TP Flutter - Netflim - Liste à regarder

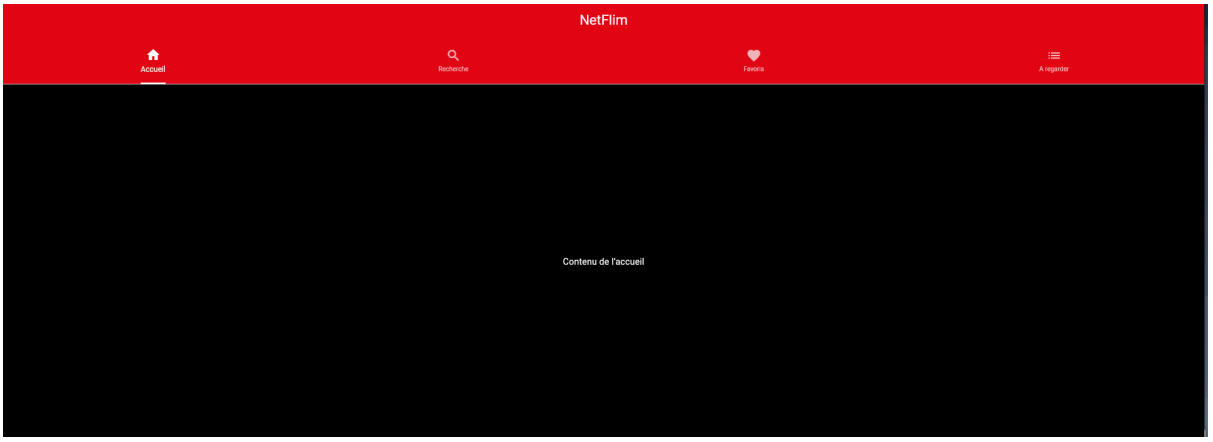
1- Gestion de la liste de films à regarder

1.1 Ajout d'une TabBar

Modification de main.dart pour afficher la tab bar :

```
switch (_loadingStatus) {
  case LoadingStatus.success:
    screen = AppTabController(
      popularMovies: _movies!,
    );
    break;
  default:
```

```
return MaterialApp(
  debugShowCheckedModeBanner: false,
  theme: AppTabController.theme(),
  home: screen,
);
```



1.2 Intégration de la liste des films populaires sur l'accueil

Ajout d'un appel à la classe MovieListScreen depuis le controller :

```
body: TabBarView(
  controller: _tabController,
  children: [
    Container(
      color: UserPreferences().backgroundColor,
      child: Center(child: MovieListScreen(movies: widget.popularMovies)),
    ), // Container
```

Afin d'avoir la liste qui s'affiche correctement, il suffit de supprimer la barre de titre pour éviter d'avoir un dédoublement de celle- ci.

1.3 Création d’un singleton pour la liste de films à regarder

```dart
// Importe shared_preferences pour le stockage persistant des données.
import 'package:flutter/material.dart';
import 'package:shared_preferences/shared_preferences.dart';
import '../models/movie.dart';

/// UserPreferences utilise le modèle Singleton pour gérer les
préférences.
///
/// Permet le stockage et la récupération persistants des préférences
utilisateur.
class ListToWatch {
 // Instance unique privée de UserPreferences pour le modèle Singleton.
 static final ListToWatch _instance = ListToWatch._internal();

 // Factory constructor retournant l'instance unique.
 factory ListToWatch() {
   return _instance;
 }

 // Constructeur privé pour l'initialisation de l'instance Singleton.
 ListToWatch._internal();

 // Référence privée à SharedPreferences pour le stockage clé-valeur.
 SharedPreferences? _prefs;

 /// Initialise SharedPreferences. Doit être appelé avant toute
opération.
 Future<void> init() async {
   _prefs = await SharedPreferences.getInstance();
 }

 /// Ajoute un film à la liste des films
 addToList(Movie movie) {
   // Récupération de la liste des liste des films depuis les
SharedPreferences.
   // Si aucune liste n'est stockée, on initialise une liste vide par
défaut.
   List<String> listToWatch = _prefs?.getStringList('ListToWatch') ??
[];
   // On utilise l'id du film comme clé dans le tableau
```

```
    final movieId = movie.id.toString();
    // L'identifiant du film est inconnu?
    if (!listToWatch.contains(movieId)) {
      // Ajout du film à la liste
      listToWatch.add(movieId);
      // Sauvegarde persistante via les préférences
      _prefs?.setStringList('ListToWatch', listToWatch);
    }
  }

  /// Vérifie si un film est dans les liste des films
  bool isInList(Movie movie) {
    // Récupération de la liste des liste des films depuis les
SharedPreferences.
    // Si aucune liste n'est stockée, on initialise une liste vide par
défaut.
    List<String> listToWatch = _prefs?.getStringList('ListToWatch') ??
[];
    // Conversion de l'ID du film en chaîne de caractères.
    final movieId = movie.id.toString();
    // Vérifie si l'identifiant du film est présent dans la liste des
liste des films.
    // Retourne true si l'identifiant est trouvé, sinon false.
    return listToWatch.contains(movieId);
  }

  /// Supprime un film des liste des films
  removeFromList(Movie movie) async {
    List<String> listToWatch = _prefs?.getStringList('ListToWatch') ??
[];
    final movieId = movie.id.toString();

    if (listToWatch.contains(movieId)) {
      listToWatch.remove(movieId);
      _prefs?.setStringList('ListToWatch', listToWatch);
    }
  }

  List<String> list() {
    List<String> idList = list();
    List<String> listToWatch = _prefs?.getStringList('ListToWatch') ??
[];
    for (var item in listToWatch) {
```

```
      idList.add(item);
    }
    return idList;
  }
}
```

1.4 Mise en “À regarder” dans la fiche détails d’un film

Après avoir ajouté le code pour afficher la pop up tout fonctionne à l'exception que le code dans l’état ne n’ajoute pas de film dans la liste, ça ne fait qu’afficher une popup.

```
onAddToListPressed: () {
  showDialog(
    context: context,
    builder: (BuildContext context) {
      if (!ListToWatch().isInList(movie)) {
        return AlertDialog(
          title: const Text('Confirmer l\'ajout'),
          content: const Text(
              'Voulez-vous ajouter ce film à votre liste à regarder ?'), // Text
          actions: <Widget>[
            TextButton(
              // Ferme la boîte de dialogue sans rien faire
              onPressed: () {···
              child: const Text('Annuler'),
            ), // TextButton
            TextButton(
              onPressed: () {
                // Logique pour ajouter le film à la liste

                ListToWatch().addToList(movie);
                print('Ajouté à la liste');
                print(ListToWatch().list());

                // Ferme la boîte de dialogue après l'action
                Navigator.of(context).pop();
              },
              child: const Text('Confirmer'),
            ), // TextButton
          ], // <Widget>[]
        ); // AlertDialog
      } else {
        return const AlertDialog(
            title: Text('Ajout impossible'),
            content: Text(
                'Le film est déja présent dans vôtre liste des films à regardé')); // Text // AlertDialog
      }
    },
  );
```

1.5 Gestion dynamique des ongles

```dart
import 'package:flutter/material.dart';
import 'package:netflim/services/api.dart';
import 'package:netflim/services/api_service.dart';
import '../models/movie.dart';
import '../services/user_preferences.dart';
import '../screens/popular_movies_screen.dart';

/// Contrôleur permettant l'affichage des onglets: Accueil, Recherche,
Favoris,
/// et à regarder.
class AppTabController extends StatefulWidget {
 /// Liste des films populaires chargés à l'initialisation de
l'application
 final List<Movie> popularMovies;

 /// Thème spécifique pour la TabBar
 static ThemeData theme() {
   return ThemeData(
     primaryColor: UserPreferences().backgroundColor,
     tabBarTheme: TabBarThemeData(
       labelColor: UserPreferences().mainTextColor,
       unselectedLabelColor:
UserPreferences().mainTextColor.withOpacity(0.7),
       indicator: UnderlineTabIndicator(
         borderSide: BorderSide(
           color: UserPreferences().mainTextColor,
           width: 3.0,
         ),
       ),
     ),
   );
 }

 /// Constructeur de notre widget
 const AppTabController({
   super.key,
   required this.popularMovies,
 });

 /// Surcharge de la gestion d'état
 @override
 _AppTabControllerState createState() => _AppTabControllerState();
```

```dart
}

class _AppTabControllerState extends State<AppTabController>
    with SingleTickerProviderStateMixin {
  TabController? _tabController;
  late Future<List<Movie>?> _moviesFuture;

  @override
  void initState() {
    super.initState();
    _tabController = TabController(length: 4, vsync: this);
    // Ajout de l'écouteur
    _tabController!.addListener(_handleTabSelection);
    // Initialisation de la liste de films par défaut
    _moviesFuture = loadMovies();
  }

  /// Méthode asynchrone chargeant la liste de films à afficher
  Future<List<Movie>?> loadMovies() async {
    switch (_tabController!.index) {
      // Recherche
      case 1:
        return [];
      // Favoris
      case 2:
        return [];
      // A regarder
      case 3:
        return [];
      default:
        return [];
    }
  }

  void _handleTabSelection() {
    if (_tabController!.indexIsChanging) {
      setState(() {
        _moviesFuture = loadMovies();
      });
    }
  }

// Construction du contexte
```

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: const Center(child: Text('NetFlim')),
      backgroundColor: UserPreferences().netflimColor,
      foregroundColor: UserPreferences().mainTextColor,
      bottom: TabBar(
        controller: _tabController,
        tabs: const [
          Tab(icon: Icon(Icons.home), text: 'Accueil'),
          Tab(icon: Icon(Icons.search), text: 'Recherche'),
          Tab(icon: Icon(Icons.favorite), text: 'Favoris'),
          Tab(icon: Icon(Icons.list), text: 'A regarder'),
        ],
        labelStyle: const TextStyle(fontSize: 12),
        unselectedLabelStyle: const TextStyle(fontSize: 10),
      ),
    ),
    body: TabBarView(
      controller: _tabController,
      children: [
        Container(
          color: UserPreferences().backgroundColor,
          child: Center(child: MovieListScreen(movies:
widget.popularMovies)),
        ),
        Container(
          color: UserPreferences().backgroundColor,
          child: const Center(
            child: Text(
              'Contenu de la recherche',
              style: TextStyle(color: Colors.white),
            ),
          ),
        ),
        Container(
          color: UserPreferences().backgroundColor,
          child: const Center(
            child: Text(
              'Contenu des favoris',
              style: TextStyle(color: Colors.white),
            ),
```

```
            ),
          ),
          Container(
            color: UserPreferences().backgroundColor,
            child: const Center(
              child: Text(
                'Contenu de la Liste',
                style: TextStyle(color: Colors.white),
              ),
            ),
          ),
        ],
      ),
    );
  }
}
```

1.6 Affichage de la liste des films à regarder

Version Anglaise :

In Flutter, the *FutureBuilder* Widget is used to create widgets based on the latest snapshot of interaction with a Future. It is necessary for Future to be obtained earlier either through a change of state or change in dependencies. *FutureBuilder* is a Widget that will help you to execute some asynchronous function and based on that function's result your UI will update.

FutureBuilder is *Stateful* by nature i.e it maintains its own state as we do in *StatefulWidgets*.

Version traduire en Français :

Dans Flutter, le Widget FutureBuilder est utilisé pour créer des widgets basé sur la dernière snapshot de l'interaction avec un Future. Il est nécessaire pour un Future d'être obtenu soit par un changement d'état ou un changement de dépendance. FutureBuilder est un Widget qui va nous aider à exécuter des fonctions asynchrone et basé sur cette fonction en résulte nôtre mise à jour de l'UI.

La nature de FutureBuilder est d'être Stateful, il gère son propre état comme nous le faisons avec les StatefulWidgets.

source : https://www.geeksforgeeks.org/flutter/flutter-futurebuilder-widget/

Après avoir ajouté plusieur film dans la liste des films à regarder, aucun film n'est chargé dans l'onglet des film à regardé.

```dart
/// Méthode asynchrone chargeant la liste de films à afficher
Future<List<Movie>?> loadMovies() async {
  switch (_tabController!.index) {
    // Recherche
    case 1:
      return [];
    // Favoris
    case 2:
      return [];
    // A regarder
    // A regarder
    case 3:
      // Récupérer la liste des identifiants
      List<String> movieIds = ListToWatch().list();
      // Créer une liste vide pour stocker les films
      List<Movie> movies = [];
      // Boucler sur chaque identifiant pour récupérer les films
correspondants
      for (String movieId in movieIds) {
        // Appeler l'API pour chaque film et attendre le résultat
        Movie? movie = await
ApiService().getMovie(int.parse(movieId));

        // Ajouter le film à la liste des films
        if (movie != null) movies.add(movie);
      }
// Retourner la liste complète des films
      return movies;
    default:
      return [];
  }
}
```

Après ajout du cas 3, on peut maintenant apprécier la présence des films dans l'onglet a regarder.