# MOSTRANT DADES DE MOVIE_DB

Ainhoa Sánchez Salvago

SALESIANS SARRIÀ | MÒBILS | S2AM

# ÍNDEX

# 1. MODIFICACIÓ DE L'APLICACIÓ

Un cop hem entès el codi original, podem començar a modificar el codi per tal de transformar l'aplicació. En primer lloc, ens crearem un compte per tal de poder accedir a l'API i un cop ens accepten la sol·licitud, ens enviaran una "key" amb la qual podrem accedir a les dades de l'API.

## 1.1. CARPETA API

Aquesta carpeta és el nostre pont entre l'aplicació i l'API, la seva única funció és portar les dades que ens facilita l'API a la nostra aplicació. Comencem modificant aquesta carpeta, per configurar la connexió. Dins d'aquesta carpeta trobem tres fitxers: "api_end_points.dart", "api_service.dart" i "api.dart".

### 1.1.1. API_END_POINTS.DART

Dins d' aquest arxiu, construïm un seguit de direccions URLs exactes a les que la nostra aplicació cridarà.

```dart
import 'package:movies/api/api.dart';

// cosntrucció de les URLs finals per les peticions de l'ApiService
// Utilitzarem en tots els mètodes statics per facilitar l'accés sense instanciar --> ApiEndPoints.getTopRatedMovies()
class ApiEndPoints {
  static const products = "products";
  static const popularMovies = "movie/popular";
  static const upcomingMovies = "movie/upcoming";
  static const getGenreList = "genre/movie/list";
  static const popularPersons = "person";

  // Top Rated movies
  static String getTopRatedMovies() =>
      '${Api.baseUrl}movie/top_rated?api_key=${Api.apiKey}&language=es-ES&page=1';

  // Endpoint flexible: rep la ruta d'allò que es vol veure (popular, upcoming, etc.)
  static String getCustomMovies(String endpoint) =>
      '$endpoint?api_key=${Api.apiKey}&language=es-ES&page=1';

  // Actors populars --> comença a la pàgina 1 per defecte
  static String getPopularActors() =>
      '${Api.baseUrl}person/popular?api_key=${Api.apiKey}&language=es-ES&page=1';

  // Reviews d'una pel·lícula concreta
  static String getMovieReviews(int movieId) =>
      'https://api.themoviedb.org/3/movie/$movieId/reviews?api_key=${Api.apiKey}&language=es-ES&page=1';

  // Detalls d'una peli
  static String getMovieDetails(int movieId) =>
      '${Api.baseUrl}movie/$movieId?api_key=${Api.apiKey}&language=es-ES';

  // Detalls d'un actor
  static String getActorDetails(int actorId) =>
      '${Api.baseUrl}person/$actorId?api_key=${Api.apiKey}&language=es-ES';

  // Cerca de pel·lícules
  static String getSearchMovies(String query) =>
      '${Api.baseUrl}search/movie?api_key=${Api.apiKey}&language=es-ES&query=$query&page=1&include_adult=false';

  // Cerca de persones/actors
  static String getSearchPeople(String query) =>
      '${Api.baseUrl}search/person?api_key=${Api.apiKey}&language=es-ES&query=$query&page=1&include_adult=false';
}
```

## 1.1.2. API_SERVICE.DART

Seguidament, editem aquest arxiu, és el responsable de demanar les dades a les rutes definides a l'arxiu "api_end_points.dart". Un cop realitzada la crida a l'API, l'arxiu espera un resposta del servidor i, quan la rep, processa les dades rebudes. Aquest procés consisteix a transformar el JSON rebut a la llista d'objectes corresponent segons el model utilitzat.

### 1.1.2.1 PEL·LÍCULES

```dart
import 'dart:convert';
import 'package:movies/api/api.dart';
import 'package:movies/models/movie.dart';
import 'package:movies/models/actor.dart';
import 'package:http/http.dart' as http;
import 'package:movies/models/review.dart';

class ApiService {

  // Top Rated: carreguem 5 pelicules pel carrusel principal --> Ens saltem les 6 primeres perque siguin diferents a les de populars
  static Future<List<Movie>?> getTopRatedMovies() async {
    List<Movie> movies = [];
    try {
      http.Response response = await http.get(Uri.parse(
          '${Api.baseUrl}movie/top_rated?api_key=${Api.apiKey}&language=es-ES&page=1'));
      var res = jsonDecode(response.body);

      //convertim cada element del array en un objecte Movie
      res['results'].skip(6).take(5).forEach(
            (m) => movies.add(Movie.fromMap(m)),
          );

      return movies;
    } catch (e) {
      return null;
    }
  }
```

```dart
// Mètode flexible per demanar qualsevol llista de pelis (popular, properament, etc.)
//Rep endpoint --> genera la petició corresponent
static Future<List<Movie>?> getCustomMovies(String url) async {
  List<Movie> movies = [];
  try {
    http.Response response =
        await http.get(Uri.parse('${Api.baseUrl}movie/$url'));
    var res = jsonDecode(response.body);
    res['results'].take(6).forEach((m) => movies.add(Movie.fromMap(m)));
    return movies;
  } catch (e) {
    return null;
  }
}
```

```dart
// Cerca de pelis --> depenent del text introduit en el buscador
static Future<List<Movie>?> getSearchedMovies(String query) async {
  List<Movie> movies = [];
  try {
    http.Response response = await http.get(Uri.parse(
        '${Api.baseUrl}search/movie?api_key=${Api.apiKey}&language=es-ES&query=$query&page=1&include_adult=false'));
    var res = jsonDecode(response.body);
    res['results'].forEach((m) => movies.add(Movie.fromMap(m)));
    return movies;
  } catch (e) {
    return null;
  }
}
```

```dart
//carreguem les ressenyes d'una pel·lícula concreta
static Future<List<Review>?> getMovieReviews(int movieId) async {
  List<Review> reviews = [];
  try {
    http.Response response = await http.get(Uri.parse(
        'https://api.themoviedb.org/3/movie/$movieId/reviews?api_key=${Api.apiKey}&language=es-ES&page=1'));
    var res = jsonDecode(response.body);
    //convertim cada element del array en un objecte Review
    res['results'].forEach((r) {
      reviews.add(
        Review(
          author: r['author'],
          comment: r['content'],
          rating: r['author_details']['rating'],
        ),
      );
    });
    return reviews;
  } catch (e) {
    return null;
  }
}
```

```dart
//Tots els detalls d'una pel·lícula concreta
static Future<Movie?> getMovieDetails(int movieId) async {
  try {
    final response = await http.get(Uri.parse(
        '${Api.baseUrl}movie/$movieId?api_key=${Api.apiKey}&language=es'));
    var res = jsonDecode(response.body);

    //Utilitzem el fromDetailsMap per agafar més camps que amb fromMap
    return Movie.fromDetailsMap(res);
  } catch (e) {
    return null;
  }
}
```

```dart
// Mètode amb el que obtenim els actors d'una pel·lícula concreta
static Future<List<dynamic>?> getMovieCredits(int movieId) async {
  try {
    //URL per obtenir el cast d'una peli --> aqui directament ja que és molt específica i no l'utilitzem en cap altre lloc
    final response = await http.get(Uri.parse(
        '${Api.baseUrl}movie/$movieId/credits?api_key=${Api.apiKey}&language=es'));
    var res = jsonDecode(response.body);
    return res['cast'] ?? []; // Retornem la llista de cast
  } catch (e) {
    return null;
  }
}
```

## 1.1.2.2. ACTORS

```dart
// Carreguem els actors populars --> comencem a la pàgina 1 per defecte
static Future<List<Actor>?> getPopularActors() async {
  List<Actor> actors = [];
  try {
    final response = await http.get(Uri.parse(
        '${Api.baseUrl}person/popular?api_key=${Api.apiKey}&language=es&page=1'));
    var res = jsonDecode(response.body);
    //Convertim cada element del array en un objecte Actor
    (res['results'] as List).forEach((p) => actors.add(Actor.fromMap(p)));    Funct
    return actors;
  } catch (e) {
    return null;
  }
}
```

```dart
// Cerca d'actors --> depenent del text introduit en el buscador
static Future<List<Actor>?> getSearchedPeople(String query) async {
  List<Actor> people = [];
  try {
    final response = await http.get(Uri.parse(
        'https://api.themoviedb.org/3/search/person?api_key=${Api.apiKey}&language=es&query=$query&page=1&include_adult=false'));
    var res = jsonDecode(response.body);
    (res['results'] as List).forEach((p) => people.add(Actor.fromMap(p)));    Function literals shouldn't be passed to 'forEach'.
    return people;
  } catch (e) {
    return null;
  }
}
```

```dart
// Paginació d'actors populars --> passem la pàgina que volem carregar
static Future<List<Actor>?> getPopularActorsByPage(int page) async {
  List<Actor> actors = [];
  try {
    final response = await http.get(Uri.parse(
        '${Api.baseUrl}person/popular?api_key=${Api.apiKey}&language=es&page=$page'));
    var res = jsonDecode(response.body);
    (res['results'] as List).forEach((p) => actors.add(Actor.fromMap(p)));    Function
    return actors;
  } catch (e) {
    return null;
  }
}
```

```dart
// Detalls d'un actor
static Future<Actor?> getActorDetails(int actorId) async {
  try {
    final response = await http.get(Uri.parse(
        '${Api.baseUrl}person/$actorId?api_key=${Api.apiKey}&language=es'));
    var res = jsonDecode(response.body);
    return Actor.fromDetailsMap(res);
  } catch (e) {
    return null;
  }
}

// Crèdits d'un actor (peli i sèries on ha participat)
static Future<List<dynamic>?> getActorCredits(int actorId) async {
  try {
    final response = await http.get(Uri.parse(
        '${Api.baseUrl}person/$actorId/combined_credits?api_key=${Api.apiKey}&language=es'));
    var res = jsonDecode(response.body);
    return res['cast'] ?? res['crew'] ?? [];
  } catch (e) {
    return null;
  }
}
```

### 1.1.3. API.DART

Aquest arxiu es essencial en el nostre codi per al funcionament de la nostra aplicació ja que centralitza tota la configuració de connexió amb l'API i actua com arxiu de configuració global. En primer lloc, definim la URL base aquesta s'utilitzarà per totes les peticions, totes les rutes definides dins el fitxer dels "end-points" es concatenaran darrere d'aquesta. En segon lloc, com l'API només ens envia el nom del fitxer i no la foto sencera, haurem de posar aquest prefix davant per veure-les. Finalment, definim la clau que ens identifica com usuari registrat i la qual ens permet accedir a les dades de l'API.

```dart
//Configuració de la connexió a la API de TMDB
class Api {
  // URL base de l'API de TMDB
  static const baseUrl = "https://api.themoviedb.org/3/";

  // URL base per obtenir les imatges (posters, backdrops, etc.)
  static const imageBaseUrl = "https://image.tmdb.org/t/p/original/";

  // Clau pública d'accés a l'API
  static const apiKey = "5aa88ce9c7886efbaa010b8070a745e8";
}
```

## 1.2. CARPETA CONTROLLERS

Aquesta carpeta conté la lògica de presentació de la nostra aplicació. La seva funció és emmagatzemar les dades que la pantalla necessita en variables observables, després fa de pont, és a dir, la vista demana les coses al controlador i aquest li demana a la API. Finalment, quan les dades canvien el controlador avisa a la pantalla perquè s'actualitzi. Dins d'aquesta carpeta hem modificat els controladors que comentem a continuació.

### 1.2.1. ACTORS CONTROLLERS

Aquest controlador, gestiona l'estat dels actors a tota l'aplicació. Carreguem les dades dels actors que volem veure i quan hi hagi canvis utilitzem l'eina de "GetX" la qual ens dona un seguit de funcionalitats per gestionar l'estat de l'aplicació i quan hi ha canvis s'actualitza la interfície.

```dart
import 'package:get/get.dart';
import 'package:movies/api/api_service.dart';
import 'package:movies/models/actor.dart';

// Controlador que gestiona l'estat dels actors a l'aplicació.
class ActorsController extends GetxController {
  // Llista d'actors populars a la pàgina actual
  var popularActors = <Actor>[].obs;
  // Llista d'actors guardats per l'usuari
  var watchListActors = <Actor>[].obs;
  // Pàgina actual --> paginació d'actors
  var currentPage = 1.obs;
  // Estat de càrrega quan s'estan obtenint més actors
  var isLoadingMore = false.obs;

  /// Mètode que s'executa quan s'inicialitza el controlador.
  /// Carrega la primera pàgina d'actors populars des de l'API.
  @override
  void onInit() async {
    var actors = await ApiService.getPopularActorsByPage(1);
    if (actors != null) popularActors.value = actors;
    super.onInit();
  }
```

A baix de tot de la pantalla, trobem unes fletxes que ens permeten passar o retrocedir la pàgina i això o controlem de la següent manera:

```dart
//Carreguem la següent pàgina d'actors populars --> incrementem currentPage i obtenim nous actors
Future<void> loadNextPage() async {
  isLoadingMore.value = true;
  currentPage.value++;
  // Obtiene los actores de la página siguiente
  var actors = await ApiService.getPopularActorsByPage(currentPage.value);
  if (actors != null) {
    popularActors.value = actors;
    popularActors.refresh(); // Fuerza actualización de observables
  }
  isLoadingMore.value = false;
}

// Carreguem la pàgina anterior d'actors populars --> decrementem currentPage i obtenim nous actors
// No permet anar a una pàgina menor que 1.
Future<void> loadPreviousPage() async {
  if (currentPage.value <= 1) return; // No permet pàgina 0 o negativa
  isLoadingMore.value = true;
  currentPage.value--;
  // Obtenim els actors de la pàgina anterior
  var actors = await ApiService.getPopularActorsByPage(currentPage.value);
  if (actors != null) {
    popularActors.value = actors;
    popularActors.refresh(); // Força actualització d'observables
  }
  isLoadingMore.value = false;
}
```

```dart
// Comprova si un actor ja està a la llista personal de l'usuari
// Retorna --> true si l'actor està a watchList/ false en cas contrari
bool isActorInWatchList(Actor actor) {
  return watchListActors.any((a) => a.id == actor.id);
}

// Afegeix o elimina un actor de la llista personal de l'usuari
// Quan l'usuari clica sobre el boto de watchList --> si l'actor ja està l'elimina. Si no està, l'afegeix.
// Mostra una notificació de confirmació a l'usuari sobre l'acció realitzada.
void addActorToWatchList(Actor actor) {
  if (watchListActors.any((a) => a.id == actor.id)) {
    // Si ya existe, lo elimina
    watchListActors.removeWhere((a) => a.id == actor.id);
    Get.snackbar('Éxito', 'Actor eliminado de la lista',
        snackPosition: SnackPosition.BOTTOM,
        animationDuration: const Duration(milliseconds: 500),
        duration: const Duration(milliseconds: 500));
  } else {
    // Si no existe, lo añade
    watchListActors.add(actor);
    Get.snackbar('Éxito', 'Actor añadido a la lista',
        snackPosition: SnackPosition.BOTTOM,
        animationDuration: const Duration(milliseconds: 500),
        duration: const Duration(milliseconds: 500));
  }
}
```

### 1.2.2. APP_MENU_CONTROLLER.DART

S'encarrega de recordar quina opció del menú hamburguesa (pel·lícules o actors) ha escollit l'usuari i així avisar a la pantalla principal per a que canviï i s'adapti a l'opció corresponent.

```dart
import 'package:get/get.dart';

//Controlador que gestiona la selecció del menú principal de l'aplicació
// Aquest controlador és observat pel drawer i la pantalla principal per actualitzar la interfície
class AppMenuController extends GetxController {
  // Menú seleccionat actualment: 'actors' o 'movies'
  var selectedMenu = 'actors'.obs; // Per defecte mostrem els actors

  //Selecciona el menú d'actors --> Actualitza la interfície per mostrar contingut relacionat amb actors
  void selectActors() {
    selectedMenu.value = 'actors';
  }

  // Selecciona el menú de pel·lícules --> Actualitza la interfície per mostrar contingut relacionat amb pel·lícules
  void selectMovies() {
    selectedMenu.value = 'movies';
  }

  // Comprova si actualment està seleccionat el menú d'actors.
  // Retorna --> true si selectedMenu és 'actors'/false en cas contrari
  bool isActorsSelected() => selectedMenu.value == 'actors';

  // Comprova si actualment està seleccionat el menú de pel·lícules.
  // Retorna --> true si selectedMenu és 'movies'/false en cas contrari
  bool isMoviesSelected() => selectedMenu.value == 'movies';
}
```

### 1.2.3. BOTTOM_NAVIGATOR_CONTROLLER.DART

Controlador que gestiona la navegació inferior de l'aplicació. S'encarrega de gestionar les tres pantalles principals: la pantalla principal que ens mostra o les pel·lícules o els actors, la pantalla de cerca i la pantalla de preferits on es mostra la llista dels actors o pel·lícules guardades.

```dart
import 'package:get/get.dart';
import 'package:movies/screens/home_screen.dart';
import 'package:movies/screens/search_screen.dart';
import 'package:movies/screens/watch_list_screen.dart';

// Controlador que gestiona la navegació inferior (bottom navigation bar) de l'aplicació.
class BottomNavigatorController extends GetxController {
  // Array amb les tres pantalles principals de l'aplicació
  var screens = <Widget>[
    HomeScreen(),
    const SearchScreen(),
    const WatchList(),
  ]; // <Widget>[]

  // Índex de la pantalla actual mostrada (0, 1 o 2)
  var index = 0.obs;

  // Canvia l'índex de la pantalla actual.
  void setIndex(indx) => index.value = indx;
}
```

### 1.2.4. MOVIES_CONTROLLER.DART

Aquest controlador, gestiona l'estat de les pel·lícules a tota l'aplicació. Funciona de la mateixa manera que el controlador d'actors.

```dart
import 'package:movies/api/api_service.dart';
import 'package:movies/models/movie.dart';

// Controlador que gestiona l'estat de les pel·lícules en tota l'aplicació.
class MoviesController extends GetxController {
  // Estat de càrrega quan s'obtenen les pel·lícules destacades
  var isLoading = false.obs;
  // Pel·lícules millor valorades per mostrar en el carrusel principal de l'app
  var mainTopRatedMovies = <Movie>[].obs;
  // Llista personal de pel·lícules guardades per l'usuari
  var watchListMovies = <Movie>[].obs;

  // Mètode que s'executa quan s'inicialitza el controlador.
  // Aquí es carreguen les pel·lícules destacades des de l'API.
  @override
  void onInit() async {
    isLoading.value = true;
    //Cridem al endpoint per obtenir les pel·lícules millor valorades
    mainTopRatedMovies.value = (await ApiService.getTopRatedMovies())!;
    isLoading.value = false;
    super.onInit();
  }

  /// Comprueba si una película ya está en la lista personal del usuario.
  /// Parámetro [movie]: película a verificar
  /// Retorna: true si la película está en watchList, false en caso contrario
  bool isInWatchList(Movie movie) {
    return watchListMovies.any((m) => m.id == movie.id);
  }
```

```dart
// Comprovem si una pel·lícula ja està a la llista de l'usuari --> Retorna: true si està a watchList/ false en cas contrari
bool isInWatchList(Movie movie) {
  return watchListMovies.any((m) => m.id == movie.id);
}

// Si la pel·lícula ja està a la llista, l'elimina. Si no està, l'afegeix.
// Mostra una notificació de confirmació a l'usuari sobre l'acció realitzada.
void addToWatchList(Movie movie) {
  if (watchListMovies.any((m) => m.id == movie.id)) {
    // Si ja existeix, l'elimina
    watchListMovies.removeWhere((m) => m.id == movie.id);
    Get.snackbar('Éxito', 'Eliminada de la lista',
        snackPosition: SnackPosition.BOTTOM,
        animationDuration: const Duration(milliseconds: 500),
        duration: const Duration(milliseconds: 500));
  } else {
    // Si no existeix, l'afegeix
    watchListMovies.add(movie);
    Get.snackbar('Éxito', 'Añadida a la lista',
        snackPosition: SnackPosition.BOTTOM,
        animationDuration: const Duration(milliseconds: 500),
        duration: const Duration(milliseconds: 500));
  }
}
```

## 1.3. CARPETA MODELS

Aquesta carpeta informa a la teva aplicació de quines dades té exactament un objecte i, a més, també li diu com traduir la resposta de l'API a l'objecte corresponent.

### 1.3.1. ACTOR.DART

Model bàsic d'un actor on definim les dades que té l'objecte d'actor, provinents de l'API.

```dart
import 'dart:convert';

// Model bàsic d'un actor
//Camps principals que te l'actor
class Actor {
  int id;
  String name;
  String profilePath;
  String biography;
  String birthday;
  String placeOfBirth;
  List<dynamic> knownFor;
  int gender;
  double popularity;

  //Constructor de l'objecte
  Actor({
    required this.id,
    required this.name,
    required this.profilePath,
    this.biography = '',
    this.birthday = '',
    this.placeOfBirth = '',
    this.knownFor = const [],
    this.gender = 0,
    this.popularity = 0.0,
```

```dart
  // Versió resumida de l'objecte
  factory Actor.fromMap(Map<String, dynamic> map) {
    return Actor(
      id: map['id'] as int,
      name: map['name'] ?? '',
      profilePath: map['profile_path'] ?? '',
      knownFor: map['known_for'] ?? [],
      gender: map['gender'] ?? 0,
      popularity: (map['popularity'] ?? 0.0).toDouble(),
    );
  }

  //Versió completa --> agafem totes les dades de l'actor
  factory Actor.fromDetailsMap(Map<String, dynamic> map) {
    return Actor(
      id: map['id'] as int,
      name: map['name'] ?? '',
      profilePath: map['profile_path'] ?? '',
      biography: map['biography'] ?? '',
      birthday: map['birthday'] ?? '',
      placeOfBirth: map['place_of_birth'] ?? '',
      knownFor: map['known_for'] ?? [],
      gender: map['gender'] ?? 0,
      popularity: (map['popularity'] ?? 0.0).toDouble(),
    );
  }

  factory Actor.fromJson(String source) => Actor.fromMap(json.decode(source));
}
```

## 1.3.2. MOVIE.DART

Model bàsic d'una pel·lícula on definim les dades que té l'objecte d'aquesta, provinents de l'API.

```dart
import 'dart:convert';

// Model bàsic d'una pel·lícula
class Movie {
  int id;
  String title;
  String posterPath;
  String backdropPath;
  String overview;
  String releaseDate;
  double voteAverage;
  List<int> genreIds;

  //Consructor de l'objecte
  Movie({
    required this.id,
    required this.title,
    required this.posterPath,
    required this.backdropPath,
    required this.overview,
    required this.releaseDate,
    required this.voteAverage,
    required this.genreIds,
  });

  //Mètode amb el que carreguem els llistats de pel·lícules
  factory Movie.fromMap(Map<String, dynamic> map) {
    return Movie(
      id: map['id'] as int,
      title: map['title'] ?? '',
      posterPath: map['poster_path'] ?? '',
      backdropPath: map['backdrop_path'] ?? '',
      overview: map['overview'] ?? '',
      releaseDate: map['release_date'] ?? '',
      voteAverage: map['vote_average']?.toDouble() ?? 0.0,
      genreIds: List<int>.from(map['genre_ids']),
    ); // Movie
  }
```

```dart
  //Mètode amb el que carreguem els detalls d'una pel·lícula concreta --> pagina de detalls
  factory Movie.fromDetailsMap(Map<String, dynamic> map) {
    List<int> genres = [];
    if (map['genres'] != null && map['genres'] is List) {
      genres = (map['genres'] as List)
          .map<int>((g) => (g['id'] ?? 0) as int)
          .toList();
    }

    //retornem l'objecte
    return Movie(
      id: map['id'] as int,
      title: map['title'] ?? '',
      posterPath: map['poster_path'] ?? '',
      backdropPath: map['backdrop_path'] ?? '',
      overview: map['overview'] ?? '',
      releaseDate: map['release_date'] ?? '',
      voteAverage: (map['vote_average'] ?? 0).toDouble(),
      genreIds: genres,
    );
  }

  factory Movie.fromJson(String source) => Movie.fromMap(json.decode(source));
}
```

### 1.3.3. REVIEW.DART

Model bàsic de les ressenyes d'una pel·lícula on definim les dades que té l'objecte, provinents de l'API.

```dart
// Mòdel de les ressenyes d'una pel·lícula
class Review {
  String author;
  String comment;
  double rating;

  //Constructor de l'objecte
  Review({
    required this.author,
    required this.comment,
    required this.rating,
  });

  //Mètode amb el que carreguem les ressenyes --> convertim en un objecte de review
  factory Review.fromJson(Map<String, dynamic> map) {
    return Review(
      author: map['name'] ?? '',
      comment: map['content'] ?? '',
      rating: map['rating']?.toDouble() ?? 0.0,
    );
  }
}
```

## 1.4. CARPETA SCREENS

Aquesta carpeta conté totes les pantalles principals de l'aplicació amb les que l'usuari interactua. Transforma totes les dades simples i sense format en una experiència visual molt més atractiva.

Fins ara, hem anat comentant tots els components de cada carpeta per entendre millor la base de tot el codi. Des d'aquest punt, donada la llarga extensió dels arxius ens centrarem únicament en les modificacions i afegits que hem implementat per a la nova funcionalitat.

### 1.4.1. DETAILS_SCREEN.DART

Pantalla de detalls tant d'una peli com d'un actor. Aquesta ens mostra els detalls de l'objecte sobre el que hem clicat. La primera modificació que trobem dins del codi és en relació a la sinopsis de la pel·lícula, abans de la modificació no era possible que si la sinopsis era molt llarga ens deixes fer "scroll" per poder llegir-la sencera i no deixar a l'usuari a mitges. En canvi, després de la modificació si ho és, per aconseguir-ho utilitzem un "SingleChildScrollView".

```
SizedBox(
  height: 400,
  child: TabBarView(children: [
    //Si el resum que es mostra a la pantalla de detalls és molt llarg ens permet veure'l sencer fent scroll
    SingleChildScrollView(
      child: Container(
        margin: const EdgeInsets.only(top: 20),
        padding:
            const EdgeInsets.symmetric(horizontal: 10),
        child: Text(
          movie.overview,
          textAlign: TextAlign.center,
          style: const TextStyle(
            fontSize: 20,
            fontWeight: FontWeight.w200,
          ), // TextStyle
        ), // Text
      ), // Container
    ), // SingleChildScrollView
```

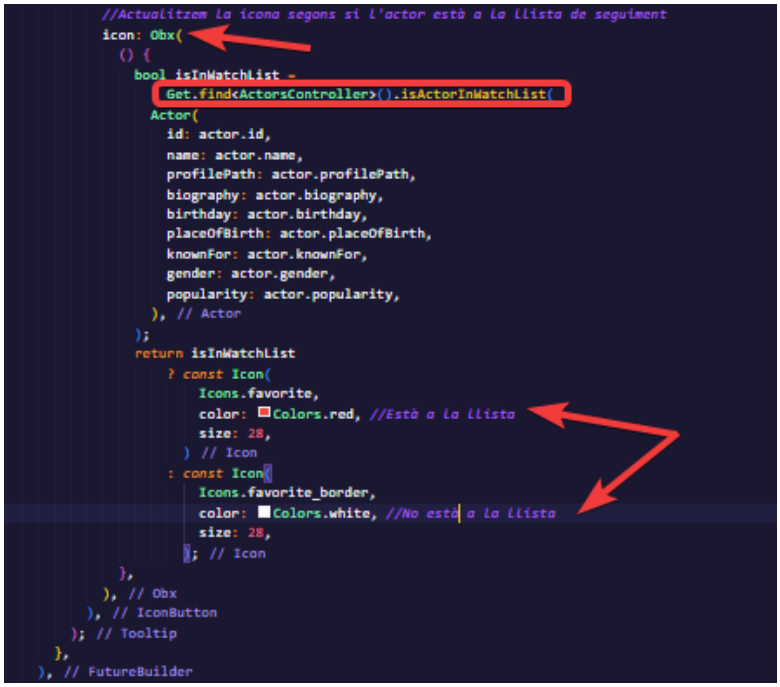
Si seguim el codi, la segona modificació que trobem és en relació amb un container buit que teníem en el codi original d'aquesta pantalla. Aquí és on mostrem tots els actors que han participat a la pel·lícula sobre la qual estem veient els detalls. En el segon li hem donat funcionalitat substituint el "Container()" buit per un "widget _CastTabContent" el qual cridarà al "endpoint" de "getMoviesCredits" i ens mostrarà als actors participants amb la seva foto i nom.

```
                } else {
                  return const Center(
                    child: Text('Wait...'),
                  ); // Center
                }
              },
            ), // FutureBuilder
            _CastTabContent(movieId: movie.id),
          ]), // TabBarView
        ), // SizedBox
```

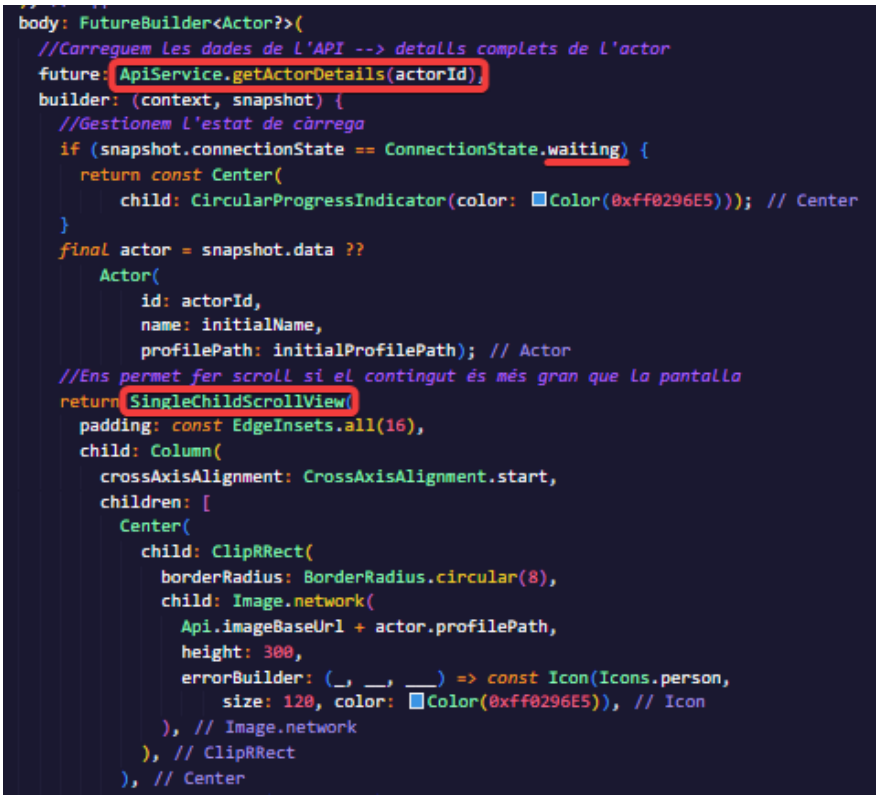
A partir d'aquí, el següent que hem afegit és tot el codi necessari per la pantalla de detalls de l'actor. Amb aquest codi, ens permet afegir un actor a la llista de favorits i ens mostra de manera dinàmica un llistat de pel·lícules on l'actor ha participat. Per controlar la llista de seguiment utilitzem un "FutureBuilder" que controlarem de la següent manera:

```
actions: [
  FutureBuilder<Actor?>(
    //Agafem els details complets de l'actor
    future: ApiService.getActorDetails(actorId),
    builder: (context, snapshot) {
      final actor = snapshot.data ??
          //Si no s'han carregat encara, utilitzem les dades inicials
          Actor(
              id: actorId,
              name: initialName,
              profilePath: initialProfilePath); // Actor
      return Tooltip(
        message: 'Guardar este actor en tu lista',
        triggerMode: TooltipTriggerMode.tap,
        child: IconButton(
          onPressed: () {
            //Afegim o eliminem l'actor a la llista de seguiment segons si ja hi és o no
            Get.find<ActorsController>().addActorToWatchList(
              Actor(
                id: actor.id,
                name: actor.name,
                profilePath: actor.profilePath,
                biography: actor.biography,
                birthday: actor.birthday,
                placeOfBirth: actor.placeOfBirth,
                knownFor: actor.knownFor,
                gender: actor.gender,
                popularity: actor.popularity,
              ), // Actor
            );
          },
```

I amb el "Obx" segons si està a la llista o no canvia el color del cor.




Seguidament, hem afegit el codi que forma el cos d'aquesta pantalla. Aquest és molt extens i únicament mostrarem les parts on hem aplicat les millores més rellevants o destacables. Primerament, assegurem la càrrega de dades amb un "FutureBuilder". També hem afegit controls per ocultar la informació que l'API no retorna. Per últim, la part més complexa és la secció 'Conocido por', on hem hagut de gestionar una segona petició a l'API dins de la mateixa pantalla per carregar i mostrar la graella de pel·lícules de l'actor.



```
const SizedBox(height: 20),
Text(
  actor.name,
  style: const TextStyle(
      fontSize: 24,
      fontWeight: FontWeight.bold,
      color: Color(0xff0296E5)), // TextStyle
), // Text
const SizedBox(height: 12),
//Mostrem la data de naixement i lloc si estan disponibles
if (actor.birthday.isNotEmpty)
  Text(
    'Nacimiento: ${actor.birthday}',
    style: const TextStyle(color: Colors.white70, fontSize: 14),
  ), // Text
if (actor.placeOfBirth.isNotEmpty)
  Padding(
    padding: const EdgeInsets.only(top: 4),
    child: Text(
      'Lugar: ${actor.placeOfBirth}',
      style:
          const TextStyle(color: Colors.white70, fontSize: 14),
    ), // Text
  ), // Padding
```

```
//Mostrem els crèdits de l'actor
FutureBuilder(
  future: ApiService.getActorCredits(actorId),
  builder: (context, creditsSnap) {
    //Gestionem l'estat de càrrega
    if (creditsSnap.connectionState ==
        ConnectionState.waiting) {
      return const Center(
        child:
            CircularProgressIndicator(color: Color(0xff0296E5)),
      ); // Center
    }
    final credits = creditsSnap.data ?? [];
    if (credits.isEmpty)
      return const Text(    Statements in an if should be enclosed in a block. Try wrapping the statement in a block.
        'No hay créditos disponibles',
        style: TextStyle(color: Colors.white70),
      ); // Text
    return GridView.builder(
      shrinkWrap: true,
      physics: const NeverScrollableScrollPhysics(), //Deshabilitem el scroll intern
      gridDelegate:
          const SliverGridDelegateWithFixedCrossAxisCount( //Definim la graella
        crossAxisCount: 2,
        crossAxisSpacing: 12,
        mainAxisSpacing: 12,
        childAspectRatio: 0.6,
      ), // SliverGridDelegateWithFixedCrossAxisCount
      itemCount: credits.length > 10 ? 10 : credits.length, //Mostrem un màxim de 10 crèdits
      itemBuilder: (context, index) {
        final credit = credits[index];
        final title =
            credit['title'] ?? credit['name'] ?? 'Sin título';
        final posterPath = credit['poster_path'] ?? '';
        final movieId = credit['id'] ?? 0;
        final mediaType = credit['media_type'] ?? 'movie';
        final year = (credit['release_date'] ??
            credit['first_air_date'] ??
            '') as String;

        //Extraiem només l'any de la data completa
        return GestureDetector(
          onTap: () async {
            if (mediaType == 'movie' && movieId != 0) {
              final movie =
                  await ApiService.getMovieDetails(movieId); //Agafem els detalls complets de la pel·lícula
              if (movie != null) {
                Get.to(
                  () => DetailsScreen(movie: movie), //Naveguem a la pantalla de detalls
                  transition: Transition.fadeIn,
                );
              } else {
                print(    Don't invoke 'print' in production code. Try using a logging framework.
                    'Error: película no se cargó (getMovieDetails retornó null)');
              }
            }
          },
```

Finalment, trobem la classe de "_CastTabContent" que hem comentat anteriorment, dins d'aquesta gestionem la paginació dels actors que ens surten i la navegació( configurem l'event onTap perquè en clicar sobre un actor en porti a la pantalla de detalls). Primerament, sempre ens surten 10 i amb la implementació d'aquesta paginació permetem a l'usuari de que esculli si vol o no vol veure més actors participants de la pel·lícula seleccionada. Per comentar aquesta part del codi ens centrarem en les parts més rellevants deixant de banda altres parts més bàsiques com els estils.

```dart
@override
Widget build(BuildContext context) {
  //Carreguem els participants de la pel·lícula
  return FutureBuilder<List<dynamic>?>(
    future: _castFuture,
    builder: (context, castSnap) {
      if (castSnap.connectionState == ConnectionState.waiting) {
        return const Center(
            child: CircularProgressIndicator(color: Color(0xff0296E5))); // Center
      }
      final cast = castSnap.data ?? [];
      if (cast.isEmpty) {
        return const Center(child: Text('No hay reparto disponible'));
      }

      // Mostrem només una part del repartiment --> opció de veure més
      final displayedCast = cast.take(_displayedCount).toList();
      final hasMore = cast.length > displayedCast.length;

      //Construïm la llista de participants
      return ListView.builder(
        itemCount: displayedCast.length + (hasMore ? 1 : 0),
        itemBuilder: (context, index) {
          if (index == displayedCast.length) {
            final remaining = cast.length - _displayedCount;
            return Padding(
              padding: const EdgeInsets.symmetric(vertical: 12),
              child: Center(
                child: ElevatedButton(
                  style: ElevatedButton.styleFrom(
                    backgroundColor: const Color(0xff0296E5),
                    padding: const EdgeInsets.symmetric(
                        horizontal: 24, vertical: 12), // EdgeInsets.symmetric
                  ),
                  onPressed: () {
                    setState(() {
                      _displayedCount =
                          (_displayedCount + 10).clamp(0, cast.length); //Augmentem el nombre de participants mostrats
                    });
                  },
                  child: Text(
                    remaining > 0
                        ? 'Ver más ($remaining restantes)'
                        : 'Ver más',
                    style: const TextStyle(
                        color: Colors.white, fontWeight: FontWeight.w500), // TextStyle
                  ), // Text
                ), // ElevatedButton
              ), // Center
            ); // Padding
          }

          final person = displayedCast[index];
          final name = person['name'] ?? 'Desconocido';
          final profile = person['profile_path'] ?? '';
          final character = person['character'] ?? 'Sin especificar';

          return Padding(
            padding: const EdgeInsets.symmetric(horizontal: 0, vertical: 8),
            child: GestureDetector(
              onTap: () {
                //Naveguem a la pantalla de details de l'actor
                Get.to(() => ActorDetailScreen(
                      actorId: person[
                          'id'], //Passem l'ID de l'actor per carregar els details
                      initialName: name,
                      initialProfilePath: profile,
                    )); // ActorDetailScreen
              },
```

### 1.4.2. HOME SCREEN

Aquesta pantalla és la pantalla d'inici o principal de la nostra aplicació la qual hem modificat per afegir el menú hamburguesa que mostra o actors o pel·lícules segons la selecció de l'usuari. Aquesta pantalla també consta d'un "AppBar" dinàmic amb un títol el qual canvia segons la selecció del menú.

```dart
//Pantalla principal que alterna entre --> pel·lícules i actors segons la selecció del menú
class HomeScreen extends StatelessWidget {
  HomeScreen({super.key});

  // Controladors per gestionar l'estat de pel·lícules, cerca i menú
  final MoviesController movieController = Get.put(MoviesController());
  final SearchController1 searchController = Get.put(SearchController1());
  final AppMenuController menuController = Get.put(AppMenuController());

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      // AppBar amb títol dinàmic basat en la selecció del menú
      appBar: AppBar(
        backgroundColor: const Color(0xFF1C1F26),
        elevation: 0,
        iconTheme: const IconThemeData(
          color: Color(0xFF0296E5),
        ), // IconThemeData
        // Títol que canvia entre "Actores" i "Pel·lícules"
        title: Obx(
          () => Text(
            menuController.isActorsSelected() ? 'Actores' : 'Peliculas',
            style: const TextStyle(
              color: Colors.white,
              fontWeight: FontWeight.bold,
              fontSize: 20,
            ), // TextStyle
          ), // Text
        ), // Obx
        centerTitle: false,
      ), // AppBar
      // Opcions --> permeten seleccionar entre pel·lícules i actors
      drawer: const AppNavigationDrawer(),
      //Escolta --> mostrar ActorsList o MoviesHomeView segons la selecció
      body: Obx(
        () {
          if (menuController.isActorsSelected()) {
            return const ActorsList(); // Vista d'actors
          } else {
            return const MoviesHomeView(); // Vista de pel·lícules
          }
        },
      ), // Obx
    ); // Scaffold
  }
}
```

### 1.4.3. MAIN.DART

Aquesta és la pantalla principal de l'aplicació i és la que controla el menú de navegació inferior, és a dir, el que controla que es mostri la pantalla corresponent quan l'usuari clica sobre alguna de les seves 3 opcions: "Inicio", "Buscar" i "Mi Lista". En aquest document no hem realitzat cap canvi respecte el codi original.

### 1.4.4. MOVIES_HOME_VIEW.DART

Aquesta pantalla ens mostra la vista principal de les pel·lícules. La pantalla es divideix en dos grans seccions dins d'un mateix "scroll". Primerament, trobem un carrusel que mostra les pel·lícules millor valorades i, d'altra banda, la segona secció esta formada d'un sistema de pestanyes que consta de 4 categories, segons la categoria ens mostra una graella amb les 6 pel·lícules corresponents.

```dart
import 'package:fade_shimmer/fade_shimmer.dart';
import 'package:flutter/material.dart';
import 'package:get/get.dart';
import 'package:movies/api/api.dart';
import 'package:movies/api/api_service.dart';
import 'package:movies/models/movie.dart';
import 'package:movies/screens/details_screen.dart';
import 'package:movies/widgets/index_number.dart';

class MoviesHomeView extends StatelessWidget {
  const MoviesHomeView({super.key});

  // Cosntruïm la URL de l'endpoint amb els paràmetres necessaris --> key i l'idioma que volem
  String _endpoint(String path) =>
      '$path?api_key=${Api.apiKey}&language=es-ES&page=1';

  // Fila de pel·lícules destacades (Top Rated) --> acompanyades d'un número d'índex
  Widget _featuredRow() {
    return FutureBuilder<List<Movie>?>(
      future: ApiService.getTopRatedMovies(), //Cridem a l'endpoint de Top Rated
      builder: (context, snapshot) {
        if (snapshot.connectionState == ConnectionState.waiting) {
          return const SizedBox(
            height: 220,
            child: Row(
              children: [
                Expanded(
                  child: FadeShimmer(
                    width: double.infinity,
                    height: 220,
                    highlightColor: Color(0xff22272f),
                    baseColor: Color(0xff20252d),
                  ),
                ),
                SizedBox(width: 16),
                Expanded(
                  child: FadeShimmer(
                    width: double.infinity,
                    height: 220,
                    highlightColor: Color(0xff22272f),
                    baseColor: Color(0xff20252d),
                  ),
                ),
              ],
            ),
          );
        }
```

```
// Un cop tenim les dades, les mostrem en un PageView
final movies = snapshot.data ?? [];
if (movies.isEmpty) return const SizedBox(height: 380); //Si no hi ha dades, retornem un espai buit
final controller =
    PageController(viewportFraction: 0.8, keepPage: true);
return SizedBox(
  height: 380,
  //Construïm com volem que es vegi la pantalla amb les pel·lícules
  child: PageView.builder(
    controller: controller,
    itemCount: movies.length,
    padEnds: false,
    physics: const BouncingScrollPhysics(), //Efecte de rebot al final de la llista
    itemBuilder: (context, index) {
      final m = movies[index];
      return Padding(
        padding: EdgeInsets.only( // Afegim marge a l'esquerra del primer element --> sense això no es veu bé
          left: index == 0 ? 16 : 0,
          right: 16,
        ),
        // Permet detectar tocs en cada element del PageView --> si en detecta  --> obre la pantalla de detalls d'aquesta pel·lícula
        child: GestureDetector(
          onTap: () => Get.to(DetailsScreen(movie: m)),
          child: Stack(
            clipBehavior: Clip.none,
            children: [
              Positioned.fill(
                child: ClipRRect(
                  borderRadius: BorderRadius.circular(16),
                  child: Image.network(
                    Api.imageBaseUrl + m.posterPath,
                    fit: BoxFit.cover,
                    errorBuilder: (_, __, ___) =>
                        const Icon(Icons.broken_image, size: 120),
                  ),
                ),
              ),
              Positioned(
                left: -12,
                bottom: -24,
                child: IndexNumber(number: index + 1),
              ),
```

```
// Llista de pel·lícules per a cada pestanya --> d'aquesta manera evitem repetir codi --> per cada pestanya només canviem l'endpoint
Widget _tabList(
  String endpoint, {
  int crossAxisCount = 3,
  double childAspectRatio = 0.62,
}) {
  // Cridem a l'endpoint corresponent i mostrem les pel·lícules en una graella --> GridView Builder
  return FutureBuilder<List<Movie>?>(
    future: ApiService.getCustomMovies(_endpoint(endpoint)), //Cridem a l'endpoint corresponent
    builder: (context, snapshot) {
      if (snapshot.connectionState == ConnectionState.waiting) {
        return const Center(child: CircularProgressIndicator());
      }
      final movies = snapshot.data ?? [];
      if (movies.isEmpty) {
        return const Center(child: Text('Sin datos'));
      }
      return GridView.builder(
        padding: EdgeInsets.zero,
        gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(
          crossAxisCount: crossAxisCount,
          mainAxisSpacing: 16,
          crossAxisSpacing: 16,
          childAspectRatio: childAspectRatio,
        ),
        itemCount: movies.length,
        itemBuilder: (_, index) {
          final m = movies[index];
          return GestureDetector(
            onTap: () => Get.to(DetailsScreen(movie: m)), //Obrim la pantalla de detalls d'aquesta pel·lícula
            child: ClipRRect(
              borderRadius: BorderRadius.circular(12),
              child: Image.network(
                Api.imageBaseUrl + m.posterPath,
                fit: BoxFit.cover,
                errorBuilder: (_, __, ___) => Container(
                  decoration: BoxDecoration(
                    color: const Color(0xff0296E5),
                    borderRadius: BorderRadius.circular(12),
                  ),
                  child: const Center(
                    child: Icon(
                      Icons.movie,
                      size: 60,
                      color: Colors.white,
                    ),
                  ),
                ),
              ),
            ),
          );
        },
      );
    },
  );
}
```

```dart
// Construïm la vista principal de pel·lícules --> amb les pel·lícules destacades i les pestanyes
@override
Widget build(BuildContext context) {
  return SingleChildScrollView(
    child: Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
        Padding(
          padding: const EdgeInsets.symmetric(horizontal: 16),
          child: _featuredRow(),
        ),
        const SizedBox(height: 24),
        Padding(
          padding: const EdgeInsets.symmetric(horizontal: 16),
      child: DefaultTabController(   // Controlador de les pestanyes
          length: 4,
          child: Column(
            crossAxisAlignment: CrossAxisAlignment.start,
            children: [
              const TabBar(
                isScrollable: false,
                indicatorColor: Color(0xff0296E5),
                indicatorWeight: 4,
                indicatorSize: TabBarIndicatorSize.label,
                labelColor: Color(0xff0296E5),
                unselectedLabelColor: Colors.white54,
                tabs: [
                  Tab(text: 'En cartelera'),
                  Tab(text: 'Próximamente'),
                  Tab(text: 'Mejor valoradas'),
                  Tab(text: 'Populares'),
                ],
              ),
              SizedBox(
                height: 1,
                child: Container(color: Color(0xFF3A3F47)),
              ),
              const SizedBox(height: 12),
              SizedBox(
                height: 460,
                child: TabBarView(
                  children: [
                    _tabList('now_playing'),
                    _tabList('upcoming'),
                    _tabList('top_rated'),
                    _tabList('popular'),
                  ],
```

## 1.4.5. POPULAR_ACTORS_VIEW.DART

Aquest arxiu és l'encarregat de mostrar tot els actors quan l'usuari escull l'opció "actors" dins del menú. Dins d'aquesta pantalla l'usuari pot agregar o retirar a un actor de la llista de preferits, entrar a la seva biografia clicant sobre la seva imatge i visualitzar més actors passant de pàgina. Tot això s'actualitza en temps real, com gairebé tota l'aplicació gràcies a la utilització de l'eina "GetX".

```dart
import 'package:flutter/material.dart';
import 'package:get/get.dart';
import 'package:movies/api/api.dart';
import 'package:movies/controllers/actors_controller.dart';
import 'package:movies/screens/details_screen.dart';

// Vista que mostra els actors més populars amb paginació
class PopularActorsView extends StatelessWidget {
  const PopularActorsView({super.key});

  @override
  Widget build(BuildContext context) {
    final ActorsController controller = Get.put(ActorsController());

    return SingleChildScrollView(
      child: Padding(
        padding: const EdgeInsets.symmetric(
          horizontal: 24,
          vertical: 20,
        ), // EdgeInsets.symmetric
        child: Column(
          mainAxisAlignment: MainAxisAlignment.start,
          children: [
            Obx(() { // Obx per actualitzar la UI quan canvia l'estat del controlador --> paginació
              return GridView.builder(
                shrinkWrap: true,
                physics: const NeverScrollableScrollPhysics(), // Evita que el GridView tingui el seu propi desplaçament
                gridDelegate: const SliverGridDelegateWithFixedCrossAxisCount( // Defineix la disposició de la graella
                  crossAxisCount: 2,
                  crossAxisSpacing: 16,
                  mainAxisSpacing: 16,
                  childAspectRatio: 0.6,
                ), // SliverGridDelegateWithFixedCrossAxisCount
```

```dart
itemCount: controller.popularActors.length,
itemBuilder: (_, index) {
  final actor = controller.popularActors[index];
  return GestureDetector( // Permet detectar tocs en cada element de la graella --> si en detecta  --> obre la pantalla de detalls d'aquest actor
    onTap: () => Get.to(ActorDetailScreen(
        actorId: actor.id,
        initialName: actor.name,
        initialProfilePath: actor.profilePath)), // ActorDetailScreen
    child: Column(
      children: [
        Stack(
          children: [
            ClipRRect(
              borderRadius: BorderRadius.circular(12),
              child: Image.network(
                Api.imageBaseUrl + actor.profilePath,
                fit: BoxFit.cover,
                height: 220,
                width: double.infinity,
                errorBuilder: (_, __, ___) => Container(
                  height: 220,
                  decoration: BoxDecoration(
                    color: const Color(0xff0296E5),
                    borderRadius: BorderRadius.circular(12),
                  ), // BoxDecoration
                  child: const Center(
                    child: Icon(
                      Icons.person,
                      size: 80,
                      color: Colors.white,
                    ), // Icon
                  ), // Center
                ), // Container
              ), // Image.network
            ), // ClipRRect
```

Per tal d'afegir un actor a la llista de preferits ho controlem de la mateixa manera que des de la pantalla de detalls. D'aquesta manera, al mostrar-se la vista dels actors quan l'usuari clica sobre l'opció del menú "actors" si hi ha algun de la primera pàgina que el tenim guardat dins la llista, ens sortirà amb el cor en vermell. Donant, també l'opció a l'usuari d'afegir o retirar de la llista per aquí també.

```
    Positioned( // Icona de cor per afegir a la llista de favorits
      top: 8,
      right: 8,
      child: Obx( // Obx per actualitzar l'estat de l'icona quan es canvia la llista de favorits
        () {
          bool isInWatchList =
              controller.isActorInWatchList(actor); // Comprova si l'actor està a la llista de favorits
          return GestureDetector(
            onTap: () =>
                controller.addActorToWatchList(actor), // Afegeix o elimina l'actor de la llista de favorits
            child: Container(
              decoration: BoxDecoration(
                color: Colors.black54,
                shape: BoxShape.circle,
              ), // BoxDecoration
              padding: const EdgeInsets.all(8),
              child: Icon( // Icona de cor plena o buida segons si està a la llista de favorits
                isInWatchList
                    ? Icons.favorite
                    : Icons.favorite_border,
                color: isInWatchList
                    ? Colors.red
                    : Colors.white,
                size: 20,
              ), // Icon
            ), // Container
          ); // GestureDetector
        },
      ), // Obx
    ), // Positioned
  ],
), // Stack
const SizedBox(height: 8),
Flexible( // Permet que el text s'ajusti dins del seu contenidor
  child: Text(
    actor.name,
    style: const TextStyle(
      fontSize: 14,
      fontWeight: FontWeight.w600,
    ), // TextStyle
    maxLines: 2,
    overflow: TextOverflow.ellipsis, // Quan el nom és mmolt llarg --> mostra punts suspensius si el text és massa llarg --> sense això surt un alert de overflow
    textAlign: TextAlign.center,
  ), // Text
```

```
const SizedBox(height: 24),
Obx(() { // Obx per actualitzar la UI quan canvia la pàgina actual o l'estat de càrrega --> al canviar de pàgina
  return Row(
    mainAxisAlignment: MainAxisAlignment.center,
    children: [
      IconButton( // Botó per carregar la pàgina anterior
        onPressed: controller.currentPage.value <= 1
            ? null
            : () => controller.loadPreviousPage(),
        icon: const Icon(Icons.arrow_back),
        color: controller.currentPage.value <= 1
            ? Colors.grey
            : const Color(0xff0296E5),
        iconSize: 24,
      ), // IconButton
      const SizedBox(width: 16),
      Text(
        'Página ${controller.currentPage.value}',
        style: const TextStyle(
          fontSize: 14,
          fontWeight: FontWeight.w600,
          color: Color(0xff0296E5),
        ), // TextStyle
      ), // Text
      const SizedBox(width: 16),
      if (controller.isLoadingMore.value) // Mostra un indicador de càrrega mentre es carreguen més actors
        const SizedBox(
          height: 24,
          width: 24,
          child: CircularProgressIndicator(
            strokeWidth: 2,
            valueColor: AlwaysStoppedAnimation(Color(0xff0296E5)),
          ), // CircularProgressIndicator
        ) // SizedBox
      else
        IconButton( // Botó per carregar la següent pàgina
          onPressed: () => controller.loadNextPage(),
          icon: const Icon(Icons.arrow_forward),
          color: const Color(0xff0296E5),
          iconSize: 24,
        ), // IconButton
```

## 1.4.6. WATCH_LIST_SCREEN.DART

La funció d'aquest arxiu és implementar la pantalla de "Mi lista", la qual, ens mostra en temps real les pel·lícules i actors que l'usuari ha guardat dins la llista amb una separació que ens permet diferenciar la secció de les pel·lícules guardades de la secció dels actors guardats. Si les dues seccions estan buides, ens mostra un missatge indicant que no hi ha contingut guardat a la llista. D'aquest arxiu comentarem les parts més rellevants afegides, en el cas de les dues seccions, segueixen la mateixa estructura excepte la part on afegim que les pel·lícules de la llista ens puguin redirigir a la pantalla de detalls d'aquestes al fer clic, des d'aquesta mateixa pantalla.

```dart
// Secció de pel·lícules
if (Get.find<MoviesController>().watchListMovies.isNotEmpty) ...[
  const Align(
    alignment: Alignment.centerLeft,
    child: Text(
      'Peliculas',
      style: TextStyle(
        fontWeight: FontWeight.w600,
        fontSize: 18,
      ), // TextStyle
    ), // Text
  ), // Align
  const SizedBox(height: 15),
  ...Get.find<MoviesController>().watchListMovies.map(
        (movie) => Column(
          children: [
            GestureDetector(
              onTap: () => Get.to(DetailsScreen(movie: movie)), // Permet obrir la pantalla de detalls de la pel·lícula --> ho hem afegit només en aquesta secció
              child: Row(
                mainAxisSize: MainAxisSize.min,
                children: [
                  ClipRRect(
                    borderRadius: BorderRadius.circular(16),
                    child: Image.network(
                      Api.imageBaseUrl + movie.posterPath,
                      height: 180,
                      width: 100,
                      fit: BoxFit.cover,
                      errorBuilder: (_, __, ___) => Container(
                        height: 180,
                        width: 100,
                        decoration: BoxDecoration(
                          color: const Color(0xff0296E5),
                          borderRadius: BorderRadius.circular(16),
                        ), // BoxDecoration
                        child: const Center(
                          child: Icon(
                            Icons.movie,
                            size: 60,
                            color: Colors.white,
                          ), // Icon
                        ), // Center
                      ), // Container
                      loadingBuilder: (_, __, ___) {
                        // ignore: no_wildcard_variable_uses
                        if (___ == null) return __;
                        return const FadeShimmer(
                          width: 150,
                          height: 150,
                          highlightColor: Color(0xff22272f),
                          baseColor: Color(0xff20252d),
                        ); // FadeShimmer
                      },
                    ), // Image.network
                  ), // ClipRRect
                  const SizedBox(
                    width: 5,
                  ), // SizedBox
                  Infos(movie: movie)
                ],
              ), // Row
            ), // GestureDetector
            const SizedBox(
              height: 20,
            ), // SizedBox
          ],
        ), // Column
      ),
  const SizedBox(height: 10),
```

```dart
// Missatge quan no hi ha res a la llista
if (Get.find<MoviesController>().watchListMovies.isEmpty &&
    Get.find<ActorsController>().watchListActors.isEmpty)
  const Column(
    children: [
      SizedBox(
        height: 200,
      ), // SizedBox
      Text(
        'No hay películas ni actores en tu lista',
        style: TextStyle(
          fontSize: 20,
          fontWeight: FontWeight.w200,
        ), // TextStyle
      ), // Text
    ],
  ), // Column
```

## 1.5. CARPETA UTILS

Dins d'aquesta carpeta només gestionem un arxiu el qual ens permet traduir codis a text. És a dir, quan l'API retorna un gènere retorna només l'ID, aquest arxiu s'encarrega de iterar la llista de identificadors que porta la pel·lícula i acumula els noms trobats en una llista. Si no hi ha coincidències, retorna un "N/A".

### 1.5.1. UTILS.DART

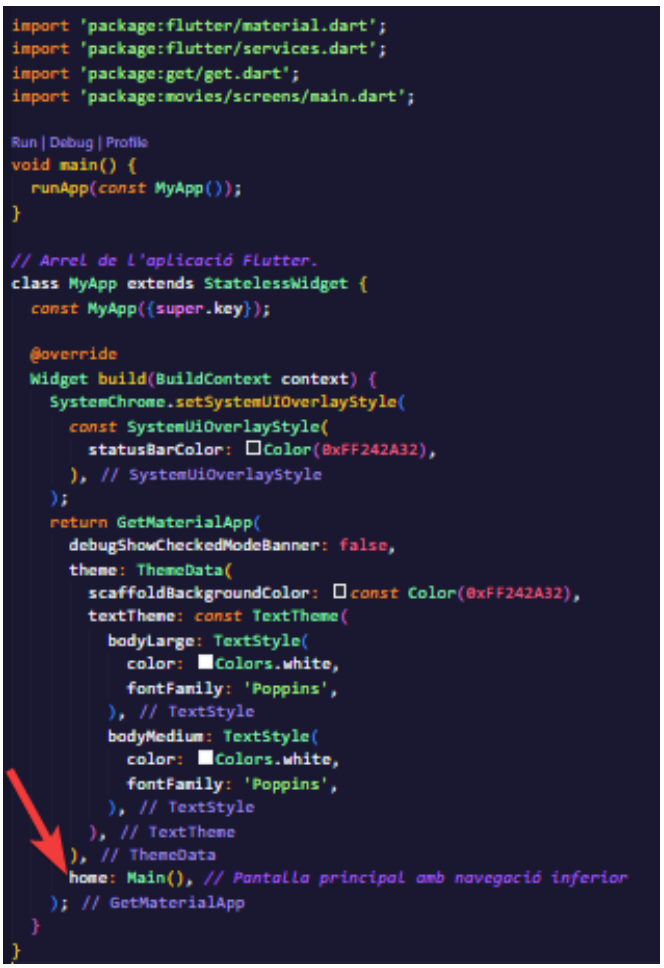
```dart
import 'package:movies/models/movie.dart';

// Utilitat per obtenir els gèneres de les pel·lícules a partir dels seus IDs
class Utils{
  static String getGenres(Movie movie) {
    // Llista per emmagatzemar els noms dels gèneres
    List<String> genres = [];

    movie.genreIds.forEach((id) {
      [
        {28: 'Action'},
        {12: 'Adventure'},
        {16: 'Animation'},
        {35: 'Comedy'},
        {80: 'Crime'},
        {99: 'Documentary'},
        {18: 'Drama'},
        {10751: 'Family'},
        {14: 'Fantasy'},
        {36: 'History'},
        {27: 'Horror'},
        {10402: 'Music'},
        {9648: 'Mystery'},
        {10749: 'Romance'},
        {878: 'Science Fiction'},
        {10770: 'TV Movie'},
        {53: 'Thriller'},
        {10752: 'War'},
        {37: 'Western'}
      ].forEach((m) {
        m.keys.first == id ? genres.add(m.values.first) : null;
      });
    });
    return genres.isEmpty ? 'N/A' : genres.take(2).join(', '); // Retornem els dos primers gèneres o 'N/A' si no n'hi ha
  }
}
```

## 1.6. MAIN.DART

És l'arxiu principal de la nostra aplicació el qual ens indica que es mostrarà en carregar l'aplicació i gràcies a aquest el codi es pot executar. A més, en aquest arxiu, es configura el tema global de l'aplicació (colors i la tipografia) i s'implementa la utilització del "GetX" .



```dart
import 'package:flutter/material.dart';
import 'package:flutter/services.dart';
import 'package:get/get.dart';
import 'package:movies/screens/main.dart';

Run | Debug | Profile
void main() {
  runApp(const MyApp());
}

// Arrel de l'aplicació Flutter.
class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    SystemChrome.setSystemUIOverlayStyle(
      const SystemUiOverlayStyle(
        statusBarColor: Color(0xFF242A32),
      ), // SystemUiOverlayStyle
    );
    return GetMaterialApp(
      debugShowCheckedModeBanner: false,
      theme: ThemeData(
        scaffoldBackgroundColor: const Color(0xFF242A32),
        textTheme: const TextTheme(
          bodyLarge: TextStyle(
            color: Colors.white,
            fontFamily: 'Poppins',
          ), // TextStyle
          bodyMedium: TextStyle(
            color: Colors.white,
            fontFamily: 'Poppins',
          ), // TextStyle
        ), // TextTheme
      ), // ThemeData
      home: Main(), // Pantalla principal amb navegació inferior
    ); // GetMaterialApp
  }
}
```

## 2. ENLLAÇ VIDEO DEMOSTRACIÓ

https://drive.google.com/file/d/1Qs8N1cD4F_ExEX1qZSLcGISvsUGh7rBd/view?usp=sharing

## 3. ENLLAÇ REPOSITORI DE GIT-HUB

https://github.com/Ainhoass13/Flutter.git