Vue : practical actity

Subject

The SigninButton component

Question 1

> Why you should not commit credentials on git?

Because it's a public repository and everyone can see it. It's a security issue.

Question 2

> Why may you want different configurations depending on the environment? Give an example.

Because you may want to use different API keys depending on the environment. For example, you may want to use a test API key for the development environment and a production API key for the production environment.

Exercice 1

> I already wrote helpers on top of msla (see the file src/lib/microsoftGraph.js above). Import them for use in your SigninButton component. For now, SigninButton stores the resolved user inside its own state (the data property of your component) and displays it in the template.

So to start the exercice, we copy the repository on Lab3 and we install the dependencies with `npm install`. Then we create a new component `SigninButton.vue` in the `src/components` folder.

We take the api code in the Azure link and we put in the .env.development.local file.

So to correctly use the api, we need to check 2 or more things :

- we have on your machine the right version of azure/msal-browser (3.2.0)
- we have the right VUE_APP_CLIENT_ID in the .env.development.local file (196ef638-917b-4f82-b655-7d3c80154af4)
- we have to verify that the redirectUri in microsoftGraph for exemple in my machine is http://localhost:8080/ (the port can change depending on the port used by the server)

Then we can start the server with `npm run serve` and we can see the button on the page.

If there are a bug, check on the browser's console

First solution, use props and events through component tree.

Exercice 2

> Use props to share the user with both SigninComponent and HomePage. Then use events to update the shared user from SigninComponent. Eventually, display the user name in HomePage.

So we modify the page signinButton.vue to add the props and the event.

We emit the event in the signinButton.vue and we catch it in the baseHeader.vue and we pass it in the baseContent.vue and finally in the homePage.vue.

Question 3

> While being a well-working solution, it suffers from maintainability issues. Please expose and discuss them.

The problem is that we have to pass the user in all the components. If we have a lot of components, it can be a problem.

Second solution, use provide and inject.

Exercice 3

> Replace every needed user props by an inject and remove transitional props that are not useful anymore. For now, you can keep the userChanged event chain. Be sure to make your inject reactive.

So we modify the page signinButton.vue to add the provide and the inject. We remove the props and the event. We use the provide to share the user with the other components inside just the basecontent so we passe the user in the footer to use it.

Question 4

> What is the bug if the inject is not reactive?

If the inject is not reactive, the user will not be updated when we change it.

Third solution, use a store library like vuex.

Exercice 4

> Move the user property from the App component state to a store managed with vuex. Drop the event chain and let SigninButton directly call a mutation in the store.

So we modify the page signinButton.vue to call a mutation in the store. We remove the provide and the inject. We remove the props and the event.

Question 5

> Build a comparison table between the various state management strategies available, especially about pro and cons. Optionally, feel free to explore other ways not covered in that tutorial.

Strategy	Pros	Cons
Props and events	Easy to implement	If we have a lot of components, it can be a problem
Provide and inject	Easy to implement	If we have a lot of components, it can be a problem
Vuex	Easy to implement	If we have a lot of components, it can be a problem

Question 6

> Imagine a developer in your team suggests to exclusively manage the state with stores. Therefore, it recommends not to rely on props and provide anymore. Would you accept this? An argued answer is expected.

I think it's a bad idea because it's not a good practice to use only one strategy. We have to use the strategy that is the most adapted to the situation.

Client-side routing

Exercice 5

> Install vue-router v4+ according to official documentation. Make sure to pick the version compatible with Vue3, (also compatible with vue-router v4+)

So we install vue-router with `npm install vue-router@4` so the file package.json is updated with the new dependency.

Exercice 6

> Using the Getting Started guide, add / and /conversations routes. Define the router inside its own ES module (ex. src/router/index.js). For now, the ConversationsIndexPage should just contain a placeholder (a small message to visually see that the route works fine).

Do to this we change the (main.js)[.\vue-oauth-microsoft-graph\src\main.js] file. We add the router view in the app.vue file. We deside to create a new page nammed About and we add the route the main.js file.

Exercice 7

> You probably tested your routes by directly changing the URL. However, a normal user clicks on links and buttons. Help them by adding a link to the header that targets the /conversations routes. That link should only be visible for logged users. It must use the .

So we modify the baseHeader.vue file to add the link to the about page and the conversation page.

Question 7

> What is the performance difference between:
>
> - `<a href="/conversations">Conversation</a>`
> - `<router-link to="/conversations">Conversations</router-link>`

The difference is that the first one will reload the page and the second one will not reload the page.

Exercice 8

> In the case the user tries to access /conversations without being logged-in, it should be redirected to the home page. Use a guard to implement this behaviour.

So we modify the (index.js)[.\vue-oauth-microsoft-graph\src\router\index.js] file to add the guard with the beforeEnter just for the page conversation.

Exercice 9

> Implement the /conversations/:id route. For now, the newly created ConversationShowPage component should access the id given in the URL and display it inside the template.

So we create a new page ConversationShowPage.vue and we add the route in the index.js file. So when you click in the conversation section, you arrive in a new route with the id of 6, (it is just a random number).