Name: Lavin Valechha Roll No: 70 Div: D15B

MAD Lab Experiment No 5

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

Theory:

Navigation

- Purpose: Navigation refers to the process of guiding users through the different screens (or "routes") within your app, allowing them to explore information and complete tasks. It establishes a clear flow and user experience.
- Techniques in Flutter:
  a. Navigator: Flutter's built-in Navigator class is the central component for managing navigation. It provides methods for pushing new routes onto the navigation stack (moving forward), popping routes from the stack (going back), and replacing the current route.
  b. Named Routes: Named routes associate a unique string identifier with each route, making navigation more readable and maintainable. You define these routes in the MaterialApp or WidgetsApp constructor's routes property.

Routing

- Purpose: Routing defines the logic behind how the app determines which route (screen) to display based on user actions or data. It establishes the mapping between events and destinations.
- Implementation in Flutter:
  a. Basic Navigation: For simple navigation, you can use the Navigator.push() and Navigator.pop() methods directly.
  b. Declarative Routing (Packages): For complex applications with deep linking or advanced navigation patterns, consider using third-party routing packages like go_router or fluro. These packages provide a more declarative approach to defining routes and handling transitions.

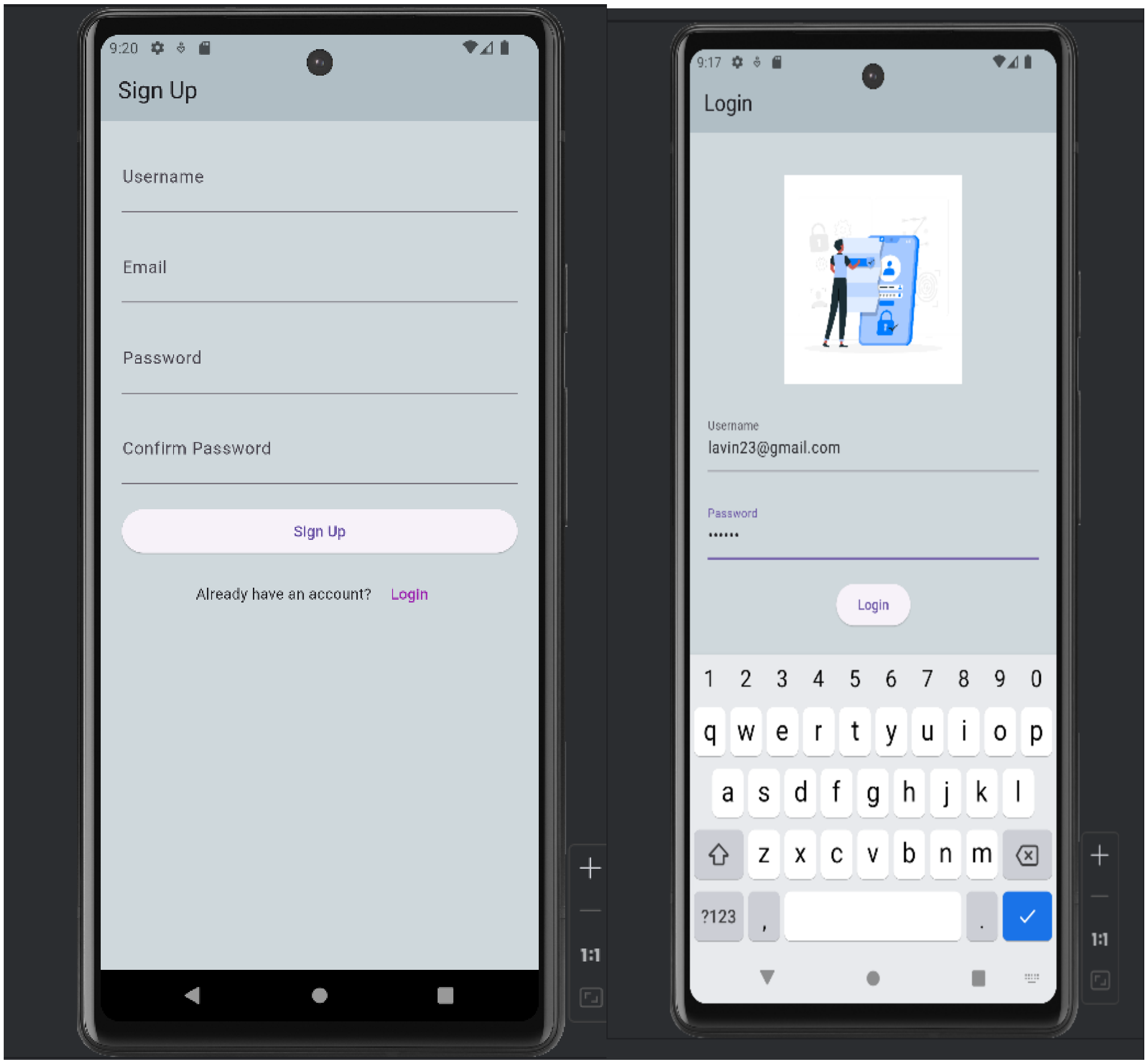
Gestures

- Purpose: Gestures are user interactions with the touch screen that control the app's behavior. They allow users to navigate, interact with UI elements, and manipulate data.
- Handling Gestures in Flutter:
  a. Gesture Detectors: Flutter provides various gesture detectors (like

GestureDetector, TapGestureRecognizer, SwipeGestureRecognizer) to capture and interpret user gestures. These detectors trigger callbacks based on the type and timing of the gesture.

Code:

```
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.dart';
import 'signup.dart';
import 'home.dart';
import 'utils/utils.dart';
import 'main.dart';

class Login extends StatefulWidget {
  const Login({Key? key}) : super(key: key);

  @override
  State<Login> createState() => _LoginState();
}

class _LoginState extends State<Login> {
  final _formKey = GlobalKey<FormState>();
  final emailController = TextEditingController();
  final passwordController = TextEditingController();
  final _auth = FirebaseAuth.instance;

  void login() {
      auth
        .signInWithEmailAndPassword(
      email: emailController.text,
      password: passwordController.text.toString(),
    )
        .then((value) {
      Navigator.push(
        context,
        MaterialPageRoute(builder: (context) => Home()),
      );
    }).onError((error, stackTrace) {
      utils().toastMessage(error.toString());
    });
  }
  @override
  void dispose() {
    super.dispose();
    emailController.dispose();
    passwordController.dispose();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text("Login"),
        backgroundColor: Colors.blueGrey[200],
        automaticallyImplyLeading: false,
      ),
      body: Container(
        height: double.infinity,
        padding: EdgeInsets.all(20),
        color: Colors.blueGrey[100],
        child: SingleChildScrollView(
          child: Column(
            crossAxisAlignment: CrossAxisAlignment.center,
            children: [
              SizedBox(height: 20),
              Image.asset(
                "images/login.jpg",
                height: 200,
              ),
              SizedBox(height: 20),
              Form(
                key: _formKey,
                child: Column(
                  children: [
                    TextFormField(
                      controller: emailController,
                      decoration: InputDecoration(
                        labelText: 'Username',
                      ),
                      validator: (value) {
                        if (value!.isEmpty) return 'Empty email';
                        return null;
                      },
                    ),
                    SizedBox(height: 20),
                    TextFormField(
                      controller: passwordController,
                      obscureText: true,
                      decoration: InputDecoration(
                        labelText: 'Password',
                      ),
                      validator: (value) {
                        if (value!.isEmpty) return 'Empty password';
                        return null;
                      },
                    ),
                  ],
                ),
              ),
              SizedBox(height: 20),
              ElevatedButton(
                onPressed: () {
                  if (_formKey.currentState!.validate()) {
                    login();
                  }
                },
                child: Text("Login"),
                style: ElevatedButton.styleFrom(),
              ),
              SizedBox(height: 10),
              Row(
                mainAxisAlignment: MainAxisAlignment.center,
                children: [
                  Text("Don't have an account? "),
                  TextButton(
                    onPressed: () {
                      Navigator.push(
                        context,
                        MaterialPageRoute(builder: (context) => Signup()),
                      );
                    },
                    child: Text(
                      "Sign Up",
                      style: TextStyle(color: Colors.purple),
                    ),
                  ),
                ],
              )
            ],
          ),
        ),
      ),
      );
     }
    }
```
Output:



Username
lavin23@gmail.com

Password
••••••

Conclusion: The integration of navigation, routing, and gestures in Flutter app development significantly enhances user experience and interaction. Leveraging Flutter's robust navigation system, hierarchical routing, and diverse gesture recognizers, developers can create intuitive, seamless, and engaging applications. By prioritizing user-centric design and functionality.