POLYTECH DIJON

TRAVAUX DIRIGÉS

Conception Web Avancée

React & Router

OBJECTIFS

- Assembler des composants existants pour créer des vues
- Mettre en place un routeur pour naviguer entre les vues

PREPARATION DU PROJET

- Clonez le dépôt GitHub https://github.com/cmeunier-ub/react-router-part-1
- Ouvrez le dossier du projet dans Visual Studio Code
- Installez les dépendances nécessaires à l'aide de la commande :

```
npm i
```

- Démarrez le serveur à l'aide de la commande :

```
npm run dev
```

- Vérifiez que le site se charge bien dans un navigateur (page grise)

LOGINVIEW

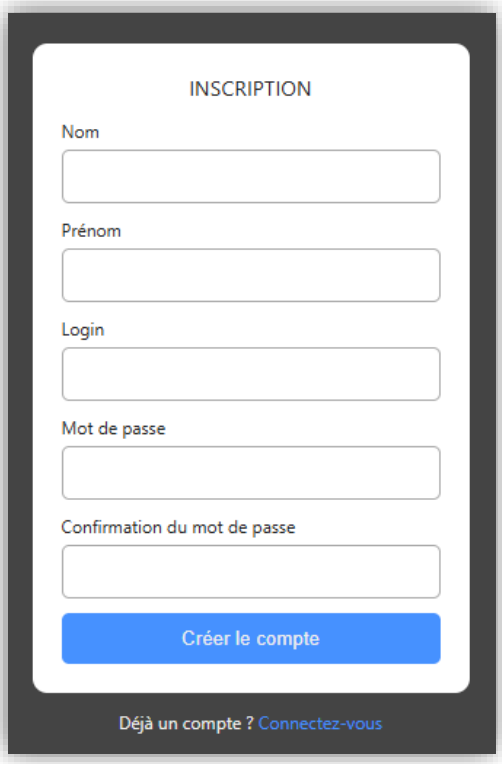
- Dans le dossier **`src/react/view`**, créez un dossier **`login-view`**
- Dans ce dossier créez les fichiers **`tsx`** et **`css`** du composant **`LoginView`**
- Ce dernier utilisera le composant **`LoginForm`** (voir annexe 1) pour fournir le visuel suivant :

> **Note**
> Pour le moment, le lien **`Créez un compte`** n'aura aucune action.

- Testez le bon fonctionnement de votre vue en l'utilisant dans le composant **`Application`** servant de base à l'application (**`/src/react/application`**)

REGISTERVIEW

- Créez une nouvelle vue **`RegisterView`** qui utilisera le composant **`RegisterForm`** (voir annexe 2) et aura l'aspect suivant :



> **Note**
> Ici aussi, pour le moment, le lien **Connectez-vous** n'aura aucune action.

- Dans le composant **Application**, remplacez le composant **LoginView** par **RegisterView** afin de tester le bon fonctionnement de votre vue.

PREMIERES ROUTES

- Installez la dépendance **React Router** à l'aide de la commande :

```
npm i react-router
```

React Router permet d'afficher des composants en fonction de l'URL de la page affichée. La mise en place d'un routeur est très simple comme le montre l'exemple suivant :

```
<BrowserRouter>
    <Routes>
        <Route path="/" element={<ComposantPageRacine />} />
        <Route path="/posts" element={<ComposantPageDesPosts />} />
    </Routes>
</BrowserRouter>
```

Ici, le routeur gère deux routes distinctes. Si l'URL du navigateur pointe sur la racine du site (**/**), le composant **ComposantPageRacine** sera affiché. En revanche, si l'URL du navigateur pointe sur la page **/posts**, c'est le composant **ComposantPageDesPosts** qui sera visible.

- A partir des éléments précédents, modifiez le composant **Application** pour que le site affiche **LoginView** si l'URL est **/login** et **RegisterView** si l'URL est **/register**.
- Testez le bon fonctionnement du routeur en modifiant l'URL de votre navigateur (ex : http://localhost:5173/login)

NAVIGATE

React Router propose un nouveau hook **useNavigate** qui permet de modifier l'URL de la page actuellement visible et de chargé le composant requis sans effectuer un rechargement complet de la page comme cela serait le cas si nous changions manuellement l'URL ou si nous utilisions une balise **<a>**

- Commencez par modifiez le composant **LoginView**.

- Stockez le résultat du hook **`useNavigate`** dans une constante déclarée dans votre composant.

```
const navigate = useNavigate();
```

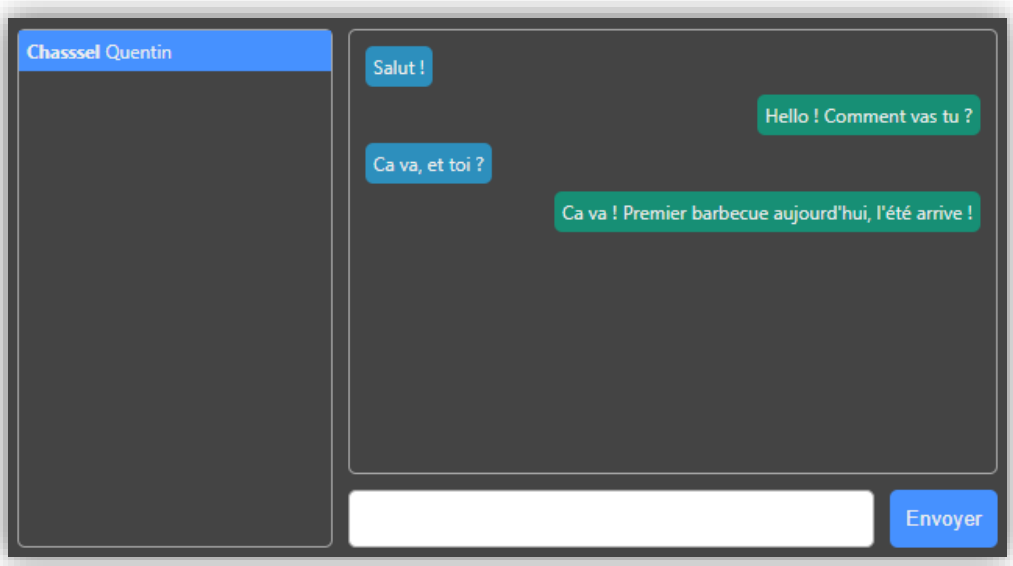
- Ajoutez une méthode **`goToRegister`** à votre composant. Cette méthode déclenchera une navigation vers l'URL **`/register`** de votre site.

```
const goToPage = () => {
    navigate("/page") ;
}
```

- Affectez la méthode **`goToRegister`** à l'attribut **`onClick`** du lien **`Créer un compte`**.
- Affichez la page de login et testez le bon fonctionnement du lien.
- Procédez de même pour la vue **`Register`** qui doit renvoyer vers **`/login`** lorsque l'on clique sur le lien **`Connectez-vous`**.

MESSENGERVIEW

- Créez une nouvelle vue **`MessengerView`** qui utilisera les composants **`ContactList`** (annexe 3) et **`Messenger`** (annexe 4) pour obtenir le rendu suivant :



- Modifiez le composant **`Application`** pour le composant **`MessengerView`** soit affiché lorsque l'on demande la racine du site (**`/`**).

SECURITE

En prévision de la seconde partie du sujet qui concernera la mise en place d'un backend avec AdonisJS, un service gérant l'authentification a été intégré au projet dans le dossier **`/src/services`**.

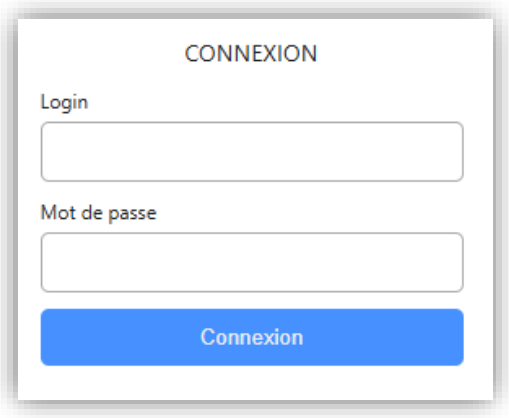
L'annexe 5 décrit en détails les méthodes proposées par ce service.

- Modifiez **`MessengerView`** pour que l'utilisateur soit automatiquement redirigé vers **`/login`** s'il tente d'envoyer des messages sans s'être connecté au préalable.

- Modifiez **`LoginView`** pour rediriger l'utilisateur vers la racine du site (**`/`**) si la connexion réussit.

ANNEXES

Annexe 1 – LoginForm



Composant permettant la saisie d'un login et d'un mot de passe. Le bouton **`Connexion`** déclenche la méthode **`onValidate`** passée comme paramètre du composant.

Paramètres du composant

- **`onValidate : (data : LoginData) => void`**
  Action à réaliser lorsque l'on clique sur le bouton Connexion

Annexe 2 – RegisterForm

Composant permettant la saisie des informations de l'utilisateur dans le but de créer un nouveau compte. Le bouton **`Créer le compte`** teste si les champs **`Mot de passe`** et **`Confirmation du mot de passe`** sont identiques puis appelle la méthode **`onValidate`** passée en paramètre du composant.

Paramètres du composant

- **`onValidate : (data : RegisterData) => void`**
  Action à réaliser lorsque l'on clique sur le bouton Créer le compte (une vérification des mots de passe sera réalisée avant d'appeler onValidate)

Annexe 3 – ContactList

Affiche une liste de contacts passée en paramètre.

Paramètres du composant

- **`contacts : ContactData[ ]`**
  Liste des contacts à afficher

- **`selection : ContactData | null`**
  Contact sélectionné, **`null`** si aucun contact sélectionné

- **`onClick : (clickedContact : ContactData) => void`**
  Action à réaliser lors d'un clic sur un contact de la liste

Annexe 4 – Messenger

Composant gérant les messages d'une conversation entre deux contacts.

Paramètres du composant

- **`from : ContactData`**
  Contact à l'origine des messages depuis cette instance de l'application (l'utilisateur actuel)
- **`to : ContactData`**
  Contact avec qui l'utilisateur est en train d'échanger

Annexe 5 – AuthService

Service gérant l'authentification de l'utilisateur sur le serveur. Actuellement, le service contient le code minimum pour effectuer des tests sans nécessité de lien avec le backend qui n'existe actuellement pas. Ce sera l'objectif du prochain sujet.

- **`register(data : RegisterData) : Promise<boolean>`**
  Fonction asynchrone demandant au serveur de créer un nouveau compte à partir des données fournies dans **`data`**.

  Retourne **`true`** si le compte a bien été créé, **`false`** sinon.
- **`connect(data : LoginData) : Promise<boolean>`**
  Fonction asynchrone demandant au serveur d'authentifier l'utilisateur. Si les identifiants fournis dans **`data`** sont valides, le serveur retourne un token de sécurité.

  Retourne **`true`** si la connexion a réussi, **`false`** sinon.
- **`isConnected() : boolean`**
  Indique si l'utilisateur est bien connecté.

  Retourne **`true`** si l'utilisateur est connecté, **`false`** sinon.
- **`getUser() : UserData | null`**
  Fournit les informations sur l'utilisateur connecté à partir des données contenues dans le token de sécurité.

  Retourne les données de l'utilisateur si ce dernier est connecté, **`null`** sinon.