Homework 4 - More Practice with Vue!

By: Constantin Miranda & Kyle Harms

1. Overview

For this homework, we will practice creating a Vue project from scratch, fetching information from an API, and using Vue Router. The goal of this assignment is to improve your general programming and development abilities to better prepare you for the final project.

2. Learning Objectives

- Understand how to initialize a Vue project from scratch.
- Understand how to query information from an API.
- Understand CORS, why it exists, and how to bypass it.
- Gain further exposure to NPM packages (Axios) and API tools (Insomnia).
- Learn how to pass props to route components (dynamic routing).
- Improve overall Vue development skills by building an application from the ground-up.

3. Deadlines and Credit

Your work should be your own for this assignment. However, you should work *together* with your peers/teammates. Almost everyone in this class is learning this material for the first time. **Help yourself. Help them. Work together with your peers!** *You will *not hold them back!* Everyone benefits when we work together!

Assignment	Deadline	Credit
Part I	11/5, 4:10pm	~20 points
Part II	11/12, 4:10pm	~30 points
Part III	11/19, 4:10pm	~20 points
Part IV	11/19, 4:10pm	~30 points

4. Git Repository & Submission

Clone `git@github.coecis.cornell.edu:info4340-fa2019/YOUR_GITHUB_USERNAME-hw4.git`. Replace **YOUR_GITHUB_USERNAME** in the URL with **your actual GitHub username**. This is usually your NetID.

Submit **all** materials to your GitHub repository for this assignment. To submit you much **commit** *and* **push**.

5. Setup & Installation

For this homework assignment, you will need to install the following pieces of software:

- Insomnia

You will also need to install the following NPM packages **in your repository** as well:

- Axios
- CORS-Anywhere

6. Requirements

- Vue Project is initialized in **the root** of student's repository and selects the proper initialization options.
- Application uses the **MetaWeather API** to fetch and display weather information for **three** cities for **three** days.
- Student may pick any 3 cities they desire.
- The application displays city weather conditions, low temperature, and high temperature (at minimum -- feel free to add more!) for each city.
- Application uses **History Mode** for Vue Router to ensure clean URLs.
- Application **displays information dynamically** by passing props to route components. Minimal information is hard coded.
- Application uses the **CORS Anywhere** service to bypass CORS restrictions.

***Not* Requirements:**

- You **are not** required to implement the 'loading' label when awaiting the API call. However, you may if you would like to, it's good UX practice.
- You **are not** required to submit sketches. but you may if you would like. Please place these in a folder named **documents** in the root of your repository.

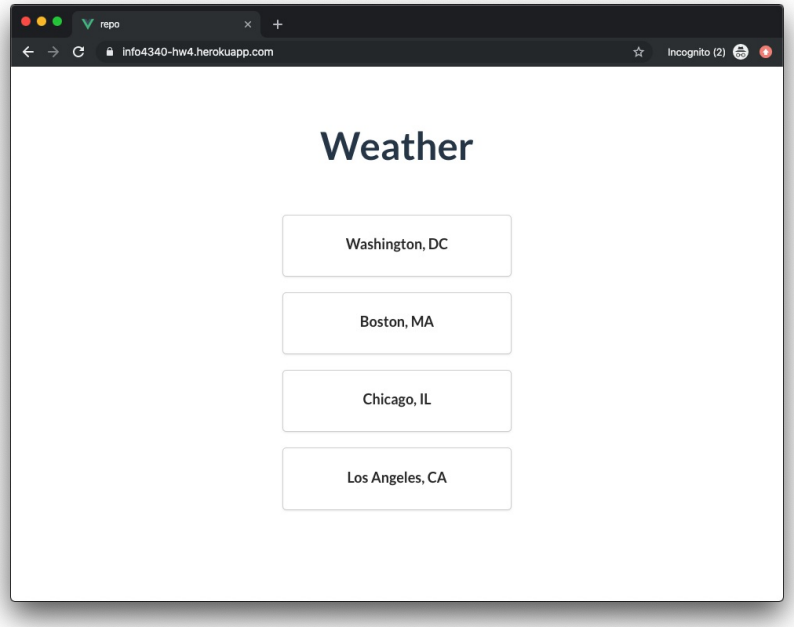
7. The Result

When finished with Homework 4, your app should look something like this example:
https://info4340-hw4.herokuapp.com/

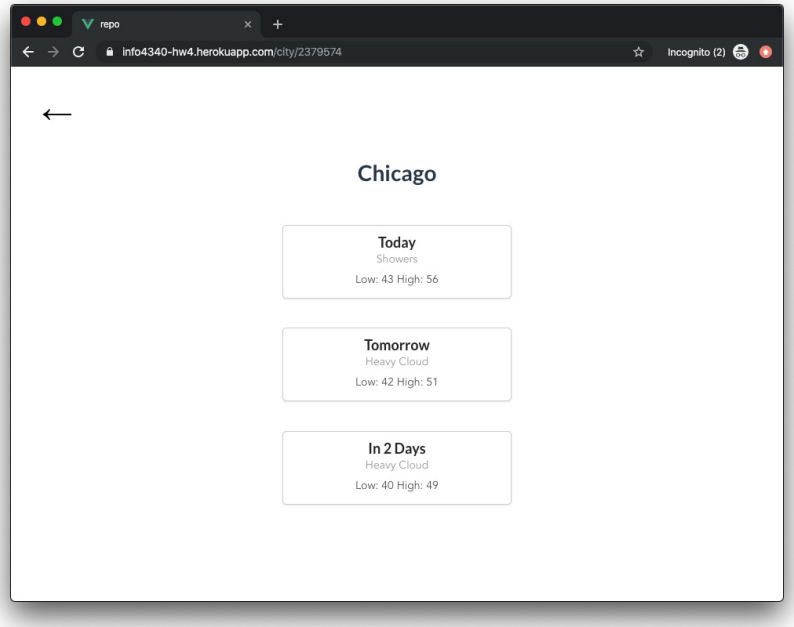
Before beginning this assignment, we recommend using the application to get a feel for what type of information to display and how the site is structured. **Pay particular attention to the URLs for the routes that you visit.** Think about how this application might be structured (components, views, routes) and how information is getting passed.

Note: When viewing the forecast for a city, please be patient while the data loads.

Note: Heroku's free Dynos sleep after 15 minutes of inactivity. Thus, if the site takes a while to respond, let the page load for about 30 seconds.



Home Screen



Weather for Chicago. ***Take notice of the URL!***

Part I: Initializing a Vue PWA from Scratch

For the past few projects, we have provided an initialized Vue PWA for you. In this assignment, we ask that you create your application from scratch.

Use the following documentation to help you figure out how to initialize a Vue Project using the Vue CLI: https://cli.vuejs.org/guide/creating-a-project.html#vue-create **Read this thoroughly!**

1. `vue create .`

Clone your homework repository. Within the **root** of this folder, you will need to initialize a Vue project with the following selected options.

To begin the setup process, execute *the appropriate* command:

- `vue create .` *: generates a Vue project **within** the current working directory.*
- `vue create <projectname>` *: generates a Vue project named **projectname**.*

  However, if a subfolder with the name **projectname** already exists, the CLI gives you the option to merge or overwrite the existing folder.

You should **manually select features**. You may save this as a preset at the end of the `create` process if you would like:

- Babel
- PWA
- Router
- Linter/Formatter
- History Mode Enabled for Vue Router
- ESLint w/ Error Prevention Only
- Lint on Save
- Store configuration in Package.json

Vue will now download and install all necessary packages and scripts to initialize your Vue Project. Run `npm run serve` to make sure everything works. Open the project in Atom to check the file structure.

Commit your newly initialized project. You should always store the initial state before your make any changes.

Part II: Vue Router: Dynamic Routes

Take a look at the example app you viewed *above* and you'll notice a few things. First, there are only 2 routes defined: the root route `/` (displays all cities) and `/city/{woeid}` (fetches & displays weather for a specific city given the Where On Earth ID). Second, all URLs are formatted as `/<path>` rather than `/#/<path>`.

This is a nice way to build an application since it allows information to be loaded dynamically into a reusable view. In other words, we can maintain consistent formatting and **don't need to hard code any information**.

1. When building your application, use **history mode** for Vue Router.

   You should have already done this when selecting options during `vue create`

2. Build and use a reusable views. Your application should have:

   - A **root** route `/` : Displays list of cities.
   - A **dynamic** route `/city/{woeid}` : Displays weather for a given city. **WOEID** is passed as a **route parameter**.

   Note: You might also consider building a reusable component to display the weather forecasts (today, tomorrow, in 2 days).

3. Read and thoroughly utilize the reference documentation to complete this part.

   - Router History Mode
   - Route Parameters & Dynamic Route Matching

1. Structure To Consider

- Views:

  Application has 2 **views** that correspond to 2 routes:

  - `/` points to home **view** (lists cities)
  - `/city/{woeid}` points to weather **view** (weather for a specific city)

- Components:

  - CityPanel - Reusable component for displaying a city name. Use in the home *view*.
  - WeatherPanel - Reusable component for displaying weather for a given day. Relevant information is passed to this component from the weather *view*.

2. Requirements

You should have an application that works like the result, *without the connectivity to the API (i.e. everything except for weather data)*.

- You should have an initialized Vue PWA project in your repository.
- You should be able to execute `npm run serve`.
- You should have functioning routes for `/` and `/city/{woeid}`.
- You should have a dynamic `/city/{woeid}` route, meaning a parameter can be passed in the url to identify the city.
- You should have placeholder Semantic UI buttons & data to differentiate the two routes.

Part III: Using an API

Apps use Application Programming Interfaces (API) to communicate between the client and the server. APIs are used to retrieve data and store data on the server. Typically most apps provide their own API. However, for this assignment, you'll practice using a third party API with your app.

You will use the MetaWeather API to retrieve the weather forecast and load that data into your app. This API is open source and aggregates weather from a variety of online sources to come to a single prediction.

Read through the API documentation to understand what type of information the API returns: MetaWeather API Documentation.

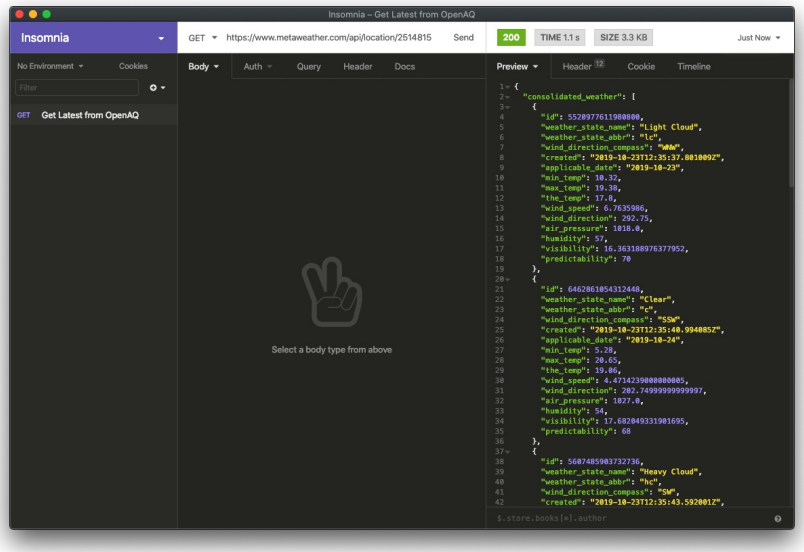
1. Insomnia: Exploring an API

APIs can be *confusing*. Don't mix trying to learn Vue and an API at the same time. Professionals explore APIs using tools like Insomnia (or Postman) *before* writing their code to utilize the API.

Use Insomnia to test querying the API.

Insomnia is a REST client that allows you to easily query an API and view the response using a graphical interface. Insomnia is the open source alternative to the very popular Postman (which is very good too). We recommend using Insomnia to explore the functionality of the API. You should use this as a debugging tool when something in your code is not working as expected.

If you need help using Insomnia, *work together with your peers* **and** view the Insomnia Documentation.



Above: An example MetaWeather query in Insomnia for Washington, DC. We are querying the 'location' endpoint (see documentation: /api/location/(woeid)/) with the 'Where On Earth ID' for DC: 2514815.

Many APIs return their data as JSON. The MetaWeather API's response will be formatted as JSON. We have compiled some resources to help you parse these results:

- Mozilla API Documentation

  Pay particular attention to the 'Methods' section.

- W3 Resource on JSON

  This is great for understanding **how** to use the methods.

- W3 Resource on Accessing JSON

  This is great for understanding how to access information within the JSON response.

2. Requirements

You should get 3 cities API calls working in Insomnia.

- You should have a good understanding of the type of information that the API is sending.
- You should have a plan for **which** data to extract and **how** you will extract it.

Part IV: Programming API Calls

Now that you've explore the MetaWeather API and you have tested your API calls, it's time to program the calls into your PWA.

You need to do your own work here. But you shouldn't work alone. Work together with your peers!

1. Axios: Querying the API

Use the Axios NPM package to query the API. **Make sure that you install this NPM package.**

Axios is an HTTP client for Node.js. In other words, Axios allows you to make API calls. Axios returns **JavaScript promises** which ensure that code is only run once a request completes. If you are not familiar with how to handle promises, reference the documentation.

Refer to the Axios Documentation.

1.1. Promises

Promises ensure that our code **first** queries the API and **then** performs some set of actions, only once the response has been received from the server: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise

Lucky for us, Axios returns a Promise object anyways, so all we have to do is call `.then()` (like in the Mozilla example) and write an inline function (also like in the Mozilla example) to pull information out and set data properties. The Axios documentation has excellent examples of this as well: https://www.npmjs.com/package/axios

2. Speed Bump: CORS & Bypassing CORS

One issue that developers often face is Cross Origin Resource Sharing (CORS). CORS is a security standard that ensures that servers mitigate the risks associated with requesting assets from different locations. However, this means that CORS must be configured on the server and thus if we do not operate the API, we cannot get around the CORS configuration. **Try making an Axios request to the MetaWeather API and see what happens in your browser's console.**

The correct way to use CORS is to specify the correct HTTP headers. Unfortunately, MetaWeather doesn't do this. ☹

We can *workaround* this by using a proxy. "CORS Anywhere" circumvents CORS by acting like a proxy server. Our request will be sent through the CORS Anywhere service and then passed along to the MetaWeather API.

Read the documentation to see how to integrate "CORS Anywhere" into your API call:

- About CORS
- How to Bypass CORS using CORS-Anywhere

*Hint: The workflow is to use **Axios** to query the **API URL** through the CORS-Anywhere proxy URL, located in the documentation. The response from the promise is then processed and the corresponding data is passed to the components.*

2.1. CORS Anywhere Example

Here is an example of how we might use Axios to query Cornell's course roster API:

Course Roster API Documentation

```
var cors_api_url = 'https://cors-anywhere.herokuapp.com/';
var api_url = 'https://classes.cornell.edu/api/2.0/search/classes.json?roster=FA14&subject=MATH'
axios.get(cors_api_url + api_url).then(function(response) {
  console.log(response)
}
```

3. Requirements

Your PWA should work like the *above* example and should pull live weather information from MetaWeather.

- Your application uses Axios to query the API and fetch the data in the `/city` route.
- The promise resolution passes the response to a function, which processes the data (extracts relevant information).
- The relevant data is passed to the component that displays the information for a given city.