Experiment 5

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

Theory:

Flutter offers a comprehensive approach to handle deep links and screen transitions. While the Navigator is suitable for smaller apps with basic deep linking, the Router is recommended for larger apps requiring intricate deep linking. This ensures proper deep link handling across Android, iOS, and web platforms.

Any mobile app's workflow is determined by the user's ability to navigate to various pages; this process is known as routing. The MaterialPageRoute class of Flutter's routing system comes with the two methods that demonstrate how to switch between two routes and they are:

1. Navigator.push() — To navigate to a new route.
2. Navigator.pop() — To return to the previous route.

The Navigator widget manages routes using a stack approach. Routes are stacked based on user actions, and pressing back navigates to the most recent route.

Working of the Navigator:
These following steps show how all the components work together as one piece:

1. The RouteInformationParser converts a new route into a user-defined data type.
2. The RouterDelegate method updates the app state and calls notifyListeners.
3. The Router instructs the RouterDelegate to rebuild via its build() method.
4. The RouterDelegate.build() returns a new Navigator reflecting the updated app state.

Steps to Start Navigation in App:

There are three steps required to start navigation in the app and they are:

1. Create Two Routes: Define the initial and destination routes.
2. Use Navigator.push(): Transition to the new route. To navigate or transition to a new page, route, or screen, the Navigator.push() method is used. The push() method adds a page or route to the stack, and then uses the Navigator to manage it. Once more, we use the MaterialPageRoute class, which enables platform-specific animations for route transitions.
3. Use Navigator.pop(): Navigate back to the initial route. The Navigator controls the stack, and the pop() method enables to remove the current route from it.

The Router and Navigator are designed to work in tandem. By executing imperative methods like push() and pop() on the Navigator, or declarative routing packages like go_router, Router API can ne navigated. Each route on the Navigator is page-backed, which means it was

made from a Page using the pages option on the Navigator constructor, when you navigate using the Router or a declarative routing package. On the other hand, any Route added to the Navigator by calling Navigator.push or showDialog will add a pageless route. Page-backed routes, as opposed to pageless ones, can always be deep-linked when utilising a routing package.

All subsequent pageless routes are also deleted when a page-backed route is deleted from the navigator. For instance, if a deep link navigates by eliminating a page-backed route from the navigator, any pageless _routes that follow up to the following page-backed route are also eliminated.

For a consistent experience when utilising the browser's back and forward buttons, apps that use the Router class integrate with the History API. A History API entry is added to the browser's history stack each time you use the Router to traverse. When the back button is pressed, the Router switches to reverse chronological navigation, which transports the user back to the last

Gestures are used to interact with an application. It is generally used in touch-based devices to physically interact with the application. It can be as simple as a single tap on the screen to a more complex physical interaction like swiping in a specific direction to scrolling down an application. It is heavily used in gaming and more or less every application requires it to function as devices turn more touch-based than ever. In this article, we will discuss them in detail.

Some widely used gestures are mentioned here :

- Tap: Touching the surface of the device with the fingertip for a small duration of time period and finally releasing the fingertip.
- Double Tap: Tapping twice in a short time.
- Drag: Touching the surface of the device with the fingertip and then moving the fingertip in a steadily and finally releasing the fingertip.
- Flick: Similar to dragging, but doing it in a speedier way.
- Pinch: Pinching the surface of the device using two fingers.
- Zoom: Opposite of pinching.
- Panning: Touching the device surface with the fingertip and moving it in the desired direction without releasing the fingertip.

The GestureDetector widget in flutter is used to detect physical interaction with the application on the UI. If a widget is supposed to experience a gesture, it is kept inside the GestureDetector widget. The same widget catches the gesture and returns the appropriate action or response.

Code in main.dart:

```
import 'package:flutter/material.dart';
import 'package:flutter/services.dart';
import 'package:flutter_riverpod/flutter_riverpod.dart';
import 'package:flutter_youtube_ui/screens/nav_screen.dart';

void main() {
 runApp(ProviderScope(child: MyApp()));
}

class MyApp extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
   SystemChrome.setPreferredOrientations([DeviceOrientation.portraitUp]);
   return MaterialApp(
     title: 'Flutter YouTube UI',
     debugShowCheckedModeBanner: false,
     theme: ThemeData(
       brightness: Brightness.dark,
       bottomNavigationBarTheme:
           const BottomNavigationBarThemeData(selectedItemColor: Colors.white),
     ),
     home: NavScreen(),
   );
 }
}
```

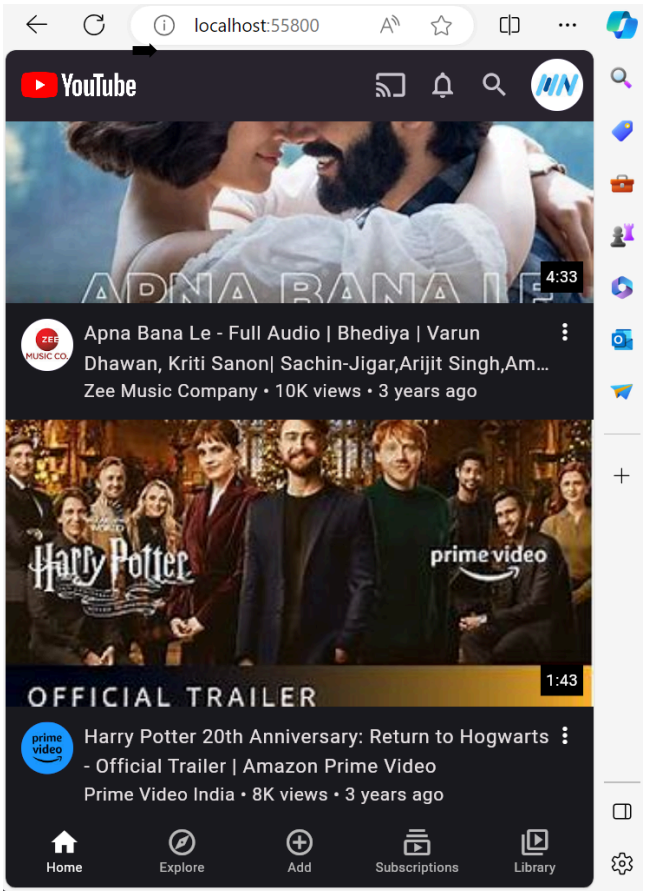
The code in home_screen.dart:

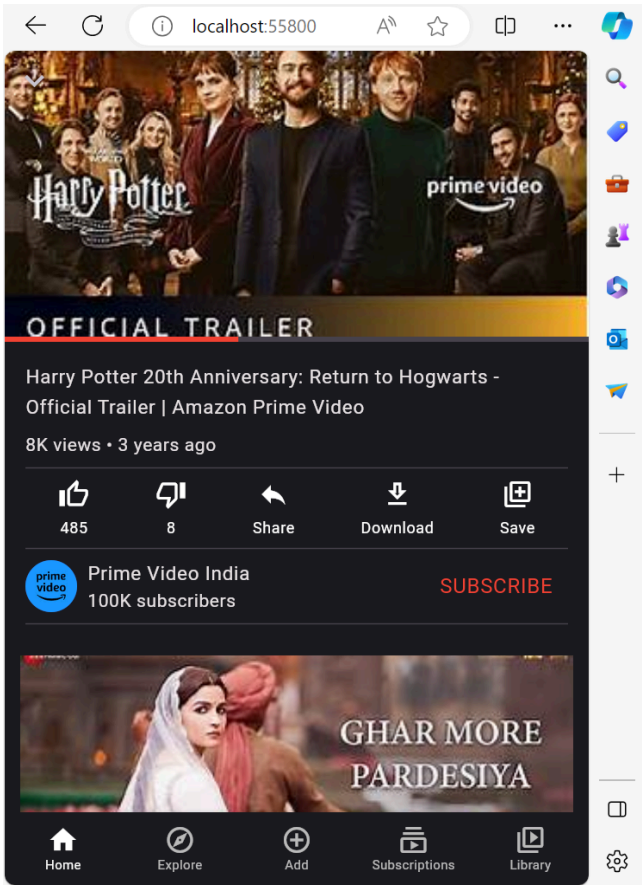
```
import 'package:flutter/material.dart';
import 'package:flutter_youtube_ui/data.dart';
import 'package:flutter_youtube_ui/widgets/widgets.dart';

class HomeScreen extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
   return Scaffold(
     body: CustomScrollView(
       slivers: [
         CustomSliverAppBar(),
         SliverPadding(
           padding: const EdgeInsets.only(bottom: 60.0),
           sliver: SliverList(
             delegate: SliverChildBuilderDelegate(
               (context, index) {
```

```
            final video = videos[index];
            return VideoCard(video: video);
          },
          childCount: videos.length,
        ),
      ),
    ),
  ],
 ),
 );
 }
}
```

Output:





(Here, clicking on the second video opens up the video player for that video i.e. the second video in this case)