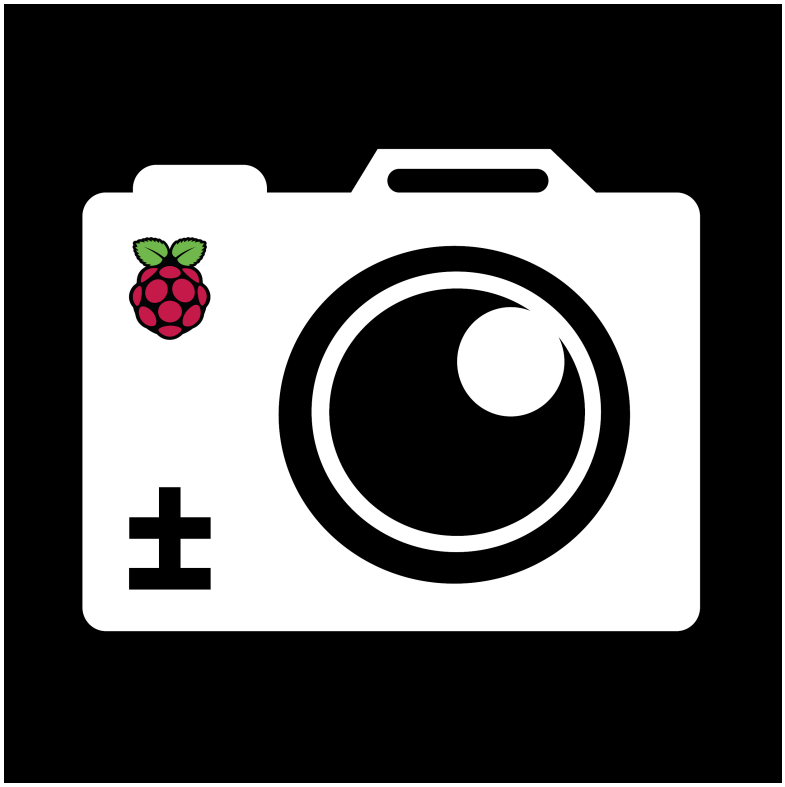

User Manual
People Counting Infrastructure
Version 1.0

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

User Manual
People Counting Infrastructure
Version 1.0 **1**

1.0 Introduction

There are certain aspects of the PiWatcher people counting infrastructure that require user interaction for setup and maintenance. These aspects are the major components that make up the people counting infrastructure including the Frontend, Backend, and IoT device. Each of these sections are split up into different subsections that include the specific installation steps that may be required for proper setup of each component.

In some cases, maintenance will be required, depending on the component. Maintenance steps will be listed below to ensure seamless operation and component cross-communication. Failure to follow these steps will result in inaccurate data collection which will affect business metrics.

Some disclaimers:
There are a few aspects of the installation that can absolutely be modified for more user friendliness, but this is not a guide of how to simplify the installation process. This guide contains the details for setup and maintenance of the beta application which the official PiWatcher github repo represents as of 4/28/2021. Any modifications made to the code from this point onward may have an affect on the installation process. It may be difficult to get into contact with the developers of the beta prototype once it is handed off officially to NAU on 4/29/2021. Any issues regarding this user manual should be directed towards NAU's IoT team.

2.0 Installation

This section will go over the procedures for installing the IoT device, backend and frontend of this application.

2.1 Installation for IoT Device

This section will review how to install all necessary components of the IoT device to ensure the application will run and update properly.

Downloading the OS Image to Micro SD

- Here is a link to the website where you can download the latest image (Note: The latest image may have differences in how to set the Pi up)
- https://www.raspberrypi.org/downloads/raspberry-pi-os/ (We recommend downloading the image without the recommended software)
- Our version of the image -> Version: August 2020

Installing the OS Image to Micro SD

- Once you have the image you can then download software to to write the image to an sd card
- Here is the software we used to so just that http://sourceforge.net/projects/win32diskimager/

- Run the Win32DiskImager utility from your desktop or menu.
- Select the image file you extracted earlier.
- In the device box, select the drive letter of the SD card. Be careful to select the correct drive: if you choose the wrong drive you could destroy the data on your computer's hard disk! If you are using an SD card slot in your computer, and can't see the drive in the Win32DiskImager window, try using an external SD adapter.
- Click 'Write' and wait for the write to complete.
- Exit the imager and eject the SD card.

Booting up the Raspberry Pi

- Insert the sd card into the Pi
- Boot up the Pi and plug it into a proper display
- Follow the set-up menu provided
- You have successfully installed the Raspberry Pi OS

Cloning The Repo Onto The Pi

- Open up a terminal window on the Raspberry Pi
- Run the following command:

HTTPS	$ git clone https://github.com/PiWatcher/PCI-Prototype-IoT.git
SSH	$ git clone git@github.com:PiWatcher/PCI-Prototype-IoT.git

Installing dependencies

- After repo has been cloned cd into it
- Run the following command:

```
$ . setup.sh
```

Now follow any prompts on the shell - This will install the correct version of Python along with any dependencies needed into a virtual environment that will be placed into the Pi's Home Directory

Starting/Deactivating the Virtual Environment

To start the virtual environment:

```
$ source ~/env/PiEnv/bin/activate
```

To deactivate the virtual environment:

```
$ deactivate
```

Running the Application

To run the application ensure you are in the cloned folder and the virtual environment is running then execute the command below:

```
$ python scheduler.py
```

Make sure to manually add the checkpoints folder to the root directory of this repo from the capstone drive

2.2 Installation for the Backend

This section will go over installing the backend for the different environments. Each section will detail how to set up the product in a production environment, development environment, and testing environment, each with it's important uses.

Setting up the product in a production environment

Download the beta release of the web application backend from this link. Once you have downloaded the product, extract the contents into a folder.

Enter the folder and edit the ***.env.prod*** file, located at the root directory of the project, and add the following environment variables into the file:

Table 2.21: Environment Variables

Variable Name	Description
MONGO_USER	The username of the MongoDB account that the REST API will use to read/write to the database. Example: MONGO_USER=apiuser
MONGO_PASS	The password of the MongoDB account that the REST API will use to read/write to the database.

	Example: MONGO_PASS=super-secure-password
JWT_SECRET_KEY	This is the secret that will be used to create JSON Web Tokens. This secret key is important for creating the appropriate user tokens needed for accessing certain routes on the Flask API. It is recommended that you use a 24 character long randomly generated string for the JWT_SECRET_KEY to ensure that signatures are secure. Example: JWT_SECRET_KEY=asupersecretjwtkey
ROOT_USER	The username to the root user of the MongoDB database. Example: ROOT_USER=admin
ROOT_PASS	The password to the root user of the MongoDB database. Example: ROOT_PASS=asecurepassword

Once you have added the required environment variables to the ***.env.prod*** file, run the **start.sh** file located at the root directory of the project. This script is responsible for loading the variables into the shell environment, composing the necessary docker containers, and creating the necessary users in the MongoDB database. When this script is executed, you will be prompted to enter either **DEV**, **PROD**, or **TEST**, type **PROD** and press Enter.

This will initialize the setup process and compose the necessary containers needed to start the backend. Once the script has finished executing head over to http://localhost:5000/api and ensure that you get this response:

```
{ “status”: 200, “message”: “Api base resource has been hit’}
```

If, for some reason, you run into an issue where this does not show up, consult the troubleshooting section for the Backend.

Setting up the product in a development environment

Open a terminal in the folder that you want the product to reside in. Use the following commands down below to clone the repository on GitHub:

Using HTTPS	$ git clone https://github.com/PiWatcher/pci-backend.git

Using SSH	$ git clone git@github.com:PiWatcher/pci-backend.git

Once the product has finished cloning to your host machine, go into the root directory of the folder and edit the ***.env.dev*** file. In this file, you will add the necessary and required environment variables to ensure that the application can run. To know what these variables are, refer to Table 2.21.

Once you have added the required environment variables, execute the start.sh script. Much similar to setting up a production environment, a prompt will ask to enter either **DEV**, **PROD**, or **TEST**, type **DEV** and press Enter.

This will initialize the setup process and compose the necessary containers needed to start the backend. Once the script has finished executing head over to http://localhost:5000/api and ensure that you get this response:

```
{ “status”: 200, “message”: “Api base resource has been hit’}
```

Setting up the product in a testing environment

Open a terminal in the folder that you want the product to reside in. Use the following commands down below to clone the repository on GitHub:

Using HTTPS	$ git clone https://github.com/PiWatcher/pci-backend.git
Using SSH	$ git clone git@github.com:PiWatcher/pci-backend.git

Once the product has finished cloning to your host machine, go into the root directory of the folder and edit the ***.env.test*** file. In this file, you will add the necessary and required environment variables to ensure that the application can run. To know what these variables are, refer to Table 2.21.

Once you have added the required environment variables, execute the start.sh script. Much similar to setting up a production environment, a prompt will ask to enter either **DEV**, **PROD**, or **TEST**, type **TEST** and press Enter. This will initialize the testing suite, to view the results of the test, open up the container and inspect the output of the console.

2.3 Installation for the Frontend

This section will go over installing the different environments for the frontend.

Setting up the production environment

Open a terminal in the folder of which you would like the codebase to reside. Use either command below to fork and clone the codebase for the frontend.

Using HTTPS	$ git clone https://github.com/PiWatcher/pci-prototype-frontend.git
Using SSH	$ git clone git@github.com:PiWatcher/pci-prototype-frontend.git

Prior to creating the Docker container, the creator will need to set the address of the accompanying backend. This can be set in the docker-compose.prod.yml under the argument ***REACT_APP_BASE_URL***. This address must be the external address of the machine that is hosting the backend.

Begin by running the following bash command:

```
$ ./start.sh
```

You will be asked for your choice of environment. Typing **PROD** and hitting enter will begin the creation of the production container. A production build of the react application will then be hosted by an NGINX service within. The container can be reached at http://localhost:80.

Setting up the development environment

Open a terminal in the folder of which you would like the codebase to reside. Use either command below to fork and clone the codebase for the frontend.

Using HTTPS	$ git clone https://github.com/PiWatcher/pci-prototype-frontend.git
Using SSH	$ git clone git@github.com:PiWatcher/pci-prototype-frontend.git

Without adjustment, the development Docker container will use localhost to look for the accompanying backend. This can be changed in the docker-compose.yml under the argument ***REACT_APP_BASE_URL***.

Begin by running the following bash command:

```
$ ./start.sh
```

You will be asked for your choice of environment. Typing **DEV** and hitting enter will begin the creation of the development container. The container can be reached at http://localhost:3001.

Running tests within the development environment

If you would like to run the testing suite for this application, open a terminal window within the codebase folder and run the following command.

```
$ npm test
```

The suite will run and notify you of any passed or failed tests within the suite.

3.0 Maintenance

This section goes over the maintenance for the people counting infrastructure will be for the front end, back end and the IoT device.

3.1 Maintenance for IoT Device

For the maintenance of the Iot device, please refer to the Raspberry Pi manual or the website: https://www.raspberrypi.org.

For maintenance on the use of the software, please refer to the reinstallation and you can go back to the installation part of this user manual. Since the Raspberry Pi is autonomous after installation. No further maintenance is required. If the Pi were to fail that would require debugging on each particular case depending on the error received.

3.2 Maintenance for the Backend

The backend is set up in docker containers and to check and see if the servers are running, input this command into the terminal.

```
$ docker container ls
```

It should show a list of containers and names of containers and if they are running. If nothing has shown up or nothing is in the container, then a reinstallation will need to be done, please refer back to the installation of the backend in this user manual.

Our team recommends checking the versioning of dependencies and packages used for the backend. All the packages used by the Flask Backend is documented in the requirements.txt file located at the root directory of the product.

REST API Endpoints

Down below are the lists of endpoints that are provided through the REST API.

Endpoint Route	Method	Description
/api	GET	This is the base URL of the REST API. Use this endpoint to test if the API is functional.
/api/mock/update	GET	A developer only endpoint that is used to mock data IoT entries in the database Request Params: building - which building you want to add to iterations - how many data points you want to add
/api/auth/signup	POST	Creates a user account in the database JSON Body: email - the email for the account password - the password for the account full_name - the full name of the user
/api/auth/signin	POST	Signs into a user account in the database JSON Body: email - the email for the account password - the password for the account
/api/auth/token	POST	Signs into a user account with a JWT token.
/api/auth/users	GET	Gets a list of users from the database
/api/auth/users	DELETE	Deletes a user from the database JSON Body: email - the email of the account being deleted
/api/auth/users/update	POST	Updates the role of a user JSON Body: email - the email of user being updated new_role - the role being changed too

/api/auth/users/update/password	POST	Updates the password of the user JSON Body: email - the email of the user being updated password - the users current password new_password - the new password to be changed too
/api/auth/roles	GET	Gets all roles from the database.
/api/auth/roles	POST	Creates a role in the database JSON Body: role_name - the name of the role is_admin - a boolean value determining if the role is an admin role can_view_raw - a boolean value determining if the role can allow raw data
/api/auth/roles	DELETE	Deletes a role in the database JSON Body: role_name - the name of the role being deleted
/api/data/iot/update	POST	Adds an entry into the database JSON Body: timestamp - a timestamp of when the entry was created building - the name of the building building_id - the id of the building count - the current count in the room endpoint - the name of the room/space endpoint_id - the endpoint name/id of the IoT device room_capacity - the capacity of the space being watched
/api/data/buildings	GET	Gets a list of buildings with entries in the database
/api/data/building/rooms	GET	Collects the most recent count from all the rooms for a building in the database Request Params: building_name - the name of the building
/api/data/building/room/live	GET	Grabs the data for the past hour for a room Request Params: building_name - the name of the building room - the name of the room
/api/data/building/room/daily	GET	Grabs the data for the past day for a room

		Request Params: building_name - the name of the building room - the name of the room
/api/data/building/room/weekly	GET	Grabs the data for the past week for a room Request Params: building_name - the name of the building room - the name of the room
/api/data/building/room/monthly	GET	Grabs the data for the past month for a room Request Params: building_name - the name of the building room - the name of the room
/api/data/building/room/quarterly	GET	Grabs the data for the past quarter for a room Request Params: building_name - the name of the building room - the name of the room
/api/data/building/room/yearly	GET	Grabs the data for the past year for a room Request Params: building_name - the name of the building room - the name of the room

3.3 Maintenance for the Frontend

The frontend does not require any maintenance other than moderating the dependencies and packages used within. We recommend checking for security patches and version updates through the Node Package Manager on a monthly basis.

4.0 Troubleshooting

This section introduces how to troubleshoot the certain components of the people counting infrastructure.

4.1 Troubleshooting for IoT Device

If counts are not sending or hitting the correct endpoint please check the environment variables in the .bashrc file in the home directory. To ensure the Pi camera is working as intended and capturing a photo of its intended location take a picture without running the application and check it manually. For further assistance in debugging print out response json received from the post request to get a more accurate representation of the error you are encountering.

4.2 Troubleshooting for the Backend

Unable to hit base resource

Check if the docker container pci-backend-dev/prod is running. If the docker container is not running, that means there was a mistake during the setup process. Taking a look at the output of the docker container will give you hints at where to start. The usual reasons for these issues are missing environment variables or typos in environment variable names.

Multiple IoT Admin accounts

To fix the issue with multiple Iot admin accounts being inside the mongodb container, go inside the docker container, login into the mongodb database as a root user. Run the following commands down below:

```
$ use Users
$ db.users.delete({“email”: “iotadmin@nau.edu”}, 1)
```

Repeat the delete command until there is exactly one Iot admin account left in the database.

4.3 Troubleshooting for the Frontend

Frontend fails to connect to the backend

Check if the ***REACT_APP_BASE_URL*** was not set up or set up correctly. If so, update the URL within the docker-compose.prod.yml and complete the **PROD** setup process again. If the ***REACT_APP_BASE_URL*** was set up correctly, check your browser’s development console as all errors and their information will be displayed within.

5.0 Conclusion

This section concludes our user manual.
From our team , we would like to thank you and appreciate you for using our People Counting Infrastructure.
With a huge amount of joyous appreciation, we are happy that our infrastructure will be serving you.