Zenika Vue.js Training

Labs

zenika

<animés par la passion>

Introduction

To practice what will be described in the main course, you will implement an application that allows you to manage TV shows.

You will start with a basic static web page and add new features based on what we will be seeing in the main course. The end goal is to have a **modern, rich and reactive** application.

The initial bootstrapped application is composed of a single HTML page containing a search form and a static list of TV shows.

To improve the design of the application and avoid writing too much CSS, the CSS framework `Bulma` and the icon font toolkit `font-awesome` have been included.

No Javascript code is included in the initial application. Any external libraries (such as Vue.js) will first be added using a CDN and later on using npm and webpack.

Getting started

Bootstrapped app

Start by extracting the received `lab.zip` archive. You should now have the following project structure:

```
├── index.html
├── css
  └── app.css
├── img
  └── logo.jpg
```

Feel free to open the `index.html` file in your browser to see what it looks like.

Lab 1 : Setting up Vue.js

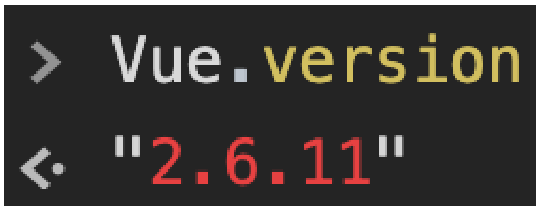
This lab will have us set up Vue.js.

In order to use Vue, we will start by importing it using a CDN. A CDN (Content Delivery Network) refers to a group of distributed servers that help us deliver content over the internet. In our case, we will obtain the code for Vue.js using the `https://cdn.jsdelivr.net/npm/vue/dist/vue.js` link.

1. Start by adding the following at the end of your `index.html`'s body:

```
<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
```

2. In your browser, try printing the current version of Vue using `Vue.version` to confirm that you have imported it successfully.



```
> Vue.version
<· "2.6.11"
```

Lab 2 : The Vue instance

Part 1: The first instance

In this lab, we will use what we've learned about the Vue instance so far.

1. Inside of `app.js`, create the Vue instance that will be the entry point to your application. Mount it on the `div` with the id `app`.

2. Print the Vue instance in your browser console. What do you see?

Part 2: Data attributes

We want to store our app's title as a `data` attribute of the Vue instance.

1. Add a `title` data attribute.

2. Print this title through an interpolation in the `h1` tag of your template.

Part 3: Lifecycle hooks

As a reminder, here's a list of the existing lifecycle hooks that Vue.js has to offer.

To understand their place in our instance's lifecycle, let's use all of them.

1. Add this `console.log` in each available hook in your Vue instance. Reload the page, to see their effect, what can you determine?

```
console.log('<hookname>', this.title, this.$el)
```

3. Call the `$destroy` instance method, what happens?

Lab 3 : Components

In this exercise we'll create a `Card` component that will only display some text.
In future labs, we will render values conditionally as well as render lists.

1. Declare a `Card` component using the `Vue.component` method. This component must contain:
    - a title
    - a description
    - a status
    - an image
    - a date of creation
    - the number of seasons
    - a list of genres
    - a boolean `isFavorite`
2. Create a `template` for our component. For now, only display textual data.
3. Let's now pass the content mentioned above as props to our component. Display each prop values accordingly using interpolation.
4. Using `isFavorite`, we want to display the `computed` property `{{ title }} is your favorite!` or `{{ title }} is NOT your favorite!` based on the value of the boolean.
5. In order to simulate a real-world example, you have been given mocked data that can be found in the `js/api.js` file. You can access these values via the `mockData` object. Use your newly created component to display two cards in your template. Don't forget to give them prop values.

Lab 4 : Vue-CLI

By the end of this lab, we will have transitioned our current project into its `@vue/cli` counterpart. Meaning we will take advantage of a cleaner architecture, single file components and all the quality of life improvements that it has to offer.

In the `lab4` workspace, you will find a bootstrapped project created using the `vue create lab4` command.
For now, we have a root `App.vue` component that displays our two static cards based on the `json` data. Let's now transition our project fully by creating a `Card` component and displaying it.

1. Install the npm dependencies

```
$ npm install
```

2. Start your development server and navigate to the link displayed in your console.

```
$ npm run serve
```

Your view will now be automatically updated whenever you modify a component.

3. Create a `Card.vue` component inside of our `components` folder, don't forget to set its props, computed value and template.

4. Import the newly created component in our root component and display two cards like before. Then try to display all of the shows from the API.

You should now have a cleaner and well structured application. As you can see, single file components allow us to easily separate functionalities & views of our applications.

5. Navigate to `main.js`, what do you see?

6. What do you think each configuration file at the root of your project does?

Lab 5 : Directives

This exercice will show us how to easily trigger a user event.

1. Write a method that toggles the value of isFavorite.
2. Bind the click on the star icon button to toggle favorite state.
3. We want to bind a `is-favorite` class to this button to indicate the favorite state.
4. We want to bind a `is-danger` class to our tag when the status is not 'Continuing'. This class has to set the color to red.
5. Display the list of genres in tags
6. In the form, we want to trigger user input & assign it to a model. The user can trigger a research by pressing enter.
7. (Bonus) We want now to create a custom directive that auto focus on the input.

Lab 6 : Filters

In this exercice we would to like use filter to display only a part of show description.

1. First we would like to limit the description to 35 chars.

2. Implement a function that display the full description when the user click on it.

Lab 7 : Routing

Now that we saw how the router works, we will re organize our architecture to display our elements in a proper way.

1. Install `vue-router` using npm

```
$ npm install --save vue-router
```

2. Create a new route named 'shows' that will render our component. This route has to match the default path.

3. Now we want to have a new route 'favorites' that will display our favorites shows. We also want a link to navigate to this component.

4. This 'favorites' route has to be also aliased with '/starred'

5. In each card, we would like to have a navigation button to navigate to a new route that displays details about the show. The routes have to match with an ID that will be passed as a prop.

6. When a user clicks to favorite a show, we also want the app to redirect him to the details of the show.

Lab 8 : Data fetching

Until now, we hard coded our shows list. This exercice will show us how we can fetch data from an API. We will use the Node.js server from the `resources` folder.

Step 1 - Run the Node.js server

In the server directory, install the npm dependencies with `npm install`.

Then you can launch the server using `npm start`.

The server should be listening on *http://localhost:4000*.

You can check if the server is working by going to `http://localhost:4000/rest/shows`.

The server can respond to theses URL:

- GET `/rest/shows` - list of all shows
- GET `/rest/shows/:id` - Get detail of one show
- POST `/rest/shows/:id/favorites` - set show as favorite or not (use *isFavorite* in the body)

Step 2 - Lab

1. Install the `axios` dependency using npm.
2. Using axios, get the list of shows from our API in the different pages. You'll need to use the `/rest/shows/:id` endpoint for the "Show detail" page.
3. We now want to use the fav or unfav feature from our API and allow the user to refresh the page.

(BONUS) Lab 9 : Plugins

In this lab, we will create a custom plugin that allow us to register our directive and filter.

1. Create a file `plugin.js` which export an `install` function
2. Register the created directive and filter inside the plugin
3. Back in the main.js file, replace the filter and directive registration by the import of our plugin

(BONUS) Lab 10 : Typescript

In this lab, we will create an app written in typescript

1. Create an application with *vue-cli 4* and select typescript

2. Rewrite the app using typescript and the `vue-class-component` library

Lab 11 : Vuex

In this lab, we will use Vuex to manage favorites and show search.

1. First, import `vuex` and configure it with and empty store

2. Then, we want to manage the search of a show with Vuex:

- it should fetch the data through api

- it should handle errors

- it should indicate the loading state

3. Bind the "Shows" page with the store

4. Implement the search feature

5. Create a Vuex module to manage favorites:

- it should have an empty list of favorites by default

- it can add a TV show as favorites

- it can remove a TV show by id

- it should tell if a TV show id is favorites

6. Update both Shows and Favorites pages to work with the store

Lab 12 : Testing

We will now test our application.

Step 1 - Installation

You can choose your framework:

```
vue add @vue/cli-plugin-unit-jest
vue add @vue/cli-plugin-unit-mocha
```

Step 2 - Workshop

Take a look inside the folder `tests`

The plugin gives us an example of a test. You can now write test for each component you have already created.

(Bonus) Lab 13 : ssr

In this lab, we will create a basic SSR server from scratch.

Create an empty directory and initalise npm

```
$ npm init
```

1 - Create a basic ssr with express and vue.

2 - Then, create a Vue instance and render it in the server

3 - Create a template file and update the entrypoint

(Bonus) Lab 14 : nuxt

1 - Transform your app with SSR using nuxt.js