Greenwatch v2
Test Document

Revision 1

Prepared by

Sharome Burton
Delton Hughes
Victor Marchesi
Calvin Walmer

April 28, 2024

Table of Contents -- 2

1. Introduction/Overview

1.1 Statement of Purpose
This document acts as the preliminary test plan for the Greenwatch greenhouse monitoring software project. It outlines the range of tests to be conducted, the tasks to be accomplished, the necessary resources, and the methodologies and procedures to be employed in testing prior to the release of a deliverable.

1.2 Product Overview
Greenwatch shall provide a system for monitoring and controlling environmental variables within the greenhouse located at Bolin Hall at Midwestern State University.

1.3 Test Types
Due to the nature of our project, which was an improvement on a previously established code base, unit testing was not as important as it might have been if we were writing all of the code from scratch. Because of the high level of interaction between the user and our final product, integration testing of some form was performed at nearly every team meeting to ensure the product was functioning as intended.

1.3.1 Integration testing
The initial modules of the product were already completed by the previous development team of the GreenWatch v1. Integration testing was performed to ensure that the modules were interacting as intended.

1.3.2 System testing
Once the system had been hosted, integration tested, and the agents deployed. We began running system tests to ensure that the user would be able to interact with the system easily and that key functions could be performed.

1.3.3 Unit testing
The purpose of unit tests in software engineering is to identify any erroneous, incomplete, or redundant code by focusing on the functions encapsulated by a class and the behavioral state of that class. Unit testing involves isolating and evaluating individual components of the source code, such as functions or methods, to ensure they function correctly across different scenarios.

1.4 Testing Process

Figure 1 shows the process we will use to test and develop our product. The sequence begins from the original product produced by the first development team. From there we will develop a requirements specification document and work through the code to add to and fix previous implementations of features.

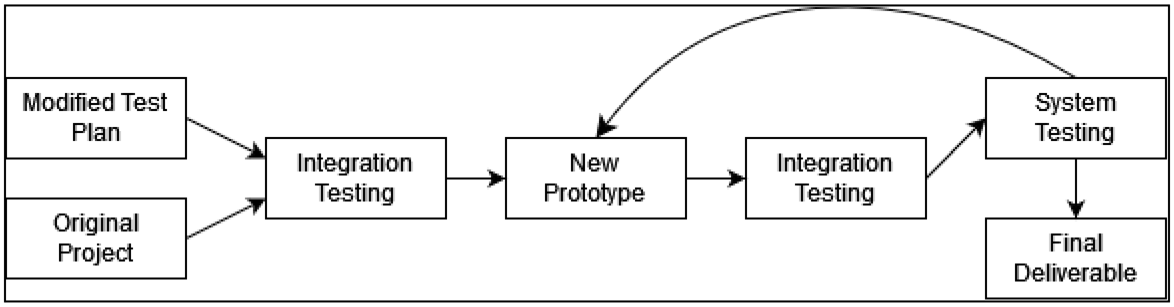

Figure 1: Testing process figure

2. Scope and Objectives

As mentioned in the requirements document, the CarRental software provides support for all levels of the Company's operations except for employee and personnel accounting tasks.
Outlined below are the main test types that will be performed. All test plans and conditions will be developed from the requirements document and project design document.

2.1 Integration testing

This examination confirms that every component of the system communicates effectively with no interruptions in data transfer. The ultimate integration test demonstrates that the system functions as a cohesive unit once all corrections have been made. Test harnesses will be developed to allow a simulated caller to activate each function, with all database interactions occurring in a mock database. The testing technique employed in this phase will be the "Black box" method. Additionally, the "Bottom-up testing strategy" will be utilized throughout the integration testing phase.

Entrance Criteria – None, base modules were completed by the Greenwatch v1 dev team.

Exit Criteria – The web application is fully fleshed out and every page within the application is able to be reached via the interface. Every button that the user may interact with performs its function correctly and completely.

2.2 System testing

This test aims to verify that the functionality provided by the team meets the specifications outlined by the Greenwatch team in the requirements document. It evaluates the software's quality and ensures that it will effectively replace or support the business functions the client needs. The validation of the entire system will also employ the "black box" method as part of the testing strategy. All potential user inputs will be thoroughly tested to identify and correct any shortcomings. Similar to the integration testing, this phase will be conducted using a "bottom-up" approach.

Entrance Criteria – The web application is able to be navigated through efficiently without error and user credentials are able to be utilized to login and interact with the basic functions of the application. Agent hardware must also be able to run python code with a high level of admin oversight.

Exit Criteria – The system should properly display given data from the agents through graphs and room cards in the application. The application should be able to run for an extended period of time without crashing or failing. The agents should be able to run independently with very minimal to no oversight from the admin.

3. Resources

3.1 Team

The Greenwatch v2 team consists of four members who are actively involved in the development of the system. The advisory team consists of two members; one of whom will be the end user of the system.

3.1.1 Development Team

The development team members and their specialities are as follows:

- Sharome Burton (Back-End Design, Microcomputers)
- Delton Hughes (Front-End Design and Planning)
- Victor Marchesi (OS, Front-End Design)
- Calvin Walmer (Planning, Microcomputers)

3.1.2 Advisory Team

The advisory team members and their roles are as follows:

- Dr. Timothy Pegg (Customer and end user of the system)
- Dr. Catherine Stringfellow (Software engineering expert)

3.2 Hardware

In order to properly test the web application the following is specifications are required:

- Computer
- Raspberry Pi's
- Sensor units on the agents (senseHAT, photoresistor with accompanying IC's and wiring components)

3.3 Software

- MacOs or Windows
- VSCode or some IDE
- Python
- Flask (Python Web Framework) installation
- JavaScript
- DigitalOcean(access to the internet)

4. Testing Procedures

During Integration tests and System tests, defects will be recorded as they are detected and will be recorded on a Kanban board to be addressed. Defects on the Kanban board will be categorized as Low, Medium, High, and Urgent priority. Low priority consists of tasks that are not necessarily important and are appended to the back of our task list. Medium priority tasks will come before low priority in terms of getting the tasks finished. High priority being before medium and Urgent coming above all other priorities. All tasks were categorized based on how detrimental the issue was regarding deployment of the project. All the identified defects that had work done will be reviewed during weekly code reviews. When a defect has been resolved, the card will be archived in the kanban board.

4.1 Test Schedule

Given that we have chosen to use test-driven development, we do not have a test schedule. As we go through the creation of the project, we primarily test interactions within the user interface to ensure they work properly. Due to the nature of our project, it naturally requires us to get the system deployed before we can actually test for system development.

4.2 Integration test results

Table 1 shows the defects discovered during weekly code reviews and their statuses as well as their causes and solutions.

Table 1: Integration defects found upon weekly code reviews

Description	Priority	Expected Behavior	Actual Behavior	Cause	Solution	Status
Update room page export button	Low	downloads .csv file	Button wouldn't export data	No logic behind button click to download file	Added logic to download file	Resolved
Add capability to display whether agent is assigned to a room	Medium	Agent IP displays once agent is connected to server	No indication is given	Agent does not send it's IP ever	Agent send IP at startup and updates it in the database	Resolved
Add agent capability to get and execute actions requested by users	Medium	Agent will be able to receive commands given to it by the user (toggle vent/ toggle shade)	Agent does not receive any communication	Agent does not receive any communication	Implement agent sending and receiving data	Resolved
Add 'status' field to action model	Low	Database stores action	Database had no way of knowing whether an action was complete	No field existed	Created a field for status of action and added logic	Resolved
Fix 'flickering' room cards issue on home page	Low	Steady display of room cards	Flickering of the room cards	Reloading entire card instead of individual values	Target individual values	Resolved
Add a dark mode to home page	Medium	Inverting all colors to dark mode preference	Colors not applying to html body	Bootstrap properties added on improperly	Adding JavaScript handling for entire page	Resolved
Fix original room charts	Low	Charts display room data and update when a new data point is added	Charts do not respond to any data in the database	API call to wrong location inside database	Rewrite API call to grab data from correct location	Resolved
Display current day's measurements by default	Low	Current measurements display on cards	Cards do not show any room data	Logic to auto refresh non-existent	Added logic to auto refresh	Resolved

on loading room page						
Get chat box to work with dark mode on home page	Medium	Chatbox changes in between light and dark mode	Chat Box does not respond to dark mode styling	Improper target for classes or id's	Found proper class names	Resolved
Fix coloring on prepopulated values and on login page	Medium	Color on login page needs to be consistent	Login box for credentials stayed white when in dark mode	Not attacking proper class names	Found proper class names	Partially Resolved
Login password color is white on light mode	Low	Textbox font colors will be appropriate to whatever color mode (light, dark) is currently active	Password textbox color is only responding to darkmode styling	CSS syntax error	Fix syntax error in login.css	Resolved
Getting User information to pre-populate user info other than password.	Medium	Admin can see all user details when going in to edit a user other than the user's password	No information about users is displayed	No logic for populating fields	Added logic for populating fields	Resolved
Create room button on mouse click bug	Medium	Room needs to be created by clicking the create room button	No room is being created	Function logic was out of order	Put function logic in order	Functional
Add logic in for deleting a user. Needs to be a check if userID = 1	High	The first user of the system (with user id = 1) must not be able to be deleted	An admin can delete ANY user	Logic did not exist for this	Added logic for deleting users	Resolved

4.3 System test results

All the defects identified while applying each of the following test cases during System test along with category of the error, status of the error and action taken to correct the error will be documented as a System test results document.

Table 2: System defects found upon weekly code reviews

Description	Priority	Expected Behavior	Actual Behavior	Cause	Solution	Status
Get Wifi working for agent	High	Connect to any wifi connection	Wifi adapter is not recognized/ no networks are displayed	Wifi adapter requires standalone drivers	Track down proper drivers and install	Resolved
Create server on initial app startup	Medium	A single server instance is created on start up	No server is created and must be done manually	Servers were originally implemented to be created manually	Add functionality that creates a single server on the startup of the app	Resolved

4.4 Unresolved Defects

Table 3 shows the unresolved known defects at the time of writing this report. As work continues in the next few days some defects might be resolved

Table 3: Unresolved known defects

Description	Priority	Expected Behavior	Actual Behavior	Cause	Solution	Status
Fix app hosting/ crashing issue	High	The app will run in the cloud without the need for frequent restarts	App will not run in production mode	Unknown	None yet	Active
Event Listener modals on rooms not properly closing add room modal	Low	When creating a room through either keypress or mouse click it should close the create room form	Clicking button with mouse stays on same create room form	Unknown	None yet	Active
Edit agent parameters (needs to work)	Low	Button should open some editing modal on the room page to edit the agent parameters	Button does nothing	No code to handle this issue	None yet	Active

4.5 System test cases

The test cases below were derived from the requirements document.

- **System Initialization and Setup**
    - Verify that the Greenwatch system initializes and connects to the server correctly on startup.
    - Validate that the admin can successfully clone the operating system image onto a Raspberry Pi 2B microcomputer.
    - Check that the system correctly recognizes the connected sensors and effectors after the OS clone.
- **User Interface and Authentication**
    - Ensure that the login page correctly authenticates user credentials and rejects invalid login attempts.
    - Test that the system logs out a user and terminates the session upon user request or after inactivity.
    - Verify that the home page correctly displays live data readings from all connected rooms.
- **Data Handling and Display**
    - Confirm that the system accurately records and displays temperature, humidity, and light readings in real-time.
    - Check the functionality to download historical data for a specified time frame in CSV format.
    - Validate the accuracy of charts and graphs displaying historical data on the user interface.
- **Alerts and Notifications**
    - Test that the system sends alerts when set thresholds for temperature and humidity are reached.
    - Verify that the notification system correctly sends emails to designated recipients under specified conditions.
- **User and Room Management**
    - Ensure that an admin can add and delete users as well as manage their privileges.
    - Validate that an admin can create and delete rooms and assign agents to rooms.
- **Agent Management and Operations**
    - Confirm that an agent can be added to a room and its operational status (active/inactive) is correctly displayed.
    - Test the execution of agent.py script on the Raspberry Pi and its ability to post measurements to the server.
    - Verify that the system updates room configurations and agent scripts when modifications are made by an admin.
- **Non-functional Requirements**
    - Evaluate system performance to ensure that responses are received within acceptable time limits.

    - Test the security features to confirm that only authorized users can access sensitive data and system functionalities.
    - Check the system's scalability by simulating multiple users accessing the system concurrently.

5. Appendix

Glossary of Terms

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.

23.**Bottom-up testing**: integrating and testing low-level components in the hierarchy, and then working up the hierarchy of modules until the final module is tested.

24.**Black box testing**:Tester is concerned only with the functionality i.e. inputs and the related outputs, not the implementation of the software.