Rapport

Mini-X

auteurs:
Ahmad HARKOUS
An Vinh Lala
Valentin Dauzere-Peres

Mai, 2024

Table des matières

1 Introduction

Ce rapport décrit la solution proposée et les problèmes rencontrés pour construire le backend d'un mini réseau social similaire à X, anciennement connu sous le nom de Twitter.
L'objectif était de développer des APIs basiques fonctionnelles avec l'intégration de plugins sous Redis et de mettre en place une infrastructure professionnelle réelle.

La solution à était déployer sur un serveur physique réel. L'application est accessible à l'adresse suivante : `https://x.devgains.com`.

-Objectifs du projet :

Développer des APIs fonctionnelles :

- APIs publiques (méthodes GET)
- APIs privées (méthodes POST/PUT/UPDATE)
- Plugin pour obtenir les logs

Mettre en place une infrastructure professionnelle :

- Utilisation d'Apache comme reverse proxy
- Séparation de la gestion des APIs publiques et privées
- Packager tous les micro-services avec docker compose

Sécuriser les APIs :

- Protection des APIs privées avec OpenID et le serveur d'identité keycloak.

2 Base de donnée

Notre schéma de base de données relationnelle est conçu pour gérer un système de tickets et de salles, permettant de suivre les modifications des tickets et les activités des utilisateurs.

La modélisation de la base de données a été réalisée à l'aide de l'**ORM**

Sequelize [1] avec **PostgreSQL** et **Node.js**.

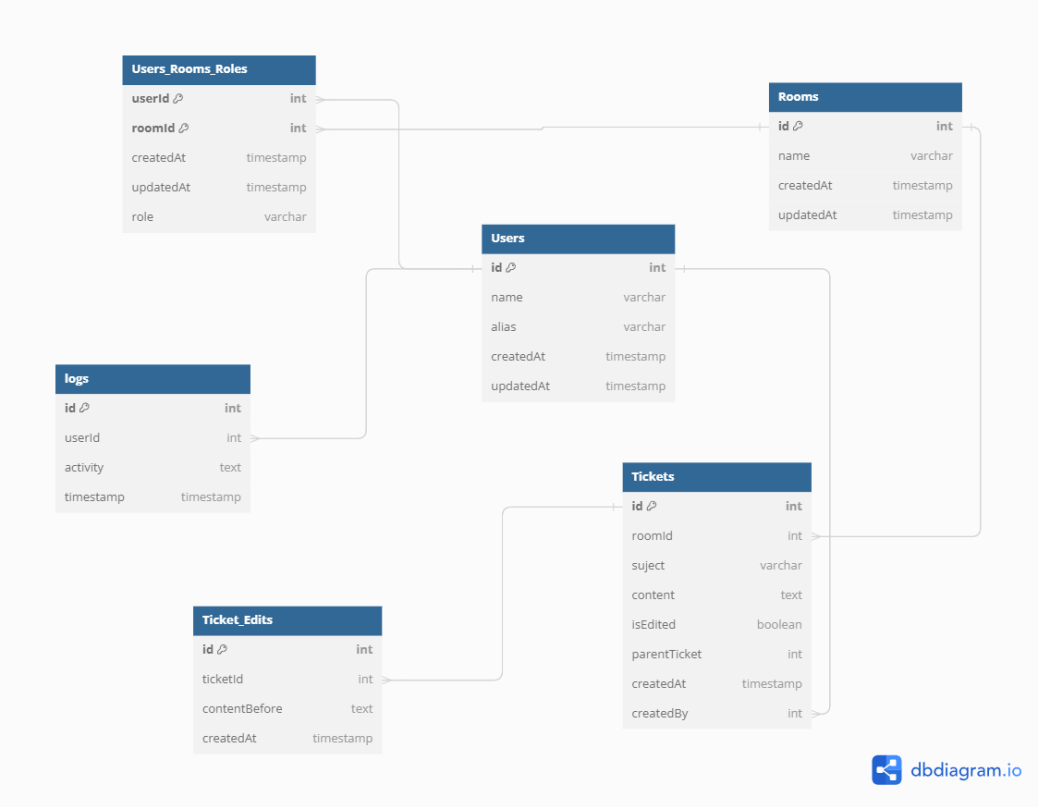

Figure 1: MLD

Voici une explication de chaque table et de leurs relations :

Table Users

Cette table contient les informations des utilisateurs.

- **id** (`int, pk`) : Identifiant unique de l'utilisateur (clé primaire).
- **name** (`varchar`) : Nom de l'utilisateur.
- **alias** (`varchar`) : Pseudonyme de l'utilisateur.
- **createdAt** (`timestamp`) : Date et heure de création de l'utilisateur.

[1] Voir `https://sequelize.org`

- **updatedAt** (`timestamp`) : Date et heure de la dernière mise à jour des informations de l'utilisateur.

Table Rooms

Cette table contient les informations des salles.

- **id** (`int, pk`) : Identifiant unique de la salle (clé primaire).
- **name** (`varchar`) : Nom de la salle.
- **createdAt** (`timestamp`) : Date et heure de création de la salle.
- **updatedAt** (`timestamp`) : Date et heure de la dernière mise à jour des informations de la salle.

Table Tickets

Cette table gère les tickets, qui sont des éléments de communication ou des rapports.

- **id** (`int, pk`) : Identifiant unique du ticket (clé primaire).
- **roomId** (`int`) : Identifiant de la salle associée au ticket, avec une référence vers la table `Rooms`.
- **subject** (`varchar`) : Sujet du ticket.
- **content** (`text`) : Contenu du ticket.
- **isEdited** (`boolean`) : Indicateur si le ticket a été édité.
- **parentTicket** (`int`) : Identifiant du ticket parent, pour les tickets de suivi ou les réponses.
- **createdAt** (`timestamp`) : Date et heure de création du ticket.
- **createdBy** (`int`) : Identifiant de l'utilisateur ayant créé le ticket, avec une référence vers la table `Users`.

Table Users_Rooms_Roles

Cette table gère les rôles des utilisateurs dans les salles.

- **userId** (`int, pk`) : Identifiant de l'utilisateur, avec une référence vers la table `Users` (clé primaire composite).
- **roomId** (`int, pk`) : Identifiant de la salle, avec une référence vers la table `Rooms` (clé primaire composite).
- **createdAt** (`timestamp`) : Date et heure de création de l'entrée.
- **updatedAt** (`timestamp`) : Date et heure de la dernière mise à jour de l'entrée.
- **role** (`varchar`) : Nom du rôle de l'utilisateur dans la salle (par exemple, 'moderator', 'super_admin').

Table Ticket_Edits

Cette table suit les modifications apportées aux tickets.

- **id** (`int, pk`) : Identifiant unique de l'édition (clé primaire).
- **ticketId** (`int`) : Identifiant du ticket modifié, avec une référence vers la table `Tickets`.
- **contentBefore** (`text`) : Contenu du ticket avant modification.
- **createdAt** (`timestamp`) : Date et heure de la modification.

Table Logs

Cette table enregistre les activités des utilisateurs.

- **id** (`int, pk`) : Identifiant unique de l'entrée de log (clé primaire).
- **userId** (`int`) : Identifiant de l'utilisateur ayant effectué l'activité, avec une référence vers la table `Users`.
- **activity** (`text`) : Description de l'activité.
- **timestamp** (`timestamp`) : Date et heure de l'activité.

Relations entre les tables

- **Users** et **Tickets** : Chaque ticket est créé par un utilisateur.
- **Rooms** et **Tickets** : Chaque ticket est associé à une salle.
- **Users** et **Users_Rooms_Roles** : Un utilisateur peut avoir différents rôles dans différentes salles.
- **Rooms** et **Users_Rooms_Roles** : Une salle peut avoir différents utilisateurs avec différents rôles.
- **Tickets** et **Ticket_Edits** : Les modifications des tickets sont suivies dans la table `Ticket_Edits`.
- **Users** et **Logs** : Les activités des utilisateurs sont enregistrées dans la table `Logs`.

3 Infra Reseau

Voici un schéma détaillant l'infrastructure globale de notre solution:

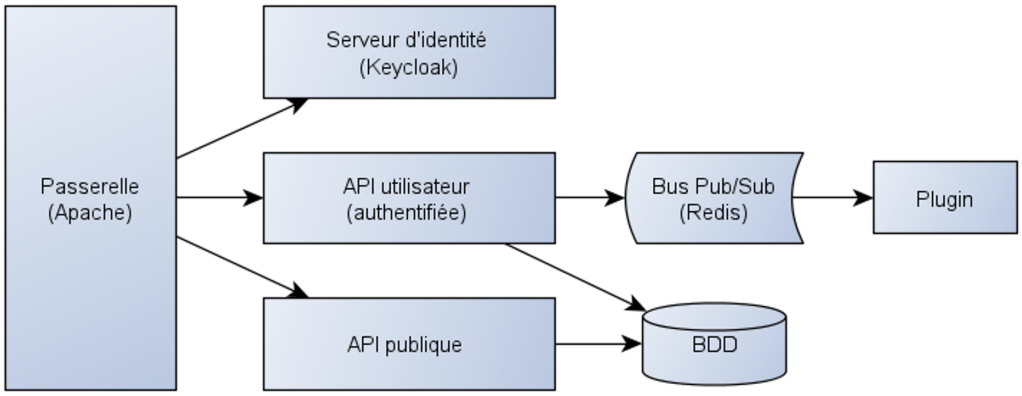

Figure 2: Archi-Infra. Ref: Fabio Guigou

3.1 Configuration de l'Apache

Pour Apache, nous avons reconstruit l'image de httpd:2.4 en y ajoutant le plugin mod_auth_openidc pré-téléchargé, ainsi que les certificats SSL valides générés avec Certbot pour le site.
Nous avons également copié la configuration globale d'Apache, disponible

dans le répertoire apache-config, et configuré le site avec les directives ProxyPass suivantes :

- /auth pour le conteneur Keycloak
- /public pour l'API publique
- /private pour l'API privée

3.2 Intégration de Keycloak

Pour Keycloak, nous avons décidé de l'intégrer avec une base de données **PostgreSQL**, afin de se rapprocher le plus possible des conditions réelles de production.

Nous avons reconstruit l'image de la base de données postgresql de Keycloak en y copiant un fichier de sauvegarde SQL et un script restore.sh permettant de mettre à jour la base de données lors du point d'entrée de Docker. Le container est dispo sur le port 8080.

L'utilisateur va être redirigé vers le formulaire de connexion de Keycloak avant d'accéder à l'API privée.

3.3 Infra des APIs

Les APIs publiques et privées sont gérées par la même application **Node.js Express** sur le port 4000, mais avec des routes différentes.

Nous avons choisi de ne pas créer deux applications séparées afin d'optimiser les ressources et la maintenance.
L'application Node.js communique avec un conteneur Redis sur le **port 6379** au sein du même réseau interne Docker.

3.4 Middleware Keycloak

Pour l'application déployée sur le serveur de production réel, nous avons également ajouté le middleware Keycloak Connect à toutes les routes. Cela permet d'assurer une gestion sécurisée et centralisée des accès utilisateurs avec les access tokens.

4 Fonctionnement de notre réseau social

4.1 Gestion des Salles de Communication

- **Rejoindre et Quitter une Salle :**
  - Les utilisateurs peuvent rejoindre ou quitter une salle de communication à tout moment.
- **Création de Salle :**
  - Lorsqu'un utilisateur crée une nouvelle salle, il est automatiquement assigné au rôle d'**administrateur**.
  - Chaque salle ne peut avoir qu'un seul administrateur, mais le nombre d'utilisateurs dans une salle est illimité.

4.2 Rôles et Permissions de l'Administrateur

- **Exclure un Utilisateur :**
  - Il peut expulser un utilisateur non administrateur de la salle.
- **Assigner des Rôles :**
  - Par défaut, les nouveaux membres n'ont pas de rôle. L'administrateur doit assigner un rôle à chaque utilisateur.
- **Modifier le Nom de la Salle :**
  - Il peut changer le nom de la salle.
- **Supprimer la Salle :**
  - Il peut supprimer la salle.

4.3 Gestion des Tickets

- **Création de Ticket :**
  - Les utilisateurs peuvent créer un ticket uniquement dans une salle qu'ils ont rejointe.
- **Vérifications et Sécurité :**

  - Des vérifications sont en place pour empêcher les utilisateurs de poster des messages dans des salles qu'ils n'ont pas rejointes, avec un code de réponse 403 (Forbidden).

- **Réponse et Édition de Ticket :**
  - Les utilisateurs peuvent répondre à un ticket et éditer leurs propres tickets.
  - Des vérifications sont aussi mise en place pour vérifier que le ticket éditer appartient à l'utilisateur en question .

4.4 Gestion des Transactions

- Chaque transaction est enregistrée dans table logs de la BD à l'aide de Redis Subscriber.
- Les logs sont récuperables à l'aide des endpoints get

4.5 Documentation des APIs

- La documentation complète des APIs est disponible sur Swagger.`https://x.devgains.com/private/api-docs` et `https://x.devgains.com/public/api-docs`.

5 Problèmes Rencontrés et Optimisations à Apporter

5.1 Problèmes Rencontrés

Nous avons rencontré plusieurs difficultés lors de la configuration du reverse proxy Apache avec le serveur d'identité Keycloak et OpenID :

- Parfois, des erreurs telles que `cannot find redirect_uri` se produisaient.
  - **Solution :** Vérifier que l'URL de redirection configurée dans Keycloak correspond exactement à l'URL utilisée par l'application. S'assurer également que les configurations de proxy dans Apache sont correctement définies.
- Au début, le souscripteur Redis ne récupérait pas les messages correctement.

- **Solution :** S'assurer que les configurations de connexion à Redis sont correctes et que le souscripteur est correctement abonné aux canaux pertinents. Vérifier également la connectivité réseau entre les conteneurs.

5.2 Optimisations à Apporter

Voici les points à optimiser pour améliorer notre solution :

- Ajouter de la documentation directement dans le code pour faciliter la compréhension et la maintenance.
- Optimiser le temps de construction des images Docker.
- Ajouter des protections avec différents rôles sur le middleware Keycloak pour renforcer la sécurité.

A Annexes