WAASMA User/Developer Manual

April 2025

By Ruth Garcia, Haley Hamilton, and Gregory Thompson

Table of Contents

User Manual

Setup

Downloading Files

Download files from https://github.com/Kooky909/WAASMA-code.git and move to a findable location.

Installing MongoDB

Begin by installing the following MongoDB tools:
MongoDB Community Edition:
https://www.mongodb.com/docs/manual/administration/install-community/
Mongosh: https://www.mongodb.com/docs/mongodb-shell/install/
Compass: https://www.mongodb.com/docs/compass/current/install/
MongoDB Command Line Database Tools:
https://www.mongodb.com/try/download/database-tools

Once the above are installed, extract WAASMA_flaskdb.zip into its own folder. From the folder containing the now unzipped WAASMA_flaskdb, right click and select 'Open in Terminal'. Enter the following command: mongorestore -db WAASMA_flaskdb WAASMA_flaskdb.

In the MongoDB install folder, navigate to bin. Run mongod.exe. This will open a command prompt with rapidly moving text. This means the database is running.

Run MongoDB Compass and connect to the now-running database. You can view all database data here. To stop the database, open the command prompt with flashing text and press Ctrl + C. This will close it and will require you to re-launch mongod.exe to continue.

Installing Python and related libraries

Is there a better way to install packages than pip install until the errors stop?
Install the latest version of Python. If you already have Python installed, but have not updated it since 2025 started, download the latest version of Python. This will not work with older Python versions, even only a year or two old.

Open the backend folder in the command prompt and run "py -m pip --version" to confirm pip is installed. Then run main.py. Python will most likely state that a module is not found. Run "py -m

pip install [module_name]" on that module and try "py main.py" again. Repeat until the program begins without error, then terminate with Ctrl + C.

Beginning Experiment

Connecting COM Ports

Haley take this part

Starting MongoDB

Navigate to and run mogod.exe. Open MongoDB Compass to view the database live.

Running From the Command Line

Within the backend folder of the codebase, right-click and "Open in Terminal." Run "py main.py".

Accessing While Running

Expected Console Messages

On setup, the program prints a series of status messages. On receiving new settings from an admin, the program will indicate that it is resetting and print status messages regarding that.

If an error occurs during operation, the program may not completely crash. Importantly, some of the sensors may still be running. Attempt to shut down the program using the remote UI. If that does not work, try CTRL+C. If that fails, use the task manager to shut down the command prompt.

Remote Access IP

????????????

Admin Privileges

????????????

Configuring Charts

????????????

Accessing Data

The data may be viewed through MongoDB Compass on the host computer or using the UI analysis tool on any connected computer.

Data export as a file is not currently provided internally by the system. We recommend using MongoDB's internal export tool mango export, (explained at https://www.geeksforgeeks.org/export-data-from-mongodb/). It should have been installed as part of MongoDB Command Line Database Tools during setup.

Shut Down

How to Shut Down from Remote Access

???????????

How to Continue After Shutdown

Double-check that the program is NOT running in another window, then simply run main.py from the backend folder. All user and sensor information will be automatically re-read from the database, and the process will continue.

In order to start a new experiment, use Mongo Compass to rename the old database for later use, and create a new extraction of the zip that comes with the software.

Known Issues and Workarounds

Reset Errors

During automatic system restarting due to new settings, connected computers may freeze up. If that happens, reload the page.

Developer Manual

System Formatting

Overview

The codebase is divided into two sections, the front-end and back-end. The back-end contains the code that will be actively running on the host computer, while the front-end code is sent to connecting computers to display the webpage. Additionally, the database system must be installed on the computer. The database program runs simultaneously in a separate console to the WAASMA system.

Back End

main.py

Primary Execution

The entry point to the server is main.py, hereafter referred to as “Main”. Main completes a series of steps as follows:

- First, Main creates an instance of the sys_state class, which is a semaphore-protected dictionary explained later.
- Main then initializes two additional threads, the mail server and the web application. Both of these threads are given access to the sys_state instance.
- Next, Main connects to the database to collect the information it will need to read from the sensors.
    - Sensor data is stored as a tuple and, soon after, is repackaged into a dictionary for easy access.
- Main creates a new thread for each of the sensors, starting from the sensor_proc function in Main. These threads are given access to the sys_state instance
- Main enters a loop where it checks if the terminate flag has been flipped.
    - If terminate is not flipped, Main checks if new settings are in the database.
    - If there are new settings, main terminates the sensor threads and rejoins them.
    - Main then creates new sensor threads using fresh settings pulled from the database.
- Once terminate is true, Main joins all other threads and terminates.

Sensor_proc

In addition to the primary execution path in Main, the sensor_proc function will run in additional threads throughout the system's operation. Sensor_proc is the entry point for the new threads created earlier by Main. Sensor_proc executions are given access to the sensor data used to create them and the sys_state instance, which is global in main.py.

- Sensor_proc reads a sensor value.
- Sensor_procthen requests a semaphore lock because its task is too complicated for sys_states' internal methods.
- Sensor_proc stores the reading in the database, and a recent readings queue is sent to clients when they connect to backfill the displayed graph.
- Sensor_proc checks if the read value is in acceptable ranges. If not, sensor_proc invokes the notification method to send notifications to the lab team.
- Sensor_proc sleeps for an amount of time determined by a sys_state value.
- If either the terminate or reset flags are flipped, sensor_proc terminates.

Supporting methods

- “new_sensor_wrapper” converts information about sensors into a dictionary similar to a JavaScript Object.
- “notification” creates an email to send to the lab team when values are out of range. These emails are deposited in the mail server's mailbox.
- “app_init”/”mail_init” instantiate and run the webapp and mail server, respectively. They both are the entry points for new threads, and both feed their objects the sys_state.

app.py

I have no idea

mail_server.py

This serves as the mail server for notifying users. To avoid the potentially time-consuming process of interacting with SMTP APIs from freezing any important program processes, emails are sent on their own thread. The mail server has a ‘mailbox’, implemented with a double-ended queue.

- After initialization, the mail server is passive until an email is placed in the outbox.
    - During this phase, it watches sys_state for the terminate flag, and terminates when it is flipped.
- Upon receiving an email in the outbox, the server connects to Gmail using the lab’s email address and a passkey.
- The server repeatedly sends the next email in the queue until the queue is empty.
- The server disconnects from Gmail and returns to the passive state.

sys_state.py

This is a semaphore-protected dictionary. Upon initialization, this class creates a semaphore lock from the Python threading library. It also takes as an initialization argument a dictionary that it will modify. It presently contains four access functions, each of which locks the semaphore

before reading or writing any data. This prevents data corruption if multiple threads modify the same data at the same time. The access functions are as follows:

- get(target)
    - Returns the value at the target index
- set(target, value)
    - Sets the value at the target index to the value. Does not return anything.
- add_to_dict(target, key, value)
    - Assumes the value at the target index is a dictionary. Sets the value at the key in that dictionary to the value.
- add_to_list(target, value)
    - Assumes the value at the target index is a list. Appends the value to that list.

Additionally, sys_state contains hard_lock and hard_release methods that directly trigger the lock or release of the semaphore. These are for complicated changes to the dictionary that would be impractical to implement with methods. These should be used carefully, as failing to release after the hard_lock due to an error will freeze any other thread that tries to access the semaphore.

random_test_sensor.py

This is a dummy sensor used to test features in other methods. It currently returns constant values for testing purposes and should never be accessed in a live build.

db_config.py

It contains the base data required to access the database

w_sensor.py

AHHHHHHHH

a_sensor.py

Are we still using this one?

Front End

SRC folder

App.jsx and App.css

Navbar.jsx and Navbar.css

Navbar.jsx controls the hamburger menu which is used to navigate through the website and takes the user to the Home, Analysis Tool page, Settings and User settings. The user's role will also

affect what you can and can’t see on the navbar based on the user's role/permissions. The Navbar.css is what styled the hamburger menu.

Src → components

Analysis.jsx and AnalysisTool.css

ChangeRangeForm.jsx

Either in the Settings page, or at the bottom of Tank 1 of the home page, the ChangeRangeForm comes up when

ChangeSettingsForm.jsx

ChangeSettingsForm.jsx Allows the user to change the frequency

ConfigSensorsForm.jsx and ConfigSensorsForm.css

CreateUserform.jsx

EditUserForm.jsx

Home.jsx

HomeDisplay.jsx

Login.jsx and Login.css

Login.jsx is the first page the user sees when wanting to access the website. They have to login with email and password. This page uses Login.css so style the page.

SensorDisplay.jsx

SensorSettings.jsx

- Implements Settings.css

Settings.jsx and Settings.css

SlideToggle.jsx and SlideToggle.css

SlideToggle.jsx implements the functionality of changing tabs from “Home” to “Tank 1” and “Tank 2” in the Home screen. SlideToggle.css is what we used to style the slide toggle and what makes the words pop out in blue to indicate what tab the user is on.

UserList.jsx

The UserList.jsx file contains the layout for the list of Users registered into the system and can only be viewed by the Admin as this page contains the users first and last name, email, and role. Through this page, the Admin can change certain user settings, like emails and their names, and it will update to the database. This file implements UsersStyles.css for styling purposes.

Users.jsx

- Implements UsersStyles.css

UserSettings.jsx

UserSettings.jsx is what allows the user to see their individual settings, and they can change their email for notifications, first and last name, or password through here. Every role has access to this page and this page implements UsersStyles.css to style the page.

UsersStyles.css

UserStyles.css is implemented in UserList.jsx, Users,jsx, and UserSettings.jsx to style the User Management page of the website.

Needed Improvements

Things we found that need improvement, polish, and/or development in the future.

Change Range Button on Home Screen

The change range button found at the bottom of the home screen in tank 1 should be moved to a different part of the screen, or the color should be changed to something else that makes the button more visible. When conducting the user acceptance tests, all of our users changed the ranges through the settings page, and when asked if they knew of other methods to do this task, ¾ said they didn’t know there was another option. Some feedback was offered to move the location of the button or change the color so the eye catched it on first glance of the website.

On Screen Out of Range Notifications

We found the notifications on screen, when a sensor is out of range, could appear more clear and obvious. Either a message that pops up or a symbol indicating a warning could be more beneficial to representing that the sensor is out of range rather than having to stare at the plots to identify if they were red (which alerts that there is a sensor out of range).

SMS Notifications

The scope of the notification system was shrunk to only emails. SMS was also requested by the client.

File Export

There is not currently a way to use the UI to create CSV files containing the database data. Using MongoDB command line tools is a temporary solution.

User Logging

The client wanted a feature where the system keeps track of who is logging into the website and what pages they are navigating through, as well as what changes were made if there were changes made.