|  |  |
| --- | --- |
| Une image contenant noir, obscurité  Le contenu généré par l’IA peut être incorrect. | **Aix Ynov Campus**  2 Rue de la Fourane  13090 Aix-en-Provence |
|  |  |

**DOSSIER PROJET**

**MusicSwipe**

**GARRIGUES Hugo**

Concepteur développeur d'applications – RNCP 37873

**Année scolaire 2024-2025**

Sommaire

[**1.** Glossaire 5](#_Toc208266354)

[**2.** **Introduction Générale** 6](#_Toc208266355)

[**2.1.** **Présentation de projet** 6](#_Toc208266356)

[**2.2.** **Objectifs Globaux** 6](#_Toc208266357)

[**2.2.1.** **Objectifs fonctionnels** 6](#_Toc208266358)

[**2.2.2.** **Objectifs techniques** 7](#_Toc208266359)

[**2.3.** **Compétences mises en œuvre** 7](#_Toc208266360)

[**3.** **Contexte et expression des besoins** 8](#_Toc208266361)

[**3.1.** **Contexte du projet** 9](#_Toc208266362)

[**3.2.** **Expression des besoins** 9](#_Toc208266363)

[**3.3.** **Présentation de l’environnement** 10](#_Toc208266364)

[**4.** **Gestion de projet** 10](#_Toc208266365)

[**4.1.** **Méthodologie adoptée** 10](#_Toc208266366)

[**4.2.** **Outils utilisés** 11](#_Toc208266367)

[**4.3.** **Planning prévisionnel** 12](#_Toc208266368)

[**4.4.** **Suivi de l’avancement** 13](#_Toc208266369)

[**5.** **Conception fonctionnelle** 14](#_Toc208266370)

[**5.1.** **Description des parcours utilisateurs** 14](#_Toc208266371)

[**5.2.** **Contraintes fonctionnelles et livrables** 15](#_Toc208266372)

[**6.** **Conception technique** 17](#_Toc208266373)

[**6.1.** **Architecture logicielle** 17](#_Toc208266374)

[**( mettre image d’un schema technique** 18](#_Toc208266375)

[**6.2.** **Base de données** 18](#_Toc208266376)

[**6.2.1.** **Présentation de la base de données** 18](#_Toc208266377)

[**6.3.** **Maquettes et interfaces** 19](#_Toc208266378)

[**7.** **Développements et réalisations techniques** 24](#_Toc208266379)

[**7.1.** **Environnent technique** 24](#_Toc208266380)

[**7.2.** **Développement Backend** 25](#_Toc208266381)

[**7.2.1.** **Choix techniques et architecture** 25](#_Toc208266382)

[**7.2.1.1.** **Frameworks et langages** 25](#_Toc208266383)

[**7.2.1.2.** **Justification des choix** 26](#_Toc208266384)

[**7.2.1.3.** **Organisation du projet** 27](#_Toc208266385)

[**7.2.2.** **Fonctionnalités implémentées** 28](#_Toc208266386)

[**7.2.2.1.** **Authentification et login sécurisée** 28](#_Toc208266387)

[**7.2.2.2.** **Gestion des utilisateurs** 29](#_Toc208266388)

[**7.2.2.3.** **Gestion des tracks** 30](#_Toc208266389)

[**7.2.2.4.** **Système d’abonnements** 31](#_Toc208266390)

[**7.2.2.5.** **Gestion des likes** 31](#_Toc208266391)

[**7.2.2.6.** **Gestions des notations** 32](#_Toc208266392)

[**7.2.3.** **Sécurité backend** 32](#_Toc208266393)

[**7.2.3.1.** **Hashage des mots de passe** 33](#_Toc208266394)

[**7.2.3.2.** **Validations des entrés** 33](#_Toc208266395)

[**7.2.3.3.** **Protection des routes** 33](#_Toc208266396)

[**7.2.3.4.** **Sécurité BDD** 33](#_Toc208266397)

[**7.3.** **Développement Admin** 34](#_Toc208266398)

[**7.3.1.** **Choix techniques et architecture** 34](#_Toc208266399)

[**7.3.1.1.** **Frameworks et langages** 34](#_Toc208266400)

[**7.3.1.2.** **Justification des choix** 34](#_Toc208266401)

[**7.3.1.3.** **Organisation du projet** 35](#_Toc208266402)

[**7.3.2.** **Fonctionnalités implémentées** 36](#_Toc208266403)

[**7.3.2.1.** **Page d’authentification** 36](#_Toc208266404)

[**7.3.2.2.** **Dashboard Principal** 37](#_Toc208266405)

[**7.3.2.3.** **Dashboard Utilisateurs** 37](#_Toc208266406)

[**7.3.2.4.** **Dashboard Tracks** 38](#_Toc208266407)

[**7.3.2.5.** **Page settings** 39](#_Toc208266408)

[**7.3.3.** **Sécurité admin** 39](#_Toc208266409)

[**7.3.3.1.** **Authentification admin** 40](#_Toc208266410)

[**7.3.3.2.** **Gestion des autorisations** 40](#_Toc208266411)

[**7.4.** **Développement PWA** 41](#_Toc208266412)

[**7.4.1.** **Test** 41](#_Toc208266413)

[**7.4.2.** **Test** 41](#_Toc208266414)

[**8.** **Déploiement et DevOps** 41](#_Toc208266415)

[**8.1.** **Environnement** 41](#_Toc208266416)

[**8.2.** **CI/CD** 41](#_Toc208266417)

[**8.3.** **Scripts de déploiement** 41](#_Toc208266418)

[**8.4.** **Monitoring** 41](#_Toc208266419)

[**9.** **Veille technologie** 41](#_Toc208266420)

[**9.1.** **Sources et outils de veilles utilisés** 41](#_Toc208266421)

[**9.2.** **Vulnérabilités et correction** 41](#_Toc208266422)

[**10.** **Bilan et conclusion** 41](#_Toc208266423)

[**10.1.** **Compétences acquises** 41](#_Toc208266424)

[**10.2.** **Perspectives d’évolution** 41](#_Toc208266425)

1. Glossaire

|  |  |
| --- | --- |
| Terme | Définition |
| AJAX | Technique permettant de mettre à jour une partie d’une page web sans la recharger entièrement. |
| Backlog | Liste des tâches ou fonctionnalités à réaliser dans un projet, souvent utilisée en gestion agile. |
| Base de données | Ensemble organisé de données stockées et structurées pour être exploitées par des logiciels. |
| CAO | Conception Assistée par Ordinateur : logiciel permettant de créer des modèles et plans numériques de produits ou de pièces |
| CamelCase | Convention d’écriture où les mots sont collés et chaque mot commence par une majuscule (ex. : MaVariableImportante). |

1. **Introduction Générale**
   1. **Présentation de projet**

Le projet MusicSwipe est une application permettant aux utilisateurs de découvrir de nouveaux morceaux de musique mais aussi de découvrir une manière ludique et interactive de noter leurs écoutes récentes. Le projet est aussi composé d’un dashboard administrateur destiné à la gestion des utilisateurs, des morceaux et des données globales de l’application.

MusicSwipe s’inspire à la fois de l’application Letterboxd, qui repose sur un système de notation et de suivi des contenus, et de Tinder, qui a popularisé le principe du « swipe » à gauche ou à droite pour exprimer une préférence. Dans ce contexte, l’application propose aux utilisateurs un mode de découverte musicale innovant et engageant, leur permettant d’indiquer facilement leurs goûts tout en interagissant avec les morceaux proposés.

L’objectif du projet est d’offrir une expérience moderne, intuitive et sociale centrée sur la découverte musicale. L’application favorise non seulement la recommandation de titres adaptés aux préférences des utilisateurs, mais également l’interaction entre eux grâce à la possibilité de suivre des profils.

* 1. **Objectifs Globaux**

Le projet MusicSwipe poursuit un double objectif. D’une part, il vise à proposer une expérience utilisateur moderne et engageante autour de la découverte musicale. D’autre part, il cherche à démontrer la mise en œuvre de compétences techniques et organisationnelles en conception, développement et déploiement d’une application complète. Ces deux dimensions, fonctionnelles et techniques, sont indissociables et garantissent à la fois la pertinence du produit pour ses utilisateurs et la solidité de sa réalisation.

* + 1. **Objectifs fonctionnels**

Sur le plan fonctionnel, l’objectif principal est de concevoir une application qui facilite la découverte musicale de manière intuitive et interactive. L’utilisateur doit pouvoir explorer de nouveaux morceaux grâce à une page d’accueil présentant les différents titres notés par la communauté. Il doit également pouvoir exprimer ses préférences en indiquant si un morceau lui plaît ou non, et enrichir son expérience grâce à des fonctionnalités sociales telles que le suivi d’autres profils, la possibilité de laisser un commentaire et bien plus encore. Le projet ambitionne ainsi de rendre la découverte et la notation musicales plus ludiques, tout en créant un espace communautaire où les interactions renforcent l’implication des utilisateurs, avec pour finalité de bâtir la meilleure communauté possible autour de la musique.

* + 1. **Objectifs techniques**

Sur le plan technique, MusicSwipe a pour objectif de mettre en œuvre une architecture simple mais robuste, sécurisée et adaptée aux standards actuels du développement web. Le backend doit garantir la fiabilité et la sécurité des données grâce à une API REST bien structurée, protégée et cohérente. Le dashboard administrateur doit offrir une interface claire et performante pour superviser l’ensemble des données de l’application. Enfin, la PWA utilisateur doit proposer une expérience fluide, responsive et mobile-first, en intégrant les mécanismes propres aux applications modernes. L’intégration de l’API Spotify constitue également un objectif technique central. En effet, c’est cette API qui, associée aux comptes des utilisateurs, alimentera automatiquement la base de données de MusicSwipe. Elle permettra ainsi d’exploiter un catalogue riche et de connecter l’application à un service reconnu, tout en respectant les contraintes de performance et de sécurité.

* 1. **Compétences mises en œuvre**

La réalisation du projet MusicSwipe m’a permis de mobiliser un ensemble de compétences variées couvrant l’ensemble du cycle de développement d’une application moderne. Ces compétences concernent à la fois la conception, le développement, la gestion de projet et l’intégration de solutions externes.

Dans un premier temps, j’ai mis en œuvre des compétences d’analyse des besoins et de formalisation du cahier des charges. Cela m’a conduit à traduire des attentes fonctionnelles en spécifications claires et exploitables, notamment à travers la réalisation de parcours utilisateurs, de diagrammes de cas d’utilisation et de maquettes d’interfaces. Cette étape m’a également permis de définir les contraintes techniques et de poser les bases de l’architecture logicielle.

J’ai ensuite exercé des compétences liées au développement côté serveur, en travaillant sur la conception de bases de données relationnelles, sur le développement de composants métier et sur la sécurisation de l’accès aux différentes ressources. J’ai aussi dû mettre en place des mécanismes d’authentification adaptés aux besoins du projet.

Du côté du développement orienté utilisateur, j’ai mobilisé des compétences dans la conception d’interfaces intuitives et responsives, la mise en place de composants réutilisables et la gestion des échanges avec la partie serveur.

En parallèle, j’ai pu mettre en pratique des compétences en gestion de projet et en organisation, en adoptant une méthodologie agile, en structurant un backlog de tâches, en utilisant des outils collaboratifs et en planifiant les étapes de réalisation. Cette organisation m’a permis d’assurer la bonne coordination du projet et de respecter les contraintes de temps et de qualité.

Enfin, j’ai travaillé sur des compétences comme la sécurité, les tests et le déploiement. J’ai veillé à mettre en place une stratégie de tests adaptée, à sécuriser les échanges, à automatiser certaines étapes de livraison et à maintenir une veille technologique sur les bonnes pratiques actuelles. Ces éléments ont contribué à garantir la fiabilité de l’application.

1. **Contexte et expression des besoins**

MusicSwipe est né d’un problème que j’ai moi-même rencontré, que j’ai pu observer autour de moi et même sur les réseaux sociaux. Les applications de notation de musique existantes sont vieillissantes et souvent peu attractives. Pour noter l’ensemble des morceaux qu’un utilisateur souhaite, il faut généralement passer par un processus long et fastidieux, consistant à évaluer chaque titre un par un. De plus, il ne s’agit pas de véritables communautés actives comme on peut en trouver dans d’autres domaines, par exemple avec Letterboxd pour les films. Côté musique, des applications comme Youratemusic ou Musicboard existent, mais elles souffrent d’un manque de dynamisme et ne sont plus vraiment maintenues.

Aujourd’hui, l’offre musicale est immense et il est parfois difficile pour un utilisateur de savoir quoi écouter. L’idée n’est pas de remplacer les algorithmes de recommandation existants, mais plutôt d’offrir un autre angle, la découverte par la communauté. Avec MusicSwipe, un utilisateur peut consulter les notes et avis laissés par d’autres, explorer ce qui est apprécié ou critiqué, et ainsi trouver de nouveaux morceaux à écouter grâce aux retours réels d’une communauté active.

C’est dans ce contexte que j’ai imaginé MusicSwipe, une application qui s’inspire de concepts ayant déjà fait leurs preuves dans d’autres domaines, comme le système de notation de Letterboxd ou le principe de swipe popularisé par Tinder, et qui les transpose dans l’univers de la musique. L’idée est de rendre la découverte musicale plus interactive, plus communautaire et surtout plus engageante.

* 1. **Contexte du projet**

Le projet s’inscrit dans le cadre de ma formation de concepteur développeur d’applications que je réalise à Aix Ynov Campus, dans la perspective de l’obtention de mon titre RNCP de niveau 6. Il répond également à la volonté de créer une application moderne qui allie utilité, innovation et dimension communautaire. La passion que je porte pour la musique a renforcé l’intérêt de ce projet et m’a permis de m’investir pleinement dans sa conception. MusicSwipe se veut à la fois un projet pédagogique, permettant de mettre en pratique l’ensemble des compétences techniques et organisationnelles acquises, et une réponse à un besoin concret que j’ai identifié dans le domaine des applications musicales.

* 1. **Expression des besoins**

Les besoins se répartissent en deux dimensions principales, l’expérience utilisateur et la gestion des données.

Du côté des utilisateurs, l’application doit proposer une interface simple, intuitive et agréable à utiliser. Le système de swipe rend la notation plus rapide et moins contraignante que sur les plateformes existantes, tout en rendant l’expérience plus ludique. Les utilisateurs doivent pouvoir suivre d’autres profils, consulter leurs notes et commentaires, et ainsi enrichir leur propre parcours de découverte musicale. Les interactions sociales sont essentielles pour créer une véritable communauté et renforcer l’engagement.

Du côté de l’administration, il est nécessaire de disposer d’un outil complet et fiable pour gérer l’ensemble des données. Cela inclut la création, la modification et la suppression des utilisateurs et des morceaux, ainsi que la supervision de l’activité globale. Des statistiques doivent également être accessibles pour suivre l’évolution de la communauté et garantir la qualité du service.

Enfin, l’application doit répondre à des besoins en matière de sécurité, de performance et d’évolutivité. Les données doivent être protégées, l’expérience utilisateur doit rester fluide quel que soit le support, et la conception doit permettre l’ajout de nouvelles fonctionnalités à l’avenir.

* 1. **Présentation de l’environnement**

Le projet a été mené dans un environnement technique et organisationnel adapté à un développement complet. Sur le plan humain, il s’agit d’un projet réalisé dans le cadre de ma formation, qui m’a permis de travailler de manière autonome tout en m’appuyant sur les bonnes pratiques étudiées en cours ainsi qu’au travers de mon alternance. Comme dans mon expérience en alternance, j’ai pu travailler seul sur un projet de A à Z, ce qui m’a permis de consolider mes compétences dans ce rôle proche de celui d’un tech lead.

Sur le plan technique, l’environnement de développement reposait sur un ordinateur fonctionnant sous Windows et équipé d’un WSL. Cependant, le matériel étant vieillissant, il m’a parfois ralenti au cours du développement. J’ai également utilisé un iPhone sous iOS afin de tester et développer la PWA dans des conditions proches de celles des utilisateurs finaux.

1. **Gestion de projet**
   1. **Méthodologie adoptée**

Pour la gestion du projet MusicSwipe, j’ai opté pour une méthodologie simple et efficace, basée sur un flux continu inspiré du Kanban. Ce choix s’est imposé naturellement car le projet était mené seul, dans le cadre de ma formation, sans équipe à coordonner ni réunions régulières à organiser. Le flux continu m’a permis de travailler de manière souple, en avançant étape par étape et en visualisant en permanence l’état d’avancement du projet.

L’outil principal que j’ai utilisé est ClickUp, qui m’a permis de centraliser toutes les tâches, de les organiser et de les suivre. J’ai mis en place un tableau composé de plusieurs colonnes représentant les différents statuts possibles d’une tâche : *à faire*, *en cours*, *en développement*, *bloqué*, *en production* et *terminé*. Cette structure offrait une vision claire et immédiate de la progression, tout en identifiant facilement les éléments nécessitant une attention particulière, comme les tâches bloquées ou critiques.

Chaque fonctionnalité, étape technique ou décision de conception était traduite en tâche dans ClickUp. J’ai également utilisé les labels de priorité (urgent, élevé, normal) pour hiérarchiser les tâches et me concentrer d’abord sur ce qui était essentiel à la progression du projet. Par exemple, les fondations techniques telles que l’architecture de la base de données ou l’authentification étaient considérées comme prioritaires, alors que des éléments visuels ou des optimisations mineures pouvaient être traités plus tard.

Même en l’absence d’équipe ou de rituels agiles (comme les daily meetings ou les rétrospectives), cette organisation m’a permis d’adopter une approche proche de l’agilité individuelle. En pratique, je travaillais en cycles courts, je définissais les tâches prioritaires du moment, je les déplaçais dans la colonne *en cours* et je validais rapidement les étapes franchies. Cela m’a aidé à garder une motivation constante en voyant le tableau évoluer et se remplir de tâches clôturées.

Cette méthodologie m’a offert plusieurs avantages. Elle a renforcé ma capacité d’auto-organisation, essentielle dans un projet individuel. Elle m’a également permis de développer une discipline de suivi, en consignant toutes les étapes importantes plutôt que de travailler de manière improvisée. Enfin, elle a facilité la mise en place d’une logique de progression incrémentale, où chaque tâche validée représentait une avancée concrète vers l’objectif final.

* 1. **Outils utilisés**

La réussite du projet MusicSwipe a reposé sur l’utilisation d’un ensemble d’outils qui m’ont accompagné tout au long du cycle de développement, depuis la planification des tâches jusqu’au suivi des versions et aux tests. Ces outils m’ont permis de travailler de manière structurée, d’organiser mes priorités et de garantir la qualité du produit final.

Pour la gestion de projet, j’ai utilisé ClickUp comme outil central, comme je l’ai précisé auparavant.

En parallèle, j’ai utilisé Git et GitHub pour la gestion des versions. GitHub a joué un rôle essentiel dans le suivi du code source et dans la sécurisation du projet. J’ai travaillé avec des branches pour séparer les développements par grandes fonctionnalités, ce qui a facilité les tests avant l’intégration dans la branche principale, tout en respectant le nommage des *conventional commits* et des *conventional branches*. Cette organisation m’a également habitué aux bonnes pratiques utilisées en entreprise pour la collaboration et le versioning, même si le projet a été mené seul.

Pour la conception et la modélisation, j’ai utilisé des outils comme dbdiagram.io pour concevoir la base de données et Figma pour les maquettes d’interfaces. Ces outils m’ont permis de clarifier la vision du projet avant de passer à l’implémentation et de garder une documentation visuelle sur laquelle m’appuyer tout au long du développement.

Côté environnement de développement, j’ai travaillé principalement avec Visual Studio Code, qui offre une grande flexibilité grâce à ses extensions, notamment pour la gestion de bases de données, le débogage et l’intégration avec Git. Pour tester la PWA, j’ai également utilisé mon iPhone, ce qui m’a permis de vérifier l’expérience utilisateur en conditions réelles sur un appareil mobile.

Enfin, pour le suivi technique et la documentation, j’ai utilisé Word tout au long du développement du projet. Cet outil m’a permis d’assurer une trace écrite et de mieux structurer mes idées en complément du tableau de tâches.

L’ensemble de ces outils a constitué un environnement de travail cohérent, couvrant la planification, la conception, le développement, les tests et le suivi, et a joué un rôle déterminant dans la bonne organisation et la réussite du projet.

* 1. **Planning prévisionnel**

Dès le lancement du projet MusicSwipe, j’ai établi un planning prévisionnel afin d’avoir une vision claire des différentes étapes à réaliser et de structurer mon travail dans le temps. L’objectif n’était pas de figer le développement dans un calendrier trop strict, mais de définir des grandes phases et d’anticiper l’ordre logique des priorités.

Pour ce faire, j’ai utilisé ClickUp, qui m’a permis de planifier les différentes tâches dans un tableau organisé par colonnes et d’indiquer des dates d’échéance. J’ai découpé le projet en plusieurs étapes principales : la mise en place de l’environnement de développement, la conception de l’architecture et de la base de données, le développement du backend et de l’API, la création du dashboard administrateur, puis enfin la réalisation de la PWA. Chaque phase comportait elle-même plusieurs sous-tâches que je pouvais suivre au quotidien.

Le planning reposait sur un **flux continu**, ce qui signifie que je ne fonctionnais pas avec des sprints fixes comme dans Scrum, mais plutôt avec une progression étape par étape, en avançant dans l’ordre des priorités définies. J’avais également attribué des labels de priorité aux tâches (urgent, élevé, normal) afin de toujours savoir sur quoi concentrer mes efforts. Par exemple, l’authentification et la sécurité des données faisaient partie des priorités, tandis que l’optimisation visuelle de certaines pages était planifiée plus tard, une fois les fonctionnalités essentielles terminées.

Ce planning m’a servi de feuille de route tout au long du projet. Même si certains ajustements ont été nécessaires en fonction des difficultés rencontrées ou du temps disponible, la structuration initiale m’a permis de garder un cap et de m’assurer que les différentes fonctionnalités étaient réalisées dans un ordre logique. Le fait d’avoir cette vision d’ensemble a été essentiel pour ne pas me disperser et pour rester productif du début à la fin du développement.

* 1. **Suivi de l’avancement**

Le suivi de l’avancement a été un élément clé tout au long du développement de MusicSwipe. J’ai utilisé ClickUp non seulement pour planifier les tâches, mais aussi pour suivre en temps réel leur progression. Chaque fonctionnalité ou étape technique était inscrite dans le tableau et passait par différentes colonnes correspondant à son état : *à faire*, *en cours*, *en développement*, *bloqué*, *en production* et *terminé*. Cette organisation m’a permis d’avoir une vision claire de l’état global du projet à chaque instant.

Pour garder une cohérence dans ma progression, j’ai pris l’habitude de mettre à jour mon tableau quotidiennement. À la fin de chaque session de travail, je déplaçais les tâches vers la colonne correspondante, ce qui me permettait d’identifier immédiatement ce qui restait à accomplir et ce qui avait déjà été validé. Cette routine simple m’a aidé à garder le rythme et à rester motivé, car je pouvais voir concrètement l’avancement du projet.

Lorsque je rencontrais une difficulté technique, la tâche concernée était déplacée dans la colonne *bloquée*. Cela me permettait de ne pas perdre de temps et de continuer à avancer sur d’autres aspects du projet en parallèle, tout en gardant une trace claire des points bloquants à résoudre plus tard.

Le suivi reposait également sur l’utilisation des labels de priorité. Ils m’ont permis de gérer les urgences et d’adapter mon travail en fonction de l’importance des tâches. Par exemple, je pouvais mettre de côté certaines optimisations secondaires pour me concentrer sur les fonctionnalités critiques, comme l’authentification ou la gestion des données utilisateurs.

Même si le projet a été mené seul, cette organisation proche d’un suivi agile m’a donné une discipline comparable à celle d’un projet en équipe. Elle m’a permis de rester constant dans ma progression, de ne pas me disperser et de m’assurer que chaque fonctionnalité importante était bien développée et validée avant de passer à la suivante.

1. **Conception fonctionnelle** 
   1. **Description des parcours utilisateurs**

Afin de répondre aux besoins identifiés, deux parcours principaux ont été définis : celui de l’utilisateur normal et celui de l’administrateur. Ces parcours décrivent l’ensemble des interactions possibles avec l’application MusicSwipe et permettent de mieux comprendre la logique de navigation ainsi que les fonctionnalités mises à disposition selon le rôle de l’acteur.

Commençons par le parcours utilisateur, il commence par la phase d’inscription et de connexion. L’utilisateur peut créer un compte à l’aide de son adresse e-mail ou via une solution OAuth2, comme Spotify ou Deezer. L’accès est sécurisé grâce à l’utilisation de jetons JWT, ce qui permet de protéger les données et les sessions.

Une fois connecté, l’utilisateur accède à la page d’accueil, qui constitue le cœur de l’application. Celle-ci présente un flux de titres, d’albums ou d’artistes proposés aléatoirement ou en fonction de ses écoutes récentes. L’interface est pensée mobile-first et optimisée pour le tactile, afin de permettre une navigation fluide basée sur le swipe. Le swipe vers la droite correspond à un like ou à une note positive, tandis que le swipe vers la gauche exprime un dislike ou une note faible. L’utilisateur peut également faire défiler le contenu vers le haut pour passer au morceau suivant.

Lorsqu’il souhaite obtenir plus d’informations, l’utilisateur peut consulter une fiche détaillée sur un titre, un artiste ou un album. Cette fiche affiche la note moyenne globale, ses propres notations, les avis qu’il a laissés ainsi que des informations complémentaires telles que la date de sortie, le genre ou encore le nom de l’artiste.

L’utilisateur dispose également d’un espace personnel à travers son profil. Celui-ci regroupe son historique de notations, ainsi que des statistiques personnalisées, comme la moyenne de ses notes, ses genres favoris ou ses artistes les plus appréciés. Enfin, une fonctionnalité de recherche est disponible pour lui permettre de trouver un morceau, un artiste ou un album précis grâce à un moteur de recherche intégré.

**Parcours administrateur**  
L’administrateur dispose d’un parcours spécifique, centré sur la gestion et le suivi global de l’application. Comme l’utilisateur normal, il commence par la connexion, qui peut s’effectuer par e-mail ou via OAuth2. L’accès est également sécurisé par JWT, mais complété par un rôle administrateur lui conférant des droits étendus.

Il accède ensuite à une page d’accueil spécifique sous forme de dashboard. Cette interface propose une vue synthétique regroupant des statistiques globales, telles que le nombre d’utilisateurs inscrits, le nombre total de swipes et de notations, ou encore les logs récents enregistrés en base de données. Bien que la présentation soit optimisée pour l’administration, l’interface conserve une ergonomie proche de celle des utilisateurs classiques, notamment sur mobile.

L’administrateur dispose de fonctionnalités de gestion avancées. Il peut intervenir sur les actions des utilisateurs grâce à un CRUD complet sur les commentaires et les notes, afin de modérer le contenu si nécessaire. Il a également accès à la gestion des utilisateurs, qui lui permet de visualiser leur historique d’activité, de désactiver ou supprimer des comptes et de contrôler les logs liés aux swipes ou aux comportements suspects.

Comme pour l’utilisateur normal, il peut accéder aux fiches détaillées des titres, albums ou artistes, mais avec la possibilité supplémentaire d’analyser des statistiques plus poussées. Enfin, il dispose d’un moteur de recherche pour trouver rapidement un utilisateur, un album ou un artiste et gérer efficacement les données de l’application.

* 1. **Contraintes fonctionnelles et livrables**

La conception de MusicSwipe s’est appuyée sur un ensemble de contraintes fonctionnelles définissant les règles de fonctionnement de l’application, ainsi que sur des livrables concrets garantissant la complétude et la qualité du projet.

L’application devait, en premier lieu, être accessible sur différents supports, avec une priorité donnée à l’expérience mobile. Le choix d’une Progressive Web App s’est imposé afin de proposer une utilisation fluide et responsive, intégrant les spécificités tactiles du swipe. La navigation devait être intuitive, avec une interface claire et des parcours utilisateurs simples à comprendre.

La sécurité a constitué une contrainte majeure. Les données des utilisateurs devaient être protégées grâce à une authentification sécurisée, une gestion rigoureuse des sessions et un contrôle des accès différencié entre utilisateurs classiques et administrateurs. De plus, la cohérence et l’intégrité des données devaient être garanties, en particulier pour les notations et les interactions communautaires.

Une autre contrainte importante concernait la performance et la rapidité de l’application. Les utilisateurs ne devaient pas subir de ralentissements notables, que ce soit lors du swipe, de la consultation des fiches ou de la recherche d’un contenu spécifique. Cela impliquait une gestion efficace des appels API et une optimisation de la base de données.

Enfin, l’application devait respecter une logique communautaire. Elle devait permettre non seulement de noter et de découvrir de la musique, mais aussi d’interagir avec les autres utilisateurs.

À l’issue du projet, plusieurs livrables ont été produits afin de répondre aux objectifs fixés. Le livrable principal est bien entendu l’application elle-même, composée de trois parties distinctes : le backend fournissant une API REST sécurisée, le dashboard d’administration offrant une interface de gestion complète et la PWA utilisateur permettant l’accès aux fonctionnalités principales de notation et de découverte musicale.

En complément, des livrables techniques et documentaires ont été réalisés. La base de données a été conçue et modélisée, puis documentée à travers un modèle conceptuel et un modèle physique. Des maquettes d’interface ont également été produites afin de guider le développement frontend. Enfin, une documentation de projet a été rédigée, comprenant notamment le présent dossier, qui détaille la conception, la réalisation et les choix techniques opérés.

Ces livrables constituent la preuve tangible du travail accompli et permettent de montrer la correspondance entre les besoins exprimés et la solution effectivement développée.

1. **Conception technique**
   1. **Architecture logicielle**

L’architecture logicielle de MusicSwipe repose sur une séparation claire entre trois grandes parties, le backend, le dashboard administrateur et la Progressive Web App utilisateur. Cette organisation modulaire permet de distinguer les rôles et responsabilités de chaque composant, tout en garantissant une communication fluide et sécurisée entre eux.

Le backend constitue le socle technique du projet. Il expose une API REST sécurisée qui centralise l’ensemble des fonctionnalités métier : gestion des utilisateurs, authentification, intégration Spotify, gestion des morceaux, notations, commentaires et système de suivi entre utilisateurs. Il est responsable de la persistance des données et assure la sécurité des échanges.

Le dashboard administrateur est une application web connectée directement à l’API du backend. Il permet aux administrateurs de gérer les utilisateurs, de superviser le catalogue musical, de modérer les notations et les commentaires, et de suivre l’évolution de l’activité grâce à des statistiques globales. Sa conception met l’accent sur l’efficacité et la clarté, afin de fournir un outil de gestion fiable et intuitif.

La PWA utilisateur constitue l’interface principale destinée aux membres de la communauté. Pensée mobile-first, elle permet de s’inscrire, de se connecter, de découvrir de nouveaux morceaux via un système de swipe, de laisser des notations et des avis, ainsi que de suivre d’autres profils. Elle communique avec le backend pour récupérer et envoyer les données, et intègre des mécanismes propres aux Progressive Web Apps, comme l’installation sur smartphone et une utilisation partielle hors ligne grâce aux services workers et au cache.

Cette architecture repose sur une communication centralisée via l’API REST, ce qui garantit une indépendance entre les différentes couches. Chaque partie peut évoluer de manière autonome tout en restant compatible avec les autres. Elle a également été pensée pour répondre aux enjeux de sécurité, de scalabilité et de maintenabilité, afin d’assurer la pérennité du projet et sa capacité à intégrer de nouvelles fonctionnalités à l’avenir.

**( mettre image d’un schema technique**

* 1. **Base de données**

La base de données constitue un élément central de MusicSwipe, car elle permet de gérer l’ensemble des informations liées aux utilisateurs, aux morceaux, aux notations, aux interactions sociales et à l’authentification. Elle a été conçue selon une approche relationnelle, afin de garantir la cohérence, l’intégrité et la sécurité des données.

* + 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.

Une image contenant capture d’écran, Modélisation 3D, conception

Le contenu généré par l’IA peut être incorrect.Follow : permet de gérer les relations sociales entre utilisateurs, en distinguant le suiveur et l’utilisateur suivi .

Image 1 : Modèle de base de données MusicSwipe (dbdiagram.io)

* 1. **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.

Une image contenant texte, capture d’écran, logiciel, conception

Le contenu généré par l’IA peut être incorrect.Une image contenant texte, capture d’écran, conception

Le contenu généré par l’IA peut être incorrect. 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.

Une image contenant capture d’écran, texte, multimédia, logiciel

Le contenu généré par l’IA peut être incorrect.Une image contenant texte, multimédia, Appareil électronique, capture d’écran

Le contenu généré par l’IA peut être incorrect.Une image contenant capture d’écran, multimédia, texte, logiciel

Le contenu généré par l’IA peut être incorrect.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.

Une image contenant Appareils électroniques, capture d’écran, texte, logiciel

Le contenu généré par l’IA peut être incorrect.Une image contenant texte, capture d’écran, logiciel, multimédia

Le contenu généré par l’IA peut être incorrect.Une image contenant Appareils électroniques, capture d’écran, texte, Appareil électronique

Le contenu généré par l’IA peut être incorrect.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

Une image contenant capture d’écran, texte, logiciel, Logiciel multimédia

Le contenu généré par l’IA peut être incorrect.Une image contenant texte, capture d’écran, multimédia, gadget

Le contenu généré par l’IA peut être incorrect.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.

1. **Développements et réalisations techniques**
   1. **Environnent technique**

Le développement de MusicSwipe s’est appuyé sur un environnement technique complet, regroupant à la fois les outils logiciels, le matériel utilisé et les services externes nécessaires à la réalisation du projet. L’objectif était de disposer d’un cadre de travail adapté à un projet moderne, sécurisé et maintenable, tout en tenant compte des contraintes matérielles.

Sur le plan matériel, j’ai travaillé principalement sur un ordinateur sous Windows, équipé d’un WSL (Windows Subsystem for Linux) afin de bénéficier d’un environnement plus proche des standards du développement backend. Bien que la machine soit relativement vieillissante, elle m’a permis de mener à bien le projet. En complément, j’ai utilisé mon iPhone sous iOS pour tester la PWA en conditions réelles et m’assurer de l’optimisation mobile-first.

Côté logiciels, l’éditeur Visual Studio Code a constitué l’outil central pour le développement. Grâce à ses nombreuses extensions, il m’a permis de gérer efficacement le code, d’intégrer Git, de réaliser du débogage et de travailler sur la base de données. Pour la conception des maquettes, j’ai eu recours à Figma, tandis que dbdiagram.io m’a servi pour la modélisation de la base de données.

La gestion du code source et du versioning a été assurée par Git et GitHub, avec un découpage en branches selon les fonctionnalités développées et l’utilisation de commits normalisés (*conventional commits*). Cela a permis d’assurer un suivi clair des évolutions et de garder un historique complet du projet.

Pour la gestion de projet, j’ai utilisé ClickUp, qui m’a permis d’organiser mes tâches dans un tableau de type Kanban et de suivre l’avancement en temps réel. Cet outil a facilité la priorisation des développements et la visualisation globale du projet.

Enfin, l’environnement technique intègre également l’utilisation de PostgreSQL comme base de données relationnelle, gérée via Prisma en tant qu’ORM, et l’intégration de services externes comme l’API Spotify pour enrichir le catalogue musical et gérer l’authentification OAuth.

Cet ensemble d’outils et de technologies a permis de mettre en place un environnement de développement complet et cohérent, répondant aux besoins du projet tout en respectant les standards actuels du développement web.

* 1. **Développement Backend**
     1. **Choix techniques et architecture**
        1. **Frameworks et langages**

Le développement du backend de MusicSwipe s’appuie sur un ensemble de frameworks et de technologies modernes, choisis pour leur robustesse, leur popularité et leur adéquation avec les besoins du projet.

Le cœur du backend repose sur NestJS, un framework Node.js basé sur TypeScript. Ce choix s’est imposé pour plusieurs raisons. Tout d’abord, NestJS propose une structure claire et modulaire, inspirée d’Angular, qui favorise la maintenabilité et la scalabilité. Ensuite, son intégration native avec TypeScript permet de bénéficier d’un typage fort, réduisant ainsi les erreurs potentielles et améliorant la lisibilité du code. Enfin, NestJS est particulièrement adapté à la mise en place d’APIs REST sécurisées, ce qui correspond parfaitement aux objectifs de MusicSwipe.

Pour la gestion des données, j’ai utilisé Prisma en tant qu’ORM. Prisma simplifie la communication avec la base de données en offrant une API fortement typée et en générant automatiquement un client adapté aux modèles définis. Cela permet non seulement de réduire le risque d’erreurs liées aux requêtes SQL, mais aussi d’accélérer le développement en facilitant les opérations CRUD.

La base de données choisie est PostgreSQL, une solution relationnelle reconnue pour sa fiabilité, ses performances et ses fonctionnalités avancées. PostgreSQL s’est révélé particulièrement adapté pour gérer les relations complexes nécessaires dans MusicSwipe (notations, likes, suivis entre utilisateurs, etc.), tout en garantissant une bonne évolutivité.

En complément, différents outils et dépendances de développement ont été intégrés pour améliorer la qualité du code et assurer la stabilité du backend. ESLint et Prettier permettent de respecter des standards de code clairs et homogènes. Les tests unitaires et d’intégration s’appuient sur Jest et Supertest, garantissant la robustesse des fonctionnalités développées. Enfin, l’utilisation de SWC et de ts-node a facilité le démarrage et l’exécution du projet, tout en maintenant une bonne productivité.

Ainsi, l’association de NestJS, TypeScript, Prisma et PostgreSQL, soutenue par un ensemble d’outils de test et de qualité de code, constitue un socle solide pour le backend de MusicSwipe. Cet environnement technique permet de répondre aux exigences de sécurité, de performance et de maintenabilité attendues d’une application moderne.

* + - 1. **Justification des choix**

Le choix des technologies utilisées pour le backend de MusicSwipe est le résultat à la fois d’un apprentissage effectué tout au long de ma formation et d’une appréciation personnelle des outils.

J’ai commencé à travailler avec NestJS dès le début de l’année et j’ai rapidement apprécié ce framework. Je le trouve particulièrement simple à prendre en main tout en restant très structuré. Son approche modulaire et son intégration native avec TypeScript en font un outil propre et agréable à utiliser au quotidien. De plus, il intègre nativement des concepts de sécurité (comme la gestion des guards et des middlewares), ce qui m’a permis de mettre en place une API robuste et protégée avec moins de complexité que sur d’autres frameworks plus bas-niveau comme Express.

L’utilisation de Prisma s’est imposée de manière naturelle avec NestJS. Cet ORM propose une approche moderne, claire et fortement typée, ce qui facilite la communication entre le code et la base de données. Prisma permet de générer automatiquement un client adapté aux modèles définis, rendant les opérations CRUD rapides et fiables. Associé à NestJS, il offre une productivité accrue tout en minimisant les risques d’erreurs.

Enfin, le choix de PostgreSQL comme base de données repose sur sa fiabilité et sa compatibilité avec Prisma. PostgreSQL est un système relationnel largement éprouvé, capable de gérer des relations complexes tout en offrant de bonnes performances et une grande stabilité. Sa maturité et sa popularité en font une solution adaptée à un projet comme MusicSwipe, où la cohérence et l’intégrité des données sont essentielles.

En résumé, l’association NestJS + Prisma + PostgreSQL m’a semblé être le meilleur compromis entre simplicité, sécurité, maintenabilité et efficacité. Elle correspond à mon apprentissage, à mes préférences techniques et aux besoins concrets du projet.

* + - 1. **Organisation du projet**

L’organisation du backend de MusicSwipe repose sur une structure modulaire, pensée pour séparer clairement les responsabilités et faciliter la maintenabilité du code. Cette organisation suit les conventions proposées par NestJS, tout en étant adaptée aux besoins spécifiques du projet.

Le projet est découpé en plusieurs modules, chacun étant responsable d’une partie précise de l’application. Cette approche modulaire permet d’isoler les fonctionnalités, de simplifier les tests et de rendre le code plus lisible. Parmi les modules principaux, on retrouve :

Auth : gère l’authentification et l’autorisation des utilisateurs. Il inclut la création et la vérification des jetons JWT, ainsi que l’intégration avec OAuth2 pour la connexion via Spotify.

Users : responsable de la gestion des comptes utilisateurs (création, lecture, mise à jour, suppression). Il inclut également la gestion des profils et des relations entre utilisateurs.

Tracks : dédié à la gestion des morceaux. Il permet d’ajouter, consulter et supprimer des titres, ainsi que de récupérer leurs informations associées.

Ratings : module qui gère les notations des morceaux par les utilisateurs, avec un contrôle empêchant une double notation du même titre.

Likes : permet de gérer les interactions rapides issues du swipe (like/dislike) et de les lier aux morceaux correspondants.

Comments : dédié aux commentaires laissés par les utilisateurs sur les morceaux.

Follows : gère le système de relations sociales (suivre un utilisateur, arrêter de le suivre).

Chaque module est organisé autour d’un contrôleur (responsable des endpoints API), d’un service (contenant la logique métier) et d’un repository qui communique avec la base de données via Prisma. Cette organisation claire rend le projet extensible : un nouveau module peut être ajouté sans impacter le fonctionnement des autres.

En complément, un dossier common regroupe les éléments partagés entre plusieurs modules, comme les DTOs (Data Transfer Objects), les middlewares, les interceptors ou les stratégies de sécurité.

Enfin, un dossier tests contient les tests unitaires et d’intégration réalisés avec Jest et Supertest, permettant de vérifier le bon fonctionnement des routes et de la logique métier.

Cette organisation modulaire et hiérarchisée assure à la fois la lisibilité, la maintenabilité et l’évolutivité du backend de MusicSwipe.

* + 1. **Fonctionnalités implémentées**
       1. **Authentification et login sécurisée**

L’authentification est un point central de MusicSwipe, car elle conditionne l’accès sécurisé aux fonctionnalités de l’application. Le système mis en place combine une authentification classique (email et mot de passe) avec une authentification via un fournisseur tiers (Spotify), afin d’offrir à l’utilisateur plusieurs options de connexion.

Dans le cas de l’authentification par email et mot de passe, le processus repose sur la création d’un compte lors de l’inscription. Le mot de passe de l’utilisateur est haché avant d’être stocké en base de données, garantissant qu’aucune donnée sensible n’est conservée en clair. Lors de la connexion, les informations fournies par l’utilisateur sont vérifiées et, en cas de succès, un jeton JWT (JSON Web Token) est généré et transmis au client. Ce jeton contient les informations essentielles sur l’utilisateur (identifiant, rôle, droits d’accès) et permet d’assurer une authentification sans nécessiter un stockage de session côté serveur.

Pour l’authentification via Spotify, MusicSwipe utilise le protocole OAuth2. L’utilisateur est redirigé vers Spotify afin de valider son identité, puis un code d’autorisation est renvoyé à l’application. Ce code est échangé contre un jeton d’accès qui permet d’associer le compte Spotify de l’utilisateur à son profil MusicSwipe. Cette méthode simplifie la connexion et permet également d’enrichir la base de données de l’application grâce aux informations musicales issues de Spotify.

Un système de rôles et permissions a également été intégré afin de différencier les droits d’accès. Les utilisateurs classiques ont accès aux fonctionnalités de notation, de swipe et d’interaction sociale, tandis que les administrateurs disposent de privilèges supplémentaires leur permettant de gérer les utilisateurs et le catalogue musical via le dashboard. Cette gestion est réalisée au moyen de guards NestJS, qui vérifient systématiquement le rôle associé au jeton JWT avant d’autoriser l’accès à une ressource protégée.

Enfin, l’ensemble de l’authentification est renforcé par des mécanismes de validation des entrées pour prévenir les attaques classiques (injections SQL, failles XSS) et par la mise en place d’une expiration des jetons JWT. Ce système garantit que les sessions ne restent pas actives indéfiniment et que l’utilisateur doit se reconnecter après un certain délai.

Grâce à cette combinaison d’authentification sécurisée, MusicSwipe assure à la fois une expérience fluide pour l’utilisateur et une protection robuste des données et des accès.

* + - 1. **Gestion des utilisateurs**

La gestion des utilisateurs constitue une fonctionnalité essentielle du backend de MusicSwipe. Elle permet de créer, consulter, modifier et supprimer des comptes, tout en garantissant la sécurité et l’intégrité des données stockées en base.

Lors de l’inscription, l’utilisateur fournit les informations nécessaires à la création de son compte : adresse e-mail, nom d’utilisateur et mot de passe. Avant d’être enregistré en base, le mot de passe est systématiquement haché afin de ne jamais être stocké en clair. Une vérification est également effectuée pour garantir l’unicité de l’email et du pseudonyme, évitant ainsi les doublons.

La consultation des profils est accessible via des endpoints sécurisés. Un utilisateur peut accéder à son propre profil, qui contient ses informations personnelles, son avatar, ses notations et ses statistiques. En revanche, l’accès aux données sensibles (comme l’email) reste restreint et n’est pas exposé publiquement. Lorsqu’un profil d’un autre utilisateur est consulté, seules les informations autorisées (pseudo, avatar, notations publiques, commentaires) sont accessibles.

La mise à jour des comptes est possible via des endpoints protégés par authentification. Un utilisateur peut modifier ses informations personnelles, telles que son pseudonyme, son mot de passe ou son avatar. Les administrateurs disposent de privilèges supplémentaires leur permettant de modifier certains paramètres liés aux utilisateurs en cas de besoin (désactivation d’un compte, mise à jour forcée d’un rôle, etc.).

La suppression des comptes peut être initiée par l’utilisateur lui-même (désactivation volontaire) ou par un administrateur en cas de non-respect des règles de la communauté. Cette action entraîne également la suppression ou l’anonymisation des données associées (notations, commentaires, likes), afin de respecter la cohérence des relations en base tout en préservant l’historique global de l’application.

Enfin, un système de rôles et permissions différencie les utilisateurs classiques des administrateurs. Cette gestion est effectuée via des guards NestJS qui vérifient systématiquement le rôle de l’utilisateur à partir de son jeton JWT avant d’autoriser l’accès aux routes sensibles.

Ainsi, la gestion des utilisateurs assure à la fois la protection des données personnelles et la mise en place d’une expérience fluide pour la communauté, tout en laissant aux administrateurs les moyens de superviser et de contrôler efficacement la plateforme.

* + - 1. **Gestion des tracks**

La gestion des morceaux est une fonctionnalité essentielle de MusicSwipe, car elle constitue la base du système de découverte et de notation. Elle permet d’ajouter de nouveaux titres, de les consulter, de les modifier et de les supprimer, tout en assurant une cohérence avec les autres entités de l’application comme les notations, les likes ou les commentaires.

Chaque track est enregistré dans la base avec ses principales métadonnées : titre, artiste, album, durée et, si disponible, un extrait audio de 30 secondes. Un identifiant Spotify peut également être associé afin de relier directement le morceau au service externe et d’enrichir automatiquement le catalogue.

Du côté utilisateur, les morceaux sont récupérés par l’API et affichés dans l’application mobile. Ils peuvent être notés, likés ou commentés. La logique métier s’assure qu’un utilisateur ne puisse pas noter ou liker un même morceau plusieurs fois, en respectant les contraintes définies dans la base.

Du côté administrateur, le dashboard offre un CRUD complet pour gérer les morceaux. L’administrateur peut ajouter un titre manuellement ou via l’intégration Spotify, modifier les informations d’un morceau existant, ou encore le supprimer si nécessaire. Ce rôle permet aussi de superviser l’ensemble du catalogue et de garantir sa pertinence.

Enfin, l’API de gestion des tracks a été pensée pour être performante et sécurisée. Les endpoints sont protégés par des guards, et des validations sont effectuées sur les données entrantes pour éviter les incohérences (par exemple, interdiction de créer un morceau sans titre). Des index ont été mis en place en base pour optimiser les recherches, notamment par titre ou par artiste.

Ainsi, la gestion des tracks constitue une brique fondamentale de MusicSwipe, en reliant les fonctionnalités sociales et communautaires à un catalogue musical riche et structuré.

* + - 1. **Système d’abonnements**

Le système d’abonnements permet de créer une dimension sociale au sein de MusicSwipe. Les utilisateurs peuvent suivre d’autres profils afin de consulter plus facilement leurs notations, leurs commentaires et leur activité récente. Cette fonctionnalité est essentielle pour renforcer l’aspect communautaire de l’application et favoriser les interactions entre membres.

Techniquement, chaque abonnement est stocké dans la table Follow, qui contient une relation entre le suiveur (*follower*) et l’utilisateur suivi (*followed*). Les contraintes définies dans la base empêchent la duplication, garantissant qu’un utilisateur ne peut pas suivre deux fois le même profil. Lorsqu’un utilisateur se désabonne, la relation est simplement supprimée.

Côté API, deux endpoints principaux ont été mis en place : l’un pour suivre un utilisateur, et l’autre pour s’en désabonner. Ces actions sont protégées par authentification et ne peuvent être effectuées que par des utilisateurs connectés. Enfin, des contrôles assurent qu’un utilisateur ne puisse pas se suivre lui-même.

* + - 1. **Gestion des likes**

La gestion des likes est directement liée au système de swipe, qui constitue le cœur de l’expérience utilisateur. Chaque swipe à droite enregistre un like pour un morceau, tandis qu’un swipe à gauche enregistre un dislike. Ces interactions rapides permettent de construire un profil musical pour chaque utilisateur et de nourrir les statistiques globales de la plateforme.

Les likes sont stockés dans la table Like, qui lie un utilisateur et un morceau. Une contrainte d’unicité empêche un utilisateur de liker plusieurs fois le même morceau, garantissant la cohérence des données.

L’API expose des endpoints permettant de créer un like ou un dislike, de consulter les morceaux likés par un utilisateur et de supprimer un like si nécessaire. Ces actions sont authentifiées afin de s’assurer que seuls les utilisateurs connectés peuvent interagir avec le catalogue musical.

* + - 1. **Gestions des notations**

En complément des likes, MusicSwipe permet aux utilisateurs de donner une note chiffrée aux morceaux qu’ils écoutent. Ce système offre une appréciation plus détaillée qu’un simple swipe et permet d’obtenir une moyenne représentative des avis de la communauté.

Les notations sont stockées dans la table Rating, qui associe un utilisateur, un morceau et une valeur de score. Là encore, une contrainte d’unicité empêche un utilisateur de noter plusieurs fois le même morceau. En cas de nouvelle notation, la valeur précédente peut être mise à jour.

Côté API, plusieurs fonctionnalités sont disponibles : ajouter une note, consulter les notes attribuées par un utilisateur, afficher la moyenne des notations pour un morceau et mettre à jour une note existante. Ces endpoints sont protégés par authentification, afin de garantir que seules les actions valides soient prises en compte.

Grâce à ce système, MusicSwipe combine la rapidité des interactions via le swipe et la richesse d’une notation détaillée, renforçant ainsi l’expérience utilisateur et la valeur des statistiques communautaires.

* + 1. **Sécurité backend**

La sécurité du backend de MusicSwipe a été une priorité tout au long du développement. L’objectif était de protéger les données des utilisateurs, de garantir l’intégrité des interactions et de limiter les risques liés aux vulnérabilités classiques des applications web.

* + - 1. **Hashage des mots de passe**

Lors de l’inscription, le mot de passe fourni par l’utilisateur n’est jamais stocké en clair dans la base de données. J’ai mis en place un système de hashage avec bcrypt, qui transforme le mot de passe en une empreinte sécurisée. Même en cas de fuite de données, les mots de passe restent donc inexploitables. Lors de la connexion, bcrypt compare l’empreinte stockée avec celle générée à partir du mot de passe saisi.

* + - 1. **Validations des entrés**

Toutes les données reçues par l’API sont validées avant d’être traitées. J’ai utilisé des DTOs (Data Transfer Objects) couplés au module class-validator de NestJS pour définir les règles de validation (formats d’e-mail, longueur minimale des mots de passe, types de données attendus, etc.). Ce mécanisme réduit les risques d’injections SQL, d’erreurs de type et d’attaques basées sur des données malformées.

* + - 1. **Protection des routes**

L’accès aux endpoints est protégé par des guards JWT. Chaque requête envoyée au backend doit contenir un token valide généré lors de la connexion. Ce token permet de vérifier l’identité de l’utilisateur et de déterminer ses droits. Les routes sensibles, comme celles réservées aux administrateurs, sont sécurisées par un contrôle supplémentaire sur le rôle de l’utilisateur.

* + - 1. **Sécurité BDD**

La base PostgreSQL a été conçue pour garantir l’intégrité et la cohérence des données grâce à l’utilisation de clés étrangères et de contraintes d’unicité. Par exemple, un utilisateur ne peut pas noter deux fois le même morceau, ni liker plusieurs fois le même titre. Les relations de type *cascade* ont été configurées afin que la suppression d’un utilisateur entraîne également la suppression des notations, likes, follows et commentaires associés, évitant ainsi la présence de données orphelines.

Enfin, des index ont été ajoutés sur les colonnes stratégiques (identifiants, relations) afin d’améliorer les performances et de réduire la surface d’attaque par déni de service lié aux requêtes lourdes.

* 1. **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.

* + 1. **Choix techniques et architecture**
       1. **Frameworks et langages**

Le développement du dashboard administrateur de MusicSwipe a été pensé pour être rapide à mettre en place, évolutif et facile à maintenir. L’objectif était de fournir une interface claire et performante permettant aux administrateurs de gérer efficacement les utilisateurs, les morceaux et l’ensemble des données de la plateforme.

J’ai choisi de m’appuyer sur Next.js comme framework principal, couplé à Refine, une surcouche spécialisée pour construire des backoffices et des outils d’administration. Cette combinaison m’a permis de bénéficier de la puissance de Next.js (rendu côté serveur, optimisation SEO, performances globales) tout en profitant de la rapidité de développement qu’offre Refine pour générer des pages de gestion (CRUD, listes, formulaires).

Le dashboard adopte une architecture orientée composants. Chaque ressource (utilisateurs, morceaux, notations, paramètres) correspond à un ensemble de pages et de composants réutilisables. Cette organisation rend l’interface plus modulaire et simplifie la maintenance. L’API backend est intégrée via le connecteur simple-rest de Refine, ce qui facilite les appels réseau et la synchronisation des données.

Enfin, l’interface est construite avec TailwindCSS, ce qui m’a permis d’obtenir un design moderne et responsive, sans devoir passer trop de temps sur la mise en forme. L’accent a été mis sur la simplicité et la clarté afin de rendre l’expérience administrateur fluide.

* + - 1. **Justification des choix**

J’ai choisi d’utiliser Next.js car c’est aujourd’hui l’un des frameworks les plus performants et les plus complets pour développer des applications React modernes. Il propose nativement le rendu côté serveur (SSR), le support du statique (SSG) et une gestion optimisée des routes. Cela me permet d’avoir une application rapide, bien structurée et facile à maintenir, tout en restant dans un environnement que je maîtrise déjà grâce à mon apprentissage de React.

L’ajout de Refine s’est imposé comme une évidence pour gagner du temps dans la construction du dashboard. Refine fournit des composants prêts à l’emploi pour la gestion CRUD, la navigation ou encore la gestion des ressources. Plutôt que de tout développer manuellement, j’ai pu m’appuyer sur ces outils afin de me concentrer sur les parties importantes du projet, comme l’intégration avec le backend et l’organisation des données.

L’utilisation de TailwindCSS complète parfaitement ce choix. Cet outil me permet de styliser rapidement mes interfaces avec un rendu moderne, sans avoir besoin de passer par des feuilles de style complexes. Je trouve que Tailwind s’intègre particulièrement bien avec l’architecture orientée composants de React et Next.js, ce qui rend le code plus propre et plus facile à maintenir.

Enfin, l’ensemble de ces choix répond aussi à un critère important : la maintenabilité et l’évolutivité. Avec Next.js, Refine et Tailwind, j’ai un socle solide et moderne qui me permet de faire évoluer facilement le dashboard à l’avenir, que ce soit en ajoutant de nouvelles fonctionnalités ou en améliorant le design.

* + - 1. **Organisation du projet**

L’organisation du projet administrateur a été pensée pour rester claire et modulaire, afin de faciliter la maintenance et l’évolution du code. J’ai suivi la logique proposée par Next.js, en combinant cette structure avec l’approche par ressources de Refine.

Chaque ressource de l’application correspond à un ensemble de pages et de composants dédiés. Par exemple, les utilisateurs, les morceaux et les paramètres sont chacun organisés dans leur propre espace, ce qui permet de séparer les responsabilités et de rendre le code plus lisible.

La structure suit principalement cette logique :

pages/ : répertoire principal géré par Next.js, contenant les routes de l’application. Chaque fichier ou dossier correspond directement à une URL. J’ai organisé ici les pages liées à l’authentification (login), au tableau de bord et aux différentes ressources.

resources/ : dossier utilisé pour déclarer les ressources Refine (users, tracks, settings). Chaque ressource est associée à ses composants CRUD (listes, création, édition, détails). Cela permet à Refine de générer automatiquement une navigation cohérente et de simplifier la gestion des données.

components/ : dossier où j’ai regroupé les composants réutilisables, comme les formulaires, les boutons personnalisés ou les cartes de statistiques. Cette organisation évite la duplication et permet de garder une cohérence visuelle dans l’ensemble du dashboard.

styles/ : contient la configuration de TailwindCSS et les éventuelles personnalisations pour ajuster l’apparence globale du dashboard.

utils/ : répertoire qui regroupe des fonctions utilitaires, notamment pour gérer les appels API via le connecteur simple-rest, ou pour centraliser certaines règles de validation côté frontend.

Cette organisation me permet de garder un projet bien structuré, où chaque fonctionnalité est isolée dans son propre espace. Elle rend aussi le code plus facile à maintenir, car il est simple de retrouver les fichiers associés à une ressource précise. Enfin, cela facilite l’ajout de nouvelles fonctionnalités, puisqu’il suffit de créer une nouvelle ressource et de l’intégrer dans la logique existante sans impacter le reste de l’application.

* + 1. **Fonctionnalités implémentées**
       1. **Page d’authentification**

La page d’authentification du dashboard administrateur constitue le point d’entrée de l’espace de gestion. Elle a été conçue pour être simple, claire et sécurisée.

L’administrateur accède à un formulaire où il doit renseigner son adresse e-mail et son mot de passe. Ces informations sont ensuite transmises au backend via une requête sécurisée. En cas de succès, un jeton JWT est généré et stocké côté client afin de maintenir la session. Ce token est systématiquement vérifié lors des appels API ultérieurs, ce qui garantit que seules les personnes autorisées peuvent accéder aux fonctionnalités du dashboard.

Une attention particulière a été portée à la gestion des erreurs. Si les identifiants saisis sont incorrects ou si l’utilisateur ne possède pas les droits d’administration, un message clair est affiché. L’objectif est de combiner une expérience utilisateur fluide avec un haut niveau de sécurité.

L’interface a été réalisée avec TailwindCSS pour assurer une mise en page moderne et responsive, adaptée aussi bien à un usage sur ordinateur qu’en mobilité. Bien que le dashboard soit principalement destiné au format desktop, j’ai veillé à ce que la page de login reste utilisable sur différents supports.

Cette page joue donc un rôle essentiel : elle filtre l’accès au dashboard et s’assure que seules les personnes disposant du rôle administrateur puissent gérer les utilisateurs, les morceaux et les données de l’application.

* + - 1. **Dashboard Principal**

Le dashboard principal constitue la page d’accueil de l’administrateur une fois connecté. Son rôle est de fournir une vue d’ensemble claire et synthétique sur l’activité de l’application afin de faciliter la supervision.

L’interface a été pensée pour être sobre et efficace. Dès l’arrivée sur cette page, l’administrateur accède à plusieurs indicateurs clés présentés sous forme de cartes et de graphiques.

L’objectif est que l’administrateur puisse, en un coup d’œil, évaluer la dynamique de la communauté et repérer d’éventuelles anomalies.

Le dashboard est également un point d’accès rapide aux autres sections de gestion. Des raccourcis permettent de naviguer directement vers la gestion des utilisateurs, des morceaux ou encore des paramètres. L’interface s’appuie sur la logique de Refine, ce qui simplifie la mise en place des ressources et permet d’actualiser les données en temps réel via l’API backend.

En résumé, le dashboard principal joue un rôle de tableau de bord stratégique pour l’administrateur : il centralise les statistiques essentielles, propose une navigation rapide vers les sections de gestion et permet de suivre en temps réel l’évolution de l’application.

* + - 1. **Dashboard Utilisateurs**

Le dashboard utilisateurs est une section clé, car il permet aux administrateurs de superviser et de gérer l’ensemble des comptes inscrits sur MusicSwipe.

L’interface se présente sous la forme d’une liste complète des utilisateurs, affichée dans un tableau clair. Chaque ligne du tableau regroupe les informations principales d’un utilisateur : son identifiant, son nom d’utilisateur, son adresse e-mail, son rôle (utilisateur classique ou administrateur) ainsi que la date de création du compte.

L’administrateur dispose de plusieurs actions possibles directement depuis cette interface, il peut consulter le profil complet d’un utilisateur et le rendre administrateur ou supprimer son compte si nécessaire.

Grâce à Refine, ce module bénéficie de composants CRUD déjà optimisés, ce qui simplifie la gestion et assure une cohérence avec le reste du dashboard. Les appels API sont sécurisés : seuls les administrateurs connectés avec un rôle valide peuvent accéder à cette section et exécuter ces actions.

Ce dashboard constitue un outil indispensable pour garantir la qualité et la sécurité de la communauté MusicSwipe. Il donne aux administrateurs la possibilité de superviser les activités des membres, de résoudre rapidement les problèmes et de maintenir un environnement sain au sein de l’application.

* + - 1. **Dashboard Tracks**

Le dashboard tracks est dédiée à la gestion du catalogue musical de MusicSwipe. Il permet aux administrateurs de superviser, d’ajouter et de modifier les morceaux présents dans la base de données, afin d’assurer un contenu de qualité et en adéquation avec les besoins des utilisateurs.

L’interface se présente sous forme d’un tableau listant l’ensemble des morceaux disponibles. Chaque ligne contient les principales informations liées à un morceau : son titre, son artiste, son album, sa durée et, lorsque disponible, un extrait audio. L’administrateur peut ainsi visualiser rapidement l’état du catalogue et intervenir si nécessaire.

Consulter les interactions associées, pour chaque morceau, l’administrateur peut accéder aux statistiques liées, comme le nombre de notations, de likes ou de commentaires.

Grâce à Refine, cette section repose sur des composants CRUD préconfigurés qui facilitent le développement et garantissent une cohérence avec le reste du dashboard. Les données sont synchronisées en temps réel avec l’API backend, et toutes les actions sont protégées par un système d’authentification JWT réservé aux administrateurs.

Le dashboard tracks joue donc un rôle essentiel, il assure le contrôle qualité du catalogue musical et offre aux administrateurs un outil fiable pour maintenir une base de données cohérente et pertinente.

* + - 1. **Page settings**

La page **Settings** du dashboard administrateur est dédiée à la gestion du compte personnel de l’administrateur connecté. Contrairement aux autres sections qui concernent la gestion globale de la plateforme, cette page se concentre uniquement sur les informations liées à l’administrateur lui-même.

Depuis cette interface, il peut consulter et modifier ses informations personnelles comme son nom d’utilisateur, son adresse e-mail, son mot de passe ou encore son avatar. Chaque modification passe par l’API backend et nécessite une authentification via JWT afin de garantir que seul l’administrateur connecté puisse mettre à jour ses propres données. J’ai également mis en place des validations côté frontend et côté backend pour m’assurer que les informations saisies respectent les formats attendus, comme une adresse e-mail valide ou un mot de passe suffisamment sécurisé.

L’interface a été conçue de manière simple et claire avec TailwindCSS, sous la forme d’un formulaire ergonomique permettant d’effectuer rapidement les changements nécessaires. Cette page renforce donc la sécurité et l’autonomie des administrateurs, puisqu’elle leur permet de gérer eux-mêmes leurs informations de connexion et de profil sans dépendre d’un autre compte.

* + 1. **Sécurité admin**

La sécurité du dashboard administrateur était un point crucial dans le développement, car il s’agit d’un espace sensible qui permet de gérer l’ensemble des utilisateurs, des morceaux et des données de la plateforme. J’ai donc mis en place plusieurs mécanismes afin de garantir que seules les personnes autorisées puissent y accéder et que les actions réalisées soient toujours protégées.

* + - 1. **Authentification admin**

L’accès au dashboard est strictement réservé aux comptes disposant du rôle administrateur. Lors de la connexion, l’authentification passe par un système basé sur JWT, généré par le backend après vérification des identifiants. Une fois connecté, l’administrateur reçoit un token qui doit être présent dans chaque requête envoyée au backend. Ce token est vérifié à chaque appel pour s’assurer que la session est valide et qu’elle correspond bien à un administrateur. En cas de tentative d’accès avec des identifiants incorrects ou avec un compte n’ayant pas les droits suffisants, l’accès est refusé et un message d’erreur clair est renvoyé. Cette gestion stricte permet de protéger l’espace d’administration contre toute tentative d’intrusion.

* + - 1. **Gestion des autorisations**

Au-delà de l’authentification, j’ai également mis en place un contrôle précis des autorisations. Chaque action disponible sur le dashboard est protégée par des guards qui vérifient le rôle et les permissions de l’utilisateur connecté. Par exemple, un utilisateur classique qui tenterait d’accéder directement à une route réservée aux administrateurs verrait sa requête bloquée automatiquement. Côté frontend, certaines sections comme la gestion des utilisateurs ou des morceaux ne sont même pas affichées si le compte connecté ne dispose pas des droits nécessaires. Cette double vérification, à la fois côté frontend et backend, garantit que les autorisations sont respectées en toutes circonstances. Elle permet d’éviter que des fonctionnalités sensibles soient accessibles par erreur et assure un contrôle complet de l’espace administrateur.

* 1. **Développement PWA**
     1. **Choix techniques et architecture**
        1. **Frameworks et langages**
        2. **Justification des choix**
        3. **Organisation du projet**
     2. **Fonctionnalités implémentées**
        1. **Authentification**
        2. **Swipe & flux de découverte**
        3. **Détail d’un morceau**
        4. **Profil & statistiques**
        5. **Recherche**
        6. **Suivre / ne plus suivre**
        7. **Paramètres**
     3. **Bilan de l’applicaiton**

1. **Déploiement et DevOps**
   1. **Environnement**
   2. **CI/CD**
   3. **Scripts de déploiement**
   4. **Monitoring**
2. **Veille technologie**
   1. **Sources et outils de veilles utilisés**
   2. **Vulnérabilités et correction**
3. **Bilan et conclusion**
   1. **Compétences acquises**
   2. **Perspectives d’évolution**