Fiche : Structure d'une Application Web avec Backend et Frontend

Objectif

Cette fiche présente une architecture d'application web bien structurée qui intègre un backend (Node.js) et un frontend (React). Elle explique la séparation des préoccupations, l'organisation des fichiers et l'utilisation de Docker pour le déploiement, ainsi qu'une gestion des rôles avec ACL.

Structure de l'Application

```
my-app/
│
├── auth/                        # Dossier pour l'authentification
│   ├── Dockerfile               # Dockerfile pour le service d'authentification
│   ├── server.js                # Point d'entrée pour le service d'authentification
│   ├── controllers/             # Contrôleurs pour gérer les requêtes
d'authentification
│   ├── routes/                  # Routes pour l'authentification
│   └── models/                  # Modèles de base de données (ex. User)
│
├── authorization/               # Dossier pour l'autorisation
│   ├── Dockerfile               # Dockerfile pour le service d'autorisation
│   ├── server.js                # Point d'entrée pour le service d'autorisation
│   ├── middleware/              # Middleware pour gérer les autorisations
│   ├── policies/                # Politiques d'autorisation
│   └── acl.js                   # Fichier de configuration des ACL
│
├── main/                        # Dossier principal de l'application
│   ├── Dockerfile               # Dockerfile pour le service principal
│   ├── server.js                # Point d'entrée pour le service principal
│   ├── config/                  # Fichiers de configuration (ex. DB config)
│   ├── routes/                  # Routes de l'application principale
│   ├── controllers/             # Contrôleurs pour gérer les requêtes principales
│   └── models/                  # Modèles de base de données
│
├── frontend/                    # Dossier pour le frontend React
│   ├── Dockerfile               # Dockerfile pour le projet React
│   ├── package.json             # Fichier de configuration npm pour le frontend
│   ├── public/                  # Dossier public pour les fichiers statiques
│   ├── src/                     # Dossier source pour le code React
│   └── README.md                # Documentation spécifique au frontend
│
├── docker-compose.yml            # Fichier de configuration Docker Compose
│
└── README.md                    # Documentation du projet
```

Explications des Dossiers

1. auth/

- **Objectif** : Gérer l'authentification des utilisateurs.
- **Contenu** :
  - **Dockerfile** : Contient les instructions pour construire l'image Docker du service d'authentification.
  - **server.js** : Point d'entrée de l'application, initialisant le serveur et les routes.
  - **controllers/** : Contient la logique pour gérer les requêtes d'authentification (ex. inscription, connexion).
  - **routes/** : Définit les endpoints pour l'authentification.
  - **models/** : Contient les modèles de données (ex. User) pour interagir avec la base de données.

2. authorization/

- **Objectif** : Gérer les autorisations des utilisateurs.
- **Contenu** :
  - **Dockerfile** : Instructions pour construire l'image Docker du service d'autorisation.
  - **server.js** : Point d'entrée pour le service d'autorisation.
  - **middleware/** : Middleware pour gérer les autorisations (ex. vérifier les rôles des utilisateurs).
  - **policies/** : Contient les politiques d'autorisation, définissant qui peut accéder à quoi.
  - **acl.js** : Configuration des Listes de Contrôle d'Accès (ACL) pour gérer les rôles et permissions.

Exemple de Code pour la Gestion des ACL

Voici un exemple simple de gestion des ACL dans un fichier `acl.js` :

```
// acl.js

// Définition des rôles et permissions
const acl = {
  admin: {
    can: ['create', 'read', 'update', 'delete'], // Admin peut tout faire
  },
  user: {
    can: ['read'], // Utilisateur peut seulement lire
  },
};

// Fonction pour vérifier si un utilisateur a accès à une certaine permission
const canAccess = (role, action) => {
  return acl[role]?.can.includes(action);
};

// Exporter la fonction pour utilisation dans d'autres parties de l'application
module.exports = { canAccess };
```

Dans ce code :

- On définit des rôles (`admin`, `user`) et leurs permissions.
- La fonction `canAccess` permet de vérifier si un rôle spécifique a le droit d'effectuer une action donnée.

3. **main/**

- **Objectif** : Gérer la logique principale de l'application.
- **Contenu** :
  - **Dockerfile** : Instructions pour le service principal.
  - **server.js** : Point d'entrée pour le service principal.
  - **config/** : Fichiers de configuration, comme les paramètres de connexion à la base de données.
  - **routes/** : Routes pour les fonctionnalités principales de l'application.
  - **controllers/** : Logique pour gérer les requêtes des utilisateurs.
  - **models/** : Modèles de données pour interagir avec la base de données.

4. **frontend/**

- **Objectif** : Gérer le code de l'application frontend en React.
- **Contenu** :
  - **Dockerfile** : Instructions pour construire l'image Docker du projet React.
  - **package.json** : Fichier de configuration npm pour gérer les dépendances et les scripts.
  - **public/** : Contient les fichiers statiques qui seront servis (ex. index.html).
  - **src/** : Dossier source pour le code React, contenant les composants, styles, etc.
  - **README.md** : Documentation spécifique au frontend, expliquant comment démarrer et développer.

5. **docker-compose.yml**

- **Objectif** : Configurer les services de l'application dans Docker.
- **Contenu** : Définit les services, les réseaux et les volumes nécessaires pour faire fonctionner l'application ensemble.

Conclusion

Cette structure d'application permet de maintenir une séparation claire des responsabilités entre les différentes parties de l'application. L'intégration des ACL permet de gérer les permissions de manière efficace et sécurisée, garantissant que seuls les utilisateurs autorisés peuvent accéder à certaines fonctionnalités. Chaque service peut être développé et déployé indépendamment, facilitant ainsi le travail en équipe et la gestion des versions. De plus, l'utilisation de Docker permet de garantir que l'application fonctionne de manière cohérente sur différentes environnements.