Exam-React-Typescript

Examen Pratique React Typescript : Application de Blog

Objectif

Créer une application de blog en React Typescript qui interagit avec l'API JSONPlaceholder pour afficher des posts et leurs détails, y compris les informations sur l'auteur du post.

Consignes

Partie 1 : Configuration de l'Environnement (2 points)

1. Initialiser un nouveau projet React avec

```
npx create-react-app exam-react --template typescript
```

2. Installer `react-router-dom`

```
npm i react-router-dom
```

3. Lancez l'application et vérifiez que React démarre correctement
4. Supprimez les fichiers
   - `App.css`
   - `App.test.tsx`
5. Créez les 2 fichiers css suivants dans le dossier `src` au même niveau que le fichier `App.tsx`

```
/* reset.css */

body {
  background-color: #f6f6f6;
  font-family: "Roboto", sans-serif;
  font-size: 16px;
  line-height: 1.5;
  color: #333;
  margin: 0;
  padding: 0 20px;
  box-sizing: border-box;
}
```

```
/* layout.css */

#root {
  display: flex;
  flex-direction: column;
  min-height: 100vh;
}

#root > header {
  height: 80px;
}

#root > main {
  flex: 1;
}

#root > footer {
  height: 80px;
}
```

6. Editez le fichier `App.tsx` comme suit :

```
import "./reset.css";
import "./layout.css";
import Router from "./Router";

function App() {
  return <Router />;
}

export default App;
```

Partie 2 : Mise en Place du Routing (3 points)

Configurer le routage dans `App.tsx` à l'aide de `react-router-dom` et créer un fichier `Router.tsx` qui définit les chemins suivants :

- `"/"` doit retourner un composant `PageTemplate` et ce chemin racine a les chemins enfants suivants

  Voici le HTML partiel du composant PageTemplate `src/components/PageTemplate/index.ts`

  ```
  <>
    <header>
      <h1>Exam Typescript React</h1>
      {/* ...une barre de navigation horizontale avec un lien vers la page home */}
    </header>
    <main>
      {/* ...le composant permettant le rendu des sous-routes par le router */}
    </main>
    <footer>
      <p>
        {/* Par [Prénom NOM] (en italique sans importance sémantique) le 10/11/2023 (attention à utiliser correctement la bal
      </p>
    </footer>
  </>
  ```

- `""` qui retourne l'élément `HomePage`
- `"posts/:postId"` qui retourne l'élément `PostPage` et redirige par défaut sur `"posts/:postId/detail"` au moyen d'une sous-route `""`(veillez à utiliser la props appropriée pour remplacer l'url en cas de redirection)
- `"detail"` est une sous-route de `"posts/:postId"` qui retourne l'élément `PostDetailPage`
- `"owner/:userId"` est une sous-route de `"posts/:postId"` qui retourne l'élément `PostOwnerPage`
- `"*"` doit retourner `Page404` dont voici le HTML partiel

```
<>
  <p>404 Page</p>
  <p>
    {/* Un lien permettant de retourner à la home page */}
  </p>
</>
```

Partie 3 : Liste des Posts (3 points)

1. Dans `HomePage`, implémenter la logique pour récupérer la liste des posts depuis `https://jsonplaceholder.typicode.com/posts`
2. Les afficher dans une liste HTML de lien vers leur page de détail: `posts/${post.id}/detail`
3. Refactorisez l'appel réseau pour utiliser une fonction `fetchPosts` exportée d'un fichier `src/services/api/jsonplaceholder/posts.ts`. le dossier `jsonplaceholder` contiendra également un fichier `index.ts` exportant des constantes propres à l'api, dans votre cas il. sera ainsi :

   ```
   export const rootUrl = "https://jsonplaceholder.typicode.com";
   ```

   Vous pourrez ainsi utiliser une constante `postsUrl` ayant pour valeur `` `${rootUrl}/posts` `` dans la fonction `fetchPosts` et d'autres dans le futur.

Partie 4 : Contexte du Post (4 points)

Créez un contexte `PostContext` pour gérer les données du post actuel et ainsi éviter de refaire des appels inutiles à l'api

1. Créez un fichier `src/models/Post.ts` pour typer le retour de l'API

```
type Post = {
  id: number;
  userId: number;
  title: string;
  body: string;
};

export default Post;
```

2. Créez un fichier `src/contexts/PostContextProvider.tsx` . Le context doit exposer uniquement le post courant et rien d'autre car le post ne sera jamais défini depuis l'extérieur, voici son type

```
type PostContextType = {
  post: Post | null;
};
```

3. Nous voulons valider l'ID du post, et rediriger l'utilisateur vers la page 404 si l'ID est invalide, `not-found` étant une route inconnue du router elle sera prise en charge par la route `"*"` qui renvoie une 404, attention à bien remplacer l'url présentant l'ID invalide par l'url `not-found` sans ça l'utilisateur sera toujours redirigé vers la page 404 en utilisant le bouton de retour de son navigateur (pensez à la propriété `replace` qu'accepte le hook de navigation de `react-router-dom`. L'ID doit être un nombre entier strictement positif (pensez à `Number.isInteger`)
4. Ajoutez une fonction `fetchPost` dans le fichier `src/services/api/jsonplaceholder/posts.ts` responsable de faire les appels réseaux pour ce qui concerne les posts
5. Vous redirigerez sur la page 404 si l'appel API échoue
6. C'est seulement si tout se passe bien que vous pourrez définir l'état du post du context avec la valeur retournée par l'API

Partie 5 : Page du Post (2 points)

Créez le fichier `src/pages/PostPage/index.tsx` pour la page `PostPage`, consommez-y le `PostContext` pour afficher :

- Un lien vers les détails du post, la sous-route `"detail"` de la route `"posts/:postId"`
- Un lien vers les détails de l'auteur du post, la sous-route `"owner/:userId"` de la route `"posts/:postId"`

Partie 6 : Détail du Post (1 points)

Créez `PostDetailPage` dans le dossier `PostPage` , cette sous page affichera le contenu du post en consommant le contexte `PostContext` pour le passer en `props` d'un composant `PostCard` dont voici le HTML

```
<div>
  <h2>{post.title}</h2>
  <p>{post.body}</p>
</div>
```

Partie 7 : Détails de l'auteur du Post (4 points)

1. Créez `PostOwnerPage` qui consommera également le `PostContext` pour retrouver l'ID de l'auteur du post
2. Utilisez cet ID pour récupérer ses détails depuis `https://jsonplaceholder.typicode.com/users/:userId`.
   - Créez un fichier `src/models/User.ts` pour typer le retour de l'API

     ```
     type User = {
       id: number;
       name: string;
       username: string;
       email: string;
     };

     export default User;
     ```

   - Créez un fichier `src/services/api/jsonplaceholder/users.ts` sur le même principe que `posts.ts` utilisant la constante `rootUrl` dans une constante `usersUrl` qui sera elle-même utilisée dans la fonction exportée `fetchUser` que vous

utiliserez.

- Redirigez vers la page 404 pour toute erreur lors de l'appels API.
- Comme pour le post, redirigez sur la page 404 si l'ID de l'utilisateur n'est pas valide, l'ID doit être un nombre entier strictement positif (pensez à `Number.isInteger`)

3. Affichez les détails de l'auteur du post au moyen d'un composant `UserCard` qui prendra les props nécessaires à utiliser ce HTML

```
<div>
  <h2>{user.name}</h2>
  <p>{user.username}</p>
  <p>{user.email}</p>
</div>
```

Voici un exemple d'implémentation fonctionnelle

https://exam-react.rael-calitro.ovh/

Critères d'Évaluation

Les étapes sont notées sur 19, et le dernier point évaluera la qualité :

- Nommage clair et en anglais
- Structure du projet (pages, components, services, utils, contexts, etc.)
- Respectez les conventions (
  - classes, interfaces, types, composants React… en PascalCase
  - variables, propriétés… en camelCase
  - urls, classes css… en kebab-case
- Utilisation privilégiée de `const`
- Indentation et lisibilité
- Interdiction d'utiliser le typage dynamique avec `any`
- Evitez les duplications (notamment pour la validation de l'ID)

Livrables

- Le projet doit être compressé et remis sur Moodle dans la section React, **en ayant supprimé le dossier `node_modules` au préalable.**