

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: 12/09/2025 / v3.01

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.
- SpringBoot/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.
- Arducam 16MP Wide Angle USB Camera: High-resolution camera that monitors the parking lot. The Pi requests an image, and the camera captures the block of the lot it is assigned to monitor.

The system uses the Raspberry Pi as a server for each lot. Each camera captures a view of the block assigned to it when requested by the Pi. The Pi then processes the frame using YOLOv8, identifying the number of parked vehicles. Then, it checks for a license plate on each car and finds possible matches. Lastly, the Pi sends the information to the remote MySQL server. The app communicates with the server, requesting a parking lot's availability and sending user interactions.

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

Development Standards & Practices Used.....	2
Practices:.....	2
Relevant IEEE Standards.....	2
Summary of Requirements.....	2
Functional Requirements:.....	2
Non-Functional Requirements:.....	2
Applicable Courses from Iowa State University Curriculum.....	3
New Skills/Knowledge acquired that was not taught in courses.....	3
Table of Contents.....	4
List of figures/tables/symbols/definitions.....	6
1. Introduction.....	7
1.1. Problem Statement.....	7
1.2. Intended Users.....	7
2. Requirements, Constraints, And Standards.....	8
2.1. Requirements & Constraints.....	8
2.1.1. Functional Requirements:.....	8
2.1.2. Non-Functional Requirements:.....	8
2.2. Engineering Standards.....	8
3. Project Plan.....	9
3.1. Project Management/Tracking Procedures.....	9
3.2. Task Decomposition.....	9
3.3. Project Proposed Milestones, Metrics, and Evaluation Criteria.....	10
3.3.1. Milestones, Metrics, and Evaluation Criteria.....	10
3.4. Project Timeline/Schedule.....	12
3.5. Risks and Risk Management/Mitigation.....	15
3.5.1. Hardware Development Risk Management/Mitigation.....	15
3.5.2. Software Development Risk Management/Mitigation.....	16
3.6. Personnel Effort Requirements.....	19
3.7. Other Resource Requirements.....	20
3.8. Project Management Methodology Shift.....	20
4. Design.....	21
4.1. Design Context.....	21
4.1.1. Broader Context.....	21
4.1.2. Prior Work/Solutions.....	21
4.1.3. Technical Complexity.....	22
4.2. Design Exploration.....	22
4.2.1. Design Decisions.....	22

4.2.2. Ideation.....	23
4.2.3. Decision-Making and Trade-Off.....	23
4.3. Proposed Design.....	24
4.3.1. Overview.....	24
4.3.2. Detailed Design and Visual(s).....	26
4.3.3. Functionality.....	27
4.3.4. Areas of Challenge.....	27
4.4. Technology Considerations.....	27
5. Testing.....	28
5.1. Unit Testing.....	28
5.2. Interface Testing.....	29
5.3. Integration Testing.....	29
5.4. System Testing.....	30
5.5. Regression Testing.....	30
5.6. Acceptance Testing.....	30
5.7. Security Testing.....	30
5.8. User Testing.....	30
5.9. Results.....	31
6. Implementation.....	31
7. Ethics and Professional Responsibility.....	34
7.1. Areas of Professional Responsibility/Codes of Ethics.....	34
7.2. Four Principles.....	35
7.3. Virtues.....	36
8. Conclusions.....	37
8.1. Summary of Progress.....	37
8.2. Value Provided.....	37
8.3. Next Steps.....	38
9. References.....	40
10. Appendices.....	41
10.1. Appendix 1 - Operation Manual.....	41
10.2. Appendix 2 - Alternative/Initial Version of the Design.....	41
10.3. Appendix 3 - Other Considerations.....	41
10.4. Appendix 4 - Code.....	41
10.5. Appendix 5 - Team Contract.....	41

List of figures/tables/symbols/definitions

Table of Figures

Figure 3.2.1: Task Decomposition Chart	9
Figure 3.5.1: High-Level Schedule	11
Figure 3.5.2: Software Schedule	12
Figure 3.5.3: Hardware Schedule	13
Figure 3.6.1: Software Development Effort Estimate	18
Figure 3.6.2: Hardware Development Effort Estimate	19
Figure 4.3.1: Simplified Design Flow	23
Figure 4.3.2: High-Level Design Flow	24
Figure 7.1.1: Areas of Professional Responsibility	36
Figure 7.2.1: Four Principles.....	37

1. Introduction

1.1. PROBLEM STATEMENT

Drivers waste valuable time trying to find open parking spaces, often circling crowded lots without direction. This inefficiency leads to frustration, congestion, and poor use of available parking. Unipark addresses these challenges by simplifying the process, reducing search times, and optimizing how parking spaces are managed.

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, as well as 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

2.1.1. 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).

2.1.2. 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 employs a hybrid approach, combining elements of both waterfall and agile methodologies. Initially, the project was broken into phases, ensuring that each group member's contributions complemented 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, a transition to an agile workflow occurred, where frequent sprint meetings ensured continuous and 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.

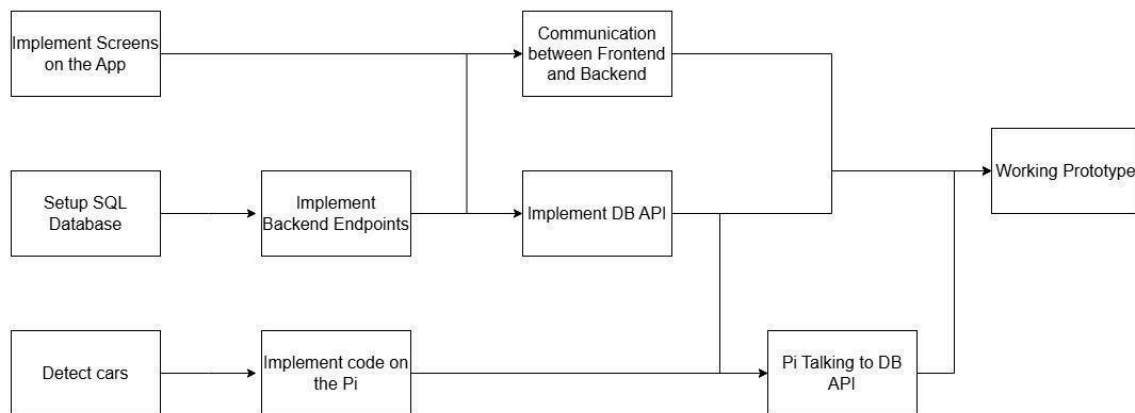


Figure 3.2.1: Task Decomposition Chart

3.3. PROJECT PROPOSED MILESTONES, METRICS, AND EVALUATION CRITERIA

3.3.1. Milestones, Metrics, and Evaluation Criteria

Front-end Iteration 1:

- 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.

Front-end Iteration 2:

- Improve User Flow
 - Create Profile Screen accessible from the map screen that allows users to logout, change passwords, manage license plates, and set preferences.
 - Merge Reservation & Payment Screens.
 - Implement login caching.
 - Convert to floating components on the map screen.
 - Adjust coloring of buttons to improve flow.
- Implement Live Map
 - Create an embedded mini map on the Reserve Screen that shows a schematic of the selected lot with color-coded polygon components representing the different blocks within the lot.
 - Fetch information like available spots and block information with live occupancy counts from the backend API.
 - Color-coded blocks by occupancy ratio, where green represents occupancy rate of < 60%, orange = 60-89%, red >= 90%, and gray = 100%, respectively.
 - A legend explains the colors so users can quickly see which sections are most full.
 - Allow selection of a block that isn't full for a reservation.
- Improve Modularity
 - Create an authentication & authorization module.
 - Utilize React Paper to reduce custom styling.
 - Create custom component files for reused components.
- Implement security features
 - Email verification & password reset.
- Integrate New Backend Features
 - Parking Blocks on reservation screen.
 - Reservation & Payment information
 - Profile features, such as password changes, and license plate management.

Back-end Iteration 1:

- 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.

Back-end Iteration 2:

- Switch to Parking blocks
- Adding Reservations
 - Parking
- Check if Reservations are in their active time
 - Scheduled job to check if reservation start time is equal to or after the current time
 - Same scheduled job should check if reservation start time plus reservation length is equal or after current time
- Find violators
 - Scheduled job to check if cars in a block have associated license plate
- Add Car Detection uploads
 - Vehicles should have the parking lot id, parking block name that they are associated with as well as a list of potential license plates it could be
- Add paid lot
 - Paid lots should display cars who are parked without a reservation
- Add notifications
 - Service to send notification to users if their reservation has been moved or refunded
 - Admins should be notified of violating vehicles

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.
- 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.

Hardware Iteration 2:

- Migrate to Raspberry Pi 5
 - Configure the Pi 5 to pull in Docker and run on boot.
- New Camera Module
 - Use a high resolution camera to monitor the parking lot.
- System Test
 - Test outside with a real parking lot.
 - Enable Wi-Fi support.
- Add find-by-plate support
 - Send a list of plates from each car object in the block.
- Connect with the rest of the system
 - Switch to block structure instead of spot.
 - Update API for server to parking lot and parking lot to server.
- Improve OCR
 - Update to PaddleOCR.

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

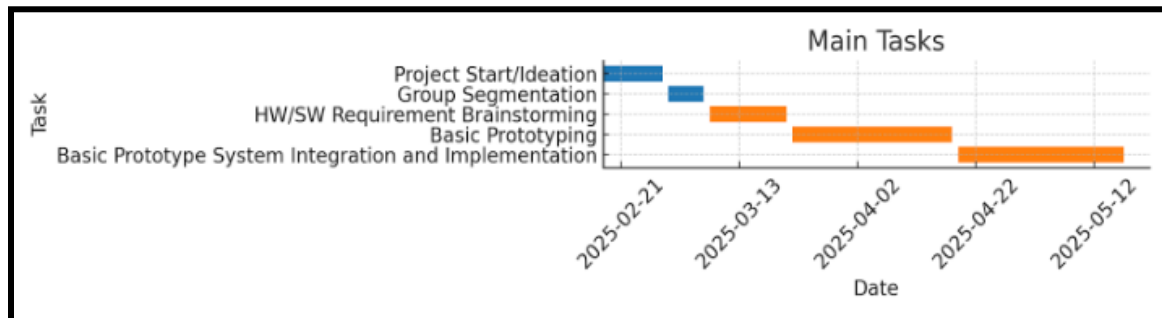


Figure 3.5.1: High-Level Schedule

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

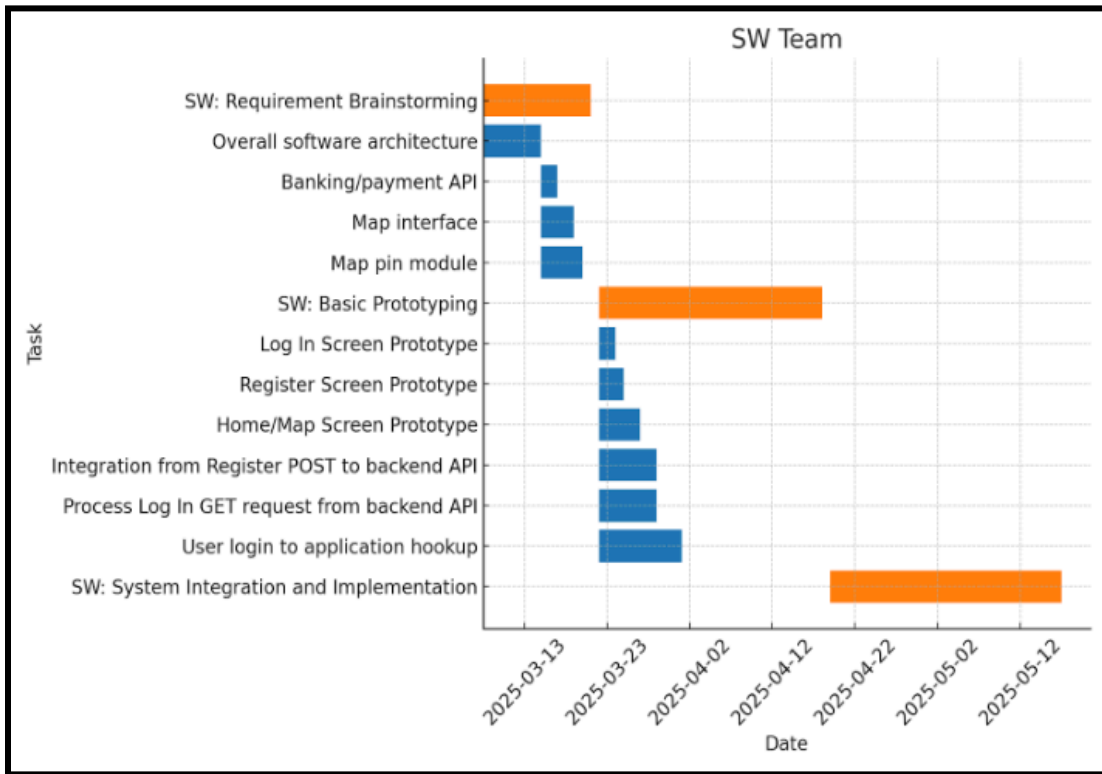


Figure 3.5.2: Software Schedule

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		

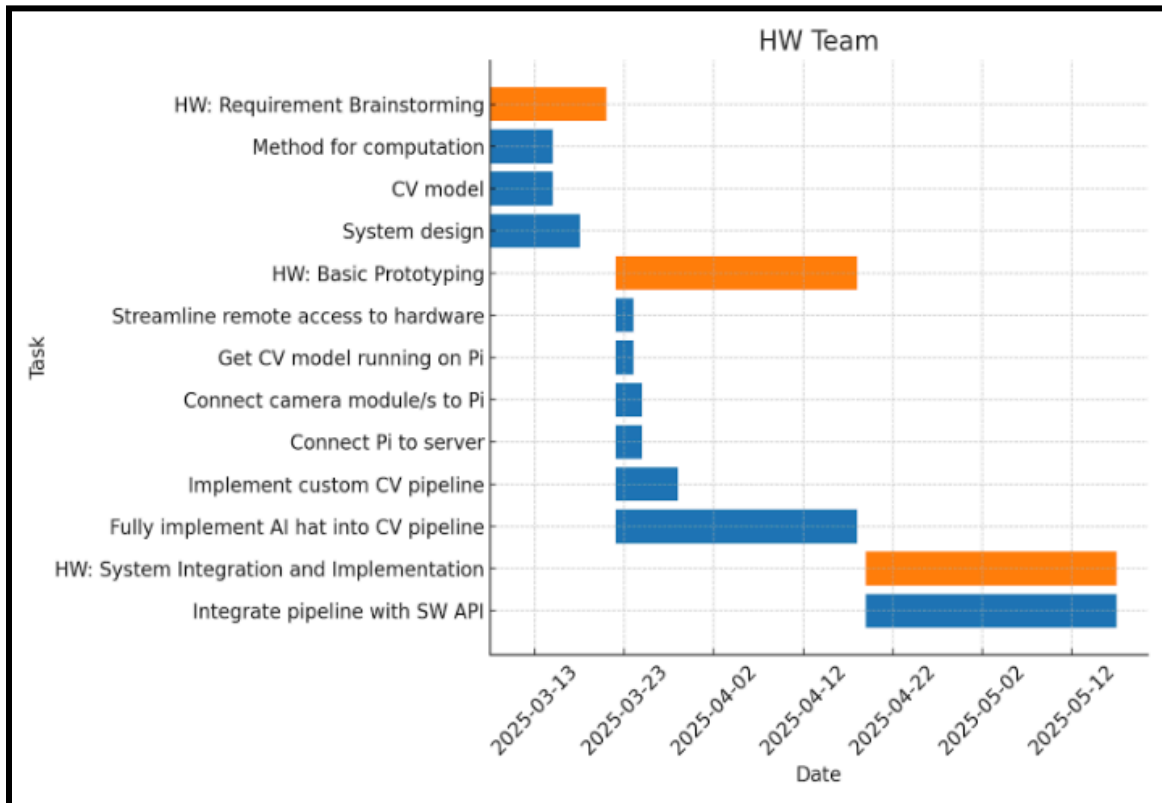


Figure 3.5.3: Hardware Schedule

3.5. RISKS AND RISK MANAGEMENT/MITIGATION

3.5.1. 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.

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.

3.5.2. Software Development Risk Management/Mitigation

Overall Software Architecture Risks

- **Risk/Constraint:** Scalability Challenges – The architecture may not effectively support growing user loads.
 - **Risk Level:** 0.4
 - **Solution/Mitigation Method:** Designing the system for horizontal scaling and using containerized services to allow for additional deployments to respond to increasing user load.
- **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, such as 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 utilized to resolve compatibility issues across various 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 HTTPS requests are used to mitigate authentication risks.

Register Screen Prototype

- **Risk/Constraint:** Spam/Fake Accounts – A 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 may not function properly 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 result in 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 5, a camera module, 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.

3.8. PROJECT MANAGEMENT METHODOLOGY SHIFT

As the team moved into the second semester, a methodology shift occurred. The team moved from a waterfall style to a more AGILE one. As a result, no Gantt charts or effort estimates were generated, as the focus was primarily on week-to-week progress rather than planning for the entire semester. This decision arose due to a significant discrepancy between our semester one actual and estimated work.

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 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 primary difference between the two implementations lies in 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 enables users to prevent crime by utilizing a website and a range of cameras, as well as to monitor vehicles. They have multiple cameras for various uses, including solar, battery, and different proximity settings. Their product focuses on retrospective and

manual searching of data. Users must log in to the product's interface to find evidence of a crime. The product is very expensive (\$2500+) per camera.

Parkopedia [3] allows users to enter a location and get parking locations in the area. Users appreciate the simplicity and effectiveness of the interface. The amount of information displayed for each lot is extensive. However, users dislike the presence of a prompted login screen, some advertisements, and interface lag on mobile devices. This product is unique because it offers extensive location support, covering hundreds of countries with detailed 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 the Raspberry Pi acting as a server for each parking lot, a remote server for all parking lots and the app's back-end, and a mobile application.

The Raspberry Pi Gateway demonstrates key engineering principles, including edge computing and distributed systems, showcasing their applications in real-world scenarios. Furthermore, this component utilizes YOLOv8 to analyze photos taken by the camera modules and assign confidence values to vehicles. Computer vision is a complex mathematical concept that further complicates 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 also increases the complexity of the project.

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 data with concurrent users significantly increases the project's complexity.

Finally, the mobile application demonstrated scientific principles, such as event-driven architecture, and engineering principles, including 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 enhances user trust by providing a familiar and polished payment experience, which is likely to boost conversion rates and reduce payment errors.
3. Determining the transmission of images (transmitted from cameras or request from Pi).
 - a. Originally, the system was designed to periodically receive images from wireless camera nodes. The Pi would hold a database of images sent by the cameras – processing the images in a queue. Although this is still an option that should be pursued, the use of wired cameras was implemented for simplicity. This allowed for easier transmission of high-resolution images that were found to be necessary after experimenting with wireless camera modules. The low-resolution images were found to not have enough clarity for license plate reading.
4. Creating the block system in place of individual parking spots.
 - a. Handling logic when a violator takes a reserved spot would require alerting the parker after moving them to a different spot. To prevent this from happening multiple times, we created the block system. Grouping multiple spots into a block would enable better handling, a simplified user interface, and more robust error handling.

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 access to the parking lot, the state of the entire lot can be determined. 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. 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:** The result is not as invasive as having a lot of cameras, but isn't modular.
6. **One camera for each block (chosen):** This was chosen due to the modularity. The administrator is able to designate blocks, and one camera is assigned to each. Along with simplification to the reservation handling, the Pi also has to only count the number of cars in frame instead of checking each spot.

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.

Additionally the team discussed moving the solution to the cloud. This would allow for better scaling and a plethora of tools. Upon exploring this approach the team quickly realized the cost far exceeded the budget of the project. As a result, the team opted for a free virtual machine given to us by the University.

4.3. PROPOSED DESIGN

4.3.1. Overview

The design comprises three main components, each with its own team: Frontend (Mobile), Backend, and Raspberry Pi. Connected to the Raspberry Pi, cameras capture data about parking spot occupancy. The Pi then sends this data to the backend, which stores it for future retrieval. Lastly, the app shows parking occupancy, takes reservations, and searches for cars by license plates.

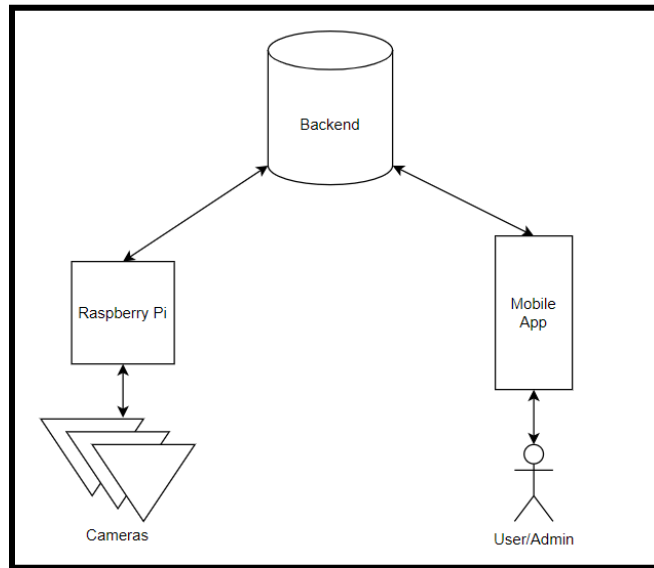


Figure 4.3.1: Simplified Design Flow

4.3.2. Detailed Design and Visual(s)

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

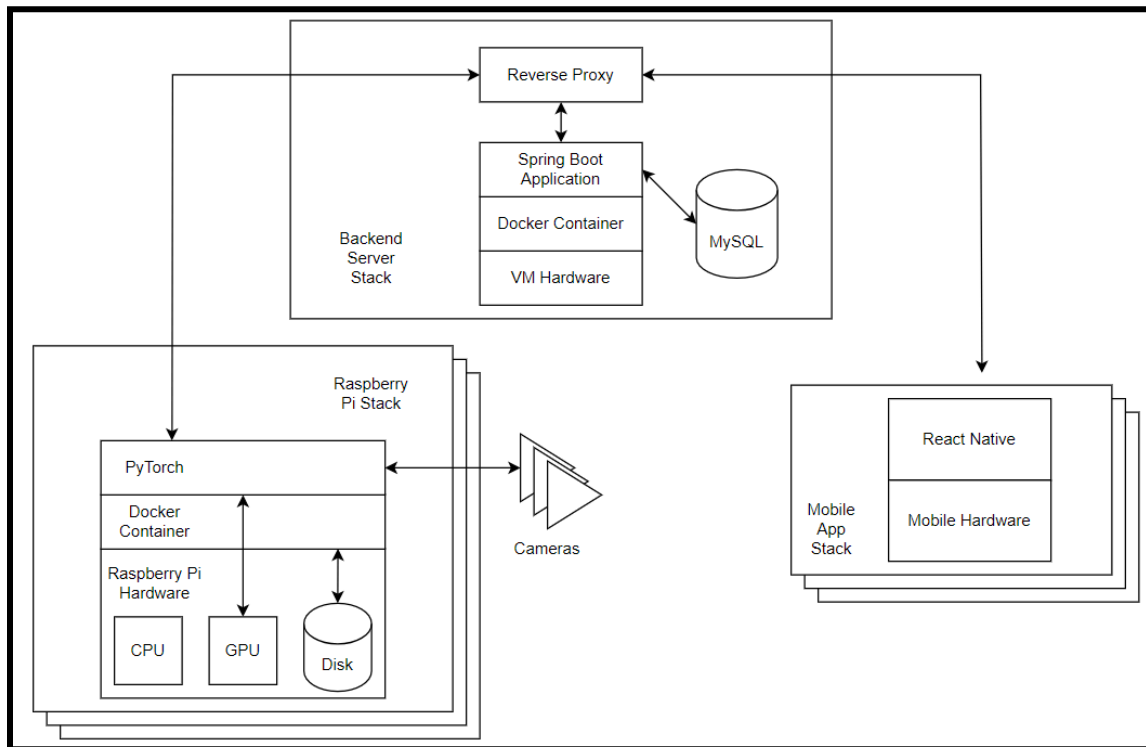


Figure 4.3.2: High-Level Design Flow

The Raspberry Pis used in the project have three main hardware components being utilized: the CPU, GPU, and Disk storage. The current design runs 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. Features include login/registration, reservations, payment, and viewing live occupancy.

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 blocks, set up cameras).

4.3.4. Areas of Challenge

The first major challenge the team faced was adjusting to the new requirements after our team size decreased. To overcome this, we were forced to delegate the work of our former teammate and take on an increased workload.

Another challenge that the team faced was finding a simple way to complete the deployment and setup of our product in a new location. If this product were to hit the market, it would need to be customizable to accommodate various lot sizes. Our admin users would likely prefer not to do this on their phones, so having a desktop or web-based GUI would be optimal.

A third challenge was enforcing reserved parking spots, as there was no indication of whether a spot was reserved or not. The team overcame this by devising the parking block solution, which meant that a parking block would have to be filled for a reservation to be affected. If it does, the reservation can be moved to a different parking block or refunded to the user.

The team wanted to move the server to the cloud for the added benefits of scalability and proprietary tooling. Upon moving to Azure [6] the team quickly realized the cost far exceeded the budget available. This resulted in a reversion to the previous VM backend server as it was free and highly customizable.

The frontend team struggled with dependencies being updated twice during development. Part of the testing process is ensuring the features work locally. Doing so requires that the version of Expo Go in our project and the version on your phone are the same. IOS does not allow you to download and run previous versions of applications. This made the testing process difficult as one member of the frontend team was simulating the application in the Expo Go app on an IOS device, but this was no longer possible after the automatic update the device pushed, which deprecated the project version. To overcome this problem, the member set up an Android emulator on their laptop and used it to simulate and test the application and working around the issues with the Expo Go IOS app. The other team member also meticulously updated every single dependency to keep the React Native application up-to-date.

The team also struggled initially with using ESP32 cameras, as the resolution of the photos was too low to obtain accurate inference results. To circumvent this, the team used a portion of the budget to purchase a high-resolution USB camera, which enabled focused images that helped yield more accurate results.

4.4. TECHNOLOGY CONSIDERATIONS

Due to the numerous components required for the project's design, various 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. Once it is deployed, it will have low latency and accurate predictions.

5. Testing

5.1. UNIT TESTING

Hardware Unit Tests:

Check for transmission of pictures while evaluating latency and resolution. This test will be performed at various temperatures, brightness levels, 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 stores and fetches the correct results, ensuring 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.

Endpoint Tests:

The backend team ensured correct endpoint behavior by running multiple edge case tests for each endpoints open on the server.

5.2. INTERFACE TESTING

Hardware System Interface:

The hardware system consists of several components: a Raspberry Pi with an AI accelerator, a variable number of cameras. The Raspberry Pi serves as the central node, requests images from the cameras, processes the images, and sends the results to the server. To test, a collection of pictures with known data points could be given to the Pi's computer vision model (without the real camera being used) to test the accuracy. System testing should be done in various parking lots in different settings. This has occurred informally, but more rigorous should be done to ensure robustness.

5.3. INTEGRATION TESTING

Pi to Camera Integration:

This integration is critically important as it allows for scalability within the system. The connections between the Pi and their cameras allow all blocks to be monitored. This integration will be tested by incrementally increasing the number of camera nodes connected to the Pi, and the performance of the Pi could be measured.

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 sample images with known values and then comparing them to the observed results.

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 utilized a dummy parking lot entity with a set remaining parking capacity enum value in the back-end to retrieve the capacity field 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 entire system is assembled, edge cases may arise that were not considered during development (such as a license plate being improperly mounted or the user having an outdated phone). 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 were handled directly with the client during weekly meetings. As the team continued demonstrating new features and implementations, the client decided whether the changes were acceptable and met his vision for the project. Furthermore, requirements had been discussed at the beginning of the project, which were continually being 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.

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 in iterative user testing, focusing on usability and user experience. Multiple rounds of testing will be conducted using emulators and 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, such as account creation, map navigation, payments, and user permissions (e.g., admin features). The following method is task-based testing, where users are assigned specific tasks, and the team evaluates 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

Unit and integration tests that the front-end team has deployed ensure that the application is ready for additional features. 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 did user tests with a few different users. The two earliest tests resulted in UI changes to enhance usability. Colors were used to represent actions and guide the user effectively. These tests also influenced the design decision to transition to the block model that the team has now adopted.

The backend code has a CI/CD deployment setup in place. 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

The Reserve Screen has a React Native Maps overlay to show a live schematic of the parking lot. When the component mounts, it triggers an asynchronous fetch to get parking lot information from the backend endpoint. The live map is built on top of react-native-maps using a MapView with polygon overlays. The component has a set of helper functions to convert backend data into map geometry with color-coding based on the parsed data from the backend endpoints.

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.

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 database through the repository layer.

For the ownership model, the codebase is set up as parking lots having a many-to-one unidirectional relationship with parking blocks, with the parking lot being the owning side. Parking blocks contain several variables for tracking block occupancy - total spots, available spots, reserved spots, and taken spots - as well as a list of reservation entities.

These reservation objects the following variables: licensePlate, reservationStartTime, reservationDurationMinutes, moveorRefund, ReservationStatus, and numberOfMoves. Reservations work by being in one of three states at all times - ACTIVE, EXPIRED, or UPCOMING. These relate to the scheduled event algorithm that is constantly run on the backend.

The backend has a scheduled event that takes place every ten seconds (configurable value). This event is to update the occupancy of the parking lot, ready reservations, and check for violators. Ready reservations involves checking to see if the start time is equal or later than the current time. Before it's turned to active though, there must be a check to see if the block the reservation is for has available spots. If not, one of two things happen based on the moveOrRefund boolean within the reservation object: true, move the reservation to an adjacent block; false, refund the user. This check happens a specified time before the reservation (configured by parking lot manager).

The parking lot manager additionally sets a max number of reservation moves to happen. Inside the reservation object there is a `numberOfMoves` variable tracking the move count and upon reaching the set max number of moves the reservation will be refunded.

If the parking lot is configured to be a paid lot then the scheduled event checks for violating cars. This is done by iterating through all cars and checking to see if a reservation is found for it in the block it is associated to. If not the car will be added to a violators list.

Additionally, to manage the many dependencies being used within the Spring Boot application, Maven has been employed. This enables easy dependency management and promotes scalability within the project.

The hardware implementation consists of connected components: Raspberry Pi, AI Hat, and 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 AI Hat has been utilized to label images and detect when cars are in spots.

The front-end implementation is a react-native app that utilizes Expo, Paper, Stripe, and Axios. The app utilizes Expo to streamline the development process and manage internal routing between screens. We leveraged Expo-Router for page navigation, 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, such as login, registration, the home map, and payment. Authentication on the frontend is implemented using `AuthProvider`, the login screen delegates authentication to a shared `AuthProvider` via a `useAuth` hook. This contains the logic for sending credentials to the backend and caching the authentication state.

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.	<i>IEEE 1.7</i> : “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.	<i>IEEE 1.6</i> : “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.	<i>IEEE 1.3</i> : “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.	<i>IEEE 1.1</i> : “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.	<i>IEEE 1.10</i> : “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.

Figure 7.1.1: Areas of Professional Responsibility

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 implement these security measures to ensure the privacy and safety of all users.

Another area of responsibility that the team discussed in the second semester was the ethical responsibility of collecting users' license plate information. Even if this data is necessary for reservations and linking vehicles to accounts, it is also sensitive information that could be misused in the wrong hands. Under IEEE Code of Ethics 1.1 and 1.5: prioritizing public welfare and respecting privacy, we recognize our obligation to restrict access to only authorized system components. This way, users are protected from privacy risks.

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, helping 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

Figure 7.2.1: Four Principles

The primary context-principle benefit is Welfare-Beneficence. This was chosen as the project's main benefit, as it could improve the moods of many users, help prevent traffic violations, and maintain street organization. This will be ensured by creating an easy-to-use and functional UI, as well as by addressing 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 balance users' privacy with 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. Many 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 essential to continually improve individually, and I have consistently delivered the necessary features on time, actively incorporating feedback from my 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 providing descriptive comments, using well-named variables, and ensuring robustness. This is important to help my future self, my teammates, and future senior design teams. I would like to utilize the industry more. I have spent a considerable amount of time on the project, but it hasn't 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 encountered several roadblocks, including 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 collaborative environment. 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 in areas where I've gained expertise (such as 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 provide a reasonable timeline and accomplish those features. A virtue that is important to me, I would like to implement more of is communicativeness. I would like to update the team on a weekly basis regarding progress and current efforts.

8. Conclusions

8.1. SUMMARY OF PROGRESS

During the second semester, the frontend evolved into a more cohesive application completed with features that align closely with the overall project goals. Some of the major application accomplishments was improving the user flow by implementing a Profile Screen accessible right from the map, and merging the reservation and payment processes into a unified flow to help a smoother user experience. Another helpful integration was transitioning UI components to React Paper that supports dark-mode and unified styling, and added reusable component files to help with future development.

Another key frontend accomplishment in this second period was the implementation of the Live Map. This embedded mini map on the Reserve Screen renders color-coded polygon blocks that represent the occupancy within specific regions of the lot. This gives users immediate feedback on lot capacity and is supported by a clear legend.

As for the backend, the switch over to parking blocks was successfully completed. Reservations were added with the ability to reserve in advance and for specified blocks. In the case were a parking block becomes full near a user's reservation time, the system can automatically move the reservation to an available block. Payments are processed securely through the Stripe API. Also, the backend supports car detection uploads with the ability for admins to mark unreserved cars as violators. Lastly, parking lot metrics are recorded, including the number of cars parked, reserved spaces, and active violators.

The hardware experimentation has revealed a lot about potential implementations. Lots of experimentation led to the conclusion that the Raspberry Pi 5 with the AI accelerator is sufficient for the task that was requested. Computing on the edge has proven to be an appropriate choice, and high resolution cameras are capable of detecting the capacity of a parking lot. The use of high resolution cameras also showed that the richer data than IR sensors can enable more features like license plate lookup.

8.2. VALUE PROVIDED

The design addresses the needs of some of the rushed, patient, and proactive parkers. The rushed parker wants to get through the flow and secure the first available spot. The simplified block model allows their needs to be addressed. The patient parker wants to find the perfect spot. The design enables users to view occupancy and select the block that best suits their needs. The design allows the proactive parker to reserve a spot before arriving at the lot.

Due to design constraints, we were not able to incorporate handicap spots, which limits the use for elderly/disabled parkers. The forgetful parker's needs would be addressed by the "find your car" feature. The infrastructure is incorporated into the design, but isn't served to the user to avoid privacy issues. However, adding limitations (GPS, authentication, etc.) could allow this feature to be ethically implemented.

The admin user's needs are not addressed within the design. An admin website with ability to handle reservations, manage/create new lot and block designations, and modify user data would be required to address the needs of the admin user, which fell out of scope.

8.3. Next Steps

Frontend:

Designing a website for admins to set up, moderate, and manage their lots would be a great next step for this project. Currently, lots and blocks are hardcoded for setup. A mechanism to set up the camera locations and block assignment would improve the deployment of the product.

Expanding the app to give users more control over their reservations is another good step to progress this project. Currently, users can create a reservation and complete a payment, but allowing them to modify, extend, or cancel their reservations directly from the app without any administrative support would make our system much more user-centered. Some more features like time reservation status, reminders, and ability to switch blocks would also be beneficial to users.

Another quality of life step would be to enable real time updates using a technology like WebSockets to give instant updates when cameras detect occupancy changes or when reservations are created or canceled. Current implementation fetches parking lot data when the screen mounts but this step would be beneficial for users.

There are other small improvements that can enhance the product, including email verification, backend session management/expiration, license plate registration, 'Find My Car', and additional preferences.

Backend:

A major next step would be Spring Security [4]. The app lacks security as data communications and user data are unencrypted. Using Spring Security would allow the project to encrypt sensitive user data and secure our endpoints via the HTTPS protocol.

There was an attempt to add Swagger[5] into the backend. It currently is dated and needs improvements to become operable.

There is some fault tolerance where a car will need to get detected for multiple update cycles. This was not tested in the real world for an ideal value. This should be changed.

The backend is comprised of multiple variable values that can be configured. These should be moved into config files that can be changed from a single location.

A CICD pipeline would be a great addition to improve efficiency and reduce hassle of manually swapping iterations.

Creating defined Unit tests would help improve validity and would help ensure past functionality.

Currently, there is no communication to users about violators or reservation updates. Implementing a notification service to send notifications such as reservation moves and refunds, as well as violators to parking administrators would improve the system's transparency.

Administrator and user profiles are currently underdeveloped. Additional features should be added such as preferences, and user settings.

In the code there is a placeholder for the refund reservation method. There should be logic implemented to refund a user in the case their reservation is occupied.

Integrating the application with the ISU parking division permits or building a system to integrate permits for a given lot would improve the design's usability.

Allowing a reservation for a specific time rather than 5 minutes away. Being able to reserve for later in the day would be an improvement to the current system. Giving the user more flexibility. This change introduces a significant amount of complexity to the existing reservation system. The system would have to check availability during a given time and for the duration of the reservation.

Hardware:

Make the Pi a more closed system (configure for SSH only, file permissions, etc.).

Add configuration process for administrator.

Revisit multi-cam support. This may have to include wireless nodes (cameras connected to Pi Zero or similar board).

Utilizing LED lights to represent whether a spot is reserved would improve backwards compatibility.

9. 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).
- [4] "Spring security," Spring Security :: Spring Security, <https://docs.spring.io/spring-security/reference/index.html> (accessed Dec. 6, 2025).
- [5] "Swagger Documentation." Swagger. <https://swagger.io/docs/> (accessed Dec. 6, 2025).
- [6] Microsoft, "Data Privacy in the Trusted Cloud – Azure," Microsoft Azure, Accessed: Dec. 9, 2025. [Online]. Available: <https://azure.microsoft.com/en-us/explore/trusted-cloud/privacy>

10. Appendices

10.1. APPENDIX 1 - OPERATION MANUAL

Link to the Operation Manual: [📖 Operation Manual](#)

10.2. APPENDIX 2 - ALTERNATIVE/INITIAL VERSION OF THE DESIGN

- *Versions considered before client's specifications have changed*
- *Versions considered before learning more about the project*
- *Versions that resulted in failure to achieve specifications, etc.*
- *Describe each major design version and why they were scrapped/revised*

Initially, the design included LED lighting to display the availability of a given parking space. Due to a reduction in work capacity, that was not pursued. Removing the LEDs reduces the backwards compatibility of the design. Users who do not use the application would not be able to see whether a spot is reserved. The lack of a visual indication would mean we could not punish non-users for taking reserved spots. If there is no way to ensure the reserver will be able to park in their spot, there is no incentive to reserve a spot.

The camera configuration was redesigned multiple times throughout the two semesters. Potential designs included one camera on each entrance / exit of the lot; this design would show capacity, but would not work with reservations or find-by-plate. ESP32s were also tried to potentially have low-cost and capacity-only nodes. The resolution was not good enough, and they struggled to have enough range for a practical implementation. Finally, a high-resolution camera was chosen for each block. This simplified the process as it didn't need to detect the spot that each car is in and included license plate data.

10.3. APPENDIX 3 - OTHER CONSIDERATIONS

- *Any miscellany you deem important, what you learned, anything funny, anecdotes from your project experience*

10.4. APPENDIX 4 - CODE

<https://git.ece.iastate.edu/sd/sddec25-09>

10.5. APPENDIX 5 - TEAM CONTRACT

Team Members

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

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

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

Project Management Style Adopted by the team

The project will use waterfall and agile methodology. By breaking development 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.

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

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. The 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