Experiment 04 : To create an interactive Form using form widget.

Experiment No 4 **1)Creating an interactive Form using form widget**	
ROLL NO	**48**
NAME	**Shivani Nikam**
CLASS	**D15-B**
SUBJECT	**MAD & PWA Lab**
LO-MAPPED	

Aim: To create an interactive Form using form widget.

Theory:
In Flutter, forms are used to collect and validate user input. The `Form` widget is a container for form fields and provides various methods for managing form submission and validation. Here's a brief overview of Flutter forms and the `Form` widget:

Flutter Forms:

1. User Input Collection : Forms are used to collect user input, such as text input, selections, and other data, typically through form fields like text fields, checkboxes, radio buttons, dropdowns, etc.

2. Validation : Forms allow you to validate user input to ensure it meets certain criteria or constraints. Validation helps prevent invalid or incorrect data from being submitted.

3. Form Submission : Once the user has completed filling out the form, the data is submitted for processing. This may involve sending the data to a server, saving it locally, or performing other actions based on the application's requirements.

4. Feedback : Forms often provide feedback to users about the validity of their input. This feedback can include error messages, validation hints, or other indicators to guide users in providing correct input.

Form Widget:

The `Form` widget is a container for form fields and manages form state, validation, and submission. Here are some key points about the `Form` widget:

1. Key : The `Form` widget requires a `GlobalKey<FormState>` to uniquely identify the form and manage its state.

2. Validation : Each form field within the `Form` widget can have a validator function assigned to it. Validators are functions that take the current value of the form field as input and return an error message if the value is invalid, or `null` if it's valid.

3. Submission : The `Form` widget provides methods to handle form submission, such as `FormState.save()` to save the form data, and `FormState.validate()` to trigger validation of all form fields.

4. FormState : The `FormState` class represents the state of the form and provides methods to interact with and manipulate the form's state, such as accessing form field values, validating form fields, and resetting the form.

5. Nested Forms : You can nest `Form` widgets within each other to create complex forms with sections or subsections, each with its own validation and submission logic.

Overall, the `Form` widget in Flutter provides a convenient and powerful way to create interactive forms with input validation and submission capabilities. By using the `Form` widget along with form fields and validators, you can build robust and user-friendly forms in your Flutter applications.

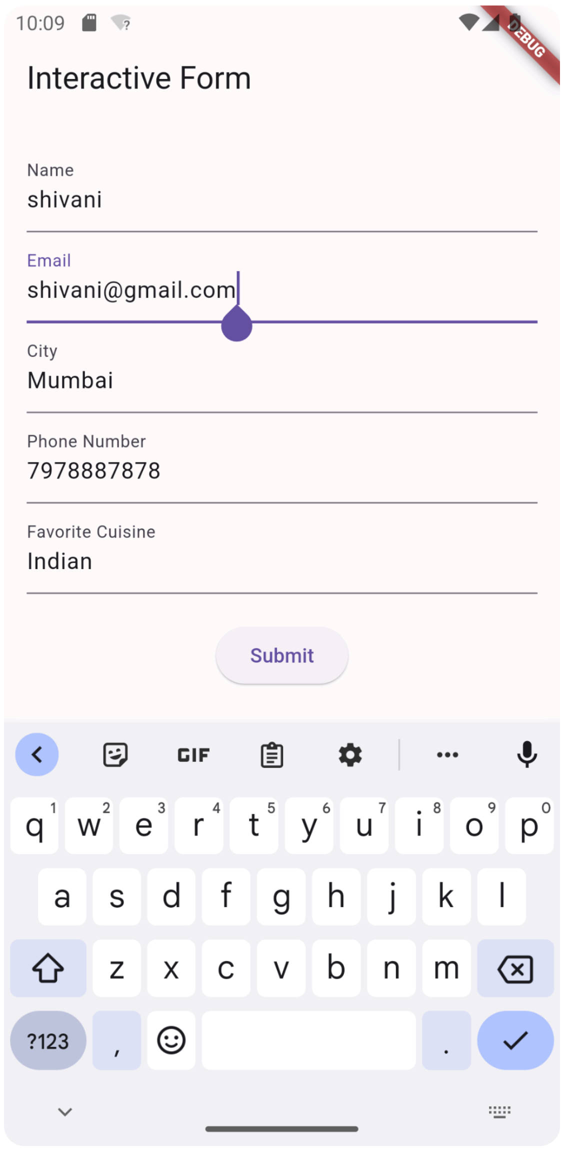
Code:

```
import 'package:flutter/material.dart';

void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Interactive Form Example',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: FormPage(),
    );
  }
}

class FormPage extends StatefulWidget {
  @override
  _FormPageState createState() => _FormPageState();
}

class _FormPageState extends State<FormPage> {
  final _formKey = GlobalKey<FormState>();
```

```
TextEditingController _nameController = TextEditingController();
TextEditingController _emailController = TextEditingController();
TextEditingController _cityController = TextEditingController();
TextEditingController _phoneNumberController = TextEditingController();
TextEditingController _foodTypeController = TextEditingController();

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text('Interactive Form'),
    ),
    body: Padding(
      padding: EdgeInsets.all(16.0),
      child: Form(
        key: _formKey,
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            TextFormField(
              controller: _nameController,
              decoration: InputDecoration(labelText: 'Name'),
              validator: (value) {
                if (value == null || value.isEmpty) {
                  return 'Please enter your name';
                }
                return null;
              },
            ),
            TextFormField(
              controller: _emailController,
              decoration: InputDecoration(labelText: 'Email'),
              validator: (value) {
                if (value == null || value.isEmpty) {
                  return 'Please enter your email';
                } else if (!_isValidEmail(value)) {
                  return 'Please enter a valid email';
                }
                return null;
              },
),
TextFormField(
  controller: _cityController,
  decoration: InputDecoration(labelText: 'City'),
  validator: (value) {
    if (value == null || value.isEmpty) {
      return 'Please enter your city';
    }
    return null;
  },
),
TextFormField(
  controller: _phoneNumberController,
  decoration: InputDecoration(labelText: 'Phone Number'),
  keyboardType: TextInputType.phone,
  validator: (value) {
    if (value == null || value.isEmpty) {
      return 'Please enter your phone number';
    }
    // Add phone number validation logic here if needed
    return null;
  },
),
TextFormField(
  controller: _foodTypeController,
  decoration: InputDecoration(labelText: 'Favorite Cuisine'),
  validator: (value) {
    if (value == null || value.isEmpty) {
      return 'Please enter the type of food you like';
    }
    return null;
  },
),
SizedBox(height: 20),
Center(
  child: ElevatedButton(
    onPressed: () {
      if (_formKey.currentState!.validate()) {
        // Form is valid, perform submission
        _submitForm();
```

```
                }
              },
              child: Text('Submit'),
            ),
          ),
        ],
      ),
    ),
  ),
 );
}

void _submitForm() {
 // Perform form submission logic here
 String name = _nameController.text;
 String email = _emailController.text;
 String city = _cityController.text;
 String phoneNumber = _phoneNumberController.text;
 String foodType = _foodTypeController.text;

 // Print form data (for demonstration)
 print('Name: $name');
 print('Email: $email');
 print('City: $city');
 print('Phone Number: $phoneNumber');
 print('Favorite Cuisine: $foodType');
}

bool _isValidEmail(String email) {
 // Regular expression for validating email format
 // This is a simple pattern, you may use a more comprehensive one for production
 return RegExp(r'^[\w-\.]+@([\w-]+\.)+[\w-]{2,4}$').hasMatch(email);
}

@override
void dispose() {
 _nameController.dispose();
 _emailController.dispose();
 _cityController.dispose();
 _phoneNumberController.dispose();
```

```
  _foodTypeController.dispose();
  super.dispose();
 }
}
```



Interactive Form

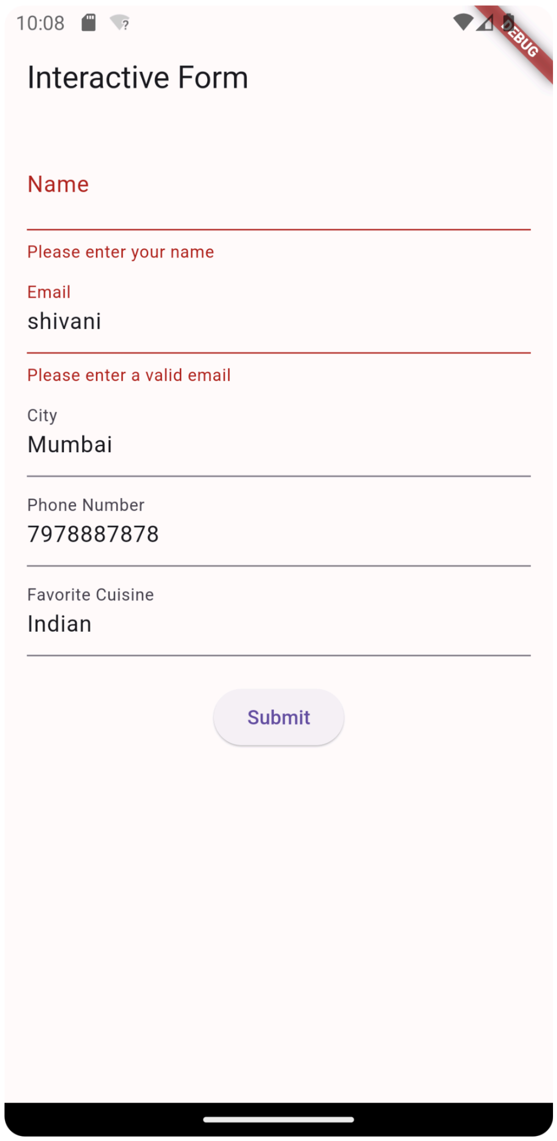
Name
shivani

Email
shivani@gmail.com

City
Mumbai

Phone Number
7978887878

Favorite Cuisine
Indian



Interactive Form

Name

Please enter your name

Email
shivani

Please enter a valid email

City
Mumbai

Phone Number
7978887878

Favorite Cuisine
Indian

Conclusion:
Form widget, helps in creating robust and user-friendly forms in Flutter applications, enabling efficient data collection and interaction with users. We have successfully implemented the Form widget in a Flutter application to create an interactive form for collecting user input.