

Critical Evaluation

(word count=545)

Human-Computer Interaction (HCI) & Application Development of Cross Platform Applications

The multidisciplinary discipline of human-computer interaction, or HCI, is concerned with the design of computer technology and, in particular, with how people, or users, interact with computers. HCI has two sides to it. On one side it is about Academic discipline which is studying the way people interact with computer technology. The other side to it which is design discipline which is about how do you create interventions with technology that make a difference to people. In terms of HCI, cross-platform apps, which may operate on several devices and operating systems, present unique potential and challenges. Developing cross-platform applications necessitates creating software that functions flawlessly on several platforms and devices, promoting multiple accessibility and usability. HCI principles play a crucial role in this context since developers must make sure that consumers have a pleasant and consistent experience in a variety of settings.

There are three major components of HCI in cross-platform development,

1. **User interface (UI) consistency:** Cross-platform apps often handle varying screen sizes, resolutions, and embedding techniques. To give users a comfortable and predictable experience, it's critical to keep the user interface (UI) consistent. The inclusion of features, layouts, and platform-specific modifications that are adaptable makes the interface cohesive and easy to use.
2. **User-Driven Concepts (UCD):** UCD is a fundamental HCI principle that highlights end users' participation in the design and development process. It's critical to comprehend user preferences and behaviors across devices while developing cross-platform apps. Feedback loops, usability testing, and prototyping may all be used to improve UCD and make sure the application fulfills the needs and expectations of the users.
3. **Performance gains:** Frameworks like Flutter, which employ a single code base for several platforms, are frequently used in cross-platform enhancements. Thorough business considerations are necessary to guarantee seamless and uninterrupted communication on all platforms. When users encounter irregular or delayed navigation response times, HCI is affected. Performance requirements, platform-specific optimization, and effective coding techniques are crucial.

Critical Evaluation of FLUTFLIX (Movie Application)

Let's see how the application used the HCI principles.

Simple Navigation System: Users may easily browse through movies using this application because it adheres to well-established navigation patterns. For film segments, the use of a horizontal slider improves visibility, allowing for easy navigation and the application of HCI concepts. The application demonstrates responsive design by adjusting to various screen

sizes and formats. This responsiveness upholds the HCI principle of continuous consistency by facilitating a smooth user experience across devices. Interactive Features: Including interactive features that allow users to directly and meaningfully interact with the system, like making movie posters to view presentations or transportation, is in line with HCI principles. Features and accessibility: Examining an application's features, including its compatibility with screen readers, amount of text visuals, and font sizes that can be changed, might improve its integration. Supporting the inclusivity and usability principles of HCI requires making sure the program is accessible to users with varying requirements and skills.

Reference

Shneiderman, B. and Plaisant, C., 2010. *Designing the user interface: strategies for effective human-computer interaction*. Pearson Education India.

Tenner, E., 2015. The design of everyday things by Donald Norman. *Technology and Culture*, 56(3), pp.785-787.

Nielsen, J. and Budiu, R., 2013. *Mobile usability*. MITP-Verlags GmbH & Co. KG.

Design And Planning

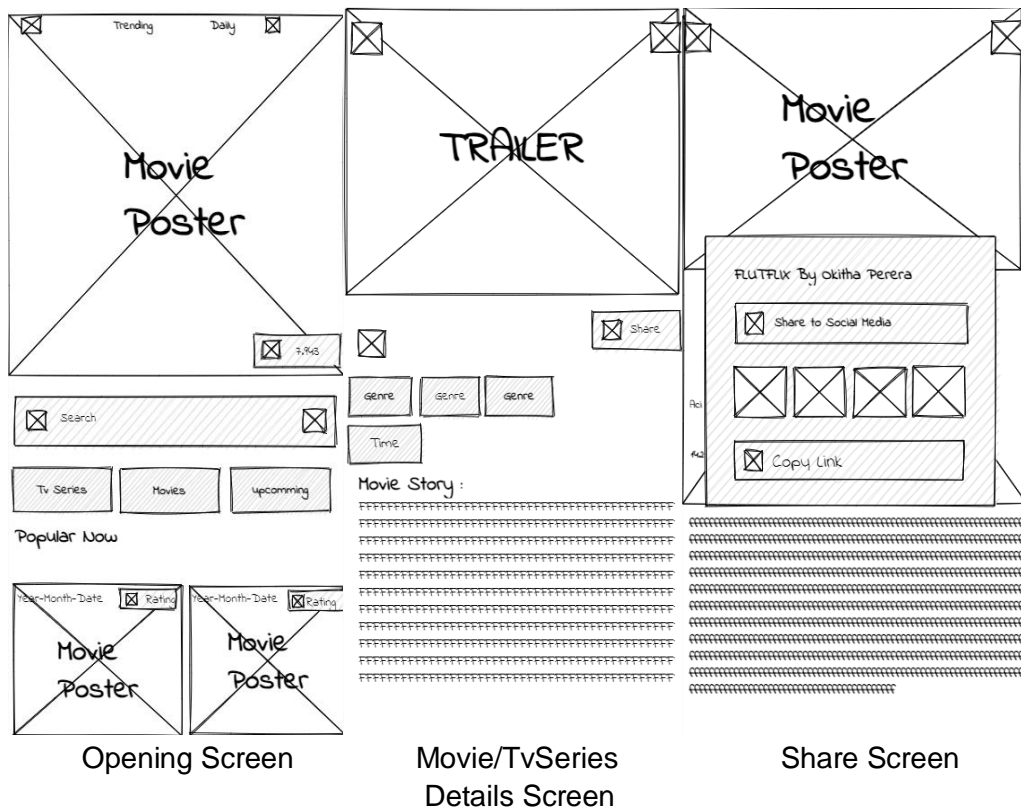
High-Level Requirements

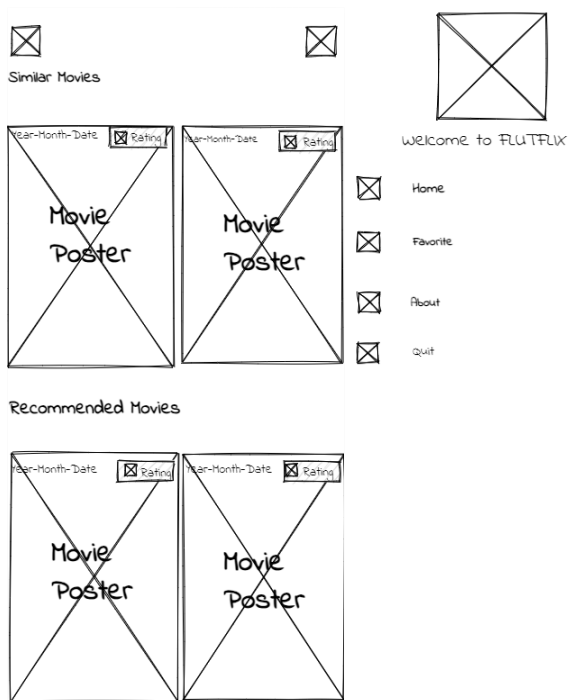
Feature	Targeted Platform
Movie List	Web,Android
TvSeries List	Web,Android
Upcoming	Web,Android
Trending	Web,Android
Popular	Web,Android
Search Functionality	Web,Android
Favourite Movies/TvSeries List	Android
Trailer PlayBack	Android

Planning User Interfaces

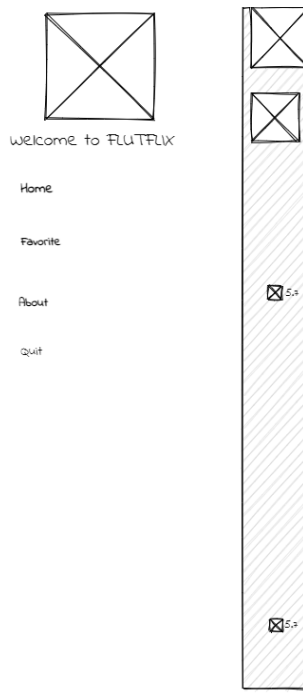
Creating Wireframes

Wireframes were first created for each of the main user interface screens, such as the home screen, detail screen, sign-in page, and search screen. In wireframes, important functionality, navigational components, and a basic layout were prioritised over graphic design details.

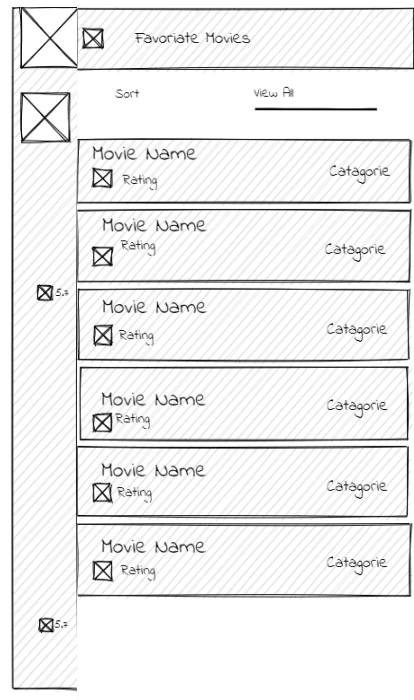




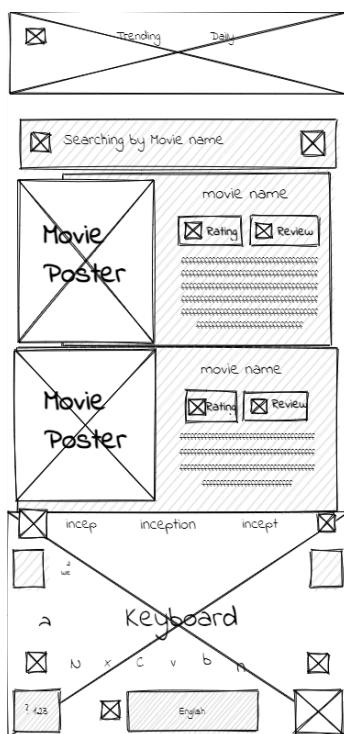
Scrolling Screen



Navigate SlideScreen



Favourite List Screen



Search Bar Function Screen

Planning Navigation

The planning of user behaviour and screen navigation processes resulted in easy transitions and smooth navigation. Navigation systems were implemented, including stack-based navigation for the details screen and bottom navigation for the main components.

Design Decisions

1. Flutter Documentations(On official Website)

For creating cross-platform applications, helpful examples, technical details, and design frameworks were made available by the official Flutter documentation and community resources. Optimising the use of Flutter widgets to generate platform-specific UI designs and interactions was made possible by investigating Flutter objects.

2. Friedman, V. and Lennartz, S., 2007. Smashing Magazine.

Smashing Magazine is a famous online e-book devoted to internet design and improvement. Their articles, tutorials, and case studies cover a wide range of subjects related to UI/UX design, the front-end development, and responsive design.

3. The UX Collective

User research, product development, and UX design are the main topics of discussion on The UX Collective, an online community forum and journal. It includes essays, case studies on design, and carefully chosen resources written by professionals in the field.

Mapping Wireframes to Flutter Layouts & Structuring

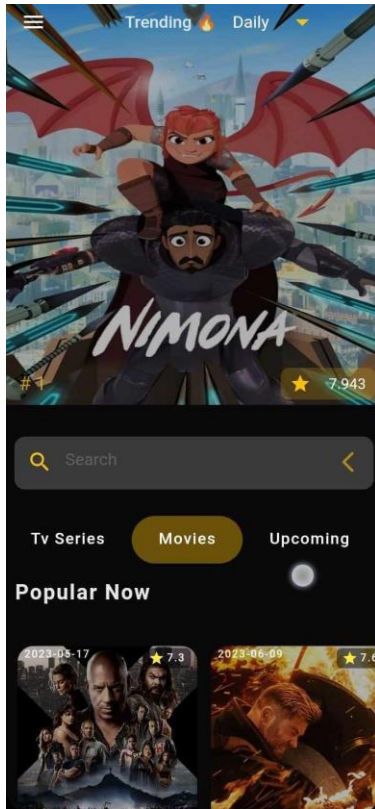
The UI elements were divided into smaller widgets and the wireframes were mapped to reusable Flutter layouts. Using Flutter widgets like MaterialApp, Scaffold, AppBar, BottomNavigationBar, ListView, and GestureDetector, the more realistic screens were set up in accordance with the target platforms' (web/android) design requirements.

Implementation

Top 5 Features

1. Opening Screen (Movie Slider)

Screenshot:



Source Code Snippet:

```
CarouselSlider(  
  options: CarouselOptions(  
    viewportFraction: 1,  
    autoPlay: true,  
    autoPlayInterval: Duration(seconds: 2),  
    height: MediaQuery.of(context).size.height  
  ),  
  items: trendingweek.map((i) {  
    return Builder(builder: (BuildContext context) {  
      return GestureDetector(  
        onTap: () {  
          Navigator.push(  
            context,  
            MaterialPageRoute(  
              builder: (context) => DescriptionCheckUI(i['id'], i['media_type'])  
            ),  
          );  
        },  
      );  
    },  
  ),  
  child: Container(  
    width: MediaQuery.of(context).size.width,  
    decoration: BoxDecoration(  
      image: DecorationImage(  
        colorFilter: ColorFilter.mode(  
          Colors.black.withOpacity(0.3),
```

```

        BlendMode.darken,
      ),
      image: NetworkImage(
        'https://image.tmdb.org/t/p/w500${i['poster_path']}',
      ),
      fit: BoxFit.fill,
    ),
  ),
  child: Column(
    mainAxisAlignment: MainAxisAlignment.end,
    children: [
      Row(
        mainAxisAlignment: MainAxisAlignment.spaceBetween,
        children: [
          Container(
            child: Text(
              ' # ${i['indexno'] + 1}',
              style: TextStyle(
                color: Colors.amber.withOpacity(0.7),
                fontSize: 18,
              ),
            ),
          ),
          Container(
            margin: EdgeInsets.only(right: 8, bottom: 5),
            width: 90,
            padding: EdgeInsets.all(5),
            decoration: BoxDecoration(
              color: Colors.amber.withOpacity(0.2),
              borderRadius: BorderRadius.all(
                Radius.circular(8),
              ),
            ),
          ),
          child: Row(
            mainAxisAlignment: MainAxisAlignment.spaceEvenly,
            children: [
              Icon(
                Icons.star,
                color: Colors.amber,
                size: 20,
              ),
              SizedBox(width: 10),
              Text(
                '${i['vote_average']}',
                style: TextStyle(
                  color: Colors.white,
                  fontWeight: FontWeight.w400,
                ),
              ),
            ],
          ),
        ],
      ),
    ],
  ),
);
});
}).toList(),
)

```

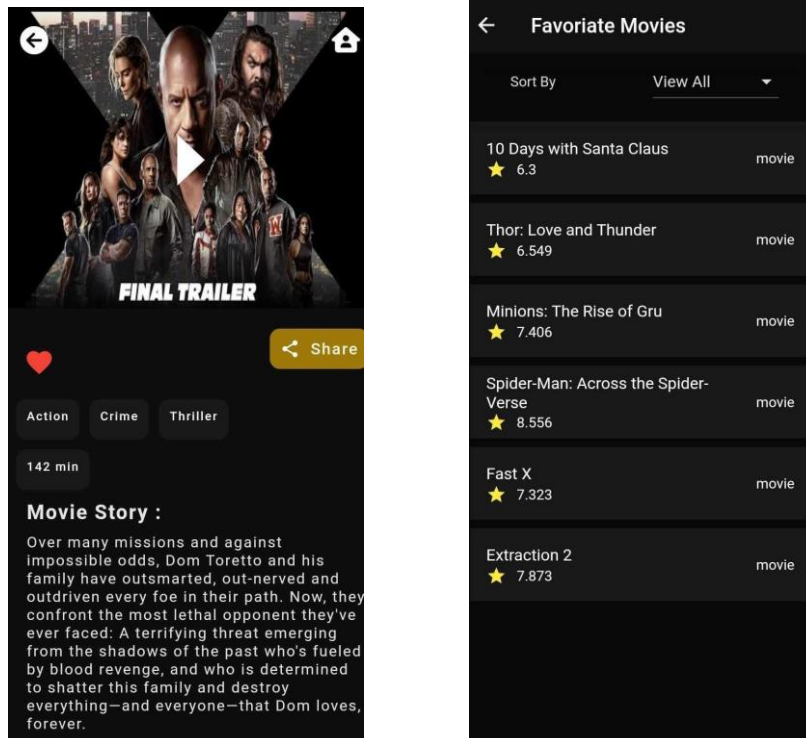
Explanation:

The movie slider list feature displays a horizontally scrollable list of movies or TV series. It fetches data from the API and dynamically populates the list with movie posters, release

dates, and ratings. Tapping on a movie poster navigates to the respective movie details screen.

2. Favourite Movies Screen

Screenshot:



Source Code Snippet:

```
class addtofavorite extends StatefulWidget {  
  final dynamic id;  
  final String type;  
  final dynamic Details;  
  
  addtofavorite({  
    required this.id,  
    required this.type,  
    required this.Details,  
  });  
  
  @override  
  State<addtofavorite> createState() => _addtofavoriteState();  
}  
  
class _addtofavoriteState extends State<addtofavorite> {  
  Future checkfavorite() async {  
    FavMovielist()
```

```

        .search(widget.id.toString(), widget.Details[0]['title'].toString(),
            widget.type)
        .then((value) {
            if (value == 0) {
                print('notanythingfound');
                favoritecolor = Colors.white;
            } else {
                //print the tmdbname and tmdbid and tmdbtype and tmdbrating from database

                print('surelyfound');
                favoritecolor = Colors.red;
            }
        });
        await Future.delayed(Duration(milliseconds: 100));
    }

```

```

Color? favoritecolor;

```

```

addatabase(
    id,
    name,
    type,
    rating,
    customcolor,
) async {
    if (customcolor == Colors.white) {
        FavMovielist().insert({
            'tmdbid': id,
            'tmdbtype': type,
            'tmdbname': name,
            'tmdbrating': rating,
        });
        favoritecolor = Colors.red;
        Fluttertoast.showToast(
            msg: "Added to Favorite",
            toastLength: Toast.LENGTH_SHORT,
            gravity: ToastGravity.BOTTOM,
            timeInSecForIosWeb: 1,
            backgroundColor: Colors.green,
            textColor: Colors.white,
            fontSize: 16.0);
    } else if (customcolor == Colors.red) {
        FavMovielist().deletespecific(id, type);
        favoritecolor = Colors.white;
        Fluttertoast.showToast(
            msg: "Removed from Favorite",
            toastLength: Toast.LENGTH_SHORT,
            gravity: ToastGravity.BOTTOM,
            timeInSecForIosWeb: 1,
            backgroundColor: Colors.red,
            textColor: Colors.white,
            fontSize: 16.0);
    }
}

```

```

@override
void initState() {
    super.initState();
}

```

```

    checkfavorite();
}

@override
Widget build(BuildContext context) {
  return Container(
    height: 80,
    width: MediaQuery.of(context).size.width,
    child:
      Row(mainAxisAlignment: MainAxisAlignment.spaceBetween, children: [
        Container(
          alignment: Alignment.centerLeft,
          width: MediaQuery.of(context).size.width / 2,
          child: FutureBuilder(
            future: checkfavorite(),
            builder: (context, snapshot) {
              if (snapshot.connectionState == ConnectionState.done) {
                return Container(
                  height: 55,
                  margin: EdgeInsets.only(top: 20),
                  padding: EdgeInsets.all(8),
                  child: Container(
                    decoration: BoxDecoration(
                      borderRadius: BorderRadius.circular(20),
                    ),
                    height: 50,
                    width: 50,
                    child: IconButton(
                      icon: Icon(Icons.favorite,
                        color: favoritecolor, size: 30),
                      onPressed: () {
                        print('pressed');
                        setState(() {
                          adddataatbase(
                            widget.id.toString(),
                            widget.Details[0]['title'].toString(),
                            widget.type,
                            widget.Details[0]['vote_average'].toString(),
                            favoritecolor,
                          );
                        });
                      },
                    ),
                  ),
                );
              } else {
                return Container(
                  height: 55,
                  width: MediaQuery.of(context).size.width,
                );
              }
            },
          ),
        ),
      ],
    ),

```

Explanation:

The Favorite Movies screen allows users to view and manage their favorite movies. It retrieves data from the local SQLite database and displays movie titles, ratings, and types. Users can swipe to delete movies from their favorites list.

3. SearchBar Functioning Screen

Screenshot:



Source Code Snippet:

```
class searchbarfun extends StatefulWidget {  
  const searchbarfun({super.key});  
  
  @override  
  State<searchbarfun> createState() => _searchbarfunState();  
}
```

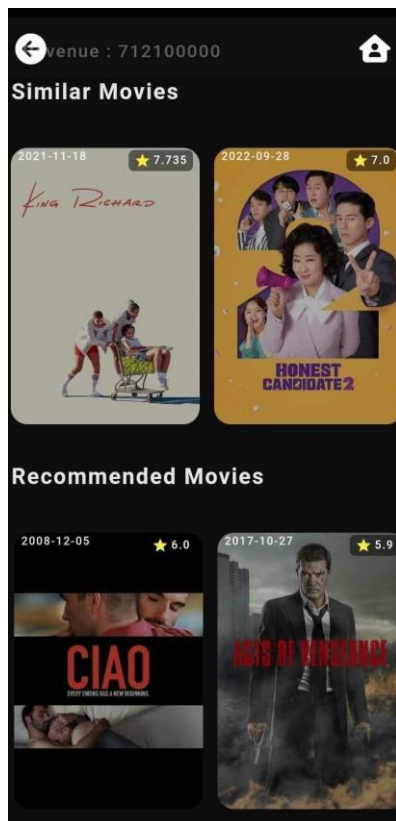
Explanation:

The search functionality allows users to search for movies or TV series based on keywords or titles. It consists of a search bar where users can input their search queries. As

the user types, the application dynamically updates the search results, displaying matching movies or TV series in real-time. The search functionality enhances user experience by providing a quick and efficient way to find specific content within the application.

4. Similar & Recommended Movies List Screen

Screenshot:



Source Code Snippet:

```
var similarmoviesresponse = await http.get(Uri.parse(similarmoviesurl));
if (similarmoviesresponse.statusCode == 200) {
  var similarmoviesjson = jsonDecode(similarmoviesresponse.body);
  for (var i = 0; i < similarmoviesjson['results'].length; i++) {
    similarmovieslist.add({
      "poster_path": similarmoviesjson['results'][i]['poster_path'],
      "name": similarmoviesjson['results'][i]['title'],
      "vote_average": similarmoviesjson['results'][i]['vote_average'],
      "Date": similarmoviesjson['results'][i]['release_date'],
      "id": similarmoviesjson['results'][i]['id'],
    });
  }
} else {}
// print(similarmovieslist);
//////////recommended movies
var recommendedmoviesresponse =
  await http.get(Uri.parse(recommendedmoviesurl));
if (recommendedmoviesresponse.statusCode == 200) {
  var recommendedmoviesjson = jsonDecode(recommendedmoviesresponse.body);
  for (var i = 0; i < recommendedmoviesjson['results'].length; i++) {
```

```

recommendedmovieslist.add({
  "poster_path": recommendedmoviesjson['results'][i]['poster_path'],
  "name": recommendedmoviesjson['results'][i]['title'],
  "vote_average": recommendedmoviesjson['results'][i]['vote_average'],
  "Date": recommendedmoviesjson['results'][i]['release_date'],
  "id": recommendedmoviesjson['results'][i]['id'],
});
}
}

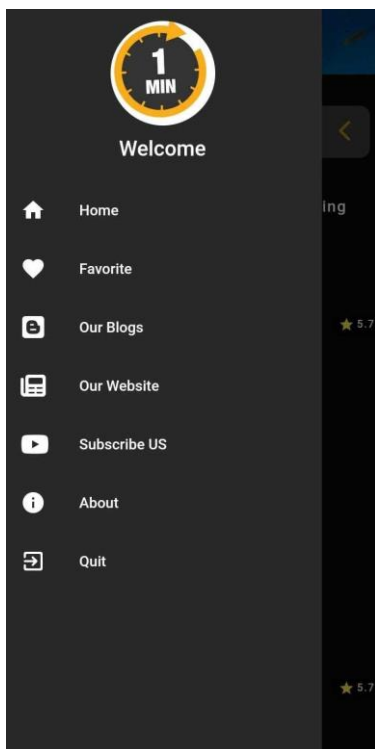
```

Explanation:

This feature displays a list of movies that are similar or recommended based on the selected movie. It utilises the TMDB API to fetch related movie data and presents it to the user. The list includes movie posters, titles, release dates, and ratings. Users can explore additional movies that match their interests and preferences, enhancing their movie discovery experience.

5. Section Home Screen

Screenshot:



Source Code Snippet:

```

class drawerfunc extends StatefulWidget {
  drawerfunc({
    super.key,
  });
}

```

```

@override
State<drawerfunc> createState() => _drawerfuncState();
}

class _drawerfuncState extends State<drawerfunc> {
  File? _image;

  Future<void> SelectImage() async {
  }

  @override
  void initState() {
    super.initState();
    SharedPreferences.getInstance().then((sp) {
      setState(() {
        _image = File(sp.getString('imagepath')!);
      });
    });
  }

  @override
  Widget build(BuildContext context) {
    return Drawer(
      child: Container(
        color: Color.fromRGBO(18, 18, 18, 0.9),
        child: ListView(
          children: [
            DrawerHeader(
              child: Container(
                child: Column(
                  children: [
                    GestureDetector(
                      onTap: () async {
                        await SelectImage();
                        //toast message
                        Fluttertoast.showToast(
                          msg: "Image Changed",
                          toastLength: Toast.LENGTH_SHORT,
                          gravity: ToastGravity.BOTTOM,
                          timeInSecForIosWeb: 1,
                          backgroundColor: Colors.grey,
                          textColor: Colors.white,
                          fontSize: 16.0);
                      },
                    child: _image == null
                      ? CircleAvatar(
                          radius: 50,
                          backgroundImage: AssetImage('asset/user.png'),
                        )
                      : CircleAvatar(
                          radius: 50,
                          backgroundImage: FileImage(_image!),
                        ),
                  ),
                ),
                SizedBox(height: 10),
                Text(
                  'Welcome to FLUTFLIX',
                  style: TextStyle(color: Colors.white, fontSize: 20),
                )
              ],
            ),
          ),
        ),
      listtilefunc('Home', Icons.home, onTap: () {
        //close drawer
        Navigator.pop(context);
      })
    );
  }
}

```

```

    )),
    listtilefunc('Favorite', Icons.favorite, onTap: () {
      Navigator.push(context,
        MaterialPageRoute(builder: (context) => FavoriteMovies()));
    )),
    listtilefunc('About', Icons.info, onTap: () {
      showDialog(
        context: context,
        builder: (context) {
          return AlertDialog(
            backgroundColor: Color.fromRGBO(18, 18, 18, 0.9),
            title: overviewtext(
              'FLUTFLIX was made by Okitha Perera by using TMDB/Flutter.'),
            actions: [
              TextButton(
                onPressed: () {
                  Navigator.pop(context);
                },
                child: Text('Ok'))
            ],
          );
        });
    )),
    listtilefunc('Quit', Icons.exit_to_app_rounded, onTap: () {
      SystemNavigator.pop();
    )),
  ],
),
);
}
}

Widget listtilefunc(String title, IconData icon, {Function? onTap}) {
  return GestureDetector(
    onTap: onTap as void Function()?,
    child: ListTile(
      leading: Icon(
        icon,
        color: Colors.white,
      ),
      title: Text(
        title,
        style: TextStyle(color: Colors.white),
      ),
    ),
  );
}

```

Explanation:

The Section Home UI feature provides a central hub for users to explore different sections of the application, such as Home, Favourite, About, Quit.

Test Plan

Unit Tests

Favorite Films Database:

Check the movie's availability after adding it to your list of preferences.
Make sure you select the movie from your list of favorites and that the right data is sent back.
Verify that the movie is not there in the list and make sure you are removing it.

API Checking:

Verify that the movie data you try to retrieve from the API follows the necessary format.

Trailer Playing:

By simulating the API response to the trailer URL, you may test the functionality of trailer playback.
Verify that trailers function properly in the app and elegantly repair any errors.

UI Elements:

Test the visibility and layout of UI elements such as buttons, images, and text.
Ensure UI elements respond correctly to user interactions such as taps and swipes.

Navigation:

Check how easy it is to navigate between the application's various screens and sections.
Make that the navigation routes are accurately configured and lead to the desired screen.

Widget Tests

1: Viewing Popular Movies

Test Item: Go to the "Popular Now" section to make sure that the list of popular movies is displayed.
Expected Outcome: Film posters with titles and release date should appear.
Results: Pass

2: Viewing Movie Results

Test Item: To view the lyrics, click on the movie poster.
Expected Outcome: A synopsis, title, Genre, Time, and release date of the movie should all be included in the description.

Results: Pass

3: Movie Trailers

Test Item: Click the "play" button for the movie

Expected Outcome: The movie trailer should play in the application.

Results: Fail (In Chrome)