**SUJET :**

Conception d’une application musicale intelligente adaptée à l’humeur de l’utilisateur

**PLAN :**

**I. Problématique**

Difficulté à trouver des playlists adaptées à notre état d’esprit.

Utilisation inefficace des plateformes musicales actuelles (manque d’émotion, personnalisation faible).

Opportunité de croiser IA et musique pour améliorer l’expérience utilisateur.

**II. Objectif de l’application**

Créer une application qui détecte l’humeur via la caméra, la voix ou les interactions de l’utilisateur.

Générer une playlist personnalisée selon cette humeur (joie, tristesse, colère, calme…).

S’adapter aux préférences musicales de l’utilisateur avec le temps.

**III. Étapes du processus de développement logiciel**

(Selon la méthode agile, par exemple Scrum)

**a. Recueil des besoins fonctionnels et non-fonctionnels**

Détection de l’humeur via IA (faciale, vocale, ou questionnaires)

Connexion à une API musicale (ex. : Spotify, Deezer)

Création de playlists

Stockage des préférences

UI/UX intuitive

**1. Besoins fonctionnels**

Les besoins fonctionnels décrivent ce que l'application doit faire. Voici ceux qui sont les plus pertinents pour ton sujet :

🎯 Fonctionnalités principales :

-Analyser l’humeur de l’utilisateur

Par reconnaissance faciale (via caméra),

Ou via analyse vocale (ton de voix),

Ou via questionnaire interactif.

-Générer une playlist personnalisée

En se basant sur l’humeur détectée,

En se connectant à une API musicale (Spotify, Deezer…).

-Afficher la playlist à l’utilisateur

Interface simple et intuitive pour écouter les morceaux.

-Permettre à l’utilisateur de donner un feedback

L’utilisateur peut indiquer s’il aime ou non la playlist,

Cela améliore les futures recommandations.

-Sauvegarder les préférences de l’utilisateur

Humeurs passées,

Genres musicaux préférés.

-Créer un compte utilisateur (connexion/inscription)

Pour stocker les préférences et l’historique.

**b. Choix de la méthodologie : Agile (Scrum)**

Justification : sprints rapides, feedback utilisateur, adaptation facile.

Organisation de l’équipe : développeur, UX designer, spécialiste IA, etc.

**c. Architecture fonctionnelle**

Module de détection d’humeur (via caméra ou micro ou saisie manuelle)

Moteur de recommandation (IA ou algorithme)

Générateur de playlists

Interface utilisateur

**d. Choix technologiques (à adapter)**

Frontend : Flutter / React Native

Backend : Node.js

IA : TensorFlow

Base de données : Firebase ou MongoDB

APIs musicales : Spotify API

**e. Tests et validation**

Tests fonctionnels

Tests UX (expérience utilisateur)

Feedback utilisateur à chaque sprint

**IV. Aspects innovants**

Intégration d’une IA émotionnelle dans le processus logiciel

Personnalisation en temps réel

Réinvention de l’expérience musicale

Peut être étendu à la santé mentale (musicothérapie)

**V. Limites et perspectives**

Défis techniques (précision de la détection d’humeur)

Respect de la vie privée (traitement d’images ou voix)

Possibilités d’évoluer vers :

Application de méditation,

Assistance émotionnelle,

Intégration domotique (musique ambiante selon humeur).

**Introduction :**

Aujourd’hui, la musique joue un rôle central dans notre bien-être émotionnel. Mais et si une application pouvait automatiquement détecter notre humeur et nous proposer une playlist qui y correspond ? C’est l’objectif de cette application intelligente. Dans cet exposé, nous allons explorer comment mettre en place un **processus de développement logiciel** innovant pour concevoir cette solution, alliant **agilité**, **intelligence émotionnelle**, et **personnalisation**.

**Conclusion**

À travers cet exposé, nous avons démontré comment un processus de développement logiciel bien structuré — notamment en mode agile — permet de transformer une idée innovante en solution concrète. L’intégration de l’intelligence émotionnelle dans une application musicale est non seulement utile, mais ouvre également la voie à de nouvelles formes d’interaction homme-machine, plus humaines et personnalisées.

**5. Bonus :** Visuel ou schéma suggéré

Tu peux créer (ou je peux t’aider à créer) un schéma fonctionnel de l'application :

**Utilisateur → Détection humeur → Moteur IA → Playlist générée → Interface de lecture**

**1. Patients en hôpitaux psychiatriques**

* **Utilité :** Apaiser les états émotionnels (anxiété, dépression, agitation).
* **Avantage :** Musique adaptée à leur état mental sans intervention constante du personnel médical.
* **Application :** Thérapie complémentaire non médicamenteuse.

**🔹 2. Personnes âgées en maison de retraite**

* **Utilité :** Briser la solitude, raviver des souvenirs (musique nostalgique).
* **Avantage :** Améliore la qualité de vie, stimule la mémoire (notamment pour les patients Alzheimer).
* **Application :** Routines musicales personnalisées selon les émotions ou moments de la journée.

**🔹 3. Enfants et bébés (crèches, maternités)**

* **Utilité :** Apaiser les pleurs, favoriser l’endormissement, stimuler les émotions positives.
* **Avantage :** Créer un environnement sonore rassurant et interactif.
* **Application :** Playlist douce quand bébé pleure, musique rythmée quand il est actif, etc.

**🔹 4. Travailleurs en stress (open space, métiers sous pression)**

* **Utilité :** Réduction du stress, concentration, motivation.
* **Avantage :** Playlist adaptée à l’humeur au travail : concentration, détente ou boost.
* **Application :** Intégration dans les casques connectés ou ambiances sonores de bureaux.

**🔹 5. Personnes malvoyantes ou non-voyantes**

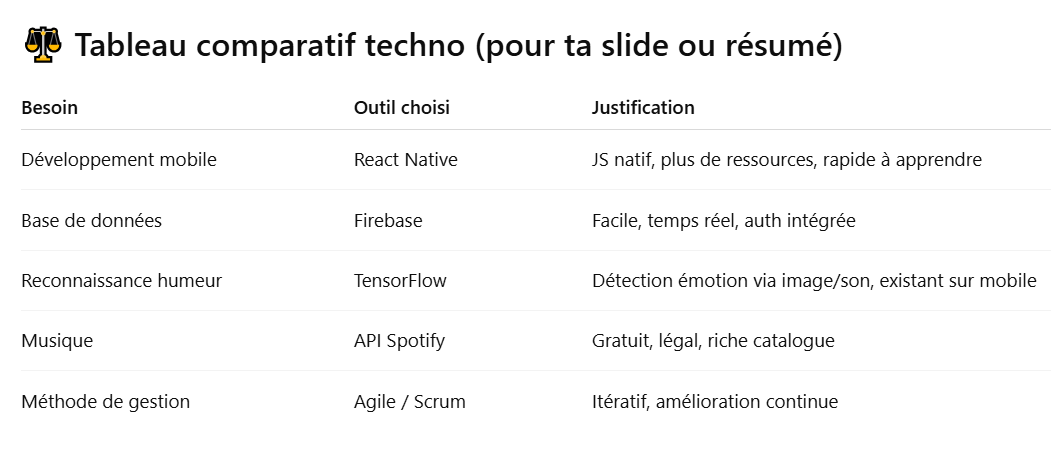
* **Utilité :** Accès intuitif et vocal à une musique adaptée à leur ressenti.
* **Avantage :** Interaction via commandes vocales et retour auditif basé sur les émotions.
* **Application :** Interface 100 % vocale et personnalisée.

**🔹 6. Étudiants en détresse émotionnelle (dépression, burn-out)**

* **Utilité :** Soulagement des troubles émotionnels par des ambiances musicales adéquates.
* **Avantage :** Peut aider à gérer les émotions sans avoir à verbaliser ou expliquer.
* **Application :** Intégration possible avec des services de soutien psychologique.

**🔹 7. Utilisateurs en situation de handicap mental ou autisme**

* **Utilité :** Régulation émotionnelle, amélioration du comportement.
* **Avantage :** Interaction simple, sans surcharge sensorielle.
* **Application :** Playlist adaptée à leurs sensibilités sensorielles et à leur humeur du moment.



## PARTIE III : Étapes du Processus de Développement Logiciel (Méthode Agile – SCRUM)

### 🟨 1. ****Analyse des besoins (ou recueil des exigences)****

#### 👉 Ce qu’on fait :

Identifier les **utilisateurs cibles** (patients psychiatriques, personnes âgées, bébés, étudiants stressés…).

Déterminer leurs **besoins spécifiques** :

Reconnaître l’humeur.

Proposer une playlist adaptée.

Avoir une interface simple, parfois vocale.

#### ✅ Pourquoi c’est important :

#### Cette étape pose ****la base fonctionnelle**** du projet. Sans elle, le logiciel risque de ne pas répondre aux attentes réelles.

### 🟨 2. ****Spécification fonctionnelle et technique****

#### 👉 Ce qu’on fait :

Rédiger des **cas d’usage** (user stories) :

"En tant qu’utilisateur triste, je veux que l’application me propose de la musique douce pour me réconforter."

Définir les **fonctions clés** :

Détection de l’humeur via l’appareil photo (reconnaissance faciale),

Appel à une API musicale (Spotify),

Interface utilisateur dynamique.

#### ✅ Pourquoi on choisit la ****méthode Agile (Scrum)**** :

Permet de **diviser** le projet en **sprints courts (1 à 2 semaines)**.

Favorise l'**amélioration continue**.

Encourage la **communication d’équipe** et les **retours rapides**.

### 🟨 3. ****Conception (Design)****

#### 👉 Ce qu’on fait :

**Maquettes UI/UX** (sur Figma par exemple).

Architecture de l’application :

Composants front-end (affichage),

Composants back-end (analyse humeur, connexion API),

Base de données (mood logs, préférences utilisateur).

#### ✅ Justification technique :

**React Native** (vs Flutter) :

Plus facile si tu as des bases en **JavaScript**.

Large communauté, nombreux composants prêts à l’emploi.

**Firebase** (vs MongoDB) :

Plus simple pour débutants, surtout pour l’authentification, stockage cloud, analytics.

Intégré facilement à une app mobile avec React Native.

### 🟨 4. ****Implémentation (développement)****

#### 👉 Ce qu’on fait :

Développer les composants selon les **sprints Scrum** :

Sprint 1 : interface simple.

Sprint 2 : reconnaissance faciale.

Sprint 3 : appel API Spotify.

Sprint 4 : recommandations basées sur l’humeur.

#### ✅ Justification :

**API Spotify** : accès légal à un catalogue de musique, possibilité de générer des playlists.

**TensorFlow.js** ou **TensorFlow Lite** :

Pour analyser les expressions faciales et détecter l’humeur.

Exemples de modèles pré-entraînés existent.

### 🟨 5. ****Tests****

#### 👉 Ce qu’on teste :

**Tests fonctionnels** : est-ce que la playlist est bien générée selon l’humeur ?

**Tests utilisateurs** : l’application est-elle facile à utiliser pour une personne âgée ?

**Tests de performance** : le temps de réponse est-il rapide ?

#### ✅ Importance :

Les tests assurent que l’application est **fiable et agréable à utiliser**.

### 🟨 6. ****Déploiement****

#### 👉 Ce qu’on fait :

Mettre l’application sur le **Google Play Store** ou l’**Apple App Store**.

Gérer la configuration de Firebase pour la production.

Proposer des mises à jour automatiques après chaque sprint.

### 🟨 7. ****Maintenance et évolutions****

#### 👉 Ce qu’on prévoit :

Corriger les bugs signalés.

Ajouter des fonctionnalités selon les retours utilisateurs (mode nuit, playlist offline, etc.).

Adapter les recommandations selon les saisons, événements ou habitudes de l’utilisateur.

**Docker** est un outil fondamental en DevOps et tu peux l’utiliser dès maintenant pour :

uniformiser l’environnement de dev/test/déploiement,

faciliter l’exécution de ton backend Spring Boot (et éventuellement d’autres services comme une base de données),

préparer ton application pour le cloud.

**Dockerfile**

# Utilise une image Java officielle

# Répertoire de travail dans le conteneur

# Copier le fichier jar dans le conteneur (remplace "nom-fichier.jar" après build)

# Expose le port utilisé par ton application (souvent 8080)

# Commande pour exécuter l’application

Un **pipeline CI/CD** (Continuous Integration / Continuous Deployment) est un ensemble d'étapes automatisées qui permettent de tester, construire et déployer une application.

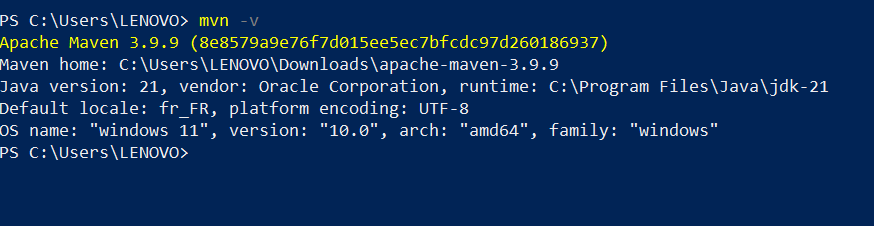
Exemple :

Tu écris du code et le pousses sur GitHub.

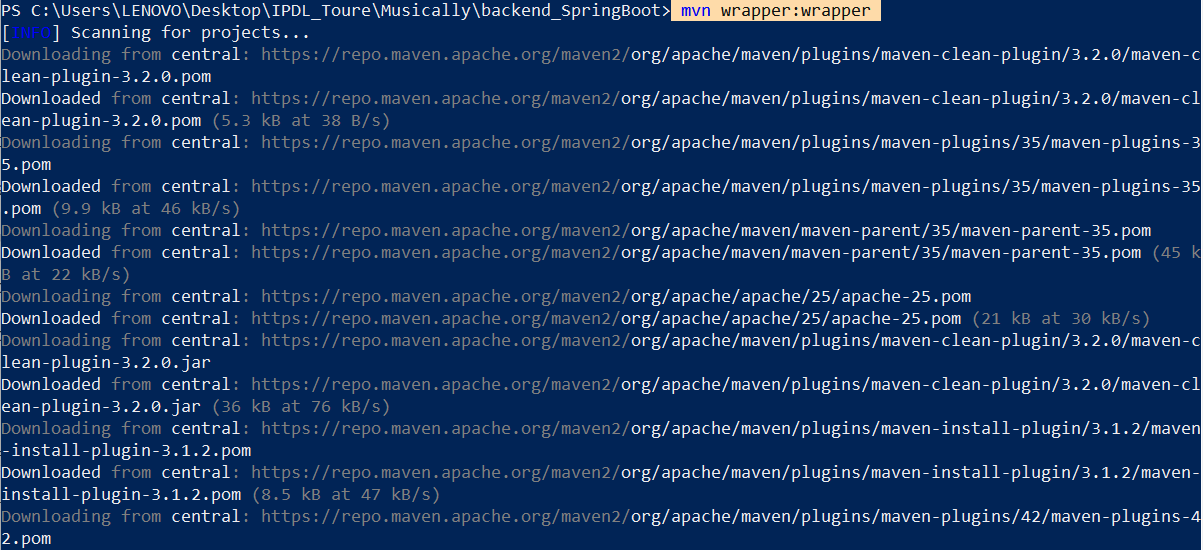
Le pipeline se déclenche :

Exécution des tests, Compilation du code, Création d'un fichier exécutable ou d'un conteneur Docker, Déploiement sur un serveur

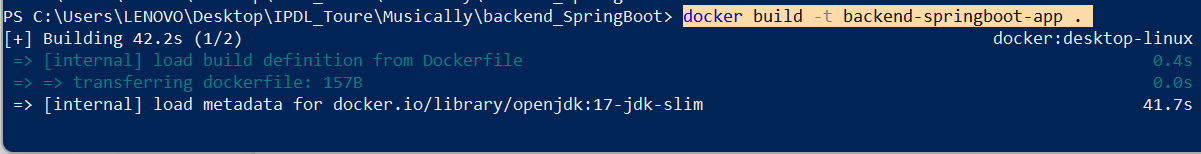
➡️ Cela permet de livrer plus rapidement des logiciels fiables.



Generer les fichiers wrapper :



Construire l’image Docker :



Lancer le conteneur :

Une fois le **build créé**, à quoi ça sert en DevOps ?

Quand tu fais un build avec Maven (ex. ./mvnw clean package -DskipTests), cela signifie que tu transformes ton code source Java + config en un fichier exécutable comme :

un **.jar** (Java Archive) pour une application Spring Boot

Ce **.jar** généré sert a :

1. Exécuter l'application localement

Tu peux tester l'app avec :

**java -jar target/nom-du-fichier.jar**

Le packager dans un conteneur Docker : le .jar est souvent utilisé dans un Dockerfile.

Le .jar peut être déployé : dans un conteneur Docker

S’assurer que le workflow GitHub Actions fonctionne : pas d’erreurs, bonne configuration Java/Maven/Docker,

En tant que responsible DevOps, je veux que chaque push fasse un build/test automatiquement. C’est plus facile de l'ajouter dès le départ.

Au fur a mesure, on mettra à jour le Dockerfile, workflows, etc, a la fin du projet on fera le Build complet + Docker + déploiement automatique



ddd