D'accord, je vais reprendre la description de la chronologie de la requête en intégrant les étapes de hachage et de routage. Voici une description détaillée de la requête dans une application frontend-centrée déployée sur AWS avec Docker, utilisant Nginx, Express.js, et Node.js, où chaque utilisateur a accès à une application spécifique parmi plusieurs disponibles, et les paramètres spécifiques à chaque utilisateur sont chargés dynamiquement.

Structure de l'Application

1. **Frontend (React.js)** :
    - Application React.js déployée dans un conteneur Docker.
    - Code de base identique pour tous les utilisateurs.
    - Composants et modules chargés dynamiquement en fonction des autorisations de l'utilisateur.
2. **Backend (Express.js/Node.js)** :
    - Serveur Express.js/Node.js déployé dans un conteneur Docker.
    - Gère l'authentification, les autorisations, et les requêtes API.
3. **Nginx** :
    - Serveur Nginx déployé dans un conteneur Docker, agissant comme reverse proxy.
4. **Base de Données** :
    - Base de données (par exemple, MongoDB) déployée dans un conteneur Docker.
    - Stocke les informations d'identification des utilisateurs, les autorisations, et les paramètres spécifiques à chaque utilisateur.

Chronologie de la Requête

1. **Connexion du Client** :
    - Le client ouvre un navigateur web et entre l'URL du nom de domaine lié à l'application (par exemple, `https://www.example.com`).
2. **Résolution DNS** :
    - Le nom de domaine est résolu en une adresse IP par le serveur DNS.
    - Le navigateur envoie une requête HTTP/HTTPS à l'adresse IP du serveur AWS.
3. **Nginx (Reverse Proxy)** :
    - Nginx écoute les requêtes HTTP/HTTPS sur le port 80/443.
    - Nginx reçoit la requête et redirige les requêtes vers le frontend ou le backend en fonction de l'URI.
4. **Requête Initiale (Frontend)** :
    - La requête initiale est une requête GET pour charger l'application React.js.
    - Nginx redirige cette requête vers le serveur frontend (React.js) :

```
location / {
    proxy_pass http://frontend:3000;
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;
}
```

5. **Chargement de l'Application React.js** :
   - Le serveur frontend (React.js) répond avec les fichiers HTML, CSS, et JavaScript nécessaires pour charger l'application.
   - Le navigateur du client télécharge et exécute ces fichiers, affichant l'interface utilisateur de l'application.

6. **Affichage de la Page de Connexion** :
   - L'application React.js détecte que l'utilisateur n'est pas authentifié et affiche la page de connexion.

7. **Soumission du Formulaire de Connexion** :
   - L'utilisateur entre ses informations d'identification (par exemple, nom d'utilisateur et mot de passe) et soumet le formulaire de connexion.
   - Une requête POST est envoyée au serveur backend pour authentification :

```
axios.post('/api/login', { username, password })
    .then(response => {
        // Gérer la réponse de connexion
    })
    .catch(error => {
        // Gérer les erreurs de connexion
    });
```

8. **Nginx (Redirection vers le Backend)** :
   - Nginx reçoit la requête POST et la redirige vers le serveur backend (Express.js) :

```
location /api {
    proxy_pass http://backend:5000;
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;
}
```

9. **Authentification (Express.js)** :

- Le serveur backend (Express.js) reçoit la requête POST et traite les informations d'identification.
- Le serveur vérifie les informations d'identification contre la base de données (par exemple, MongoDB).
- Les mots de passe sont hachés avant d'être stockés dans la base de données pour des raisons de sécurité.
- Si les informations d'identification sont correctes, le serveur génère un token JWT (JSON Web Token) et le renvoie au client.

10. **Réponse de Connexion** :
    - Le serveur backend envoie une réponse avec le token JWT :

```
res.json({ token });
```

11. **Stockage du Token JWT** :
    - Le client stocke le token JWT dans le stockage local (LocalStorage) ou dans un cookie sécurisé.
12. **Accès à l'Application** :
    - Le client inclut le token JWT dans les en-têtes des requêtes suivantes pour accéder aux ressources protégées :

```
axios.get('/api/user-applications', {
    headers: {
        'Authorization': `Bearer ${token}`
    }
})
.then(response => {
    // Gérer la réponse des applications disponibles
})
.catch(error => {
    // Gérer les erreurs d'accès
});
```

13. **Nginx (Redirection vers le Backend)** :
    - Nginx reçoit les requêtes avec le token JWT et les redirige vers le serveur backend (Express.js).
14. **Vérification du Token JWT (Express.js)** :
    - Le serveur backend (Express.js) vérifie le token JWT pour s'assurer qu'il est valide et non expiré.
    - Si le token est valide, le serveur traite la requête et renvoie les données demandées.
15. **Réponse des Applications Disponibles** :
    - Le serveur backend envoie une réponse avec la liste des applications disponibles pour l'utilisateur.
    - Le client reçoit les données et affiche la liste des applications dans l'application React.js.

16. **Sélection d'une Application** :
    - L'utilisateur sélectionne une application parmi celles disponibles.
    - Une requête GET est envoyée au serveur backend pour charger les paramètres et les projets de l'utilisateur pour l'application sélectionnée :

```
axios.get(`/api/user-applications/${applicationId}/params`, {
    headers: {
        'Authorization': `Bearer ${token}`
    }
})
.then(response => {
    // Gérer la réponse des projets disponibles
})
.catch(error => {
    // Gérer les erreurs d'accès
});
```

17. **Chargement des Paramètres et Projets** :
    - Le serveur backend récupère les paramètres et les projets de l'utilisateur depuis la base de données et le dossier personnel de l'utilisateur.
    - Le serveur backend envoie une réponse avec les paramètres et les projets de l'utilisateur.

18. **Affichage de l'Application Personnalisée** :
    - Le client reçoit les données et affiche l'application personnalisée avec les paramètres et les projets de l'utilisateur.
    - L'utilisateur peut maintenant travailler sur les différentes versions de l'application en fonction du projet sélectionné.

Hachage et Routage

- **Hachage** :
    - Les mots de passe des utilisateurs sont hachés avant d'être stockés dans la base de données pour des raisons de sécurité.
    - Lors de l'authentification, le mot de passe fourni par l'utilisateur est haché et comparé au hachage stocké dans la base de données.

- **Routage** :
    - Nginx agit comme un reverse proxy et redirige les requêtes vers les serveurs frontend et backend appropriés en fonction de l'URI.
    - Express.js gère le routage des requêtes API et les redirige vers les contrôleurs appropriés pour traitement.

Exemple de Code pour le Hachage des Mots de Passe

controllers/auth.js

```
const bcrypt = require('bcrypt');
const jwt = require('jsonwebtoken');
const User = require('../models/User');

const login = async (req, res) => {
  const { username, password } = req.body;


 const

 user = await User.findOne({ username });
  if (!user) {
    return res.status(401).json({ message: 'Invalid credentials' });
  }

  const isMatch = await bcrypt.compare(password, user.password);
  if (!isMatch) {
    return res.status(401).json({ message: 'Invalid credentials' });
  }

  const token = jwt.sign({ id: user._id, username: user

.username

 }, 'secret_key', { expiresIn: '1h' });
  res.json({ token });
};

module.exports = { login };
```

models/User.js

```
const mongoose = require('mongoose');
const bcrypt = require('bcrypt');

const userSchema = new mongoose.Schema({
  username: { type: String, required: true, unique: true },
  password: { type: String, required: true },
});

userSchema.pre('save', async function (next) {
  if (!this.isModified('password')) {
    return next();
  }
  const salt = await bcrypt.genSalt(10);
  this.password = await bcrypt.hash(this.password, salt);
  next();
});
```

```
const User = mongoose.model('User', userSchema);
module.exports = User;
```

Conclusion

Dans cette architecture, Nginx agit comme un reverse proxy qui redirige les requêtes HTTP/HTTPS vers les serveurs frontend et backend appropriés. L'application React.js gère l'interface utilisateur et les interactions avec l'utilisateur, tandis que le serveur backend (Express.js) gère l'authentification, les requêtes API, et la gestion des paramètres et des projets de l'utilisateur. Le hachage des mots de passe est effectué avant de les stocker dans la base de données pour des raisons de sécurité. Le routage des requêtes est géré par Nginx et Express.js pour assurer une expérience utilisateur fluide et sécurisée, tout en offrant une application personnalisée en fonction des projets de l'utilisateur.

Similar code found with 1 license type