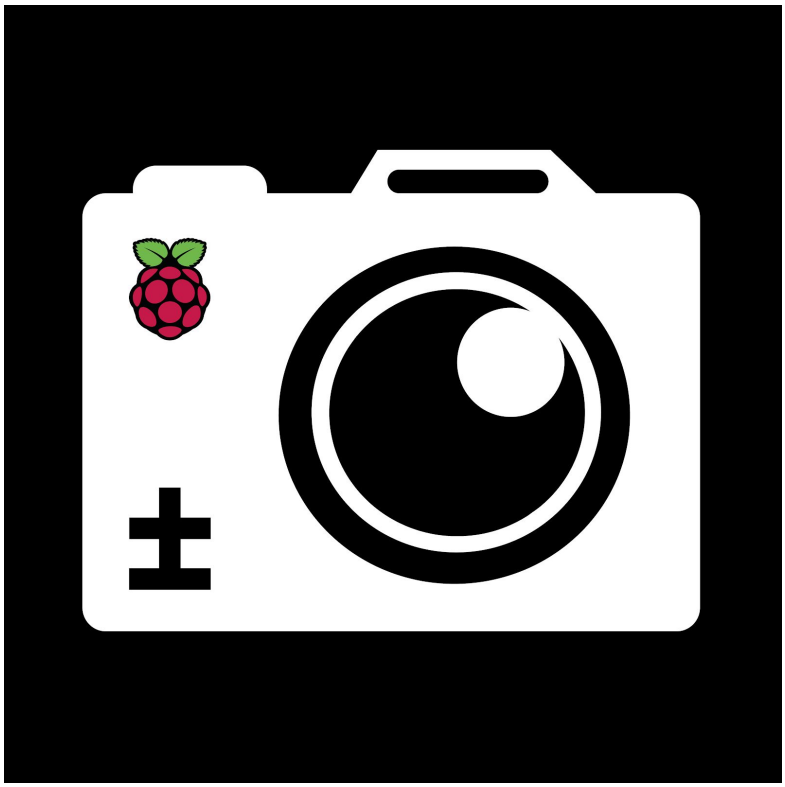

Technological Feasibility

Date: 10/09/2020

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

Table of Contents

1.0 Introduction

The rules and restrictions set by local and Federal governments worldwide in response to the global pandemic has become paramount for larger organizations to provide structures that keep their members, as well as the public, safe from harm. One aspect to the pandemic that affects transmission potential is population density. There is no straightforward way for large organizations, whose buildings are set up to accommodate large numbers of people, to keep good track of the population flow in and out of their buildings.

Current solutions for controlling population density are incredibly expensive. Each solution utilizes a combination of technologies ranging from infrared, lasers, and cameras to keep track of the number of people entering and exiting buildings. This is unfortunate when an organization has multiple buildings because cost limits the number of devices that can be deployed. NAU currently utilizes a technology that costs upwards of $1000 per deployment. This technology provides a people counting infrastructure that allows administrators to keep track of population density, trends, and various other metrics.

Our proposed solution aims to solve the challenges that the current technology does not address. These challenges are both price, flexibility, and scalability. Using a Raspberry Pi with an attached PiCamera, each device would cost less than $100 per deployment, allowing the university to scale our people counting infrastructure to nearly all buildings and classrooms. Much like the current solution, our team will provide a web application that would allow administrators to view metrics and trends. The most important aspect of this project is that it will be open source for anyone to use. This allows any organization to utilize this technology in their environments. This eliminates the need for organizations to purchase expensive, closed source solutions. In the end, our proposed solution will provide these organizations with the ability to properly track population density and appropriately allocate business resources.

2.0 Technological Challenges

This section will investigate the design decisions that will be made in order to have a set of technologies that will produce a working product to our client. Below is a brief outline of our technological challenges that will be discussed in this document.

2.1 Object Detection/Tracking Library

The main obstacle our team will face is how we detect and track people moving in and out of spaces. Our system will need to accurately find and track people entering and exiting spaces in real time with a live video feed. This will be done utilizing libraries from languages that give us the fast up to date processing we need.

2.2 Database Management System (DBMS)

We will need a DBMS to store and access data we gather from tracking people going in and out of spaces. Both our RaspberryPi and web application will need to have easy access to the database in order to give accurate information to our users.

2.3 Front End Web Application Framework

This project will utilize a front end framework to give a visual representation of the data collected. This framework will need to have a seamless connection with our backend in order to display accurate and up to date data.

2.4 Back End Web Application Framework

The back end framework section will investigate frameworks whose main job is connecting and ingesting data from the database into a usable format for the front end to display. Our solution must be capable of handling an influx of information and managing it properly on demand.

3.0 Technological Analysis

This section will address the major technological issues, challenges, and design decisions that our team has identified as critical to our project's success.

3.1 Object Detection/Tracking Library

The main requirement for our capstone project is to detect and track people from a live video feed. The main point of detecting and tracking people is strictly for counting the occupancy of a room/building thus there is no need for any other type of recognition. The only way to do this is by using multiple Deep Learning algorithms therefore we have decided to use a library previously made in order to get all the information we need. This library will be able to ideally implement both image detection and tracking. If the library is only capable of one we will need to choose another that can do the other portion.

The main methodology used for detecting objects is a type of Convolutional Neural Network (CNN). This CNN will have two stages; one being extracting certain regions of the video and the second being classifying that region, in our case classifying it as a person. This CNN will work behind the scenes and will be done by the library we decide to use.

3.1.1 Alternatives

The main challenge our team faces with our implementation of object detection and tracking is the following:

- **Easy-of-use**: How easy is the library to use, is it well documented? Are there any tutorials in place already to go off of?
- **Accuracy**: How accurate is the library on a live video feed vice picking people out of a photo?
- **Raspberry Pi Compatibility**: How compatible is this library when implemented on a Raspberry Pi?

3.1.1.1 OpenCV

Open Source Computer Vision Library (OpenCV) is an open source computer vision and machine learning (ML) library. This library offers a variety of algorithms to use for tracking and detecting objects [2]. These libraries can be used as a base to build off of for detecting and tracking people/objects.

Pros
- Vast methods and classes
- Provides set up for tracking people
- Tutorials exist for implementing basic people tracking software on a Raspberry Pi

Cons
- Documentation is not straight forward
- Models not as customizable

3.1.1.2 TensorFlow/Tensorflow Lite

Tensorflow is an open source platform that offers multiple levels of ML models. TensorFlow Lite is a set of tools that assist us in running TensorFlow on IoT devices. This is a more efficient form of TensorFlow specifically optimized for putting and using models on IoT devices [3]. It also offers a variety of pretrained models to choose from so we do not have to create our own.

Pros
- Able to be optimized for a IoT Device
- A lot of well done documentation
- Tutorials exist for simple people tracking system

Cons
- Slower compared to competitors
- Can be difficult to debug
- Steep learning curve

3.1.2 Analysis

Our team looked over different tutorials on implementing detection and tracking of objects on a Raspberry Pi. This confirms the Raspberry Pi capability and from the tutorials we see that not much is needed from us to implement the tracking. We mainly need to set up the infrastructure for when the Pi receives a video feed which will require some external libraries but ones that are standard when dealing with ML. As with any ML model there is going to be a learning curve with testing and optimizing our model.

OpenCV's provided documentation is not the friendliest . While it does provide different documentation for different versions the documents themselves are not the easiest to navigate. They do however provide a lot of examples to go off of for various implementations of their modules [4].

While using TensorFlow, once a model is chosen or created it needs to be run through a converter in order to get it in a TensorFlow Lite format and be compatible with the Raspberry Pi. Then the model can not only be run on our device but we can also do optimizations to reduce the size and increase the efficiency [3]. This is offered through TensorFlow so nothing will have to be done on our part to optimize our model thus lightening the load and improving the accuracy on the Pi.

Our team ran a quick tutorial to choose, run and optimize a pre-trained model with input data given to us. It then walked us through how to optimize the model with the toolkit provided to ease the process. Most of the work is done by TensorFlow, though we do have the option to make our own model and provide our own data if needed. The API is well documented with descriptions for all methods as well as source code to see what is happening behind the scenes.

3.1.3 Chosen Approach

Technology	Easy of Use	Accuracy	Pi Compatibility	Total Score
OpenCV	**4**	**3**	**5**	**12**
TensorFlow/ TensorFlow Lite	3	4	5	12

Table 1: Object Detection/Tracking Library Summary Table (ratings out of 5)

We decided to choose OpenCV for our library. While the models we create may not be as customizable thus yielding a lower accuracy we believe it will be enough for our use case. Some of our members have used this in the past making the learning curve lower than the other thus making it easier to use in the long run.

3.1.4 Proving Feasibility

For our demo, we will be creating a model to run against a live video feed to begin the tracking of people. We will use predesigned models and data to do this given by other open source resources or within OpenCV itself. After this we will be working on testing it against “crowds” of people to test the accuracy and tweak the model if needed to improve said accuracy.

3.2 Database Management System (DBMS)

For our platform to track the number of people passing through our cameras, an efficient database management system with quick query times is essential. The sometimes massive data throughput from our many Raspberry Pi devices must not bottleneck our platform or miscounts will be probable and the accuracy of the platform as a whole is tarnished.

Connected through our backend, the DBMS will record the given data into a database and update the entries on a regular basis. When connected to our front end, the databases will provide the necessary data to give the user the ability to check the counts often and in real time. For our team to achieve an optimal system, the DBMS needs to be correct for the job. *MongoDB*, *CouchDB*, and *MySQL* are the top three of which our team has found during our research for a suitable DBMS.

3.2.1 Alternatives

The main challenges our team faces with our implementation of a DBMS are the following:

- **Ease of Use:** Is the database easy to implement? Are endpoints easy to add and remove from the system?
- **Compatibility:** Is the DBMS compatible with our chosen language and schema?
- **Performance:** Does the database system have acceptable performance for our situation?
- **Cost:** What does the DBMS cost? Is it open source?

3.2.1.1 MongoDB

MongoDB is a popular general purpose NoSQL database focused on scalability and ease of development. This DBMS was conceived in 2007 by the team at DoubleClick to suit their needs of a less rigid schema. MongoDB is now developed by MongoDB Inc. and is open source . The database system is written in C++. [9]

Pros:

- Extensive documentation
- Large community
- Multiple language support

Cons:

- Does not have expansive data replication features

- Open source (free to use)
- Fast
- Scalable
- Flexible document model

3.2.1.2 CouchDB

CouchDB is an open source NoSQL DBMS by Apache. They're tagline is "relax" to emphasize their focus on reliability and data safety. [11] It was originally introduced as an Apache Software Foundation project in 2008 [10]. The database is written in Erlang and is open source.

Pros:

- Extensive documentation
- Multiple language support
- Open source (free to use)
- Better data safety features
- Flexible document model
- Large community

Cons:

- Not as a fast
- Less scalable

3.2.1.3 MySQL

MySQL is possibly the most used database system of all. Originally released in 1995 by the Swedish company, MySQL AB, it has since been acquired by Oracle and is one of the go to RDBMS systems with its inclusion in the LAMP (Linux, Apache, MySQL, PHP) stack. Written in C and C++, the RDBMS is open source and free to use. [12]

Pros:

- Extensive documentation
- Easy to use
- Large community due to age
- Multiple language support
- Open source (free to use)

Cons:

- Relational model could be too rigid for our application

3.2.2 Analysis

From our research, we have found that each of our prospective databases is representative of a specific niche within the DBMS spectrum. All three have shared characteristics, such as being open source and being highly compatible with different languages and libraries. Yet, their differences are what separate them from qualifying better or worse for our project.

MongoDB and CouchDB are both document model databases with flexible (or no) schema with large amounts of documentation available, yet they market themselves as having different purposes. MongoDB is a DBMS created for performance with easy project scaling and quick query times. CouchDB is slightly less about those characteristics and more about data safety and reliability. Our third option, MySQL is a relational model database with a rigid schema, but with decades of backing and work behind it the database software seems to be the go-to for many developers looking for a quick and simple database to employ in their project.

3.2.3 Chosen Approach

Technology	Ease of Use	Compatibility	Performance	Cost	Score
MongoDB	**5**	**5**	**4**	**5**	**19**
CouchDB	5	4	3	5	17
MySQL	5	4	5	5	19

Table 2: Database Management System (DBMS) Summary Table (ratings out of 5)

Although MongoDB and MySQL have been rated the same scores in our findings. MongoDB will be the chosen database management system for our project due to it's document based storage platform. The DBMS is a fast, scalable option with lots of documentation and a large community behind it.

3.2.4 Proving Feasibility

To prove the feasibility of our chosen DBMS, we will first check for the successful connection with our front end and back end to be sure they are compatible. We will then check the database's performance by attempting to simulate the data collection of the Raspberry Pi endpoints. By sending queries at various intervals, we can get an idea about how the database will respond in a real world scenario. In the event of MongoDB not satisfying our requirements, CouchDB will be given a second look for testing.

3.3 Front End Web Application Framework

Providing the users of our people counting solution with a professional, clean looking user interface is paramount to its effectiveness. Luckily, there is no shortage of libraries and frameworks to aid in the process of user interface design and creation. A good user interface design is simple to understand, useful for the customer's needs, and should be able to provide them with accurate information in a well thought out format. The interface should be responsive and consistently refresh to collect the most up to date information from the back end of the people counting solution. The front end should also be mobile friendly, although the focus of the front end design for this particular project will be toward desktop users. We'll most likely be refining the design of this front end to reflect the ideas of our client, but make sure to focus specifically on the base functionality and user experience before more advanced features are added.

3.3.1 Alternatives

Upon further investigation, there seem to be a few different options available when it comes to front end application frameworks. Each technology listed has its pros and cons, but there is a consensus on what requirements we'd like to abide by when utilizing a particular front end library. These libraries are: *Bootstrap*, *React*, and *Angular*. The requirements that we'd like to fulfill when it comes to utilizing these technologies are:

- **Performance:** How responsive is the front end library? When a user executes an event on the sight, are they able to quickly access the data that they're requesting?

- **Ease of Use**: How quickly can the customer access the data that they need? Is the data presented cleanly such that it is not a headache to interact with the system? Does the data cleanly represent what the database contains at any given time the user decides to log into the web interface?

- **Data integrity**: Does the front end contain accurate information from the database? Are there tools available for representing the data? Is the user allowed to access all the data or are there restrictions? What kind of data should be represented?

- **Learning Curve**: Is the front end simple enough to design so that we won't get bogged down trying to understand complicated language libraries, and be able to present an adequate solution to the client within a given time frame?

3.3.1.1 Bootstrap

Bootstrap is a front end tool kit that simplifies nearly everything about the layout and design portions of a website. The library focuses exclusively on making the construction of the website as simple as knowing the classes provided by Bootstrap. This allows the developer to focus less on the technical aspects of CSS and javascript since Bootstrap is primarily composed of these two front end languages.

A website becomes increasingly challenging to develop when mobile devices are considered. Bootstrap takes complete care of this issue by providing classes that are both desktop and mobile responsive. There is no need to dig into complicated CSS media queries, Bootstrap takes care of that for you. In fact, the developer doesn't really need to use CSS at all for design unless you care to implement something very specific.

It can be cumbersome to implement a Bootstrap site that allows for excellent performance. Other event driven javascript intensive libraries are most likely going to be more effective when performance is the driving force behind the design of a particular site. It is possible to merge both a front end library with an event driven library, such as React JS. This could end up happening for this particular people counting project.

Although the more developed templates that are built using the bootstrap framework may cost money, Bootstrap is mainly a free tool to use and develop with.

Pros

- Easy to use
- Desktop and mobile compatible
- Extensible
- Very responsive
- Extensive documentation
- Free

Cons

- Not the best performance
- Void of back end communication library
- Not meant for enterprise level development

3.3.1.2 React

React is Facebook's way of dealing with interactive websites. There are many advantages to utilizing this powerful framework to build responsive websites that include

a lot of user interaction. It may be useful for us to implement a front end solution that includes something like React JS.

The efficiency of React is one of its biggest pluses. When a user interacts with a site built on React, only the component that the user interacted with at that point is updated. This prevents the entire page from having to reload when the user interacts with anything on the website. React can improve performance dramatically, especially for something with less processing capability such as a mobile device. These components can be stateful, such that, when a user interacts with them, the state of said components are updated. Once a component is developed, it can be used throughout the site wherever it is necessary. This makes websites scalable, and because of the nature of components in general, more complex components can be built upon simple foundations.

There are however, a few cons to consider when dealing with React JS. One of these is a steep learning curve that includes how to structure React components. The style of language used to structure components is a bit complex, especially for someone just getting into programming or learning how to write in JavaScript for the first time. In fact, a new way of styling JS code needs to be learned, called JSX. The braces and code that reside within this particular styling are a bit different than what would be expected from vanilla javascript styling.

React is not the only front end javascript library available on the market. Because of this, it is possible for alternatives to come up that may have less of a learning curve or better functionality for our customer's needs.

Pros

- Backed by Facebook
- Reusable components
- Scalable
- Multi-page
- Performance Oriented
- Extensive documentation
- Free

Cons

- Steep learning curve
- Not automatically mobile compatible

3.3.1.3 Angular

Angular is a single page web application framework that was developed by Google. Using a tool like Angular ensures that you'll be able to accomplish what you want with the backing of the massive amount of development Google has poured into the

platform. It includes many features such as tooling for communicating with the back end as well as a powerful command line that allows you to develop quickly and efficiently. This can also be a con however. Given the expansive tooling around angular, it is easy to find multiple different ways to do one specific task. Angular's documentation is thorough, but it can be irritating to start building a feature and realize that you have been implementing it with two different methods at once.

One important feature of Angular is that it is easy to scale web applications. Tools are included that allow for development on both desktop and mobile platforms. The structure of Angular is such that the client is able to load what they need locally with minimal communication with the server. This makes Angular applications very snappy, which from a client's point of view, is a very good thing!

One problem with Angular is that it is focused around single page web applications. While this is good for many projects, ours will be built around the idea of multiple user views, so Angular in our case is not a great pick for a front end framework. Another issue with Angular is the steep learning curve associated with its initial implementation. This comes with most front end frameworks, but Angular is especially difficult to learn. In our case, we'd like to minimize the impact of spending time learning difficult technologies, yet another reason why Angular is not ideal for our purposes.

Pros

- Backed by Google
- Reusable components
- Scalable
- Performance oriented
- Extensive documentation
- Free

Cons

- Steep learning curve
- Single page app oriented
- Documentation redundancy
- Divided community

3.3.2 Analysis

The three front end frameworks mentioned in this document are all fully supported tools that simplify the front end development process significantly over raw full stack development. Each different framework can be used in different ways, for different purposes. It's clear after going through each of these technologies that there are a couple options that speak to us more clearly than the other.

Learning curve is one common aspect of these tools that our team is focused on. Only one of these tools, Bootstrap, has a very slight learning curve. In fact, it is so easy to learn bootstrap, it's likely easier to build a bootstrap webpage than it is to use raw

HTML, CSS, and JavaScript. Both Angular and React are difficult tools to learn. That's not to say Angular and React aren't useful or powerful, they certainly are, but unfortunately that means we'll have to focus on the one whose learning curve is the slightest.

Front end frameworks can often overlap, and that's what we see with Bootstrap and React. There's an entire development combining the two frameworks called React Bootstrap that takes the advantages of both frameworks and pools them into one more powerful framework. It is likely that we'll end up using this tool to simplify the development process.

3.3.3 Chosen Approach

Technology	Performance	Ease of Use	Data Integrity	Learning Curve	Total Score
Bootstrap	**3**	**5**	**5**	**5**	**18**
React	**5**	**4**	**5**	**4**	**18**
Angular	5	2	5	1	13

Table 3: Front End Framework Summary Table (ratings out of 5)

Strangely enough, there is no one chosen approach between the three frameworks. Both React and Bootstrap have their place for different aspects of web development. While React is focused around the performance and responsiveness of websites, Bootstrap is focused around layout and design for both mobile and desktop devices. Bootstrap is very easy to learn while React has a decently sized learning curve. Luckily there is a development in the works called React Bootstrap that combines the two frameworks in such a way that allows for both responsiveness and elegant design.

3.3.4 Proving Feasibility

In order to prove the feasibility of React and Bootstrap, we will be learning how to develop in these languages through documentation analysis and building a sample website. This will allow us to more adequately judge what we can expect in terms of a learning curve for getting into development using these languages. Since they are free, we know right off that bat that we don't need to get into contact with our client to help us fund these tools which is a pretty big deal. Web debuggers on Firefox and Chrome offer metrics on performance so we'll be able to determine how responsive the sample site is as we develop it. To make sure that we can make server side calls, we'll implement

code that tests this feature. All in all, it should be pretty easy to determine the feasibility of these two web technologies.

3.4 Back End Web Application Framework

The ecosystem of our project solution is going to be composed of Raspberry Pis', a web application, and a database. Our product needs to have a back end framework to connect all these pieces of the overall product together. Each of these Raspberry Pis' will need to communicate and send processed video feed data to the database and web application. Since our solution is going to utilize multiple Raspberry Pis' across multiple buildings on campus and potentially the Flagstaff community, we need to make sure that these devices are able to communicate effectively without bottlenecks.

3.4.1 Alternatives

During our research, the potential backend frameworks that were found that could address our technological issues and challenges are: *Django*, *Flask*, and *Express*.

Making the correct decision between these frameworks rely on key criteria. The identified key criteria are:

- **Ease-of-use**: Is the back end framework easy to use? What is the learning curve of this framework? How large is the documentation? Does it have a well established community?

- **Scalability**: How does it handle multiple connections coming in? Will it be able to handle the load?

- **Security**: How can we utilize the framework so that the interaction between the users and the web application is secured?

- **Performance:** How fast is the framework? Does the performance decrease when the application scales?

3.4.1.1 Django

Django is a popular and widely used free and open source web application framework that utilizes the Python programming language. This web framework was created back in 2005 to allow developers to build web applications rapidly with an emphasis on security and scalability [5]. Along with that, Django also provides plugins, modules, and a front end framework that follows a model-template-view architectural pattern.

Pros

- Scalable
- Focuses on security to prevent things like (SQL injections and XSS)
- Rich documentation
- Large community
- Provides user authentication

Cons

- Steep learning curve
- Performance dependent on good design
- Not good for static one-pagers and microservices [6]

3.4.1.2 Flask

Flask is another popular free open source Python-based web application microframework that was initially released in 2010 to develop simple and small web applications. Flask keeps the core of the back end simple while also making it extensible allowing us to add our own functionality and libraries to allow our application to work effectively. This allows us to easily add and integrate numerous extensions that Flask does not provide such as database integration, form validation, uploading handling, open authentication technologies and more [7]. Flask provides useful documentation on how to install, setup, and get started on a simple application. They also provide documentation on how to test Flask applications.

Pros

- Minimalistic and lightweight
- Easy to learn and easy to set up
- Lots of useful documentation
- Very flexible and extensible

Cons

- Does not provide much out of the package
- Need to find extensions for database integration, form validation, authentication, etc.
- Not-asynchronous - can only handle a single request at a time

3.4.1.3 Express

Express is a popular free open source web application framework that utilizes Node.js that was initially released in 2010. Express paired with Node.js allows developers to easily create robust API quickly using JavaScript. Along with that, Express has extensive documentation on how to install the framework, how to utilize the tools it provides, guides on how to create routes, and so much more [8]. On top of the easy learning curve and great documentation, Express also allows support for multiple plugins while also being lightweight and flexible.

Pros

- Minimalistic and lightweight
- Easy to learn and easy to set up
- Lots of useful documentation
- Very flexible and extensible
- Large community

Cons

- Does not provide a lot as it has very few dependencies

3.4.2 Analysis

After further analysis into each tech, we can easily see that these three frameworks are popular, supplied with rich active communities, and are able to solve the technological challenges that our product faces.

Comparing between Express and Flask, we can easily see that they provide the same pros and cons with the only difference being the programming language that is used. Both Express and Flask offer a lightweight and minimalistic option that is easy to learn and easy to set up. Since the framework is very lightweight and minimalistic, the framework is a lot easier to learn than other frameworks. However, further analysis on different libraries/frameworks to use for database integration, user authentication, and many more needs to be inspected since they do not already provide these tools.

Taking that into consideration, Django offers a scalable and secure option. Just like Flask, Django utilizes Python for development. In contrast to Express and Flask, Django comes preloaded with packages for database integration, user authentication, and its own front end framework. However, because of that Django has a steep learning curve in comparison to Express and Flask.

In terms of scalability, large companies such as Pinterest, Instagram, Netflix, and IBM utilize these technologies. However, Django is often recommended over Flask if you are looking to build an application that needs to scale.

3.4.3 Chosen Approach

Technology	Easy of Use	Scalability	Security	Performance	Total Score
Django	**3**	**5**	**5**	**4**	**17**
Flask	5	4	3	4	16
Express	4	4	3	4	15

Table 4: Back End Framework Summary Table (ratings out of 5)

The chosen approach that we will be using for the back end framework for our product is Django. As discussed in our analysis, Django offers a scalable and secure solution, while also not sacrificing a lot of performance.

3.4.4 Proving Feasibility

In order to show that Django is the best option for our solution, our teams will create a demo website. We will use Django to connect our demo website and our chosen database together along with a test REST API to provide a place for stress testing the back end of our application. This stress test will check the application performance and simulate the amount of connections that our back end will need to handle from the Raspberry Pis'. In the scenario where Django does not provide the metrics that we need, we will use the same approach for the second alternative, Flask.

4.0 Technological Integration

This section defines how our team will address the following integration challenges on how our platform's pieces will fit together as well as what challenges we aim to solve with each of them.

4.1 Integration Challenges

All of our decisions regarding the design of our platform will have to run through one final challenge: "will our software choices run together efficiently and effectively?". To ensure that these software choices are able to run together, our team has outlined the envisioned product to address our integration challenge.

4.2 Envisioned Product

Our platform as a whole will begin at the endpoints with multiple Raspberry Pi devices and PiCameras utilizing the OpenCV library. These endpoints will process how many people enter and exit an enclosed space. The processed data will be sent to our Django back end. Django will be our link between the front end web service and our database management system. Once the processed data has been received, MongoDB will be sent the data to be recorded into the database. Every update from each endpoint will immediately be sent to our Django back end and queried into our database. From there, we will utilize our React based front end tied with our Django back end to interpret the data such that it can be displayed in an appealing and informational manner to the user.

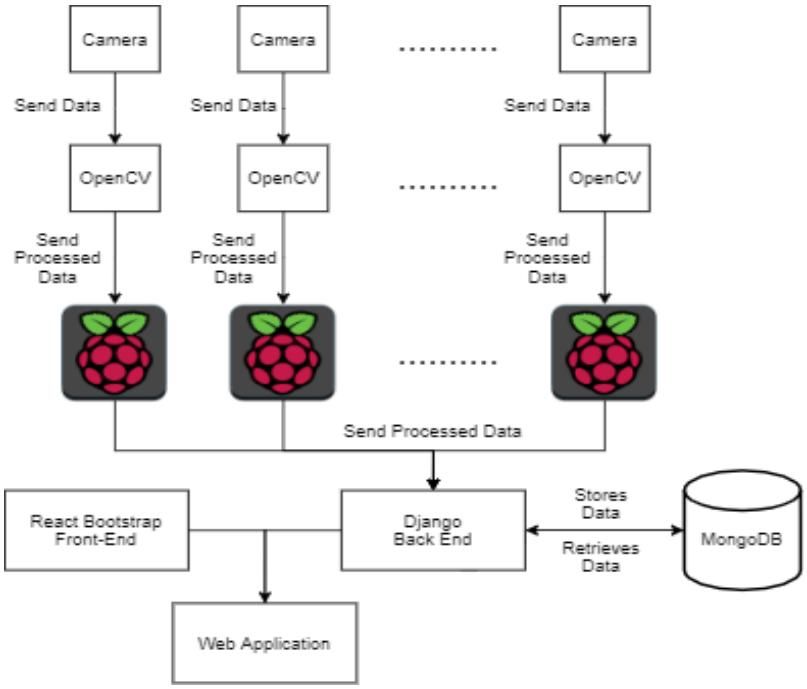

Figure 1: Diagram of envisioned product

5.0 Conclusion

Our project's purpose is to provide an effective solution to monitoring room population count and space utilization during the COVID era and beyond. Our goal with this document is to gain more knowledge of the many technologies available to be incorporated into our platform and choose our prospective solution. Our solution consists of front end and back end web services, a database management system, and an object detection library using Raspberry Pi devices with PiCameras as our endpoint data collection tools.

Technological Challenge	Chosen Technology
Object Detection	OpenCV
Database Management System	MongoDB
Web Application Front End Framework	React Bootstrap
Web Application Back End Framework	Django

With the software we have chosen above, our team is confident our solution will successfully solve our problem. PiWatcher will be a cheap, effective, and accurate people counting software platform.

6.0 References

[1] OpenCV People Counter. (2018). Retrieved October 1, 2020, from https://www.pyimagesearch.com/2018/08/13/opencv-people-counter/

[2] About. (2020). Retrieved October 1, 2020, from https://opencv.org/about/

[3] TensorFlow Lite guide. (2020). Retrieved October 5, 2020 from https://www.tensorflow.org/lite/guide

[4] OpenCV Modules. (2020). Retrieved October 6, 2020 from https://docs.opencv.org/master/index.html

[5] Django Project. (2020). Retrieved October 7, 2020 from https://djangoproject.com

[6] Django Pros and Cons. (2020). Retrieved October 7, 2020 from https://www.netguru.com/blog/django-pros-and-cons

[7] Flask Background Information. (2020). Retrieved October 7, 2020 from https://flask.palletsprojects.com/en/1.1.x/foreword/

[8] ExpressJS. (2020). Retrieved October 7, 2020 from https://expressjs.com/

[9] MongoDB. (2020). Retrieved October 7, 2020 from https://docs.mongodb.com/manual/introduction/

[10] IBM Cloud Learn Hub. (2020). Retrieved October 8, 2020 from https://www.ibm.com/cloud/learn/couchdb

[11] CouchDB. (2020). Retrieved October 7, 2020 from https://couchdb.apache.org/

[12] What is MySQL?. (2020). Retrieved October 8, 2020 from https://www.atlantic.net/dedicated-server-hosting/what-is-mysql/