Software Design Specification

for

W A T E R R A

Version 1.0 approved

Prepared by

Group 15: Cynthia (Yingxue) Liu, Jennie (Jian) Li, Zoe (Huaijin) Ning

McMaster University

November 25, 2022

Table of Content

Revision History

Version	Name	Reason For Changes	Date
1.0	*Cynthia Jennie Zoe*	• *Add two more APIs for the system* • *Change some API design* • *Edit some sentences in user interface design section to be more specific*	*November 25*

1. Introduction

1.1 Purpose

This document contains the high-level requirements for the water data visualization of the Water Sensor website (later abbreviated as *the WS*[1]), and water data integration of *Terrastories*[2](later abbreviated as *the TI*[3]) presented at *COP27*[4]. The technical specifications for this project have been drafted following several meetings between the *Ohneganos*[5] development team. This *Waterra*[6] project will both integrate and implement the data collection of water quality sensors that enables Indigenous and Secondary Education communities to interact with the *Ohneganos* project using a computer or remote interface.

1.2 System Overview

This project extends the functionality of the data visualization part that is currently active on the Water Sensor website.

The WS should use the data from the time-series database and improve the *nodes*[7] by selecting a function and current data visualization. Currently, users can select the *nodes* and then the *probe*[8], then the initial data visualization graph would be generated below the map.

The TI is a follow-on development of the existing indigenous community. Currently, the website displays video/audio and literal stories for each marker on the map, our project aims to add the water quality data to the markers as well. That is, the water quality data would be integrated from the time-series database provided by *the Ohneganos*, and *the Terrastories* team would do the implementation to display it together with other contents.

2. Design Considerations

2.1 Assumptions

It is assumed that our users have normal or corrected-to-normal vision, hearing, and common sense, and be proficient in at least one language that is supported by *the WS* and *the TI* website. We also assume that users use the latest several versions of mainstream browsers on either desktops or mobile devices to make sure both of our projects are adapted properly.

2.2 Constraints

The TI:

- Since we are only integrating the data into the tool used by the *Terrastories* team, we cannot create unique display formats and make exact changes on the map.
- All the content integrated must fit the frames and paradigms of *Terrastories*.
- This website is aimed at indigenous people and high school students, but it is also universal and educational. As a result, data presentation should not be overly specialized and difficult to understand.

The WS:

- We can not use other technical tools, i.e. new programming languages or frameworks, as we need to follow the existing format of the website.
- The website software must be maintainable and stable so that other developers can easily understand and continue to develop.

2.3 System Environment

- Both parts of the project require a browser and an operating system.
- The browser should be one of the several recent versions of *Google Chrome*, *Firefox*, *MS Edge*, and *Safari*.
- Operating system: *Windows 7*+, *Mac OS X 10.7*+, or *Ubuntu*[9] *10*+.
- *The WS* should work well on both phone and desktop versions of the above browsers and must have an internet connection. The ideal internet connection is around 0.3+ Mbps.
- *The TI* should be able to work online and offline, so an internet connection is unnecessary.
- *The TI* must have *Docker* installed on desktop devices to develop. Mobile device users can view websites without *Docker*. For *Windows* systems, *WSL*[10] 2.0 or virtualization supported by *Hyper-V*[11] is required to work.

3. Architecture

3.1 Overview

This project is made up of three components, the front end using *React*, the back end using *Go*, and the database using *InfluxDB*. Each component is developed and maintained as independent modules on separate platforms. The front end presents the user interface that users directly interact with. When users take actions, the actions would be translated to the backend using API for the functionality that drives the behavior of the application. The data for *probes* are stored in two tables in *InfluxDB*, where the sensor information is recorded by time,

id, name/sensor type, and value. When the data is required, the *MacWater* API on the backend will retrieve the data in *InfluxDB* and provide it to the front end, which will generate the chart.

Data stored in the database is in two similar table formats, one for *nodes*, and one for *probes*. Both of them must have a timestamp column and sort data by time series using *InfluxDB*. In the backend, ten APIs are used to provide a faster and more productive service. Among them are eight “GET” methods for retrieving data from the database and two “POST” methods for receiving data from remote *probes*. Names of the eight “GET” methods are *getAllNodes, getIdNodes, getNodeProbes, getAllProbeData, getProbeNodes, getAllProbes, getProbeLevels,* and *getTimeProbeData* respectively, and all of them will be explained in further detail in the following text. Both of the “POST” methods are hardware related so they will only be mentioned briefly later. With the data from “GET” methods, the front end can make use of *Leaflet* to generate a functional map and *Chart.js v3* to draw line graphs. The dashboard of the website consists of four components — Map, Real-time, About, and Settings, where users can switch between them using a button in the sidebar.

3.2 Design Rationale

- The website should display a line chart for the same *node*, and same *probe* at different time periods. We choose to do this in the backend because this would be a more efficient way in comparison to doing it in the front end. In this case, we decide to make an API (4.2.8) that input *node*, *probe*, start time, and end time, and return the data within that period
- The website should display a different color for each *node* after selecting a *probe*. The multiple colors should represent different levels of data. We decide to do this comparison in the backend, as the front end does not need to know the exact data but only the level that indicates color. This is done by an API mentioned later in 4.2.7, which takes *probes* types as input and returns an array containing levels so that it could be better maintained in the future.

4. Database Schema

4.1 Data Description

MacWater, a time-series database using *InfluxDB* format is used to save both the sensor and the *node* data. Time series databases *InfluxDB* with small footprints and simple data storage are designed with the assumption that insertions are more frequent than queries, allowing for the rapid insertion of large amounts of data, such as water quality data. The database contains two tables, the *sensorInfo* table, and the *sensorMeasurements* table.
The *sensorInfo* table has four columns: time, id, name, and value. The value represents the coordinates of the *node* at the time. The id, name, and value are all stored as strings.

The *sensorMeasurements* table has four columns: time, id, sensortype, and value. The value is the measurement of the sensortype from the *node* at the time. The id and sensortype are stored as strings. The value is stored as a float.

4.2 Data Dictionary

1. *getAllNodes*: a GET method that returns all *nodes* information
    - Parameters: null
    - Return:
        - locations (Array): id (id: String), name (name: String), latest location (value of *sensorInfo* table: String), latest location update timestamp (time of *sensorInfo* table: String).
          Eg.

          ```
          {
            "locations": [
              [
                "2",
                "End_Node_2",
                "43.268383,-79.920265",
                "1663894193000000000"
              ],
              [
                "5",
                "End_Node_5",
                "43.268330,-79.920113",
                "1663180712000000000"
              ]
            ]
          }
          ```

2. *getIdNodes*: a GET method that gets *node* location data trace by *node* id
    - Parameters:
        - id (id: Number).
          Eg.
          ```
          id: 2
          ```
    - Return:
        - name (name: String),
        - locations (Array): time (time of *sensorInfo* table: String), value (value of *sensorInfo* table: String)
          Eg.

          ```
          {
            "name": "End_Node_2",
            "locations": [
              [
          ```


```
            "1663179116000000000",
            "43.268257,-79.920189"
          ],
          [
            "1663179183000000000",
            "43.268261,-79.920143"
          ]
        ]}
```


3. *getNodeProbes*: a GET method that gets sensor *probes* type by *node* id
    - Parameters:
        - id (id: Number)
    - Return:
        - types (Array): type (sensortype of *sensorMeasurements* table: String)
4. *getAllProbeData*: a GET method that gets sensor *probe* data by *node* id and type
    - Parameters:
        - id (id: Number)
        - type (sensortype of *sensorMeasurements* table: String)
    - Return:
        - id (id: Number),
        - type (sensortype of *sensorMeasurements* table: String),
        - data (Array): time (time of *sensorMeasurements* table: String), value (value of *sensorMeasurements* table: String)
5. *getProbeNodes*: a GET method that gets *node* information by sensor *probe* type
    - Parameters:
        - type (sensortype of *sensorMeasurements* table: String)
    - Return:
        - locations (Array) : id (id: String), name (name: String), latest location (value of *sensorInfo* table: String), latest location update timestamp (time of *sensorMeasurements* table: String)
6. *getAllProbes*: a GET method that gets all sensor *probe* types of all *nodes*
    - Parameters: null
    - Return:
        - types (Array) : type (sensortype of *sensorMeasurements* table: String)
7. *getProbeLevels*: a GET method that gets all data levels by sensor *probes* types
    - Parameters:
        - type (sensortype of *sensorMeasurements* table: String)
    - Return:
        - type (sensortype of *sensorMeasurements* table: String)
        - data (Array): id (id: String), name (name: String), latest location (value of *sensorInfo* table: String), latest location update timestamp (time of *sensorMeasurements* table: String), value (value of *sensorMeasurements* table: String), level (level: Number)

8. *getTimeProbeData*: a GET method that gets sensor *probe* data by *node* id, type, and time period
    - Parameters:
        - id (id: Number),
        - type (sensortype of *sensorMeasurements* table: String)
        - start time (time: Number)
        - end time (time: Number)
    - Return:
        - id (id: Number)
        - type (sensortype of *sensorMeasurements* table: String)
        - data (Array): time (time of *sensorMeasurements* table: String), value (value of *sensorMeasurements* table: String)
9. POST Create *nodes* information (post a line of the doc on *sensorInfo* table)
10. POST Create sensor *probe* data (post a line of the doc on the *sensorMeasurements* table)

Data is detected from water quality sensors. For this *Waterra* project, we do not deal with data collection.

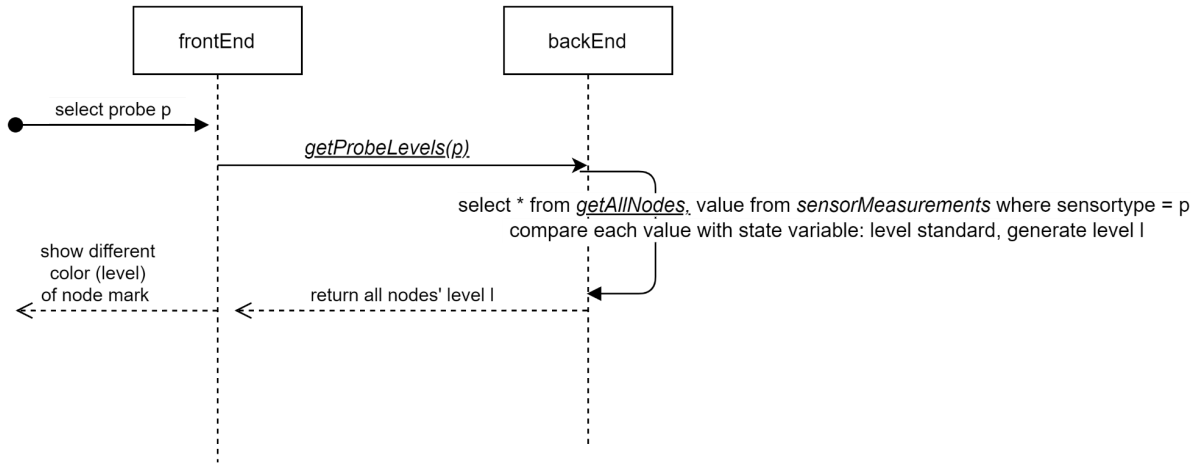
Table Name	**Field Name**	**Data Type**	**Allow Nulls**	**Field Description**
sensorInfo	time	bigint(20)	No	• Get this field from Create *nodes* info POST method • Latest location update timestamp
sensorInfo	id	varchar(20)	No	• Get this field from Create *nodes* info POST method • Unique id for the *node*
sensorInfo	name	varchar(50)	No	• Get this field from Create *nodes* info POST method • *Node* name
sensorInfo	value	varchar(50)	No	• Get this field from Create *nodes* info POST method • The coordinates of the *node* location

sensorMeasurements	time	bigint(20)	No	• Get this field from Create sensor *probe* data POST method • Latest sensor update timestamp
sensorMeasurements	id	varchar(20)	No	• Get this field from Create sensor *probe* data POST method • Same id for the *node* of *sensorInfo* table
sensorMeasurements	sensortype	varchar(50)	No	• Get this field from Create sensor *probe* data POST method • Sensor *probe* type
sensorMeasurements	value	float	No	• Get this field from Create sensor *probe* data POST method • value detected by the sensor

5. Component Design

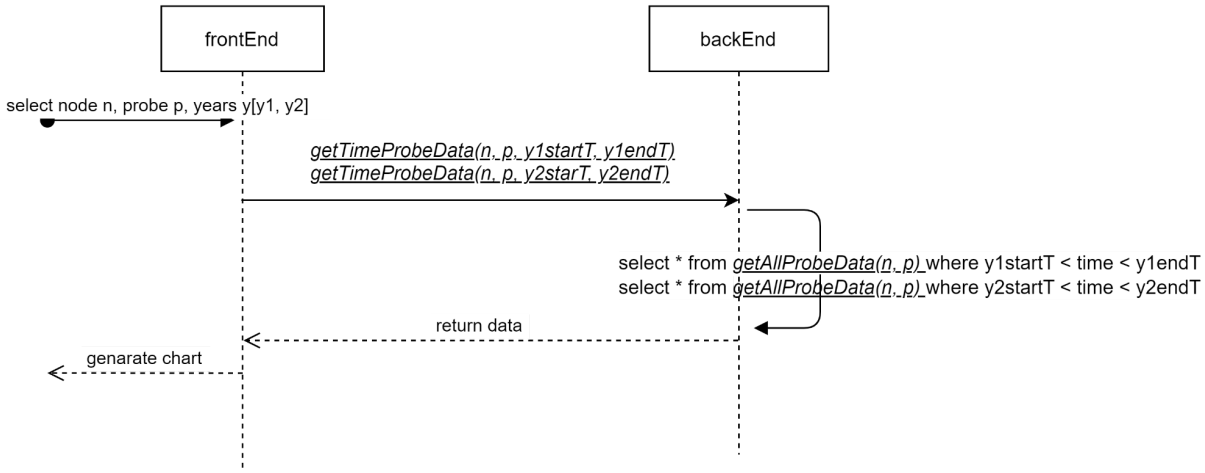
- **Colors can be used to differentiate data levels.**

Users will be able to see the data level of a selected *probe* type on t*he WS* website's initial map page by seeing different colors of *node* marks after selecting a *probe* type.



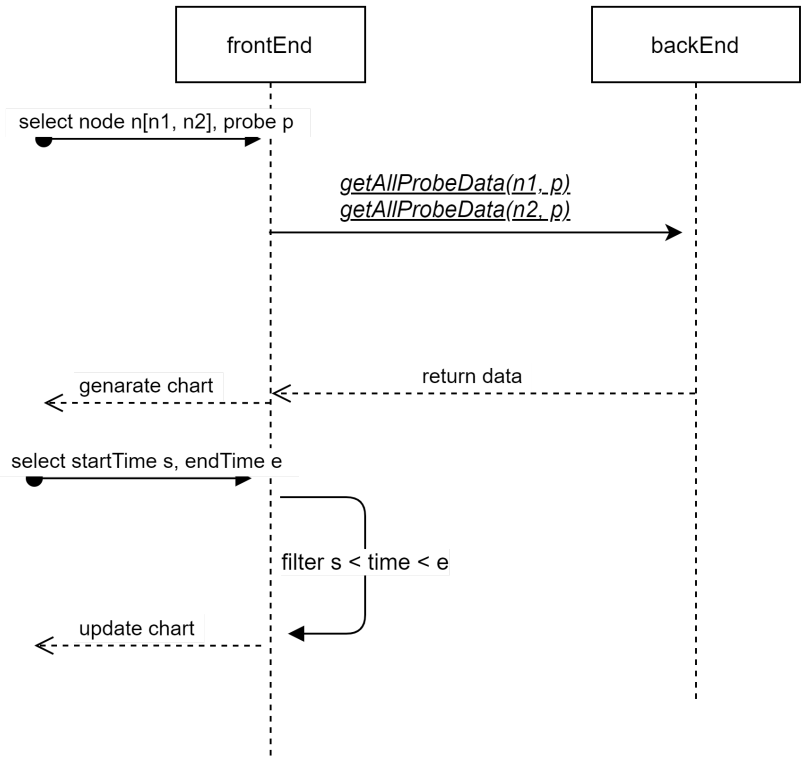
- **Annual/monthly/weekly data of fixed *nodes* and *probe* types are compared.**

By selecting a *node*, a *probe* type, and multiple time periods, users will be able to compare yearly/monthly/weekly data on an overlapping line chart.



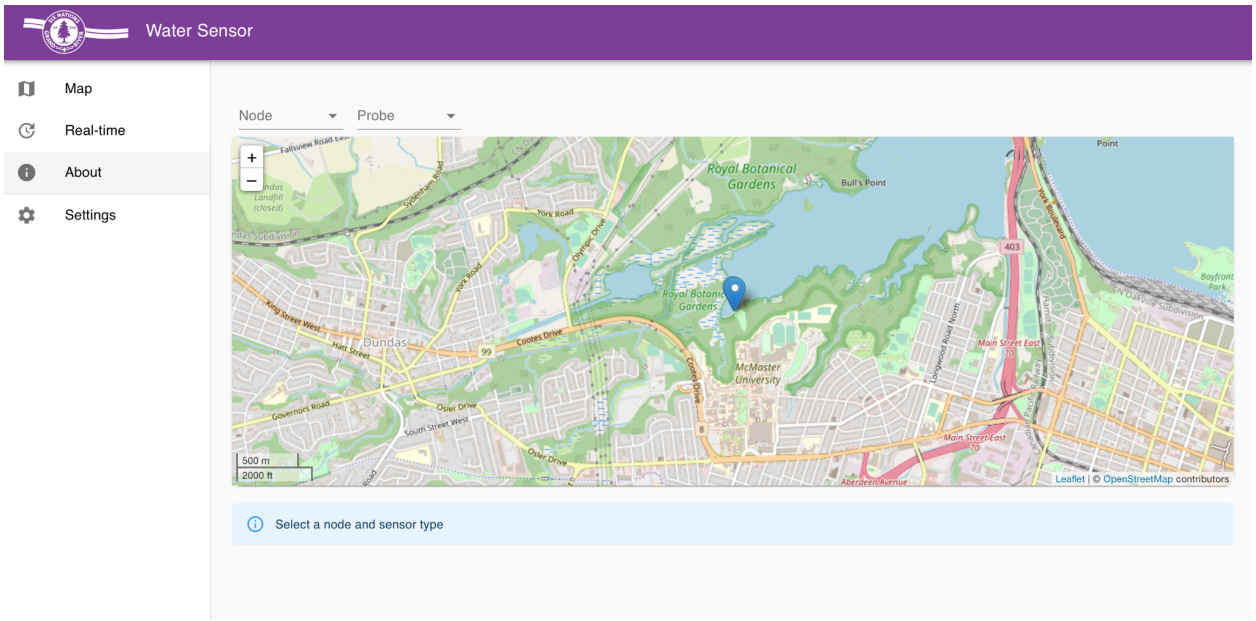
- **Compare specific *probe* type data from different *nodes* and choose a time period.**

By selecting multiple *nodes*, and a *probe* type, users will be able to compare a specific *probe* type data of different *nodes* on an overlapping line chart. An additional time filter action is available by selecting the start and end times.



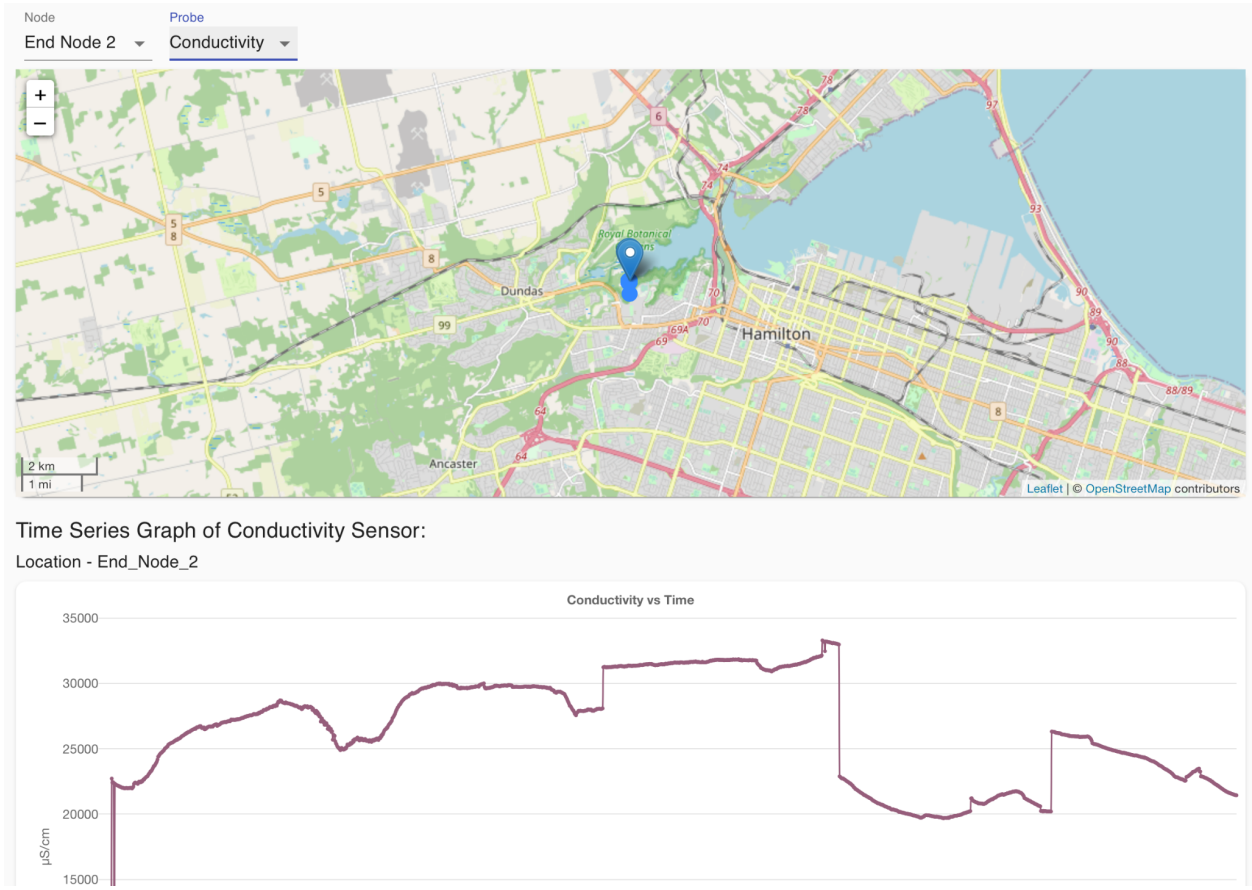
6. User Interface Design

The overall page layout of *the WS* consists of three parts: the title bar, the sidebar, and the main section.

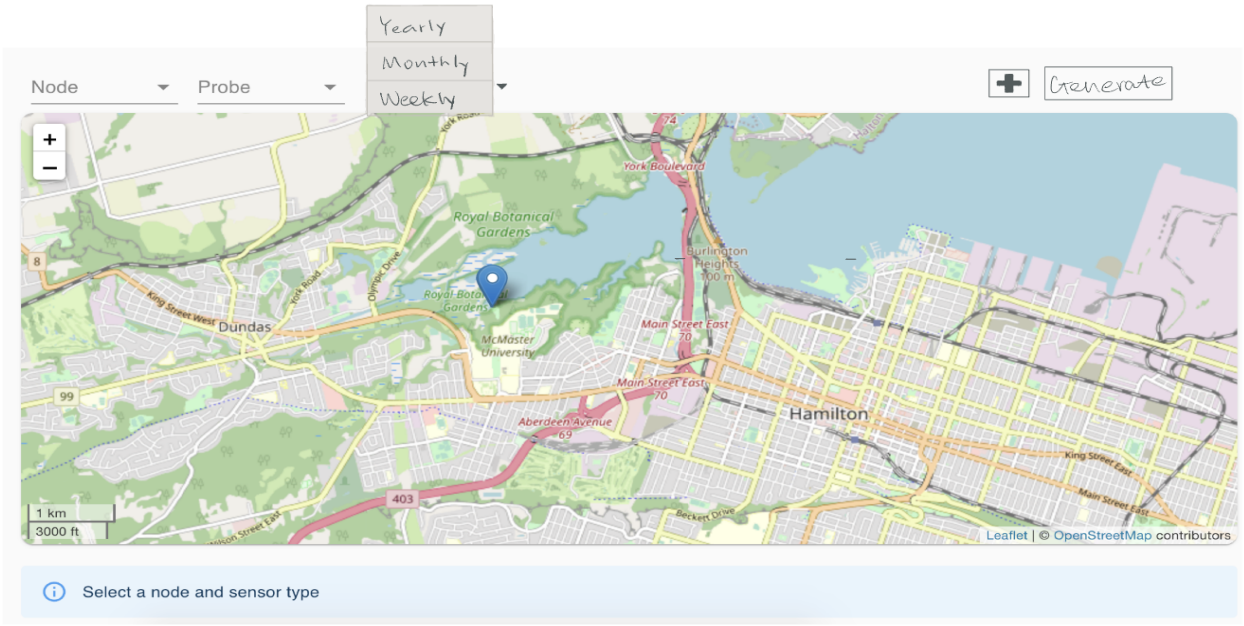


The title bar displays the name of the website and the program logo with a background in the theme color McMaster maroon.
The sidebar has four buttons corresponding to four pages, and users can switch between the four pages through the buttons.
The main section provides most of the information. As we are in charge of the map page only, details about other pages will not be mentioned below. The main section of the map page includes a map and a data analysis area.
The map is fully functional using *OpenStreetMap* provided by *Leaflet*, with markers at all sensor locations, two zoom in/out buttons, and a scale in the corner.
The data analysis area contains several drop-down lists and buttons that allow users to manually select *nodes*, *probes*, and data time periods. After selecting a *probe*, the map will display the corresponding selection by changing the color of all *node* markers. The color of each marker is determined according to the latest updated data of the *node*.

When users choose a *node* and a *probe*, clicking the “Generate” button gives them a line graph of the corresponding selection generated with all related data.

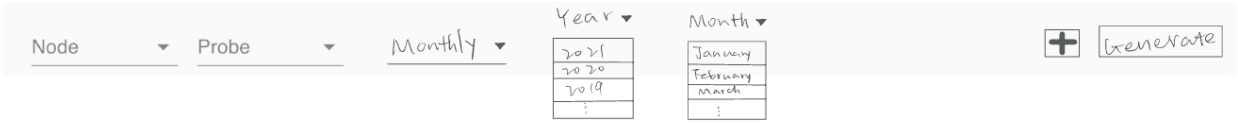


When users choose a *node*, or a *probe* and also select one of the yearly/monthly/weekly options, more drop-down lists will show up to allow them to choose the specific period. Then, click the “Generate” button to get a line graph of the corresponding selection in the chosen period.

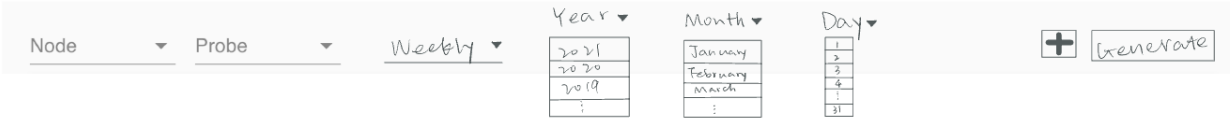


For example, yearly: one more dropdown to choose a year.

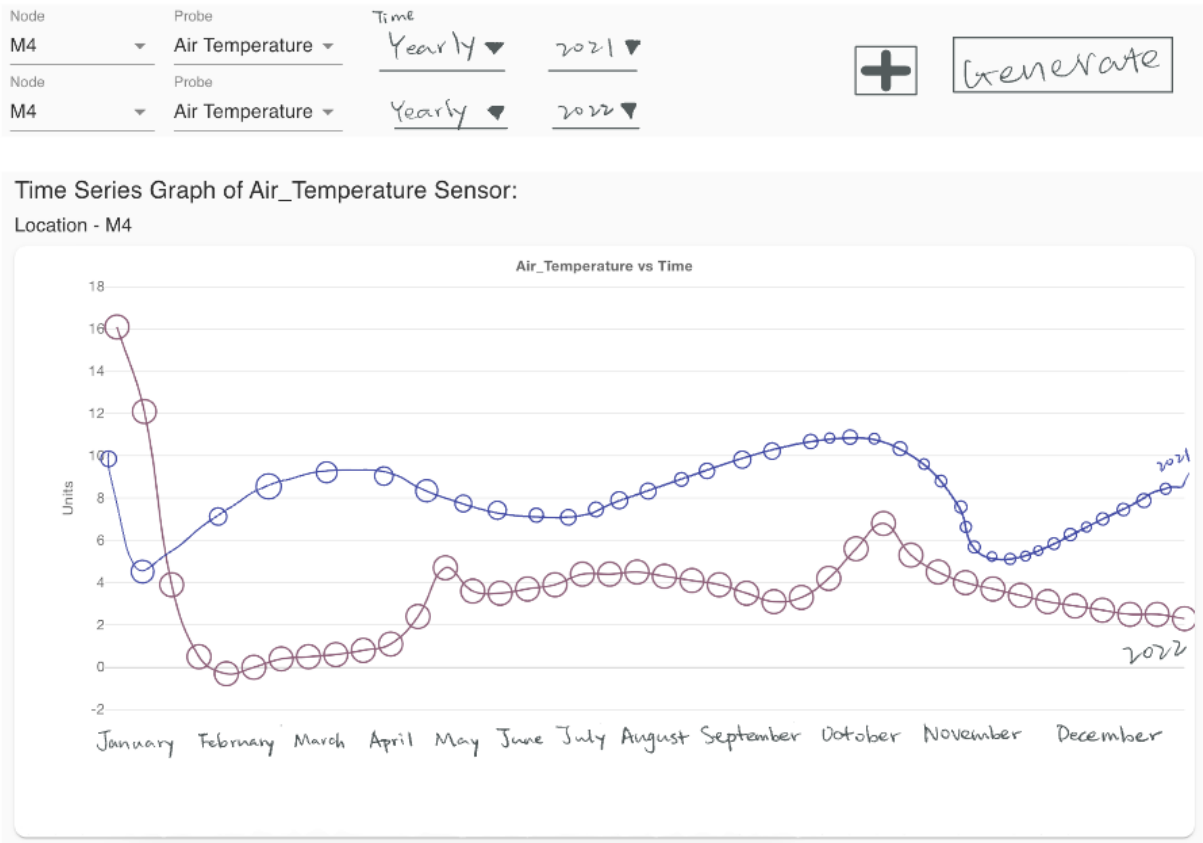
Monthly: two more dropdowns to choose a month in a specific year.



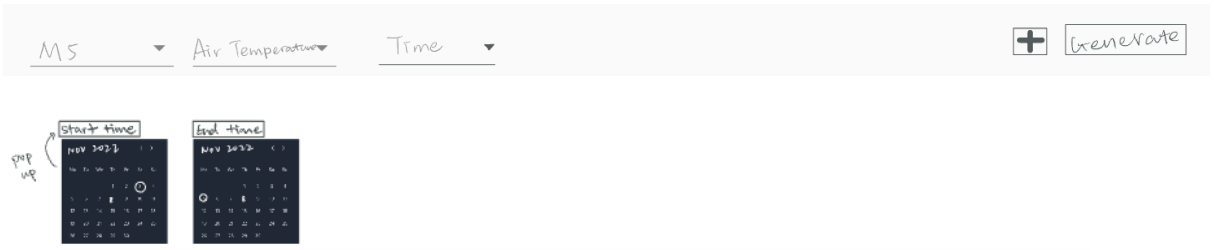
Weekly: three more dropdowns to choose a specific day, and a graph of data for the next seven days from that selected day will be presented.



The website can also generate overlapping graphs for comparison. After selecting a *node*, a *probe*, and a time interval, users can click the plus sign on the right to select another time interval of the same *node* and *probe*. Up to five selections can be made for one overlapping graph. When finishing selecting, click the “Generate” button to view the graph. Note that time intervals for all selections must be the same size, and the *node* and *probe* selected must also be exactly the same.



Displaying the graph of a certain period only is also supported. When a graph is generated, users can arbitrarily choose the start time and end time, then regenerate the graph to view the chosen interval only.


M5
Air Temperature
Time
Generate
Start time
End time
Time Series Graph of Air_Temperature Sensor:
Location - M5
Air_Temperature vs Time
Units
Time Series Graph of Air_Temperature Sensor:
Location - M5
Air_Temperature vs Time
Units
(multiple timestamp)

Appendix: Glossary

1. *The WS*: refers to the water sensor data visualization part of our project.
2. *Terrastories*: an open-source application built to enable local communities to locate and map their oral storytelling traditions about places of significant meaning or value to them.
3. *The TI*: refers to the *Terrastories* integration part of our project.
4. *COP27*: United Nations Climate Change Conference held in November 2022
5. *Ohneganos*: an Indigenous water research program led by McMaster University Professor, Dr. Dawn Martin Hill. The word "*Ohneganos*" means water in Iroquoian Language.
6. *Waterra:* the project name, explained as the water of the earth.
7. *Node*: locations that have installed the water sensor gateway.
8. *Probe*: the water sensor of each *node* includes multiple *probes* which may include an air temperature *probe*, conductivity *probe*, dissolved oxygen *probe*, humidity *probe*, PH *probe*, turbidity *probe*, water temperature *probe*, etc.
9. *Ubuntu*: a Linux distribution based on Debian and composed mostly of free and open-source software.
10. *WSL 2.0*: a new version of the Windows Subsystem for Linux architecture that powers the Windows Subsystem for Linux to run ELF64 Linux binaries on Windows.
11. *Hyper-V*: a Microsoft hardware virtualization product that lets you create and run a software version of a computer, called a virtual machine.