# 4.2 Design Exploration

## 4.2.1 Design Decisions

The current design is the result of many design decisions. Specifically, some decisions include the framework for the mobile application, the connections between the hardware, and the placement of cameras. Each one of these decisions will significantly impact the users' experience with the product, and the development over the next year.

## 4.2.2 Ideation

Where the camera is placed will impact the hardware that can be used (cables, WiFi, batteries), accessibility for maintenance, how the code works, and many more aspects of the project. The design decisions were developed through discussions and brainwriting. This resulted in multiple potential approaches, including:

- **One camera at each entrance/exit**: By monitoring the accesses to the parking lot, the state of the whole lot can be assumed. However, the status of each individual 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

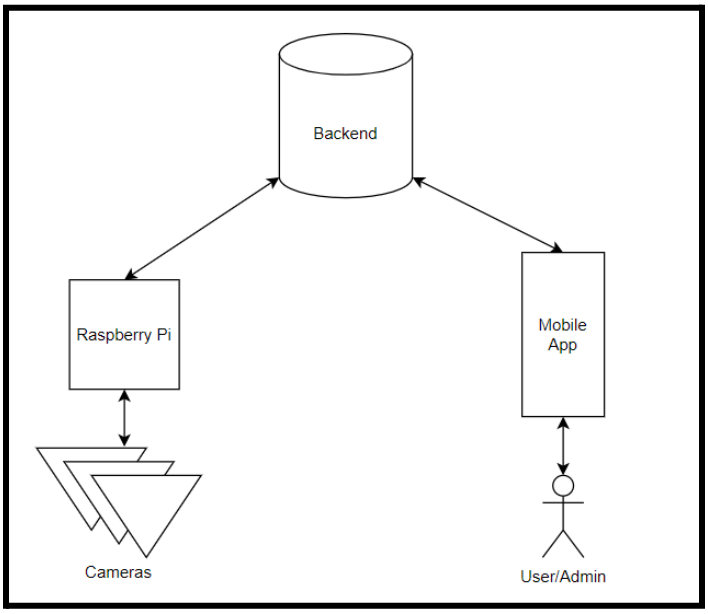
To decide which approach to use, each potential decision's pros and cons were evaluated by discussion and mind-mapping. These are the benefits and disadvantages of each option:

- **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.
- **One camera monitoring each parking spot**: Having one camera per spot would be the most robust solution and would require the least complex software. However, it would be the most expensive, require the most installation, and require ceilings or poles in line with each spot.
- **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.
- **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.
- **One camera for each row (chosen)**: This was the decision that was chosen. This was chosen by compromising software complexity and hardware installation. The result is not as invasive as having a lot of cameras, is modular, and will maintain reliability.

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

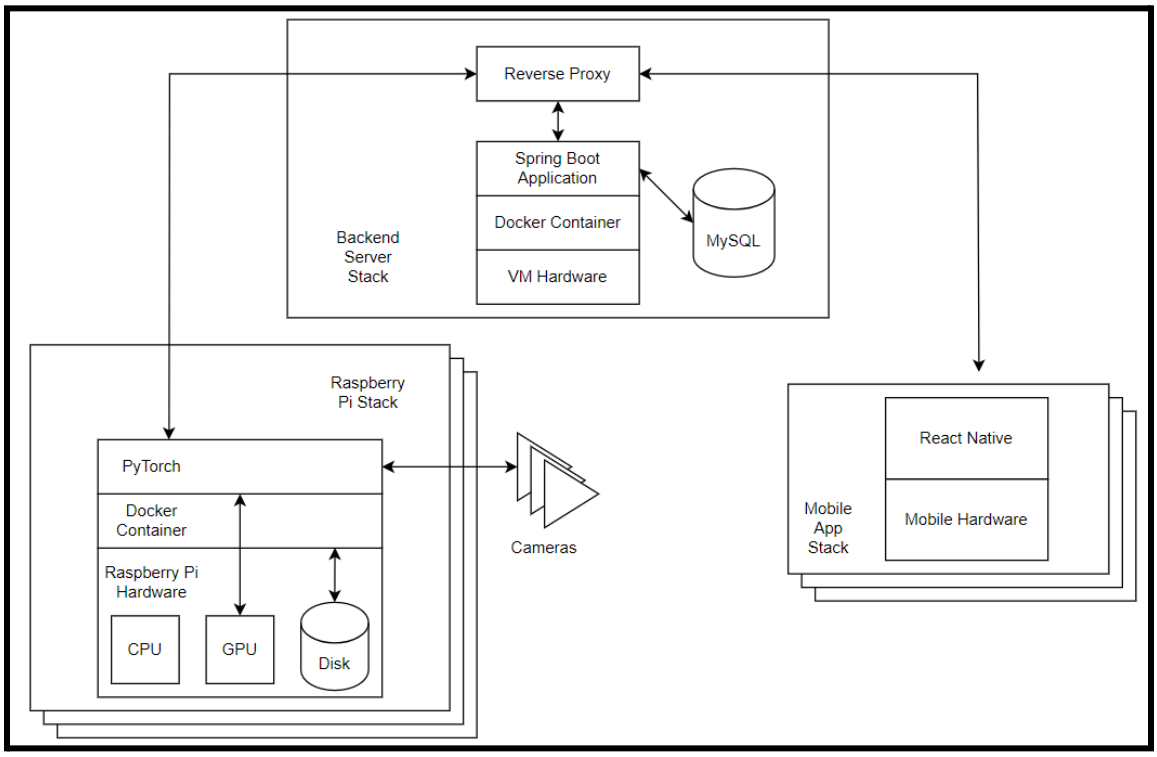
# 4.3 Proposed Design

## 4.3.1 Overview




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

## 4.3.2 Detailed Design and Visual(s)




This diagram shows the three main components' software and hardware organization.

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

The backend consists of a reverse proxy, Spring Boot application, 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 did not allow 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 can interact with the application through the mobile app. The frontend mobile app is being developed using React Native. This app will interact with the backend fetching and storing parking lot data while also holding user data.

## 4.3.3 Functionality

***Describe how your design is intended to operate in its user and/or real-world context. What would a user do? How would the device/system/etc. respond? This description can be supplemented by a visual, such as a timeline, storyboard, or sketch.***

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

### 4.3.4 Areas of Concern and Development

The design experienced trade-offs and iteration to satisfy the requirements given to the team. However, user needs dictated more requirements than those provided in the initial request. This means more features and functionality are needed than the initial designs included.

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

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

## 4.4 Technology Considerations

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

- **React Native**: React Native was chosen to develop the application because of its ability to create apps that work on IOS and Android without too much extra code. With React Native, generic elements are converted into native OS elements; this avoids duplicate code.
- **Spring Boot**: Spring Boot is a production-ready software that allows for rapid development and scales well. Some negatives about using Spring Boot are the high memory cost and the hidden complexities with technologies such as auto-configuration.
- **MySQL**: MySQL was selected as the relational database management system because of its reliability and open-source licensing. MySQL gives efficient storage with querying capabilities for structured data making it easy to manage user and parking lot information. While a database management system like PostgreSQL has more advanced features, the project does not require those capabilities, and MySQL's more straightforward configuration was aligned to this project.
- **Docker**: Docker was chosen for its scalability and real-world uses. By utilizing Docker, code and updates can be deployed easily to hardware. Docker is an industry standard, so issues have been documented and solved. This adds complexity to the development, but it makes the product scalable.
- **Raspberry Pi**: The Raspberry Pi has integration with its AI hat, and it is one of few products that can host the chosen model. Raspberry Pi has lots of features that allow for easy integration with the other components in the system. Although it is more expensive at production-scale than a custom computer, it is a good fit for the prototyping stage.
- **YOLOv8**: YOLOv8 is capable of high accuracy while still being considered a fast model when hardware acceleration is used. It requires training, but that overhead is just in the development stage. Once it is deployed, it will have low latency and confident predictions.

## 4.5 Design Analysis

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

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

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