# Monolithique

users(userID, firstName, lastName, email, location, deezerID) userStatistics(userID, statistic, value)

profileFields(profileFieldID, name, desc) userProfileFields(profileFieldID, userID, content)

likedGenres(userID, genreID, strength) hatedGenres(userID, genreID, strength)

blockedUsers(blockerID, blockedID) userSocialMedias(userID, socialMediaID, profileLink)

socialMedias(socialMediaID, name, url) (url + profileLink = adresse)

genres(genreID, name, subgenreOf) artists(artistID, name)

musics(musicID, genreID, artistID, title)

playlists(playlistID, userID, name, desc) playlistMusics(playlistID, musicID) playlistSubscribers(playlistID, userID)

musicComments(musicCommentID, musicID, userID, replyTo, rating, content, date) (1 rating/user

+ music)

eventComments(eventCommentID, eventID, userID, replyTo, content, date) playlistComments(playlistCommentID, playlistID, userID, replyTo, content, date)

messages(messageID, senderID, receiverID, content, date, status)

events(eventID, organiserID, name, location, date, desc, status) (status = closed, open, private etc.) eventUsers(eventID, userID)

eventUserInvitations(eventID, inviterID, inviteeID, comment) eventGenreInvitation(eventID, genreID, comment) eventArtistInvitation(eventID, artistID, comment)

matches(matchID, userID1, userID2, status, compatibility float, feedbackUser1, feedbackUser2) badges(badgeID, name, desc)

userBadge(badgeID, userID, date, progress) logs(logID, dateTime, content)

# Architecture Générale:

# L'architecture générale du système repose sur une approche basée sur les microservices, favorisant la scalabilité, la flexibilité et la maintenabilité. Parmi quelques points supplémentaires à considérer pour enrichir la description :

# 1. Décomposition Fonctionnelle: Chaque microservice est conçu pour accomplir une tâche spécifique du domaine métier, ce qui favorise une décomposition fonctionnelle claire et une séparation des préoccupations. Par exemple, un microservice gère l'authentification des utilisateurs, un autre gère les profils des utilisateurs, et ainsi de suite.

# 2. Indépendance des Services: Les microservices sont conçus pour être indépendants les uns des autres, ce qui permet à chaque équipe de développement de se concentrer sur un seul domaine fonctionnel sans être affectée par les autres services. Cela favorise également une évolutivité horizontale, car chaque microservice peut être mis à l'échelle individuellement en fonction de la charge.

# 3. Évolutivité: L'architecture basée sur les microservices permet une évolutivité facile en raison de sa nature modulaire. Les nouvelles fonctionnalités peuvent être implémentées et déployées rapidement en ajoutant de nouveaux microservices ou en modifiant les existants, sans perturber le fonctionnement global du système.

# 4. Flexibilité Technologique: Chaque microservice peut être développé et déployé avec les technologies les plus adaptées à ses besoins spécifiques. Par exemple, un microservice peut utiliser Java avec Spring Boot, tandis qu'un autre peut préférer Node.js avec Express. Cette flexibilité technologique permet d'exploiter au mieux les avantages de chaque technologie pour résoudre des problèmes spécifiques.

# 5. Gestion des Erreurs et de la Tolérance aux Pannes: L'architecture des microservices met l'accent sur la résilience du système en isolant les erreurs au niveau des services individuels. Chaque microservice peut gérer ses propres erreurs de manière appropriée, ce qui limite l'impact des pannes sur l'ensemble du système.

# 6. Sécurité: La sécurité est intégrée à chaque microservice, avec des mécanismes d'authentification et d'autorisation appropriés. Par exemple, le microservice d'authentification utilise OAuth pour garantir l'identification sécurisée des utilisateurs, tandis que d'autres microservices appliquent des contrôles d'accès en fonction des rôles et des autorisations.

# En résumé, l'architecture générale basée sur les microservices favorise une conception modulaire, une évolutivité facile, une flexibilité technologique et une meilleure gestion des erreurs, ce qui en fait une approche adaptée aux systèmes complexes et évolutifs.

# Distribué

## Microservice Authentification (Oauth)

## Gère l'authentification et l'autorisation des utilisateurs.

*users(userID, firstName, lastName, email)*

## Microservice Profiles

## Gère les profils des utilisateurs, y compris les informations personnelles, les statistiques et les liens vers les réseaux sociaux

*- profileFields(profileFieldID, name, desc) userProfileFields(profileFieldID, userID, content)*

*- userStatistics(userID, statisticID, value) statistics(statisticID, name)*

*- userSocialMedias(userID, socialMediaID, profileLink) - socialMedias(socialMediaID, name, url) (url + profileLink = adresse)*

## Microservice Musique

## Responsable de la gestion des genres, des artistes, des pistes musicales, des listes de lecture et des abonnements.

*genres(genreID, name, subgenreOf) artists(artistID, name)*

*musics(musicID, genreID, artistID, serviceID, link, title) (serviceID = socialMediaID) playlists(playlistID, userID, name, desc)*

*playlistMusics(playlistID, musicID) playlistSubscribers(playlistID, userID)*

## Microservice Commentaires

## Gère les commentaires des utilisateurs sur la musique, les événements et les listes de lecture.

*musicComments(musicCommentID, musicID, userID, replyTo, rating, content, date) (1 rating/user*

*+ music)*

*eventComments(eventCommentID, eventID, userID, replyTo, content, date) playlistComments(playlistCommentID, playlistID, userID, replyTo, content, date)*

## Microservice Messagerie

messages(messageID, senderID, receiverID, content, date, status) blockedUsers(blockerID, blockedID)

## Microservice Evènements

events(eventID, organiserID, name, location, date, desc, status) (status = closed, open, private etc.) eventUsers(eventID, userID)

eventUserInvitations(eventID, inviterID, inviteeID, comment) eventGenreInvitation(eventID, genreID, comment) eventArtistInvitation(eventID, artistID, comment)

## Microservice Matching

likedGenres(userID, genreID, strength)

hatedGenres(userID, genreID, strength) → fusionné, strength positif/négatif

matches(matchID, userID1, userID2, status, compatibility float, feedbackUser1, feedbackUser2)

## Microservice Badges

badges(badgeID, name, desc)

userBadges(badgeID, userID, date, progress)

## Microservice Logs

logs(logID, dateTime, content)

Microservice des Événements : Gère les événements, les invitations et la participation des utilisateurs.

Microservice de Musique : Inclut l'identifiant de service et le lien pour les pistes musicales, permettant aux utilisateurs d'accéder et de diffuser de la musique à partir de sources externes.

**Microservice de Correspondances** : Fusionne les genres aimés et détestés, introduisant des forces positives et négatives pour les calculs de compatibilité

### Modèle de communication:

La communication entre les microservices peut être implémentée à l'aide d'API RESTful, de files d'attente de messages (par exemple, RabbitMQ, Kafka) ou d'architectures orientées événements (par exemple, Pub/Sub).

Chaque microservice expose des points de terminaison ou des événements pour communiquer avec d'autres services. La communication asynchrone peut être utilisée pour le découplage et la scalabilité, tandis que la communication synchrone peut être utilisée pour les interactions en temps réel.

### Modèle de stockage:

Le modèle de stockage utilise une combinaison de bases de données relationnelles et éventuellement NoSQL pour répondre aux différents besoins en données des microservices :

* Base de données relationnelle : Utilisée pour les données structurées telles que les profils utilisateur, les commentaires, les événements et les correspondances.
* Base de données NoSQL (par exemple, MongoDB, Redis) : Utilisée pour les données semi-structurées ou non structurées telles que les journaux, les préférences utilisateur et les données temporaires.
* Stockage d'objets (par exemple, Amazon S3, Azure Blob Storage) : Adapté pour stocker des fichiers multimédias tels que des pistes musicales et du contenu téléchargé par l'utilisateur.

Le choix de la technologie de stockage dépend de facteurs tels que la cohérence des données, la scalabilité et les exigences de performance de chaque microservice.

# Topics

|  |  |  |
| --- | --- | --- |
| **Topic** | **Producer** | **Consumer** |
| UserCreation | Authentification | Profiles |
| UserListensToMusic | Musique | Profiles, Matches |
| UserStatisticsHasChanged | Profiles | Badges |
| UserComments | Commentaires | Messagerie |
| UserAttendsEvent | Evènements | Matches |
| UserSubscribesToPlaylist | Musique | Matches, Messagerie |
| UserNews | Badges, Evènements | Messagerie |
| UserLogsIn | Authentification | Profiles, Evènements, Matches |
| UserCreatesPlaylist | Musique | Matches |
| UserCreatesEvent | Evènements | Matches |
| UserInvitesToEvent | Evènements | Messagerie |
| LogIsCreated | All minus Logs | Logs |