarding step.
2. **Responsive Design**:
    - The layout adjusts seamlessly across different screen sizes to maintain usability.

6.6.6.6. Parking Availability Screen

Purpose: This screen introduces users to another core feature of the app: checking for free parking spots in real time. It helps users understand the convenience of remotely monitoring parking availability.



Key Features

1. **Feature Description**:
    - Highlights the parking availability feature.
    - Includes a short description:
      "From anywhere, check if there's any free spot for you."
2. **Visual Representation**:
    - Includes an isometric illustration emphasizing the feature visually.
3. **Navigation Buttons**:
    - A "Next" button (or equivalent action) proceeds to the main app screen.
4. **Progress Indicator**:
    - Displays the user's progress in the onboarding flow, indicating the final step.

Design Elements

- **Background**:
    - Maintains a consistent light theme with the previous onboarding screen for visual harmony.
- **Typography**:
    - Features simple and clear text for the feature description.

Functionality

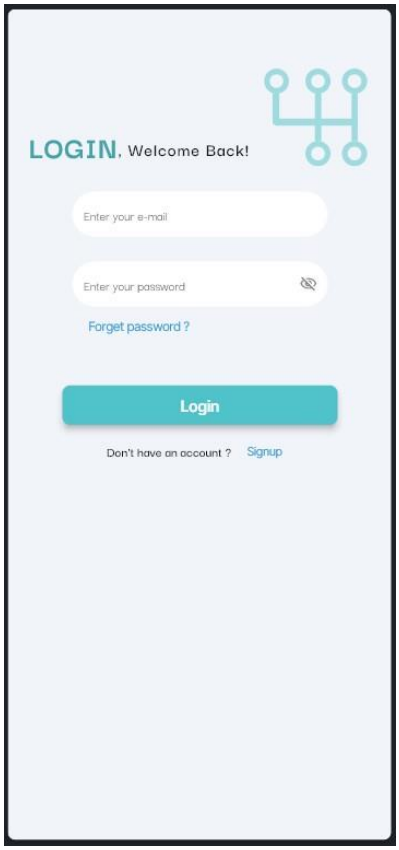
- The progress indicator updates dynamically to reflect the user's final position in the onboarding sequence

Implementation in Flutter Flow

1. **Navigation Setup**:
   - Navigation is configured to enable a smooth transition from the onboarding flow to the main authentication screens.
2. **Progress Indicator**:
   - The progress indicator is dynamically linked to align with the current onboarding step.
3. **Responsive Design**:
   - Ensures proper scaling and alignment of the illustration across different device sizes to maintain a visually appealing layout.

6.6.6.7. Login Screen

Purpose: The Login Screen provides a secure entry point for users to access their accounts. It is designed for returning users who have already created an account.



Key Features

1. **Email and Password Input Fields**:
   - Users can enter their registered email and password to log in.
2. **Forgot Password Option**:
   - Provides a link to recover or reset a forgotten password, ensuring users can regain access to their accounts.
3. **Sign-Up Redirect**:
   - Includes a link for users who do not have an account, directing them to the **Sign-Up Screen** to create one.
4. **Login Button**:
   - Validates user credentials and logs them into the app.

Design Elements

- **Illustration**:
    - The app's gearshift logo is placed in the top-right corner for brand consistency.
- **Buttons**:
    - A prominent "Login" button is styled with a solid teal color for a clear call to action.
- **Color Scheme**:
    - The screen uses a light background with teal accents for the logo, button, and links, creating a clean and modern look
- **Typography**:
    - Clean and simple text is used for labels and links to ensure readability.

Functionality

1. **Forgot Password**:
    - This link initiates **Supabase's** built-in password recovery feature, which typically sends a reset link to the user's email.
2. **Sign-Up Redirect**:
    - Navigates users to the Sign-Up Screen.
3. **Login Button**:
    - On click, the app validates the entered credentials using **Supabase Authentication**. If successful, the user is navigated to the Home Page

Implementation in FlutterFlow

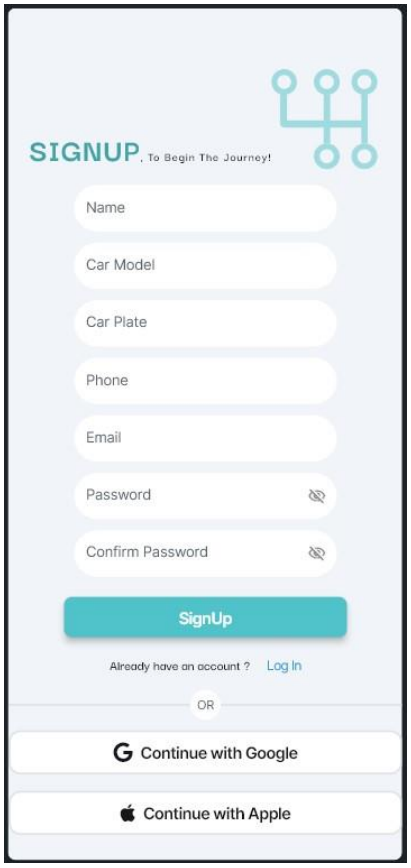
1. **Navigation Setup**:
    - Navigation is configured for the "Forgot Password" link to trigger the Supabase password reset action and for the "Signup" link to navigate to the Sign-up Screen.
2. **Form Validation**:
    - Validation rules are set up for the email and password fields to ensure correctness before submission.
3. **Login Button Integration**:
    - The "Login" button is linked to the **Supabase** 'User Login' action, which handles authentication.

6.6.6.8. Sign-Up Screen

Purpose: The Sign-Up Screen allows new users to create an account, enabling access to the app's features.

Key Features

1. **Input Fields**:
    - **Full Name**: Allows users to provide their name for personalization.
    - **Car Model & Car Plate**: Captures vehicle details essential for the parking service.
    - **Phone Number** Collects the user's contact number.
    - **Email**: Captures the user's email for authentication and communication.
    - **Password and Confirm Password**: Ensures secure account creation by requiring matching passwords.
2. **Sign-Up Button**:
    - Validates the input fields and registers the user.
3. **Social Media Integration**:
    - Offers options to sign up using Google or Apple for added convenience.
4. **Login Redirect**:
    - Includes a link for users who already have an account, guiding them to the **Login Screen**.



Design Elements

- **Illustration**:
    - A gearshift icon or similar graphic symbolic of the app's theme.
- **Input Fields**:
    - Styled with rounded corners and a light background for a modern aesthetic.
- **Buttons**:
    - A prominent "SignUp" button and clearly defined options for Google and Apple sign-ups enhance ease of use.

Functionality

1. **Sign-Up Button**:
    - The button's action flow performs two tasks in sequence:
        1. It creates a new user in **Supabase Auth** with the provided email and password.
        2. It then inserts the user's name, car model, car plate, and phone number as a new row in the **Supabase** database.
        3. Upon success, the user is navigated to the Home Screen.
2. **Social Media Sign-Up**:
    - Connects to third-party APIs (e.g., Google or Apple) for quick and secure registration.
3. **Login Redirect**:
    - Redirects users to the **Login Screen** if they already have an account.
4. **Form Validation**:
    - Ensures valid email formatting and matching passwords.

Implementation in Flutter Flow

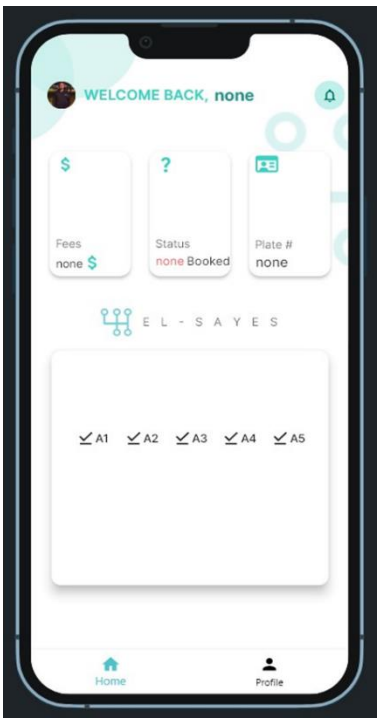
1. **Backend Authentication**:
    - The "SignUp" button's action flow is configured to first call the **Supabase** 'Create Account' action, followed by a **Supabase** 'Insert Row' action. The social login buttons are linked to their respective **Supabase** authentication actions.
2. **Navigation Setup**:
    - Navigation is configured for the "Log In" redirect link and for the successful completion of the sign-up flow, which directs the user to the Home Page
3. **Form Validation**:

Real-time validation is set up for user inputs, such as checking email formatting and verifying that passwords match.

6.6.6.9. Parking System Interface (Home Page)

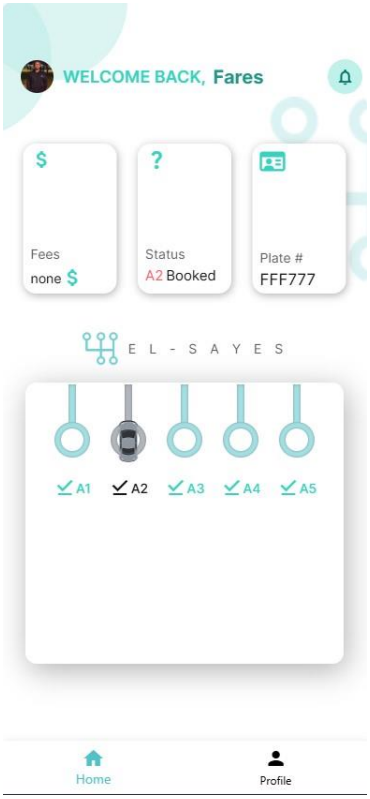
Purpose: The Home Page serves as the central hub for users, providing access to essential information and navigation options. It allows users to view parking details, check parking availability in real-time, and interact with a parking map to reserve a spot.

Key Features



1. **User Information Header**:
    - Displays a personalized welcome message (e.g., "WELCOME BACK, [Name]") and the user's profile picture:
    - Includes a notification icon for alerts and updates.
2. **Interactive Parking Map:**
    - A visual representation of available and reserved parking spots.
    - Users can click on a parking spot to initiate the reservation process**.**
3. **Navigation Bar**:
    - Provides quick access to other app functionalities, including "Home" and "Profile".

Design Elements



- **Header Section**:
    - Features a calming teal background with abstract circular patterns.
    - A rounded container holds the user's profile picture, with bold, white typography for the welcome message.
- **Dashboard Cards**:
    - The cards are designed with a clean, white background, rounded corners, and subtle shadows for a modern, layered look. Each card includes a simple icon for quick visual identification.
- **Parking Map Section**:
    - A central white container holds the list of parking spots. The app's logo ("EL-SAYES") is displayed above it.
    - Each parking spot's status is visually indicated by a dynamic image: a teal checkmark icon for available spots and a greyed-out icon for reserved spots.
- **Navigation Bar**:
    - A minimalist design with icons and labels provides clear and intuitive navigation

Functionality

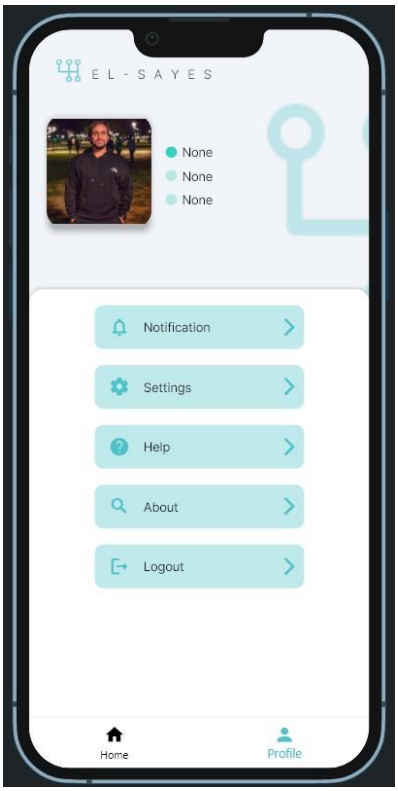
1. **Dynamic User Information**:
    - Displays data such as the user's name, profile picture, fees, booking status, and car plate, all retrieved dynamically from the **Supabase** backend.
2. **Interactive Parking Map**:
    - The availability of each parking spot is determined by real-time data from a **Supabase** table.
    - When a user clicks on a parking spot:
        1. If the spot is **available**, the user is navigated to the Reservation Screen to proceed with the booking.
        2. If the spot is **reserved**, a pop-up alert dialog appears with the message "Not available, this slot already occupied" to inform the user.
3. **Navigation Bar**:
    - Provides seamless transitions between the **Home Page** and **Profile Page**.

Implementation in FlutterFlow

1. **Backend Integration:**
    - The page is connected to **Supabase**. On page load, it queries the necessary tables to fetch user profile data and the current status of all parking spots.
2. **Conditional Display**:
    - The images representing each parking spot use conditional logic. Based on the 'status' field for a spot from the **Supabase** query, the app displays one of two images from a web URL: a teal image for "available" or a grey image for "reserved".
3. **Navigation Bar**:
    - The icons are configured with Maps to actions, linking them to the appropriate screens in the app.
4. **Interactive Actions**:
    - Each parking spot element is configured with a conditional `onTap` action. The condition checks the spot's status from the backend.
    - If available, the action is `Maps to` the Reservation Screen.
    - If reserved, the action is `Alert Dialog` with the specified message.

6.6.6.10. Profile Page Design

Purpose: The Profile Page provides users with an interface to view their personal details and manage their account and app settings. It serves as a central point for user management, offering access to notifications, settings, and the option to log out securely.



Key Features:

1. **Header Section:**
    - Displays the user's profile picture, name, email, and phone number.
    - This information is fetched dynamically from the **Supabase** database, with "None" displayed as a default value if the data is not available.
2. **App Menu Section:**
    - Organized into a list of options for easy navigation and usability:
        - **Settings:** Navigates to a page where the user can edit profile details or app settings.
        - **Notification:** Allows the user to manage notification preferences.
        - **Help:** Provides access to a help center or support resources.
        - **About:** Links to a screen with information about the application.
        - **Log Out Button:** Provides an option to securely log out of the account.

Design Elements:

1. **Header Section:**
    - Uses a centered layout with a large, rounded profile picture at the top left.
    - The background features the app's signature teal color and abstract logo elements, ensuring consistency with the app's theme.
2. **App Menu Section:**
    - The menu options are displayed as a vertical list of buttons within a clean, white container.
    - Each button is styled with a light teal background, rounded corners, an icon, and clear text for intuitive interaction.

3. **Log Out Button:**

    - Styled consistently with the other menu buttons for a uniform look while being positioned at the bottom of the list for easy access

Functionality:

1. **Header Section:**
    - Dynamically fetches and displays the user's profile picture, name, and email address from the backend.
2. **App Settings Section:**
    - The language and currency settings can open modal dialogs or dropdowns for user selection.
    - Notification settings are managed using toggle switches.
3. **Log Out Button:**
    - On click, it logs the user out and redirects them to the login screen.

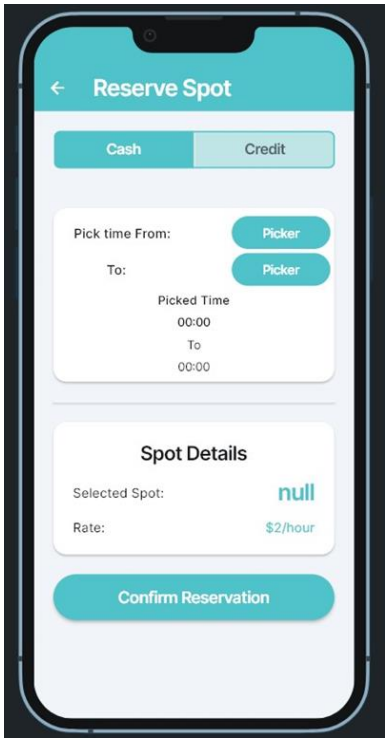
Implementation in FlutterFlow:

1. **Header Section:**
    - The page performs a query to **Supabase** on page load to retrieve the current user's data.
    - The profile picture image widget and the text fields for name, email, and phone are bound to the backend data for dynamic display. A default value of "None" is set for text fields.
2. **App Settings Section:**
    - A vertical list of buttons is used to display the settings options.
    - Each button is configured with a navigation action to open the appropriate screen.
3. **Log Out Button:**

    - The button is configured with an `onTap` action that first calls the **Supabase** 'Log Out' function, then navigates the user to the Login screen.

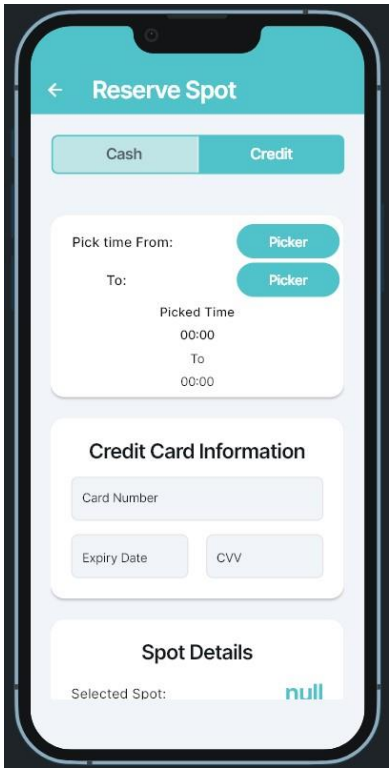
6.6.6.11. Reserve Spot Screen Design

Purpose: The Reserve Spot screen allows users to enter the necessary information to finalize their reservation. It guides them through selecting a time, choosing a payment method, and confirming the booking details.



Key Features

- **Payment Method Selection**: A prominent tab bar allows users to toggle between "Cash" and "Credit" as their preferred payment method.
- **Time Selection**: Users can choose their desired reservation start and end times using interactive time "Picker" buttons.
- **Credit Card Information**: When "Credit" is selected, a dedicated section appears for entering credit card details, including the Card Number, Expiry Date, and CVV.
- **Spot Details**: A card displays a summary of the selected spot, including its identifier (which is passed from the home screen) and the hourly rate.
- **Confirmation Button**: A "Confirm Reservation" button allows the user to submit their details and finalize the booking process.



Design Elements

- **Header**: A standard app header with a back arrow and the title "Reserve Spot".
- **Layout**: The screen is divided into clear, sectioned cards for each part of the process (Time, Payment, and Spot Details), enhancing readability. The layout is scrollable to accommodate all fields.
- **Tabs and Buttons**: The payment method tabs and action buttons are styled with the app's primary teal color, ensuring a consistent and intuitive user interface.

Functionality

- **Dynamic UI**: The "Credit Card Information" card is conditionally displayed based on the user's selection in the payment method tab bar.
- **Data Passing**: The screen receives the chosen spot's ID from the Home Page and displays it in the "Spot Details" section, ensuring the user is reserving the correct spot.

- **Reservation Confirmation**: When the user clicks the "Confirm Reservation" button, the app validates the inputs and initiates the booking. If "Credit" is selected, it securely processes the payment through a third-party gateway. A new record is then created in the **Supabase** database to log the reservation.

Implementation in FlutterFlow

- **State Management**: Page state is used to manage the active payment tab (Cash/Credit) and to store the selected reservation times, which controls the conditional visibility of the payment form.
- **Page Parameters**: The screen is configured to accept a spot_id as a page parameter, which is passed from the Home Page.
- **Payment Gateway Integration**: For credit card payments, the screen integrates with a third-party payment gateway like Stripe to securely handle transactions.

Backend Integration

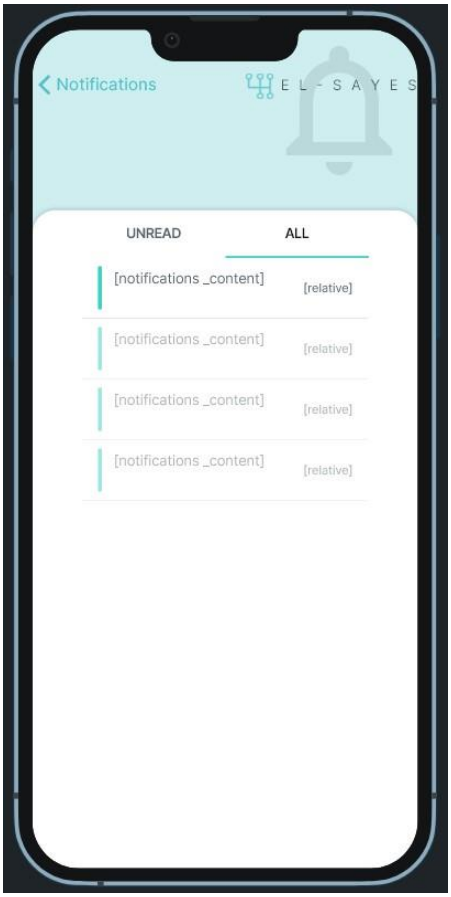
- The "Confirm Reservation" button triggers an action flow that creates a new row in the **Supabase** reservations table, saving the user_id, spot_id, start time, and end time.

User Flow

1. **Section 1: Reservation Details**
    - The user enters their alternate phone number (optional) and selects the reservation time using the time selection widget.
2. **Section 2: Payment Information**
    - The user enters their credit card details (card number, expiry date, CVV) and sees a summary of the selected spot with its identifier and hourly rate.
3. **Section 3: Confirmation and Submission**
    - The user reviews their reservation details and clicks the "Confirm Reservation" button to submit the information and process payment.
4. **Payment Confirmation:**
    - After the payment is processed, the user receives a confirmation message and a digital receipt or reservation details.

6.6.6.12. Notifications Screen

Purpose: To provide users with a dedicated space to view all alerts and communications from the app. This includes important updates like booking confirmations, reminders for expiring parking times, or special offers, ensuring the user stays informed about their parking activity.



Key Features

- **Tabbed View**: Features "UNREAD" and "ALL" tabs, allowing users to easily filter and organize their notifications.
- **Dynamic Notification List**: Displays a list of all incoming notifications. Each entry shows the notification content and a relative timestamp (e.g., "2 hours ago").
- **Back Navigation**: A back arrow in the header allows users to return to the previous screen.
- **Clear All Button**: A button that gives users the option to clear all notifications from their list.

Design Elements

- **Header**: The screen includes the standard app header with a back button and the "EL-SAYES" logo. A large, faint notification bell icon is used as a background watermark to reinforce the screen's purpose.
- **Notification Container**: A white container with rounded corners holds the notification list, creating a clean, card-like appearance.
- **Tabs and List Style**: A simple tab bar with an underline indicator highlights the active section. Each notification in the list is accented with a vertical teal bar on the left, improving readability.
- **Buttons**: The "Clear All" button is styled with the app's primary teal color and rounded corners, making it a clear and accessible call to action at the bottom of the screen.

Functionality

- **Data Fetching**: The screen dynamically fetches and displays a list of notifications for the current user from a dedicated table in the **Supabase** database.
- **Tab Filtering**: Tapping on the "UNREAD" or "ALL" tabs filters the list based on the notification's read status in the backend.

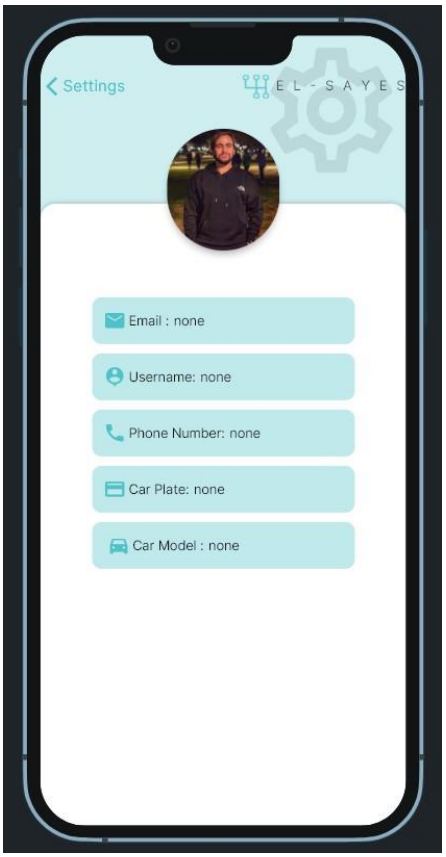
- **Clear All**: When a user presses the "Clear All" button, a backend action is triggered to delete or update the user's notifications in the **Supabase** database, and the list is refreshed.

Implementation in FlutterFlow

- **Backend Integration**: The notification list is generated by a ListView widget that is connected to a **Supabase** query. This query retrieves all notifications associated with the logged-in user.
- **Tab Logic**: A TabBar widget is used to manage the view. The state of the selected tab is used to apply a filter to the Supabase query, showing all items or only those where an is_read column is false.
- **Button Action**: The "Clear All" button is configured with an onTap action that calls a **Supabase** function (RPC) or a delete query to remove the user's notification records from the database. The page state is then rebuilt to reflect the change.
- **Navigation**: The back arrow in the AppBar is configured with a "Navigate Back" action.

6.6.6.13. Settings Screen

Purpose This screen allows users to view their core profile and vehicle information that is stored in the system. It serves as a personal data overview, providing a clear, read-only display of the user's registered details, accessible from the main Profile page.



Key Features

- **Profile Picture Display**: A prominent, circular view of the user's current profile picture.
- **User Information Fields**: A comprehensive list of the user's account and vehicle details, including:
  - Email
  - Username
  - Phone Number
  - Car Plate
  - Car Model
- **Back Navigation**: An arrow in the header allows the user to easily navigate back to the Profile screen.

Design Elements

- **Header**: The screen features the standard app header with a back button and the "EL-SAYES" logo. A large, faint gear icon is used as a background watermark to visually signify the "Settings" theme.
- **Profile Picture**: A large, circular image is centered at the top of the content area, drawing focus to the user's identity.
- **Information Layout**: User details are organized in a vertical list. Each piece of information is presented in its own container with a light teal background, rounded corners, and a relevant icon, making the information easy to scan and understand.

Functionality

- **Data Fetching**: When the screen is loaded, it dynamically fetches all displayed user information (profile picture, email, car plate, etc.) from the user's record in the **Supabase** database.
- **Read-Only Display**: The fields on this screen are for informational purposes, allowing the user to review their current data. If a particular detail is not available in the database, the screen displays "none" as a default placeholder.

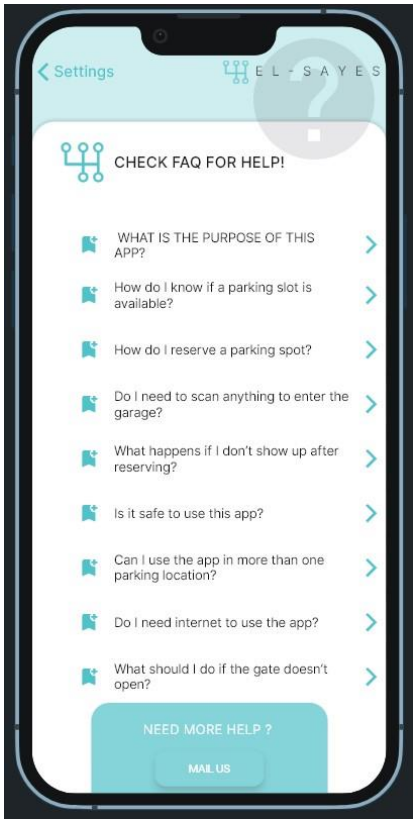
Implementation in FlutterFlow

- **Backend Integration**: The page is configured to query the **Supabase** database on load to retrieve all necessary data for the currently authenticated user.
- **Data Binding**: Each widget, including the profile image and the various text fields, is bound to the corresponding column from the Supabase query result. A default value is set for each text field to handle cases where data might be null.
- **Navigation**: The back arrow in the AppBar is configured with a 'Navigate Back' action, providing a simple way for the user to return to the Profile screen.

6.6.6.14. Help Screen

Purpose: To provide a self-service support center that allows users to quickly find answers to Frequently Asked Questions (FAQs). This screen aims to resolve common queries without the need for direct contact, covering topics from app usage to security.

Key Features



- **FAQ List**: A comprehensive, scrollable list of common questions related to app functionality, such as reserving a spot, garage entry, and payment.
- **Contact Option**: A "MAIL US" button is provided for users who require further assistance beyond the scope of the FAQs.
- **Back Navigation**: A back arrow in the header allows users to return to the Profile Screen.

Design Elements

- **Header**: The screen is titled "CHECK FAQ FOR HELP!" and features the app's gearshift logo to maintain brand consistency.
- **FAQ List Style**: Each question is presented in a separate row with a bookmark icon to the left and a chevron icon to the right, indicating that it is a clickable item that leads to more information.
- **Contact Button**: The "MAIL US" button is designed as a large, prominent teal button, making it an accessible option for users needing more help.

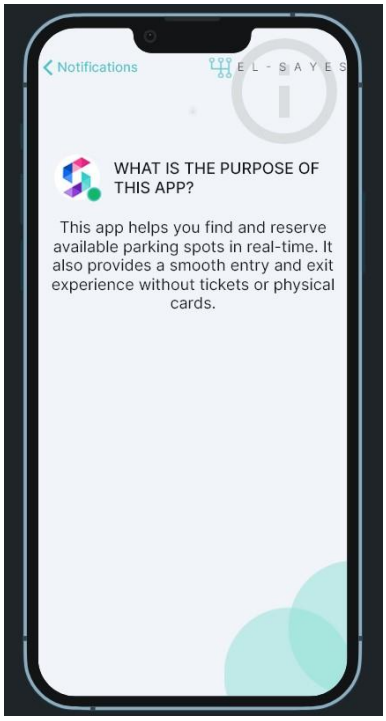
Functionality

- **FAQ Navigation**: When a user taps on a question, they are navigated to a "Help Details Screen" where the corresponding answer is displayed.
- **Email Support**: Tapping the "MAIL US" button launches the device's native email client, allowing the user to send a message directly to the support team.

Implementation in FlutterFlow

- **Navigation Actions**: Each item in the FAQ list is configured with an `onTap` action to navigate to the "Help Details Screen". It passes the selected question and its answer as page parameters.
- **Mail To Action**: The "MAIL US" button is configured with a 'Launch URL' action using a `mailto:` link to open the user's email application.

6.6.6.15. Help Details Screen



Purpose: To provide a clear and focused view of the answer to a specific question selected by the user from the main Help Screen.

Key Features

- **Dynamic Content Display**: The screen dynamically displays the question selected by the user and its detailed answer.
- **Back Navigation**: A standard back arrow in the header allows the user to return to the FAQ list on the Help Screen.

Design Elements

- **Layout**: The content is presented within a clean, white container. A colorful app icon is displayed next to the question for visual branding.
- **Typography**: The question is styled in a bold, uppercase font to serve as a clear title, while the answer is presented in a simple, readable paragraph format.
- **Background**: The screen maintains the app's consistent light blue background with subtle branding elements.

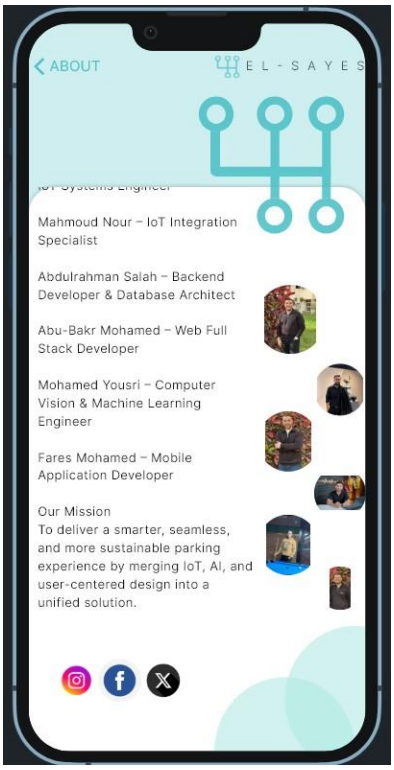
Functionality

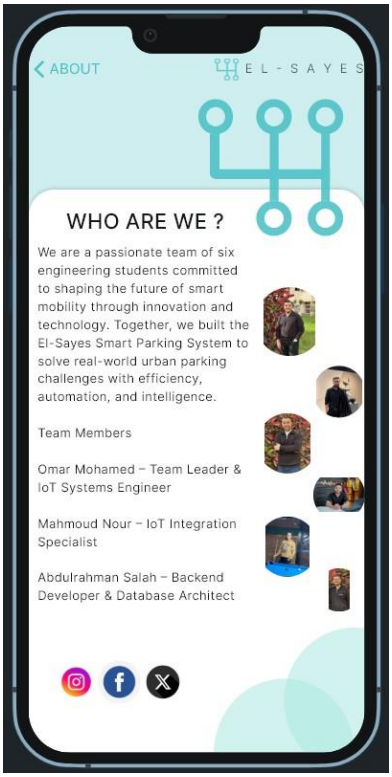
- **Content Display**: The screen receives the question and answer text as navigation parameters from the Help Screen and displays them accordingly.
- **Navigation**: The back button returns the user to the previous screen, allowing them to browse other questions in the FAQ list.

Implementation in FlutterFlow

- **Page Parameters**: The page is designed to accept `question` and `answer` as string parameters.
- **Data Binding**: The text widgets on the screen are bound to the page parameters, ensuring the correct information passed from the Help Screen is displayed.
- **Navigation**: The `AppBar` includes a back arrow that is configured with a 'Navigate Back' action.

6.6.6.16. About Screen





Purpose: To introduce the users to the development team behind the El-Sayes Smart Parking System. This screen provides transparency about who created the app, details their roles and mission, and offers ways to connect with the project on social media.

Key Features

- **Team Introduction**: A section titled "WHO ARE WE?" that describes the team's background and commitment to solving urban parking challenges through technology.
- **Team Member List**: A detailed list identifying each member of the development team and their specific role in the project, including:
    - Omar Mohamed – Team Leader & IoT Systems Engineer
    - Mahmoud Nour – IoT Integration Specialist
    - Abdulrahman Salah – Backend Developer & Database Architect
    - Abu-Bakr Mohamed – Web Full Stack Developer
    - Mohamed Yousri – Computer Vision & Machine Learning Engineer
    - Fares Mohamed – Mobile Application Developer
- **Mission Statement**: A dedicated section outlining "Our Mission" to deliver a smarter and more sustainable parking experience by merging IoT, AI, and user-centered design.
- **Social Media Links**: Clickable icons for Instagram, Facebook, and X (formerly Twitter) located at the bottom of the screen.
- **Back Navigation**: A back arrow in the header to return to the previous screen.

Design Elements

- **Layout**: The content is organized within a single, scrollable white container with rounded corners, set against the app's standard light blue background. The app's gearshift logo is used as a faint watermark.
- **Team Showcase**: The names and roles of team members are listed clearly. Small, circular profile pictures are displayed alongside the list to add a personal touch.

- **Social Media Icons**: The social media links are represented by clean, circular icons containing the official logos of the platforms.

Functionality

- **Informational Content**: This screen is primarily static, designed to provide information about the project's creators and their goals.
- **Social Media Navigation**: Tapping on any of the social media icons will launch the corresponding social media platform, directing the user to the project's or team's page.

Implementation in FlutterFlow

- **Layout Structure**: The screen is built using a SingleChildScrollView to ensure all content is accessible, even on smaller devices.
- **Static Assets**: All text and images on this page are implemented as static assets within the FlutterFlow project.
- **Launch URL Action**: Each social media icon is configured with an onTap action that uses the 'Launch URL' function to open the respective social media link.
- **Navigation**: The AppBar contains a back arrow with a 'Navigate Back' action to return the user to the Profile Screen.

6.7 Web Application

6.7.1 Web Overview

Elsayes is a modern web-based solution designed to streamline smart parking services. The application consists of two major modules:

- A **customer-facing marketing page**, showcasing the platform's services and allowing users to download the mobile app.
- An **admin/operator dashboard**, built for garage operators to monitor real-time parking data, manage reservations, and maintain operational oversight.

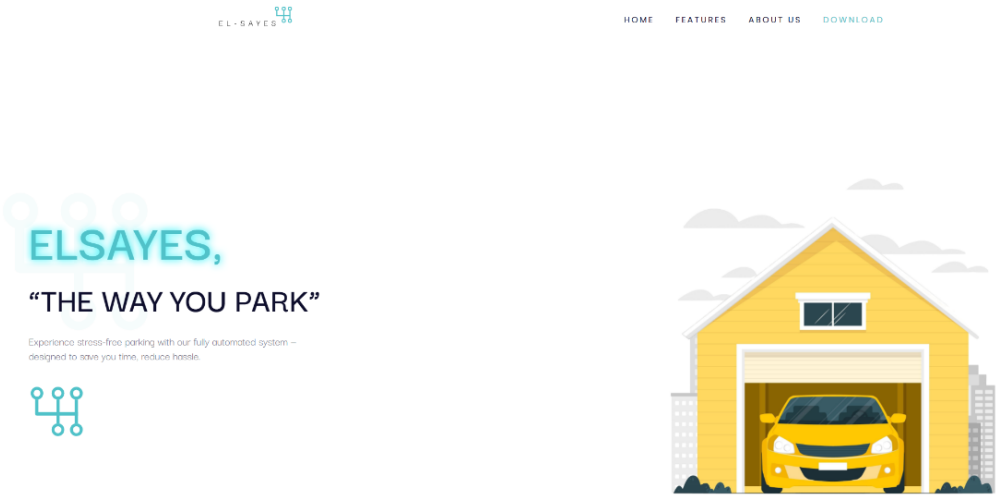
The project is built using **Angular 20** and follows modern development standards to ensure scalability, responsiveness, and maintainability.

6.7.2 Web Implementation

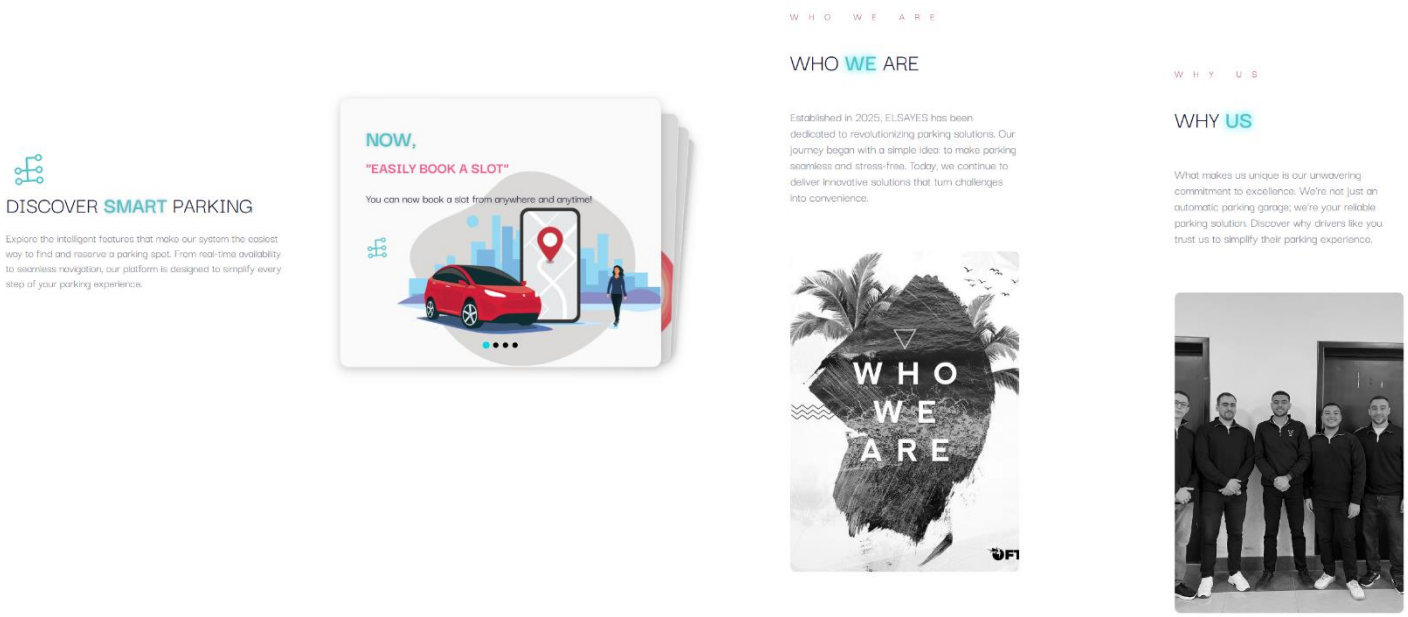
So using SPA (Single Page Application) we built 2 different pages each with it’s own purpose:

Marketing Page (Public Interface)

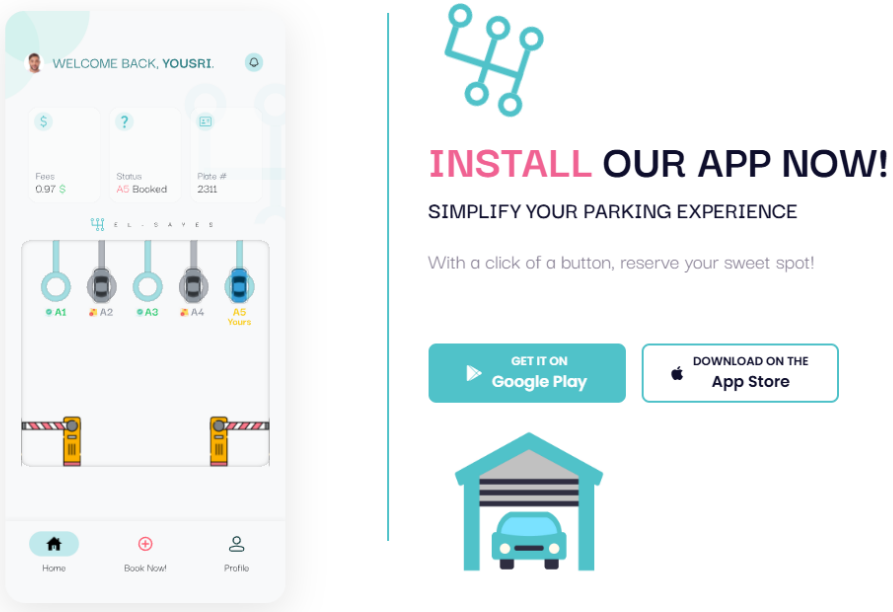
- Designed and implemented a clean, modern landing page.



- Integrated animations using GSAP to enhance user engagement.

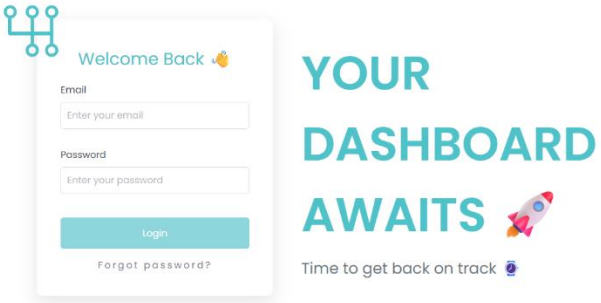


- Added download links for the mobile application to drive adoption.



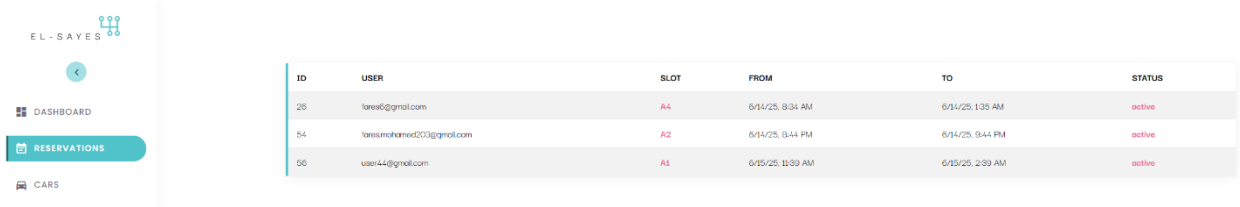
Admin/Operator Dashboard

- Developed a secure login system with role-based access.

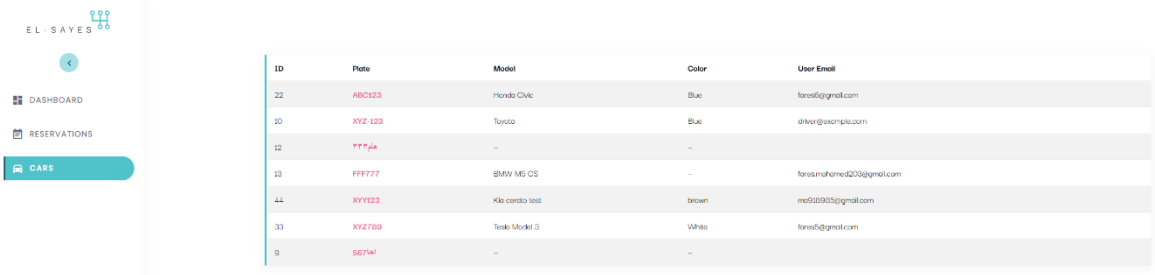


- Created a metrics dashboard with charts and statistics to monitor garage status.

- Built dynamic table views for reservations and car data, ensuring real-time visibility.



- Implemented authenticated routing and protected views using Angular Guards.



6.7.3 Technical Stack

This project leverages a combination of modern front-end frameworks, cloud services, and third-party libraries to deliver a robust and scalable web application. The following technologies were used:

Frontend Technologies

- **Framework: Angular 20.0.0**
    - A cutting-edge front-end framework maintained by Google.
    - Used for building single-page applications (SPAs) with a modular, component-based architecture.
    - Offers built-in tools for routing, form handling, dependency injection, and state management.
- **Language: TypeScript 5.8.2**
    - A statically typed superset of JavaScript.
    - Provides type checking at compile time, making the code more reliable and maintainable.
    - Enables the use of modern JavaScript features along with strong tooling support in IDEs.
- **Styling: Bootstrap 5.3.6**
    - A popular CSS framework for creating responsive, mobile-first layouts.
    - Used to ensure consistent design, spacing, and alignment across different screen sizes.
    - Customized to match the branding and design guidelines of the project.

Database & Backend Services

- **Supabase (@supabase/supabase-js 2.50.0)**
    - A backend-as-a-service platform that offers real-time PostgreSQL databases, authentication, and file storage.

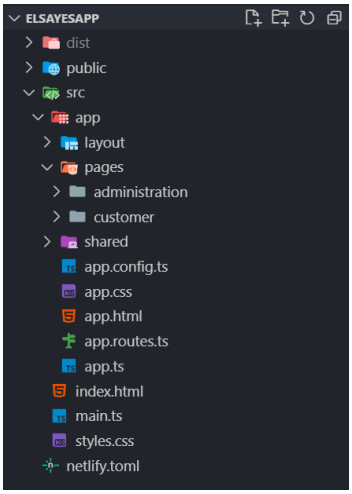
- Used for:
    - Authentication: Secure user sign-up and sign-in processes.
    - Database Operations: CRUD operations on entities like reservations and cars.
    - Realtime Subscriptions: Live updates for data tables using Supabase's event-based listeners.
    - Storage: Uploading and serving assets (e.g., user profile images, logos).

Key Libraries and Tools

- **UI & UX Enhancements**
    - **Angular Material**
        1. Provides a set of pre-built, well-designed UI components following Google's Material Design principles.
        2. Used for form inputs, buttons, dialog modals, and toolbars.
    - **Font Awesome 6.7.2**
        - Icon library used to add visual clarity and branding to buttons, headers, and navigation elements.
- **Animations: GSAP (GreenSock Animation Platform)**
    - A professional-grade animation library.
    - Used for smooth transitions, hero text animations, scroll-triggered effects, and entrance animations on the marketing page.
    - Enhances user engagement and improves perceived performance.
- **Carousels: Swiper 11.2.8**
    - A lightweight, mobile-friendly carousel/slider library.
    - Used on the homepage to display key features and app screenshots interactively.
    - Supports autoplay, navigation buttons, and touch gestures.

- **Data Visualization**
    - Chart.js 4.4.9
        - JavaScript charting library for rendering responsive and animated charts (e.g., bar, line, pie charts).
    - ng2-charts 8.0.0
        - Angular wrapper for Chart.js.
        - Used to integrate dashboard metrics such as total reservations, parking statistics, and revenue insights seamlessly.

6.7.4 Application Architecture



We followed a clean & scalable architecture based on new modern structure of SPA projects

6.7.5 Key Features

Authentication

- Secure login with Supabase Using the built in authentication
- Role-based access and routing

Garage Monitoring Dashboard

- Real-time metrics on parked cars, reservations, payments
- Dynamic charts using Chart.js
- Responsive design for desktop and tablets

Data Management

- Table views for reservations and cars with sorting and editing
- Real-time updates through Supabase subscriptions

Customer Site

- Interactive marketing content
- Animated sections powered by GSAP
- Download links for mobile applications

6.7.6 Development Highlights

Performance Optimization

- Lazy loading for major modules
- Tree-shaking and optimized production builds
- Efficient image handling and minification

Testing

- Jasmine for unit testing
- Karma test runner with code coverage reporting

Security

- Protected admin routes
- Email/password auth using Supabase
- Input sanitization and client-side validations

6.7.6.1 Source Control and Version Management

To ensure efficient collaboration, change tracking, and code integrity throughout the development lifecycle, we used **Git** as the primary version control system, along with **GitHub** as the remote repository hosting platform.

Git Workflow

- Followed a **feature-branch workflow** to isolate development of new features and bug fixes.
- Used main as the stable production branch, and dev for staging and testing.
- Feature branches (e.g., feature/auth, feature/dashboard-ui) were merged into dev through pull requests.

Collaboration & CI/CD

- GitHub was used for code collaboration and issue tracking.
- Netlify was connected directly to GitHub for **continuous deployment**, triggering new builds automatically on push to the main branch.

Benefits of Source Control

- Enabled safe rollback to previous versions if needed.
- Simplified code reviews and collaboration with team members.
- Facilitated automated deployment through GitHub-Netlify integration.

6.7.7 Deployment

- **Hosting**: Netlify (CI/CD setup with GitHub)
- **Environment Handling**: Separate configurations for development and production
- **Build Optimization**: Minified assets, lazy loaded charts, and modular routes

6.7.8 Challenges and Solutions

Challenge	Description	Solution
Swiper styles breaking in production	Swiper styles were not loading correctly on Netlify	Included styles manually and configured global styles properly
Netlify deployment issues	Environment variables and lazy modules causing build errors	Fixed build settings and ensured proper route fallback
Auth guarding logic	Dashboard was accessible without login during dev	Implemented role-based AuthGuard and fallback redirection

6.7.9 Future Enhancements

- Develop a **dedicated mobile app** with deep linking to web features.
- Add **push notifications** for reservation updates.
- Implement **advanced analytics** for garage usage trends.
- Introduce **PWA (Progressive Web App)** support.
- Expand payment integration (e.g., Stripe, Apple Pay).

6.7.10 Conclusion

The Elsayes project showcases the application of modern web development techniques to solve real-world problems in the parking domain. By separating customer and admin functionalities, and utilizing a scalable architecture, the system ensures ease of use, performance, and maintainability.

Chapter Seven

Conclusion

7. Chapter seven

The **"El-Sayes" Smart Parking System** represents a groundbreaking solution to the persistent challenges of urban parking. By leveraging advanced technologies such as **IoT, computer vision, and real-time analytics**, the system directly addresses critical issues, including traffic congestion caused by prolonged parking searches, inefficient space utilization, security vulnerabilities, and the environmental toll of idling vehicles.

The system's **IoT-enabled features**, including **real-time slot monitoring** and **automated gate operations powered by license plate recognition**, are seamlessly integrated with a centralized **Supabase backend**. This ensures reliable synchronization, operational efficiency, and a smooth user experience for both drivers and administrators.

Insights gathered from extensive surveys with car owners and garage operators highlighted key pain points—91.6% of users reported difficulty finding parking, while delays and security concerns were also prominent. These findings directly informed the functional and non-functional requirements, resulting in a design that effectively addresses real-world needs. The system combines **hardware components** like IR sensors, servo motors, and ESP32-CAM modules with **software elements** such as OCR-based plate detection, ensuring seamless interaction between users and infrastructure.

The project's **business model** is equally robust, offering revenue opportunities through subscriptions, pay-per-use options, and analytics services. A SWOT analysis highlighted system strengths in scalability and innovation, as well as opportunities for integration with smart city initiatives. Implementation of **YOLOv9 for license plate recognition** has further improved detection accuracy and system reliability.

Looking ahead, the **"El-Sayes" system is future-ready**, with plans for multi-garage support, EV charging integration, valet automation, and on-street slot detection. Sustainability goals, such as the use of solar-powered infrastructure, further align the system with global environmental initiatives.

By combining cutting-edge technology with a user-centered design approach, "El-Sayes" sets a new benchmark for intelligent parking solutions—paving the way for smarter, safer, and more sustainable urban mobility.

References

- https://egyptinnovate.com/en/users/rakna
- https://www.smartparking.com/uk
- https://ramec-misr.com/
- https://thaki.online/en/
- https://www.parkassist.com/solutions-2/
- https://www.parksmarter.org.uk/home.aspx
- https://www.parkme.com/en-gb/map
- https://www.voicepark.org/
- https://www.montgomerycountymd.gov/DOT-Parking/Resources/Files/FY18_ParkingSurvey.PDF
- https://www.mtsac.edu/president/cabinet-notes/2016-17/PTK_College_2016_Parking_Student_Survey.pdf
- https://www.astoria.gov/Assets/dept_1/pm/pdf/parkingsurveyreport_final.pdf
- https://www.mapc.org/wp-content/uploads/2010/02/HTD_5.pdf
- https://randomnerdtutorials.com/esp32-cam-save-picture-firebase-storage/
- https://www.handsontec.com/dataspecs/RC522.pdf
- https://makerselectronics.com/product/ws2811-addressable-rgb-led-12v?campaignid=20503411856&adgroid=up&network=x&device=c&campaignname=sales_pmax&gad_source=1&gclid=Cj0KCQiA4rK8BhD7ARIsAFe5LXIZfWWOeNF_fNqolNB1GqK2MjGG-0PFuuz4o5j0uIHyZBV4bbSUp10aAqTvEALw_wcB
- https://mostelectronic.com/shop/arduino-development-boards/esp32-cam-development-board-with-camera/?gad_source=1&gad_campaignid=22607246765&gbraid=0AAAAA9fT608izZ0z2QJ3gqc0sNWrxjqby&gclid=CjwKCAjw3rnCBhBxEiwArN0QEy3bawmZDef6odk3Nz8cv2A94STYpaQ9HLMdXTZVe3_nsOa9sPxiZBoCF5cQAvD_BwE
- https://supabase.com
- https://roboflow.com

- https://www.flutterflow.io/enterprise?utm_term=low-code%20application%20platform&utm_campaign=US-Enterprise&utm_source=adwords&utm_medium=ppc&hsa_acc=2507013754&hsa_cam=22383834188&hsa_grp=178684838762&hsa_ad=742293197165&hsa_src=g&hsa_tgt=kwd-340881025901&hsa_kw=low-code%20application%20platform&hsa_mt=b&hsa_net=adwords&hsa_ver=3&gad_source=1&gad_campaignid=22383834188&gbraid=0AAAAAB3BfxPtD1ribiPifcwDajinDVYkS&gclid=CjwKCAjw3rnCBhBxEiwArN0QE0NUQiWh7A4CBmIVf_kMz3rf_OIk8kLOPdiRZCTwpSWGqGioG938QxoCFRIQAvD_BwE