Name:Komal Singh
Div:D15B
Roll No:56

MPL Practical 5

Aim:

To implement navigation, routing, and gesture detection in a Flutter application, ensuring smooth transitions between screens and interactive user interactions.

Theory:

Flutter provides built-in navigation and gesture detection capabilities that enable seamless user experiences in mobile applications.

1. **Navigation in Flutter:**
    - Navigation allows users to move between different screens (routes) in an application.
    - The `Navigator` widget manages a stack-based approach for screen transitions using `push()` and `pop()`.
    - Named routes (`Navigator.pushNamed()`) provide better management and reusability of navigation logic.
2. **Gesture Detection in Flutter:**
    - Gesture detection enhances user interactivity by responding to touch inputs such as taps, swipes, and drags.
    - The `ElevatedButton`'s `onPressed` function is commonly used for button interactions.
    - The `GestureDetector` widget allows custom touch detection (e.g., double taps, long presses).
3. **Routing Methods:**
    - **Direct Routing:** Using `Navigator.push()` to navigate to new screens dynamically.

    - **Named Routing:** Pre-defining routes in `MaterialApp` and using `Navigator.pushNamed()` for structured navigation.

Steps to Apply Navigation, Routing, and Gestures in a Flutter App

1. Set Up the Flutter Project

- Create a new Flutter project or use an existing one.
- Ensure all necessary dependencies are available in `pubspec.yaml`.

2. Create the Required Screens

- Design a `LoginScreen` with input fields and a login button.
- Create a `HomeScreen` where users will be navigated after login.

3. Implement Navigation (Route Handling)

- Use `Navigator.push()` for navigating from the login screen to the home screen.
- Ensure navigation occurs only when form validation is successful.

4. Add Gesture Detection

- Implement button clicks (`ElevatedButton` onPressed) for navigation.
- Use `GestureDetector` for handling custom touch interactions if needed.

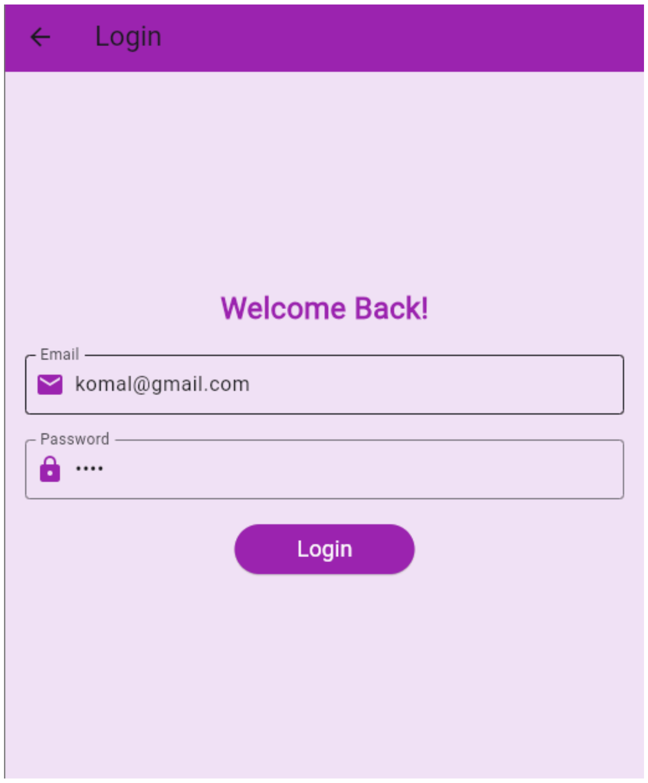
5. Define Named Routes (Optional for Better Management)

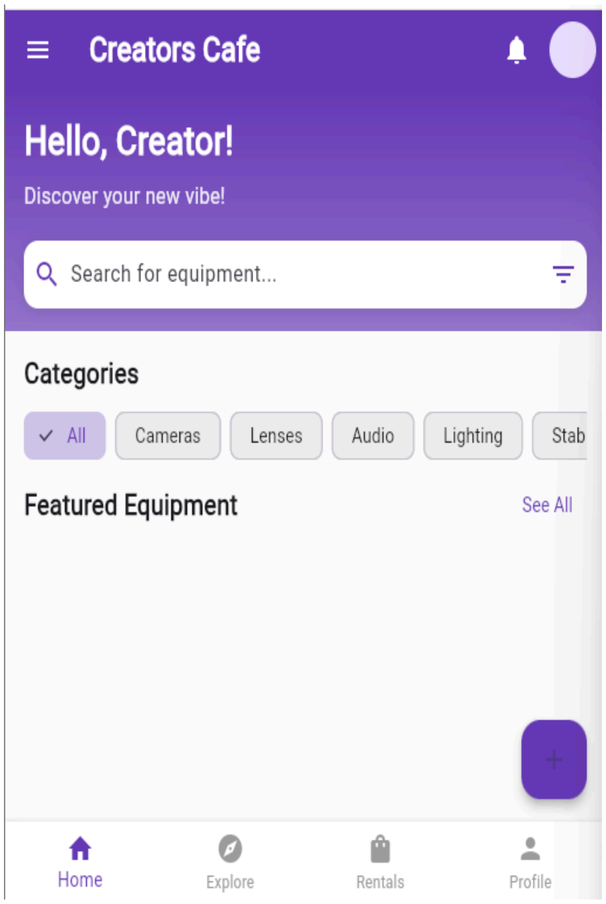
- Set up named routes in the `MaterialApp` widget.
- Navigate using `Navigator.pushNamed()` instead of defining routes inline.

6. Test the Navigation and Gestures

- Run the app and ensure tapping the login button successfully navigates to the home screen.
- Validate that gestures (e.g., button press) are working correctly.

Output:


Login
Welcome Back!
Email
komal@gmail.com
Password
••••
Login


Creators Cafe
Hello, Creator!
Discover your new vibe!
Search for equipment...
Categories
All
Cameras
Lenses
Audio
Lighting
Stab
Featured Equipment
See All
Home
Explore
Rentals
Profile

Conclusion:

By implementing navigation, routing, and gesture detection, a Flutter app can provide a smooth and interactive user experience. Proper screen transitions, form validations before navigation, and handling gestures improve usability and responsiveness, making the app more user-friendly and intuitive.