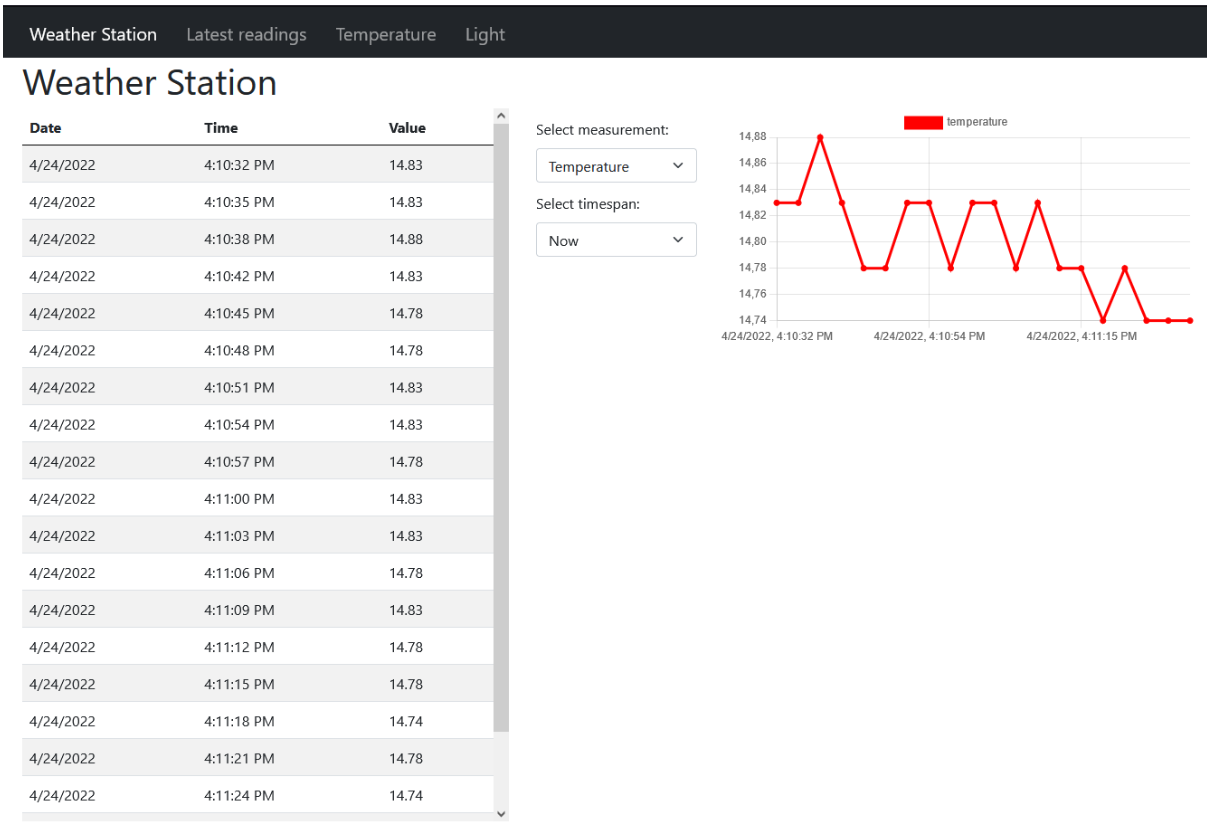
Front End Documentation- Oskari Lindroos

Documentation for the front end of a web page that fetches data from TAMK's public weather API and displays it to the user.



Weather Station

Date	Time	Value
4/24/2022	4:10:32 PM	14.83
4/24/2022	4:10:35 PM	14.83
4/24/2022	4:10:38 PM	14.88
4/24/2022	4:10:42 PM	14.83
4/24/2022	4:10:45 PM	14.78
4/24/2022	4:10:48 PM	14.78
4/24/2022	4:10:51 PM	14.83
4/24/2022	4:10:54 PM	14.83
4/24/2022	4:10:57 PM	14.78
4/24/2022	4:11:00 PM	14.83
4/24/2022	4:11:03 PM	14.83
4/24/2022	4:11:06 PM	14.78
4/24/2022	4:11:09 PM	14.83
4/24/2022	4:11:12 PM	14.78
4/24/2022	4:11:15 PM	14.78
4/24/2022	4:11:18 PM	14.74
4/24/2022	4:11:21 PM	14.78
4/24/2022	4:11:24 PM	14.74

Select measurement: Temperature

Select timespan: Now

This is a screenshot of the website's main page. The website has a total of 4 pages: the main page, all latest readings, latest temperature- and latest light readings.

In this documentation only the main page is talked about since the other pages are just small variations of the main page.

HTML & CSS

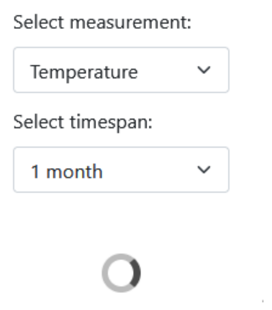
The whole webpage was built using Bootstrap. The navigation bar is an unordered list element wrapped in a Bootstrap navbar component.

The content under the navigation bar is all wrapped in a Bootstrap container component to make it nice and centered.

The container is also divided into 3 bootstrap columns that are of variable width.

The leftmost column houses the data table. The table is an empty html element styled with Bootstrap. An additional style was added to the table to make it a fixed height and to add a scrollbar and a sticky header. This I thought was a nice addition to the user experience since this allows the data and the chart to be displayed at the same time if the user selects a larger timespan.

The center column consists of two select elements with which the user can select the measurement and the timespan. A CSS loading animation was added to improve the user experience. When selecting larger timespans e.g. a month, the API request takes a while so the loading icon lets the user know that something is actually happening.

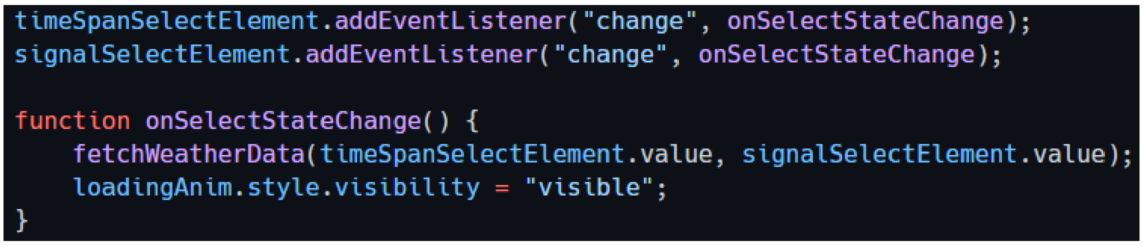


The rightmost column has a HTML canvas element to house the chart that is made with Chart.js.

JavaScript

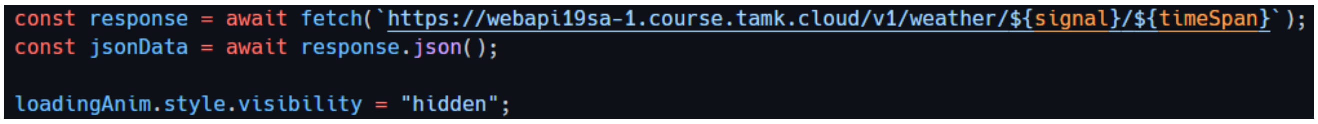
Once the website loads, the select elements are stored as variables and an event listener is added to them.

The event listener is listening to the change event, so that when the user selects a different measurement or timespan a function is called that calls the fetchWeatherData function with the current selected values as parameters. At the same time the loading icon is made visible to the user to let them know that the API request is taking place.



```
timeSpanSelectElement.addEventListener("change", onSelectStateChange);
signalSelectElement.addEventListener("change", onSelectStateChange);

function onSelectStateChange() {
    fetchWeatherData(timeSpanSelectElement.value, signalSelectElement.value);
    loadingAnim.style.visibility = "visible";
}
```

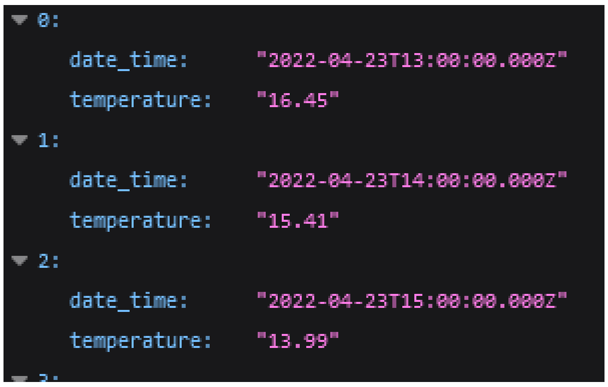


```
const response = await fetch(`https://webapi19sa-1.course.tamk.cloud/v1/weather/${signal}/${timeSpan}`);
const jsonData = await response.json();

loadingAnim.style.visibility = "hidden";
```

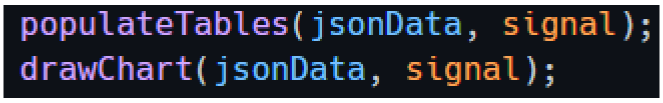
Once the data has been returned, the loading icon is hidden.

The default API response looks like this:



```
▼ 0:
    date_time:     "2022-04-23T13:00:00.000Z"
    temperature:   "16.45"
▼ 1:
    date_time:     "2022-04-23T14:00:00.000Z"
    temperature:   "15.41"
▼ 2:
    date_time:     "2022-04-23T15:00:00.000Z"
    temperature:   "13.99"
```

This response is then converted to JSON, stored as an object, and used as a parameter to call two functions:

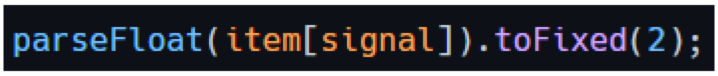


```
populateTables(jsonData, signal);
drawChart(jsonData, signal);
```

The populateTables function works as the name implies. It loops through the JSON data and for each item, populates the html table with a data cell with the corresponding weather data.

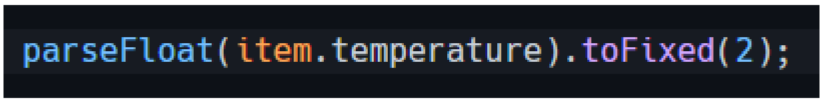
The date and the time are easily converted from the API response format to a nicer format using JS's built-in toLocaleTimeString and toLocaleDateString methods.

The actual weather values are accessed by using the property name as a variable. This is also parsed as a float and converted to a 2 decimal number:



```
parseFloat(item[signal]).toFixed(2);
```

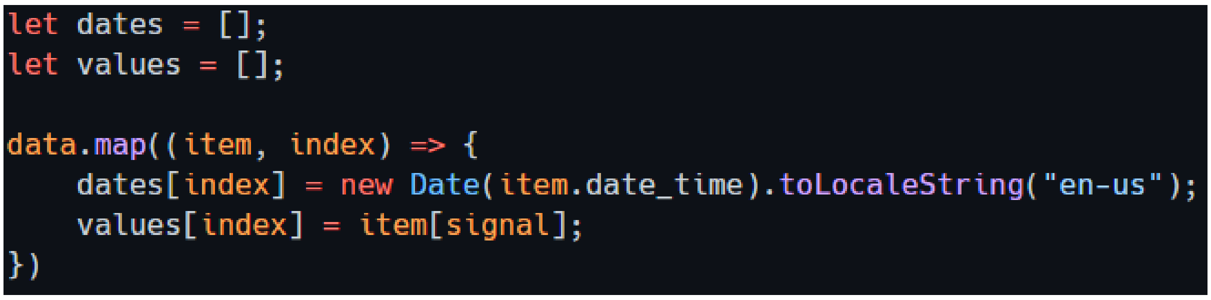
With the default select values that does the same as this:



```
parseFloat(item.temperature).toFixed(2);
```

But since the user can select the measurement, the property name had to be used as a variable.

The drawChart function is responsible for drawing a chart for the data using Chart.js. Before doing anything with the chart the dates and values are stored to arrays in a loop. This makes it easy to use them as X- and Y-axes for the chart and formats the date to the proper format simultaneously.



```
let dates = [];
let values = [];

data.map((item, index) => {
    dates[index] = new Date(item.date_time).toLocaleString("en-us");
    values[index] = item[signal];
})
```

Next the actual chart is created with the weather values as the Y-axis and the dates as X-axis. Additional options are also used to limit the number of date ticks on the X-axis to make it look tidier.