**Shravani Pore**
**D15A_45**

# EXPT 4

**AIM:**
To create an interactive Form using form widget

**THEORY:**
**Grouping Form Fields:** The Form widget allows us to group multiple form fields together. This grouping is important for managing the state of the form and handling form submission.

**State Management:** The Form widget manages the state of the form fields within it. It automatically keeps track of the current value of each form field, their validation status, and any errors associated with them.

**Validation:** The Form widget provides built-in support for form field validation. We can specify validation logic for each form field, and the Form widget will automatically validate the form when it's submitted. Validation ensures that the user input meets certain criteria, such as required fields, valid email addresses, or password length.

**Error Reporting:** If a form field fails validation, the Form widget displays error messages associated with the invalid fields. These error messages are typically displayed below the corresponding form fields to inform us about what went wrong.

**Submission Handling:** When the user submits the form, the Form widget can trigger a callback function, allowing us to handle form submission. This callback function can perform actions such as sending data to a server, saving data locally, or navigating to another screen.

**GlobalKey:** To access the state of a Form widget, we typically use a GlobalKey<FormState>. This key allows us to interact with the Form widget programmatically, such as triggering form validation or resetting the form fields.

**Nested Forms:** We can nest Form widgets within each other to create complex forms with different sections or groups of fields. Each nested Form manages its own set of form fields independently.

**SYNTAX:**
```
Form(
  key: _formKey, // GlobalKey<FormState>
  child: Column(
    children: [
      // Form fields go here
    ],
  ),
```

)

**WIDGET AND PROPERTIES:**

In this experiment we have focussed on the form widget. We have used form widget for the change password page and have given different validation criteria for the submission of the form. Thus we have used properties of form widget like validator and decoration.

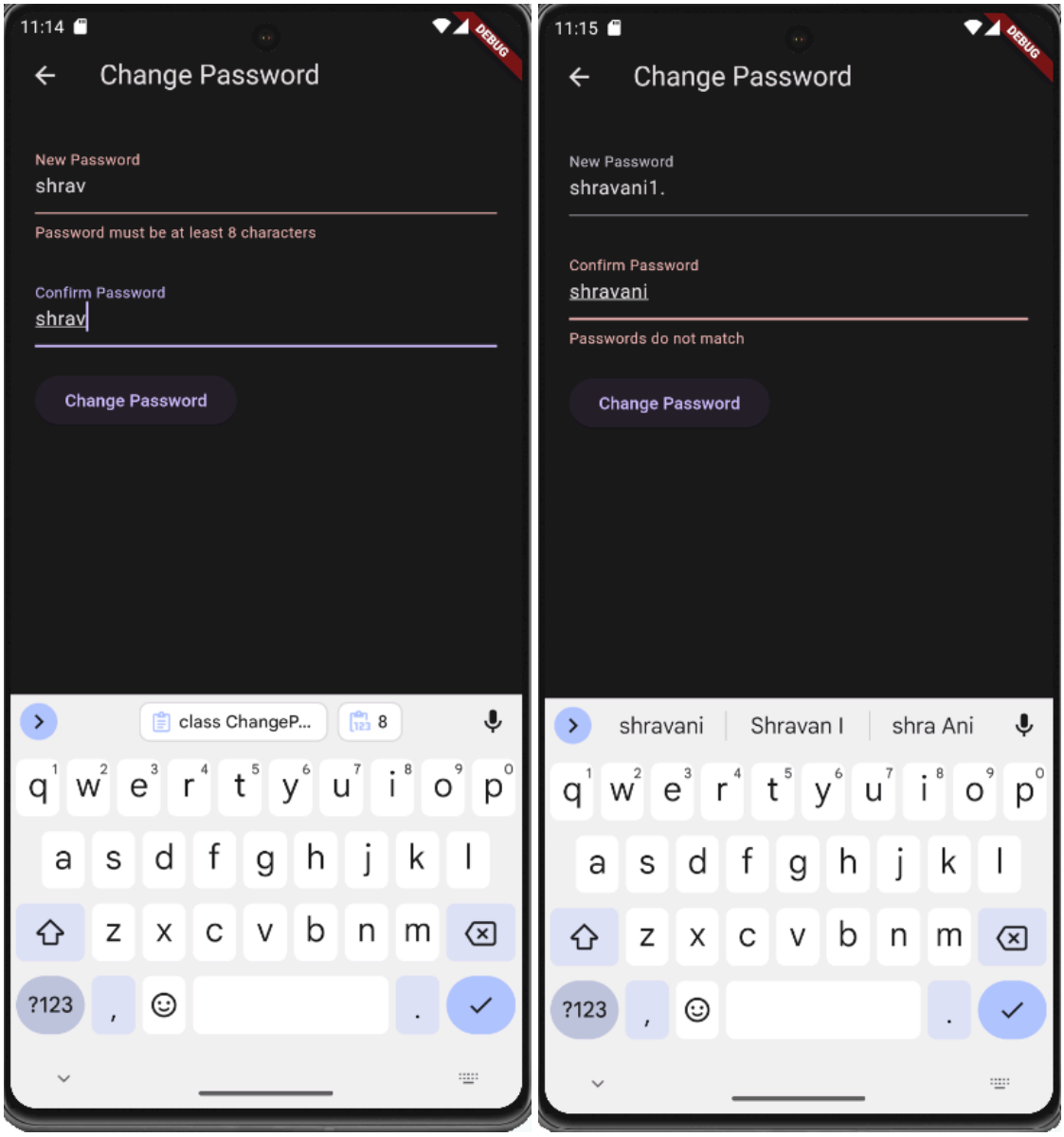
**CODE:**

```
class ChangePasswordPage extends StatefulWidget {
  @override
  _ChangePasswordPageState createState() => _ChangePasswordPageState();
}

class _ChangePasswordPageState extends State<ChangePasswordPage> {
  final GlobalKey<FormState> _formKey = GlobalKey<FormState>();
  TextEditingController _passwordController = TextEditingController();
  TextEditingController _confirmPasswordController =
TextEditingController();

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Change Password'),
      ),
      body: Padding(
        padding: const EdgeInsets.all(20.0),
        child: Form(
          key: _formKey,
          child: Column(
            crossAxisAlignment: CrossAxisAlignment.start,
            children: [
              TextFormField(
                controller: _passwordController,
                decoration: InputDecoration(
                  labelText: 'New Password',
                ),
                validator: (value) {
                  if (value!.isEmpty) {
                    return 'Password is required';
```

```
                }
                if (value.length < 8) {
                  return 'Password must be at least 8 characters';
                }
                if (!value.contains(RegExp(r'[!@#$%^&*(),.?":{}|<>]')))
{
                  return 'Password must contain at least one symbol';
                }
                return null;
              },
            ),
            SizedBox(height: 20),
            TextFormField(
              controller: _confirmPasswordController,
              decoration: InputDecoration(
                labelText: 'Confirm Password',
              ),
              validator: (value) {
                if (value != _passwordController.text) {
                  return 'Passwords do not match';
                }
                return null;
              },
            ),
            SizedBox(height: 20),
            ElevatedButton(
              onPressed: () {
                if (_formKey.currentState!.validate()) {
                  Navigator.pushNamed(context, '/password');
                  ScaffoldMessenger.of(context).showSnackBar(
                    SnackBar(
                      content: Text('Password changed successfully.'),
                    ),
                  );
                }
              },
              child: Text('Change Password'),
            ),
          ],
```

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

  @override
  void dispose() {
    _passwordController.dispose();
    _confirmPasswordController.dispose();
    super.dispose();
  }
}
```

**OUTPUT:**




**CONCLUSION:**

**Learnt about form widget and applied it in my project.**