Implementation Document

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

January 30, 2023

Revision History

Version	Name	Reason For Changes	Date
1.0	*Cynthia* *Jennie* *Zoe*	• Add more details to backend and frontend naming convention • Improve function structure graphs	*January 29*

1. Architecture

The entire system of our project is composed of three essential components, frontend, backend, and database.
The frontend is responsible for presenting the user interface, allowing the user to interact with the application and receive real-time updates.
The backend, on the other hand, is designed to manage the data and serve it to the frontend. It implements APIs that interact with the database, fetching the necessary data and processing requests from the frontend.
The database is the central repository where all the data is stored, including user information, application settings, and any other data that the application requires. Data sent by all deposited sensors is also received, stored, and updated by it.
The frontend and backend communicate with each other through APIs, enabling the seamless exchange of data and ensuring the application remains functional. The data is stored in a database which is secured and maintained by the backend, ensuring the data is safe and consistent across the entire system. These three components work together to form a complete and cohesive system, allowing the project to function as intended and delivering a smooth user experience.

2. Implementation Detail

2.1 Languages

The language used for the backend of our project is Go. The backend of our project requires the implementation of three new APIs and several helper functions that enhance the functionality of the backend and improve the overall performance of the system.
The three APIs get the data from the database. The first one gets all nodes information by sensor probe types, especially the type level calculated by the latest data. The second one gets data of a specific sensor probe within a period of time entered by users. The third one modifies the data in the database by SQL commands; sensor id, table name, and target column name are required as input.

For the frontend, we use typescript to do interface design and modify the map layout. React and leaflet are also imported to support map usage.
After selecting a probe, all markers on the map are displayed in different colours according to the latest value of the selected probe. The greater the value, the darker the marker label. As for the markers without this probe data, they will be greyed out.

We stored the data in a time-series database named InfluxDB. The time series databases assume insertions are more frequent than queries, so it allows for the fast insertion of large amounts of data such as water quality data. The SQL statements are used in the backend when achieving data from the database.

2.2 Tools

2.2.1 FileZilla & Winscp

FileZilla and Winscp are free and open-source, cross-platform FTP applications we used to upload our source code to the test environment.

2.2.2 Postman

Postman is an API platform we use to build, test and iterate their APIs. After generating the API link, the information can be clearly entered using Postman, and we use the POST method to put data into the database, and use the GET method to test if our function works appropriately.

2.2.3 Frontend - React leaflet, Yarn

Leaflet is a lightweight, open source mapping library that utilizes OpenStreetMap, a free editable geographic database. We use Leaflet to render our map and markers, and use yarn to configure our code. One benefit for yarn is that it will show errors when configuring.

2.2.4 InfluxDB Workbench

As mentioned above, InfluxDB Workbench is a time-series database for retrieving data. It is used to visualize the records in the database when we are debugging, which helps to identify and fix any issues.

2.2.5 VSCode Live Share

VSCode Live Share is an extension of Visual Studio Code published by Microsoft. It allows us to conduct real-time collaborative development on the project and easily focus on or track the progress of other members when anyone in the group is sharing her workspace.

3. Description of Interfaces

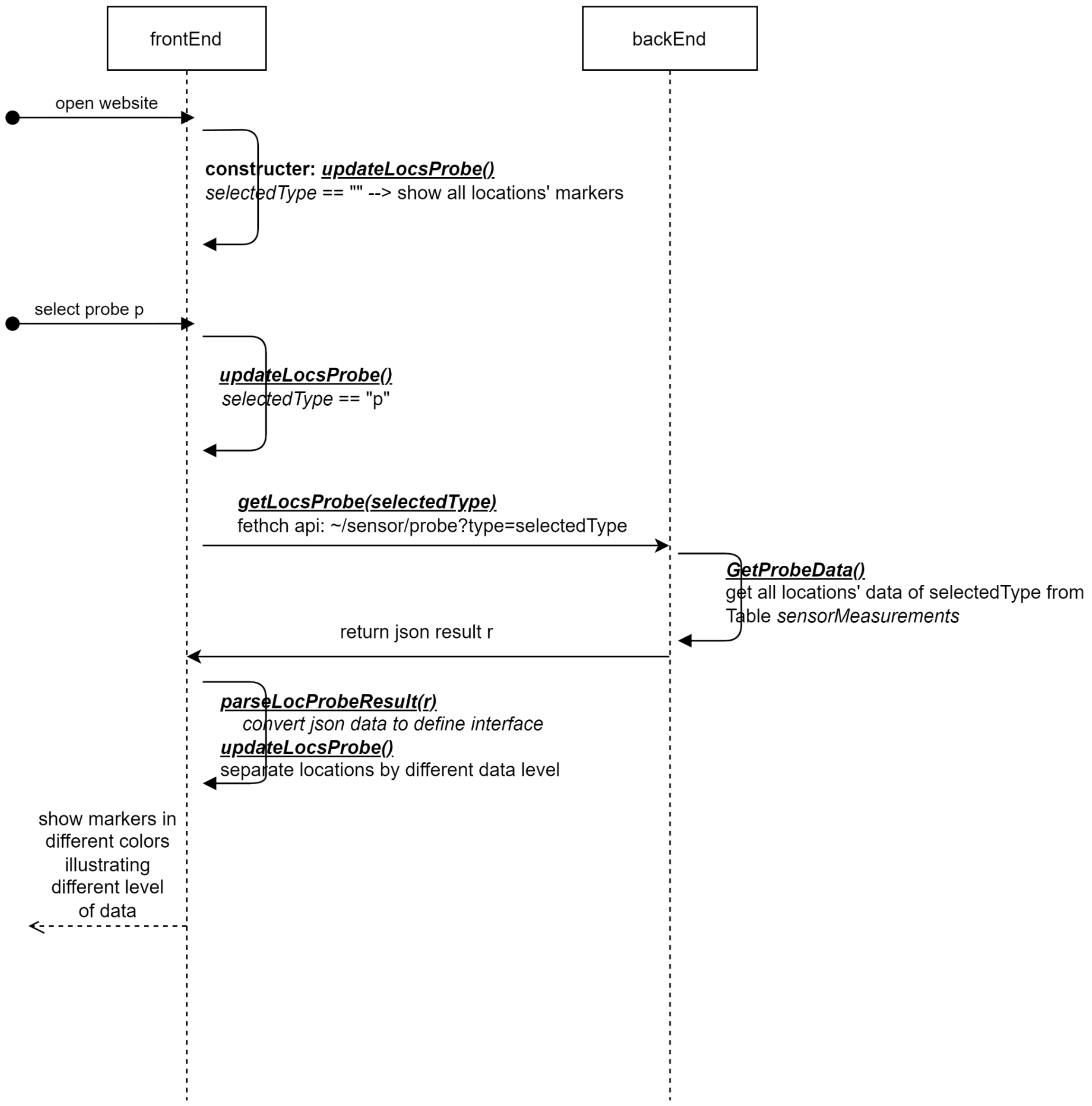
3.1 Programmatic description of the interfaces

The main page of our project consists of three parts: a map, two dropdown bars, and a data analysis area.

The code for the map page is in Map.tsx and MapDetail.tsx. In Map.tsx, it gets all data from the database and shows each sensor as markers. The dropdown bar of Node and Probe contains the sensor and sensor types available for users to select. After selecting an option, the map will display the corresponding changes, such as highlighting the selected *nodes*. If only the Probe is selected, the colors of markers will change, which are controlled in MapDetail.tsx by the value of level.
The code for the data analysis area is in Chart.tsx. It displays all data visualizations, primarily in table and graph formats. Currently because we are using the testing database,

only a limited number of selections will generate line graphs and tables. An example is when Node = s2 and Probe = PH, a smooth line graph will be generated below the map, with the data under the graph.

- **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.


frontEnd
backEnd
open website
constructer: updateLocsProbe()
selectedType == "" --> show all locations' markers
select probe p
updateLocsProbe()
selectedType == "p"
getLocsProbe(selectedType)
fethch api: ~/sensor/probe?type=selectedType
GetProbeData()
get all locations' data of selectedType from Table sensorMeasurements
return json result r
parseLocProbeResult(r)
convert json data to define interface
updateLocsProbe()
separate locations by different data level
show markers in different colors illustrating different level of data

3.2 Naming convention

3.2.1 Backend

GetProbeData: 2 GET methods for our new APIs
Url: /sensor/probe
if id == “”: 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 (coordinates value of *sensorInfo* table: String), latest location update timestamp (time of *sensorMeasurements* table: String), value (value of *sensorMeasurements* table: String), level (level: Number, calculated by function ***levelCompare()***)

else: a GET method that gets sensor *probe* data by *node* id, type, and time period
- Parameters:
    - id (sensorId, sensor ids in both *sensorMeasurements* and *sensorInfo* are the same: String, can be an empty string “”),
    - type (sensortype of *sensorMeasurements* table: String)
    - startTime (time: String, e.g. ‘2023-01-05’, can be an empty string “”)
    - endTime (time: String, e.g. ‘2023-01-05’, can be an empty string “”)
- Return:
    - id (sensorId: String)
    - name (sensorName of sensorInfo table: String)
    - type (sensortype of *sensorMeasurements* table: String)
    - data (Array): time (time of sensorMeasurements table: String), value (value of sensorMeasurements table: String)

ModifyById: a PUT method that update data by *node* id, table, typeBefore, typeAfter
Url: /sensor/modify_by_id
- Parameters:
    - id (id: Number, not null),
    - table (table: String, not null),
    - type (sensortype of *sensorMeasurements* table: String)

levelCompare: a helper function to calculate level (low/middle/high)
- Parameters:
    - type (sensortype of *sensorMeasurements* table: String)
    - value (value of *sensorMeasurements* table: String)
- Return:
    - level (level: Number, 0 represents low, 1 represents medium, 2 represents high)

3.2.2 Frontend

Interface Data: a new interface that stored the information we need for generating the probes on the map
- Attributes:
    - id: string,
    - name: string,
    - coordinates: [number, number],
    - time: string,
    - value: number,
    - level: number

updateLocsProbe(): separate the data by their levels into four sub-arrays in array

- Return:
    - Update locsProbeLevelData as the value of new array

<u>parseLocProbeResult(result)</u>: separate the raw data got from API to array

- Parameters:
    - result (raw data get from the database through API, JSON)
- Return:
    - currArr (array of type Data)

<u>getLocsProbe(selectedType)</u>: fetch data using API from the database

- Parameters:
    - selectedType (sensortype of *sensorMeasurements* table: String)
- Return:
    - res (JSON file of id, name, coordinate, time, value)

<u>createIcon(url)</u>: a function to generate marker icons of specific image and size

- Parameters:
    - url (file path of marker image: String)
- Return:
    - Leaflet Icon object using the image specified by url in a specific size

<u>getMarkerIcon(index)</u>: generate markers in different colors according to level

- Parameters:
    - index (level in locsProbeLevelData: Number)
- Return:
    - Leaflet Icon objec (call createIcon(url), the input of createIcon(url) varies according to the index)

3.2.3 Database for Water Data

Two tables are used to support the website in total. All fields of both tables are listed below.

Table Name	**Field Name**	**Data Type**	**Allow Nulls**	**Field Description**
sensorInfo	time	Number (timestamp)	No	● Get this field from Create *nodes* info POST method ● Latest location update timestamp
sensorInfo	id	String	No	● Get this field from Create nodes info POST method ● Unique id for the node

sensorInfo	name	String	No	● Get this field from Create nodes info POST method ● Node name
sensorInfo	value	String	No	● Get this field from Create nodes info POST method ● The coordinates of the node location
sensorMeasurements	time	Number (timestamp)	No	● Get this field from Create sensor probe data POST method ● Latest sensor update timestamp
sensorMeasurements	id	String	No	● Get this field from Create sensor probe data POST method ● Same id for the node of sensorInfo table
sensorMeasurements	sensortype	String	No	● Get this field from Create sensor probe data POST method ● Sensor probe type
sensorMeasurements	value	Number	No	● Get this field from Create sensor probe data POST method ● value detected by the sensor

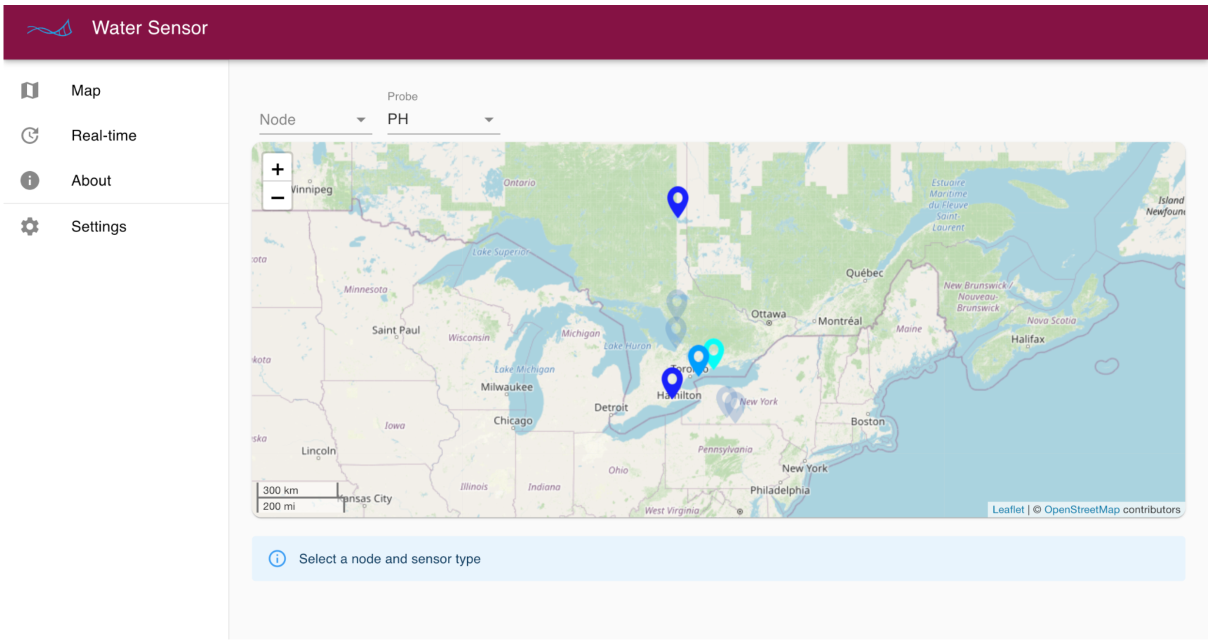
4. Setup Instructions

Preparations:
- Clone the GitLab repo to a local folder in your device
- Connect to McMaster Campus Wi-fi or use McMaster VPN

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

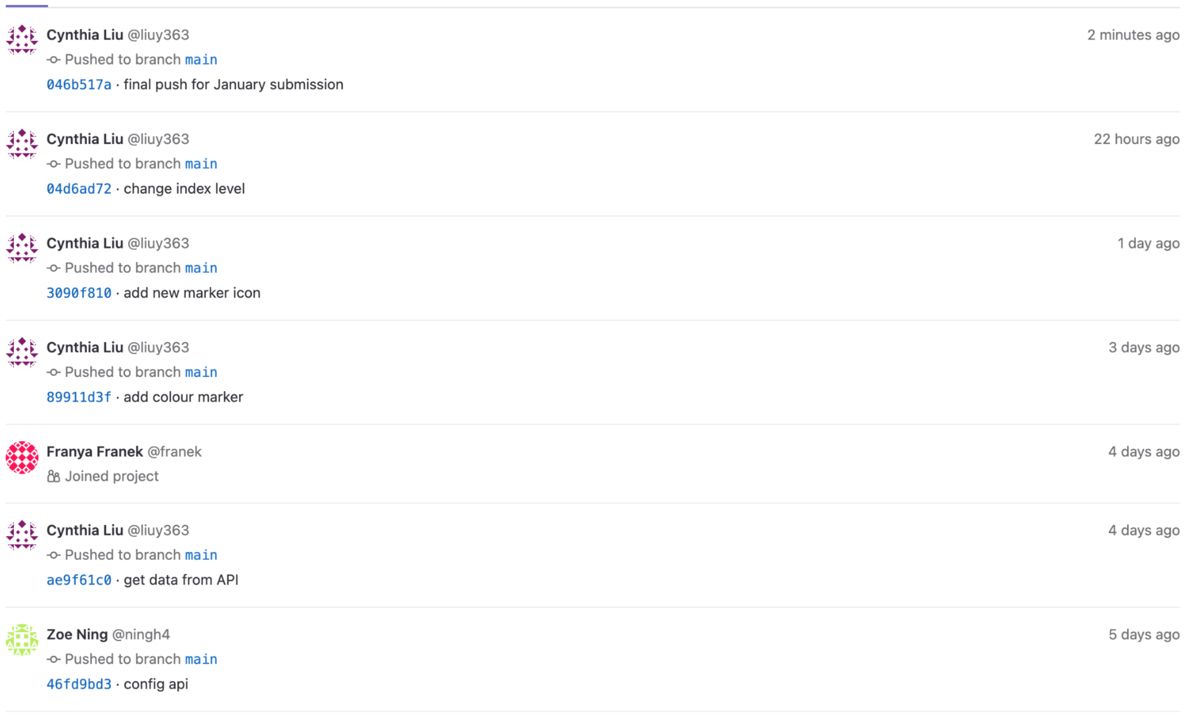
- If the text “Starting server for testing HTTP GET POST...” is displayed, the backend is set up.
- Then, visit our test environment website: https://dev.macwater.re-mote.tk/.

The website should look like this:



5. URL & Copy of Commit

GitLab repository: https://gitlab.cas.mcmaster.ca/lij416/capstone_2022
GitLab repository commit history:



Cynthia Liu @liuy363 — 2 minutes ago
Pushed to branch main
046b517a · final push for January submission

Cynthia Liu @liuy363 — 22 hours ago
Pushed to branch main
04d6ad72 · change index level

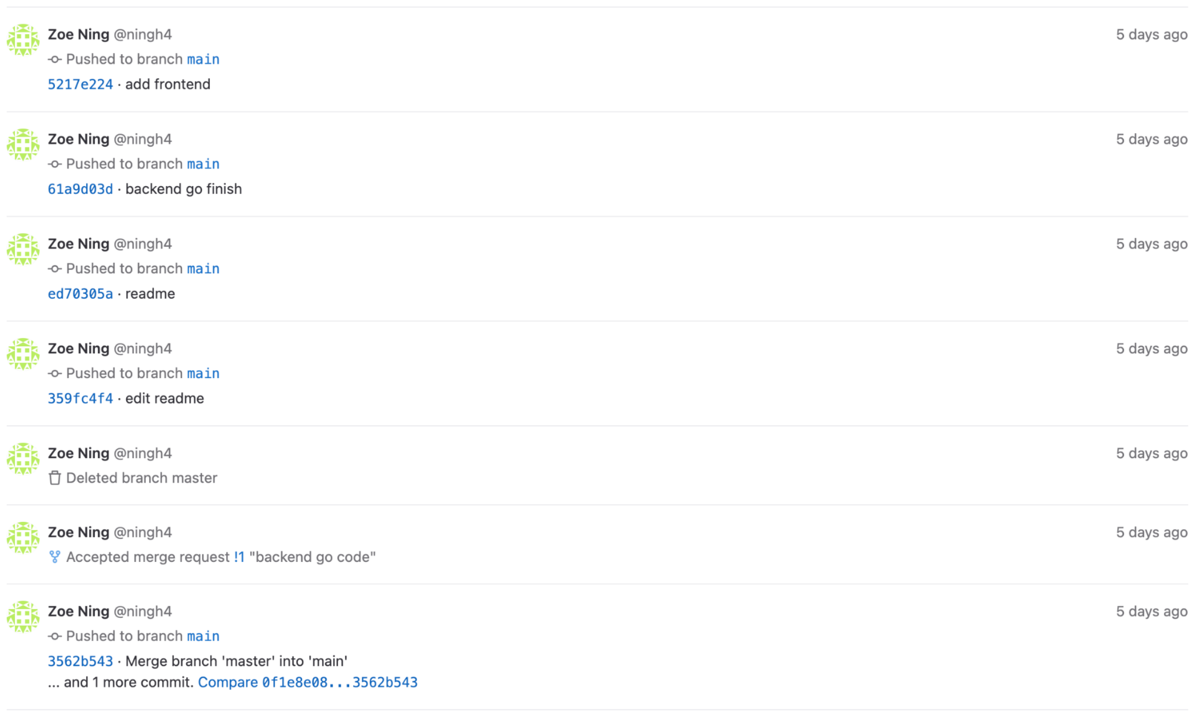
Cynthia Liu @liuy363 — 1 day ago
Pushed to branch main
3090f810 · add new marker icon

Cynthia Liu @liuy363 — 3 days ago
Pushed to branch main
89911d3f · add colour marker

Franya Franek @franek — 4 days ago
Joined project

Cynthia Liu @liuy363 — 4 days ago
Pushed to branch main
ae9f61c0 · get data from API

Zoe Ning @ningh4 — 5 days ago
Pushed to branch main
46fd9bd3 · config api



Zoe Ning @ningh4 — 5 days ago
Pushed to branch main
5217e224 · add frontend

Zoe Ning @ningh4 — 5 days ago
Pushed to branch main
61a9d03d · backend go finish

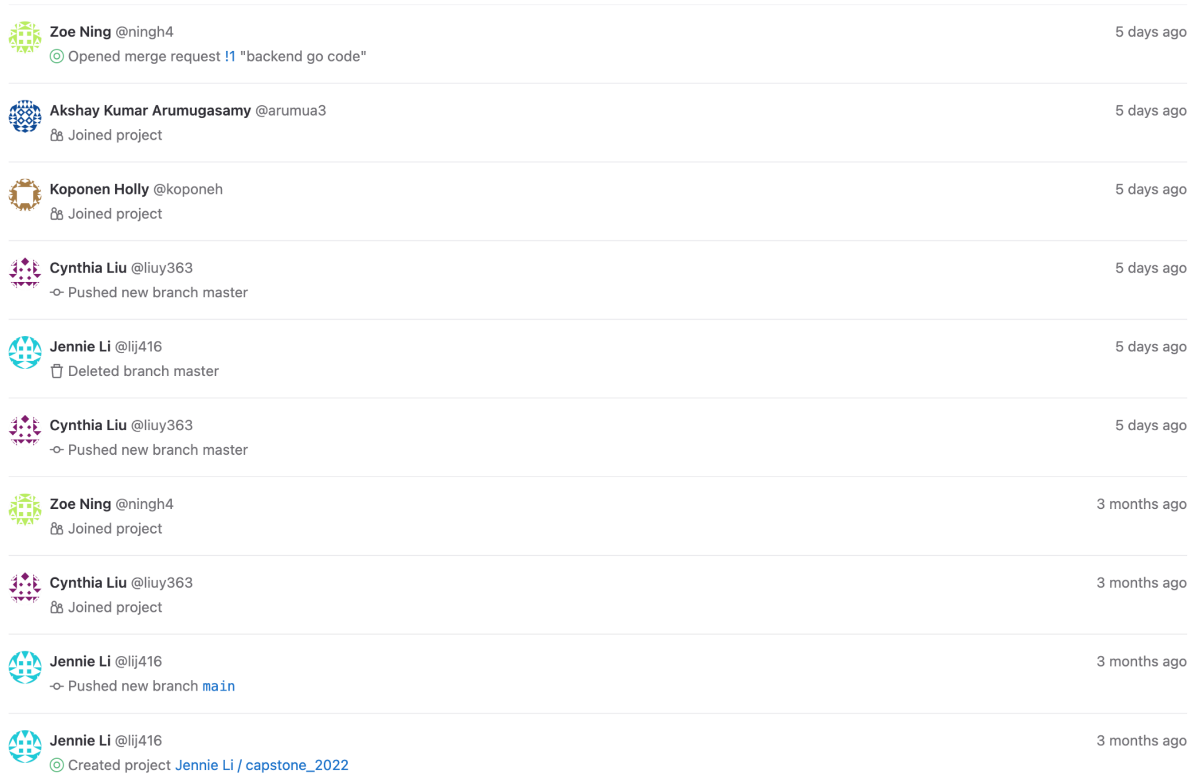
Zoe Ning @ningh4 — 5 days ago
Pushed to branch main
ed70305a · readme

Zoe Ning @ningh4 — 5 days ago
Pushed to branch main
359fc4f4 · edit readme

Zoe Ning @ningh4 — 5 days ago
Deleted branch master

Zoe Ning @ningh4 — 5 days ago
Accepted merge request !1 "backend go code"

Zoe Ning @ningh4 — 5 days ago
Pushed to branch main
3562b543 · Merge branch 'master' into 'main'
... and 1 more commit. Compare 0f1e8e08...3562b543



Zoe Ning @ningh4 — 5 days ago
Opened merge request !1 "backend go code"

Akshay Kumar Arumugasamy @arumua3 — 5 days ago
Joined project

Koponen Holly @koponeh — 5 days ago
Joined project

Cynthia Liu @liuy363 — 5 days ago
Pushed new branch master

Jennie Li @lij416 — 5 days ago
Deleted branch master

Cynthia Liu @liuy363 — 5 days ago
Pushed new branch master

Zoe Ning @ningh4 — 3 months ago
Joined project

Cynthia Liu @liuy363 — 3 months ago
Joined project

Jennie Li @lij416 — 3 months ago
Pushed new branch main

Jennie Li @lij416 — 3 months ago
Created project Jennie Li / capstone_2022