

---

A

## **Project Report**

On

### **Movie Db App in Flutter**

Submitted in Partial Fulfilment of the Requirement

**for the Award of Degree of Bachelor of Technology**

in Department of Computer Science & Engineering

**Submitted To:**

Dr. Deepika Shekhawat

**Submitted By:**

Shivam Singh (18BCON078)

Ras Bihari Agarwal (18BCON084)



**Department of Computer Science & Engineering**

**JECRC UNIVERSITY,**

**JAIPUR 2020**

---

## **CERTIFICATE**

This is to certify that the Project entitled “Movie Db App in Flutter” presented by “Shivam Singh” bearing Registration No. “18BCON078” and “Ras Bihari Agarwal” bearing Registration No. “18BCON084” of JECRC University has been completed successfully.

This is in partial fulfilment of the requirements of Bachelor Degree in Computer science & Engineering as prescribed by the JECRC University during academic year 2021-22

---

## ACKNOWLEDGEMENT

I am greatly indebted to my guide Dr. Deepika Shekhawat for her invaluable guidance during the course of the seminar. She always gave useful suggestions and also helped me when the work was not moving ahead at times.

SHIVAM SINGH (18BCON078)  
RAS BIHARI AGARWAL (18BCON084)  
JECRC UNIVERSITY



---

## **CONTENTS**

## **PAGE NO.**

<b>1. Abstract</b>	<b>1</b>
<b>2. Introduction</b>	
<b>2.1. Project Profile</b>	<b>3</b>
<b>2.2. Project Requirements</b>	<b>4</b>
<b>2.3. What is Flutter</b>	<b>5</b>
<b>3. Application Architecture</b>	<b>4</b>
<b>3.1. The Importance of Software Architecture</b>	
<b>3.2. Clean Architecture</b>	
<b>3.3. What are those Layers?</b>	
<b>4. Application Features</b>	
<b>5. Code Screenshots</b>	
<b>6. References</b>	<b>17</b>

---

## **ABSTRACT**

The aim of this project was to develop a hybrid Movie Db app that runs on both Android and iOS by using the principles of Clean Architecture, Dependency Injection, Error Handling, State Management and App localization to make user experience better. The specific function of the application is to allow a user to see data about any movie and store it as favorites. It uses TMDB api to fetch data about currently playing, coming soon and popular movies right from its cast to its trailers. The framework and the SDK used in this project is Flutter (Google's Framework to create hybrid apps), language used is Dart Programming language and TMDB api. Flutter is used to take care of the Frontend and TMDB api is acting as a data source for the application.

---

## INTRODUCTION

### 2.1 Project Profile

**Title** : Movie Db App

**Definition** : A Movie DB app made in Flutter that uses TMDB api to fetch data about currently playing, coming soon and popular movies. Works on both Android and iOS.

**Description** :

This application displays the information about current playing, coming soon and popular movies by using the TMDB api. It shows the desired movie's summary, rating, cast and its trailer. Users also have an option to search for any movie other than the above three categories. The application also stores the favorite marked movies of the users in the database. It also provides language localization support with English and Spanish currently available as of now. It has a responsive and clean UI layout. It uses Clean Architecture and SOLID principles to maintain a scalable and clean code under the hood.

## 2.2 Project Requirements

### → Project Hardware Requirements

- ◆ Android/iOS device
- ◆ Computer

### → Project Software Requirements

- ◆ Android SDK
- ◆ Java Runtime Environment
- ◆ IDE (VSC or Android Studio)
- ◆ Flutter
- ◆ Dart

### → Project Network Requirements

- ◆ Stable Internet Connection



---

## 2.3 What is Flutter

Flutter is a free and open-source mobile UI framework created by Google and released in May 2017. In a few words, it allows you to create a native mobile application with only one codebase. This means that you can use one programming language and one codebase to create two different apps (for iOS and Android).

Flutter consists of two important parts:

- An SDK (Software Development Kit): A collection of tools that are going to help you develop your applications. This includes tools to compile your code into native machine code (code for iOS and Android).
- A Framework (UI Library based on widgets): A collection of reusable UI elements (buttons, text inputs, sliders, and so on) that you can personalize for your own needs. To develop with Flutter, you will use a programming language called Dart. The language was created by Google in October 2011, but it has improved a lot over these past years. Dart focuses on front-end development, and you can use it to create mobile and web applications. If you know a bit of programming, Dart is a typed object programming language. You can compare Dart's syntax to JavaScript.

---

## Application Architecture

### 2.1 The Importance of Software Architecture

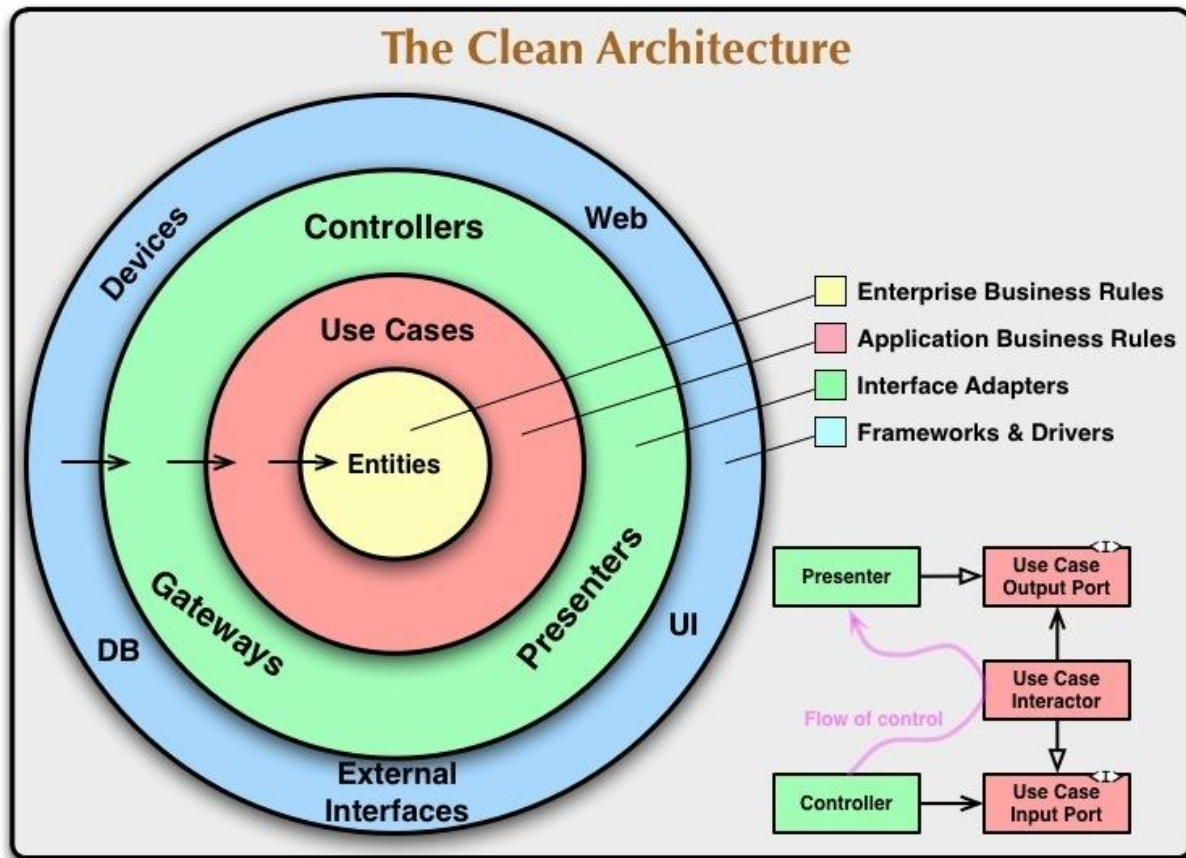
Many companies do technical tests to candidates who are submitted to a selection process. The tests may vary, but there is something that never changes, and that is that they all ask for good practices, including applying good software architecture.

*A good software architecture allows the system to be easy to understand, to develop, to maintain and to implement [Clean Architecture, Chapter 15].*

Developing apps in Flutter gives you free hand to choose the architecture and libraries. With so many options in your hand, you'll often donate a lot of your time in selecting a specific architecture. The most popular and trustworthy architecture is Clean architecture, where you've separate layers for Presentation, Domain, and Data.

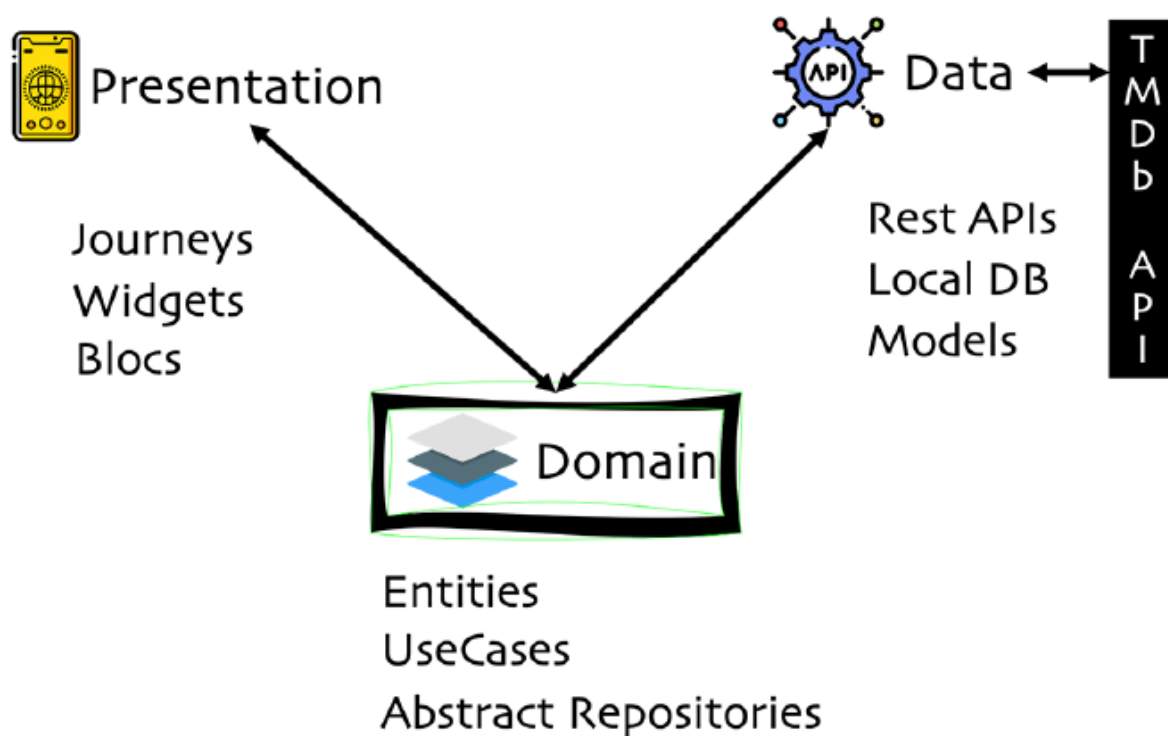
### 2.2 Clean Architecture

As proposed by our friendly Uncle Bob, we should all strive to separate code into independent layers and depend on abstractions instead of concrete implementations.



For example the most common visualization of this concept is given on the layered "onion" image above, the horizontal arrows ---> represent dependency flow. For example, Entities do not depend on anything, Use Cases depend only on Entities etc.

Also, while the essence of clean architecture remains the same for every framework, the devil lies in the details. The specific Architecture that we've used in the project is given below:



## 2.2 What are those layers?

- **The Presentation Layer**

One of the primary reasons people have switched to Flutter is the ease with which you can create cool and flexible UIs. While it is easy to write Flutter widgets, it is even easier to put business logic and taking critical decisions in your widgets.

The presentation layer mainly consists of Widgets. Many widgets combine to create a screen. A screen in Clean architecture is considered as Journey. For example, when a user moves to Movie Detail Screen, users are supposed to be in a Movie Detail Journey.

**Widgets** folder in the presentation layer will consist of small UI building blocks which will be used throughout your application across the different screens like Button, Logo, App Bar, and so on. You'll see more as we go in this series.

**Blocs** will be the heart of your UI, where you'll make decisions about what and when to show in the UI. For example, till the time movie details are fetched from API, UI will show a loader and once details are fetched from the API, you'll see the movie details on the screen.

All the themes, be it Text styles, colors, button themes, dialog themes will go in the Themes folder. So, there will be one place to maintain all theme-related information.

- **The Data Layer**

Data Layer is exposed to the outside world, whose sole responsibility is to bring data from Rest APIs, Local Database or Firebase, basically any service that gives data to your application.

Based on the application features, different APIs, and local databases that it has to fetch data from, you can have as many **DataSources** as required. Each of them will only interact with repositories.

**Repositories** will decide whether to fetch data from the remote data source or local data source, behaving as a single source of truth for the UI. UI should not know from where the data is fetched. In Data Layer, repositories will be implementations of repository abstract classes from the domain layer. More on that, in the Domain Layer section.

**Models** and **Tables** are again extensions of the entities present in the Domain Layer. Models are mapped directly with the API response and Tables are directly mapped with Database response.

There will be a Core folder as well, to segregate common code of fetching and parsing remote data.

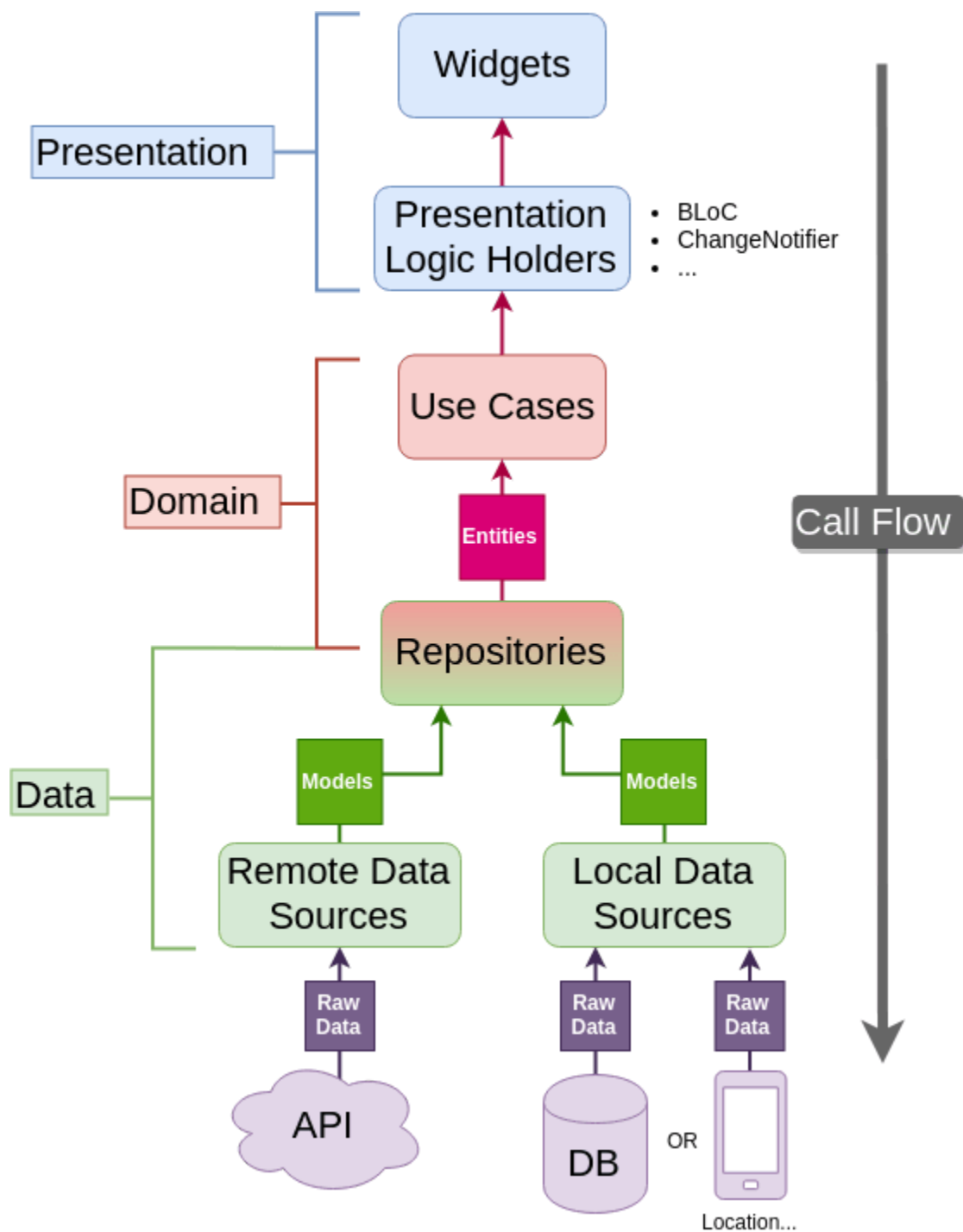
- **The Domain Layer**

Now, you've one layer for UI interaction and the other for API interaction. Domain Layer acts as a communication channel between Data Layer and Presentation Layer.

**Entities** represent data that will be required by the UI. These entities will be extended by Models and Tables in Data Layer, to maintain a level of abstraction.

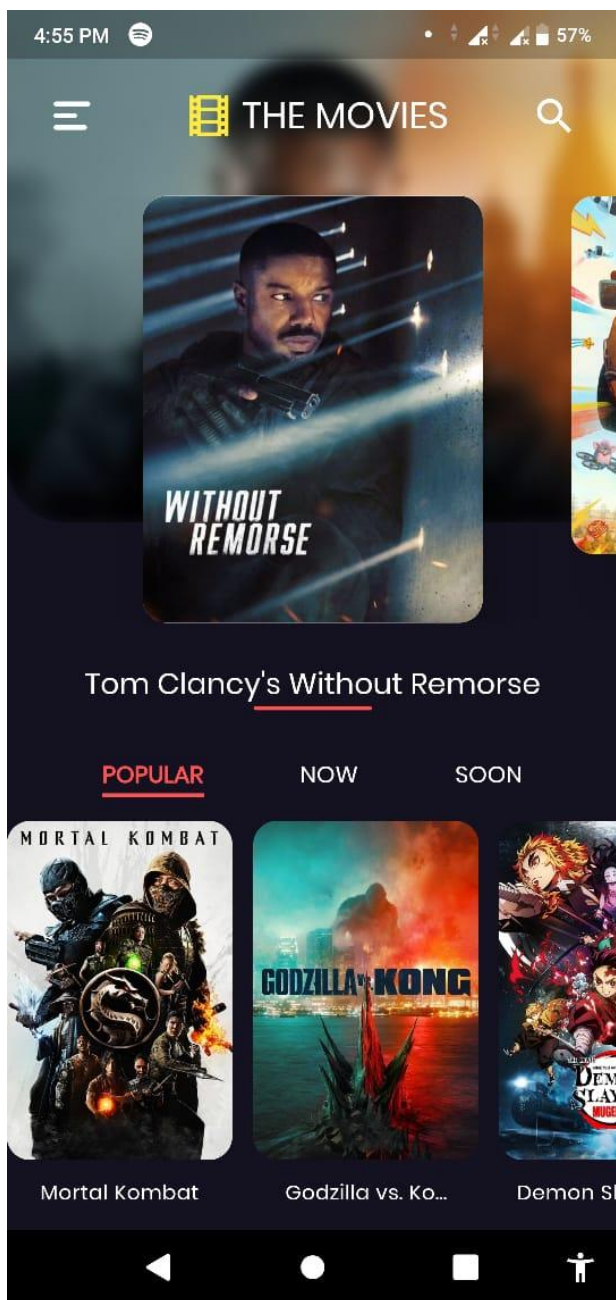
**Repositories** in the domain layer are abstract classes that only tell what data has to be fetched. But, the decision of how and from where data has to be fetched, is made by the repository implementations in the Data layer.

**UseCases** consist of the features that the app will work on. Like, fetching popular movies, trending movies, movie details, etc. UseCases are simple classes that directly pass the input parameters required to fetch details to the repository. UseCase will directly interact with the blocs.

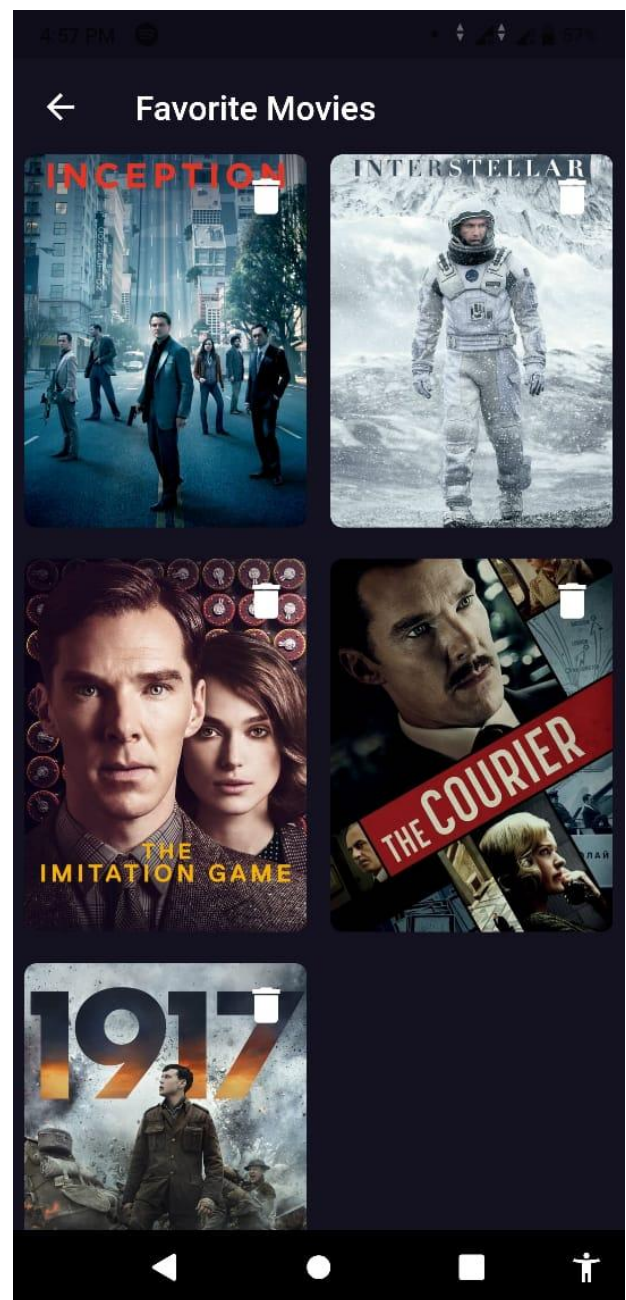


## APPLICATION FEATURES

- Home Screen

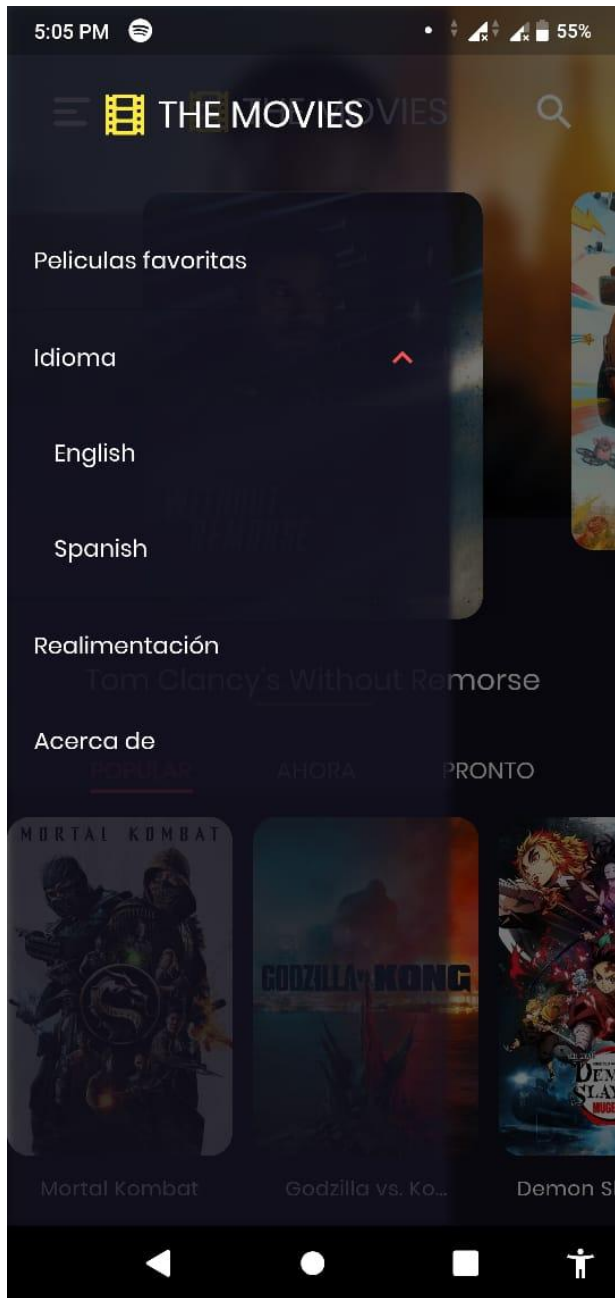


- Favorite Movies

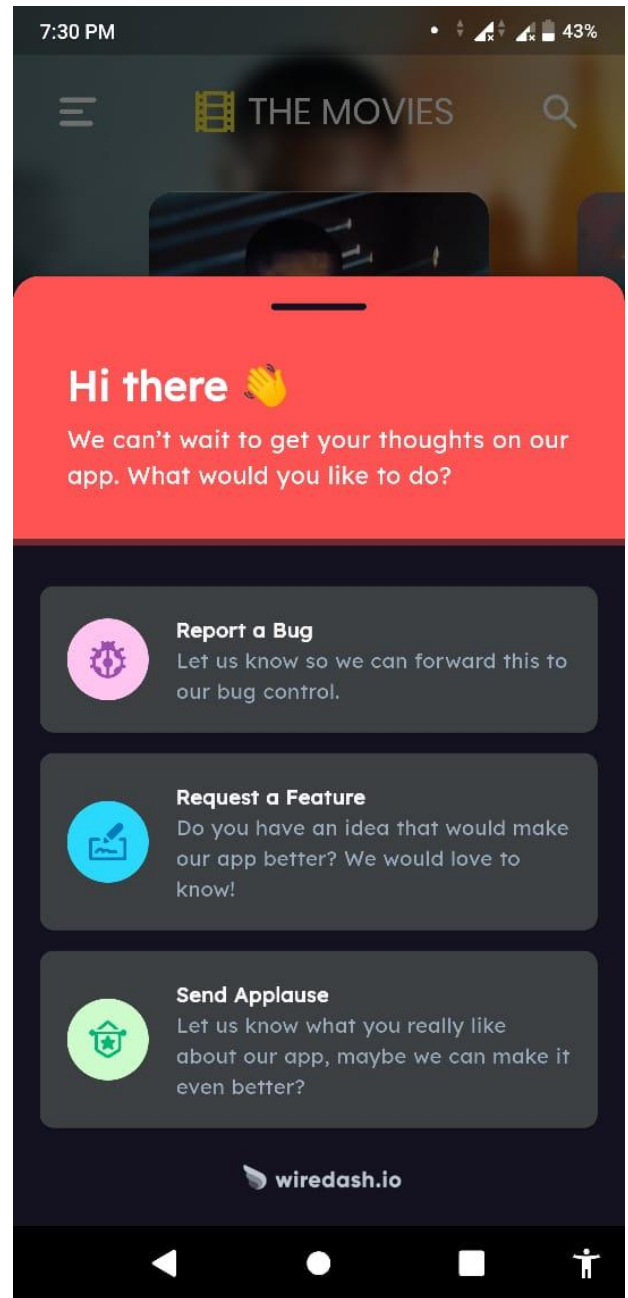




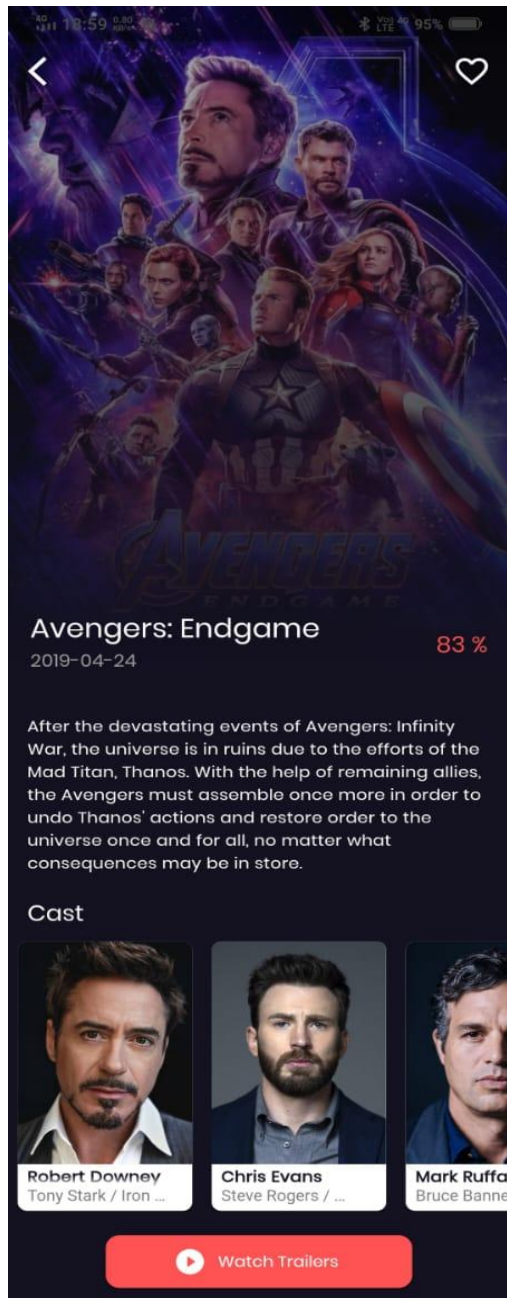
- Language Localization



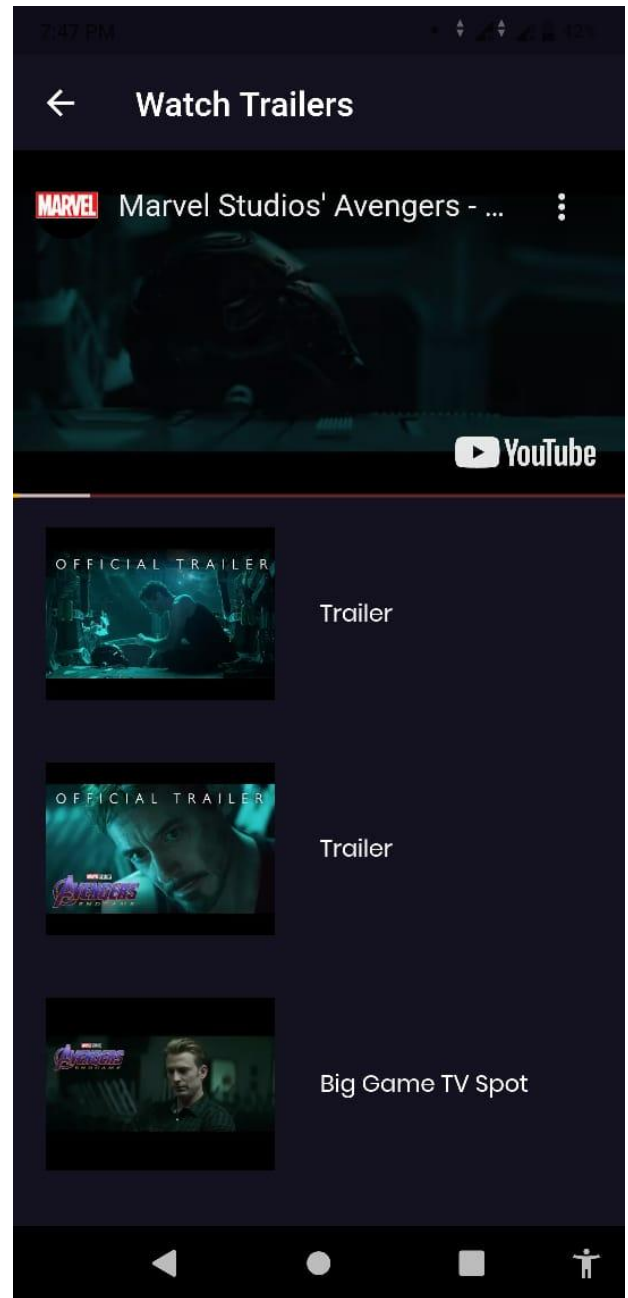
- Feedback



- **Movie Detail Screen**



- **Movie Trailer Screen**



- **Search Movie Screen**

7:56 PM



1917



### 1917

At the height of the First World War, two young British soldiers must cross enemy territory and deliver a message that will stop a deadly...



### Allied Forces: Making 1917

Learn how the one shot, 360-degree format was executed and the pivotal role Academy Award® winner Roger Deakins served in ...



### 1917: The Real Story

'1917: The Real Story' documents the true story behind the international blockbuster film '1917' - providing an insight into the real-life ...



### Russia 1917: Countdown to Revolution

Russia, 1917. After the abdication of Czar Nicholas II Romanov, the struggle for power confronts allies, enemies, factions and ideas;...

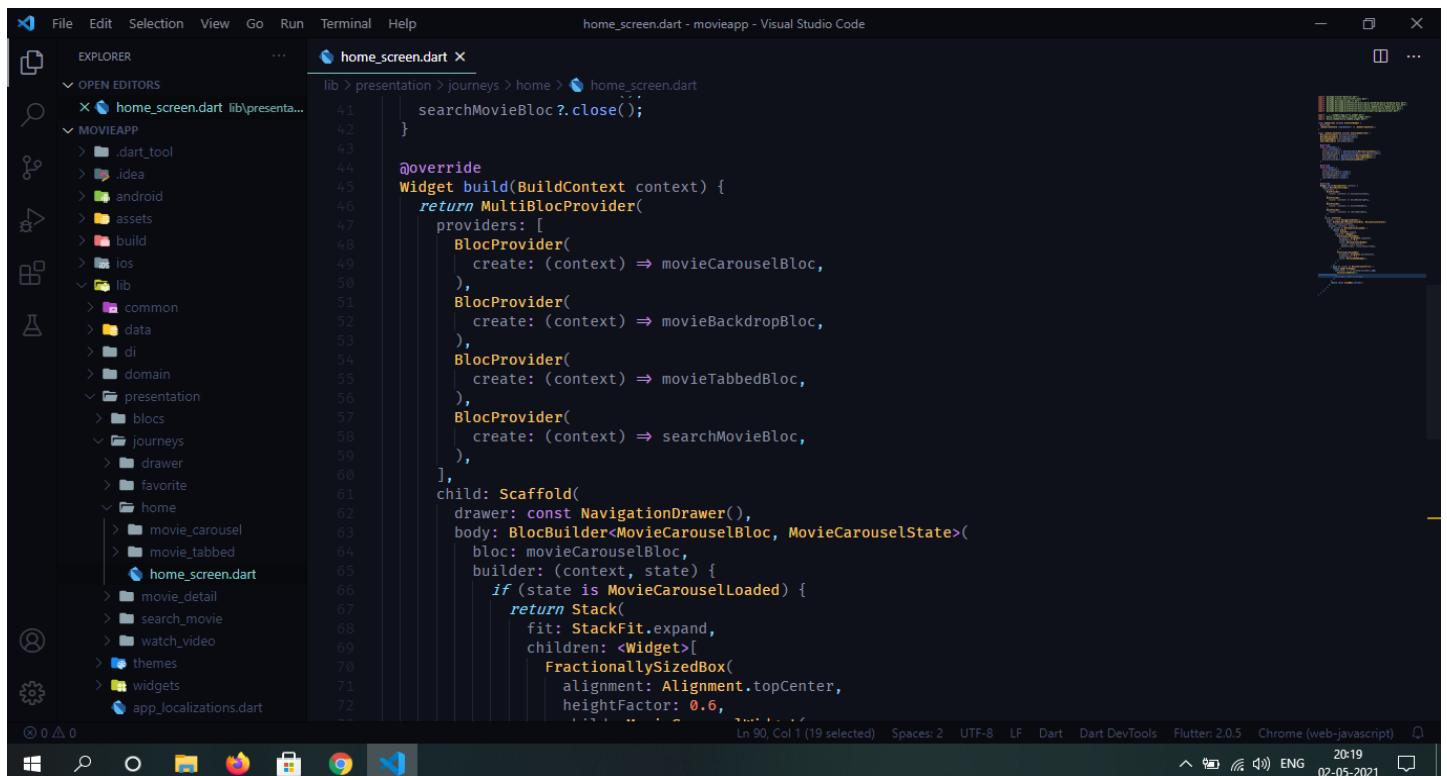
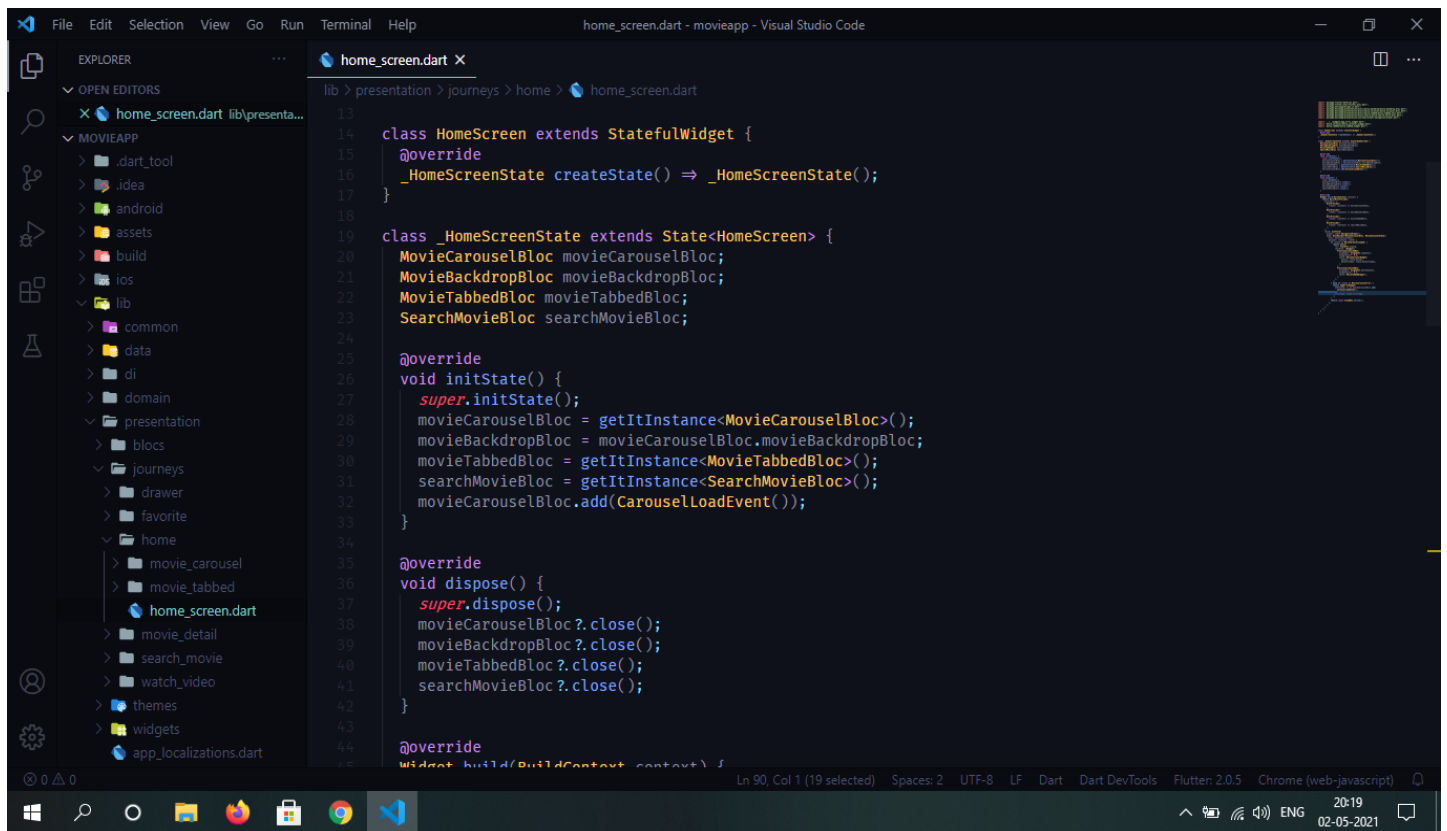


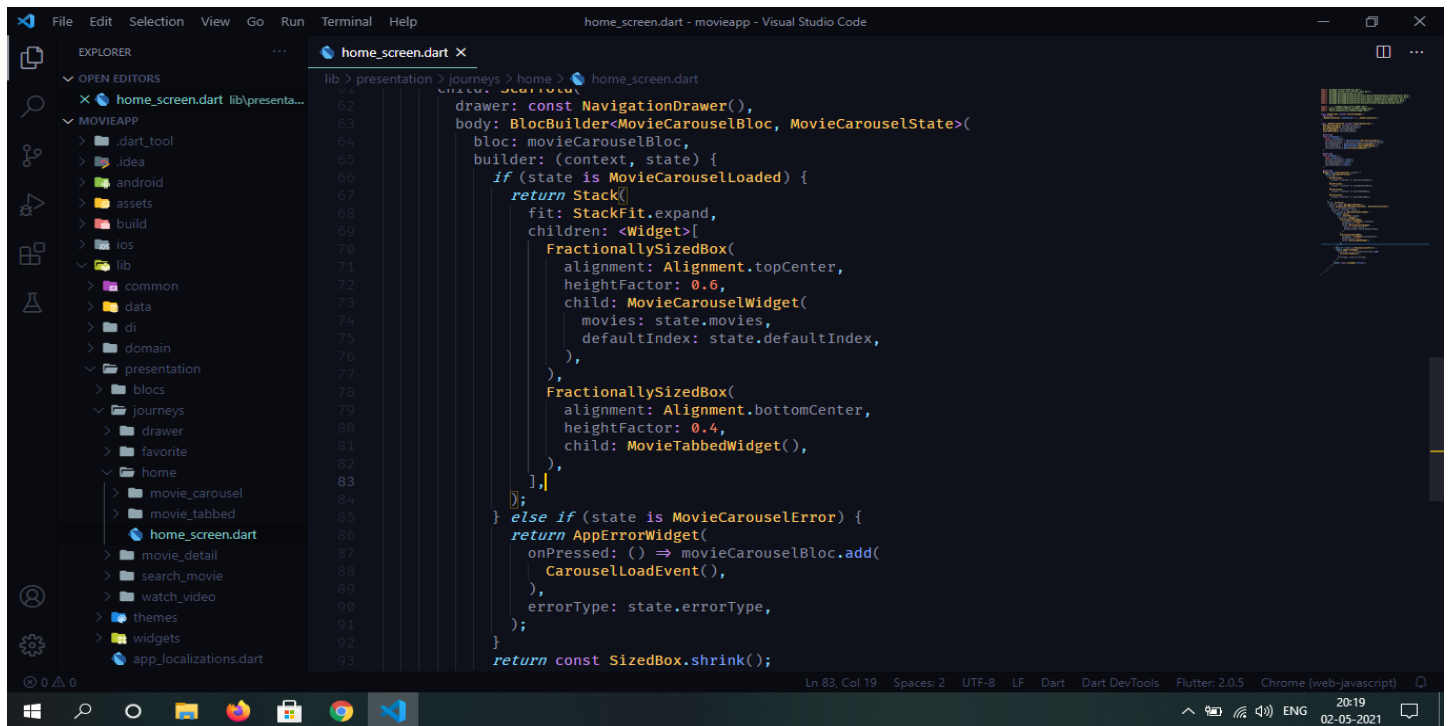
### Shock Troop

This powerful anti-war film statement focuses on the plight of a German unit in World War I that finds itself surrounded by British and ...

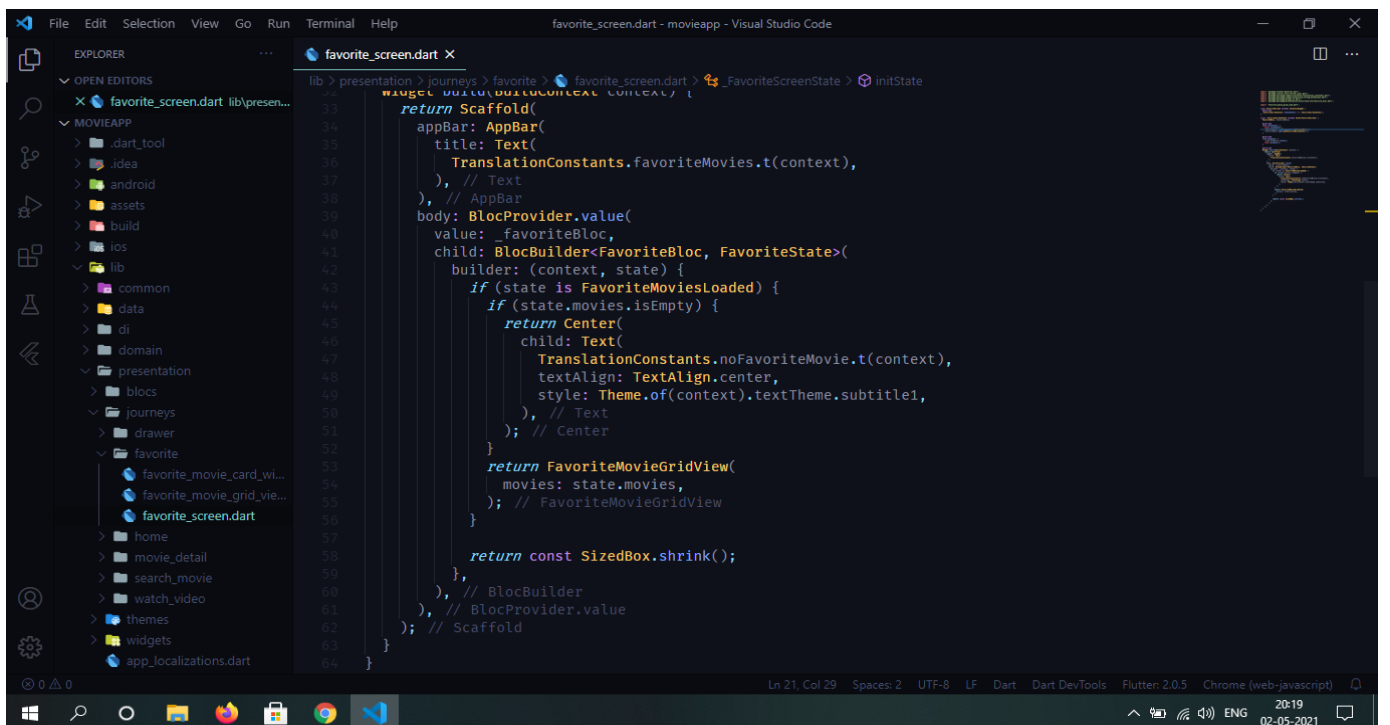
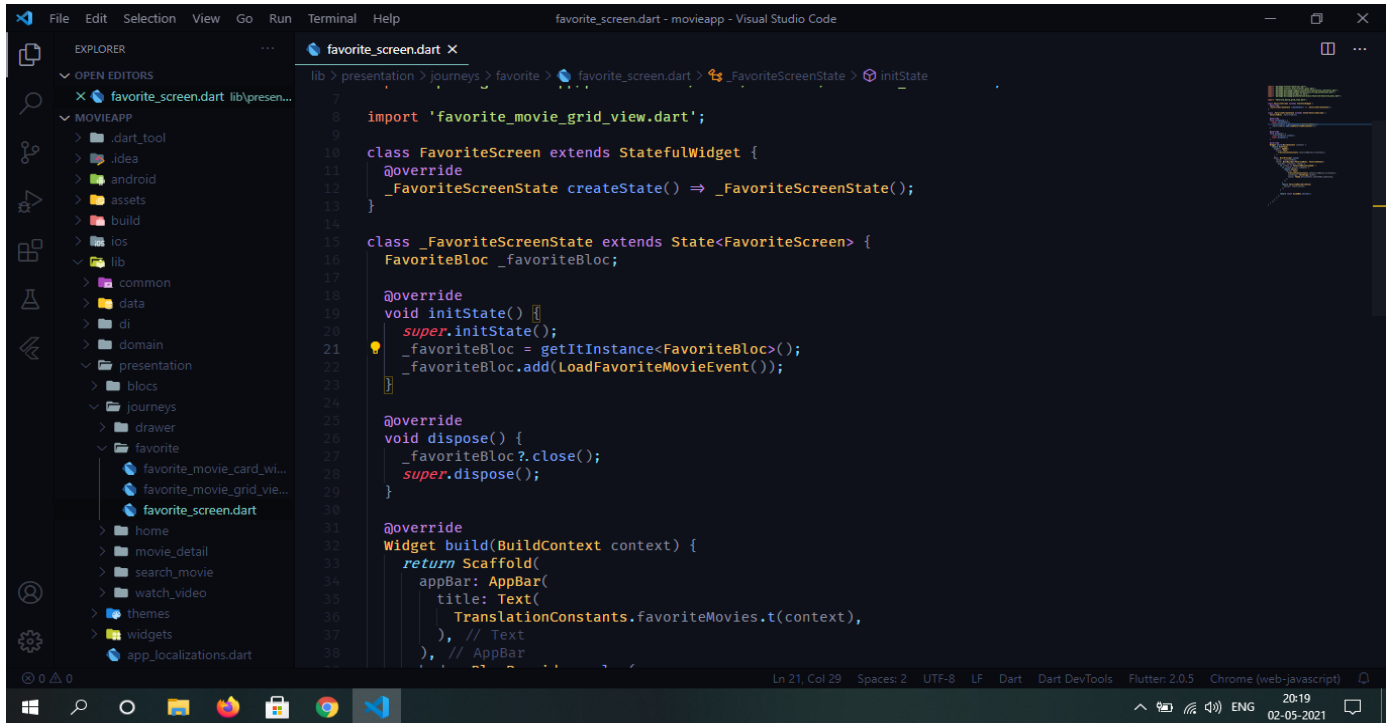


## CODE SCREENSHOTS

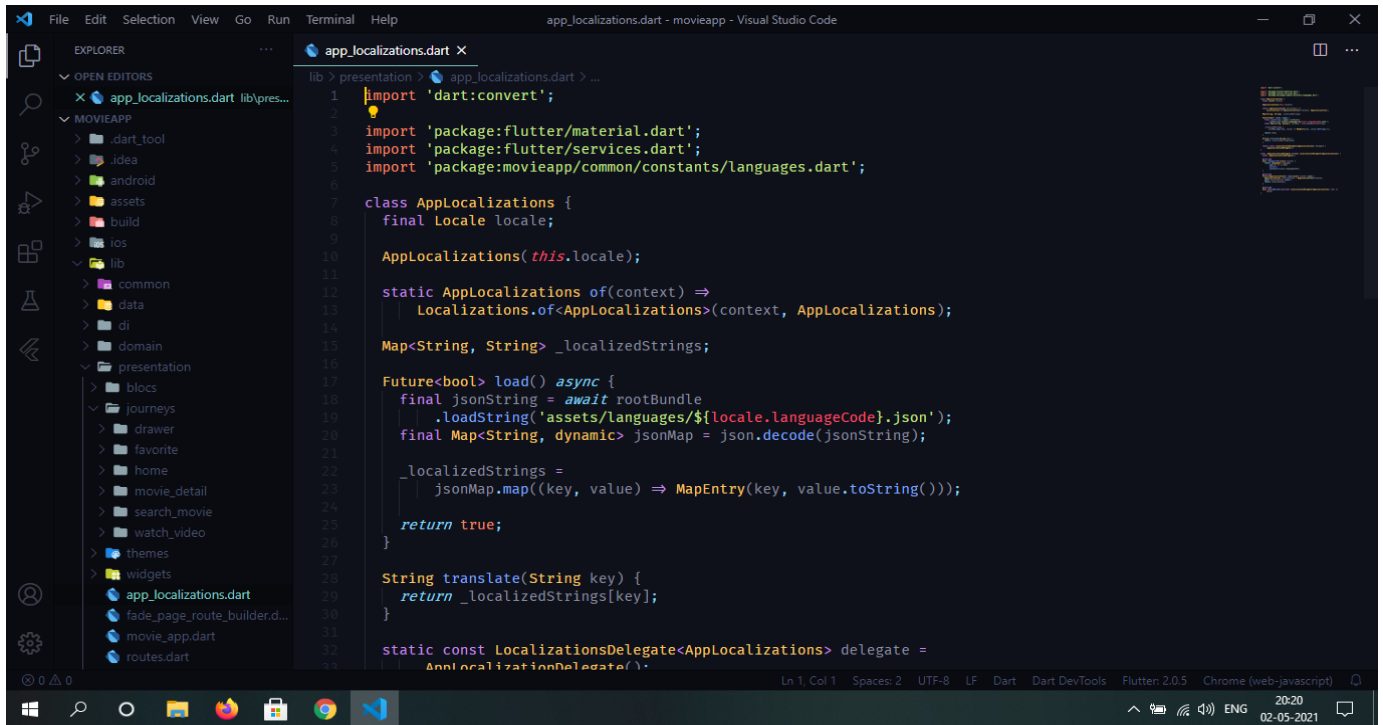




- Favorite Movies



- App Localization



```
lib > presentation > app_localizations.dart > ...
1  import 'dart:convert';
2
3  import 'package:flutter/material.dart';
4  import 'package:flutter/services.dart';
5  import 'package:movieapp/common/constants/languages.dart';
6
7  class AppLocalizations {
8    final Locale locale;
9
10   AppLocalizations(this.locale);
11
12   static AppLocalizations of(context) =>
13     Localizations.of<AppLocalizations>(context, AppLocalizations);
14
15   Map<String, String> _localizedStrings;
16
17   Future<bool> load() async {
18     final jsonString = await rootBundle
19       .loadString('assets/languages/${locale.languageCode}.json');
20     final Map<String, dynamic> jsonMap = json.decode(jsonString);
21
22     _localizedStrings =
23       jsonMap.map((key, value) => MapEntry(key, value.toString()));
24
25     return true;
26   }
27
28   String translate(String key) {
29     return _localizedStrings[key];
30   }
31
32   static const LocalizationsDelegate<AppLocalizations> delegate =
33     AppLocalizationDelegate();
```



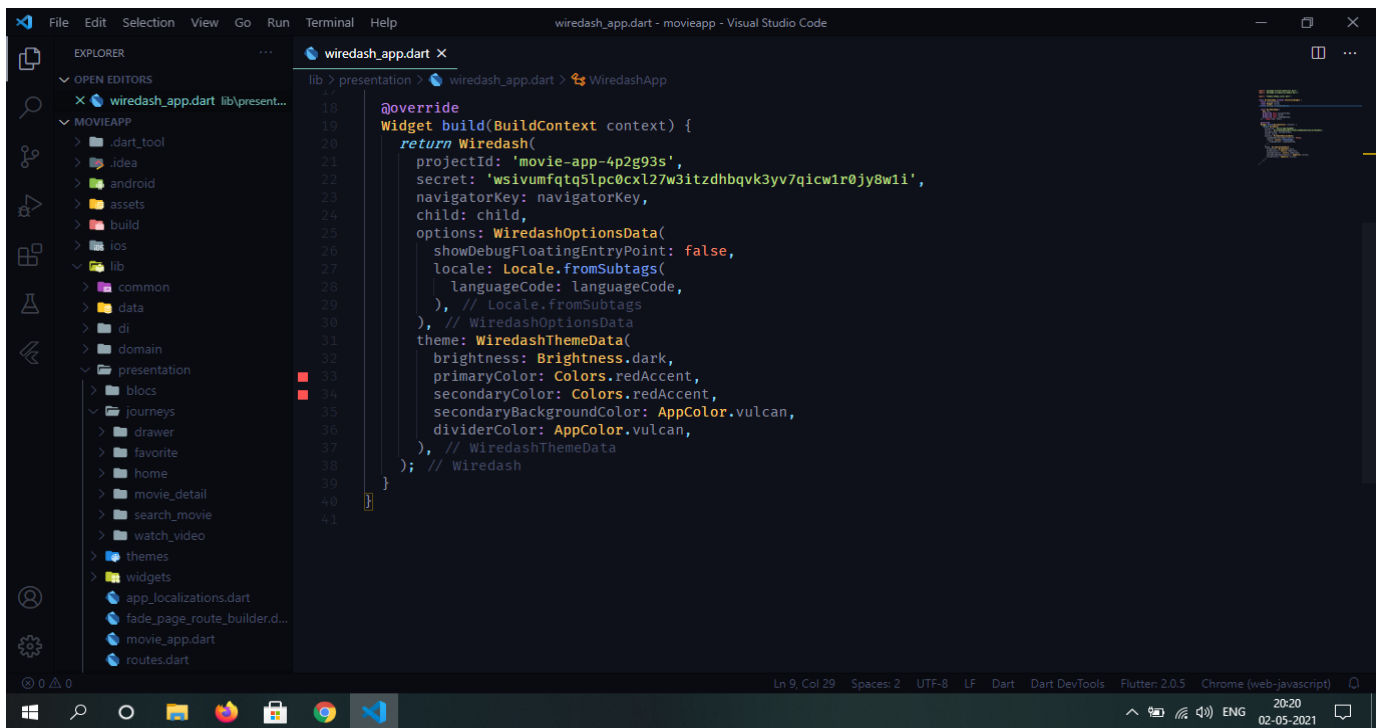
The screenshot shows the Visual Studio Code editor with the file `app_localizations.dart` open. The Explorer panel on the left shows the project structure, including the `lib` directory and the `presentation` folder. The code in the editor defines a `LocalizationsDelegate` for `AppLocalizations` and a `AppLocalizationDelegate` class that extends it. The class implements `isSupported`, `load`, and `shouldReload` methods.

```
lib > presentation > app_localizations.dart > ...
1  import 'package:flutter/material.dart';
2  import 'package:wiredash/wiredash.dart';
3
4  import 'themes/theme_color.dart';
5
6  class WiredashApp extends StatelessWidget {
7    final navigatorKey;
8    final Widget child;
9    final String languageCode;
10
11    const WiredashApp({
12      Key key,
13      @required this.navigatorKey,
14      @required this.child,
15      @required this.languageCode,
16    }) : super(key: key);
17
18    @override
19    Widget build(BuildContext context) {
20      return Wiredash(
21        projectId: 'movie-app-4p2g93s',
22        secret: 'wsivumftq5lpc0cx12/w3itzdhbqvk3yv7qicw1r0jy8w1i',
23        navigatorKey: navigatorKey,
24        child: child,
25        options: WiredashOptionsData(
26          showDebugFloatingEntryPoint: false,
27          locale: Locale.fromSubtags(
28            languageCode: languageCode,
29          ), // Locale.fromSubtags
30        ), // WiredashOptionsData
31        theme: WiredashThemeData(
32          brightness: Brightness.dark,
33          primaryColor: Colors.redAccent
34        ),
35      );
36    }
37  }
38
39  static const LocalizationsDelegate<AppLocalizations> delegate =
40    _AppLocalizationDelegate();
41
42  class _AppLocalizationDelegate extends LocalizationsDelegate<AppLocalizations> {
43    const _AppLocalizationDelegate();
44
45    @override
46    bool isSupported(Locale locale) {
47      return Languages.languages
48        .map((e) => e.code)
49        .toList()
50        .contains(locale.languageCode);
51    }
52
53    @override
54    Future<AppLocalizations> load(Locale locale) async {
55      AppLocalizations localizations = AppLocalizations(locale);
56      await localizations.load();
57      return localizations;
58    }
59
60    @override
61    bool shouldReload(covariant LocalizationsDelegate<AppLocalizations> old) =>
62      false;
63  }
```

- Feedback

The screenshot shows the Visual Studio Code editor with the file `wiredash_app.dart` open. The Explorer panel on the left shows the project structure, including the `lib` directory and the `presentation` folder. The code in the editor defines the `WiredashApp` class, which extends `StatelessWidget`. It implements the `build` method to return a `Wiredash` widget with specific configuration.

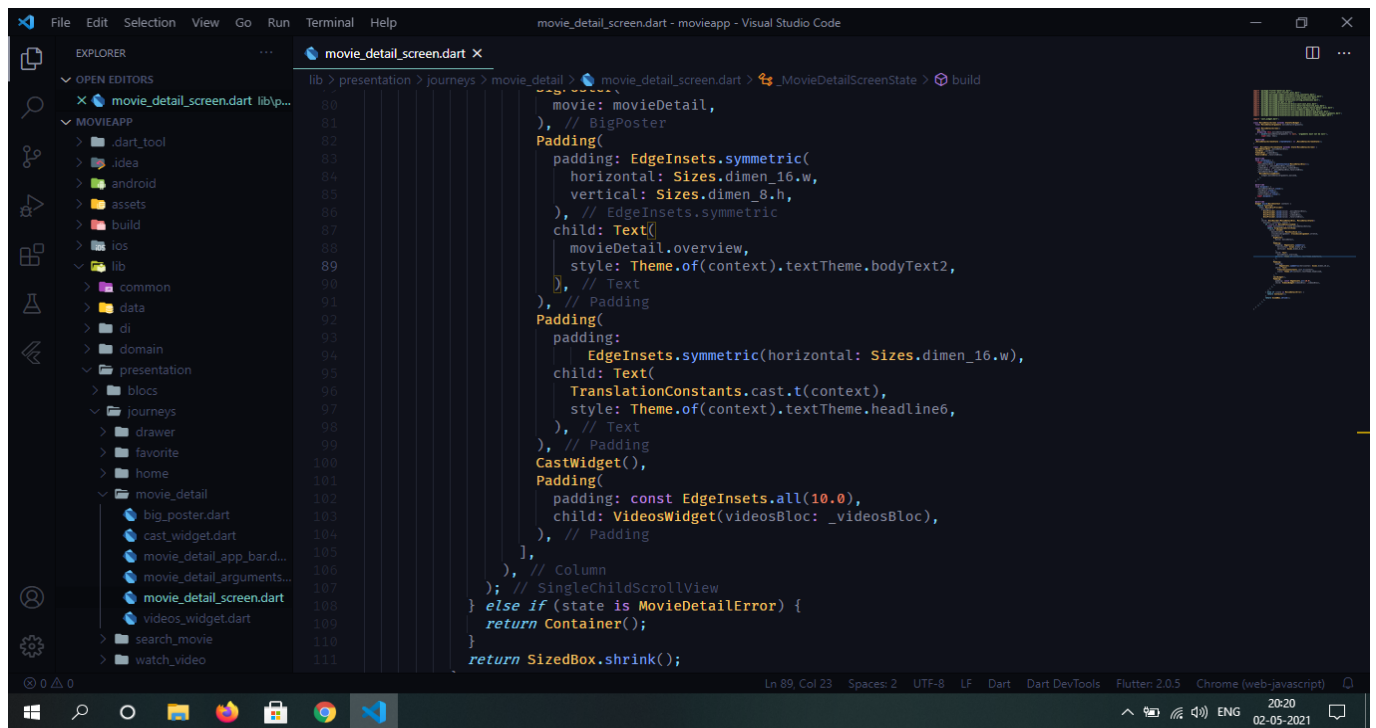
```
lib > presentation > wiredash_app.dart > WiredashApp
1  import 'package:flutter/material.dart';
2  import 'package:wiredash/wiredash.dart';
3
4  import 'themes/theme_color.dart';
5
6  class WiredashApp extends StatelessWidget {
7    final navigatorKey;
8    final Widget child;
9    final String languageCode;
10
11    const WiredashApp({
12      Key key,
13      @required this.navigatorKey,
14      @required this.child,
15      @required this.languageCode,
16    }) : super(key: key);
17
18    @override
19    Widget build(BuildContext context) {
20      return Wiredash(
21        projectId: 'movie-app-4p2g93s',
22        secret: 'wsivumftq5lpc0cx12/w3itzdhbqvk3yv7qicw1r0jy8w1i',
23        navigatorKey: navigatorKey,
24        child: child,
25        options: WiredashOptionsData(
26          showDebugFloatingEntryPoint: false,
27          locale: Locale.fromSubtags(
28            languageCode: languageCode,
29          ), // Locale.fromSubtags
30        ), // WiredashOptionsData
31        theme: WiredashThemeData(
32          brightness: Brightness.dark,
33          primaryColor: Colors.redAccent
34        ),
35      );
36    }
37  }
```



- **Movie Detail Screen**

```
lib > presentation > journeys > movie_detail > movie_detail_screen.dart > MovieDetailScreenState > build
11 import 'package:movieapp/presentation/blocs/videos/videos_bloc.dart';
12 import 'package:movieapp/presentation/journeys/movie_detail/big_poster.dart';
13 import 'package:movieapp/presentation/journeys/movie_detail/movie_detail_arguments.dart';
14 import 'package:movieapp/presentation/journeys/movie_detail/videos_widget.dart';
15
16 import 'cast_widget.dart';
17
18 class MovieDetailScreen extends StatefulWidget {
19   final MovieDetailArguments movieDetailArguments;
20
21   const MovieDetailScreen({
22     Key key,
23     @required this.movieDetailArguments,
24   }) : assert(movieDetailArguments != null, 'arguments must not be null'),
25       super(key: key);
26
27   @override
28   _MovieDetailScreenState createState() => _MovieDetailScreenState();
29 }
30
31 class _MovieDetailScreenState extends State<MovieDetailScreen> {
32   MovieDetailBloc _movieDetailBloc;
33   CastBloc _castBloc;
34   VideosBloc _videosBloc;
35   FavoriteBloc _favoriteBloc;
36
37   @override
38   void initState() {
39     super.initState();
40     _movieDetailBloc = getItInstance<MovieDetailBloc>();
41     _castBloc = _movieDetailBloc.castBloc;
42     _videosBloc = _movieDetailBloc.videosBloc;
43     _favoriteBloc = _movieDetailBloc.favoriteBloc;
```

```
42 _videosBloc = _movieDetailBloc.videosBloc;
43 _favoriteBloc = _movieDetailBloc.favoriteBloc;
44 _movieDetailBloc.add(
45   MovieDetailLoadEvent(
46     widget.movieDetailArguments.movieId,
47   ),
48 );
49
50 @override
51 void dispose() {
52   _movieDetailBloc?.close();
53   _castBloc?.close();
54   _videosBloc?.close();
55   _favoriteBloc?.close();
56   super.dispose();
57 }
58
59 @override
60 Widget build(BuildContext context) {
61   return Scaffold(
62     body: MultiBlocProvider(
63       providers: [
64         BlocProvider.value(value: _movieDetailBloc),
65         BlocProvider.value(value: _castBloc),
66         BlocProvider.value(value: _videosBloc),
67         BlocProvider.value(value: _favoriteBloc),
68       ],
69       child: BlocBuilder<MovieDetailBloc, MovieDetailState>(
70         builder: (context, state) {
71           if (state is MovieDetailLoaded) {
72             final movieDetail = state.movieDetailEntity;
```



- **Movie Trailer Screen**

```
lib > presentation > journeys > watch_video > watch_video_screen.dart > WatchVideoScreen
import 'package:movieapp/presentation/journeys/watch_video/watch_video_arguments.dart';
import 'package:youtube_player_flutter/youtube_player_flutter.dart';

class WatchVideoScreen extends StatefulWidget {
  final WatchVideoArguments watchVideoArguments;

  const WatchVideoScreen({
    Key key,
    @required this.watchVideoArguments,
  }) : super(key: key);

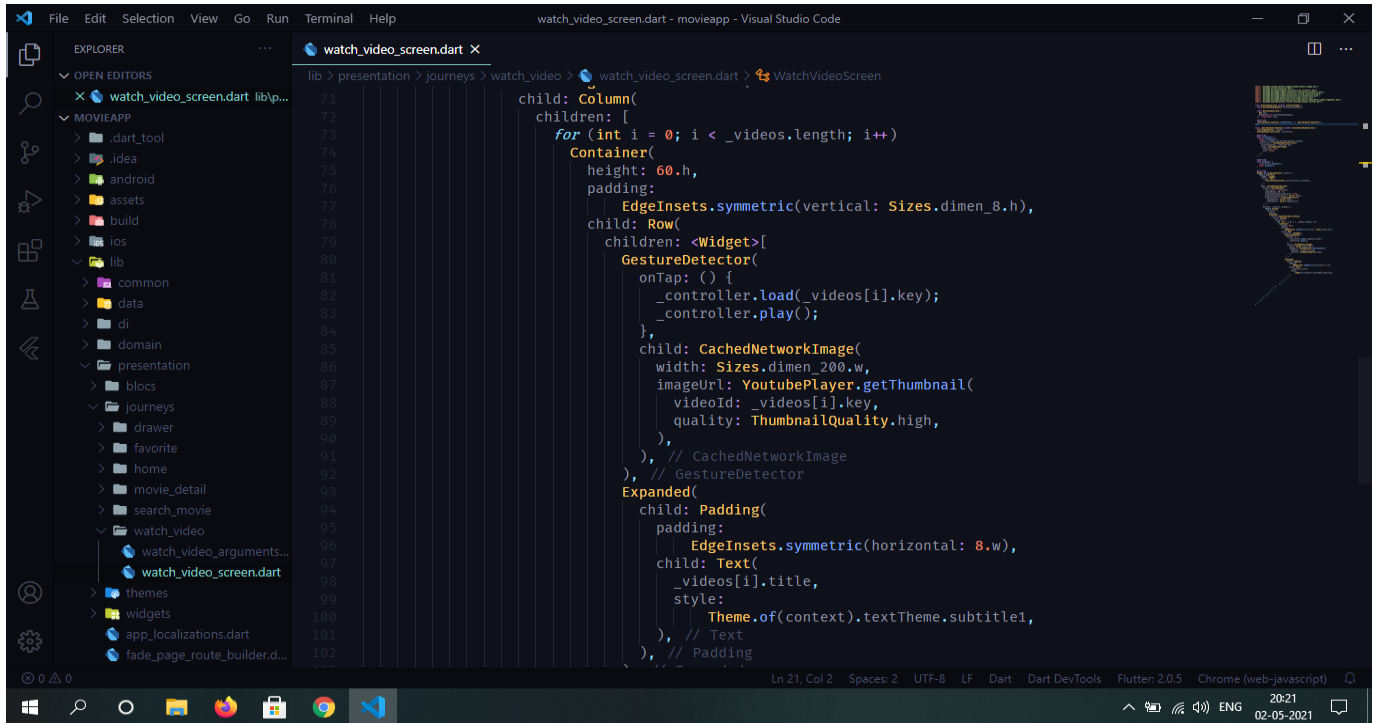
  @override
  _WatchVideoScreenState createState() => _WatchVideoScreenState();
}

class _WatchVideoScreenState extends State<WatchVideoScreen> {
  List<VideoEntity> _videos;
  YoutubePlayerController _controller;

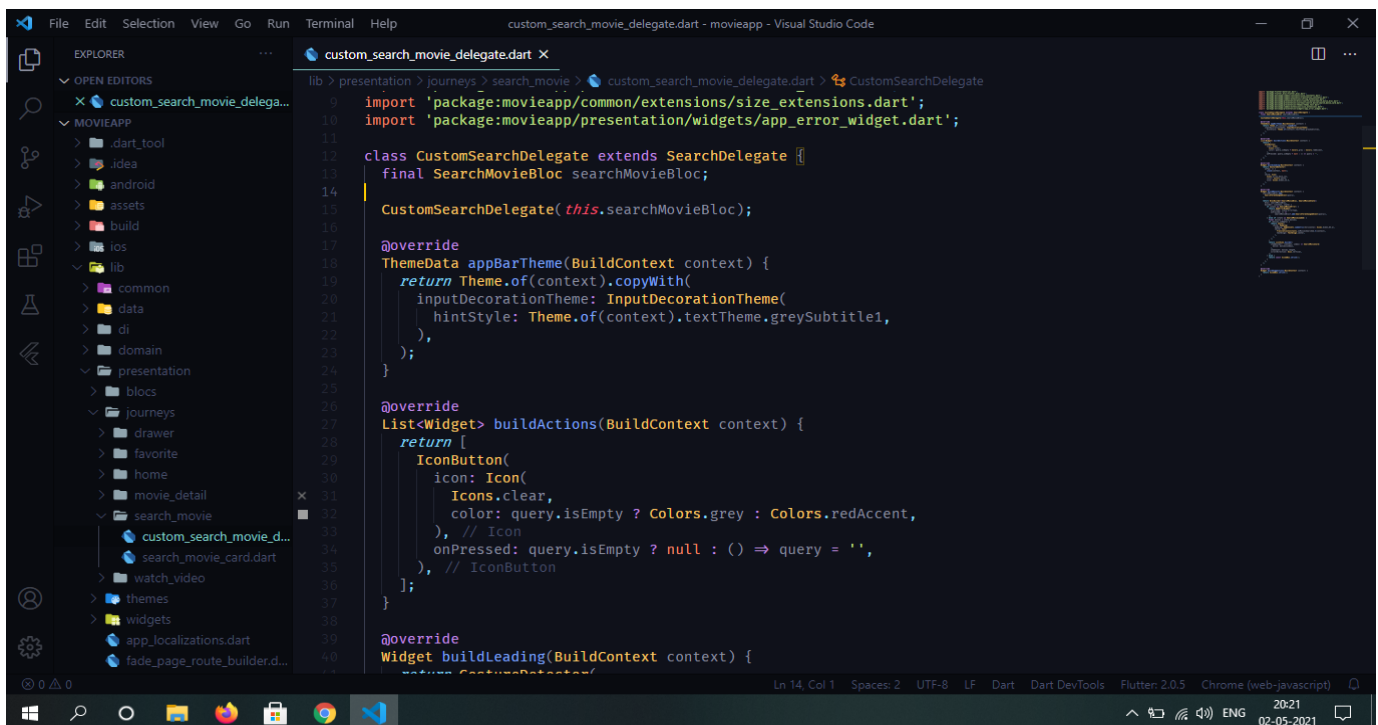
  @override
  void initState() {
    super.initState();
    _videos = widget.watchVideoArguments.videos;
    _controller = YoutubePlayerController(
      initialVideoId: _videos[0].key,
      flags: YoutubePlayerFlags(
        autoPlay: true,
        mute: false,
      ), // YoutubePlayerFlags
    ); // YoutubePlayerController
  }

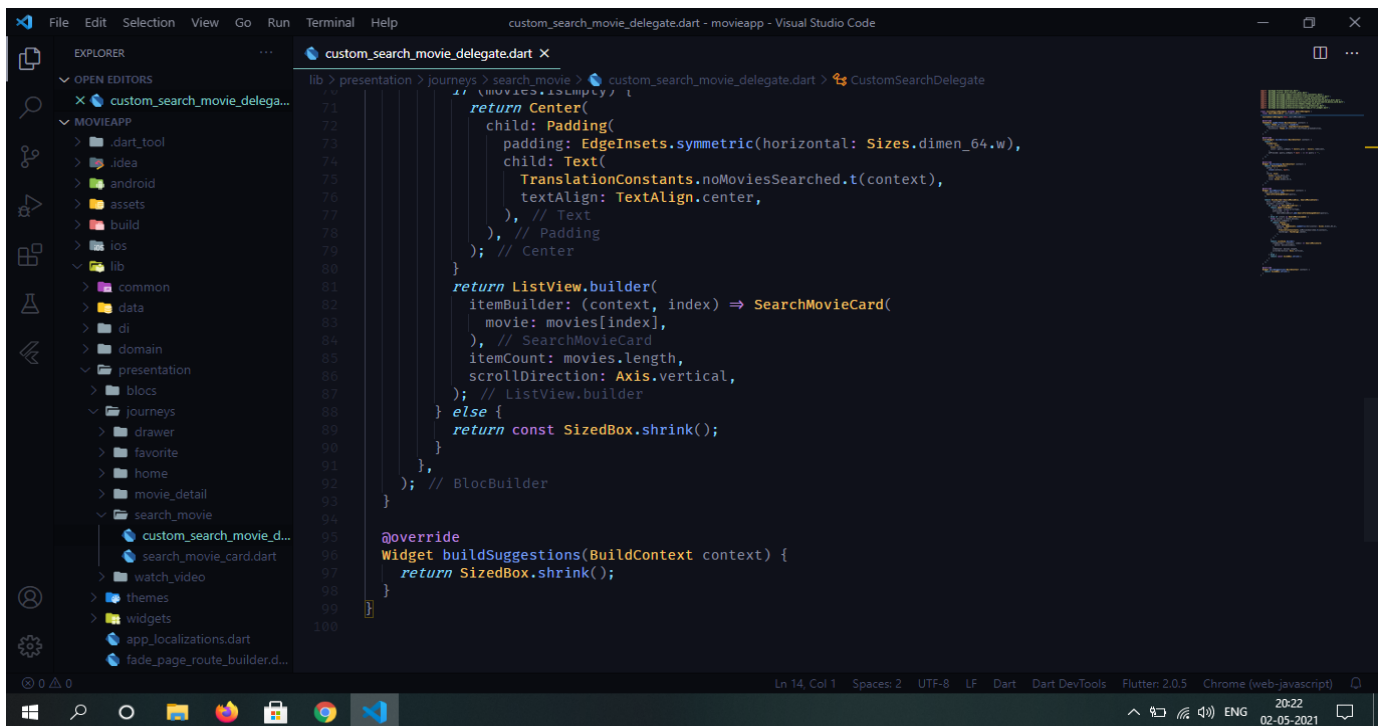
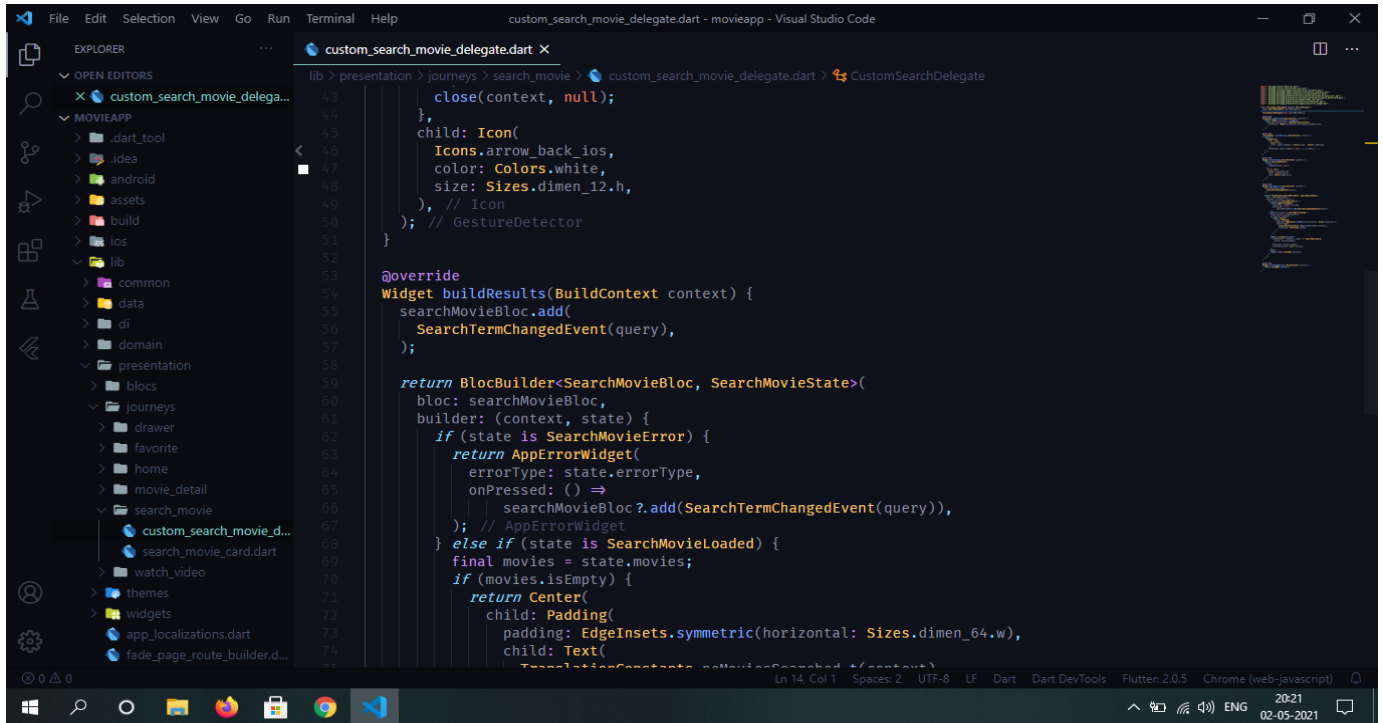
  @override
```

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text(
        TranslationConstants.watchTrailers.t(context),
      ), // Text
    ), // AppBar
    body: YoutubePlayerBuilder(
      player: YoutubePlayer(
        controller: _controller,
        aspectRatio: 16 / 9,
        showVideoProgressIndicator: true,
        progressIndicatorColor: Colors.amber,
        progressColors: ProgressBarColors(
          playedColor: Colors.amber,
          handleColor: Colors.amberAccent,
        ), // ProgressBarColors
      ), // YoutubePlayer
      builder: (context, player) {
        return Column(
          children: [
            player,
            Expanded(
              child: SingleChildScrollView(
                child: Column(
                  children: [
                    for (int i = 0; i < _videos.length; i++)
                      Container(
                        height: 60.h,
                        padding:
                          EdgeInsets.symmetric(vertical: Sizes.dimen_8.h),
```



- Search Movie Screen





## REFERENCES

- <https://flutter.dev>
- <https://flutter.dev/docs>
- <https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html>
- <https://www.freecodecamp.org/news/a-quick-introduction-to-clean-architecture-990c014448d2/>
- <https://api.flutter.dev/flutter/widgets/PageView-class.html>
- <https://developers.themoviedb.org/3>
- <https://wiredash.io/>