IoT Plant Sensor

Design Document

sddec-08
Client: Mark Easley
Advisers: Daji Qiao
Zachary Kauffman
Daniel Phalen
Mason Gil
Walter Gilbert
Thomas Smeed
sddec21-08@iastate.edu
https://sddec21-08.sd.ece.iastate.edu

Revised: 04/21/2021 / Version 3

Executive Summary

Development Standards & Practices Used

Agile Development

Continuous Testing

Summary of Requirements

- Given the sensor module and some basic instructions, a non-technical user should be able to get the sensor working and begin monitoring their garden/nursery
- The front-end interface needs to have a way to view the battery status of the sensors within 5% accuracy
- The front-end interface needs to keep track of the value of the sensors and report if any level is above/below the acceptable level, reporting within 1 hour.
- The front-end interface needs to have available settings for different types of plants. Example: If a user has a garden with 3 plant types, they should be able to set different sensor levels for each
- The hardware should run for long periods of time (few weeks) with minimal user interaction.
- The sensor module should be able to monitor light, moisture, temperature and pH, all within 5% accuracy and <5 minute delay.
- Sensor module should have an attached screen which can quickly display status so that the user doesn't always need to log onto the website

Applicable Courses from Iowa State University Curriculum

CprE 288

CprE 488

ComS 309

SE 319

EE 201

EE 230

New Skills/Knowledge acquired that was not taught in courses

AWS
Project definition
Leadership skills

Table of Contents

List of figures/tables/symbols/definitions (This should be the similar to the project plan)

1 Introduction

1.1 Acknowledgement

We would like to thank Texas Instruments and Mark Easley for sponsoring this project. Without their contributions, we would not have the necessary resources to complete this project.

1.2 Problem and Project Statement

Gardening, farming, or even caring for a houseplant, takes care and time. As life gets busy, people (especially those new to growing plants) are especially prone to forget to water and care for their plants, causing them to die. Even professionals like those who run nurseries need to expend a lot of time and manpower to keep their plants healthy.

IoT sensors can make this easier for both groups. By planting a sensor module along with a plant, users can take the mental work out of gardening, and let algorithms guide their actions. Rather than guess if their plant needs watering again, they can log onto the provided website and examine the plant's water consumption and current supply. If the days are getting longer and sunlight levels are changing, they can view sunlight charts and see if the plant should be moved somewhere else to get more/less sunlight.

The goal of our project is to develop an easy-to-use IoT sensor ecosystem to be used in conjunction with growing and monitoring plant life. The final project will consist of a sensor module that will have attached sensors for light, fertilizer, and moisture, which will connect to a microcontroller that will send the data to an AWS server. Users will be able to go on this website and view the status and relative position of their sensors, as well as generated graphs of each sensor's value over time. Recommendations on repositioning (for better light), watering, and re-fertilizing the soil will be generated off this data. For home gardeners who likely don't tend to their gardens every day, they will be able to receive reminders when the moisture sensor detects less moisture than their plant requires.

Once completed, this project will be a massive boon for farmers and gardeners, both hobbyists and professionals alike.

1.3 Operational Environment

The end product will be exposed to the elements, especially water. High temperatures and rain will likely be a frequent occurrence where the sensors are positioned. While these sensors are not expected to be positioned outdoors during the winter and offseason, they will still likely experience some cold temperatures, especially when used incorrectly and left for longer periods of time than intended.

1.4 Requirements

Functional:

- Given the sensor module and some basic instructions, a non-technical user should be able to get the sensor working and begin monitoring their garden/nursery
- The front-end interface needs to have a way to view the battery status of the sensors within 5% accuracy
- The front-end interface needs to have available settings for different types of plants. Example: If a user has a garden with 3 plant types, they should be able to set different sensor levels for each
- Sensor module should have an attached screen which can quickly display status so that the user doesn't always need to log onto the website

Non-functional:

- The front-end interface needs to keep track of the value of the sensors and report if any level is above/below the acceptable level, reporting within 1 hour.
- The hardware should run for long periods of time (few weeks) with minimal user interaction.
- The sensor module should be able to monitor light, moisture, temperature and pH, all within 5% accuracy and <5 minute delay.

1.5 Intended Users and Uses

Our intended users are all who wish to grow plants and need some help in that process. This group can be broken down into a few subcategories.

- New to gardening. These users will likely not have large gardens to tend to, and will rely more on the reminder and recommendation features (when they should water next, how much, etc), using only a single sensor module.
- Experienced gardeners with larger setups. These people know how to care for plants, but as they expand their gardens/hydroponics setup, they require a bit more help keeping everything straight. These users will likely have a few modules, so they need to be able to view all the data in the cloud and have an easy way to make sense of it. They will likely not need the recommendations feature as much, but the reminder will still be used.
- Professionals. Nursery businesses or farmers* who have lots of experience, lots of plants, and lots of sensors. These users have a very clear understanding of what they need to do and will likely be using the sensors as more of an error detection system, helping them know if there's a part of their field that needs extra attention. This group will not be our core focus.

1.6 Assumptions and Limitations

Assumptions: users will have network connection available, users will have access to electricity/batteries, users will be able to procure the water and nutrients they need for their plants, maximum registered users will be 10

Limitations: we will be unable to test large-scale setups and thus the project will be focused on houseplants and ordinary garden plants.

1.7 Expected End Product and Deliverables

Sensor module

This sensor module goes in the soil with the plant and has interfaces to each of the sensors we will be using (light, pH, moisture) and sends all that data to the microcontroller. This is what would be considered our main deliverable. We've divided this deliverable into several deadlines. The basic hardware should be mocked up and prototyped with a breadboard by mid-April. The PCB design should be designed and ready to go by the end of the spring semester so that it can be purchased and ready to use in the fall.

Front-end interface

This will be built with React and connect to our AWS backend. It will display current and past sensor readings, recommendations and other useful information. This will be used by the user to monitor their sensors and plants. This should be ready by the start of the fall semester, with notable progress made by the end of the spring semester. Unit testing will be done as development proceeds.

AWS backend

Our microcontroller will stream the data over the network to our AWS backend, which will collect the sensor data and process it. The backend will store sensor data so that history graphs can be generated. It will also store user preferences and thresholds for alerts on water/light/nutrient needs. The backend timeline is similar to the frontend timeline and we expect to develop them at the same time. This should be ready at the start of the fall semester, with notable progress and preliminary testing done by the end of the spring semester.

2 Project Plan

2.1 Task Decomposition

Core tasks:

- Development of hardware PCB which will connect to our sensors and stream data to a microcontroller
- Development of a front-end interface with AWS which will take the data provided by the aforementioned microcontroller, parse and store it, and present it to the user

Integration tasks:

- Integration of the microcontroller to the sensor module
- Integration of the microcontroller to the cloud service

2.2 Risks And Risk Management/Mitigation

One potential risk is that the pH/nutrient sensor will be difficult and/or expensive to work with reliably. Our initial research on the types of sensors available shows that this should be possible with an NPK (nitrogen, phosphorus, potassium) sensor, although none of us on the team are familiar and some of the sensors available have spotty reviews (many don't seem terribly accurate). We estimate the risk of this occurring to be roughly 0.4. In the event that it does end up being difficult to use, we can try to find an alternative method of tracking fertilizer.

Another safety concern that we will need to manage is the fact that our electronics will be near water. This will be a main factor in our design and the risk of any issues occurring with this is very low.

2.3 Project Proposed Milestones, Metrics, and Evaluation Criteria

Our key milestones will be

- Hardware prototype on the breadboard
- PCB design
- Front end React app
- Backend AWS server
- PCB integrated with sensors
- Software can interact with the sensor module/controls

Our evaluation criteria for this project will include the following

- Ease with which users can sign up and login to the website
- Reliability and speed of sensor updates to front-end
- Functionality provided by the history and recommendation features
- Reliability of the connection between sensor module and AWS
- Accuracy of the sensor data values

2.4 Project Timeline/Schedule

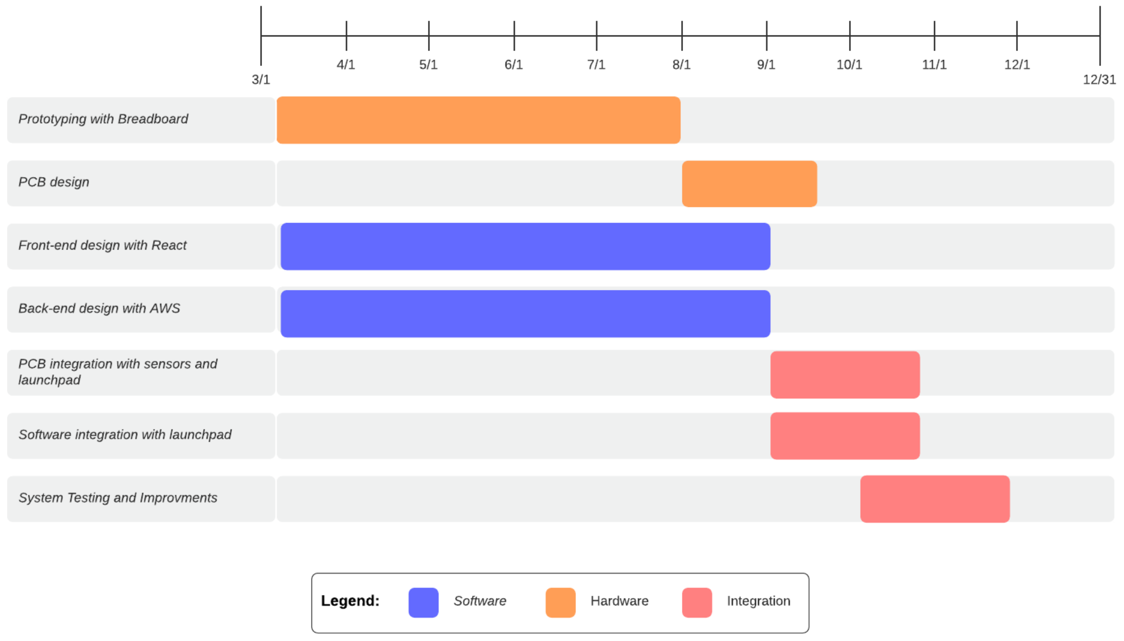


Figure 1

2.5 Project Tracking Procedures

Our team plans on using Git, Trello, and Discord to manage our project.

Discord will be our main, and fastest, method of communication. We will post announcements, ideas, and reminders in our discord server and host our team meetings in the voice room.

Trello will be how we will keep track of our deadlines and tasks. Trello is handy for organization, and our page helps by having lists for backlog, to do, in progress, under review, and finished tasks. This keeps everyone on the same page as to what needs to be done, and by when.

All of our work and important documentation will be stored in the group Gitlab repository. As we develop, we will each have our own branches in that repo and do reviews before merging code into the master branch. This will ensure our code is working properly and the version control will help roll back the code if anything goes wrong.

2.6 Personnel Effort Requirements

Hardware prototype on the breadboard	20 hrs	Prototyping should go quickly once we have the sensors
PCB design	40 hrs	Want to go through iterative development and make sure the circuit will work correctly before we purchase them for use.
React app	60 hrs	Frontends can be tricky, feature creep is real and Javascript based webdev has a literacy barrier
AWS server	20 hrs	Should be easy to crank out in a day, but we will certainly find ourselves researching and debugging unforeseen problems.
PCB integration with sensors	60 hrs	We will need to do a lot of testing on this integration to make sure everything is working correctly.
Software/hardware interaction	60 hrs	Lots of interface testing and moving parts. A large amount of time will be spent communicating hardware and software requirements/limitations.

Figure 2

2.7 Other Resource Requirements

AWS accounts and credentials

CAD software for PCB design

2.8 Financial Requirements

We will need to purchase our sensors and microcontroller, as well as order our PCB once designed.
The total price for this should fall under $100.
The AWS free tier should suffice for our initial implementation.

3 Design

3.1 Previous Work And Literature

digiPlant[1] is the most notable result that will pop up if you start looking for other IoT plant sensors. This project is developed by Microsoft and is primarily focused on teaching the process of IoT, making it pretty simple but effective. It can integrate with Twitter and tweet out the status, which is a handy feature. Outside of that, there aren't many ways to monitor the plant's status remotely, and Twitter as your main remote monitoring mechanism doesn't provide all the information we would like to provide with our solution. [2]

There are many solutions out there similar to digiPlant. Focused on hobbyists, these guides help users set up their own IoT solutions, but are not very user friendly. They only support one or two sensor nodes and require lots of knowledge on the part of the user. Our solution aims to be much more approachable.

Another commercial product on the market is the WANFEI Plant Monitor [3]. This is a sensor node with light, temperature, humidity and nutrient sensors and a bluetooth capability. It's marketed as being easy to use, but the bluetooth connection only lets users monitor when they are right there with their plants. Our solution will be accessible from anywhere and will support multiple sensor nodes.

3.2 Design Thinking

Our project was presented to us relatively open-ended. Thus, most of the first few weeks of the semester were very much focused on defining the project. One of the first aspects of the design that was defined was "smart home". It was clear from the start of the project that we wanted to focus our sensor use on the home and making people's lives easier. Here we explored a few different ideas:

- Smart thermostat system with temperature sensors in every room and an app that let you see and control the temperature of each room
- Hydroponics vertical farming system with moisture, temperature, light sensors
- Pandemic focused biometrics sensors that would let companies track the locations and temperatures of employees in their building for early phase transitioning back to the office

The next aspect that was defined was "plant growth". During brainstorming we had discovered hydroponics setups and that lined up nicely with the smart home focus. By focusing on residential agriculture, we make it easier for people to participate in the very therapeutic act of gardening in their own home.

3.3 Proposed Design

Proposed hardware design:

- PCB daughter board for a TI Launchpad or Raspberry Pi which will have ports for several sensors
    - Since the board will have the capability to support multiple sensors, we will be able to use one board that will monitor all of light, moisture, temperature, and pH.
- Enclosure for board with sensors exposed so that they can be placed while treating the daughter board as a black box

    - Enclosing the board in a box with only sensors exposed may help some consumers be less confused about the setup process since they will only have to worry about the sensors and not the board itself.
- Launchpad/Raspberry Pi will stream data to the AWS server
    - Using the AWS server will be our method of having the hardware report any imbalances in the monitored levels to the user(s) as well as allowing the user to check the levels at any time from any capable device.
- LED panel which will get sent data from the server and display status
    - Receiving data from AWS, the TI launchpad will display some of the information to an LCD that is close to the system. This will allow the user to view the information without having to access the application as well as the battery level.

Proposed software design:

- AWS backend will receive data from the microcontroller
    - Using the AWS server will be our method of having the hardware report any imbalances in the monitored levels to the user(s) as well as allowing the user to check the levels at any time from any capable device.
- React app will take the data from the backend and generate sensor graphs for history of sensor values
    - This will allow for the user to check the history and trends of the sensors in case there is a problem with the monitored values.
- React app will generate alerts when sensor values drop below a certain threshold
    - The user will be sent a notification from the react app that the plant(s) is/are low on water, not enough light, or other issues. This will allow the user to make sure that their plant(s) is/are staying alive.
- Data gets sent to status LED panel
    - Receiving data from AWS, the TI launchpad will display some of the information to an LCD that is close to the system. This will allow the user to view the information without having to access the application as well as the battery level.

This design satisfies our requirements because it allows us to design an easy to use plant sensor module that lets users view their plant status from the cloud.

3.4 Technology Considerations

React: Slow but easy to build user friendly apps on the web.

AWS: Easy to build IoT applications on, free to use for us. Powerful IoT tools and its IAM and Cognito offerings make it easy to allow users to make secure accounts and assign them the right permissions.

Custom PCB: Nothing extraneous, might not be as optimized as an off-the-shelf solution

3.5 Design Analysis

The project is not yet far enough in the development process to assess success/failure qualifications on a large scale. We can, however, speak to the success/failure of what we've done so far.

The components of our software design so far have been successful, which has mostly consisted of getting an AWS backend connected with a frontend as a skeleton from which we can build the project specifics. We have been able to deploy an Amplify app with AWS and mock MQTT data to display on the front-end. This software skeleton we've designed uses AWS Cognito and lets users sign up, sign in, and gives them the proper permissions to view our AWS topics. See Appendix for some more information on what work has been done on the software side.

Additionally, we've been able to get real sensor data up to AWS via a Raspberry Pi gateway talking to a TI SensorTag (over bluetooth). This is good for prototyping, although as the hardware team advances, we might need to develop a different interface for the devices talking to the Raspberry Pi. The sensors exposed via services on the SensorTag are not what we will end up using, so this is a temporary solution.

The parts (as of revision 3 of this document) have all been delivered but basic prototyping is still going on. This, however, qualifies as a success based on the timeline we have proposed and discussed with our client and advisor.

What we've designed so far has presented us with an effective and clear framework to continue to build our functionality off of. Potential modifications will certainly be necessary in the device->AWS interface. We also want to upgrade our frontend, since everything we've designed there so far has been for a proof-of-concept.

3.6 Development Process

We are going to use an Agile development approach because it encourages continuous improvement and evaluation. We've used Agile approaches in other courses and internships and prefer using it in practice to other development processes.

3.7 Design Plan

Preliminary block diagram

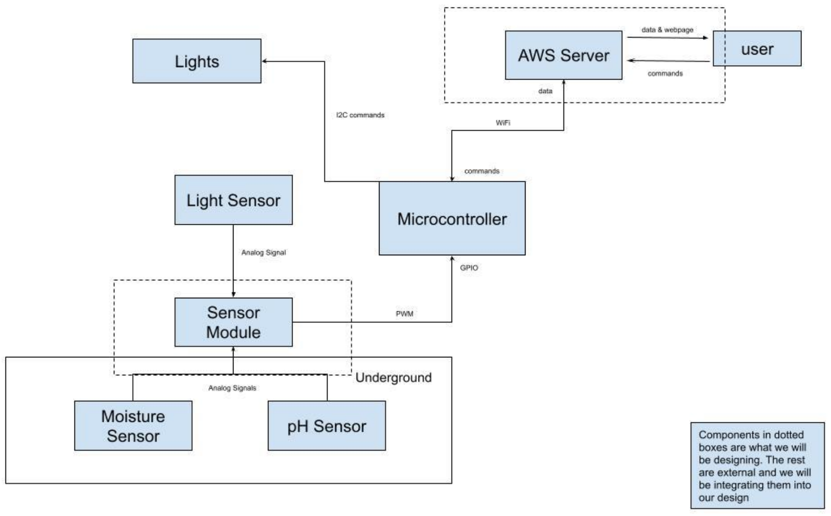


Figure 3

4 Testing

4.1 Unit Testing

AWS server: Building api test scripts where the AWS server's endpoints are called and expected to send back the correct data.

React Front end: The React app will be tested by rigorously trying to break the user interface and writing a small amount of tests for the little amount of states/internal logic the app will hold.

Custom daughter board: This will be mocked up on a breadboard that is connected to the microcontroller, be it the Ti Launchpad or a raspberry pi. The role of this PCB is to connect the GPIO of the microcontroller to the sensors. We will model the breadboard in KiCad and use the test tools that the software provides to make sure that the layout works. When the final PCB design is done, we have it printed and tested to make sure that there are no flaws.

Microcontroller: We will make sure that the microcontroller can read out the proper information from the GPIO ports, since this is what is mainly going to be used.

Sensors: We will have to calibrate the sensors, like the pH or moisture, to make sure that they are working properly.

Hardware sensor module will be tested in isolation. It will be tested in isolation to ensure that the proper sensor values are being taken from the sensors and outputted together to the microcontroller.

Backend will be tested in isolation to ensure that, given fake data in lieu of the sensor module, it will correctly read and process it to present to the frontend.

Similarly, the frontend will be tested to ensure that it displays data correctly.

4.2 Interface Testing

Interface testing would involve the intercommunication of the microcontroller with the sensor to the React Front end displaying the correct values for each sensor like moisture value. This will be done through rigorous testing by having different controls, like high moist to very dry environment.

Communication from the microcontroller with plant interfacing components like water pump and lights, will be tested outside the system. This will allow us to set values for the flow rate of the water pump or brightness of the lights.

Frontend/backend interface will be tested to ensure that data from the backend is correctly displayed by the frontend and that no errors occur as data moves through the pipeline.

Hardware/software interface will be tested to observer sensor values correctly read by the sensor module and displayed to the user via the React app.

4.3 Acceptance Testing

We will demonstrate that the design requirements are being met in section 1.4 by using our project setup on testing setups and using it like a customer would as we are developing it as well as getting outside opinions on the interface. We will be able to notice the areas where the product isn't living up to expectations as we are using it. Our customer will be involved in this acceptance testing during our biweekly meetings.

4.4 Results

Our first semester of work on the project has been successful, leaving us with a solid software prototype and a clear framework to continue to build our core functionality off of. Potential modifications will certainly be necessary in the device->AWS interface. We also want to upgrade our frontend, since everything was put together in order to get the AWS integration working correctly. Everything we've put together so far is in line with our proposed schedule.

The project is still not far enough along that we don't have testing substantive results to conclude success/failure from. From what we have developed on the software side, we have performed some basic testing on different devices to make sure that users can sign up via our front-end and accurately see our back-end data, but these are preliminary tests not covered by our main test plan.

5 Implementation

For the next semester, we plan on getting a PCB mock up and printed, integrating the PDB into the system, AWS Backend, and software monitoring and control integration. We plan on finishing up hardware prototyping by the end of the spring semester and getting a basic framework setup on the software side that will let us

Some things will be started, and worked on throughout the summer and continuing through the fall semester. Towards the end of the fall semester we will be doing our final testing of the project.

6 Closing Material

6.1 Conclusion

So far, our project has mostly consisted of requirements gathering, design sessions, and defining our project. Our project's idea has transformed from a general "IoT" specification to a specific sensor application involving plants and a cloud interface. We have developed a solid proof of concept cloud interface and gotten to work getting real sensor data up to the cloud, while developing a hardware solution with breadboards. Next semester, we will provide a sensor hardware module and cloud interface for the client.

6.2 References

[1] ms-iot (2016). PlantSensor [source code] https://github.com/ms-iot/PlantSensor.

[2] W. I. T. M. Nagase, "Plant App," *Hackster.io*, 27-Dec-2017. [Online]. Available: https://www.hackster.io/windowsiot/plant-app-1167ed#team. [Accessed: 08-Mar-2021].

[3]"WANFEI Plant Monitor Soil Test Kit Flower Care Soil Tester Smart Plant Tracker Intelligent Sensor Plants Detector Bluetooth Monitor for Light Moisture Fertility Temperature Level, for iOS and Android", *Amazon.com*. [Online]. Available: https://www.amazon.com/WANFEI-Monitor-Flowers-Sensor-Moisture/dp/B07ZH7FQJ7/ref=sr_1_5?dchild=1&keywords=plant+sensor&qid=1618934778&sr=8-5. [Accessed: 21-Apr-2021]

6.3 Appendices

Software Figures

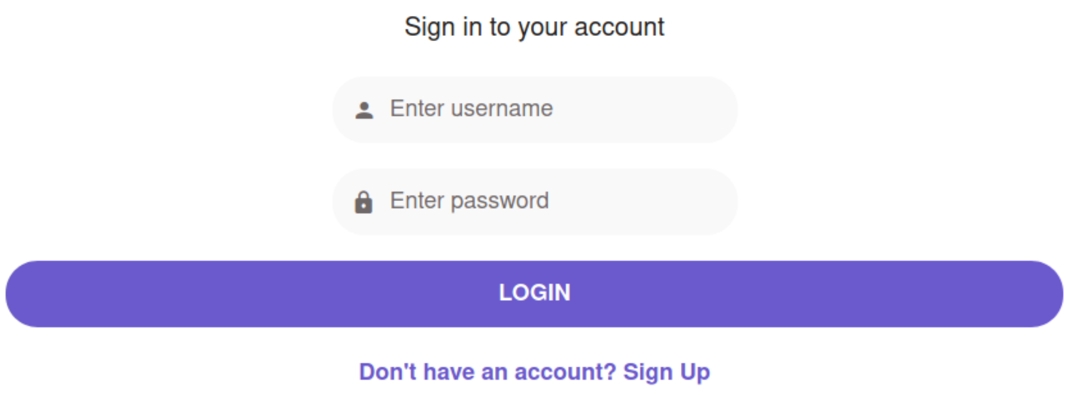


Figure 4: Sign In Page

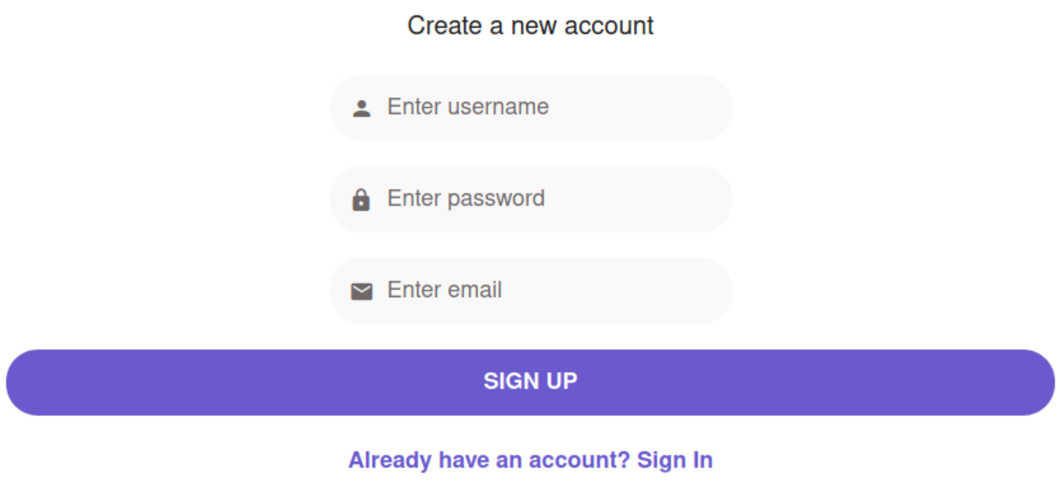
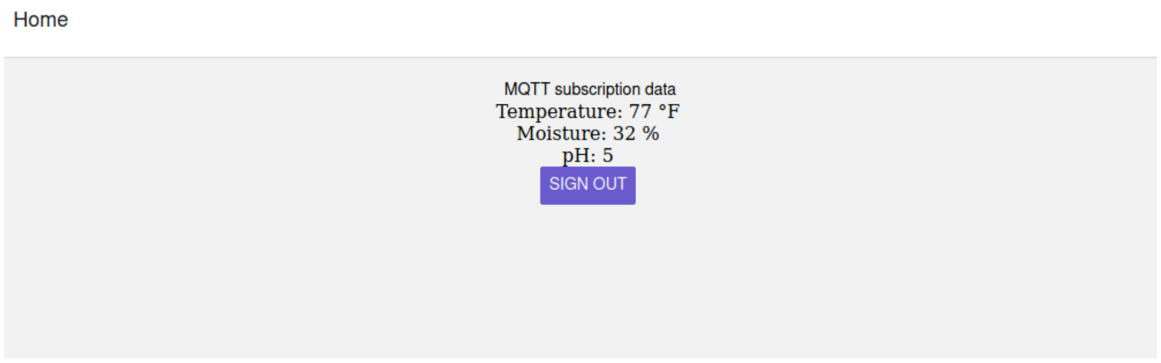
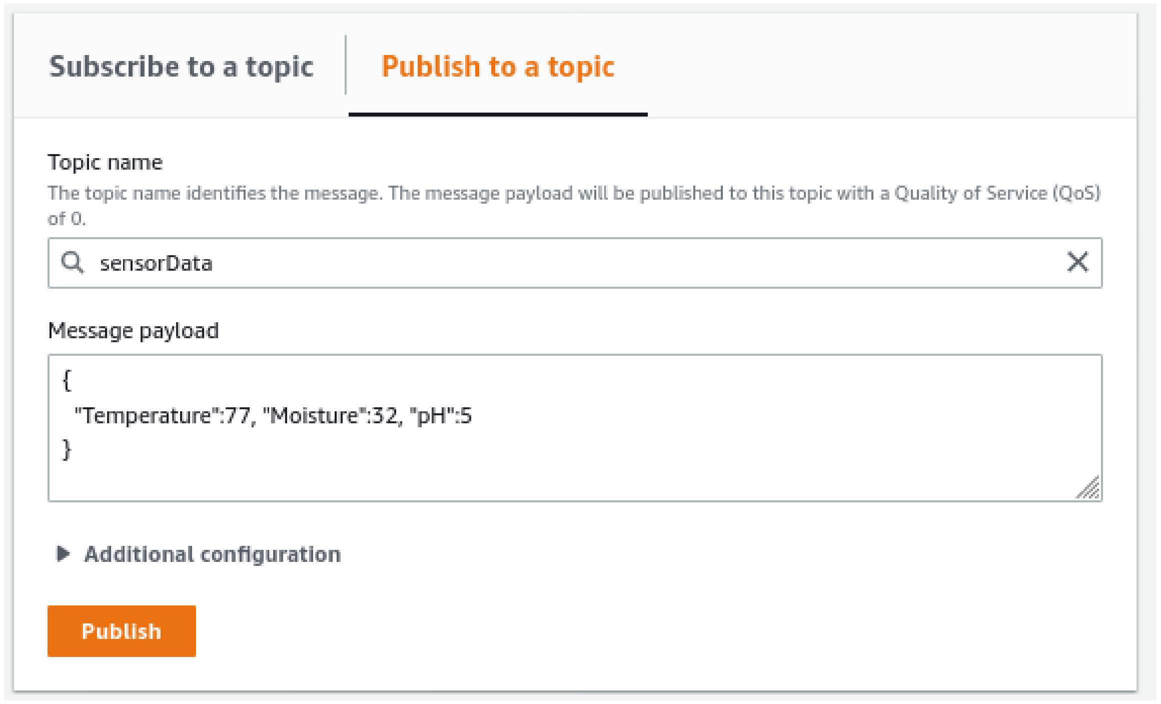


Figure 5: Sign Up Page



MQTT subscription data
Temperature: 77 °F
Moisture: 32 %
pH: 5

Figure 6: Home Page



Topic name
The topic name identifies the message. The message payload will be published to this topic with a Quality of Service (QoS) of 0.

sensorData

Message payload

```
{
  "Temperature":77, "Moisture":32, "pH":5
}
```

Figure 7: AWS's MQTT Test interface which is publishing the data displayed in Figure 6

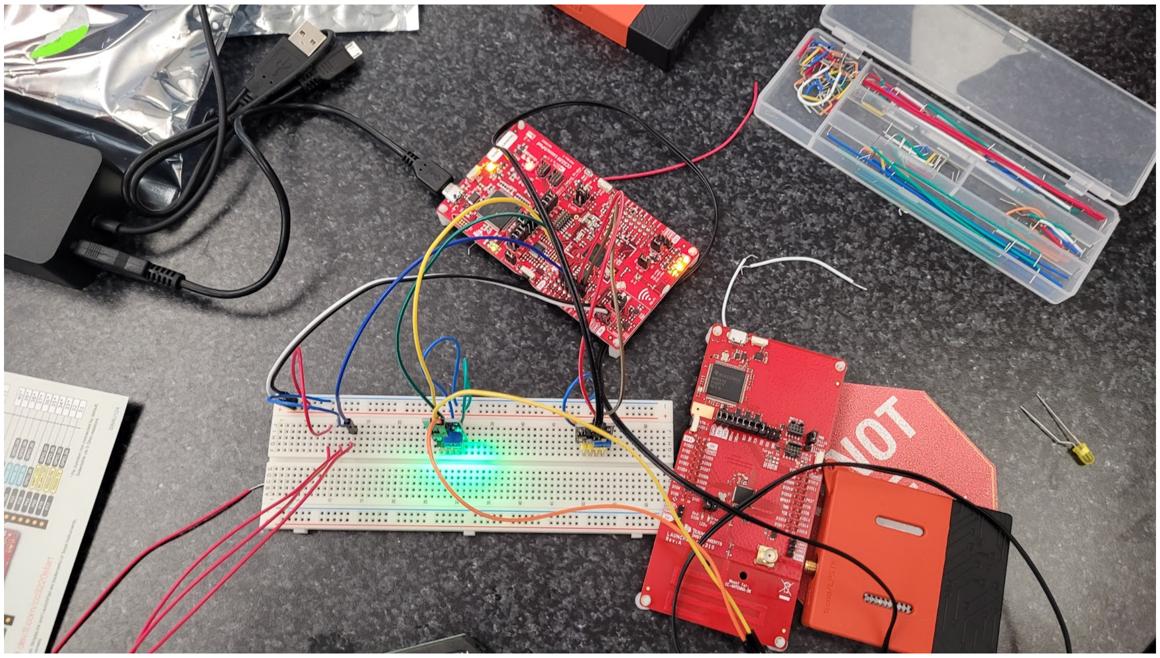


Figure 8: TI Launchpad wired to breadboard that leads to sensors

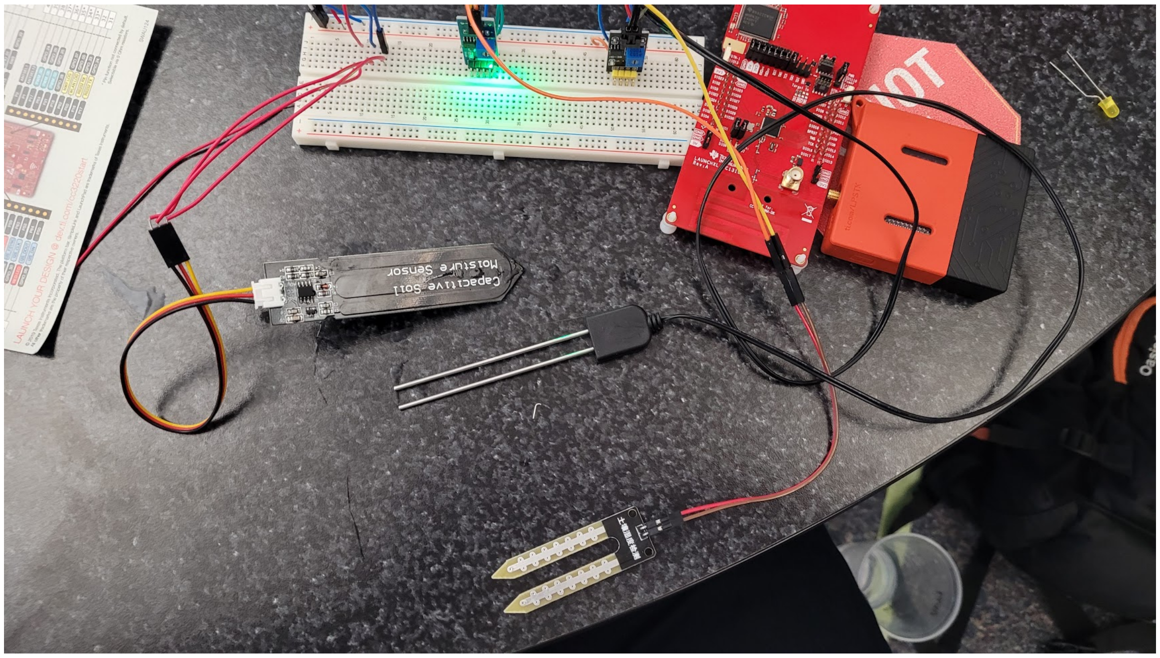


Figure 9: Sensors that are connected to the breadboard

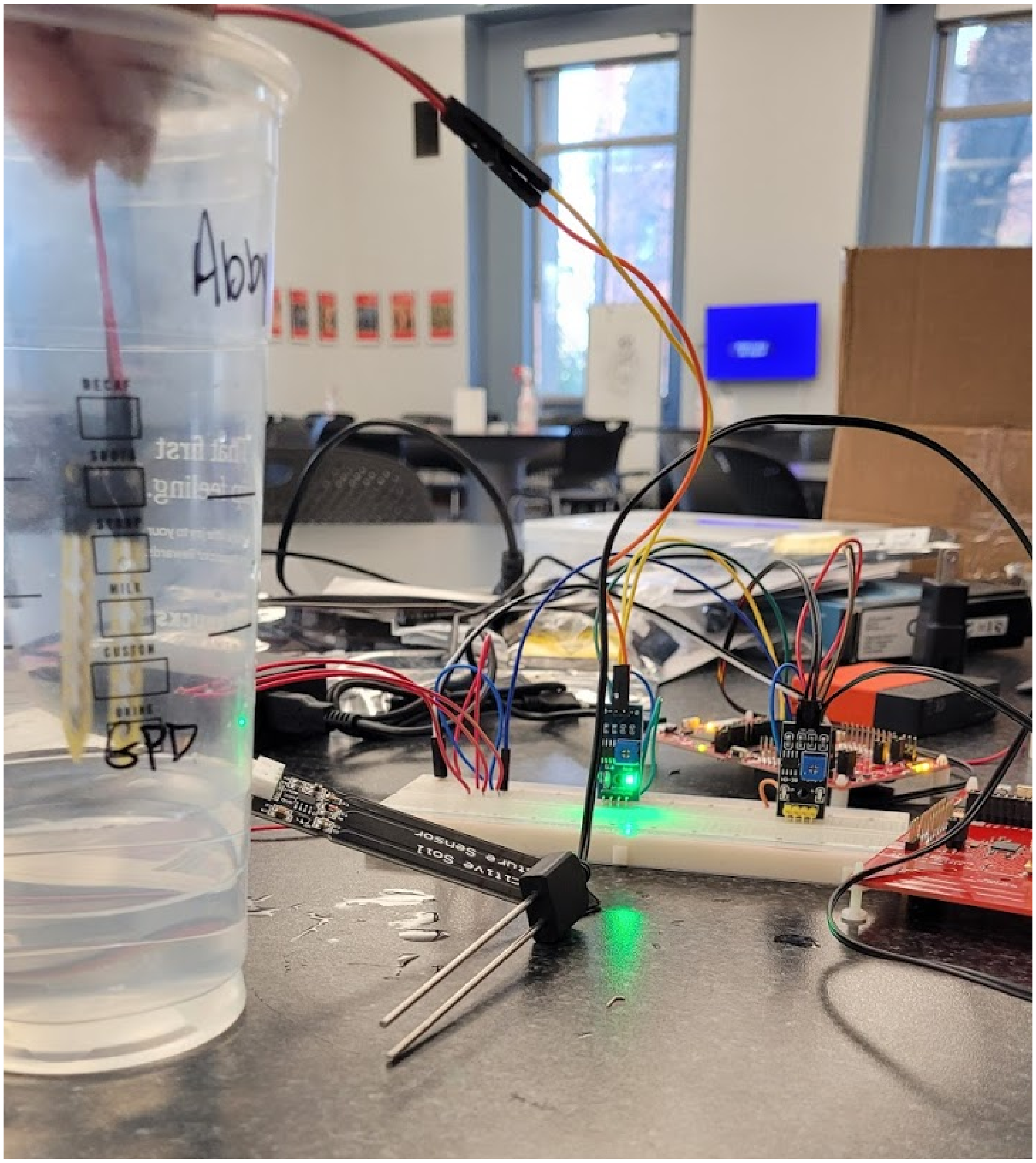


Figure 10: Resistive sensor not detecting moisture

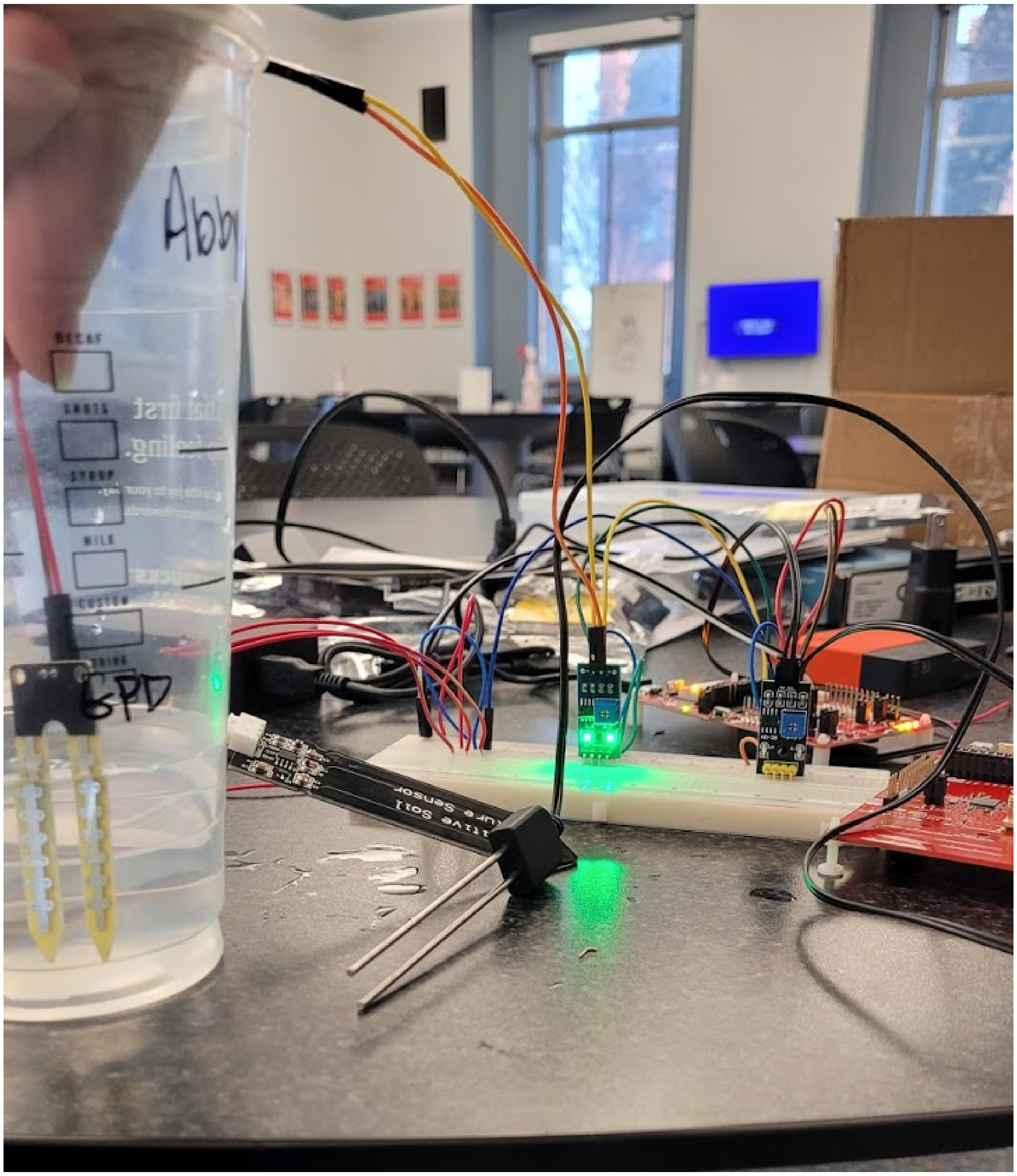


Figure 11: Resistive moisture sensor detecting moisture