VueJS

Labs

2022-12-11#8c555e7

Introduction

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

You'll start with a basic static webpage and add new functionality based on what we'll see in the main course. The end goal is to have a **modern, rich and reactive** application.

The initial bootstrapped application consists 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. All external libraries (such as Vue.js) will be added first using CDN, then using npm and webpack.

Getting started

Strigo

In a remote environment, using Strigo, you'll find in your **Lab** section a terminal running **Ubuntu 20.04**. If this terminal doesn't output a **code-server password** then open a new terminal by clicking on the **+** button.

In your `home` folder (accessible by the `cd ~` command), you'll find the following structure

```
├── ~/training-vuejs/
  └── resources/
    └── posters/                => it contains the movie images for our
application
    └── server-formation-vue/   => server / API for the labs
  └── workspaces/
    └── lab1/...                => lab folders
    └── ...to lab12
    └── Workbook.pdf            => this document
    └── Slides.pdf              => the training slides
```

Verify your Node version: `node -v` should be > 14.x (otherwise run `nvm use 14`)

Face-to-face training

Extract the ZIP file given by your trainer and you'll find the above structure.

Run the workspaces

Navigate to the `training-vuejs` folder and run the following command:

`npm install && npm run start`

This will start a micro http server to navigate the lab.

And then, depending on your environment (Strigo or your local machine) you can:

- **local**: open Chrome and navigate to http://localhost:4000.
- **Strigo**: open a new Chrome tab to `<Your_Strigo_URL>:4000`

Then navigate to `<URL>:4000/workspaces/lab1` to see the first lab.

You must the *My TV shows* app 👌

Lab 1 : Setting up Vue.js

This lab will allow us to configure 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://unpkg.com/vue@3/dist/vue.global.js` link.

1. Create a `js` folder
2. Then create the app file in it (`app.js`) and print something in it using `console.log()` method.
3. Import the newly created file by adding the following at the end of your `index.html` 's body tag:

```
<script src="https://unpkg.com/vue@3/dist/vue.global.js"></script>
<script type="module" src="./js/app.js"></script>
```

3. Modify the app file to print the Vue version (`console.log(Vue.version)`).

> By using the `type="module"` we can import script file as ES modules and use modern syntax like import and export

Lab 2 : The Application Instance

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

1. Inside of `app.js`, create the Application Instance that will be the entry point to your application. Mount it on the `div` with the id `app`.
2. Print the Application Instance in your browser console. What do you see?

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

3. Add a `title` data attribute.
4. Print this title through an interpolation in the `h1` tag of your `index.html` file using the mustache syntax `{{ title }}`.

Lab 3 : Tooling

By the end of this lab, we will have transitioned our current project to its `Vite` counterpart. Meaning we will take advantage of a cleaner architecture, single-file components, and all the enhancements to the development experience it has to offer.

Create a project

1. Shutdown your previous server launch in the first lab by using `Ctrl+C` command (the one you used to see the `lab1` and `lab2`).
2. To get started with Vite + Vue, you could simply run:

```
npm init vue@latest
```

and select the following options:

- Eslint
- Prettier

For the sake of the example, **we've already created** a project using `npm init vue@latest` command. You will find it in the `workspaces/lab3` folder.

For now, we have a root `App.vue` component that renders the same HTML file we had earlier but using Single File Component (SFC) syntax.

1. Start your lab 3 development server and navigate to the link displayed in your console.

   ```
   # Go to the `workspaces/lab3` folder
   $ cd lab3

   # Start the dev server
   $ npm install && npm run dev
   ```

   Your view will now be automatically updated whenever you modify a component thanks to the **Hot reload** provided by **Vite**.

2. Open the `src/main.ts` file and see how the Vue instance is created.
3. What do you think each configuration file in the root of your project does ?

Install Vue Devtools

For Vue 3 support, you will install vue-devtools.

- Chrome extension
- Firefox extension

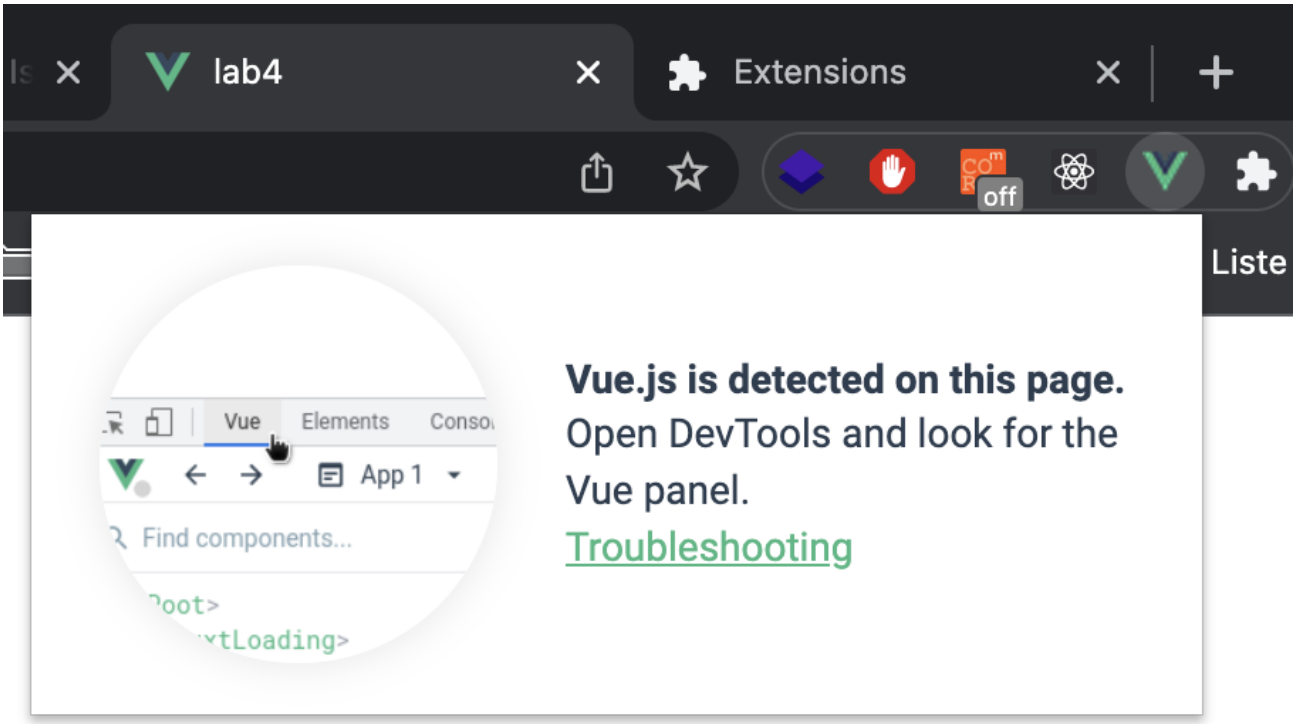
To install or update the Firefox beta version of the devtools, go to one of repository releases and download the xpi file.

- Repository releases

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

   ```
   # In your lab3 project directory
   $ npm run dev
   ```

   In your browser, check that your application is detected by the extension.



2. Open your browser devtools (f12) and click on the Vue tab.

   You must see one component here: `App`.

Lab 4 : Syntax

> ⚠️ You will have to launch the `server-formation-vue` so that the images are served correctly. Launch it by going to the `training-vuejs` folder and running `npm run start`

In this lab, we'll turn our Vue application into a true dynamic application using the syntax we've seen. We will use local data during this lab.

1. First, import TV shows data by adding an import statement in your `App.vue` file. Add a new reactive property `shows` to the `data` Option of the component `App.vue`

```
// src/App.vue

import shows from '../../../resources/server-formation-vue/shows.js';

export default defineComponent({
   data() {
      return {
         // ...
         shows
      }
   }
})
```

2. Iterate through the list of TV shows to display them all using the `v-for` directive, replacing all

```
<div class="card card-result">...</div>
```

sections with one, holding the v-for directive.

3. Replace the current hard-coded text displaying the information for each TV shows in the previously iterated list. Use interpolation to display:
    - title
    - status
    - date of creation
    - the number of seasons
    - comma separated list of genres
    - description
4. Bind the image `src` attribute to the TV show `images.poster` key.

5. Bind an `is-favorite` class to the star icon to indicate favorite status (corresponding to the `user.favorited` key).

6. Bind an `is-danger` class to the tag when the TV show status is not `Continuing`.

Now create a custom directive to automatically focus the input on page load.

1. Add the code in the `src/main.js` file using the `app.directive` syntax seen in the slides.

2. Use the directive on the `input` element.

Lab 5 : Components

In this exercise we'll create a `CardShow` component.

1. Create a `CardShow.vue` component in the directory `src/components`. This component will declare a `show` prop. The prop must be typed as an `Object`.

2. Add a `template` for the `CardShow.vue` component. To do so, move the html template from `App.vue` of the element having the class `.card-result`:

```
<template>
    <div class="card card-result">...</div>
</template>
```

3. Bind the `show` prop from `App.vue` to `CardShow.vue`

4. In `CardShow.vue`, based on the value of the `show.user.favorited` property, create a **computed** property `{{ title }} is your favorite!` or `{{ title }} is NOT your favorite!`. Display this property instead of the show title:

```
  <p class="card-header-title">{{ titleFavorite }}</p>
```

We will now toggle the favorite status. In order to respect the props immutability and the data ownership, we need to propagate the event to the parent that owns the data. 5. Emit a `toggle-favorite` event when we click on the star icon having the class `.card-header-icon`. From `App.vue`, intercept the `toggle-favorite` event by using the `v-on` directive and update the favorite property accordingly.

We will now use the input to filter the shows. 6. Bind a `v-model` directive to a `searchTerm` data to filter shows with the existing input field. You can use the `filter()` function on the `data` imported. Filter only on the `title` field of each show. Finally, on the `v-model`, apply the `.lazy` modifier to trigger the search only when the user has finished writing.

```
<script>
  import shows from '../../../resources/server-formation-vue/shows.js';

  export default {
    // ...
    data: () => ({
      shows,
    }),
    computed: {
      filteredShow () {
        return this.shows.filter(show =>
show.title.includes(this.searchTerm))
      }
    }
  }
```

```
</script>
```

Lab 6 : Plugins

In this lab, we'll create a custom plugin that will register the `v-focus` directive we created earlier.

1. Create a `plugins` folder
2. Create a file `focus.js` which export an `install` function
3. Register the directive `v-focus` inside the plugin
4. Back in the `main.ts` file, install the plugin by using the `app.use` method

Lab 7 : Routing

Now that we saw how the router works, we will reorganize our architecture to display our components properly.

1. Install `vue-router` using npm

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

2. Create the router instance inside `src/router.js` with an empty routes array. Use the `createWebHistory()` method as history mode. Export the router instance and use it in your `main.js` file as a plugin.

3. Create a new directory `pages` inside `src`. Create a new component `ShowList.vue` inside the `src/pages` directory. Move all the tag `<div class="hero-body">` and its children from `App.vue` to the `ShowList.vue` component.

4. Create a new route named `shows` which will render the page created above to display our list of shows. This route should match the default path: `/`.

5. Copy the page `ShowList.vue` to create a new page `ShowListFavorites.vue`. Update the component to display only starred shows.

6. Create a new route named `favorites` which will display the component created above. The component associated to the route should be lazy-loaded. This route should match the path `/favorites`. Add a link in the toolbar in `App.vue` to access this page.

7. Create a new page `ShowListItemDetails.vue` in `src/pages`. This page will have a props `showId` and will find the show when mounted. It will only display the title of the show and the `CardShow` component.

8. In each card, we'd like the title to be clickable to navigate to a new route (`/shows/:id`) that displays the page created above. The route must match an ID that will be passed as a prop to the detail component. The component associated to the route should be lazy-loaded.

9. Catch any other route by using a regexp after the param and redirect to the homepage own this scenario. For more details: https://router.vuejs.org/guide/essentials/dynamic-matching.html#catch-all-404-not-found-route

Lab 8 : Data fetching

Until now, we imported a hard-coded shows list (the `shows.js` file). This exercice will show us how we can fetch data from an API. We will use our Node.js server from the `resources/server-formation-vue` folder.

Explore the API

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

The server can respond to these URLs:

- GET `/rest/shows` - list all shows
- GET `/rest/shows/:id` - Get details of one show
- POST `/rest/shows/:id/favorites` - toggle favorite status of a show (use *isFavorite* in the body with a Boolean value). For instance:

```
POST http://localhost:4000/rest/shows/12345/favorites
{
isFavorite: true
}
```

Interact with the API

1. Install the `axios` dependency using `npm install axios`.
2. Display the list of shows by retrieving the data returned by the `/rest/shows` API endpoint.
3. Display the details of a show by retrieving the data returned by the `/rest/shows/:id` API endpoint.
4. Toggle the favorite status of the shows by using the `/rest/shows/:id/favorites` API endpoint.

Lab 9 : Store

In this lab, we will use Pinia to set up the state management of our application.

All API calls will be made through the store, as well as searching for shows and retrieving favorited shows.

Thus, our store will have a state, some getters and some actions.

You will find below a relatively step by step guide and a proposal to properly structure your store. As often in software development, there are several ways to solve a problem. So feel free to structure your store differently if you wish.

1. Install Pinia

Install Pinia by following the getting started guide of the documentation. Add the `pinia` dependency and use the `createPinia` method in the `main.ts` file to instantiate the plugin.

2. Define the store

Define a store in a `src/stores/index.js` file:

```
import { defineStore } from 'pinia'

export const useStore = defineStore('main', {
  state: () => {
    return {
    }
  },
  getters: {
  },
  actions: {
  },
})
```

2.1 State

Add some properties to the state:

- `shows` : this property is an array of shows and will store the shows retrieved from the API
- `searchTerm` : this property is a string and will store the term entered by the user during the search

See https://pinia.vuejs.org/core-concepts/state.html

2.2 Getters

Add some getters:

- `searchedShows` : this getter will return a list of shows matching the search criteria entered by the user
- `favoriteShows` : this getter will return the favorite shows

See https://pinia.vuejs.org/core-concepts/getters.html

2.3 Actions

Define some actions:

- `fetchShows` : this asynchronous method retrieves the list of shows via the API and uses the API response to set the `shows` property of the state
- `toggleFavorite` : this asynchronous method takes a show ID in argument. Its role is to toggle the favorite status of the show, updating the `shows` property defined in the state. It also makes a POST request to the API to persist the new favorite status.

See https://pinia.vuejs.org/core-concepts/actions.html

3. Use the store

Use the mapActions helper to fetch the shows and toggle the favorite status.

Use the mapWritableState helper to update the `searchTerm` property defined in the state.

Use the mapState helper to retrieve the list of shows from the state.

(Bonus) Lab 10: Testing

Unit testing with Vitest

vitest is are already installed.

Take a look inside the `src/components/__tests__/CardShow.spec.js` file.

Run the `npm run test:unit` command. The unit tests should pass.

Complete the tests that are empty.

E2E testing with Cypress

Like Vitest, Cypress is already configured. Cypress tests are inside the `cypress/e2e` folder.

In Strigo, it is not possible to start Cypress in interactive mode. Therefore, run the `test:e2e:ci` command to run the tests in headless mode.

Ask yourself the following question: which other tests would make me confident?

Write these tests

Lab 11 : Typescript

In this lab, we will gradually migrate our project to Typescript.

To make it easier for you to get started, Typescript is already configured.

1. Create a `src/model.ts` file. Inside this file, export a minimal `ShowInterface` that contains the required properties of a show.

```ts
// src/model.ts

export interface ShowInterface {
  id: number
  title: string
  genres: string[]
  images: {
    poster: string
  }
  seasons: string
  user: {
    favorited: boolean
  }
  description: string
  creation: string
  status: string
}
```

2. Rename the `src/store/index.js` into `src/store/index.ts` and make it Typescript compliant.

3. Finally, use Typescript in your components inside the `pages` and `components` folders. Start by adding the `lang="ts"` attribute to the `script` tag of your component, then add the necessary return types so that the Typescript server no longer reports errors.

(Bonus) Lab 12 : Composition API

TP 1: beginner

Open the `workspaces/lab12-1` folder.

Transform the 2 components inside the application to use only the setup syntax.

TP 2: intermediate

Open the `workspaces/lab12-2` folder.

The main goal of this lab will be to migrate from the Options API to the Composition API.

1. Refactor in our components and pages the `data` method to use the Composition API. Make sure not to break anything.
2. Do the same for the `methods`. Don't forget about the non-existing `this`.
3. Now do it for the `computed`. Make sure to replace `this` to use the component `props`.
4. Create a composable for communication with the API.

(Bonus) Lab 13 : nuxt

1. Create a nuxt app from scratch
2. Transform your app with SSR using Nuxt