5️⃣

Cahier des charges technique

Version	Date	Auteur	Validation	Commentaires
V1	28/02/2026	Antonin DRUJON	Équipe : à vérifier	RAS

1. Introduction

Le projet *Le Vinyle* vise à développer une application web permettant aux streamers Twitch d'organiser des sessions musicales collaboratives avec leur communauté. L'application offre la possibilité aux viewers de proposer des morceaux via Spotify, aux modérateurs de gérer les propositions et à l'hôte de superviser l'ensemble de la session en temps réel.

Le cahier des charges fonctionnel a défini les besoins utilisateurs ainsi que les fonctionnalités attendues. Le présent cahier des charges technique a pour objectif de décrire les moyens techniques mis en œuvre pour répondre à ces exigences. Il précise l'architecture du système, les principes d'implémentation, l'intégration des services tiers, ainsi que les choix techniques structurants.

Ce document constitue une référence pour le développement, la maintenance et les évolutions futures du projet. Il ne redéfinit pas les besoins fonctionnels, mais en propose une traduction technique cohérente et structurée.

2. Architecture générale du système

L'application *Le Vinyle* repose sur une architecture web de type client–serveur. Elle est composée d'une interface utilisateur accessible depuis un navigateur, d'un serveur applicatif centralisant la logique métier et d'une base de données assurant la persistance des informations. L'ensemble communique via le protocole HTTPS pour les échanges classiques et via un mécanisme temps réel pour les mises à jour dynamiques.

L'interface utilisateur (frontend) permet aux différents profils — hôte, modérateur et viewer — d'interagir avec la plateforme. Elle gère l'authentification via Twitch, l'affichage des sessions, la proposition de morceaux et la modération. Elle communique avec le serveur au moyen d'une API sécurisée et reçoit les mises à jour en temps réel afin de garantir une expérience fluide et synchronisée pour tous les participants.

Le serveur applicatif (backend) constitue le cœur du système. Il traite les requêtes envoyées par le frontend, applique les règles métier définies dans le cahier des charges fonctionnel et gère les droits d'accès selon le rôle de chaque utilisateur. Il assure également l'intégration avec les API externes, notamment Twitch pour l'authentification et Spotify pour la recherche de morceaux, la création de playlists et l'ajout de titres validés. Le backend est responsable de la diffusion des événements en temps réel, tels que l'ajout d'une proposition ou la validation d'un morceau.

La base de données stocke les informations nécessaires au fonctionnement de l'application : utilisateurs authentifiés, sessions, propositions musicales, décisions de modération et éventuelles

sanctions. Elle garantit la cohérence et la persistance des données pendant la durée d'une session et permet, si nécessaire, leur conservation temporaire.

Le système dépend enfin de services tiers, principalement Twitch et Spotify, dont les API officielles sont utilisées conformément à leurs contraintes techniques et réglementaires. L'architecture retenue permet une séparation claire des responsabilités entre les différentes composantes, favorise la maintenabilité du projet et facilite son évolution future.

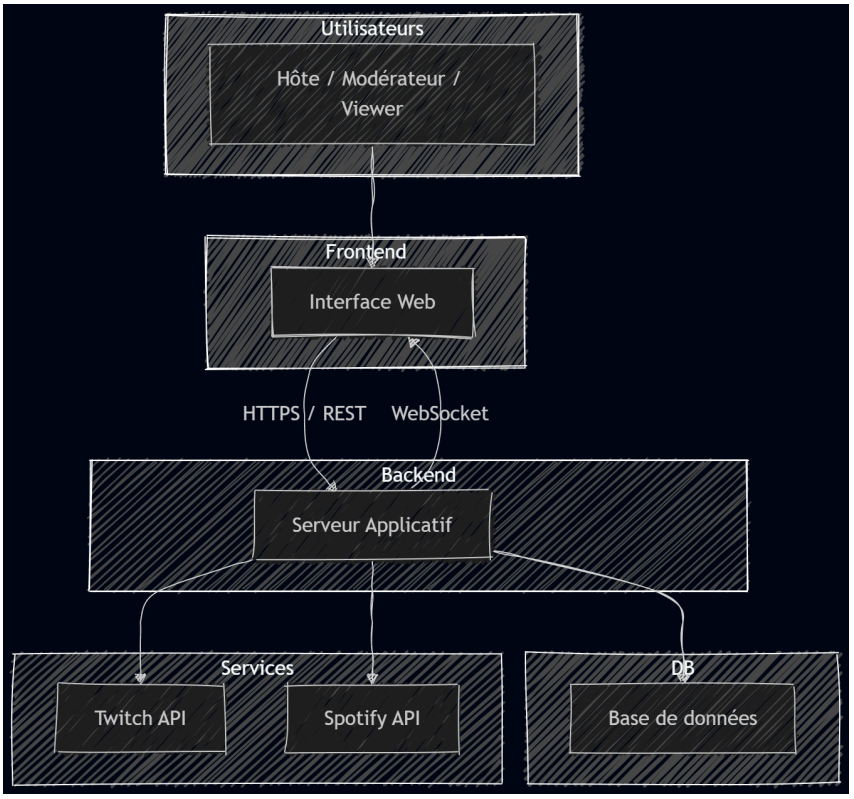


Schéma d'architecture globale – Le Vinyle (avec WebSocket)

3. Choix technologiques

Pour le projet *Le Vinyle*, les choix technologiques ont été définis afin de garantir **performance, évolutivité, maintenabilité et compatibilité avec les API Twitch et Spotify**, tout en assurant une expérience utilisateur fluide et sécurisée.

Frontend

- **Technologie choisie : React.js avec TypeScript**
- **Raisons :**
  - Permet de créer une interface réactive et *componentisée*, adaptée aux différents formats d'écran.
  - Bonne compatibilité avec **Socket.io** pour la communication temps réel.
  - Large communauté et composants prêts à l'emploi.
- **SEO ou « Search Engine Optimization » :**

    - Pour les pages publiques (landing page, documentation), un **prérendu statique (SSG)** peut être utilisé afin d'assurer une indexation correcte par les moteurs de recherche.
    - Les pages dynamiques de l'application (sessions, propositions musicales) resteront en SPA, le SEO n'étant pas critique pour les utilisateurs authentifiés.
- **Framework animation : GSAP ?**

Backend

- **Technologie choisie : Node.js avec Express**
- **Raisons :**
    - Intégration fluide avec React et Socket.io.
    - Gestion simple des API REST et des événements temps réel.
    - Performance adaptée aux sessions simultanées avec de nombreux participants.
- **Modules clés :**
    - **Express.js** : framework backend léger et performant.
    - **Socket.io** : communication bidirectionnelle en temps réel.

Base de données

- **Technologie choisie : PostgreSQL**
- **Raisons :**
    - Base relationnelle robuste, adaptée à la gestion des utilisateurs, sessions, propositions et playlists.
    - Supporte les relations complexes et les transactions.
    - **ORM : Prisma**
    - Simplifie la gestion des modèles de données et assure des requêtes sécurisées.

Intégration de services tiers

- **Spotify API :** recherche de morceaux, création et ajout à playlist.
- **Twitch API :** authentification des utilisateurs et récupération de leurs informations.
- **Authentification :** OAuth2 pour Twitch et Spotify, avec gestion sécurisée des tokens côté backend.

Communication temps réel

- **Technologie choisie : Socket.io (WebSocket)**
- **Raisons :**
    - Diffusion instantanée des événements : nouvelle proposition, validation/refus, sanctions, fin de session.
    - Gestion automatique des reconnexions et des événements simultanés.

Hébergement et déploiement

- **Infrastructure :** serveur cloud (AWS EC2)
- **Raisons :** haute disponibilité, évolutivité et déploiement simple pour Node.js.
- **CI/CD :** GitHub Actions pour automatiser tests et déploiements.

4. Modélisation et gestion des données

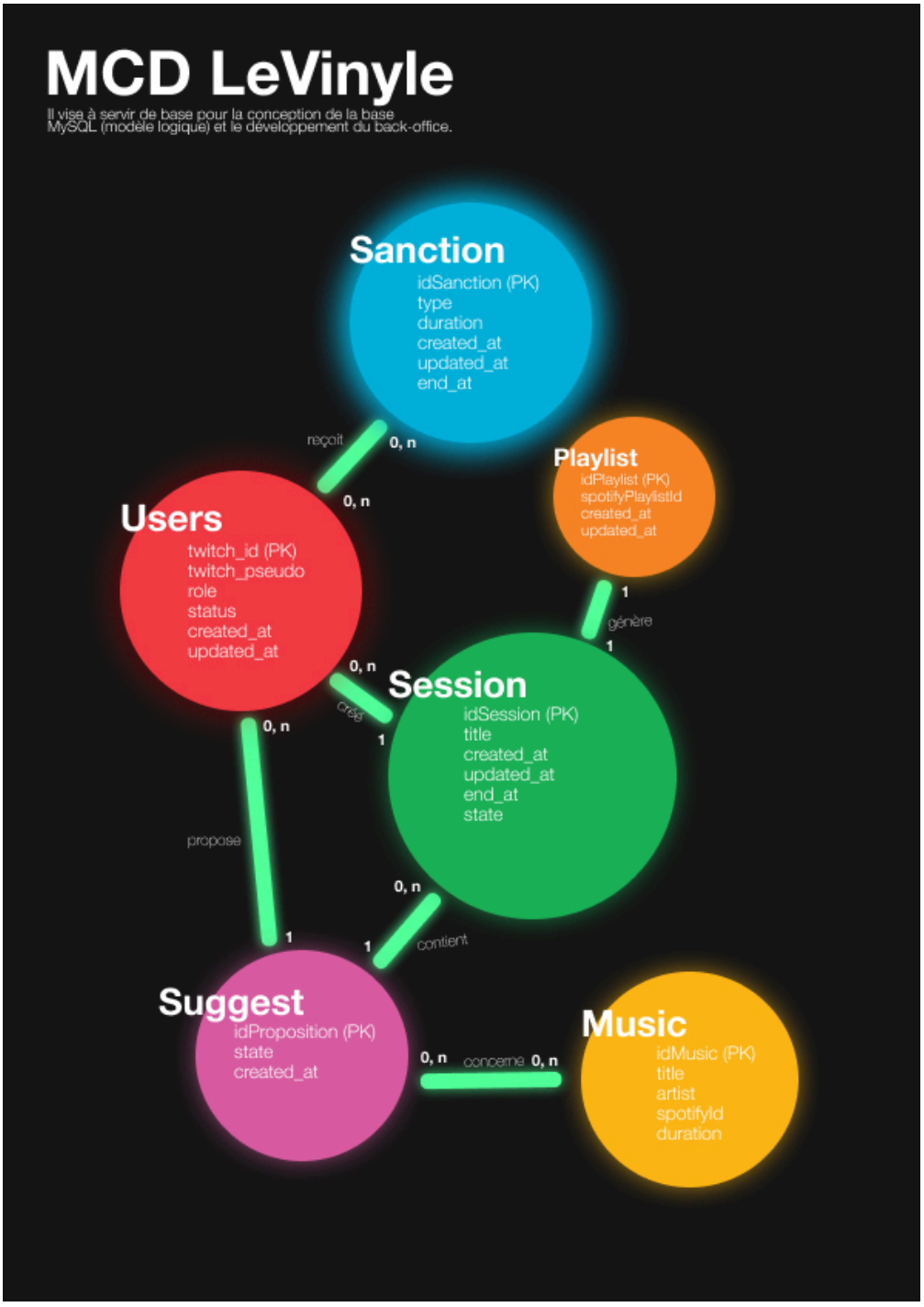
La modélisation des données de l'application *Le Vinyle* repose sur un modèle conceptuel (MCD) structuré autour des entités principales du système : utilisateurs, sessions, propositions musicales, musiques, playlists et sanctions.

Ce modèle constitue la base de la conception de la base de données relationnelle sous PostgreSQL.

Vue d'ensemble du modèle

Le système s'organise autour de la **Session**, qui représente une session musicale active.

Les utilisateurs interagissent dans ces sessions en proposant des morceaux, qui peuvent être acceptés et intégrés à une playlist.



Modèle Conceptuel de Données (MCD)

Description des entités principales

Users

Représente un utilisateur authentifié via Twitch.

Nom de l'attribut	Type de donnée	Contraintes	Description
twitch_id	VARCHAR(50)	PRIMARY KEY	Identifiant unique fourni par Twitch
twitch_pseudo	VARCHAR(100)	NOT NULL	Nom d’affichage Twitch de l’utilisateur

Nom de l'attribut	Type de donnée	Contraintes	Description
status	VARCHAR(20)	NOT NULL DEFAULT 'ACTIVE'	Statut du compte (ACTIVE, BANNED, MUTED)
created_at	TIMESTAMP	NOT NULL DEFAULT NOW()	Date de création du compte
updated_at	TIMESTAMP	NOT NULL DEFAULT NOW()	Date de dernière mise à jour

User_Session

Représente la participation d'un utilisateur à une session avec un rôle spécifique.

Attribut	Type	Contraintes	Description
user_id	VARCHAR(50)	PK, FK → Users(twitch_id)	Utilisateur
session_id	UUID	PK, FK → Session(id_session)	Session
role	VARCHAR(20)	NOT NULL	(HOST, MODERATOR, VIEWER)
joined_at	TIMESTAMP	NOT NULL DEFAULT NOW()	Date d'entrée

Session

Représente une session musicale en cours ou terminée.

Nom de l'attribut	Type de donnée	Contraintes	Description
id_session	UUID	PRIMARY KEY	Identifiant unique de la session
title	VARCHAR(150)	NOT NULL	Titre de la session
state	VARCHAR(20)	NOT NULL	État de la session (ACTIVE, CLOSED)
host_id	VARCHAR(50)	FOREIGN KEY → Users(twitch_id)	Identifiant de l'hôte
created_at	TIMESTAMP	NOT NULL DEFAULT NOW()	Date de création
updated_at	TIMESTAMP	NOT NULL DEFAULT NOW()	Date de mise à jour
end_at	TIMESTAMP	NULL	Date de fin de session

Suggest

Représente une proposition de musique faite par un utilisateur.

Nom de l'attribut	Type de donnée	Contraintes	Description
id_proposition	UUID	PRIMARY KEY	Identifiant unique de la proposition
state	VARCHAR(20)	NOT NULL DEFAULT 'PENDING'	Statut (PENDING, ACCEPTED, REFUSED)
user_id	VARCHAR(50)	FOREIGN KEY → Users(twitch_id)	Utilisateur ayant proposé
session_id	UUID	FOREIGN KEY → Session(id_session)	Session concernée
music_id	UUID	FOREIGN KEY → Music(id_music)	Musique concernée
created_at	TIMESTAMP	NOT NULL DEFAULT NOW()	Date de proposition
decision_at	TIMESTAMP	NULL	Date de validation/refus
justification	TEXT	NULL	Justification en cas de refus

Music

Représente une musique issue de Spotify.

Nom de l'attribut	Type de donnée	Contraintes	Description
id_music	UUID	PRIMARY KEY	Identifiant interne de la musique
spotify_id	VARCHAR(100)	NOT NULL UNIQUE	Identifiant Spotify du morceau
title	VARCHAR(200)	NOT NULL	Titre du morceau
artist	VARCHAR(200)	NOT NULL	Nom de l'artiste
duration	INTEGER	NOT NULL	Durée en secondes
created_at	TIMESTAMP	NOT NULL DEFAULT NOW()	Date d'enregistrement en base

Playlist

Représente la playlist Spotify générée pour une session.

Nom de l'attribut	Type de donnée	Contraintes	Description
id_playlist	UUID	PRIMARY KEY	Identifiant interne de la playlist
spotify_playlist_id	VARCHAR(100)	NOT NULL UNIQUE	Identifiant Spotify de la playlist
session_id	UUID	FOREIGN KEY → Session(id_session)	Session associée
created_at	TIMESTAMP	NOT NULL DEFAULT NOW()	Date de création
updated_at	TIMESTAMP	NOT NULL DEFAULT NOW()	Date de mise à jour

Sanction

Représente une sanction appliquée à un utilisateur.

Nom de l'attribut	Type de donnée	Contraintes	Description
id_sanction	UUID	PRIMARY KEY	Identifiant unique de la sanction
user_id	VARCHAR(50)	FOREIGN KEY → Users(twitch_id)	Utilisateur sanctionné
session_id	UUID	FOREIGN KEY → Session(id_session)	Session concernée
type	VARCHAR(20)	NOT NULL	Type (MUTE, KICK, BAN)
duration	INTEGER	NULL	Durée en secondes (si temporaire)
created_at	TIMESTAMP	NOT NULL DEFAULT NOW()	Date d'application
end_at	TIMESTAMP	NULL	Date de fin automatique (si applicable)

Explication des Relations

- **Users ↔ Session** : Un utilisateur (ayant le rôle HOST) peut créer et gérer plusieurs sessions. En revanche, chaque session possède un seul hôte identifié par `host_id`. La relation est donc de type **1,N** (un hôte → plusieurs sessions).
- **Users ↔ Suggest** : Un utilisateur peut proposer plusieurs musiques au cours d'une session. Chaque proposition est obligatoirement liée à un seul utilisateur. La relation est de type **1,N** (un utilisateur → plusieurs propositions).

- **Session ↔ Suggest** : Une session peut contenir plusieurs propositions musicales. En revanche, chaque proposition appartient à une seule session. La relation est de type **1,N** (une session → plusieurs propositions).
- **Suggest ↔ Music** : Chaque proposition concerne une seule musique. Cependant, une même musique peut être proposée plusieurs fois dans différentes sessions (ou par différents utilisateurs). La relation est de type **1,N** (une musique → plusieurs propositions).
- **Session ↔ Playlist** : Chaque session génère une seule playlist Spotify dédiée. Une playlist est rattachée à une seule session. La relation est donc de type **1,1**.
- **Users ↔ Sanction** : Un utilisateur peut recevoir plusieurs sanctions au cours du temps (mute, kick, ban). Chaque sanction est associée à un seul utilisateur. La relation est de type **1,N** (un utilisateur → plusieurs sanctions).
- **Session ↔ Sanction** : Une sanction est appliquée dans le cadre d'une session spécifique. Une session peut contenir plusieurs sanctions. La relation est donc de type **1,N** (une session → plusieurs sanctions).

5. Spécifications techniques de l'API interne

5.1 Structure générale de l'API

L'API interne de *Le Vinyle* est basée sur un modèle **REST**, garantissant des échanges sécurisés et structurés entre le frontend et le backend. Toutes les communications se font en **JSON** via **HTTPS**, assurant la sécurité et l'intégrité des données. La base URL de l'API est `/api/v1/`.

Les méthodes HTTP principales utilisées sont celles standard du protocole REST : `GET` pour la récupération des données, `POST` pour la création de ressources, `PUT` ou `PATCH` pour la mise à jour et `DELETE` pour la suppression.

L'authentification est gérée via **OAuth2** avec Twitch. Chaque requête nécessitant un utilisateur authentifié est validée côté backend. La gestion des droits d'accès est effectuée selon le rôle de l'utilisateur :

- **HOST** : accès complet sur les sessions qu'il a créées, y compris la validation des propositions et la gestion des participants.
- **MODERATOR** : possibilité de gérer les propositions musicales et d'appliquer des sanctions.
- **VIEWER** : consultation des sessions et possibilité de proposer des morceaux, sans droits de modération.

Cette organisation permet de sécuriser chaque action tout en maintenant une séparation claire des responsabilités selon le rôle de l'utilisateur.

5.2 Principales fonctionnalités couvertes par l'API

L'API interne prend en charge l'ensemble des fonctionnalités nécessaires au fonctionnement de l'application, regroupées autour de quatre axes principaux :

1. **Création et gestion de sessions**

   Les utilisateurs disposant du rôle HOST peuvent créer une session musicale, consulter l'état de celle-ci et la mettre à jour (ACTIVE ou CLOSED). Tous les utilisateurs peuvent rejoindre une session existante afin de participer aux propositions musicales.

2. **Propositions musicales**

Les participants peuvent ajouter de nouvelles propositions de morceaux à une session. Le système vérifie les droits de l'utilisateur et, si l'option de doublon est activée, interdit ou autorise les répétitions de morceaux selon la règle définie.

3. **Validation ou refus des propositions**

   Le HOST ont la possibilité de valider ou refuser les propositions musicales. En cas de refus, ils peuvent fournir une justification afin d'informer l'utilisateur concerné.

4. **Sanctions des utilisateurs**

   Le HOST et les MODERATOR peuvent appliquer des sanctions telles que MUTE, KICK ou BAN. Le backend gère la durée des sanctions temporaires et permet d'appliquer plusieurs sanctions successives si nécessaire.

Chaque fonctionnalité est associée à des **codes HTTP standardisés** et à une gestion des erreurs cohérente, garantissant une interaction sécurisée et fluide entre le frontend et le backend.

5.3 Gestion des erreurs et sécurité

L'API interne de *Le Vinyle* assure une gestion cohérente et standardisée des erreurs afin que le frontend reçoive toujours des informations claires et exploitables. Chaque erreur est renvoyée sous forme de message JSON contenant la description de l'erreur et le code HTTP correspondant. Les principaux codes utilisés sont les suivants :

- `400 Bad Request` : les données fournies sont invalides ou incomplètes.
- `401 Unauthorized` : l'utilisateur n'est pas authentifié ou son token est invalide.
- `403 Forbidden` : l'utilisateur ne dispose pas des droits nécessaires pour effectuer l'action.
- `404 Not Found` : la ressource demandée n'existe pas ou n'est pas accessible.
- `500 Internal Server Error` : une erreur serveur empêche le traitement de la requête.

En parallèle, plusieurs mesures de sécurité garantissent la protection des échanges et des données :

- Toutes les communications entre le frontend et le backend se font exclusivement via **HTTPS**.
- Les tokens OAuth2 sont systématiquement vérifiés côté backend avant toute action.
- Le contrôle des rôles est appliqué pour chaque requête afin de limiter les accès aux fonctionnalités selon le profil de l'utilisateur.
- Des mécanismes de limitation des appels (rate limiting) sont mis en place pour prévenir les abus ou les surcharges du système.

Ces principes garantissent que l'API est à la fois **robuste, sécurisée et fiable**, tout en fournissant aux utilisateurs des messages d'erreur clairs et compréhensibles.

6. Intégration des API externes

L'application *Le Vinyle* utilise des services externes pour enrichir l'expérience musicale et gérer l'authentification des utilisateurs. Les deux principaux services intégrés sont **Twitch** et **Spotify**, via leurs API officielles.

L'authentification des utilisateurs repose sur le protocole **OAuth2**, garantissant un accès sécurisé aux données et aux fonctionnalités. Lorsqu'un utilisateur se connecte via Twitch, le backend récupère un **access token** et un **refresh token**, qui sont stockés côté serveur de manière sécurisée

et jamais exposés au frontend. Le refresh token permet de renouveler automatiquement l'accès en cas d'expiration, assurant ainsi une expérience fluide et continue pour l'utilisateur. De manière similaire, Spotify est utilisé pour rechercher des morceaux, créer des playlists dédiées à chaque session et ajouter les morceaux validés, toujours via des tokens OAuth2 sécurisés.

6.1 Intégration Twitch

L'API Twitch permet de :

- Authentifier les utilisateurs et récupérer leurs informations publiques (pseudo, identifiant Twitch, avatar).
- Vérifier les droits pour la gestion de la session et la modération.
- Respecter les quotas imposés par Twitch afin d'éviter toute surcharge ou blocage du service.

Toutes les requêtes vers Twitch sont sécurisées via HTTPS, et les scopes OAuth utilisés sont limités au strict nécessaire pour assurer la confidentialité des données.

6.2 Intégration Spotify

L'API Spotify est utilisée pour gérer les contenus musicaux proposés par les utilisateurs. Elle permet de :

- Rechercher des morceaux à proposer dans les sessions.
- Créer des playlists dédiées pour chaque session.
- Ajouter ou retirer des titres validés dans les playlists associées.

Le backend veille à respecter les quotas Spotify et les droits d'accès de chaque utilisateur. Les erreurs provenant de l'API (morceau non disponible, token expiré, quota dépassé) sont capturées côté backend et renvoyées au frontend sous forme de messages clairs et compréhensibles, afin de garantir une expérience utilisateur cohérente.

7. Gestion du temps réel (WebSocket)

Pour assurer une expérience utilisateur fluide et interactive, l'application *Le Vinyle* utilise la technologie **WebSocket** via **Socket.io**, en complément de l'API REST. Cette approche permet de diffuser instantanément les événements de chaque session musicale à tous les participants connectés, sans nécessiter de rafraîchissement ou de requêtes répétitives.

Lorsqu'un utilisateur rejoint une session, le frontend ouvre une connexion WebSocket avec le backend. Ce dernier maintient une liste des clients connectés pour chaque session et diffuse uniquement les événements pertinents aux participants concernés. Cette architecture garantit la **synchronisation en temps réel** de l'interface et réduit la charge serveur par rapport à un système de polling classique.

7.1 Types d'événements temps réel

Les événements principaux transmis via WebSocket couvrent toutes les interactions critiques de la session musicale :

- **Nouvelle proposition musicale (** `new_suggestion` **)** : notification de l'ajout d'un nouveau morceau proposé par un participant.
- **Validation ou refus d'une proposition (** `suggestion_accepted` **/** `suggestion_refused` **)** : mise à jour immédiate de l'état de la proposition pour tous les utilisateurs.

- **Sanction d'un utilisateur (`user_sanctioned`)** : annonce qu'un participant a été muté, kické ou banni.
- **Fin de session (`session_closed`)** : signal que la session est terminée et verrouillée pour tous les participants.

Ces événements permettent de refléter en temps réel toutes les décisions prises par l'Hôte ou les Modérateurs, garantissant que tous les utilisateurs voient l'état exact de la session.

7.2 Fonctionnement technique

Chaque action réalisée via l'API REST déclenche, si nécessaire, l'émission d'un événement WebSocket correspondant. Par exemple, lorsqu'un morceau est proposé via l'API, le backend enregistre la proposition dans la base de données puis émet l'événement `new_suggestion` à tous les clients connectés à la session.

Le frontend écoute ces événements et met à jour automatiquement l'interface utilisateur, affichant les nouvelles propositions, les changements de statut et l'ajout de morceaux à la playlist Spotify.

Socket.io gère également les **reconnexions automatiques**. Si un participant perd sa connexion, le backend renvoie l'état complet de la session lors de la reconnexion afin de synchroniser correctement l'interface.

7.3 Avantages pour Le Vinyle

Cette architecture temps réel apporte plusieurs bénéfices :

- **Synchronisation instantanée** de tous les participants pour toutes les actions.
- **Expérience utilisateur fluide et interactive**, sans latence perceptible.
- **Réduction des appels API redondants**, limitant la charge serveur.
- **Extensibilité facile**, permettant d'ajouter de nouveaux événements en temps réel (votes, alertes, statistiques, etc.).

8. Sécurité

La sécurité est un élément central du projet *Le Vinyle*, car l'application gère à la fois des comptes utilisateurs, des sessions en temps réel et l'intégration avec des services externes. L'architecture est conçue pour protéger les données, garantir la confidentialité et contrôler les accès selon les rôles des utilisateurs.

8.1 Authentification et gestion des rôles

L'accès à l'application repose sur **OAuth2** avec Twitch pour authentifier tous les utilisateurs. Les tokens OAuth2 sont vérifiés côté backend à chaque requête afin d'assurer leur validité et leur authenticité.

La gestion des droits d'accès se fait via un **système RBAC (Role-Based Access Control)** :

- **HOST** : contrôle complet sur la session qu'il a créée, gestion des propositions et des participants.
- **MODERATOR** : possibilité de modérer les propositions et d'appliquer des sanctions.
- **VIEWER** : consultation des sessions et proposition de morceaux, sans droits de modération.

Chaque route sensible est protégée par la vérification du rôle correspondant, garantissant que les utilisateurs ne peuvent effectuer que les actions autorisées.

8.2 Protection des communications et des données

Toutes les communications entre le frontend, le backend et les services externes (Twitch, Spotify) sont **chiffrées via HTTPS**, garantissant la confidentialité et l'intégrité des échanges. Les clés API et secrets OAuth2 sont stockés dans des **variables d'environnement sécurisées** côté serveur et ne sont jamais exposés au frontend.

Des mesures supplémentaires assurent la sécurité des opérations :

- **Contrôle des accès** pour toutes les requêtes REST et événements WebSocket.
- **Protection contre les attaques XSS**, notamment dans les formulaires de proposition et dans l'interface temps réel.
- **Limitation des appels API** (rate limiting) pour prévenir les abus ou attaques par saturation.

9. Performance et scalabilité

L'application *Le Vinyle* est conçue pour rester fluide et réactive même avec un grand nombre de participants simultanés. La combinaison de REST API pour les actions ponctuelles et WebSocket pour les mises à jour temps réel permet de limiter les latences et d'optimiser les échanges de données.

Pour garantir la performance :

- Les requêtes vers la base de données sont optimisées et indexées selon les besoins des sessions et des propositions.
- Le backend est capable de gérer plusieurs connexions simultanées grâce à Node.js et Socket.io, assurant une diffusion efficace des événements en temps réel.
- Les mécanismes de limitation et de contrôle des appels API permettent de protéger le système en cas de pic de trafic.

L'architecture retenue offre également une **scalabilité horizontale**, permettant d'ajouter facilement de nouveaux serveurs pour supporter un nombre croissant de sessions ou d'utilisateurs, sans dégrader l'expérience.

10. Déploiement et infrastructure

L'application *Le Vinyle* sera déployée sur une infrastructure cloud afin d'assurer **haute disponibilité, évolutivité et fiabilité**. Le serveur backend Node.js et la base de données PostgreSQL seront hébergés sur des instances cloud (AWS EC2), avec possibilité d'extension horizontale si le nombre d'utilisateurs augmente.

La procédure de déploiement repose sur un **pipeline CI/CD** automatisé via **GitHub Actions**, permettant de tester, construire et déployer automatiquement les nouvelles versions de l'application.

Les variables d'environnement (tokens OAuth, clés API, paramètres de configuration) sont gérées de manière sécurisée, et seules les ressources nécessaires sont exposées au frontend.

Pour la maintenance et la surveillance, le système inclut :

- **Monitoring et logs** des serveurs et des événements applicatifs pour détecter rapidement tout problème.
- **Sauvegardes régulières** de la base de données pour garantir la continuité des données.
- **Gestion des versions** pour suivre l'évolution du code et faciliter le rollback si nécessaire.

Cette infrastructure garantit un **déploiement fiable, sécurisé et évolutif**, tout en permettant un support efficace pour les mises à jour futures et la montée en charge.

11. Maintenance et évolutivité

L'application *Le Vinyle* est conçue pour faciliter la **maintenance et l'évolution** de ses fonctionnalités tout au long de son cycle de vie. L'organisation du code suit des principes modulaires et componentisés, permettant aux développeurs de comprendre rapidement chaque partie de l'application et de corriger efficacement les éventuels bugs. Une documentation interne claire accompagne le code, décrivant les modèles de données, les endpoints de l'API et les flux temps réel, ce qui facilite la prise en main par de nouveaux développeurs.

La procédure de correction des incidents et des bugs repose sur un suivi centralisé des tickets et sur des tests automatisés. Les déploiements des correctifs sont réalisés via le pipeline CI/CD, garantissant la **stabilité et la continuité du service**. Les logs applicatifs et les outils de monitoring permettent d'identifier rapidement les anomalies et de réagir efficacement.

L'architecture choisie offre également une **bonne évolutivité** pour intégrer de nouvelles fonctionnalités et répondre à une augmentation du nombre d'utilisateurs. Parmi les évolutions envisagées pour le futur, on peut citer :

- L'ajout d'un **système de vote** pour les propositions musicales afin de renforcer l'interaction des viewers.
- L'intégration d'**autres plateformes musicales** en complément de Spotify.
- La mise en place de **statistiques avancées** sur les sessions, les propositions et l'activité des utilisateurs.

Ces choix permettent de garantir que l'application reste **robuste, maintenable et prête à évoluer** en fonction des besoins des utilisateurs et des nouvelles fonctionnalités à venir.