zenika-formation-vuejs

Practical Works

zenika
ARCHITECTURE INFORMATIQUE

Intro

To implements what is described in the main course, you will implements an application allowing you to manage your TV shows.

You will start from a static web page and add features accordingly to the main courses to finally have a modern, rich and reactive application.

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

To improve the design of the application, the CSS framework `Bulma` and the icon font toolkit `font-awesome` are included.

No Javascript code are included in this application, and the external library will be added first via CDN and later with npm + webpack.

Getting started

Bootstraped app

After extracting the received `PW.zip` archive, you should have the following structure:

```
├── index.html
├── css
  └── app.css
├── img
  └── ...
└── js
  └── app.js
```

PW 1 : Setting up Vue.js

In this first PW, we will setting up Vue.js.

1. Import Vue from the CDN. Add in the index.html:

```
<script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js"></script>
```

2. In your browser, print the current version of Vue (`Vue.version`).

PW2 : The Vue instance

Part 1: The first instance

In this PW, we will create our first instance of Vue.

1. Create a Vue instance mounted on the div with the id `app`. Use both way to mount your instance.
2. Display the interpolation of `3 + 5` in the title of the app.
3. Print the Vue instance in your browser console. What do you see ?

Part 2: data attribute

We want to store the title as data of the Vue instance.

1. Update the instance to achieve this behavior
2. Check that you can change the title directly from the instance.

Part 3: lifecycle

1. Add this code in each hook of the Vue instance.

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

2. Update the title from the instance
3. Trigger a `$destroy`

PW3 : Components

In this exercise we'd like to create a Card component.

In this lab, we only want to display some text. We will have conditional display and list display in a later labs

1. Declare a Card component that will be called in our vue instance and it template.
2. This component must contains:

- a title
- a description
- a status
- an image
- a date of creation
- the number of season
- a list of genres
- a boolean isFavorited.

3. Then we would like to associate a template for our component. This one must display our textual data.
4. Instead of declaring the data in the component, we would like to pass this data as props from the parent to the component.
5. From isFavourited, we want to display with a computed property '{{title}} is favorited!' when the boolean is true & '{{title}} is not favorited!' when it is not.

PW4 : Directives

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

1. Write a method that toggle the value of isFavourited.
2. Bind the click on the star icon button to toggle favorites state
3. We want to bind a is-favorite class to this button to indicate the favorites state.
4. Back in our vue instance, we would like to create a collection of series and call our Card component for each of them.
5. We want to bind a warning class to our tag when the status is not 'Continuing'. This class has to set the color to red.
6. In the form, we want to trigger user input & assign it to a model. The user can create a research by pressing enter.
7. Display the list of genres in tags
8. (Bonus) We want now to create a custom directive that auto focus on the input.

PW5 : Filters

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

1. First we would like to limit the description to 35 chars.
2. Implement a function that display all the description when the user click on.

PW6 : Routing

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

1. First we need to import our router via CDN.

```
<script src="https://unpkg.com/vue-router/dist/vue-router.js"></script>
```

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

3. Now we want to have a new component 'Favorites' that will display our series filtered by the fact that if they have been favourited or not. We also want a link to navigate to this component, route has to be named.

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 display us details about show. The routes have to match with an ID that will be passed as a prop.

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

PW7 : Data fetching

We have until now hard coded our shows list. This exercice will show us how we can fetch data directly from an API. We will use the Node.js server that the trainer has sent to you.

Step 1 - installation

You should install this on your computer in order to run the server:

- Node.js (Version > 8.x.x)
- npm (It will be installed at the same time as Node.js)

To install Node.js, you can use NVM (Node Version Manager) to have a easier installation. On Unix :

```
wget -qO- https://raw.githubusercontent.com/creationix/nvm/v0.33.6/install.sh | bas
```

On Windows, go to https://github.com/coreybutler/nvm-windows

When NVM is installed you can do :

```
nvm install node
nvm use node
```

Step 2 - run the Node.js server

In the directory of the server, install npm dependecy with `npm install`.

Then you can launch the server with : `npm start`.

The server is listening on *http:///localhost:4000*.

Check with your browser if the server is working with `http://localhost:4000/rest/shows` url.

The server car 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 *isFavorites* in the body)

Step 3 - lab

1. Import axios in our app:

```
<script src="https://unpkg.com/axios/dist/axios.min.js" />
```

2. Replace the collection of shows that we declared before by a function that will retrieve shows from our API.

3. Create a new page that will display the detail of a show. When we navigate to a show details, we want now a function that we will be called before the component is mount and that get it from the API with the ID.

4. Also we want to fav or unfav our API. For that we will create a function that will call the fav endpoint and pass the id of the show.

PW8 : Vue-CLI

We've seen how to use Vue as a simple library imported via CDN, but now we want to use modern tools like **vue-cli** & **webpack**.

Before anything, make sur you have Node.js and a package manager (**npm** or **yarn**) installed on your computer.

```
$ node -v
10.x.x
$ npm -v
5.x.x
$ yarn -v
1.x.x
```

1. Globally install `@vue/cli`
2. Create a new project inside *TP8* with the following features: Router, Linter / Formatter (ESLint + Prettier), don't forget to install `axios` too.
3. Run a development server
4. Migrate the app in our new architecture and use Single file components.
    - Move `img/` and `css/` into `assets/`
    - Instead of declaring `app.css` from `index.html`, import it in `main.js`
    - Move the `App.vue` template by the `#app` div from our `index.html`
    - Remove predefined style from `App.vue`
    - Replace the content of `index.html` by our content without the `<script>` tags
    - Install *axios*
    - Create a `.vue` file for each component
    - Move the router object in `router.js`
    - Move our previously defined filter and directive in `main.js`
    - Now that component are not registered globally, you'll need to import them where you need them

(BONUS) PW9 : Plugins

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

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

(BONUS) PW10 : Typescript

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

1. Create an application with `vue cli 3` and select typescript
2. Rewrite the app using typescript and the `vue-class-component` library

PW11 : vuex

In this PW, we will use vuex to manage favorites and show search.

1. First, import vuex and configure it with and emtpy store
2. Then, we want to manage the search of a TV with vuex:

- it should fetch the data through api
- it should handle errors
- it should indicate the loading state

3. Bind the the Search component with the store
4. 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

5. Update both Search and Favorites components to work with the store

PW12 : testing

We will now test our application.

Installation

You can choose your framework:

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

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) PW13 : ssr

In this PW, 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) PW13 : nuxt

1 - Transform your app with SSR using nuxt.js