EXPERIMENT 5

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

Theory

1. Navigation and Routing

In Flutter, navigation refers to moving from one screen (or "route") to another. There are several key concepts:

- **Routes**: These are the different screens or pages in your app. Every route is typically represented by a Widget in Flutter. The default route is usually the home screen of the app, but you can define multiple routes for different screens.

- **Navigator**: This is a widget that manages a stack of routes. You can "push" a new route onto the stack to navigate to another screen, or "pop" the top route off to go back.

- **Named Routes**: These are routes that are identified by a string. Instead of pushing or popping routes directly, you can refer to routes by their name (e.g., /home, /settings).

- **Custom Route Transitions**: Flutter allows you to define custom animations and transitions when navigating between routes. You can create smooth, custom page transitions using PageRouteBuilder.

- **Route Arguments**: You can pass data between routes using arguments. This is particularly useful when navigating to a screen that requires specific data (e.g., opening a product page with product details).

2. Gestures in Flutter

Gestures are interactions that a user performs with the screen, such as taps, swipes, or long presses. Flutter provides a flexible way to detect these gestures.

- **GestureDetector**: This is the most commonly used widget for detecting gestures. You can wrap it around any widget to detect gestures like tap, double tap, long press, swipe, and others.

- **Tap Gesture**: A simple touch interaction, typically detected using onTap or onLongPress callbacks.

- **Swipe Gestures**: Swiping is usually detected via onHorizontalDragUpdate, onVerticalDragUpdate, or onPanUpdate. These allow you to track the user's finger movement and respond accordingly.

- **Custom Gesture Detection**: Flutter also allows you to implement more complex gestures. For example, you can detect drag gestures to create features like a sliding menu or draggable elements.

- **Dismissible Widget**: This widget enables swipe-to-dismiss behavior, commonly used for items in a list that users can swipe left or right to remove.

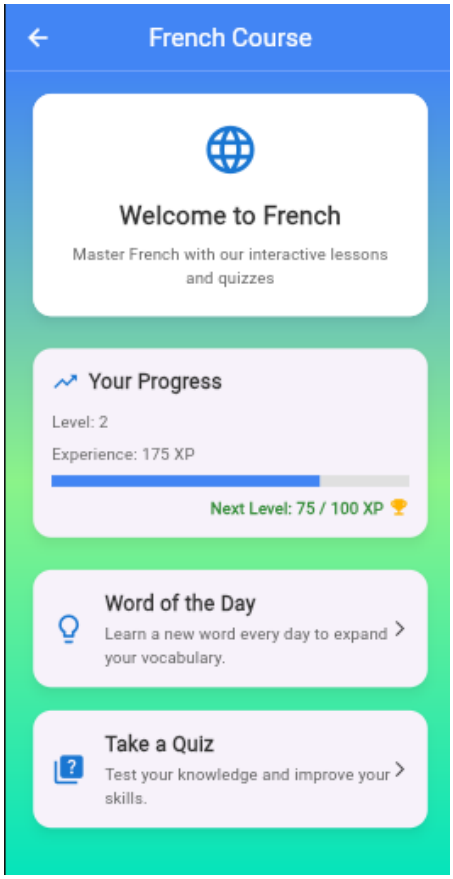
3. Managing Navigation and Gestures Together

When you combine navigation with gestures, you can create more interactive and dynamic UIs. For instance, a user could swipe to navigate between screens, or tap a button that triggers navigation while performing a gesture on a different part of the screen.

4. Back Button Handling

On Android devices, there is a system-wide back button that users can press to navigate backward. Flutter provides a way to intercept and customize this behavior using WillPopScope, allowing you to decide what happens when the user tries to go back (e.g., prevent the user from leaving the current screen, show a confirmation dialog, or allow normal back navigation).

Screenshots:



Code:

Gesture Detector

```
child: GestureDetector(
  onTap: () => Navigator.push(
    context,
    MaterialPageRoute(builder: (context) => SearchPage()),
  ),
  child: SearchBar(),
),
```

Using Navigator.push

```
Navigator.pushReplacement(
```

```
  context,
  MaterialPageRoute(builder: (context) => HomeScreen()),
);


Navigator.push(
  context,
  MaterialPageRoute(builder: (context) => const LoginPage()),
);




import 'package:flutter/material.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:cached_network_image/cached_network_image.dart';
import 'book_detail_page.dart';

class BrowsePage extends StatefulWidget {
  const BrowsePage({super.key});

  @override
  _BrowsePageState createState() => _BrowsePageState();
}

class _BrowsePageState extends State<BrowsePage> {
  int _selectedIndex = 1;
  String searchQuery = "";
  String? selectedGenre;

  final List<String> genres = [
    "Fiction", "Mystery", "Fantasy", "Sci-Fi", "Romance", "Non-Fiction"
  ];

  void _onItemTapped(int index) {
    switch (index) {
      case 0:
        Navigator.pushNamed(context, '/dashboard');
        break;
      case 1:
```

```
        break;
      case 2:
        Navigator.pushNamed(context, '/events');
        break;
      case 3:
        Navigator.pushNamed(context, '/profile');
        break;
    }
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text("Browse Books", style: TextStyle(fontSize: 22, fontWeight:
FontWeight.bold)),
        backgroundColor: const Color(0xFF04511A),
      ),
      body: Padding(
        padding: const EdgeInsets.all(16),
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            // Search Bar
            TextField(
              onChanged: (value) {
                setState(() {
                  searchQuery = value.toLowerCase();
                });
              },
              decoration: InputDecoration(
                hintText: "Search books...",
                prefixIcon: const Icon(Icons.search),
                border: OutlineInputBorder(borderRadius: BorderRadius.circular(8)),
              ),
            ),
            const SizedBox(height: 16),

            // Genre Filter
```

```
        const Text("Genres", style: TextStyle(fontSize: 18, fontWeight:
FontWeight.bold)),
        const SizedBox(height: 8),
        Wrap(
          spacing: 8,
          children: genres.map((genre) {
            return ChoiceChip(
              label: Text(genre),
              selected: selectedGenre == genre,
              onSelected: (isSelected) {
                setState(() {
                  selectedGenre = isSelected ? genre : null;
                });
              },
            );
          }).toList(),
        ),
        const SizedBox(height: 16),

        // Book List
        const Text("Trending Books", style: TextStyle(fontSize: 18, fontWeight:
FontWeight.bold)),
        const SizedBox(height: 8),
        Expanded(
          child: StreamBuilder(
            stream: FirebaseFirestore.instance.collection('books').snapshots(),
            builder: (context, AsyncSnapshot<QuerySnapshot> snapshot) {
              if (snapshot.connectionState == ConnectionState.waiting) {
                return const Center(child: CircularProgressIndicator());
              }
              if (!snapshot.hasData || snapshot.data!.docs.isEmpty) {
                return const Center(child: Text('No books found.'));
              }

              var books = snapshot.data!.docs.where((doc) {
                var bookData = doc.data() as Map<String, dynamic>;

                // Check if book has a valid cover URL
                if (bookData['coverUrl'] == null || bookData['coverUrl'].toString().isEmpty) {
                  return false;
```

```
  }

  // Check genre filtering
  if (selectedGenre != null) {
    var bookGenre = bookData['genre'];
    if (bookGenre is String) {
      if (bookGenre != selectedGenre) return false;
    } else if (bookGenre is List) {
      if (!bookGenre.contains(selectedGenre)) return false;
    }
  }

  // Search filtering
  if (searchQuery.isNotEmpty) {
    var title = bookData['title']?.toLowerCase() ?? "";
    var author = bookData['author']?.toLowerCase() ?? "";
    if (!title.contains(searchQuery) && !author.contains(searchQuery)) {
      return false;
    }
  }

  return true;
}).toList();

if (books.isEmpty) {
  return const Center(child: Text('No books match your filters.'));
}

return ListView.builder(
  itemCount: books.length,
  itemBuilder: (context, index) {
    var book = books[index];

    return ListTile(
      contentPadding: const EdgeInsets.symmetric(vertical: 8),
      leading: ClipRRect(
        borderRadius: BorderRadius.circular(8),
        child: CachedNetworkImage(
          imageUrl: book['coverUrl'],
          width: 60,
```

```
                    height: 90,
                    fit: BoxFit.cover,
                    placeholder: (context, url) => const Center(child:
CircularProgressIndicator()),
                    errorWidget: (context, url, error) => const Icon(Icons.broken_image,
size: 50),
                  ),
                ),
                title: Text(book['title'], style: const TextStyle(fontWeight:
FontWeight.bold)),
                subtitle: Text(book['author']),
                onTap: () {
                  Navigator.push(
                    context,
                    MaterialPageRoute(
                      builder: (context) => BookDetailPage(
                        bookId: book.id,
                        bookTitle: book['title'],
                        bookCover: book['coverUrl'],
                        bookAuthor: book['author'],
                        bookDescription: book['description'],
                      ),
                    ),
                  );
                },
              );
            },
          );
        },
      ),
    ),
    bottomNavigationBar: BottomNavigationBar(
      currentIndex: _selectedIndex,
      onTap: _onItemTapped,
      backgroundColor: const Color(0xFF04511A),
      selectedItemColor: Colors.white,
      unselectedItemColor: Colors.white70,
```

```
        type: BottomNavigationBarType.fixed,
        items: const [
          BottomNavigationBarItem(icon: Icon(Icons.dashboard), label: 'Home'),
          BottomNavigationBarItem(icon: Icon(Icons.search), label: 'Browse'),
          BottomNavigationBarItem(icon: Icon(Icons.event), label: 'Events'),
          BottomNavigationBarItem(icon: Icon(Icons.person), label: 'Profile'),
        ],
      ),
    );
  }
}
```

Conclusion:

This experiment provided a comprehensive understanding of navigation and routing in Flutter, which is essential for creating multi-screen applications. We learned how to implement smooth transitions between different screens using both direct and named routes. The use of the Navigator widget helped us manage route stacks effectively, improving the overall navigation flow of the app.

Additionally, the integration of gesture detection using widgets like GestureDetector enabled us to build more interactive and responsive user interfaces. By responding to user actions such as taps and swipes, we were able to make the application more dynamic, intuitive, and user-friendly. These features are crucial for enhancing the user experience and building polished, production-ready apps.

Overall, this experiment strengthened our understanding of managing screen navigation and incorporating user gestures, both of which are fundamental for modern mobile app development in Flutter.