Sorbonne Université
Faculté de Science et d'Ingénierie
Département Informatique

Rapport du Projet PC3R

Informatique

Spécialité : Science et Technologie Logiciel

MerYouZik

Encadré par

- Romain Damangeon

Réalisé par

- Benaissa Meriem - 21418006
- AHMED Youra 21416343

Promotion 2024 - 2025

Contents

1 Introduction

MerYouZik est un réseau social autour de la musique, conçu sous forme d'une application web. Il permet aux utilisateurs de découvrir des chansons en accédant à diverses informations telles que le titre, l'artiste, la durée et le classement. Chaque morceau est présenté sous forme de publication interactive, où l'on peut aimer, commenter et répondre aux commentaires, créant ainsi une véritable communauté musicale en ligne.

2 Schéma global

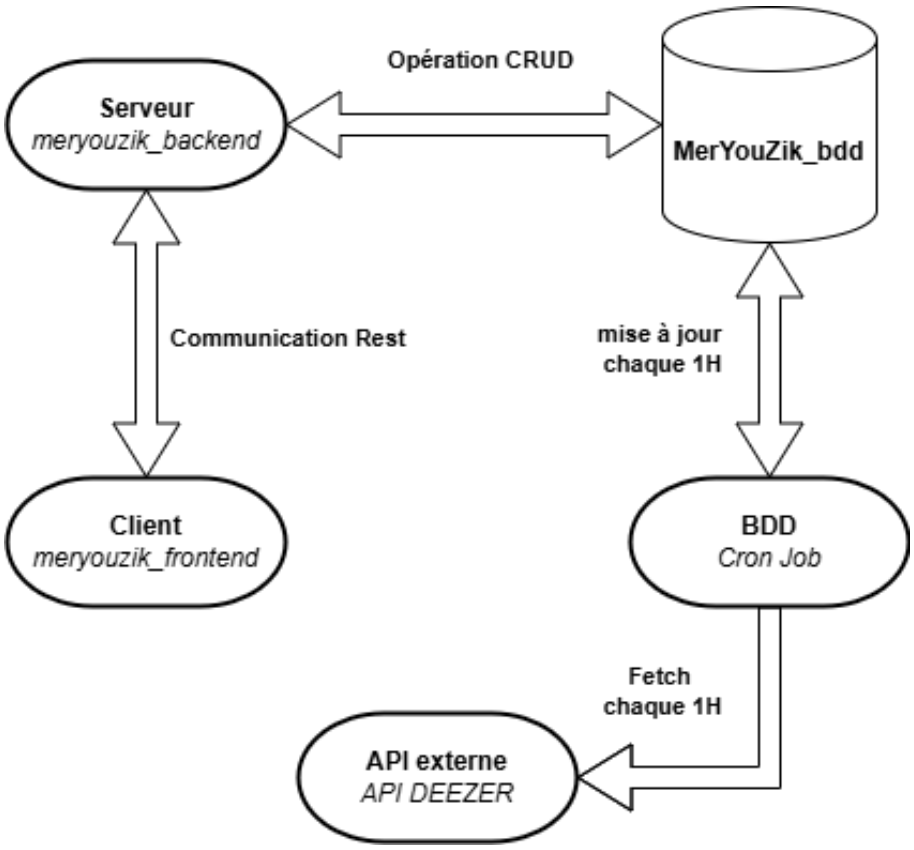

Figure 1: Schéma global

3 Choix de téchnologie

Nous avons opté pour une architecture basée sur **Go** côté backend et **React** côté frontend pour les raisons suivantes :

- **Performance** :
  - Go offre des performances natives avec une consommation mémoire optimisée
  - React permet un rendu virtuel efficace du DOM
- **Productivité** :
  - Simplicité de développement avec Go (syntaxe claire, outils intégrés)
  - Composants réutilisables et écosystème riche avec React
- **Sécurité** :
  - Typage statique fort de Go réduisant les erreurs runtime
  - Isolation des composants React limitant les failles XSS
- **Maintenabilité** :
  - Code Go clair
  - Structure prévisible des composants React

4 Déploiement en ligne

Nous avons choisi de déployé notre application en utilisant Render, voila ce qu'on a fait dedans;

4.0.1 Étapes de déploiment

Les déploiements de meryouzik_backend, meryouzik_frontend et bdd_refresh ont plusieurs points communs :

- Le code source des trois composants se trouve dans un même dépôt GitHub, chacun dans son propre répertoire.
- Les variables d'environnement utilisées contiennent les informations nécessaires à la connexion avec la base de données (nom de la base, mot de passe, etc.).

Voici les différents déploiements qui ont été effectués :

4.1 meryouzik_bdd

c'est notre base de données psql qu'est déployé sur render.

4.2 bdd_refresh

c'est le cron job qui permet de rajouter les nouveaux données de DEEZER, tel que chaque 1h le programme de bdd.go se lance et rajoute les nouveaux tuples dans notre base de donnée (ce qu'est dans le dossier BDD dans notre dépôt github).

4.3 meryouzik_backend

Utiliser pour lancer le serveur indéfiniment et premet de répondre au requête du client (le répertoire concerné est backend).

4.4 meryouzik_frontend

Utiliser pour lancer notre application MerYouZik aprés avoir rajouté les variable d'environnement tel que le liens vers le meryouzik_backend pour permettre la communication entre le clien et le serveur (le répertoire concerné est frontend).

4.5 Lien de déploiement

Voici le lien : `https://meryouzik-frontend.onrender.com`.

5 L'API Source :

L'API Web choisie pour ce projet propose des données sur les chansons (Tracks) qui sont disponible sur DEEZER, ses albums, ses artists, ses genres, ses charts (TOP

10 des chansons), ainsi que podcast et radio, dans notre application on se sert des tables (Album, Artist, Chart, Genre, Track).

Voici le lien vers l'API de DEEZER : `https://developers.deezer.com/`.

5.1 Use Cases

Voila les cas d'utilisation existant dans notre application, on commence de la premiere connexion de l'utilisateur :

- L'utilisateur clique sur "Créer un compte"
- L'utilisateur entre son Username (nom d'utilisateur de MerYouZik), Email et Mot de passe deux fois pour confirmer.
- L'utilisateur clique sur le bouton "S'inscrire".
- Le serveur confirm que l'utilisateur n'existe pas et crée un nouveau utilisateur dans la base de donnée et envoie un token au front qui lui il se sert de lui pour envoyé un email de confirmation d'email à l'utilisateur, tel que la durée est de 24h de validité, puis le front rederige l'utilisateur vers la page aprés 5 secondes de l'inscription, une fois on clique sur le lien envoyé à l'email le compte utilisateur sera confirmer et peut se connecter à l'application.
- L'utilisateur peut maintenant entrer ses informations de connexion (email et mot de passe).
- Après la connexion l'utilisateur va être rederiger vers la page home où y a tout les publication.
- Après la connexion le serveur envoies les données des tracks et des charts et artists pour fromé des publications, en tête le TOP 10 et les affiche dans la page home.
- L'utilisateur à ce stade il peut faire des likes au publication.
- L'utilisateur peut rajouter, modifier, supprimer des commentaires sur les publications (supprimer et modifier c'est par rapport à ses propre commentaires uniquement).

- L'utilisateur peut répondre au commentaires des autres utilisateur autrement dit ajouté des réponses au commentaire, modifier et supprimer aussi.
- L'utilisateur peut effectué une recherche sur une publication soit par le titre de chanson, soit nom d'album ou artists.
- L'utilisateur peut se déconnecter et revenir à la page de connexion à nouveau.

6 Conception de la base de données

Le fichier qui permet de définir la base de données est trouvé dans le dossier Schema_bdd dans le dépot github.

les tables qu'on a sont les suivantes :

- up_users(id, username, email, password, confirmation_token, confirmation_expiry, confirmed)
- tracks(id, id_track*, title, link, id_album*, id_artist*, duration, rank)
- commentaires(id, contenu, id_user*, id_track*, date_commentaire)
- responses(id, contenu, date_response, id_comment*, id_user*)
- likes(id_like, id_user*, id_track*)
- genres(id, id_genre*, name)
- album_genres(id, id_genre*, id_album*)
- albums(id, title, link, release_date, id_album*, id_artist*)
- artists(id, name, id_artist*, link, picture)
- charts(id, title, link, id_artist*, id_album*, id_chart*, nom_artist, picture_artist, link_artist, nom_album, duration, rank)

7 Fonctionnalités

- Regarder le TOP 10 des chansons sur DEEZER, le reste des des chansons.
- Chercher les publication par nom d'album, nom de l'artist et titre de la chanson.
- Interagir avec les autres utilisateurs dans les commentaires et réponses.
- Pouvoir liker des chansons.
- S'inscrire/ se connecter à MerYouZik.

8 Backend

8.1 Fonctionnement global du backend

- L'application utilise une architecture RESTful avec des endpoints clairement définis pour chaque fonctionnalité.
- La connexion à la base de données PostgreSQL est gérée via le driver lib/pq, avec une initialisation au démarrage du serveur.
- Les handlers traitent les requêtes HTTP et interagissent avec la base de données pour fournir les données demandées.

9 Structure et architecture des fichiers

Le projet coté backend est organisé en trois fichiers principaux :

9.1 Fichiers principaux

- **database.go** : Contient la configuration et l'initialisation de la connexion à la base de données.
- **handlers.go** : Implémente tous les handlers pour les différentes routes de l'API.
- **main.go** : Point d'entrée du serveur, configure les routes et démarre l'application.

9.2 Gestion des requêtes asynchrones

- Les opérations de base de données sont exécutées de manière asynchrone pour ne pas bloquer le thread principal.
- L'utilisation de defer rows.Close() garantie la bonne gestion des ressources après chaque requête SQL.

9.2.1 Validation et sécurité

- Validation des entrées utilisateur pour l'inscription et la connexion.
- Hachage des mots de passe avec bcrypt avant stockage en base.
- Implémentation d'un système JWT pour l'authentification sécurisée.
- Vérification de l'email via un système de confirmation par token.

9.3 Handlers

L'application expose une API REST complète organisée en plusieurs catégories fonctionnelles :

9.3.1 Gestion des utilisateurs

- **Inscription** :
  - Requête : `POST /register` avec `{username, email, password}`
  - Réponse : Token de confirmation ou message d'erreur
  - Statut : 201 (Créé), 409 (Conflit) ou 400 (Requête invalide)
- **Connexion** :
  - Requête : `POST /login0` avec `{email, password}`
  - Réponse : Token JWT et informations utilisateur
  - Statut : 200 (OK), 401 (Non autorisé) ou 403 (Compte non confirmé)
- **Confirmation d'email** :

  - Requête : `GET /confirm-email?token={token}`
  - Réponse : Message de confirmation
  - Statut : 200 (OK), 400 (Token invalide) ou 410 (Lien expiré)

- **Profil utilisateur** :
  - Requête : `GET /api/me` (header Authorization)
  - Réponse : Informations de l'utilisateur connecté
  - Statut : 200 (OK) ou 401 (Non autorisé)

9.3.2 Recherche musicale

- **Recherche** :
  - Requête : `GET /search?q={query}&type={artist|track|album}`
  - Réponse : Liste des résultats
  - Statut : 200 (OK) avec résultats ou message "Aucun résultat"
- **Morceaux aléatoires** :
  - Requête : `GET /tracks?page={page}`
  - Réponse : Liste paginée
  - Statut : 200 (OK)
- **Classements** :
  - Requête : `GET /charts`
  - Réponse : Meilleurs morceaux
  - Statut : 200 (OK)

9.3.3 Interactions sociales

- **Likes** :
  - Ajout : `POST /like` avec `{id_user, id_track}` (201 Créé)
  - Suppression : `DELETE /unlike` avec `{id_user, id_track}` (200 OK)
  - Statut : `GET /likes?track_id={id}&user_id={id}` (200 OK)

9.3.4 Système de commentaires

- **Commentaires** :
  - Ajout : `POST /comment` (201 Créé)
  - Liste : `GET /comments?track_id={id}` (200 OK)
  - Modification : `PUT /comment/update` (200 OK)
  - Suppression : `DELETE /comment/delete` (200 OK)
- **Réponses** :
  - Ajout : `POST /response/add` (201 Créé)
  - Liste : `GET /response/get?comment_id={id}` (200 OK)
  - Nombre : `GET /response/count` (200 OK)
  - Modification : `PUT /response/update` (200 OK)
  - Suppression : `DELETE /response/delete` (200 OK)

9.3.5 Schéma des endpoints

```
User:
  POST   /register            → Création de compte
  POST   /login0              → Connexion
  GET    /confirm-email       → Confirmation email
  GET    /api/me              → Profil utilisateur

Music:
  GET    /search              → Recherche musicale
  GET    /tracks              → Morceaux aléatoires
  GET    /charts              → Classements

Social:
  POST   /like                → Ajouter un like
  DELETE /unlike              → Retirer un like
  GET    /likes               → Statut des likes
```

```
Comments:
  POST   /comment             → Ajouter un commentaire
  GET    /comments            → Lister les commentaires
  PUT    /comment/update      → Modifier un commentaire
  DELETE /comment/delete      → Supprimer un commentaire

Responses:
  POST   /response/add        → Répondre à un commentaire
  GET    /response/get        → Lister les réponses
  GET    /response/count      → Nombre de réponses
  PUT    /response/update     → Modifier une réponse
  DELETE /response/delete     → Supprimer une réponse
```

9.4 Gestion des erreurs

- Chaque handler inclut une gestion d'erreur appropriée avec des codes HTTP spécifiques.
- Les erreurs de base de données sont loguées et renvoyées sous forme structurée au client.

9.5 Sécurité et bonnes pratiques

- Utilisation de CORS pour les requêtes cross-origin.
- Protection contre les injections SQL via l'utilisation de requêtes paramétrées.
- Stockage sécurisé des informations sensibles via les variables d'environnement.
- Séparation claire entre la logique métier et la couche d'accès aux données.

10 Frontend

Notre application frontend développée avec **React TypeScript** suit une architecture modulaire organisée en plusieurs pages principales. L'interface a été conçue

selon les principes du **Responsive Web Design** pour garantir une expérience utilisateur optimale sur tous les appareils.

10.1 Architecture

Application monopage (SPA) avec les écrans suivants :

- **Page d'Accueil** (`/Home`)
  - Contenu : Barre de recherche, charts, recommandations
  - Appels serveur :
    - `GET /charts` (chargement initial)
    - `GET /tracks` (morceaux aléatoires)
    - `GET /search` (lors d'une recherche)
    - et autre
- **Page de Connexion** (`/login`)
  - Contenu : Formulaire d'authentification
  - Appels serveur : `POST /login0`
- **Page d'Inscription** (`/register`)
  - Contenu : Formulaire d'inscription
  - Appels serveur : `POST /register`

10.2 Expérience mobile

Pour ce genre d'application la majorité des utilisateur utilise les appareils mobile, pour cela on a essayé de garantir une bonne experience utilisateur mobile.

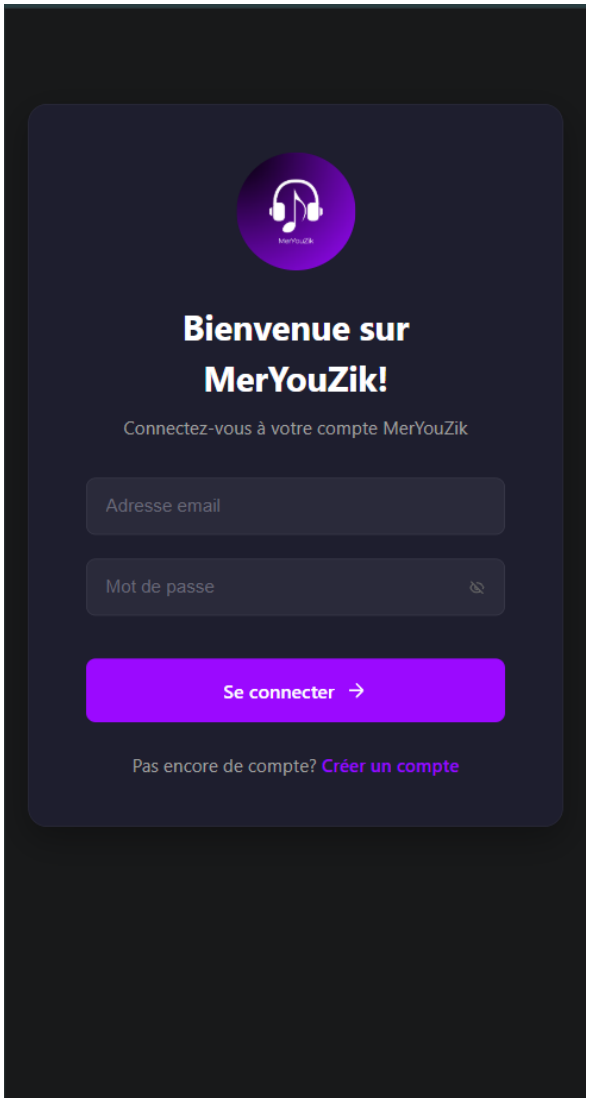

Figure 2: page de connexion

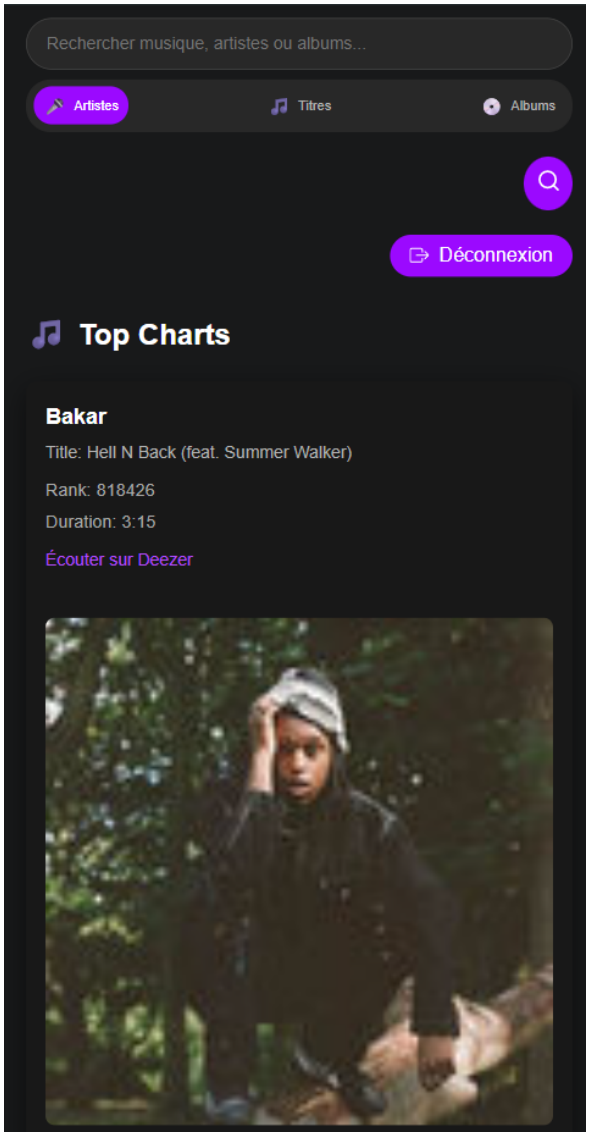

Figure 3: Home

Figure 4: Experience Mobile

10.3 Expérience sur les ordinateurs et grands écrans

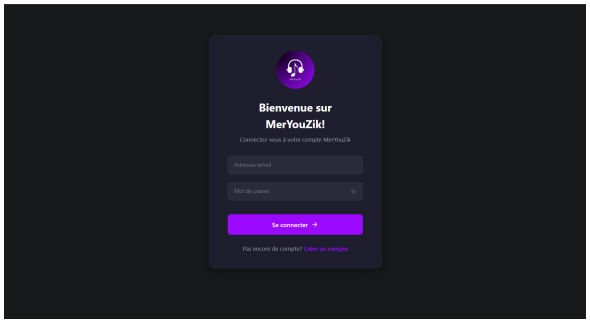

Figure 5: page de connexion

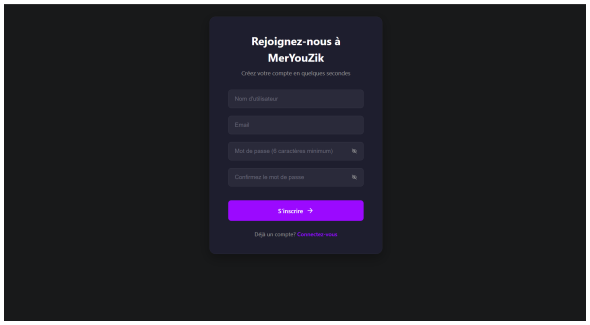

Figure 6: page d'inscription

Figure 7: Authentification & inscription

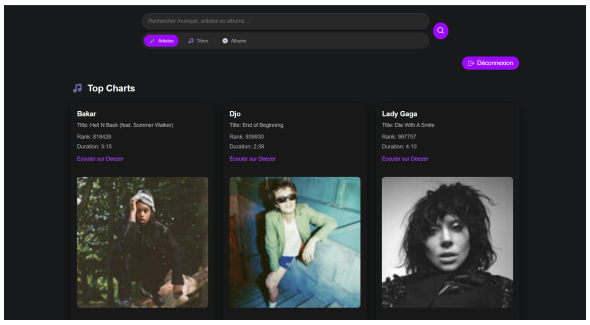

Figure 8: image 1 de home

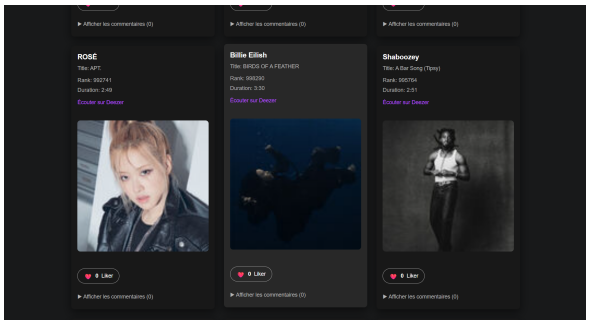

Figure 9: Home

Figure 10: image 2 de home

11 Perspectives

L'application nous a apporté beaucoup de bénifice et nous avons aprecié le faite de travail sur ce language, malheureusement nous avons pas eu beaucoup de temps, on a aimé améliorer plusieurs points parmi aux le faite le faite de refresh la page home et inscription qui ne ne marche pas.

References

[1] Documentation de render : https://render.com/docs

[2] Documentation de react : https://react.dev/

[3] LaTeX pour le rapport : https://go.dev/doc/

[4] LaTeX pour le rapport : https://fr.overleaf.com/project

[4] Emailjs pour la confirmation d'email : https://www.emailjs.com/