Rapport de projet

Présentation générale du projet

Le projet est une application web de gestion des utilisateurs, conçue pour permettre l'ajout, la modification, et la suppression d'utilisateurs. L'application utilise une architecture client-serveur, où le frontend est développé avec React, et le backend est construit avec Node.js et Express. La base de données MySQL est utilisée pour stocker les informations des utilisateurs. Docker est employé pour la conteneurisation, ce qui facilite le déploiement et la gestion des environnements.

Étapes de mise en place du backend et frontend Backend

- Installation des dépendances : Utilisation de npm install pour installer les packages nécessaires tels qu'Express pour le serveur web et MySQL pour la gestion de la base de données.
- Configuration du serveur : Mise en place d'un serveur Express avec des routes pour les opérations CRUD (Create, Read, Update, Delete) sur les utilisateurs. Les routes sont définies dans le fichier index.js .
- Connexion à la base de données : Configuration de la connexion MySQL dans le fichier database.js , en utilisant le module mysql2 pour gérer les interactions avec la base de données.
- Démarrage du serveur : Utilisation de npm start pour lancer le serveur, qui écoute sur le port 5000. Frontend
- Installation des dépendances : Utilisation de npm install pour installer React et d'autres bibliothèques comme Axios pour les requêtes HTTP et React Toastify pour les notifications.
- Démarrage de l'application : Utilisation de npm start pour lancer l'application React, qui s'exécute sur le port 3000.
- Intégration API : Configuration des appels API dans le composant UserList.js pour interagir avec le backend, permettant de récupérer, ajouter, modifier et supprimer des utilisateurs.

Explication de la base de données

La base de données MySQL est structurée avec une table users contenant les colonnes id , name , et email . Les opérations CRUD sont implémentées pour gérer les utilisateurs. La connexion à la base de données est gérée par le module mysql2 , et des requêtes SQL sont utilisées pour interagir avec les données.

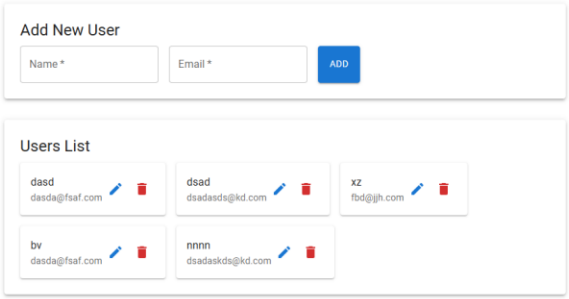
Dockerisation : étapes et choix faits

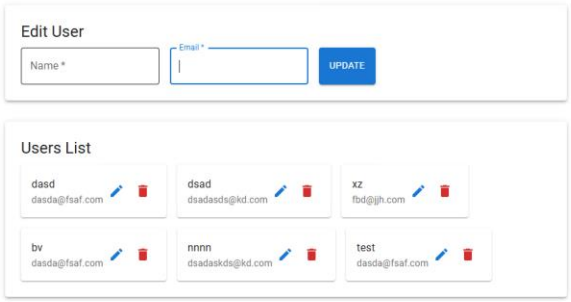
- Dockerfile : Création de Dockerfiles pour le backend et le frontend, permettant de construire des images Docker. Le Dockerfile du backend utilise Node.js pour exécuter le serveur, tandis que celui du frontend utilise Nginx pour servir l'application React.
- docker-compose : Utilisation de docker-compose.yml pour orchestrer les conteneurs, incluant MySQL, le serveur backend, et le frontend. Cela permet de gérer facilement les dépendances et de lancer tous les services avec une seule commande.
- Choix techniques : Docker est choisi pour sa capacité à isoler les environnements, assurer la portabilité entre les systèmes, et simplifier le déploiement en production.

GitHub Actions : pipeline expliqué étape par étape

- CI/CD Pipeline : Mise en place d'un fichier YAML pour automatiser les tests et le déploiement. Le pipeline est déclenché à chaque push sur le dépôt GitHub.
- Étapes du pipeline :
    - Checkout : Récupération du code source depuis le dépôt.
    - Installation : Installation des dépendances pour le frontend et le backend.
    - Tests : Exécution des tests unitaires pour s'assurer que le code fonctionne correctement.
    - Build : Construction des images Docker pour le frontend et le backend.
    - Déploiement : Déploiement des conteneurs sur un serveur distant, si applicable.

Captures d'écran des tests, conteneurs Docker


Add New User
Name *
Email *
ADD
Users List
dasd
dasda@fsaf.com
dsad
dsadasds@kd.com
xz
fbd@jjh.com
bv
dasda@fsaf.com
nnnn
dsadaskds@kd.com


Edit User
Name *
Email *
UPDATE
Users List
dasd
dasda@fsaf.com
dsad
dsadasds@kd.com
xz
fbd@jjh.com
bv
dasda@fsaf.com
nnnn
dsadaskds@kd.com
test
dasda@fsaf.com

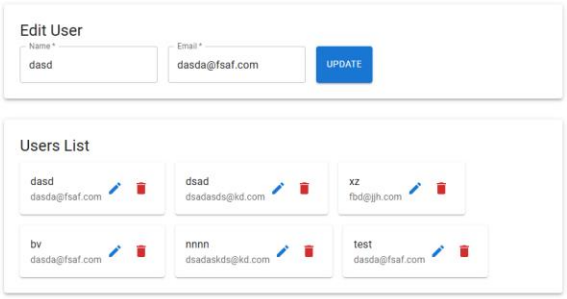


Image 1 : Ajout d'un nouvel utilisateur

- Cette image montre l'interface pour ajouter un nouvel utilisateur. En haut, il y a un formulaire avec des champs pour entrer le nom et l'email de l'utilisateur. Une fois les champs remplis, cliquer sur le bouton "ADD" enverra une requête au serveur pour créer un nouvel utilisateur. En dessous du formulaire, il y a une liste des utilisateurs existants, chacun affiché avec son nom et son email. Chaque entrée d'utilisateur a des icônes pour modifier et supprimer l'utilisateur. Le bouton rouge est utilisé pour supprimer un utilisateur.

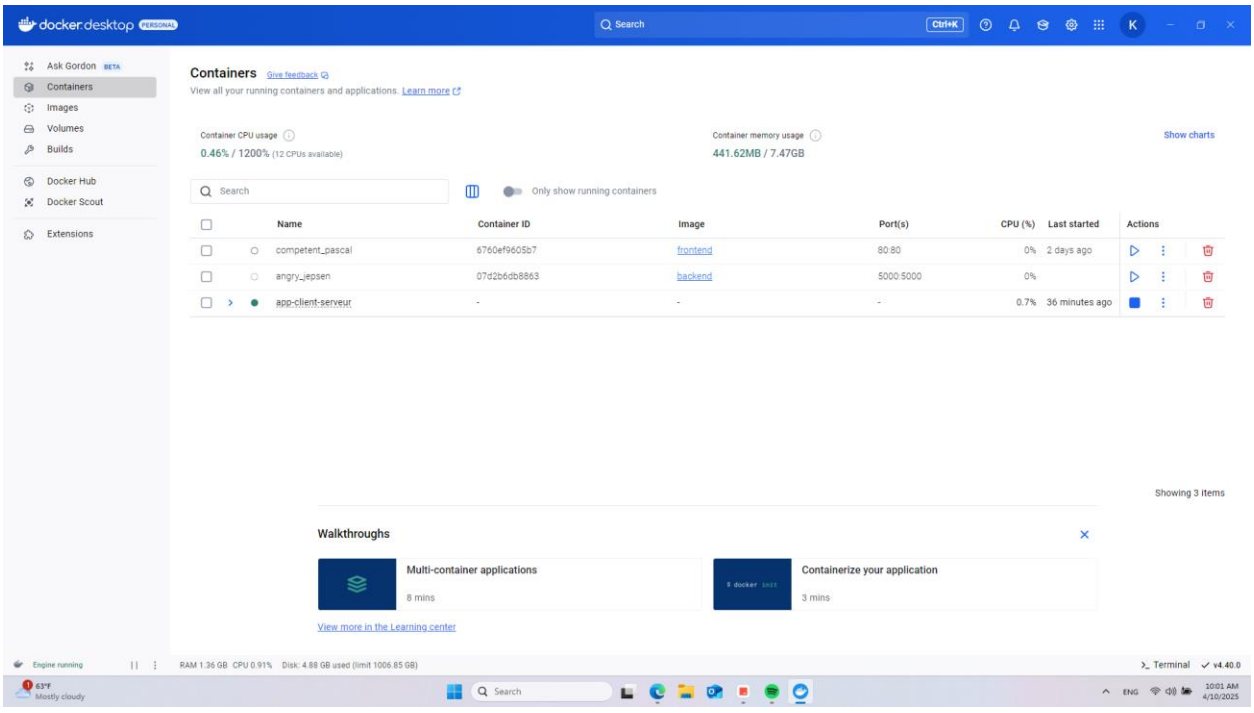
Image 2 : Modification d'un utilisateur

- Cette image illustre l'interface pour modifier un utilisateur existant. Le formulaire en haut est intitulé "Edit User" et contient des champs pré-remplis avec le nom et l'email de l'utilisateur sélectionné. L'utilisateur peut modifier ces champs et cliquer sur le bouton "UPDATE" pour enregistrer les modifications. La liste des utilisateurs est affichée en dessous, similaire à la première image, avec des options pour modifier ou supprimer chaque utilisateur. Le bouton rouge permet de supprimer un utilisateur.

Image 3 : Mise à jour d'un utilisateur

- Cette image montre le processus de mise à jour des informations d'un utilisateur. Le formulaire en haut est utilisé pour modifier les détails de l'utilisateur, avec les champs affichant le nom et l'email actuels de l'utilisateur en cours de modification. Après avoir apporté des modifications, cliquer sur le bouton "UPDATE" enverra une requête pour mettre à jour les informations de l'utilisateur sur le serveur. La liste des utilisateurs est affichée en dessous, reflétant les mises à jour effectuées. Le bouton rouge est utilisé pour supprimer un utilisateur.

Ces images démontrent collectivement la fonctionnalité de l'application pour gérer les utilisateurs, y compris l'ajout de nouveaux utilisateurs, la modification des utilisateurs existants et la mise à jour des informations des utilisateurs. Chaque entrée d'utilisateur dans la liste offre des options pour modifier et supprimer, permettant une gestion complète des utilisateurs. Le bouton rouge est spécifiquement utilisé pour supprimer un utilisateur de la liste.



Description du conteneur app-client-serveur

- Nom du conteneur : Le conteneur est nommé app-client-serveur , ce qui indique qu'il est lié à l'application client-serveur que vous avez développée.
- Image : La colonne "Image" est vide pour ce conteneur, ce qui peut suggérer qu'il s'agit d'un conteneur composite ou d'un service orchestré par Docker Compose.
- Ports : La colonne "Port(s)" est également vide, ce qui peut indiquer que ce conteneur n'expose pas directement de ports ou que les ports sont gérés par les conteneurs individuels frontend et backend .
- Utilisation CPU et mémoire : Le conteneur app-client-serveur utilise une faible quantité de ressources CPU et mémoire, ce qui est typique pour un conteneur en veille ou un conteneur qui orchestre d'autres services.
- Actions : À droite, il y a des icônes pour démarrer, arrêter ou supprimer le conteneur. Le bouton rouge avec une icône de poubelle est utilisé pour supprimer le conteneur. Ce conteneur semble être une partie intégrante de l'application, assurant la coordination entre les services frontend et backend. Il est essentiel pour le bon fonctionnement de l'application client-serveur, permettant une gestion efficace des ressources et des services.

Difficultés rencontrées et solutions

- Problèmes de connexion à la base de données : Des erreurs de connexion ont été résolues en ajustant les paramètres de connexion et en ajoutant des mécanismes de reconnexion automatique.
- Erreurs de déploiement : Des problèmes de déploiement ont été résolus en ajustant les configurations Docker et en utilisant des logs pour le débogage.

Conclusion et axes d'amélioration

Le projet a permis de mettre en place une application web robuste avec une architecture conteneurisée. Pour l'avenir, des améliorations peuvent être apportées en optimisant les performances, en ajoutant de nouvelles fonctionnalités, et en améliorant l'expérience utilisateur.