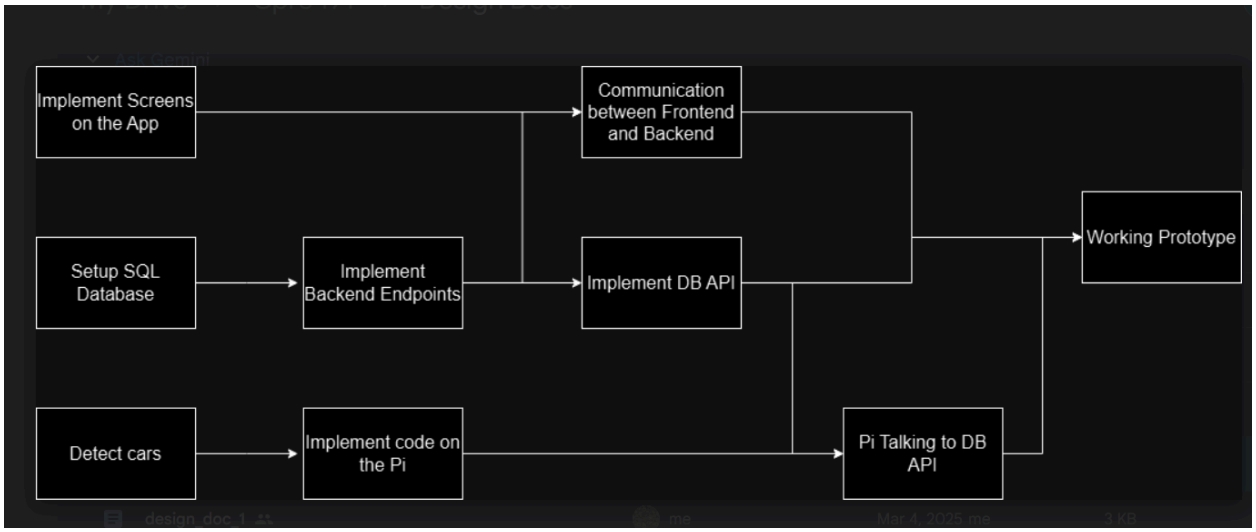
# 3 Project Plan - sddec25-09

## 3.1 PROJECT MANAGEMENT/TRACKING PROCEDURES

The project will follow a hybrid waterfall and agile methodology. Initially, breaking the project into phases will ensure that each group member's contributions complement one another—this represents the waterfall portion. Progressing through the phases of research, development, and integration will lead to a working product that can be tested and refined. At that point, the team will transition to an agile workflow. By meeting frequently to review sprints, the team can continuously improve the project toward its best possible outcome. Iterating on a functional design is more effective than iterating on individual components in isolation. We are still currently in the waterfall phase of development, but plan to break out of this at the end of the semester with a somewhat functional prototype of the system.

## 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.

# 3.3 PROJECT PROPOSED MILESTONES, METRICS, AND EVALUATION CRITERIA

## Hardware Milestones, Metrics, and Evaluation Criteria

- Choose a computer for running a CV model
    - The computer (whether it is the actual server or a microcontroller) should have the hardware necessary to run a CV model in real time.
- Choose the best CV model to run
    - The CV model should be able to run on the chosen hardware and be relatively easy to implement (good developer community and documentation).
- Computer successfully running a CV pipeline
    - The CV pipeline should be running in real time.
    - The accuracy of object detection should be upwards of 95%.
    - The pipeline should be able to take camera input.
- Computer successfully sending data to server
    - Should be able to send simple requests to server
    - Should be able to send formatted data (car ID and position) to the server
    - Should be able to send the formatted data in real time off of the CV pipeline
- Computer successfully running CV on multiple camera inputs
    - The system should be able to run multiple camera inputs through the CV pipeline and still maintain the 95% accuracy of object detection.
    - The system should still be able to send the formatted data relevant to the cars in the lot with multi-camera input.

## Software Milestones, Metrics, and Evaluation Criteria

Front-end:

- Completing Basic Screens
    - Having a welcome, register, login, and landing page would be a great starting place for this project
- 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 should be completed for this milestone to be reached.
- Integration of More Complex Screens with the Back-end API
    - The aforementioned screens should have a connection to the back-end API to complete this milestone.
- All Screens Are Completed and Integrated (Completed Prototype)

        - The app should have complete and intended functionality. All criteria listed for each screen should be met.

Backend:

- 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 our code coverage report has a high rating
- Advanced Database Information
    - Keeping track of license plates and payment methods

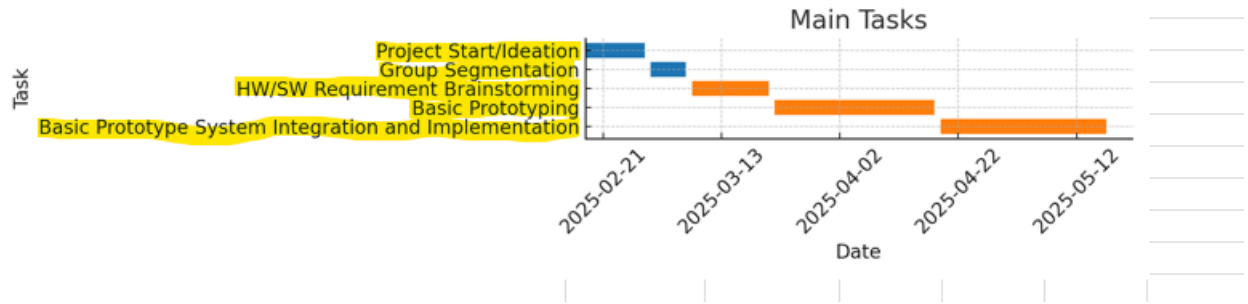
## 3.4 PROJECT TIMELINE/SCHEDULE

Because our project requires both hardware and software efforts, we decided to divide our planning into two branches: hardware and software. During the first few weeks of the class, we took the time to develop a clear understanding of the project's high-level requirements and determine who was interested in each type of work. We then created the following Gantt charts to help us schedule deadlines for deliverables. A high-level schedule was also developed to ensure that we had clear goals for integrating the different parts of the system. The software and hardware schedules were designed to align with the scope defined by the high-level schedule.

*We still need to work out the exact schedule for the system integration, but dependencies are preventing us from making those plans.

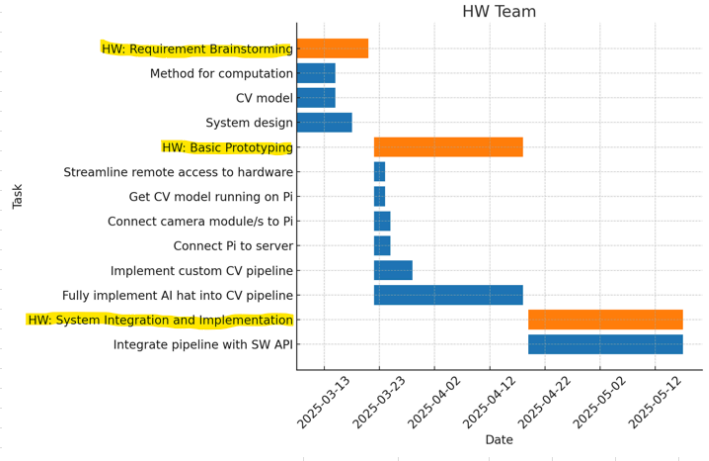
## High Level Schedule:

| Task | Start | End | Duration | Category | Notes |
|---|---|---|---|---|---|
| **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 | |



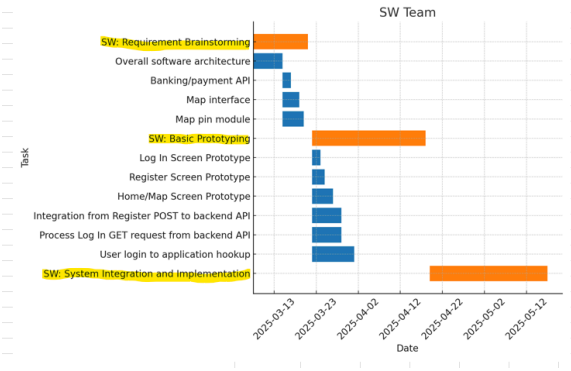

## Hardware Schedule:

| Task | Start | End | Duration | Category | Specifics |
|---|---|---|---|---|---|
| **HW: Requirement Brainstorming** | **2025-03-08** | **2025-03-21** | **13** | HW Team | |
| Method for computation | 2025-03-08 | 2025-03-15 | 7 | | Decide "where" the computation for the CV model will be ran. |
| CV model | 2025-03-08 | 2025-03-15 | 7 | | Decide which CV model will be the best fit for our use case. |
| System design | 2025-03-08 | 2025-03-18 | 10 | | Come up with a system using our proposed hardware that can detect and keep track of cars in the lot. |
| **HW: Basic Prototyping** | **2025-03-22** | **2025-04-18** | **28** | HW Team | |
| Streamline remote access to hardware | 2025-03-22 | 2025-03-24 | 2 | | Implement a way for anybody in the group to remote into the Pi. |
| Get CV model running on Pi | 2025-03-22 | 2025-03-24 | 2 | | Get a basic CV model running on the Pi (using the AI hat accelerator to help computation). |
| Connect camera module/s to Pi | 2025-03-22 | 2025-03-25 | 3 | | Connect a camera/cameras to the Pi and use them in tandem with the basic CV model. |
| Connect Pi to server | 2025-03-22 | 2025-03-25 | 3 | | Connect the Pi to the senior design server and send a simple GET request. |
| Implement custom CV pipeline | 2025-03-22 | 2025-03-29 | 7 | | Implement a CV pipeline that detects cars and reads license plates using the Pi. |
| Fully implement AI hat into CV pipeline | 2025-03-22 | 2025-04-18 | | | Integrate the AI hat accelerator with the custom CV pipeline to increase performance. |
| **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 | | | Format the output of the CV pipeline to hook into the API for the application. |



**Software Schedule:**

| Task | Start | End | Duration | Category | Specifics |
|---|---|---|---|---|---|
| **SW: Requirement Brainstorming** | **2025-03-08** | **2025-03-21** | **13** | SW Team | *For software brainstorming we plan on using the code/app architecture from the previous group and editing it as we see fit. |
| Overall software architecture | 2025-03-08 | 2025-03-15 | 7 | | Decide the functionality of the application and the overall user experience |
| Banking/payment API | 2025-03-15 | 2025-03-17 | 2 | | Used similar fields but was started from scratch on own "Payment Screen" |
| Map interface | 2025-03-15 | 2025-03-19 | 4 | | Used the same React components to implement our own Map interface on home screen |
| Map pin module | 2025-03-15 | 2025-03-20 | 5 | | Recycled pin button logo, which upon press was displayed popup with brief parking lot info and host its own "Reserve button" |
| **SW: Basic Prototyping** | 2025-03-22 | 2025-04-18 | **28** | SW Team | |
| Log In Screen Prototype | 2025-03-22 | 2025-03-24 | 2 | | Log In screen prototype completed with fields for email and password, button that upon press cross references with backend GET request |
| Register Screen Prototype | 2025-03-22 | 2025-03-25 | 3 | | Register Screen prototype completed with fields for email, password, confirm password, and Register Button that upon press sends POST request to backend API |
| Home/Map Screen Prototype | 2025-03-22 | 2025-03-27 | 5 | | Display interactive map with a basic placeholder lot upon successful login |
| Integration from Register POST to backend API | 2025-03-22 | 2025-03-29 | 7 | | Takes user input and uses React components to interact with Spring Boot executable on the live server |
| Process Log In GET request from backend API | 2025-03-22 | 2025-03-29 | 7 | | Takes user input and uses React components to pull data from the database and compare it for successful log in |
| User login to application hookup | 2025-03-22 | 2025-04-01 | 10 | | When Log In cross reference was successful a popup for "Successful Log In!" appears and the user's screen was switched to the Home page |
| **SW: System Integration and Implementation** | **2025-04-19** | **2025-05-17** | **29** | SW Team | |
| HW Backend API | 2025-04-19 | | | | Implement a basic API for the backend to send data pertaining to a specific car and its location in the lot |




# 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:* We currently plan on using a Raspberry Pi with an AI hat accelerator to do the computer vision computations. With tests that we have conducted, the accuracy needed should be no problem for the unit. If we run into a condition where it is not accurate enough, we plan on piping the video to a computer with more horsepower.
- **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:* Our current setup of the Pi 5 and AI hat totals to $230. That is well within our budget. The remainder of our budget will be allocated to the cameras needed to cover the lot, a total of $70. We believe each lot should need at most four cameras.

- **Reliability:**
    - *Risk/Constraint:* The system must be able to run overnight (for prototyping phase).
    - *Risk Level:* 0.5
        - *Solution/Mitigation Method:* We plan on using different cameras for both night and day (we are including these cameras in our planned total of four per lot). The Pi will also likely run into some cooling issues. There are cheap cooling options to try if it ends up overheating.
    - *Risk/Constraint:* The physical server we are working on crashes.
    - *Risk Level:* 0.1
        - *Solution/Mitigation Method:* If this is something frequent we can migrate to a more reliable virtual vendor.
- **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 relatively easy to develop for.
    - *Risk Level:* 0.1
        - *Solution/Mitigation Method:* We are using a very common CV architecture called YOLOv8 and running it using the Raspberry Pi AI Hat SDK and pipeline. This is a very common combination in the CV world, so help via the vibrant community will keep this project alive for anybody who wants to update it.

## Software Development Risk Management/Mitigation

**Overall Software Architecture Risks:**

*Scalability Challenges* – The architecture might not support growing user loads effectively. Probability = 0.4

*Security Weaknesses* – Weak authentication/authorization mechanisms could expose sensitive data. Severe risk due to compliance requirements.

- Utilizing industry-standard tooling within our application will mitigate possible issues with authentication and authorization. All potentially sensitive data should be encrypted using SHA256 or better encryption.

**Banking/Payment API:**

*Compliance Issues* – Failure to meet regulations like PCI DSS could potentially lead to legal action. Severe risk because of the legality.

- Utilizing libraries and ensuring that we are educated on these standards before implementing them will allow us to bypass any possible issues.

*Fraud/Unauthorized Transactions* – Weak authentication in this system could lead to financial fraud. Severe risk because of the legality.

- Ensuring that we follow industry standards on authentication and properly utilize available libraries will protect our users against fraud.

**Map Interface:**

*Third-Party API Downtime* – Relying on external map services introduces external risks. Probability = 0.3

*Device Compatibility Issues* – The map might not function properly on all mobile devices or browsers. We utilize a library within React-native to avoid any compatibility issues between platforms. Probability < 0.1

**Login Screen Prototype:**

*Weak Authentication Design* – Insecure login mechanisms could expose user credentials. Severe risk due to compliance standards.

- Utilizing industry-standard libraries for form submission and HTTP requests mitigates risk involved with login authentication.

*Lack of Error Handling* – Users may become frustrated if they do not receive on screen feedback for errors encountered within the application. Probability = 0.3

**Register Screen Prototype:**

*Spam/Fake Accounts* – No email/phone verification could allow bot registrations. Probability = 1

- There is no verification system to prevent bot registrations. This could cause bots to book entire parking lots. To ensure this does not happen, implementation of email verification must be completed.

**Home/Map Screen Prototype:**

*Data Refresh Issues* – Inconsistent data updates could make the map unreliable. Probability = 0.4

*Device-Specific Bugs* – Some features might not work across all devices. React-native should mitigate platform compatibility issues. Probability < 0.1

**Integration from Register POST to Backend API:**

*API Downtime* – Backend unavailability could prevent new user registrations. Probability = 0.5

- To mitigate this issue a queue could be created to make requests to the API once it is available again. Users could be notified of registration when sending the verification email.

**Process Login GET Request from Backend API:**

*Data Leakage* – Sensitive user data could be exposed in responses from the API. Severe risk due to compliance standards.

- Data being sent to and from the API should be within the body of requests. The body of these requests is encrypted, which mitigates the risk of data leakage.

**User Login to Application Hookup:**

*Session Expiry Issues* – Poor session handling causes unexpected logouts, which increases user frustration. Probability = 0.5

- The user's login token should be stored on the front end securely to avoid unintended logouts.

*Multiple Login Handling* – Lack of support for multi-device login could frustrate users. This application is designed for mobile devices due to the nature of its use. The likelihood of a user having multiple phones that they wish to login with is unlikely. However, having a desktop version available for parking administration users could be beneficial and might run into this issue. Probability = 0.3

**System Integration and Implementation:**

*Lack of Rollback Plan* – No contingency plan exists for failed deployments. Probability = 0.5

- This application is still in the prototype phase. Future iterations should include sufficient versioning to avoid this issue.

*API Versioning Conflicts* – Changes in the back-end API versions could break existing integrations. Probability = 0.3

**Backend Application Crashing**

*Unhandled Exception in Backend* - Edge cases could potentially crash our SpringBoot backend application.

- We can set a Docker policy on our container to automatically rerun if the SpringBoot application crashes.

Probability = 0.2

# 3.6 PERSONNEL EFFORT REQUIREMENTS

The tables below provide a detailed breakdown of the estimated effort required for each task in total person-hours:

Software Development Effort Estimate

| 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 | |

## Hardware 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 | |

The projected person-hour estimates ensure an efficient allocation of resources, with time dedicated to critical phases such as requirement gathering, prototyping, and system integration. These estimates help maintain project timelines and ensure smooth execution.

## 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.