Movie Browser

Movie Browser — это Android-приложение, написанное на языке Kotlin с использованием Jetpack Compose, которое позволяет пользователю:

* Искать фильмы в онлайн-базе OMDb (The Open Movie Database).
* Просматривать краткую информацию о результатах поиска (мини-постер, название, год выпуска).
* Открывать диалог с расширенными деталями выбранного фильма (постер в большем разрешении, год, рейтинг, сюжет, режиссёр, актёры и т. д.).
* Помечать понравившиеся фильмы как «Избранные» и просматривать весь список «Избранного» независимо от текущего результата поиска.
* Локально сохранять идентификаторы «Избранных» фильмов (SharedPreferences), чтобы они сохранялись между запусками приложения.

Приложение полностью работает асинхронно: все сетевые запросы и операции с локальным хранилищем выполняются в фоновом потоке с помощью Kotlin Coroutines, не блокируя главный (UI) поток.

## Задачи и проблемы, которые решает проект

1. Уменьшение рутинной работы  
   Пользователю не нужно вручную открывать браузер, искать сайт OMDb (или IMDb) и вводить там запрос. Вместо этого достаточно одного поля поиска в приложении.
2. Быстрый доступ к информации о фильмах  
   Приложение запрашивает данные онлайн и отображает их в структурированном виде (постер, название, год, рейтинг, сюжет, режиссёр, актёры и пр.). Это позволяет за секунды получить всю необходимую информацию.
3. Удобное управление «Избранным»  
   Любой найденный фильм можно пометить звёздочкой. Список «Избранного» хранится локально (в SharedPreferences) и обновляется мгновенно при добавлении/удалении фильмов. При повторном запуске приложения «Избранное» остаётся неизменным.
4. Единая точка доступа к данным  
   Вместо того чтобы использовать разные сайты или приложения для поиска фильмов, Movie Browser объединяет запросы к OMDb API и выводит результаты в удобном интерфейсе.

# Архитектура проекта

## Компоненты

* **data/models —** Kotlin data-классы для представления JSON-ответов OMDb: MovieSearchResponse и MovieDto.
* **network —** Интерфейс OmdbApiService для выполнения HTTP-запросов к OMDb API через Retrofit.
* **repository —** MovieRepository — получение и маппинг данных из сети; FavoritesRepository — хранение IMDb-ID избранных в DataStore.
* **model —** MovieModel — упрощенная модель фильма для UI, копирует поля title, year, imdbID, posterUrl.
* **ui —** Components.kt — Composable-функции: MovieApp, MovieItem, MovieDetailsDialog; theme — настройка цветов и шрифтов.
* **viewmodel —** MovieViewModel — бизнес-логика: поиск, управление состоянием UI, подписка на Flow избранного; MainActivity — точка входа и установка темы.
* **datastore —** DataStore Preferences через preferencesDataStore для реактивного хранения списка избранных фильмов.
* **utils —** Вспомогательный код (если имеется) для расширений и утилит, например, преобразования DTO в модели.

**Итог**: каждый класс и файл отвечает за строго ограниченную часть логики:

* network — только API-интерфейс.
* repository — только получение и подготовка данных.
* model — только модели данных.
* ui — только отображение интерфейса.
* viewmodel — только связывание UI и репозиториев.
* FavoritesRepository — только работа с локальным хранилищем избранного.

# Основные зависимости

|  |  |  |
| --- | --- | --- |
| Категория | Название библиотеки | Для чего используется |
| Асинхронность | Kotlin Coroutines | Выполнение сетевых запросов и операций с локальным хранилищем в фоновом потоке, не блокируя UI. |
| Сеть | Retrofit 2 | Формирование и выполнение HTTP-запросов к OMDb API |
| JSON-парсинг | Gson | Десериализация JSON-ответов OMDb в DTO (MovieSearchResponse, MovieDetailResponse) и модели приложения. |
| UI | Jetpack Compose Coil | Jetpack Compose, Material 3,  иконки (Material Icons), а также Coil (асинхронная загрузка и кэширование постеров). |
| Хранение | Preferences | Локальное сохранение списка избранных фильмов |

# Пошаговая функциональность проекта

**1. Поиск фильма**

* Пользователь вводит запрос в поле и нажимает «Search».
* Выполняется асинхронный HTTP-запрос к OMDb API.

**2. Отображение результатов**

* Результаты поиска показываются списком карточек с постером, названием и годом.
* Если произошла ошибка, отображается сообщение об ошибке.

**3. Просмотр деталей**

* При клике на карточку открывается диалог с большим постером, годом и IMDb ID.
* Пользователь может закрыть диалог кнопкой «Close».

**4. Управление избранным**

* Рядом с каждой карточкой есть звезда для добавления/удаления из избранного.
* Список избранного хранится в DataStore и обновляется в реальном времени.

**5. Переключение вкладок**

* Можно переключаться между вкладками «Search» и «Favorites».
* Во вкладке «Favorites» видны только добавленные фильмы.

## Тестирование

В проекте предусмотрены четыре основных юнит-теста.

MovieDtoTest  
Проверяет, что метод toModel() у MovieDto корректно копирует все поля (название, год, IMDb ID, URL постера) в модель MovieModel.

MovieDetailResponseTest  
Убеждается, что метод toModel() у MovieDetailResponse правильно переносит все доступные поля (название, год, рейтинг, длительность, жанр, сюжет, режиссёр, актёры, страна, награды, URL постера, IMDb-рейтинг, число голосов, IMDb ID, тип, кассовые сборы) в модель MovieDetailModel.

MovieSearchResponseParsingTest  
Проверяет, что «сырый» JSON-ответ OMDb для поиска десериализуется в MovieSearchResponse, что вложенные объекты MovieDto содержат корректные поля (Title, Year, imdbID, Poster), и что MovieDto.toModel() правильно создаёт MovieModel.

FavoritesRepositoryTest  
Удостоверяется, что FavoritesRepository (SharedPreferences + StateFlow) при старте возвращает пустой набор IMDb-ID, что метод addFavorite(id) записывает новый ID в SharedPreferences и сразу обновляет поток, а метод removeFavorite(id) удаляет ID и корректно отражает это изменение во Flow.

Как работает API и парсинг  
Приложение отправляет HTTP-GET запросы к внешнему OMDb-сервису через библиотеку Retrofit. Для поиска фильмов используется запрос вида

<https://www.omdbapi.com/?s>={query}&apikey={ключ}

, где {query} — введённая пользователем строка,

а {ключ} — API-ключ OMDb.

В ответ сервис возвращает JSON с массивом объектов (каждый содержит название, год, IMDb ID, URL постера и т. п.). Для получения подробной информации о конкретном фильме отправляется запрос https://www.omdbapi.com/?i={imdbID}&apikey={ключ}, где {imdbID} — уникальный идентификатор фильма. Retrofit, подключённый вместе с GsonConverterFactory, автоматически десериализует этот JSON в DTO-классы (MovieSearchResponse и MovieDetailResponse), после чего каждый DTO конвертируется в внутренние модели (MovieModel, MovieDetailModel) для отображения в UI.

# Возможные направления развития

• Экран деталей с навигацией и дополнительной информацией.

• Интеграция с YouTube API для трейлеров.

• Расширенный поиск: фильтры.

• Пользовательские рейтинги и отзывы (Firebase).