Rapport

Group 5

Marie Holme Kjær, studentnr: 539668, mariehkj@stud.ntnu.no

Truls Teige Pettersen, studentnr: 539673, trulstp@stud.ntnu.no

https://cloudoblig3.herokuapp.com/

https://github.com/trulstp/cloudOblig2final

Sadly, we could not get the backend to work on Heroku. We are not sure why, because it works fine on localhost. In the GitHub repository we have the version that works on localhost. We believe Axios is the problem, because we got a cors issue that we could not find a fix for.

The scenario

Sensors gather data about the weather around beaches in California. This data is saved on a server and displayed on a website. The data is publicly available and can be used by both researchers and everyday people. There are sensors that gather data about the air temperature.

The requirements

This project requires sensors that capture air temperature, a database to store the data and a website client to display the data. We use MongoDB as the database.

The plan

The plan is to gather data with sensors from weather stations around beaches in California. The weather stations ended up being 63rd Street Weather Station, Foster Weather Station and Oak Street Weather Station. Then we send this data to the database through a MQTT broker. The database saves the data along with the date it was pushed to the database.

The website fetches the data from the database and analyses it. It will find the minimum temperature, maximum temperature, the average temperature of each station, and overall. The biggest jump in temperature is also calculated for each station. The website will then print the analyzed data.

The data

The data we used came from the city of Chicago, and is real data gathered from sensors. We decided to use the air temperature for this assignment.

The system

Fetch data

```
async componentDidMount() {
    const response = await this.fetchData();
    this.setState({ array: response.data[0].topic });
    console.log(response.data[0].topic);
}

fetchData() {
    return axios.get("http://localhost:5000/app/");
}
```

This function fetches the data from the database. We use the Axios packet to make the frontend and backend work together. To fetch the data, we use the Axios get-request, and save the response the state “array”.

Arrays

Air temperature from all stations

```
fetchAirTemps() {
    const tempData = this.state.array;
    let temperature = [];
    tempData.forEach((data) => {
        temperature.push(data.airTemperature);
    });
    return temperature;
}
```

To get the temperature values from all the station, we first save the state as const *tempData* to make it easier to work with. We also create an empty array, which we push all the temperature values to. We then return the array. This function gets called as a parameter in the analysis functions later.

Air temperature from specific station

```
fetchAirTempsStation(name) {
    const tempData = this.state.array;
    let temperature = [];
    tempData.forEach((data) => {
        if (data.stationName === name) {
            temperature.push(data.airTemperature);
        }
    });
    return temperature;
}
```

This function works mostly the same way as the *fetchAirTemps* function. The difference is that it takes the name of the station as a parameter. It then checks with an if-test whether the station name matches the parameter, and if it does, it pushes the temperature to the array and returns it.

Minimum

```
calcMin(values) {
    return Math.min(...values);
}
```

The *calcMin* function finds the minimum value in an array. It takes the array as a parameter, and uses the built in Math function *Math.min* to find the lowest value in the array.

Maximum

```
calcMax(values) {
    return Math.max(...values);
}
```

This function works mostly the same as the *calcMin* function, but it uses *Math.max* to find the maximum value in the array.

Average

```
calcAverage(values) {
    let sum = 0;

    values.forEach((value) => {
        sum += value;
    });

    return (sum / values.length).toFixed(2);
}
```

To calculate the average temperature, we first create a variable *sum* with the value 0. We then loop though the array and add all the temperature values to the *sum*. We then divide the sum with the length of the array and round up to two decimals, and return this value.

Difference

```
calcJump(values) {
    let highestJump = 0;
    for (let i = 0; i < values.length - 1; i++) {
        const dif = Math.abs(values[i] - values[i + 1]).toFixed(2);

        if (dif > highestJump) {
            highestJump = dif;
        }
    }
    return highestJump;
}
```

The *calcJump* function finds the biggest difference between two temperatures that are 1 hour apart. By using a for loop, we cycle through each temperature, and subtract the value from the next temperature. We convert this to an absolute number and compare it to the highest jump. If the new value is higher, we save that as the new highest jump and continue doing this until we have gone through the array. Lastly, we return the value of the highest jump.

The graph

```
const graphData = {
    labels: timestamps,
    datasets: [
        {
            label: "63rd Street Weather Station",
            backgroundColor: "rgb(255, 100, 130)",
            borderColor: "rgb(255, 100, 130)",
            data: station63rdTemp,
        },
```

```
<Line data={graphData} options={{ scales: { y: { title: { display: true, text: "Air temperature" } } } }} />
```

To create the graph over the temperatures, we used the Graph.js packet. As the labels we put an array of the timestamps of when the temperature was recorded. The dataset label is one of the weather stations, and the data of that dataset is the array of temperatures that belong to that station.

Running both the frontend and backend

```
app.use(express.static(path.join(__dirname,"../frontend/build")));
app.use("*",(req,res)=>{
    res.sendFile(path.join(__dirname, "../frontend/build", "index.html"));
});
```

By using this function we can run both the frontend and the backend at the same Heroku server. First it creates a path to the html page, and then it sends that file when the backend loads.

MQTT

```
client.on('connect', () => {
    console.log('Connected')
    client.subscribe([topic], () => {
        console.log(`Subscribe to topic '${topic}'`)
    })
    client.publish(topic, JSON.stringify(message), { qos: 0, retain: false }, (error) => {
        if (error) {
            console.error(error)
        }
    })
})

client.on('message', (topic, payload) => {
    console.log('Recieved Message:', topic, JSON.parse(payload.toString()))
})
```

We send the data from the sensors to a MQTT broker. This allows us to publish the array, but also subscribe with the database. By doing this the data is transmitted to the database.

The webpage

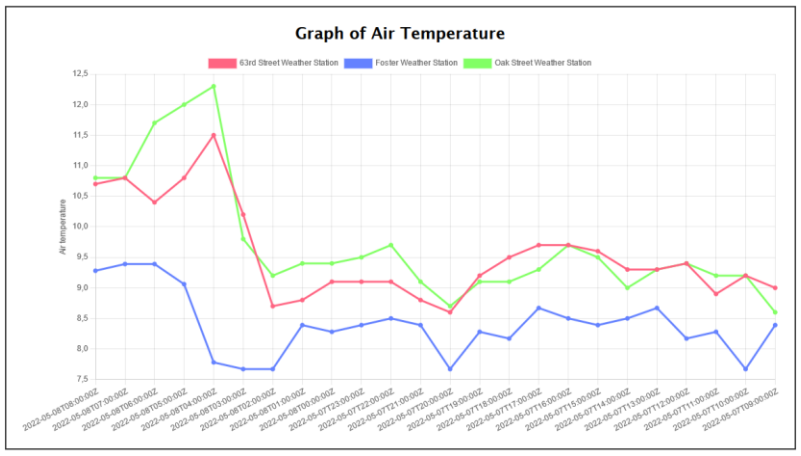
Here is how the data is printed after the functions have run. The data is categorized by the different stations, as well as a section with the overall results.

Analysis

Overall average Air Temperature: 9.23
Overall lowest Air Temperature: 7.67
Overall highest Air Temperature: 12.3

63rd Street Weather Station analysis

Average Air Temperature: 9.56
Lowest Air Temperature: 8.6
Highest Air Temperature: 11.5
Biggest jump in Air Temperature: 1.50



Here is how the graph turned out. By clicking on one of the station labels on top, you can hide that specific station. You are also able to see each specific value by hovering over each point on the graph.

Source of data

https://data.cityofchicago.org/Parks-Recreation/Beach-Weather-Stations-Automated-Sensors/k7hf-8y75