

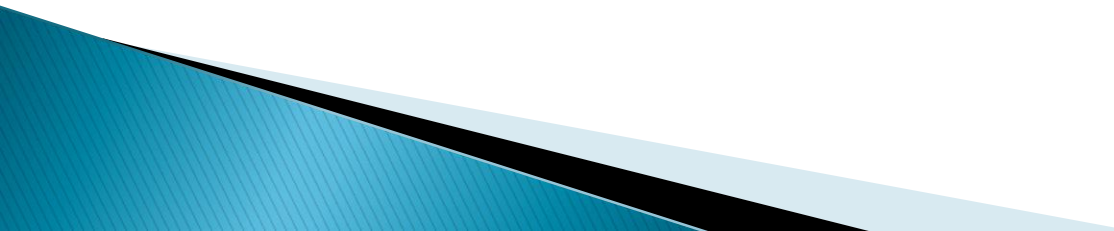
Créez un réseau social d'entreprise

Gwenaël GRESSIER

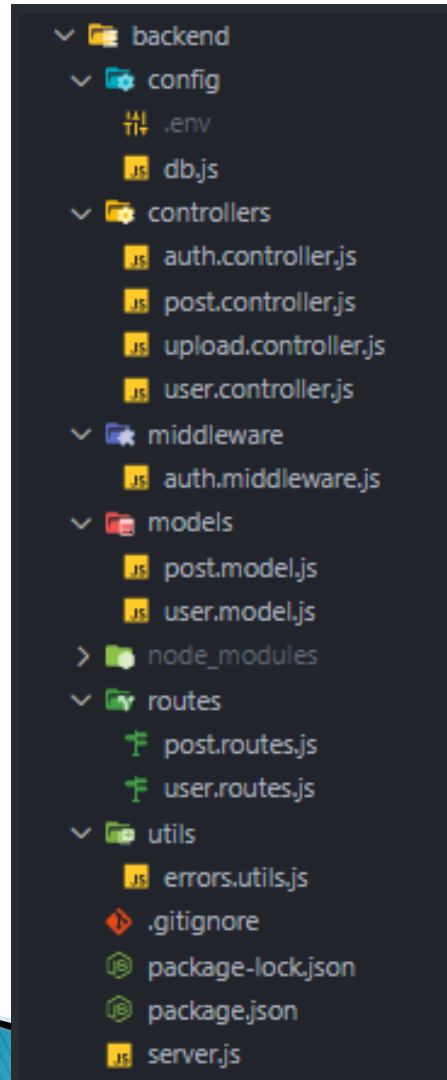
Introduction

- ▶ Pour ce projet j'incarnerais un développeur en poste depuis plus d'un an chez Connect-E une agence web
- ▶ Ou j'ai été missionné pour créer un réseau social interne pour les employés de Groupomania. Le but de cet outil est de faciliter les interactions entre collègues.

Mon Projet MERN

- ▶ Pour ce projet j'ai choisie une approche dite MERN stack ce qui veut dire que je vais faire appelle à:
 - ▶ MongoDB
 - ▶ Express
 - ▶ React
 - ▶ Node.js
- 

Le code



Config

Ici sera stock tout les donner sensible et la connexion à ma base de donne

Controlers

C'est mon code. En gros qui dit qui fait quoi

Middleware

Se sont mes autorisation

Models

Sert à définir mes schémas (interfaces) post et user

Routes

Définit les règles du CRUD et ses la ou je vais définir les chemin avec ou sans autorisation

Utils

Va me servir à définir les messages d'erreur

server.js

Définit mes routes et mes headers(CORS)

Back-end/server.js a voir si je garde

```
14 const corsOptions = {  
15   origin: process.env.CLIENT_URL,  
16   credentials: true,  
17   allowedHeaders: ["sessionId", "Content-Type"],  
18   exposedHeaders: ["sessionId"],  
19   methods: "GET,HEAD,PUT,PATCH,POST,DELETE",  
20   preflightContinue: false,  
21 };
```

→ Définition des CORS de la ligne 14 à 21

```
28 // jwt token  
29 app.get("*", checkUser);  
30 app.get("/jwtid", requireAuth, (req, res) => {  
31   res.status(200).send(res.locals.user._id);  
32 });  
33  
34 // routes  
35 app.use("/api/user", userRoutes);  
36 app.use("/api/post", postRoutes);
```

→ Gestion de la vérification de mes users et de la gestion des tokens

→ principaux chemin de l'API: user et post

Back-end / utils / errors-utils.js

```
1 //messages d'erreur à la création de profil
2 module.exports.signUpErrors = (err) => {
3   let errors = { pseudo: "", email: "", password: "" };
4   //Le pseudo n'est pas supporté
5   if (err.message.includes("pseudo")) errors.pseudo = "Pseudo invalide";
6   //L'email n'est pas valide en temps qu'email
7   if (err.message.includes("email")) errors.email = "Email invalide";
8   //mot de passe trop court ou trop long
9   if (err.message.includes("password"))
10     errors.password = "Le mot de passe doit faire 6 caractères minimum";
11   /*Le code d'erreur 11000 s'affiche quand un élément unique existe déjà,
12    on distingue simplement Les pseudo et mails dans ce cas de figure*/
13   if (err.code === 11000 && Object.keys(err.keyValue)[0].includes("pseudo"))
14     errors.pseudo = "Ce pseudo est déjà enregistré";
15
16   if (err.code === 11000 && Object.keys(err.keyValue)[0].includes("email"))
17     errors.email = "Cet email est déjà enregistré";
18
19   return errors;
20 };
21 //erreurs à la connexion
22 module.exports.signInErrors = (err) => {
23   let errors = { email: "", password: "" };
24
25   if (err.message.includes("email")) errors.email = "Email Inconnu";
26
27   if (err.message.includes("password"))
28     errors.password = "Le mot de passe ne correspond pas";
29
30   return errors;
31 };
32
33 //erreurs d'upload
34 module.exports.uploadErrors = (err) => {
35   let errors = { format: "", maxSize: "" };
36
37   if (err.message.includes("invalid file"))
38     errors.format = "Format incompatible";
39
40   if (err.message.includes("max size"))
41     errors.maxSize = "Le fichier dépasse 5000ko";
42
43   return errors;
44 };
```

- ▶ J'ai créé un fichier errors - utils qui me sert à centraliser mes erreurs pour les parties inscription, connexion et d'upload d'image

Back-end/models par exemple pour le user

```
8  const userSchema = new mongoose.Schema(  
9    [  
10     pseudo: {  
11       type: String,  
12       required: true,  
13       minLength: 3,  
14       maxLength: 30,  
15       unique: true,  
16       trim: true,  
17     },  
18     email: {  
19       type: String,  
20       required: true,  
21       validate: [isEmail],  
22       lowercase: true,  
23       unique: true,  
24       trim: true,  
25     },  
26     password: {  
27       type: String,  
28       required: true,  
29       max: 1024,  
30       minlength: 6,  
31     },  
32     picture: {  
33       type: String,  
34       default: "./img/random-user.png",  
35     },  
36     likes: {  
37       type: [String],  
38     },  
39     isAdmin: {  
40       type: Boolean,  
41       allowNull: false,  
42       defaultValue: false,  
43     },  
44   ],  
45 );
```

- ▶ Grace a l'importation du package mongoose pour la base de donnée je peut faire la définition de mon schéma sous forme de JSON
- ▶ Plusieurs type possible: String, number, Array...
- ▶ Ici ligne 39 à 43 j'ai rajouter un champ isAdmin se ce dernier qui va me permettre de définir mon user admin

Back-end/models/user.model.js

```
53 // fonction pour "saler"(crypter) les mots de passe.
54 userSchema.pre("save", async function (next) {
55     const salt = await bcrypt.genSalt();
56     this.password = await bcrypt.hash(this.password, salt);
57     next();
58 });
59
60 //fonction pour décrypter un mot de passe
61 userSchema.statics.login = async function (email, password) {
62     const user = await this.findOne({ email });
63     if (user) {
64         const auth = await bcrypt.compare(password, user.password);
65         if (auth) {
66             return user;
67         }
68         throw Error("incorrect password");
69     }
70     throw Error("incorrect email");
71 };
```

Utilisation du hook de mongoose .pre sur la fonction ligne 54 à 58 cela me permet d'exécuter ma fonction avant mon user schema

Et enfin de la ligne 61 à 71 j'ai une Fonction qui permet de comparer mon mot de passe en fonction du mail renseigné

Back-end/middleware/auth.middleware.js

```
1 const jwt = require("jsonwebtoken");
2 const UserModel = require("../models/user.model");
3
4 module.exports.checkUser = (req, res, next) => {
5   const token = req.cookies.jwt; //recupere le token dans les cookies
6   if (token) {
7     jwt.verify(
8       token,
9       process.env.TOKEN_SECRET,
10      async (err, decodedToken) => {
11        if (err) {
12          res.locals.user = null;
13          res.cookie("jwt", "", { maxAge: 1 }); //si une err sur le token, on supprime le cookie
14          next();
15        } else {
16          let user = await UserModel.findById(decodedToken.id);
17          res.locals.user = user;
18          console.log(user);
19          next();
20        }
21      }
22    );
23  } else {
24    res.locals.user = null;
25    next();
26  }
27 };
28
29 //Verification de l'existence et de la validité du token de l'utilisateur
30 module.exports.requireAuth = (req, res, next) => {
31   const token = req.cookies.jwt;
32   if (token) {
33     jwt.verify(
34       token,
35       process.env.TOKEN_SECRET,
36      async (err, decodedToken) => {
37        if (err) {
38          console.log(err);
39          res.send(200).json("no token");
40        } else {
41          console.log(decodedToken.id);
42          next();
43        }
44      }
45    );
46  } else {
47    console.log("no token");
48  }
49 };
```

Ces dans cette
partie que je fait la
gestion de mes
tokens
récupération
suppression et
validation

Back-end/controllers/auth.controllers.js

```
7  const maxAge = 1000 * 60 * 60 * 24;
8  |
9  > /**...
15 const createToken = (id) => {
16   return jwt.sign({ id }, process.env.TOKEN_SECRET, {
17     expiresIn: maxAge,
18   });
19 };
20
21 > /**...
27 module.exports.signUp = async (req, res) => {
28   const { pseudo, email, password } = req.body;
29
30   try {
31     /*Lorsqu'on veut créer un utilisateur, ces trois éléments
32     obligatoires doivent être présents, sinon ça passe au catch en dessous.*/
33     const user = await UserModel.create({ pseudo, email, password });
34     res.status(201).json({ user: user._id });
35   } catch (err) {
36     const errors = signUpErrors(err);
37     res.status(200).send({ errors });
38   }
39 };
40
41 > /**...
47 module.exports.signIn = async (req, res) => {
48   const { email, password } = req.body;
49
50   try {
51     const user = await UserModel.login(email, password);
52     const token = createToken(user._id);
53     res.cookie("jwt", token, { httpOnly: true, maxAge });
54     res.status(200).json({ user: user._id });
55   } catch (err) {
56     const errors = signInErrors(err);
57
58     res.status(200).json({ errors });
59   }
60 };
61
62 > /**...
68 module.exports.logout = async (req, res) => {
69   //on attribue un cookie vide qui va vivre 1ms, puis rediriger l'utilisateur
70   res.cookie("jwt", "", { maxAge: 1 });
71   res.redirect("/api/post");
72 };
```

Ligne 7 je mets un temps de valider le token égal à 24h

Lorsque l'on veut créer un utilisateur vérifie que pseudo, l'email et le password sont présents

Fonction de connexion

Fonction de déconnexion qui supprime les cookies et qui renvoie à la page principale

différent renvoie vers mon fichier utils/error

Post controller

```
17 module.exports.createPost = async (req, res) => {
18   let fileName;
19   if (req.file) {
20     try {
21       if (
22         req.file.mimetype !== "image/jpg" &&
23         req.file.mimetype !== "image/png" &&
24         req.file.mimetype !== "image/jpeg" &&
25         req.file.mimetype !== "image/gif"
26       ) {
27         throw Error("invalid file");
28       }
29       //verif du poids du fichier
30       if (req.file.size > 5000000) throw Error("max size");
31     } catch (err) {
32       const errors = uploadErrors(err);
33       return res.status(201).json({ errors });
34     }
35     //nouveau nom du fichier
36     const images = req.file.mimetype.split("/");
37     const extension = images.slice(-1).pop();
38     fileName = req.body.posterId + Date.now() + "." + extension;
39
40     //stockage de la nouvelle image.
41     fs.writeFile(
42       `../frontend/public/uploads/posts/${fileName}`,
43       req.file.buffer,
44       (err) => {
45         if (err) throw err;
46       }
47     );
48   }
49
50   const newPost = new postModel({
51     posterId: req.body.posterId,
52     message: req.body.message,
53     picture: req.file ? `../uploads/posts/` + fileName : "",
54     likers: [],
55     comments: [],
56   });
57
58   try {
59     const post = await newPost.save();
60     return res.status(201).json(post);
61   } catch (err) {
62     return res.status(400).send(err);
63   }
64 };
```

▶ Je définit tout mon CRUD:

▶ Le CREATE

définition de mes mimetype
de la ligne 22 à 25

Définit un poid max de 5MO

Modifie le nom du fichier

Stock l'image dans le dossier
correspondant

Création de mon post avec
son model .

Save le nouveau post dans la
base de donnée

Post controller

```
66 //CRUD : Read
67 > /** ...
73 module.exports.readPost = (req, res) => {
74   PostModel.find((err, docs) => {
75     if (!err) res.send(docs);
76     else console.log("Error to get data : " + err);
77   }).sort({ createdAt: -1 });
78 };
79
80 //CRUD : Update
81 > /** ...
88 module.exports.updatePost = (req, res) => {
89   if (!ObjectID.isValid(req.params.id))
90     return res.status(400).send("ID unknown : " + req.params.id);
91
92   const updatedRecord = {
93     message: req.body.message,
94   };
95
96   PostModel.findByIdAndUpdate(
97     req.params.id,
98     { $set: updatedRecord },
99     { new: true },
100     (err, docs) => {
101       if (!err) res.send(docs);
102       else console.log("update error : " + err);
103     }
104   );
105 };
106
107 //CRUD : Delete
108 > /** ...
115 module.exports.deletePost = (req, res) => {
116   if (!ObjectID.isValid(req.params.id))
117     //
118     return res.status(400).send("ID unknown : " + req.params.id); //
119
120   PostModel.findByIdAndRemove(req.params.id, (err, docs) => {
121     if (!err) {
122       fs.unlink(docs.picture, () => {});
123       res.send(docs);
124     } else console.log("Deleting error : " + err);
125   });
126 };
```

Le Read qui va récupérer mes posts dans ma base de donnée

grâce au `.sort({creatAT:-1})` je vais pouvoir afficher les post dans l'ordre anté-chronologique

Le Update qui me sert à re-inject le nouveau contenu de mon message

Et le DELETE qui cherche dans la base de donner le message par son id et le supprime grâce a `findByIdAndRemove`

Ensuite je fait un `fs.unlink` de mon image afin de la supprimer

Post controler les like et unlike

```
102 //Like-post
103 module.exports.likePost = async (req, res) => {
104   if (!ObjectID.isValid(req.params.id))
105     return res.status(400).send("ID unknown : " + req.params.id);
106
107   try {
108     let updatedLikers = await PostModel.findByIdAndUpdate(
109       req.params.id,
110       /*j'ajoute avec $addToSet l'id dans le tableau likers du post*/
111       { $addToSet: { likers: req.body.id } },
112       //true pour renvoyer le document modifié
113       { new: true }
114     );
115     res.json({ updatedLikers });
116     let updatedLikes = await UserModel.findByIdAndUpdate(
117       req.body.id,
118       { $addToSet: { likes: req.params.id } },
119       { new: true }
120     );
121     res.json({ updatedLikes });
122   } catch (err) {
123     return;
124   }
125 };
126
127 //unlike-post
128 module.exports.unlikePost = async (req, res) => {
129   if (!ObjectID.isValid(req.params.id))
130     //
131     return res.status(400).send("ID unknown : " + req.params.id);
132
133   try {
134     let updatedLikers = await PostModel.findByIdAndUpdate(
135       req.params.id,
136       { $pull: { likers: req.body.id } },
137       { new: true }
138     );
139     res.json({ updatedLikers });
140     let updatedLikes = await UserModel.findByIdAndUpdate(
141       req.body.id,
142       { $pull: { likes: req.params.id } },
143       { new: true }
144     );
145     res.json({ updatedLikes });
146   } catch (err) {
147     return;
148   }
149 };
```

Grace à la commande `addToSet` j'ajoute la nouvelle id post dans le tableaux de mon user

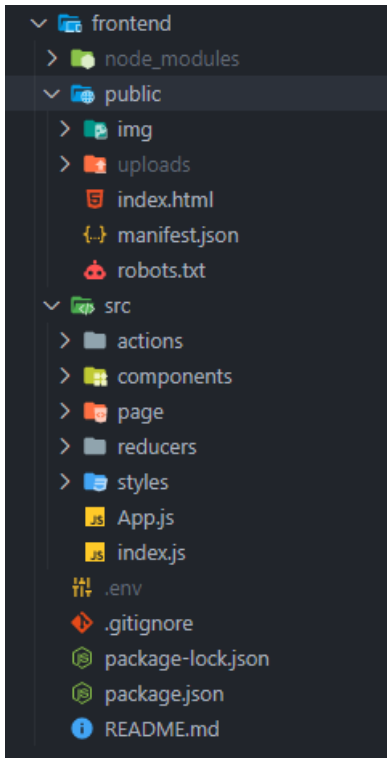
Et je fait de même pour mon post à l'inverse en ajoutant l'id de l'user a mon post

En se qui concerne le unlike ses la même fonction que la précédente sauf que au lieu de `addToSet` je pull mes ids

front

- ▶ Donc pour ma partie front je vais utiliser react et redux ainsi que du sass pour la mise en page, je ne m'attarderais pas sur se dernier sauf si vous le souhaitais
- ▶ Pour la mise en place de react j'ai utiliser la commande `npx creat-react-app`

Architecture du code



public

Ici sera stocké tout les images de mon site et les uploads des utilisateurs
Il y a aussi la page index qui est ma page de référence pour tout mon site

src

C'est le dossier ou il va y avoir tout mon code

actions

Se sont mes action qui vont déclencher mes reducers

components

Ses ici que son coder les composant qui vont formée ma page

page

Définit l'ensemble des page de mon site ses ici que nous faisons l'assemblage de nos composant

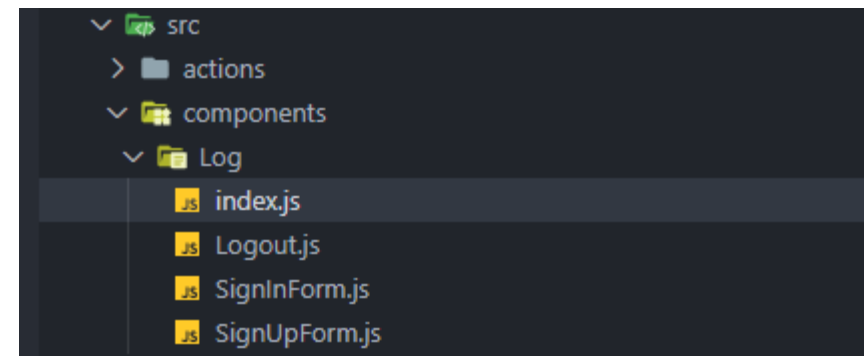
reducers

Gestion de mes reducers

Styles

Ses ici que sera définit toute la mise en page du site j'ai utiliser la méthode 7:1 pour une meilleur maintenabilité

Authentification



```
5 // eslint-disable-next-line
6 /*suivant la page de par la quelle on appelle le component on aura signup ou signin afficher*/
7 const Log = (props) => {
8   const [signUpModal, setSignUpModal] = useState(props.signup);
9   const [signInModal, setSignInModal] = useState(props.signin);
10
11   // eslint-disable-next-line
12   /*si on appuie sur s'inscrire change les etat si on appuie sur se connecter change les etat a l'inverse*/
13   const handleModals = (e) => {
14     if (e.target.id === "register") {
15       setSignInModal(false);
16       setSignUpModal(true);
17     } else if (e.target.id === "login") {
18       setSignUpModal(false);
19       setSignInModal(true);
20     }
21   };
22 }
```

➤ Déclaration de mes variables vi les usestate

➤ Fonction qui gères l'état de mes variables suivant de par qui elle a étai appelé

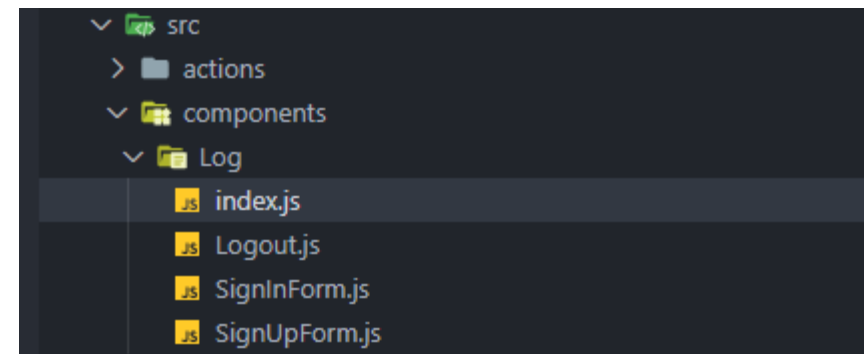
```
const [signUpModal, setSignUpModal] = useState(props.signup);
```

ma variable

Commande qui sert à modifier ma variable

Props useState sert à déclarer l'état initial de ma variable

Authentification

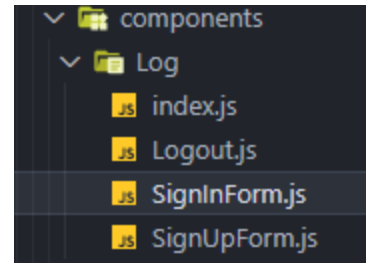


```
23 return (  
24   <div className="connection-form">  
25     <div className="form-container">  
26       <ul>  
27         <li  
28           onClick={handleModals}  
29           id="register"  
30           className={signInModal ? "active-btn" : null}  
31         >  
32           S'inscrire  
33         </li>  
34         <li  
35           onClick={handleModals}  
36           id="login"  
37           className={signInModal ? "active-btn" : null}  
38         >  
39           Se connecter  
40         </li>  
41       </ul>  
42       {signInModal && <SignUpForm />}  
43       /*si on click sur inscription alors SignUpForm*/  
44  
45       {signInModal && <SignInForm />}  
46       /*si on click sur se connecter alors SignInForm*/  
47     </div>  
48   </div>  
49 );  
50  
51 export default Log;
```

- Dans le return nous allons retourner toute notre mise en page
- Ligne 32 et 39 Au click nous allons déclencher la fonction handleModal qui sera traite l'information en fonction de l'id
- La class de mon bouton changera en fonction de mes variables
- Condition pour l'appelle de mes composants

Par soucie de temps je vous passerais toute les parties return sauf celle que je juge pertinente nous pourrons cependant revenir ensemble sur ces dernière à la fin si vous avez la moindre question

Components/SignInForm



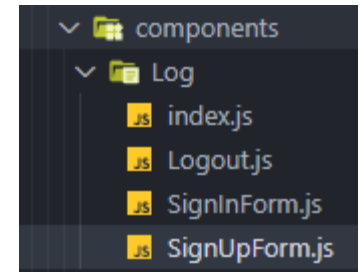
```
8  const handleLogin = (e) => {
9    e.preventDefault();
10   const emailError = document.querySelector(".email.error");
11   const passwordError = document.querySelector(".password.error");
12
13   axios({
14     method: "post",
15     url: `${process.env.REACT_APP_API_URL}api/user/login`,
16     withCredentials: true,
17     data: { email, password },
18   })
19   .then((res) => {
20     if (res.data.errors) {
21       emailError.innerHTML = res.data.errors.email;
22       passwordError.innerHTML = res.data.errors.password;
23     } else {
24       window.location = `/`;
25     }
26   })
27   .catch((err) => {
28     console.log(err);
29   });
30 }
```

Envoie une requêtes à la base de donnée avec axios (cette partie va être récurrente donc je vais l'expliquer une fois ici et je ne l'expliquerais pas pour les fois prochaine)

Traite les messages d'erreur

Si aucune erreur sur l'adresse mail et le mot de passe alors me redirige vers la page principal

Components / SignUpForm



```
23 passwordConfirmError.innerHTML = "";
24 termsError.innerHTML = "";
25
26 if (password !== controlPassword || !terms.checked) {
27   if (password !== controlPassword)
28     passwordConfirmError.innerHTML =
29       "Les mots de passe ne correspondent pas";
30
31   if (!terms.checked)
32     termsError.innerHTML =
33       "Veuillez valider les conditions générales";
34 } else {
35   await axios({
36     method: "post",
37     url: `${process.env.REACT_APP_API_URL}api/user/register`,
38     data: {
39       pseudo,
40       email,
41       password,
42     },
43   })
44     .then((res) => {
45       console.log(res);
46       if (res.data.errors) {
47         pseudoError.innerHTML = res.data.errors.pseudo;
48         emailError.innerHTML = res.data.errors.email;
49         passwordError.innerHTML = res.data.errors.password;
50       } else {
51         setFormSubmit(true);
52       }
53     })
54     .catch((err) => console.log(err));
55 }
56
```

Je mets des strings vide dans mes const

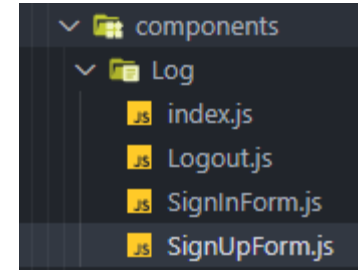
Vérifie que les deux mots de passe soit similaire

Validation des conditions général avec acceptation de cookies

Gestion des erreurs

Passage à true de la constante FormSubmit

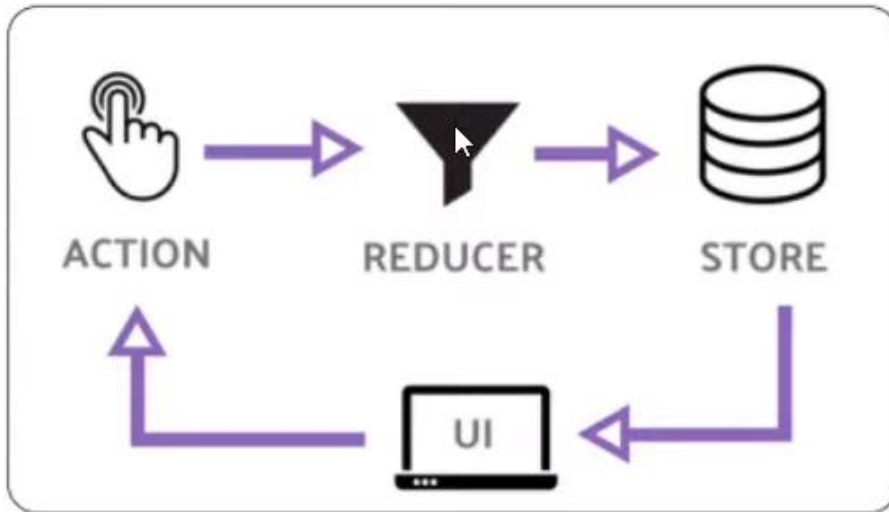
Components / SignUpForm



```
63 | <|
64 |   <SignInForm />
65 |   <h4 className="success">...
67 |   </h4>
68 | </>
```

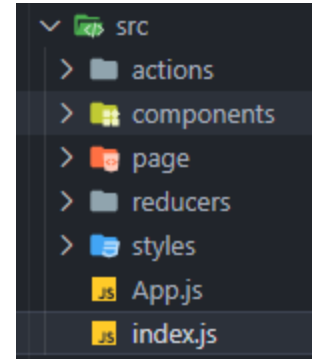
Mise en place de fragment car en jsx pour un bon fonctionnement si deux parties sont au même niveau elle doivent être encapsuler

Redux



- ▶ Alors redux comment sa marche
- ▶ comme vous le voyais il y a 3 élément 1 action 1 reducer 1 store
- ▶ il faut comprendre ses 3 élément donc le store ses la ou on va stocker toute nos donner.
- ▶ Toute les donner stocker dedans sont accessible a n'importe quelle component dans la userinterface
- ▶ donc par exemple on se dit tien je vais follow un utilisateur donc ceci est une action sur la userinterface
- ▶ a partir de la l'action va dire a la base de donne telle user follow tel user
- ▶ et sa passe au reducer qui va dire la meme chose au store et force de redux ses que toute info change dans le store l'est immédiatement a l'écran
- ▶ donc par exemple le bouton ou il y avais écrit suivre deviendra abonner à la personne
- ▶ le user aura un component de suivi des user il va changer instantanément alors que se sont des component qui sont différent et qui n'interagisse pas entre eux
- ▶ mes leur point commun ses que il prenne tous leur data du store

Mise en place de redux



```
11 // dev tools
12 import { composeWithDevTools } from "redux-devtools-extension";
13
14 const store = createStore(
15   rootReducer,
16   composeWithDevTools(applyMiddleware(thunk))
17 );
18
19 store.dispatch(getUsers());
20
21 const container = document.getElementById("root");
22 const root = createRoot(container);
23
24 root.render(
25   <Provider store={store}>
26     <App tab="home" />
27   </Provider>
28 );
```

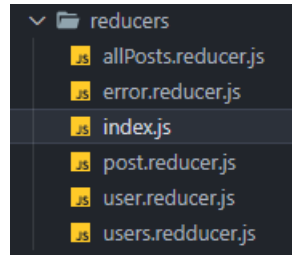
Mise en place d'un devtools pour redux a enlever lors de la mise en prod

Création de mon store

Thunk est une sorte de middleware qui permet de faire des requête asynchrone avec redux

Mise en place de notre provider qui sert a connecter notre store à react

Je combine ensemble tout mes reducer



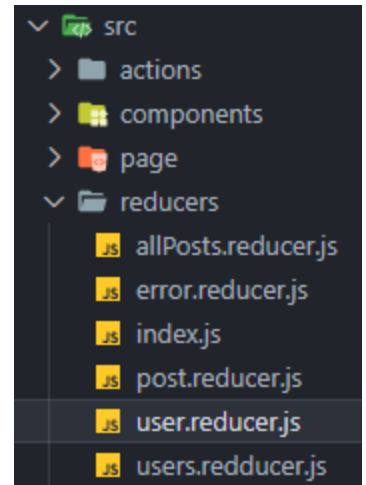
```
1 import { combineReducers } from "redux";
2 import userReducer from "../user.reducer";
3 import usersReducer from "../users.reducer";
4 import postReducer from "../post.reducer";
5 import errorReducer from "../error.reducer";
6 import allPostsReducer from "../allPosts.reducer";
7
8 export default combineReducers({
9   userReducer,
10  usersReducer,
11  postReducer,
12  errorReducer,
13  allPostsReducer,
14 });
```

- ▶ Le fichier index.js va être celui qui me permet d'organiser tout les reducers
- ▶ Import de tout mes reducer
- ▶ combine ensemble tout mes reducer

```

1 import { GET_USER, UPDATE_BIO, UPLOAD_PICTURE } from "../actions/user.actions";
2
3 const initialState = {};
4
5 export default function userReducer(state = initialState, action) {
6   switch (action.type) {
7     case GET_USER:
8       return action.payload;
9     case UPLOAD_PICTURE:
10      return {
11        ...state, //retourne toute les donnees
12        picture: action.payload, //et modifie que la picture
13      };
14     case UPDATE_BIO:
15      return {
16        ...state,
17        bio: action.payload,
18      };
19     default:
20      return state;
21   }
22 }
23

```

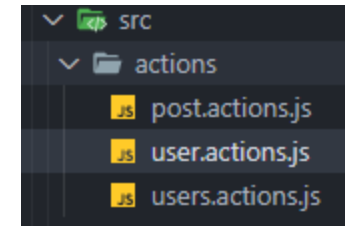


Tout nos reducers doivent avoir un initialState

Creation de ma fonction avec en parametre mon state et une action

me retourne la data en fonction de l'action qui a été appelé

Quand nous modifions quelque chose du store si nous ne voulons pas tout écraser il est important de rappeler toutes les données

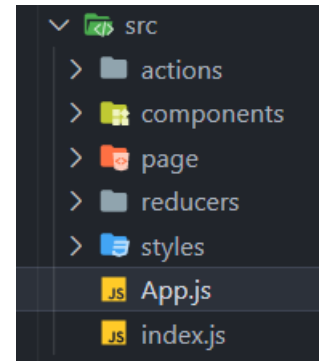


```
1 import axios from "axios";
2
3 export const GET_USER = "GET_USER";
4 export const UPLOAD_PICTURE = "UPLOAD_PICTURE";
5 export const UPDATE_BIO = "UPDATE_BIO";
6 export const GET_USER_ERRORS = "GET_USER_ERRORS";
7
8 export const getUser = (uid) => {
9   //dispatch ses se qui est envoyer au reducer
10  return (dispatch) => {
11    return axios
12      .get(`${process.env.REACT_APP_API_URL}api/user/${uid}`)
13      .then((res) => {
14        dispatch({ type: GET_USER, payload: res.data });
15      })
16      .catch((err) => console.log(err));
17  };
18 };
19
20 export const uploadPicture = (data, id) => {
21  return (dispatch) => {
22    return axios
23      .post(`${process.env.REACT_APP_API_URL}api/user/upload`, data)
24      .then((res) => {
25        return axios
26          .get(`${process.env.REACT_APP_API_URL}api/user/${id}`)
27          .then((res) => {
28            dispatch({
29              type: UPLOAD_PICTURE,
30              payload: res.data.picture,
31            });
32          });
33      })
34      .catch((err) => console.log(err));
35  };
36 };
```

Ici je défini 'la table des matières' de toute nos actions

Dispatch ses se que je mets dans mon reducer

Et récupère les info pour les faire passe a mon reducer (donc les GET_USER contiendra le res.data en info)



```
12 const App = () => {
13   const [uid, setUid] = useState(null);
14   const dispatch = useDispatch();
15   //a chaque fois que app est appelé controle du token
16   useEffect(() => {
17     const fetchToken = async () => {
18       await axios({
19         method: "get",
20         url: `${process.env.REACT_APP_API_URL}jwtid`,
21         withCredentials: true,
22       })
23       .then((res) => setUid(res.data)) //fait evoluer le uid
24       .catch((err) => console.log("No token"));
25     };
26     fetchToken();
27     if (uid) dispatch(getUser(uid));
28   }, [uid, dispatch]); // [uid] à chaque fois que uid evolue tu rejoue App
29
30   return (
31     <UidContext.Provider value={uid}>
32       <BrowserRouter>
33         <Navbar />
34         <LeftNav />
35         <Routes>
36           <Route path="/" element={<Home />} />
37           <Route path="/profil" element={<Profil />} />
38           /*path="*" fonctionne si jamais l'url ne correspond à rien déclaré au dessus */
39           <Route path="*" element={<Home />} />
40         </Routes>
41       </BrowserRouter>
42     </UidContext.Provider>
43   );
44 };
45
46 export default App;
```

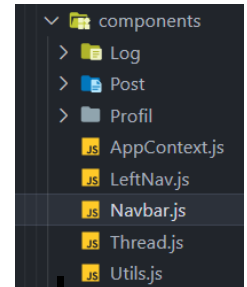
A chaque fois que App est appelé nous vérifions le token

appelle de mon getUser avec en param mon uid

Suivant le chemin taper en adresse sa va me renvoyer sur les component suivant

Apelle de mon component appcontext qui va me permettre de stocker mon user id au plus aux de mon application

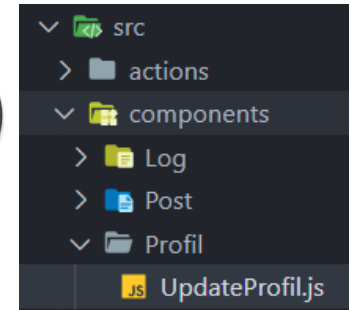
Components / Navbar.js



```
7  const Navbar = () => {  
8    const uid = useContext(UidContext);  
9    const userData = useSelector((state) => state.userReducer);  
10  
11    return (  
12      <nav>  
13        <div className="nav-container">  
14          <div className="logo">  
15            <NavLink to="/">  
16              <div className="logo">  
17                  
18                <h3>Groupomania</h3>  
19              </div>  
20            </NavLink>  
21          </div>  
22          {uid ? (  
23            <ul>  
24              <li className="welcome">  
25                <NavLink to="/profil">  
26                  <h5>Bienvenue {userData.pseudo}</h5>  
27                </NavLink>  
28              </li>  
29              <Logout />  
30            </ul>  
31          ) : (  
32            <ul>  
33              <li>  
34                <NavLink to="/profil">  
35                    
36                </NavLink>  
37              </li>  
38            </ul>  
39          )}  
40        </div>  
41      </nav>  
42    );  
43  };
```

La mise en place
du reducer va me
permettre de
pouvoir récupérer
ma valeur
dynamique comme
ici mon pseudo

UpdateProfil (DeleteProfil)

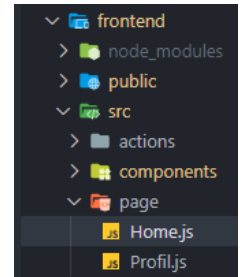


```
7  const UpdateProfil = () => {
8    const userData = useSelector((state) => state.userReducer);
9
10   const deleteUser = async () => {
11     await axios({
12       method: "DELETE",
13       url: `${process.env.REACT_APP_API_URL}api/user/${userData._id}`,
14       withCredentials: true,
15     })
16     .then(async () => {
17       await axios({
18         method: "get",
19         url: `${process.env.REACT_APP_API_URL}api/user/logout`,
20         withCredentials: true,
21       }).catch((err) => console.log(err));
22       window.location = "/";
23     })
24     .catch((err) => {
25       console.log(err);
26     });
27   };
28 }
```

useSelector sert à récupérer les info du store

lors de la suppression d'un user je le déconnecte aussi

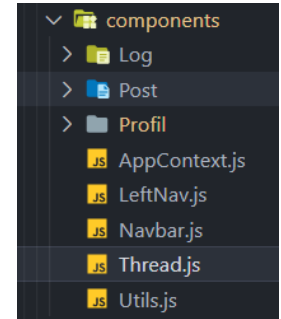
et je le redirige vers la page d'accueil



```
7  const Home = () => {  
8      const uid = useContext(UidContext);  
9      const showlogin = () => {  
10         if (uid !== "") {  
11             return <NewPostForm />;  
12         }  
13         return <Log signin={true} signup={false} />;  
14     };  
15  
16     return (  
17         <div className="home">  
18             <div className="main">  
19                 <div className="home-header">{showlogin()}</div>  
20                 <Thread />  
21             </div>  
22         </div>  
23     );  
24 };  
25  
26 export default Home;
```

Si je suis
connecter il
m'affiche le
component
newpost form
sinon la page de
log

Fil d'actualité infinite scroll

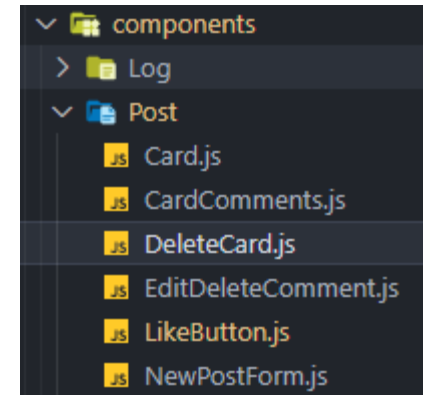


```
7  const Thread = () => {
8    const [loadPost, setLoadPost] = useState(true);
9    const [count, setCount] = useState(5);
10   const dispatch = useDispatch();
11   const posts = useSelector((state) => state.postReducer);
12
13   const loadMore = () => {
14     //si on arivent en bas de la scroll barre alor on r'affiche les 5 prochains posts
15     if (
16       window.innerHeight + document.documentElement.scrollTop + 1 >
17       document.scrollingElement.scrollHeight
18     ) {
19       setLoadPost(true); //on affiche les 5 prochains posts
20     }
21   };
22
23   useEffect(() => {
24     if (loadPost) {
25       dispatch(getPosts(count)); //recupere les 5 prochains posts
26       setLoadPost(false); //passe a false pour ne pas charger de nouveaux posts
27       setCount(count + 5); //incrimente le nombre de posts a recuperer
28     }
29
30     window.addEventListener("scroll", loadMore);
31     return () => window.removeEventListener("scroll", loadMore);
32   }, [loadPost, dispatch, count]);
33
34   return (
35     <div className="thread-container">
36       <ul>
37         {!isEmpty(posts[0]) &&
38           posts.map((post) => {
39             return <Card post={post} key={post._id} />;
40           })}
41       </ul>
42     </div>
43   );
44 }
```

- ▶ Gestion d'un infinite scroll
- ▶ Si on arrive a 1px du bas la scroll-bar alors on affiche les 5 message suivant

Gestion des droits d'admin

Vérification si on est l'auteur ou si on est un admin alors les boutons de suppression seront disponibles

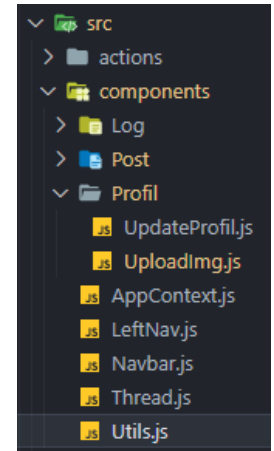


```
35     return (  
36       <div className="edit-comment">  
37         {(isAuthor || userData.isAdmin) && edit === false && (  
38           <span onClick={() => setEdit(!edit)}>  
39               
40           </span>  
41         )}  
42         {(isAuthor || userData.isAdmin) && edit && (  
43           <span onClick={() => setEdit(!edit)}>  
44               
45           </span>  
46         )}  
47       </div>  
48     )  
49   )  
50 }
```

Chose importante en jsx lors des condition les && ont la priorité sur les || donc il faut mettre des parenthèses sur les conditions si on ne veut pas avoir des erreurs

utils

```
1 > /**...
2
3 export const dateParser = (num) => {
4   //creation de mes option d'affichage de la date
5   let options = {
6     hour: "2-digit",
7     minute: "2-digit",
8     second: "2-digit",
9     weekday: "long",
10    year: "numeric",
11    month: "short",
12    day: "numeric",
13  };
14  //recupere la date en millisecondes depuis le 1er janvier 1970
15  let timestamp = Date.parse(num);
16  //convertie ma date en millisecondes au format de date fr
17
18  let date = new Date(timestamp).toLocaleDateString("fr-FR", options);
19
20  return date.toString();
21 };
22
23 > /**...
24
25 export const timestampParser = (num) => {
26   let options = {
27     hour: "2-digit",
28     minute: "2-digit",
29     second: "2-digit",
30     weekday: "long",
31     year: "numeric",
32     month: "short",
33     day: "numeric",
34   };
35
36   let date = new Date(num).toLocaleDateString("fr-FR", options);
37
38   return date.toString();
39 };
```



```
_id: ObjectId('62cd12f4a5aed685c2b7256b')
posterId: "62cd12daa5aed685c2b72562"
message: "qzzfzb"
picture: "./uploads/posts/62cd12daa5aed685c2b725621657606900193.png"
likers: Array
comments: Array
  0: Object
    commenterId: "62cd12daa5aed685c2b72562"
    commenterPseudo: "test"
    text: "wxfcg"
    timestamp: 1657612866201
    _id: ObjectId('62cd2a423abeb2c26b73ae1f')
    createdAt: 2022-07-12T06:21:40.195+00:00
    updatedAt: 2022-07-12T08:01:06.202+00:00
    __v: 0
```


annexe



MongoDB

- ▶ MongoDB permet de gérer les bases de données. Il se différencie du **SQL**. MongoDB est utilisé dans la MERN stack car les données se manipulent sous format JSON et il est vraiment très simple de transformer des données **JavaScript vers MongoDB** et inversement grâce à des librairies tel que mongoose par exemple.

Express

- ▶ Express est un langage de **modélisation des données** qui spécifie formellement les données. Il donne la capacité de développer une **application web complexe**. Express fait appel au système de middleware qui est dédié à la gestion **d'application web** de grande complexité. Express est dédié à la création de la REST API. La REST API correspond à un site web qui va récupérer des données via des requêtes HTTP. Ces requêtes sont exécutées avec le langage React.js.

React.js

- ▶ React.js est une **bibliothèque JavaScript** développée par Facebook depuis 2013. React.js développe la partie front-end du site web. Ce langage est réputé pour **accélérer les vitesses de chargement de sites internet**. Les animations, chargements et tout autre transition sont beaucoup plus rapides. Netflix utilise du React.js par exemple, c'est le meilleur choix technologique si vous souhaitez développer **une application web**, une application mobile ou encore un logiciel rapidement.



Node.js

- ▶ Node.js fonctionne **avec le JavaScript**. Ces deux langages sont fortement liés et sont très simples à faire interagir entre eux. Par conséquent, un **développeur maîtrisant le JavaScript** n'aura aucune difficulté à apprendre du Node.JS.
- ▶ En outre, Node.JS a la particularité d'être un langage extrêmement rapide, l'un des langages les plus rapides de tous les autres langages de la programmation informatique, car c'est également un langage asynchrone. Des entreprises de renom telles que Facebook utilisent Node.js.