

ARGENTBANK

Utilisez une API pour un compte utilisateur bancaire avec React

Utilisez une API pour un compte utilisateur bancaire avec React

Soutenance du vendredi 3 mars 2023

Table des matières :

Objectif du projet

et Spécifications fonctionnelles et Techniques

Partie 1 - Création du prototype :

Vue d'ensemble du prototype réalisé avec démonstration

Mise en œuvre des fonctionnalités

Partie 2 - Création du fichier au format YAML :

Mise en œuvre de la document décrivant les API proposées pour les transactions

Conclusion

Objectif du projet et Spécifications fonctionnelles et Techniques :

L'entreprise Argent Bank est une nouvelle banque sur le marché. L'objectif est de l'aider à mettre en place son application. Le projet contient deux phases.

Phase 1 :

Authentification des utilisateurs - Création d'une application web permettant aux clients de se connecter et de gérer leurs comptes et leur profil.

Phase 2 :

Transactions - Il s'agirait de spécifier les endpoints d'API nécessaires pour une éventuelle deuxième mission une fois que nous aurons terminé la première.

PHASE 1 : APPLICATION WEB COMPLÈTE ET RESPONSIVE AVEC REACT

- L'UTILISATEUR PEUT VISITER LA PAGE D'ACCUEIL
- L'UTILISATEUR PEUT SE CONNECTER AU SYSTÈME
- L'UTILISATEUR PEUT SE DÉCONNECTER DU SYSTÈME
- L'UTILISATEUR NE PEUT VOIR LES INFORMATIONS RELATIVES À SON PROPRE PROFIL QU'APRÈS S'ÊTRE CONNECTÉ AVEC SUCCÈS
- L'UTILISATEUR PEUT MODIFIER LE PROFIL ET CONSERVER LES DONNÉES DANS LA BASE DE DONNÉES.

POUR LE PROJET : UTILISATION DE REDUX (ET DE REDUX TOOLKIT), BIBLIOTHÈQUE DE GESTION D'ÉTAT POUR LES APPLICATIONS REACT.

=> STOCKAGE DES DONNÉES D'APPLICATION DANS UN MAGASIN DE DONNÉES CENTRALISÉ ET LEUR ACCESSIBILITÉ À TOUS LES COMPOSANTS DE L'APPLICATION.

Création du store, des actions et du reducer

```
import { configureStore } from '@reduxjs/toolkit';
import userReducer from './user.slice';

const store = configureStore({
  reducer: {
    user: userReducer,
  },
});

export default store;
```

```
import { createSlice } from '@reduxjs/toolkit';

const initialState = {
  token: null,
  userFirstName: '',
  userLastName: '',
  isAuthenticated: false,
  errorLogin: '',
  errorUpdateIdentity: '',
};
```

Etat initial de l'application

```
const userSlice = createSlice({
  name: 'user',
  initialState,
  reducers: {
    setToken: (state, action) => {
      state.token = action.payload;
    },
    setUserFirstName: (state, action) => {
      state.userFirstName = action.payload;
    },
    setUserLastName: (state, action) => {
      state.userLastName = action.payload;
    },
    setIsAuthenticated: (state, action) => {
      state.isAuthenticated = action.payload;
    },
    logout: (state) => {
      state.token = null;
      state.userFirstName = '';
      state.userLastName = '';
      state.isAuthenticated = false;
    },
    setErrorLogin: (state, action) => {
      state.errorLogin = action.payload;
    },
    setIsLoadingLogin: (state, action) => {
      state.isLoadingLogin = action.payload;
    },
    setErrorUpdateIdentity: (state, action) => {
      state.errorUpdateIdentity = action.payload;
    },
  },
});
```

Conservation du token

Suppression des données utilisateur lors de la déconnexion

Connexion au système

La gestion de l'authentification de l'utilisateur

Gestion de l'email et du password : Le composant <FormLogin />

```
export default function FormLogin() {  
  const dispatch = useDispatch();  
  const navigate = useNavigate();  
  const token = useSelector((state) => state.user.token);  
  const isAuthenticated = useSelector((state) => state.user.isAuthenticated);  
  const errorLogin = useSelector((state) => state.user.errorLogin);  
  const [email, setEmail] = useState('');  
  const [password, setPassword] = useState('');  
  const [isLoading, setIsLoading] = useState(false);
```

```
  useLocalStorageToken(dispatch, token, setToken, setIsAuthenticated)
```

```
  const {rememberMe, handleRememberMe} = useLocalStorageLogin(email, setEmail, password, setPassword, isAuthenticated);
```

```
  const handleFormLogin = (event) => {  
    event.preventDefault()  
    login(setIsLoading, email, password, dispatch)  
  }
```

```
  useEffect(() => {  
    dispatch(setErrorLogin(''));  
  }, [email, password, dispatch]);
```

```
  useEffect(() => {  
    if (isAuthenticated) {  
      navigate('/profile')  
    }  
  }, [isAuthenticated, navigate]);
```

Gestion du token dans le localStorage en cas de rafraîchissement alors que l'utilisateur est connecté

Appel à l'api pour l'authentification
=> seul lors du rafraîchissement
durant une connexion authentifiée,
la connexion est automatique,
sinon appel api.

Gestion de
l'enregistrement et de la
récupération des données
de connexion de
l'utilisateur dans le
localStorage

Navigation vers la page de profile lorsque la connexion a réussi
(isAuthenticated en true)

Mise en place de la gestion de l'email et du password dans le localStorage

```
import { useEffect, useState, useCallback } from 'react';
import { AES, enc } from 'crypto-js';

const KeyLogin = 'g5yF01236Dxilp';
const KeyToken = "groqIU25xdd";

export function useLocalStorageLogin(email, setEmail, password, setPassword, isAuthenticated){
  const [rememberMe, setRememberMe] = useState(false);

  const loadDataFromLocalStorage = useCallback(() => {
    const encryptedLoginUser = localStorage.getItem("loginUser");
    if (encryptedLoginUser) {
      const decryptedLoginUser = JSON.parse(AES.decrypt(encryptedLoginUser, KeyLogin).toString(enc.Utf8));
      setEmail(decryptedLoginUser.email);
      setPassword(decryptedLoginUser.password);
      setRememberMe(true);
    }
  }, [setEmail, setPassword, setRememberMe]);

  const saveLoginToLocalStorage = (email, password) => {
    const loginUser = { email, password };
    localStorage.setItem(
      "loginUser",
      AES.encrypt(JSON.stringify(loginUser), KeyLogin).toString()
    );
  };
}
```

Fonction de récupération de l'email et du password dans le localStorage s'ils y sont présents

Fonction d'enregistrement de l'email et du password dans le localStorage

Mise en place de la gestion de l'email et du password dans le localStorage (suite)

```
useEffect(() => {  
  loadDataFromLocalStorage();  
}, [ loadDataFromLocalStorage]);
```

Lancement de la récupération de l'email et du password lors du chargement du composant.

```
useEffect(() => {  
  if (isAuthenticated && rememberMe) {  
    saveLoginToLocalStorage(email, password);  
  }  
}, [isAuthenticated,rememberMe, email, password]);
```

Lancement de l'enregistrement de l'email et du password seulement si l'authentification a réussi (=> isAuthenticated) et si la case rememberMe est cochée.

```
const handleRememberMe = (event) => {  
  setRememberMe(event.target.checked);  
  if(!event.target.checked) {  
    localStorage.removeItem("loginUser");  
  }  
};
```

Fonction de suppression des données de login (email et password) si la case rememberMe est décochée.

```
return {  
  rememberMe,  
  handleRememberMe,  
}  
};
```

Gestion de la sauvegarde du token dans le localStorage en cas de rafraîchissement de la page pour permettre une reconnexion

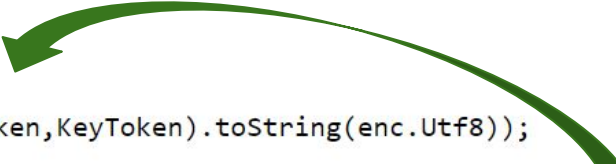
```
export function useLocalStorageToken(dispatch, token, setToken, setIsAuthenticated) {
```

```
  useEffect(() => {  
    const encryptedToken = localStorage.getItem("token");  
    if (encryptedToken) {  
      const decryptedToken = JSON.parse(AES.decrypt(encryptedToken, KeyToken).toString(enc.Utf8));  
      dispatch(setToken(decryptedToken));  
      dispatch(setIsAuthenticated(true));  
    }  
  }, [dispatch, setToken, setIsAuthenticated ]);
```


```
  useEffect(() => {  
    if (token) {  
      localStorage.setItem(  
        "token",  
        AES.encrypt(JSON.stringify(token), KeyToken).toString()  
      );  
    }  
  }, [token]);
```

```
  return
```

```
}
```



Récupération du token dans le localStorage s'il y est présent et son décryptage.
Enregistrement de celui-ci dans le store,
Passage de isAuthenticated en true pour la redirection vers la page profile



Enregistrement du token dans le localStorage dès sa réception et son enregistrement dans le store après cryptage de celui-ci,

L'appel à l'api lors de l'authentification

```
import { setToken, setIsAuthenticated, setErrorLogin } from '../store/user.slice';
```

```
export const login = async (setIsLoading, email, password, dispatch) => {  
  setIsLoading(true);  
  dispatch(setErrorLogin(''));  
  try {  
    const response = await fetch('http://localhost:3001/api/v1/user/login', {  
      method: 'POST',  
      headers: {  
        'Content-Type': 'application/json'  
      },  
      body: JSON.stringify({ email, password })  
    });  
    if (!response.ok) {  
      throw new Error('Erreur de connexion');  
    }  
    const data = await response.json();  
    const res = await data.body;  
    dispatch(setToken(res.token));  
    dispatch(setIsAuthenticated(true));  
  } catch (error) {  
    dispatch(setErrorLogin(error.message));  
  } finally {  
    setIsLoading(false);  
  }  
}
```

Utilisation de isLoading pour bloquer une nouvelle tentative de connexion par un nouvel appel à l'api tant que celui-ci est en cours avec isLoading en true

Passage à l'api de l'email et du password dans le body

Enregistrement du token dans le store,
Mise en true de isAuthenticated dans le store

```
<button type="submit" className={`${errorLogin ? 'redAlert' : 'noAlert'} sign-in-button`}  
  disabled={isLoading}>Sign In</button>  
{errorLogin && <p>UserName or Password is incorrect</p>}
```

Gestion de l'erreur de login et Déblocage du bouton de soumission

Déconnexion du système

La gestion de la déconnexion du système par l'utilisateur :

Le composant <HeaderPage />

```
const handleLogout = () => {  
  localStorage.removeItem("token");  
  dispatch(logout())  
  navigate('/')  
}
```

Suppression du token de l'utilisateur dans le localStorage.

Réinitialisation de l'état de l'utilisateur en utilisant la fonction Redux créée pour cela.

Redirection de l'utilisateur vers la page d'accueil.

Cela permet de s'assurer que l'utilisateur est déconnecté complètement et que toutes les données relatives à son authentification sont effacées.

RENDU MAQUETTE DU COMPOSANT <HEADERPAGE />:

Gestion du rendu de la barre de navigation. Affichage différencié en fonction de la page en cours

```
return (  
  <header>  
    <nav className="main-nav">  
      <NavLink to="/" className="main-nav-logo">  
        <img src={Logo} className="main-nav-logo-image" alt="ArgentBank logo" />  
        <h1 className="sr-only">Argent Bank</h1>  
      </NavLink>  
      <div className="box_sign">  
        {isAuthenticated ?  
          (<<NavLink className="main-nav-item router-link-exact-active" to="/  
profile"><FaUserCircle className='nav-icon' />{userFirstName}</NavLink>  
          <button className="main-nav-item btnLogout" onClick={handleLogout }>  
            <FaSignInAlt className='nav-icon' />  
            <span>Sign Out</span>  
          </button></>)  
          :  
          (<NavLink className={({ isActive }) =>  
            `main-nav-item ${isActive ? ' router-link-exact-active' : ''}` to="/  
login"><FaUserCircle className='nav-icon' size="30px"/><span>Sign In</span></NavLink>  
          )  
        }  
      </div>  
    </nav>  
  </header>  
)  
}
```

```
.main-nav a.router-link-exact-active {  
  color: #42b983;  
}  
  
.FaUserCirclerouter-active {  
  color: #42b983;  
}
```


L'utilisateur ne peut voir les informations relatives à son propre profil qu'après s'être connecté avec succès

La gestion de la navigation vers la page profile de l'utilisateur

Le composant <Profile />

```
export default function Profile() {  
  document.title = 'Argent Bank - Profile Page'  
  const isAuthenticated = useSelector((state) => state.user.isAuthenticated);  
  const navigate = useNavigate();  
  
  useEffect(() => {  
    if (!isAuthenticated) {  
      navigate("/login");  
    }  
  }, [isAuthenticated, navigate])  
  
  return (isAuthenticated &&  
    <main className="main bg-dark main_profile">  
      <HeaderProfile />  
      {dataAccount.map((item, index) => (<Account key={`account-${index}`} title={item.title} amount={item.amount} text={item.text}  
    ))}  
    </main>  
  )  
}
```

Récupération de la valeur de isAuthenticated dans le store

Redirection vers la page de login si isAuthenticated a pour valeur false

Rendu conditionnel des informations de l'utilisateur

L'appel à l'api pour la récupération de l'identité de l'utilisateur

```
import { setUserFirstName, setUserLastName } from '../store/user.slice';

export const readUserIdentity = async (setIsLoading, setError, token, dispatch) => {
  setIsLoading(true);
  setError(null);
  try {
    const response = await fetch('http://localhost:3001/api/v1/user/profile', {
      method: 'POST',
      headers: {
        'Authorization': `Bearer ${token}`,
        'Content-Type': 'application/json'
      },
    });
    if (!response.ok) {
      throw new Error(response.statusText);
    }
    const data = await response.json();
    const res = await data.body;
    dispatch(setUserFirstName(res.firstName))
    dispatch(setUserLastName(res.lastName))
  } catch (error) {
    setError(error)
  }
  setIsLoading(false)
}
```

Utilisation de la méthode POST pour la récupération de l'identité de l'utilisateur comme le prévoit la documentation de l'api

Passage à l'api du token pour qu'elle autorise la récupération de l'identité de l'utilisateur

Enregistrement dans le store du prénom et du nom donnés dans le corps de la réponse de l'api,

L'utilisateur peut modifier le profil et conserver les données dans la base de données.

Gestion du formulaire pour la modification de l'identité de l'utilisateur

```
export default function HeaderProfile() {  
  const dispatch = useDispatch();  
  const userFirstName = useSelector((state) => state.user.userFirstName);  
  const userLastName = useSelector((state) => state.user.userLastName);  
  const token = useSelector((state) => state.user.token);  
  const [firstName, setFirstName] = useState(userFirstName);  
  const [lastName, setLastName] = useState(userLastName);  
  const [activeNameForm, setActiveNameForm] = useState(false);  
  const [isLoading, setIsLoading] = useState(false);  
  const [errorFirstName, setErrorFirstName] = useState('');  
  const [errorLastName, setErrorLastName] = useState('');  
  const [error, setError] = useState('');
```

mise en place d'un state au niveau du composant pour le prénom et pour le nom entrés dans le formulaire avant leur vérification et soumission à l'api pour modification de ceux-ci dans la base de données de l'api

```
  const editNameForm = () => {  
    setFirstName(userFirstName)  
    setLastName(userLastName)  
    setErrorFirstName('')  
    setErrorLastName('')  
    setActiveNameForm(!activeNameForm)  
  }  
}
```

gestion de l'ouverture et de la fermeture du formulaire de modification du prénom et du nom avec

```
useEffect(() => {  
  readUserIdentity(setIsLoading, setError, token, dispatch)  
}, [dispatch, token])
```

Récupération de l'identité de l'utilisateur déjà enregistré dans la base de données de l'api pour pouvoir l'afficher

```
useEffect(() => {  
  setFirstName(userFirstName)  
  setLastName(userLastName)  
}, [userFirstName, userLastName])
```

Gestion de l'identité à mettre dans le formulaire

Gestion du formulaire pour la modification de l'identité de l'utilisateur (suite)

Fonction de contrôle des entrées dans le formulaire du prénom et du nom

```
const handleFirstNameChange = handleNameChange(setErrorFirstName, 'firstname', setFirstName);  
const handleLastNameChange = handleNameChange(setErrorLastName, 'lastname', setLastName);
```

```
const handleUpdateIdentityUser = async (e) => {  
  e.preventDefault();  
  if(!controlLenghtName(setErrorFirstName, firstName, 'firstname') || !controlLenghtName  
    (setErrorLastName, lastName, 'lastname')) return;  
  setIsLoading(true);  
  setError(null);  
  updateUserIdentity(setIsLoading, setError, token, firstName, lastName, dispatch)  
  if (!error) {  
    setActiveNameForm(false)  
  }  
}
```

Vérification qu'aucun input n'est vide

Appel à l'api pour la modification des données de l'utilisateur

L'appel à l'api pour la modification de l'identité de l'utilisateur

```
import { setUserFirstName, setUserLastName } from '../store/user.slice';

export const updateUserIdentity = async (setIsLoading, setError, token, firstName, lastName, dispatch) => {
  setIsLoading(true);
  setError(null);
  try {
    const response = await fetch('http://localhost:3001/api/v1/user/profile', {
      method: 'PUT',
      headers: {
        'Authorization': `Bearer ${token}`,
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({ firstName, lastName })
    });
    if (!response.ok) {
      throw new Error(response.statusText);
    }
    const data = await response.json();
    const res = await data.body;
    dispatch(setUserFirstName(res.firstName))
    dispatch(setUserLastName(res.lastName))
  } catch (error) {
    setError(error)
  } finally {
    setIsLoading(false);
  }
}
```

Passage à l'api du token

Passage à l'api du prénom et du nom entrés par l'utilisateur dans le body

Enregistrement du nouveau prénom et du nouveau nom dans le store lorsque l'api répond avec succès.

Gestion de l'erreur de login et déblocage du bouton de soumission pour permettre une nouvelle modification

Utilisation de isLoading pour bloquer une nouvelle tentative de modification par un nouvel appel à l'api tant que celui-ci est en cours avec isLoading en true

PHASE 2 : FICHER DÉCRIVANT LES API PROPOSÉES POUR LES TRANSACTIONS, EN SUIVANT LES DIRECTIVES DE SWAGGER.

LA FONCTIONNALITÉ POUR LES TRANSACTIONS DOIT POUVOIR PERMETTRE AUX UTILISATEURS :

- DE VISUALISER TOUTES LEURS TRANSACTIONS POUR LE MOIS EN COURS ;
- DE VISUALISER LES DÉTAILS D'UNE TRANSACTION DANS UNE AUTRE VUE ;
- D'AJOUTER, DE MODIFIER OU DE SUPPRIMER DES INFORMATIONS SUR UNE TRANSACTION.

CONCLUSIONS :

Le token n'est pas forcément le plus sécurisé des choix possibles quant à l'authentification de l'utilisateur auprès de l'API surtout sur une application bancaire. Cependant, il peut être utilisé en conjonction avec un autre facteur d'authentification pour former une double authentification, tel que le code PIN, le mot de passe, ou une empreinte digitale pour former un système à deux facteurs.

Pour minimiser ces risques, il est important de mettre en place des mesures de sécurité pour protéger les tokens, notamment en utilisant des mécanismes de chiffrement forts, en limitant la durée de validité des tokens et en surveillant l'utilisation des tokens pour détecter toute activité suspecte.