# Experiment 5

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

## Theory:

Navigation in Flutter refers to moving between different screens (called routes). Each screen in Flutter is typically represented by a Widget. Navigation helps in organizing content into different views/pages and managing user interaction flow in an app.

**Types of Navigation:**

***Basic Navigation (Push & Pop)***

Used to move to a new screen or go back.
Commonly uses Navigator.push() and Navigator.pop().

Steps:

- Define two or more widgets (screens).
- Use Navigator.push() to navigate to the next screen.
- Use Navigator.pop() to return to the previous screen.

***Named Navigation (Named Routes)***

More scalable and organized.
Routes are registered with names and used by calling those names.

Steps:

- Define routes in the MaterialApp's routes property.
- Assign a string name to each route.
- Use Navigator.pushNamed(context, '/routeName') for navigation.
- Use Navigator.pop(context) to return.

**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.

**Gestures in Flutter**

Gestures are physical interactions users make with the screen, such as taps, drags, swipes, pinches, etc. Flutter provides powerful gesture recognition through widgets and detectors.

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 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.

Flutter provides the GestureDetector widget to detect these gestures on any widget. You wrap any widget with GestureDetector and specify the gesture callbacks (e.g., onTap, onLongPress, etc.). 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.

**Multi-Gestures**

Multi-gestures involve detecting and responding to multiple gestures simultaneously or in combination, such as pinch-to-zoom with two fingers or dragging and scaling together.

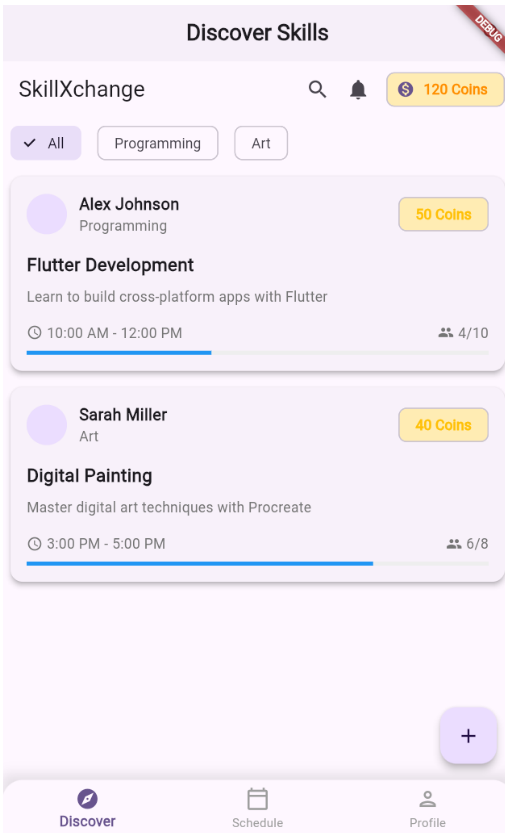
Use Case:
For advanced interaction like image viewers, games, or interactive maps, Flutter provides gesture recognition support through:

- GestureDetector
- RawGestureDetector
- Listener
- InteractiveViewer (for scale, pan, zoom)

**Github Link: https://github.com/brijeshforcollege/flutter**

Code:
```
import 'package:flutter/material.dart';
import 'package:intl/intl.dart';
import 'package:skillxchange/data/mock_data.dart';
import 'package:skillxchange/models/skill_model.dart';
import 'package:skillxchange/screens/post_skill_screen.dart';
import 'package:skillxchange/widgets/skilll_card.dart';
import 'dart:math';

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

  @override
  State<HomeScreen> createState() => _HomeScreenState();
}

class _HomeScreenState extends State<HomeScreen> {
  bool _isLoading = false;
  String _selectedCategory = 'All';

  Future<void> _handleSkillExchange(Skill skill) async {
    final selectedDate = await showDatePicker(
      context: context,
      initialDate: DateTime.now().add(const Duration(days: 1)),
      firstDate: DateTime.now(),
      lastDate: DateTime.now().add(const Duration(days: 365)),
    );
    if (selectedDate == null) return;
    final selectedTime = await showTimePicker(
      context: context,
      initialTime: TimeOfDay.now(),
    );
    if (selectedTime == null) return;

    final confirmed = await showDialog<bool>(
      context: context,
      builder: (context) => AlertDialog(
        title: const Text('Confirm Exchange'),
        content: Column(
          mainAxisSize: MainAxisSize.min,
          children: [
            Text('Exchange skill "${skill.title}" for ${skill.skillCoins} coins?'),
            const SizedBox(height: 10),
            Text('Selected: ${DateFormat('MMM d, y').format(selectedDate)} at
  ${selectedTime.format(context)}'),
          ],
        ),
        actions: [
          TextButton(
            onPressed: () => Navigator.pop(context, false),
            child: const Text('Cancel'),
          ),
          ElevatedButton(
            onPressed: () => Navigator.pop(context, true),
            child: const Text('Exchange'),
          ),
        ],
      ),
    );

    if (confirmed == true) {
      if (currentUser.skillCoins >= skill.skillCoins) {
        setState(() {
          // Update user coins after skill exchange
          currentUser = UserProfile(
            name: currentUser.name,
            email: currentUser.email,
            imageUrl: currentUser.imageUrl,
            skillCoins: currentUser.skillCoins - skill.skillCoins,
            skillsTeaching: currentUser.skillsTeaching,
            skillsLearning: currentUser.skillsLearning,
            bio: currentUser.bio,
            mobile: currentUser.mobile,
          );

          // Add to enrolled skills
          currentUser.skillsLearning.add(skill.title);

          // Update skill participants
          final index = mockSkills.indexWhere((s) => s.id == skill.id);
          if (index != -1) {
            mockSkills[index] = Skill(
              id: skill.id,
              title: skill.title,
              description: skill.description,
              teacherName: skill.teacherName,
              teacherImage: skill.teacherImage,
              timeSlot: skill.timeSlot,
              skillCoins: skill.skillCoins,
              date: skill.date,
              category: skill.category,
              maxParticipants: skill.maxParticipants,
              currentParticipants: skill.currentParticipants + 1,
            );
          }
        });

        ScaffoldMessenger.of(context).showSnackBar(
          SnackBar(
            content: Text('Successfully exchanged for "${skill.title}"!'),
            backgroundColor: Colors.green,
          ),
        );
      } else {
        ScaffoldMessenger.of(context).showSnackBar(
          const SnackBar(
            content: Text('Not enough Skill Coins!'),
            backgroundColor: Colors.red,
          ),
        );
      }
    }
  }

  String _generateMeetLink() {
    const chars = 'abcdefghijklmnopqrstuvwxyz0123456789';
    final random = Random();
    return                'https://meet.google.com/${List.generate(10,               (index)               =>
chars[random.nextInt(chars.length)]).join()}';
  }

  @override
  Widget build(BuildContext context) {
    final categories = ['All', ...mockSkills.map((e) => e.category).toSet().toList()];

    return Scaffold(
      appBar: AppBar(
        title: const Text('SkillXchange'),
        actions: [
          IconButton(
            icon: const Icon(Icons.search),
            onPressed: () {},
          ),
          IconButton(
            icon: const Icon(Icons.notifications),
            onPressed: () {},
          ),
          Padding(
            padding: const EdgeInsets.symmetric(horizontal: 8.0),
            child: Chip(
              backgroundColor: Colors.amber[100],
              label: Text(
          '${currentUser.skillCoins} Coins',
          style: TextStyle(
            color: Colors.amber[800],
            fontWeight: FontWeight.bold,
          ),
        ),
        avatar: const Icon(Icons.monetization_on, size: 18),
      ),
    ),
  ],
),
floatingActionButton: FloatingActionButton(
  onPressed: () {
    Navigator.push(
      context,
      MaterialPageRoute(builder: (context) => const PostSkillScreen()),
    );
  },
  child: const Icon(Icons.add),
),
body: Column(
  children: [
    SizedBox(
      height: 50,
      child: ListView.builder(
        scrollDirection: Axis.horizontal,
        itemCount: categories.length,
        itemBuilder: (context, index) {
          return Padding(
            padding: const EdgeInsets.symmetric(horizontal: 8.0),
            child: ChoiceChip(
              label: Text(categories[index]),
              selected: _selectedCategory == categories[index],
              onSelected: (selected) {
                setState(() {
                  _selectedCategory = categories[index];
                });
              },
                  ),
                );
              },
            ),
          ),
          Expanded(
            child: _isLoading
                ? const Center(child: CircularProgressIndicator())
                : RefreshIndicator(
                    onRefresh: () async {
                      setState(() => _isLoading = true);
                      await Future.delayed(const Duration(seconds: 1));
                      setState(() => _isLoading = false);
                    },
                    child: ListView.builder(
                      padding: const EdgeInsets.all(8),
                      itemCount: mockSkills.length,
                      itemBuilder: (context, index) {
                        final skill = mockSkills[index];
                        if (_selectedCategory != 'All' && skill.category != _selectedCategory) {
                          return const SizedBox.shrink();
                        }
                        return InkWell(
                          onTap: () => _handleSkillExchange(skill),
                          borderRadius: BorderRadius.circular(12),
                          child: SkillCard(skill: skill),
                        );
                      },
                    ),
                  ),
          ),
        ],
      ),
    );
  }
}
```

Output:



Discover Skills

DEBUG

SkillXchange

120 Coins

All

Programming

Art

Alex Johnson

Programming

50 Coins

Flutter Development

Learn to build cross-platform apps with Flutter

10:00 AM - 12:00 PM

4/10

Sarah Miller

Art

40 Coins

Digital Painting

Master digital art techniques with Procreate

3:00 PM - 5:00 PM

6/8

+

Discover

Schedule

Profile

Conclusion:

Navigation and gestures are essential for building interactive and user-friendly Flutter apps. With **basic and named routing**, developers can structure apps efficiently and enhance maintainability. On the other hand, **gestures and multi-gestures** empower the app with intuitive user interactions, improving the overall user experience.