



SafeCampus Final Documentation

March 14th, 2023

Prepared by Ryan Angliss, Michael Fleagle, Jared Gogel, Elisabeth Jenkins, and George Kim

Table of Contents

Table of Contents	2
1. Introduction	5
1.1 Purpose	5
1.2 Scope	5
1.3 Definitions, acronyms, and abbreviations	5
1.4 References	5
1.5 Overall Description	6
1.5.1 Product perspective	6
1.5.2 User interfaces	6
1.5.3 Hardware interfaces	6
1.5.4 Software interfaces	6
1.5.5 Memory	6
1.5.6 Operations	7
1.6 Product functions	7
1.7 User characteristics	7
1.8 Constraints	7
1.9 Assumptions and dependencies	7
2. Problem Statement	8
3. Legal Considerations & Ethics	8
3.1 Central Washington University Policies and Procedures	8
3.1.1 CWUP 2-40-010 Acceptable & Ethical Use of University Information Technology	8
3.2 WASHINGTON CYBERCRIME ACT	8
3.2.1 RCW 9A.90.040 Computer trespass in the first degree.	8
3.2.2 RCW 9A.90.050 Computer trespass in the second degree.	8
3.2.3 RCW 9A.90.100 Electronic data theft.	8
3.2.4 RCW 9A.90.130 Cyberstalking.	9
3.3 Ethics	9
4. Requirements	10
4.1 User interaction	10
4.1.1 Outputs	10
4.1.2 Inputs	10
4.2 General Requirements	10
4.3 Functional Requirements	10
4.3.1 Exact sequence of operations	11
4.3.2 Responses to abnormal situations	11
4.3.3 Relationship of outputs to inputs	11
4.4 Non-Functional Requirements	11
4.5 Feasibility Analysis	11
4.5.1 Financial Feasibility	11

4.5.2 Technical Feasibility	12
4.5.3 Resource and Time Feasibility	12
4.5.4 Risk Feasibility	12
4.5.5 Development environment risks	13
5. Design	13
5.1 High Level Design	13
5.1.1 System Overview	13
5.2 Low Level Design	14
5.2.1 Application Structure	14
5.2.2 Module 1: Wi-Fi Data	15
5.2.3 Module 2: SMS Polling Data	15
5.2.4 Module 3: Occupancy Estimation	16
6. Software Tools Requirements	17
7. Hardware Requirements	17
8. Software Documentation	17
8.1 Wi-Fi Passive Listening	17
8.2 Occupancy Estimation	18
8.3 Rave Alert Data Upload	18
8.4 Registrar Data Upload	19
8.5 SMS Polling	19
8.6 Database Structure	20
8.6.1 Building	20
8.6.2 DeviceCount	21
8.6.3 RegData	21
8.6.4 EmergencyModeIndicator	21
8.6.5 EmergencyModePollResult	21
9. Installation	21
10. User Documentation	22
11. Conclusion	26
11.1 What we have seen in the document	26
11.1.1 Introduction Summary	27
11.1.2 Problem Statement Summary	27
11.1.3 Legal Considerations and Ethics Summary	27
11.1.4 Requirements Summary	27
11.1.5 Design Summary	27
11.1.6 Software Tools Requirements Summary	27
11.1.7 Hardware Requirements Summary	27
11.1.8 Software Documentation Summary	28
11.1.9 Installation Summary	28
11.1.10 User Documentation Summary	28
11.2 What have we learned?	28

11.3 What would we do differently?

28

1. Introduction

1.1 Purpose

The purpose of this project documentation is to define the problem, legal considerations, requirements, design, software tools, hardware requirements, software documentation, installation, and the user documentation for the Central Washington University Emergency Management software SafeCampus. The intended audience of this project documentation is the client, the development team, and others interested in the project.

1.2 Scope

The software to be produced is a web-application that allows the emergency management team and authorized personnel at Central Washington University to determine the number of people in any building on campus as accurately as possible, while still maintaining everyone's privacy. By using this app, the university will be better equipped to respond to emergency situations on campus.

1.3 Definitions, acronyms, and abbreviations

SRS – Software Requirements Specification

User – The person, or persons, who operate or interact directly with the product. The user(s) and the customer(s) are often not the same person(s)

Product – The software being developed

Access Point (AP) – A Wi-Fi connection router in a building

Short Message Service (SMS) polling - Asking individuals registered to the campus for information via text messaging

Web-application - Software that is accessible via a web browser with network access

RaveGuardian - An app currently being used by CWU to give individuals a virtual guardian while walking on campus, as well as a way of contacting and providing the user's location to 911 in the case of an emergency.

Rave Alert - An emergency use only mass-broadcast system provided to CWU by Rave Mobile Safety, the same group behind RaveGuardian. This allows for the mass communication of information and offers the ability to request exact locations from threatened individuals using SMS polling.

1.4 References

Break down on how to get probe requests and what data they collect:

<https://blog.samcater.com/capturing-beacons-and-probe-requests-of-public-wifi-access-points-the-why-how-and-stats/>

SRS Template:

<https://www.cse.msu.edu/~cse870/IEEEExplore-SRS-template.pdf>

Software Video Demonstration: <https://www.youtube.com/watch?v=i5AeVMISM4I>

GitHub: <https://github.com/CWU-Emergency-Management-Project/Emergency-Management>

1.5 Overall Description

1.5.1 Product perspective

This product will rely on three methods for gathering data - APs and Wi-Fi monitoring used to get queries of phones with Wi-Fi enabled near an AP, an occupancy estimation calculation based on registrar data, building floor plans, and probabilities, and SMS polling nominally using data from RaveAlert. If the RaveAlert integration does not go through, then the product will draw from our own SMS polling system. The SMS polling data will be integrated with the system when an emergency is triggered by the user. The product will be a web-application format.

1.5.2 User interfaces

The user interfaces for the product will include a list of buildings on campus, a search function to look up buildings, a display of two numbers for the estimation of people currently in a building based on two sources (Wi-Fi monitoring and the occupancy estimation), analytics showing overall trends in the person counts over an academic quarter, and a toggle to switch the web-app to emergency mode and back to regular mode. Once in emergency mode, the display will also include SMS polling data as a third source. The product will also include a map interface that facilitates clicking on visualizations of buildings instead of using the search function or list view. There will also be an admin portal that any superuser can navigate to that can be used to regulate the databases and the users.

1.5.3 Hardware interfaces

This product will be hosted on a secure server and will be accessible from any web-browser. The server will be configured to handle a high volume of requests and will be protected by a robust security system. Additionally, existing APs and Wi-Fi monitors will be needed.

1.5.4 Software interfaces

This product is built for Windows. Verified to run on Windows 10 education version 21H2. Containers will also be used to run a web application, Linux, and Python, and the software will utilize either an Arduino or a tool provided by Central Washington University to generate query files.

1.5.5 Memory

This product temporarily uses memory and a central processing unit (CPU) to calculate and modify the occupancy estimation and analytics. It will create storage for a query file of probe requests from an AP. Queries will be deleted after use and the estimation will be overwritten with every new estimation every half hour in regular mode and every five minutes in emergency mode.

1.5.6 Operations

This product has periods of interactive operations when accessing information for a specific building and when activating or deactivating the emergency mode. This product will operate in two different modes:

Standard Mode, where the person count per building is determined by Wi-Fi data and the occupancy estimation once every half hour.

Emergency Mode, which will be initiated manually by the user. In this mode, the data update frequency increases to every five minutes, data is sent to a backup system, and SMS location data displays as Safe or Unsafe dots on the map.

Also, when in either mode, clicking or tapping on a building card will take the user to a page with the current counts and information for that building, as well as a graph that shows trends in the data over an academic quarter.

1.6 Product functions

This product will perform mathematical calculations to create an estimation of the number of people in a building on campus every half hour. It will create query files of unique probe requests an AP is receiving. There will be an emergency mode which queries and gets a new number every five minutes. It will display this information in both the list of buildings and overlaid on the interactive map of the CWU campus.

1.7 User characteristics

Users are intended to be either emergency management specialists or high-ranking individuals in the campus police department who coordinate with emergency services during an emergency event.

1.8 Constraints

- Have a range of accuracy for the number of people in a building, and the client prefers that the accuracy margin be as small as possible.
- Needs a tool to create the queries of probe requests.
- New or existing APs are required.
- Must be secure from unauthorized access due to the nature of the information.
- Relies on data from RaveAlert.
- Development has a ten week timeframe.

1.9 Assumptions and dependencies

There is the assumption that this product will run hardware that supports Windows OS with a query tool installed and existing APs to query.

2. Problem Statement

The Central Washington University (CWU) campus requires an efficient system to display the current count of people within any building in case of an emergency. The Emergency Management Coordinator wants a web application that will provide real-time information about the number of people present in a particular building to help emergency services respond effectively.

3. Legal Considerations & Ethics

3.1 Central Washington University Policies and Procedures

3.1.1 CWUP 2-40-010 Acceptable & Ethical Use of University Information Technology

Section 5.4: Using Central Washington University information technology resources to attempt to break into, gain root access, probe, disrupt, or obstruct any system is not permissible. Installation of invasive software or testing security flaws without authorization on any system is not permissible.

3.2 WASHINGTON CYBERCRIME ACT

3.2.1 RCW [9A.90.040](#) Computer trespass in the first degree.

(1) A person is guilty of computer trespass in the first degree if the person, without authorization, intentionally gains access to a computer system or electronic database of another; and

(a) The access is made with the intent to commit another crime in violation of a state law not included in this chapter; or

(b) The violation involves a computer or database maintained by a government agency.

(2) Computer trespass in the first degree is a class C felony.

3.2.2 RCW [9A.90.050](#) Computer trespass in the second degree.

(1) A person is guilty of computer trespass in the second degree if the person, without authorization, intentionally gains access to a computer system or electronic database of another under circumstances not constituting the offense in the first degree.

(2) Computer trespass in the second degree is a gross misdemeanor.

3.2.3 RCW [9A.90.100](#) Electronic data theft.

(1) A person is guilty of electronic data theft if he or she intentionally, without authorization, and without reasonable grounds to believe that he or she has such

authorization, obtains any electronic data with the intent to:

- (a) Devise or execute any scheme to defraud, deceive, extort, or commit any other crime in violation of a state law not included in this chapter; or
 - (b) Wrongfully control, gain access to, or obtain money, property, or electronic data.
- (2) Electronic data theft is a class C felony.

3.2.4 RCW [9A.90.130](#) Cyberstalking.

- (1) A person commits the crime of cyberstalking if, without lawful authority and under circumstances not amounting to a felony attempt of another crime:
 - (a) The person knowingly and without consent:
 - (i) Installs or monitors an electronic tracking device with the intent to track the location of another person

The team for this project was given explicit permission by Information Services at CWU to run and test this software on campus.

3.3 Ethics

Due to the tracking nature of this project, we had to evaluate the ethics of the tracking that we were doing. In order to understand the ideas and values of our peers, the project was presented in concept to fellow Computer Science students in the context of getting ethical feedback. The results of the feedback were mixed, but the general consensus was that the ways that we implemented the project were ethical. However, some peers did not feel comfortable with the idea of being tracked at all. The major ethical question is whether or not the tracking that we are doing is not intrusive enough to still justify the safety aspects of the project. The safety aspect of the project, and the fact that we are not tracking the exact location of any individual made the majority of students feel comfortable with the project. Those who did not agree stated that they were uncomfortable with the tracking in general. Everyone in the group agreed that there were concerns regarding the potential for the tracking to be expanded to pinpoint locations in the future under the guise of “more safety.” The takeaway from the group was that the project is ethically acceptable in its current configuration, but future expansions should be considered individually and carefully.

4. Requirements

4.1 User interaction

4.1.1 Outputs

- A list of all buildings on campus, as well as the current number of people in any particular building and other relevant statistics like when the numbers were determined, the capacity of the building, and the overall trend data for the building for that quarter.
- A digital map that enables the user to select buildings and display the previously stated statistics.

4.1.2 Inputs

- Search Function – A search field to look up specific buildings
- Emergency Mode Switch – A toggle switch that when pressed will switch the system to a new query interval of five minutes and will also cause the system to retain current data and display SMS location information in the map. The switch can be pressed again to transfer back to the normal query interval of half an hour.
- A form to add new buildings and APs to the list of queried buildings
- A form to upload Registrar data
- A form to upload RaveAlert data
- Mouse/Touch - Interacting with the map using the mouse for desktop or laptop and touch for mobile will allow for building specific data to be shown and displayed to the user

4.2 General Requirements

The items below are the general requirements for our software and are listed in order of priority. General requirements refers to the big picture items that the client requested.

- Ability to view the number of people inside a building at Central Washington University
- Ability to access building data from any computer connected to the internet, including mobile devices
- Ability to initiate an emergency mode which increases the frequency and accuracy of data collection
- Ability to view a map of Central Washington University with data
- Ability to add or remove possible buildings from the data and map view
- Ability to view advanced data trends from a list format of all Central Washington University buildings

4.3 Functional Requirements

The items below are specifically the functional requirements, or those requirements that specifically dictate how the software will operate. They are listed in order of priority.

- Calculate the number of people inside a building with +/- 5 person accuracy
- Authorized personnel should be able to access software with verified credentials
- Handle incorrect query files, unknown inputs, and calculation accuracy failures
- Create a modular program that allows for the future integration of different deterministic methods

4.3.1 Exact sequence of operations

- Run script to collect a query file every half hour for one minute from APs in specific buildings currently listed in the program
- Filter files to only include unique probe request from phones
- Take that number and set it as the current estimation for people in the building of the corresponding AP.
- If the Emergency Mode is activated, switch the query interval to every five minutes and add SMS polling data to the map display.
- Display current occupancy estimation and building statistics for corresponding buildings.

4.3.2 Responses to abnormal situations

This system shall have error handling for incorrect query files, unknown inputs, and calculation accuracy failures. There will also be a pop-up that appears whenever the user attempts to turn Emergency Mode either on or off. The user will just need to click "OK" to continue if they did mean to turn Emergency Mode either on or off, and "Cancel" if they clicked or tapped the switch by accident.

4.3.3 Relationship of outputs to inputs

This system shall use queries to create the calculation of people in a specific building and then display them to the user in both list form and map form.

4.4 Non-Functional Requirements

The items below are the non-functional requirements and are listed in order of priority.

- Program must be secure from unauthorized access
- Ability to interact with a map of Central Washington University to see building data
- Program can be accessed from any internet connected web browser

4.5 Feasibility Analysis

4.5.1 Financial Feasibility

This software will be developed with free Integrated Development Environments (IDEs), programming languages, databases, and other tools. This software is freeware and will follow those standards. No cost will be charged to customers. These conditions make the software financially feasible.

4.5.2 Technical Feasibility

The technologies and tools associated with the software are Linux, Microsoft Word, Windows 10 Education, and Arduino micro controllers. The tools used to build the software will be React, Django, MariaDB, OpenStreetMaps with Leaflet, and VisualStudio Code. These tools are all easily and freely accessible to students and the development team. They are also all tools that the team members either have prior experience with or can learn fairly quickly. These conditions make the software technically feasible.

4.5.3 Resource and Time Feasibility

Resources that are required are a programming device (desktop), programming tools, and developers. All these things are already available to use for the software currently. These conditions make the software resource feasible.

4.5.4 Risk Feasibility

Risk associated with size: The software itself will take up a small amount of space on the memory of the computer and will temporarily store query files.

Projected changes to the requirements before and after delivery: The software utilizes known methods for tracking phones and will include a form to add new buildings. A more integrated system could be implemented later and capabilities to enable location services or Wi-Fi services on phones during an emergency could also be included. The method for tracking the phones if the accuracy is too low or if the tools/APs are unavailable can change too.

Customer related risk: The software is specific to the customer and organization. Privacy of data is considered. Query files are deleted after use so no data from people's phones is saved. The software does depend on the network security of the organization.

Accuracy: The client requested an ideal accuracy of +/- 5 people within a building. Due to limitations in the deterministic methods, an accuracy of +/- 5 people is impossible. In addition, regardless of the technology or method used, there are benefits and issues with every method. First, two of our methods rely on tracking devices within a building. This means that counts can be potentially inflated with people using multiple devices. Also, in an emergency, not everyone grabs their device/cell phones. This could lead to artificial flags for the number of people in a

building. Finally, if an individual has no trackable devices, they would not be included in the count at all. The third method relies on data from the registrar and floor plan, as well as assumptions about people's behaviors during the day. These assumptions will not be accurate all the time, and this method will therefore have a wide range for accuracy. Since we know the method will be fairly inaccurate, we informed the client of this and they requested that the methods err on the upper end, so we have adjusted this third method to be an overestimation.

4.5.5 Development environment risks

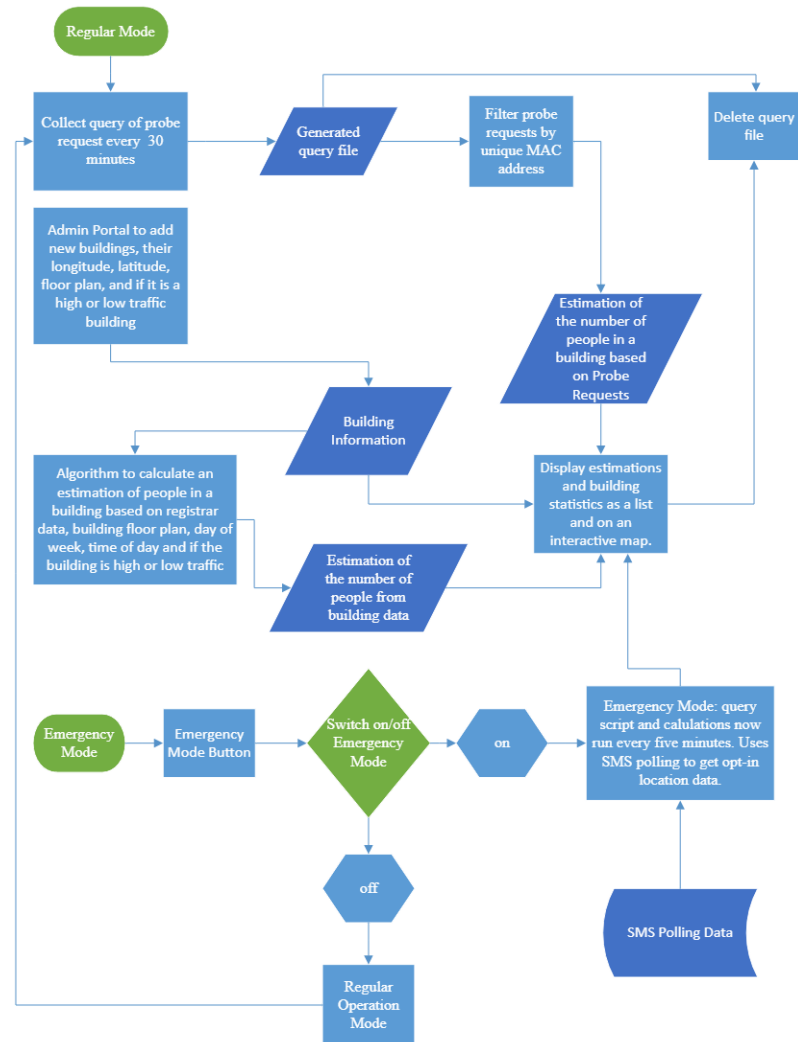
The tools necessary to make the software are readily available. The technology used for the software is well established and the implementation methods are not new. These conditions make the software risk feasible.

5. Design

5.1 High Level Design

5.1.1 System Overview

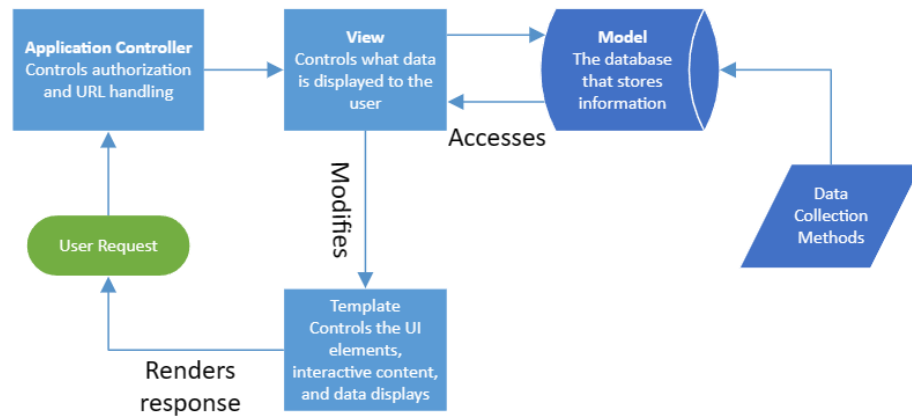
This product's main function is to create an estimation of the number of people in a specific building before and during an emergency. The software will gather probe request queries from listed buildings and then use the number of requests to determine an estimation of the number of people in the corresponding building. The query will run every thirty minutes and be deleted after use. The estimation will be displayed alongside building statistics relevant to emergency situations like capacity and number of exits. There will be a switch for Emergency Mode which will increase the query interval to every five minutes and send SMS polls to CWU Rave users. There will be an admin portal to add new buildings to those queried and an occupancy estimation which will be displayed alongside the probe request data.



5.2 Low Level Design

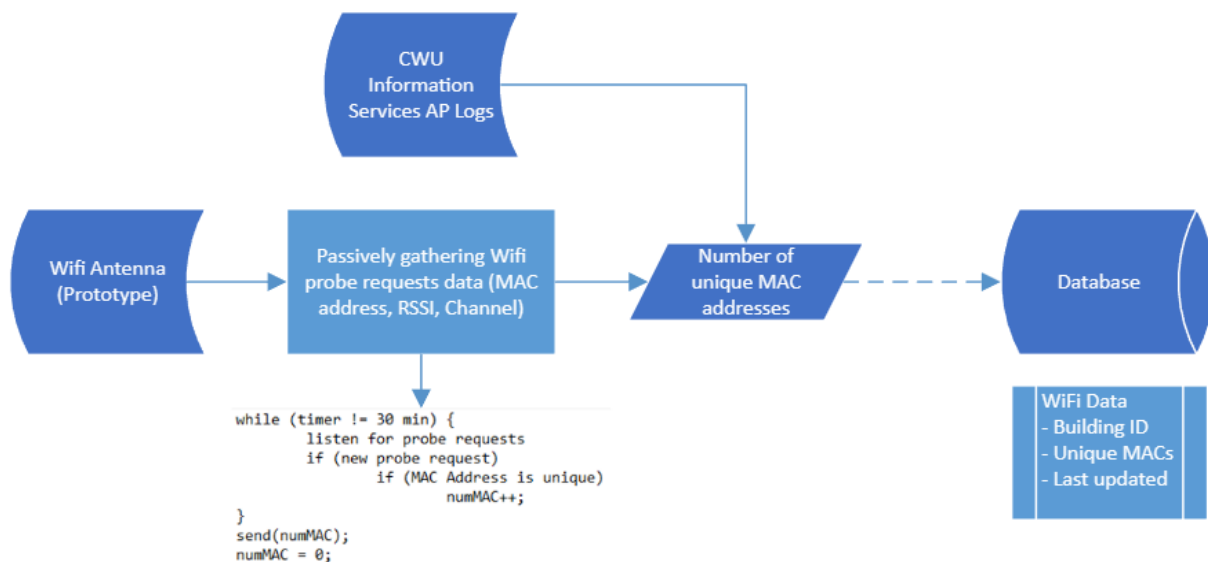
5.2.1 Application Structure

The Emergency Management Software uses an architectural layout known as a “Model-Template-View” layout. It consists of a model module which is the underlying database to the project, a view module which controls what data is displayed, a template module that acts as a blueprint for how the data is displayed, and an application controller module which handles user authentication and URL handling. The figure below shows how a “Model-Template-View” layout is structured.



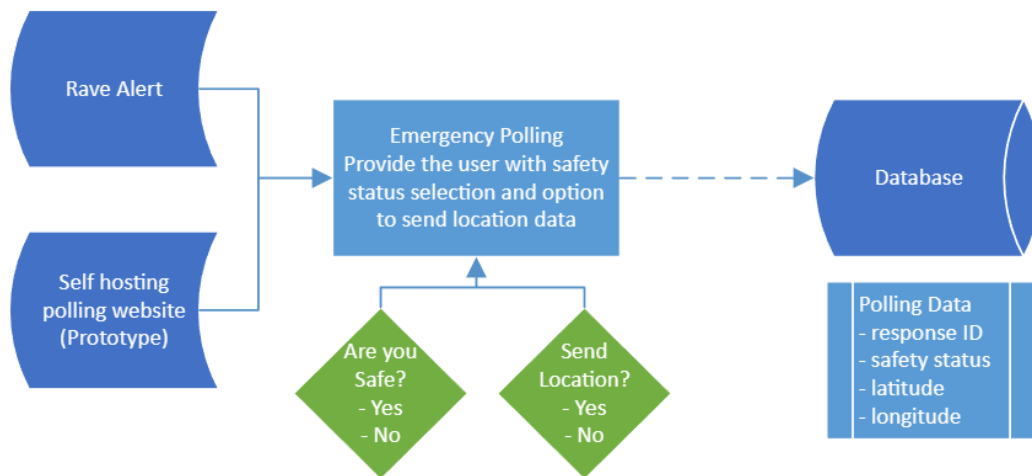
5.2.2 Module 1: Wi-Fi Data

The Wi-Fi data module collects Wi-Fi probe request data and stores the number of unique devices in the database. The two input methods for this module are the access point logs from the network infrastructure services at the organization this software is deployed at (in this case, CWU Information Services), or a Wi-Fi antenna. These input methods are used to determine the number of unique MAC addresses in a given building, and then send the count to the database. Below is an example of how this module works.



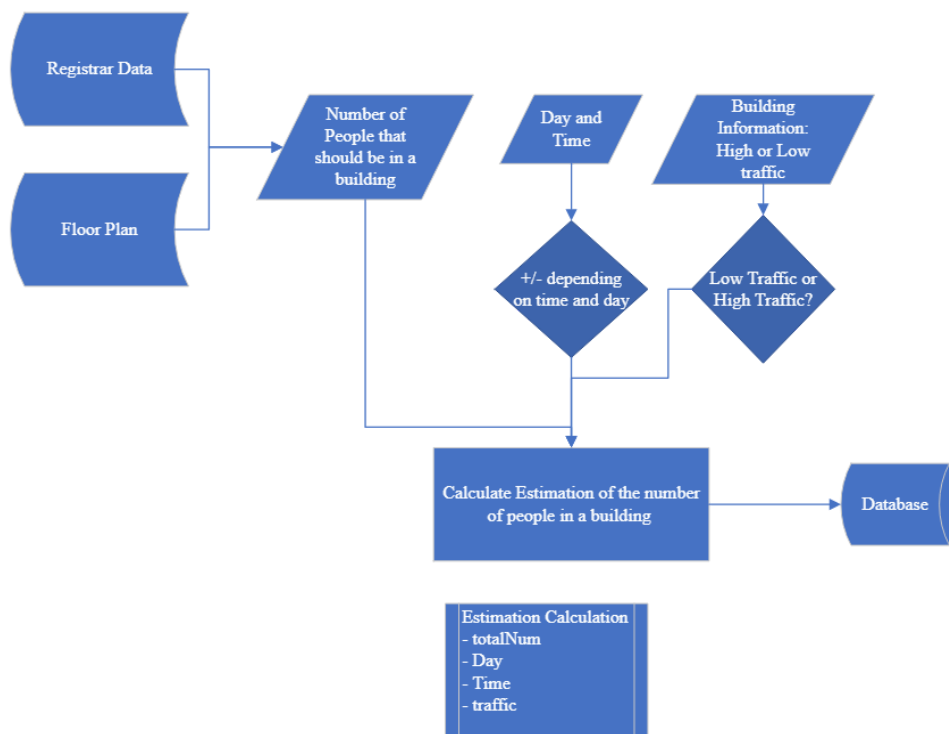
5.2.3 Module 2: SMS Polling Data

The SMS polling data module is responsible for gathering poll results from users during emergency mode operation. The two inputs of this module is the Rave Alert backend data and our own polling website. The poll is designed to ask the user questions about their safety status and gather their location if they choose to opt in. Below is an example of the SMS module.



5.2.4 Module 3: Occupancy Estimation

Our final data collection module is in charge of estimating building occupancy. The two data inputs for this module are the registrar data, or data of how many people should be in a building, as well as the floor plan data for each building. This data is used, along with other factors like the time of day, day of the week, and whether or not the building is high or low traffic, to calculate an estimation of the number of people in a building at a given time. This method is the most inaccurate, and as mentioned in [Risk Feasibility](#), is mainly used as an upper bound. Below is an example of the structure of this module



6. Software Tools Requirements

- Recommended
 - To successfully deploy this software at scale, these are the necessary software tools
 - Python 3.11 or higher
 - MariaDB
 - Web server (Apache or nginx)
 - Windows, Mac OS X, Linux
- Minimum requirements
 - This software can be run with the development server and development database, in which case these are the only necessary requirements
 - Python 3.6 or higher
 - Windows, Mac OS X, Linux

7. Hardware Requirements

- Recommended:
 - x86 multi-core CPU
 - 4 GB of ram
 - 30 GB Storage*
- Minimum requirements
 - Any multi-core CPU
 - 2 GB of ram
 - 10 GB Storage*

* Storage is dependent on data input and storage methods from the user. This amount may vary based on the deployment of the product.

8. Software Documentation

8.1 Wi-Fi Passive Listening

The main data collection method for this software is Wi-Fi device data from passively gathered probe requests. This software implements this collection method by using a small Wi-Fi antenna built into a micro controller, however it is designed to be connected to a much more robust network infrastructure of the organization this software is deployed at. The main algorithm of this module is listening to incoming probe requests, and storing the number of unique MAC addresses over a certain time period. The pseudo code for this module can be seen below.

```

while (timer != 30 min) {
    listen for probe requests
    if (new probe request)
        if (MAC Address is unique)
            numMAC++;
}
send(numMAC);
numMAC = 0;

```

8.2 Occupancy Estimation

The software needed a data collection method that could be compared to the other data collection methods. The occupancy estimation accomplishes this by taking registrar data and a number derived from the floor plan to create the number of people that should be in the building at a specific time. The two inputs are pulled from the Building database. The registrar data specifically uses a function in the Building model class to filter the RegData database and return the count for a specific building at the current time and day. Then it gets the time, day, and if the building is low or high traffic. Those parameters return a percentage of the total number to be added into a final estimation; for example, Friday is -50% and 8am on any weekday is -10%. Once the final estimation is created, it is uploaded to the Buildings database. This whole process runs in a background thread every hour. The algorithm was tested with snapshot data provided by the CWU Registrar. This file contained 100 entries for one building on campus. It was tested against null, zero, and negative input values. This would be considered white box testing.

8.3 Rave Alert Data Upload

Due to limited constraints regarding access to the backend Rave Alert systems, the only way to integrate data from a Rave Alert poll result is to upload a .csv file of the data. The user downloads the data from the Rave Alert portal. Due to not having access to the Rave Alert system, we cannot provide any information for how to access and retrieve this data. However, once the data file is retrieved, the user can upload this file to our software. We created a form, view, and template button to control the upload of the Rave Alert data to the EmergencyModePollResult database. From the Rave Alert .csv file, five fields are extracted for each received poll response:

1. Site UID (Unique Identification)
2. Answer Latitude
3. Answer Longitude
4. Answer
5. Answer Received

These five fields are directly imported to the EmergencyModePollResult database table under the five fields from the modal:

1. Site UID -> id
2. Answer Latitude -> lat

3. Answer Longitude -> long
4. Answer -> status
5. Answer Received -> created_at

With the data added to the database, every response can be added to the map as an individual point if the Emergency Mode is enabled.

8.4 Registrar Data Upload

The user needed a way to give the software registrar data for the occupancy estimation. We created a form, view, and template button to control the upload of registrar data to the RegData database with an excel file. The important functionality happens in view.py where the code to handle the file upload checks to make sure it is an excel file with an explicit format.

```
def process_data(f):
    workbook = openpyxl.load_workbook(f)
    worksheet = workbook.active
    for row in worksheet.iter_rows(min_row=2, values_only=True):
        reg_data = RegData.objects.update_or_create(
            name=row[0],
            date=row[1],
            start_time=row[2],
            count=row[3]
        )
```

The form and upload code is made with django standards. To process the data we made an algorithm that can open an excel workbook and then it reads the fields in the excel sheet and sets them to the corresponding fields in the RegData database. This was tested with invalid file formats like .csv and files with more fields than what the database accepts. This would be considered white box testing.

8.5 SMS Polling

This software provides its own SMS alert webpage that can be sent to individuals on campus in order to provide a more integrated SMS polling system that doesn't rely on external systems. The user can select between "Safe", "Unsafe", or "Not sure", and can also select if they want to send their location. The location that is sent is based on the geolocation of the device that the web browser is running on, and is strictly opt-in, meaning the user must select that they want to send location, and must have location services turned on. This is the user interface for the alert webpage.

CWU Safety Status

Status:

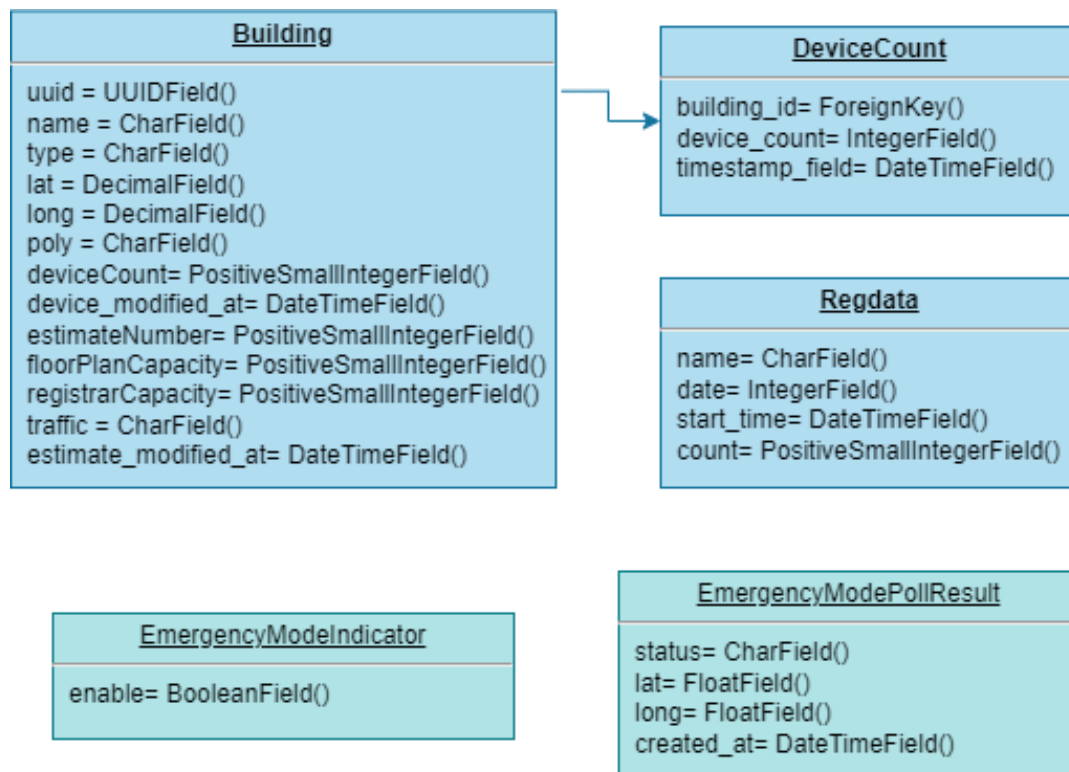
- ☐ Safe
☐ Unsafe
☐ Not sure

☐ Send location

Submit

This web page should only be accessible from a specific domain or subdomain and does not have any UI connection to the main software. The page will display a confirmation message if the poll response has been successfully sent to the database.

8.6 Database Structure



8.6.1 Building

The purpose of Building database is to store information about each building's information, including:

- Building ID with uuid, name and type.
- Location and shape of the building with latitude, longitude, and polygon shape of the building
- Current Wifi passive listening information with deviceCount and device_modified_at
- Current Occupancy Estimation information with estimateNumber, floorPlanCapacity, registrarCapacity, traffic, and estimate_modified_at

8.6.2 DeviceCount

The purpose of DeviceCount is to store the count and timestamp of WiFi passive listening devices in a building. This data is then used to display trend data by filtering it by the day, hour, and building uuid, and then computing the average. The average data is then displayed to the user on the trend data page for each hour of the selected day and building. After testing 720 different entries with random values based on the hour and day in this table, we concluded that the trend data average is accurate.

8.6.3 RegData

The purpose of RegData is to store occupancy estimation calculation for a timestamp. Similar to device count, it stores the occupancy estimation calculation and the timestamp, and then filters the database to display the calculation to the user in the trend data page.

8.6.4 EmergencyModeIndicator

The purpose of EmergencyModeIndicator is to indicate whether emergency mode is on or not. This information is passed into the webpage then changes the mode of the website to emergency mode, therefore changing functions of the webpage. This database helps sync all other users to the same mode, not just locally.

8.6.5 EmergencyModePollResult

The purpose of EmergencyModePollResult is to store the SMS poll results including the status, timestamp and location. This data is then sent to the map interface. Then the data is used to display the location and status of the victims onto the map.

9. Installation

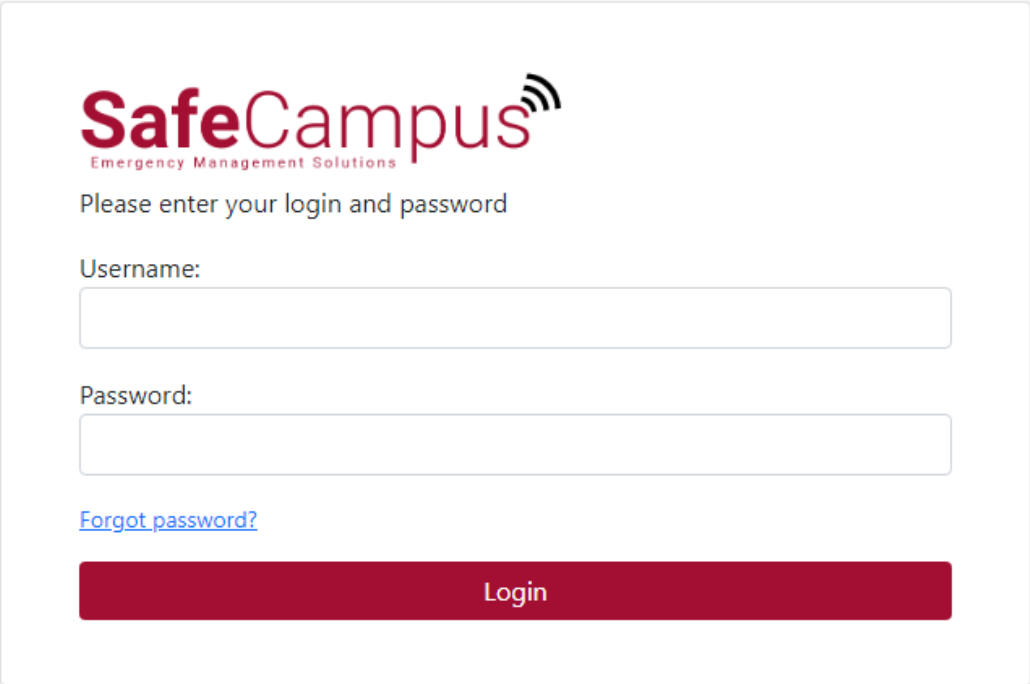
1. Download the source code from <https://github.com/CWU-Emergency-Management-Project/Emergency-Management>
2. Install python dependencies from requirements.txt file or by using pip
 - a. django
 - b. djangorestframework
 - c. Django-cors-headers
 - d. pandas
 - e. requests

- f. `openpyxl`
3. Navigate to the folder with the `manage.py` file in the command prompt
 - a. create a superuser with `python manage.py createsuperuser` and follow the steps to create a new admin user
 - b. Start the development server with `python manage.py runserver` and navigate to `127.0.0.1:8000` in a web browser
4. For more detailed installation instructions, see Django installation documentation <https://docs.djangoproject.com/en/4.1/topics/install/>
5. If you plan to deploy this software with an external database, you need to install MariaDB on the system
 - a. Download the correct version of MariaDB from <https://mariadb.org/download/?t=mariadb>
 - b. Modify `settings.py` file to connect to your MariaDB server
 - c. Use `python manage.py migrate` to transfer the current tables to MariaDB.

10. User Documentation

If you want a video demonstration of the software here is a link to a video demo of the software <https://www.youtube.com/watch?v=i5AeVMISM4I>.

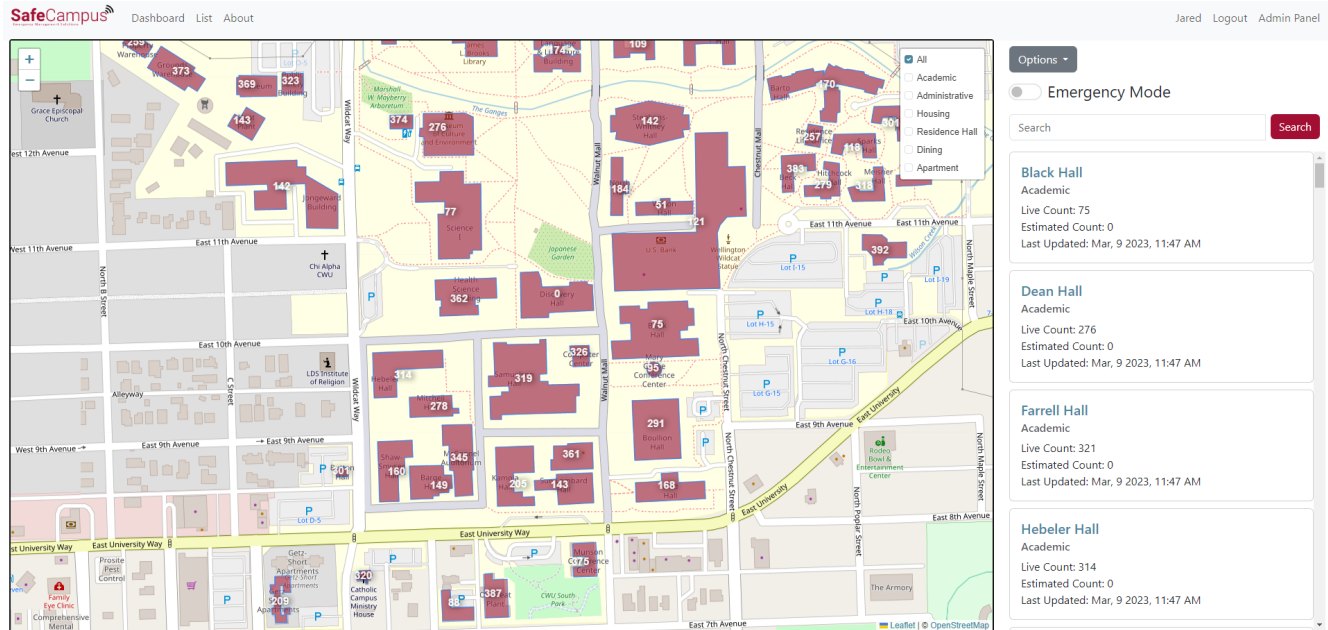
1. Open a web browser like Google Chrome and type in the website name. (Not yet hosted). There is a demo website `SafeCampus.dev` which is available until 02/23/2024. The first thing you will see is the login page.



The image shows a login page for SafeCampus. At the top is the SafeCampus logo, which includes the text "SafeCampus" in a large, bold, red font, with "Emergency Management Solutions" in a smaller, red font below it. To the right of the text is a red icon of a signal tower. Below the logo, the text "Please enter your login and password" is displayed. There are two input fields: "Username:" followed by a text box, and "Password:" followed by a text box. Below the password field is a blue link that says "Forgot password?". At the bottom is a large red button with the word "Login" in white text.

Please type in your username and password to login. If you're using the demo website, you can enter username "demoUser" and password "demoPassword". The forgot password button currently does nothing as an email server needs to be linked to it. If you do not already have an account please contact the site admin. If there is no site admin yet, you can create one if you have access to the server running the site by following the steps in the installation section.

2. Once logged in, the first thing you will see is the interactive map display.



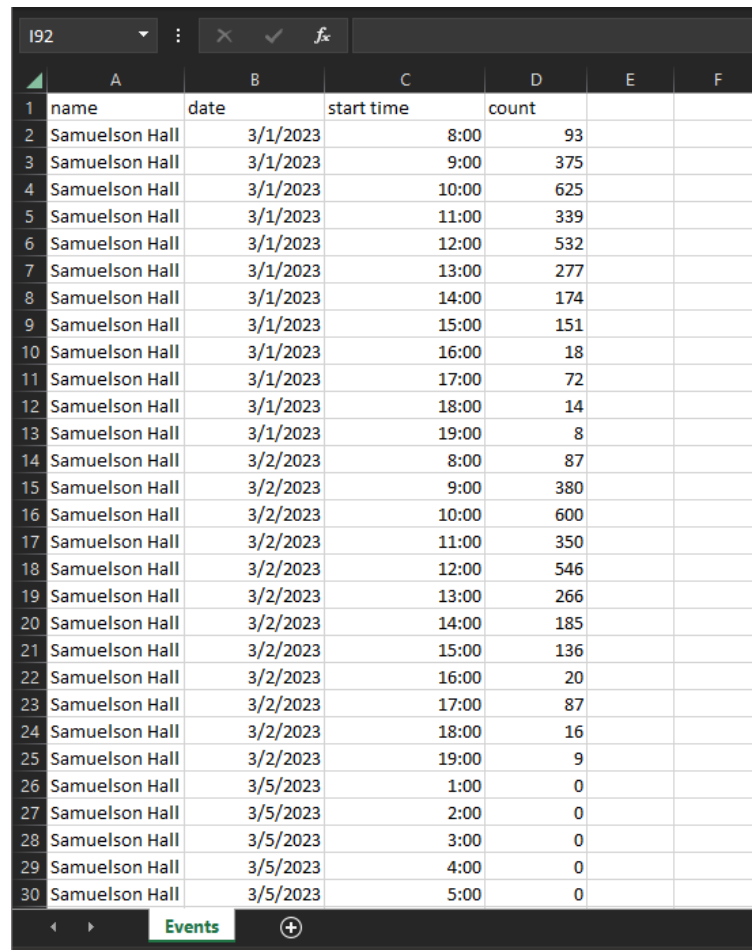
The functionality on this page is as follows:

- Use ctrl+scroll-wheel to zoom in and out.
- Click on a red building to pop up an information card.
- Click a building card on the right side of the screen to center the map on that building.
- Change what type of building is showing by selecting a filter from the list found in the upper right corner of the map.
- Use the 'Search' field on the right side of the screen to find a specific building in the card list.
- Click the 'Emergency Mode' toggle switch on the right side of the screen to activate that mode. There will be a pop-up to confirm enabling and disabling this switch.
- Click the 'Options' drop down button to access the 'Upload Rave Alert Data', 'Upload Registrar Data', and the 'Add Building' button. See part three.
- If the name of a building is clicked on, in either a pop-up card or on a card in the list, it will take you to the building's trend data page.
- In the nav-bar across the top of the screen, you can find the 'Dashboard' button which will take you to the map screen. The 'List' button takes you to a page which only contains the building card list and a search bar. The 'About' button takes you

to the about page. The 'Logout' button will end your session and send you to the logout screen. The 'Admin Panel' button will take you to the admin portal.

3. For the 'Options' button there are three different functions:

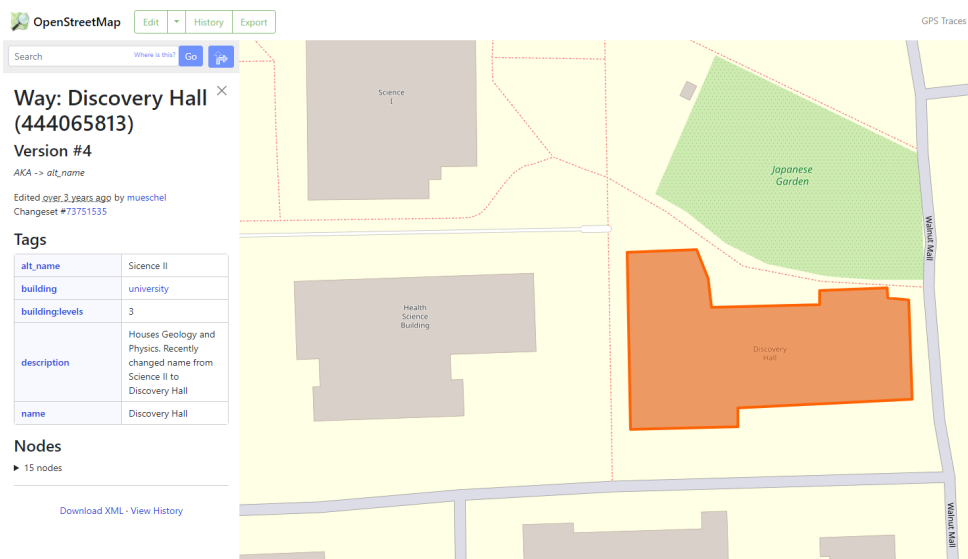
- 'Upload Rave Alert Data' will open a window where you can upload a .csv file containing the locations of those who opted in through the Rave backend. Rave needs to provide this file.
- 'Upload Registrar Data' will open a window where you can upload a .xlsx file containing the building name, day, time, and count in that order.



	A	B	C	D	E	F
	name	date	start time	count		
2	Samuelson Hall	3/1/2023	8:00	93		
3	Samuelson Hall	3/1/2023	9:00	375		
4	Samuelson Hall	3/1/2023	10:00	625		
5	Samuelson Hall	3/1/2023	11:00	339		
6	Samuelson Hall	3/1/2023	12:00	532		
7	Samuelson Hall	3/1/2023	13:00	277		
8	Samuelson Hall	3/1/2023	14:00	174		
9	Samuelson Hall	3/1/2023	15:00	151		
10	Samuelson Hall	3/1/2023	16:00	18		
11	Samuelson Hall	3/1/2023	17:00	72		
12	Samuelson Hall	3/1/2023	18:00	14		
13	Samuelson Hall	3/1/2023	19:00	8		
14	Samuelson Hall	3/2/2023	8:00	87		
15	Samuelson Hall	3/2/2023	9:00	380		
16	Samuelson Hall	3/2/2023	10:00	600		
17	Samuelson Hall	3/2/2023	11:00	350		
18	Samuelson Hall	3/2/2023	12:00	546		
19	Samuelson Hall	3/2/2023	13:00	266		
20	Samuelson Hall	3/2/2023	14:00	185		
21	Samuelson Hall	3/2/2023	15:00	136		
22	Samuelson Hall	3/2/2023	16:00	20		
23	Samuelson Hall	3/2/2023	17:00	87		
24	Samuelson Hall	3/2/2023	18:00	16		
25	Samuelson Hall	3/2/2023	19:00	9		
26	Samuelson Hall	3/5/2023	1:00	0		
27	Samuelson Hall	3/5/2023	2:00	0		
28	Samuelson Hall	3/5/2023	3:00	0		
29	Samuelson Hall	3/5/2023	4:00	0		
30	Samuelson Hall	3/5/2023	5:00	0		

The format shown is explicitly required.

- ‘Add Building’ opens a window where you can enter the Way ID from Open Street Maps for a building on campus.



Once the Way ID is added and the page has been refreshed, the new building should have a red outline on the map.

4. The ‘List’ page is where you will find the list of building information cards separate from the map. The ‘Search’ field and building name links on this page function the same way the same features discussed in part two function.
5. The ‘About’ page is where the information on the project will be displayed.
6. The ‘Admin Panel’ page is where the site admin can manage the users and databases connected to the site.

SafeCampus Admin Dashboard

Site administration

AUTHENTICATION AND AUTHORIZATION	
Groups	+ Add Change
Users	+ Add Change
WEBAPP	
Buildings	+ Add Change
Emergency mode indicators	+ Add Change
Emergency mode poll results	+ Add Change
Reg datas	+ Add Change

Recent actions

My actions

- [Samuelson Hall](#)
Building
- [+ JaredTest](#)
User

The site admin can create and manage users and assign them to specific permissions groups if needed. The main database to manage is the Buildings database. Select ‘Buildings’ from the list under ‘WEBAPP’ and you will see a list of all the buildings

currently in the database. From here you can search and filter the list. You can select entries to delete. You can click an entry to open a page with editable fields.

The screenshot shows the 'SafeCampus Admin Dashboard' with a breadcrumb trail: Home > Webapp > Buildings > Samuelson Hall. On the left is a sidebar menu with sections: 'AUTHENTICATION AND AUTHORIZATION' (containing 'Groups' and 'Users', each with a '+ Add' link) and 'WEBAPP' (containing 'Buildings' (highlighted in yellow with a '+ Add' link), 'Emergency mode indicators' (+ Add), 'Emergency mode poll results' (+ Add), and 'Reg datas' (+ Add)). The main content area is titled 'Change building' and shows details for 'Samuelson Hall'. The form includes fields for: Name (Samuelson Hall), Type (Academic dropdown), Traffic (Low dropdown), FloorPlanCapacity (150), RegistrarCapacity (empty), Lat (47.0014192704), Long (-120.5406343296), and Poly ([[47.0018015, -120.5410974], [47.0014545, -120.5406343]]). A red 'Delete' button is at the bottom.

On this screen you can set the building type, pass through traffic, and floor plan capacity (This is a manually set number derived from a building's floor plan, where the number of offices and labs is counted to account for people not attending classes; this number should almost always be lower than the registrar data). Registrar Capacity is pulled from a different unseen database which is filled by the 'Upload Registrar Data' button. Lat, Long, and Poly all control how the building is being displayed on the map and are all set with the 'Add Building' button in part three. To return to the active site, there is a 'View Site' button in the upper right corner of the screen.

11. Conclusion

11.1 What we have seen in the document

This document contained 10 main sections: Introduction, Problem Statement, Legal Considerations and Ethics, Requirements, Design, Software Tools Requirements, Hardware Requirements, Software Documentation, Installation, and User Documentation.

11.1.1 Introduction Summary

The Introduction section looked at a general overview and scope of this project and this specific documentation. The goal of this documentation is to serve as the culminating documentation from the project. This section also included important definitions and references.

11.1.2 Problem Statement Summary

The Problem Statement section covered how we needed to create a web-based application that can give an estimation on the number of people in a building at a specific time before and during an emergency.

11.1.3 Legal Considerations and Ethics Summary

The Legal Considerations and Ethics section is where we looked at the laws of Washington state and the policies of Central Washington University to direct the overall design and privacy policy we implemented for the software. We addressed the ethical issues that naturally accompany tracking software and made sure to protect users and those being counted.

11.1.4 Requirements Summary

In the Requirements section, we outlined all of the requirements of the project that were extracted from the requests of the client. These requirements were broken into functional, non-functional, and general requirements to help differentiate them from each other. We also discussed the feasibility of different aspects of the project.

11.1.5 Design Summary

The Design section contained the high and low level designs of the project and its data collection methods.

11.1.6 Software Tools Requirements Summary

The Software Tools Requirements section covered the dependencies, packages, and language that are required in order to run and operate the project. We included two options, with one option being for at scale deployment (which requires more dependencies) and a local deployment that requires less dependencies.

11.1.7 Hardware Requirements Summary

The Hardware Requirements section covered the minimum and recommended hardware requirements in order for the system to run, including details of the CPU, ram, and storage requirements.

11.1.8 Software Documentation Summary

The Software Documentation section covered how the code is organized into its different modules, functions, parameters, and specific algorithms that the project uses. This section includes screenshots from the actual code and project to illustrate the modules to the reader.

11.1.9 Installation Summary

The Installation section contained a step by step guide on installing the software and setting up a super user

11.1.10 User Documentation Summary

The User Documentation section walked a new user through the different pages of the site and their features.

11.2 What have we learned?

Due to the limitations of both the timeline and lack of possibility of system integration with Central Washington University systems, this project only serves as a proof of concept software for the data collection methods that we prototyped. Despite just being a proof of concept, we still learned a lot as a group. One major thing that we learned was the legality regarding tracking systems. The major takeaway was that we were within legality so long as we had explicit permission from the network owner and/or administrators, who was the CWU Information Services department in our case. While learning about the legality of our project and of tracking in general, we also looked into the possible methods for collecting the data that we need. We learned that there are many different versions and methods of tracking, with varying levels of accuracy. We learned that there was a way to produce 100% accuracy in the tracking, but it required a closed campus with issued tracking fobs for every person on campus. This would cost a lot of money and require CWU to operate in a method that is not intuitive for campus life. From this research, we also learned about SMS polling, which we integrated into the project and took the back-end data from Rave Data for.

In addition to the conceptual ideas for legality and tracking, we learned a lot about building, developing, and deploying web applications. Using the Django framework took a lot of difficulty out of understanding and creating a web application due to the built-in functionality that comes with it. The Django framework did require learning the framework and intricacies of setting up the aspects of the framework. We also had to learn and understand how to use the Leaflet system in order to deploy it in the project.

11.3 What would we do differently?

There are several aspects of this project that we would do differently if we were to complete this project all over again. Since this information would be important for anyone wanting to continue this project or possibly create something similar, we have included those items here.

If we had more time for development during this project, we would have been able to work on other features that we wanted to include or planned on including, such as a way to handle events on campus, like commencement. Commencement is just one event of many that happen throughout the year, but it is one of the largest, with most of the graduating students and their families and friends attending. Our software currently would only reflect that event if there was an Access Point nearby, and that would not ensure an accurate estimate of the number of people present. We could have gotten more time by starting on development sooner - earlier on, we made the choice to concentrate on the SRS and the midterm presentation before starting development, and if we were to do this project again, we would start working on development earlier.

The other main aspect we would do differently would be our organization. While we did have a Discord server that we used to communicate and a GitHub to make collaboration easier, we could have organized both better and we should have also created a Trello board to better keep track of what we were working on, who was working on it, and what still needed to happen, as well as any bugs that were present in the software. We should have organized the Discord better by creating more channels so that discussion about the same issues was all in the same place and more easily accessible. As for GitHub, when we created the code repository and first started using it, we did not create separate branches for us to each work on. This was not an issue at the beginning, when we were all working on more separate aspects, but once we got further in, we started having major difficulties with merging changes. If we were to redo the project, we would have a separate branch for each person on the team, and we would set up pull requests so that anything going to the main branch would have to be approved by at least one other person.

In addition, the database we created worked for our purposes, but if we were given more time or we were to do the project over again, we would organize it differently so that it was set up more efficiently and in a more secure way. We would have learned Django earlier and tried to adapt to it sooner - one of our team members knew Django and was able to set it up, but we all had to use it to code the software, and we could have learned it and adapted to it sooner than we did. We also would have done more testing, and we would have done gray box testing instead of the white box testing we did do. White box testing worked, but if we had done gray box testing, with at least two of the team members who did not develop the feature doing the testing, we might have been able to discover bugs sooner and been able to more thoroughly test the software. Finally, we would actually integrate with live data - we did attempt to integrate with live data through the campus WiFi APs, but if we were to do this project over, we would put an Arduino in one of the buildings on campus to gather the data ourselves for multiple days. This live data would have enabled us to test the occupancy estimation more thoroughly.