Corvallis Hazard Watch Application Documentation

Last Updated 6/2/2024

Table of Contents

1 - API Documentation

6 - Links to Documentation for Outside Resources

7 - Considerations For Project Handoff
- Database Creation / Usage
- Auth0 Starting Point
- An Expansion On The Idea of “Plug and Play Hardware”

10 - Final Thoughts / Things to look into / Ideas for improvements

API Documentation

app.get(‘/’)

Description
Basic endpoint that will return REACT html.

Requirements
None

Example Query
{appurl}/

Example Response

React HTML

app.get(‘/api/Hazards’)

Description
Gets all hazards as well as all their data from Supabase and returns them in json format.

Requirements
- Working Supabase URL and Key fed to API
- No body requirements

Example Query
appurl/api/Hazards

Example Response


```
{
    "data": [
        {
            "id": 182,
            "created_at": "2024-05-17T20:11:59.454297+00:00",
            "type": 0,
            "latitude": 44.94538185033746,
            "longitude": -123.0723398923874,
            "text": "",
            "creator_id": 10,
            "image": null,
            "icon_type": null,
            "radius": 24,
            "location": "Salem,Oregon, US"
        },
{a bunch more hazards}
    ]
}
```

app.get(‘/api/addHazard’)

Description
Gets all hazards as well as all their data from Supabase and returns them in json format.

Requirements
- Working Supabase URL and Key
- Allowed Hazard Body Inputs
    - Type, Latitude, Longitude, Text, creator_id
- The user must be logged in to post a hazard

Example Query
appurl/api/addHazards

Example Response

If good: No body
If bad: Error 500: Internal Server Error

app.put(‘/api/sensor’)

Description
Updates a sensor’s data point in the database based on a unique sensor_name value.

Requirements
- Working Supabase URL and Key
- Required body elements of
    - sensor_name that exists
    - sensor_status within 0 and 3

Example Query
appurl/api/sensor

Example Response

If good:
Code 200:
```
{
    "message": "Sensor status updated"
}
```
Code 201:
```
{
    "message": "Sensor not in need of update"
}
```

If bad:
Error 400:
```
{
    "message": "sensor_status is only allowed to be 0-3"
}
```
Error 401:
```
{
    "message": "Missing required fields"
}
```
Error 500: Internal Server Error

app.get(‘/api/getsensor’)

Description
Gets the current sensor data for all sensors in the sensor table.

Requirements
- Working Supabase URL and Key
- Required body elements of
    - sensor_name that exists
    - sensor_status within 0 and 3

Example Query
appurl/api/getsensor

Example Response

If good:
Code 200:


```
{
    "data": [
        {
            "req_id": 7,
            "last_updated": "2024-05-29T21:05:18.662+00:00",
            "latitude": 44.94538185033746,
            "longitude": -123.0723398923874,
            "sensor_name": "test",
            "sensor_status": 0
        },
        {
            "req_id": 3,
            "last_updated": "2024-06-02T21:03:33.941+00:00",
            "latitude": 96.57200488893162,
            "longitude": -446.2820510864258,
            "sensor_name": "sensor1",
            "sensor_status": 1
        }
    ]
}
```


If bad:
Error 500: Internal Server Error

Links To Outside Resources Documentation

Leaflet:

Leaflet is used to power the map visualization. It is an open-source JS library for creating maps.

Link:
https://leafletjs.com/reference.html

Supabase:

We used Supabase to store all of the crowd-reported hazards, and our ECE counterpart used it to store the data being uploaded by their sensor.

Link:
https://supabase.com/docs/guides/api
https://supabase.com/dashboard/project/tkmthnhmpgonqiwlgjvu/api

Auth0

Auth0 was used as a way to manage Usernames and Passwords for users without having to store them ourselves. This was done for security purposes.

Link:
Overall: https://auth0.com/docs
JS Specific: https://auth0.com/docs/quickstart/webapp/express/interactive

Considerations For Project Handoff

This page was made in the event that this project is handed off to another group. We wanted to make this so we could give you some things to keep in mind so that you aren't running blind into this project.

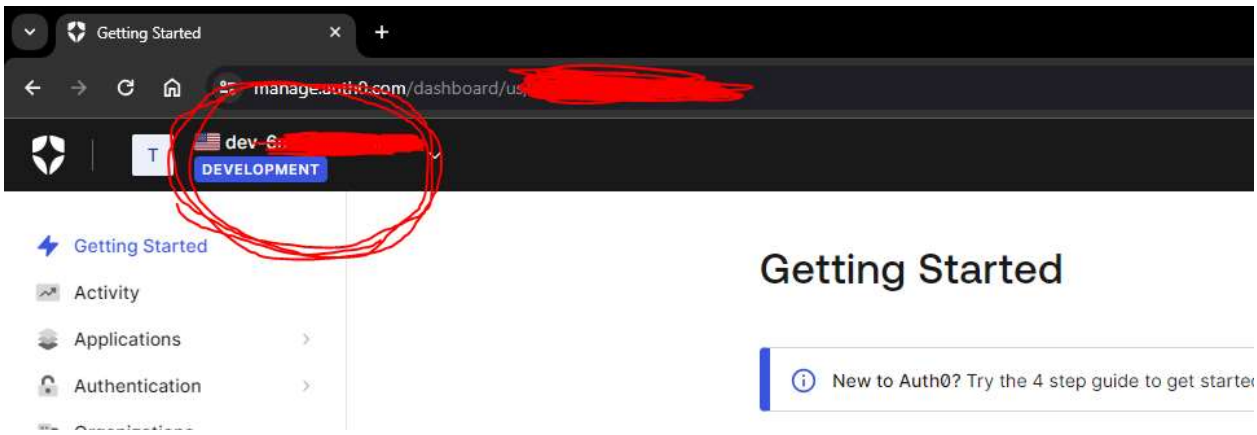
Auth0 and Supabase

The Auth0 and Supabase accounts we originally used for this project were either ONID based or completely personal emails. You will need to set up a new Auth0 account, as well as a new Supabase account to be able to run this project in the way that we did. We talked over the idea of creating a project-specific email that we could handoff, but did not execute upon this idea.

Auth0 Starting Point

In the event that you have not worked with Auth0 (curriculum for CS493 Cloud App had already changed at time of writing), here is a very basic set of instructions to get things running properly. **We would highly recommend reading the provided documentation for Auth0 if you run into any issues as things are always subject to change with 3rd party applications.**

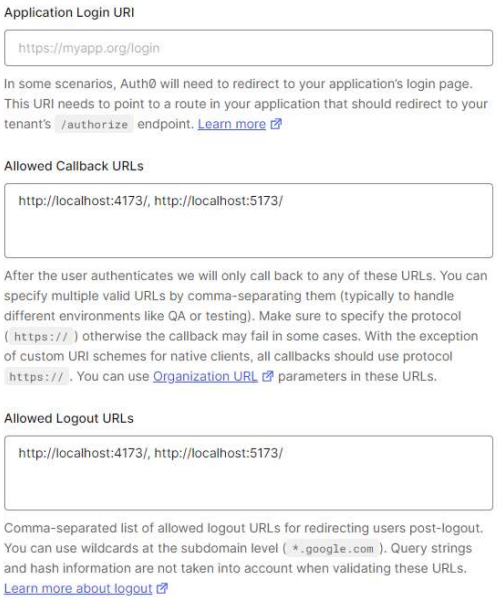
When you create an auth0 account, it will take place under a **Domain**. This domain can be seen in the top left of your screen when you are looking at the auth0 webpage, it is usually auto-generated if you have a free account.



To start, you will want to go to the **Applications** tab on the left, and create a new application for this capstone project. Choose "Regular Web Applications". For the type, we used Express.js.

If you click on the “I want to integrate my app” selection, it will give you some help as well as some example setup code for using Auth0 with Express.js.

An important section to become familiar with inside your App Settings section is the Application URI’s section seen here:



Make note of the login URL in case you made it something different inside of the API.

“Callback” url’s are where Auth0 will send the user once they have logged in using Auth0’s 3rd party login page.

“Logout” is the same idea, but for when the user logs out of the app.

To set up an Auth0 database for passwords, go “Authentication -> Database” and create a DB connection. Inside the settings for this you can create requirements such as username length, etc.

Now, back inside of your app settings in Application->Applications->[your app name], make sure that you go to “Connections” and assure that the database you want is being used for this project.

Down in the “Advanced Settings” dropdown inside of the App settings, go to “Grant Types” and make sure that “Password” is checked.

Now, inside of User Management->Users, you should be able to create new users with a username and password that will be stored inside of your Auth0 database. If you have properly added your domain and Client ID to the API, then you should be able to go to the login page and login as users you have created here. You should also be able to create a new user on the 3rd party Auth0 page, and it will show up as a user here.

Hopefully this gets your Auth0 dependencies up and running.

An Expansion Upon The Idea of “Plug and Play Hardware”

For our project, we focused entirely upon getting the website to take basic shape, as well as for the map and icon features to be working with as few bugs as possible. In the project description, you may have read some wording along the lines of “Plug and Play Hardware”. This section is intended to expand upon what this idea means.

Our project featured a water sensor that an ECE team was responsible for building. The data from this sensor is transmitted over LoRa low-frequency radio, and is stored in an independent Supabase database.

A core idea that was discussed throughout the duration of our project was the idea that you could create an appropriate database for any kind of sensor that can transmit data, and then in the HazardWatch app you could add a new icon that would represent a place where users could go to check on this data. This idea was something that we weren’t able to completely see through, as we only had the one sensor type to work with. If the project handoff has been handled properly, then you should be working with another ECE team, or possibly multiple ECE teams, to fulfill this vision of Plug and Play hardware using different types of sensors.

Some ideas we came up with for other sensor types:

- Bridge pressure sensor
- Temperature sensors
- Earthquake sensors
- Humidity sensors
- UV Index sensors
- Wildfire sensors
- Gas leak sensors
- Traffic Density sensors
- Integration with Blue Light Emergency phones
    - (slide 9 of this pdf)
- Air Quality Index sensors

Of these ideas, many of them are already tracked by some existing online resources, so you could look into how to integrate these resources as well.

Final Thoughts / Things to look into / Ideas for improvements

Lastly, we wanted to include some final thoughts on where we think the app should go, as well as some ideas we didn't get around to implementing within our project's scope.

First and foremost, is security. The app as it stands is using the 3rd party Auth0 software to handle user login, as well as Supabase for all of its data. The issue is, as we ran our app we just set our Supabase tables to be virtually open for any anonymous user to access or make changes to, as long as they had the correct domain and key. We would recommend looking into locking this down a bit more, so that the app can take on a much more "public" identity and image. Changing a lot of the hazard posting logic to be server-side would be a great place to start. This includes the logic that governs the profanity filter as well.

Second, we cannot stress enough how good of a project it would be to port this whole thing as seamlessly as possible to mobile. Look into this as early as you can if you end up wanting to do it.

Lastly, since it's possible new sensors just don't get made by an ECE team, and that you might not work with anyone in the same regard we did this term, you can always just create different fake sensor types or try and plug outside data streams into the webapp as well. **Other non-hardware-dependent changes could include** just completely redesigning the app from the ground up. Your perspective of being able to bounce off what we created makes it much easier for you to correct our mistakes. By this we mean you could implement ideas such as pagination, and any other scalability features (we know it fetches every hazard every time). The codebase could stand for a potential ground-up redesign (hindsight is always 20/20).