Godfrin
Louis
N° étudiant : 20210589

PARIS PANTHÉON-ASSAS UNIVERSITÉ

Interface Development and Design

TP 4

Professor: NIMAGA, Mahamadou

Godfrin
Louis
N° étudiant : 20210589

Table des matières

Godfrin
Louis
N° étudiant : 20210589

Questions:

Question 1: Why should you not commit credentials on git?

Because they may be exposed to others, even unintentionally, private and confidential data. Git repositories can be public or shared, and exposing sensitive information can lead to security risks like unauthorized access, data breaches, or even financial losses if those credentials are tied to critical services.

Question 2: Why may you want different configurations depending on the environment? Give an example.

Different environments require different configurations, and so we can modify a part or try something, without damage the production part. For example, in a development environment, we can use a local database with debugging features enabled. But in production, we'll use a real database, with optimized settings.

This separation makes it possible to work on the code without impacting real user data, and to avoid any problems with production operations.

Question 3: While being a well-working solution, it suffers from maintainability issues. Please expose and discuss them.

Some well-working solutions can suffer from maintainability issues as they grow.

First, more we add features, more the codebase becomes harder to understand and maintain.

Second, coupling issues. Tight interdependencies between components mean that a change in one part of the system can have huge consequences.

Third, scaling systems can struggle to scale as demands grow.

Finally, testing and deployment are problematic. Making small changes requires redeploying the entire system, which increases downtime and the risk of introducing bugs.

Decoupling components through microservices can help reduces these issues.

Godfrin
Louis
N° étudiant : 20210589

Question 5: Build a comparison table between the various state management strategies available, especially about pros and cons.

Strategy	**Pros**	**Cons**
Props & Events	Simple, no extra libraries required	Can get messy with deep or wide component trees
Vuex (Global Store)	Centralized state, easy to debug, and track	Learning curve, can be overkill for smaller apps
Pinia (Alternative to Vuex)	Simpler API, easier to use than Vuex	May not offer all the features needed for complex apps
Provide/Inject (Vue API)	Simple for parent-child communication	Less clear, not reactive for deep hierarchy changes
Local Storage/Session Storage	Persistent between sessions	Limited to client-side, potential security risks

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?

I would say no. Stores are powerful tools that centralized state management and can work even if we completely remove props and provide/inject. But it might not always be the best approach. Props and injection patterns are lightweight and intuitive for simple parent-child communication. On other hand, using only stores for all state management could add unnecessary complexity.

So, for me the use of stores is for complex state but rely on props and provide/inject when it’s for little elements, to avoid over complicating the codebase.

Godfrin
Louis
N° étudiant : 20210589

Question 7: What is the performance difference between:

- Conversation
- <router-link to="/conversations">Conversations</router-link>

The difference between these two elements is simple.

- **<a href="/conversations">** This way of doing it triggers a full-page reload.

  The browser makes a new request to the server, to fetch the new page. It's consuming in resources and can lead to slower navigation and a less smooth user experience.

- **<router-link to="/conversations">** This way of doing it, uses Vue Router to perform client-side navigation. This way we don't ask for reloading the entire page. And the result is a faster transitions and that preserves the app state, providing a better user experience.

To conclude, <router-link> offers better performance as it avoids full page reloads.

Data that I use for this lab:

I will display the user.username in this lab, so l.godfrin@yahoo.fr until exercise 8, afterward I will display user.name, so louis G.

```
User already signed in:        microsoftGraph.js:48
  {homeAccountId: '00000000-0000-0000-70c7-02fa9ab3
  365d.9188040d-6c67-4c5b-b112-36a304b66dad', envir
▶ onment: 'login.windows.net', tenantId: '9188040d-
  6c67-4c5b-b112-36a304b66dad', username: 'l.godfri
  n@yahoo.fr', localAccountId: '00000000-0000-0000-
  70c7-02fa9ab3365d', …}
```

Exercise 1:

For this lab we need to install azure:

.env.developpement.local -> .env to erase issues inside of the code that was not understand the file.

Godfrin
Louis
N° étudiant : 20210589

Connexion to microsoft services with Signin button:

Confirmation that it works:

Godfrin
Louis
N° étudiant : 20210589

Exercise 2:

Before I already had something inside the button to say hello to the user inside of the SiginButton Component.

```
<template>
    <div>
      <button @click="signIn">Sign In</button>
      <div v-if="user">
        <p>Welcome, {{ user.username }}</p>
      </div>
    </div>
  </template>
```

That why we can see it in the screenshot for the connexion in exercise 1.
But I added a method handleUserSignedIn(user) in HomePage to receive the informations of the user that are emit from SiginButton Component.

And we can see in the screenshot bellow, that works.

HomePage changements:

```
<template>
    <div>
      <SigninButton @user-signed-in="handleUserSignedIn"/>
    </div>
      <div v-if="user">
        <p>Welcome, {{ user.username }}</p>
      </div>
```

```
    methods:
    {
      // Retrieve user from SigninButton
      handleUserSignedIn(user)
      {
        this.user = user;
        this.isDisabled = true;
      },
```

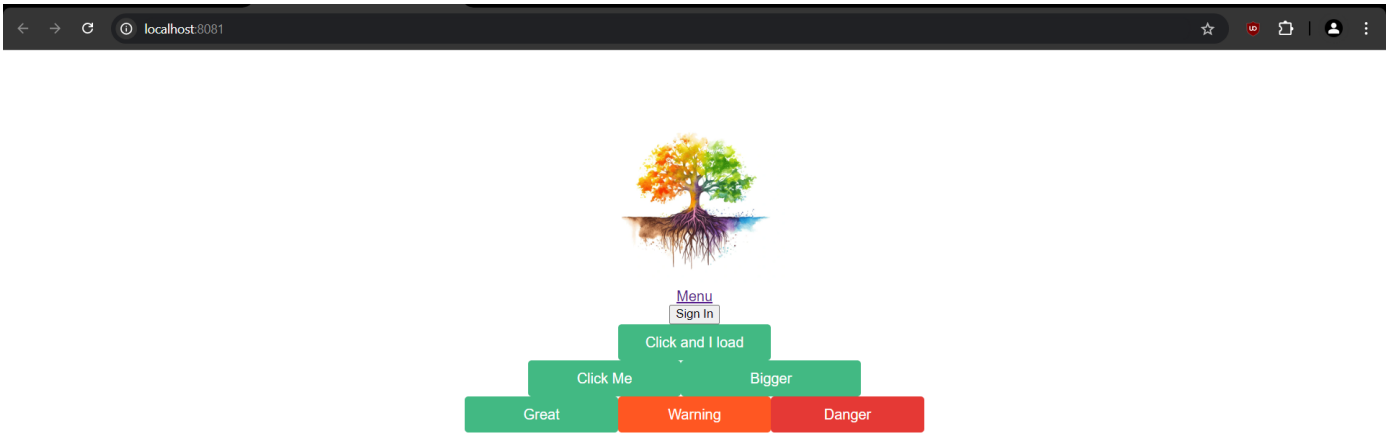
Godfrin
Louis
N° étudiant : 20210589
SigninButton template, now we transmit the information to the HomePage:

```vue
<template>
    <div>
      <button @click="signIn">Sign In</button>
    </div>
  </template>
```

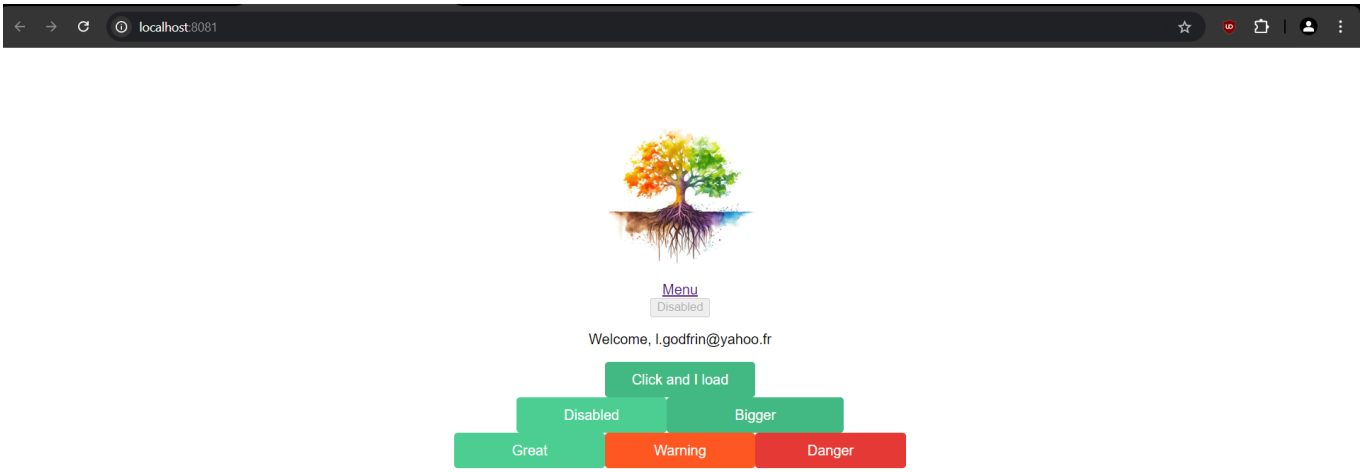
```
        try
        {

          this.user = await signInAndGetUser();
          this.$emit('user-signed-in', this.user); // Transmit the user to HomePage
        }
```

Result:

Before login:



Godfrin
Louis
N° étudiant : 20210589
<u>After login:</u>



I add a disable state after login, to have a better user experience. So we have the information of the user inside the SiginButton Component and the HomePage Component. The HomePage now display the information of the user.

Godfrin
Louis
N° étudiant : 20210589

Exercise 3:

In HomePage:

import { reactive } from 'vue';

```
setup()
{
  const state = reactive({
    isCut: false,
    user: null,
  })
  const setUser = (user) => {
  state.user = user;
  state.isCut = true;
  };

  return {
    state,
    setUser,
  };
},

provide()
{
  return {
    state: this.state,
    setUser: this.setUser,
  };
},
```

In SiginButton (Injection) :

```
export default
{
  name: 'SigninButton',
  inject: ['state', 'setUser'],
```

Use of Reactive and encapsulation of attributes by ref to emit them. With the implementation of inject and Reactive, that works.

We have above some examples of elements that have change in the code, it’s not all the changes of course.

Godfrin
Louis
N° étudiant : 20210589

Changes:

Replacing props with injects:

The SigninButton component no longer uses props to pass information about the user by using isCut to disable the button and setUser.

Use of ref and reactive for reactive state management :

A reactive state object has been created with reactive() in HomePage to centralize shared states such as isCut and user.

Reactive deactivation of the Sign In button:
The Sign In button in SigninButton now uses state.isCut to determine whether or not it should be disabled. This enables reactive management of the button's state in multiple components from the state shared object. Avoid conflict with an other button that have to be disable.

Centralized user management:

User state is now managed globally in state.user in HomePage via the state reactive object. When a user logs in, their information is stored in state.user, which is shared with all child components.

The Sign In button in SigninButton component calls setUser injected after a successful login, which updates state.user in HomePage.

Synchronization of user displays in HomePage:

The HomePage component now uses state.user to display logged-in user information (for example, {{ state.user.username }}), ensuring that the interface reacts automatically to changes in user information.

To conclude:

These changes have for objective to eliminate the use of props, for implementing provide/inject, and making the application more reactive by using centralized state by the reactive object.

Godfrin
Louis
N° étudiant : 20210589

Exercise 4:

Command: npm install vuex@next --save

We use Vuex to manage the data of the user. So, we create a folder Store, and inside of it we put a index.js with inside mutations that will manage how the user's data are access and shared.

```
import { createStore } from 'vuex';

const store = createStore({
  state:
  {
    user: null,
    isCut: false
  },

  mutations:
  {
    setUser(state, user)
    {
      state.user = user;
      state.isCut = true;
    },

    clearUser(state)
    {
      state.user = null;
      state.isCut = false;
    },

    setIsCut(state, isCut)
    {
      state.isCut = isCut;
    },
  },

  getters:
  {
    isUserSignedIn: (state) => !!state.user,
    getUser: (state) => state.user,
    isButtonDisabled: (state) => state.isCut
  }
});

export default store;
```

So, we don't use inject anymore, but mapState and mapMutations inside the SiginButton to access the data of the user inside the SiginButton Component.

Godfrin
Louis
N° étudiant : 20210589
And mapState inside the HomePage component to display the username to show th

at the data are shared through all the component that are built to access the data.

```
computed:
{
  ...mapState(['user']) // Map the `user` state from Vuex
},
```

Exercise 5:

Here we just install vue router dependencies.

Command: npm install vue-router@4

Exercise 6:

Add a router folder, with a index.js inside of it:

```
// Router configuration

import { createRouter, createWebHistory } from 'vue-router';
import HomePage from '../pages/HomePage.vue';
import ConversationsIndexPage from '../pages/ConversationsPage.vue'; // Créez ce composant

const routes = [
  {
    path: '/',
    name: 'Home',
    component: HomePage,
  },
  {
    path: '/conversations',
    name: 'Conversations',
    component: ConversationsIndexPage, // Affiche une page de placeholder pour le moment
  },
];

const router = createRouter({
  history: createWebHistory(),
  routes,
});

export default router;
```

Pages folder :

-HomePage

-Conversations

Godfrin
Louis
N° étudiant : 20210589
Add a Conversations Page :

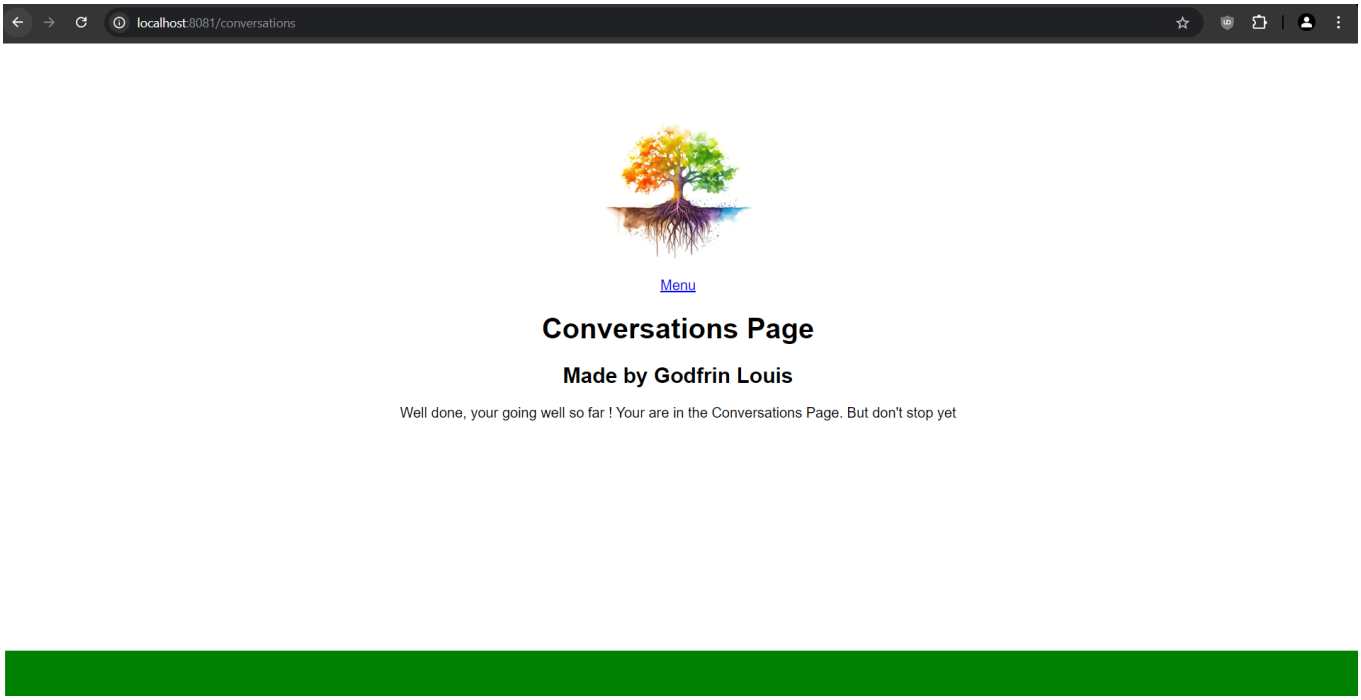
```vue
<!-- ConversationsPage.vue -->
<template>
    <div>
      <h1>Conversations Page</h1>
      <h2> Made by Godfrin Louis </h2>
      <p>Well done, your going well so far ! Your are in the Conversations Page. But don't stop yet :p </p>
    </div>
  </template>

  <script>
  export default
    {
        name: 'ConversationsIndexPage',
    };
  </script>
```

Modification of the Main.js :

```js
import { createApp } from 'vue'
import App from './App.vue'
import store from './store';
import router from './router';

const app = createApp(App);

// Use both Vuex store and Vue Router
app.use(store);   // Use the store with Vuex
app.use(router);  // Use the router

app.mount('#app');
```

Home Page: http://localhost:8081/

Godfrin
Louis
N° étudiant : 20210589
Conversations Page: http://localhost:8081/conversations

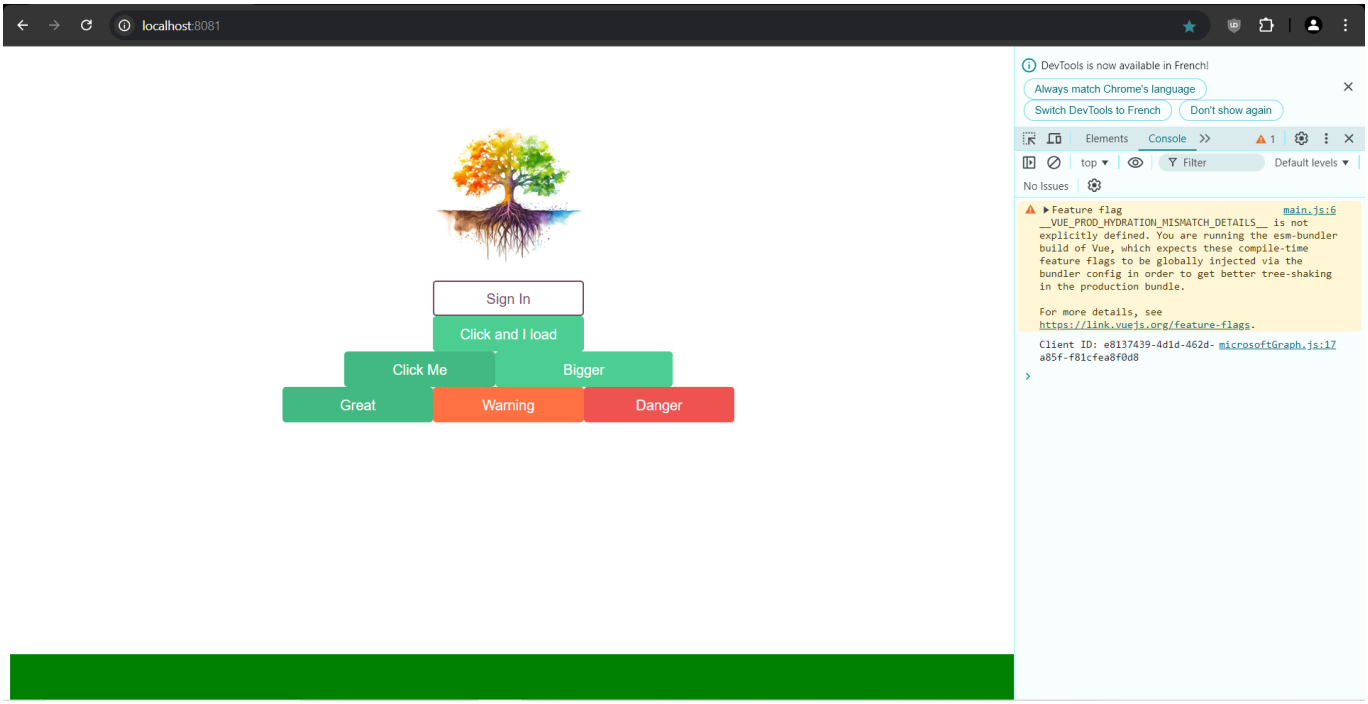


Exercise 7:

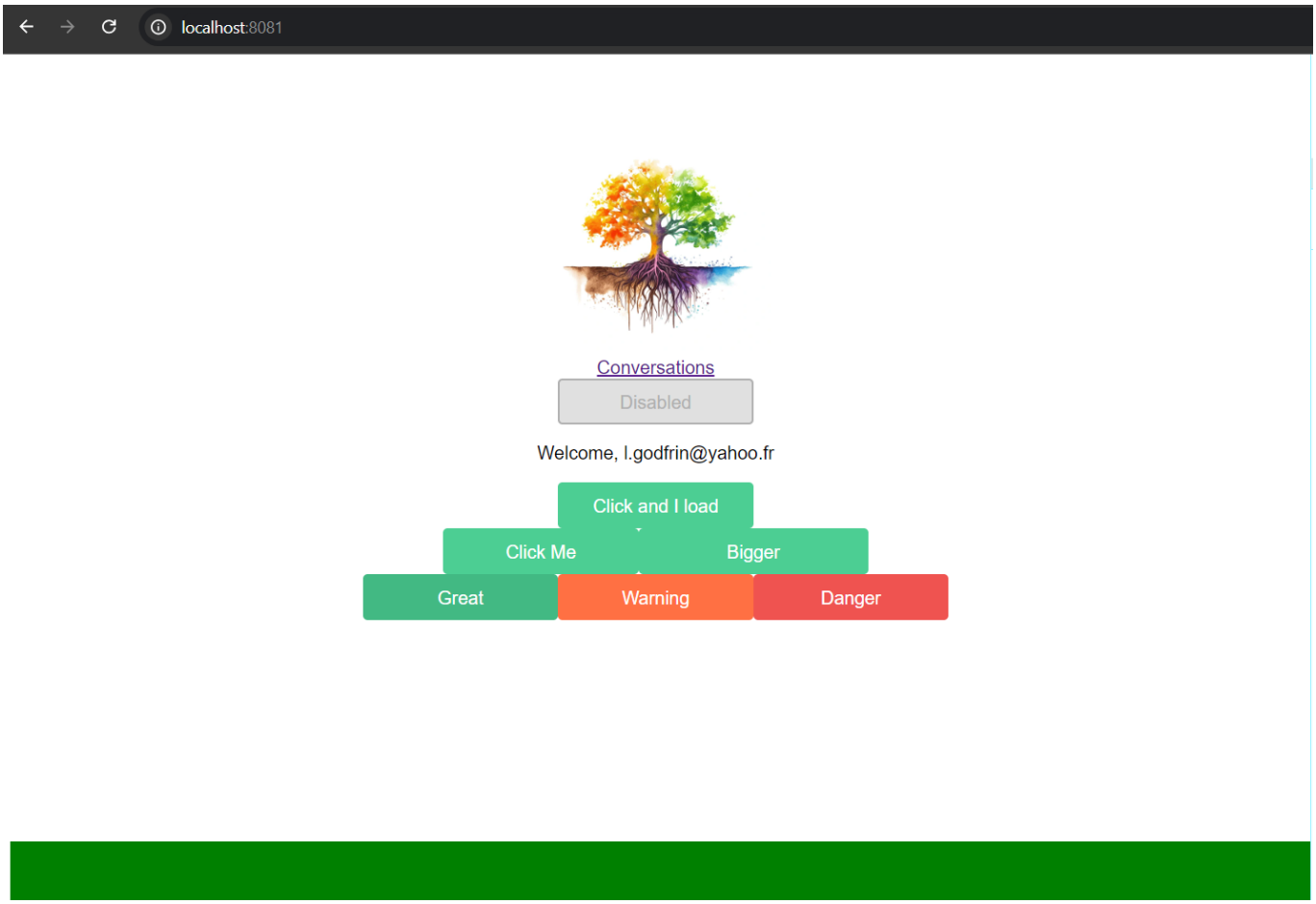
Add route inside baseHeader:

```
<template>
    <nav>
        <!-- Use of $route.path to have the good link in the nav to go to the good page  -->
        <router-link v-if="$route.path !== '/'" to="/">Go back to Home</router-link>

        <router-link v-if="user && $route.path === '/'" to="/conversations">Conversations</router-link>
    </nav>
</template>
```

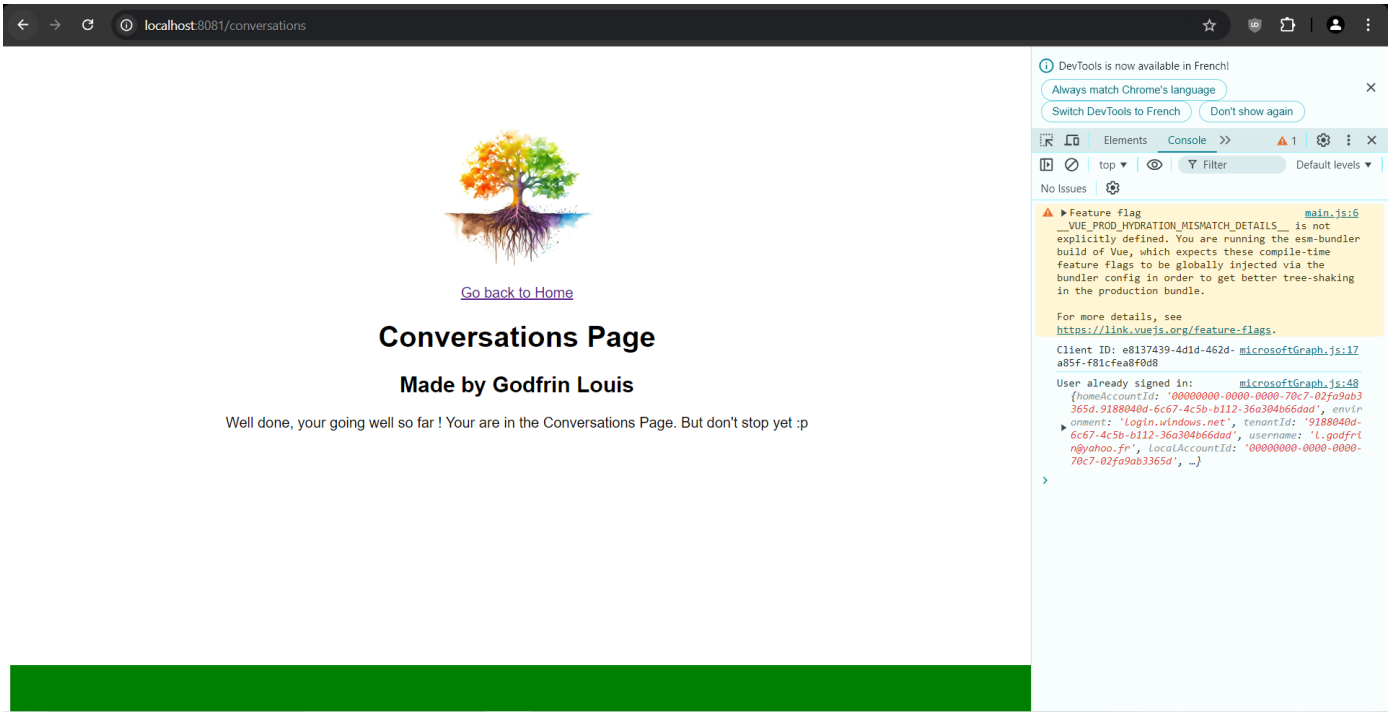
Godfrin
Louis
N° étudiant : 20210589
Vue of when the user is not sign in:



Vue from HomePage:



Godfrin
Louis
N° étudiant : 20210589
Vue from conversations page:



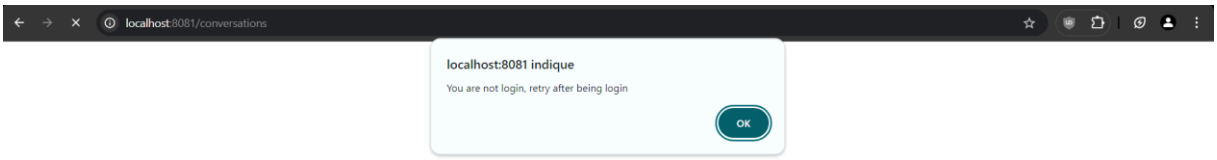
Exercise 8:

Add a guard:

```
// Guard, if the user is not loggin, he can't go to conversation page
router.beforeEach((to, from, next) => {

  // Verify if the route require a guard
  if (to.matched.some(record => record.meta.requiresAuth))
  {
      // If the requirement of the sigin is equal to false, so if the user is not login
      if (!store.state.user)
      {
        // Go to HomePage
        alert("You are not login, retry after being login")
        next({ path: '/' });
      }
      else
      {
        // else access to conversation page
        next();
      }
  }
  else
  {
    // if no guard, let access to any page without login
    next();
  }
});

export default router;
```

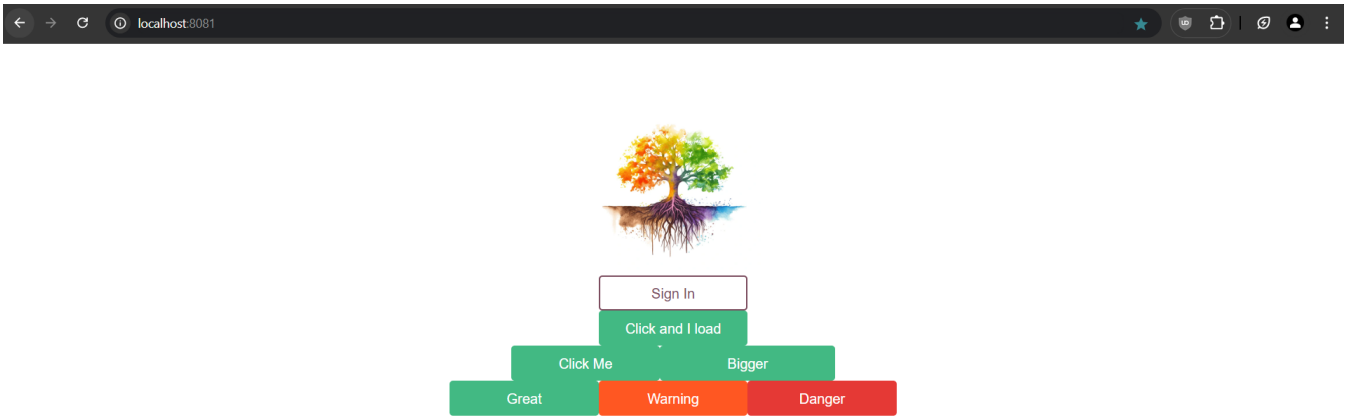
Godfrin
Louis
N° étudiant : 20210589
We can see that I have an alert to alert the user.

If I try to access to conversation by the URL, I can’t, the guard is up.

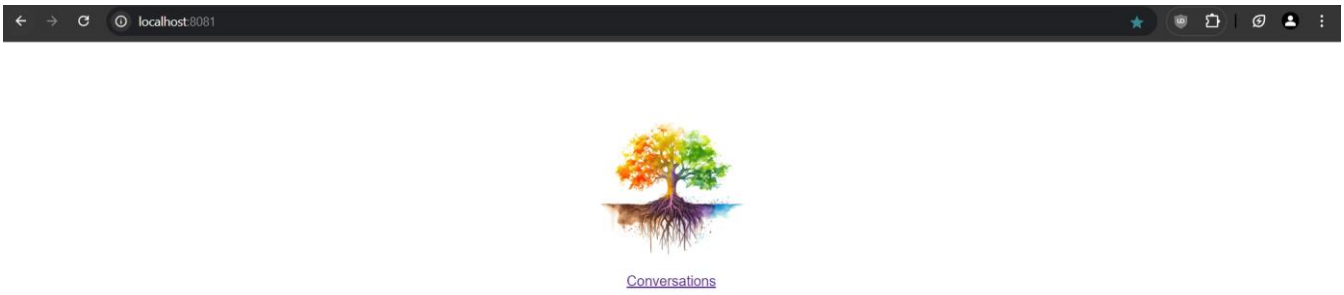


And I add something else, so we can’t access to the “<router-link>" of any pages if we are not login, with v-if.

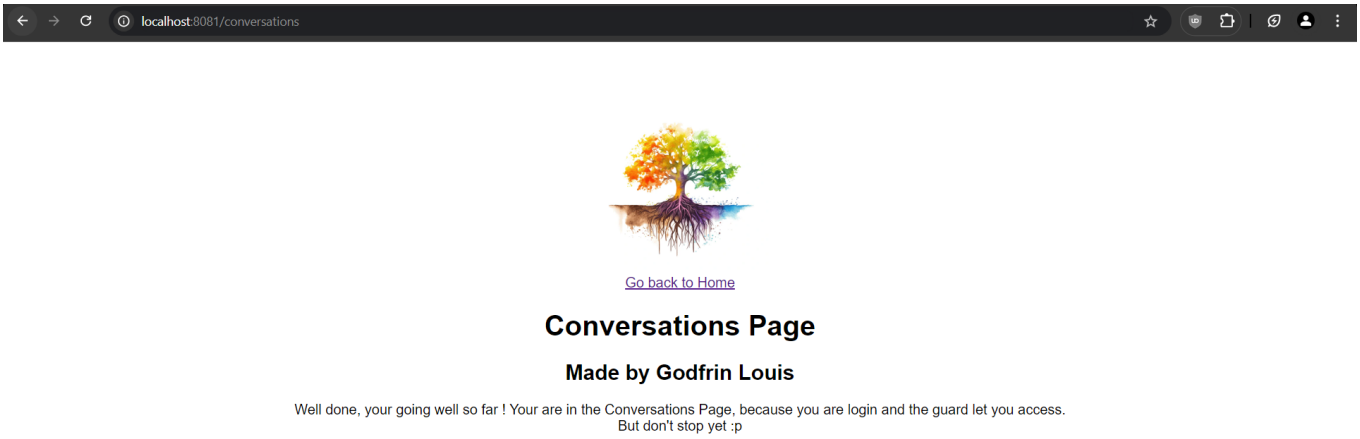
Without being connected:



Godfrin
Louis
N° étudiant : 20210589
And when we are connected:



And when we are connected, we can access to conversations page:

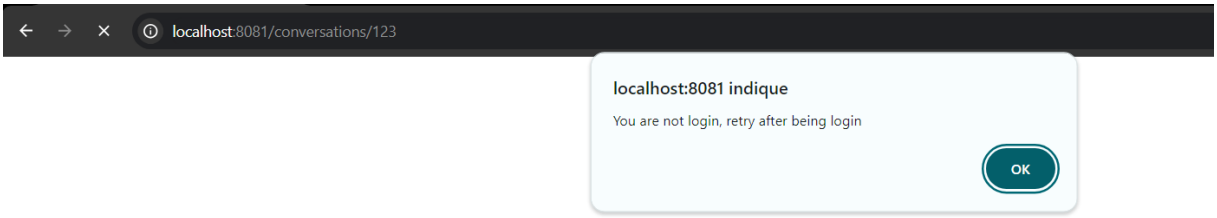


And I add that we can come back to Home Page from Conversation Page with router-link.

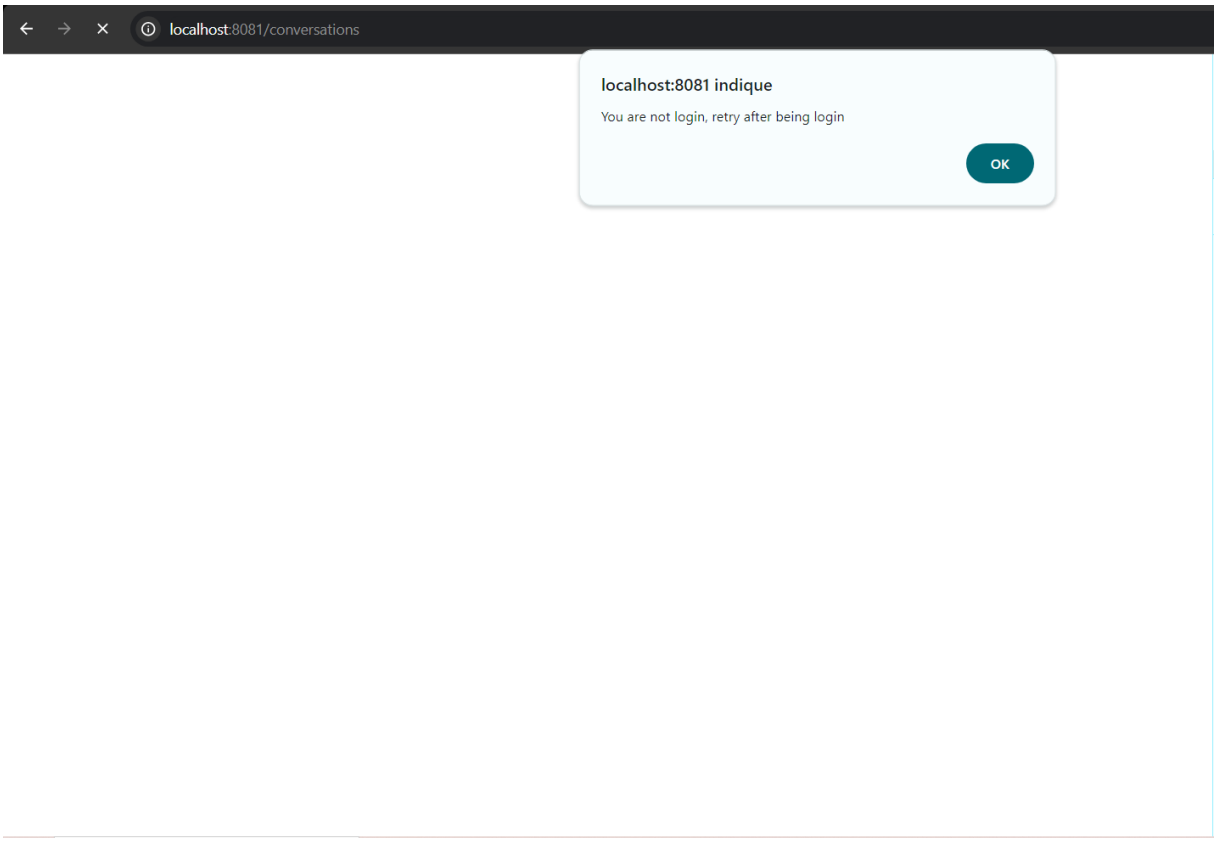
Godfrin
Louis
N° étudiant : 20210589

Exercise 9:

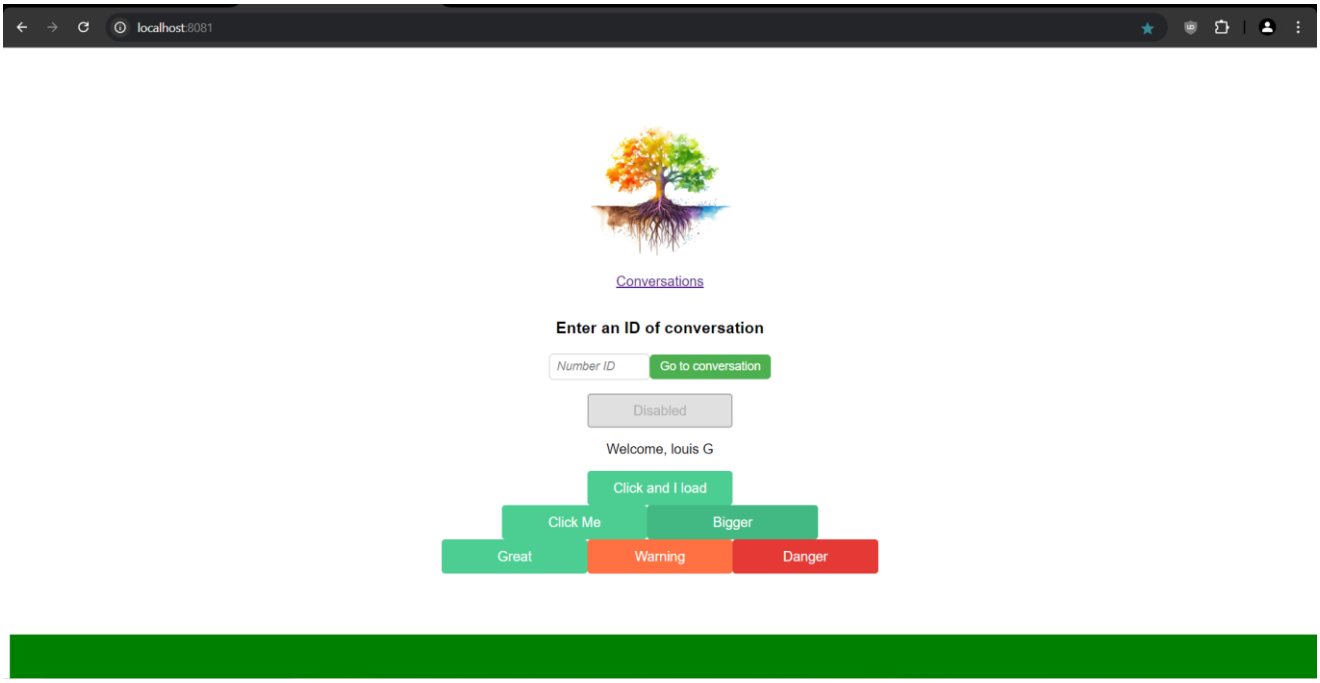
The guard is up for the conversation/:id



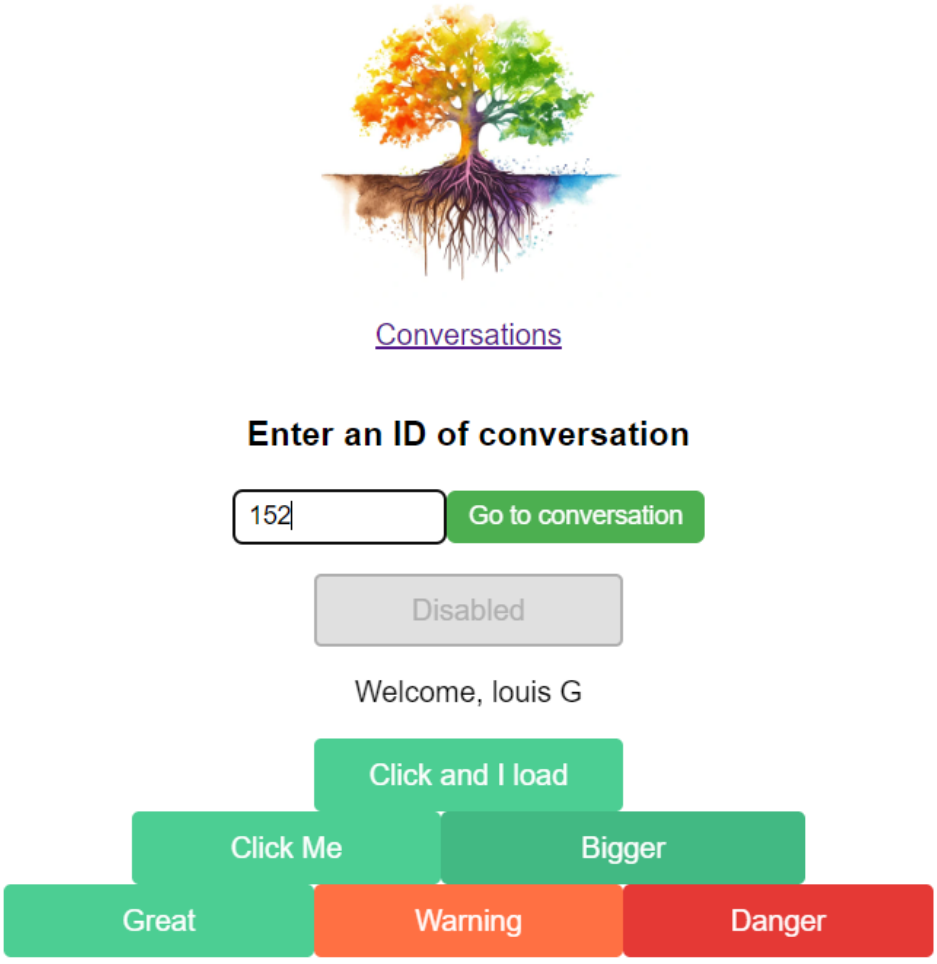
If the user is not login and try to access the conversation page, the guard and the alert are triggered and we will have that, for conversations and conversations/:id:



Godfrin
Louis
N° étudiant : 20210589
But when we are login:



We will be able to access to the conversation with the id that we give:



Godfrin
Louis
N° étudiant : 20210589
Then we access to conversations/:id (here 152):