Document de mi-parcours. Technologies du web

Schéma initial des composants React

Point de départ pour l'analyse et l'évolution

Sara Castro López
sara.castro_lopez@etu.sorbonne-universite.fr
Pablo Alcalá Mato
pablo.alcala@etu.sorbonne-universite.fr

Contents

1 Graphe des dépendances entre les composants **2**

2 Présentation et fonctionnement des composants **2**

2.1 main.jsx . . . 2
2.2 App.jsx . . . 2
2.3 Login.jsx . . . 3
2.4 Enregistrement.jsx . . . 3
2.5 PagePrincipale.jsx . . . 3
2.6 SideBar.jsx . . . 3
2.7 Header.jsx . . . 4
2.8 Recherche.jsx . . . 4
2.9 AdminUtilities.jsx . . . 4
2.10 Pages.jsx . . . 4
2.11 ProfileView.jsx . . . 4
2.12 Forum.jsx . . . 5
2.13 Inscriptions.jsx . . . 5
2.14 AjoutMessage.jsx . . . 5
2.15 MessageList.jsx . . . 5
2.16 Message.jsx . . . 6
2.17 ReplyList.jsx . . . 6
2.18 Reply.jsx . . . 6
2.19 AvatarList.jsx . . . 6
2.20 Avatar.jsx . . . 6

3 Conclusion **7**

4 ANNEXE. Premiers croquis et designs **8**

1 Graphe des dépendances entre les composants

Ci-dessous, un schéma représentant les relations entre les composants React de notre projet.

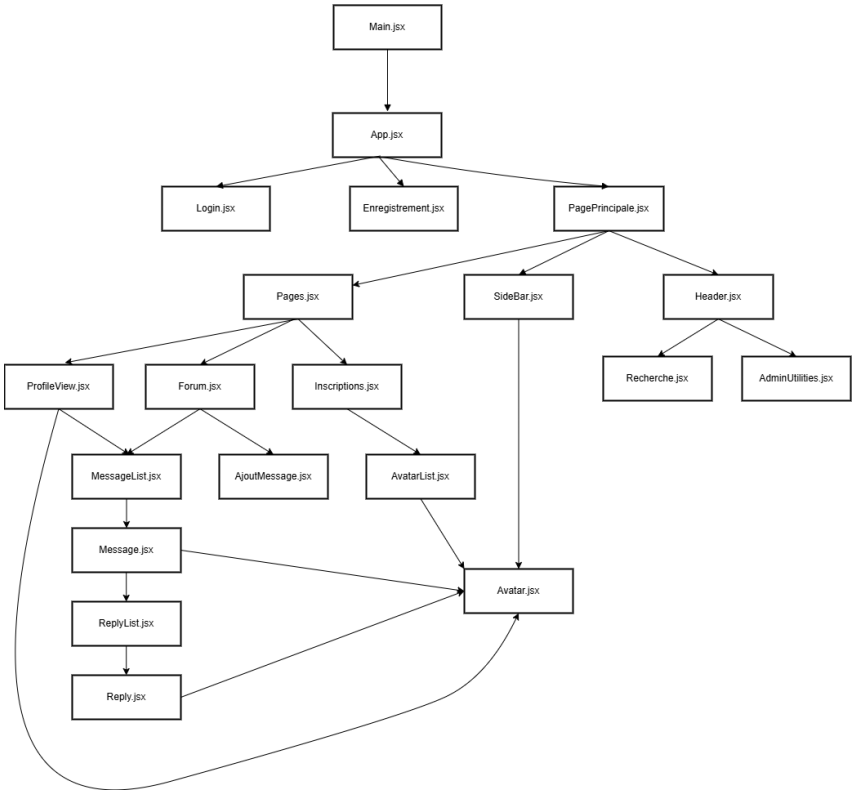

Figure 1: Graphe de dépendances entre les composants React

2 Présentation et fonctionnement des composants

2.1 main.jsx

- **Rôle** : Initialise l'application React et monte le composant principal `App` dans l'élément HTML `root`.
- **Props** : Aucune.
- **État** : Aucun état géré, il sert uniquement de point d'entrée à l'application.
- **Composants inclus** : `App`

2.2 App.jsx

- **Rôle** : Composant racine de l'application. Il gère la navigation entre les différentes pages et l'état global de connexion de l'utilisateur.
- **Props** : Aucune.
- **État** : `current_user`, qui contient les informations de l'utilisateur connecté. À partir de cet état, on décide quel page montrer (si current_user est null, cela signifie que l'utilisateur n'est pas connecté, donc on affiche le Login)

- **Composants inclus** : `Login`, `Enregistrement`, `PagePrincipale`

2.3 Login.jsx

- **Rôle** : Composant qui gère l'authentification des utilisateurs. Il vérifie d'abord si l'utilisateur existe dans la base de données et s'il a été accepté par un administrateur avant de permettre l'accès.
- **Props** : `set_current_user`, pour changer l'utilisateur actuel.
- **État** : Un état pour chaque champ que l'utilisateur doit remplir (login et mot de passe).
- **Composants inclus** : Aucun.

2.4 Enregistrement.jsx

- **Rôle** : Composant qui gère l'enregistrement des utilisateurs. Une fois l'inscription effectuée, l'utilisateur reste en attente de validation par un administrateur avant d'accéder au forum.
- **Props** : Aucune.
- **État** : Un état pour chaque champ que l'utilisateur doit remplir (nom, prénom, login, mot de passe, confirmation du mot de passe).
- **Composants inclus** : Aucun.

2.5 PagePrincipale.jsx

- **Rôle** : Page principale de notre application. Elle constitue l'interface principale où l'utilisateur peut naviguer entre les différentes sections, telles que le forum, le profil et d'autres fonctionnalités de l'application.
- **Props** : `current_user`, `set_current_user`
- **État** :
  - `current_page` : Pour stocker la page actuelle sélectionnée
  - `selected_user` : Stocke l'utilisateur dont on souhaite afficher le profil. Si `null`, le profil affiché est celui de l'utilisateur connecté (`current_user`).
  - `mode_forum` : Détermine si le forum est public ou privé (option réservée aux administrateurs). Ce paramètre permet de basculer entre les deux modes.
- **Composants inclus** : SideBar, Header, Pages.

2.6 SideBar.jsx

- **Rôle** : Le composant SideBar affiche l'avatar de l'utilisateur connecté, une brève description de son profil ainsi que des options pour consulter son profil et se déconnecter.
- **Props** :
  - `current_user`, utilisé pour afficher les informations de base de l'utilisateur connecté.
  - `set_current_user` pour mettre à null l'état si l'utilisateur veut se déconnecter
- **État** : Aucun.
- **Composants inclus** : Avatar.

2.7 Header.jsx

- **Rôle** : Le composant Header de notre forum inclut des champs de recherche permettant de filtrer les messages. De plus, s'il s'agit d'un administrateur connecté, des adminUtilities sont affichées pour gérer le contenu et les utilisateurs.
- **Props** : `current_user`, pour montrer `adminUtilities.jsx` si notre user est un admin, et `set_mode_forum` (qui sera utilisé par AdminUtilities)
- **État** : Aucun.
- **Composants inclus** : Recherche, AdminUtilities.

2.8 Recherche.jsx

- **Rôle** : Panel de recherche pour les messages. C'est possible de chercher des messages par mot-clé, par dates de début et de fin, et par auteur.
- **Props** : Aucun.
- **État** : Aucun.
- **Composants inclus** : Aucun.

2.9 AdminUtilities.jsx

- **Rôle** : Composant avec des functions de l'administrateur: Le forum privé et la liste d'inscriptions. Ce composant va être affiché **uniquement** si l'utilisateur actuel est admin.
- **Props** : `set_mode_forum`, permet de définir quel forum doit être affiché sur la page (public ou privé).
- **État** : Aucun.
- **Composants inclus** : Aucun.

2.10 Pages.jsx

- **Rôle** : Le composant Pages est utilisé pour afficher soit le forum, soit les profils des utilisateurs, soit la gestion des inscriptions (uniquement accessible aux administrateurs).
- **Props** : `current_user`, `selected_user`, `set_selected_user`, `mode_forum`
- **État** : Aucun.
- **Composants inclus** : ProfileView, Forum, Inscriptions.

2.11 ProfileView.jsx

- **Rôle** : Le composant ProfileView affiche le profil d'un utilisateur. Si le profil correspond à celui de l'utilisateur connecté, il peut modifier ses informations. De plus, une liste des messages de cet utilisateur est affichée.
- **Props** :
  - `selected_user` : Permet d'afficher les informations de l'utilisateur sélectionné.
  - `set_selected_user` : Permet de réinitialiser l'état à `null` lorsqu'on retourne sur le forum.

  - `current_user` : Permet de modifier les informations si `current_user` est égal à `selected_user`. Si l'utilisateur est administrateur, il a l'option de promouvoir un membre en administrateur.

- **État** : Aucun.
- **Composants inclus** : Avatar, MessageList.

2.12 Forum.jsx

- **Rôle** : Le composant Forum affiche une liste des messages publiés sur le forum. Il permet aussi d'ajouter des nouvelles discussions et de répondre aux messages.
- **Props** :
  - `current_user`
  - `mode_forum` : Détermine quel forum afficher en fonction de sa valeur (cet état sera modifié par `AdminUtilities`).
  - `set_selected_user`
- **État** : Aucun
- **Composants inclus** : MessageList.

2.13 Inscriptions.jsx

- **Rôle** : Page exclusive des utilisateurs pour gérer la validation des incriptions du forum.
- **Props** : Aucun.
- **État** : Aucun.
- **Composants inclus** : AvatarList.

2.14 AjoutMessage.jsx

- **Rôle** : Le composant AjoutMessage permet d'ajouter le titre et le contenu d'un message. Si le titre existe déjà, le message sera enregistré comme une réponse au message portant ce titre.
- **Props** : Aucun.
- **État** : Un état pour le titre et un état pour le contenu
- **Composants inclus** : Aucun.

2.15 MessageList.jsx

- **Rôle** : Ce composant affiche la liste des messages d'un sujet dans le forum. Il récupère et organise les messages dans l'ordre chronologique
- **Props** : `current_user`, `set_selected_user`
- **État** : Aucun.
- **Composants inclus** : Message.

2.16 Message.jsx

- **Rôle** : Représente un message individuel dans le forum. Il affiche le contenu du message, l'auteur, la date et permet d'afficher les réponses associées.
- **Props** :
  `current_user` (Pour supprimer les messages, l'ID doit correspondre à celui de l'utilisateur ou à celui d'un administrateur).`set_selected_user` : Permet de sélectionner l'utilisateur ayant écrit un message afin d'afficher son profil.
- **État** : Aucun.
- **Composants inclus** : Avatar, ReplyList.

2.17 ReplyList.jsx

- **Rôle** : Ce composant affiche la liste des réponses associées à un message. Il permet de structurer les discussions et les échanges entre utilisateurs.
- **Props** : `current_user`, `set_selected_user`
- **État** : Aucun.
- **Composants inclus** : Reply.

2.18 Reply.jsx

- **Rôle** : Représente une réponse individuelle à un message. Il affiche le contenu de la réponse ainsi que l'avatar de l'utilisateur qui l'a postée.
- **Props** :
  - `current_user` (Pour supprimer les Replys, l'ID doit correspondre à celui de l'utilisateur ou à celui d'un administrateur).
  - `set_selected_user` : Permet de sélectionner l'utilisateur ayant écrit une reply afin d'afficher son profil.
- **État** : Aucun.
- **Composants inclus** : Avatar.

2.19 AvatarList.jsx

- **Rôle** : Affiche une liste d'avatars des utilisateurs. Il est utilisé pour représenter visuellement les membres qui ont demandé de joindre le forum
- **Props** : Aucun.
- **État** : Aucun.
- **Composants inclus** : Avatar.

2.20 Avatar.jsx

- **Rôle** : Ce composant favorise la modularisation. Il affiche une image de profil accompagnée du prénom d'un membre. Il est largement utilisé dans plusieurs composants de l'application.
- **Props** : Aucun.
- **État** : Aucun.
- **Composants inclus** : Aucun.

3 Conclusion

Ce document détaille les relations entre les composants React du notre projet en intégrant une gestion dynamique des forums via un seul composant. Cette approche simplifie la navigation et offre une meilleure expérience utilisateur tout en respectant la distinction entre membres et administrateurs.

Cependant, il reste encore plusieurs états et props à définir et préciser. Nous avons présenté comment gérer le type de page affiché au démarrage, ainsi que la gestion de la connexion et déconnexion à l'aide des états `current_user` et `set_current_user`. De plus, nous avons décrit la manière dont le passage entre le forum public et privé sera réalisé à l'aide de l'attribut `mode_forum`, qui pourra être modifié par les options administratives (`AdminUtilities`).

Enfin, bien que la gestion de la visualisation des profils d'autres utilisateurs ne soit pas encore totalement finalisée, nous avons essayé de structurer cette fonctionnalité. Le projet est encore en phase de développement et reste sujet à des modifications, mais il est clair qu'un état `selected_user` sera nécessaire. Il permettra d'accéder aux profils depuis les messages et réponses du forum, ce qui nécessite de transmettre `set_selected_user` à travers toute la hiérarchie des composants jusqu'à `Reply`. De plus, cet état sera utile pour identifier si l'utilisateur peut modifier ses informations personnelles : si `selected_user` et `current_user` correspondent, alors l'utilisateur est sur son propre profil.

Ce travail de structuration des composants et de définition des états constitue une première étape vers la mise en place d'une architecture claire et évolutive pour l'application.

4 ANNEXE. Premiers croquis et designs

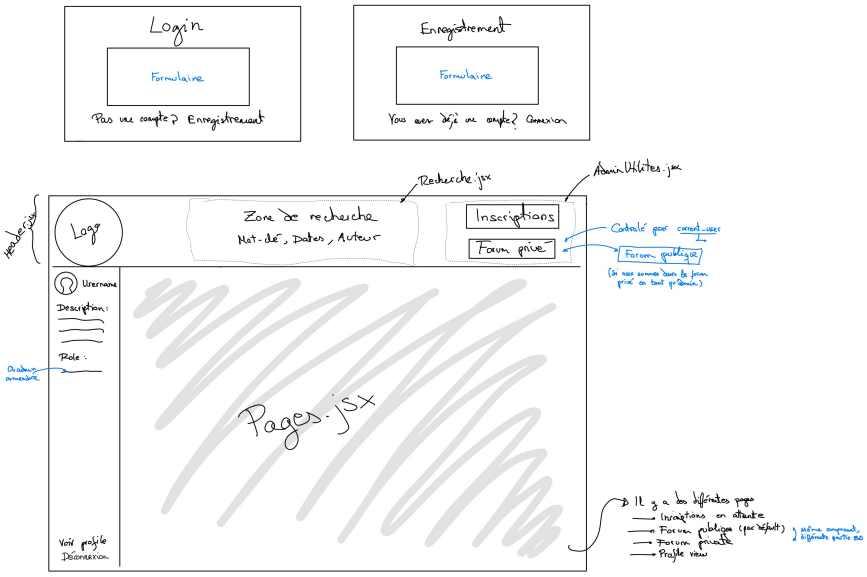

Figure 2: Login, Enregistrement et Page Principale de la web

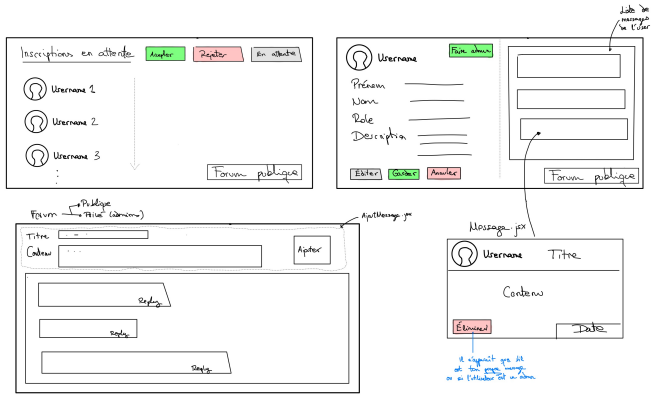

Figure 3: Types de pages