Experiment No.4

- **Aim:** To create an interactive Form using form widget.

- **Theory:**

Forms are crucial components in mobile and web applications, facilitating user input and data submission. In Flutter, forms are created using the Form widget, which serves as a container for form fields and manages their state.

Key Concepts:

1. Form Widget: The Form widget is a container for form fields, allowing developers to organize and manage user input within a structured layout. It provides functionalities for validation, submission, and data handling.

2. FormField Widget: Each input field within a form is encapsulated by a FormField widget, which represents a single form element. These widgets handle user input, validation, and interaction with the form.

3. FormState: The Form widget maintains the state of its form fields through an associated FormState object. This state includes validation status, error messages, and user-entered data for each form field.

Building Interactive Forms:

Creating interactive forms in Flutter involves integrating various form fields, implementing validation logic, and managing user interactions. The following steps outline the process:

1. Add Form Widget: Begin by wrapping your form fields within a Form widget, providing a context for validation and state management.

2. Implement FormFields: Utilize different form field widgets such as TextFormField, CheckboxFormField, RadioFormField, etc., based on the type of data input required.

3. Validate User Input: Define validation logic for each form field by setting validators within the corresponding form field widget. Validators can be functions that check for required fields, valid email addresses, numeric values, etc.

4. Handle Form Submission: Implement a function to handle form submission, typically triggered by a button press. Within this function, access the current form state using the Form.of(context) or FormState.of(context) method to retrieve form data and perform further processing.

Advanced Form Functionality and Error Handling

To enhance the functionality of interactive forms in Flutter, consider implementing additional features such as dynamic form fields, error handling, and form field customization.

1. Dynamic Form Fields: Allow users to dynamically add or remove form fields based on user actions or requirements, such as adding multiple email addresses or selecting multiple options.

2. Error Handling: Customize error messages and UI feedback to provide users with clear guidance on input errors. Utilize the FormState object to access error information and update error messages dynamically.

3. Form Field Customization: Customize the appearance and behavior of form fields using properties and parameters provided by Flutter widgets. This includes styling text inputs, adding icons, and implementing custom validation logic.

By mastering these concepts and techniques, developers can create interactive forms that provide a seamless user experience and effectively capture user input in Flutter applications.

- **Code:-**

```
class LoginPage extends StatelessWidget {
  Widget build(BuildContext context) {
            const SizedBox(height: 50),

            // welcome back, you've been missed!
            Text(
              'Welcome back you\'ve been missed!',
              style: TextStyle(
                color: Colors.grey[700],
                fontSize: 16,
              ), // TextStyle
            ), // Text

            const SizedBox(height: 25),

            // username textfield
            MyTextField(
              controller: usernameController,
              hintText: 'Username',
              obscureText: false,
            ),

            const SizedBox(height: 10),

            // password textfield
            MyTextField(
              controller: passwordController,
              hintText: 'Password',
              obscureText: true,
            ),
```

```
          Padding(
            padding: const EdgeInsets.symmetric(horizontal: 25.0),
            child: Row(
              mainAxisAlignment: MainAxisAlignment.end,
              children: [
                Text(
                  'Forgot Password?',
                  style: TextStyle(color: Colors.grey[600]),
                ), // Text
              ],
            ), // Row
          ), // Padding

          const SizedBox(height: 25),

          // sign in button
          MyButton(
            onTap: signUserIn,
          ),
```

TextFormField Widgets:

Instead of MyTextField, we'll use TextFormField widgets for username and password fields. These widgets come with built-in validation and error handling capabilities.

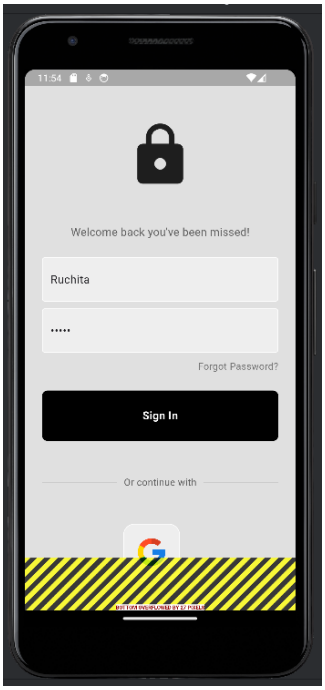
Validation:

We'll add validation logic to the TextFormField widgets by providing a validator function. This function will return an error message if the input is invalid or null otherwise.

Form Submission:

We'll handle form submission by defining a function to be executed when the sign-in button is pressed. This function will first validate the form and then proceed with the sign-in process if the form is valid.

- **Output:-**



- **Conclusion:**

By implementing the interactive form using the Form widget in Flutter, I have successfully created a user-friendly interface for capturing user input and handling form submission. This experience has enhanced my understanding of form management and validation within Flutter applications.