Experiment no: 5

Aim: To apply navigation, routing and gestures in Flutter App

Theory:

Introduction:Navigation, routing, and gestures are essential aspects of a Flutter application to ensure seamless interaction and smooth user experience. Navigation allows users to move between different screens, routing defines how screens are structured and managed, and gestures enable touch interactions like taps, swipes, and long presses. Implementing these functionalities properly enhances app usability and improves the overall flow of the application.

2. Objective:The primary objectives of this experiment are:

- To understand navigation and routing concepts in Flutter.
- To implement navigation between different screens using routes.
- To integrate gestures for better user interaction.

3. Importance of Navigation, Routing, and Gestures

Navigation enables users to switch between different screens or pages in an app. It ensures smooth transitions between different UI components, making the app interactive and user-friendly.

Routing:Routing defines the structure of app screens and their navigation paths. Flutter supports two main types of routing:

1. Basic Routing (Navigator Class) – Used for simple navigation between screens using push and pop methods.
2. Named Routing – Predefined routes that make navigation structured and manageable.

Gestures:Gestures allow users to interact with the app through touch-based inputs like tapping, swiping, dragging, and pinching. Flutter's GestureDetector widget helps in detecting and responding to these interactions effectively.

4. Navigation in Flutter:Flutter provides a built-in navigation system using the Navigator class. Navigation can be implemented using:

- Navigator.push() – Moves to a new screen.
- Navigator.pop() – Returns to the previous screen.
- Navigator.pushNamed() – Uses named routes for structured navigation.

5. Implementing Routing in Flutter

Flutter supports two routing mechanisms:

1. Direct Navigation – Using Navigator.push() for simple screen transitions.
2. Named Routes – Defined in the MaterialApp widget, allowing better organization and scalability.

Using named routes is beneficial for large applications with multiple screens, making navigation management more efficient.

6. Gesture Handling in Flutter

Flutter provides the GestureDetector widget to handle user interactions. Common gestures include:

- Tap Gesture – Used for button clicks or selecting UI elements.
- Swipe Gesture – Enables horizontal or vertical movement within the app.
- Long Press – Triggers additional actions on prolonged touch.
- Drag and Drop – Allows moving objects within the app.

Code:

Downloads_bloc.dart:

```
import 'dart:developer';
import 'package:bloc/bloc.dart';
import 'package:dartz/dartz.dart';
import 'package:injectable/injectable.dart';
import 'package:meta/meta.dart';
import 'package:freezed_annotation/freezed_annotation.dart';
import 'package:netflix_clone/domain/downloads/i_downloads_repos.dart';
import '../../domain/core/failures/main_failure.dart';
import '../../domain/downloads/models/downloads.dart';
part 'downloads_event.dart';
part 'downloads_state.dart';
part 'downloads_bloc.freezed.dart';
@injectable
class DownloadsBloc extends Bloc<DownloadsEvent, DownloadsState> {
  final IDownloadsRepo _downloadsRepo;
  DownloadsBloc(this._downloadsRepo) : super(DownloadsState.inital()) {
    on<_GetDownloadsImage>((event, emit) async {
      emit(
        state.copyWith(
          isLoading: true,
          downloadsFailureSucessOption: none(),
        ),
      );
      final Either<MainFailure, List<Downloads>> downloadsOption =
          await _downloadsRepo.getDownloadsImage();
      // log(downloadsOption.toString());
      emit(
        downloadsOption.fold(
          (failure) => state.copyWith(
            isLoading: false,
            downloadsFailureSucessOption: Some(
              Left(failure),
            ),
```

```
        ),
        (sucess) => state.copyWith(
          isLoading: false,
          downloads: sucess,
          downloadsFailureSucessOption: Some(
            Right(sucess),
          ),
        ),
      ),
    );
  });
 }
}
```

Description_bloc.dart:

```
import 'dart:developer';
import 'package:bloc/bloc.dart';
import 'package:freezed_annotation/freezed_annotation.dart';
import 'package:injectable/injectable.dart';
import 'package:netflix_clone/domain/description/description_service.dart';
import 'package:netflix_clone/domain/description/model/description_resp/description_resp.dart';
import '../../domain/core/failures/main_failure.dart';
import '../../domain/new_and_hot/model/discover.dart';
part 'description_event.dart';
part 'description_state.dart';
part 'description_bloc.freezed.dart';

@injectable
class DescriptionBloc extends Bloc<DescriptionEvent, DescriptionState> {
  final DescriptionService _descriptionService;
  DescriptionBloc(this._descriptionService)
      : super(DescriptionState.initial()) {
    on<LoadDataMovie>((event, emit) async {
      emit(
        const DescriptionState(
          title: "",
          posterPath: "",
          overview: "",
          status: "",
          relaseDate: "",
          voteAverage: 0,
          isLoading: true,
          isError: false,
        ),
      );
```

```
    final movies =
        await _descriptionService.getMovieDescription(id: event.id);
    final result = movies.fold(
      (l) => const DescriptionState(
        title: "",
        posterPath: "",
        overview: "",
        status: "",
        relaseDate: "",
        voteAverage: 0,
        isLoading: true,
        isError: false,
      ),
      (r) => DescriptionState(
        title: r.title!,
        posterPath: r.posterPath!,
        overview: r.overview!,
        status: r.status!,
        relaseDate: r.releaseDate!,
        voteAverage: r.voteAverage!,
        isLoading: false,
        isError: false,
      ),
    );
    emit(result);
  });
 }
}
```

Fast_laugh_bloc.dart:

```
import 'dart:developer';
import 'package:bloc/bloc.dart';
import 'package:flutter/material.dart';
import 'package:freezed_annotation/freezed_annotation.dart';
import 'package:injectable/injectable.dart';
import 'package:netflix_clone/domain/core/failures/main_failure.dart';
import 'package:netflix_clone/domain/downloads/i_downloads_repos.dart';
import '../../domain/downloads/models/downloads.dart';
part 'fast_laugh_event.dart';
part 'fast_laugh_state.dart';
part 'fast_laugh_bloc.freezed.dart';
final dummyVideoUrls = [

"https://pagalstatus.com/wp-content/uploads/2021/10/Game-Of-Thrones-Dragon-Scene-Status-Video.mp4
```

```
",
  "https://statuspaji.com/wp-content/uploads/2021/09/Iron-Man-Full-Screen-Status-Video.mp4",
  "https://statusguide.com/anykreeg/2021/04/Naruto-vs-pain-amv-whatsapp-status.mp4",

"https://gostatusguru.com/siteuploads/files/sfd25/12133/Stranger%20Things%20Whatsapp%20Status(Go
StatusGuru.Com).mp4",

"https://statusour.com/wp-content/uploads/2021/09/Money-Heist-Whatsapp-Status-Video-Download-Full
-Screen-4k-Status-9.mp4"
];
ValueNotifier<Set<int>> likedVideosIdNotifier = ValueNotifier({});
@injectable
class FastLaughBloc extends Bloc<FastLaughEvent, FastLaughState> {
  FastLaughBloc(
    IDownloadsRepo _downlodsService,
  ) : super(FastLaughState.initial()) {
    on<Initialize>((event, emit) async {
      emit(
          // sending loading to ui
          const FastLaughState(
        videosList: [],
        isLoading: true,
        isError: false,
      ));
      // get trending movies
      final _result = await _downlodsService.getDownloadsImage();
      // log(_result.toString());
      final respon = _result.fold(
          (l) => FastLaughState(
                videosList: [],
                isError: true,
                isLoading: false,
              ),
          (r) => FastLaughState(
                videosList: r,
                isLoading: false,
                isError: false,
              ));
      // send to ui
      emit(respon);
    });
    on<LikeVideo>((event, emit) async {
      likedVideosIdNotifier.value.add(event.id);
      likedVideosIdNotifier.notifyListeners();
```

```
    });
    on<UnlikeVideo>((event, emit) async {
      likedVideosIdNotifier.value.remove(event.id);
      likedVideosIdNotifier.notifyListeners();
    });
  }
}
```

Hot_and_new_bloc.dart:

```
import 'package:bloc/bloc.dart';
import 'package:freezed_annotation/freezed_annotation.dart';
import 'package:injectable/injectable.dart';
import 'package:netflix_clone/domain/new_and_hot/hot_and_new_service.dart';
import 'package:netflix_clone/domain/new_and_hot/model/discover.dart';
import '../../domain/core/failures/main_failure.dart';
part 'hot_and_new_event.dart';
part 'hot_and_new_state.dart';
part 'hot_and_new_bloc.freezed.dart';
@injectable
class HotAndNewBloc extends Bloc<HotAndNewEvent, HotAndNewState> {
  final HotAndNewService _hotAndNewService;
  HotAndNewBloc(this._hotAndNewService) : super(HotAndNewState.initial()) {
    // get hot and new movie data
    on<LoadDataInComingSoon>((event, emit) async {
      // send loading to ui
      emit(const HotAndNewState(
          comingSoonList: [],
          everyOneIsWatchingList: [],
          isLoading: true,
          hasError: false));
      // get data from remote
      final _result = await _hotAndNewService.getHotAndNewMovieData();
      // data in state
      final newState = _result.fold((MainFailure f) {
        return const HotAndNewState(
            comingSoonList: [],
            everyOneIsWatchingList: [],
            isLoading: false,
            hasError: true);
      }, (HotAndNewDataResp resp) {
        return HotAndNewState(
            comingSoonList: resp.results,
            everyOneIsWatchingList: state.everyOneIsWatchingList,
            isLoading: false,
```

```
          hasError: false);
    });
    emit(newState);
  });
  // get hot and new tv data
  on<LoadDataInEveryOnesWatching>((event, emit) async {
    emit(const HotAndNewState(
        comingSoonList: [],
        everyOneIsWatchingList: [],
        isLoading: true,
        hasError: false));
    // get data from remote
    final _result = await _hotAndNewService.getHotAndNewTvData();
    // data in state
    final newState = _result.fold((MainFailure f) {
      return const HotAndNewState(
          comingSoonList: [],
          everyOneIsWatchingList: [],
          isLoading: false,
          hasError: true);
    }, (HotAndNewDataResp resp) {
      return HotAndNewState(
          comingSoonList: state.comingSoonList,
          everyOneIsWatchingList: resp.results,
          isLoading: false,
          hasError: false);
    });
    emit(newState);
  });
 }
}
```

Search_bloc.dart:

```
import 'package:bloc/bloc.dart';
import 'package:freezed_annotation/freezed_annotation.dart';
import 'package:injectable/injectable.dart';
import 'package:netflix_clone/domain/core/failures/main_failure.dart';
import 'package:netflix_clone/domain/downloads/i_downloads_repos.dart';
import 'package:netflix_clone/domain/search/search_service.dart';

import '../../domain/downloads/models/downloads.dart';
import '../../domain/search/model/search_resp/search_resp.dart';

part 'search_event.dart';
```

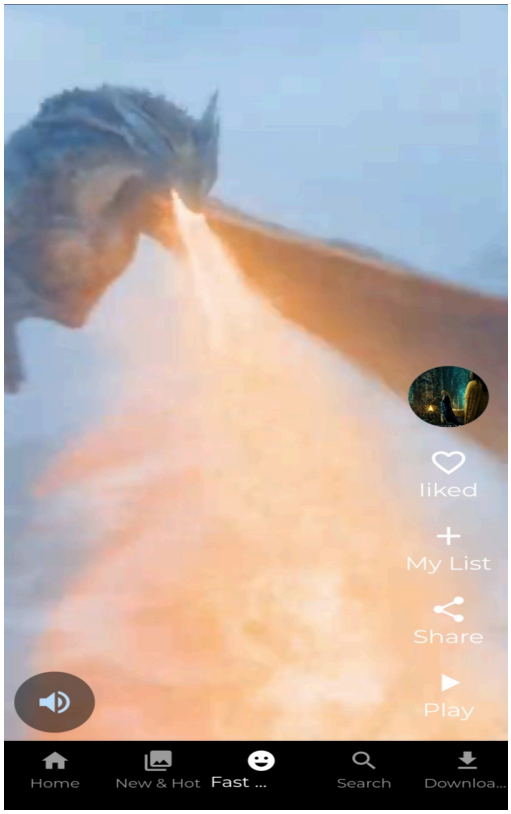
```
part 'search_state.dart';
part 'search_bloc.freezed.dart';

@injectable
class SearchBloc extends Bloc<SearchEvent, SearchState> {
  final IDownloadsRepo _downloadsService;
  final SearchService _searchService;
  SearchBloc(this._searchService, this._downloadsService)
      : super(SearchState.initial()) {
    // idel state
    on<Initalize>((event, emit) async {
      if (state.ideleList.isNotEmpty) {
        emit(SearchState(
          searchResultList: [],
          ideleList: state.ideleList,
          isLoading: false,
          isError: false,
        ));
        return;
      }
      emit(
        const SearchState(
          searchResultList: [],
          ideleList: [],
          isLoading: true,
          isError: false,
        ),
      );
      // get trending
      final _result = await _downloadsService.getDownloadsImage();
      final _state = _result.fold((MainFailure f) {
        return const SearchState(
          searchResultList: [],
          ideleList: [],
          isLoading: false,
          isError: true,
        );
      }, (List<Downloads> list) {
        return SearchState(
          searchResultList: [],
          ideleList: list,
          isLoading: false,
          isError: false,
        );
```

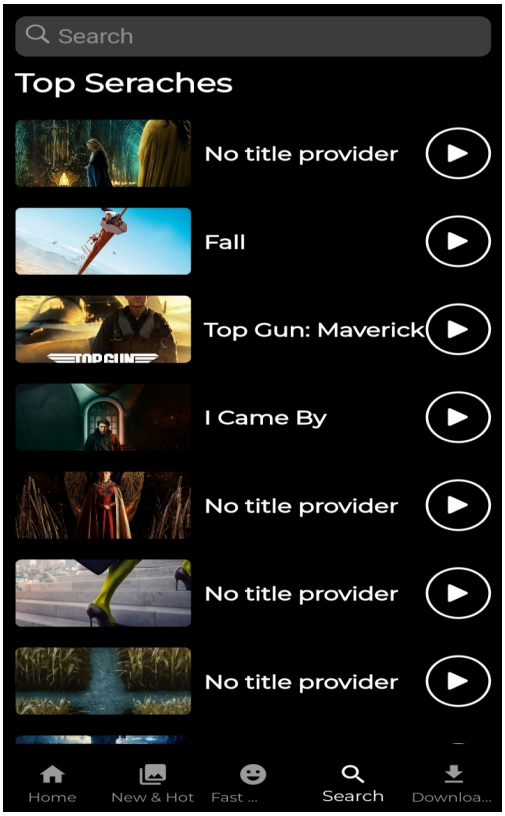
```
    });
    emit(_state);
    // show to ui
   });
   // serch result state
   on<SerachMovie>((event, emit) async {
    // call search movie api
    emit(
     const SearchState(
      searchResultList: [],
      ideleList: [],
      isLoading: true,
      isError: false,
     ),
    );
    final _result =
      await _searchService.searchMovies(movieQuery: event.movieQuery);
    final _state = _result.fold((MainFailure f) {
     return const SearchState(
      searchResultList: [],
      ideleList: [],
      isLoading: false,
      isError: true,
     );
    }, (SearchResp r) {
     return SearchState(
      searchResultList: r.results,
      ideleList: [],
      isLoading: false,
      isError: false,
     );
    });
    // show to ui
    emit(_state);
   });
  }
}
```

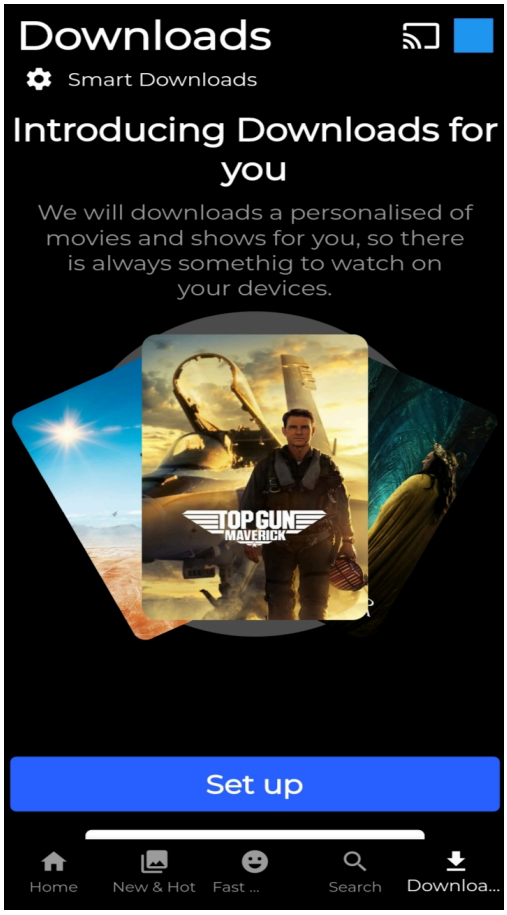
OUTPUT:

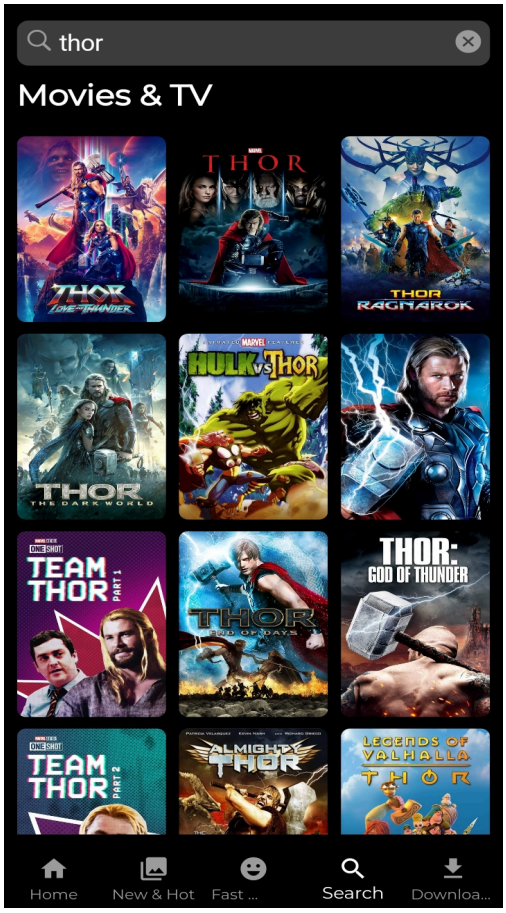

TV Shows
Movies
Categories
HOUSE OF THE DRAGON
My List
Play
More
Released in the past year
Home
New & Hot
Fast ...
Search
Downloa...


New & Hot
Coming Soon
Everyone's Watching
JUN
06
ドラゴンボ･･･
Remind me
Info
coming on Friday
ドラゴンボール超 スーパーヒーロー
The Red Ribbon Army, an evil organization that was once destroyed by Goku in the past, has been ...
JUL
07
Home
New & Hot
Fast ...
Search
Downloa...


liked
My List
Share
Play
Home
New & Hot
Fast ...
Search
Downloa...


Search
Top Seraches
No title provider
Fall
Top Gun: Maverick
I Came By
No title provider
No title provider
No title provider
Home
New & Hot
Fast ...
Search
Downloa...





Conclusion:

Navigation, routing, and gestures are fundamental in creating a dynamic and interactive Flutter application. Proper navigation ensures users can move seamlessly between screens, routing organizes app structure efficiently, and gestures provide a smooth and intuitive user experience. By implementing these features effectively, developers can create well-structured, responsive, and user-friendly Flutter applications.