GRADUATION PROJECT II

Smart Parking System

El-Sayes

Submitted By

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

Program of Data Science

Faculty of Computers and Data Science

Alexandria University

2025 Semester 8

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 Firebase 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, RFID readers, 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

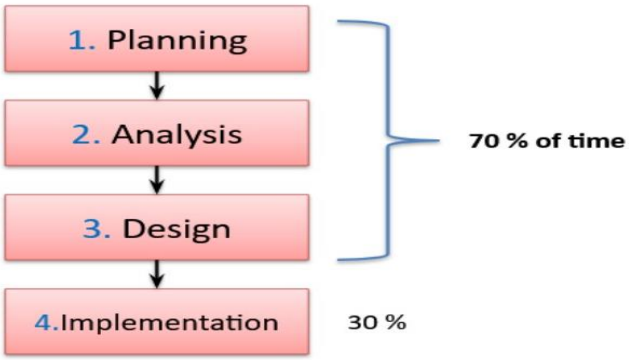
- **Objective:**
  - Define user requirements and understand the system's needs.
- **Activities:**
  - Collect user feedback through surveys and interviews.
  - Document functional requirements (e.g., slot reservation, flexible payments) and non-functional requirements (e.g., scalability, security).
  - 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

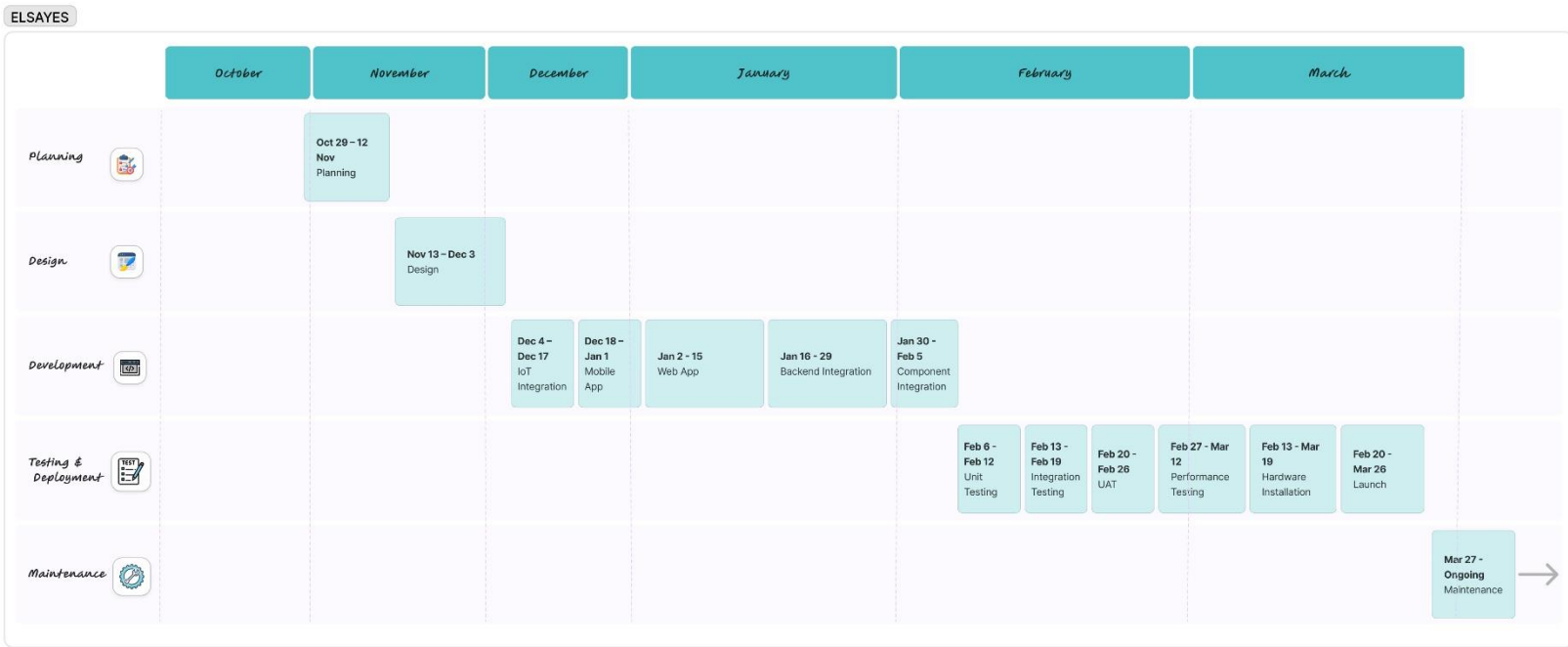
Development Phase:

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").

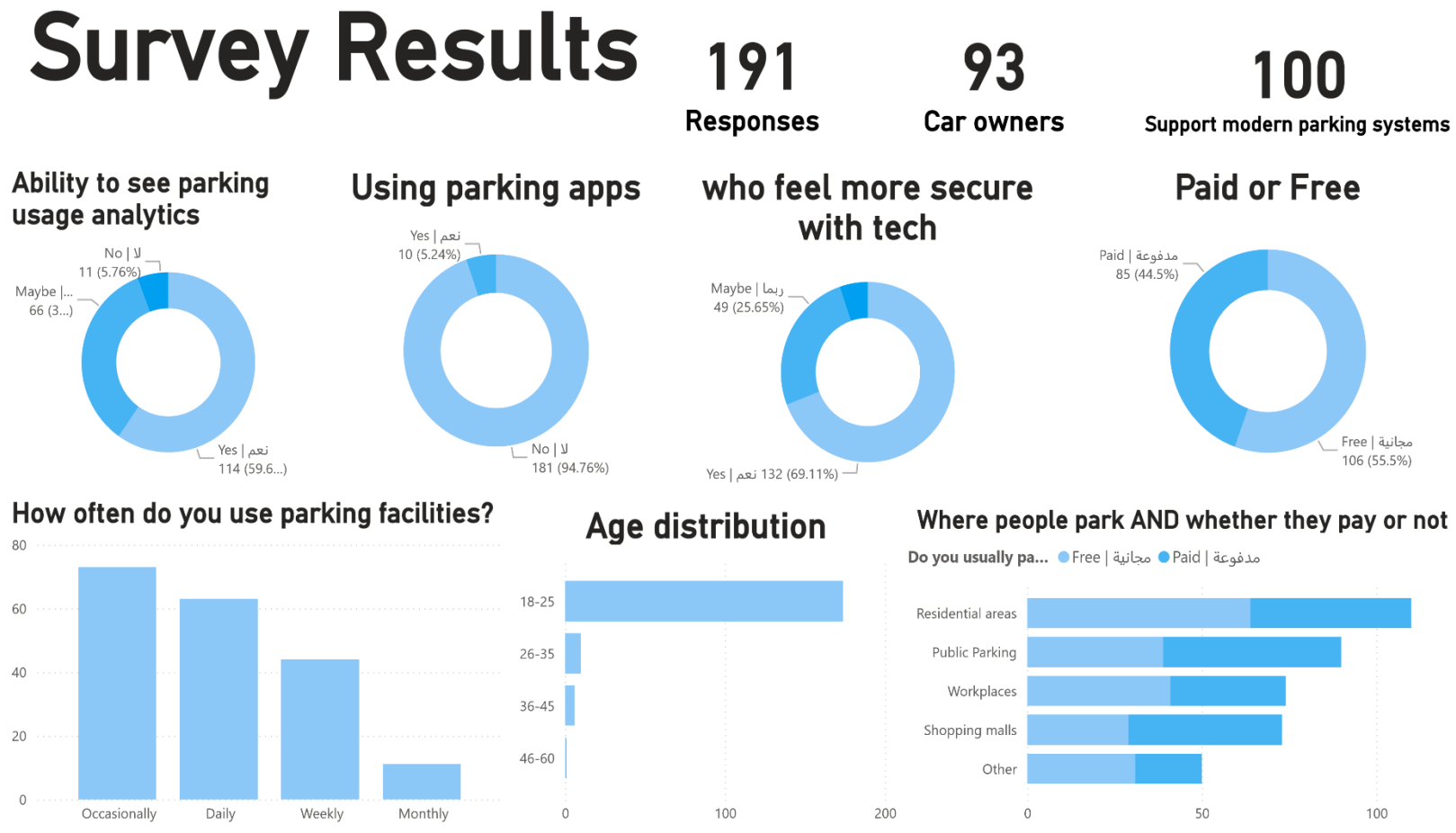
Below are the details of the methodology used:

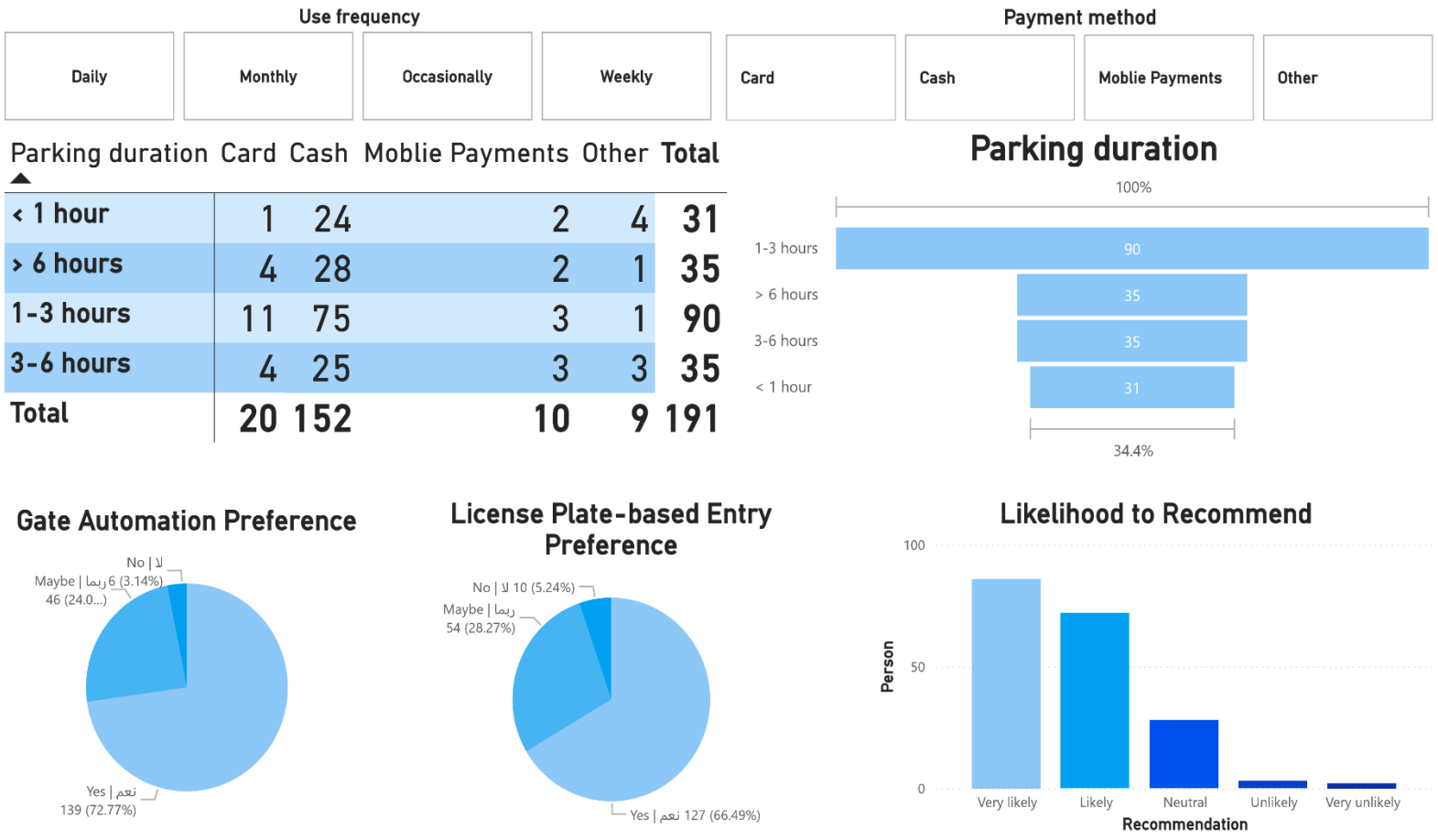
1. **Survey Design**:
    - The survey consisted of 24 structured questions, covering areas such as parking frequency, preferred payment methods, major frustrations, and interest in modern parking solutions.
    - Both multiple-choice and 2 open-ended questions were included to gather quantitative and 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

Actors:

1. **Admin**:
    - The **Admin** is a user responsible for managing the overall operation of the parking garage system. They can access and manage data, create reports, and perform administrative tasks, such as updating queue information and managing parking slots.
2. **User (Driver)**:
    - The **User** or **Driver** is the customer who wants to park their vehicle. They interact with the system to reserve a parking slot, make payments, and check the status of available parking spaces.
3. **Web Application**:
    - The **Web Application** acts as an interface for both the **Admin** and **User** to interact with the system. It provides the necessary interfaces for slot reservations, viewing slot statuses, and managing data like reports.
4. **Machine Learning Model**:
    - The **Machine Learning Model** is responsible for processing images of cars and recognizing their license plates. It interacts with the system to extract text (license plate numbers) from photos uploaded by the cameras.
5. **Supabase**:
    - **Supabase** is the cloud database that stores all of the real-time data. It stores information like parking slots, user data (car plate numbers, payment details), queue information, and reservation details.

4.1.1 Use Case for Admin

1. **Request Data**:
    - The **Admin** can request and retrieve data from the system, such as current parking slot statuses, the list of cars in the garage, and payment information. This will likely be handled through the **Web Application** that connects to **Supabase**.
2. **Generate Report**:
    - The **Admin** can generate reports for analysis purposes. Reports could include parking slot usage, total payments collected, or the number of reservations made. The **Web Application** handles the report generation and interfaces with **Supabase** for data retrieval.
3. **Update Queue**:
    - The **Admin** can update or manage the queue of cars waiting for parking. The queue data is updated in **Supabase**, and the changes are reflected in the **Web Application**.
4. **Manage Slots**:
    - The **Admin** can manage parking slots, including marking them as available, occupied, or reserved. This is done through the **Web Application**, which communicates with **Supabase** to update the slot status in real-time.

4.1.2 Use Case for User (Driver)

5. **Reserve Slot**:
    - The **User** can reserve a parking slot through the **Mobile Application**. This involves selecting a free parking space, entering reservation details (time, payment method), and confirming the booking. The reservation information is stored in **Supabase**, and the status of the slot is updated.

6. **Make Payment**:
    - The **User** can make a payment through the **Mobile Application** to confirm their reservation. This process involves interacting with a payment gateway and updating **Supabase** with the payment status.
7. **View Slot Status**:
    - The **User** can view the status of parking slots. This includes checking for available, occupied, or reserved slots. This data is fetched in real-time from **Supabase** and displayed through the **Mobile Application**.

4.1.3 Web Application

8. **Fetch Slot Data**:
    - The **Web Application** fetches real-time data on the availability of parking slots from **Supabase**. This data is presented to both **Admin** and **Users** in an interactive format.
9. **Display Reports**:
    - The **Web Application** generates and displays reports for the **Admin**. This could include statistical data on the number of cars, parking trends, and payment summaries.

4.1.4 Machine Learning Model

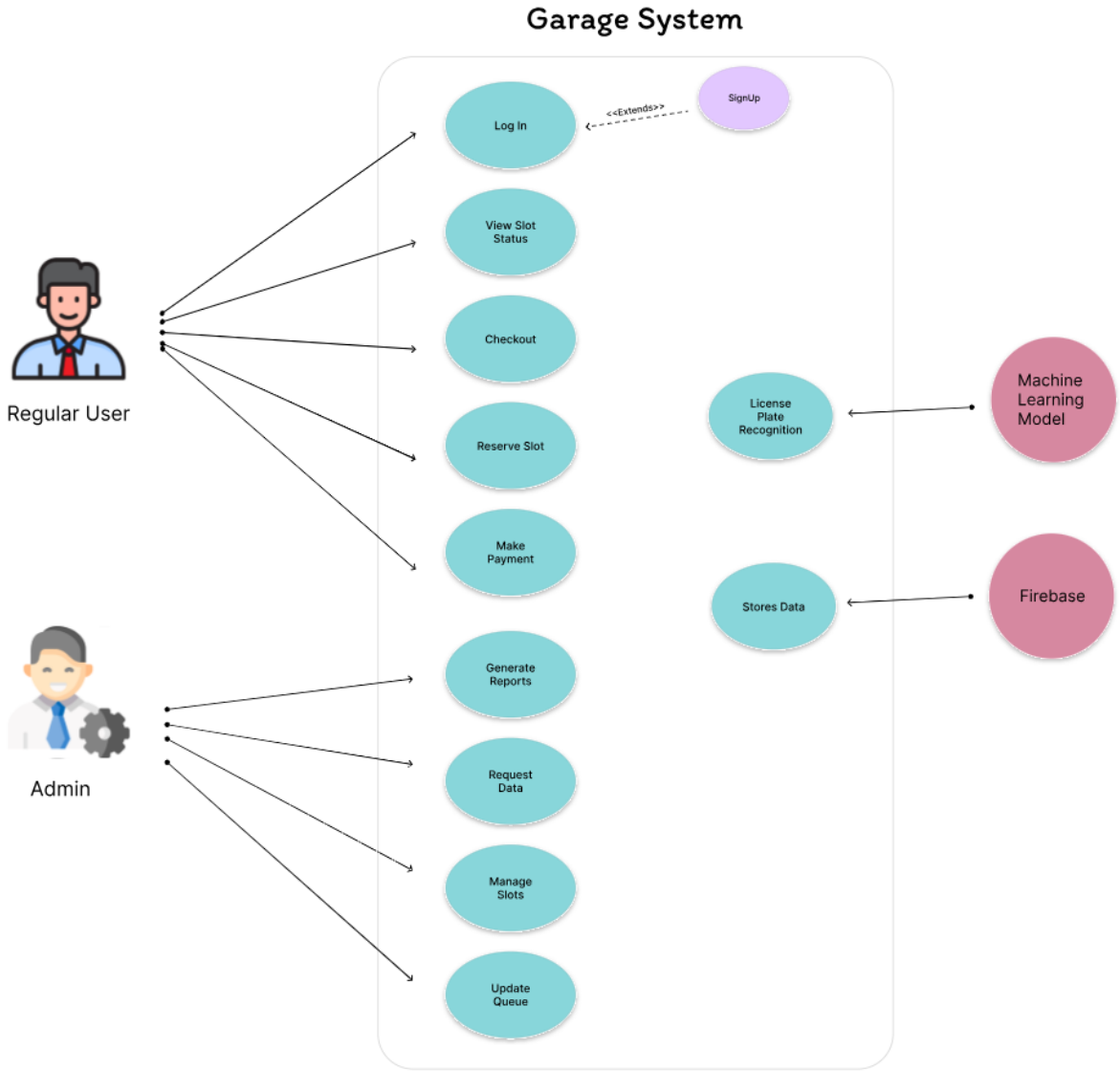
10. **License Plate Recognition**:

- The **Machine Learning Model** processes images captured by cameras at the parking gate to extract the license plate number of the car. This is crucial for tracking the vehicles entering and exiting the garage. The extracted license plate number is sent to **Supabase** for storage and linked to the **User**'s profile or the transaction.

4.1.5 Firebase

11. **Store Data**:

- **Supabase** stores various types of data for the Smart Garage System:
    - **Slot Data**: Status of parking slots (available, occupied, reserved).
    - **Car Data**: User information, license plate numbers, and associated reservations.
    - **Payment Data**: Details of the payments made by the users.
    - **Queue Data**: Information about cars waiting for parking and their status.



Explanation of the Code:

- **Actors**: These represent the users and systems interacting with the Smart Garage system (Admin, User, WebApp, ML_Model, Firebase).
- **Use Cases**: These represent the actions or tasks that each actor can perform (e.g., generating reports, reserving slots, making payments, license plate recognition, storing data).
- **Arrows**: Arrows show the relationships between actors and their use cases. For example, the **Admin** can request data, generate reports, update the queue, and manage parking slots.

How to Use:

1. **Admin** has full control over the system. They can request data, manage slots, update the queue, and generate reports.
2. **User** interacts primarily with the mobile app to reserve slots, make payments, and view slot statuses.
3. **Web Application** provides an interface for both **Admin** and **User** to interact with real-time data, reports, and slot information.
4. **Machine Learning Model** is responsible for processing the license plates of cars that enter or exit the garage.
5. **Supabase** serves as the backend database where all data is stored and managed.

4.2 Sequence Design (Flow Chart)

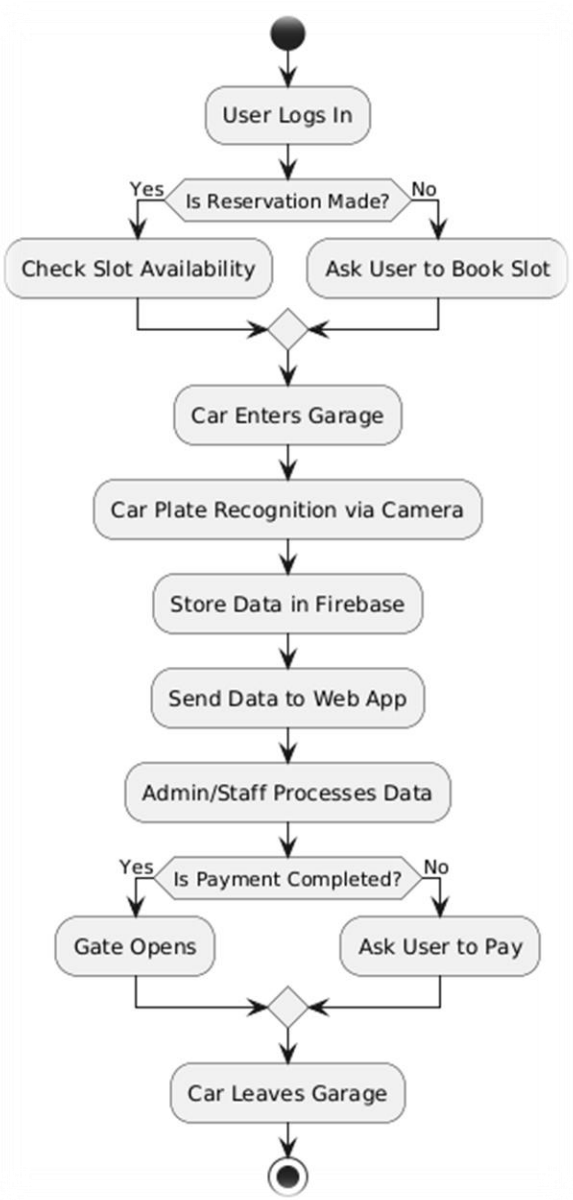
4.2.1 Introduction

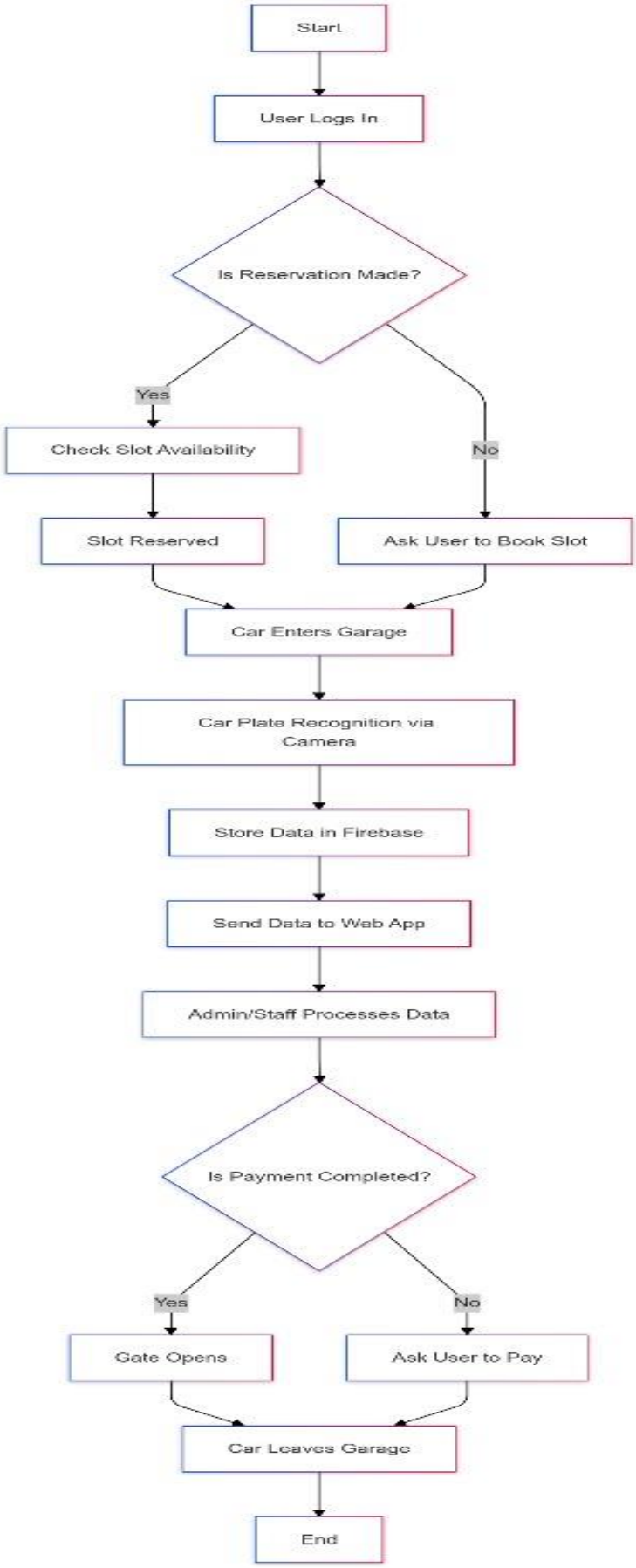
The **Smart Garage System** is a comprehensive parking management solution that integrates IoT sensors, machine learning models, and cloud databases to provide an efficient and automated parking experience. The flow diagram presented here outlines the various stages involved in car parking, from reservation and entry to payment and exit, highlighting the system's components and their interactions.

4.2.2 Flow Diagram

Flow Diagram Overview

The following flow diagram illustrates the main processes of the **Smart Garage System**, demonstrating the interactions between the **user**, **admin**, **IoT sensors**, **mobile/web applications**, **machine learning model**, and **database**




Start
User Logs In
Is Reservation Made?
Yes
No
Check Slot Availability
Slot Reserved
Ask User to Book Slot
Car Enters Garage
Car Plate Recognition via Camera
Store Data in Firebase
Send Data to Web App
Admin/Staff Processes Data
Is Payment Completed?
Yes
No
Gate Opens
Ask User to Pay
Car Leaves Garage
End

4.2.3 Explanation of Flow Diagram Steps

1. **User Logs In:** The process begins when the user logs into the mobile or web application. This step ensures that only authenticated users can access the system.
2. **Reservation Check:** The system checks whether the user has made a reservation.
    - If **Yes**, it proceeds to check the availability of the reserved slot.
    - If **No**, the system prompts the user to make a reservation.
3. **Slot Reservation & Car Entry:** After confirming the reservation, the car enters the garage. If no reservation is made, the car proceeds directly to an available slot.
4. **Car Plate Recognition:** Once the car enters the garage, a camera captures an image of the license plate. The image is then processed using the machine learning model to extract the license plate number.
5. **Data Storage in Firebase:** The extracted license plate number and other car data (such as entry time) are stored in the **Firebase** database for real-time data access and future reference.
6. **Web Application Update:** The data is sent to the **web application** for admin or staff to view in real time, enabling better management of the garage.
7. **Admin Processing:** The **admin** or staff member verifies the car's data, including entry time, and ensures payment is completed.
8. **Payment Check:** If the user has completed the payment:
    - If **Yes**, the system proceeds to open the gate for the car to exit.
    - If **No**, the user is prompted to make the payment before exiting.
9. **Car Exit:** Once payment is confirmed, the system opens the gate, and the car exits the garage.
10. **End:** The process concludes once the car leaves the garage, marking the end of the parking session.

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.).

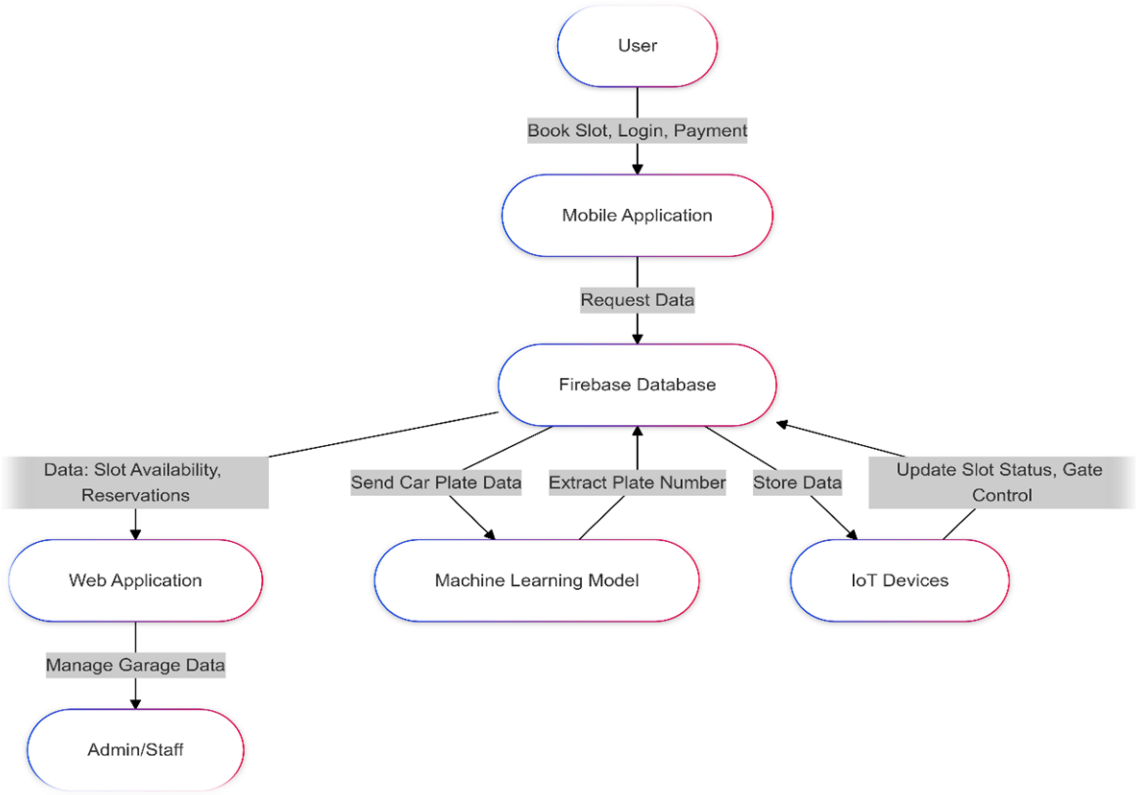
For the **Smart Garage System**, the DFDs are structured to show how data flows between the IoT components, machine learning model, Firebase database, web application, and mobile application. The DFD levels provide insight into system functionality, data interactions, and integration points.

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.



Key Features:

- The user interacts with the system through the mobile application to book slots, make payments, and view garage details.
- IoT devices (IR sensors, RFID readers, and cameras) detect cars, capture images, and update the database.
- Firebase acts as a central repository for real-time data sharing among all subsystems.
- The machine learning model processes images to extract car plate numbers, sending the results back to the database.
- Admins use the web application to monitor the garage, generate reports, and manage operations.

4.3.2.2 DFD Level 1: Subsystems Overview

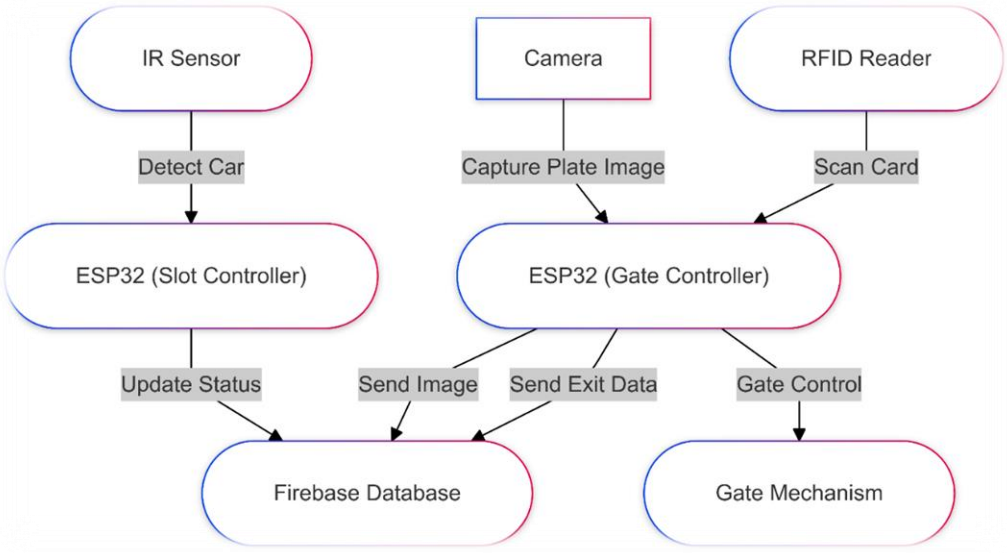
The system is divided into four main subsystems:

1. **IoT Subsystem**
   Handles sensor data and controls hardware like gates and LED indicators for parking slots.
2. **Web Application Subsystem**
   Enables admins to monitor data, manage reports, and analyze garage usage.
3. **Mobile Application Subsystem**
   Allows users to book slots, view real-time parking availability, and make payments.
4. **Machine Learning Model Integration**
   Processes license plate images to extract text and update databases.

DFD1: IoT Subsystem

Description:

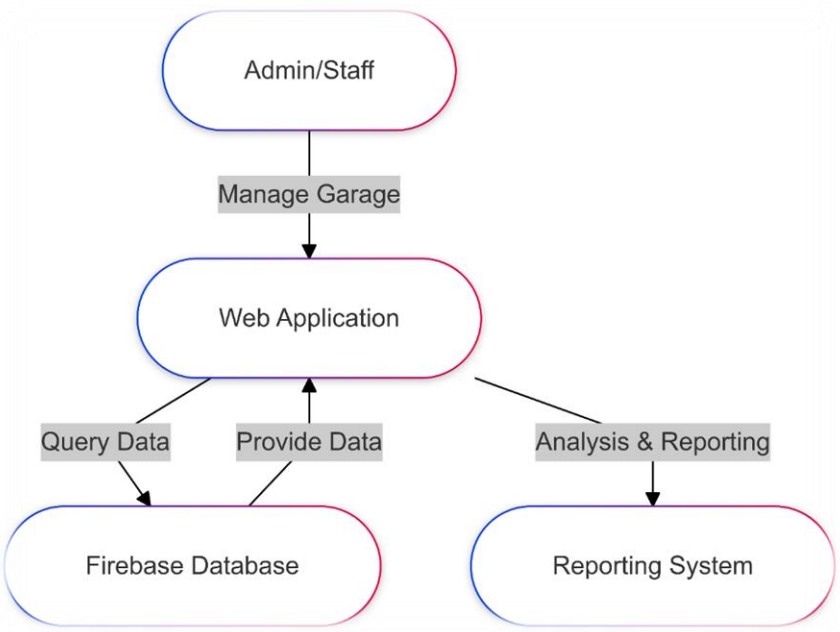
The IoT subsystem manages the hardware components in the garage, including IR sensors, RFID readers, cameras, and ESP32 controllers. These devices interact with Firebase to update real-time data, manage slot availability, and process car entry/exit.



DFD1: Web Application Subsystem

Description:

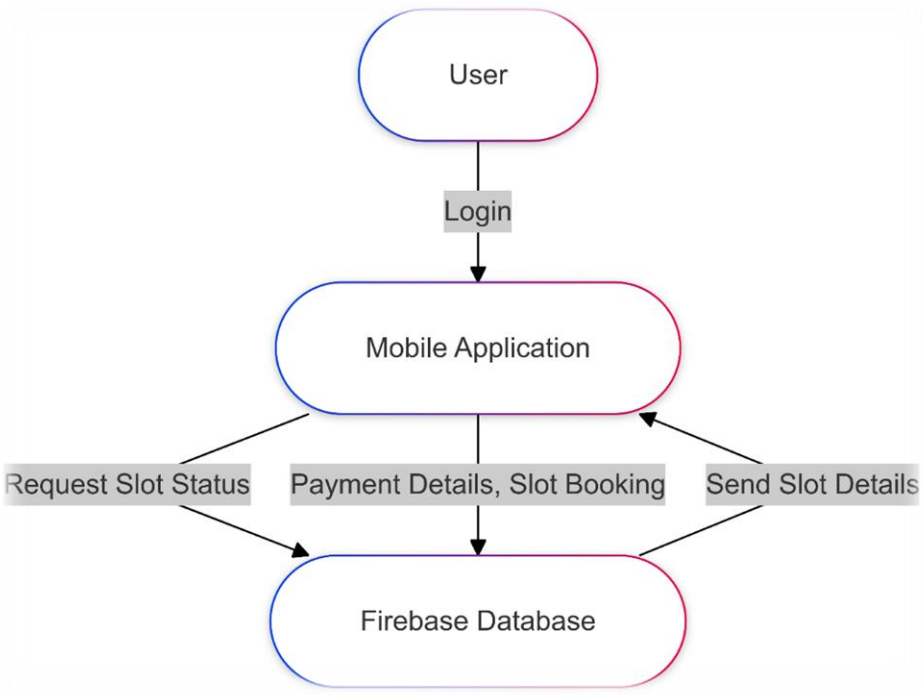
The web application is designed for administrators and staff. It allows them to manage the garage, view slot statuses, generate reports, and analyze performance. Real-time data from Firebase ensures accurate and updated information.



DFD1: Mobile Application Subsystem

Description:

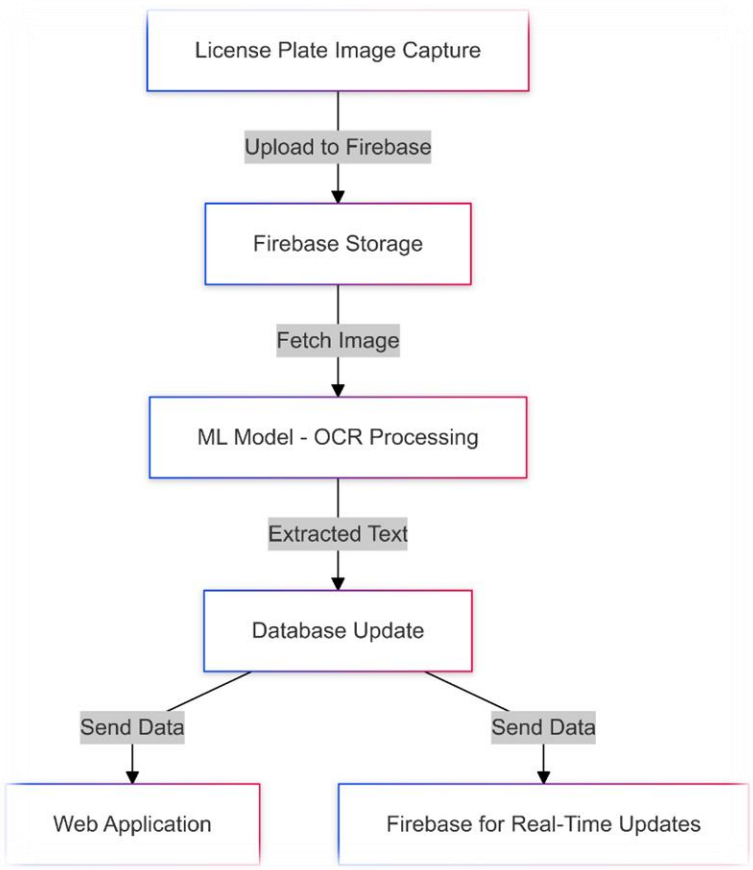
The mobile application is user-oriented, providing features like slot booking, payment processing, and real-time updates. It interacts with Firebase to fetch and send data, ensuring users have a seamless parking experience.



DFD1: Machine Learning Model Integration Subsystem

Description

The Machine Learning Model Integration subsystem is responsible for processing license plate images to extract text and update the system's database. This subsystem uses Optical Character Recognition (OCR) technology to analyze images captured by the IoT subsystem. Once processed, the extracted license plate numbers and related metadata are uploaded to Firebase and integrated with the database. This ensures real-time availability of vehicle information for the web and mobile applications, facilitating efficient garage management and accurate record-keeping.

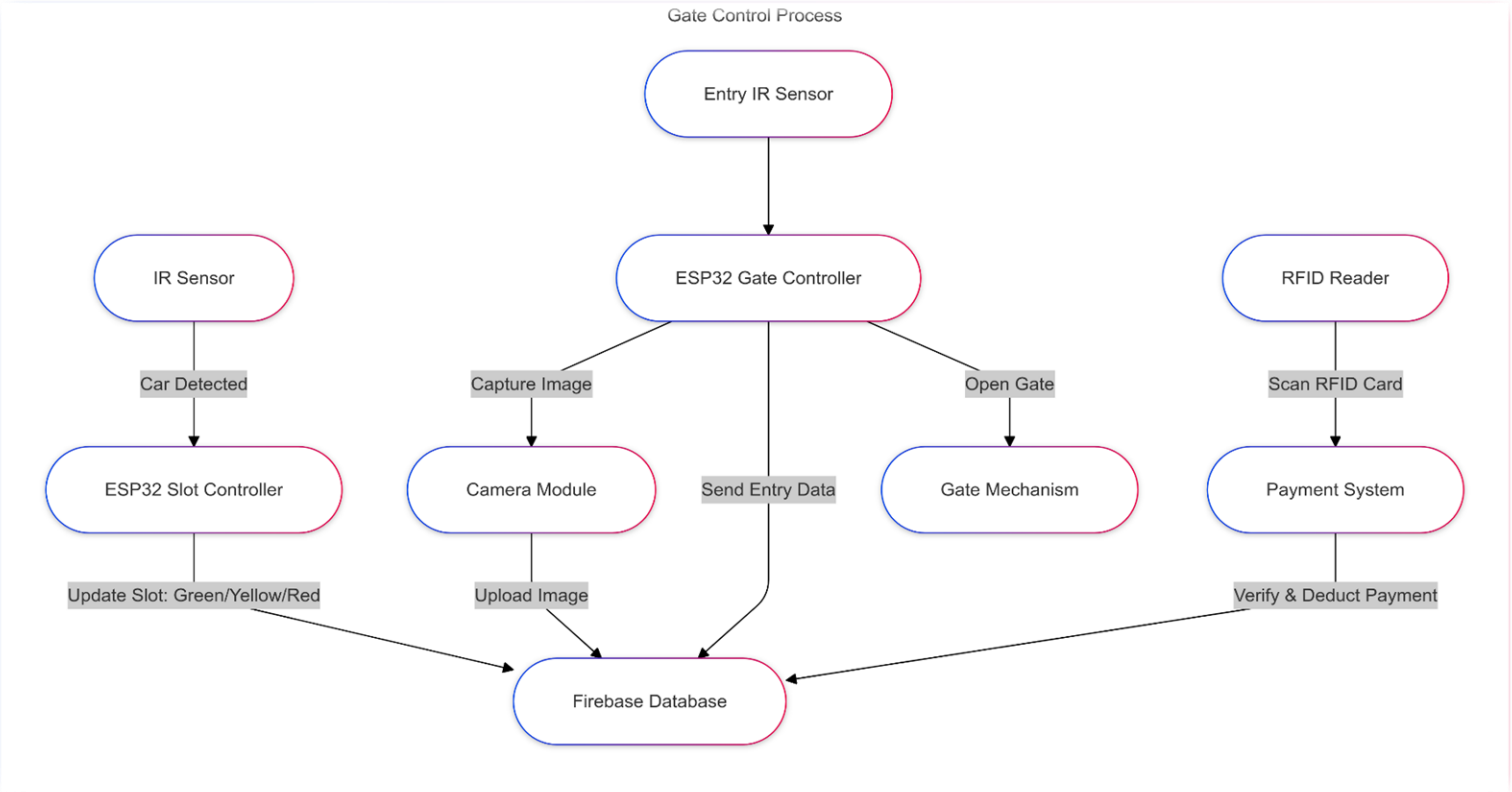


4.3.2.3 DFD Level 2: Subsystem Details

1. IoT Subsystem

The IoT subsystem automates hardware operations using ESP32 controllers.

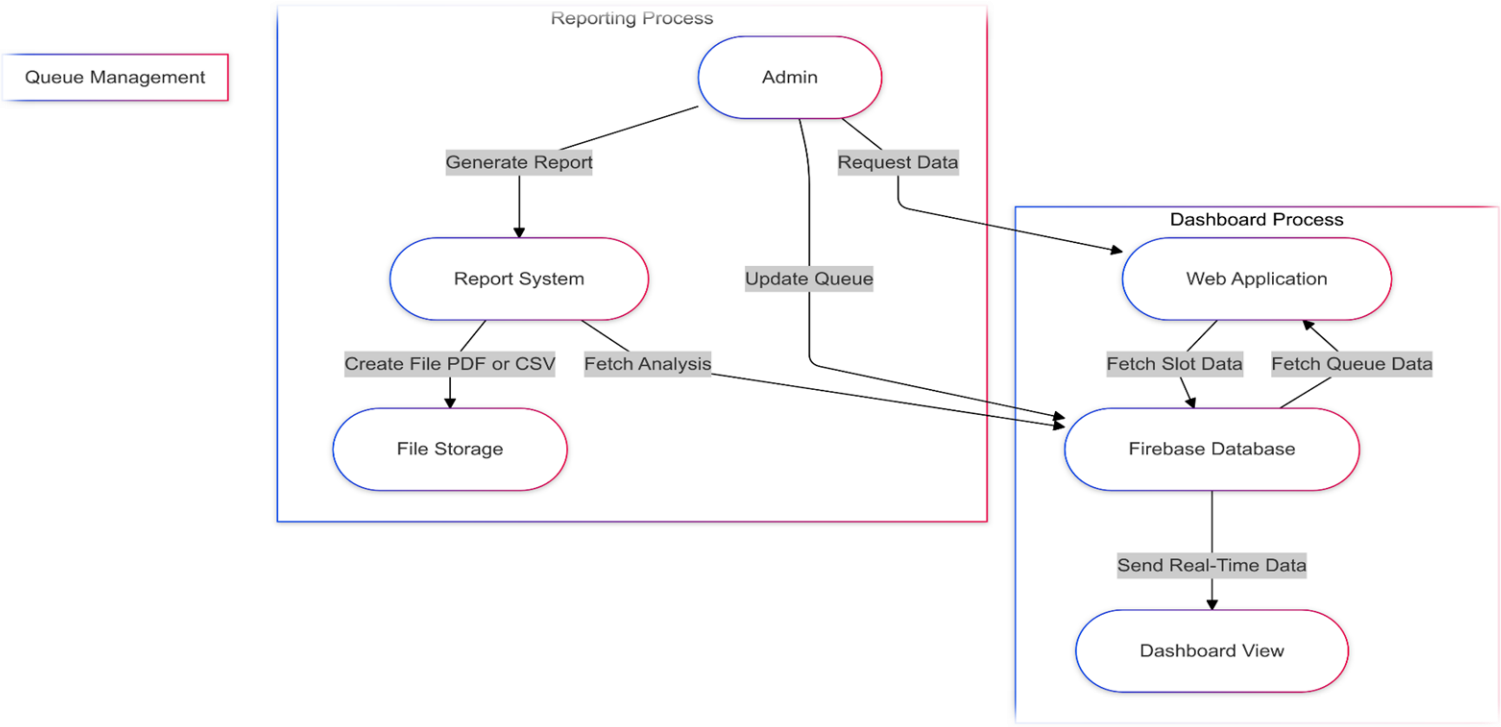
- **Slot Management:**
  IR sensors detect car presence and update slot status (green, yellow, red) via Firebase.
- **Gate Management:**
  Controls gates based on IR detection, RFID scans, and payment verification.
- **Image Processing:**
  Captures license plate images, uploads to Firebase, and forwards them to the machine learning model.



DFD Level 2: Web Application Subsystem

Details to Include:

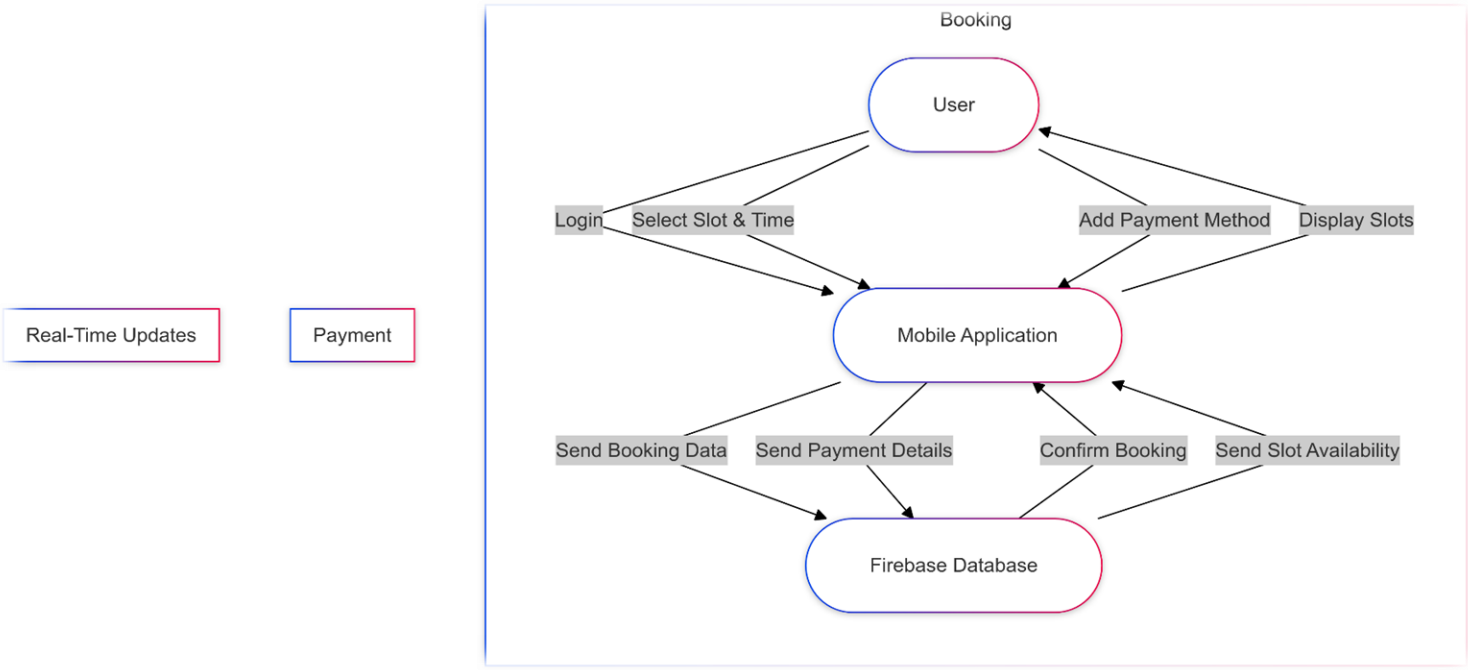
- Dashboard for admins showing data from Firebase.
- Reporting functionality (daily/monthly analysis, downloadable reports).
- Queue and slot management workflow.



DFD Level 2: Mobile Application Subsystem

Details to Include:

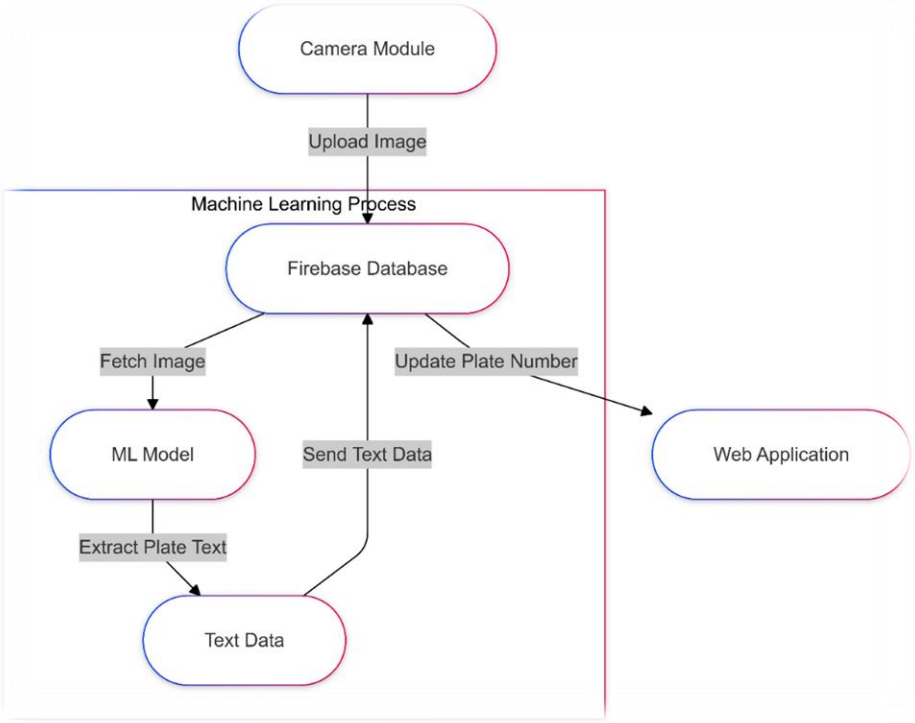
- Booking process: user selects slot, time, and payment.
- Real-time updates on slot availability.
- Communication between mobile app and Firebase.



DFD Level 2: Machine Learning Model Integration

Details to Include:

- How images are processed through the model.
- The workflow for extracting text and updating Firebase.

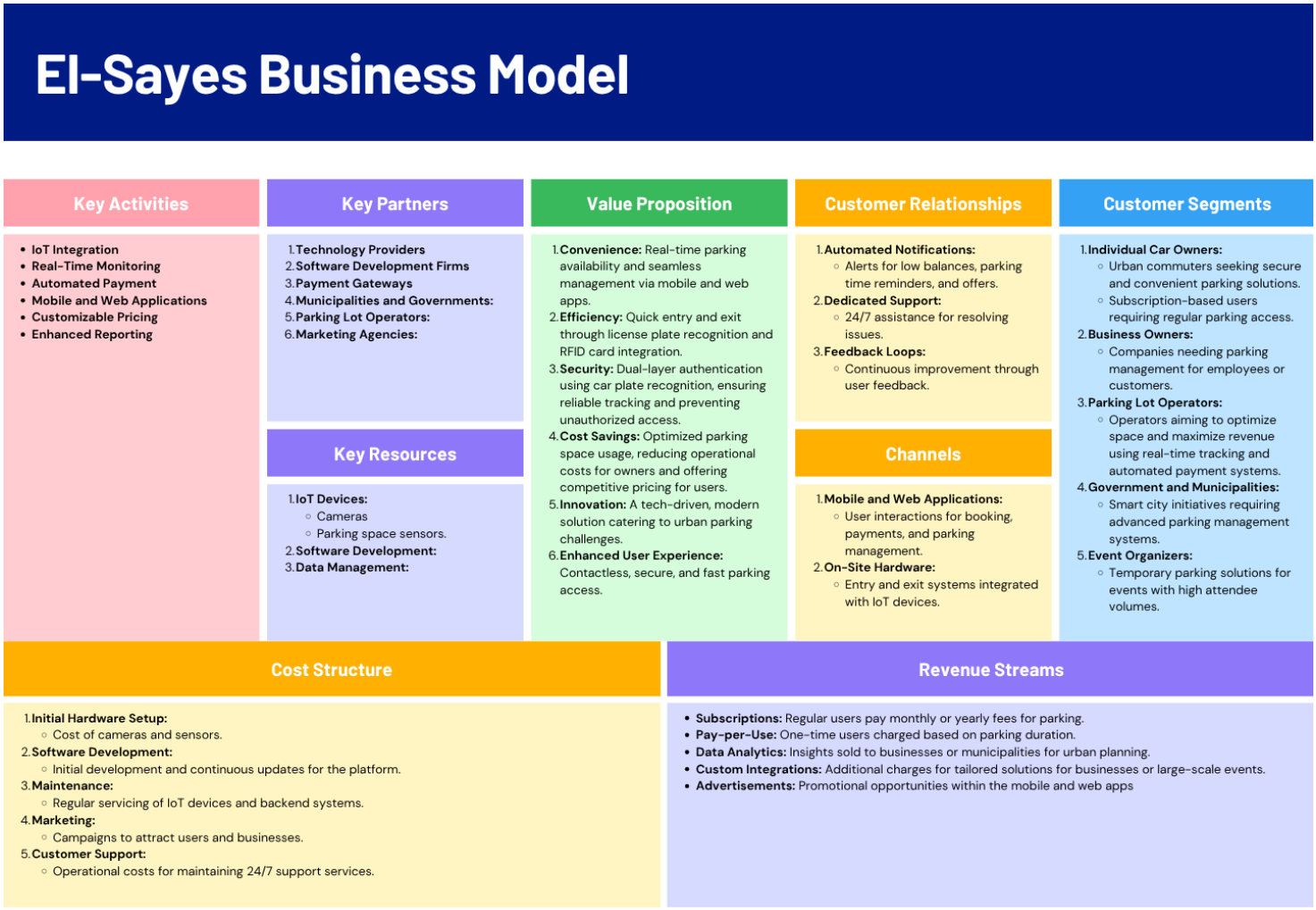


Chapter Five

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.

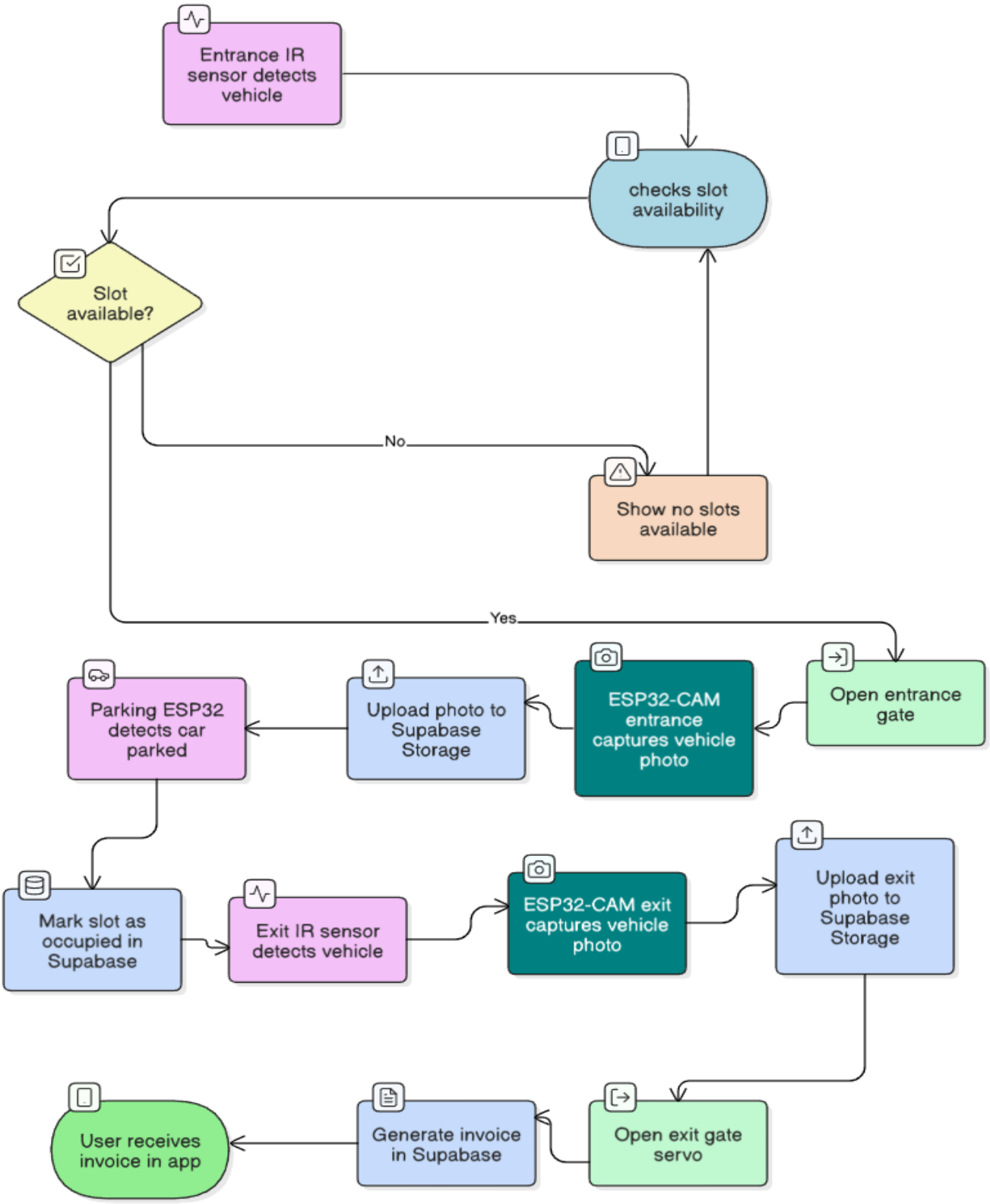
6.2.2. System Architecture

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.

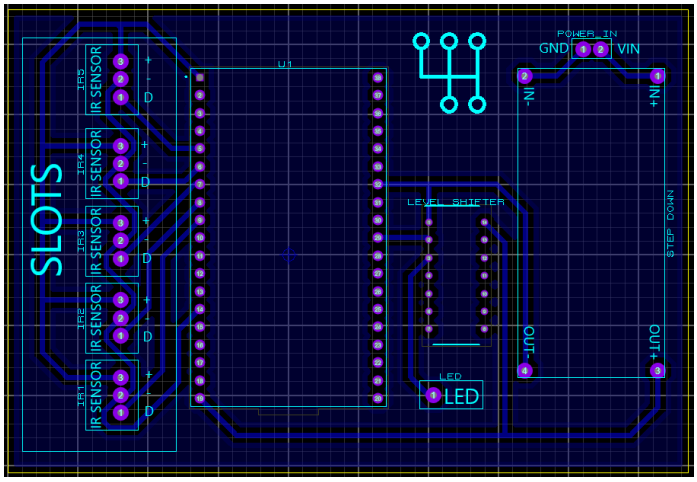
Communication Protocols

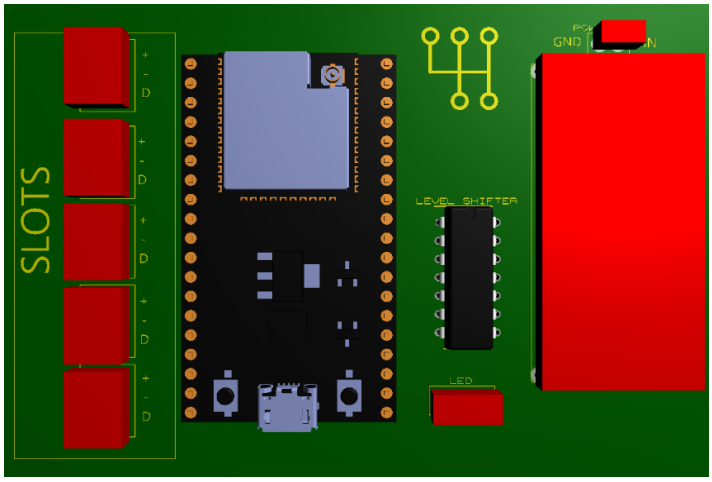
- **Wi-Fi:** Facilitates real-time data exchange with **Supabase**.
- **HTTP (REST**): Used to send real-time PATCH requests for updating the is_occupied status.

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.

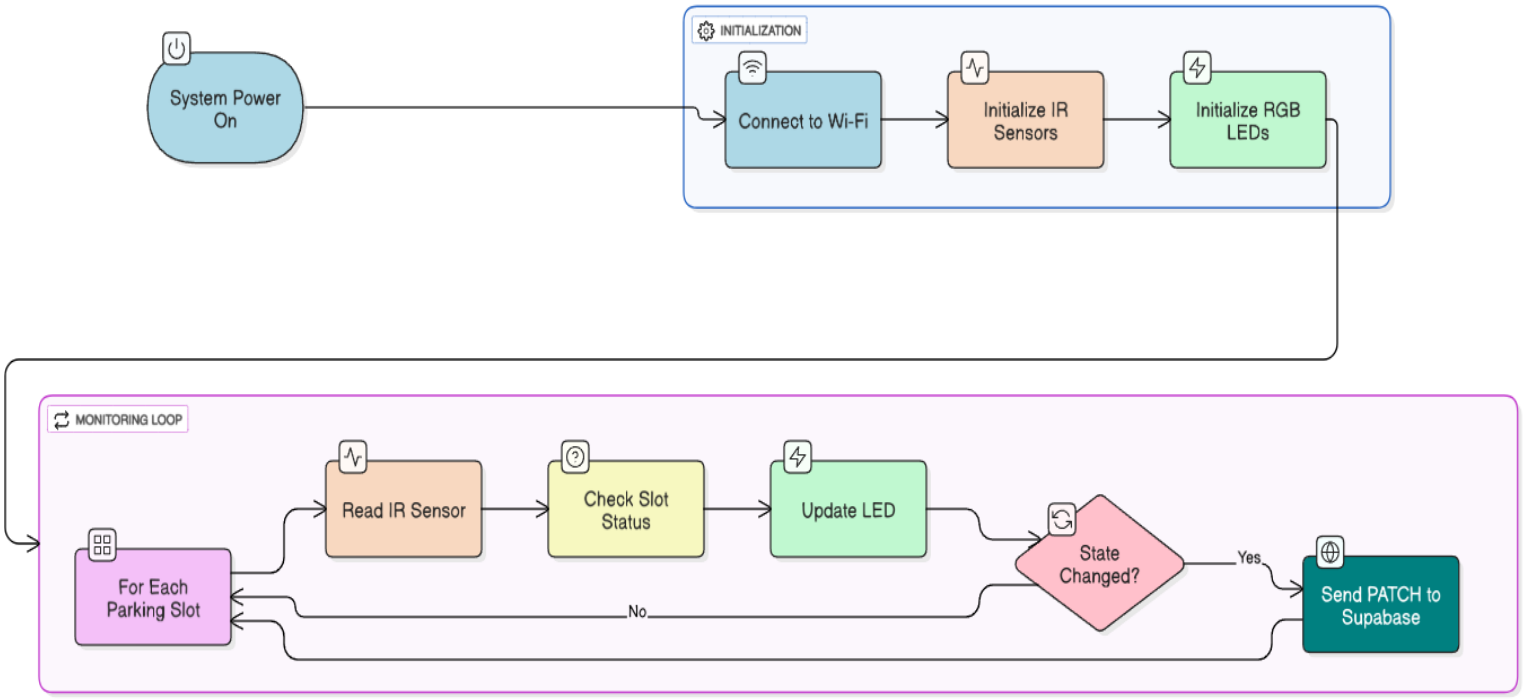
6.2.3.3 Data Flow and Communication

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

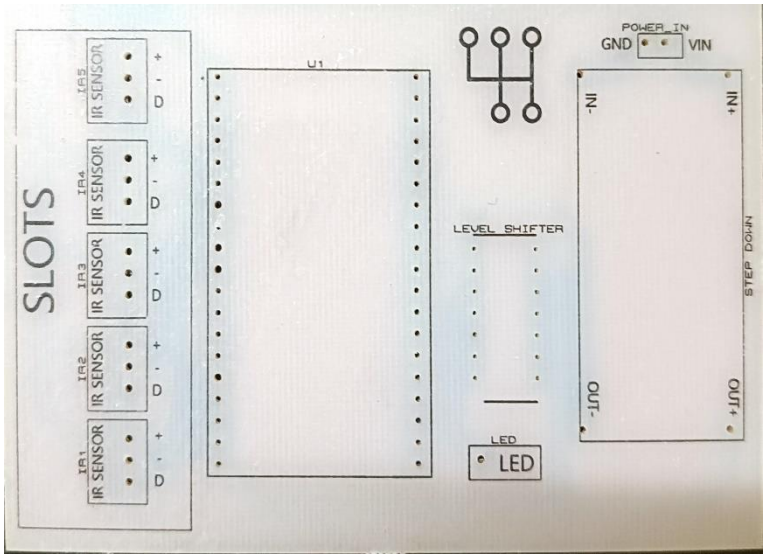
Challenges

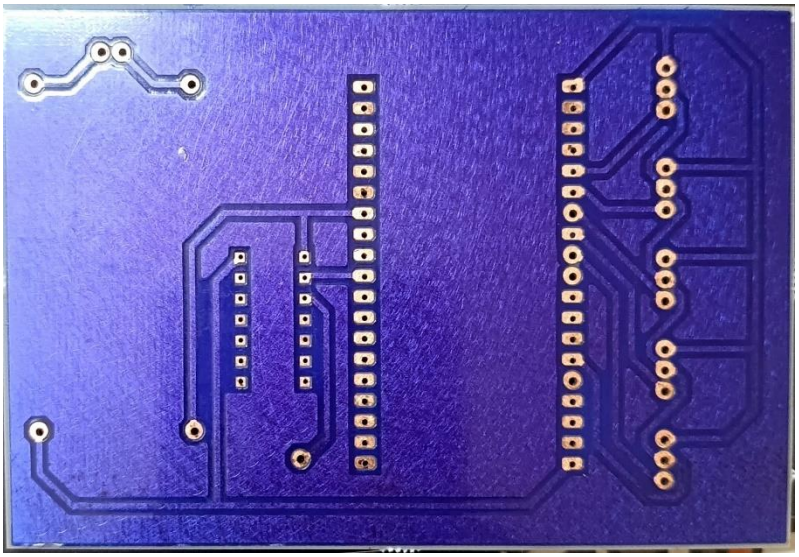
1. **Signal Interference:** May affect sensor communication.
2. **Voltage Regulation:** Critical for stable LED operation.
3. **Hardware Costs:** Need for cost-effective sourcing.

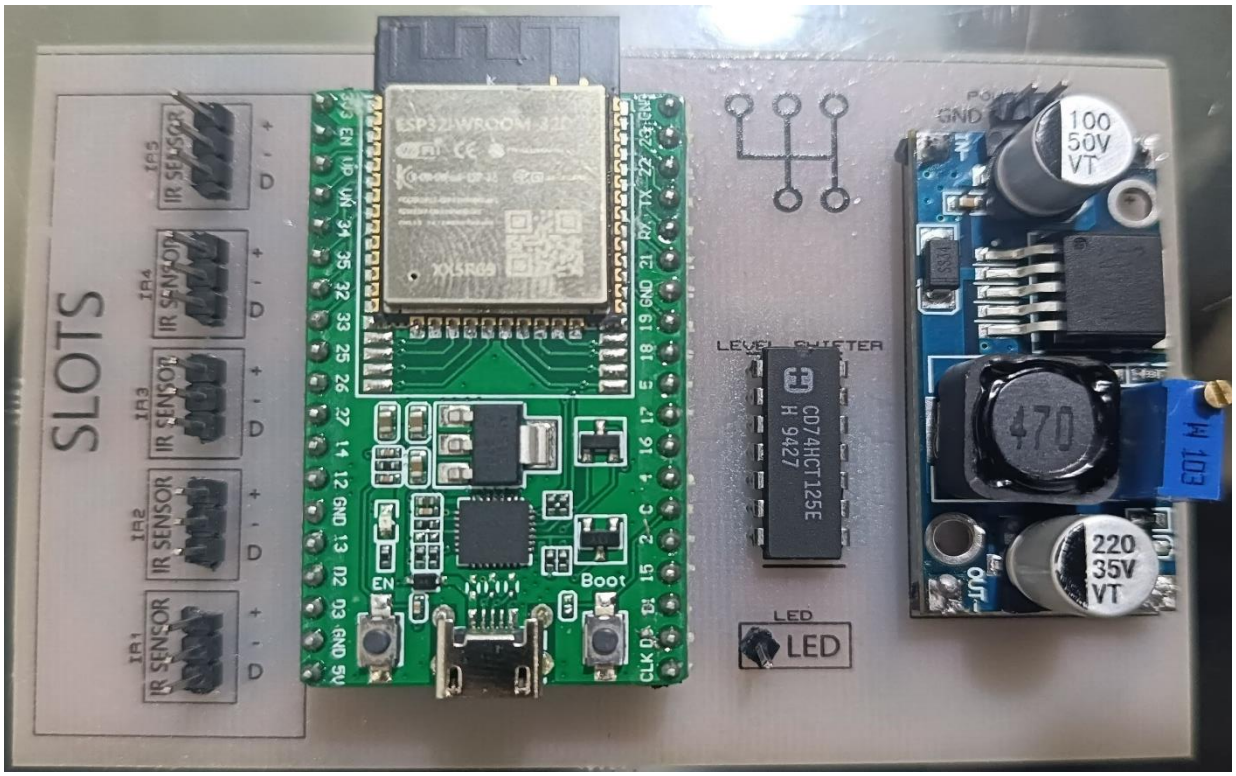
Mitigation Strategies

- **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.

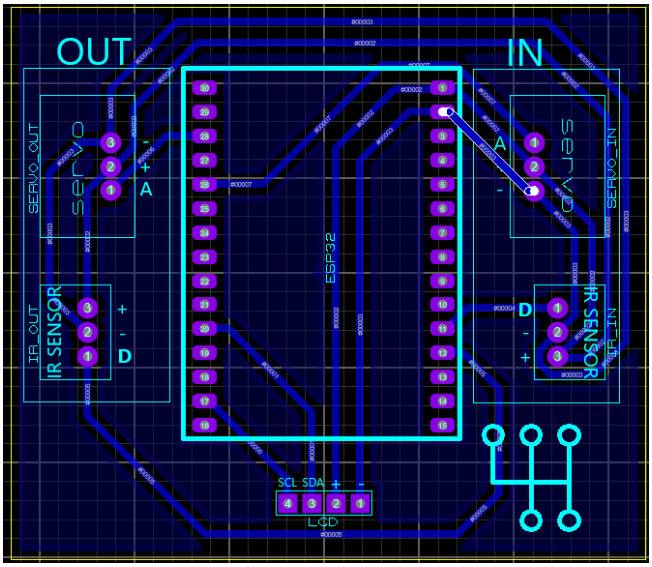
Communication Protocols

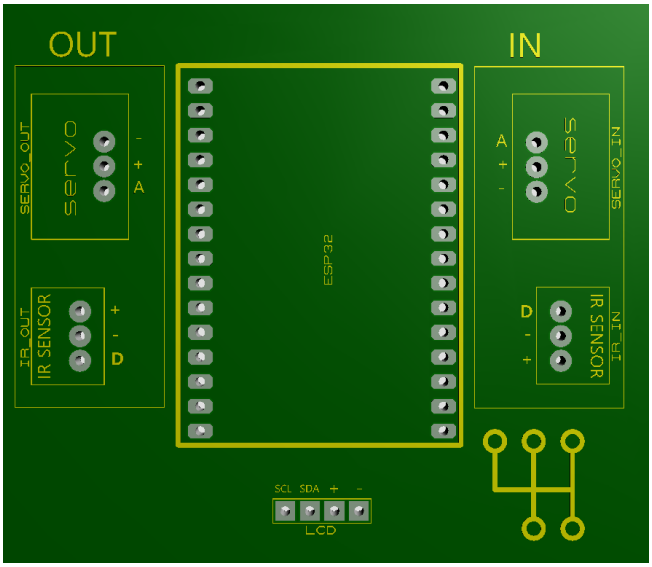
- **Wi-Fi:** Connects all ESP32 modules to the internet for backend interaction.
- **HTTP (REST):** Used to send/receive real-time data from Supabase (e.g., camera triggers, slot checks, entry logs).

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

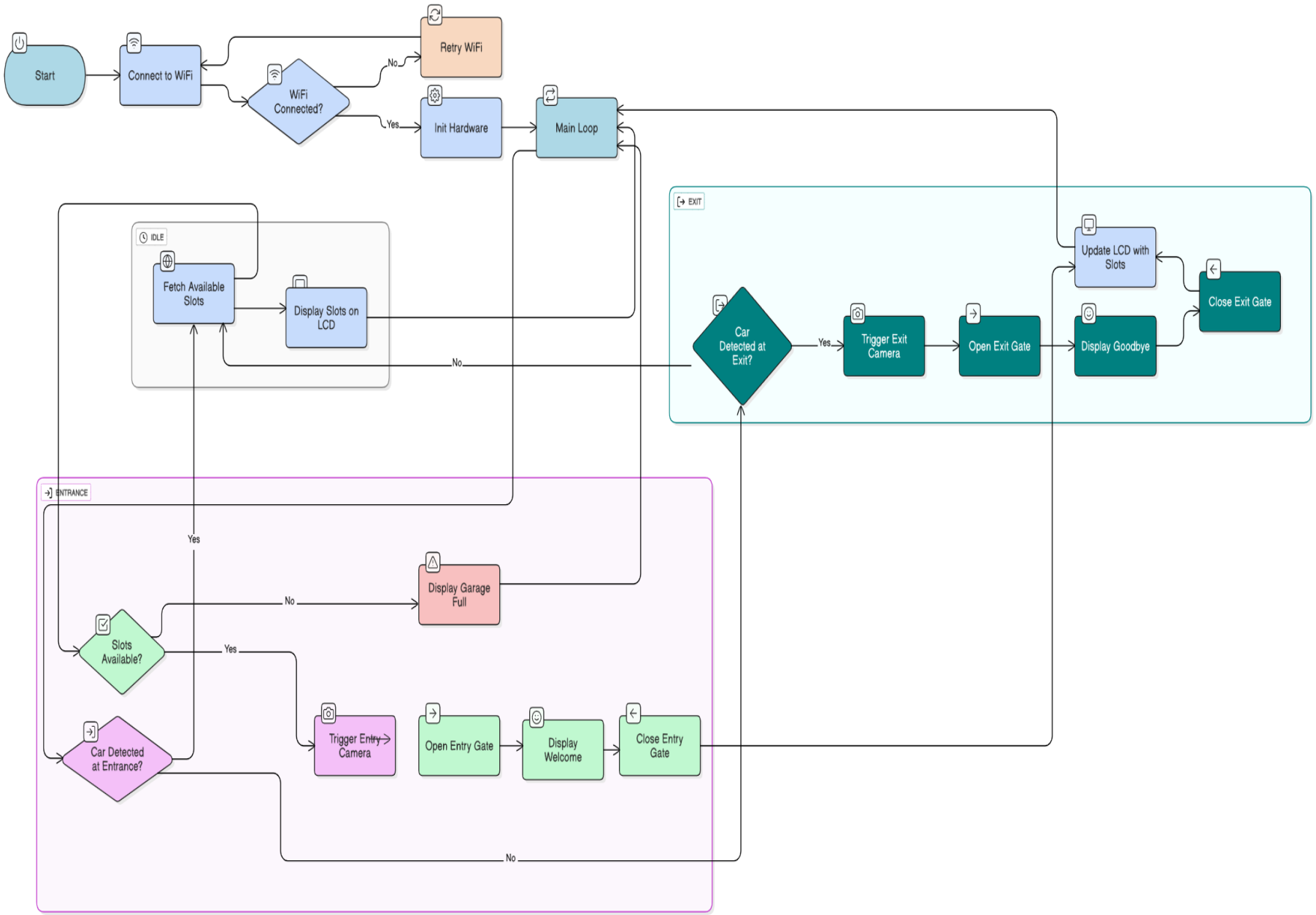
6.2.4.3 Data Flow and Communication

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

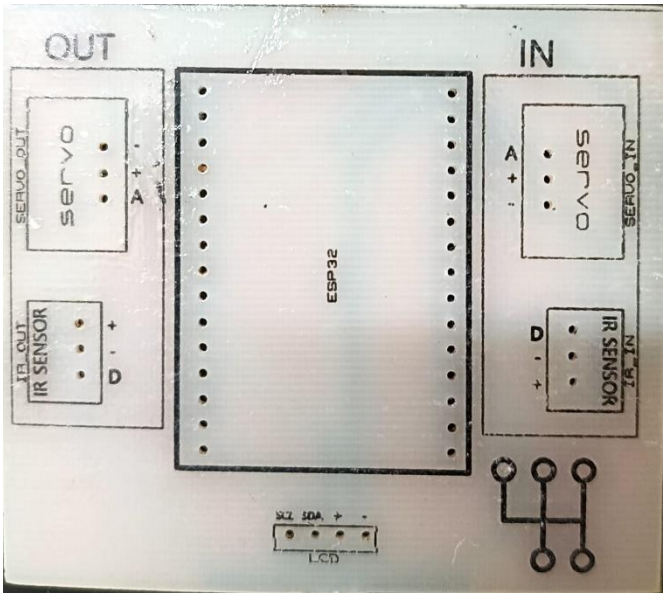
Challenges

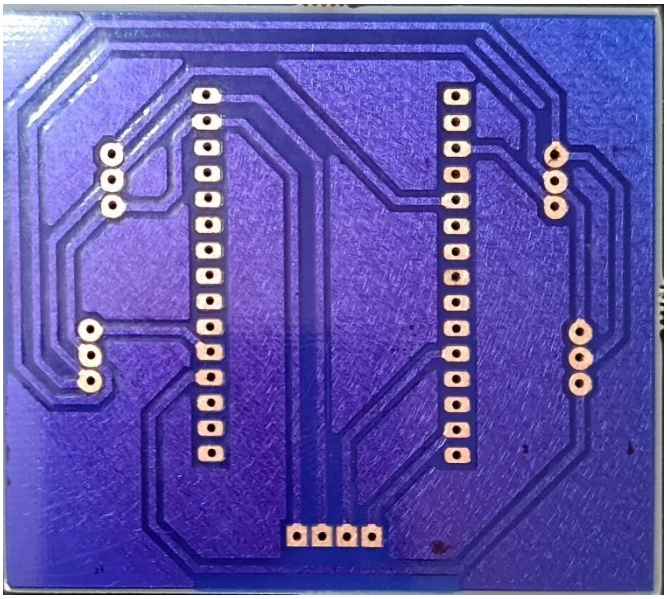
- **IR Sensor Accuracy:** Risk of false detection due to environmental factors (sunlight, shadows, etc.).
- **Servo Motor Durability:** Potential wear and tear from frequent gate operations.
- **Camera Reliability:** Lighting conditions may affect license plate image clarity.
- **Wi-Fi Stability:** Network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

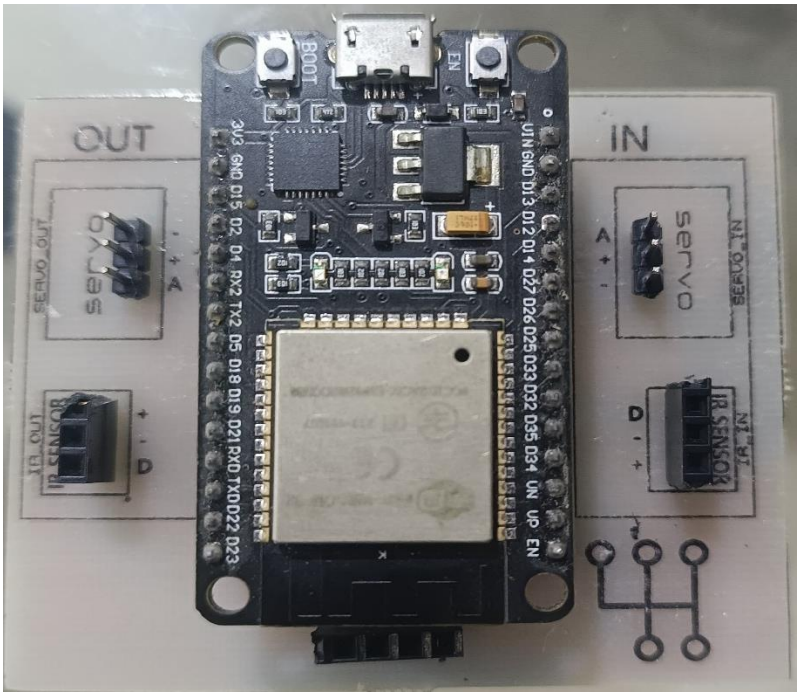
Mitigation Strategies

- **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

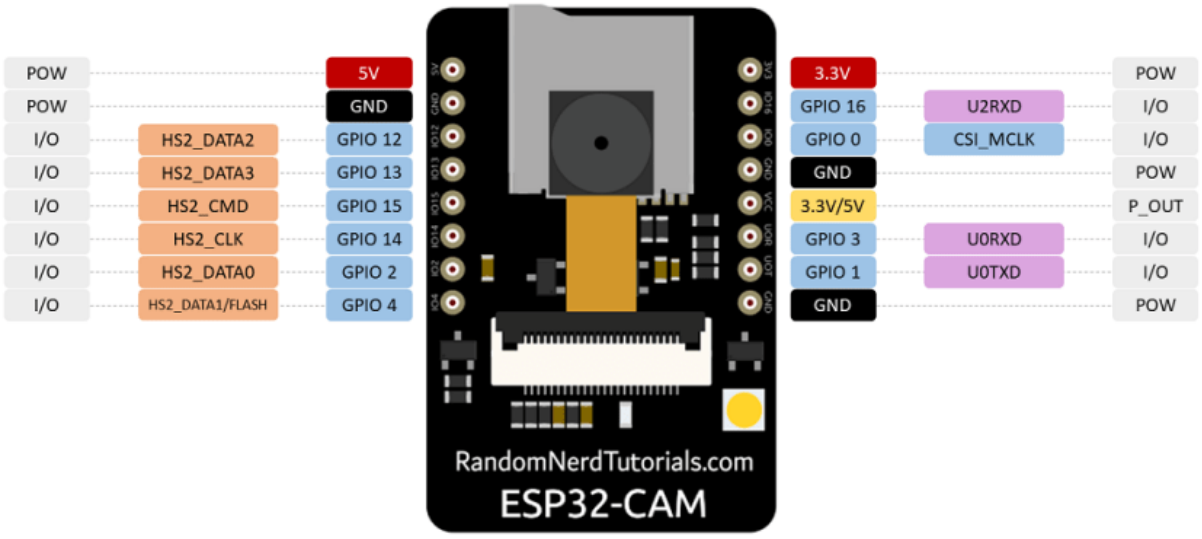
- **ESP32-CAM Module:** Equipped with an OV2640 camera sensor for capturing images in JPEG format.
- **Wi-Fi Connectivity:** Connects the ESP32-CAM to the internet for uploading captured images to Supabase.
- **Supabase Storage:** Stores uploaded photos with timestamped filenames for easy identification.
- **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

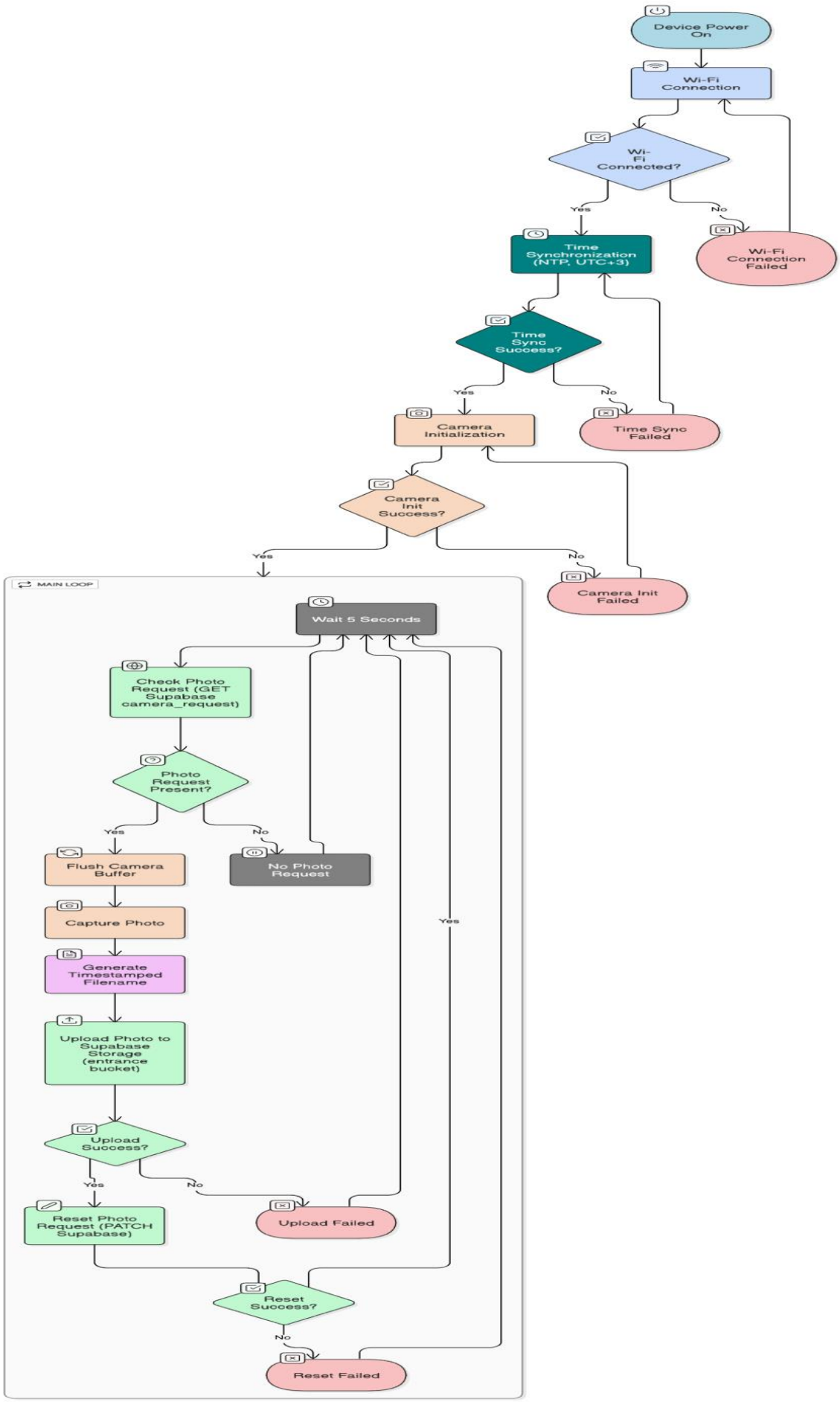
- After a successful upload, the ESP32-CAM resets the camera_request flag to false to avoid duplicate captures.
- Simultaneously, a backend OCR service monitors the Storage bucket:
    - When a new image is detected, the image is downloaded and processed.
    - The extracted plate number is inserted 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);
```

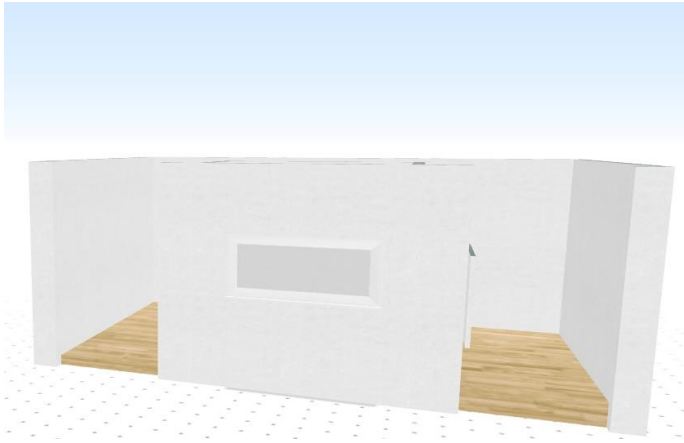
- Uploads the JPEG image to a predefined Supabase bucket (entrance or exit) using a RESTful POST request.
- File name includes a timestamp for easy tracking and retrieval.

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

The Machine Learning (ML) phase of our **Smart Garage System** project, focusing on the development of a **License Plate Recognition (LPR)** system using **YOLOv8** for license plate detection and **OCR (Optical Character Recognition)** for character extraction. Our goal is to create a robust and accurate system capable of detecting and recognizing Egyptian license plates in real-time. The phase involves **dataset preparation**, **YOLO model training**, **OCR integration**, and **real-time deployment**. We chose the **Egyptian Cars Plates Dataset** from Kaggle because it is highly suitable for this project due to its relevance and diversity. This report provides a detailed explanation of our workflow, tools, and methodologies, along with justifications for the choices we made.

6.3.2 Introduction

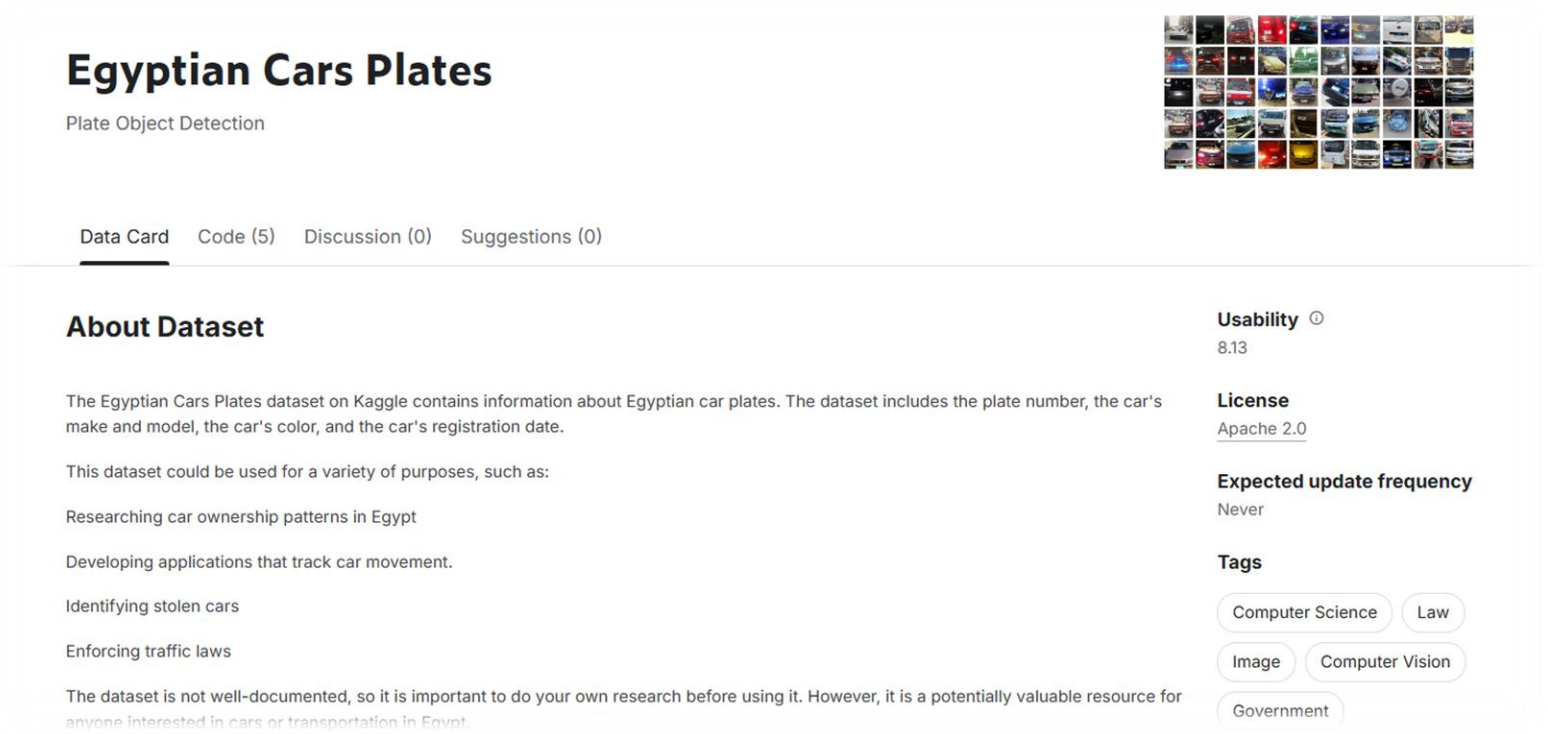
The **Smart Garage System** aims to provide a seamless and efficient solution for managing parking spaces, monitoring vehicle entry/exit, and offering real-time updates to users. A critical component of this system is the **License Plate Recognition (LPR)** module, which uses machine learning to automatically 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**.

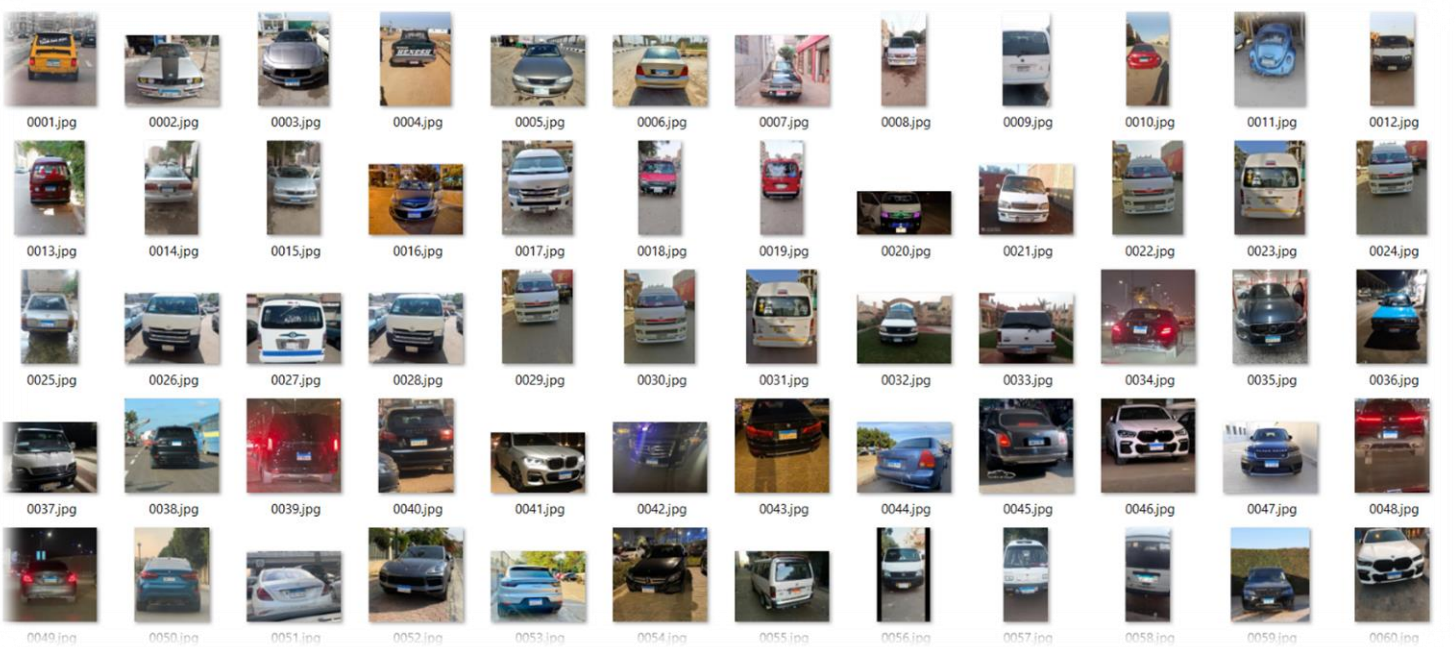
6.3.3 Dataset Selection

We chose the **Egyptian Cars Plates Dataset**

(https://www.kaggle.com/datasets/mahmoudeldebase/egyptian-cars-plates/data**)** from Kaggle for this project due to the following reasons:



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 Why YOLOv8?

We selected YOLOv8 (You Only Look Once version 8) 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.

6.3.5 Overall Flow of Data in the Project

The LPR system follows a structured workflow to process data from input to integration:

1. **Data Input:**
    - Images or video feed from the IoT camera system (real-time or pre-recorded).
    - Input resolution and orientation are standardized.

2. **License Plate Detection (YOLO):**
    - YOLO detects the bounding boxes of license plates in each frame.
    - Outputs:
        - Bounding box coordinates.
        - Cropped images of license plates.
3. **Plate Preprocessing:**
    - Cropped license plates undergo preprocessing:
        - Resizing to standard dimensions.
        - Removing noise, adjusting contrast, and handling rotation.
4. **Character Recognition (OCR):**
    - OCR processes the preprocessed plate images.
    - Outputs: Text (license plate number) extracted from the plate.
5. **Post-Processing:**
    - Validate and correct the extracted text (e.g., using regex or a predefined plate format).
    - Handle real-time errors (e.g., unclear images or incomplete recognition).
6. **Integration into IoT System:**
    - Combine predictions with the video feed and send real-time alerts or updates.
    - Store recognized plates and corresponding metadata (e.g., timestamp, car entry/exit) in a database.

6.3.6 Phases Breakdown

6.3.6.1 Phase 1: Dataset Preparation and Annotation

Objective: Prepare a high-quality dataset for training the YOLO model.

Tasks:

1. **Collect Data:**
    - Source images/videos of Egyptian license plates from the Kaggle dataset.
    - Ensure diversity in lighting conditions, plate angles, and plate types.
2. **Annotate Dataset:**
    - Use tools like **LabelImg** or **Roboflow** to label license plates with bounding boxes.
    - Export annotations in YOLO format (a .txt file for each image containing the bounding box coordinates and class).
3. **Standardize Images:**
    - Resize images to a consistent resolution (e.g., 200x200).
    - Rotate and orient images correctly.
    - Apply data augmentation techniques (e.g., flipping, blurring, brightness/contrast adjustments).

Deliverables:

- A fully annotated and augmented dataset in YOLO format.
- Standardized images ready for training.

6.3.6.2 Phase 2: Training YOLO for License Plate Detection

Objective: Train a YOLO model for license plate detection.

Tasks:

1. **Set Up YOLO Framework:**
    - Install YOLOv8 using Ultralytics.
    - Configure the environment on a local machine or cloud platform (e.g., Google Colab).
2. **Train YOLO Model:**
    - Load the dataset and configure the data.yaml file (define dataset paths, classes, etc.).
    - Train the YOLO model using pre-trained weights (e.g., COCO weights) for transfer learning.
    - Monitor training metrics like precision, recall, and mAP (mean average precision).
3. **Evaluate Model:**
    - Test the trained model on a validation dataset (images not used in training).
    - Ensure the model detects license plates accurately under various conditions.

Deliverables:

- Trained YOLO model with saved weights (.pt file).
- Evaluation report with metrics (mAP, precision, recall).

6.3.6.3 Phase 3: OCR Integration and Character Recognition

Objective: Integrate OCR for character recognition.

Tasks:

1. **Set Up OCR Framework:**
    - Install and configure **EasyOCR** or **Tesseract** for character recognition.
    - Test OCR on sample cropped license plate images to verify setup.
2. **Preprocess Cropped Plates:**
    - Automate the pipeline for cropping plates detected by YOLO.
    - Preprocess cropped plates to enhance contrast, remove noise, and standardize size.
3. **Combine YOLO + OCR Pipeline:**
    - Integrate YOLO's bounding box output with the OCR framework.
    - Automate the pipeline: Plate detection → Plate cropping → OCR processing.

Deliverables:

- Functional YOLO + OCR pipeline.
- Verified OCR outputs for various test images.

6.3.6.4 Phase 4: Model Integration and Real-Time Testing

Objective: Deploy the YOLO + OCR model for real-time testing.

Tasks:

1. **Deploy the YOLO + OCR Model:**
    - Deploy the trained YOLO model and OCR pipeline into the IoT system.
    - Use OpenCV to process real-time video feeds.
2. **Implement Real-Time Feedback:**
    - Display detected plates and recognized text on the video feed.
    - Add a confidence score for each prediction (e.g., confidence > 0.8).

3. **Test Under Real-World Conditions:**
    - Test the system in diverse environments (e.g., different lighting, moving vehicles).
    - Add post-processing rules to handle OCR errors (e.g., regex for validating plate formats).
4. **Store Results:**
    - Save predictions (plate numbers, timestamp, confidence) in a database for further analysis.

Deliverables:

- Fully integrated system with real-time video processing.
- Testing report detailing performance under real-world conditions.
- Final deployment-ready YOLO + OCR pipeline.

Tools Used in Each Step

Step	Tools
Annotation	LabelImg, Roboflow
YOLO Training	YOLOv8 (Ultralytics), PyTorch, Jypyter Notebook
OCR Framework	EasyOCR, Tesseract
Preprocessing	OpenCV, PIL (Python Imaging Library)
Integration and Testing	OpenCV for real-time video, Python for pipeline automation
Deployment	Flask/FastAPI for APIs (optional), Firebase for IoT database

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.

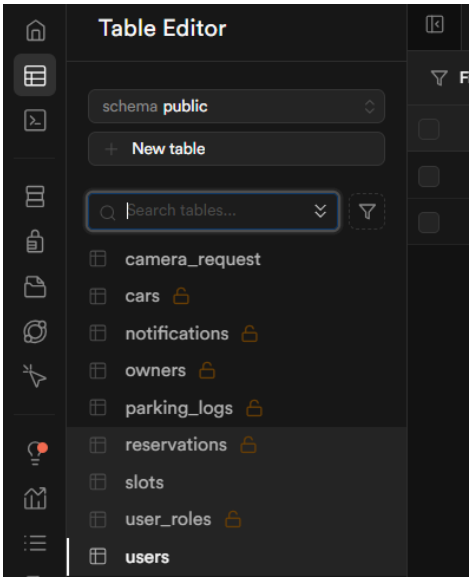
6.4.2 Why Supabase?

Supabase was chosen because:

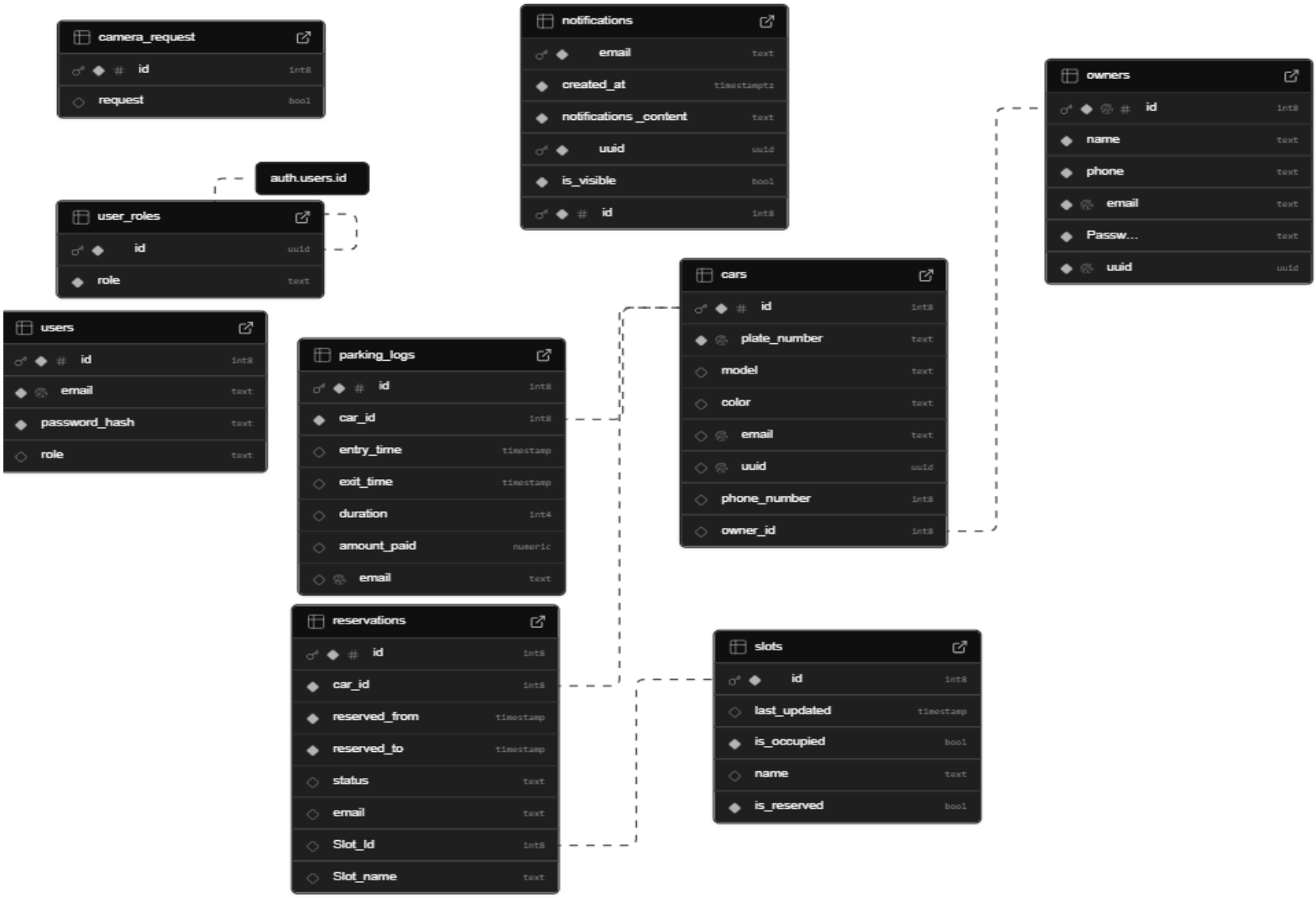
- It is built on **PostgreSQL**, which is powerful, scalable, and ACID-compliant.
- It provides **RESTful APIs** and **Realtime** features out-of-the-box.
- It integrates easily with IoT devices (via HTTP requests) and frontend apps (via SDKs).
- Offers **Supabase Storage** to manage images from the 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:

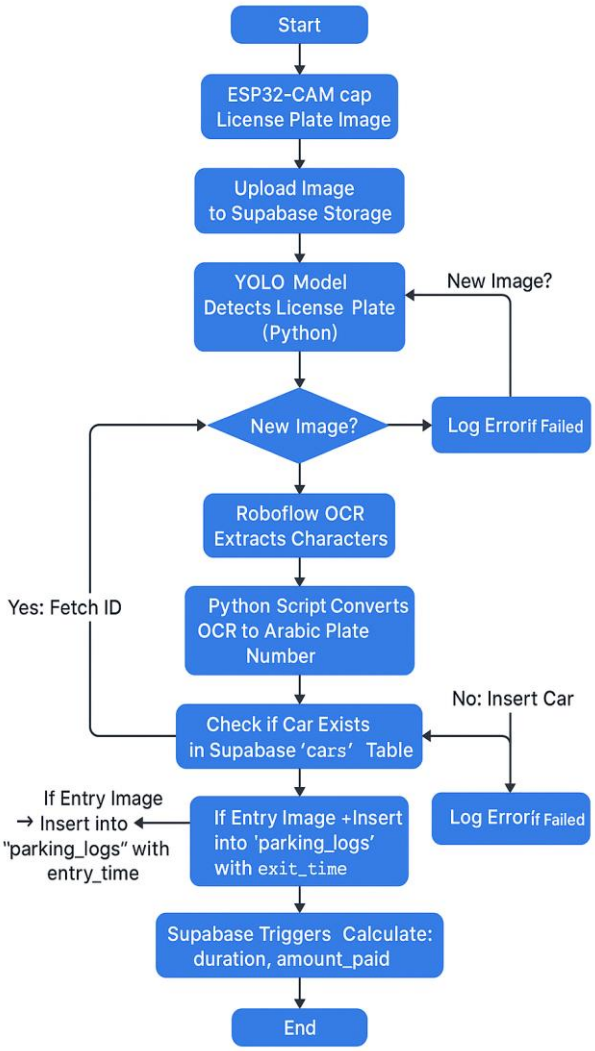
```
CREATE TRIGGER trg_calculate_duration
BEFORE UPDATE ON parking_logs
FOR EACH ROW
EXECUTE FUNCTION calculate_parking_duration();

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

Integration with IoT and Applications

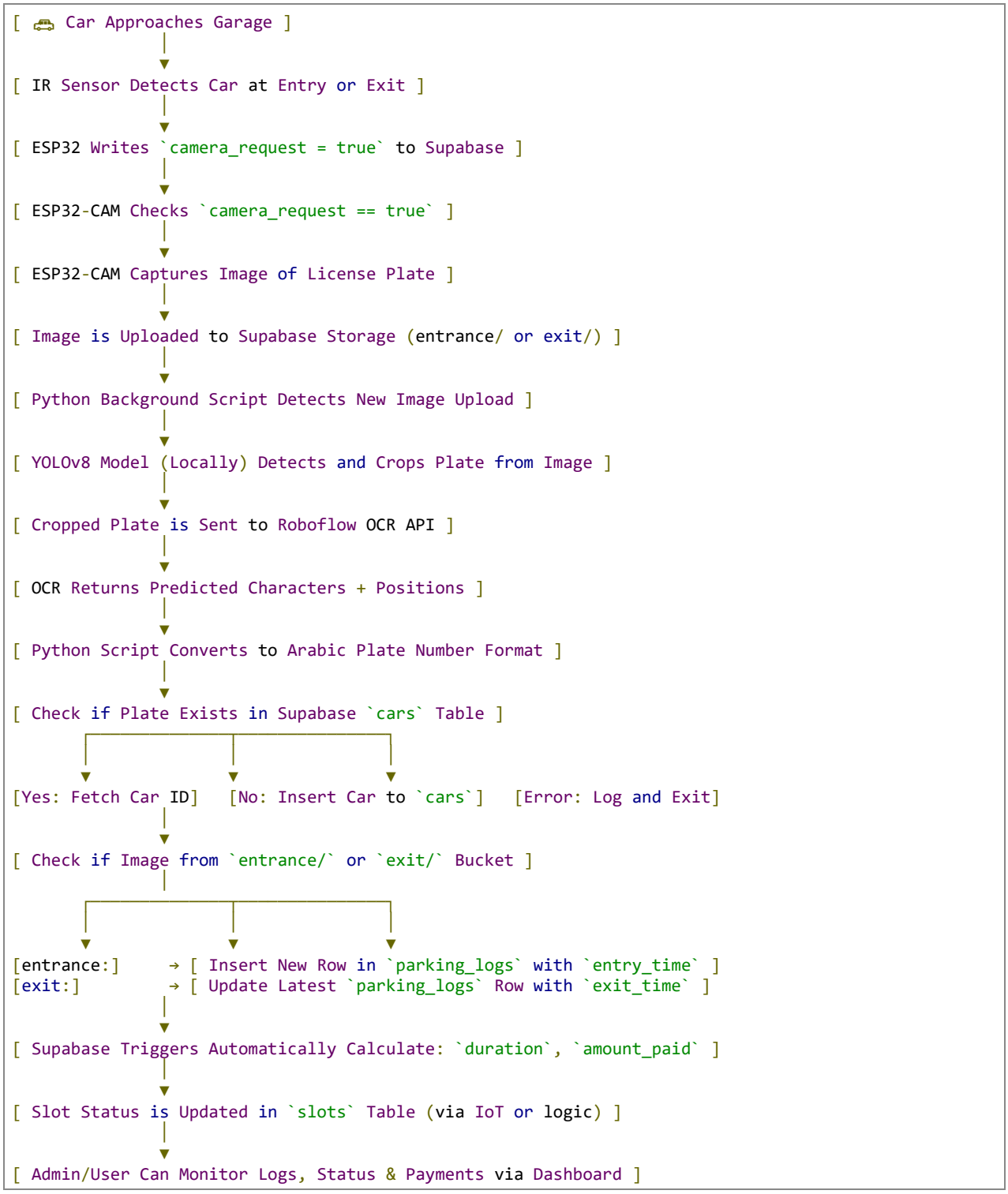
Device/Module	How it interacts with DB
ESP32	Updates slots.is_occupied, sends camera_request
ESP32-CAM	Monitors camera_request → uploads image to Supabase Storage
Python/Streamlit	Sends reservation data, displays slot status
Supabase Functions	Used for Edge Functions (image OCR integration)

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**.

6.4 UI / UX Design

6.4.1 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.

6.4.2 Importance of UI/UX in Modern Applications

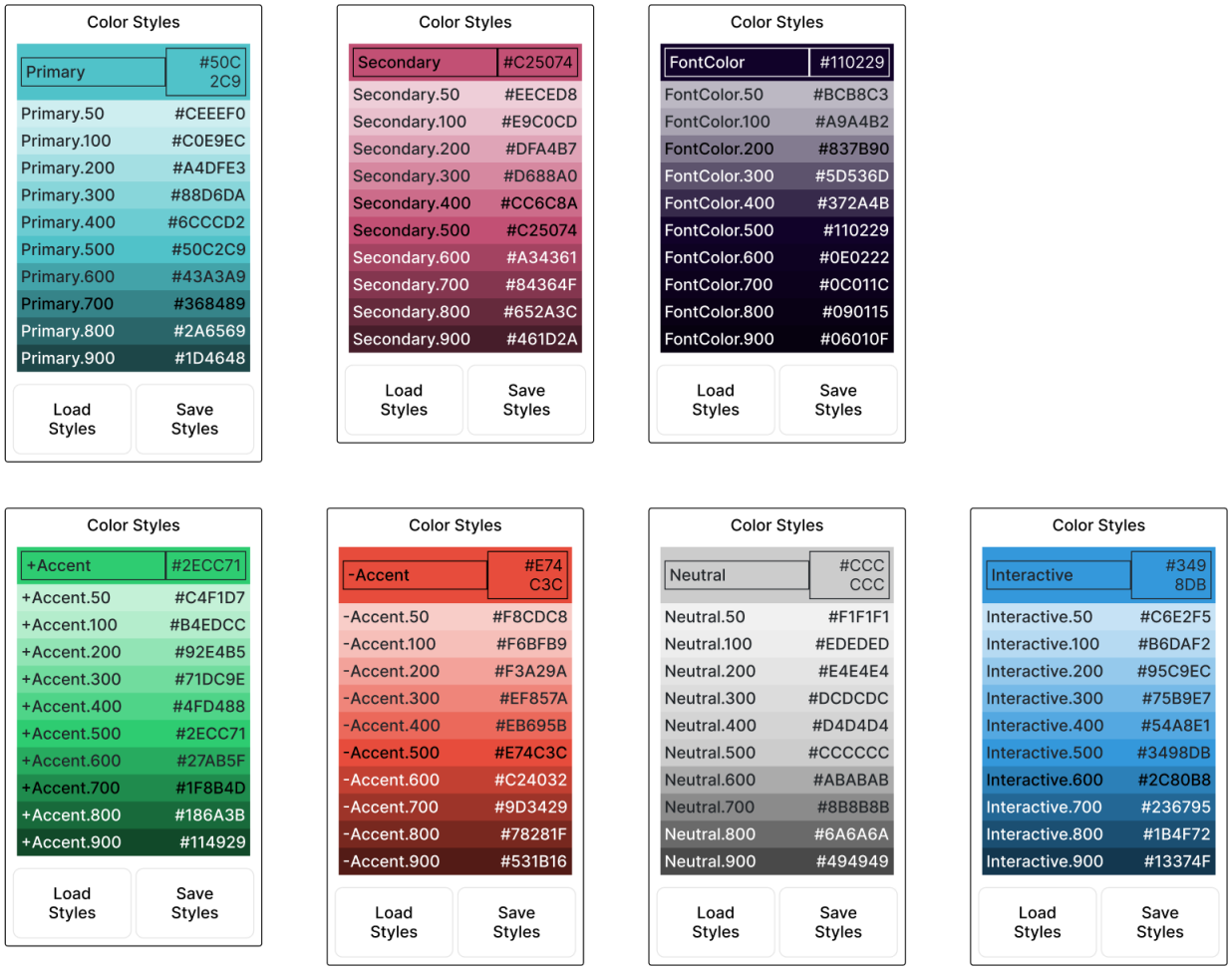
- The importance of UI/UX design cannot be overstated in the development of modern applications. As technology evolves, users expect intuitive and visually engaging experiences. A well-designed interface not only enhances usability but also fosters user satisfaction and retention. In our project, UI/UX 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.

6.4.3 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.

6.4.4 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.

HEADING O

HEADING 1

HEADING 2

Heading 3

Heading 4

Paragraph Regular

Paragraph Bold

Buttons

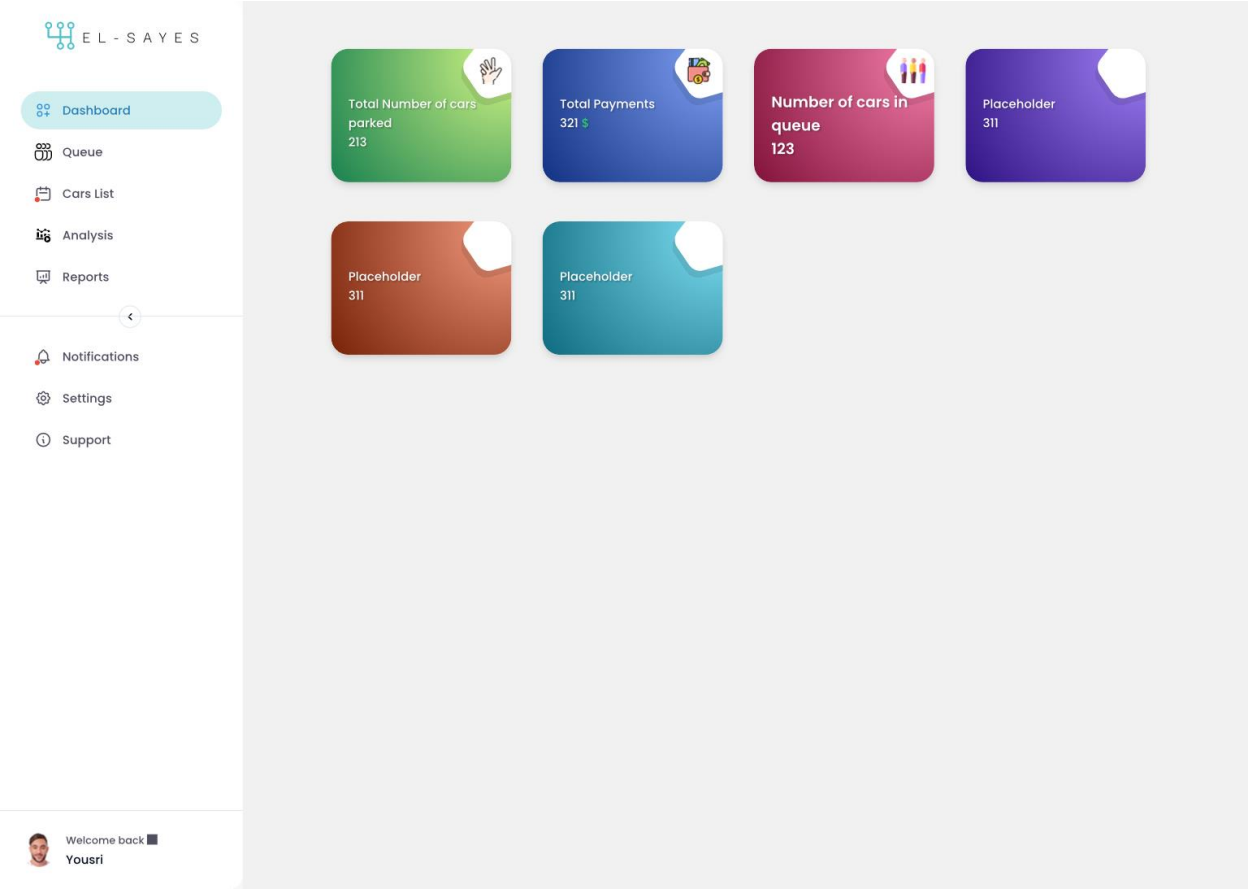
Icon Label

- 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.

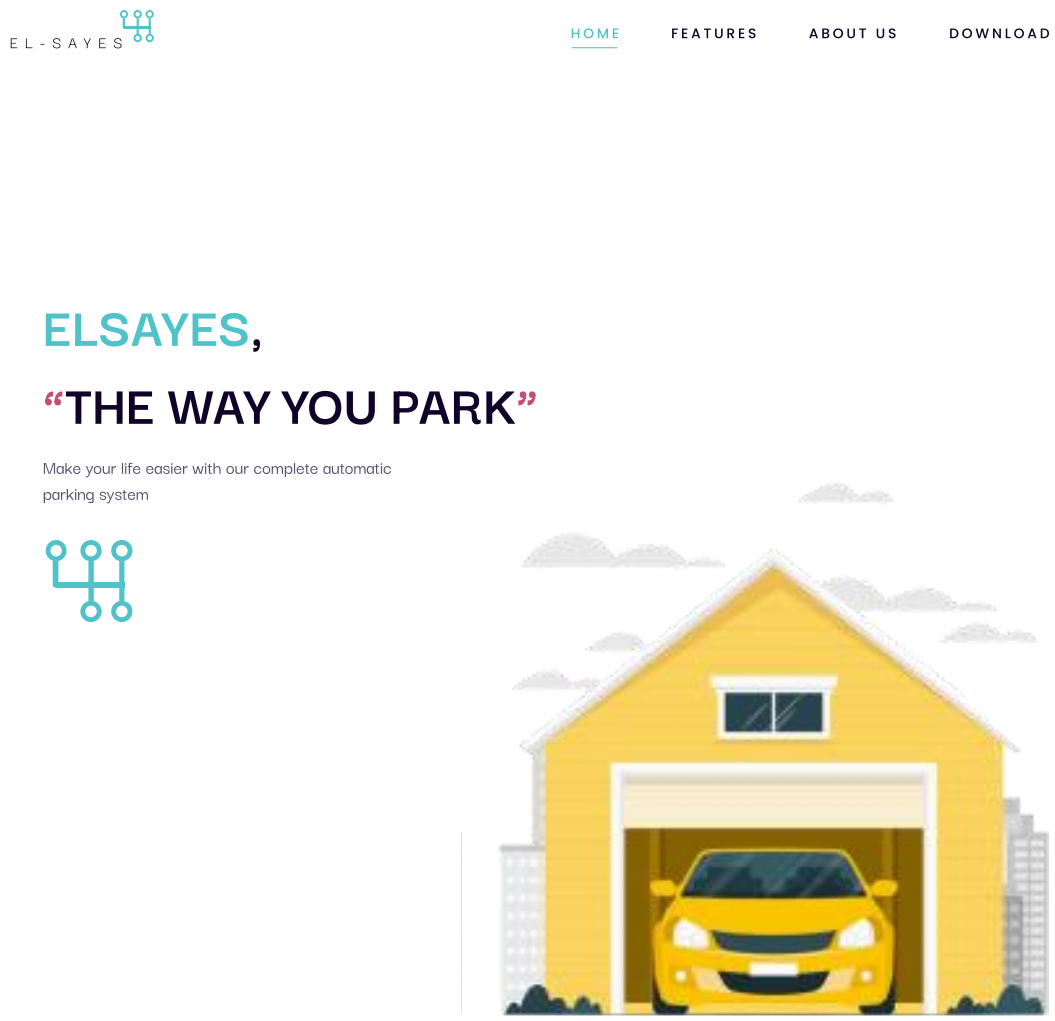
EL-SAYES

6.4.5 Design Process for Web Application

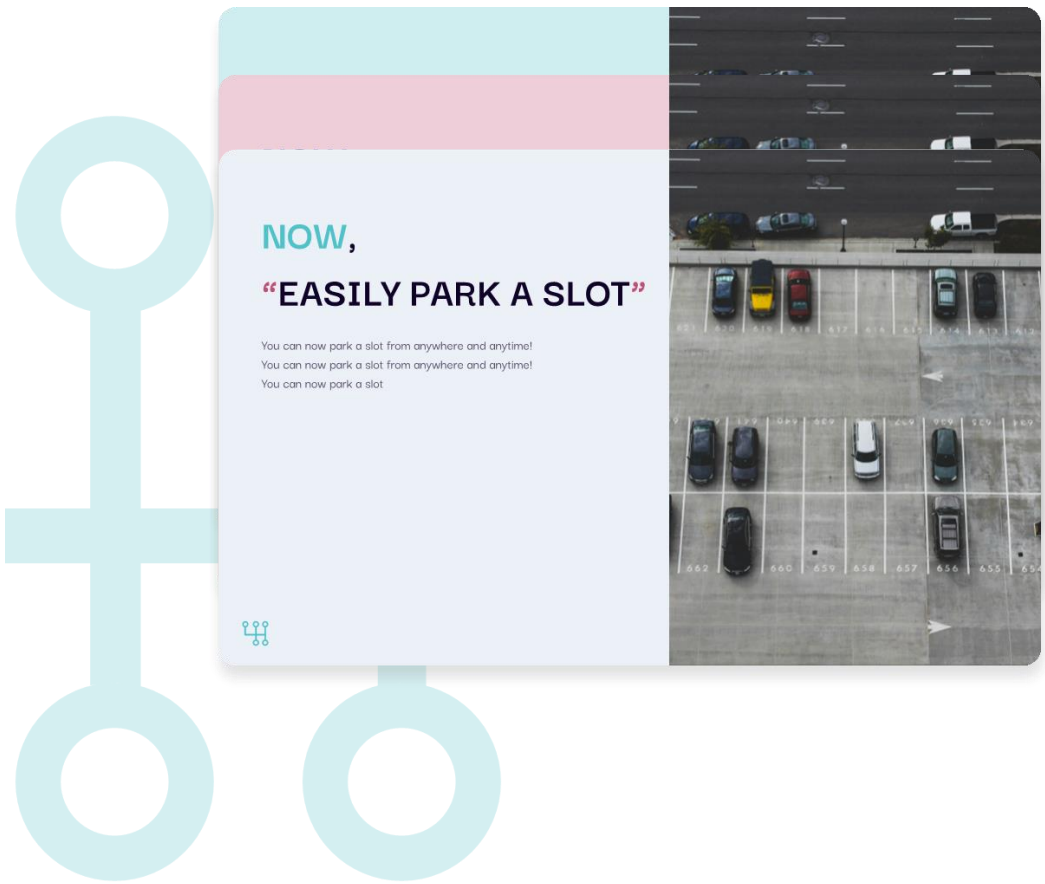
- After establishing the identity for our application, we designed an intuitive dashboard tailored for the admin to effectively monitor and manage all operations within the garage. The dashboard provides a comprehensive overview, including key analytics and management tools. Admins can access a dedicated 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.

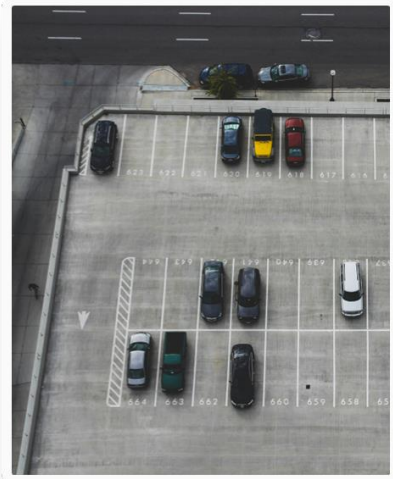


- An About Us section to introduce ourselves to the users and provide insight into the purpose and vision behind our application. This section highlights our mission, the challenges we aim to solve, and the dedication of our team in creating a solution tailored to meet user needs. By sharing our story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

W h o W e A r e

WHO WE ARE

Established in 2025, ELSAYES has been dedicated to revolutionizing parking solutions. Our journey began with a simple idea: to make parking seamless and stress-free. Today, we continue to deliver innovative solutions that turn challenges into convenience.

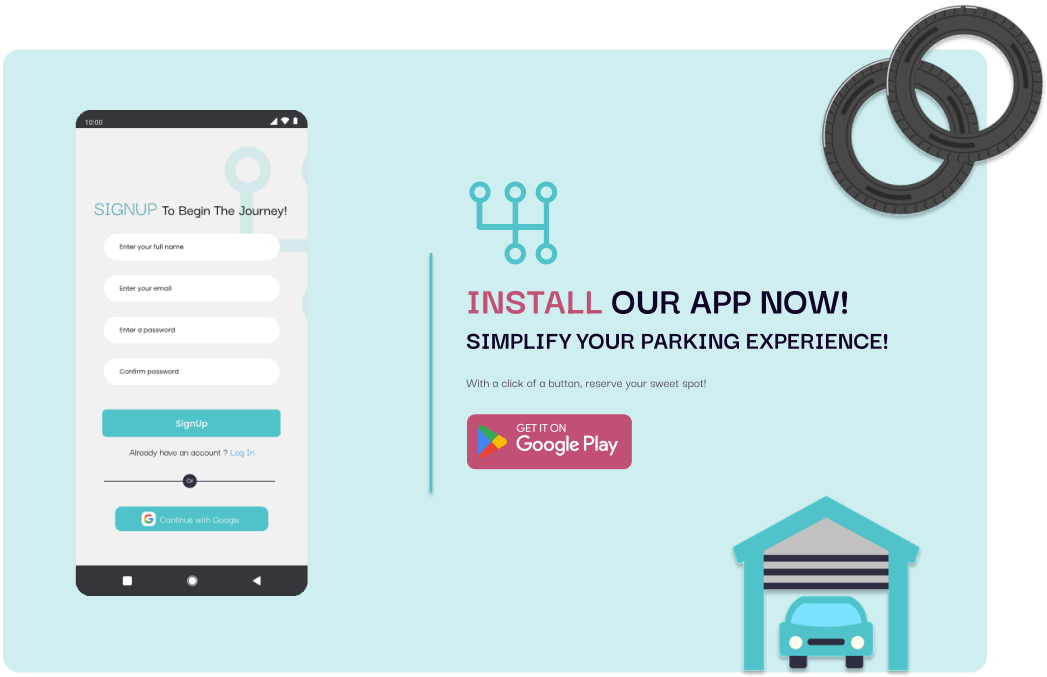


W h y U s

WHY US

What makes us unique is our unwavering commitment to excellence. We're not just an automatic parking garage; we're your reliable parking solution. Discover why drivers like you trust us to simplify their parking experience.

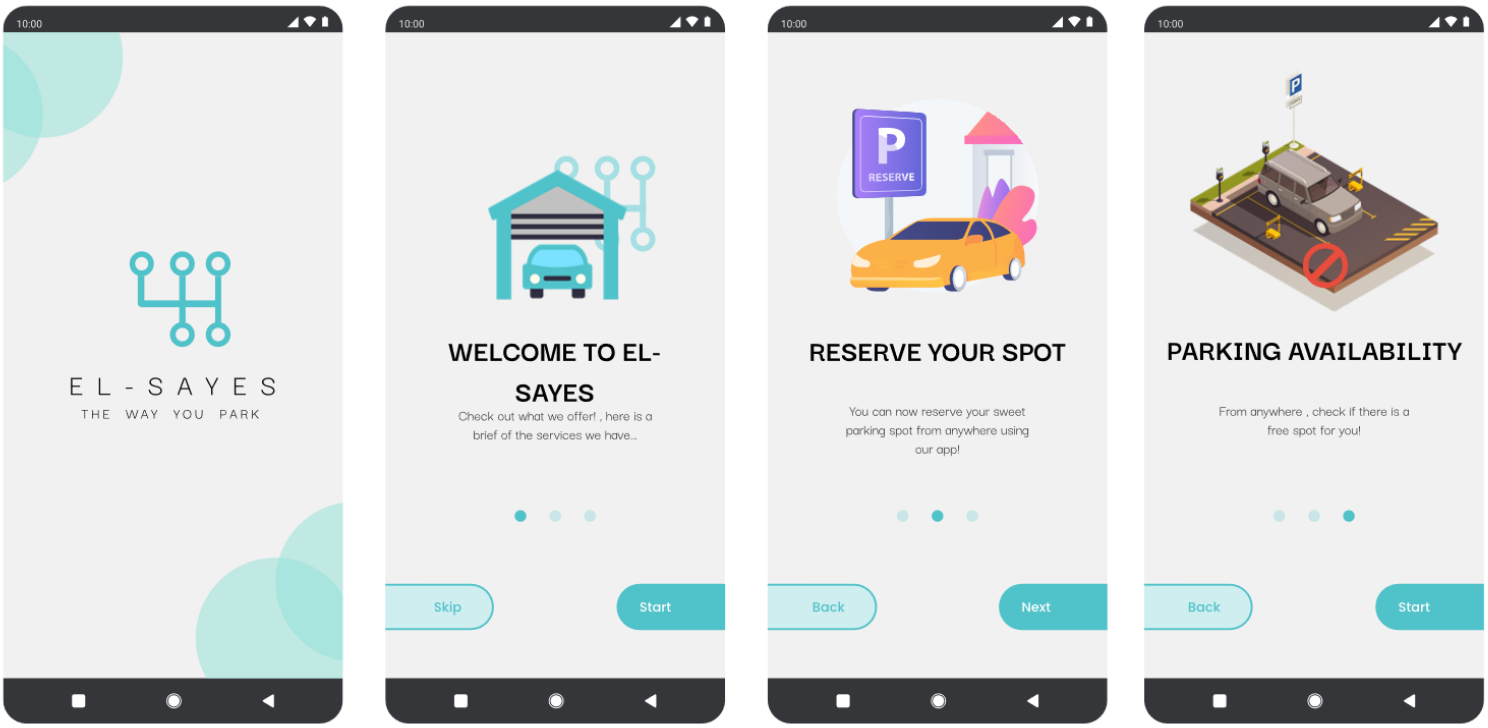
- We also designed a Download Section to guide users seamlessly when they want to download our application. This section features clear and prominent call-to-action buttons, allowing users to download the app for their preferred platform, such as iOS or Android. It includes a brief description of the 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.



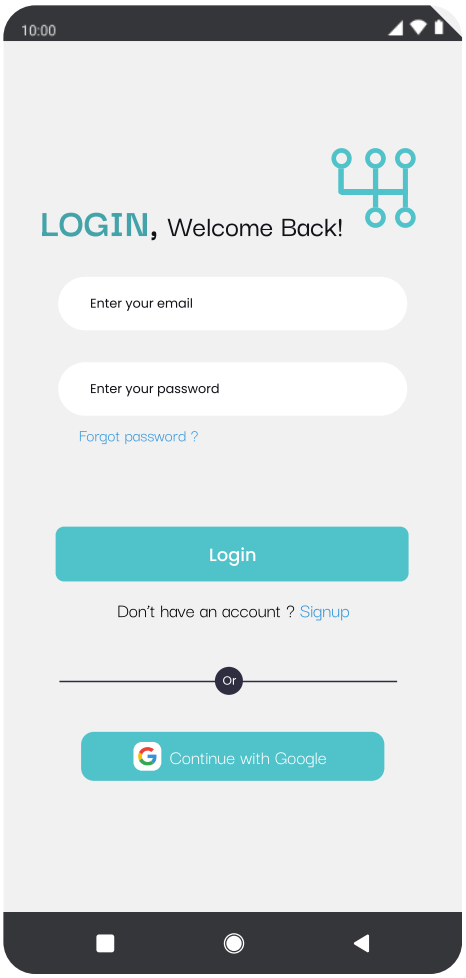
- The footer features a minimalist design with the "EL-SAYES" logo and name on the left, reinforcing branding. On the right, it provides links to Instagram, Facebook, and Twitter, encouraging users to connect with us for updates and engagement. Its clean layout ensures simplicity and usability.

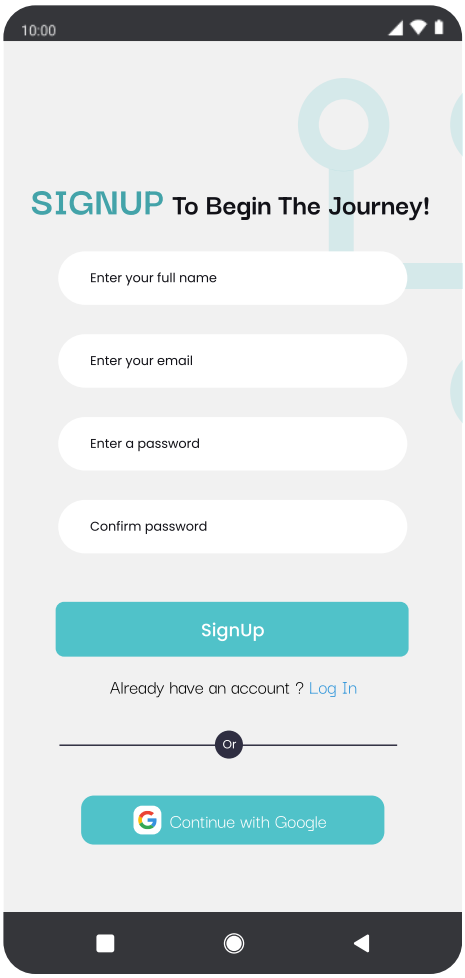
6.4.6 Design Process for Mobile Application

- 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.





6.5 Flutter Development

6.5.1 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.

6.5.2 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.

6.5.3 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.

6.5.4 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.

6.5.5 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: Firebase for real-time database and authentication
Payment Integration: Stripe/PayPal SDKs
Navigation: Google Maps API for GPS and indoor navigation

6.5.6 Development Roadmap

6.5.6.1. 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.

6.5.6.2. Adding Actions to Buttons

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

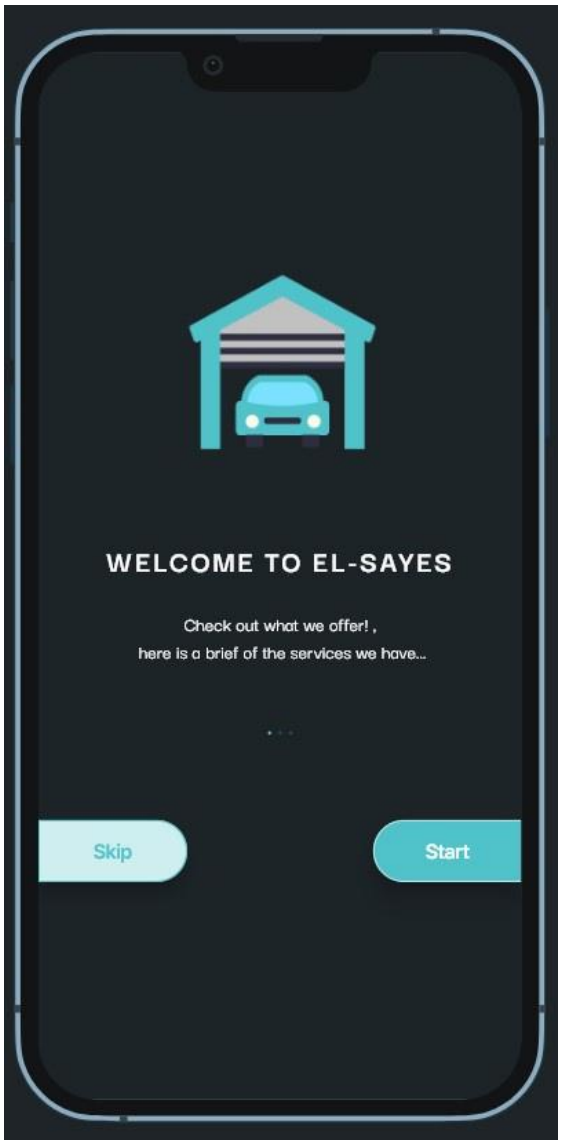
- **Key Tasks**:
  - Linking buttons to their corresponding pages (e.g., navigating to the reservation page when the "Reserve" button is clicked).
  - Adding functionalities like form submission, user authentication, and real-time data fetching using Firebase.
  - Implementing error handling for invalid 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.

6.5.6.3. 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.

6.5.6.4. Welcome Screen

- **Purpose**: Introduces users to the app with a user-friendly onboarding experience.
- **Key Features**:
  - A welcome message and a short description of the app's offerings.
  - Prominent icons or images representing the app's functionality.
  - Navigation buttons (e.g., "Skip" and "Start").
  - A progress indicator to show the user's position in the onboarding process.
- **Implementation in FlutterFlow**:
  - Navigation actions for the "Skip" and "Start" buttons.
  - Dynamic updates to the progress indicator for each onboarding step.

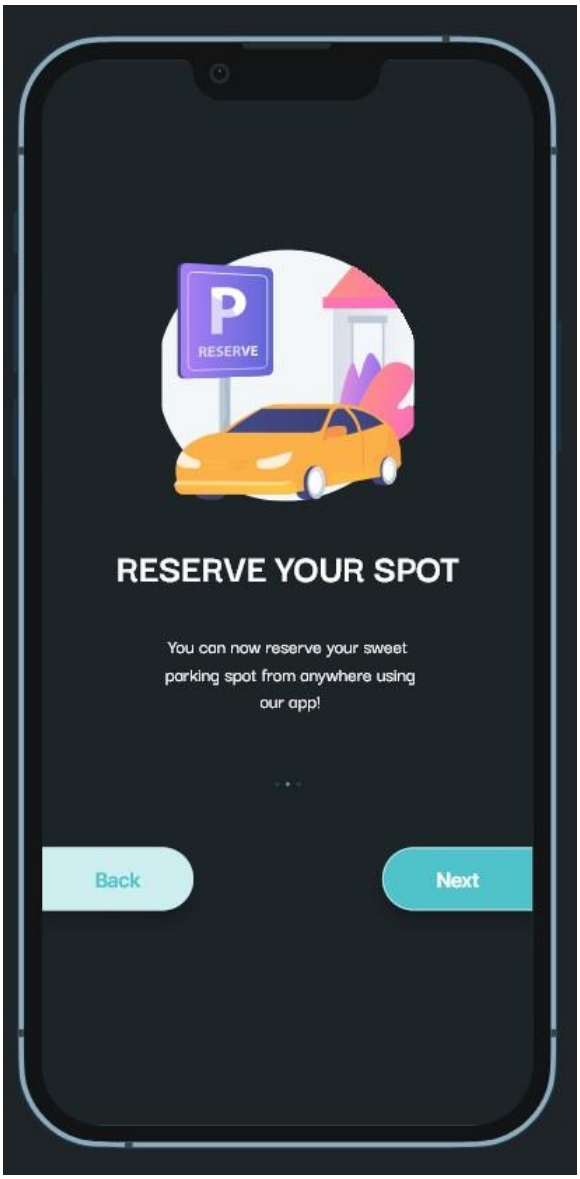


6.5.6.5. Reserve Your Spot Screen

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 before diving into its features.

Key Features



1. **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."*
2. **Visual Representation**:
   - Contains an illustration or icon (e.g., a car) to visually communicate the feature.
3. **Navigation Buttons**:
   - **Back Button**: Navigates the user to the previous onboarding screen (e.g., Welcome Screen).
   - **Next Button**: Proceeds to the next onboarding screen or main app screen.
4. **Progress Indicator**:
   - Displays the user's progress in the onboarding flow (e.g., dots indicating the current step).

Design Elements

- **Background**:
  - Maintains a dark theme consistent with the Welcome Screen for visual continuity.
- **Font Style**:
  - Features easy-to-read typography that aligns with the app's overall design language.
- **Buttons**:
  - Clearly labeled "Back" and "Next" buttons for intuitive navigation.

Functionality

- The **Back Button** takes the user to the Welcome Screen.
- The **Next Button** advances the user to the next onboarding or main app screen.
- The progress indicator updates dynamically to reflect the user's current position in the onboarding sequence.

Implementation in FlutterFlow

1. **Navigation Actions**:
    - Configure the "Back" and "Next" buttons with appropriate navigation logic.
2. **Progress Indicator**:
    - Dynamically link the progress indicator to the current onboarding step.
3. **Responsive Design**:
    - Ensure the layout adjusts seamlessly across different screen sizes to maintain usability.

6.5.6.6. Parking Availability Screen

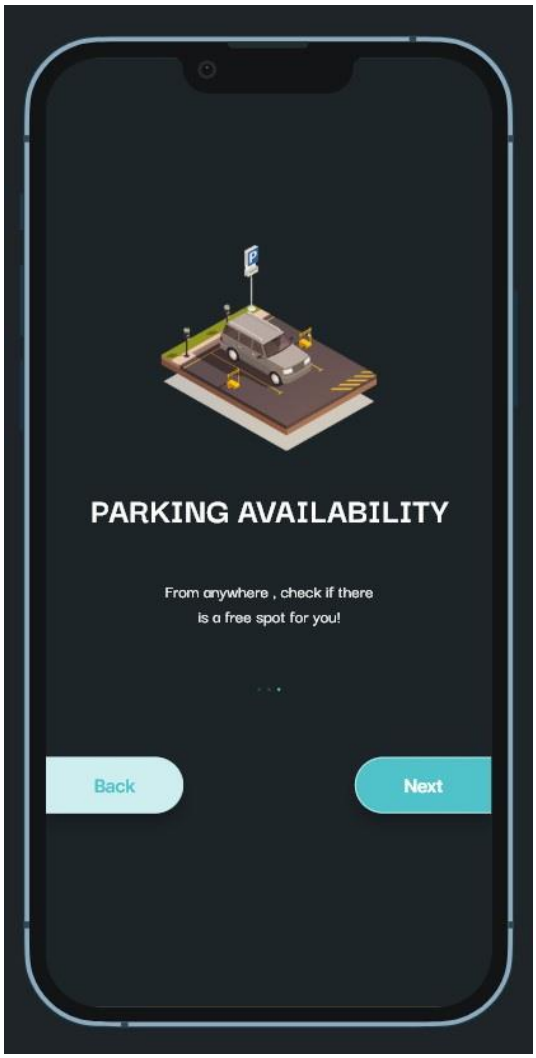
Purpose

The 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 illustration or icon, such as a car inside a parking garage, emphasizing the feature visually.

3. **Navigation Buttons**:
    - **Back Button**: Allows users to return to the previous screen (e.g., Reserve Your Spot Screen).
    - **Next Button**: Proceeds to the next onboarding screen or main app screen.
4. **Progress Indicator**:
    - Displays the user's progress in the onboarding flow (e.g., dots indicating the current step).



Design Elements

- **Background**:
    - Maintains a consistent dark theme with other onboarding screens for visual harmony.
- **Typography**:
    - Features simple and clear text for the feature description.
- **Buttons**:
    - Clearly styled "Back" and "Next" buttons consistent with the app's UI theme.

Functionality

- The **Back Button** navigates the user to the Reserve Your Spot Screen.
- The **Next Button** advances the user to the next onboarding or main app screen.
- The progress indicator updates dynamically to reflect the user's current position in the onboarding sequence.

Implementation in Flutter Flow

1. **Navigation Setup**:
    - Configure the Back and Next buttons to enable smooth transitions between screens.
2. **Progress Indicator**:
    - Dynamically link the progress indicator to align with the current onboarding step.

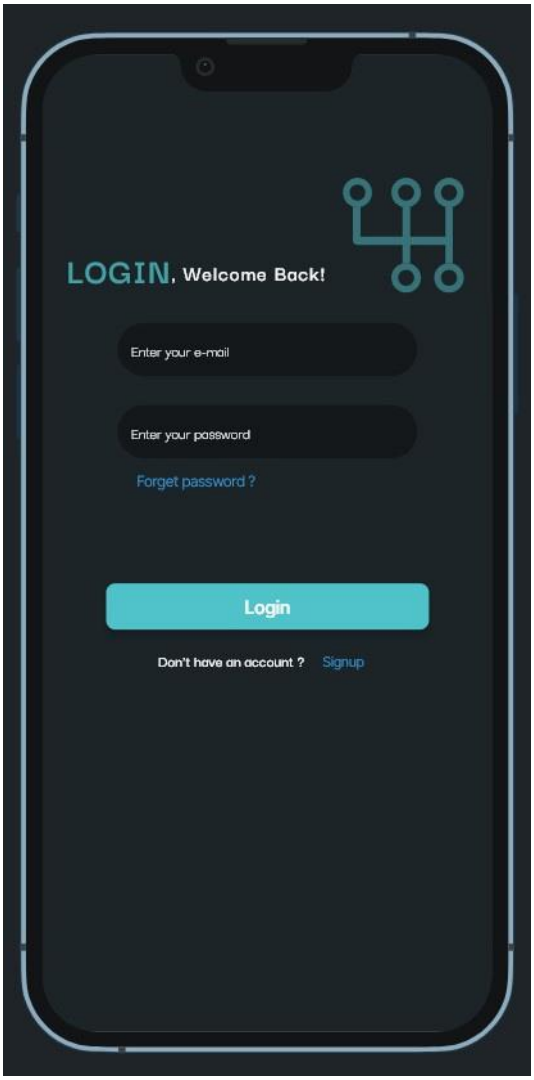
3. **Responsive Design**:
    - Ensure proper scaling and alignment of the car-in-garage illustration or icon across different device sizes to maintain a visually appealing layout.

6.5.6.7. 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

- **Typography**:
    - Clean and simple text for labels and links to ensure readability.
- **Buttons**:
    - A prominent Login button for accessibility and ease of use.
- **Color Scheme**:
    - Consistent dark theme that aligns with the overall app design.

Functionality

1. **Forgot Password**:
    - Redirects users to a password recovery screen or process.
2. **Sign-Up Redirect**:
    - Navigates users to the **Sign-Up Screen**.
3. **Login Button**:
    - Validates form inputs (email and password) to ensure correctness.
    - Logs the user into the app using Firebase or a backend authentication service.

Implementation in Flutter Flow

1. **Navigation Setup**:
    - Configure navigation for the **Forgot Password** and **Sign-Up Redirect** links.
2. **Form Validation**:
    - Set up validation for email format and password length.
3. **Login Button Integration**:
    - Link the Login Button to Firebase or any other backend authentication service to handle user login securely.

6.5.6.8. Sign-Up Screen

Purpose

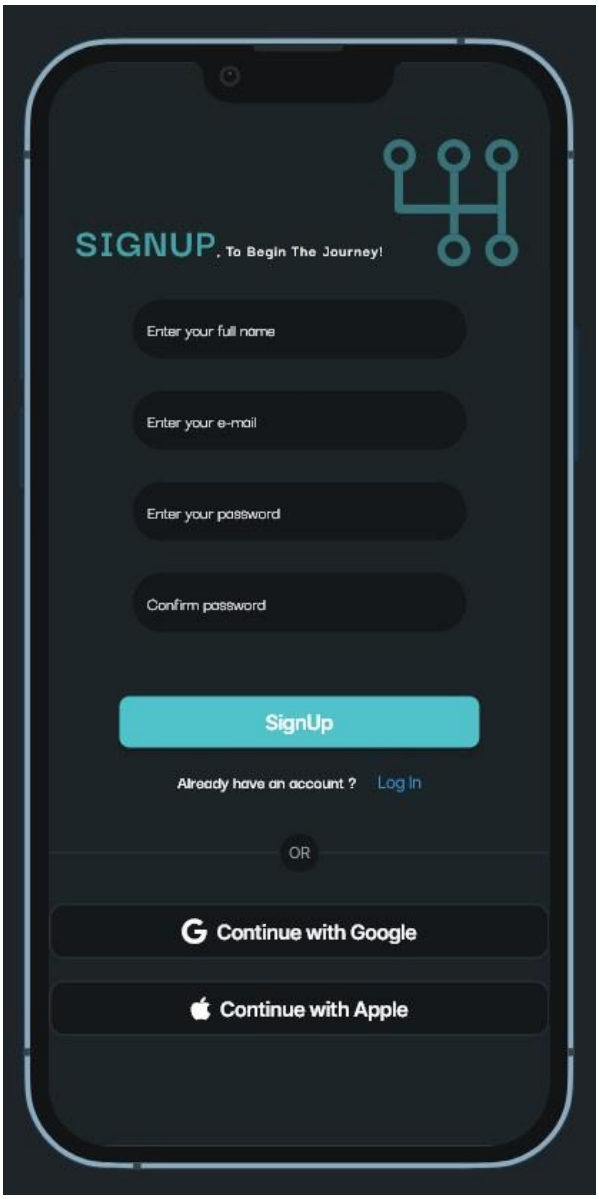
The Sign-Up Screen allows new users to create an account, enabling access to the app's features

Key Features

1. **Input Fields**:
    - **Full Name**: Allows users to provide their name for personalization.
    - **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 dark background for a modern aesthetic.
- **Buttons**:
    - A prominent Sign-Up Button for accessibility and ease of use.
    - Clearly defined options for Google and Apple sign-ups.
- **Form Validation**:
    - Ensures valid email formatting and matching passwords.

Functionality

1. **Sign-Up Button**:
    - Registers the user and saves their data to the backend.
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.

Implementation in Flutter Flow

1. **Backend Authentication**:
    - Configure email/password authentication and social media integration using Firebase or another backend service.
2. **Navigation Setup**:
    - Add navigation for the **Login Redirect** link.
3. **Form Validation**:
    - Set up real-time validation for user inputs, such as:
        - Checking email formatting.
        - Verifying that passwords match.

6.5.6.9. Parking System Interface (Home Page)

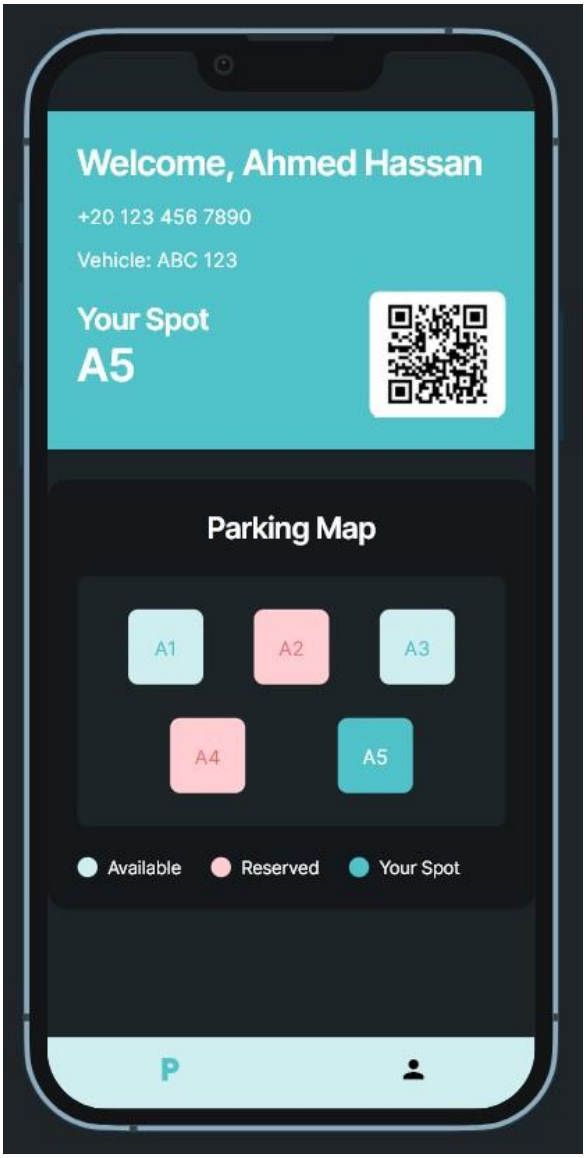
Purpose

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, and interact with a parking map.

Key Features

1. **User Information Header**:
    - Displays personalized user information, such as:
        - **Name**: E.g., Ahmed Hassan.
        - **Phone Number**: The registered contact number.
        - **Vehicle Details**: The vehicle's registration or license plate number.
        - **Your Spot**: Shows the user's reserved parking spot (e.g., A5).
        - **QR Code**:
            - Represents the user's reserved parking information.
            - Can be scanned for easy access or verification at the parking facility.

2. **Parking Map**:
    - A visual representation of available and reserved parking spots.
    - **Legend**:
        - Available spots: Marked in light blue.
        - Reserved spots: Marked in pink.
        - User's reserved spot: Highlighted in teal.
    - **Interactive Buttons**:
        - Clicking on a parking spot (e.g., A1, A2) navigates the user to the **Reservation Screen**, where they can reserve or manage parking.
3. **Navigation Bar**:
    - Provides quick access to other app functionalities:
        - **Home (P icon)**: Links to the current screen.
        - **Profile (User icon)**: Links to the **Profile Page** for account management.



Design Elements

- **Header Section**:
    - Prominent teal background for user details.
    - Bold, white typography for readability.
    - A large, scannable QR code for convenience.
- **Parking Map Section**:
    - Dark background with contrasting colors for easy differentiation of parking statuses.
    - Easy-to-tap buttons (e.g., A1, A2) for intuitive interaction.
- **Navigation Bar**:
    - Minimalist design with icons for quick navigation.
    - Consistent with the app's dark theme.

Functionality

1. **Dynamic User Information**:
    - Displays data retrieved dynamically from the backend, ensuring up-to-date details.
2. **Interactive Parking Map**:
    - Allows users to click on available spots to proceed with the reservation process.
    - Reserved spots are disabled or unclickable.
3. **QR Code**:
    - Encodes the user's parking details for use at the parking facility.
4. **Navigation Bar**:
    - Provides seamless transitions between the **Home Page** and **Profile Page**.

Implementation in FlutterFlow

1. **User Information Header**:
    - Bind user data (e.g., name, contact, vehicle, and parking spot) dynamically using backend integration.
    - Generate QR codes using a third-party library or API.
2. **Parking Map**:
    - Use a grid layout to create the parking map.
    - Implement button states (e.g., available, reserved, user spot) with conditional styling.
3. **Navigation Bar**:
    - Add navigation actions for each icon, linking them to the appropriate screens.
4. **Interactions**:
    - Configure button actions to navigate to the **Reservation Screen**.
    - Pass the selected spot as a parameter to enable the reservation or management process.

6.5.6.10. Profile Page Design

The Profile Page provides users with an interface to view and manage their personal details and app settings. It serves as an overview of the user's profile, allowing them to update their information and adjust app preferences, as well as log out securely.

Key Features:

1. **Header Section:**
    - Displays the user's profile picture, name, and email address.
    - Provides a quick, visually appealing summary of the user's identity within the app.
2. **User Information Section:**
    - Displays detailed user information:
        - Phone Number: Registered contact number.
        - Additional personal details: Retrieved dynamically from the backend.
3. **App Settings Section:**
    - Options for managing app-related settings:
        - **Language:** Allows the user to change the app's language.
        - **Currency:** Enables the user to choose their preferred currency for transactions.
        - **Profile Settings:** Allows the user to edit details like name, email, and phone number.
        - **Notification Settings:** Lets the user enable/disable notifications or manage preferences.
4. **Log Out Button:**
    - Provides an option to securely log out of the account.

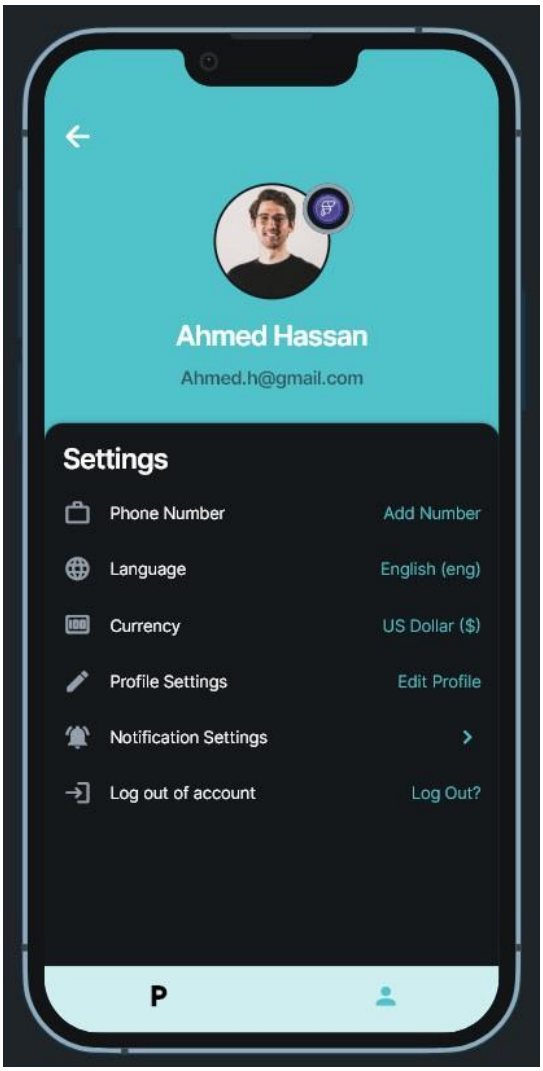
Design Elements:

1. **Header Section:**
    - Uses a centered layout with a rounded profile picture at the top.
    - The user's name is presented in bold text, with the email address in a smaller, lighter font for clarity and visual hierarchy.

2. **User Information Section:**
    - Clearly labeled fields for easy identification of the user's details.
    - The background is dark with contrasting colors, ensuring consistency with the app's theme.
3. **App Settings Section:**
    - Organized into categories for easy navigation and usability.
    - Includes toggles or dropdowns for settings like language and notifications.
4. **Log Out Button:**
    - Styled distinctively (e.g., using red text or a bordered button) for clear visibility.
    - Positioned at the bottom of the page for easy access.

Functionality:

1. **Header Section:**
    - Dynamically fetches and displays the user's profile picture, name, and email address from the backend.
2. **User Information Section:**
    - Displays read-only user information.
    - Editable fields (e.g., phone number) are linked to profile settings, allowing for easy updates.
3. **App Settings Section:**
    - The language and currency settings can open modal dialogs or dropdowns for user selection.
    - Notification settings are managed using toggle switches.
4. **Log Out Button:**
    - On click, it logs the user out and redirects them to the login screen.



Implementation in Flutter Flow:

1. **Header Section:**
    - Use a container with a circular image widget to display the profile picture.
    - Bind the name and email fields to backend data for dynamic display.
2. **User Information Section:**
    - Display a vertical list of labeled text fields for user details.
    - Provide navigation to profile settings to allow the user to edit their information.
3. **App Settings Section:**
    - Use a vertical list of buttons, dropdowns, and toggles to display the settings.
    - Configure navigation actions to open the settings screens or modals.
4. **Log Out Button:**
    - Place the button at the bottom of a scrollable container.
    - Set up the button to log the user out and navigate them to the login screen.

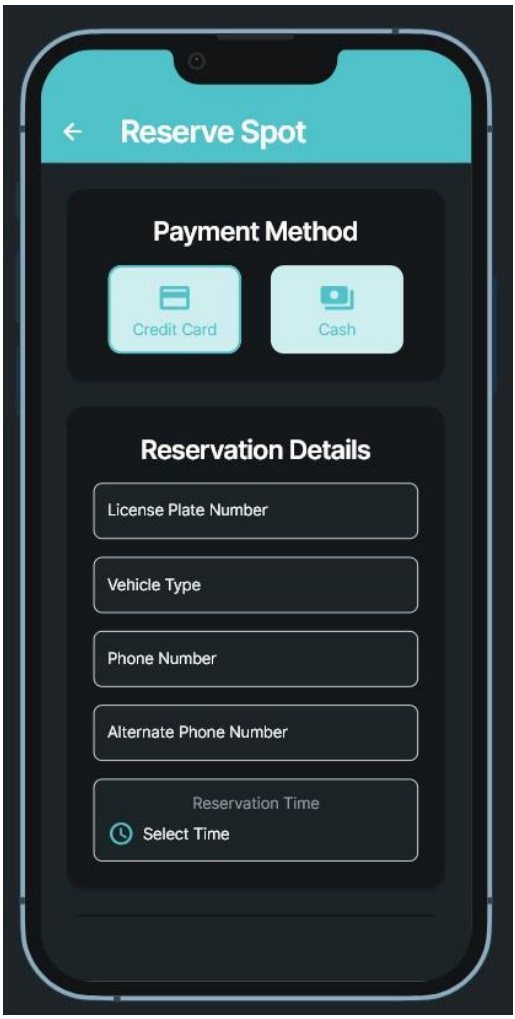
6.5.6.11. Reserve Spot Screen Design

Purpose: The Reserve Spot screen allows users to enter necessary information for reserving a spot, including contact details, reservation time, and payment information. It is divided into three main sections to guide users through the reservation process.

Sections Overview:

1. **Section 1: Reservation Details**
    - **Purpose:** Gathers user details necessary for the reservation.
    - **Key Features:**
        - **Alternate Phone Number:** An optional field where users can enter a backup phone number for contact purposes.
        - **Reservation Time:** A time selection widget that allows users to choose their preferred reservation time.

2. **Section 2: Payment Information**
    - **Purpose:** Handles the payment details for reservations made via credit card.
    - **Key Features:**
        - **Credit Card Information:** Fields for entering credit card details, including:
            - Card Number
            - Expiry Date
            - CVV
        - **Spot Details:** Displays a summary of the selected spot, including its identifier (e.g., "A3") and the hourly rate.
3. **Section 3: Confirmation and Submission**
    - **Purpose:** Allows users to review their reservation details and confirm the reservation.
    - **Key Features:**
        - **Confirm Reservation Button:** A prominent button that finalizes the reservation process, submits the entered information, and initiates the payment transaction.



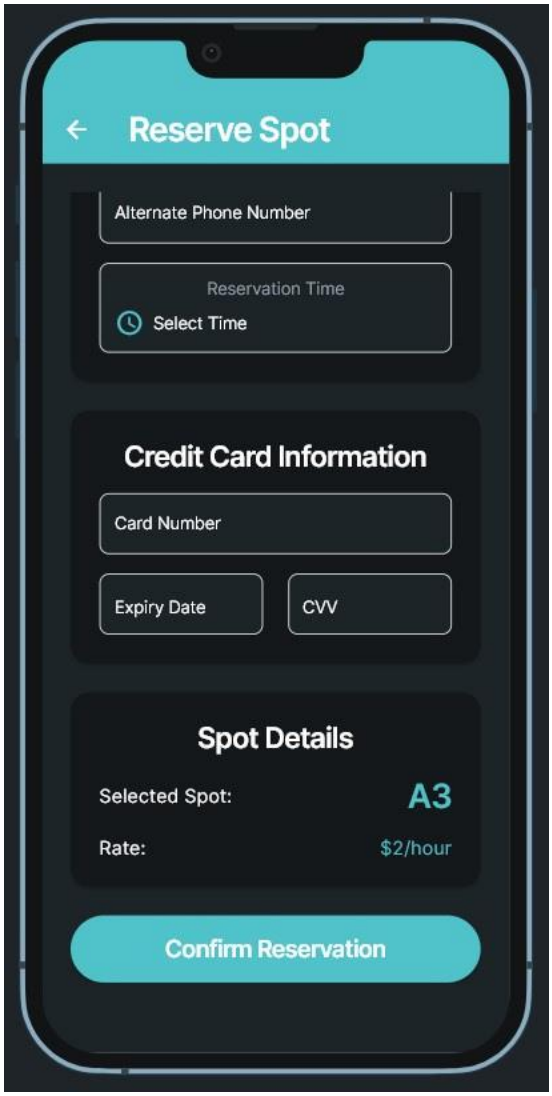
Design Considerations:

1. **Clear Sectioning:**
    - The screen is divided into distinct, clearly separated sections using spacing and subtle design elements to enhance readability and user understanding.
2. **Consistent Styling:**
    - The dark theme and consistent typography from previous screens are maintained, ensuring a cohesive visual experience.
3. **Informative Labels:**
    - Each input field has a clear and concise label, guiding users on the required information.
4. **Input Validation:**
    - The application likely performs real-time validation on user inputs to ensure accuracy and prevent errors (e.g., validating the phone number format and credit card details).

5. **Time Selection:**
    - The time selection widget offers an intuitive interface for users to select their desired reservation time.
6. **Payment Processing:**
    - Upon confirmation, the application will securely process the payment using the provided credit card details via a third-party payment gateway (e.g., Stripe).
7. **Reservation Confirmation:**
    - After successful payment, the application will generate a confirmation message and potentially provide a digital receipt or reservation details.

Functionality:

1. **Reservation Details:**
    - Users input their alternate phone number and select the reservation time.
2. **Payment Information:**
    - Credit card details are entered, and the spot summary is displayed, showing the selected spot's identifier and the hourly rate.
3. **Confirmation and Submission:**
    - Users confirm their reservation by clicking the "Confirm Reservation" button, which submits the entered data and initiates payment processing.
4. **Real-Time Data Binding:**
    - The screen likely uses FlutterFlow's data binding to update the UI based on user input (e.g., displaying the selected spot's details dynamically).
5. **Payment Gateway Integration:**
    - The application integrates with a third-party payment gateway (e.g., Stripe) to securely process credit card payments.



FlutterFlow Implementation:

1. **Scrollable Layout:**
    - FlutterFlow's scrollable layout components (e.g., SingleChildScrollView or ListView) are used to accommodate the multiple sections and allow users to scroll through the form.
2. **Page View Integration:**
    - This screen builds upon the Page View integration from previous screens, allowing users to switch between payment method options (e.g., Cash or Credit Card) and display the appropriate fields for each option.
3. **State Management:**
    - FlutterFlow's state management system will be used to manage user input, dynamically update the UI (e.g., showing or hiding fields), and handle the reservation process.
4. **Third-Party Integrations:**
    - The screen integrates with third-party payment gateways (e.g., Stripe) for securely processing credit card payments. This integration ensures that sensitive payment data is handled safely and that the user experience remains smooth.

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.

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