Greenhouse Website Design Document:

Author: William Kerr

# Introduction:

This website is designed to monitor data that the Greenhouse (located at <insert location here>) generated from a Raspberry Pi. The types of data that the Greenhouse can generate are:

BMS monitoring data (see datasheet)

Tristar monitoring data (see datasheet)

Faculty Sensor monitoring data (see datasheet)

Student Sensor monitoring data (see datasheet)

You can only request data from one source at a time. If the client wants, I can probably change that to include up to 5 data lines.

This website is programmed in Python, using the Flask framework.

Run the website in debug mode using the command python run.py. Do not use debug mode in production! Use gunicorn instead! See Website Deployment section for more details.

# Requirements:

This website requires Python version 3.5 or later, its corresponding pip, and these Python modules (also found in requirements.txt):

flask

flask\_SQLAlchemy,

flask\_datepicker

flask-wtforms

matplotlib

To get the website running, go to <http://python.org/> and download the latest version of Python 3. When you install Python, be sure to specify that you want pip installed as well. Then, run this command:

pip install <path to greenhouse>/requirements.txt

# General Details:

This website uses the Python Flask framework. The Flask framework is optimized for accelerated development. If this project is to be expanded further, this project should be rewritten using the Django framework (that also uses Python).

# Generating Webpages:

This website relies heavily on the Jinja2 templating engine that comes with Flask. Whenever someone requests a webpage, the server will pass the requested HTML template through the Jinja2 template engine. The Jinja2 template engine supports inheritance. I have programmed the website to rely heavily on inheritance. So, all websites will most likely inherit from “layout.html” in the templates folder.

## Layout.html:

This is a basic template that contains the basic information that the webpage needs, such as the navigation bar, meta tags, stylesheets, favicon inks, flash message templates (if a flash() command is called in the routes file), Javascript files, Bootstrap files, etcetera. Most importantly, there is a block content area where all child webpages put their content into.

This piece of code:

{% block content %} {% endblock %}

Will be replaced with whatever content the child webpages want to. That is the goal of a layout template.

All other websites are children of layout.html.

The website should load properly on mobile devices because the website uses Bootstrap to format the pages.

## BMS.html:

This website contains a webform that allows you to ask the web server to send you a graph of whatever data points you want from whatever time you want it to come from. Allows for formatting functions.

When you submit the form, the website will reload, and then your graph will be at the bottom of the website.

There is a test option that allows the developer to test various functions on their server. The client should not choose that option.

## Tristar.html:

TODO

## Sensors.html:

This website allows you to login(TODO), and then see what sensors you have registered.

Then, a web form will appear when you select one of the sensors asking you if you want data from them.

When you fill out and submit that form, a graph will appear at the bottom (eventually).

# Receiving Data:

To send data from Raspberry Pi to the server, you must create a POST request, put your JSON data in it, and send it to the appropriate address.

## For BMS:

All data will be sent as a POST request to:

http://<hostname>/BMS/post-json

Where <hostname> is the name of the host the website is using (e.g. arboretum-backend.soe.ucsc.edu).

Once the server has received your request, it will:

1. Extract JSON data from your request
2. Add it to the database, alongside the date the server received it.

## For Tristar:

All data will be sent as a POST request to:

http://<hostname>/Tristar/post-json

Where <hostname> is where the website is hosted (e.g. arboretum-backend.soe.ucsc.edu).

Once the server has received your request, it will:

1. Extract JSON data from your request
2. Add it to the database, alongside the date the server received it.

## For Sensors:

All data will be sent as a POST request to:

http://<hostname>/sensors/post-json/<owner>/<sensor>

Where:

* <hostname> is where the website is hosted (e.g. arboretum-backend.soe.ucsc.edu)
* <owner> is the name of the person who owns the sensor (e.g. Tela)
* <sensor> is the name of the sensor (e.g. VEML7700)

If there’s no route to accept POST data, then the website will return a 403 (forbidden) error.

When the server receives your valid POST request, it will execute these steps:

1. Extract the JSON data from your request.
2. Create a new instance of a Sensor Data entry with your JSON data, Sensor name, and your name on it, alongside the date that the server received it.
3. Add it to the Sensor Data Entries table in our SQL database.

# Error Handling

This application handles errors using the functions in the errors/handlers.py file. This file is exported as a Flask blueprint, and applied to the main function by registering the blueprint in the \_\_init\_\_.py file.

It can handle these HTTP error codes: 403,404,500,401,405,408,410,413,414,415,429,451,502,503,504.

When an error is thrown by the web server, the software will return an HTML template with information regarding the error.

Example:

Not Found (404): The page you are looking for is not here.

Server Malfunction (500): Our server is experiencing technical difficulties.

# Deployment:

To deploy this website to a Linux Ubuntu web server properly, follow these steps:

## Update Linux

Call: apt update && apt upgrade

## Set Hostname.

Call:

hostnamectl set-hostname flask-greenhouse

nano /etc/hosts

While in the nano editor, add a new line, and add this data:

<hostname><tab>flask-greenhouse

<hostname> is where the website is hosted.

<tab> is a tab indent.

## Add limited user

Add a new user that can execute privileged commands. It’s safer than doing everything at root. Let’s not have hackers here.

Call:

adduser <username>

#Then fill in all the information necessary.

adduser <username> sudo

#This gives <username> admin privileges.

**If you’re going to do a GitHub transfer, skip the next two steps.**

## Make a .ssh directory

Call mkdir .ssh

## Upload your public key

On your local machine (on either Cygwin or on a Linux machine):

Call:

ssh-keygen -b 4096

# generates a SSH key public-private key pair

scp ~/.ssh/id\_rsa.pub <username>@<hostname> ~/.ssh/authorized\_keys

# moves the public key to your server into the .ssh folder.

On your server:

Call:

sudo chmod 700 ~/.ssh

sudo chmod 600 ~/.ssh/\*

sudo nano /etc/ssh/sshd\_config

Modify these parameters in the sshd\_config file as necessary so these parameters have these values:

PermitRootLogin no

PasswordAuthentication no

For safety reasons, we cannot permit a root login, nor can we allow hackers to brute-force any user’s password. Since we have an SSH public key in our server now, we can disable password authentication, and disable root login.

## Setup firewall:

Call:

sudo apt install ufw # this is an uncomplicated firewall.

sudo ufw default allow outgoing

sudo ufw default deny incoming

sudo ufw allow ssh

sudo ufw allow <port number>

# allows our port number to go through. For our test server, it will be 5000.

sudo ufw enable # activae our firewall.

sudo ufw status # see what we have allowed, and not allowed.

## Transfer Files.

You must get the source code of our website onto your server. You have several options to do this:

1. git clone [https://github.com/<repository\_url](https://github.com/%3crepository_url)>
2. Execute an SSH file transfer of the website’s files using FileZilla or another FTP client. Ask someone who works in IT if you are unsure of how to do a SFTP over SSH. Ensure your private SSH key is applied to your transfer, and that you have uploaded your public key to the server properly.

## Install Packages

We need to install python3, pip, venv, our requirements from our requirements.txt file, and create our virtual environment.

Call:

sudo apt install python3-pip

sudo apt install python3-venv

python3 -m venv flask\_greenhouse/venv

# creates our virtual environment.

# if you accidentally created the venv folder in the wrong place, you can move it with

# the mv command.

source flask\_greenhouse/venv/bin/activate

# activates our virtual environment

pip install -r requirements.txt

## Set Global Variables:

Instead of using environment variables, you should use a secure config.json file.

Call:

sudo touch /etc/config.json

sudo nano/etc/config.json

Your config.json file should contain:

{

‘SECRET\_KEY’: “<secret key (sensitive)>”,

‘SQLALCHEMY\_DATABASE\_URI’: “<SQLALCHEMY database URI (sensitive)”,

‘MAIL\_USERNAME’: “<username (sensitive)>”,

‘MAIL\_PASSWORD’: “<password (sensitive)>”,

}

In your flask\_greenhouse folder, there should be a config.py file.

Have our config.py file load config.json instead of use environment variables.

## Running a Debug Server

To run a debug server, call:

export FLASK\_APP=run.py

flask run –host-0.0.0.0

Your website should be running in debug mode. When you navigate to your website, it should load properly. If there’s an error when loading your website, the console will display what type of error occurred. If you have set up error handling on your website (like I did), it should also work, too. My website should serve error templates back to the client in case of an error.

## Install nginx and gunicorn

These will be the two main engines that will serve our website for us while we are away. They are high performance engines, dedicated to serving websites.

Call these commands:

cd ~/

sudo apt install nginx

pip install gunicorn

# ensure you’re still within your virtual environment.

Nginx will handle static files (CSS, JavaScript, Pictures).

Gunicorn will run the Python code.

## Update Nginx configuration files

Nginx does not come with our website enabled by default. To rectify that,

Call:

sudo rm /etc/nginx/sites-enabled/default

sudo nano /etc/nginx/sites-enabled/flask\_greenhouse

The flask\_greenhouse file will contain:

server {

listen 80;

server\_name <hostname>; # example: arboretum-backend.soe.ucsc.edu

location /static {

alias /home/<username>/greenhouse\_website/flask\_greenhouse/static;

}

location / {

proxy\_pass <http://localhost:8000>; #forwards all other traffic to port 8000

include /etc/nginx/proxy\_params;

proxy\_redirect off;

}

}

Call:

sudo ufw allow http/tcp

sudo ufw delete allow <port number> # where port number is your test port

## Create your supervisor that handles traffic without your presence

Your website will run while you’re in an SSH session, but without a supervisor, once you close your SSH session, your website will shut down. To have your website be served while you’re not in an SSH session, install a supervisor to run your server without you.

Call:

sudo apt install supervisor

sudo nano /etc/supervisor/conf.d/flaskblog.conf

The file flaskblog.conf will contain:

[program:flask\_greenhouse]

directory=/home/<username>/greenhouse\_website

command=/home/<username>/greenhouse\_website/venv/bin/gunicorn -w 3 run:app

user=<username>

autostart=true

autorestart=true

stopasgroup=true

killasgroup=true

stderr\_logfile=/var/log/greenhouse\_website/greenhouse\_website.err.log

stdout\_logfile=/var/log/greenhouse\_website/greenhouse\_website.out.log

Call:

sudo mkdir -p /var/log/greenhouse\_website

sudo touch /var/log/greenhouse\_website/greenhouse\_website.err.log

sudo touch /var/log/greenhouse\_website/greenhouse\_website.out.log

sudo supervisorctl reload

# now, the supervisor should be restarted.

**Your website should be up and running now.**

Sources:

<https://docs.nginx.com/nginx/admin-guide/basic-functionality/runtime-control/>

<https://gunicorn.org/>

<https://ubuntu.com/download/server>

<https://www.youtube.com/watch?v=MwZwr5Tvyxo&list=PL-osiE80TeTs4UjLw5MM6OjgkjFeUxCYH>