Name: Mohit Kerkar **Div: D15C** **Roll No: 23**

Experiment 02
Aim: To design Flutter UI by including common widgets.

Theory:

Flutter provides a rich set of widgets such as Text, Container, Row, Column, ListView, etc., that allow flexible UI design. These widgets form the building blocks of any Flutter interface.

Sample code for the registration page containing common widgets is as follows:

```
import 'package:flutter/material.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'society_list_screen.dart';

class RegisterScreen extends StatefulWidget {
  final User user;

  const RegisterScreen({Key? key, required this.user}) : super(key: key);

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

class _RegisterScreenState extends State<RegisterScreen> {
  final FirebaseFirestore _db = FirebaseFirestore.instance;
  final TextEditingController _contactController = TextEditingController();
  final TextEditingController _societyController = TextEditingController();
  final TextEditingController _flatNumberController = TextEditingController();
  final _formKey = GlobalKey<FormState>();

  String _userType = "Resident"; // Default user type
  bool _isResident = true;
  bool _isLoading = false;

  Future<void> _registerUser() async {
  if (!_formKey.currentState!.validate()) return;
  setState(() => _isLoading = true);

  try {
    String societyId = ""; // Initialize society ID

    // If user is a Manager, create a society
    if (_userType == "Manager") {
      DocumentReference societyRef = await _db.collection("societies").add({
        "name": _societyController.text.trim(),
        "location": "Not specified", // Default location, can be updated later
        "createdBy": widget.user.uid, // Store Manager's UID
      });
      societyId = societyRef.id; // Save society ID
    }

    // Store user details in Firestore
```

```
    await _db.collection("users").doc(widget.user.uid).set({
      "uid": widget.user.uid,
      "name": widget.user.displayName ?? "Unknown",
      "email": widget.user.email,
      "contactNumber": _contactController.text.trim(),
      "userType": _userType,
      "societyName": _societyController.text.trim(),
      "societyId": _userType == "Manager" ? societyId : "", // Assign society to Managers only
      "flatNumber": _isResident ? _flatNumberController.text.trim() : "N/A",
      "photoUrl": widget.user.photoURL ?? "",
      "createdAt": FieldValue.serverTimestamp(),
    });

    Navigator.pushReplacement(
  context,
  MaterialPageRoute(
    builder: (context) => SocietyListScreen(
      userId: widget.user.uid,
      userType: _userType, // Ensure correct value is passed
    ),
  ),
);


    ScaffoldMessenger.of(context).showSnackBar(
      const SnackBar(content: Text("✅ Registration Successful!")),
    );
  } catch (e) {
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(content: Text("❌ Error: ${e.toString()}")),
    );
  } finally {
    setState(() => _isLoading = false);
  }
}


  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text("Register")),
      body: GestureDetector(
        onTap: () => FocusScope.of(context).unfocus(),
        child: Padding(
          padding: const EdgeInsets.all(16.0),
          child: Form(
            key: _formKey,
            child: SingleChildScrollView(
              child: Column(
                crossAxisAlignment: CrossAxisAlignment.start,
                children: [
```

```
            const Text(
              "Complete Your Registration",
              style: TextStyle(fontSize: 22, fontWeight: FontWeight.bold),
            ),
            const SizedBox(height: 20),

            // Full Name
            TextFormField(
              initialValue: widget.user.displayName ?? "Unknown",
              readOnly: true,
              decoration: const InputDecoration(
                labelText: "Full Name",
                border: OutlineInputBorder(),
              ),
            ),
            const SizedBox(height: 10),

            // Email
            TextFormField(
              initialValue: widget.user.email,
              readOnly: true,
              decoration: const InputDecoration(
                labelText: "Email",
                border: OutlineInputBorder(),
              ),
            ),
            const SizedBox(height: 10),

            // Contact Number
            TextFormField(
              controller: _contactController,
              keyboardType: TextInputType.phone,
              decoration: const InputDecoration(
                labelText: "Contact Number",
                border: OutlineInputBorder(),
              ),
              maxLength: 10,
              validator: (value) {
                if (value!.isEmpty) return "Enter Contact Number";
                // if (!RegExp(r'^[0-9]{10}\$').hasMatch(value)) return "Enter a valid 10-digit
number";
                return null;
              },
            ),
            const SizedBox(height: 10),

            // User Type Dropdown
            DropdownButtonFormField<String>(
              value: _userType,
              items: ["Resident", "Manager"].map((type) {
                return DropdownMenuItem(value: type, child: Text(type));
```

```
            }).toList(),
            onChanged: (value) {
              setState(() {
                _userType = value!;
                _isResident = _userType == "Resident";
              });
            },
            decoration: const InputDecoration(
              labelText: "User Type",
              border: OutlineInputBorder(),
            ),
          ),
          const SizedBox(height: 10),

          // Society Name
          TextFormField(
            controller: _societyController,
            decoration: const InputDecoration(
              labelText: "Society Name",
              border: OutlineInputBorder(),
            ),
            validator: (value) => value!.isEmpty ? "Enter Society Name" : null,
          ),
          const SizedBox(height: 10),

          // Flat Number (Only for Residents)
          if (_isResident)
            TextFormField(
              controller: _flatNumberController,
              decoration: const InputDecoration(
                labelText: "Flat Number",
                border: OutlineInputBorder(),
              ),
              validator: (value) => value!.isEmpty ? "Enter Flat Number" : null,
            ),
          const SizedBox(height: 20),

          // Register Button
          SizedBox(
            width: double.infinity,
            child: ElevatedButton(
              onPressed: _isLoading ? null : _registerUser,
              child: _isLoading
                  ? const CircularProgressIndicator(color: Colors.white)
                  : const Text("Complete Registration"),
            ),
          ),
        ],
      ),
    ),
  ),
```

```
        ),
      ),
    );
  }
}
```

In the SignUp form, multiple commonly used Flutter widgets have been utilized to create an interactive and well-structured UI:

- **TextField**: Used to capture user input such as email, name, and flat number. These are interactive fields allowing text input.
- **DropdownButtonFormField**: Used to allow users to select options from a dropdown list — for example, selecting "Resident" or "Manager" as the user type.
- **Column**: Used to align widgets vertically — making sure the form fields are placed one below the other in a clean stacked layout.
- **Row**: Used wherever horizontal alignment is required, such as displaying form elements side-by-side or organizing parts of the layout.
- **Padding and SizedBox**: Used for spacing and improving layout aesthetics — giving proper margin and breathing room between widgets.
- **ElevatedButton**: A clickable button that triggers actions, such as submitting the registration form.
- **Form and GlobalKey<FormState>**: While not visual widgets, they are crucial for validating the form and maintaining form state.
- **Responsive Design Considerations**: The use of flexible layouts and widgets ensures that the form looks good on different screen sizes.

Conclusion: Understanding and utilizing core widgets allowed us to build a structured, user-friendly Society Profile UI.