Bien sûr, voici un commentaire détaillé du code pour vous aider à comprendre ce qui se passe :

```
// Importation des modules nécessaires
const express = require('express'); // Framework web pour Node.js
const bcrypt = require('bcrypt'); // Bibliothèque pour le hachage des mots de
passe
const jwt = require('jsonwebtoken'); // Bibliothèque pour la création et la
vérification des tokens JWT
const User = require('../models/User'); // Modèle utilisateur pour interagir avec
la base de données MongoDB
const validator = require('validator'); // Bibliothèque pour valider et nettoyer
les entrées utilisateur

// Création d'un routeur Express
const router = express.Router();

// Route pour l'inscription des utilisateurs
router.post('/register', async (req, res) => {
  // Extraction des données de la requête
  const { email, password, role } = req.body;

  // Valider et nettoyer les entrées utilisateur
  if (!validator.isEmail(email)) {
    return res.status(400).send('Invalid email format'); // Vérifie si l'email est
valide
  }
  if (!validator.isStrongPassword(password)) {
    return res.status(400).send('Password is not strong enough'); // Vérifie si le
mot de passe est suffisamment fort
  }
  if (role && !['user', 'admin'].includes(role)) {
    return res.status(400).send('Invalid role'); // Vérifie si le rôle est valide
  }

  try {
    // Hachage du mot de passe avec un facteur de coût de 10
    const hashedPassword = await bcrypt.hash(password, 10);
    // Création d'un nouvel utilisateur avec l'email, le mot de passe haché et le
rôle
    const user = new User({ email: validator.escape(email), password:
hashedPassword, role });
    // Sauvegarde de l'utilisateur dans la base de données
    await user.save();
    // Réponse avec un statut 201 (créé) si l'inscription est réussie
    res.status(201).send('User registered');
  } catch (error) {
    // Réponse avec un statut 400 (mauvaise requête) en cas d'erreur
    res.status(400).send('Error registering user');
  }
});

// Route pour la connexion des utilisateurs
```

```
router.post('/login', async (req, res) => {
  // Extraction des données de la requête
  const { email, password } = req.body;

  // Valider et nettoyer les entrées utilisateur
  if (!validator.isEmail(email)) {
    return res.status(400).send('Invalid email format'); // Vérifie si l'email est
valide
  }

  try {
    // Recherche de l'utilisateur par email dans la base de données
    const user = await User.findOne({ email: validator.escape(email) });
    if (!user) {
      // Réponse avec un statut 400 (mauvaise requête) si l'utilisateur n'est pas
trouvé
      return res.status(400).send('User not found');
    }
    // Comparaison du mot de passe fourni avec le mot de passe haché stocké
    const isMatch = await bcrypt.compare(password, user.password);
    if (!isMatch) {
      // Réponse avec un statut 400 (mauvaise requête) si les mots de passe ne
correspondent pas
      return res.status(400).send('Invalid credentials');
    }
    // Création d'un token JWT avec l'ID de l'utilisateur et son rôle, expirant
dans 1 heure
    const token = jwt.sign({ userId: user._id, role: user.role },
process.env.JWT_SECRET, { expiresIn: '1h' });
    // Réponse avec le token JWT en cas de succès
    res.json({ token });
  } catch (error) {
    // Réponse avec un statut 400 (mauvaise requête) en cas d'erreur
    res.status(400).send('Error logging in');
  }
});

// Exportation du routeur pour l'utiliser dans d'autres parties de l'application
module.exports = router;
```

Explication détaillée :

1. **Importation des modules** :

express

: Utilisé pour créer des routes et gérer les requêtes HTTP.

bcrypt

: Utilisé pour hacher les mots de passe avant de les stocker dans la base de données.

- `jsonwebtoken` : Utilisé pour créer et vérifier les tokens JWT pour l'authentification.
-

User

: Modèle utilisateur pour interagir avec la collection d'utilisateurs dans MongoDB.

- `validator` : Utilisé pour valider et nettoyer les entrées utilisateur afin de prévenir les injections et autres attaques.

2. **Création du routeur** :

express.Router()

: Crée un nouvel objet routeur pour définir les routes d'authentification.

3. **Route d'inscription (`/register`)** :

router.post('/register', async (req, res) => { ... })

: Définit une route POST pour l'inscription des utilisateurs.

- `const { email, password, role } = req.body` : Extrait les données de la requête.
- `if (!validator.isEmail(email)) { ... }` : Vérifie si l'email est valide.
- `if (!validator.isStrongPassword(password)) { ... }` : Vérifie si le mot de passe est suffisamment fort.
- `if (role && !['user', 'admin'].includes(role)) { ... }` : Vérifie si le rôle est valide.
-

const hashedPassword = await bcrypt.hash(password, 10)

: Hache le mot de passe avec un facteur de coût de 10.

const user = new User({ email: validator.escape(email), password: hashedPassword, role })

: Crée un nouvel utilisateur avec l'email, le mot de passe haché et le rôle.

await user.save()

: Sauvegarde l'utilisateur dans la base de données.

res.status(201).send('User registered')

: Répond avec un statut 201 si l'inscription est réussie.

res.status(400).send('Error registering user')

: Répond avec un statut 400 en cas d'erreur.

4. **Route de connexion (`/login`)** :

router.post('/login', async (req, res) => { ... })

: Définit une route POST pour la connexion des utilisateurs.

- `const { email, password } = req.body` : Extrait les données de la requête.
- `if (!validator.isEmail(email)) { ... }` : Vérifie si l'email est valide.
-

const user = await User.findOne({ email: validator.escape(email) })

: Recherche l'utilisateur par email dans la base de données.

- `if (!user) { return res.status(400).send('User not found') }` : Vérifie si l'utilisateur existe.
-

const isMatch = await bcrypt.compare(password, user.password)

: Compare le mot de passe fourni avec le mot de passe haché stocké.

- `if (!isMatch) { return res.status(400).send('Invalid credentials') }` : Vérifie si les mots de passe correspondent.
-

const token = jwt.sign({ userId: user._id, role: user.role }, process.env.JWT_SECRET, { expiresIn: '1h' })

: Crée un token JWT avec l'ID de l'utilisateur et son rôle, expirant dans 1 heure.

res.json({ token })

: Répond avec le token JWT en cas de succès.

res.status(400).send('Error logging in')

: Répond avec un statut 400 en cas d'erreur.

5. **Exportation du routeur** :

module.exports = router

: Exporte le routeur pour qu'il puisse être utilisé dans d'autres parties de l'application, comme dans le fichier `server.js`.

Ce code gère l'inscription et la connexion des utilisateurs en utilisant des mots de passe hachés et des tokens JWT pour l'authentification, tout en validant et en nettoyant les entrées utilisateur pour prévenir les vulnérabilités.