Weather Monitoring System – Final Report

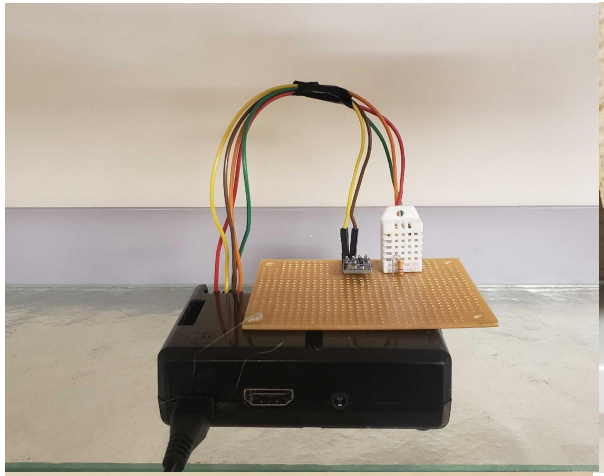
Sagar Pathare and Vimal Kakaraparthi

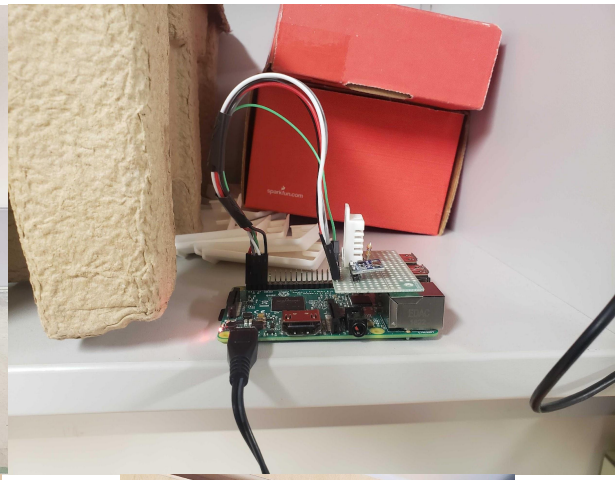
Final State of the system - Hardware

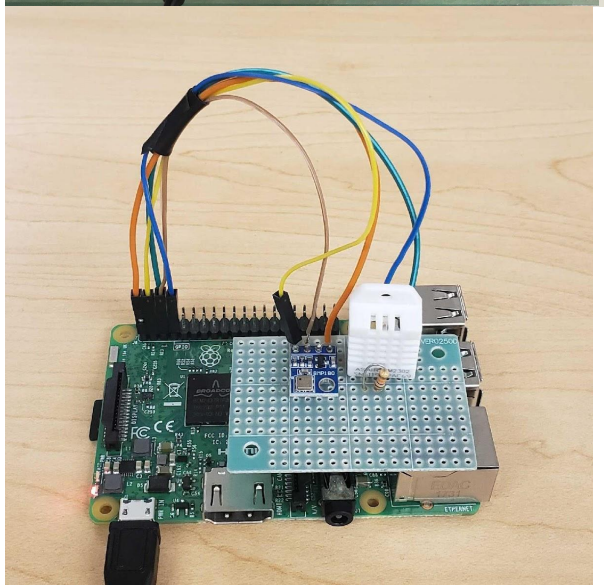
Hardware:

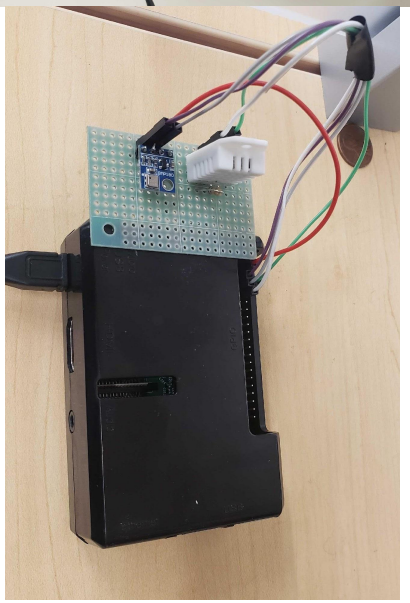
- Vimal implemented the hardware which included:
  - Setting up the Raspberry Pis
  - Configuring and testing the sensors (BMP 180, DHT 22)
  - Soldering the sensors onto a compact breadboard
  - 3D printing the cases to contain the weather stations (2 out of 4 stations)
  - Http post call implementation in case of sensor reading variations

- We implemented four weather stations. All of them check the sensor readings every minute and update the server values in case of variations

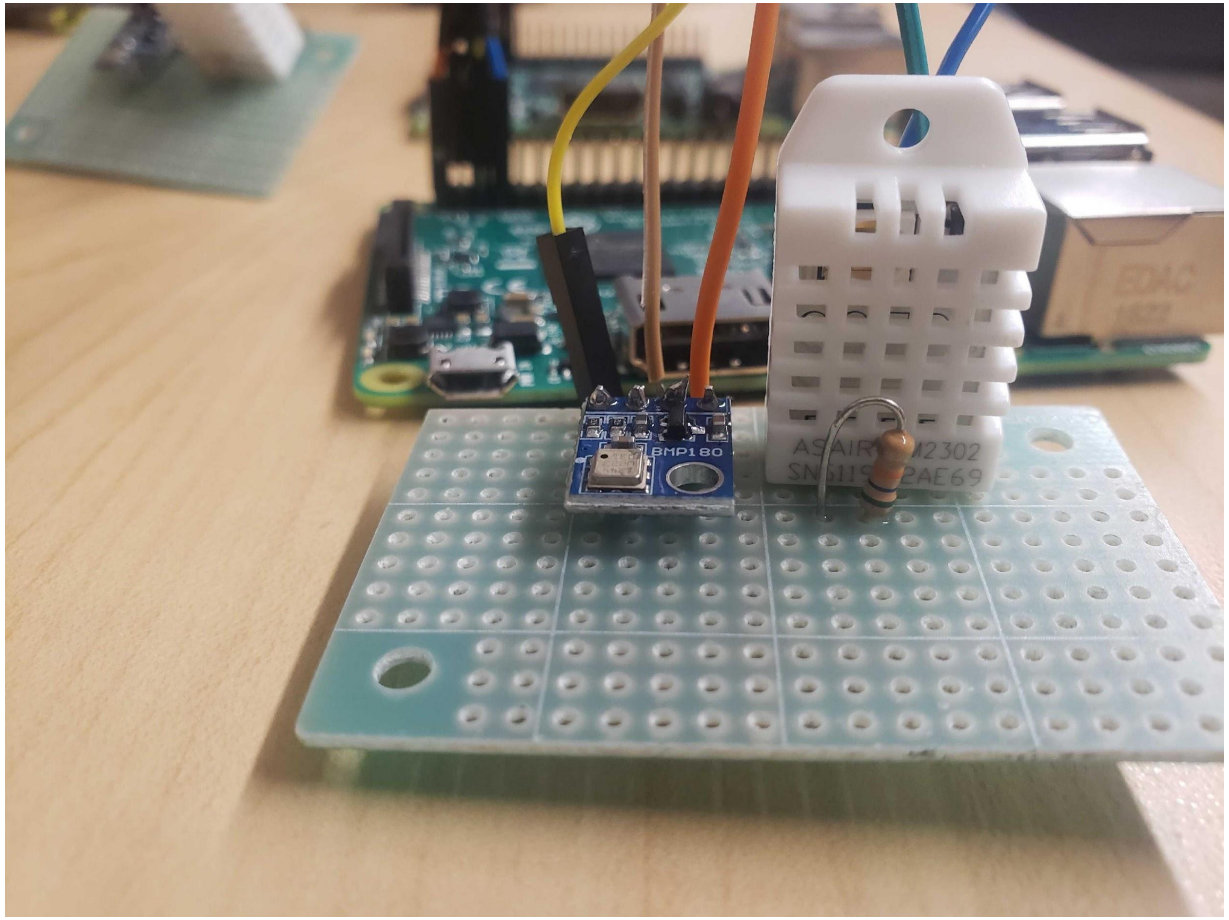


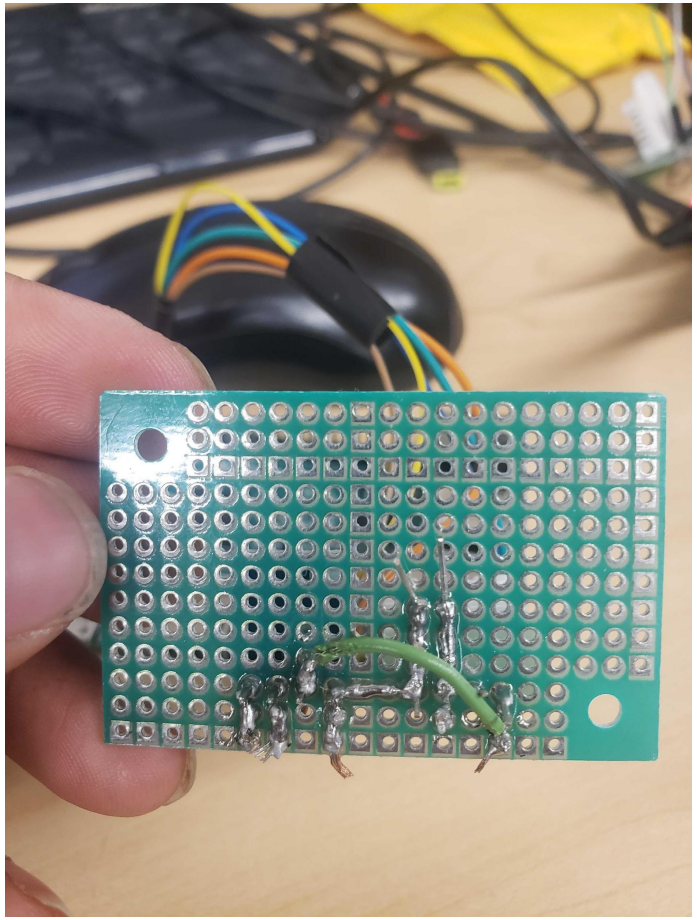






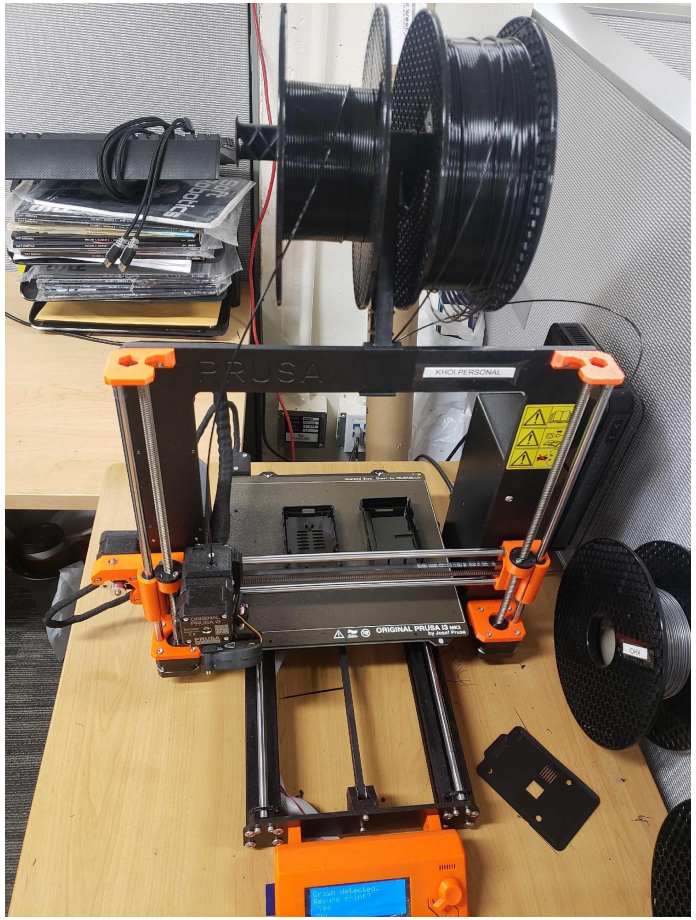
Final State of the system - Hardware





Sensors compactly soldered to a breadboard

Final State of the system - Hardware



3D printed Raspberry Pi cases for some worthy stations

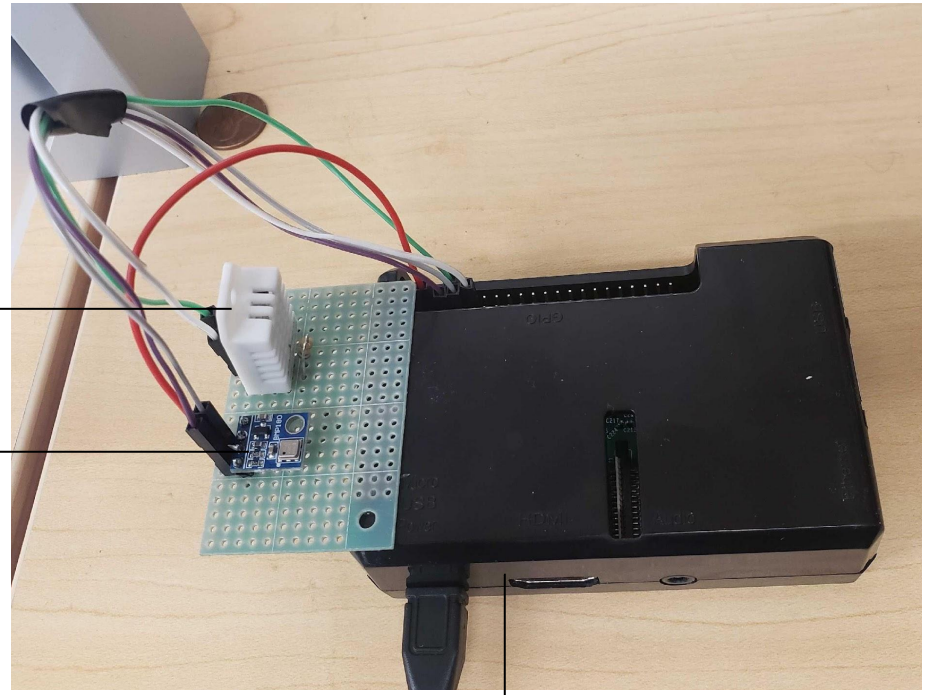
Final State of the system - Hardware

Weather Station

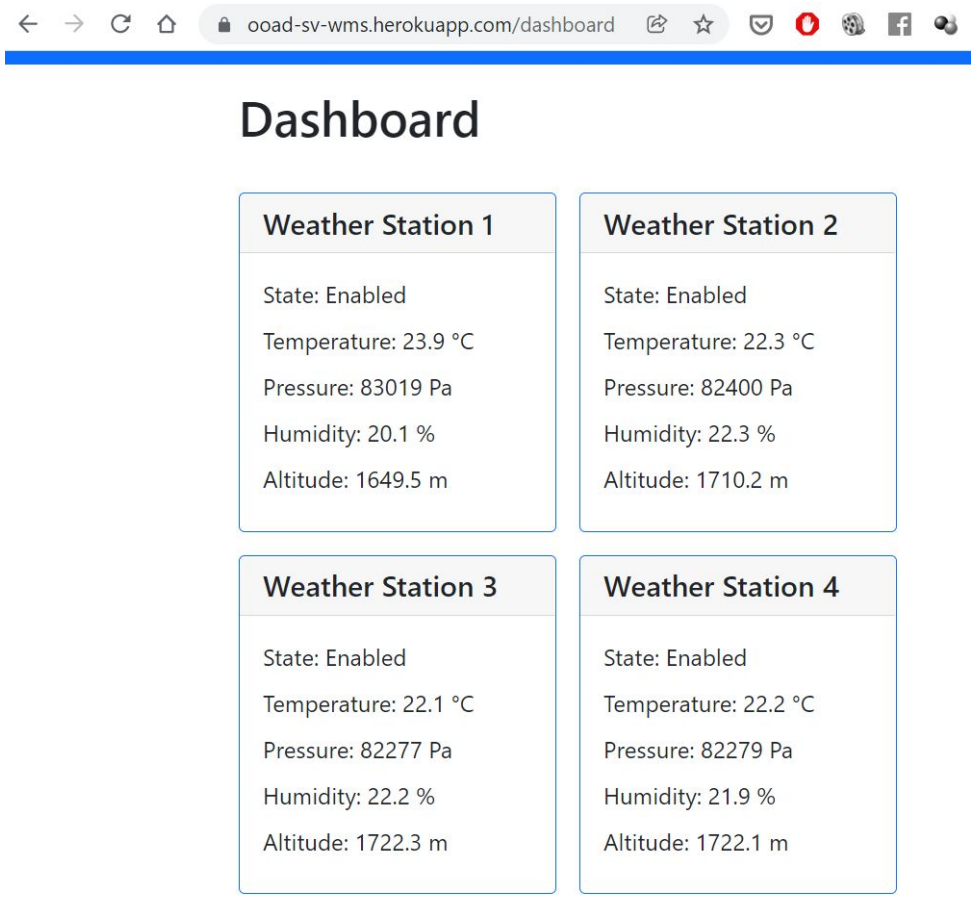
DHT22 (temperature and humidity sensor)

BMP180 (Pressure and altitude sensor)

Raspberry Pi 3



Hardware – Software Interface (Server Communication)



Final state of the system - Software

- Sagar worked on the software part of the project. He set up a node backend web server connected to a PostgreSQL database with an MVC pattern. He also deployed it on Heroku and set up the auto-deployment pipeline.

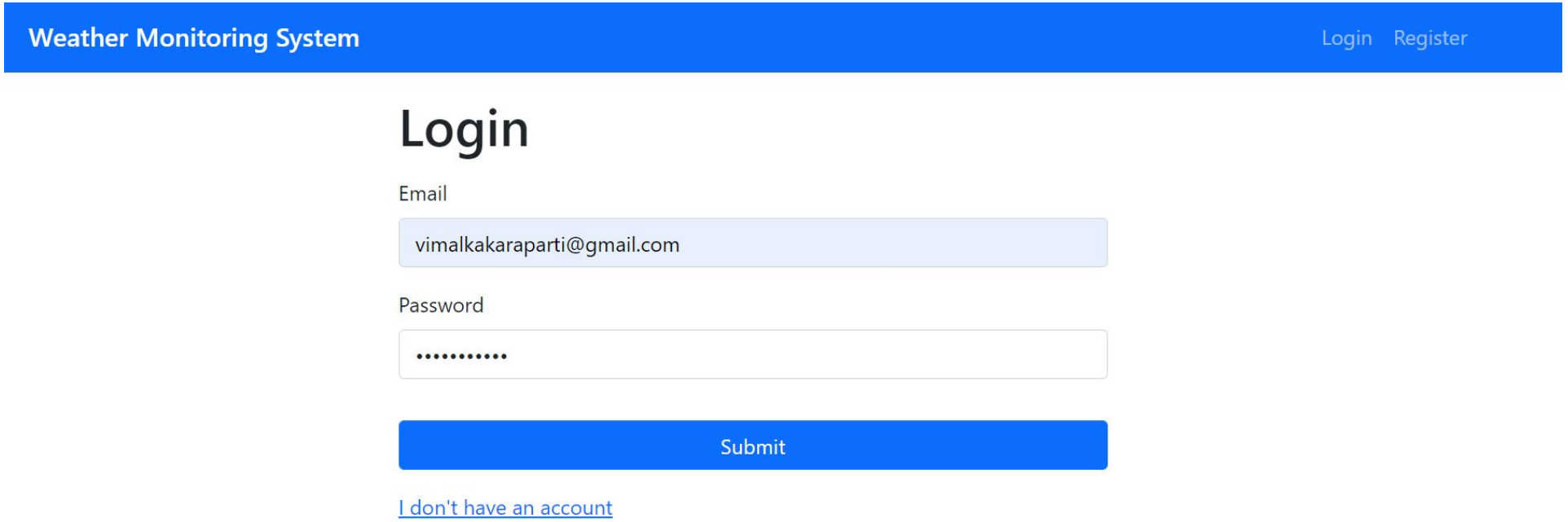
- He implemented the following pages, including the frontend UI and the backend logic- registration, login, dashboard, update profile, update preferences.

- He also implemented an API that the weather station will call to update the weather values in the database. Here's the link for the project- https://ooad-sv-wms.herokuapp.com/

Final state of the system - Software

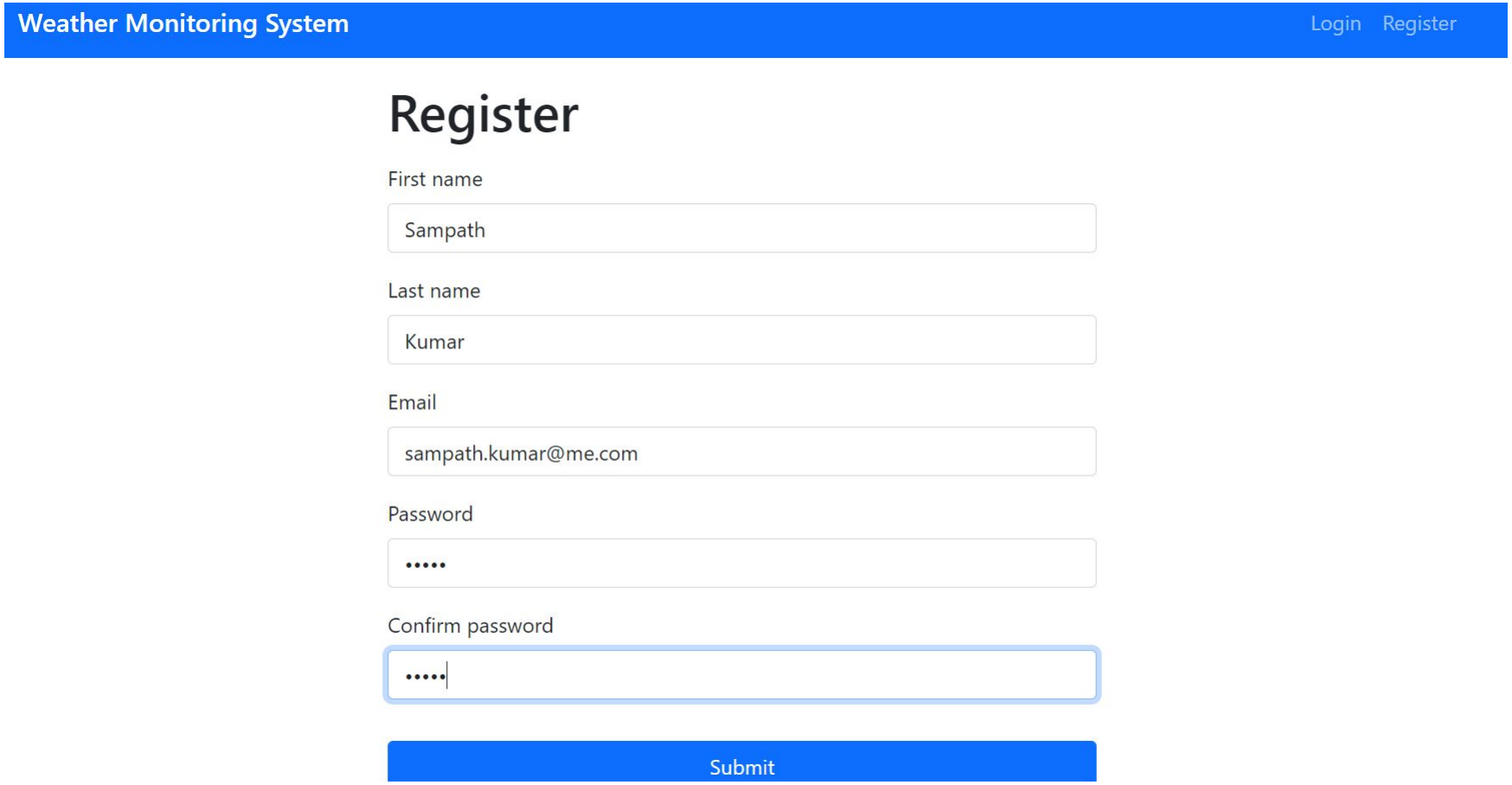
- We also implemented:
  - Role-based different UI for user and admin
  - Allowing admin to update the state of the weather station
  - Email notification service
  - Cron job for sending the interval notifications to users
  - Alarm notification service

Final state of the system - Software



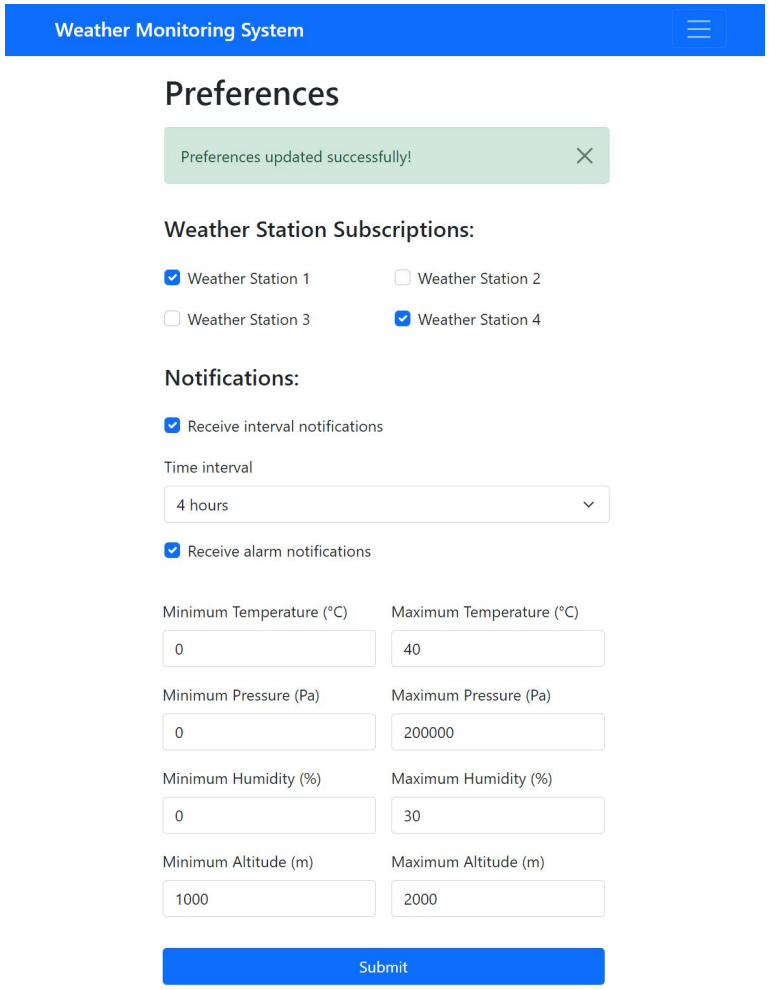
Login page

Final state of the system - Software



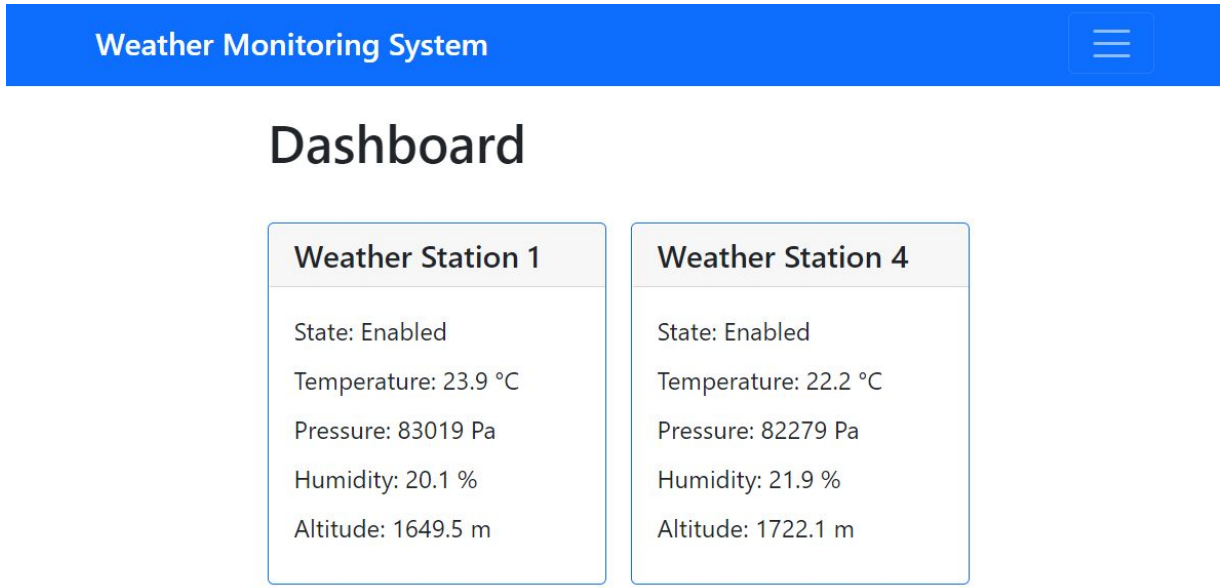
User registration page

Final state of the system - Software



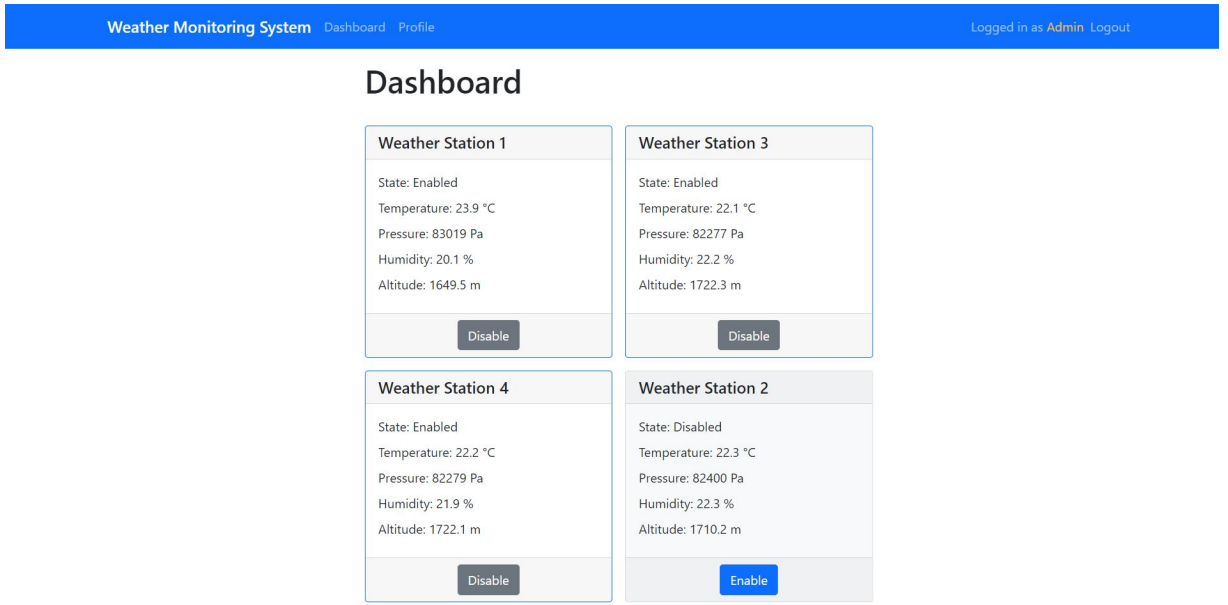
User Preferences page

Final state of the system - Software



User Dashboard

Final state of the system - Software

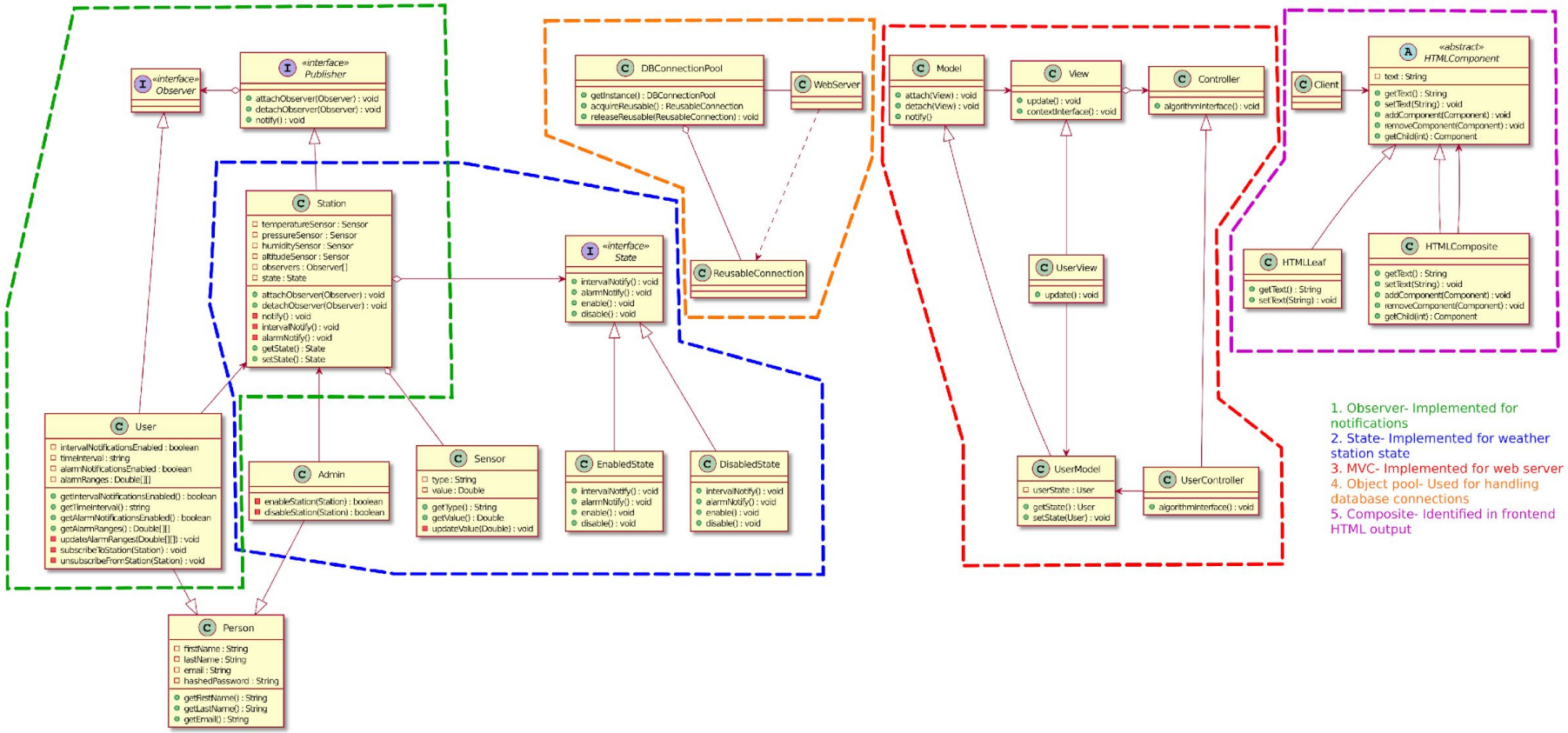


Admin Dashboard

Changes/Issues encountered

- Originally, we were going to have a single registration page where the user enters all the registration details and the preferences data.
- But we realized that if the registration details have an error like invalid email or password, then the preferences data entered by the user will be a waste of time as they would have to fill it all over again.
- So, we simplified the registration page only to include the registration details. Once the user is registered, after logging in, if they have not set the preferences, they are automatically taken to the preferences page first instead of the dashboard.

Class diagram (Annotated)



Patterns implemented

Observer: The User class implements the Observer interface. The Station class implements the Publisher interface and stores references to the Observer interface and implements attachObserver, detachObserver and notify methods. The station class has two other notify methods named intervalNotify and alarmNotify, which internally call the same notify method.

State: The Station class stores a reference to the State interface which stores its state. Concrete classes EnabledState and DisabledState implement the State interface and override the intervalNotify, alarmNotify, enable and disable methods.

Object Pool: We used Node.js for the web server and PostgreSQL for the database. The WebServer gets an instance of the DBConnectionPool using the getInstance method. It gets an instance of ReusableConnection using the acquireReusable method and later releases it using the releaseReusable method.

MVC: We used Node.js for the web server along with the Express.js framework. We applied the MVC pattern where user interacts with the View, Controller changes the Model state and renders the View and Model updates the View. For example, a webpage for displaying user information involves UserModel to store the user information, UserView to display it and UserController to update the state and render the UserView.

Composite: We used HTML to display the web UI. The DOM tree in HTML is a good example of the Composite pattern. Each element in the DOM tree implements the HTMLComponent interface. An element can either be a HTMLComposite, which further contains more elements, or it could be an HTMLLeaf, the leaf of that tree node. Both HTMLLeaf and HTMLComposite implement the HTMLComponent interface.

Key changes to the system since Projects 5 and 6

- Apart from the registration flow change that we mentioned in project 6 report, there haven’t been any changes in the class diagram as we had designed the flows and the UML diagrams by thinking through the use-cases well.

Third party code vs original code statement

We used the following sources:

1. Backend Web Server Engine: Node.js- https://nodejs.org/en/
2. Backend Web Framework: Express.js- https://expressjs.com/
3. Relational Database: PostgreSQL- https://www.postgresql.org/
4. Package for connecting with PG database with connection pool: node-postgres- https://www.npmjs.com/package/pg
5. Package for environment variables: dotenv- https://www.npmjs.com/package/dotenv
6. Package for sending emails: Nodemailer- https://nodemailer.com/about/
7. Package for JSON Web Tokens: jsonwebtoken- https://www.npmjs.com/package/jsonwebtoken
8. Package for scheduling cron jobs: Node Cron- https://www.npmjs.com/package/node-cron
9. Package for rending template views in MVC: EJS- https://ejs.co/
10. Package for validating input: express-validator- https://express-validator.github.io/docs/
11. Package for hashing password cryptographically: bcrypt.js- https://www.npmjs.com/package/bcrypt
12. Package for DHT22 sensor: https://github.com/adafruit/Adafruit_Python_DHT
13. Package for BMP180 sensor: https://github.com/BoschSensortec/BME280_driver

Third party code vs original code statement

We have referred to the following tutorials/articles for implementing the project-

1. https://geshan.com.np/blog/2021/01/nodejs-postgresql-tutorial/
2. https://www.digitalocean.com/community/tutorials/how-to-use-ejs-to-template-your-node-application
3. https://www.geeksforgeeks.org/working-with-forms-using-express-js-in-node-js/
4. https://www.section.io/engineering-education/how-to-build-authentication-api-with-jwt-token-in-nodejs/
5. https://dev.to/franciscomendes10866/using-cookies-with-jwt-in-node-js-8fn
6. https://dev.to/mr_cea/remaining-stateless-jwt-cookies-in-node-js-3lle
7. Raspberry Pi Humidity Sensor
8. Interfacing with BMP180
9. Build your own weather station

Statement on the OOAD process for the semester project

- The MVC pattern really helped in the separation of concerns. The modularity of the codebase as a result of MVC made it really easy to identify and debug any bugs we encountered.

- The design process before starting the development is often not taken seriously, even in the industry. However, since we had to submit the UML designs of the project in the beginning even before starting the development, it forced us to really think through all the requirements, the scope of the projects, and all the use-cases of the system.

- This solid foundation really helped us during the development as we already had a skeleton/blueprint and we were just building using it as reference. Also, there were hardly any last moment changes/surprises since we were clear on what we're building.

Thank you!

(Please fill up our remaining bonus points if deemed worthy)