Updated Notes

for

W A T E R R A

Prepared by

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

McMaster University

March 28, 2023

1. Description

1.1 Introduction

Water quality is a critical issue that affects everyone, yet it can be overwhelming to understand the complexities of this topic. Our project is a sub-project of the *Ohneganos* program, which is an indigenous water research program focused on indigenous and other local communities to help them easily access and understand water quality data.
By integrating and analyzing data from a time-series database and using the tools *InfluxDB*, *Postman*, *FileZilla*, we have created a website that is user-friendly and informative. Our target audience includes both professional users, who may want to perform analyses on these data, and casual users who may have limited knowledge in biology and chemistry. Through easy access to this information, we hope to increase awareness about the importance of water quality and empower individuals to take action in protecting and preserving our water resources.

1.2 Design

The website has a simple yet elegant design, with a title bar that displays the website's name and program logo. The sidebar contains four buttons that correspond to four different pages, Map, Real-time, About, and Settings, allowing users to switch between them with ease. During application, we will be focusing specifically on the map page.
The main section of the map page contains 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 and zoom out buttons, and a scale in the corner. The data analysis menu contains several drop-down lists and buttons that allow users to manually select nodes, probes, and time periods.

1.3 Functions

There are three main functions that we implemented.

The first one is color code markers. After selecting a probe, the color of each marker on the map will change according to the latest updated data of the node. Here we use three different shades of blue to differentiate data levels. For example, a dark blue marker means that the latest data of the node is at a high level. Similarly, a normal blue marker represents middle level, a light blue one represents low level, and a faded gray marker represents the absence of data.

The second one is to generate overlapping graphs for comparison of data trends of the same probe and node in different time periods. Users can select a specific time period in yearly, monthly, or weekly duration, and then generate a line chart of all selections. The website can also generate overlapping graphs for comparison. Users can select two different time intervals

for the same node and probe, however, note that all time intervals must be of the same size (i.e. Both yearly/monthly/weekly), and the node and probe selected must also be exactly the same.

The third function is similar to the second one, which is to generate overlapping graphs for comparison of data trends of different nodes on the same probe, same time period. The probe and time selected must also remain unchanged.

2. Implementation Notes

2.1 Architecture

The entire project is composed of three essential components, the frontend, the backend, and the database.

- The frontend uses *React* to build user interfaces in *TypeScript*. It is responsible for presenting all visual components like HTML elements, and allowing the user to interact with the application and receive real-time updates.
- The backend is designed to manage the data and serve it to the frontend. It implements APIs that interact with the database using *GO* language.
- The database is the central repository where all the data sent by deposited sensors is received, stored, and updated. It consists of two tables in InfluxDB, where the sensor information is recorded by time, id, name/sensor type, and value.

When users take action at the front end, the website generates a URL accordingly and fetches necessary data from the database using APIs for the functionality that drives the behavior of the application. The database is secured and maintained by the backend, ensuring the data is safe and consistent across the entire system.

2.2 Backend Code

All backend code is integrated into one file named main.go. It contains several functions that implement the APIs, constructors of six types that are used in API functions, and a main function to boot up everything.

Every time the Go file is modified, the new version needs to be uploaded to the server and replaces the old one. Then, compile the Go file to create the main executable. To prevent the SSH disconnection from breaking the main execution, a screen session is required in advance.

2.3 Frontend Code

All frontend code is located in the directory */frontend/src/components*. There are nine *TypeScript* files in total, and each of them corresponds to one component in the user interface.

- **Map.tsx**: the functional map in the Map section. It is also in charge of handling State variable changes and fetching and parsing some sensor data.
- **MapDetail.tsx**: all elements on the map, such as markers and marker pop-ups. Markers of different colors and pop-ups containing different information are also generated here.
- **Menu.tsx**: the menu bar and buttons on the right. Handlers of buttons and Props variable changes are included.
- **CustomLayout.tsx**: define a customized button layout for buttons in the menu.
- **SensorTable.tsx**: fetch and prepare sensor data for charts and data tables. Display chart titles with selected nodes and probes.
- **Chart.tsx**: draw line charts using data from SensorTable.tsx.
- **DataCard.tsx**: display all data from SensorTable.tsx in table format.
- **Info.tsx**: display introduction, news, and social media posts of the project.
- **Settings.tsx**: website language selection and sensor configuration toolkit setup.

Once a frontend file is edited, developers need to update the corresponding file in the server. For testing and debugging purposes, local files must be built in the *Yarn* environment to bundle the app into static files for visualization.

2.4 Database

An *InfluxDB* time-series database is used to save both the node and the probe 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 also has four columns: time, id, sensortype, and value. The value is the measurement of the probe from the node at the time. The id, sensortype, and value are stored as strings. Time in both tables are using the Unix timestamp method.

The database is initialized and set up using a helper tool of *InfluxDB* and *Postman*. All fetching URLs from the frontend are converted into SQL commands in backend APIs. Different actions from users form different SQL commands to manipulate and query the database.

3. Test Notes

3.1 Introduction

The notes describe a series of test cases and results for the Water Sensor website that helps users analyze water quality data. It outlines the system's features and functionality, as well as the testing process used to validate its performance and identify any potential issues or areas for improvement. There are three key features that we perform testing on. The test plans of each feature and the rationale behind them will be discussed separately. All of the tests will use fake data we upload to the *influxDB* database through *Postman*. Tests will be mainly implemented as *requirement-based testing.*

3.2 Test Objectives

The test objectives include ensuring the system's ability to generate color-coded node points based on the data level, compare data for fixed nodes and probes under different time periods, and compare different nodes for fixed probe types and times. These objectives aim to validate the system's performance and ensure it provides accurate and reliable data analysis tools to users.

3.3 Test Cases

- **Color indicating data level:**
  The test involves selecting probes like "PH" and "Air Temperature" and observing the color of the data points on the map. The goal is to ensure the system generates colors for data points based on their value and can handle probes with spaces or special characters in their names.
- **Compare different time for fixed node and probe:**
  The test involves selecting different dates for fixed nodes and probes to ensure the system can generate a chart displaying data for the selected node and probe under a specific time period. The test also checks non-functional features, such as comparing data for the same node and probe in different time periods.
- **Compare different nodes for fixed probe type and time:**
  The test involves selecting different nodes to observe data for a fixed probe type and time. The goal is to ensure the system can generate a multi-line graph, even if one node lacks some portion of data.

3.4 Test Results

- **Color indicating data level:**

The testing confirmed the system's ability to generate color-coded node points based on the data level. Deep blue, normal blue, light blue, and faded gray markers represented high, middle, low, and no data levels, respectively. Corrections were made in Map.tsx and MapDetail.tsx to pass all test cases.

- **Compare different time for fixed node and probe:**
  The testing confirmed the system's ability to generate a multi-line graph for fixed nodes and probes under different time periods. Corrections were made in Menu.tsx, SensorTable.tsx, and Chart.tsx to pass all test cases.
- **Compare different nodes for fixed probe type and time:**
  The testing confirmed the system's ability to generate a multi-line graph for different nodes with fixed probe type and time, even if one node has missing data. Corrections made here were made in Menu.tsx, SensorTable.tsx, and Chart.tsx to pass all test cases.

3.5 Conclusion

The test results demonstrate that the Water Sensor website data visualization features are capable of providing reliable and accurate data analysis tools to users. The testing process identified and resolved potential issues, improving the system's reliability and performance.

4. Setup Instructions

4.1 Preparations

- Clone the GitLab repo to a local folder on your device
  - GitLab repository: https://gitlab.cas.mcmaster.ca/lij416/capstone_2022
- Connect to McMaster Campus Wi-fi or use McMaster VPN

4.2 Setup Steps

- SSH remote connection:
  - Command: ssh pi@dev.macwater.local.re-mote.tk -p 2222
  - Username: pi
  - Password: raspberry
  - Go to server program location:
    - cd Desktop/webserver
    - sudo ./main

- If the text “Starting server for testing HTTP GET POST...” is displayed, the backend is set up.
- Go to the frontend folder in your cloned repo and install *Yarn*.
- Execute the “yarn build” command in the frontend folder.
- Then, open the index.html file in the dist folder.