Organiz-Asso

Axel CERESA - 21103270
Daba KONE -

1 Rapport

1.1 Répartition du travail

Notre binôme est un petit peu particulier. Daba était dans un autre groupe puis dans une autre classe avant de me rejoindre (Axel). A ce moment, une grande partie du travail avait déjà été réalisée.

- Axel : Tout le backend et le frontend.
- Daba : Tests Postman de toutes les requêtes.

1.2 Difficultés

La principale difficulté que nous avons rencontrée concerne l'utilisation de express-session pour générer la connexion et la déconnexion d'un utilisateur. L'utilisation des cookies de session n'était pas évidente à comprendre au départ.

L'autre difficulté était de comprendre comment bien utiliser les requêtes `axios` et les `useEffect` pour actualiser en permanence les messages des forums et que la page reste à jour sans avoir besoin de recharger la page.

1.3 Fonctionnalités

L'ensemble des fonctionnalités demandées dans le cahier des charges ont été réalisées et d'autres supplémentaires ont été ajoutées, notamment :

1. La possibilité de créer/supprimer un forum et de modifier son nom et ses restrictions.
2. La possibilité de modifier un message.
3. Options de modérations pour les administrateurs qui peuvent supprimer n'importe quel message posté sur le site.

Si nous avions eu le temps, nous aurions aimé implémenter d'autres fonctionnalités tels que l'ajout d'une photo de profil, la possibilité créer une biographie pour son profil ou même la possibilité de like des messages et de les ajouter en favoris. De plus, l'ajout d'un genre de calendrier / liste d'évènements (création et suppression) aurait pu être intéressant.

1.4 Choix d'implémentation

Concernant les choix d'implémentation, nous avons utilisé `react-router-dom` pour gérer les routes et les urls du site côté client.

Pour la base de données, notre choix de créer une collection différente pour les commentaires au lieu de les intégrer directement dans la description d'un message n'est peut-être pas la meilleur solution car cela multiplie le nombre de requête au serveur, ce qui n'est pas très efficace.

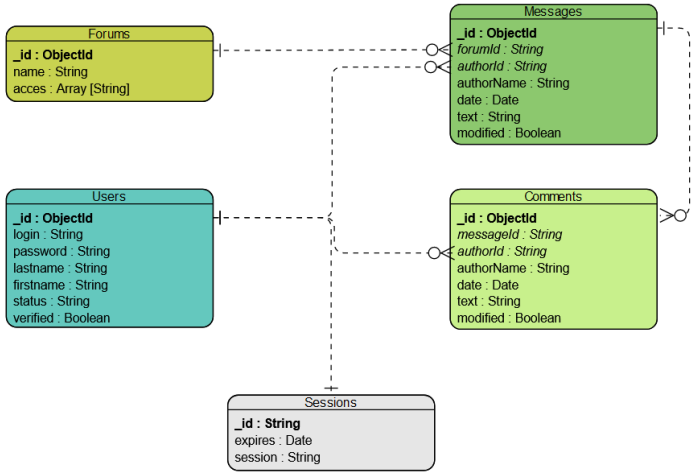

Figure 1: Base de données

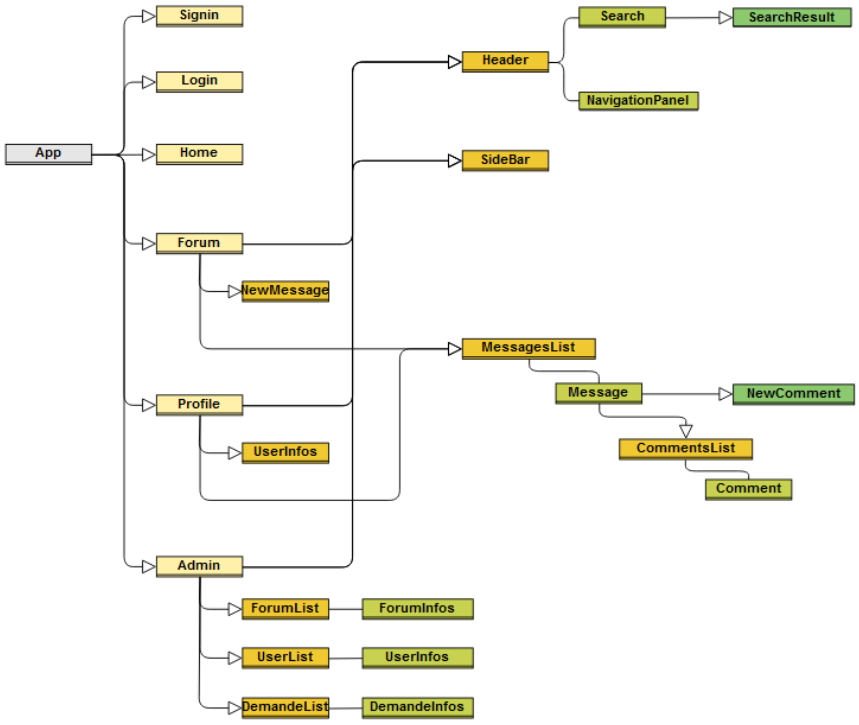

Figure 2: Architecture des composants react

2 Installation

Une fois les fichiers téléchargés, dans un terminal faire un `npm install` dans le répertoire `/server` et dans le répertoire `/client` pour installer les dépendances nécessaires.

Pour relier la base de données au backend, modifier côté serveur dans le fichier `/config/db.js` la ligne 2 `const url = "url_mongodb";` par le lien de votre base de données MongoDB. Créer une base de données Organiz-Asso et y importer les collections sous format .json présentes dans le répertoire `Collections`.

3 BACKEND

Architecture des fichiers

Dans le dossier `/server` se trouve tous les fichiers réalisant le backend de notre application dont le point d'entrée est `index.js`. Ensuite, `app.js` lance le server et fait appel à `api.js` qui gère toutes les routes de notre backend.

1. `/config` : Contient le fichier `db.js` qui définit les fonctions utilisées dans notre application pour se connecter et se déconnecter de la base de données. Elles sont appelées dans le fichier `db.middleware.js`.

2. `/controllers` : Contient les fichiers définissant toutes les actions possibles sur notre base de données. Elles sont appelées en callback des différentes routes de notre application. Chaque fichier correspond aux fonctions liées à une entité spécifique. Ces fonctions font appel aux méthodes des classes définies dans les fichiers de `/entities`.

3. `/entities` : Contient les fichiers définissant les classes de chaque entités de notre application (user, forum, message, comment). Les méthodes définies dans ces classes effectuent les requêtes auprès de la base de données MongoDB passée en paramètres.

4. `/middleware` : Contient les middlewares `checkUser` et `requireAuth` utilisés à la racine de notre serveur pour sécuriser les requêtes et restreindre leur accès uniquement à un utilisateur connecté (dont une session express a été créée). A chaque requête, les middlewares `connectDBMiddleware` et `closeDBMiddleware` sont appelés.

5. `/routes` : Contient les fichiers définissant les routes des requêtes serveur de notre application pour chaque famille de services.

4 FRONTEND

Architecture des fichiers

Dans le dossier `/client` se trouve tous les fichiers réalisant le frontend de notre application. Le point d'entrée est `index.js`, qui fait appel au composant `<App/>` qui gère le routage de notre application avec `react-router-dom`.

1. `/public` : Contient le fichier `index.html` dans lequel est généré le render des composants react de l'application.
2. `/assets/img` : Contient les images/logo temporairement utilisés dans l'application.
3. `/components` : Contient les fichiers définissant les différents composants appelés dans les pages de notre application react. Ces composants disposent chacun de leur fichier CSS correspondant. `AppContext.js` est le composant englobant toute notre application et permet de stocker dans une variable accessible dans tout nos composants les informations de session de l'utilisateur connecté.
4. `/pages` : Contient les fichiers définissant les pages de l'application, chacune d'elles fait appel aux composants définis dans le répertoire `/components`.

5 BASE DE DONNÉES

La base de données est définie grâce à MongoDB.

Définition des collections

Les champs _id de type `ObjectId` sont générés automatiquement lors de l'ajout d'un document dans la base de données.

users

```
{
   "_id" : ObjectId(),
   "login" : String,
   "password" : String,
   "lastname" : String,
   "firsname" : String,
   "status" : String,
   "date" : Date(),
   "verified" : Boolean
}
```

Le password est crypté à l'aide de la bibliothèque `bcrypt` avant l'ajout dans la base de données. Le status d'un user est initialisé de base à "member".

forums

```
{
   "_id" : ObjectId(),
   "name" : String,
   "acces" : [ String ]
}
```

Le champs acces définit les status de user pouvant voir et écrire dans le forum.

messages

```
{
   "_id" : ObjectId(),
   "forumId" : String,
   "userId" : String,
   "userName" : String,
   "date" : Date(),
   "text" : String,
   "modified" : Boolean
}
```

comments

```
{
   "_id" : ObjectId(),
   "messageId" : String,
   "authorId" : String,
   "authorName" : String,
   "date" : Date(),
   "text" : String,
   "modified" : Boolean
}
```

sessions

```
{
   "_id" : String,
   "expires" : Date(),
   "session" : String
}
```

Les documents de la collection sessions sont générés automatiquement par express-session et contiennent l'id de la session ainsi que le cookie associé.

6 Lancement

Pour lancer le serveur, se placer dans le dossier `/server` et exécuter la commande `npm start` dans un terminal. Cela lancera le serveur à l'adresse `http://localhost:4000` par défaut.

Pour lancer le client, se placer dans le dossier `/client` et exécuter la commande `npm start` dans un terminal. Cela compilera et lancera le client à l'adresse `http://localhost:3000` par défaut (le navigateur s'ouvre automatiquement).