EXPERIMENT 4

AIM : To create an interactive Form using a form widget.

1. Form Widget:

- The Form widget is a container for managing form-related interactions. It allows for validation and saving the form data.
- It keeps track of the state of all the fields within the form (like TextFormField widgets) through a GlobalKey<FormState>.

2. TextFormField:

- This is the main widget used for collecting user input (such as text). It integrates easily with form validation and submission.
- You can apply validators to the TextFormField to ensure the input meets specific criteria (e.g., required fields, correct format).

3. GlobalKey:

- A GlobalKey<FormState> is essential for managing the form's state (e.g., validating fields, saving data). It's assigned to the Form widget and can be used to trigger actions like form validation or saving the data.

4. Validation:

- You can define validation rules on each form field. The TextFormField widget has a validator property, which allows you to write logic that will run whenever the form is validated.
- A validator checks whether the input meets the required format (such as checking for valid email format or a non-empty field).

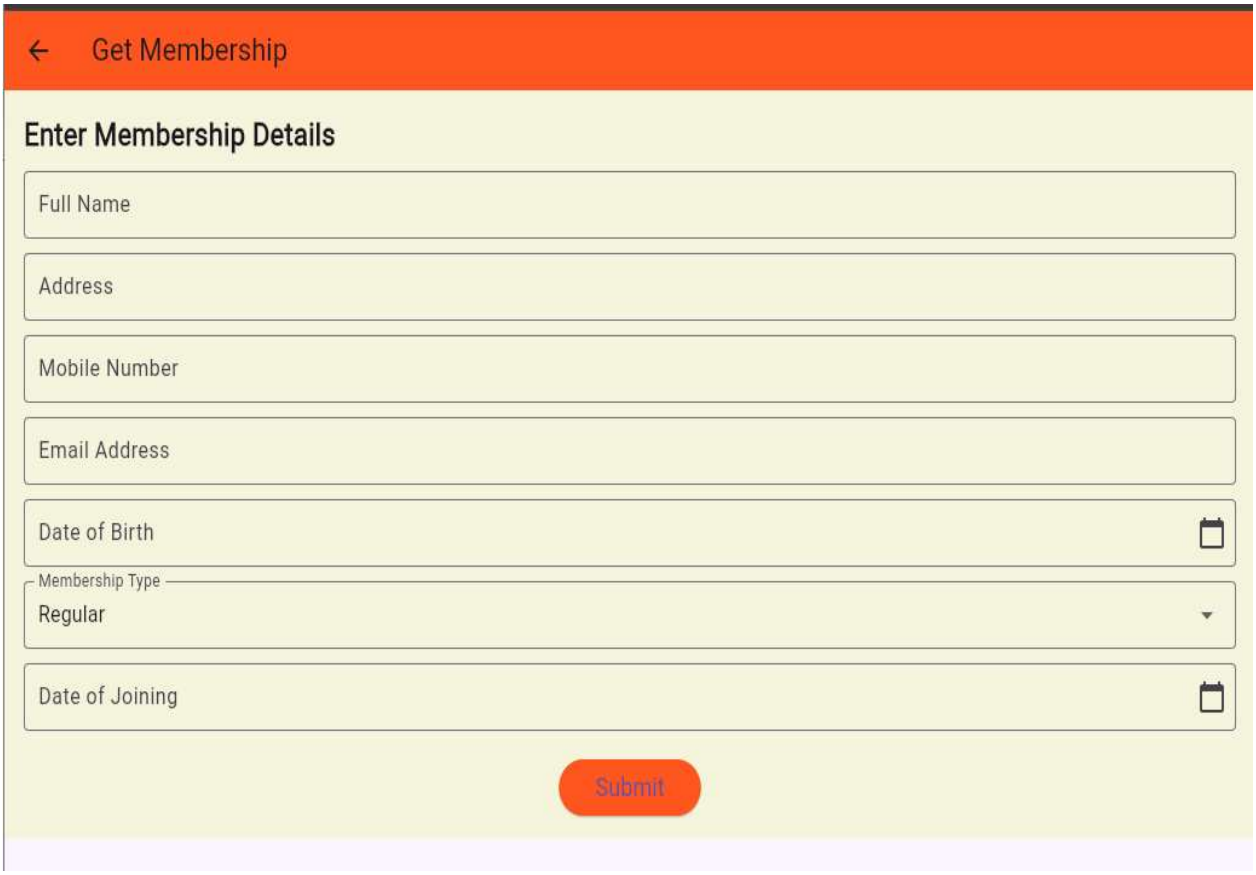
5. Form Submission:

- When you're ready to submit the form, you can call formKey.currentState?.save() to trigger the save method for all fields or formKey.currentState?.validate() to check if all the fields pass the validation checks.

6. Saving Data:

- After validation, data entered in the form fields can be saved to variables or used for further processing, such as sending it to a server.

Screenshots:



Code:

```
final _formKey = GlobalKey<FormState>();

Form(
  key: _formKey,
  child: Column(
    crossAxisAlignment: CrossAxisAlignment.stretch,
    children: [
      const Text(
        "Create an Account",
        textAlign: TextAlign.center,
        style: TextStyle(
          fontSize: 24,
          fontWeight: FontWeight.bold,
          fontFamily: 'Roboto',
          color: Color(0xFF1E3D34),
        ),
      ),
      const SizedBox(height: 20),

      // Name Field
      TextFormField(
        controller: _nameController,
        decoration: const InputDecoration(
          labelText: "Full Name",
          border: OutlineInputBorder(),
          prefixIcon: Icon(Icons.person, color: Color(0xFF1E3D34)),
        ),
        validator: (value) {
          if (value == null || value.isEmpty) {
            return "Please enter your full name";
          }
          return null;
        },
      ),
      const SizedBox(height: 15),

      // Email Field
      TextFormField(
        controller: _emailController,
```

```
        keyboardType: TextInputType.emailAddress,
        decoration: const InputDecoration(
          labelText: "Email",
          border: OutlineInputBorder(),
          prefixIcon: Icon(Icons.email, color: Color(0xFF1E3D34)),
        ),
        validator: (value) {
          if (value == null || !value.contains('@') || !value.contains('.')) {
            return "Enter a valid email address";
          }
          return null;
        },
      ),
      const SizedBox(height: 15),

      // Password Field
      TextFormField(
        controller: _passwordController,
        obscureText: true,
        decoration: const InputDecoration(
          labelText: "Password",
          border: OutlineInputBorder(),
          prefixIcon: Icon(Icons.lock, color: Color(0xFF1E3D34)),
        ),
        validator: (value) {
          if (value == null ||
!RegExp(r'^(?=.*[A-Z])(?=.*\d)(?=.*[@#\$%^&+=]).{8,}$').hasMatch(value)) {
            return "Password must be 8+ chars, include uppercase, number, and special
char";
          }
          return null;
        },
      ),
      const SizedBox(height: 15),

      // Confirm Password Field
      TextFormField(
        controller: _confirmPasswordController,
        obscureText: true,
        decoration: const InputDecoration(
```

```
          labelText: "Confirm Password",
          border: OutlineInputBorder(),
          prefixIcon: Icon(Icons.lock, color: Color(0xFF1E3D34)),
        ),
        validator: (value) {
          if (value != _passwordController.text) {
            return "Passwords do not match";
          }
          return null;
        },
      ),
    ],
  ),
);
```

Conclusion:

In this experiment, we successfully created an interactive form using Flutter's Form and TextFormField widgets. The form included multiple input fields such as full name, email, password, and confirm password. We used a GlobalKey<FormState> to manage the form's state and implemented custom validators for each field to ensure data integrity and proper user input.

The experiment demonstrated how Flutter allows developers to build responsive and user-friendly forms with built-in validation, easy data handling, and clean UI design. This approach is highly efficient for applications that require user registration, login, or any kind of data input.

Overall, this experiment enhanced our understanding of form validation, form state management, and user interaction handling in Flutter.