Application de Gestion et Lecture de Musique Locale

Contexte du projet :

"MusicStream" est une application musicale simple basée sur Angular, offrant une expérience basique aux utilisateurs pour écouter et organiser leur musique locale. Pour assurer une gestion efficace des états du lecteur et des tracks, nous adoptons des services Angular avec RxJS/Signals comme solution de gestion d'état.

L'objectif principal est de créer une application musicale simple et fonctionnelle, offrant une expérience utilisateur fluide et une architecture maintenable, tout en assurant une gestion efficace des fichiers audio locaux.

Fonctionnalités

- Gestion des tracks
- Système CRUD complet incluant pour chaque track :
  Nom de la chanson
  Nom du chanteur
  Description optionnelle (max 200 caractères)
  Date d'ajout (automatique)
  Durée de la chanson (calculée automatiquement)
  Catégorie musicale (pop, rock, rap, etc.)

Pages principales requises

- Page Bibliothèque : Liste complète des tracks avec barre de recherche et quelques filtres
- Page Track : Affichage détaillé et lecture du track sélectionné
- Lecteur audio
  Développer les contrôles essentiels (play, pause, next, previous)
  Implémenter le contrôle du volume et de la progression
  Utiliser Web Audio API ou HTMLAudioElement
- Autres pages optionnelles selon votre vision du projet

Tâches Principales

1. Configuration initiale

- Mettre en place la bonne structure de base Angular
- Structurer les composants principaux selon les bonnes pratiques
- Configurer le routing avec lazy loading

2. Gestion d'état avec Services

AudioPlayerService : Gestion de l'état du lecteur

- États : 'playing', 'paused', 'buffering', 'stopped'
- Contrôles : play, pause, next, previous, volume, progression

- Utilisation de BehaviorSubject ou Signals pour les états réactifs

TrackService : Gestion CRUD des tracks

- États de chargement : 'loading', 'error', 'success'
- Opérations : create, read, update, delete
- Communication avec le StorageService

StorageService : Persistance des données

- Gestion du stockage des métadonnées et des fichiers audio
- Gestion des erreurs de lecture/écriture
- Interface uniforme pour les opérations de persistance

Autres services que vous voyez pertinents de mettre

3. Gestion des fichiers audio

Mettre en place un système de persistance côté client pour stocker :

- Les fichiers audio
- Les métadonnées associées (informations sur les tracks)

Vous êtes libre de choisir la technologie de stockage la plus adaptée selon vos besoins :

- IndexedDB (recommandé pour fichiers volumineux)
- localStorage
- sessionStorage
- Ou toute autre solution de stockage côté client

Contraintes :

- Limiter la taille des fichiers à 10MB maximum
- Supporter les formats MP3, WAV et OGG

4. Validations

- Limites de caractères (titre: 50, description: 200)
- Validation des formats de fichiers (audio et images)
- Gestion des erreurs de upload/stockage
- Messages d'erreur UI correspondants aux différents états
- Autres validations pertinentes

Technos et exigences techniques :

- Angular 17 ou plus
- RxJS/Observables pour la gestion réactive
- Signals (optionnel mais recommandé pour Angular 17+)
- TypeScript
- Reactive Forms
- Injection de dépendance

- Routing avec lazy loading
- Bootstrap ou Tailwind CSS ou autres
- injection de dependance

Concepts Angular à utiliser

- Components
- Modules
- Services avec BehaviorSubject/Signals
- Form handling
- Data binding
- Pipes
- Observables & Async Pipe
- Routing avec lazy loading

Bonus :

- Ajouter pour chaque track une image de couverture optionnelle (formats acceptés : png, jpeg)
- Drag & drop pour réorganiser les tracks dans la bibliothèque
- Tests unitaires et d'intégration (Jasmine/Karma)
- Intégration API de lyrics pour afficher les paroles d'une chanson en cours
- Configuration Docker

Livrables :

- Lien GitHub pour le code source
- Lien Jira
- Fichier README.md

Modalités pédagogiques :

- Projet individuel
- Durée du projet : 10 jours
- du 05/01/26 à 16/01/26
- Deadline d'envoi des rendus : 16/01/26 avant minuit

Critères de performance

- L'application doit implémenter toutes les fonctionnalités CRUD
- L'architecture Angular avec services, RxJS/Signals et lazy loading doit être respectée
- Les validations (formulaires, taille fichiers, formats audio) et gestion d'erreurs doivent être opérationnelles
- Le code doit être propre et respecter les standards TypeScript/Angular