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.

The Vue CLI as already committed the newly generated project.

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?

When you install a package locally using npm, it is installed in the `node_modules` directory of your specific project. Local installations are typically used for project-specific dependencies. These are packages that are required for the project to run successfully.. Local installations ensure that the project has all the necessary dependencies and can be easily shared with others.

Global installation, on the other hand, installs packages globally on your system, outside of any specific project. Global installations are generally used for tools or utilities that are meant to be accessed from the command line interface (CLI) regardless of the current project context. For instance, development tools like Vue CLI, create-react-app, or utilities like nodemon are typically installed globally.

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

A webpack is essential for the Vue CLI because it builds multiple JavaScript files into a single output file. The .vue extension is very useful because it gathers functions, scripts, templates all in one file too.

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?

Babel is a javaScript compiler that enables developers to write code with a very recent set of rules and ignore the risk of compatibility with older browsers. Browserlist is used by libraries such as babel to specify which browsers your web application can run in.

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.

Eslint is a program that can analyze your JavaScript code and point the errors in it. The rules depend on the project configurations, but it concerns a lot of aspects such as the variable naming, syntax errors and many more. It helps developers to follow good guidelines while writing code.

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.

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`.

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.

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

Scoped CSS is made in Vue.js to define a style for a specific component, doing this vue puts an unique attribute to the component,isolating the style within the component. However, for a non-scoped CSS style is applied globally, which can lead to a lot of conflicts.

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).

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.

When non-prop attributes are passed down to a component, they will be applied to the root element of the component template. This only happens when there is only a single root of the component's template.

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.

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.

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()`?

The asyncButton works asynchronously, it has a color prop and a “ispending”. When clicking the button, the onclick function is activated (parent component) and returns a promise. When the promise is pending “isPending” is true now, which disables the button.
The callback .finally() is used to reset the “isPending” to false whether it is good or not. The finally() is used instead of then() is used to get back to the original state of the button while not depending on the result of the asynchronous task so it will work at all times. The AsyncButton(child) is aware of the returned promise by directly calling it’s parent function (OnClisk).It does not need to explicitly wait for the Promise since the Promise itself is returned by onClick..

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

If it is set to true or missing, any attributes that are not defined as props in Asyncbutton are added to the root element of the component. All of this can create a bug when there are attributes not supposed to use AsyncButton, they are passed to the button element.