Name - Tejas Shailendra Rokade
Div - D15A Roll no - 50
Batch - C

Experiment - 5

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

Theory -

Navigation and Routing in Flutter:
Flutter offers two fundamental approaches to manage navigation and routing:
1. Navigator:
   - The built-in Navigator widget provides imperative methods for pushing, popping, and replacing routes.
   - Use Navigator.push() to move forward in the navigation stack, Navigator.pop() to go back, and Navigator.popUntil() to clear the stack up to a certain point.
   - Suitable for small to medium-sized apps with straightforward navigation structures.
2. Router Packages:
   - Packages like go_router and routemaster offer declarative routing paradigms, handling deep linking and more complex navigation scenarios.
   - Define routes as data structures, and the router handles navigation seamlessly.
   - Useful for larger apps or ones with dynamic navigation requirements.

Choosing the Right Approach:
Consider these factors when selecting a navigation system:
   - App size and complexity
   - Deep linking needs
   - Preference for imperative or declarative routing styles

Example with Navigator:
Dart

```
// First screen (home)
ElevatedButton(
  onPressed: () => Navigator.push(
    context,
    MaterialPageRoute(builder: (context) => SecondScreen()),
  ),
  child: const Text('Go to Second Screen'),
),

// Second screen
ElevatedButton(
  onPressed: () => Navigator.pop(context),
```

```
  child: const Text('Go Back'),
),
```

Use code with caution. Learn more
content_copy
Example with go_router:
Dart
```
// main.dart
final myRouter = GoRouter(
  initialLocation: '/',
  routes: [
    GoRoute(
      path: '/',
      builder: (context, state) => Home(),
      routes: [
        GoRoute(
          path: '/second',
          builder: (context, state) => SecondScreen(),
        ),
      ],
    ),
  ],
);

// Second screen
ElevatedButton(
  onPressed: () => context.go('/'),
  child: const Text('Go Back'),
),
```

Use code with caution. Learn more
content_copy
Gestures in Flutter:
Flutter's rich gesture system allows you to create interactive and intuitive experiences. Here are some key types:

- Taps: Detect single, double, or long taps using GestureDetector or InkWell.
- Drags: Handle drags, swipes, and pan gestures using RawGestureDetector or GestureDetector.
- Scrolls: Use ListView, GridView, or custom scrollable widgets.
- Other gestures: Use packages like flutter_slidable for swipe/delete actions, pinch_zoom for pinch-to-zoom, and more.

Tips for Effective Gestures:

- Provide visual feedback (e.g., animations, sound effects) on gesture interaction.
- Test gestures on different devices and screen sizes.

- Prioritize accessibility by considering users with different abilities.

Example with Taps:

Dart

```
GestureDetector(
  onTap: () => print('Tapped!'),
  child: Container(
    color: Colors.blue,
    child: const Text('Tap me'),
  ),
),
```

Use code with caution. Learn more

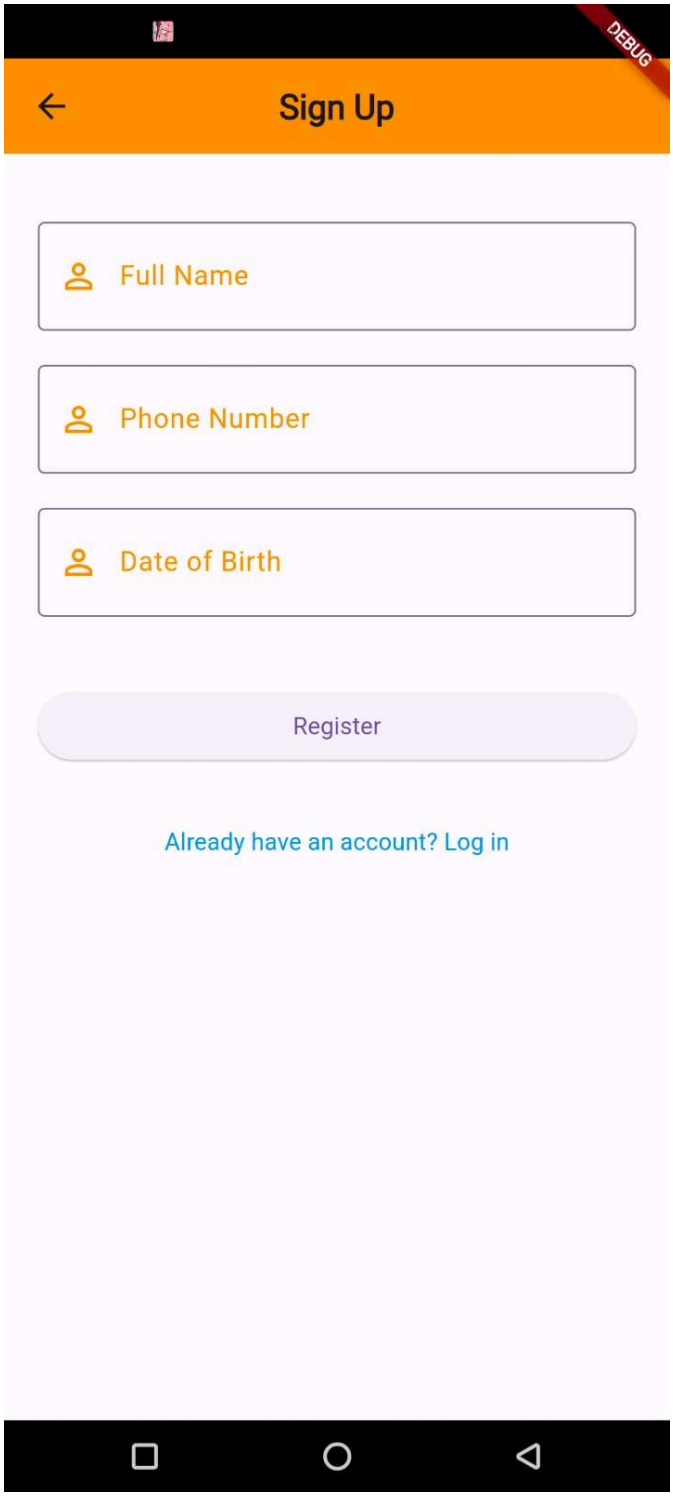
content_copy

Example with Drags:

Dart

```
RawGestureDetector(
  onPanDown: (details) => print('Drag started'),
  onPanUpdate: (details) => print('Dragging (dx: ${details.delta.dx}, dy: ${details.delta.dy})'),
  onPanEnd: (details) => print('Drag ended'),
  child: Container(
    color: Colors.green,
    child: const Text('Drag me'),
  ),
),
```

Screenshot -


DEBUG
Sign Up
Full Name
Phone Number
Date of Birth
Register
Already have an account? Log in

Lunching

The best of best

E-mail

Password

Forget Password?

LOG IN

OR

Don't have an account?Register

Sign Up

Conclusion -

Through this experiment we understood implementation of navigator , haptics and gestures for implementing navigation from one tab to another tab as per need and gestures to recognise and for events to occur in the flutter application.