Abstract DLBCSPJW01

By Simone Moima

This application uses the openWeatherMap API to receive temperature data of predefined places and extracts the relevant information using user input. The result displays the city names, the countries and their current or monthly average temperature, based on the preference of the application user.

Architectural design

The application has consist of four components, of which two include the most important functionalities of the application, the CurrentWeather.vue as well as the MonthlyAverage.vue files. They communicate with the OpenWeatherMap API by fetching predefined data through API calls. Several functions inside the components select data based on user input and extract user defined information so that it can be displayed in the browser. Since the fetched data of both components is different and the functions needed to be adjusted accordingly, most of the code could not be stored in separate components and be reused. The components do not interact with each other to avoid accidental changes as the logic of both components is quite different. Yet the structure and naming conventions are intentionally similar, as the output of the functions are almost equal.

A sidebar component is embedded into these two major components, as it doesn't change and its function is only to provide router links and an offcanvas with information. This avoids code redundancy.

The landing page is saved in the TheWelcome.vue file. It includes a carousel used entirely for aesthetic reasons and to encourage the users desire to travel. Multiple router links are embedded to make it easy for the user to navigate the pages.

A navbar is included in the entry point of the application, the App.vue file, so that it can be displayed in all router views and code redundancy can be avoided.

Most of the CSS code is saved in the main.css file in the assets folder. Since the UI design of the router views TravelLater and TravelNow are almost identical, they can be simultaneously changed.

When the application is viewed on a mobile device, the elements will be displayed vertically below each other. While used on a larger screen, the elements will be displayed next to each other, from left to right.

Multiple router views were added to visually differentiate the components and their functionality.

Technical choices

Since the application was meant to be Java Script based, Vue.js was a good choice. It is a popular framework that is easy to understand, yet gives great flexibility.

Bootstrap is also a popular framework that helps with the HTML and CSS setup. It aids in adding responsiveness to the web application.

Implementation challenges

It is difficult to make software design choices in advance, without knowing the functionalities of vue.js in detail. Therefore, the setup of the code needed to be changed several times during the development. Also, not all functionalities that vue.js offers are used since the time to learn and implement every aspect would exceed the timeframe for the course.

A difficulty that was encountered was when to call a function. For example the load() function was implemented, since multiple conditions needed to apply and different functions called, which would have been too extensive if all were called by the html code for the submit button. This difficulty also applies to the sequence in which the functions are called to be logically sound and give the expected results.

Another coding challenge was to make sure the application waited for the API to return the results, before running the rest of the code. This was fixed by creating a promise array.

Since the historical data returned by the API for the monthly average temperature did not include the city names but only a city id, a JSON file needed to be created that would pair the city id with the city name and the country the city is in.

As a final challenge error messages needed to be created for incorrect use of the application. To make sure all error cases are covered, a third party tested the application and gave feedback on challenges in using the application. A considerable amount of time was used to test the application for all use cases and to ensure a smooth and error free user experience.

Lessons learned

To efficiently design a software, there needs to be efficient knowledge of the frameworks and technical components that are being used. A learning by doing approach might deliver results, yet there will be a lot of time used in rewriting and restructuring the code. In addition the code might not be as clean and convenient as it could be and some functionality offered by a framework might not be used at all.

Using frameworks, extensions and prewritten code can make the programming experience a lot more pleasant and time efficient. Even though it is important to know how code is written, not all needs to be coded from scratch. Reusability is an essential part of modern application development.