

DOSSIER PROJET

MusicSwipe

GARRIGUES Hugo

Concepteur développeur d'applications – RNCP 37873

Année scolaire 2024-2025



Sommaire

1. Introduction Générale	5
1.1. Présentation de projet	5
1.2. Objectifs Globaux.....	5
1.2.1. Objectifs fonctionnels	5
1.2.2. Objectifs techniques	5
1.3. Compétences mises en œuvre	6
2. Contexte et expression des besoins.....	7
2.1. Contexte du projet.....	7
2.2. Expression des besoins	7
2.3. Présentation de l'environnement	8
3. Gestion de projet.....	9
3.1. Méthodologie adoptée	9
3.2. Outils utilisés	10
3.3. Planning prévisionnel	11
3.4. Suivi de l'avancement	12
4. Conception fonctionnelle	12
4.1. Description des parcours utilisateurs	12
4.2. Contraintes fonctionnelles et livrables	13
5. Conception technique.....	14
5.1. Architecture logicielle	14
5.2. Base de données	15
5.2.1. Présentation de la base de données	15
5.3. Maquettes et interfaces	16
6. Développements et réalisations techniques	21
6.1. Environnement technique	21
6.2. Développement Backend	22
6.2.1. Choix techniques et architecture	22
6.2.1.1. Frameworks et langages	22
6.2.1.2. Justification des choix	23
6.2.1.3. Organisation du projet	24
6.2.2. Fonctionnalités implémentées.....	24
6.2.2.1. Authentification et login sécurisée.....	24
6.2.2.2. Gestion des utilisateurs.....	25
6.2.2.3. Gestion des tracks	25
6.2.2.4. Système d'abonnements	25



6.2.2.5.	Gestion des likes	26
6.2.2.6.	Gestions des notations.....	26
6.2.3.	Sécurité backend.....	26
6.2.3.1.	Hashage des mots de passe	26
6.2.3.2.	Validations des entrés	26
6.2.3.3.	Protection des routes	26
6.2.3.4.	Sécurité BDD	26
6.3.	Développement Admin.....	27
6.3.1.	Choix techniques et architecture	27
6.3.1.1.	Frameworks et langages	27
6.3.1.2.	Justification des choix	27
6.3.1.3.	Organisation du projet	28
6.3.2.	Fonctionnalités implémentées.....	28
6.3.2.1.	Page d'authentification.....	28
6.3.2.2.	Dashboard Principal.....	28
6.3.2.3.	Dashboard Utilisateurs.....	29
6.3.2.4.	Dashboard Tracks	29
6.3.2.5.	Page settings	29
6.3.3.	Sécurité admin	30
6.3.3.1.	Authentification admin.....	30
6.3.3.2.	Gestion des autorisations.....	30
6.4.	Développement PWA.....	31
6.4.1.	Choix techniques et architecture	31
6.4.1.1.	Justification des choix	31
6.4.1.2.	Organisation du projet	31
6.4.2.	Fonctionnalités implémentées.....	32
6.4.2.1.	Authentification.....	32
6.4.2.2.	Swipe & flux de découverte	32
6.4.2.3.	Détail d'un morceau	32
6.4.2.4.	Profil & statistiques	32
6.4.2.5.	Recherche	33
6.4.2.6.	Suivre / ne plus suivre.....	33
6.4.2.7.	Paramètres	33
7.	Déploiement et DevOps	33
7.1.	Environnement	33
7.2.	Scripts de déploiement	34



8. Veille technologie.....	35
8.1. Sources et outils de veilles utilisés	35
8.2. Vulnérabilités et correction.....	37
9. Bilan et conclusion.....	38
9.1. Compétences acquises.....	38
9.2. Perspectives d'évolution	38



1. Introduction Générale

1.1. Présentation de projet

MusicSwipe est une application conçue pour aider les utilisateurs à découvrir de nouveaux morceaux de musique tout en rendant la notation de leurs écoutes plus simple et interactive. Le projet inclut également un dashboard administrateur qui permet de gérer les utilisateurs, les morceaux et l'ensemble des données de l'application.

L'idée s'inspire de deux références. D'un côté, Letterboxd qui repose sur un système de notation et de suivi de contenus. De l'autre, Tinder qui a popularisé le geste du swipe pour exprimer une préférence. MusicSwipe reprend ces concepts et les transpose dans l'univers musical afin de proposer un mode de découverte plus engageant où l'utilisateur peut exprimer rapidement ses goûts en interagissant avec les morceaux proposés.

L'objectif général est de créer une expérience moderne, intuitive et sociale. L'application ne se limite pas à recommander des titres en fonction des préférences de chacun, elle intègre aussi une dimension communautaire où il devient possible de suivre d'autres profils et de partager ses avis autour de la musique.

1.2. Objectifs Globaux

1.2.1. Objectifs fonctionnels

Sur le plan fonctionnel, l'idée principale est de rendre la découverte musicale intuitive et interactive. L'utilisateur peut explorer de nouveaux morceaux grâce à une page d'accueil mettant en avant les titres notés par la communauté. Il peut indiquer facilement ses préférences en aimant ou non un morceau et enrichir son expérience grâce à des fonctionnalités sociales comme le suivi de profils ou les commentaires. L'ambition est de rendre la découverte et la notation musicales plus ludiques tout en créant un espace communautaire où les interactions renforcent l'engagement. Le but est de construire une communauté vivante et active autour de la musique.

1.2.2. Objectifs techniques

Sur le plan technique, MusicSwipe repose sur une architecture simple mais robuste, sécurisée et adaptée aux standards actuels du développement web. Le backend a été conçu pour assurer la fiabilité et la sécurité des données à travers une API REST structurée et protégée. Le



dashboard administrateur propose une interface claire et efficace pour superviser l'ensemble des données. La Progressive Web App, destinée aux utilisateurs, offre une expérience fluide, responsive et mobile-first qui s'appuie sur les mécanismes attendus des applications modernes.

Un point central du projet est l'intégration de l'API Spotify. Reliée aux comptes des utilisateurs, elle permet d'alimenter automatiquement la base de données et de donner accès à un catalogue riche. Elle assure également la connexion avec un service reconnu tout en respectant les contraintes de sécurité et de performance.

1.3. Compétences mises en œuvre

La réalisation de MusicSwipe m'a permis de mobiliser un ensemble de compétences couvrant l'ensemble du cycle de développement d'une application moderne. J'ai travaillé en amont sur l'analyse des besoins et la formalisation d'un cahier des charges afin de traduire les attentes fonctionnelles en spécifications claires. Cette étape a donné lieu à la création de parcours utilisateurs, de diagrammes et de maquettes qui ont permis de poser les bases de l'architecture logicielle.

Côté serveur, j'ai acquis de l'expérience sur la conception de bases de données relationnelles, le développement de composants métier et la sécurisation des accès aux ressources. La mise en place d'un système d'authentification adapté a également fait partie des compétences développées. Du côté utilisateur, j'ai travaillé sur la conception d'interfaces responsives, sur la création de composants réutilisables et sur la gestion des échanges avec le backend.

En parallèle, j'ai renforcé mes compétences en gestion de projet grâce à l'adoption d'une méthodologie agile adaptée à un travail individuel. J'ai utilisé un backlog pour organiser les tâches, des outils collaboratifs pour garder une trace et une planification claire pour structurer la progression. Cela m'a permis de garder une organisation constante et de respecter les contraintes de temps et de qualité.

Ces éléments ont permis de garantir la fiabilité de l'application et de poser des bases solides pour ses futures évolutions.



2. Contexte et expression des besoins

2.1. Contexte du projet

MusicSwipe est né d'une observation simple que j'ai pu faire pour moi-même, mais aussi en discutant avec d'autres et en regardant les retours sur les réseaux sociaux. Les applications qui permettent de noter de la musique existent déjà, mais elles sont souvent vieillissantes et peu attractives. Pour évaluer une grande quantité de morceaux, il faut en général passer par un processus long et fastidieux où chaque titre doit être noté un par un. En plus de cette lourdeur, il ne s'agit pas vraiment de communautés actives et dynamiques comme on peut en trouver dans d'autres domaines. Letterboxd a réussi à créer cela pour le cinéma, mais côté musique, des plateformes comme Youratemusic ou Musicboard n'ont pas eu le même impact. Elles manquent de dynamisme et ne semblent plus être maintenues.

Dans le même temps, l'offre musicale ne cesse de s'élargir, ce qui rend parfois compliqué le simple fait de savoir quoi écouter. MusicSwipe n'a pas pour but de remplacer les algorithmes de recommandation, mais d'apporter un angle différent, centré sur la communauté. Le principe est de pouvoir consulter les notes et avis laissés par d'autres, de voir ce qui plaît ou déplaît réellement aux utilisateurs et de découvrir de nouveaux morceaux grâce aux retours de la communauté.

C'est dans ce contexte qu'a émergé l'idée de MusicSwipe, une application qui reprend des concepts qui ont déjà prouvé leur efficacité ailleurs. La notation inspirée de Letterboxd et le swipe popularisé par Tinder sont ici transposés à l'univers de la musique. L'objectif est de rendre la découverte musicale plus interactive, plus communautaire et surtout plus engageante.

2.2. Expression des besoins

Les besoins identifiés se répartissent en deux dimensions principales, à savoir l'expérience utilisateur et la gestion des données.

Du côté des utilisateurs, il est indispensable que l'application reste simple et agréable à utiliser. Le swipe doit permettre une notation rapide, fluide et moins contraignante que sur les autres plateformes. Il faut aussi offrir la possibilité de suivre d'autres profils, de consulter leurs notes



et leurs commentaires afin de favoriser les interactions et de créer une vraie communauté. L'aspect social est donc central, car il donne du sens à l'ensemble et encourage l'implication.

Du côté de l'administration, il faut un outil fiable pour gérer toutes les données. Cela implique la possibilité de créer, modifier et supprimer des utilisateurs ou des morceaux, mais aussi de superviser l'activité globale. Des statistiques doivent être disponibles pour suivre l'évolution de la communauté et s'assurer de la qualité du service.

Enfin, l'application doit être pensée pour répondre à des besoins de sécurité, de performance et d'évolutivité. Les données doivent être protégées, l'expérience utilisateur doit rester fluide quel que soit l'appareil, et la conception doit être prête à accueillir de nouvelles fonctionnalités sans tout remettre en question.

2.3. Présentation de l'environnement

Le projet a été mené dans un cadre technique et organisationnel conçu pour permettre un développement complet. Sur le plan humain, il s'agit d'un projet individuel mené dans le cadre de ma formation. Cela m'a permis de travailler de manière autonome, tout en appliquant les bonnes pratiques apprises en cours et en alternance. Le fait de conduire seul ce projet m'a placé dans une position proche de celle d'un tech lead, en devant gérer toutes les dimensions de bout en bout.

Sur le plan technique, j'ai travaillé principalement sur un ordinateur Windows équipé de WSL afin d'avoir un environnement plus proche des standards du développement backend. Le matériel n'étant pas tout récent, il m'est arrivé d'être ralenti dans certaines étapes, mais cela ne m'a pas empêché d'avancer. J'ai également utilisé un iPhone sous iOS pour tester la Progressive Web App dans des conditions réelles et vérifier que l'expérience mobile-first répondait bien aux attentes.

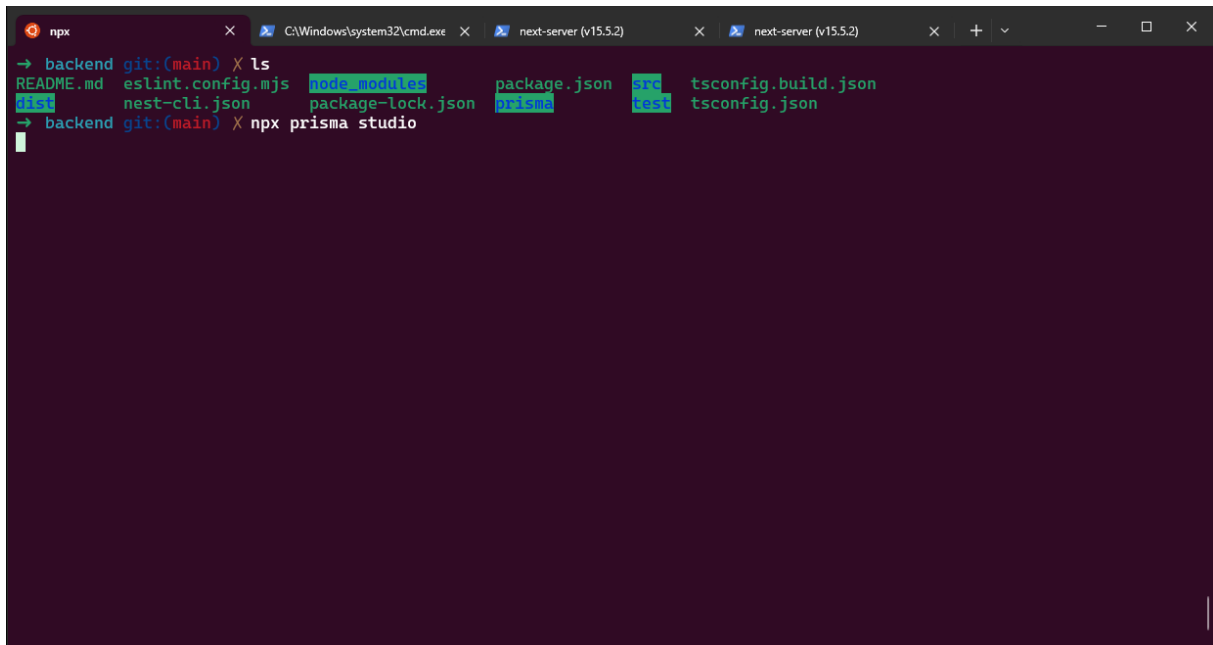


Image 1 : Terminal wsl Ubuntu

3. Gestion de projet

3.1. Méthodologie adoptée

Pour gérer le projet MusicSwipe, j'ai choisi une méthodologie simple inspirée du Kanban. Comme je travaillais seul, il n'était pas nécessaire de mettre en place des rituels d'équipe ou des sprints fixes. Le flux continu s'est imposé naturellement, car il me permettait de visualiser en permanence l'avancement et de progresser étape par étape sans alourdir l'organisation.

J'ai utilisé l'outil ClickUp comme support principal. Chaque fonctionnalité, choix technique ou étape de conception était transformé en tâche et placée dans un tableau avec plusieurs colonnes correspondant à son état d'avancement. Cela allait de "à faire" à "terminé" en passant par "en cours", "bloqué" ou "en production". Cette vision claire m'a permis de savoir rapidement où en était le projet et de repérer immédiatement les points bloquants.

Même sans daily meetings ou rétrospectives, cette approche proche de l'agilité m'a aidé à garder une discipline de travail régulière. Je me fixais de petites étapes à atteindre, je déplaçais les tâches validées et j'avais ainsi une motivation constante en voyant le tableau évoluer. Cette méthode m'a donné une vraie capacité d'auto-organisation et m'a permis de rester concentré sur l'essentiel.



3.2. Outils utilisés

Plusieurs outils ont accompagné le développement du projet. ClickUp servait pour la planification et le suivi. Git et GitHub assuraient la gestion des versions et le suivi du code, avec une organisation en branches séparées selon les fonctionnalités, ce qui permettait de tester avant d'intégrer sur la branche principale. J'ai respecté les conventions de nommage, ce qui m'a habitué aux bonnes pratiques de versioning utilisées en entreprise.

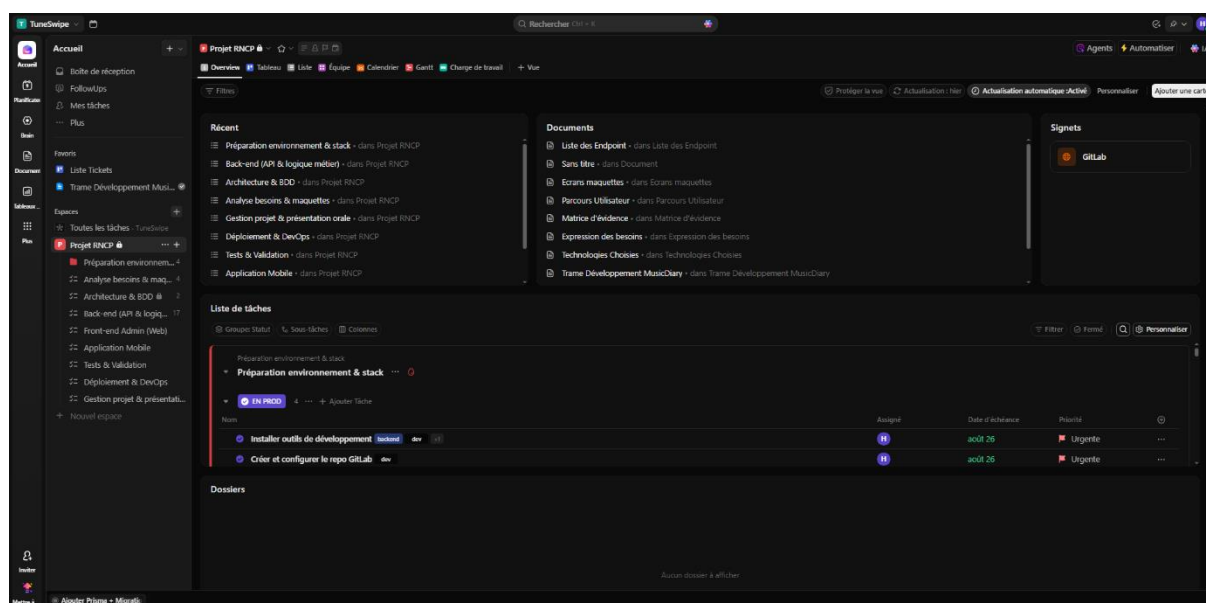


Image 2 : Présentation dashboard ClickUp

Pour la conception, j'ai utilisé dbdiagram.io pour modéliser la base de données et Figma pour réaliser les maquettes. Ces deux outils ont facilité la transition entre la phase de conception et le développement. Visual Studio Code a été mon éditeur principal, avec ses extensions pour



Git, la base de données et le débogage. Enfin, j'ai utilisé mon iPhone pour tester la PWA et vérifier son comportement en conditions réelles.

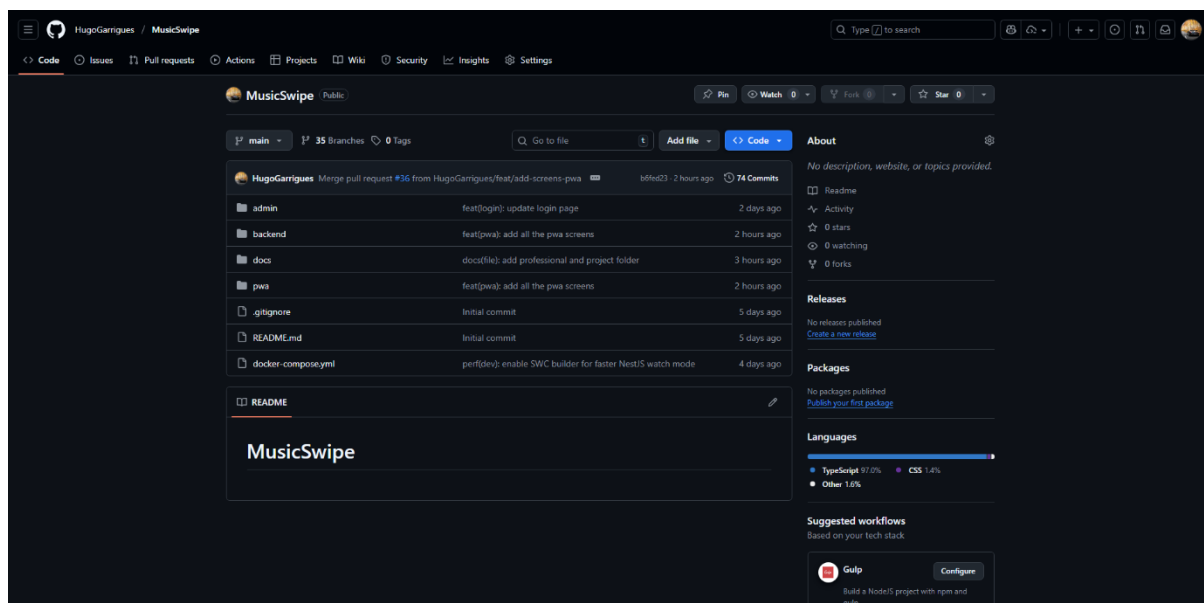


Image 3 : Github de MusicSwipe

Tout au long du développement, j'ai aussi rédigé une documentation dans Word pour garder une trace et organiser mes idées en parallèle du tableau de tâches. L'ensemble de ces outils a constitué un environnement cohérent qui couvrait toutes les étapes, de la conception à la mise en test.

3.3. Planning prévisionnel

Dès le lancement du projet, j'ai défini un planning prévisionnel afin de donner un ordre logique aux différentes étapes. L'idée n'était pas de figer le développement dans un calendrier strict, mais de disposer d'une feuille de route claire.

Le projet a été découpé en plusieurs grandes phases. D'abord la mise en place de l'environnement de développement, puis la conception de l'architecture et de la base de données. Ensuite le développement du backend et de l'API, suivi par la création du dashboard administrateur. Enfin, la réalisation de la PWA est venue clôturer le tout.

Ce planning m'a permis d'anticiper les priorités. J'ai par exemple donné la priorité à l'authentification et à la sécurité des données avant de travailler sur des éléments visuels ou secondaires. Même si certains ajustements ont été nécessaires en cours de route, cette vision d'ensemble m'a aidé à rester productif et à avancer sans me disperser.



3.4. Suivi de l'avancement

Le suivi du projet reposait sur la mise à jour quotidienne du tableau ClickUp. Chaque fin de session de travail était l'occasion de déplacer les tâches vers la colonne correspondante, ce qui me donnait une vue immédiate sur ce qui restait à accomplir.

Quand je rencontrais une difficulté technique, je déplaçais la tâche concernée dans la colonne bloquée et je continuais sur un autre point en attendant de trouver une solution. Cela m'a permis de ne pas perdre de temps et de garder une progression constante.

L'utilisation des niveaux de priorité m'a également aidé à m'organiser. Je pouvais mettre en attente certaines optimisations pour me concentrer sur les fonctionnalités critiques. Même si le projet était individuel, cette organisation proche d'un suivi agile m'a permis d'avancer avec régularité et de m'assurer que chaque fonctionnalité essentielle était bien validée avant de passer à la suivante.

4. Conception fonctionnelle

4.1. Description des parcours utilisateurs

Pour répondre aux besoins identifiés, j'ai défini deux parcours principaux, celui de l'utilisateur et celui de l'administrateur. Ils décrivent la manière dont chacun interagit avec l'application et permettent de comprendre les fonctionnalités mises à disposition selon le rôle.

Le parcours utilisateur commence par l'inscription et la connexion. L'utilisateur peut créer un compte avec une adresse e-mail ou via une solution OAuth comme Spotify ou Deezer. L'accès est sécurisé grâce à des jetons JWT, ce qui garantit la protection des données et des sessions. Une fois connecté, il arrive sur la page d'accueil qui constitue le cœur de l'application. Cette page présente un flux de titres, d'albums ou d'artistes proposés aléatoirement ou en fonction des écoutes récentes. L'interface est pensée mobile-first et optimisée pour le tactile. Le swipe à droite correspond à un like ou à une note positive, celui à gauche à un dislike ou une note faible. Il peut aussi faire défiler le contenu pour passer au morceau suivant.

Lorsqu'il veut plus d'informations, l'utilisateur peut consulter une fiche détaillée sur un titre, un artiste ou un album. Il y retrouve la note moyenne de la communauté, ses propres notations, ses avis et des informations complémentaires comme la date de sortie, le genre ou le nom de l'artiste. L'utilisateur dispose aussi d'un profil personnel regroupant son historique



de notations et des statistiques comme sa moyenne de notes, ses genres favoris ou ses artistes les plus écoutés. Enfin, un moteur de recherche intégré permet de trouver rapidement un morceau, un artiste ou un album.

Le parcours administrateur reprend une structure similaire mais adaptée à son rôle. Après s'être connecté, l'administrateur accède à un dashboard qui centralise les informations clés. Cette interface lui donne une vue synthétique sur l'activité globale avec des statistiques comme le nombre d'utilisateurs, le volume de swipes et de notations ou encore les logs récents. L'administrateur peut gérer les utilisateurs, superviser leurs activités, désactiver ou supprimer des comptes et contrôler les logs liés aux comportements suspects. Il peut également consulter les fiches des morceaux avec des statistiques plus poussées et utiliser un moteur de recherche pour naviguer efficacement dans les données de l'application.

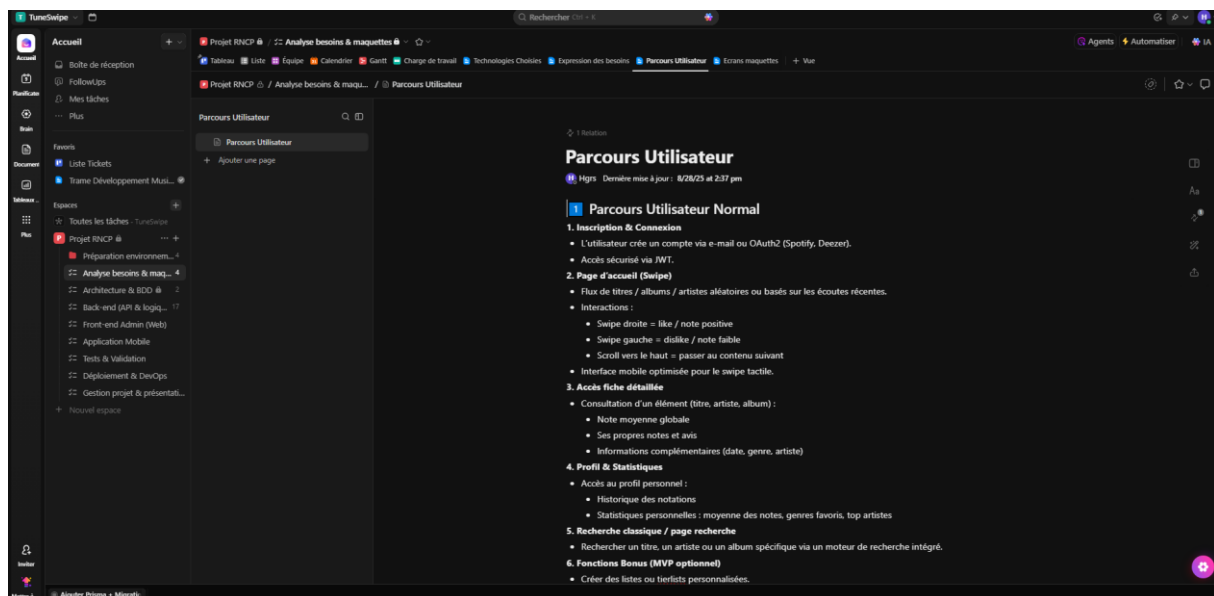


Image 4 : Document dans music swipe sur le parcours utilisateur

4.2. Contraintes fonctionnelles et livrables

La conception de MusicSwipe a été guidée par plusieurs contraintes fonctionnelles qui ont orienté le développement et assuré la qualité du projet. L'application devait être accessible sur différents supports, avec une priorité donnée à l'expérience mobile. Le choix d'une Progressive Web App s'est donc imposé, car elle permet une navigation fluide, responsive et adaptée au tactile. L'interface devait rester claire et intuitive pour simplifier les parcours utilisateurs.



La sécurité constituait une contrainte essentielle. Il fallait protéger les données avec une authentification sécurisée, une gestion stricte des sessions et un contrôle des accès différenciés entre utilisateurs classiques et administrateurs. La cohérence des données devait être garantie, en particulier pour les notations et les interactions sociales.

La performance représentait un autre enjeu. L'utilisateur ne devait pas subir de ralentissements, que ce soit lors du swipe, de la consultation d'une fiche ou d'une recherche. Cela a nécessité une gestion optimisée des appels API et une base de données bien structurée.

Enfin, le projet devait respecter une logique communautaire. L'application ne devait pas seulement permettre de noter ou découvrir de la musique, mais aussi de créer des interactions entre utilisateurs afin de renforcer l'implication.

À la fin du développement, plusieurs livrables venaient attester de la réalisation. L'application elle-même, composée du backend, du dashboard admin et de la PWA utilisateur, constituait le livrable principal. En complément, la base de données a été modélisée et documentée, des maquettes ont été réalisées pour cadrer le développement des interfaces et une documentation de projet a été rédigée pour expliquer les choix de conception et les étapes techniques. Ces livrables démontrent la correspondance entre les besoins exprimés et la solution développée.

5. Conception technique

5.1. Architecture logicielle

L'architecture de MusicSwipe repose sur trois grandes parties qui fonctionnent ensemble tout en ayant des rôles bien distincts. Le backend constitue le socle technique et expose une API REST sécurisée. Cette API regroupe l'ensemble des fonctionnalités métier comme la gestion des utilisateurs, l'authentification, l'intégration avec Spotify, la gestion des morceaux, des notations, des commentaires et du système de suivi. Elle est responsable de la persistance des données et de la sécurité des échanges.

Le dashboard administrateur est une application web connectée directement à cette API. Il permet aux administrateurs de superviser l'activité, de gérer les utilisateurs, de contrôler le catalogue musical, de modérer les notations et les commentaires et de consulter des



statistiques globales. Sa conception met l'accent sur la clarté et l'efficacité pour fournir un outil de gestion fiable et simple à utiliser.

La Progressive Web App est destinée aux utilisateurs. Elle est pensée mobile-first et permet de s'inscrire, de se connecter, de découvrir des morceaux via le système de swipe, de donner des notes, de laisser des avis et de suivre d'autres profils. Elle communique avec le backend pour envoyer et récupérer les données, tout en intégrant les mécanismes spécifiques aux PWA comme l'installation sur smartphone et la possibilité d'une utilisation partielle hors ligne.

Cette architecture modulaire, organisée autour de l'API REST, garantit une indépendance entre les différentes parties. Chaque couche peut évoluer séparément sans casser l'ensemble. Elle répond aussi aux enjeux de sécurité, de scalabilité et de maintenabilité, ce qui assure la pérennité du projet et la possibilité d'intégrer facilement de nouvelles fonctionnalités.

5.2. Base de données

5.2.1. Présentation de la base de données

La base de données de MusicSwipe est organisée autour de plusieurs tables principales, qui représentent les entités clés de l'application.

User : stocke les informations de chaque utilisateur, comme son e-mail, son pseudonyme, son mot de passe haché, son avatar et son rôle (utilisateur ou administrateur).

UserOAuth : gère les connexions via des fournisseurs externes (comme Spotify), en stockant les identifiants et tokens nécessaires.

Track : correspond aux morceaux musicaux disponibles dans l'application, avec leurs métadonnées (titre, artiste, album, durée, extrait audio).

Rating : enregistre les notes attribuées par les utilisateurs aux morceaux, avec une contrainte garantissant qu'un utilisateur ne peut noter un même morceau qu'une seule fois.

Like : conserve les interactions rapides issues du système de swipe (like ou dislike) et lie chaque utilisateur aux morceaux concernés.

Comment : stocke les commentaires laissés par les utilisateurs sur les morceaux, avec la date et le contenu.



Follow : permet de gérer les relations sociales entre utilisateurs, en distinguant le suiveur et l'utilisateur suivi .

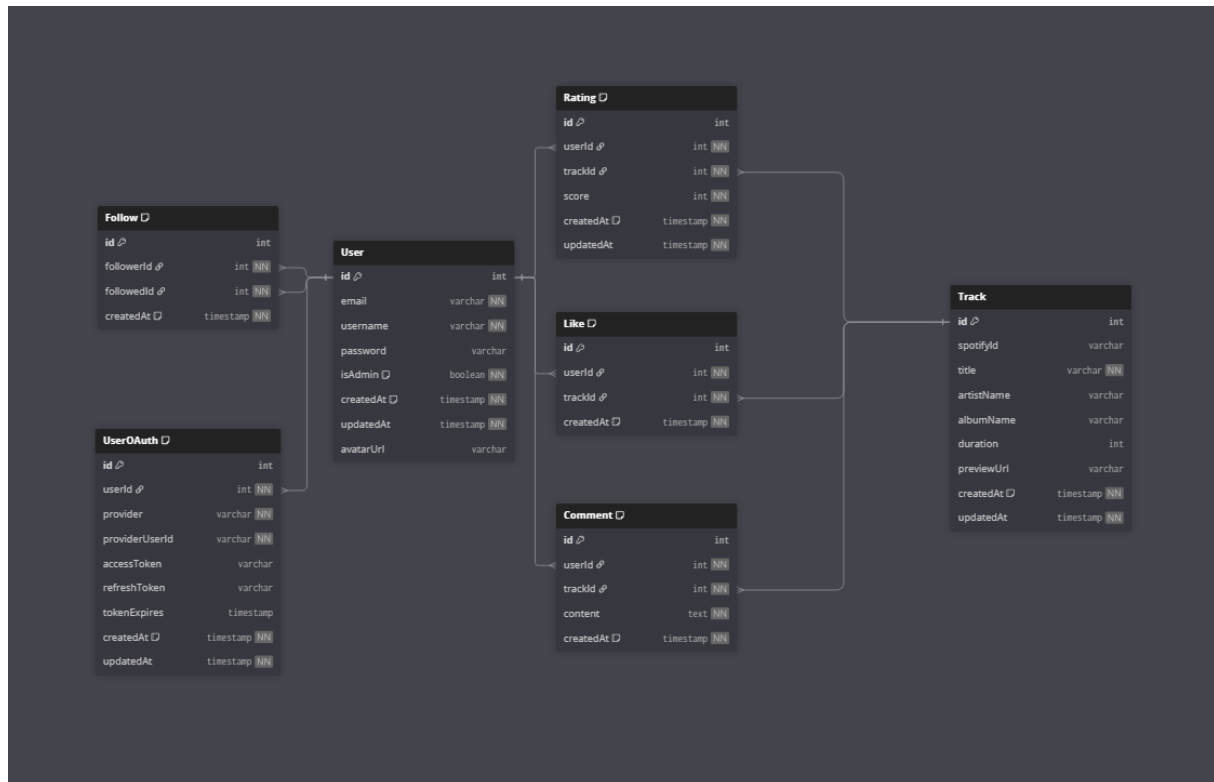


Image 5 : Modèle de base de données MusicSwipe (dbdiagram.io)

5.3. Maquettes et interfaces

La conception des maquettes a constitué une étape essentielle dans le développement de MusicSwipe. Elle a permis de définir en amont l'apparence visuelle de l'application, d'anticiper l'ergonomie des parcours utilisateurs et de valider les choix graphiques avant de passer au code. Les maquettes offrent une vision claire et concrète du produit final, tout en facilitant la cohérence entre la partie utilisateur et la partie administrateur.

La démarche adoptée a été mobile-first, puisque l'application est pensée prioritairement pour une utilisation sur smartphone. Les écrans mobiles ont donc été les premiers réalisés, avec une attention particulière portée à la simplicité, à la fluidité de navigation et à l'optimisation du swipe tactile. Une version desktop de la PWA a également été conçue afin d'assurer une expérience adaptée aux ordinateurs.

En parallèle, un ensemble de maquettes spécifiques a été réalisé pour le dashboard administrateur. Celui-ci se distingue par une interface sobre et fonctionnelle, centrée sur la gestion des utilisateurs, des morceaux et des statistiques globales. L'objectif était de fournir



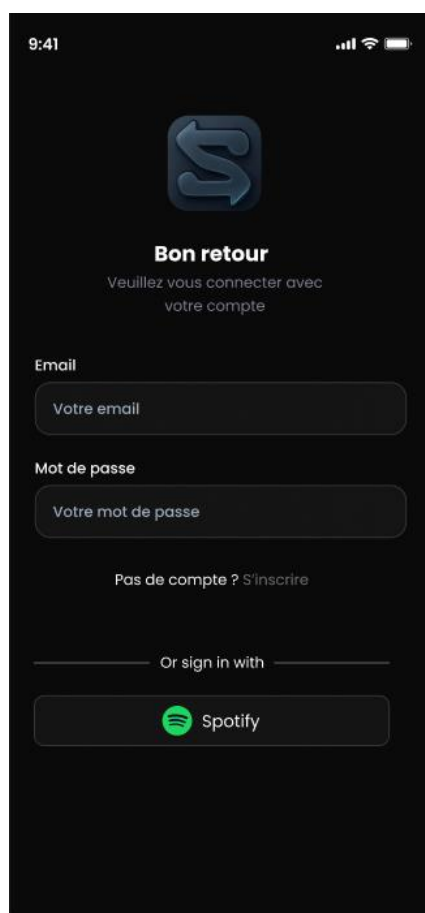
aux administrateurs un outil complet et intuitif, leur permettant de superviser efficacement l'activité de la plateforme.

Ces maquettes constituent un support visuel indispensable. Elles ont servi à définir les interfaces avant le développement et à garantir une expérience homogène et cohérente sur l'ensemble des supports et des rôles.

Présentation des écrans – PWA mobile

Login (01_login) : écran de connexion avec email/mot de passe ou via OAuth (Spotify). Conçu pour être rapide et simple, il constitue le premier point de contact avec l'application.

Register (02_register) : écran d'inscription où l'utilisateur crée un compte en renseignant ses informations de base. L'objectif est de rendre l'inscription accessible en quelques étapes seulement.

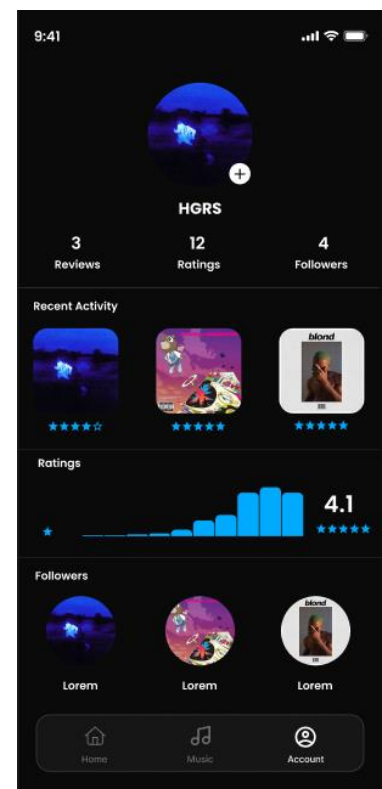
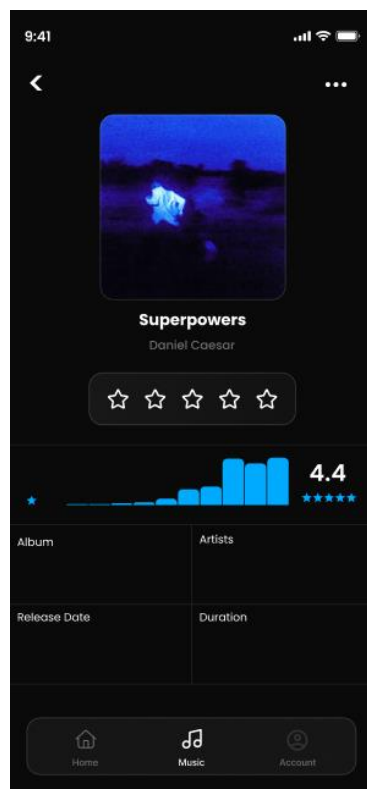




Swipe : cœur de l'application. Les morceaux apparaissent sous forme de cartes, et l'utilisateur interagit en swipant à droite pour liker ou à gauche pour passer. L'expérience tactile est optimisée pour rendre l'utilisation fluide et ludique.

Détail (detail) : page présentant toutes les informations liées à un morceau : titre, artiste, album, durée, extrait audio, moyenne des notes de la communauté et possibilité de commenter.

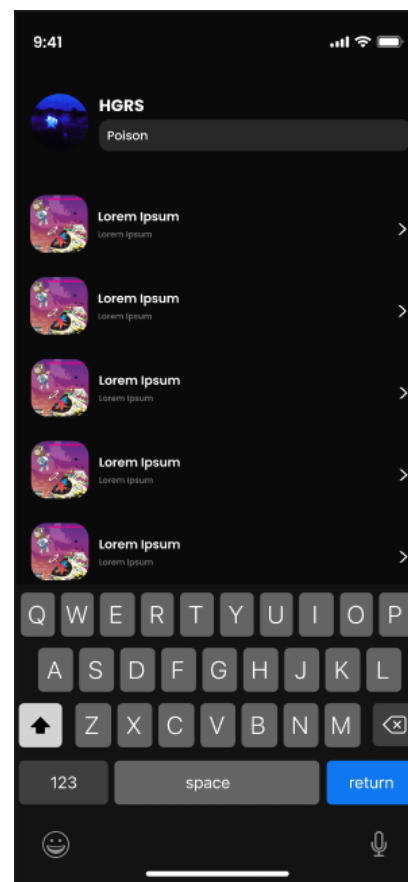
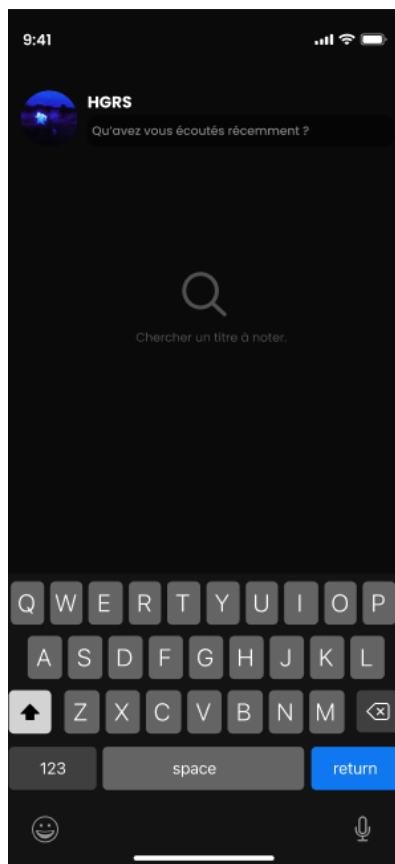
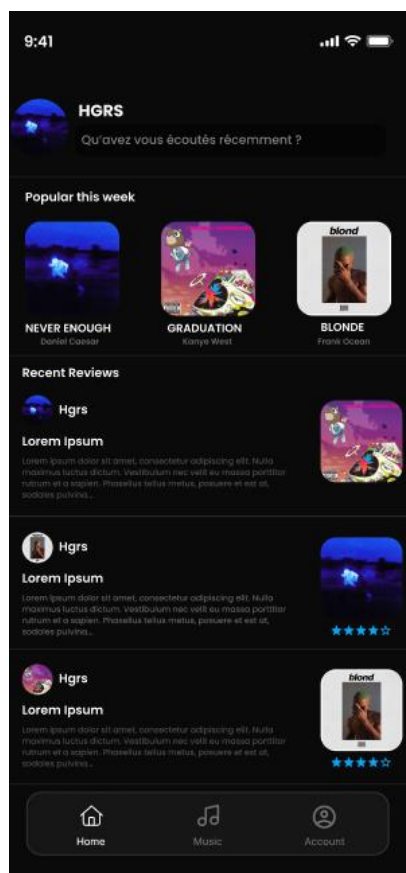
Profil (profile) : espace utilisateur regroupant ses informations personnelles, ses notations passées, ainsi que ses statistiques (moyenne des notes, genres favoris, artistes les plus appréciés).





Home : point d'entrée global de l'application. Il met en avant les morceaux populaires ou tendances, et donne accès aux principales fonctionnalités.

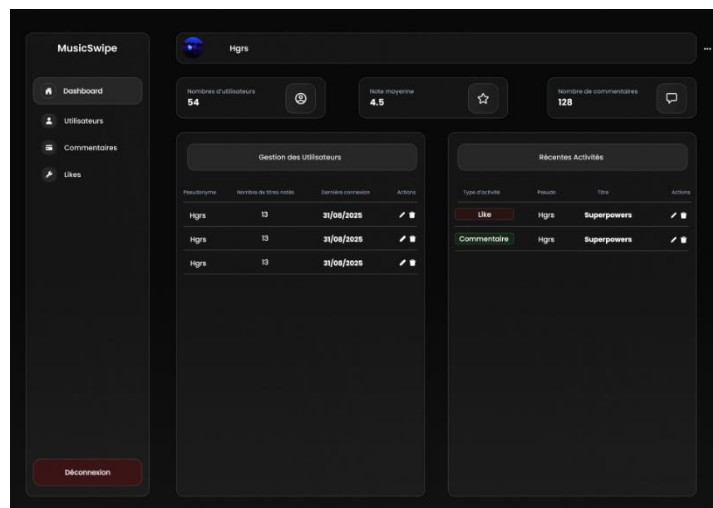
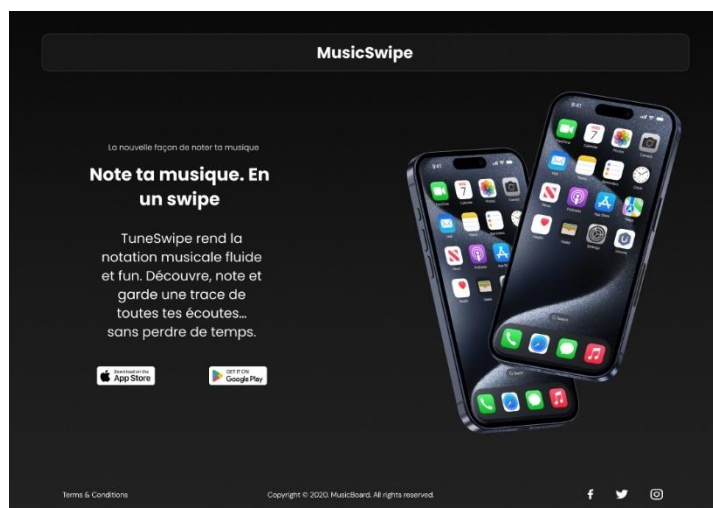
Search & Search 2 : deux versions de la recherche. La première propose un champ simple et une liste rapide de résultats. La seconde offre une présentation plus détaillée, avec des filtres et des catégories pour affiner les recherches.





Présentation des écrans – Desktop utilisateur

Page d'installation : lorsqu'un utilisateur accède à MusicSwipe depuis un ordinateur, il est invité à installer la PWA. Cette page explique la procédure et met en avant les avantages de l'installation (accès direct, mode hors ligne, meilleure ergonomie).



Présentation des écrans – Dashboard administrateur

Login admin : page de connexion spécifique aux administrateurs, avec un contrôle d'accès renforcé.

Dashboard principal : tableau de bord offrant une vue synthétique des statistiques globales : nombre d'utilisateurs inscrits, nombre de morceaux, volume de swipes/notations et activité récente.

Utilisateurs : page de gestion des comptes, permettant de consulter la liste des utilisateurs, leurs activités, et d'effectuer des actions comme désactiver ou supprimer un compte.

Titres : section dédiée au catalogue musical. L'administrateur peut ajouter, modifier ou supprimer des morceaux, et consulter leurs informations détaillées.

Paramètres : espace de configuration permettant d'ajuster les réglages généraux de l'application, notamment en matière de sécurité et d'administration.



6. Développements et réalisations techniques

6.1. Environnement technique

Le développement de MusicSwipe s'est déroulé dans un environnement complet pensé pour couvrir toutes les étapes, de la conception au test. J'ai travaillé principalement sur un ordinateur Windows, en m'appuyant sur WSL afin de bénéficier d'un environnement plus proche des standards du développement backend. La machine n'étant pas toute récente, il m'est arrivé d'être ralenti dans certaines étapes, mais cela ne m'a pas empêché d'avancer. Pour les tests côté utilisateur, j'ai utilisé mon iPhone, ce qui m'a permis de vérifier en conditions réelles que la PWA répondait bien à une logique mobile-first.

L'éditeur Visual Studio Code a été au centre du développement. Ses extensions m'ont facilité la gestion du code, le suivi Git, le débogage et même la communication avec la base de données. Pour la modélisation j'ai utilisé dbdiagram.io, qui m'a permis de concevoir rapidement le modèle relationnel, et pour les maquettes Figma, afin d'avoir une vision claire des interfaces avant de coder.

La gestion des versions s'est faite avec Git et GitHub. J'ai adopté une organisation en branches distinctes selon les fonctionnalités, en respectant les conventions de commits. Cela m'a permis de garder une trace propre de chaque évolution et de simuler un fonctionnement proche de celui que l'on retrouve en entreprise, même si je travaillais seul. En parallèle, j'ai utilisé ClickUp pour gérer mes tâches et suivre ma progression. Enfin, PostgreSQL a servi de base relationnelle et Prisma d'ORM pour assurer un lien clair et typé entre la base et le code backend. L'intégration de l'API Spotify est venue compléter cet environnement en apportant les données nécessaires pour alimenter l'application.

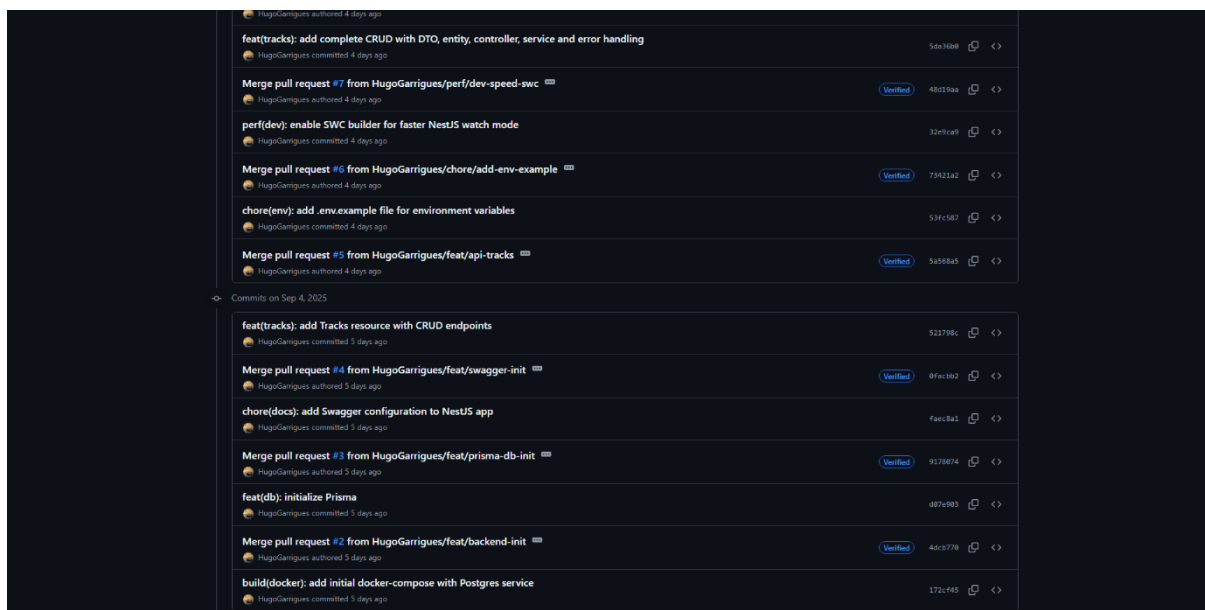


Image 6 : Suivi des conventionnal commits

6.2. Développement Backend

6.2.1. Choix techniques et architecture

6.2.1.1. Frameworks et langages

Le backend repose sur NestJS en TypeScript. Cette base m'a donné une structure modulaire et lisible, adaptée à une API REST sécurisée. J'ai relié l'application à PostgreSQL via Prisma, ce qui simplifie les accès aux données et limite les erreurs grâce au typage fort. PostgreSQL gère sans difficulté les relations entre utilisateurs, morceaux, notations, likes, commentaires et abonnements. Pour garder un code propre et stable, j'ai utilisé ESLint et Prettier pour assurer la cohérence du style et limiter les erreurs de syntaxe.

J'ai aussi intégré Swagger directement au projet afin de documenter automatiquement l'API. Cet outil me permet de générer une interface claire où chaque endpoint est listé avec ses paramètres, ses réponses possibles et ses codes d'erreur. Swagger facilite à la fois mes tests durant le développement et l'utilisation future par d'autres développeurs, car il donne une vue complète et interactive des routes disponibles. Grâce à cette documentation vivante, je peux vérifier en continu la conformité de l'API et garantir qu'elle reste compréhensible et exploitable.



L'ensemble forme un socle solide, maintenable et prêt à évoluer, avec une base technique fiable, un code standardisé et une documentation claire qui accompagne le projet tout au long de son cycle de vie.

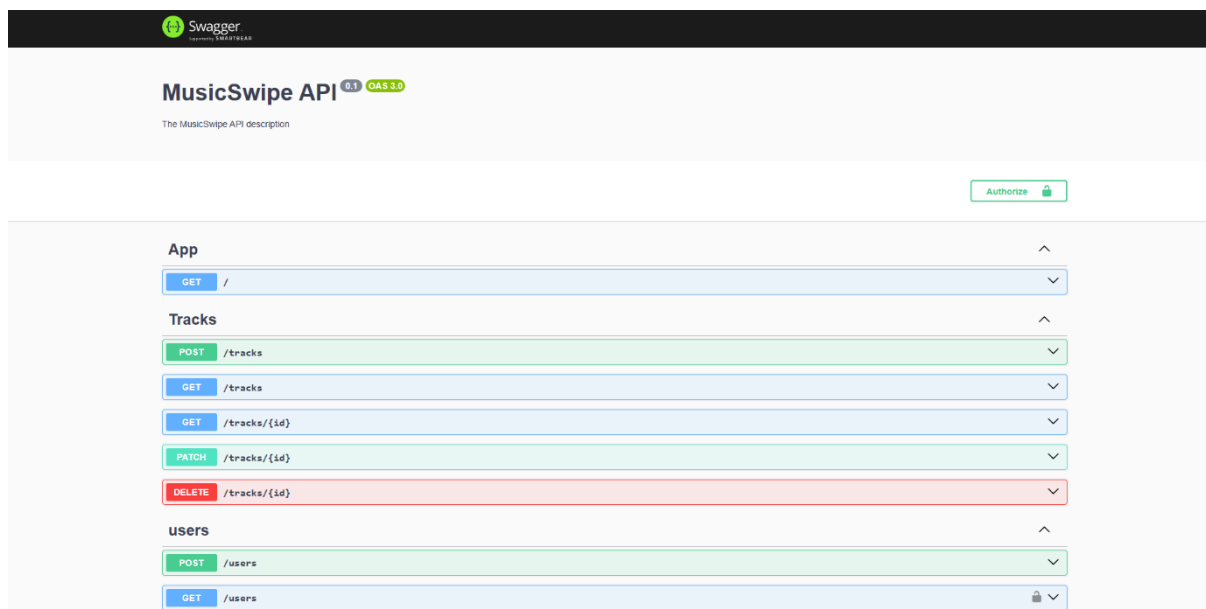


Image 7 : Présentation swagger

6.2.1.2. Justification des choix

J'ai choisi NestJS pour son intégration naturelle avec TypeScript et pour son organisation par modules, plus nette que des approches Node.js plus minimalistes. Prisma s'intègre bien dans ce cadre et apporte une couche d'accès aux données moderne et sûre. PostgreSQL reste un choix fiable en production, avec de bonnes performances et un écosystème mature. Ce trio m'a permis d'avancer vite tout en respectant des standards de qualité.

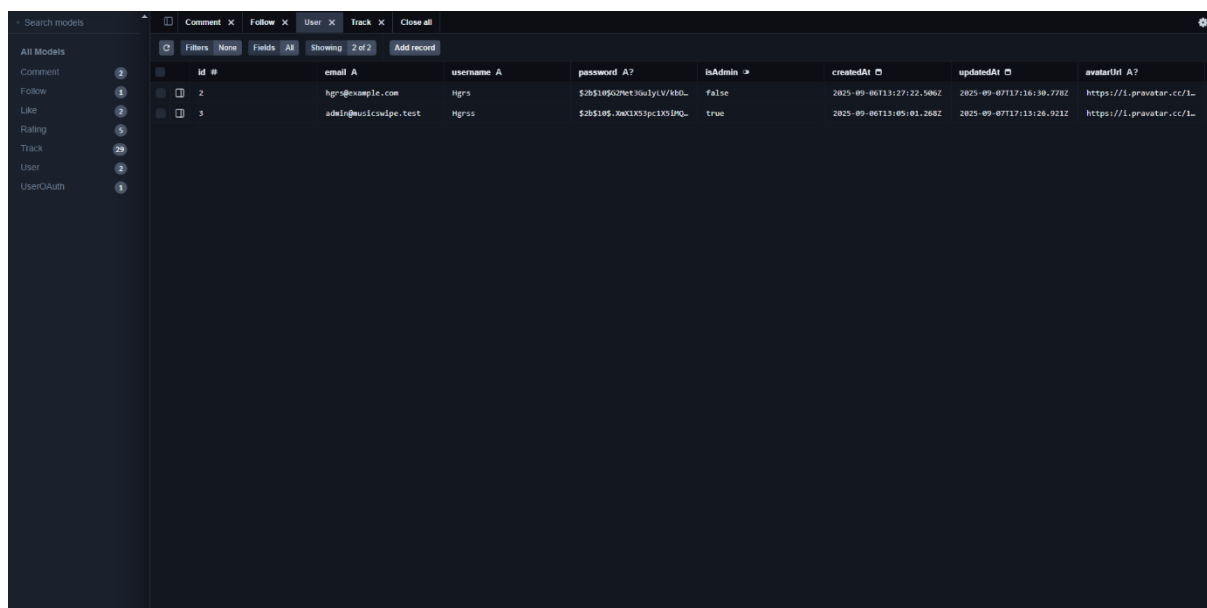


Image 8 : Prisma Studio

6.2.1.3. Organisation du projet

Le backend est découpé en modules dédiés à chaque domaine métier. Chaque module expose un contrôleur pour les routes, un service pour la logique et un repository branché sur Prisma. L'authentification, les utilisateurs, les morceaux, les notations, les likes, les commentaires et les abonnements sont isolés, ce qui rend le code plus simple à lire et plus facile à tester. Un dossier commun regroupe les éléments partagés et un espace de tests héberge les scénarios unitaires et d'intégration. Cette organisation limite le couplage et prépare bien les évolutions futures.

6.2.2. Fonctionnalités implémentées

6.2.2.1. Authentification et login sécurisée

L'authentification combine un parcours classique par e-mail et mot de passe et une connexion via Spotify en OAuth2. Lors de l'inscription, le mot de passe est haché avec bcrypt et n'est jamais stocké en clair. À la connexion, le serveur émet un jeton JWT qui transporte l'identifiant et le rôle de l'utilisateur. Le serveur ne conserve pas d'état de session, ce qui allège l'infrastructure. Pour Spotify, l'utilisateur valide son identité chez le fournisseur, l'application reçoit un code et l'échange contre un jeton d'accès, ce qui rattache le compte Spotify au profil MusicSwipe. Un système de rôles distingue les utilisateurs classiques des administrateurs. Des



guards NestJS vérifient le rôle à chaque accès à une ressource sensible. Les entrées sont validées systématiquement afin de limiter les attaques usuelles et les jetons expirent pour éviter des sessions trop longues. Finalement, l'accès reste fluide côté utilisateur et la protection des données est assurée côté serveur.

6.2.2.2. Gestion des utilisateurs

La gestion des comptes couvre la création, la consultation, la mise à jour et la suppression. À l'inscription, l'e-mail et le pseudonyme sont contrôlés pour éviter les doublons et le mot de passe est haché avant enregistrement. Les profils sont exposés via des endpoints sécurisés. Un utilisateur consulte ses informations, ses notations et ses statistiques, tandis que les champs sensibles restent protégés. La mise à jour passe par des routes authentifiées, avec la possibilité de modifier le pseudonyme, l'avatar ou le mot de passe. Les administrateurs peuvent désactiver un compte, ajuster un rôle ou intervenir en cas d'abus. La suppression tient compte des relations en base et déclenche la suppression ou l'anonymisation des données liées pour conserver la cohérence d'ensemble.

6.2.2.3. Gestion des tracks

La gestion des morceaux alimente le cœur de la découverte et de la notation. Chaque track possède ses métadonnées et peut être relié à un identifiant Spotify pour enrichir automatiquement le catalogue. Côté utilisateur, les morceaux arrivent via l'API et peuvent être notés, likés ou commentés. La logique métier empêche les doublons de like et de note. Côté administrateur, un CRUD complet permet d'ajouter, d'éditer ou de supprimer un morceau et de suivre ses statistiques. Les endpoints sont protégés par des guards et les données entrantes passent par une couche de validation. Des index en base accélèrent les recherches par titre ou par artiste pour garder une navigation fluide.

6.2.2.4. Système d'abonnements

Le système d'abonnements crée la dimension sociale. Un utilisateur peut suivre un autre profil pour retrouver facilement ses notations, ses avis et son activité. La table Follow stocke la relation entre suiveur et suivi et empêche les doublons. L'API expose une route pour suivre et une autre pour se désabonner. Ces actions nécessitent une authentification. Un contrôle évite bien sûr le suivi de soi-même.



6.2.2.5. Gestion des likes

Le like est lié au geste de swipe. Un mouvement vers la droite enregistre un like, un mouvement vers la gauche enregistre un dislike. La table Like relie l'utilisateur au morceau et une contrainte évite les répétitions. L'API permet de créer ou de retirer un like et de consulter la liste des morceaux aimés. Toutes ces opérations sont authentifiées. Ces interactions rapides alimentent le profil musical et les statistiques globales.

6.2.2.6. Gestions des notations

La notation chiffrée apporte une appréciation plus fine qu'un simple like. La table Rating relie un utilisateur, un morceau et une note unique, que l'on peut mettre à jour si l'avis évolue. L'API propose l'ajout, la consultation par utilisateur, le calcul de la moyenne pour un morceau et la mise à jour. Les routes sont sécurisées et valident les entrées. Ce mécanisme complète le swipe et renforce la qualité des statistiques communautaires.

6.2.3. Sécurité backend

6.2.3.1. Hashage des mots de passe

Les mots de passe sont hachés avec bcrypt à l'inscription. En cas de fuite de données, l'empreinte reste inutilisable. À la connexion, bcrypt vérifie l'empreinte sans jamais manipuler la valeur d'origine.

6.2.3.2. Validations des entrées

Les DTOs associés à class-validator contrôlent formats et longueurs. Les champs d'e-mail, les mots de passe et les types attendus sont vérifiés avant tout traitement. Cette barrière limite les injections, les erreurs de type et les charges inutiles côté serveur.

6.2.3.3. Protection des routes

Les routes passent par des guards JWT qui vérifient la validité du jeton et le rôle associé. Les ressources sensibles destinées aux administrateurs restent inaccessibles aux autres profils. Ce filtrage s'applique à chaque requête.

6.2.3.4. Sécurité BDD

Le schéma PostgreSQL s'appuie sur des clés étrangères et des contraintes d'unicité pour préserver l'intégrité. Un utilisateur ne peut pas noter ni liker deux fois le même morceau. Les



suppressions en cascade nettoient les données liées pour éviter les orphelins. Des index ciblés accélèrent les requêtes fréquentes et réduisent l'impact de charges lourdes.

6.3. Développement Admin

Le dashboard administrateur de MusicSwipe a été conçu comme un outil de gestion complet et intuitif, permettant de superviser l'ensemble de la plateforme. Il offre aux administrateurs la possibilité de gérer les utilisateurs, les morceaux et les données globales, tout en fournissant une vue synthétique de l'activité grâce à des statistiques claires.

6.3.1. Choix techniques et architecture

6.3.1.1. Frameworks et langages

Le dashboard administrateur a été pensé pour être rapide à mettre en place, évolutif et simple à maintenir. L'objectif était d'offrir une interface claire et performante afin de gérer efficacement les utilisateurs, les morceaux et l'ensemble des données. J'ai retenu Next.js comme socle applicatif et j'y ai ajouté Refine pour accélérer la construction d'un back-office. Cette combinaison apporte le rendu côté serveur, une navigation réactive et des bases solides pour organiser les ressources et leurs écrans de gestion. L'API du backend est consommée via le connecteur simple-rest proposé par Refine, ce qui simplifie les appels réseau et leur synchronisation. L'interface est réalisée avec TailwindCSS, ce qui permet d'obtenir un rendu moderne et responsive sans multiplier les feuilles de style. Le résultat est une application d'administration qui reste lisible, modulaire et agréable à faire évoluer.

6.3.1.2. Justification des choix

Next.js offre une architecture robuste et familière pour une application React moderne. Le framework gère proprement les pages, les routes et les performances, ce qui facilite la maintenance à long terme. Refine s'est imposé pour gagner du temps sur les écrans standards de gestion. Les composants prêts à l'emploi pour les listes, formulaires et vues détaillées évitent de réécrire du code générique et permettent de me concentrer sur l'intégration métier avec l'API. TailwindCSS complète l'ensemble en accélérant la mise en forme et en gardant une identité visuelle cohérente. Ce trio apporte un bon équilibre entre vitesse de développement, qualité et maintenabilité. Il me laisse la liberté d'ajouter des fonctionnalités ou d'améliorer l'ergonomie sans remettre en cause la base technique.



6.3.1.3. Organisation du projet

L'organisation suit la logique de Next.js et l'approche par ressources de Refine. Chaque domaine possède ses pages et ses composants dédiés, par exemple les utilisateurs, les morceaux ou les paramètres. Les routes de l'application sont structurées dans l'arborescence des pages pour conserver une navigation claire. Les ressources déclarées côté Refine associent directement les vues de liste, de création, d'édition et de détail, ce qui uniformise les parcours. Les composants réutilisables sont rassemblés dans un espace commun afin d'éviter la duplication et de préserver une cohérence visuelle. La configuration de TailwindCSS est centralisée et les utilitaires regroupent les fonctions partagées, notamment pour les appels réseau, la gestion des erreurs et certaines validations côté interface. Cette structuration rend le code facile à parcourir et facilite l'ajout d'une nouvelle ressource sans impacter le reste.

6.3.2. Fonctionnalités implémentées

6.3.2.1. Page d'authentification

L'accès à l'administration commence par une page de connexion simple et sécurisée. L'administrateur renseigne son e-mail et son mot de passe. Le backend vérifie les identifiants et renvoie un jeton qui maintient la session côté client. Ce jeton est contrôlé lors des requêtes suivantes afin de s'assurer que seules les personnes autorisées accèdent aux fonctionnalités. Les erreurs sont traitées de manière explicite afin de guider l'utilisateur. L'interface reste responsive pour être utilisable en mobilité même si la cible principale demeure le desktop. Cette page filtre l'accès à l'espace de gestion et protège les données sensibles.

6.3.2.2. Dashboard Principal

Après connexion, l'administrateur arrive sur une vue d'ensemble de l'activité. Les indicateurs clés sont rassemblés afin de prendre rapidement la température de la plateforme. L'objectif est de repérer en un coup d'œil la dynamique de la communauté ou une anomalie éventuelle. Cette page sert aussi de hub de navigation vers les sections de gestion. Les données sont actualisées via l'API afin de conserver une vision fiable de la situation.

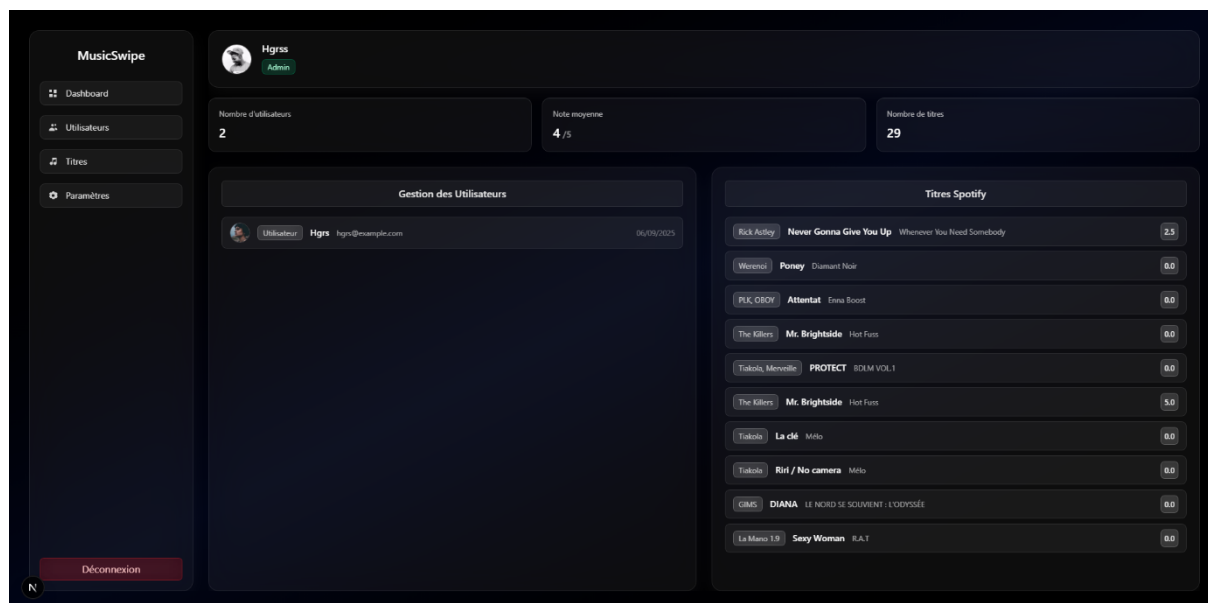


Image 9 : Dashboard principal

6.3.2.3. Dashboard Utilisateurs

Cette section liste les comptes et expose les informations essentielles pour superviser la communauté. L'administrateur peut ouvrir un profil, ajuster un rôle lorsque c'est nécessaire ou supprimer un compte qui ne respecte pas les règles. Les actions passent par des routes sécurisées et ne sont accessibles qu'aux profils autorisés. L'objectif est de garantir un espace sain et de pouvoir réagir vite en cas de problème.

6.3.2.4. Dashboard Tracks

Cette partie concerne le catalogue musical. L'administrateur peut ajouter un morceau, en modifier les informations ou le retirer. Chaque ligne présente les données principales et renvoie vers des statistiques utiles comme le volume de notations ou de likes. Les écrans s'appuient sur les composants de Refine, ce qui accélère le développement tout en gardant une interface homogène et performante.

6.3.2.5. Page settings

La page des paramètres est centrée sur le compte de l'administrateur connecté. Elle permet de modifier le pseudonyme, l'adresse e-mail, le mot de passe ou l'avatar. Les requêtes



transitent par l'API authentifiée et la validation s'effectue côté interface et côté serveur. L'objectif est de renforcer l'autonomie des administrateurs et de sécuriser la gestion de leurs informations.

6.3.3. Sécurité admin

6.3.3.1. Authentification admin

L'espace d'administration est réservé aux comptes disposant du rôle approprié. Lors de la connexion, le serveur émet un jeton et chaque requête suivante vérifie sa validité. Les tentatives d'accès non autorisées sont refusées avec des messages clairs. Ce mécanisme protège l'outil de gestion et limite les risques d'intrusion.

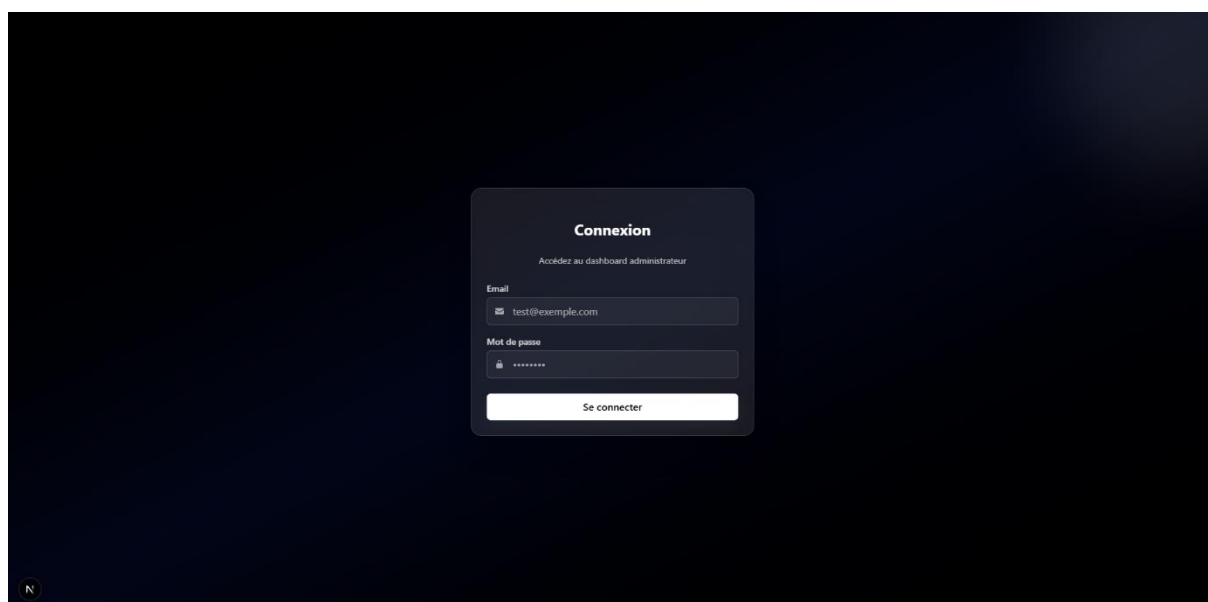


Image 10 : Login dashboard administrateur

6.3.3.2. Gestion des autorisations

Les autorisations sont contrôlées à chaque action. Les guards côté backend filtrent les routes sensibles et l'interface masque les sections non accessibles pour éviter toute manipulation involontaire. Cette double barrière maintient un périmètre d'action strict pour chaque rôle et sécurise les opérations menées dans le dashboard.



6.4. Développement PWA

6.4.1. Choix techniques et architecture

Pour la PWA, j'ai voulu rester cohérent avec ce que j'avais déjà mis en place sur le dashboard admin. Du coup, le choix s'est porté sur Next.js, basé sur React. C'est un framework qui colle bien à ce type de projet, notamment parce qu'il gère facilement le rendu côté serveur et qu'il dispose de beaucoup d'outils pour structurer une application moderne. Pour l'aspect PWA, j'ai intégré next-pwa, qui simplifie l'ajout des fonctionnalités hors-ligne et l'installation directement sur mobile.

Côté design, j'ai utilisé TailwindCSS, comme sur le dashboard. Ça m'a permis de garder une charte graphique uniforme et de gagner du temps sur la mise en forme. L'approche par classes utilitaires facilite aussi la réutilisation des composants. La PWA consomme les données du backend via des appels API REST, donc je n'ai pas eu besoin de réinventer la roue pour la partie communication serveur.

6.4.1.1. Justification des choix

Next.js s'est imposé assez naturellement. C'est un framework largement utilisé, bien documenté, et qui permet de construire des applications web modernes rapidement. Il fonctionne très bien avec React, que je connaissais déjà, et l'ajout de next-pwa vient compléter le tout pour gérer la partie Progressive Web App. Ça m'a fait gagner un temps précieux au lieu de tout configurer à la main.

Pour le style, TailwindCSS m'a semblé être la meilleure solution. Rapide à prendre en main, adapté au responsive et parfaitement intégré à l'écosystème React. En plus, comme je l'avais déjà utilisé pour le dashboard, j'ai pu répliquer facilement mes maquettes Figma. Globalement, ce stack me permet d'assurer la cohérence entre les différentes parties du projet tout en simplifiant la maintenance à long terme.

6.4.1.2. Organisation du projet

J'ai organisé la PWA pour garder un maximum de clarté. La partie frontend est structurée avec Next.js : chaque page correspond à une route, et les composants réutilisables (boutons, cartes,



formulaires) sont regroupés dans un dossier dédié. Les appels à l'API sont centralisés dans un répertoire services, ce qui rend le code plus lisible et plus facile à maintenir.

Pour la gestion des données, j'ai choisi React Query. Ça me permet de gérer automatiquement le cache, la synchronisation avec le serveur et la revalidation des requêtes, tout en gardant une interface fluide. J'ai d'abord mis en place l'architecture de base (navigation, intégration API, configuration PWA), puis j'ai construit les composants réutilisables. Ensuite seulement, je suis passé au développement des pages, ce qui a rendu le travail plus structuré et progressif.

6.4.2. Fonctionnalités implémentées

6.4.2.1. Authentification

L'utilisateur peut créer un compte ou se connecter via un formulaire classique (email/mot de passe). Le backend génère un JWT stocké côté client (localStorage) et utilisé pour sécuriser les requêtes. J'ai aussi ajouté la possibilité de s'inscrire directement avec Spotify grâce à OAuth2. Dans ce cas, un compte est généré automatiquement et lié au profil Spotify.

6.4.2.2. Swipe & flux de découverte

C'est le cœur de l'application. L'utilisateur fait défiler les morceaux un par un et choisit de liker ou de passer. Chaque action met à jour la base et affiche immédiatement le morceau suivant. Quand un compte Spotify est connecté, les dernières écoutes sont importées automatiquement pour alimenter le flux.

6.4.2.3. Détail d'un morceau

Une page permet de consulter les infos d'un titre précis (titre, artiste, album, durée, pochette). Les données sont récupérées via l'API backend en temps réel.

6.4.2.4. Profil & statistiques

Chaque utilisateur a accès à son profil avec pseudo, avatar et email. Une partie statistiques affiche ses habitudes : nombre de morceaux notés, proportion de likes/dislikes, genres préférés et temps passé à découvrir de la musique.



6.4.2.5. Recherche

Un champ de recherche en temps réel permet de retrouver des morceaux ou d'autres utilisateurs. Dès qu'un mot-clé est saisi, une requête est envoyée et les résultats s'affichent instantanément.

6.4.2.6. Suivre / ne plus suivre

La dimension sociale passe par le système de follow. Les utilisateurs peuvent suivre ou se désabonner d'un autre profil. L'interface met à jour l'état du bouton en direct, ce qui rend l'expérience plus interactive.

6.4.2.7. Paramètres

L'utilisateur peut modifier ses infos perso, changer son mot de passe ou se déconnecter. Tout passe par des endpoints sécurisés, avec des validations côté frontend et backend pour éviter les entrées incorrectes.

7. Déploiement et DevOps

7.1. Environnement

Le projet n'est pas encore déployé en production, mais toute l'architecture a été pensée pour que ce soit possible rapidement. Mon idée est de transformer mon ordinateur en homelab afin qu'il serve de serveur dédié. Cela me permet de tester le projet dans un environnement stable et isolé, proche de ce qu'on retrouverait en conditions réelles. Ce choix me donne aussi un contrôle total sur la configuration et la sécurité, tout en restant libre de faire évoluer l'infrastructure plus tard.

L'utilisation de Docker est au cœur de cette préparation. Grâce aux conteneurs, l'application peut tourner de la même manière quel que soit l'environnement. Cette approche réduit les risques de bugs liés aux différences entre machines locales et serveurs. Même si le déploiement définitif n'est pas encore réalisé, la base est prête et ne demande qu'à être mise en place.



7.2. Scripts de déploiement

Pour simplifier le lancement de l'application, j'ai mis en place un fichier `.env` qui centralise toutes les variables sensibles comme les clés API ou les informations de connexion à la base de données. Ce système rend la configuration claire et réutilisable.


```
backend >  .env
1  # Database Configuration
2  DATABASE_URL="test"
3
4  # Sécurité
5  JWT_SECRET="test"
6
7  # Spotify OAuth2 Configuration
8  SPOTIFY_CLIENT_ID=test
9  SPOTIFY_CLIENT_SECRET=test
10 SPOTIFY_REDIRECT_URI=test
11 SPOTIFY_SCOPES="test"
```

Image 11 : Exemple `.env`

En complément, un fichier `docker-compose` orchestre l'ensemble du projet. Chaque partie de l'application, qu'il s'agisse du backend, du dashboard admin ou de la PWA, possède son propre `Dockerfile`. Docker Compose se charge de lancer ces trois services en même temps et de gérer leurs dépendances.



```
1 FROM refinedev/node:18 AS base
2
3 FROM base AS deps
4
5 RUN apk add --no-cache libc6-compat
6
7 COPY package.json yarn.lock* package-lock.json* pnpm-lock.yaml* .npmrc* ./
8
9 RUN \
10   if [ -f yarn.lock ]; then yarn --frozen-lockfile; \
11   elif [ -f package-lock.json ]; then npm ci; \
12   elif [ -f pnpm-lock.yaml ]; then yarn global add pnpm && pnpm i --frozen-lockfile; \
13   else echo "Lockfile not found." && exit 1; \
14   fi
15
16 FROM base AS builder
17
18 COPY --from=deps /app/refine/node_modules ./node_modules
19
20 COPY . .
21
22 RUN npm run build
23
24 FROM base AS runner
25
26 ENV NODE_ENV production
27
28 COPY --from=builder /app/refine/public ./public
29
30 RUN mkdir .next
31 RUN chown refine:nodejs .next
32
33 COPY --from=builder --chown=refine:nodejs /app/refine/.next/standalone ./
34 COPY --from=builder --chown=refine:nodejs /app/refine/.next/static ./next/static
35
36 USER refine
37
38 EXPOSE 3001
39
40 ENV PORT 3001
41 ENV HOSTNAME "0.0.0.0"
42
43 CMD ["node", "server.js"]
```

Image 12 : Exemple de dockerfile

Pour finir, une seule commande suffit pour démarrer l'application complète dans un environnement propre et isolé. Cela simplifie énormément le déploiement et garantit que le projet peut être lancé aussi bien sur mon homelab que sur un serveur externe quand viendra le moment de la mise en ligne.

8. Veille technologie

8.1. Sources et outils de veilles utilisés

Pendant le développement, j'ai pris l'habitude de suivre une veille régulière pour rester à jour sur les technologies que j'utilisais. Je me suis appuyé sur plusieurs sources comme Reddit et Stack Overflow pour les retours d'expérience et la résolution de problèmes, ainsi que sur la documentation officielle de Next.js, TailwindCSS et Docker pour les aspects plus techniques.



J'ai aussi consulté des articles publiés sur Medium et Dev.to, souvent écrits par des développeurs qui partagent leurs bonnes pratiques ou des astuces issues de projets similaires. Enfin, GitHub et Twitter m'ont servi pour repérer rapidement les nouveautés et tendances autour des librairies et frameworks que j'utilisais.

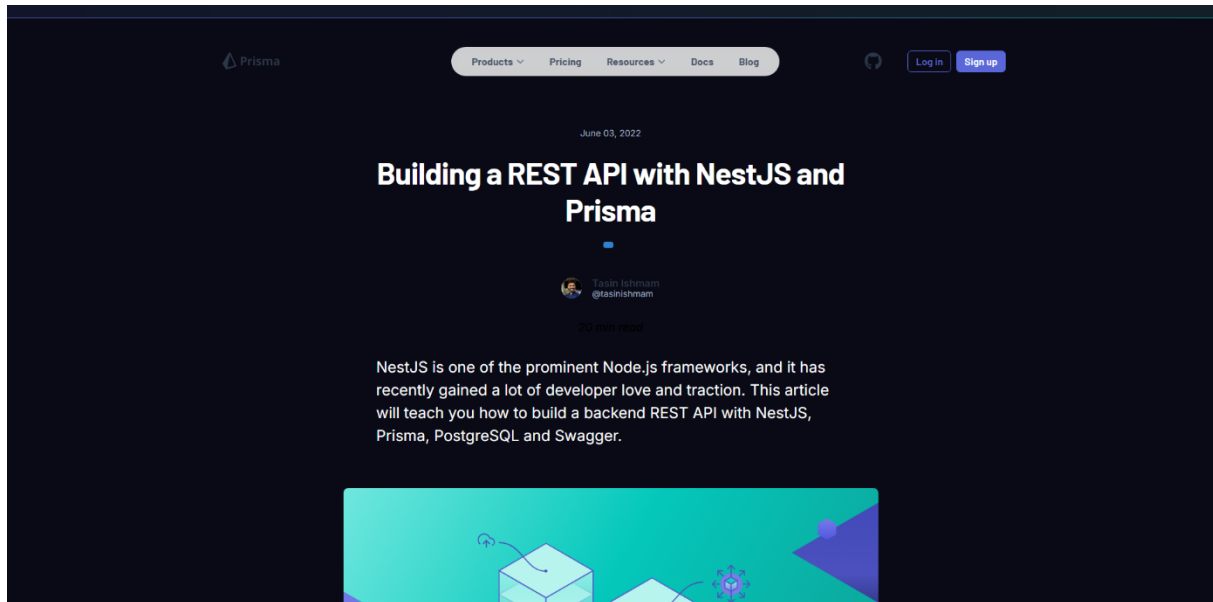


Image 13 : Exemple documentation prisma

Cette veille a joué un rôle important car elle m'a aidé à faire des choix techniques plus pertinents et à trouver des solutions quand je bloquais sur certains points. Elle m'a aussi permis de garder une vision claire des évolutions en cours dans l'écosystème du développement web.

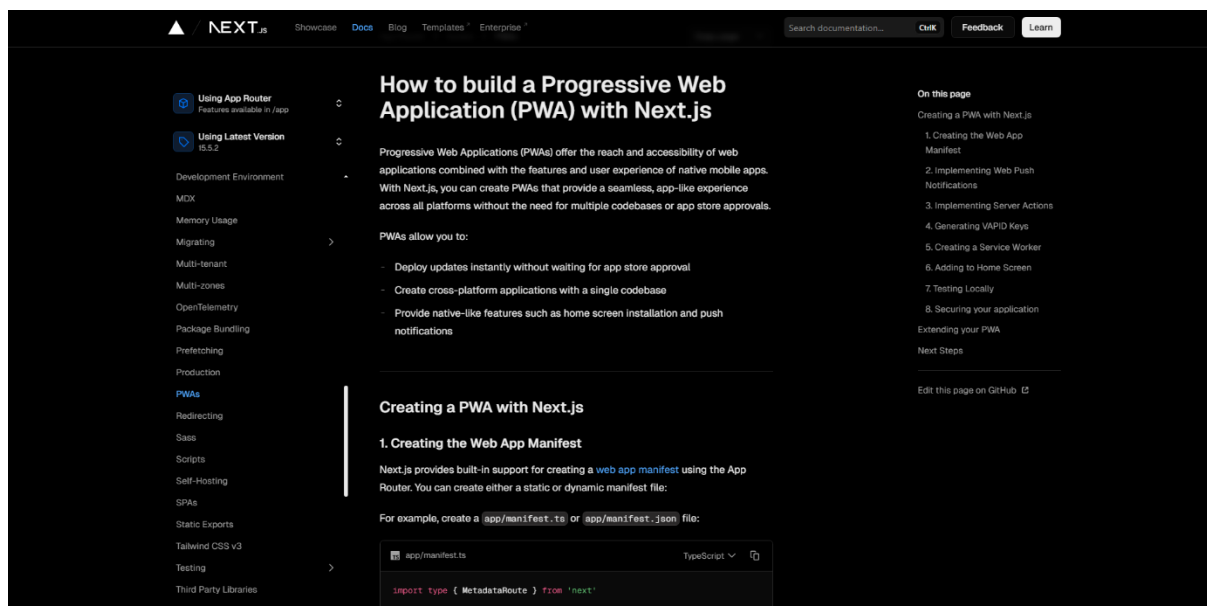


Image 14 : Exemple documentation next.js prisma

8.2. Vulnérabilités et correction

Pour moi, le développement de MusicSwipe a surtout été une façon de pousser un projet à grande échelle afin de tester mes compétences. Ce n'est pas encore une version finale prête pour une sortie publique, et certains points liés à la sécurité avancée n'ont pas encore été entièrement traités. L'objectif était d'abord de valider les choix techniques, de voir jusqu'où je pouvais aller dans la mise en place d'une PWA connectée à un backend et un dashboard admin, puis de construire sur cette base.

Je préfère prendre le temps plus tard d'intégrer des bonnes pratiques supplémentaires comme l'utilisation de cookies pour l'authentification, le renforcement des validations côté backend ou encore la sécurisation des conteneurs lors du déploiement. Pour l'instant, l'application constitue une base solide, qui fonctionne et qui montre le potentiel du projet. Les corrections de failles et l'optimisation de la sécurité feront partie des prochaines étapes avant de viser une mise en production réelle.



9. Bilan et conclusion

9.1. Compétences acquises

La réalisation de MusicSwipe m'a permis de consolider de nombreuses compétences techniques et organisationnelles. J'ai travaillé sur l'ensemble de la chaîne de développement web, depuis la conception d'une base de données relationnelle et d'une API sécurisée jusqu'à la création d'interfaces modernes et responsives avec une Progressive Web App. J'ai aussi appris à structurer un projet complexe en séparant clairement le backend, l'admin et la PWA tout en assurant leur communication via une API REST cohérente.

Sur le plan organisationnel, j'ai gagné en autonomie grâce à l'adoption d'une méthodologie agile adaptée à un projet individuel. J'ai appris à découper mon travail en étapes, à planifier les priorités et à garder une progression constante. La gestion des versions avec Git et GitHub, l'utilisation de branches et de conventions de commits m'ont également permis d'adopter des pratiques proches de celles utilisées en entreprise.

Enfin, le projet m'a donné l'occasion d'approfondir des notions essentielles en sécurité, en authentification et en déploiement. Même si tout n'est pas encore finalisé pour une sortie publique, cette première version m'a permis de tester mes compétences dans un cadre réaliste et de poser des bases solides pour la suite.

9.2. Perspectives d'évolution

L'application est aujourd'hui fonctionnelle, mais elle reste pour l'instant un prototype avancé. Je compte la faire évoluer en renforçant la sécurité, en améliorant encore l'expérience utilisateur et en développant de nouvelles fonctionnalités sociales comme la création de playlists collaboratives ou le partage de morceaux entre membres. L'intégration avec Spotify pourra aussi être poussée plus loin pour offrir une expérience toujours plus riche.

Sur le plan technique, je prévois de déployer l'application d'abord sur un homelab dédié, puis éventuellement de la migrer vers une infrastructure cloud pour la rendre accessible publiquement. À terme, l'objectif est de transformer MusicSwipe en une application complète, stable et sécurisée, capable de rassembler une vraie communauté autour de la découverte musicale.