Creating an Automated Parking System Enabled by IoT Technology

Design Document

Team: sddec25-09

Client: Md Maruf Ahamed

Adviser: Md Maruf Ahamed

Team Members:

Thomas Olson - Frontend

Harley Peacher - Frontend

Sullivan Hart - Hardware

Joseph Schmidt - Server

Team Email: sddec25-09@iastate.edu

Team Website: https://sddec25-09.sd.ece.iastate.edu/

Revised: 05/04/2025 / v2.01

Executive Summary

Parking can be a tedious and inefficient process. In industrial settings (malls, airports, etc.), this is solved with complex capacity-monitoring systems. These systems are expensive and must be constructed for each site. This leads to a large barrier of entry, preventing small and medium-sized lots from having efficient parking.

To solve this, a smart parking system using modern IoT and machine learning technology will be created. The design will be modular and require no expert installation. This design includes:

- React Mobile App: A mobile application that displays live parking lot statuses and lets users request lot availability by location.
- MySQL Server: Centralized backend for storing availability and occupancy data.
- Raspberry Pi 5 & AI Hat: Acts as the edge device, receiving camera input and processing detection jobs locally using YOLOv8.
- ESP32-CAM Modules: Low-power wireless cameras deployed across parking lots to periodically capture images of parking spaces, and transmit them to the Pi.

The system uses the Raspberry Pi as a server for each lot. Each ESP32-CAM captures frames at a timed interval and sends them wirelessly to the Pi. The Pi then processes these frames in a queue using YOLOv8, identifying which spots contain parked vehicles. If a change in availability is detected, the Pi sends updated information to a remote MySQL server. The React mobile app communicates directly with this server, requesting a parking lot's availability and sending user interactions.

The project currently collects and labels photos, stores spot information, and transmits it to the hardware; then, data is sent to the server, and the app accesses it. App users can see the availability of configured lots.

Next semester, the project's modularity will be improved; an admin interface will be created, and more parking lots will be added. Features (such as license plate detection, fewer cameras, and different spot types) will be added to make the project compete with similar systems. Implementing testing standards and code coverage will ensure the prototype is complete and ready to use. User testing with individuals outside of the development team will guarantee a usable project.

Learning Summary

Development Standards & Practices Used

Practices:

1. Rapid Prototyping
2. Modular Design
3. Iterative Design
4. Source Control
5. Agile

Relevant IEEE Standards

1. IEEE 3156-2023 - “IEEE Standard for Requirements of Privacy-Preserving Computation Integrated Platforms”
2. IEEE 1686-2022 - IEEE Standard for Intelligent Electronic Devices Cyber Security Capabilities
3. IEEE 802 - Family of Networking Standards

Summary of Requirements

Functional Requirements:

- Display capacity of parking lots.
- Display capacity within lots.
- Incorporate a secure payment portal.
- Enable reporting of issues (misparking, blockages, etc).
- Enable saving of spots.
- Enable a portal to manage a lot (toggle payment, resolve issues, etc).

Non-Functional Requirements:

- Easy to use interface (for parkers and admins).
- Consistent behavior (reliable app, server, and hardware).
- Quick use of key features.
- More details that are also intuitive (but may not be at the forefront of the app).
- Visually appealing (app and hardware).
- Modular hardware (deployable).

Applicable Courses from Iowa State University Curriculum

- Frontend/Backend software development and version control - Coms 3090 Software Development Practices
- Networking protocols for VM - Cpre 4890 Computer Networking and Data Communications
- Computer vision car detection - Cpre 4870 Hardware Design for Machine Learning
- Advanced data structures - Coms 2280 Introduction to Data Structures
- Configuring database - Coms 3630 Introduction to Database Management Systems
- Creating website and frontend interface - Coms 3190 Construction of User Interfaces
- Developing code for Raspberry Pi - Cpre 2880 Embedded Systems I: Introduction
- Connecting and testing camera charging port - EE 2300 Electronic Circuits and Systems
- Setting up container environments on devices - Cpre 3080 Operating Systems: Principles and Practice
- Writing concise and descriptive documentation, and presenting information compellingly - ENGL 2500: Written, Oral, Visual, and Electronic Composition
- Enhanced professionalism in documentation and design - ENGL 3140 Technical Communication

New Skills/Knowledge acquired that was not taught in courses

- Using Docker to containerize projects
- Gained experience creating Dockerfiles to define container environments
- Developed skills in Docker Compose to orchestrate multiple containers working together
- Implemented MySQL into a Docker Container to enable persistent storage
- Worked with Gitlab to maintain project code consistency
- Built proficiency in Spring Boot framework to implement REST APIs
- Strengthened networking skills, setting up reverse proxy for campus virtual machine to allow external network traffic to access ports
- Acquired React Native knowledge
- Utilized Expo to allow for easy simulation on a mobile device
- Raspberry Pi setup and configuration
    - Docker & Watchtower
    - Hotspot
    - Utilization of AI Hat
- Strengthened machine learning knowledge
- Learned to solder
- Developed familiarity with the ESP32 camera

Table of Contents

List of figures/tables/symbols/definitions

Table of Figures

1. Introduction

1.1. Problem Statement

Finding a parking spot is currently inefficient, time-consuming, and frustrating. Drivers often resort to circling lots aimlessly, especially during busy hours. This project will increase convenience, reduce parking search time, and optimize lot usage.

1.2. Intended Users

Admin User - In charge of configuring and maintaining the parking lot infrastructure. This includes modeling and assigning cameras on the configuration website and setting up physical cameras on-site.

Parker - The system's primary user utilizes the app to show available parking and set reservations, streamlining the parking process.

Different parking users:

- Rushed Parker - Focused on quickly finding a spot and is most likely in a heightened emotional state. The app must be easy to use in this case, being free of clutter, and allow for fast and reliable retrieval of available spots.
- Elderly and Disabled Parker - Looking for a close handicap parking spot. Prioritizes ease of use and clear information to find parking spots with ideal accessibility.
- Patient Parker - Determined to obtain optimal parking and willing to spend additional time and effort to achieve this goal. The app enables users to pinpoint specific spots to meet their preferences.
- Proactive Parker - A user who wants to secure a parking spot in advance to minimize uncertainty. Using the reservation feature in the app allows this user to stay on schedule.
- Forgetful Parker - May forget the location of their parked vehicle. The app assists by locating the vehicle’s license plate and directing the user to the correct parking spot.

2. Requirements, Constraints, And Standards

2.1. Requirements & Constraints

Functional Requirements:

- Display capacity of parking lots.
- Display capacity within lots.
- Incorporate a secure payment portal.
- Enable reporting of issues (misparking, blockages, etc).
- Enable saving of spots.
- Enable a portal to manage a lot (toggle payment, resolve issues, etc).

Non-Functional Requirements:

- Easy to use interface (for parkers and admins).
- Consistent behavior (reliable app, server, and hardware).
- Quick use of key features.
- Visually appealing (app and hardware).
- Modular hardware (deployable).

2.2. Engineering Standards

Engineering standards are important because they prevent errors and ensure thoroughness in the final design. These standards give guidelines to maintain the health and security of others.

Relevant IEEE Standards

1. IEEE 3156-2023 - “IEEE Standard for Requirements of Privacy-Preserving Computation Integrated Platforms”
2. IEEE 1686-2022 - IEEE Standard for Intelligent Electronic Devices Cyber Security Capabilities
3. IEEE 802 - Family of Networking Standards

IEEE 3150-2023 provides various requirements for multi-computer systems. The standard specifies security and performance requirements to ensure private and accurate computations.

IEEE 1686-2022 provides requirements relevant to cybersecurity in Intelligent Electronic Devices. These include Internet of Things applications and specify how to ensure data is protected between nodes.

IEEE 802 includes a variety of standards relevant to networking. These standards ensure compatibility between both ends of the communication as well as reliability.

The chosen standards will be implemented by power saving, encrypting data, and using trusted transaction services. These methodologies will maintain alignment with established standards, ensuring that the design is safe and secure.

3 Project Plan

3.1 Project Management/Tracking Procedures

The project follows a hybrid approach, combining waterfall and agile methodology. Initially, the project was broken into phases, ensuring that each group member's contributions complement one another, representing the waterfall portion. Once the project progressed through the research, development, and integration phases, the team had a product that could be tested and refined. At that stage, there was a transition to an agile workflow where frequent sprint meetings ensured continuous, incremental improvement in the project. Iterating on a functional design proved more effective than iterating on individual components in isolation.

The team primarily utilizes Discord, GitLab, and Google Docs to keep track of progress. Discord allows communication about current sprint activities and progress toward project goals. Employing GitLab enables the senior design group to monitor team pushes to ensure consistent progress and maintain accountability amongst team members. Lastly, leveraging Google Docs allows for internal documentation and API updates.

3.2 Task Decomposition

This task decomposition allows for parallel development across the different teams in the project: the front-end developing screens, the back-end database API, and the Raspberry Pi hardware, all being developed concurrently while following SCRUM procedures to ensure seamless integration.

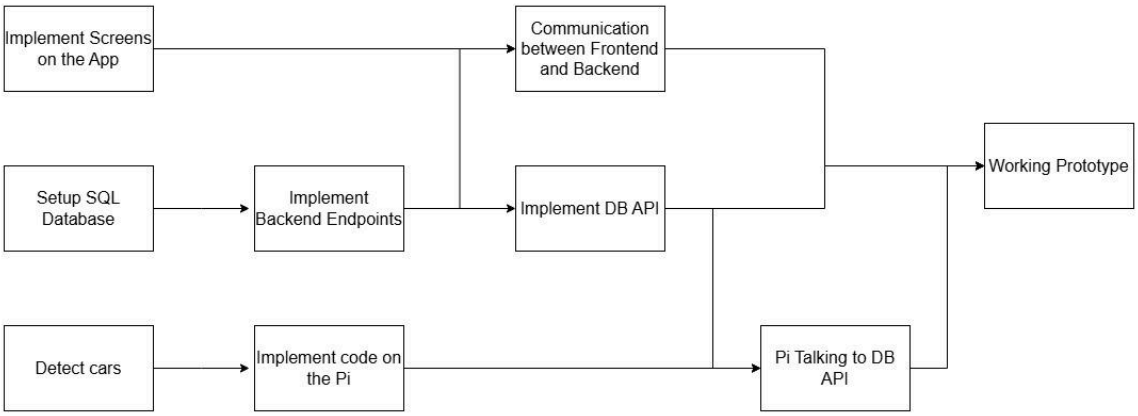

Figure 3.2.1: Task Decomposition Chart

3.3 Project Proposed Milestones, Metrics, and Evaluation Criteria

Milestones, Metrics, and Evaluation Criteria

Front-end:

- Completing Basic Screens
    - Create welcome, register, login, and landing pages.
- Integration with the Back-end API
    - These screens should be communicating with the back-end. This will allow for user registration and login.
- More Complex Screens Completed
    - Map, Reservation, and Payment Screen added.
- Integration of More Complex Screens with the Back-end API
    - The complex screens should connect to the back-end API.
- All Screens Are Completed and Integrated (Completed Prototype)
    - The app should have complete and intended functionality.

Back-end:

- Completing Basic Repositories
    - Person, Parking Lot, Parking Spots, Pi.
- Switching to Persistent Storage DB
    - Instead of H2 in memory, using MySQL to preserve data.
- Setting up CI/CD Pipeline
    - Automatic Junit testing and deployment of new code.
- Encrypting User Data
    - Ensuring sensitive user data is encrypted and sent over secure network protocols such as HTTPS.
- Generating Code Coverage
    - Ensuring all backend code is tested by unit tests.
- Advanced Database Information
    - Keeping track of license plates and payment methods.

Hardware:

- Selection of Parts
    - Identify which parts need to be ordered.
- Raspberry Pi Configuration
    - Set up the Raspberry Pi to update its Docker Container when appropriate.
    - Create network credentials.
    - Get the Pi to run once power is received.
    - Configure the Pi to create a hotspot that the cameras connect to.
- Connection with Server
    - Transmit dummy data to the server.
- Camera Module Creation
    - Solder the camera module, charging board, boost converter, and battery.
    - Develop code to periodically wake-up" and transmit still frames to Raspberry Pi.
- Connection with Camera
    - Configure the Pi to host a server with endpoints to receive camera frames
    - Filter the hotspot to allow only valid MAC addresses.

- Raspberry Pi Database
    - Allow persistent storage (between Docker refreshes) of parking lot information and camera frames.
- Machine Learning
    - Update spot information when a new camera frame is available in the database.
    - Offload work to AI Hat.
- GUI for Admin
    - Create an interface that allows the administrator to configure cameras without using the Raspberry Pi's command line interface.
- Identify License Plates
    - Update machine learning and cameras to read license plates of parked cars.
- Reduce Number of Cameras
    - Optimize the number of spots that can be processed by one camera.

3.4 Project Timeline/Schedule

The project was divided into software and hardware branches. During the first few weeks of project development, time was taken to develop a clear understanding of the project's high-level requirements and to determine interest in each type of work. The senior design team then created the following Gantt charts to help schedule deadlines for deliverables. A high-level schedule was also developed to ensure that clear goals for integrating the different parts of the system were set. The software and hardware schedules were designed to align with the scope defined by the high-level schedule.

Task	Start	End	Duration	Category
Project Start/Ideation	2025-02-18	2025-02-28	11	Main
Group Segmentation	2025-03-01	2025-03-07	7	Main
HW/SW Requirement Brainstorming	2025-03-08	2025-03-21	13	Main
Basic Prototyping	2025-03-22	2025-04-18	28	Main
Basic Prototype System Integration and Implementation	2025-04-19	2025-05-17	19	Main

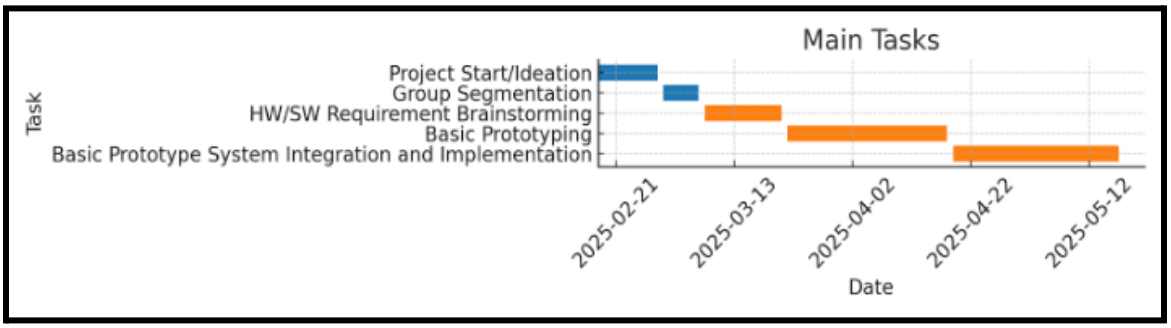

Figure 3.5.1: High-Level Schedule

Task	Start	End	Duration	Category
SW: Requirement Brainstorming	**2025-03-08**	**2025-03-21**	**13**	SW Team
Overall software architecture	2025-03-08	2025-03-15	7	
Banking/payment API	2025-03-15	2025-03-17	2	
Map interface	2025-03-15	2025-03-19	4	
Map pin module	2025-03-15	2025-03-20	5	
SW: Basic Prototyping	2025-03-22	2025-04-18	**28**	SW Team
Log In Screen Prototype	2025-03-22	2025-03-24	2	
Register Screen Prototype	2025-03-22	2025-03-25	3	
Home/Map Screen Prototype	2025-03-22	2025-03-27	5	
Integration from Register POST to backend API	2025-03-22	2025-03-29	7	
Process Log In GET request from backend API	2025-03-22	2025-03-29	7	
User login to application hookup	2025-03-22	2025-04-01	10	
SW: System Integration and Implementation	**2025-04-19**	**2025-05-17**	**29**	SW Team

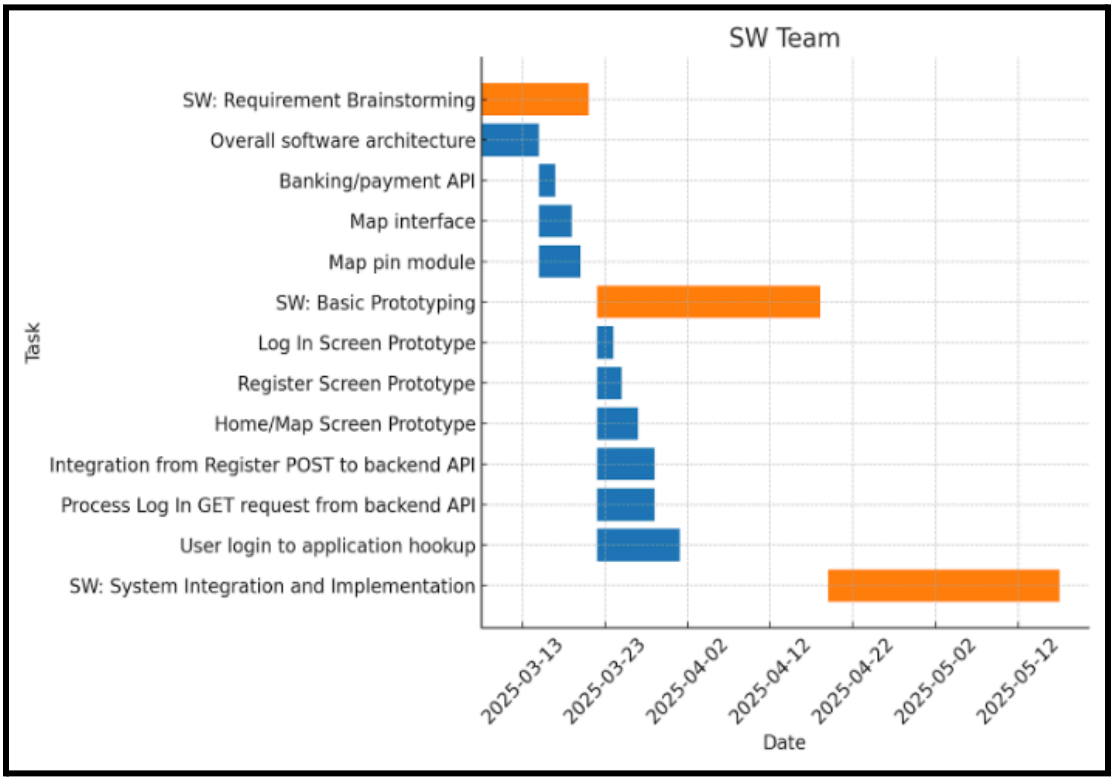

Figure 3.5.2: Software Schedule

Task	Start	End	Duration	Category
HW: Requirement Brainstorming	**2025-03-08**	**2025-03-21**	**13**	HW Team
Method for computation	2025-03-08	2025-03-15	7	
CV model	2025-03-08	2025-03-15	7	
System design	2025-03-08	2025-03-18	10	
HW: Basic Prototyping	**2025-03-22**	**2025-04-18**	**28**	HW Team
Streamline remote access to hardware	2025-03-22	2025-03-24	2	
Get CV model running on Pi	2025-03-22	2025-03-24	2	
Connect camera module/s to Pi	2025-03-22	2025-03-25	3	
Connect Pi to server	2025-03-22	2025-03-25	3	
Implement custom CV pipeline	2025-03-22	2025-03-29	7	
Fully implement AI hat into CV pipeline	2025-03-22	2025-04-18		
HW: System Integration and Implementation	**2025-04-19**	**2025-05-17**	**29**	HW Team
Integrate pipeline with SW API	2025-04-19	2025-05-17		

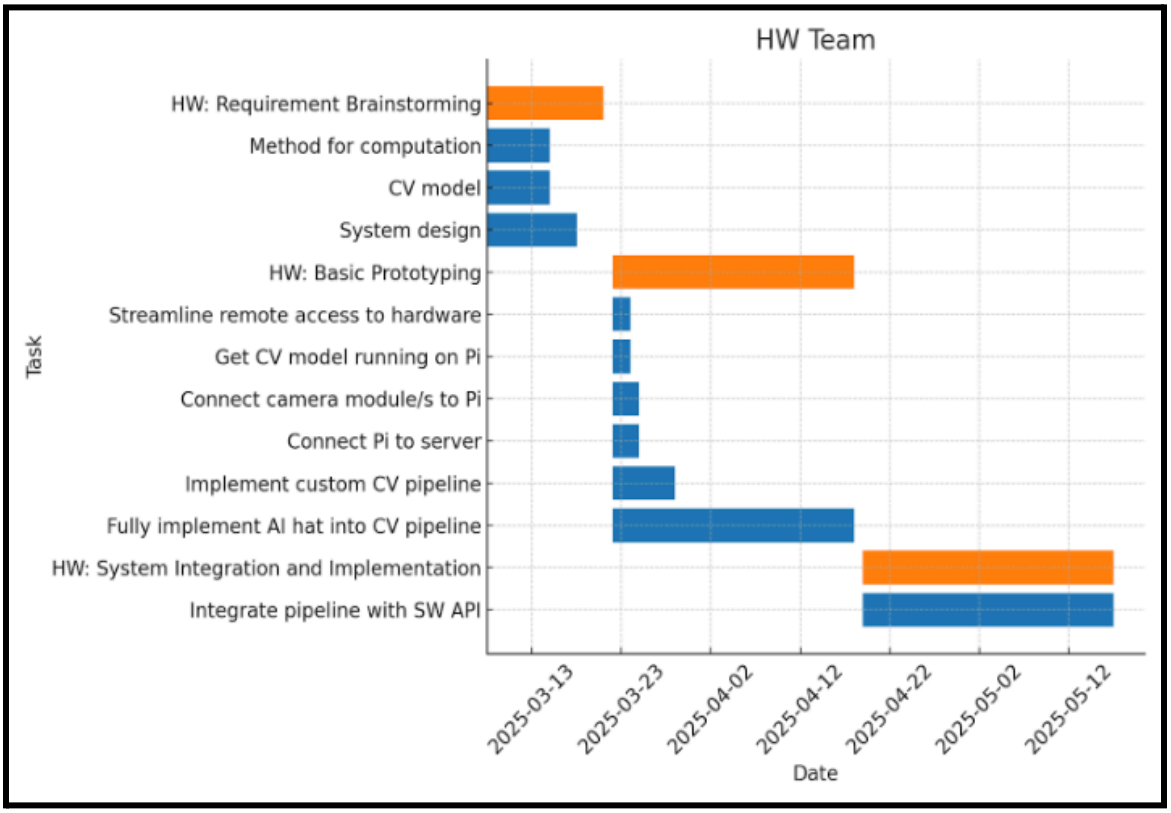

Figure 3.5.3: Hardware Schedule

3.5 Risks and Risk Management/Mitigation

Hardware Development Risk Management/Mitigation

Computation:

- **Risk/Constraint:** The system must be able to run a computer vision model in active time with an identification accuracy (car and plate number) of greater than 95 percent.
    - **Risk Level:** 0.1
    - **Solution/Mitigation Method:** A Raspberry Pi with an AI hat accelerator is used for computer vision computations. Initial testing indicates 95% accuracy can be obtained by the unit. If the accuracy isn't sufficient, a new model or computation on the server may be used.

Cost:

- **Risk/Constraint:** The system must be able to cover at least one lot while staying within the $500 budget.
    - **Risk Level:** 0.1

- **Solution/Mitigation Method:** The current setup of the Pi 5 and AI hat is less than $300. The remainder of the budget will be allocated to the cameras needed to cover the lot, a total of $70.

Reliability:

- **Risk/Constraint:** The system must be able to run overnight (for prototyping phase).
  - **Risk Level:** 0.4
  - **Solution/Mitigation Method**: The Pi AI hat comes with a fan to combat cooling issues. If the issue persists, the location of the Pi can be changed, the workload on the Pi may be decreased, or a more powerful cooling system can be pursued.

Robustness:

- **Risk/Constraint:** The system must be able to reboot to operable condition if an error occurs. Specifically, network or power interruptions.
  - **Risk Level:** 0.1
  - **Solution/Mitigation Method:** The Pi will know to return to operating condition by executing code with time-outs and other error catching strategies. By using persistent memory and setting the code to execute from connection to power, it will return to its operating state without human intervention.The ESP32 modules will also connect to the network and transmit images upon receiving power.

Ease of Development/Use:

- **Risk/Constraint:** The hardware should be up to date and easy to maintain.
  - **Risk Level:** 0.1
  - **Solution/Mitigation Method**: A widely supported CV architecture, YOLOv8, is run with the Raspberry Pi AI Hat SDK and pipeline. This is a combination, so development and rapid prototyping can be sustained by this team and any others that work on this project.

Software Development Risk Management/Mitigation

Overall Software Architecture Risks

- **Risk/Constraint:** Scalability challenges – the architecture might not support growing user loads effectively.
  - **Risk Level**: 0.4
  - **Solution/Mitigation Method**: Utilizing industry-standard tooling within the application will mitigate possible issues with authentication and authorization. All potentially sensitive data should be encrypted using SHA256 or better encryption.
- **Risk/Constraint:** Security weaknesses – weak authentication/authorization mechanisms could expose sensitive data.
  - **Risk Level**: Severe

    - **Solution/Mitigation Method**: Utilizing industry-standard tooling within the application will mitigate possible issues with authentication and authorization. All potentially sensitive data should be encrypted using SHA256 or better encryption.

Banking/Payment API

- **Risk/Constraint:** Compliance issues – failure to meet regulations like PCI DSS could potentially lead to legal action.
    - **Risk Level**: Severe
    - **Solution/Mitigation Method**: Utilizing libraries and ensuring that all relevant standards are understood prior to implementation will help avoid compliance violations.
- **Risk/Constraint:** Fraud/unauthorized transactions – weak authentication could lead to financial fraud.
    - **Risk Level**: Severe
    - **Solution/Mitigation Method**: Ensuring industry standards are followed for authentication and libraries are used appropriately to protect users against fraud.

Map Interface

- **Risk/Constraint:** Third-party API downtime – reliance on external map services introduces external risks.
    - **Risk Level:** 0.3
    - **Solution/Mitigation Method:** N/A (already embedded in statement above).
- **Risk/Constraint:** Device compatibility issues – the map may not function properly on all mobile devices or browsers.
    - **Risk Level:** <0.1
    - **Solution/Mitigation Method:** A library within React Native is used to avoid compatibility issues across platforms.

Login Screen Prototype

- **Risk/Constraint:** Weak authentication design – insecure login mechanisms could expose user credentials.
    - **Risk Level:** Severe
    - **Solution/Mitigation Method:** Industry-standard libraries for form submission and HTTP requests are used to mitigate authentication risks.

Register Screen Prototype

- **Risk/Constraint:** Spam/fake accounts – lack of email/phone verification may allow bot registrations.
    - **Risk Level:** 0.4
    - **Solution/Mitigation Method:** Currently, there is no verification system to prevent bot registrations. This could allow bots to create multiple accounts and block legitimate users.

Home/Map Screen Prototype

- **Risk/Constraint:** Data refresh issues – inconsistent updates could make the map unreliable.
    - **Risk Level:** 0.4
    - **Solution/Mitigation Method:** N/A (already embedded in statement above).
- **Risk/Constraint:** Device-specific bugs – some features might not work across all devices.
    - **Risk Level:** <0.1
    - **Solution/Mitigation Method:** React Native is used to ensure cross-platform compatibility.

Integration from Register POST to Backend API

- **Risk/Constraint:** API downtime – backend unavailability could prevent new user registrations.
    - **Risk Level:** 0.5
    - **Solution/Mitigation Method:** A queue can be implemented to retry API requests when the backend becomes available. Users can be notified upon successful registration.

Process Login GET Request from Backend API

- **Risk/Constraint:** Data leakage – sensitive user data may be exposed in API responses.
    - **Risk Level:** Severe
    - **Solution/Mitigation Method:** All data exchanges will occur within encrypted request bodies to mitigate data leakage risks.

User Login to Application Hookup

- **Risk/Constraint:** Session expiry issues – poor session handling may lead to unintended logouts.
    - **Risk Level:** 0.5
    - **Solution/Mitigation Method:** Login tokens will be securely stored on the front end to ensure session stability.
- **Risk/Constraint:** Multiple login handling – limited support for multi-device access may frustrate users.
    - **Risk Level:** 0.3
    - **Solution/Mitigation Method:** While multi-device login is unlikely for mobile-focused users, a desktop version for parking administrators could benefit from support for multiple sessions.

System Integration and Implementation

- **Risk/Constraint:** Lack of rollback plan – no fallback exists for failed deployments.
    - **Risk Level:** 0.5
    - **Solution/Mitigation Method:** Future versions will include versioning and rollback mechanisms to ensure recoverability.
- **Risk/Constraint:** API versioning conflicts – changes in backend APIs may break existing integrations.
    - **Risk Level:** 0.3

- **Solution/Mitigation Method:** Versioned endpoints and proper documentation will be maintained to avoid conflicts.

Backend Application Crashing

- **Risk/Constraint:** Unhandled exceptions in backend – edge cases may cause the SpringBoot backend to crash.
  - **Risk Level:** 0.2
  - **Solution/Mitigation Method:** A Docker container policy will be applied to ensure automatic restart if the application fails.

3.6 Personnel Effort Requirements

Task	Duration	Category
SW: Requirement Brainstorming	**22**	SW Team
Overall software architecture	5	
Banking/payment API	5	
Map interface	6	
Map pin module	6	
SW: Basic Prototyping	**52**	SW Team
Log In Screen Prototype	7	
Register Screen Prototype	10	
Home/Map Screen Prototype	15	
Integration from Register POST to backend API	5	
Process Log In GET request from backend API	5	
User login to application hookup	10	
SW: System Integration and Implementation	**TBD**	SW Team
HW Backend API	TBD	

Figure 3.6.1: Software Development Effort Estimate

Task	Duration	Category
HW: Requirement Brainstorming	**15**	HW Team
Method for computation	5	
CV model	5	
System design	5	
HW: Basic Prototyping	**35+**	HW Team
Streamline remote access to hardware	5	
Get CV model running on Pi	10	
Connect camera module/s to Pi	3	
Connect Pi to server	2	
Implement custom CV pipeline	15	
Fully implement AI hat into CV pipeline	TBD	
HW: System Integration and Implementation	TBD	HW Team
Integrate pipeline with SW API	TBD	

Figure 3.6.2: Hardware Development Effort Estimate

3.7 Other Resource Requirements

The physical resources are limited to the hardware for the prototype. This includes the Raspberry Pi 3 Model B, a power supply Raspberry Pi, multiple ESP32 camera modules, a battery for each ESP32, antennas for each ESP32, and miscellaneous mounting hardware. In addition to the physical resources, the prototype also utilizes a server on Iowa State's network that has been opened to the public.

4 Design

4.1 Design Context

4.1.1 Broader Context

Area	Description	Examples
Public health, safety, and welfare	Unipark reduces the stress and frustration associated with searching for parking, which can contribute to road rage and distracted driving. By streamlining the parking process, it enhances the well-being and safety of drivers and pedestrians in high-traffic areas.	Reduces risk of accidents from distracted driving; lowers stress for drivers; improves traffic flow in congested parking areas.
Global, cultural, and social	The app promotes equitable access to parking and can be adapted for various urban environments. By supporting multilingual interfaces and considering diverse user needs (e.g., accessibility), it reflects values of inclusivity and respect for community practices.	Encourages fair parking access; avoids discrimination in space allocation; supports user trust across cultural groups.
Environmental	Unipark can help reduce fuel consumption and emissions by minimizing the time spent idling or circling lots in search of parking. This contributes to improved air quality in urban settings and supports sustainability efforts.	Decreases unnecessary driving; reduces CO_2 emissions; promotes environmentally efficient use of parking infrastructure.
Economic	The app can improve economic efficiency by optimizing existing parking resources, potentially increasing revenue for parking lot operators. For users, reduced search time translates to savings on fuel and time. It may also reduce the need for costly new parking infrastructure.	Saves drivers money on fuel; creates new revenue streams for parking operators; reduces infrastructure expansion costs.

4.1.2 Prior Work/Solutions

The project design is another solution to the same problem that a group of senior design students previously worked on [1]. The similarities between the two implementations are limited to the tooling used for the frontend and backend. Both projects utilized a react-native framework for the frontend, including react-native-maps for the map screen, and Stripe for payment. Additionally, both implementations employed Spring Boot to create a RESTful API that connects both the hardware and the front-end application. The main difference between the two implementations is the use of cameras to detect vehicles as opposed to ultrasonic sensors. Other differences include using MySQL instead of Firebase due to familiarity from previous work.

FlockSafety.com [2] is a company that allows users to prevent crime by using a website and a variety of cameras and to monitor vehicles. They have multiple cameras for different uses including solar,

battery, and different proximities. Their product focuses on retroactive and manual searching of data. Users have to log into the product's interface to find evidence of a crime. The product is very expensive ($2500+) per camera.

Parkopedia [3] allows for users to enter a location and get parking locations in the area. Users like how simple yet effective the interface is. The amount of information that the product shows for each lot is extensive. However, users do not like that there is a prompted login screen, some advertisements, and some interface lag on mobile devices. This product is unique because it has extensive location support that covers hundreds of different countries with extensive information about each garage.

4.1.3 Technical Complexity

The design consists of multiple components/subsystems that each utilize distinct scientific, mathematical, and engineering principles. Major components include ESP32s with camera modules, a Raspberry Pi Gateway, a Back-end Server, and a mobile application.

The ESP32 component demonstrates engineering/mathematical/scientific principles. Firstly, the cameras demonstrate embedded systems design and integration with digital cameras. Additionally, the cameras must operate within tight hardware constraints while maintaining performance. The modules must maintain reliable wireless communication with limited power usage. The ESP32 module is the least technically complex module developed for this project, but the complexity of this module should still be considered.

The Raspberry Pi Gateway demonstrates key engineering principles, including edge computing and distributed systems. Further, this component utilizes YOLOv8 to analyze the photos taken by the camera modules and assign confidence values for vehicles. Computer vision is a complex mathematical concept that increases the complexity of the project. The Pi also needs to receive and analyze data from multiple camera modules, process the data, and send it to the server. Handling a distributed system increases the complexity of the project as well.

Without a properly designed back-end, it is difficult to maintain and refactor the link between the application and the hardware. The back-end server utilizes principles such as containerization, RESTful development, and database design to maintain performance and improve durability. Additionally, maintaining accurate occupancy with concurrent users greatly increases the complexity of the project.

Finally, the mobile application demonstrated scientific principles such as event-driven architecture and engineering principles such as UI/UX design and state management. Designing a UI that works with multiple operating systems increases the technical complexity of the project. Some of the complexity is mitigated by react-native libraries, but this still requires additional consideration.

This smart parking system represents a complex, integrated engineering project involving embedded systems, edge computing, real-time data processing, cloud services, and mobile development. Each subsystem operates under distinct constraints and employs specific scientific or engineering principles.

4.2 Design Exploration

4.2.1 Design Decisions

1. Using Spring Boot for the backend application.
    a. Spring Boot was chosen for the backend application because the senior design team is familiar with the framework, and it provides robust support for implementing complex data relationships with minimal configuration.
2. Employing Stripe to make payments secure and streamlined
    a. Employing Stripe significantly reduces development time and ensures strong compliance with security standards. It improves user trust by offering a familiar and polished payment experience, likely boosting conversion rates and minimizing payment errors.
3. Determining the transmission of images (transmit from cameras or request from Pi).
    a. The decision to periodically transmit images to the Raspberry Pi came from a middle-ground of complexity and efficiency. Adding a Pi for processing within the lot creates a more modular and secure system, but adds complexity. No images are transmitted to the server, and the ESP32s aren't connected to the user's network. Hosting a server on the Pi allows cameras to save power by transmitting and then going into a power-saving mode.

4.2.2 Ideation

The placement of the cameras will impact the hardware that can be used (cables, WiFi, batteries), accessibility for maintenance, code functionality, and many other aspects of the project. The design decisions were developed through discussions and brainstorming. This resulted in multiple potential approaches, including:

- **One camera at each entrance/exit**: By monitoring the accesses to the parking lot, the state of the whole lot can be assumed. However, the status of each spot can't be obtained.
- **One camera monitoring each parking spot**: This implementation would place a camera directly facing each parking spot.
- **Motorized rail with a moving camera**: By moving the camera, the camera can be set to predetermined positions for each spot.
- **One camera in the middle of a lot**: This approach would utilize a ceiling and a 360-degree camera.
- **One camera for each row**: A camera would be placed at an angle toward the row. This would display each car/license plate in the row at once.

4.2.3 Decision-Making and Trade-Off

To decide which approach to use, each potential decision's pros and cons were evaluated by discussion and mind-mapping. One example of the team's decision-making is shown in the placement of the camera. These are the benefits and disadvantages of each option:

1. **One camera at each entrance/exit**: This would decrease reliability, increase the software complexity, and ultimately would not meet the requirements of the project. However, it would save money, reduce hardware complexity, and allow for video processing.
2. **One camera monitoring each parking spot**: Having one camera per spot would be the most robust solution and would require the least complex software. However, it would be

the most expensive, require the most installation, and require ceilings or poles in line with each spot.

3. **Motorized rail with a moving camera**: This would require the least amount of cameras, would require extra overhead to move the camera, and would not be a modular system.
4. **One camera in the middle of a lot**: Although this may work for the parking lot that the project is being tested on, it would not work for others; this does not meet the project requirements.
5. **One camera for each row (chosen)**: This was the decision that was chosen. This was chosen by compromising software complexity and hardware installation. The result is not as invasive as having a lot of cameras, is modular, and will maintain reliability.

One camera for each row was chosen because it provides a trade-off between cost, installation effort, reliability, and software complexity. It meets the requirements of the project while still allowing for modularity in new parking lot layouts.

4.3 Proposed Design

4.3.1 Overview

The design has three main components, each with a team: Frontend (Mobile), Backend, and the Raspberry Pi. Connected to the Raspberry Pi, cameras capture data about parking spot occupancy. The Raspberry Pi then sends this data to the backend, which stores it for future retrieval. Lastly, the mobile app allows users to observe which parking spots are open, reserve spots, and find their car by license plate.

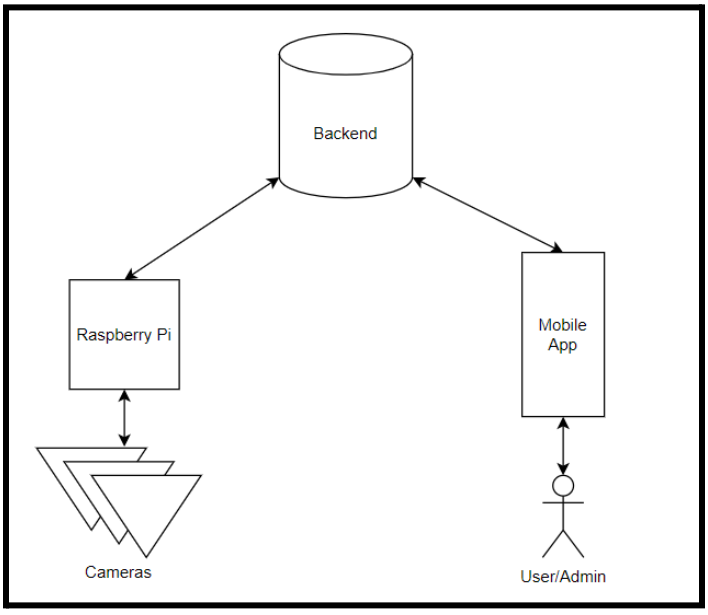

Figure 4.3.1: Simplified Design Flow

4.3.2 Detailed Design and Visual(s)

This diagram shows the organization of the three main components of our design.

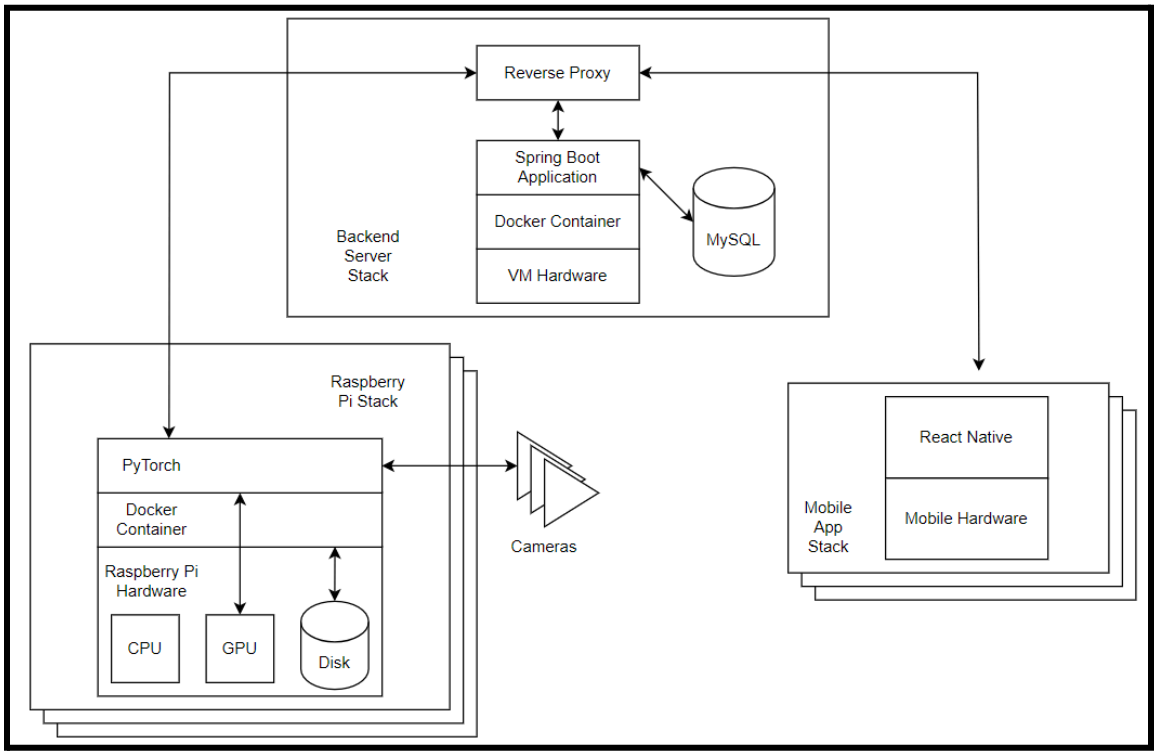

Figure 4.3.2: High-Level Design Flow

The Raspberry Pis used in the project have three main hardware components being utilized: the CPU, GPU, and Disk storage. The current design has a PyTorch application running inference operations on the GPU, and the Docker Container saves persistent data between image runs to the disk. The Raspberry Pi collects camera image data to identify spots and locate license plates. The HTTP protocol sends the parking spot location data to the backend.

The backend consists of a reverse proxy, a Spring Boot application, a Docker container, and a MySQL database. The reverse proxy is set up to take requests from outside campus networks and forward them to the VM server. On-campus services prohibit open ports to campus VMs, so a reverse proxy was implemented. Utilizing Spring Boot, the backend can handle and manage user requests. Having a Docker container allows the backend application to be portable and scalable. The backend then saves data persistently on a MySQL database.

Users interact with the product through the mobile application. The mobile application was developed using React Native. This app will interact with the backend, fetching and storing parking lot data while holding user data.

4.3.3 Functionality

The system is designed for two different types of users: the default user and the admin. The default user utilizes the application for the built-in features. The admin user will manage and set up the parking lot for the default users. There will be a GUI that allows admin users to configure parking lots (create and delete spots, set up cameras).

4.3.4 Areas of Concern and Development

The current design meets the basic needs of the average user. In the future, changes will be implemented in the design to incorporate a broader range of users. Additionally, parking lot administration is not implemented.

The primary concerns include making a usable and intuitive application. In the mobile application, many users need to be considered to make the application easy to use. This means a lot of features need to be included. However, the app still needs to be simple to maintain usability. The hardware also needs to be usable for the administrator of the parking lot while being applicable for as many parking lots as possible. Balancing all these factors is the primary concern, but they will be resolved through further discussion with the client.

Regular meetings with the client are being held to address the concerns and needs on both sides. The development process so far has been successful, so continuing to make time and put in effort to the project should yield good results. Meetings and work time are being held every week.

4.4 Technology Considerations

Because of the many parts required for the project's design, many different technologies had to be selected. The trade-offs for each one were discussed to obtain the most reasonable solution. These are the technologies that are being utilized:

- **React Native**: React Native was chosen to develop the application because of its ability to create apps that work on IOS and Android without too much extra code. With React Native, generic elements are converted into native OS elements; this avoids duplicate code.
- **Spring Boot**: Spring Boot is a production-ready software that allows for rapid development and scales well. Some negatives about using Spring Boot are the high memory cost and the hidden complexities with technologies such as auto-configuration.
- **MySQL**: MySQL was selected as the relational database management system because of its reliability and open-source licensing. MySQL gives efficient storage with querying capabilities for structured data making it easy to manage user and parking lot information. While a database management system like PostgreSQL has more advanced features, the project does not require those capabilities, and MySQL's more straightforward configuration was aligned to this project.
- **Docker**: Docker was chosen for its scalability and real-world uses. By utilizing Docker, code and updates can be deployed easily to hardware. Docker is an industry standard, so issues have been documented and solved. This adds complexity to the development, but it makes the product scalable.
- **Raspberry Pi**: The Raspberry Pi has integration with its AI hat, and it is one of few products that can host the chosen model. Raspberry Pi has lots of features that allow for easy integration with the other components in the system. Although it is more expensive at production-scale than a custom computer, it is a good fit for the prototyping stage.

- **YOLOv8**: YOLOv8 is capable of high accuracy while still being considered a fast model when hardware acceleration is used. It requires training, but that overhead is just in the development stage. Once it is deployed, it will have low latency and confident predictions.

4.5 Design Analysis

The current design has all of the components implemented. Testing on each component has been successful, but system testing will demonstrate if the previously outlined design will be sufficient. The image processing model successfully recognizes vehicles given test images, the Raspberry Pi monitors cameras and transmits data, the app runs on emulators, and the server is successfully communicating and storing information.

Future design will include connecting the components, creating a more testable product. Specifically, integrating the AI model into the current Raspberry Pi loop, getting the app to utilize data on the server, and establishing which data needs to be transmitted to the server from the Pi.

The scope of this project has increased as development progressed. Creating a deployable system that is easy for users to set up without programming or networking experience has been challenging. This has impacted our design, but this is just another example of a problem the team has solved.

5 Testing

5.1 Unit Testing

Hardware Unit Tests:

Check for transmission of pictures while evaluating latency and resolution. This test will be performed in various temperatures, brightnesses, and ranges.

CV Efficiency:
Evaluate the computer vision model's accuracy across various video inputs, including high and low visibility scenarios with differing traffic densities.

Temperature:
Ensure the Pi's components remain within safe operating temperatures (throttling begins at 85°C). The CV model will be run at different ambient temperatures—room (72°F), hot (100°F), and cold (32°F)—while monitoring the CPU and AI accelerator to identify any thermal issues.

Connectivity:
Test the Pi's ability to connect to the school's network by streaming data from the parking lot to the server over an extended period. The hardware team will monitor data loss, connection speed, and dropouts.

Database Logic:

To ensure the backend code is storing and fetching the correct results to ensure proper behavior for the front-end.

Software Unit Tests:

The front-end team uses component-level testing to prevent unexpected behavior or micro failures within React modules. The team uses the React Testing Library to ensure the user interface functions correctly and plans to use Jest to test the JavaScript behavior.

5.2 Interface Testing

Hardware System Interface:

The hardware system consists of several components: the Raspberry Pi with an AI accelerator, multiple ESP32s, and cameras connected to both the Pi and the ESP32s. The Raspberry Pi serves as the central node, sending processed information to the server, while the ESP32s and their connected cameras relay video back to the Pi for processing. The ESP32s send pictures to the Pi via the HTTP protocol. This connection will be tested by continuously sending photos from the ESP32s to the Pi in various conditions (temperature range, power outages) while verifying data integrity and connectivity.

Hardware to Server Interface:

The Pi will stream data related to detected cars and license plates to the server for processing and integration into the application. This data will be sent in JSON format. A collection of pictures with known data points will be given to the Pi's computer vision model to test the accuracy. Then, the expected results will be compared to the actual values received from the hardware. This approach also allows for scaling the number of inputs to match the expected number of cameras in the parking lot.

5.3 Integration Testing

ESP32 and its Camera to Pi Connection Integration:

This integration is critically important as it allows for scalability within the system. The connections between the ESP32s and their cameras allow us to monitor all the spots in the lot. This integration will be tested by incrementally increasing the number of ESP32 nodes connected to the Pi and streaming video at each point, monitoring the data's integrity. The team will then scale the integration by adding additional nodes (ESP32s and their respective cameras) and re-conducting the test.

Hardware Server Communication Integration:

For the application to work, the hardware team needs to connect the hardware that collects information from the Pi to the server. The connection between the Pi and the server serves as a critical integration point in the system. The data will be formatted according to the specifications of the software. The team plans on testing this integration by running the model on video input with known values and then cross-checking the results with them (this was outlined above).

Server to Application Integration:

The front-end team has already run integration tests with the account registration process, where the application cross-references the user-inputted email address to ensure validity and then processes a POST request to the Spring Boot API on the live server. This integration was tested with the Login process, where the application checks the user-inputted fields and returns a valid, already registered user account pulled from a live server GET request. The front-end team also used a dummy parking lot entity with a set remaining parking capacity enum value in the back-end to pull the capacity field from and display it on the Map Page upon selecting the specific lot. When the Expo application pulls the capacity data, it checks the spot's "status" field and calculates the number of available spots. Future front-end integration tests will be handled similarly, where the pulled and pushed data to the live server is manipulated and compared on the application side.

5.4 System Testing

The user experience will be emulated to conduct complete system tests. When the whole system is assembled, edge cases may appear that haven't been considered in development (a license plate is improperly mounted, the user has an outdated phone, etc.). The system tests will be updated to account for these cases, resulting in a robust testing process as the product is used.

The stress testing on the individual system components will continue to be tested while the whole system runs (temperature testing for the Pi, heavy user requests from the application, etc.). Doing so will avoid unexpected outages.

5.5 Regression Testing

Unit tests cover software regression testing. Running the same unit tests as the software changes ensures no potential regressions as the front-end team continues to develop code. Although tests can be updated and added, running all the tests before deployment ensures all functionality remains working.

The hardware is responsible for capturing the most critical data for the primary process of this project, so regression testing with the hardware implementation is crucial to the team's success. All the features can be verified by checking a collection of tests before deploying the code to the Pis. Also, running the same Docker container on all devices allows new device types to be used. The old devices will still be tested before deploying when new devices are added.

5.6 Acceptance Testing

The acceptance testing will be handled directly with the client during weekly meetings. As the team continues demonstrating new features and implementations, the client will decide whether the changes are acceptable and meet his vision for the project. Furthermore, requirements have been discussed at the beginning of the project, which will continue to be updated according to the client's concerns.

5.7 Security Testing

By using Iowa State's network for the server and Raspberry Pi, the network will follow Iowa State's standards. These standards include a reverse proxy and authenticated logins. The ESP32 nodes are connected to the Pi using a local WiFi hotspot (without an internet connection). Doing so isolates the nodes from Iowa State's network.

Additionally, the security of any Docker container can be analyzed using Docker Scan. Since all backend code and Raspberry Pi code is running within Docker containers, all the project's code can be scanned using Docker libraries.

Because of these protections, Docker and Iowa State assume the testing burden.

5.8 User Testing

Real users will be involved with iterative user testing focused on usability and user experience. Multiple rounds of testing will be conducted utilizing emulators, real iOS and Android mobile devices.

The first key testing method is usability testing. Usability testing involves the team observing users interacting with core features like account creation, map navigation, payments, and user permissions (e.g., admin features). The following method is task-based testing, where users will be assigned specific tasks and the team will evaluate intuitiveness and ease of use. Error Recovery and Role-Based testing are other methods the team will use to simulate failures and test user permissions. Finally, First-Time user testing will evaluate the onboarding and interface clarity for new service users.

Feedback will be gathered through direct observation, post-test surveys, and quantitative metrics like task completion time, success rate, error frequency, and satisfaction scores.

5.9 Results

As the early version of the project progresses, the unit and integration tests that the front-end team is deploying ensure that the application is ready for the next step. The team has run unit tests on the Login/Register with input handling and analysis. The case-sensitivity unit tests failed, and the team made the proper adjustments to meet the spec requirements regarding this edge case. Another integration test that caused a design adjustment was the failure to navigate to the Reservation page prototype when the app could not fetch parking lot capacity data from the live

server on the Map Page. The team has learned of this unique case, is currently working on the necessary debugging steps to fix this error, and has plans to communicate it to the client.

The backend code has a CI/CD deployment setup. Because of this integration, deploying code and debugging have been much more efficient. Verifying the code builds before sending it to the Raspberry Pis has caught issues. Further functionality tests will be included in the CI/CD process.

6 Implementation

For the backend implementation, there are two main sections – server setup and backend application development.

The server is currently running two containers via Docker Compose: a MySQL and Spring Boot container. When booting the server via Docker Compose, the Spring Boot application tries to connect to the MySQL server upon boot. The problem is that the MySQL server takes significantly longer to boot than the Spring Boot application. To combat this, the docker-compose.yml does a health check before booting the Spring Boot container. This check pings the MySQL container to make sure it is running before deploying the Spring Boot container. Currently, the MySQL container is running on port 3306 while the Spring Boot container is running on 8080. To ensure consistent data between Docker builds, the server has created a volume on the hard disk to maintain persistence.

For Spring Boot, there is software logic in place for people, parking lots, parking spots, reservations, and payments. The codebase is organized in a controller-service-repository model where data is received through endpoints and passed to a service layer that handles logic, eventually storing it in the repository. For the ownership model, the codebase is set up as parking lots having a many-to-one unidirectional relationship with parking spots, with the parking lot being the owning side. Parking spots have a one-to-one bidirectional relationship with reservations, with the parking spot being the owning side. Parking lots contain minimal data as of now, including the cost and a list of parking spots. Furthermore, parking spots only contain one field for the current implementation, which is an enum with the spot's current state. These include: EMPTY, TAKEN, RESERVED, and HANDICAP. Reservations hold the email of the reserver and a reference to its owning parking spot. If a reservation is made, there is logic to ensure the status always updates to RESERVED. Additionally, to manage the many dependencies being used within the Spring Boot application, Maven has been employed. This allows for easy dependency management and allows for scalability within the project.

The hardware implementation consists of connected components: Raspberry Pi, AI Hat, and ESP32 Cameras.

The Raspberry Pi has been configured to load a Docker container whenever available using Watchtower. This Docker container becomes available through the CICD process. When code is pushed to GitLab, it is also pushed to a mirrored GitHub repository. This GitHub repository checks that the Docker Container builds and then updates it. If it fails, the team will receive a Discord notification, and it won't be pushed.

The current Docker container is a server that receives images from the Pi. The Pi is configured to save pictures and camera frames to a persistent database. It also transmits status information periodically to the server.

The Raspberry Pi also hosts a hotspot. This hotspot isn't connected to the internet, but creates a network that the ESP32 cameras connect to. The cameras receive power, connect to the network, transmit a still frame, and then go back to power-saving mode.

The AI Hat has been utilized to label images and detect when cars are in spots.

Future development will include moving the car-detection code to the Docker container and combining the car-detection code with the license plate reading code.

The front-end implementation is a react-native app that utilizes Expo, Stripe, and Axios.
The app utilizes Expo to streamline the development process and manage internal routing between screens. Since Expo handles routing, we did not need to utilize the navigation library in react-native. For navigation, we leveraged Expo Router, a file-based routing system that handles screen transitions automatically based on the folder and file structure of our project. This approach eliminated the need to manually configure navigation stacks using traditional libraries like react-navigation. By adhering to a clear folder structure (e.g., placing pages in the app/ directory), Expo Router provides intuitive and scalable routing, making it easier to manage navigation logic across different screens like login, registration, the home map, and payment confirmation.

We wrapped our main application component with the StripeProvider component from the Stripe library to allow the application to employ Stripe's API. This component requires a publishable key, which we securely retrieve from the back-end API. The StripeProvider makes the Stripe context available throughout the app, allowing us to easily integrate payment elements such as the Payment Sheet or Checkout functionality on any screen. This approach adheres to Stripe's recommended best practices.

Requests to the back-end API are managed by Axios, a promise-based HTTP client. Axios was chosen due to its simplicity, robust configuration options, and widespread community support. It allows us to standardize the way HTTP requests are made throughout the app, improving code readability and maintainability.

7 Ethics and Professional Responsibility

7.1 Areas of Professional Responsibility/Codes of Ethics

This discussion is with respect to the paper by J. McCormack and colleagues titled "Contextualizing Professionalism in Capstone Projects Using the IDEALS Professional Responsibility Assessment", *International Journal of Engineering Education* Vol. 28, No. 2, pp. 416–424, 2012

Area of Responsibility	Definition	Relevant Item from Code of Ethics
Work Competence	Ensuring tasks are completed with the necessary technical skills, care, and diligence.	*IEEE 1.7*: "To seek, accept, and offer honest criticism of technical work..." – Our team regularly holds peer code reviews and design walkthroughs to maintain high technical standards and continuous improvement.
Financial Responsibility	Using resources wisely and staying within budget constraints.	IEEE 1.6: "To maintain and improve our technical competence and to undertake technological tasks for others only if qualified..." – We carefully scoped our cloud usage to avoid exceeding VM time or API limits.
Communication Honesty	Being truthful in documentation, updates, and all forms of communication.	*IEEE 1.3*: "To be honest and realistic in stating claims or estimates..." – We give accurate weekly progress updates and clearly report blockers or setbacks to stakeholders.
Health, Safety, Well-Being, Property Ownership	Protecting people and respecting intellectual and physical property.	*IEEE 1.1*: "To hold paramount the safety, health, and welfare of the public..." – We ensured our deployed system adheres to basic cybersecurity principles and doesn't compromise user data.
Sustainability	Creating systems with consideration for long-term environmental and resource impacts.	*IEEE 1.10*: "To support...sustainable development to protect the environment..." – We avoided unnecessary compute cycles and chose energy-efficient

		solutions when selecting libraries and hosting platforms.
Social Responsibility	Acting in a way that benefits society and avoids harm.	IEEE 1.5: "To improve the understanding of technology...by the public" – We designed our interface to be accessible and documented usage so non-technical users can benefit from our tool.

The area of responsibility that the senior design team has primarily followed is Communication Honesty. The team maintains transparency by following an open-source approach, making both code and documentation publicly available via the team's GitLab repository and team website. This allows the community to understand decisions made and provide questions and considerations to the team.

The area of responsibility the senior design team needs to work on is health and safety. While the team has discussed various approaches to securing the application, these implementations have been delayed due to other priorities. Moving forward, the team's goal is to put these security measures in place to ensure the privacy and safety of all users.

7.2 Four Principles

Broader Context	Beneficence	Nonmaleficence	Respect for Autonomy	Justice
Public health, safety, and welfare	Users are able to quickly find spots, reducing stress and clearing roads for urgency	System outages can cause leaders to be misdirected and lead to congestion	Users opt to enable location and notifications	Ensure coverage across neighborhoods to allow equal access
Global, cultural, and social	Allows for quicker parking and reservations to help users get to where they need to be.	The model trained on a specific area might not perform the same as in other areas	Allow users to disable data sharing	Inclusive design process and receiving feedback from all cultural backgrounds
Environmental	Cuts CO_2 costs by more efficiently guiding drivers to spots	Edge devices constantly running increase energy draw	Admins can set their Raspberry Pis to operate in low-power mode	Using low-power solutions to maintain sustainability
Economic	Lowers users' fuel usage by reducing parking search time	May strain small towns' budgets compared to bigger cities	Having tiered options	Maintaining a cheap price to allow access to low-income neighborhoods

The primary context-principle benefit is Welfare-Beneficence. This was chosen as the project's main benefit as this could increase many users' moods, helping prevent traffic violations, and helping maintain street organization. This will be ensured by creating an easy-to-use and functional UI and listening to the concerns of users.

The primary context-principle negative is Welfare-Nonmaleficence. The project will contain many users' private data (location, passwords, email, payments, etc.) as well as potential camera data of them. It is the project team's responsibility to preserve users' privacy and not let it overshadow the benefits. The team is committed to tackling security concerns while maintaining the safety and well-being of all application users.

7.3 Virtues

Thomas Olson -

Throughout our project, I have tried to show as much adaptability and initiative as possible. A lot of the frontend challenges arose, but I had to be proactive in my solutions when working with a disconnected backend while keeping the team's frontend objectives and implementation goals in mind. It is important to individually improve, and I have pulled through many times in getting the necessary features out in time, and actively taking in feedback from my other team members.

Sullivan Hart -

I have demonstrated cleanliness throughout the project. I have attempted to utilize clean code, and I believe it has resulted in a more maintainable and readable codebase. My contributions have included descriptive comments, well named variables, and robustness. This is important to help my future self, my teammates, and future senior design teams. I would like to utilize industry more. I have spent a lot of time on the project, but it hasn't all been as productive as it could have been.

Harley Peacher -

Challenges are inevitable in any complex project. Perseverance ensures that I stay committed and push through technical obstacles, which is crucial for delivering a successful outcome. During the initial stages of integrating the Stripe API, I faced several roadblocks with authentication errors and unexpected API behavior. Rather than giving up or relying heavily on others, I took the initiative to dive deep into the Stripe documentation, debug thoroughly, and seek out developer forums. Eventually, I was able to solve the issues and share a clear implementation guide with the team.

Leadership isn't just about directing others—it's about guiding, supporting, and creating a space for collaboration. It helps elevate the entire team's performance and morale. I aim to step up in upcoming development cycles by helping coordinate tasks, mentoring teammates on areas where I've gained expertise (like backend payment flows), and initiating team check-ins to ensure we stay on track.

Joseph Schmidt -

I believe one virtue that I have showcased in our team is honesty. I feel that when group members ask about new features, I gave a reasonable timeline and accomplished said features. A virtue that is important to me, I would like to implement more is communicativeness. I would like to update the team weekly on progress and current efforts.

8 Closing Material

8.1 Conclusion

The project is currently a group of connected components. The frontend 's map, payment, and reservations features implementation have been implemented. The server has endpoints receiving and providing information for the frontend and hardware. The hardware has all the components communicating with each other, and transmits the appropriate data within the system.

This meets the goal for this semester, to have a functioning prototype to improve upon.

Next semester, the prototype will be easier to implement for a non-experienced user, and it will have more features like license plate recognition and a variety of spot types.

By working over the summer and maintaining the same enthusiasm for the project, next semester's goals can be achieved.

8.2 References

[1]"sddec24-17 • Creating an Automated Parking System Enabled by IoT Technology," *Iastate.edu*, 2025. https://sddec24-17.sd.ece.iastate.edu/ (accessed May 03, 2025).

[2] "Safety for every situation," Flock Safety, https://www.flocksafety.com/ (accessed May 3, 2025).

[3] "Find parking, car parks, street parking, private garages - book parking," Find Parking, Car Parks, Street Parking, Private Garages - Book Parking, https://www.parkopedia.com/ (accessed May 3, 2025).

9 Team

9.1 Team Members

- Sullivan Hart
- Joseph Schmidt
- Thomas Olson
- Harley Preacher

9.2 Required Skill Sets

- React Native
  - App development
- API Integration
  - App features
- MySQL
  - Server and Raspberry Pi
- Docker
  - Server and Raspberry Pi
- Embedded C
  - ESP32 Cameras
- Python
  - Raspberry Pi
- Soldering
  - Camera Modules
- Git
  - Collaborative development

9.3 Skill Sets covered by the Team

- React Native
  - Harley and Thomas
- API Integration
  - Harley and Thomas
- MySQL
  - Joseph
- Docker
  - Joseph and Sullivan
- Embedded C
  - Joseph and Sullivan
- Python
  - Joseph and Sullivan
- Soldering
  - Joseph and Sullivan
- Git
  - All team members

9.4 Project Management Style Adopted by the team

The project will use waterfall and agile methodology. By breaking developoment into phases, each group member's contributions will complement one another – waterfall management style. Once the project progresses through the research, development, and integration phases, the team will have a product that can be tested and refined – an agile workflow. This will guarantee continuous improvement in the project.

9.5 Initial Project Management Roles

1. Frontend design lead: Thomas Olson
2. Backend design lead: Joe Schmidt
3. Client interaction lead: Sullivan Hart
4. Team manager: Harley Peacher

9.6 Team Contract

Team Members:

5. Thomas Olson
6. Harley Peacher
7. Sullivan Hart
8. Joseph Schmidt

Team Procedures:

Weekly team meeting, 10:30 AM on Discord. Wednesday 12:30 PM client meeting. Preferred method of communication for project updates and scheduling is Discord. Pressing reminders will be communicated on text messages. A majority vote for important decisions if a consensus cannot be reached. Records will be kept in a shared folder in Google Drive.

Participation Expectations:

1. All group members are expected to arrive on time and if members can not attend letting other team members know ahead of time.
2. The team member assigned to a task is responsible for its completion. If an individual can't complete their task on time, they are expected to alert the other members. Team assignments will be completed on time without exception.
3. The team is expected to communicate frequently about collaborative components (API, team assignments, etc.), and report progress of their components at weekly meetings.
4. Team members are expected to follow the consensus of the group. The team will work towards unanimous decisions. However, non-unanimous decisions will be respected with the same level of commitment.

Leadership:

- Frontend design lead: Thomas Olson
- Backend design lead: Joe Schmidt
- Client interaction lead: Sullivan Hart
- Team manager: Harley Peacher

Collaboration and Inclusion

The team will encourage and support contributions/ideas from all team members by hearing each member's suggestion and fully discussing them. Collaboration issues will be identified and addressed as an entire team; team members will listen and find a long term solution.

Goal-Setting, Planning, and Execution

The team aims to have a prototype by the end of the first semester with one parking spot monitored. The app's frontend and backend will be working. The hardware will be connected with the app.

To meet the goals, the team will meet weekly and assign the individual and team work. They will stay on task by setting attainable weekly goals.

Consequences for Not Adhering to Team Contract

1. Warning - warn the individual and make the infraction clear
2. 2nd warning - second warning and threat to contact professor
3. Contact the advisor Professor
4. Senior design professor contacted

****

a) I participated in formulating the standards, roles, and procedures as stated in this contract.

b) I understand that I am obligated to abide by these terms and conditions.

c) I understand that if I do not abide by these terms and conditions, I will suffer the

consequences as stated in this contract.

1) *Thomas Olson* DATE *2/10/2025*

2) *Harley Peacher* DATE *2/10/2025*

3) *Sullivan Hart* DATE *2/10/2025*

4) *Joseph Schmidt* DATE *2/10/2025*