Aim: To create an interactive Form using form widget

Theory:

In mobile app development, forms are essential for collecting user input, such as login details, preferences, or order information. Flutter provides a robust way to create forms that can manage validation, saving input, and handling submission events.

A form is a container that groups together multiple input fields (like text fields, dropdowns, checkboxes) and allows collective validation and saving of user input. It ensures that the input meets certain criteria before processing or submitting it.

Form Widget in Flutter

The Form widget in Flutter acts as a container for grouping and managing multiple input widgets like TextFormField, DropdownButtonFormField, etc. It uses a GlobalKey<FormState> to uniquely identify the form and enable form-wide operations such as:

- Validation: Checking if all inputs satisfy validation rules.
- Saving: Triggering callbacks to save the current input values.
- Resetting: Clearing or resetting the form fields.

By using a Form widget, developers can efficiently handle complex user input scenarios in a clean and maintainable way.

Basic Syntax of Forms in Flutter:

The typical steps to implement a form in Flutter include:

Define a GlobalKey for the FormState:

final _formKey = GlobalKey<FormState>();

Wrap input widgets inside a Form widget:

Form(

key: _formKey, child: Column(children: [

```
TextFormField( validator: (value) {
if (value == null || value.isEmpty) { return 'Please enter some text';
}
return null;
},
onSaved: (value) {
// Save the value to a variable
},
),
ElevatedButton( onPressed: () {
if (_formKey.currentState!.validate()) {
_formKey.currentState!.save();
// Process the data
}
},
child: Text('Submit'),
),
],
),
);
```

1. **Validation and Saving:**

○ Each input widget like TextFormField has a validator function that returns a validation error string or null if valid.

○ The onSaved callback captures and stores the input value when the form is saved.

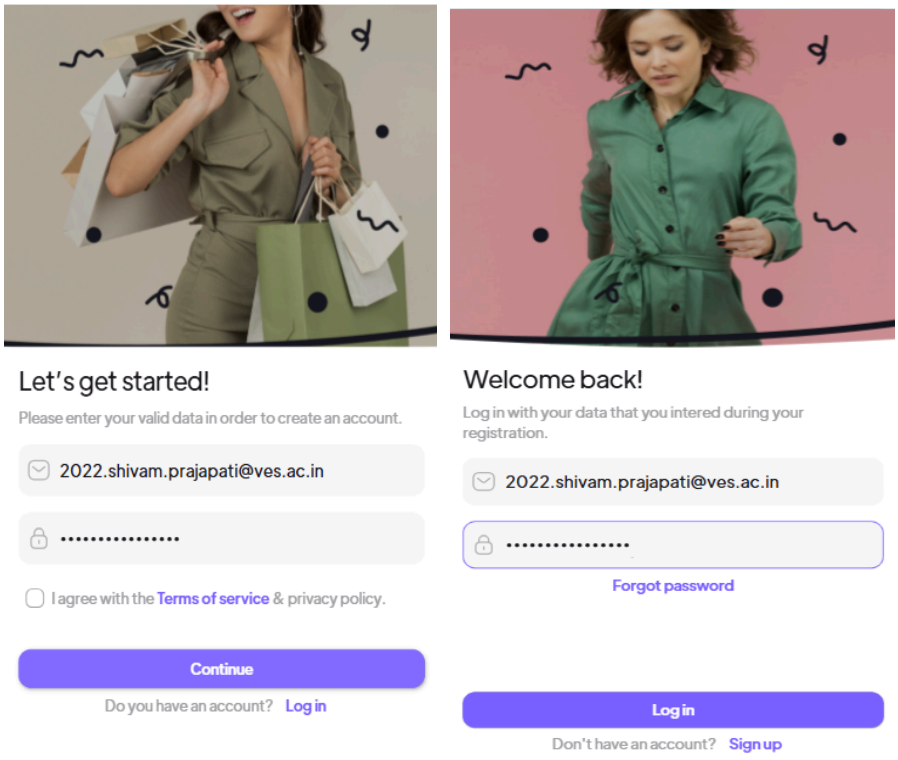
2. **Submit Handling:**

○ On submit, call _formKey.currentState!.validate() to validate all fields.

○ If valid, call _formKey.currentState!.save() to invoke onSaved for all fields.

Code: **https://github.com/Kingmaker-2/MPL-APP.git**

Output:



Basic form created

Conclusion: By creating an interactive form using the Form widget in Flutter, we have learned how to efficiently collect, validate, and manage user input within a mobile application. Understanding form fields, validation logic, and state management enables you to build robust and user-friendly interfaces for features like registration, login, and data entry. This forms an essential part of developing interactive and responsive Flutter applications.