RAPPORT WEB

Rapport ProgWeb - ClientSide

RESUME

Rapport Web pour la partie ClientSide du projet d'affichage des informations des stations d'essences

Quentin Larose

SI5 – AL – Polytech Nice Sophia

Table des matières

Équipe

L'équipe ayant réalisé ce projet est composé des étudiants suivants :

- Larose Quentin
- Martin D'Escrienne Yann
- Nersissian Tigran
- Tognetti Yohan

URL du projet

L'URL du projet est la suivante : https://github.com/SI5-QTYY-Web/si5-prog-web-project

Identifiant Github

Mon **identifiant Github** est le suivant : « ***QuentinLarose*** ».

Tâches effectuées

Durant ce projet j'ai réalisé différentes tâches que ce soit du côté serveur ou client, ici, nous parlerons uniquement des tâches effectuées sur le côté client du projet qui sont les suivantes :

- Mise en place du système d'authentification (backend et frontend)
- Mise en relation du backend et du frontend (appel au backend avec ***axios*** et traitement des données retournées)
- Réalisation des pages de connexion et de création de compte
- Mise en place de contexte permettant l'accès aux informations de compte (utilisateur, token, stations favorites → Contexte d'authentification)
- Traitement des cas où l'utilisateur est connecté (gestion de son session token, etc.)
- Récupération de la position de l'utilisateur et traitement de cette position (stockage de la position qui est utilisée pour centrer la carte et sert de point de référence autour duquel récupérer les stations dans un rayon défini)
- Mise en place du système de stations favorites (côté frontend)
- Mise en relation du backend et du frontend pour la communication des stations favorites (appels de route et traitement des données retournées)

Stratégie employée pour la gestion des versions avec Git

En ce qui concerne la stratégie que nous avons employé pour la gestion des versions avec ***Git***, nous avons décidé d'utiliser des features branchs en créant une feature branch pour chaque feature ou tâche. Le but est que chaque feature soit isolée du projet pour ne pas être perturbée par les changements de chacun et que le développement de la feature soit stable et n'affecte pas les autres développeurs. Une fois cette feature fonctionnelle, la branche est fusionnée dans la branche principale, qui était ***dev*** pour nous, afin de mettre à jour les changements et que les autres personnes y aient accès sur la branche dev. Une option aurait été de mettre en place un système de pull request lors de merge des features branch mais cela aurait été un peu trop complexe pour un projet d'une envergure pas si imposante et nous aurait rajouté plus de temps à traiter les pulls requests plutôt que se concentrer sur le développement à proprement parler.

De plus, nous avons mis en place des issues pour chaque tâche et feature à réaliser, qui correspondent plus ou moins à un besoin/une user story ainsi qu'un kanban afin d'avoir une organisation de travail agile et traiter dynamiquement toutes les tâches et avoir une visibilité plus nette de l'état du projet et de l'avancée du travail de chacun.

Solutions choisies

Initialement, j'avais prévu d'utiliser des composants de la librairie ***react-boostrap***, comme ce fut le cas pour les pages de connexion login et signup. Mais après une discussion d'équipe nous avons décidé de ne pas généraliser son utilisation pour ne pas dépendre de son utilisation et de ses styles par défaut qui se superposent sur nos thèmes de base et sont donc compliqués à traiter lorsque l'on ne veut pas appliquer le thème et le style de ***react-bootstrap***.

De plus nous avons décidé d'utiliser plusieurs contextes, dont un que j'ai réalisé : ***AuthContext***. Ce choix s'explique par le fait de généraliser certaines informations et les rendre plus facilement accessible étant donné que dans le cas des informations d'authentification et propres à l'utilisateur, beaucoup de composants requièrent d'avoir accès à ces informations, pour cela l'utilisation d'un contexte est largement justifié. En effet, le token est utilisé plusieurs fois à de nombreux endroits, l'user aussi, et les stations favorites sont également utilisées à travers le contexte et l'***AuthContext*** est encore une fois utilisé dans le composant de stations favorites. Passer ces informations à travers les ***props*** des composants aurait été redondant et il aurait fallu passer ces informations à de nombreuses reprises et cela aurait fait un grand nombre de props sur le long terme également.

Enfin, nous avons utilisé ***axios*** pour faire nos requêtes vers le backend, cela nous facilite l'envoi de données, sa récupération ainsi que les traitements d'erreurs liés à ces requêtes. Cela a été un choix naturel puisque nous sommes habitués à utiliser ***axios*** que ce soit avec ***react***, ***angular***, etc. De plus ***axios*** est bien documenté, facile de prise en main et très répandu.

Difficultés rencontrées

Une des difficultés rencontrées lors de ce projet a été le soucis d'Authorization et de CORS lors de requête sur le backend avec un token. En effet, sur certaines requêtes du backend, nous avons mis un Guard dans le but de pouvoir utiliser ces routes uniquement lorsque l'on est authentifié et que l'on a un token fonctionnel. Cela a posé quelques soucis lors de l'appel de ces routes depuis le frontend puisqu'on a dû spécifier les autorisations dans les ***headers*** des requêtes axios et cela demandait un format particulier que l'on avait défini et il fallait le faire correspondre à cette définition du backend. Pour résoudre ce souci on a mis en place un hook sur les requêtes axios pour spécifier les autorisations dans le headers en récupérant le token depuis le localStorage. Ensuite, le token était également initialement stocké dans un json et on récupérait le json, puis le token dans le json. On a donc changé cela pour récupérer directement le contenu du token, c'est-à-dire, une chaîne de caractères.

J'ai également eu quelques de soucis de validation lors de l'utilisation des formulaires de connexion et de création de compte puisqu'il fallait que le contenu d'authentification du frontend corresponde à ce qui est attendu dans le backend. Pour cela on a restreint l'utilisateur du côté des formulaires puis on a traité certaines erreurs à l'aide de try and catch pour éviter que l'erreur provoque une erreur sur la page et qu'au contraire, cette erreur soit interceptée et traitée pour afficher des messages à l'utilisateur le renseignant sur ce que son action a produit : par exemple, si les identifiants sont mauvais et ne correspondent à aucun compte, etc.

Temps de développement / tâche

Tâche	Temps de développement
Page /login	5%
Page /signup	5%
Validation sur les formulaires d'authentification	15%
Récupération de la position et utilisation de la position	5%
Traitement des erreurs dues à la communication avec le backend	15%
Mise en place des stations favorites côté frontend	20%
Mise en place du contexte d'authentification	10%
Traitement utilisateur authentifié frontend	10%
Debug général	15%

Code

Dans notre projet, il n'y a pas vraiment de composant que j'aie réalisé qui soit réellement beaucoup plus optimal qu'un autre ni un autre qui soit beaucoup plus améliorable, etc. Cependant, j'ai tout de même relevé un composant qui me semble bien écrit et fait correctement ce qu'il est censé faire sans pouvoir être plus améliorable, tout comme un composant qui lui pourrait être un peu amélioré avec plus de fonctionnalité derrière ce bout de code.

Composant optimal

Pour le composant que je trouve « optimal », j'ai retenu le suivant (cf. captures d'écran). Je le trouve optimal dans le sens où, c'est un composant pour se connecter sur notre webapp. Il fait donc ce qu'on attend de lui, il propose un formulaire avec de la validation pour s'assurer que l'utilisateur saisit bien ce qu'on attend de lui lors de la saisie du formulaire. De plus il propose une redirection si l'utilisateur veut retourner sur d'autres pages ce qui permet de ne pas restreindre l'utilisateur et qu'il se retrouve dans un cas où il est prisonnier de notre page. Enfin en ce qui concerne la demande de connexion à proprement parler, elle est réalisée avec axios, si elle échoue, on traite correctement l'erreur en affichant un pop-up à l'utilisateur en mentionnant ce qui s'est mal déroulé (que les identifiants sont incorrects). Dans le cas où tout se déroule bien, le session token et l'user sont récupérés et stockés dans l'AuthContext implémenté, qui regroupe toutes les informations liées à une session utilisateur.

```
function LoginPage() {
  const navigate = useNavigateNoUpdates();
  const [username, setUsername] = useState( initialState: "");
  const [password, setPassword] = useState( initialState: "");
  const { setUser,setLogged} = useContext(AuthContext);

  function validateForm() {
    return username.length > 0 && password.length > 0;
  }

  async function handleSubmit(event: { preventDefault: () => void; }) {
    event.preventDefault();
    await axios.post( url: `http://localhost:3333/api/auth/login`,  data: { username: username, password: password })
      .then(res => {
        const token = res.data.access_token;
        localStorage.setItem('token', token);
        setLogged(true)
        localStorage.setItem('user', username);
        setUser(username);

        navigate( to: "/")
      }).catch((err)=>{
        alert('Adresse mail ou mot de passe invalide');
        console.log('Login failed : ', err);
      })
  }
```

```
  return(
    <div>
      <Breadcrumb>
        <Breadcrumb.Item href="http://localhost:4200">Home</Breadcrumb.Item>
        <Breadcrumb.Item active>Login</Breadcrumb.Item>
      </Breadcrumb>
      <div className="Login">
        <Form onSubmit={handleSubmit}>
          <Form.Group className="mb-3" controlId="formBasicEmail">
            <Form.Label>Adresse mail</Form.Label>
            <Form.Control
              placeholder="username"
              value={username}
              onChange={(e :ChangeEvent<HTMLInputElement | HTMLTextAreaElement> ) => setUsername(e.target.value)}
            />
          </Form.Group>
          <Form.Group className="mb-3" controlId="formBasicPassword">
            <Form.Label>Mot de passe</Form.Label>
            <Form.Control
              type="password"
              placeholder="password"
              value={password}
              onChange={(e :ChangeEvent<HTMLInputElement | HTMLTextAreaElement> ) => setPassword(e.target.value)}
            />
          </Form.Group>
          <Form.Group className="mb-3" controlId="formBasicCheckbox">
            <Form.Check type="checkbox" label="Remember me" />
          </Form.Group>
          <button className="buttonStyle" type="submit" disabled={!validateForm()}>
            Se connecter
          </button>
          <button className="buttonStyle" onClick={(e :MouseEvent<HTMLButtonElement> ) => navigate( to: `signup`)}>
            Se créer un compte
          </button>
        </Form>
      </div>
    </div>
  );
}
```

Composant améliorable

Enfin, pour parler d'un composant améliorable, on peut prendre l'exemple la page d'inscription à notre webapp. Ce composant peut être améliorable pour avoir plus de fonctionnalités déclenchées une fois un compte crée pour avoir du traitement sur celui-ci. On peut observer que notre création de compte se résume à une validation du formulaire sur les identifiants qui seront créés, une création de compte en cas de succès, et… c'est à-peu-près tout. Nous avions pensé à connecter l'utilisateur dès le compte créé mais on a orienté notre réflexion sur le fait que l'utilisateur, même s'il crée un compte, ne veut pas forcément se connecter ni se connecter directement avec ce compte s'il possède plusieurs comptes. Surtout que cela permet de confirmer que l'utilisateur a bien saisi les identifiants qu'il souhaitait lors de sa création de compte.

```
11  function SignupPage() {
12    const [email, setEmail] = useState( initialState: "");
13    const [password, setPassword] = useState( initialState: "");
14    const navigate = useNavigate();
15
16    function validateForm() {
17      return email.length > 0 && (password.length > 3 && password.length < 21);
18    }
19
20    function handleSubmit(event: { preventDefault: () => void; }) {
21      event.preventDefault();
22      const signupUrl = `http://localhost:3333/api/auth/signup`;
23      try{
24        axios.post(signupUrl,  data: { username: email, password: password })
25          .then(res => {
26            alert('Votre compte a bien été créé');
27            navigate( to: "/login")
28          });
29      } catch (e) {
30        console.log(e);
31      }
32    }
```