{EPITECH}

TIME MANAGER

BOOTSTRAP - WEB INTERFACES

TIME MANAGER

JS framework with *components* and *templates* are useful to build user interfaces.
Let's use `Vue.JS` to:

- ✓ create a single page application to display weather forecasting data ;
- ✓ make calls to an external API ;
- ✓ present API data ;
- ✓ create graphics.

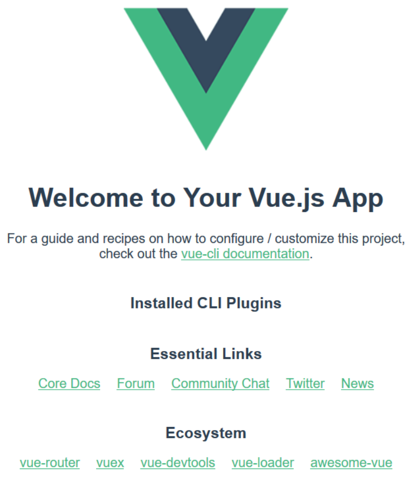
At the end of this Bootstrap, you should be able to create a dynamic user interface and query an external API then process its data to make it clear and readable.

An empty Vue.JS application

Install the npm package manager, Vue.JS and view-cli.

The use of Git Bash or Ubuntu Bash on Windows is **very strongly discouraged** or you will encounter bugs. If you are working on Windows, install the Windows version of npm and use the powershell for all your commands.

Then, create your first application and launch it using `npm`.
Checking your localhost on a browser, you should see this:



Welcome to Your Vue.js App

For a guide and recipes on how to configure / customize this project,
check out the vue-cli documentation.

Installed CLI Plugins

Essential Links

Core Docs Forum Community Chat Twitter News

Ecosystem

vue-router vuex vue-devtools vue-loader awesome-vue

The component

Before creating our first component, we must understand what is a View in `Vue.JS`. It is made up of three things:

1. a `<template>`, which contains the HTML code of the View (required) ;
2. a `<script>`, which contains the Javascript code of your view (and therefore its declaration, which is mandatory) ;
3. a `<style>`, which contains the CSS code of your View (optional).

It's considered as an app, with a unique id and associated with a unique HTML element.
As a result, in most cases, you will only have one view in your web application, which will contain your entire page, so we could imagine something like this for the `<body>` of your index.html page:

```
<body>
    <div id=app> <!-- here is the identifier of the View -->
        <!-- NO html code here; everything is in <template> in App.vue -->
    </div>
</body>
```

A component is a **reusable View instance** that has almost the same attributes / options as a View (except for a few exceptions), the most common being a **unique name** (its identifier, mandatory), a **template** (the HTML code, optional) and **data** (dynamic data, optional).

There are several methods for creating components, but we **demand** here a Single File Component which is surely the most used and the most practical / intuitive in terms of code architecture.

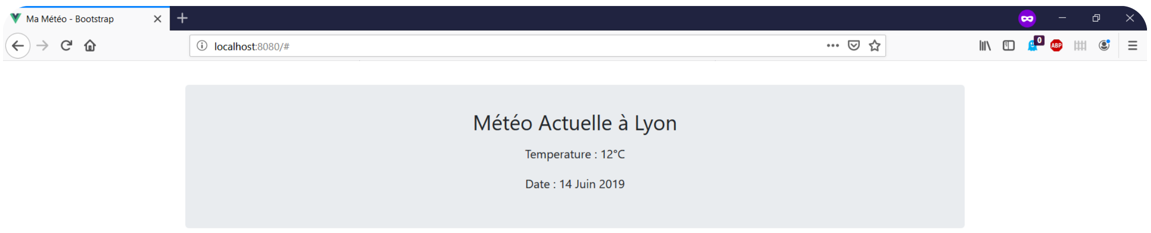
Create a **CurrentWeather** component to display data retrieved from the external API.

For the moment, in the absence of connection with an external API, your **data** will be an object hard-initialized with at least a town, a temperature and a date, initialized on today's date.

Your template should display the **data** of your component using the syntax `{{data_name}}`.

To retrieve and format the date of the day in French, we recommend you look at the Moment.js library available on npm.

Add your CurrentWeather component to your main View in the App.Vue file.
With a bit of CSS, you should get something like this:



Météo Actuelle à Lyon

Temperature : 12°C

Date : 14 Juin 2019

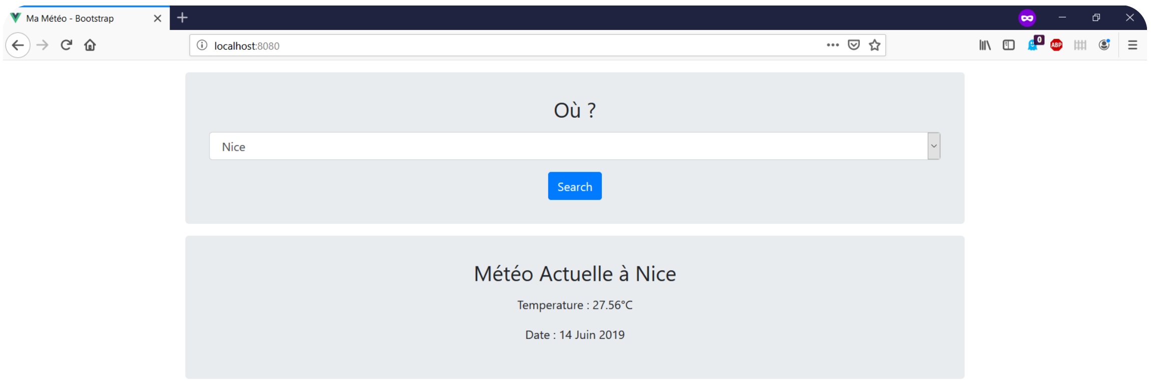
Now let's add **data** to the component.

In the template, add a `<select>` input (to select the city) and a *Search* button and a to your component's template.
The input must be linked to the **data** of your component using the v-model directive.
It needs to be built thanks to the v-for directive, so that a change in the input would impact the component.

Add a *refreshCurrentWeather* method to the component, to assign the *temperature* **data** with a random value.

v-on:click

Using some CSS Bootstrap classes, you should get something similar to this:



Où ?

Météo Actuelle à Nice

Temperature : 27.56°C

Date : 14 Juin 2019

Routing

The routing consists of associating a route (for example `/currentWeather`) to a component. As a result, when the user clicks on a link that points to the route, `Vue.JS` will display the associated component in the `<router-view>` tag (so do not forget to add it to the *template* of your main view!).

First, install vue-router.

You will find here how to import the router and use throughout your application.

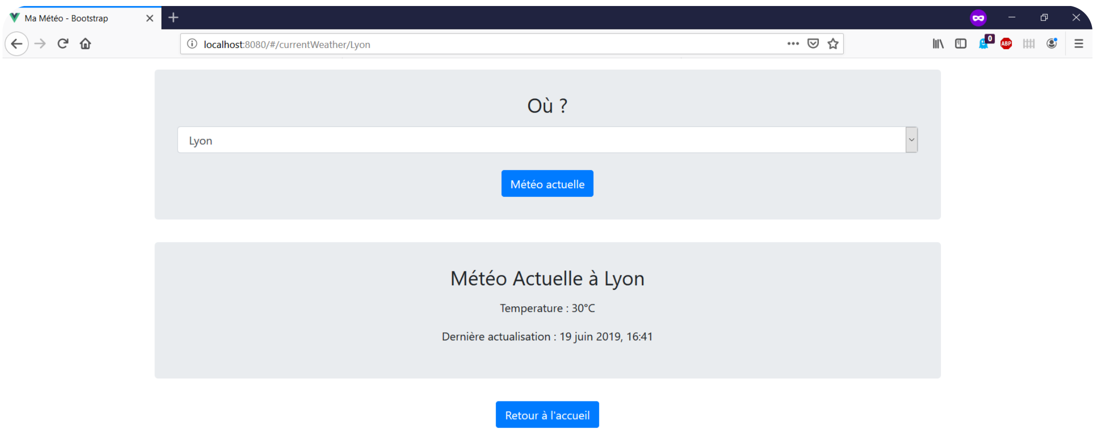
Then, define the `/currentWeather` route and associate it with your component. The route must take a `city` parameter.
Update the *city* **data** of the component so that it is initialized with the value of the city parameter.
Finally, move the `<select>` and everything connected to it in the Main View. It must now contain a *city* **data**, linked to `<select>`.

To get the value of the city parameter, use the $route property.

To display the component, link your Main View to the CurrentWeather component route.

You **must** use the `<router-link>` tag and the **v-bind:to** directive in order to set the *href* attribute of your link and pass it to your *city* **data** in parameter.

Once all this is functional, you should get something similar to the picture below (note the difference in the URL of your browser with the previous step):


Ma Météo - Bootstrap

localhost:8080/#/currentWeather/Lyon

Où ?

Lyon

Météo actuelle

Météo Actuelle à Lyon

Temperature : 30°C

Dernière actualisation : 19 juin 2019, 16:41

Retour à l'accueil

Interaction with the API

You now have all the information you need to view your data, but no data...
The next step is to interact with an external API in order to retrieve the weather data.

There are many weather APIs, free or not. No matter which one you use, but we recommend openweathermap.org which is free (but limited to 60 queries per minute) and which offers a forecast for the next 5 days and the current weather. Its documentation is available here. You will need to register on their website, get an **API KEY**.

First, install the *axios* npm package. Then, import it into the component this way:

```
<script>
    import axios from 'axios'

    axios.get(request).then(....);
</script>
```

There is only the modification of the *refreshCurrentWeather* function left, so that it makes a request to the API. Just do it, thanks to the *get* function of axios which uses the promises:

```
axios.get(requestUrl).then(resolveCallback, rejectCallback)`;
```

Use an anonymous function arrow `(response)=> {.....}` to stay into the scope of your component and to be able to use **this** (the argument response contains the JSON response returned by the API). For more information on anonymous functions and anonymous arrow functions, visit this link.

For testing purpose, simply display the response into the console in your callback functions, so that you get the API response in your browser console.

The data property of the object contains the weather data returned by the API.

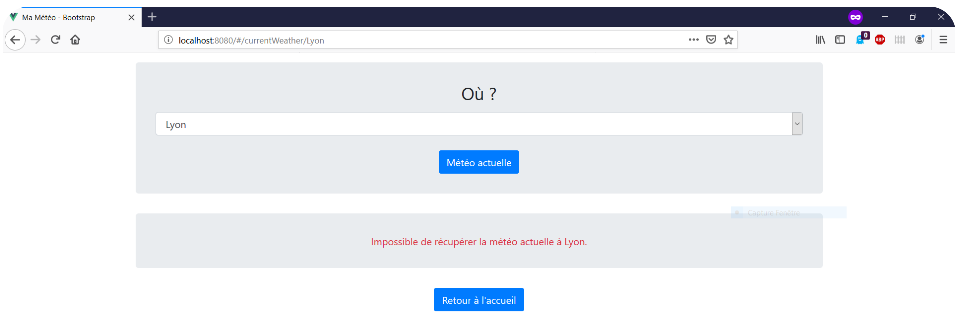
Data visualisation

Thanks to the **template / data** system of `Vue.JS`, you just have to modify your *temperature* **data** by assigning to it the value retrieved by the API in the *resolveCallback* callback of your promise; the refresh will be automatic.

However, you must also display an error message if the weather could not be retrieved.
To do this, add a *error* **data** to your component, initialized to an empty string.
Also add an HTML element to your template to display our *error* **data** when needed.

You **must** use the v-if directive.

Modify and update what needs to be, and you should get something like:



Impossible de récupérer la météo actuelle à Lyon.

Now let's create a graph with the vue-chartmaker npm package.
Install it using the following command :

```
Terminal
~/T-POO-700> npm install vue-chartmaker -\-save
```

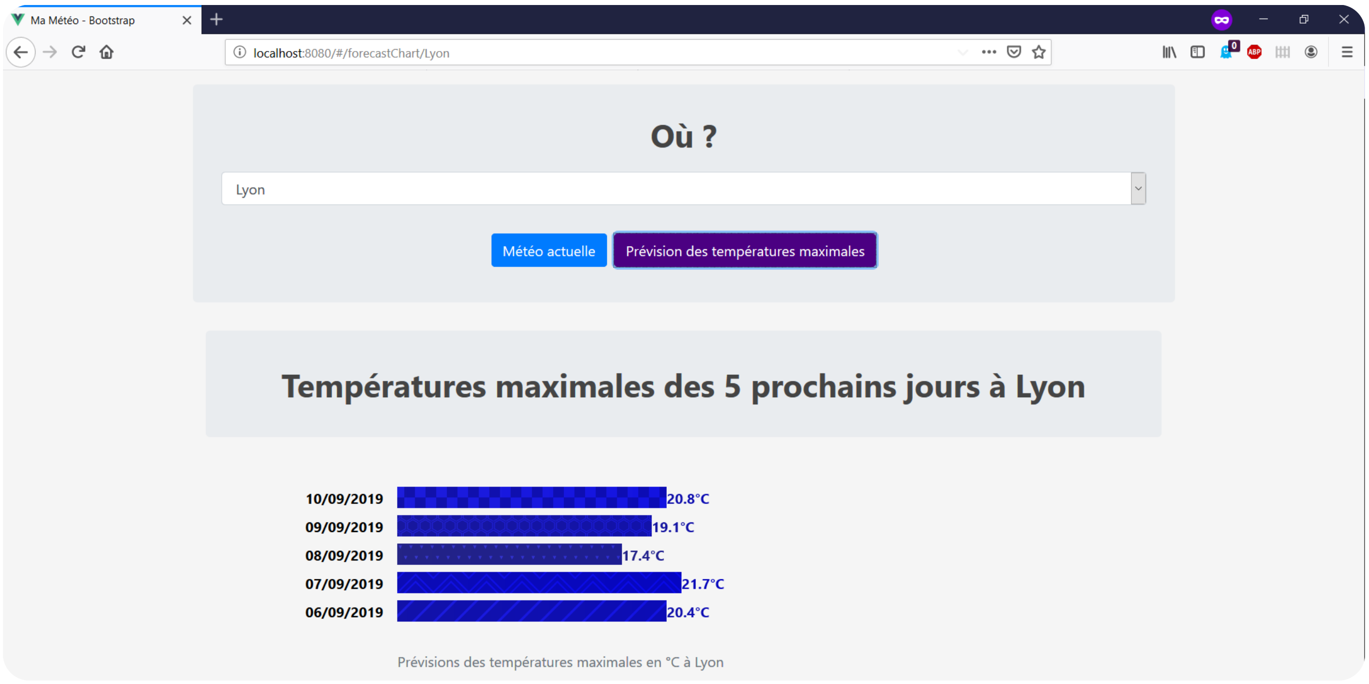
Your graph should be accessible through the route `/forecastChart/:city`.
It must retrieve the weather data from the API and process it for display in the graph.

Here is a draft of your template that you **must** use:

```
<template>
    <div id=YOUR_COMPONENT_ID>
        <chart-maker :params=YOUR_CHART_PARAMS v-if=YOUR_BOOLEAN_VARIABLE_NAME>
            <tr v-for=YOUR_FOR_LOOP :key=YOUR_UNIQUE_ID>
                <!-- TODO -->
            </tr>
        </chart-maker>
    </div>
</template>
```

Use a *v-if* directive instead of *v-show* will only create the HTML element when the condition is validated, which is very useful for asynchronous operations.

Here is an example of rendering :


Ma Météo - Bootstrap
localhost:8080/#/forecastChart/Lyon
Où ?
Lyon
Météo actuelle
Prévision des températures maximales

Températures maximales des 5 prochains jours à Lyon

10/09/2019	20.8°C
09/09/2019	19.1°C
08/09/2019	17.4°C
07/09/2019	21.7°C
06/09/2019	20.4°C

Prévisions des températures maximales en °C à Lyon v2

{EPITECH}