Greenwatch v2 Requirement Specifications Document

Revision 2

Prepared by

Sharome Burton
Delton Hughes
Victor Marchesi
Calvin Walmer

February 28, 2024

Table of Contents-- 2

1. Introduction/Overview

1.1 Purpose of Document

The purpose of this document is to provide an outline of the goals, constraints, risks, and requirements of the Greenwatch greenhouse monitoring system.

1.2 Scope and Objectives

This section provides the main objectives for the greenwatch project as well as the scope of the project so that the management and development teams can be on the same page with regards to the goals of the project.

1.2.1 Main Objective

Greenwatch shall provide a system for monitoring and controlling environmental variables within the greenhouse located at Bolin Hall at Midwestern State University. The project will be considered a success once an agent has been set up by the customer (Dr. Timothy Pegg) and is actively monitoring and reporting data back to the web server. The current scope of this project is to improve the front-end of the existing project and add better functionality for the end users.

1.3 Overview of Document

This document aims to offer a comprehensive summary, focusing on identifying the target audience, outlining practical applications, and presenting diagrams and scenarios for system development. It will provide detail on both functional and non-functional requirements, address user interface design considerations, and identify potential risks. This overview encapsulates the essence of a requirements specification document, ensuring a clear roadmap for development and implementation.

2. Users

2.1 Who are the users?

The primary user of the Greenwatch system will be the customer, Dr. Timothy Pegg acting as the primary administrative user of the system. Three additional admins will have access to the system. Additional graduate students may be added later by admins, but will not necessarily be afforded admin privileges.

2.2 Use cases and use case diagrams

2.2.1 Server Setup

Because server and connectivity are in a later phase of the project (See Section 3.3) we intend to flesh this area out once we have a better understanding of how a server will be implemented. Preliminary information follows:
The server will be hosted on the cloud using DigitalOcean. The server will be paid for by the MSU biology department and will be set up primarily by the development team. Upon project completion the customer will be provided with documentation that will allow them to maintain the server when needed.

2.2.2 Greenwatch Environment Setup

The steps for the setting up Greenwatch in the Bolin greenhouse are as follows:

1. Admin clones operating system image onto agent device (Raspberry Pi 2B microcomputer) with sensors and effectors connected.
2. Admin verifies system boots up, Wi-Fi is operating, and agent software is functioning properly.
3. Admin sets up an agent device into a physical greenhouse room.
4. Admin logs into Greenwatch website as a user with admin privileges.
5. Admin creates a new room by filling the new room form on the homepage.
6. Admin creates a new agent for the room using the ‘create new agent’ button on the room page.
7. Admin downloads the agent.py file generated by the website and copies it to the designated directory on the agent device.
8. Admin runs `agent.py` file on the agent device.
9. Admin verifies that the measurements taken by the agent device are being posted to the webpage for the specified room.

2.2.3 Use cases

The intended use cases are shown in a simplified manner at the top of Figure 1 and in a detailed manner at the bottom of Figure 1.

The use cases for a user to observe as shown at the top of Figure 1:

- Observe Data (Temperature and Humidity)
- Manage user notes (notes written by the same user on the rooms)
- Login and Logout of the system

The use cases for an admin are far more extensive as noted towards the bottom of Figure 1 which also encompasses much of the same use cases for Users.

- Observe Data (Temperature and Humidity)
- Manage All notes (notes written by any user)
- Login and Logout of the system
- Manage Users (Add/delete users, etc.)
- Manage Alerts (Alerts that automatically send emails should certain thresholds on measurements are met I.e. Temperature being too high or low)
- Manage Rooms (Adding or deleting rooms within the greenhouse
- Manage Agents (Agents are responsible for automatically taking measurements and compiling them on the database)

The Use cases for room Agents are simply:

- Accessing and logging General Data into the database

Figure 1 shows the detailed access diagram of a user and admin in the Greenwatch environment. Most users at deployment will be admins.
ADD use case stuff from class today

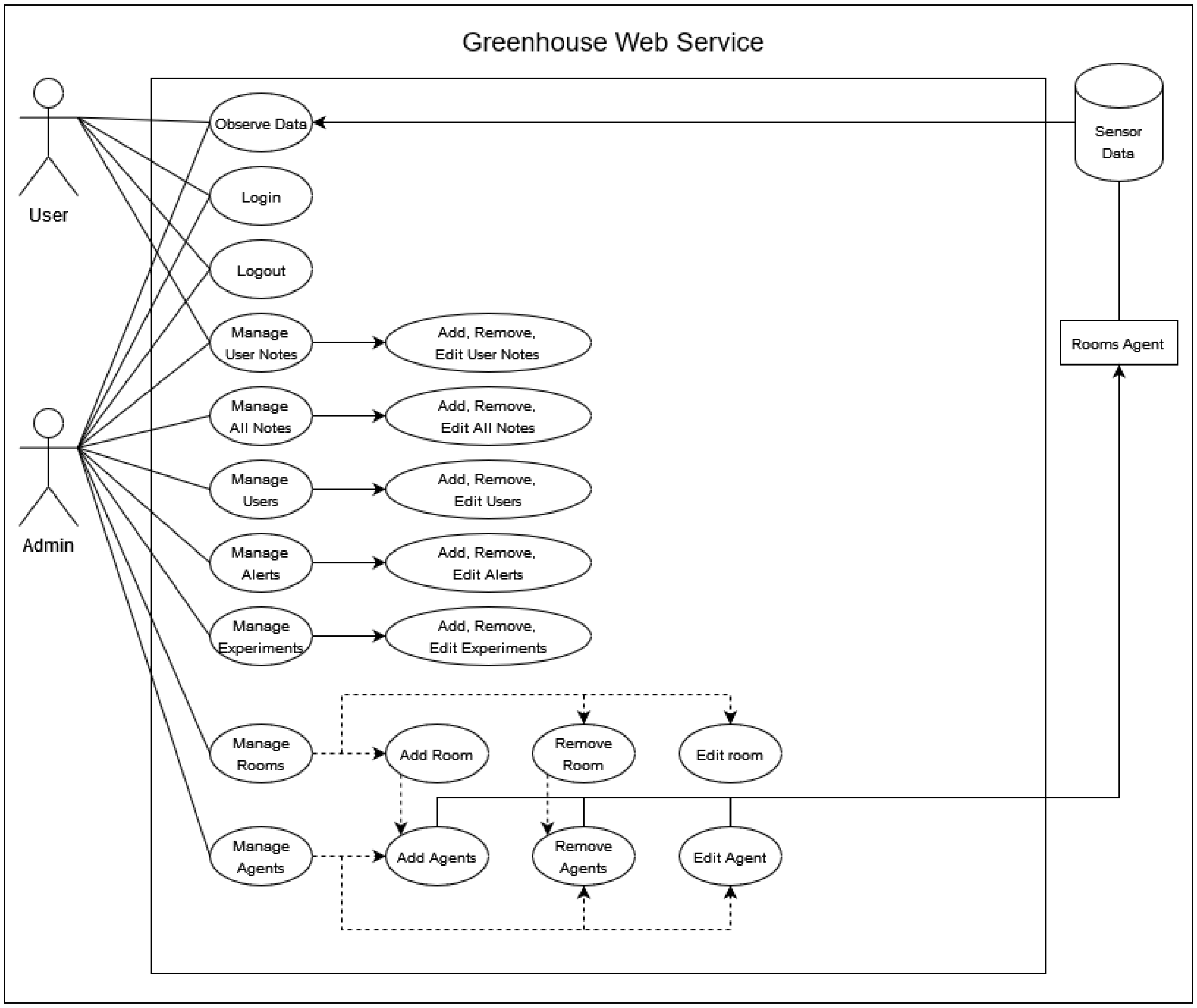

Figure 1: Use case diagram for users and admins

2.3 Scenarios

Here is a scenario that might prove the usefulness of our Greenwatch project. For example, the time of year is late Fall and the current weather is somewhat dry and starting to get somewhat cold. The user first logs into the Greenwatch website, and then pulls up the overnight data from greenhouse room1. The user then decides to export the data for further analysis in .csv format which will currently include temperature, humidity, and light intensity. Afterwards the user decides to create a new monitoring parameter to alert them when the humidity gets too low and activates the closing of the roof vents. The user then logs off of the application.

3. System

3.1 Development Environment and Tools

3.1.1 Original Project

The environment we started working with uses **Docker** for multi-platform interoperability. The back end is composed of **Python**, **Docker**, **SQL**, and **Flask**, and various other modules. The communication to the front end is done through the server hosted on **DigitalOcean**. The front end mostly consists of JavaScript, CSS, and html.

3.1.2 Current Project

The current plan is to keep docker and all of the backend modules, but to implement **react.js** to the front-end to scale down the application size. This is to provide future engineers with a more malleable development environment. We are also changing the **database** to contain less entities, and changing the code to improve the transfer of data from Raspberry Pi's to the server.

3.1.3 VScode

Visual Studio Code will be the IDE for developing the user interface section of the project. VScode allows for simple installation of extensions useful for several types of development. With built in source control tools VScode enables seamless collaboration between the frontend developers of this project. VScode is also supported on all platforms including web and mobile, giving flexibility needed to develop this project.

3.1.4 React.js/JavaScript/HTML5/CSS

With the complexity of the user interface that is planned to be developed, the web development process will contain communication with the backend and dynamic showcasing of the data being sent from the backend. JavaScript will function as the backbone of the frontend primarily within the section of the user interface which communicates with the backend. We aim to develop an aesthetically pleasing user interface that will not require updating within the near future. The combination of HTML and CSS will allow the front-end developers of this project to design and implement a professional user interface and experience for the faculty and graduate students using the Greenwatch remote monitoring system

3.2 Target Environment

There are two major target environments. The primary environment will be a dedicated server that runs the Greenwatch system. The system will store and allow access to information on users, experiments and rooms. The system will also allow users to request the agents to perform actions to alter the physical environment within the greenhouse. The server will run the Ubuntu distribution of Linux.

The secondary environment is the Raspberry Pi microcomputer that will be gathering and uploading data to the server at the desired intervals. Each Raspberry Pi will be connected to the internet and will upload their data to the server to display on the Greenwatch website. Inspection of the greenhouse on the MSU campus showed several locations that will be suitable for the agents. Near the doors of each bay there are areas that are out of direct sunlight and away from the plants. Placing the agents in these locations will minimize the effects of the environment on the agents. Additionally, there are unutilized wall outlets near these areas to power the agents.

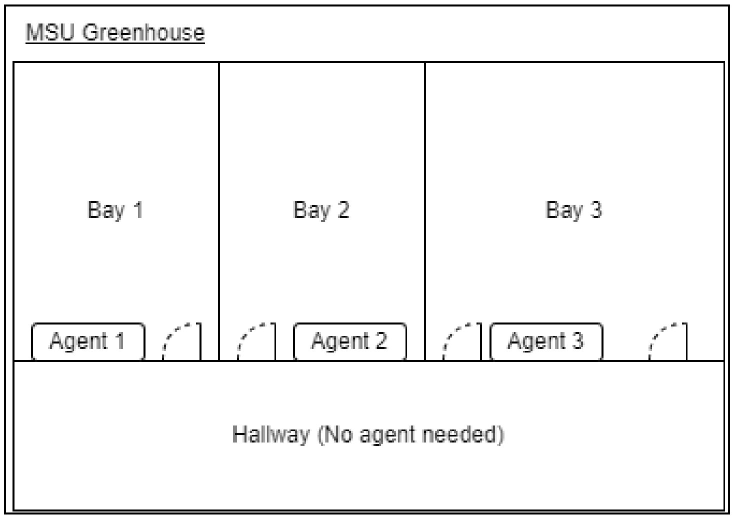

Figure 2: Rough layout of MSU greenhouse and positioning of agents

3.3 Functional Requirements

3.3.1 Primary Requirements:

- The agents must be able to gather and store temperature, humidity, light, and pressure data from the greenhouse.
- A user must be able to download data from the server for a specified time frame.
- The server must be capable of sending an alert to certain devices.
- The user must be able to set thresholds for physical environment variables in order for the server to send notifications to their devices.

3.3.2 Secondary Requirements:

- Each room must have charts to display the recorded data.
- Each room must have the ability to perform actions. Such as open/close shades and vents.

- Each user must have the ability to perform messages, each room must store a message.
- Each room must contain the ability to create/manipulate and set thresholds for an experiment.

3.3.3 Major Subsystems

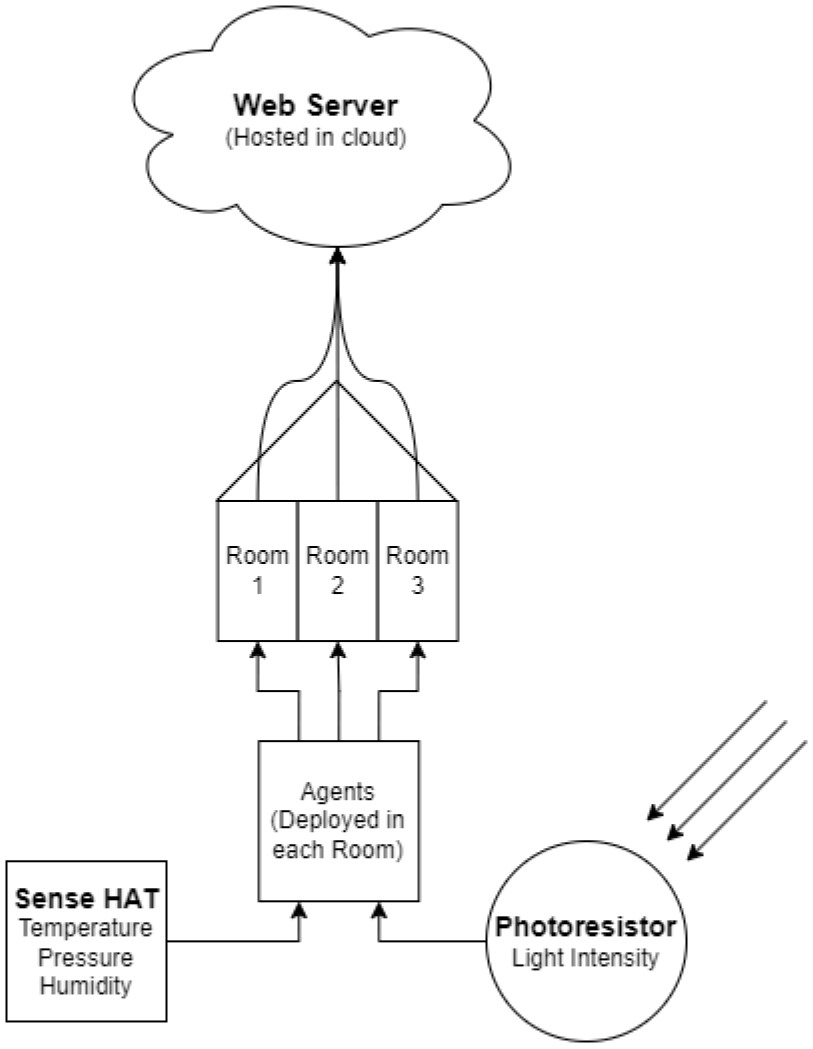

Figure 3: Major subsystems

RoomAgent will handle functionality of the application on the Raspberry PIs equipped with the **SenseHATs**. The main point of this system is to package the temperature, humidity, and brightness readings that will take place every 3 minutes and send it to the raspberry pi acting as the **RoomsServer** when a **RoomAgent** is generated by the user adding a room, it will be pre-configured based off of the room name to report back data for the room the user specified. Room names must be unique. **RoomsServer** will handle all of the requests coming from this constant influx of requests every minute or so on average from each **RoomAgent**. There will be many predefined requests/response scenarios built into this sub system all documented within a swagger-ui that they will be able to ask for in the “{server_ip}/docs” route. This will make the documentation for the backend auto generated and in a very easily understandable format.

3.3.4 Major Endpoints

Because the project will be hosted on the cloud and accessed via a web interface, major functions will be listed as routes to endpoints. Therefore, major endpoint functionally will be specified. Note that 127.0.0.1:5000 is a temporary address for development purposes and a final address will be established upon deployment.

- **127.0.0.1:5000/** Login screen of the application, this will be the first thing the user sees.
- **127.0.0.1:5000/login** Provides a user with an authentication key to use while logged in
- **127.0.0.1:5000/rooms** Allows additional rooms to be created
- **127.0.0.1:5000/home** Main homepage of the application, provides a peek into the room conditions.
- **127.0.0.1:5000/home/rooms/{room_id}** Room page that will provide a chart for past measurements as well as interface buttons to allow the creation of agent scripts. Further functionality will be added in later iterations. This is the meat and potatoes of the application.

3.3.5 Major Classes

Figures 3 and A1 show the class diagram for the system at the end of phase 1. Figure A2 shows a class diagram for the greenwatch system at the end of phase 4, by this point the application will have 8 classes.

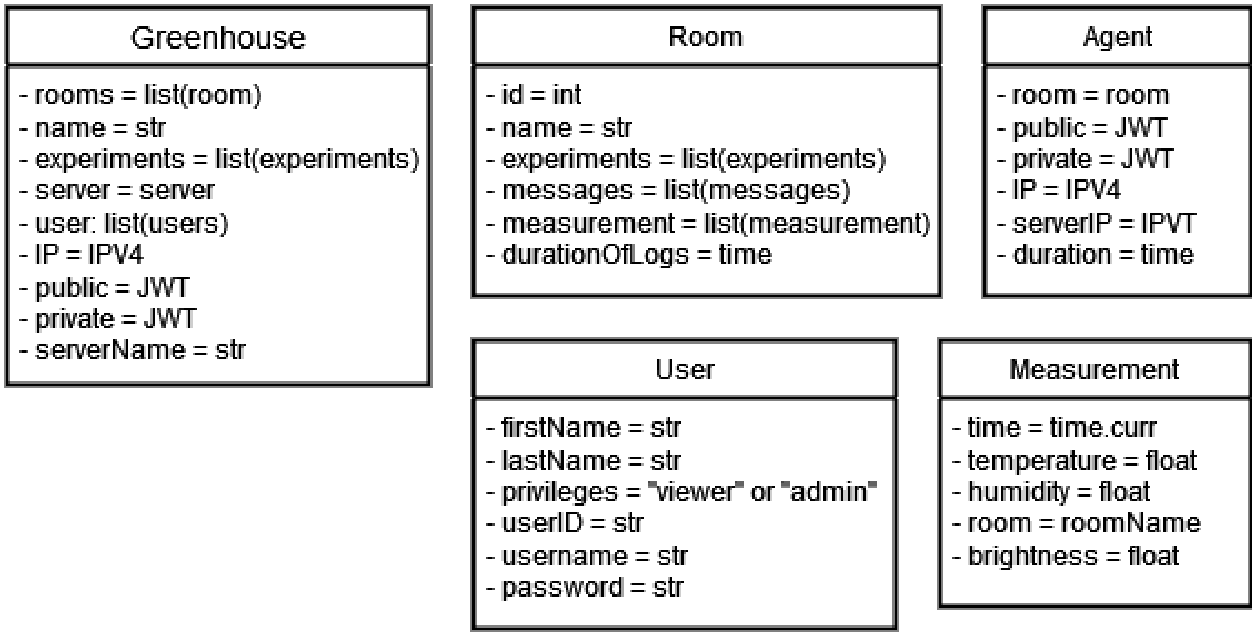

Figure 4: Major class tables for Phase 1

- **Greenhouse**: Greenhouse holds essential information that is required to connect the app to the agents. A major change from the Greenwatch v1 project is that the Server table will be absorbed into the Greenhouse class. Greenhouse holds an aggregation of rooms.

- **Room**: Room represents a room in the greenhouse, Holds attributes like name, id, agent_id, readings=[(temp, humidity, brightness, timestamp)]. Creating one will only need a name for the user. Once the room is created, the agent.py script will begin downloading into the browser. This agent script will be tied to the room using a hashed key that the room will generate upon creation. Room holds an aggregation of measurements.

- **Measurement**: A measurement is an object that will be created by a particular agent script and then will be destroyed upon being sent to the server. A measurement will have an id that will make it distinct from other measurements taken by the agent as well as the temperature, humidity, light, and pressure levels of the room. The measurement will also have a timestamp, a room ID and an experiment ID.

- **Agent**: Agent represents the individual agent script that is produced for the room. This class will provide all the mechanisms for receiving and sending data to and from the server.

- **User**: A user will hold attributes like first_name, last_name, username, password, email, phone, admin=False. This is the user account used to login to the web application. Only an admin user will have the option to create users and if they will be admin or not

3.3.6 Minor System Endpoints

- **127.0.0.1:5000/user/{user_id}** Update a user by id
- **127.0.0.1:5000/refresh** refreshes and maintains access for the client
- **127.0.0.1:5000/greenhouses** creates a new greenhouse

3.4 User Interface Specifications

This section describes the user interfaces, and their uses in detail for the users and future developers.

3.4.1 - Minimum Viable Product

The front-end team will be developing a login screen using HTML/CSS as well as the page that shows a preview of each room and its current readings for temperature, humidity, etc. This will be the very least we accomplish in case getting the UI to communicate with the backend gives us more trouble than expected. The customer would like for us to develop a user interface which contains a login screen and then a screen that will show the data for each of the bays of the greenhouse.

3.4.2 - Plans for UI

Once we have accomplished developing the minimum viable product, we would like to add the following features:

- Viewing each individual room in depth
- Notes sections for each bay for non-verbal communication
- Ability to add a room to the readings screen
- Application accessible on all platforms (web, mobile, mac, windows)
- Have a filter option to show readings over different periods of time
- Have a settings option for admins to adjust rooms' floor/ceiling for readings

3.4.3 - Overview of UI

This section will outline the various pages within the UI as well as their uses and interactions with the users so that when designing the pages, the development team will know what they are designing for.

3.4.3.1 Login Page
- Text Boxes for user email and password
- Account authentication will communicate with the server on the Raspberry Pi
- Ability for user to sign up for email updates on the greenhouse bays

3.4.3.2 Home Page
- Displays preview of all rooms with live readings from the server (Temp. and Humidity)
- User is able to select a room which will take them to the page for that specific room
- Settings cog that will take user to set up screen or a settings popup window within the home screen
- Notes section that regards to the entire Greenhouse

3.4.3.3 Room Page
- Each bay of the greenhouse is accessible via a container on the room page
- Containers display a preview of the room's stats with charts or measurements
- Settings cog for user to change settings for the room
- Collapsable notes section within room page

3.4.3.4 Settings
- User is able to adjust the floor/ceiling level that the temperature and humidity should be above or below
- User is able to adjust the frequency that the agent will fetch the readings from the server
- If admin user is able to adjust settings for other users and add or remove users

3.4.3.5 Notifications Box
- Hide able notes section for each individual room
- Notes section will be used for users to leave notes whenever they have checked on a bay or attended the plants
- Different Icons or labels for notes pertaining to a specific experiment and indicating what form of work was done

3.5 Non-Functional Requirements
The nonfunctional requirements below refer to the criteria that specify the operation and qualities of the Greenwatch system. Unlike functional requirements which define what a system should do, nonfunctional requirements describe how the system performs certain functions or the qualities it must have.

3.5.1 Management
- An admin must have the ability to add/remove rooms in the greenhouse environment.
- An admin must have the ability to add an agent to each room.
- An admin must have the ability to add new users
- An admin must have the ability to assign and revoke privileges of users

3.5.2 Technical
- The homepage must have a preview of the current measurements.
- The charts in each room must be able to be sliced down to a time frame specified by the user.
- The agents must be able to connect directly to the web server
- The server must be able to determine whether an agent is active or inactive

3.5.3 Performance
- The server must be able to respond in a timely manner
- The UI must be user friendly and adhere to common UI structures

3.5.4 Security
- The user must be able to login with a username and password
- The server must use JWT's to validate user activity
- All interactions with the server must travel through the routes specified by flask
- Server must only accept data from agents created by a credentialed user

3.6 System Evolution/Maintenance
As the system evolves and moves into the later phases of the project, there might be some additional maintenance procedures that will need to be done. A knowledge of basic server maintenance will be required as well as some degree of familiarity with **DigitalOcean** and web hosted applications. These requirements however, will not be very difficult to pass on to the customer. There will be a continued maintenance document that will be produced near the end of the project.

4. Other Deliverables

At the minimum, the software requirements, project plan, user manual, GitHub repository and a minimum viable product should be completed by the end of this semester.

- **User Manual** The user manual should include a step-by-step, in-depth guide that properly explains how to use the web application. It should contain visual assistance to help explain how to navigate throughout the application.

- **GitHub Repository** This online repository will contain any documentation, layout designs, and source code used throughout the lifetime of the project. Within the repository, a list of all past contributors should also exist.

- **Developer Manual** The purpose of this manual is to allow new developers joining the project to get brought up to speed on the inner workings of the software. This is to prevent any misinterpretations of documentation in the future.

5. Glossary

1. **Agent**: A software entity that performs tasks on behalf of another entity, typically operating autonomously within a networked environment to collect, process, and exchange data.

2. **Front-End**: The part of a software system or website that interacts directly with the user, presenting data and controls for the user to operate the application.

3. **Back-End**: The server-side of a software application, which handles data storage, business logic, and application performance, not directly seen by the user.

4. **Server**: A computer or software system that provides data, services, or resources to other computers (clients) over a network.

5. **Raspberry Pi**: A small, affordable, single-board computer popular for educational, hobbyist, and industrial applications, often used for programming, electronics projects, and embedded systems.

6. **User-Interface (UI)**: The graphical layout and elements through which a user interacts with a computer, application, or machine.

7. **Sensor**: A device that detects or measures physical properties (temperature, humidity, light intensity, etc.) and converts them into signals which can be read by an observer or by an instrument.

8. **Actuator**: A device used to effect a change in an environment by moving or controlling a mechanism or system, e.g., opening a valve, turning on a light.

9. **Database**: A structured set of data held in a computer, especially one that is accessible in various ways, used for storing historical records of sensor readings and system actions.

10. **API (Application Programming Interface)**: A set of rules and definitions that allows software programs to communicate with each other, facilitating data exchange and integration between different systems.

11. **Cloud Computing**: The delivery of computing services—including servers, storage, databases, networking, software—over the internet ("the cloud"), offering faster innovation, flexible resources, and economies of scale.

12. **HTML (HyperText Markup Language)**: The basic language used to create web pages. It tells the web browser how to display text, links, and images on a page.

13. **CSS (Cascading Style Sheets)**: A language used to make web pages look nicer. It controls the style of a web page, including colors, layouts, and fonts.

14. **Python**: A programming language that is easy to learn and use. It's used for many things like websites, games, and analyzing data.

15. **JavaScript**: A programming language that makes websites interactive. It lets you add things like animations and buttons that respond when clicked.

16. **React**: A JavaScript tool for building user interfaces, especially for web applications. It lets developers create reusable components that update in real-time as data changes.

17. **Docker**: A tool that makes it easier to create, deploy, and run applications by using containers. Containers allow a developer to package up an application with all the parts it needs, such as libraries and other dependencies, and ship it all out as one package.

18. **Flask**: A simple framework for building web applications using Python. It's lightweight and easy to use for creating small to medium websites, APIs, and web services.

19. **SQLite**: A lightweight database system that is stored in a single file, making it very easy to use and manage without the need for a separate database server.

20. **JSON (JavaScript Object Notation)**: A format for storing and transporting data, often used when data is sent from a server to a web page. It's easy for humans to read and write, and easy for machines to parse and generate.

21. **DigitalOcean**: A service that offers easy-to-use cloud computing to help you set up and run applications online. It's popular with developers and small businesses for its simple setup, affordable prices, and straightforward tools for managing websites and apps.

22. **JWT**: A compact, URL-safe means of representing claims to be transferred between two parties. It consists of three parts: a header, a payload, and a signature, which are encoded as a string of characters and used primarily for secure authentication and information exchange.

6. Appendix

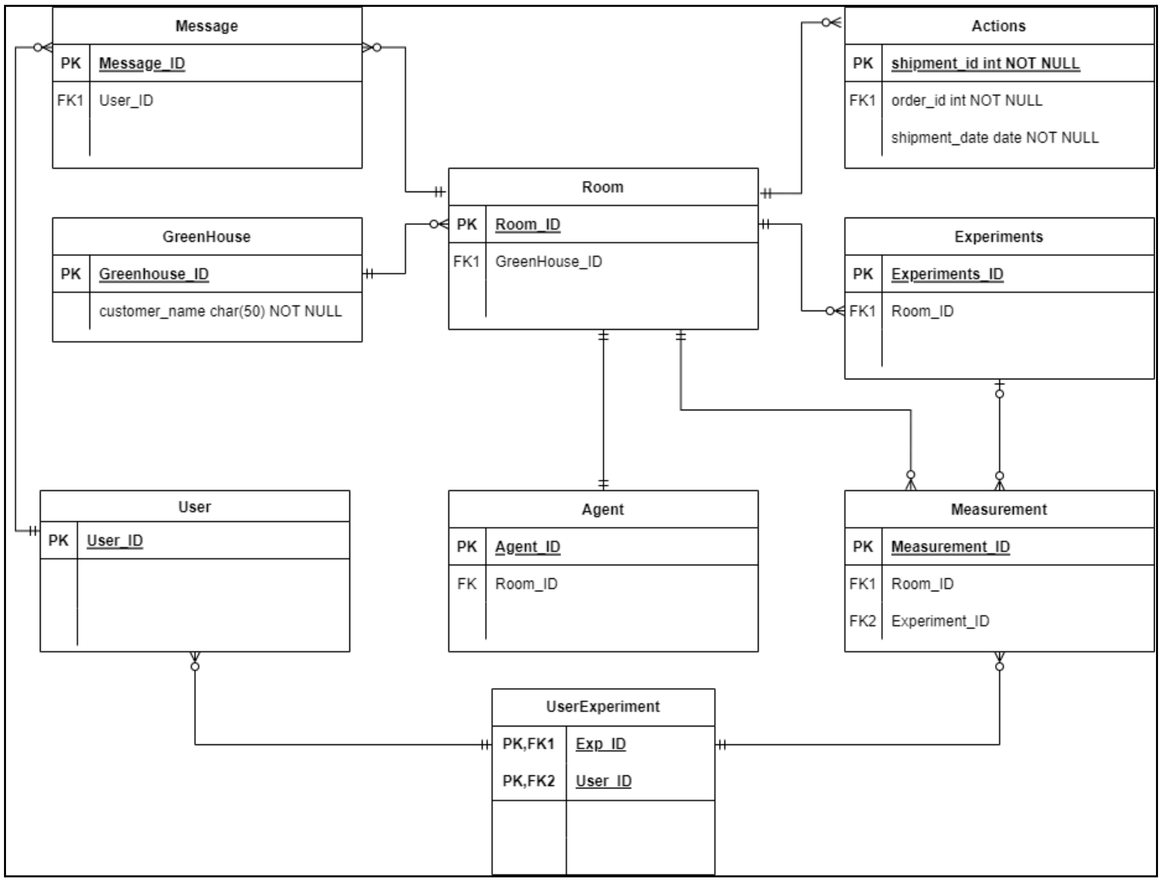

Figure A1: Entity Relationship Diagram

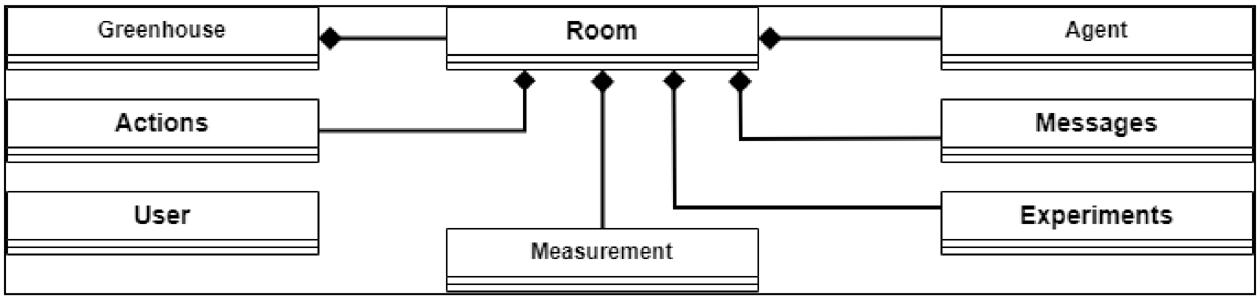

Figure A2: Final Class Diagram

7. References

A. Wilcox-Hirst, “Interfacing a K30 CO2 sensor with a Raspberry Pi for Remote Air Quality Sensing,” *Compute Nodes*, 17-Dec-2020. [Online]. Available: https://computenodes.net/2017/08/18/__trashed-4/. [Accessed: 13-Feb-2023].

B. Hughes, Greencode Logo. 2024.

Initial State, “How to build a raspberry pi temperature monitor,” *Medium*, 02-Dec-2021. [Online]. Available: https://medium.com/initial-state/how-to-build-a-raspberry-pi-temperature-monitor-8c2f70aca ea9. [Accessed: 13-Feb-2023].

“IOT based Green House Monitoring System using Raspberry Pi,” *YouTube*, 18-Nov-2018. [Online]. Available: https://www.youtube.com/watch?v=RYaguyzxSIA. [Accessed: 13-Feb-2023].

Jenfoxbot and Instructables, “Raspberry pi irrigation controller,” *Instructables*, 13-Oct-2017. [Online]. Available: https://www.instructables.com/Raspberry-Pi-Irrigation-Controller/. [Accessed: 13-Feb-2023].

N. Rawlinson, “Build a greenhouse monitor with a Raspberry Pi,” *IT PRO*, 04-Dec-2020. [Online]. Available: https://www.itpro.com/network-internet/internet-of-things-iot/357985/build-a-greenhouse-m onitor-with-a-raspberry-pi. [Accessed: 13-Feb-2023].

Peter Czanik March 9, P. Czanik, M. 9, |, 1 Comment, Steve Ovens (Alumni, Stephan Avenwedde (Correspondent), and Peter Czanik Peter is an engineer working as open source evangelist at Balabit (a One Identity business), “Collect sensor data with your raspberry pi and open source tools,” *Opensource.com.* [Online]. Available: https://opensource.com/article/21/3/sensor-data-raspberry-pi. [Accessed: 13-Feb-2023].

R. Barnes and R. runs R. P. Press, “Hydroponic gardening with a Raspberry Pi,” *The MagPi magazine*. [Online]. Available: https://magpi.raspberrypi.com/articles/hydroponic-gardening. [Accessed:

13-Feb-2023].