

- **Aim:** To create an interactive Form using form widget
- **Theory:** In Flutter, forms are created using the Form widget, which serves as a container for input fields (like TextFormField) and allows you to handle the validation and submission of data.
- **Key Components:**
 - **Form Widget :** The Form widget is used to group multiple input fields. It provides the ability to manage and validate the form data. ○ A GlobalKey is required to track the form's state for validation and saving the data.
 - **TextForm Field :** TextFormField is the primary widget for collecting text input in Flutter forms. It comes with built-in support for validation and user input handling.
 - **Validation :** The validator property of the TextFormField allows you to define rules that ensure user inputs are valid (e.g., email format, required fields). ○ Flutter will automatically show error messages when the validation fails.
 - **Form State:** To manage the state of the form, you use FormState which allows you to validate, save, and reset the form. ○ You can trigger validation by calling formKey.currentState?.validate().
 - **Saving Data :** Once the form is valid, you can save the data using the onSaved property of the TextFormField.
- **Steps to Create an Interactive Form :**
 - **Create a Form Widget:** You use the Form widget to wrap the form fields and manage validation. A GlobalKey is used to access the form's state.
 - **Add Form Fields (TextFormField):** For each input field, you use the TextFormField widget. Each TextFormField can have a validator to ensure the input is valid (e.g., ensuring that the user provides a valid email, password, etc.). Use onSaved to store the user input when the form is submitted.
 - **Validate the Form :** You can validate the form using the formKey.currentState?.validate() method, which triggers the validator for each field. If any field is invalid, it prevents form submission.
 - **Handle Form Submission :** Once the form is validated, the form data is saved by calling formKey.currentState?.save(). The form can then be processed (e.g., sending data to a server, storing locally)
- **Code :**

```
class Validator {
  static String? validateEmail(String? value) {
    if (value == null || value.isEmpty) {
```

```

        return 'Please enter your email';
    }
    // Regex for validating email format
    final emailRegex =
    RegExp(r"[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$");
    if (!emailRegex.hasMatch(value)) {
        return 'Please enter a valid email address';
    }
    return null;
}

static String? validatePassword(String? value) {
    if (value == null || value.isEmpty) {
        return 'Please enter your password';
    }

    if (value.length < 6) {
        return 'Password must be at least 6 characters long';
    }

    final regex = RegExp(r'^(?=.*[A-Z])(?=.*[a-z])(?=.*\d){6,}$');
    if (!regex.hasMatch(value)) {
        return 'Password must contain at least one uppercase letter, one lowercase
letter, and one
number';
    }

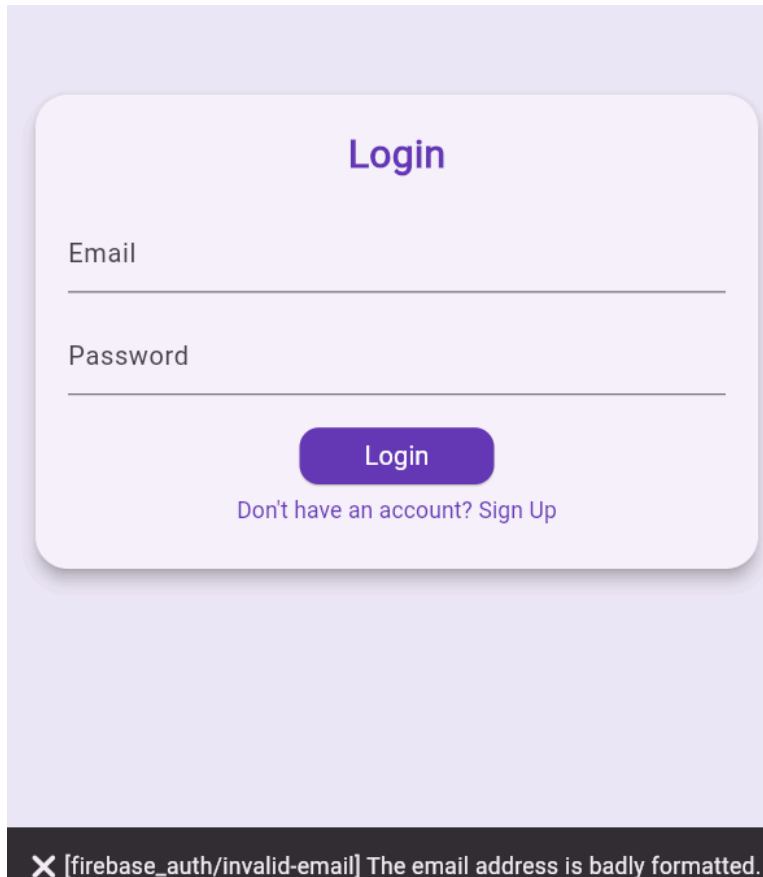
    return null;
}

static String? validateName(String? value) {
    if (value == null || value.isEmpty) {
        return 'Please enter a valid name';
    }
    if (value.length < 2){
        return 'Name must be at least 2 characters long';
    }
    return null;
}

```

```
static String? validateConfirmPassword(String value, String password) {  
  if (value.isEmpty) {  
    return 'Confirm password cannot be empty';  
  }  
  if (value != password) {  
    return 'Passwords do not match';  
  }  
  return null;  
}  
}
```

- Output :




Login

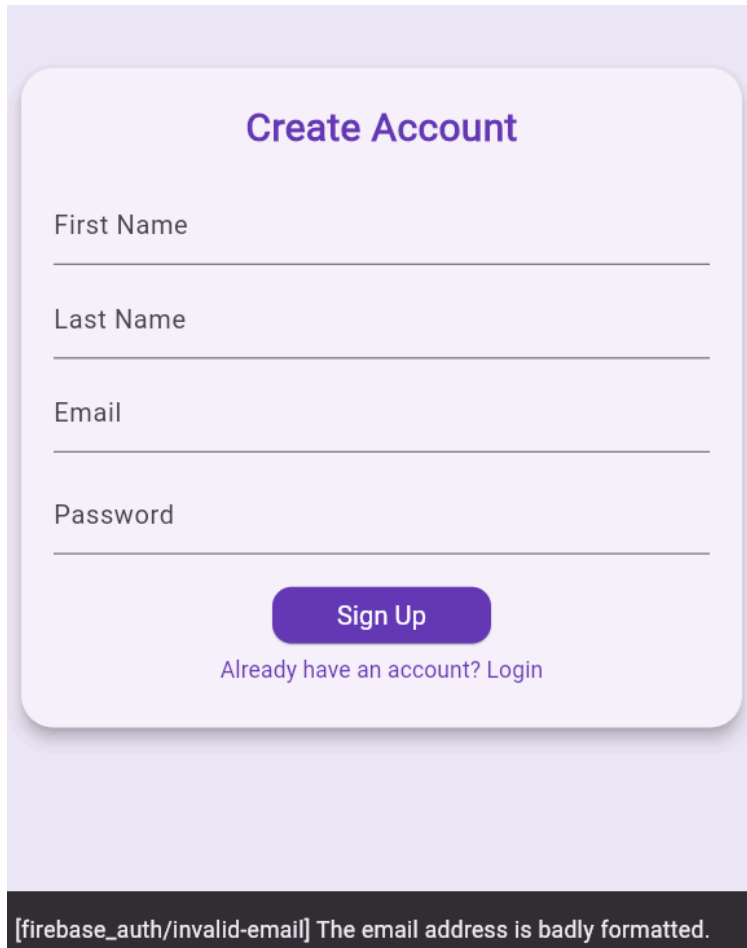
Email

Password

Login

[Don't have an account? Sign Up](#)

 [firebase_auth/invalid-email] The email address is badly formatted.



Create Account

First Name

Last Name

Email

Password

Sign Up

[Already have an account? Login](#)

`[firebase_auth/invalid-email]` The email address is badly formatted.

- **Conclusion :** In this experiment, you learned how to create an interactive form in Flutter using the Form widget. The Form widget allows for easy management of form fields, validation, and submission, while TextFormField provides the necessary functionality for input fields. By using validation and form state management, you can ensure that data entered by the user is valid before proceeding with any further operations, such as sending the data to a backend or storing it locally. This is a fundamental concept in Flutter for collecting and processing user input effectively.