EXPERIMENT NO 6

Aim: To apply navigation, routing and gesture in flutter

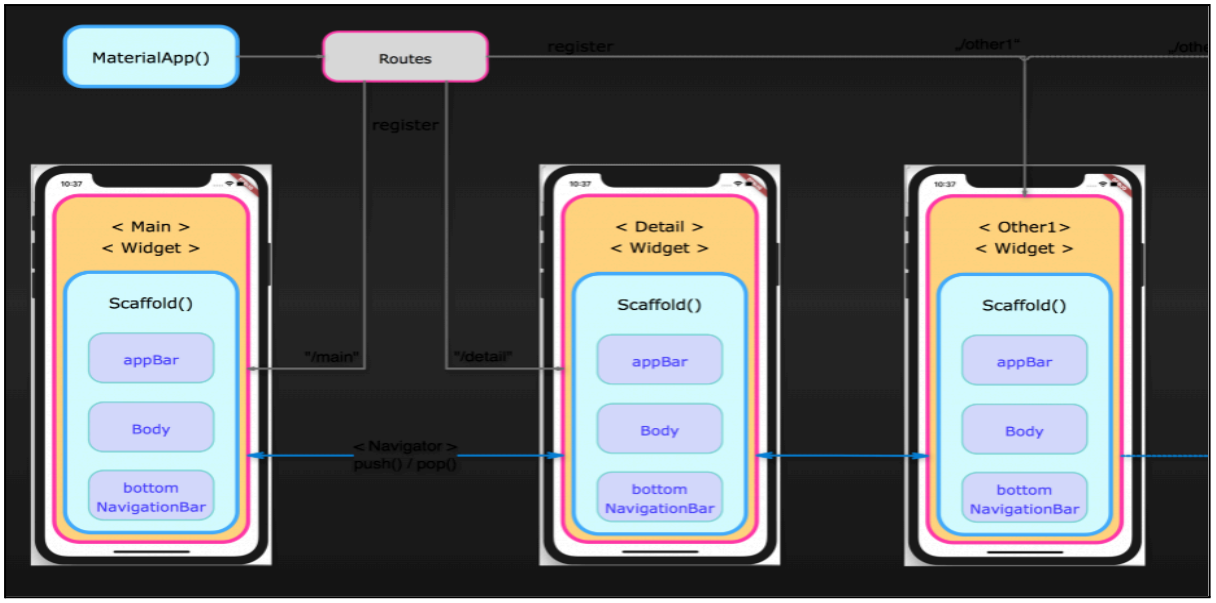
Theory:

Flutter is an open-source UI framework developed by Google used to build fast, natively compiled applications for mobile, web, and desktop from a single codebase. It uses the Dart programming language and follows a widget-based architecture, where everything in the UI is a widget, making layouts highly customizable and flexible.

Flutter renders its own UI using the Skia graphics engine, which ensures high performance and a consistent look across platforms. Features like hot reload allow developers to see changes instantly, improving productivity, while rich built-in widgets and strong community support make Flutter a popular choice for modern app development.

Navigation in Flutter is used to move between different screens (pages) in an application. Flutter follows a stack-based navigation system, where each new screen is pushed onto the navigation stack and removed when the user goes back. The Navigator widget manages this stack and helps in screen transitions. Proper navigation improves user experience by allowing smooth movement across different sections of an app, such as from a login page to a dashboard or from a product list to a details page.

Routing in Flutter defines how an application moves between screens using predefined paths or routes. Routes can be handled in two main ways: named routes and direct routes. Named routing allows developers to define all routes in one place, making large applications easier to manage and scale. Routing helps maintain clean code structure, improves readability, and supports deep linking and organized navigation flow within the app.



Gestures in Flutter allow the application to respond to user interactions such as tapping, double-tapping, long pressing, swiping, and dragging. Flutter provides widgets like GestureDetector and InkWell to detect and handle these touch-based interactions. Gestures make applications more interactive and intuitive, enabling users to interact naturally with UI elements instead of relying only on buttons.

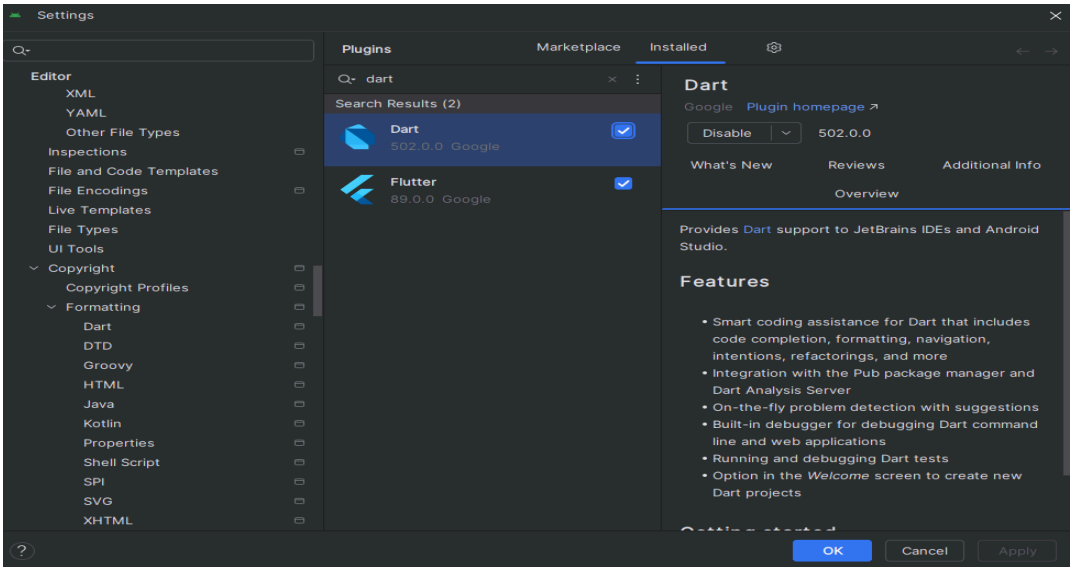
Navigation, routing, and gestures work together to create a smooth and responsive user experience. For example, a user may tap on a card (gesture) to navigate to a detailed screen (navigation) using a defined route (routing). Combining these concepts allows developers to build feature-rich applications where user actions directly control screen transitions and app behavior.

In real-world Flutter applications such as e-commerce apps, social media platforms, and dashboards, navigation and routing ensure structured screen flow, while gestures enhance interactivity and usability. Together, they make applications more dynamic, user-friendly, and professional. Understanding these concepts is essential for building scalable Flutter apps and is commonly tested in practical exams, interviews, and project evaluations.

Procedures:

STEP 1: Install Flutter & Dart Plugins in Android Studio

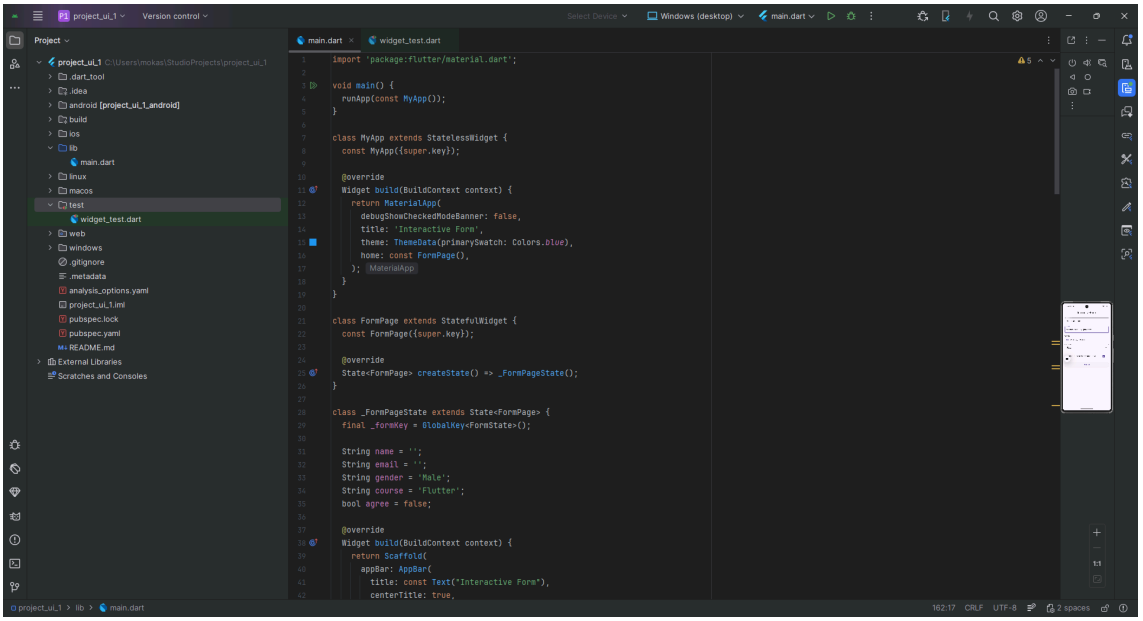
1. Open Android Studio
2. Go to Settings → Plugins
3. Search and Install- Flutter & Dart
4. Restart android studio



STEP 2: Create a New Flutter Project

1. Open Android Studio
2. Click New Flutter Project
3. Select Flutter Application
4. Choose Flutter SDK path (example):

   C:\flutter
5. Enter:
   - Project name: project_ui_1
   - Language: Dart
6. Click Finish



STEP 3: Code

```
import 'package:flutter/material.dart';
import 'package:fl_chart/fl_chart.dart';
void main() {
  runApp(const MyApp());
}
class MyApp extends StatelessWidget {
  const MyApp({super.key});
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      home: const AdvancedLineChartPage(),
    );
  }
}

class AdvancedLineChartPage extends StatelessWidget {
  const AdvancedLineChartPage({super.key});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: const Color(0xfff4f6fb),
      appBar: AppBar(
        title: const Text("Advanced Line Chart"),
        centerTitle: true,
      ),
      body: Padding(
```

```
padding: const EdgeInsets.all(16),
child: Card(
  elevation: 6,
  shape: RoundedRectangleBorder(
    borderRadius: BorderRadius.circular(20),
  ),
  child: Padding(
    padding: const EdgeInsets.all(16),
    child: Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
        Row(
          children: const [
            Icon(Icons.show_chart, color: Colors.blue),
            SizedBox(width: 8),
            Text(
              "Sales Analytics",
              style:
                  TextStyle(fontSize: 18, fontWeight: FontWeight.bold),
            ),
          ],
        ),
        const SizedBox(height: 20),
        SizedBox(
          height: 250,
          child: LineChart(_lineChartData()),
        ),
      ],
    ),
```

```
        ),
      ),
    ),
  );
}
/// ADVANCED LINE CHART DATA
LineChartData _lineChartData() {
  return LineChartData(
    gridData: FlGridData(
      show: true,
      drawVerticalLine: false,
      horizontalInterval: 1,
    ),
    borderData: FlBorderData(show: false),

    ///  Axis Titles
    titlesData: FlTitlesData(
      leftTitles: AxisTitles(
        sideTitles: SideTitles(
          showTitles: true,
          interval: 1,
          reservedSize: 40,
        ),
      ),
      bottomTitles: AxisTitles(
        sideTitles: SideTitles(
          showTitles: true,
          interval: 1,
          getTitlesWidget: (value, meta) {
```

```
        const labels = ['Mon', 'Tue', 'Wed', 'Thu', 'Fri'];
        return Padding(
          padding: const EdgeInsets.only(top: 8),
          child: Text(
            labels[value.toInt()],
            style: const TextStyle(fontSize: 12),
          ),
        );
      },
    ),
  ),
  topTitles: AxisTitles(sideTitles: SideTitles(showTitles: false)),
  rightTitles: AxisTitles(sideTitles: SideTitles(showTitles: false)),
),

///  Touch Interaction
lineTouchData: LineTouchData(
  enabled: true,
  touchTooltipData: LineTouchTooltipData(
    tooltipBgColor: Colors.blue,
    getTooltipItems: (spots) {
      return spots.map((spot) {
        return LineTooltipItem(
          "Value: ${spot.y}",
          const TextStyle(color: Colors.white),
        );
      }).toList();
    },
  ),
```

```
),
///  Line Data
lineBarsData: [
  LineChartBarData(
    isCurved: true,
    barWidth: 4,
    dotData: FlDotData(show: true),
    spots: [
      FlSpot(0, 2),
      FlSpot(1, 3.5),
      FlSpot(2, 2.8),
      FlSpot(3, 4.5),
      FlSpot(4, 3.8),
    ],
    gradient: const LinearGradient(
      colors: [Colors.blue, Colors.purple],
    ),
    ///  Area below line
    belowBarData: BarAreaData(
      show: true,
      gradient: LinearGradient(
        colors: [
          Colors.blue.withOpacity(0.3),
          Colors.purple.withOpacity(0.1),
        ],
      ),
    ),
  ),
],
```

```
    );
  }
}
```

STEP 6: Output

Conclusion:

The advanced line chart implementation in Flutter demonstrates how complex data can be presented in a clear, interactive, and visually appealing manner. By using Flutter's flexible widget system along with third-party libraries such as fl_chart, developers can create smooth animations, gradient effects, tooltips, and well-labeled axes to enhance data understanding. This approach improves user experience and makes applications suitable for analytics dashboards, business reports, and data-driven systems. Overall, advanced charts in Flutter help transform raw data into meaningful insights while maintaining a professional and modern application interface.