HubUniversity – Guide d'Installation et d'Utilisation

Ce projet regroupe trois composants essentiels :

- **PostgreSQL** pour la base de données
- **Backend Node.js/Express** pour la logique serveur
- **Frontend React.js** pour l'interface utilisateur

Ce guide vous aidera à installer et lancer l'ensemble via Docker et Docker Compose.

Prérequis

Avant de commencer, assurez-vous d'avoir :

- Un système Windows ou Mac
- Une connexion internet stable
- Un accès administrateur sur votre machine
- Docker Desktop installé

1. Installer Docker Desktop

Pour Windows / Mac

1. **Téléchargement**
   Rendez-vous sur le site officiel de Docker Desktop et cliquez sur **Download Docker Desktop** en choisissant la version adaptée à votre système.
2. **Installation**
   - Exécutez l'installateur téléchargé et suivez les instructions.
   - **Sur Windows :** Activez WSL2 (Windows Subsystem for Linux) si demandé.
   - Redémarrez votre ordinateur après l'installation.
3. **Vérification**
   Ouvrez Docker Desktop et vérifiez l'installation avec la commande :
   ```
   docker --version
   ```
4.

2. Structure du Projet

Assurez-vous que le projet dispose de la structure suivante :

```
HubUniversity/
├── docker-compose.yml
├── database/
```

```
│   └── init.sql
├── backend/
│   ├── Dockerfile
│   └── (vos fichiers backend : server.js, etc.)
└── frontend/
    ├── package.json
    ├── public/
    └── src/
        ├── App.js
        └── firebase.js
```

3. Fichiers Clés

3.1. docker-compose.yml

Ce fichier définit trois services : PostgreSQL, Backend et Frontend.

```
version: '3.8'
services:
  db:
    image: postgres:17
    container_name: projet_universite_db
    environment:
      POSTGRES_USER: postgres
      POSTGRES_HOST_AUTH_METHOD: trust
      POSTGRES_DB: projet_universite
    ports:
      - "5432:5432"
    volumes:
      - ./database/init.sql:/docker-entrypoint-initdb.d/
init.sql
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U postgres"]
      interval: 5s
      timeout: 5s
      retries: 5

  backend:
    build: ./backend
    container_name: projet_universite_backend
    environment:
      - DATABASE_URL=postgres://postgres@db:5432/
projet_universite
      - NODE_ENV=production
    depends_on:
```

```
      - db
    ports:
      - "5000:5000"
    volumes:
      - ./backend:/app
      - /app/node_modules

  frontend:
    image: node:18
    container_name: projet_universite_frontend
    working_dir: /app
    environment:
      - CHOKIDAR_USEPOLLING=true   # Pour une bonne détection
des changements
      - HOST=0.0.0.0                # Pour écouter sur toutes
les interfaces
    volumes:
      - ./frontend:/app
      - /app/node_modules
    ports:
      - "3000:3000"
    command: npm start
```

3.2. database/init.sql

Ce fichier initialise la base de données et crée la table `users` :

```
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    created_at TIMESTAMP DEFAULT NOW()
);

INSERT INTO users (name, email) VALUES
  ('Alice Dupont', 'alice.dupont@example.com'),
  ('Bob Martin', 'bob.martin@example.com');
```

3.3. Backend

Assurez-vous que le dossier `backend` contient :

- Un **Dockerfile** (pour construire l'image Node.js)
- Vos fichiers de code (ex. `server.js`)
- La variable d'environnement `DATABASE_URL` est configurée dans docker-compose pour pointer vers `postgres://postgres@db:5432/projet_universite`

Exemple de Dockerfile (dans `backend/`):

```
# Utiliser l'image officielle Node.js
FROM node:18

# Définir le dossier de travail
WORKDIR /app

# Copier les fichiers package.json et installer les
dépendances
COPY package*.json ./
RUN npm install

# Copier le reste du code
COPY . .

# Exposer le port sur lequel l'application écoute
EXPOSE 5000

# Lancer le serveur
CMD ["node", "server.js"]
```

3.4. Frontend

Le frontend est une application React.js. Dans le dossier `frontend` :

1. **Initialiser l'application et installer les dépendances**
   Depuis le dossier `frontend`, exécutez :
   ```
   npx create-react-app .
   ```
2. `npm install axios react-router-dom firebase`
3.
4. **Créer/Modifier `src/firebase.js`**
   ```
   import { initializeApp } from "firebase/app";
   ```
5. `import { getAuth, GoogleAuthProvider } from "firebase/ auth";`
6.
7. `const firebaseConfig = {`
8. `    apiKey: "VOTRE_API_KEY",`
9. `    authDomain: "VOTRE_AUTH_DOMAIN",`
10. `    projectId: "VOTRE_PROJECT_ID",`
11. `    storageBucket: "VOTRE_STORAGE_BUCKET",`
12. `    messagingSenderId: "VOTRE_MESSAGING_SENDER_ID",`
13. `    appId: "VOTRE_APP_ID"`
14. `};`
15.
16. `const app = initializeApp(firebaseConfig);`
17. `export const auth = getAuth(app);`

```
export const provider = new GoogleAuthProvider();

```

Modifier `src/App.js`

```
import { auth, provider } from "./firebase";

import { signInWithPopup } from "firebase/auth";

function App() {
    const login = async () => {
        const result = await signInWithPopup(auth, provider);
        console.log("Utilisateur connecté :", result.user);
    };

    return (
        <div>
            <h1>Bienvenue sur l'Université Connectée 🚀</h1>
            <button onClick={login}>Connexion avec Google</button>
        </div>
    );
}

export default App;

```

4. Lancer le Projet

Depuis la racine du projet (`HubUniversity/`), lancez la commande suivante pour démarrer tous les services :

```
docker-compose up --build -d
```

Vérification

- **Base de données (PostgreSQL) :**
  Vérifiez que le conteneur `projet_universite_db` est en cours d'exécution avec :
  `docker ps`
-
- **Backend :**
  Accessible via http://localhost:5000.
- **Frontend :**
  Accessible via http://localhost:3000.

5. Connexion à PostgreSQL

5.1. Via Docker

Pour accéder à la base de données depuis le terminal :

```
docker exec -it projet_universite_db psql -U postgres -d projet_universite
```

Ensuite, vous pouvez tester la table `users` :

```
SELECT * FROM users;
```

5.2. Via un Client PostgreSQL

Si vous utilisez un outil comme DBeaver ou pgAdmin, configurez la connexion avec :

- **Host :** 127.0.0.1
- **Port :** 5432
- **User :** postgres
- **Database :** projet_universite
- **Password :** (laisser vide, l'authentification en mode "trust" est activée)

6. Arrêt et Suppression des Services

Pour arrêter tous les conteneurs :

```
docker-compose down
```

Pour supprimer également les volumes (attention : toutes les données seront supprimées) :

```
docker-compose down -v
```

7. Dépannage

Si vous rencontrez des problèmes :

- **Vérifier que Docker fonctionne :**
  `docker info`
-
- **Consulter les logs :**
  - PostgreSQL : `docker logs projet_universite_db`
  -
  - Backend : `docker logs projet_universite_backend`
  -
  - Frontend : `docker logs projet_universite_frontend`
  -