Flutter Forms & User Input: TextFields, Validation & Buttons

Subtitle: Learn how to handle user input, validate data, and build interactive forms in Flutter apps.

Website Name: haas.dev
Website Link: https://dev-roast-app.vercel.app

Introduction

User input is critical for most apps—login forms, registration, search bars, or settings. Flutter makes input easy, but beginners often struggle with **TextFields, controllers, validation, and buttons**. This guide teaches you to handle input correctly.

Step 1: Using TextField

Basic TextField for input:

```
TextField(
  decoration: InputDecoration(
    labelText: "Enter your name",
    border: OutlineInputBorder(),
  ),
)
```

Exercise: Add a TextField to take the user's name.

Step 2: Using TextEditingController

- Controller helps **read input values**.
- Assign controller to TextField.

```
TextEditingController nameController = TextEditingController();

TextField(
  controller: nameController,
)
```

Exercise: Print input value to console on button press.

Step 3: Buttons to Submit Data

- Use **ElevatedButton** or **TextButton**
- On press, read value from controller

```
ElevatedButton(
  onPressed: () {
    print("Name: ${nameController.text}");
  },
  child: Text("Submit"),
)
```

Exercise: Display entered name below TextField after pressing submit.

Step 4: Input Validation

- Prevent empty or invalid input
- Use `if` checks inside button handler

```
ElevatedButton(
  onPressed: () {
    if (nameController.text.isEmpty) {
      print("Please enter your name");
    } else {
      print("Hello ${nameController.text}");
    }
  },
  child: Text("Submit"),
)
```

Exercise: Add validation for an **email field**.

Step 5: Form Widget

- Wrap multiple fields in a **Form**
- Use **GlobalKey** for validation

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

Form(
  key: _formKey,
  child: Column(
    children: [
      TextFormField(
        validator: (value) {
          if (value == null || value.isEmpty) {
            return "Field required";
          }
          return null;
        },
      ),
      ElevatedButton(
        onPressed: () {
          if (_formKey.currentState!.validate()) {
            print("Form valid");
          }
        },
        child: Text("Submit"),
      )
    ],
  ),
)
```

Exercise: Build a login form with **email and password** validation.

Step 6: Handling Focus & Keyboard

- Use **FocusNode** to move between fields
- Automatically close keyboard after submission

```
FocusScope.of(context).unfocus();
```

Exercise: Add two fields and move focus from email → password automatically.

Step 7: Mini Project

Build a **Simple Registration Form**:

Features:

Features:

- Name, Email, Password fields
- Validation for empty input & email format
- Submit button to display entered data
- Keyboard closes after submission

Key Takeaways

- TextField + TextEditingController = basic input
- Use Form + GlobalKey for multiple fields & validation
- Always validate user input
- Use buttons to trigger actions and update UI
- Handling focus improves user experience

Mastering forms is crucial because **every real app requires user input**. Messing this up makes your app unprofessional.

Visit **haas.dev** for more Flutter guides, tutorials, and complete project resources.

Website Name: haas.dev
Website Link: https://dev-roast-app.vercel.app
