Web Development for IT engineer

Lab3 : VueJs

A) Quelques precisions:

- #vue create → Utilise Vue CLI, basé sur Webpack
  Recommandé pour les projets Vue 2 ou anciens projets Vue 3
- #npm create vue@latest → Crée un projet Vue 3 « moderne »
  Utilise Vite en interne, configuré par l'équipe Vue officielle
- npm create vite@latest → Crée un projet Vite générique
  Tu choisis Vue (ou React, Svelte, etc.) pendant l'installation
- Comme Webpack, Vite permet de bundle les fichiers JS/CSS et supporte les « .vue ». Le fonctionnement interne par contre est différent rendant Vite plus léger au lancement.

B) Pour maintenir une continuité des concepts de bundling et autres. Pour ce TP vous utiliserez la commande « vue create mon-projet ».

Vue: practical activity, part n°1

Learning outcomes

- Understand how Vue create related to other well-known tools (webpack, babel...).
- Practise the Vue's essentials (basic components, templates, scoped styles...).
- Organise your source code in various files and folders.
- Build reusable components (isolation, single responsibility, props and slots).
- Do not repeat yourself (DRY!)
- Work with Promise and events inside a Vue application.
- Import third party packages providing components with npm.
- Commit and reset staged changes with git.

Expected result

At the end of the tutorial series, you got a lightweight SPA client for exploring mails and contacts from the Outlook product, through the Microsoft Graph API. This SPA targets both computers and phones, requiring responsive capabilities and appreciating resilience to network issues.

The first tutorial of the Vue's series focuses on setting up the project, understanding tools behind the Vue (those discussed in the last tutorial), setting up the layout and adding some shared reusable components.

Prepare your development environment

In a general manner, your productivity and code quality are affected by used tools. Personally:

- vscode as my primary code editor, especially suitable for JavaScript development.
- vue-devtools extends browser's debugging capabilities (beta channel for Vue3).
- npm, node, typescript and the rest of well-known JS tools...

Vue project setup

Question 1: That is the main difference between local installation and global installation of packages with npm? What kind of packages do you generally install locally? What kind is generally installed globally?

Exercise 1: Create a new Vue project (called `vue-oauth-microsoft-graph`). Opt for the Vue3 recipe that relies on webpack and babel for the build chain.

Question 2: Webpack is internally used. Why is it required to deal with both multiple JavaScript files and special extensions like `.vue`?

Babel is configured by default with `@vue/cli-plugin-babel/preset`, as specified in `babel.config.js`. By reading the package's documentation, you see it uses the `browserslist` configuration defined in `package.json`.

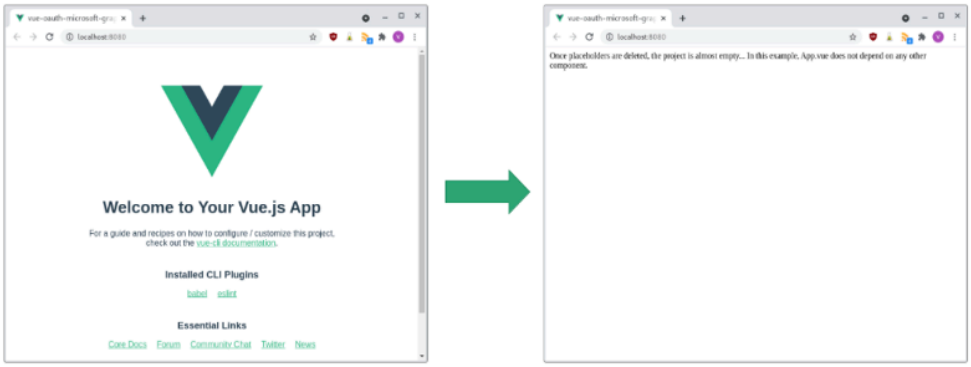
Question 3: What is the role of babel and how `browserslist` may configure its output?

Question 4: What is eslint and which set of rules are currently applied? The eslint configuration may be defined in a `eslint.config.js` or in `package.json` depending on the setup.

Exercise 2: Run `npm run serve` and open the app in your browser. Remember that npm looks at the `package.json` file (specially the `scripts` object) to find which command to execute.

Did you notice that `npm run serve` launches a program called `vue-cli-service`? This is a cli locally installed by npm inside the `node_modules` folder. This dependency is dedicated to development experience, so it is a `devDependencies` in your `package.json`.

Exercise 3: The newly generated project contains a few placeholders. Cleanup your project so it does not contain neither useless assets, nor the hello world. In other words, delete `HelloWorld.vue`, its related assets and all its references. As at the end of each exercise, the vue cli should not report any error or warning.



On the left, the vue app with placeholders. On the right, result after cleanup.

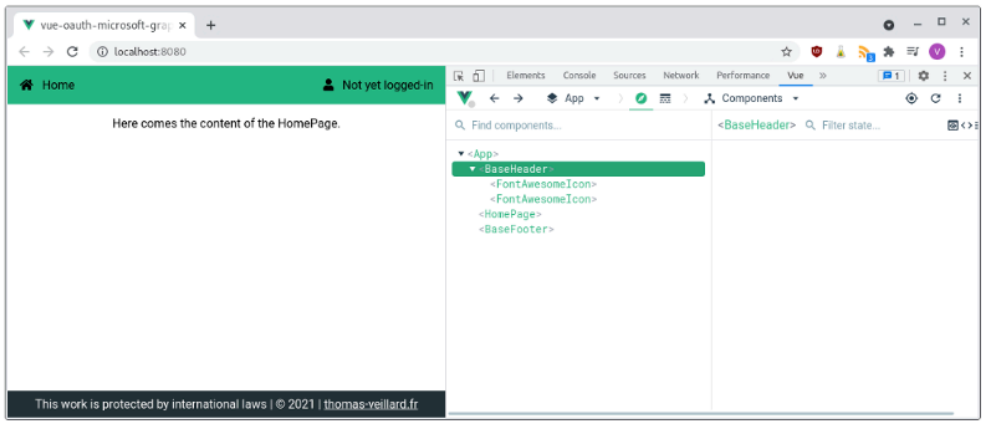
By default, only the `src/components` folder is intended for storing `.vue` files (aka. vue components). But we are free to use other directories depending on the nature of each component. Let's add `src/pages` to our code base.

- `src/pages` contains top level components that produce a particular page (ex. the home page, the index of mails page, the contact page...). Those components are intended to be mounted with vue-router in the future.
- `src/components` contains shared components required by pages or other components (ex. navigation headers, buttons, user's cards, short preview of an email...).

Exercise 4: Create the `HomePage` component inside the right folder. Do not spend too much time on the template content, as it could be a simple sentence. Import it inside `App.vue`.

Commit changes with message « Ex 4: create HomePage component »

Base layout



Basic possible visual at the end of the next exercise and its components tree.

Tips: use font awesome icons with Vue 3,

Exercise 5: Let's begin with the root component, formally `App` (in `src/App.vue`). Replace its template with the following content and create the missing components. Add some content to the header (ex. fake home link, fake user name...) and legal credits to the footer. Eventually, polish the looks and feels with scoped CSS.

```
<!-- template of file src/App.vue -->
<template>
  <div>
    <base-header />
    <home-page />
    <base-footer />
  </div>
</template>

<!-- do not remove the <script /> markup -->
```

Commit changes with message « Ex 5: create BaseHeader and BaseFooter »

Question 5: What is the difference between scoped and non-scoped CSS?

Exercise 6: In order to keep the root component `App` as simple as possible, extract everything related to the layout into a `BaseLayout` component. Using the slot API, allow `BaseLayout` to receive children (to be rendered between the header and the footer).

Tips: If you integrated font awesome, try extracting most logic out of `App`. When it comes to configure third party dependencies, I generally work with ES-modules inside a `src/lib` folder. Example: `src/lib/fontAwesome.js`.

```
<!-- template tag of src/App.vue -->
<template>
  <base-layout>
    <home-page />
  </base-layout>
</template>
```

Commit changes with message « Ex 6: create BaseLayout that uses slot API »

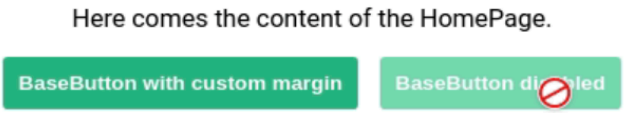
Reusable BaseButton

For the need of the UI, let's create a `BaseButton` component. Basically, this is just a styled `<button>`. Its usage should be mostly the same (possibility to pass children, styles, classes, `role="button"` or `role="submit"`...).

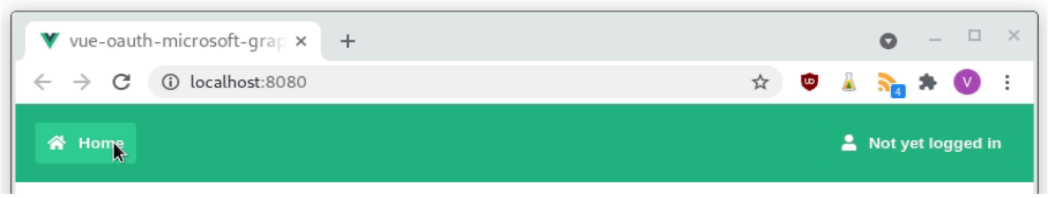
Question 6: How behaves non-prop attributes passed down to a component, when its template has a single root element? **Tips**: it is well documented by vue, but you can also try it youself by passing the `style` attribute with a straight visual effect.

Exercise 7: Implement such a `BaseButton`, animated on hover and focus. Do not forget the disabled state. You may try these buttons on your `HomePage` for now.

Slot API documentation : https://vuejs.org/guide/components/slots.html

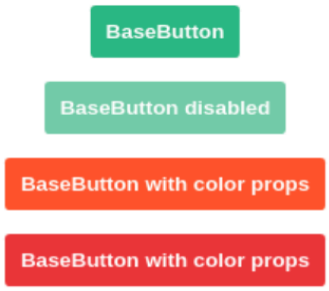


Commit changes with message « Ex 7: create BaseButton with primary color »



Tips: it makes sense to use them in the navigation in header.

Exercise 8: Add the `color` prop to `BaseButton`. This prop accepts one of `'primary'`, `'warn'` or `'danger'` values. It defaults to `primary` and you should validate the given value matches the enum. Then, dynamically apply styles to the button based on that prop.



Tips: in a first time, ensure you can pass the props from `PageHeader` template to the `BaseButton` component (ex. by temporary rendering the color name in the template). Then, use that value to apply some conditional styles (2 proposed solutions above). Remember the DRY principal (Don't Repeat Yourself) and do not duplicate code sections.

Solution 1: using CSS classes (simpler)

You may split your scoped CSS in 2 parts: the styles common to all components, and specific classes for each color in the palette. So, the component's style could look like:

« Props » documentation : https://vuejs.org/guide/components/props.html

```
<style scoped>
.button {
  /* css properties common to all buttons */
}

.button-primary {
  /* css properties specific to the primary color */
  background-color: #42b983;
}
</style>
```

Solution 2: using CSS variables & computed properties

Declare in a single place the allowed values for the colour enum and its corresponding colours. On my side, I just declared the above object and always take it as a single source of truth in `BaseComponent`. In other words, this object contains the colour palette indexed by names I use as reference everywhere in the component.

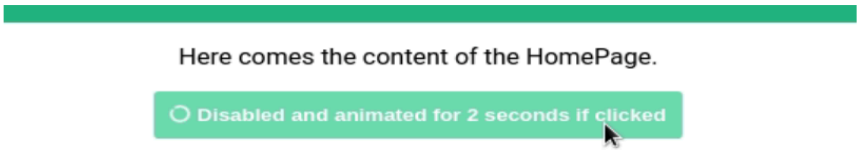
```
const colorPalette = {
  primary: { bg: '#42b983', hoverBg: '#4cce93', focusBorder: '#47d696' }
  warn: { bg: '#ff5722', hoverBg: '#ff7043', focusBorder: '#ff8a65' },
  danger: { bg: '#e53935', hoverBg: '#ef5350', focusBorder: '#e57373' },
}
```

Then, you need a way to apply variables to your CSS. In short, this is a conjugaison of style binding, CSS variables, computed properties and scoped styles.

Commit changes with message « Ex 8: color palette and prop for BaseButton »

Reusable AsyncButton

Now, let's add `AsyncButton` built on top of `BaseButton`. It prevents the user from clicking multiple times on the button while a Promise is in progress. That Promise is returned by the parent's `onClick` listener.



Here you have its source code (if you did not install FontAwesome, just replace the `<font-awesome-icon />` by a simple text, while preserving the `v-if` directive.

```
<template>
  <base-button
    :disabled="isPending"
    :color="color"
    @click.stop.prevent="handleClick"
  >
    <font-awesome-icon
      v-if="isPending"
      :icon="['fas', 'circle-notch']"
      pulse
    />
    <slot />
  </base-button>
</template>
```

```
<script>
import BaseButton from './BaseButton.vue'

export default {
  name: 'AsyncButton',
  components: { BaseButton },
  inheritAttrs: false,

  props: {
    color: {
      type: String,
      default: 'primary'
    }
  },

  data () {
    return {
      isPending: false
    }
  },

  methods: {
    handleClick () {
      const originalOnClick = /** @type {() => Promise<void>} */ (this.$att
      this.isPending = true
      originalOnClick().finally(() => { this.isPending = false })
    }
  }
}
</script>
```

Exercise 9: Add a button to the `HomePage` that is disabled for 2 seconds each time it is clicked. According to the above code, this just means the `@click` event listener attached to the instance of `AsyncComponent` instance returns a Promise that waits for 2 seconds before resolving. You can create such a Promise using its constructor and a setTimeout. Also, please write the event handler inside a dedicated method since at is a bit complex.

Commit changes with message « Ex 9: add AsyncButton »

Exercise 10. Change the behaviour of the previous button, so its waiting time increases by one second each it is clicked. Because `AsyncButton` waits for any promise, whatever how long it takes to resolve, you do not need and you should not change it. Instead, keep trace of the number of clicks in the internal state (data) of the `HomePage` component (see the counter app example) and use it while forging new promises.

Commit changes with message « Ex 10: slowing down the button on click »

Question 7: Analyse how works the `AsyncButton`. How the child component is aware of the returned Promise by the parent onClick handler? When is executed the callback passed to `.finally()`? Why use `.finally()` instead of `.then()`? Etc.

Question 8: Which bug is introduced if `inheritAttrs: false` is missing or set to `true` in `AsyncButton`? Why?

To continue

Next time, you will handle Oauth2 authentication against the Microsoft Graph API, fetch the user identity, use it at various locations of the UI and eventually implement some routes. Maybe improved state management will also be useful...

« Event » documentation https://vuejs.org/guide/components/events.html