Documentation Technique de l'Application

Introduction

Cette documentation explique les fonctionnalités, les technologies utilisées, et les étapes nécessaires pour installer et exécuter l'application. L'application est une solution de gestion des utilisateurs, permettant l'inscription et la connexion avec des rôles spécifiques.

Fonctionnalités

1. **Inscription des utilisateurs**
    - Les utilisateurs peuvent s'inscrire en fournissant un nom d'utilisateur, un mot de passe, et un rôle.
    - Les rôles disponibles sont `admin` et `user`.
    - Les données d'inscription sont stockées dans une base de données MongoDB.
2. **Connexion des utilisateurs**
    - Les utilisateurs peuvent se connecter avec leur nom d'utilisateur et leur mot de passe.
    - Une session est créée pour l'utilisateur connecté, et un jeton JWT est émis pour l'authentification.
    - Les informations de session incluent le nom d'utilisateur et le rôle.
3. **Gestion des sessions**
    - Utilisation de `express-session` pour gérer les sessions utilisateur côté serveur.
    - Les sessions sont stockées en mémoire et peuvent être configurées pour être persistantes.
4. **Déconnexion des utilisateurs**
    - Les utilisateurs peuvent se déconnecter, ce qui détruit leur session actuelle.

Technologies Utilisées

Backend

Node.js

- **Pourquoi ?** : Node.js permet l'exécution du code JavaScript côté serveur. C'est une plateforme légère et efficace grâce à son modèle événementiel non-bloquant, ce qui en fait un choix idéal pour les applications en temps réel et intensives en E/S.

Express.js

- **Pourquoi ?** : Express.js est un framework web minimaliste et flexible pour Node.js, qui simplifie la création d'API REST. Il offre une multitude de fonctionnalités puissantes, telles que la gestion des routes et des middlewares, tout en étant très léger.

MongoDB avec Mongoose

- **Pourquoi ?** : MongoDB est une base de données NoSQL qui stocke les données dans des documents JSON flexibles. Elle est idéale pour les applications nécessitant une haute scalabilité et une grande flexibilité des schémas de données. Mongoose est une bibliothèque ODM (Object Data Modeling) qui facilite l'interaction avec MongoDB en offrant une API plus structurée et des schémas de données robustes.

bcrypt.js

- **Pourquoi ?** : bcrypt.js est utilisé pour le hachage des mots de passe avant de les stocker dans la base de données. Il utilise un algorithme de hachage sécurisé, garantissant que les mots de passe ne sont jamais stockés en clair et augmentant ainsi la sécurité de l'application.

JSON Web Tokens (JWT)

- **Pourquoi ?** : Les JWT sont utilisés pour l'authentification sécurisée des utilisateurs. Ils permettent de transmettre des informations entre le client et le serveur sous forme de token signé, garantissant ainsi que les informations sont sécurisées et non modifiables sans la clé secrète.

express-session

- **Pourquoi ?** : express-session est utilisé pour la gestion des sessions utilisateur. Il permet de stocker les informations de session sur le serveur, facilitant ainsi la gestion des utilisateurs connectés et leur authentification sur plusieurs requêtes HTTP.

PM2

- Gestionnaire de processus Node.js utilisé pour le déploiement en production. Permet de démarrer, arrêter, surveiller et redémarrer les

processus Node.js, assurant ainsi la disponibilité et la stabilité de l'application en production.

Frontend

Flutter

- **flutter_toastr** et **in_app_notification** sont utilisés pour fournir des notifications à l'utilisateur dans l'application Flutter, telles que des confirmations de connexion réussie, des erreurs d'inscription, etc.

- **http** est utilisé pour effectuer des requêtes HTTP vers le backend Node.js, par exemple pour les opérations d'authentification (connexion, inscription) et pour récupérer des données depuis le serveur.

- **socket_io_client** facilite les communications en temps réel entre le frontend Flutter et le backend Node.js via des connexions WebSocket, permettant des fonctionnalités comme la mise à jour en temps réel des positions, des alertes ou d'autres données dynamiques.

Structure des Répertoires

```
backend/
│
├── middleware/
│   ├── auth.js
│   ├── authSocket.js
│   ├── isAdmin.js
│   └── userMiddleware.js
│
├── models/
│   ├── Position.js
│   └── User.js
│
├── routes/
│   ├── admin.js
│   ├── auth.js
│   └── positions.js
│
├── utils/
│   ├── jwt.js
│   └── VehiculeSimulator.js
│
```

```
├── .env
├── ecosystem.config.cjs
└── server.js
```

Description des Fichiers

Backend

Middleware

auth.js

- Gère l'authentification des utilisateurs à l'aide de JSON Web Tokens (JWT). Vérifie si un utilisateur est authentifié et valide le jeton JWT.

authSocket.js

- Middleware spécifique pour l'authentification des connexions socket, garantissant que seules les connexions authentifiées peuvent accéder aux fonctionnalités nécessitant une authentification.

isadmin.js

- Vérifie si l'utilisateur authentifié a le rôle d'administrateur (`admin`). Utilisé pour restreindre l'accès aux routes et fonctionnalités réservées aux administrateurs.

userMiddleware.js

- Gestion des opérations liées aux utilisateurs comme la validation des données utilisateur.

Models

Position.js

- Modèle décrivant la structure des données pour les positions dans l'application (ex: latitude, longitude, timestamp, etc.).

User.js

- Modèle représentant la structure des données des utilisateurs (ex: nom d'utilisateur, mot de passe haché, rôle, etc.).

Routes

admin.js

- Routes et contrôleurs spécifiques aux administrateurs pour la gestion des utilisateurs, des rôles, etc.

auth.js

- Routes pour l'authentification des utilisateurs, telles que l'inscription (`/register`) et la connexion (`/login`). Utilise les middlewares `auth.js` et `isadmin.js`.

positions.js

- Routes pour la gestion des positions, incluant l'enregistrement de nouvelles positions et la récupération des positions par utilisateur.

Utils

jwt.js

- Utilitaire pour la gestion des JSON Web Tokens (JWT) avec des fonctions de création, validation et gestion des jetons.

VehiculeSimulator.js

- Utilitaire pour simuler les données de véhicules, utilisé pour des tests ou la simulation de données en temps réel.

Fichiers et Configuration

.env

- Fichier de configuration contenant les variables d'environnement telles que les clés secrètes, les URI de bases de données, etc.

ecosystem.config.cjs

- Fichier de configuration pour PM2, un gestionnaire de processus Node.js. Il définit les configurations telles que le nombre de processus, les variables d'environnement, etc., pour le déploiement et la gestion en production.

server.js

- Fichier principal de démarrage du serveur backend. Configure et lance l'application Express, définit les middlewares globaux, les routes principales, établit la connexion aux bases de données, et utilise PM2 pour la gestion des processus en production.

Frontend

- **lib/ui/screens/login_screen.dart** : Interface utilisateur pour la connexion des utilisateurs.

- **lib/ui/screens/register_screen.dart** : Interface utilisateur pour l'inscription des utilisateurs.
- **lib/services/api_service.dart** : Service pour les appels API entre le frontend et le backend.
- **lib/models/user.dart** : Modèle de données pour l'utilisateur dans l'application Flutter.
- **pubspec.yaml** : Fichier de configuration de Flutter, listant les dépendances du projet.
- **.env** : Fichier contenant les variables d'environnement pour la configuration de l'application Flutter.