



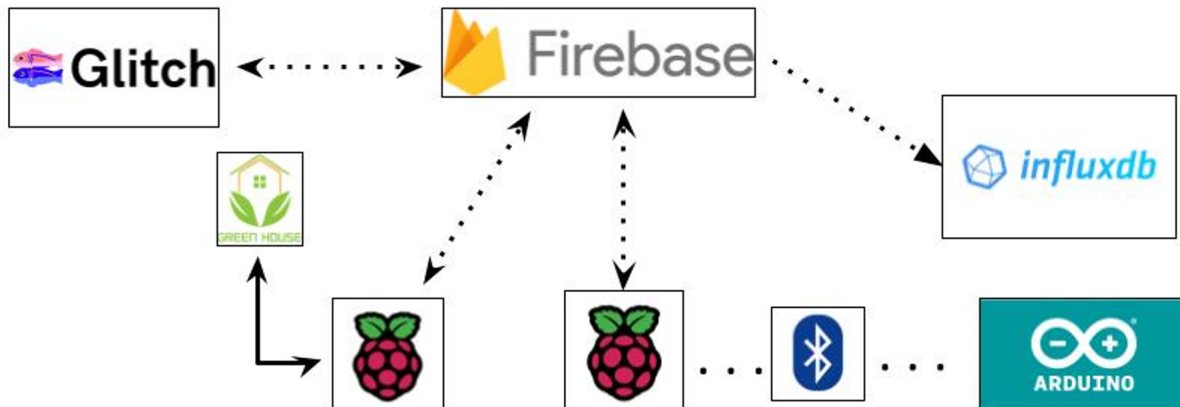
Smart Greenhouse

Ben Lepsch, Adnane Ezouhri, Sebastian Rivera, and Karla Zaragoza

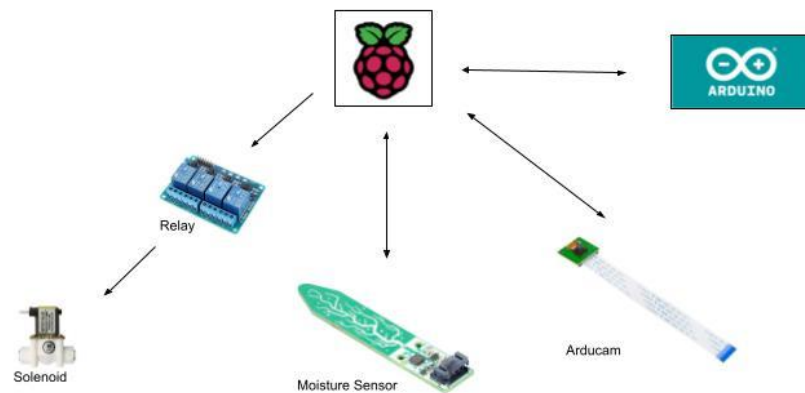
Project Background and Motivation

Our final project is a smart greenhouse. We wanted to create an environment for gardening that utilized modern day technology to successfully monitor and assist in the growth of plants. We monitored the humidity of the air, temperature of the air, temperature of the soil, and the moisture of the soil. Using the data collected from our sensors, we are able to notify the user of the conditions of their garden. We wanted to create this system in order to help manage the waste and overall negative impact farming has on the environment. Although our project is implemented for only a small garden, in theory, we can utilize the system that we have created on a larger scale. During our research, we found that about seventy percent of the water that is used in a single year is used to grow crops, and of that seventy percent, forty percent is lost due to overwatering, poor irrigation, and other factors [1]. With the system we have created, we are able to monitor the environment of the plants and manage the watering of them. Using the data, provided by our sensors we believe that we would be able to taper the effects that farming has on the environment. Another one of our motivations for our project was to build a system that implements and encompasses all of the concepts, ideas, and labs that we have been taught in lecture. These include things like bluetooth, databases, and encryption.

Overview of System Diagram



Overview of Greenhouse System Diagram



Technical Aspects

Sensors and Communication

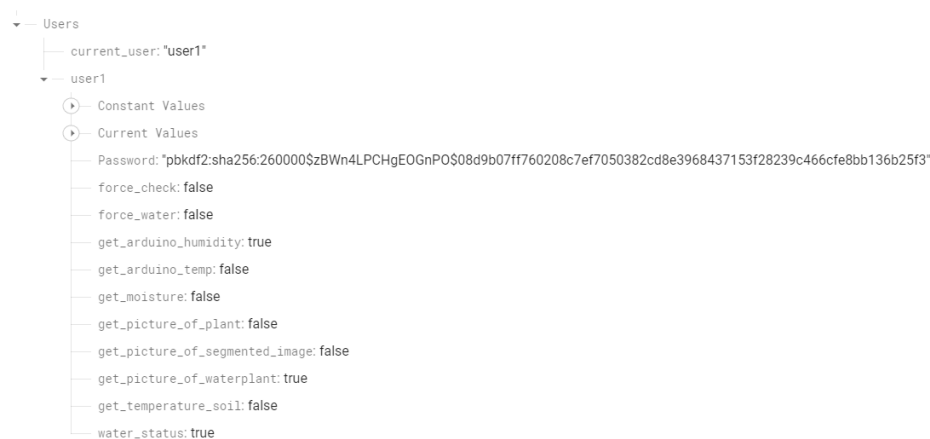
To implement our smart greenhouse system, we used two Raspberry Pis, an Arduino, an Arducam for the Raspberry Pi, a capacitive soil moisture sensor, a 12V PSU, an inlet solenoid, and a four-channel relay. We utilized the Arduino to sense the humidity and temperature of the air. This Arduino then communicated to our first Raspberry Pi using bluetooth. The Raspberry Pi then sent these values to our real time database on Firebase. The second Raspberry Pi was connected to the capacitive moisture sensor. The moisture sensor got the moisture of the soil as well as the temperature of the soil, these values were also sent to our realtime database. This Raspberry Pi was also used for our automatic watering system. We used a container which we connected to the solenoid using tubing. The solenoid is used to control the flow of water, and it is connected to the four-channel relay. The relay is used to control the state of the solenoid, it turns it on and off. The relay is connected to the Raspberry Pi which is then connected to Firebase. From there, the user is able to implement commands using Glitch and our Glitch web app is connected to our Firebase realtime Database. The last device on this Raspberry Pi was the Arducam. The camera was used to take photos of our plants, which we would also segment, and store to our Firebase Cloud Storage. On top of Firebase, we utilized the InfluxDB, we continuously send data from the Raspberry Pi.

Cloud-based Storage

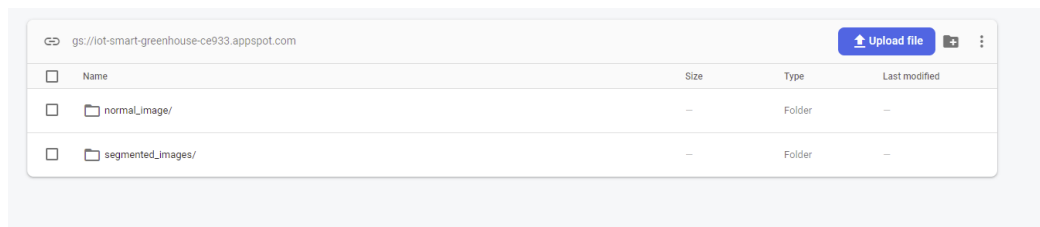
As was mentioned previously, for our cloud-based storage, we utilized the real time database from Firebase, Cloud Storage from Firebase, and InfluxDb. The structure of the real time database is implemented so that it is sorted by users. For new user entities, we would create a new key with a username. Each user then has a set of constant values and current values associated with them. The constant values are the humidity, and temperature of the air, and the moisture and temperature of the soil. We have the user set these constant values so that we can determine if the values we get for the sensors are

too high in comparison and we alert the user. The current values of the user include the current humidity, moisture, and current temperature of the soil and air. The user also has their password. The boolean values are used to indicate a request from our website. These changes in their values will activate our listeners which then force our sensors to write their most recent data back to the database. The Cloud Storage from Firebase is used to store the images that we take using the Arducam. Again, we have boolean variables that control when we get picture requests from our website. Once requested, the camera will take the photo and store it in our normal_images directory. Then when a segmented request is pushed, the segmented image will be stored in our segmented_images directory. The InfluxDB is used to hold the history of sensor's values which may be used for analytics at a later time. An authorized user is able to view these values that are represented with nice visuals.

Real Time Database



Cloud Storage



InfluxDB



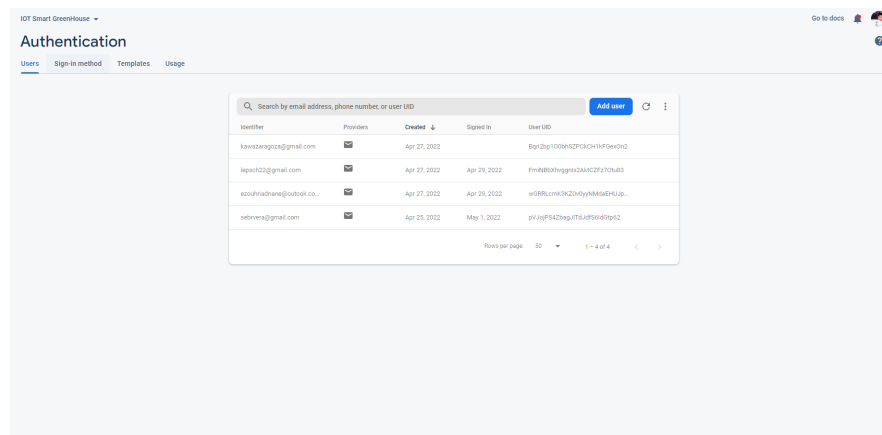
Security

For our project, we used Firebase Authentication for one form of security. Only users registered with Firebase who are authorized are able to read and write to our database. Another form of security we implemented is in our real time database. As was mentioned previously, we store the password of the user in the real time database. In order to make this more secure we hashed the user's password. The password was hashed using SHA 256 and is stored in the Firebase (using werkzeug library in Flask).

Hashed Password stored in Database

```
https://iot-smart-greenhouse-ce933-default-rtdb.firebaseio.com/
└─ Users
  └─ current_user: "user1"
    └─ user1
      └─ Constant Values
      └─ Current Values
        └─ Password: "pbkdf2:sha256:260000$zBwn4LPCHgEOGnPO$08d9b07ff760208c7ef7050382cd8e3968437153f28239c466cfe8bb136b25f3"
        └─ force_check: false
        └─ force_water: false
        └─ get_arduino_humidity: true
        └─ get_arduino_temp: false
        └─ get_moisture: false
        └─ get_picture_of_plant: false
        └─ get_picture_of_segmented_image: false
        └─ get_picture_of_waterplant: true
        └─ get_temperature_soil: false
        └─ water_status: true
```

Authenticated Users



Username	Provider	Created	Signed in	User UID
kavacangon@gmail.com		Apr 27, 2022		8qrd2p1008n52P0CH11F0e0rDn2
hpech02@gmail.com		Apr 27, 2022	Apr 28, 2022	Fn0d80A0hgges2A4K5ZPz70u03
ezouliachane@outlook.co...		Apr 27, 2022	Apr 28, 2022	u08M6_c0m3K23v0y0y0M0d0K1ap...
setevens@gmail.com		Apr 28, 2022	May 1, 2022	g0r3a0P420agJ7F6U0F0000p02

Analytics

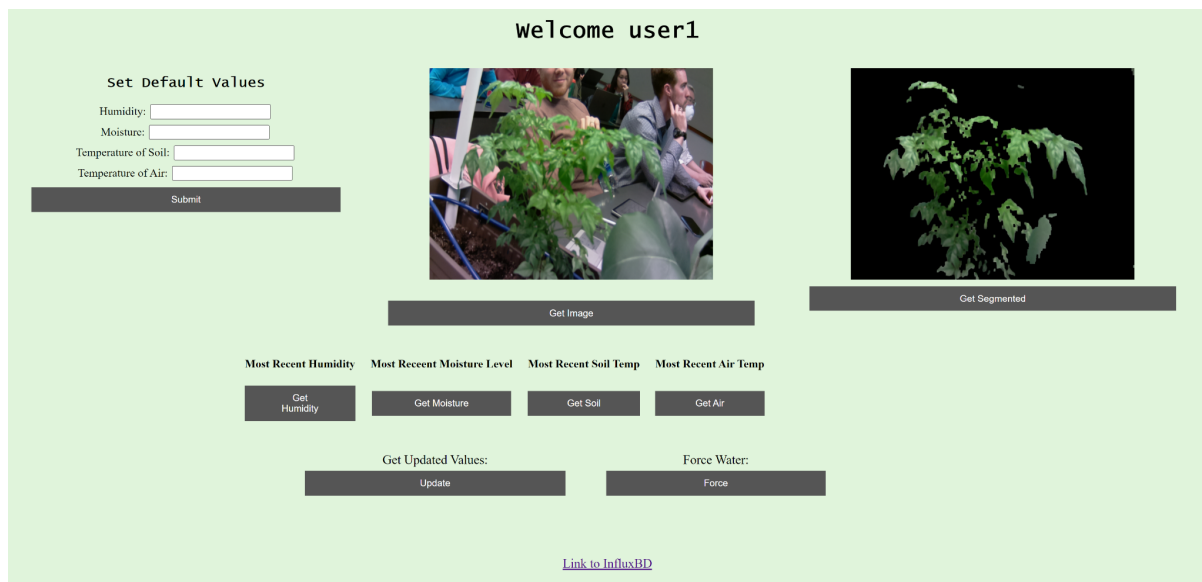
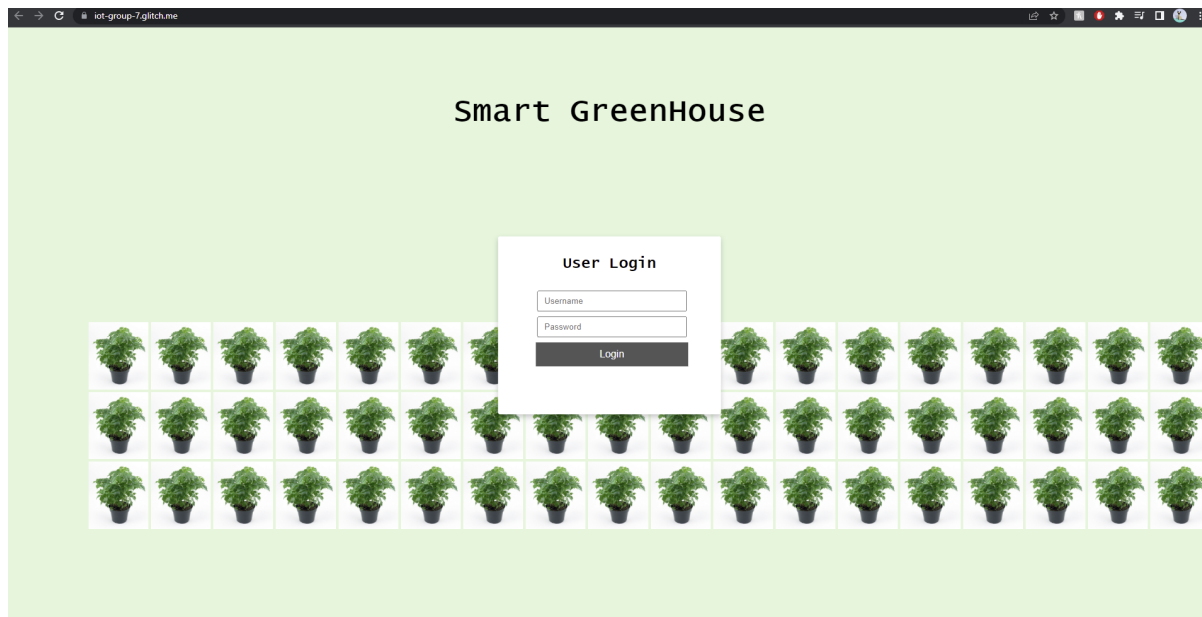
We implemented an automatic watering system that can be found in the `concurrent_jobs.py` file. This just calls a function that looks at all the values of the sensors on a specified time interval to determine if the plant needs watering. We did not get to actually implement it because of bad sensor values but the code is there, and with a good sensor it would work. The next piece of analytics is on the glitch website. Once values are fetched from the database those values are checked up against threshold values. If the value fetched from firebase breaks the threshold conditions it then sends an email notification to the user that the value threshold was broken and gives the relevant information. The next piece of analytics was taking all the values of sensors and displaying relevant information on influxDB with color coded charts, graphs, etc. for easy user readability of data. The last thing we did for data analytics was computer vision. For the computer vision the ultimate goal was to take a picture of the plant, segment that image based off a range of colors green and other then take the total pixel values of green vs other and calculate if the plant likely was infected. What we got around to actually implementing was taking a segmented picture of green pixels, I tried to do a process of detecting the green pixels and other pixels then classifying the image but I never got a correct classification. The issue was the Pi camera would change the color of the image no matter how we switched around the color saturation settings. Even if I took the picture

right up next to the pi camera and it was only of a green leaf it would change the picture to a blue, or a really dark green. This was really demotivating so we decided to just stick with only segmenting the green pixels, and just displaying that image alongside the normal image to the user.

Action

Our smart greenhouse allows the user to interact with our system using Glitch. One of the actions the user is able to take is to set the default or constant values in the database. If this action is implemented then the user will receive a notification via email that these values have been changed. The user is also able to retrieve the most recent humidity and temperature values of the air, and the moisture and temperature values of the soil individually. They can also retrieve all these values at once using the “Update” button. Another action the user is able to implement is to get an image from the Arducam as well as get a version of that image that has been segmented. If the user receives a value that is too high in comparison to the constant value, they will receive an email warning them of this fact. The last action that a user can take, is to water the plant. If they press the “Force” button then with a small delay the plant will be watered. All of these actions implemented by the user will change the boolean value associated with the command in the real time database. This was created using the Flask framework. The linking with Firebase was done by using pyrebase and our backend was written in Python.

User Interface



Group Contribution

Ben - Hardware construction, Pi code, and Computer vision

Sebastian - Website backend (Firebase to Backend linking)

Adnane - Arduino Bluetooth connection for temperature, and humidity

Karla - Report, Frontend of Website, Plant, Dirt

Conclusion

Overall, we were able to successfully implement our smart greenhouse system. We did have some difficulties with the quality of the sensors. We found that the moisture sensor was not accurate in its measurements. We compared the values of the sensor in moist soil and in water, and found that values were not sensible. The value of the moisture of the water was far lower than the soil. In a future iteration of our system, we would use better sensors. Another improvement would be a higher quality camera for the Raspberry Pi. This would allow us to take better pictures and help with the correct segmentation of our images. With this improvement, we could potentially utilize these segmented images to create an algorithm that allows us to detect more discoloration in plants, helping us to detect whether or not a plant is healthy. Although we were unable to implement this in the current iteration of our project we were able to set up the framework for it. Using the InfluxDB, there is also a potential for future analytics with that. Another improvement that would need to be implemented, is to allow for the creation of a new user. Currently on our Glitch site a user is only able to log in and the user we have created was made in Firebase. Ideally we would want a registration page where a user can sign up which would create an entity in our database. A forgotten password feature would be nice too. With these improvements, this project would be able to be scaled on a larger level. This could be implemented in a larger environment, with more plants and more sensors as well as improved hardware. The framework for multiple users is already there, the real time database was created in a way that will allow it to receive data from multiple sensors from multiple users. In conclusion, our final project was successful, we were able to meet our goals for our project. We created a smart greenhouse that is scalable and would be beneficial in lessening the environmental impact of farming. We were also able to create a project that utilized the concepts and ideas that we learned throughout the semester.