Rapport de Projet - SpotYnov

Application de gestion de groupes musicaux avec Spotify API

Date de rendu : 14/03/2025

Auteurs : Matt, Thomas et Hugo

Projet réalisé dans le cadre du module « Développement API »

---

1. Introduction

Spotynov est une application web permettant aux utilisateurs de créer et gérer des groupes musicaux, de partager leurs goûts et de synchroniser leurs musiques préférées via l'API Spotify. Le projet repose sur React.js pour le front-end, avec une architecture modulaire et évolutive. L'utilisation de Vite et TypeScript permet d'améliorer la performance et la robustesse du projet.

Objectifs du projet :

- Mise en place d'un système d'authentification via Spotify
- Gestion des groupes musicaux avec des interactions simplifiées
- Synchronisation et gestion des morceaux préférés
- Expérience utilisateur optimisée et fluide

---

2. Respect des Fonctionnalités Définies

Le projet implémente les fonctionnalités requises avec quelques limitations identifiées :

Fonctionnalité	Statut
FT-1 : Création d'utilisateur	Fonctionne
FT-2 : Connexion	Fonctionne
FT-3 : Rejoindre un groupe	Fonctionne
FT-4 : Liaison avec Spotify	Fonctionne
FT-5 : Consultation des groupes et utilisateurs	Fonctionne (récupération du titre en cours ne marche pas)
FT-6 : Analyse des préférences musicales	Fonctionne
FT-7 : Synchronisation musicale	Ne fonctionne pas
FT-8 : Création de playlist personnalisée	Affiche la playlist mais ne la copie pas

Certaines améliorations sont donc à prévoir, notamment pour la récupération du titre en cours et la synchronisation musicale.

3. Architecture du Projet

Le projet est conçu avec une structure modulaire facilitant l'évolutivité et la maintenabilité.

Structure principale du projet :

/src (Dossier principal du projet)

- **components/** : Composants réutilisables (Navbar.tsx, GroupsList.tsx, MemberProfile.tsx, CreateGroup.tsx, JoinGroup.tsx)
- **pages/** : Pages principales (Dashboard.tsx, Login.tsx, Register.tsx, Home.tsx)
- **services/** : Gestion des requêtes et logique métier (spotifyServices.ts, groupServices.ts, authServices.ts)
- **styles/** : Fichiers CSS (Dashboard.css, Auth.css)
- **App.tsx** : Point d'entrée principal de l'application
- **vite.config.ts** : Configuration de Vite pour optimiser le développement

---

4. Fonctionnalités Implémentées

Authentification avec Spotify

Connexion via OAuth 2.0 avec stockage du token

Récupération du profil utilisateur (spotifyUserName, token stocké dans localStorage)

Gestion des Groupes

- Création et suppression de groupes avec useState
- Attribution dynamique des administrateurs en cas de départ (getUserGroup dans Dashboard.tsx)
- Interaction avec l'interface (GroupsList.tsx, CreateGroup.tsx)

Intégration Spotify API

- Extraction et affichage des morceaux likés (fetchLikedTracks dans Dashboard.tsx)
- Génération automatique de playlists (createPlaylist dans MemberProfile.tsx)
- Recommandations musicales (fetchRecommendations basé sur seedTracks)
- Gestion des erreurs API (setError, try-catch)

Expérience Utilisateur

- Interface dynamique avec useEffect et useState
- Messages d'erreur et validation (setMessage, setError)
- Navigation fluide via useNavigate() dans Dashboard.tsx

---

5. Justification des Technologies

Choix de *Vite*

Vite a été sélectionné pour optimiser le temps de développement et les performances du projet. Contrairement à Webpack, Vite permet :

- Un **temps de démarrage quasi-instantané** grâce à son serveur basé sur esbuild
- Un **rechargement en direct (HMR)** optimisé pour une meilleure fluidité
- Une **configuration simplifiée**, réduisant la charge cognitive

Exemple d'utilisation dans vite.config.ts :

```
// https://vite.dev/config/
export default defineConfig({
  plugins: [react()],
})
```

Choix de *React.js*

React.js a été choisi pour sa gestion optimisée du DOM et sa modularité :

- **Utilisation du Virtual DOM**, réduisant les coûts de rendu
- **Organisation en composants réutilisables**, facilitant la maintenance (Navbar.tsx, GroupsList.tsx, etc.)
- **Gestion de l'état via useState et useEffect**, garantissant une interaction fluide

Exemple dans Dashboard.tsx :

```
const [groups, setGroups] = useState<any[]>([]);
```

```
useEffect(() => {
  setGroups(getAllGroups());
}, []);
```

Utilisation de *TypeScript*

L'intégration de TypeScript renforce la robustesse du projet :

- **Typage strict** évitant les erreurs d'exécution
- **Documentation implicite** facilitant la lecture du code
- **Interopérabilité avec les API** via des interfaces (LikedTrack, PlayedTrack dans Dashboard.tsx)

Exemple de typage dans Dashboard.tsx :

```
interface LikedTrack {
  id: string;
  name: string;
  artist: string;
  popularity: number;
  durationMs: number;
  addedAt: string;
}
```

Gestion des Données avec *localStorage*

L'utilisation de localStorage permet une persistance locale sans serveur :

- **Évite un backend complexe pour le MVP**
- **Permet une transition facile** vers une base de données distante
- **Stockage rapide et sécurisé des informations utilisateur**

Exemple d'utilisation dans Dashboard.tsx :

```
const [username, setUsername] = useState<string | null>(localStorage.getItem("username"));
```

Intégration de l'*API Spotify*

L'authentification et l'accès aux données sont gérés via OAuth 2.0. Cela permet :

- Une **connexion sécurisée** et fluide
- Un **accès aux playlists et titres likés**
- Une **gestion efficace des tokens** pour éviter les erreurs d'expiration

Exemple d'authentification dans Dashboard.tsx :

```
const authUrl = `https://accounts.spotify.com/authorize?client_id=${clientId}&redirect_uri=
${encodeURIComponent(redirectUri)}&scope=${encodeURIComponent(scopes)}&response_type=token&show_dialog=true`;
window.location.href = authUrl;
```

6. Problèmes rencontrés et solutions

Problème : **Mauvaise récupération des titres likés**

Cause : Le token expirait fréquemment, générant des erreurs 403.
Solution : Implémentation d'une vérification et d'un rafraîchissement automatique du token.

Problème : **Rechargement nécessaire après la création d'un groupe**

Cause : useState ne se mettait pas à jour immédiatement après modification de localStorage.
Solution : Utilisation d'un useEffect pour détecter les changements et forcer la mise à jour.

Problème : **URI bloquant les requêtes**

Cause : Requêtes mal formées et cache renvoyant des données obsolètes.
Solution : Correction des URIs en respectant le format attendu, encodage des paramètres et invalidation du cache pour éviter les réponses périmées.

7. Conclusion et Améliorations Futures

Bilan du projet

Le projet **Spotynov** a permis de créer une application fonctionnelle offrant une gestion efficace des groupes musicaux. L'intégration avec Spotify permet une expérience enrichissante en facilitant la découverte et le partage musical.

Améliorations possibles

- **Ajout d'un système de chat** entre les membres d'un groupe
- **Amélioration du design et de l'UX**
- **Ajout de statistiques détaillées** sur les habitudes d'écoute
- **Synchronisation des musiques** entre les utilisateurs

Fin du rapport,

Merci pour votre lecture