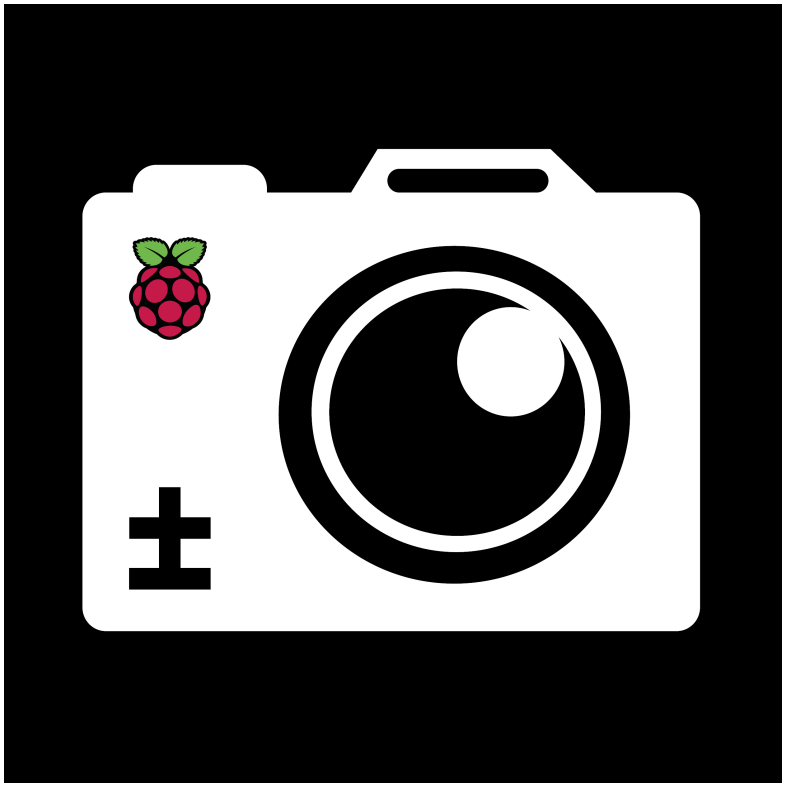

# Final Report Document
# People Counting Infrastructure
# Version 1.1

**Date:** April 28th, 2021

**Team Name:** PiWatcher

**Project Sponsor:** Duane Booher, NAU ITS

**Team's Faculty Mentor:** Volodymyr Saruta

**Team Members:** Seth Burchfield, Champ Foronda, Joshua Holguin, Brigham Ray, Brandon Thomas

# Table of Contents

# 1.0 Introduction

These trying times have deemed it necessary for large organizations to provide technologies that will help keep people safe. Currently, there isn't any way for large organizations to keep track of population flow without enormous investment. This is where our PiWatcher system steps in to fill the gap.

Current solutions for controlling population density are expensive. They utilize several complex technologies to help count people, some of which are unnecessary and further add to the cost. This is unfortunate when an organization has multiple buildings because this cost limits the number of devices that can be deployed, and therefore the number of locations that NAU can keep safe.

Our proposed solution aims to solve the challenges that the current technology does not address. These challenges are price, flexibility, and scalability. Using a Raspberry Pi with an attached PiCamera, each device would cost around $100 per deployment, allowing the university to scale our people-counting infrastructure to nearly all buildings and classrooms.

For this solution to be effective, a web application will be provided for administrators to customize the flow of information from the Raspberry Pis to suit their organization's business needs. Of course, this particular interface will need to be responsive and utilize the latest web technologies to keep the site secure.

All the data provided in the web application will be processed asynchronously. Different endpoints on the flask backend will streamline the amount of data being sent to the user so that their web application remains responsive. MongoDB will store this sent data in JSON format which is easily translated to human-readable information once it is sent to the web application. These technologies are capable of handling a high volume of requests which may be necessary once a large number of IoT devices are deployed on campus.

Most importantly there remains the Raspberry Pi portion of the solution. Each Raspberry Pi will be configured to collect data at a specific location. Data will be recorded using a camera and processed using machine learning technologies. Afterward, it will be sent over the NAU network in JSON format to be stored in the MongoDB database.

The components of this solution will reside on the NAU Mountain campus. Ideally, the Raspberry Pis will maintain a low profile as to not alarm the students or faculty. The

application that administrators utilize to perform data analytics that this solution provides will scale appropriately to both desktop and mobile devices.

# 2.0 Process Overview

This section outlines the roles and responsibilities each member of the team had and outlines the procedures and tools used to carry out the bulk of development.

## *2.1 Roles and Responsibilities*

| | |
|---|---|
| Team Leader | The team member that coordinates task assignments and ensures work is progressing, runs meetings, and makes initial efforts to resolve conflicts. |
| Customer Communicator | The team member that coordinates and conducts customer communications. |
| Recorder | This team member maintains detailed meeting minutes. |
| Release Manager | This team member coordinates project versioning and branching, reviews, and cleans up commit logs for accuracy, readability, and understandability, and ensures that any build tools can quickly generate a working release. |
| Developer | This team member is responsible for designing the architecture and developing quality code as well as maintaining and testing to ensure full code coverage. |
| Editor | This team member is responsible for ensuring that every document is following document standards. This includes, but is not limited to, checking formatting, |

| | |
|---|---|
| | grammar and spelling errors, consistent sentence flow structure, etc. |

## *2.2 Members and Responsibilities*

| | |
|---|---|
| **Champ Foronda** | Team Lead, Customer Communicator, Developer |
| **Seth Burchfield** | Developer, Editor, Release Manager |
| **Joshua Holguin** | Developer, Editor, Release Manager |
| **Brigham Ray** | Customer Communicator, Developer, Recorder |
| **Brandon Thomas** | Developer, Editor, Release Manager |

## *2.3 Expectations Introduction*

The following section details the expectations each team member needed to follow. This includes where meetings were held, attendance, and how the meetings were structured. Consequences for certain behavior are also outlined.

## *2.4 Meeting Times*

Our team conducted meetings on Tuesday and Thursday from 3:30pm to 4:30pm. Aside from client meetings held on Zoom, all meetings were held on Discord. Meetings were cancelled whenever the team agreed to cancel as a whole. Impromptu team meetings were called by anyone on the team who thought it was necessary. Members were not penalized for attendance issues related to impromptu meetings.

Mentor meetings were held on Tuesdays from 4:45 to 6:00 PM. Like team meetings, mentor meetings were held in Discord.

## *2.5 Agenda Structure*

The following is the agenda structure we used for team meetings:

- A 5-minute breakdown of things that each team member is working on, will be working on, and any roadblocks that they currently have.
- A 15-20 minute grooming session, where we establish tasks for the week, to do lists for each task, acceptance criteria for each task, and a deadline for each task. At the end of the grooming session, each team member will be assigned tasks such that the contribution efforts will be roughly 25% for each team member.
- The rest of the meeting is flexible. This is where our team will start/complete tasks, address any sidebars that were brought up, and propose important topics of discussion.

## *2.6 Decision-Making Process*

Team members were allowed to express their opinions regarding development decisions. Any disagreements were resolved democratically with a majority vote.

## *2.7 Meeting Attendance*

Behaviors listed below were penalized as follows:

Missing Meetings:
- One missed meeting: Conversation with the team
- Continual missed meetings: -5 points to team evaluation
- Fourth missed meeting: Team discussion with mentor/capstone organizer.

Late To Meetings:
- More than three 10-20 minutes late to meeting is equivalent to one meeting absence.
- Missing at least half of the meeting results in one meeting absence.

Excused Meeting Responsibilities:
- Send a five-minute breakdown done at the beginning of the meeting in the Discord server.

Notices For Missing a Meeting
- At least a six hour notice before the scheduled meeting

## *2.8 Conduct*

This lists the rules that governed our team meetings, such as how meetings were organized and how our team avoided non-constructive dialogue.

Changes without team consent:

- First offense: Initial polite-heads up discussion from team lead.
- Second offense: An internal team discussion with the team member
- Third offense: A formal team discussion with the team mentor and capstone organizer

Interpersonal Disputes:

- Interpersonal disputes will be handled immediately and discussed with the team as soon as possible.
- In extreme cases, the team will have a formal team discussion with the team mentor and capstone organizer.

Nonparticipating Member:

- First offense: Initial polite-heads up and discussion from team lead.
- Second offense: An internal team discussion with the team member
- Third offense: A formal team discussion with the team mentor and capstone organizer

## *2.9 Tools Introduction*

This section outlines the tools that our team used and expectations for how they were used. This includes version control software, issue tracking software, word processing and presentation, composition and reviews.

## *2.10 Version Control*

Git and GitHub were our main software for version control. Each repository had a develop branch where each task spawned a new branch off the develop branch. Once tasks were completed and pull requests were approved, the branch was merged into the develop branch and the task branch was deleted. Commit messages had to be professional, clear, and concise.

### *2.11 Issue Tracking*

All tasks were tracked through a kanban board on our GitHub organization.

### *2.12 Word Processing and Presentation*

All documentation and presentation aspects were done using Google Docs and Google Slides. Any video editing was done by the Recorder. Any needed audio and video recordings were sent to the Recorder at the latest two days before the due date at 5:00pm.

### *2.13 Composition and Review*

Final rough draft for any deliverable needed to be completed two days before the team assigned deadline. Afterwards, editors would finalize the deliverable for submission.

### *2.14 Team Self Review*

The team will conduct self reviews for the first meeting of every month. The efficiency and success of the group was assessed and decisions were made based on what to start doing, keep doing, and stop doing as a whole. Weekly meetings reviewed the completion and incompletion of tasks from the prior week to measure team progress and diagnose any problems that may have arisen.

## 3.0 Requirements

All requirements for this project were gathered by Duane Booher for addition to the official requirements document. At first, specific questions for Duane about the project were gathered by team members on a spreadsheet. This allowed the team to sift through Duane's answers and generate official requirements.

The requirements gathered were split into different sections: Domain, Functional, Performance, and Environmental requirements.

Domain requirements gathered by Duane are as follows: User Interface, Object Recognition from Camera, Count people from video frame, and User Authentication. These are the high level requirements from which lower level requirements were generated, each representing major components of the production system.

The high level functional requirements for the system include: Elements of the User Interface, Elements of Object Recognition, Elements of People Recording, and Elements of User Authentication. From these main functional requirements, lower level functional requirements were generated.

Performance requirements specifying performance expectations in detail were also gathered by Duane. These include: Performance of User Interface, Performance of Person Recognition, and Performance of People Count Storage (modified). An example of one such requirement found within the Performance of Person Recognition section was the requirement for a count to be generated in less than 5 seconds from the moment a Raspberry Pi takes the picture. Environmental requirements detailed where and how specific components of the project were to be displayed, including: Usability on Desktop, Usability on Mobile, and Host Platform. These requirements include details about which operating systems should be compatible with the system and that the system's usability should maintain similar performance irrespective of the host environment.

# 4.0 Implementation Overview

Our solution includes utilizing an IoT device and camera to accurately count people within buildings and classrooms, utilizing OpenCV, an open source computer vision library, and TensorFlow Lite, an open source deep learning framework for IoT devices. In combination these libraries combined with a YOLOv4 model and hardware will be placed in buildings and classrooms strategically in order to get the most accurate count possible.

The information gathered from these devices will be sent to a Flask backend that is tied with a MongoDB database. Thus, all information is through our backend and not the IoT device itself. This backend will then clean up the data as necessary to prepare it for ingestion through the front end.

Our front end will be utilizing React to create an elegant login and landing page with search functionality. The login will be utilizing our own authentication system so only authorized persons will be able to access it. Then there will be a search functionality to get specific buildings and classrooms within said building. This information will then be

displayed via a graphical representation along with a fraction and percentage of the current usage. The graph will also be configurable to the user so they can give specific time intervals and the data will then be displayed as requested.

This is all being done to help our client build and utilize a system for keeping record of how spaces within buildings are used and to find a way to best utilize these spaces in the future based on gathered data. For example, we can allocate more people to certain spaces for club meetings or create space for offices for certain departments to be put depending on the space at hand. Through the use of IoT devices this offers an inexpensive but resourceful way to accomplish this task.

# 5.0 Architectural Overview

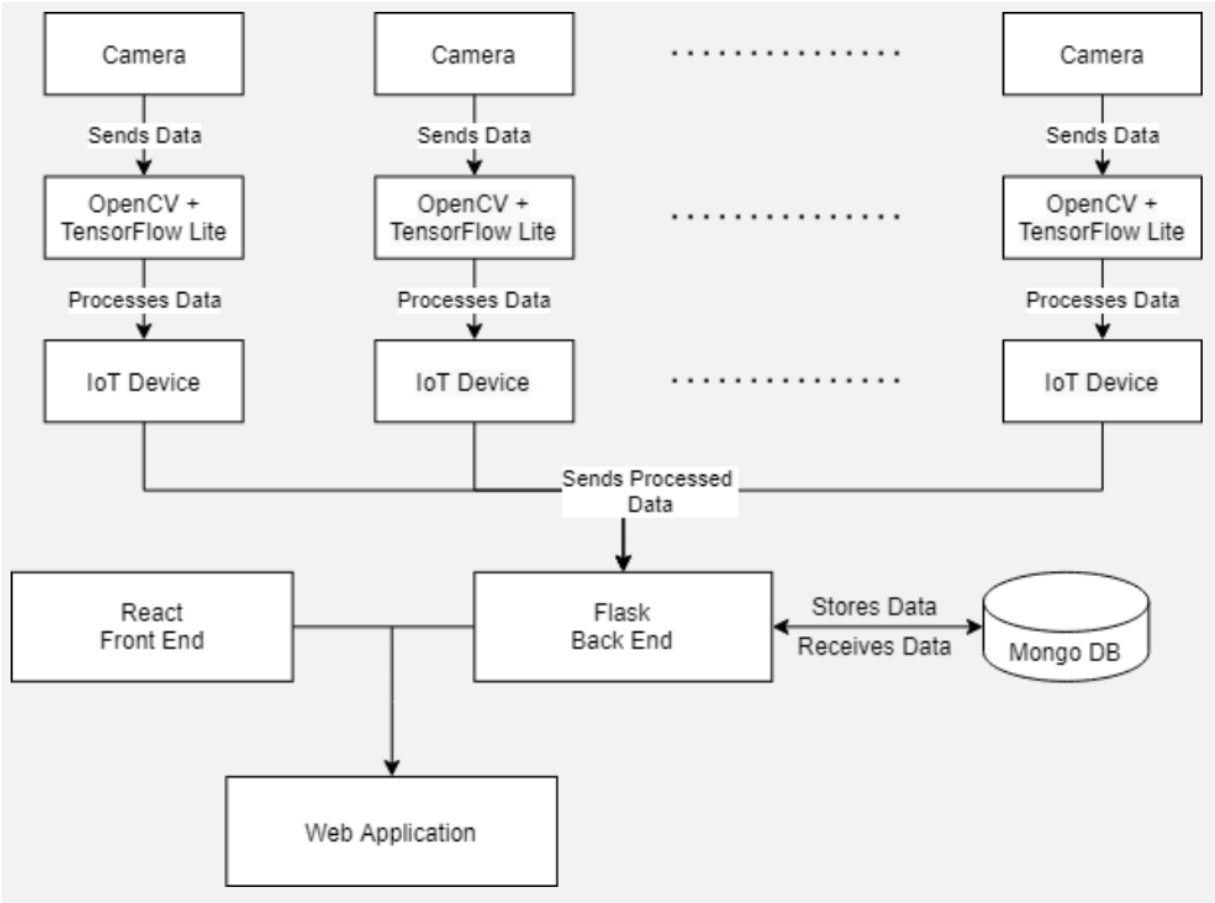


Figure 1: Architectural Overview Diagram

Our design includes eight modules. The first consists of many IoT devices connected through NAU's network. These devices do not talk to each other, however, they communicate to our backed end hosted locally at NAU. Each device will be placed in areas throughout buildings and classrooms and will have a camera mounted on them to gather data. This camera will both count people in its field of view and report that count to our backend. This counting will be done utilizing OpenCV and TensorFlow Lite. This information will be relayed to the backend with information specifically for that endpoint. This information will then be processed and delivered to our front end correctly formatted.

The front end will then display the data to the users via a time series graph to see the recent activity in the building and/or room. This graphical representation can then be configured by the user to determine the time interval they wish to see. This information will be stored in a database making it easily accessible to our backend to push and pull data from. The front end application will also consist of a login screen to give users different levels of access. This access will consist of admins and regular users within the role based system. In order for users to find what they need efficiently, there will be a search feature breaking up the results into buildings and rooms where the IoT devices are currently set up and operational.

The whole structure will be discussed in depth in section 6.0 with more explanation on the design of each module.

# 6.0 Module and Interface Descriptions

## *6.1 Web Application Front End*

To display the desired information to the user, we used React to build an appealing and easy to use interface. Our frontend design consists of an object oriented programming model with the use of components and contexts. Utilizing contexts gave us the ability to quickly pull the desired data and provide an overarching connection to all of our visualization components in a non convoluted manner. Components provide the ability to use them within our chosen context easily for simple data consumption and display. These components are within four main modules, Dashboard, User Authentication, Administrator Settings, and User Settings.

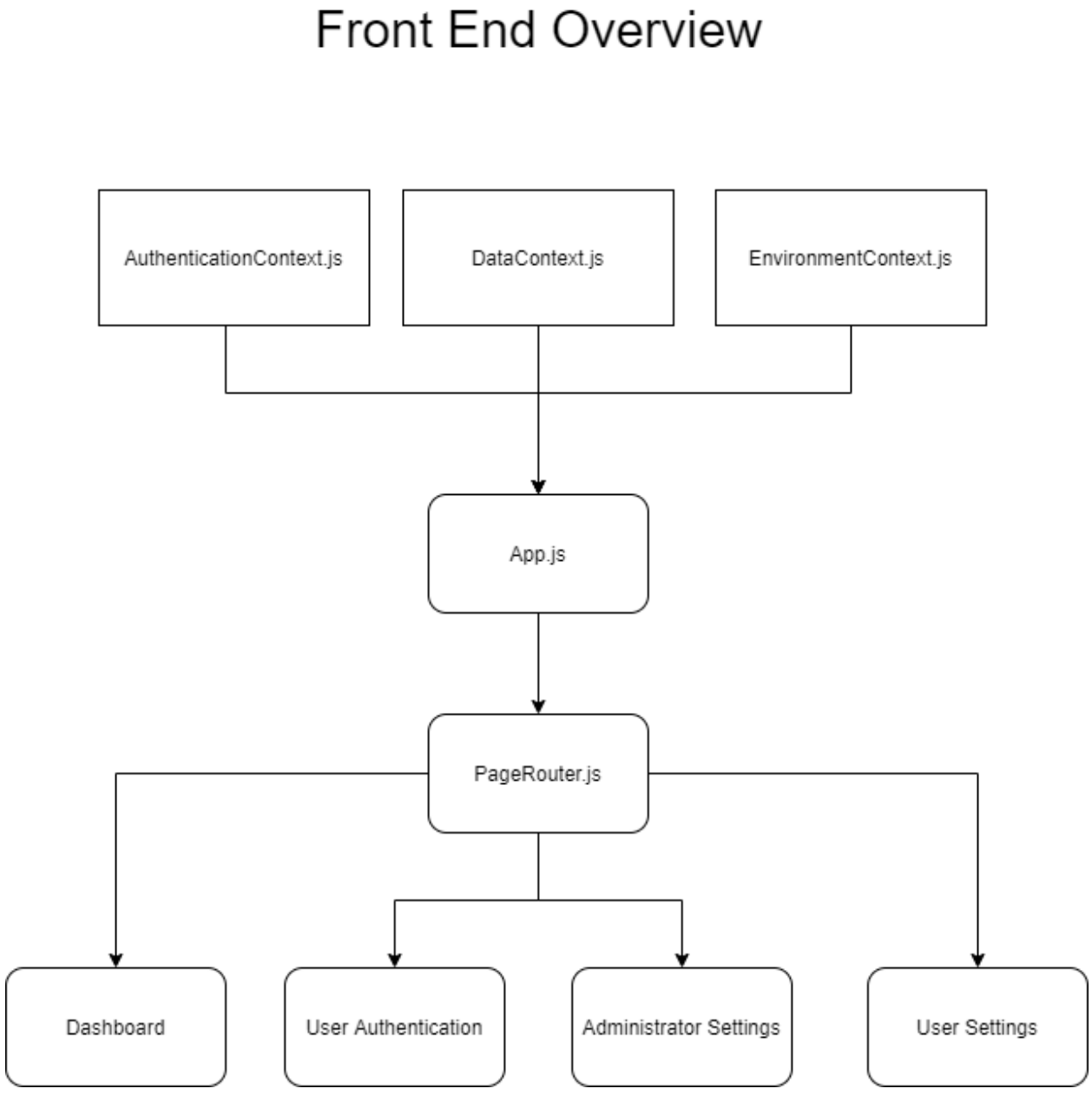


Figure 2: Front End Design Diagram

### *6.1.1 User Dashboard*

The user dashboard encompasses a navigation bar, consisting of a search bar and the different settings available depending on the user's allocated permissions. The search bar allows the user to search for and select the desired building. When a building is selected, a list of rooms from within the selected building will then be displayed in a searchable, scrollable format in our room list component. Each room in the list has

visual cues alongside for a quick look at room usage. The user then has the option to select a room. On selection, the data from the chosen room will then be pulled and parsed for display within the data visualization component, a time series. The time series can accommodate varying intervals for displaying room counts and trends in past or current tense. The graph will fit and change size according to the user's chosen data set as well as being draggable and resizable around the dashboard.

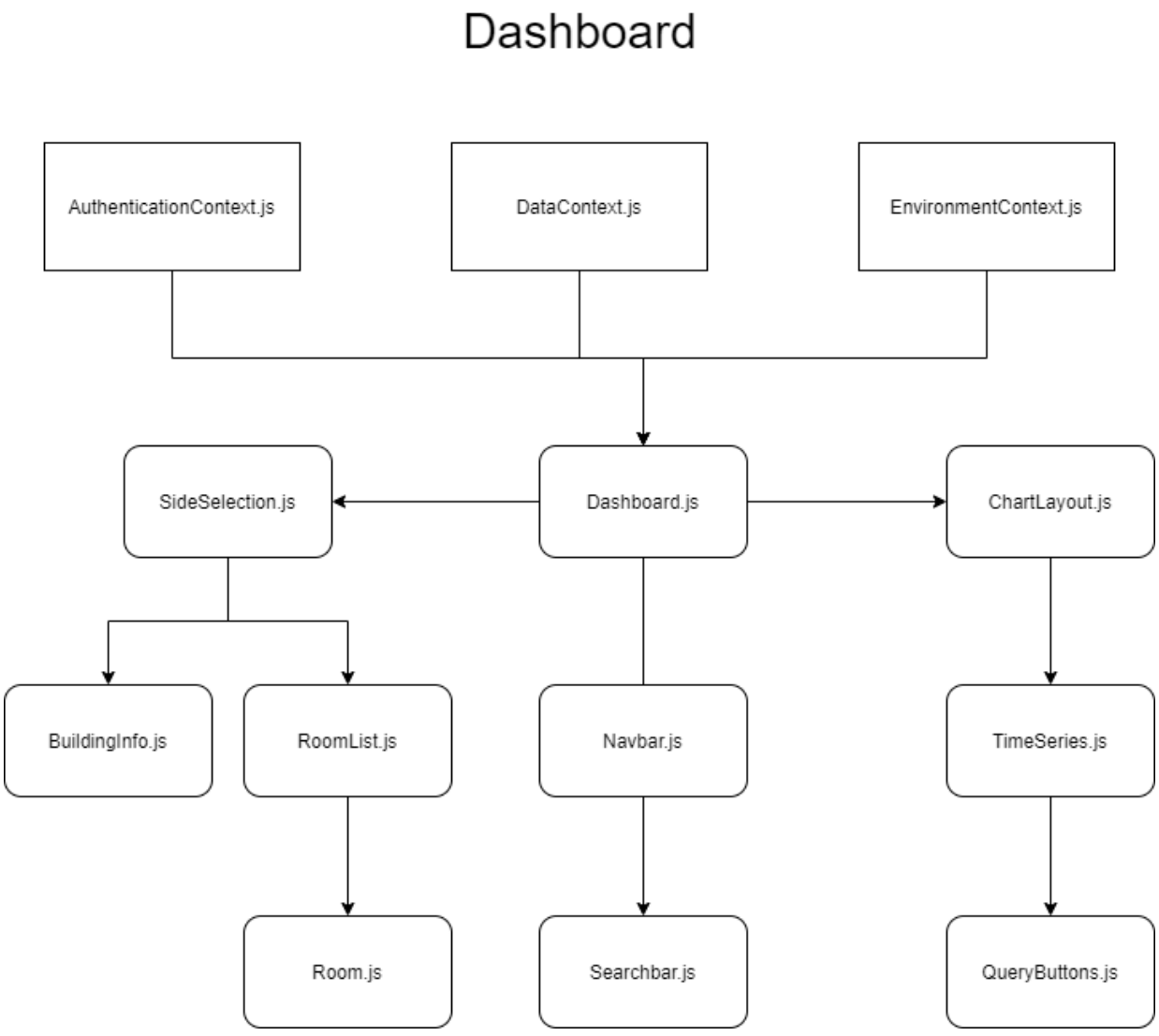


Figure 3: Dashboard Design Diagram

### *6.1.2 Administrator Settings*

A user with administrator privileges can adjust other users and their accounts. On load, all users and roles are pulled from the database and displayed in their respective list component. New roles can be created with varying permission sets and assigned to these users. Administrators can also delete user accounts and roles from the system.

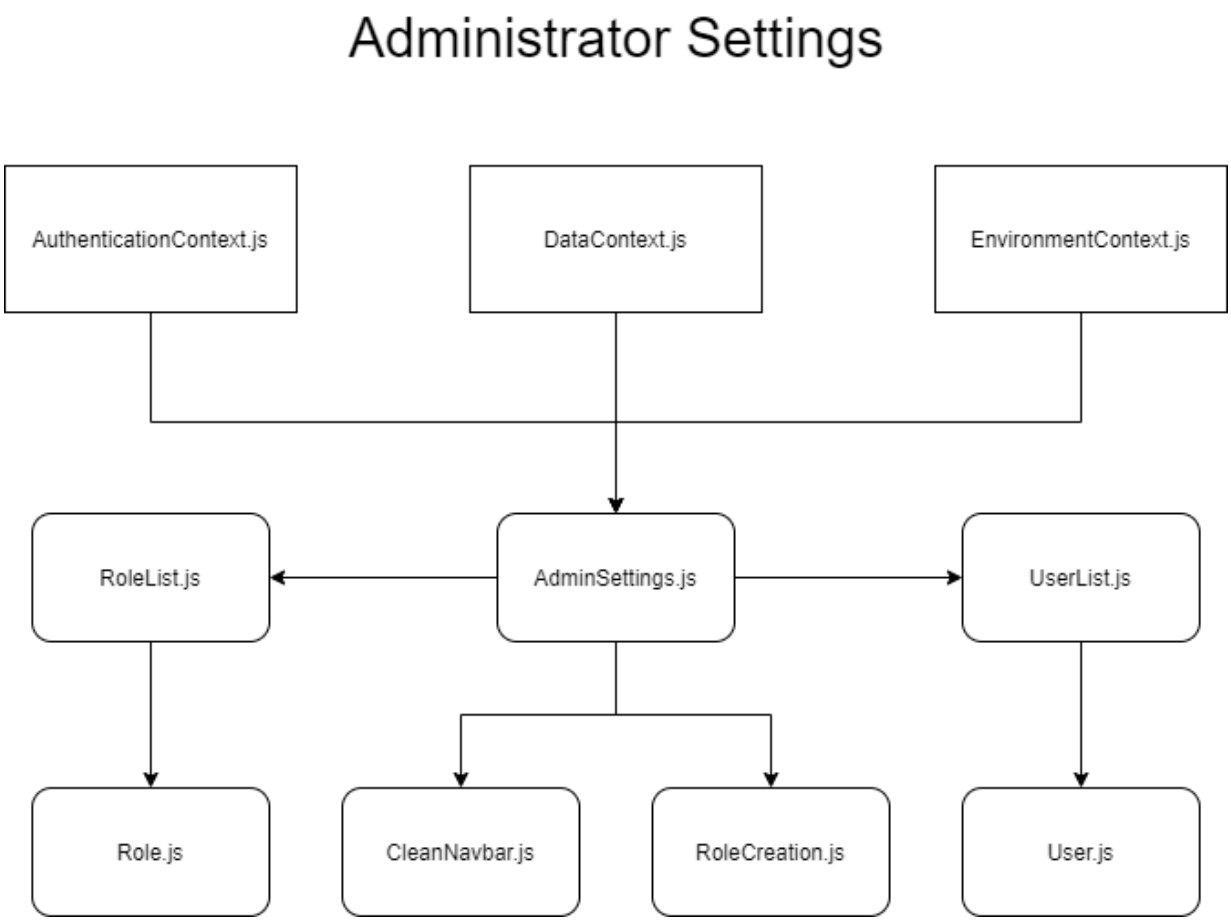


Figure 4: Administrator Settings Design Diagram

### *6.1.3 User Authentication*

Users can sign up and sign in to the web application through its authentication component. The component handles all user input for sending to the backend for processing with notification alerts if the user's information is not able to be authenticated.

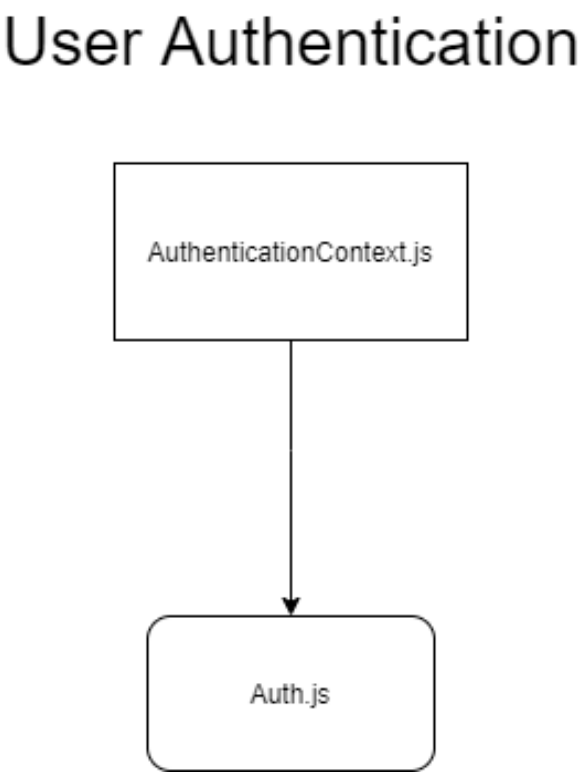


Figure 5: User Authentication Design Diagram

### *6.1.4 User Settings*

Users have the option to change their password within the user settings of the web application. All input is stored within the component and sent to the backend for processing. An alert is displayed to the user notifying them of a successful or failed password change.

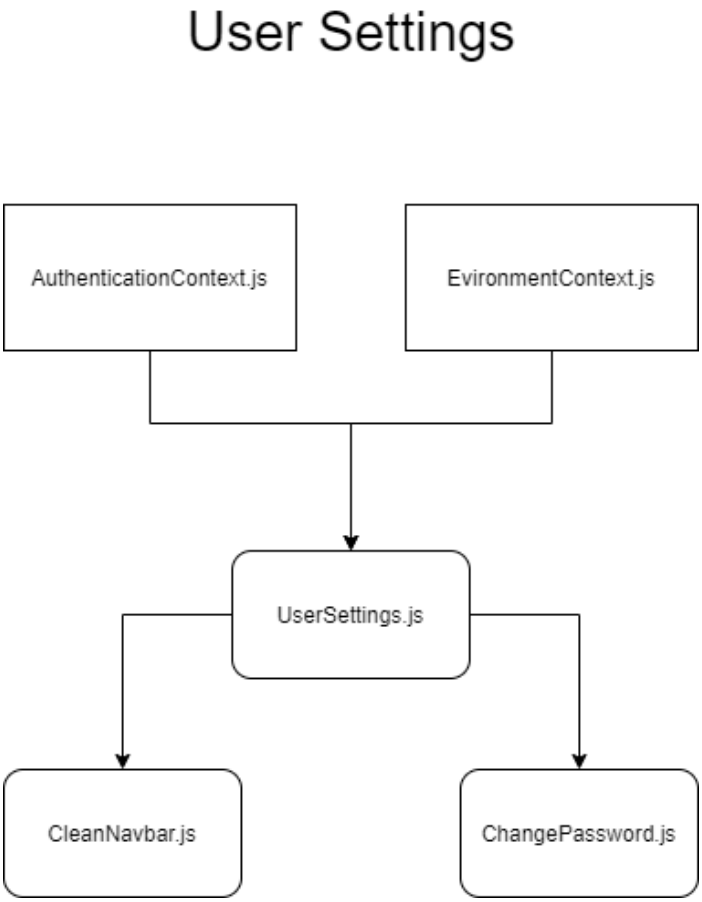


Figure 6: User Settings Design Diagram

## *6.2 Web Application Back End*

This following section describes the back end portion of our People Counting Infrastructure. The backend contains a REST API which is called People Counting Infrastructure REST API (PCI REST API). PCI REST API provides resources and endpoints to call the different services that connect the entire infrastructure together. The services that our backend provides is a Login Authentication Service (LAS) and a MongoDB Manager Service (MMS).

### *6.2.1 Login Authentication Service (LAS)*

The Login Authentication Service (LAS) is an internal service that provides a layer of security that ensures only authorized users are able to access the web application as well as the resources/endpoints that the PCI REST API provides. This service will provide end users the ability to login or create an account to gain access to the resources that our application provides. A flow diagram of the service is shown in figure

7.

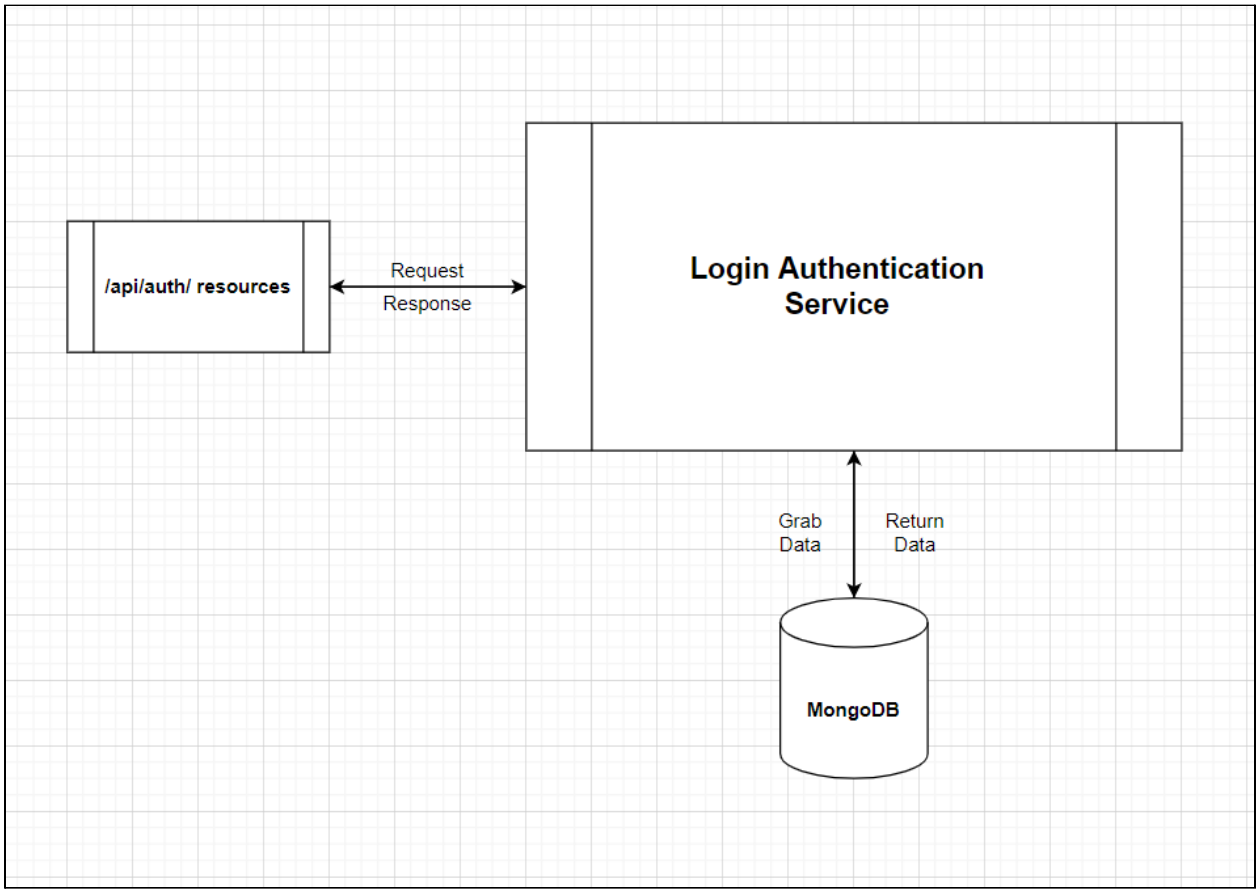


Figure 7: Login Authentication Service (LAS) Design Diagram

The front end of our web application creates a HTTP request with the following data user_name, user_email, and user_password. This data is fed into LAS, which handles creating a user and storing it into our database. If a user needs to be created, then it will interact with the LAS to create an account based on the values that were passed into the request. If the end user already has an account with the system, it will validate that the user exists within our application through the LAS and give the user the appropriate access to the resources needed.

### *6.2.2 MongoDB Manager Service*

The MongoDB Manager Service (MMS) is an internal service that provides functionality for both the front end and the IoT device to gain access to our database. This service allows the IoT device to store it's entries directly into the database. This service also allows the front end to pull information from the database to create metrics based on the count data. In figure 8, is a flow diagram of the service:

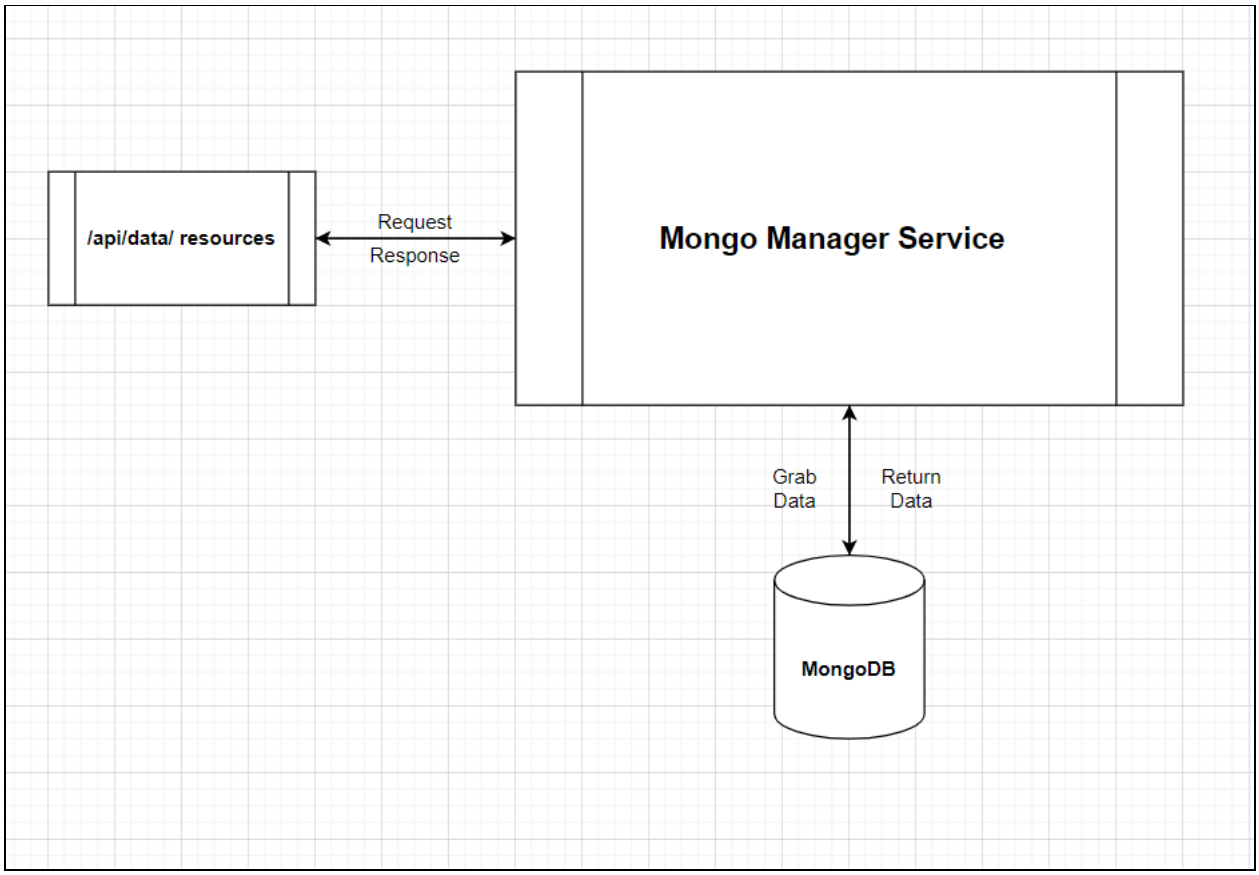


Figure 8: MongoDB Manager Service Design Diagram

As shown in the diagram above, the MMS gives IoT devices the ability to insert each entry by room into the database. Along with that it also provides the front end the functionality needed to collect information about the data the IoT device is gathering.

## *6.3 Internet of Things Device*

In figure 9 is a complete UML diagram that encompases all of the different modules within the IoT device. This section will explain how this whole model fits into the overall structure of our solution. In addition, there will be a sub section explaining each module within the diagram in more detail. Each module is split up into three sections as follows: Name, Parameters, Functions.

This model envelops the camera, OpenCV and TensorFlow Lite, and IoT Device portion of the diagram seen in Section 5.0 of this document. What the IoT device is doing is taking a video input, counting the number of people within the video via OpenCV and TensorFlow and outputting that count to our backend which in return sends it to our database. This is the first and most crucial step in getting the data to our users. Without this step we will have no data for the backend to process and our front end to show.

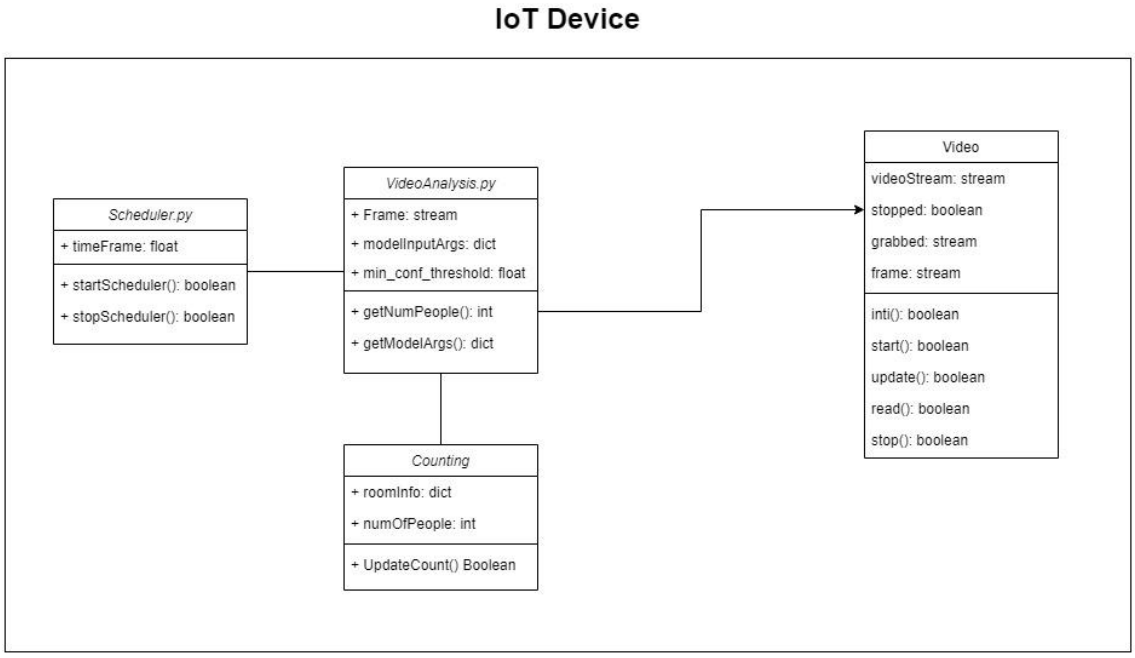


Figure 9: IoT Device Design Diagram

### *6.3.1 Scheduler Module*

This module will be used to set a schedule on when we wanted the Video Analysis module to run. It does this by taking in a time frame and using the Video Analysis module based on that time frame. This will be done using a cron job within python. The job will have defaulted parameters. The only thing changing will be how often it runs per day given by the number of seconds entered. There will be global constraints on the number of seconds entered to ensure the Video Analysis module has enough time to run. Once an average for that module has been determined we will make it so a time below that average cannot be entered. The stop schedule feature is simply there as a fail safe to end the cron job for whatever reason. We do not envision this to be utilized unless the admin sees fit.

### *6.3.2 Video Analysis Module*

This is the main module within the diagram. The purpose of this module is to output a count based on the incoming video stream, threshold, and model. The threshold is used to determine the confidence in which the model given should detect a person. This is mainly for developer access so if the model is changed or improves the threshold can be adjusted as needed. The model input arguments are used to receive a pre-trained model to be utilized in the functions. This model will have to be a specific type in order

to work properly with the hardware and any accelerators that we plan to use. The get number of people function will take in a Video class and model arguments to do its analysis needed to output the number of people it finds within a video frame. This function will utilize getModelArgs() in order to get the proper arguments needed from the model for python to use OpenCV properly. The model that was initially passed in will be utilized here to get the desired arguments for OpenCV and TensorFlow to use. Upon completion the counting module will be called to execute the final part of this process.

### *6.3.3 Video Class*

This is a class that will be utilized by the Video Analysis module. This class consists of a video stream, stopped, grabbed, and frame variables. The video stream is a variable that will store the live feed of the video to be accessed in later methods. Stopped is a boolean that ensures the feed is live and has not been interrupted. Grabbed and frame are used to get a frame from the video stream. This is done utilizing a function within OpenCV. Grabbed is a boolean to tell if the frame was successfully stored. Frame is where we are storing the image to be analysed. The methods within this class are to edit, set or update said variables. Start: begins the stream, update: updates the current frame stored in the class, and read: will return said frame.

### *6.3.4 Counting Module*

This module will execute after the Video Analysis module has returned its count of the area being read in. This module's main purpose is to send an updated count to our backend. The room info is provided as a dictionary containing strings that will tell the function which endpoint to hit on our backend to ensure we are updating the current count as well as ensure each count is different based on what IoT device this module is being run on. The function will then return a boolean value to ensure it executed properly.

# 7.0 Testing

Our project utilized unit, integration, and usability testing to ensure our modules function and interact correctly with one another. The testing section of our report outlines how we accomplished our testing goals.

## *7.1 Unit Testing*

### *7.1.1 Unit Testing for Frontend*

The frontend was unit tested by verifying each component was created and rendered correctly. By using Enzyme and Jest, each component was tested by being rendered in a virtual mock DOM with verifications to ensure that the expected visuals were rendered. If the expected outcome was realized, the test was passed.

### *7.1.2 Unit Testing for Backend*

Unit testing on the backend was done by creating a test database and utilizing a test client using Flask. The current testing suite implemented tests all resource endpoints and ensures that the expected responses are received. During the process of developing the testing suite, no major design changes were made to the backend to ensure that the unit tests worked. As a result of this, all unit tests passed and ensured that the endpoints and services, as it stands, work as intended.

### *7.1.3 Unit Testing for IoT*

The IoT device was tested by ensuring the shell scripts to set up the device have run correctly by checking all of the necessary dependencies are created and ready for the device to utilize. This includes checks of environment variables and virtual environment(s). The main test that is run upon running the testing suite is ensuring the model used for identifying and counting people is loaded properly.

## *7.2 Integration Testing*

### *7.2.1 Integration Testing for Frontend*

The frontend of this project is the web application that graphically displays the number of people in a room over time. It is protected by a sign-in page that is located on Northern Arizona University's network, where students and staff will need to be located physically or connected through the NAU VPN to access. User accounts and their assigned roles dictate the viewable information for that account.

After signing into the frontend sign-up page, the dashboard fetches any buildings with people counting in action. Then, after the desired room is selected, the room's data is fetched and displayed. An account with administrator privileges has the option of viewing more specific data than non-admin users. Administrators also can manage other users and their roles.

Integration testing the frontend of our project consisted of utilizing Jest for mocking API calls. While writing the integration tests, it was realized that the current method of creating these API calls was incorrect. In response, the frontend required a large rewrite based on decoupling the API query code from within the React components. After completion, the tests were written in compliance with the new methods. These tests check that the frontend gracefully handles successes and failures returned from the backend. If the connection is entirely broken and no response is returned, the frontend also gracefully handles these occurrences. Each query function was tested separately from one another, with a total of two tests each. This ensures that the connection between the frontend and backend is working as intended with both successful and unsuccessful queries.

### *7.2.2 Integration Testing for Backend*

Since the backend does not communicate to external services, integration testing was not needed for the backend.

### *7.2.3 Integration Testing for IoT Device*

The Raspberry Pi connects specifically to the backend of the solution so that the backend can send its counts to the database. This involves communication with the backend's IP. This needs to be correct, otherwise, no connection can be made. Other than the HTTP response string sent back from the database, it is a one-way communication from the Raspberry Pi to the backend.

A Pi makes a post request to the backend each time it sends a count. This post request contains all the information regarding the endpoint and must be entered into the database in a specific format for the web application to display properly. A function was created to assert the data is formatted correctly before submission to the backend.

Once the string is verified, it is sent to the backend. The post request returns a status 200 verifying that the data was in the proper format and that the data was sent correctly. Whenever a status 400 or 500 is returned the string may be in an improper format or the connection to the backend is configured improperly. The message will return valuable information to help a developer determine if it is a connection issue or the JSON string was built in an improper format. A function harness was created asserting the post request string contains a status 200 each time data is sent to the backend.

## *7.3 Usability Testing*

### *7.3.1 Usability Testing for Frontend*

Usability testing focused on confirming that user interaction and experience with the frontend are at an acceptable level. This testing highlights if there is any trouble or misunderstandings about the layout of the web application as a whole, whether that is on desktop or mobile devices. Each page was tested for visual and functional usability, on multiple different screen sizes.

### *7.3.2 Usability Testing for Backend*

The backend of this solution is not built for basic user interaction. Instead, only a developer will ever be interacting with the core components of the backend. We tested this portion of the backend by utilizing the documentation provided that details the setup of how the backend will run. The usability testing for the backend was successful and users were able to use the provided endpoints.

### *7.3.3 Usability Testing for IoT*

Due to the autonomous nature of the IoT device, usability testing will not be available.

# 8.0 Project Timeline

Section 8.0 outlines what modules we designed, and implementation decisions that our team made to create the final product for our client.

Figure 10.1 displays the fall semester where we planned out our project by doing initial research into what software could be used to develop the final product.

The fall semester was mainly focused on documentation. The first of many tasks was the team website that introduced the team, client, and mentor, along with details about the project.

The next part of the project was the technological feasibility document which talked about what software and technologies were to be used for developing the project, and why the team decided to use said tools.

The next big task of the project was the software specification document, which included updated information about the project, and a roadmap to begin development.

Finally, the last big task in the fall semester was the design review where the team finalized how the project was to be developed and created a development timeline.

**Team PiWatcher - Fall Schedule**

| Deliverable | Start Date | End Date | Duration |
|---|---|---|---|
| Team Standards | 8/31/2020 | 9/18/2020 | 3 |
| Team Inventory | 8/31/2020 | 9/19/2020 | 3 |
| Mini-Intro | 9/7/2020 | 9/25/2020 | 3 |
| Team Website | 9/14/2020 | 10/2/2020 | 3 |
| Technological Feasibility Draft | 9/21/2020 | 10/9/2020 | 3 |
| Technological Feasibility Final | 10/12/2020 | 10/23/2020 | 2 |
| Requirements Specification Draft | 10/5/2020 | 11/6/2020 | 5 |
| Requirements Specification Final | 11/9/2020 | 11/20/2020 | 2 |
| Design Review I Draft | 10/19/2020 | 11/6/2020 | 3 |
| Design Review I Final | 11/9/2020 | 11/13/2020 | 1 |
| Technical Prototype Demos | 11/9/2020 | 11/27/2020 | 3 |

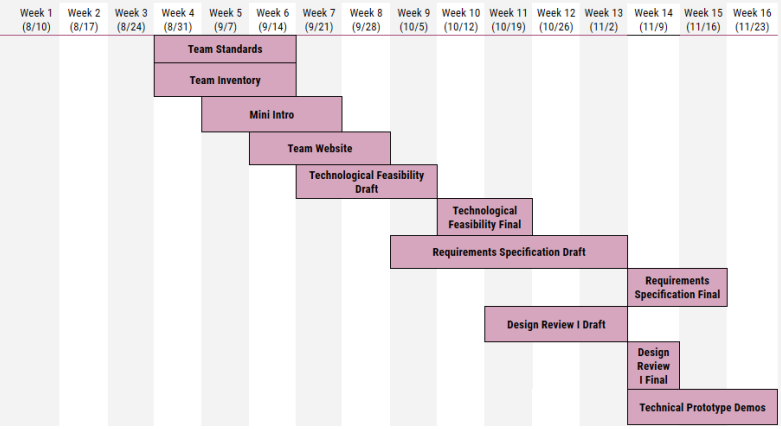


Figure 10.1: PiWatcher Fall Semester Schedule

**Team PiWatcher - Spring Schedule**

| Deliverable | Start Date | End Date | Duration |
|---|---|---|---|
| Milestone I: People Recognition & Counting People | 1/11/2021 | 2/5/2021 | 4 |
| Software Design Document | 1/11/2021 | 2/12/2021 | 5 |
| Milestone II: Cont. Counting People & Connecting Infrastructure (MVP) | 2/8/2021 | 3/5/2021 | 4 |
| Design Review II | 2/15/2021 | 2/26/2021 | 2 |
| Full Prototype Tech Demo Milestone Spec. | 3/1/2021 | 3/12/2021 | 2 |
| Milestone III: Bugfixing & Optimization | 3/8/2021 | 4/2/2021 | 4 |
| Software Testing Plan | 3/8/2021 | 4/2/2021 | 4 |
| Design Review III | 3/22/2021 | 4/2/2021 | 2 |
| Completed Team Website | 4/5/2021 | 4/16/2021 | 2 |
| Capstone Presentation | 4/5/2021 | 4/16/2021 | 3 |
| Milestone IV: Software Testing and Product Finalization | 4/5/2021 | 4/23/2021 | 4 |
| Final Product Acceptance Demos | 4/12/2021 | 4/23/2021 | 2 |
| Final Project Delivery | 4/26/2021 | 4/30/2021 | 2 |

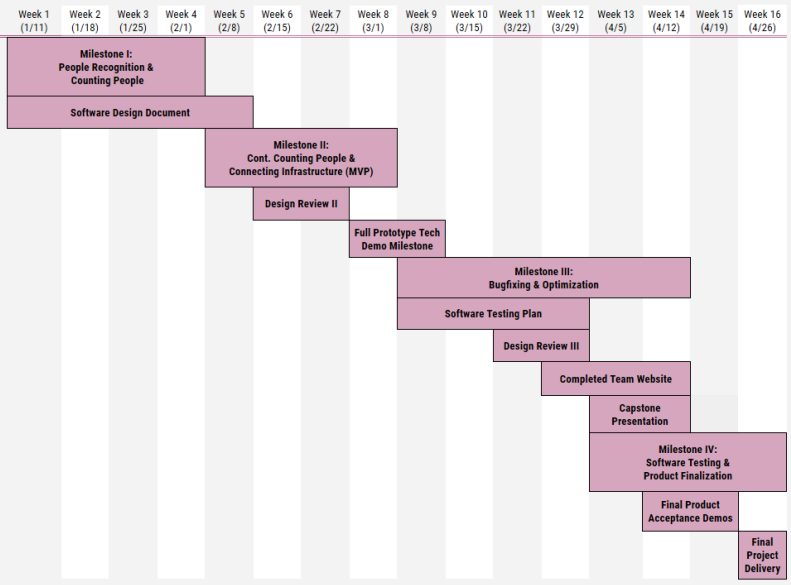


Figure 10.2: PiWatcher Spring Semester Schedule

In figure 10.2, the Gantt chart displays our team's spring timeline for the project based on the requirements and major milestones that our team needed to reach.

During the first milestone of spring semester, the main focus was to get a working implementation of our people recognition counting module for the IoT device. In this phase, the IoT device needed to be able to process a video feed from the camera, and recognize and count how many people were present in the space that was being observed.

During the second milestone in the spring semester of our project, our team's main goal was to finish the development of the minimum viable product of each module of the entire infrastructure. Once the minimum viable product of each module was completed, we started integrating and connecting the entire system together to complete the minimum viable product of our system.

During the third milestone in the spring semester of our project, our team's main goal was to focus on application optimization and further development. Our team used the baseline of our minimum viable product to determine which additional features to add. We also looked for ways to optimize our people recognition and counting module to increase accuracy and maintain as well as improve performance. This phase of the project aimed to meet the final requirements of our client as we moved on towards software testing and product delivery.

During the fourth milestone in the spring semester of our project, our team's main goal was to prepare the product for product delivery. Our team focused on implementing unit testing and integration testing, as well as needed bug fixing, along with final features that were added to the final versions of our product.

After the fourth milestone in the spring semester, our project was complete. We began the process of formally delivering the product to our client. We introduced the product and demonstrated how to properly use the tool as envisioned. Along with that, we included a user manual that contains detailed information including setting up each IoT device and utilizing the metrics provided by the web application.

# 9.0 Future Work

Future work for this project includes more in depth development regarding our frontend, backend, and people detection system. The user role management system could be expanded to further tailor permissions to particular users in specific buildings and handle more minute use cases. The inclusion of CAS would further improve integration of NAU users and provide better cohesion with other applications used within NAU. The frontend could expand its CSS and styling to better fit multiple screen sizes and its alert system within the app could be expanded on to better convery successful and failed

actions. SSL encryption could also be handled within the production environment in a future release. Improvements on our people detection accuracy could be achieved by training custom models for better handling of our room detection system.

# 10.0 Conclusion

The goal of our team was to develop a people counting solution to accumulate data regarding the footfall of buildings and classrooms on the NAU campus all while being cost-efficient. There are many edge cases needed to be accounted for in the act of counting people directly.

During our research, our solution has come down to four issues. First, we needed to find the most capable hardware for our use case that would fit into our limited budget. We chose the Raspberry Pi, due to its low cost and moderately capable processing power. Next, was finding an open-source software that would work properly with our chosen hardware and provide accurate results when keeping track of people. OpenCV and Tensorflow Lite ran on the Pi and accounted for people successfully. After our first two solutions had manifested, we began to picture our accompanying web application. A modular, scalable front-end framework was needed with the ability to update and refresh in real-time to display the accumulated data. We planned our solution in React due to its high capability in our testing. In between the Pi and the application's front end is our back end, Flask. Flask handles data consumption from the Pis and is accompanied by MongoDB, our storage system. All connected, our IoT devices count the people in view and send these counts to the back end for storage. The front end connects to the back end and retrieves this data for viewing with authentication being handled by our own LAS system. Our solution has given an inexpensive and effective way to count people within NAU buildings and classrooms as well as provide these short and long-term statistics in a business-centric way.

# Glossary

IoT - (Internet of Things) a device that connects to the internet that is embedded with a sensor that transmits data. In people counting infrastructure is the Raspberry Pi with a camera that sends the count of people to the back end.

LAS - Login Authentication Service.

MMS - Mongo Manager Service.

MongoDB - a NoSQL database program.

OpenCV - an open-source computer vision and machine learning software learning.

Raspberry Pi - a low-cost, credit-card-sized computer that plugs into a computer monitor or TV, and uses a standard keyboard and mouse.

TensorFlowLite - an open-source deep learning framework for on-device inference.

YOLO (You Only Look Once) - a deep learning model used on the IoT device for recognizing people.

# Appendix A

This section outlines the hardware, toolchain, setup, and production cycle of our product.

*Hardware*

- Raspberry Pi 4
- Pi Camera

*Toolchain*

In this section is a list of software used to develop the People Counting Infrastructure:

- Docker
- MongoDB
- Visual Studio Code
- Node JS

*Setup & Production Cycle*

There are multiple parts to the people counting infrastructure to set up and it will start with the IoT device, then the backend, and finally the frontend.

The IoT device is set up on a Raspberry Pi device, a detailed setup of the IoT device is here:
https://github.com/PiWatcher/pci-iot/blob/main/README.MD.

Once the setup of the IoT device is complete, the next step is to set up the backend. The backend is set up on docker containers, in the containers is the database, MongoDB, the server host, Flask, and finally the server for the front end that is the react server. A more detailed setup and production cycle of the back end is here:
https://github.com/PiWatcher/pci-backend/blob/main/README.md

The last part of the setup is the front end, it is where the user will be seeing a web application that displays the data on the People Counting Infrastructure onto graphs. A more detailed setup of the front end is here:
https://github.com/PiWatcher/pci-frontend/blob/master/README.md

Each part is essential and should be connected to each other in order to work together.