

Experiment 4

Aim: To create an interactive Form using form widget

Theory:

Forms are essential in mobile applications for collecting user input, such as text fields, checkboxes, radio buttons, and dropdown selections. Flutter provides the Form widget along with form field widgets like TextFormField, DropdownButtonFormField, and CheckboxListTile to create interactive and validated forms.

Key Components

Form Widget: Acts as a container for multiple form fields and manages their state.

FormField Widgets: These are input fields like TextFormField for text input, DropdownButtonFormField for selections, etc.

GlobalKey<FormState>: Used to validate and save form data.

Validation: Ensures correct user input by checking conditions (e.g., non-empty fields, email format).

Form Validation

- validator Property: Each FormField can have a validator function that returns an error message if input is invalid.
- FormState.validate(): Checks all validators in the form.
- FormState.save(): Saves form data when submitted.

Form Submission

- A button (e.g., ElevatedButton) triggers form submission.
- On submission, formKey.currentState.validate() checks all fields.
- If validation passes, formKey.currentState.save() stores the data.

Interactive Features

- Dynamic Fields: Adding/removing fields based on user actions.
- Real-time Validation: Providing feedback as the user types.
- Custom Input Controls: Using switches, sliders, or date pickers within forms.

State Management in Forms

- Forms can use local state (setState) for simple apps.
- For complex forms, state management solutions like Provider or Bloc may be used.

Code:**1. Form creation with global key and form field validation**

```
// At the top of _AuthPageState class
final _formKey = GlobalKey<FormState>();

// In build method (complete Form widget)
Form(
  key: _formKey,
  child: Column(
    mainAxisAlignment: MainAxisAlignment.min,
    children: [
      if (!isLogin) ...[
        TextFormField(
          controller: nameController,
          decoration: InputDecoration(
            labelText: 'Full Name',
            prefixIcon: const Icon(Icons.person),
            border: OutlineInputBorder(
              borderRadius: BorderRadius.circular(8),
            ),
          ),
          validator: (value) =>
            (value == null || value.isEmpty)
              ? 'Please enter your name'
              : null,
        ),
        const SizedBox(height: 16),
      ],
      TextFormField(
        controller: emailController,
        decoration: InputDecoration(
          labelText: 'Email',
          prefixIcon: const Icon(Icons.email),
          border: OutlineInputBorder(
            borderRadius: BorderRadius.circular(8),
          ),
        ),
        validator: (value) =>
          (value == null || !value.contains('@'))
            ? 'Please enter a valid email'
            : null,
      ),
      const SizedBox(height: 16),
      TextFormField(
        controller: passwordController,
        obscureText: true,
        decoration: InputDecoration(
          labelText: 'Password',
```

```

    prefixIcon: const Icon(Icons.lock),
    border: OutlineInputBorder(
      borderRadius: BorderRadius.circular(8),
    ),
  ),
  validator: (value) =>
    (value == null || value.length < 6)
    ? 'Password must be at least 6 characters'
    : null,
),
if (!isLogin) ...[
  const SizedBox(height: 16),
  TextFormField(
    controller: favTeamController,
    decoration: InputDecoration(
      labelText: 'Favorite Team (optional)',
      prefixIcon: const Icon(Icons.sports_basketball),
      border: OutlineInputBorder(
        borderRadius: BorderRadius.circular(8),
      ),
    ),
  ),
),
],
const SizedBox(height: 24),
if (errorMsg != null)
  Padding(
    padding: const EdgeInsets.only(bottom: 16),
    child: Text(
      errorMsg!,
      style: const TextStyle(
        color: Colors.red,
        fontSize: 14,
      ),
      textAlign: TextAlign.center,
    ),
  ),
SizedBox(
  width: double.infinity,
  height: 50,
  child: ElevatedButton(
    onPressed: _isLoading ? null : _authenticate,
    style: ElevatedButton.styleFrom(
      backgroundColor: Colors.blue[800],
      shape: RoundedRectangleBorder(
        borderRadius: BorderRadius.circular(8),
      ),
    ),
    elevation: 2,
  ),
  child: _isLoading

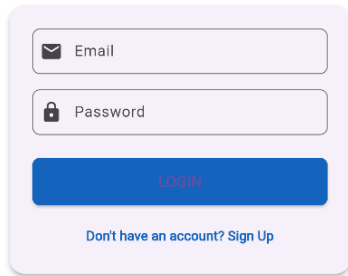
```

```

      ? const CircularProgressIndicator(
        color: Colors.white,
      )
      : Text(
        isLogin ? 'LOGIN' : 'SIGN UP',
        style: const TextStyle(
          fontSize: 16,
          fontWeight: FontWeight.bold,
        ),
      ),
    ),
  ),
  const SizedBox(height: 16),
  TextButton(
    onPressed: _isLoading
      ? null
      : () {
        setState(() {
          isLogin = !isLogin;
        });
      },
    child: Text(
      isLogin
        ? "Don't have an account? Sign Up"
        : "Already have an account? Login",
      style: TextStyle(
        color: Colors.blue[800],
        fontWeight: FontWeight.w600,
      ),
    ),
  ),
],
),
),

```

Welcome Back

A login form with a light purple background. It contains two input fields: 'Email' with an envelope icon and 'Password' with a lock icon. Below these is a blue 'LOGIN' button. At the bottom, there is a link that says 'Don't have an account? Sign Up'.

[Forgot Password?](#)



2. Form submission

// Submit button

```
SizedBox(  
  width: double.infinity,  
  height: 50,  
  child: ElevatedButton(  
    onPressed: _isLoading ? null : _authenticate,  
    style: ElevatedButton.styleFrom(  
      backgroundColor: Colors.blue[800],  
      shape: RoundedRectangleBorder(  
        borderRadius: BorderRadius.circular(8),  
      ),  
      elevation: 2,  
    ),  
    child: _isLoading  
      ? const CircularProgressIndicator(  
        color: Colors.white,  
      )  
      : Text(  
        isLogin ? 'LOGIN' : 'SIGN UP',  
        style: const TextStyle(  
          fontSize: 16,  
        ),  
      ),  
  ),  
)
```

```
        fontWeight: FontWeight.bold,
      ),
    ),
  ),
),

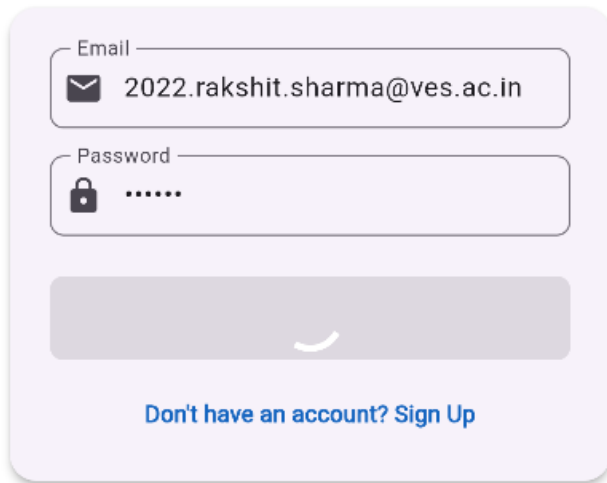
// Complete authentication function
Future<void> _authenticate() async {
  if (!_formKey.currentState!.validate()) return;

  setState(() {
    _isLoading = true;
    errorMsg = null;
  });

  try {
    if (isLogin) {
      await _auth.signInWithEmailAndPassword(
        email: emailController.text.trim(),
        password: passwordController.text.trim(),
      );
    } else {
      final cred = await _auth.createUserWithEmailAndPassword(
        email: emailController.text.trim(),
        password: passwordController.text.trim(),
      );

      await _firestore.collection('users').doc(cred.user!.uid).set({
        'name': nameController.text.trim(),
        'email': emailController.text.trim(),
        'favTeam': favTeamController.text.trim(),
        'createdAt': FieldValue.serverTimestamp(),
      }, SetOptions(merge: true));
    }
  } on FirebaseAuthException catch (e) {
    setState(() {
      errorMsg = 'Authentication Error: ${e.message}';
    });
  } catch (e) {
    setState(() {
      errorMsg = 'Unexpected Error: $e';
    });
  } finally {
    if (mounted) {
      setState(() {
        _isLoading = false;
      });
    }
  }
}
```

```
}
```



Forgot Password?

3. Form toggle

```
TextButton(  
  onPressed: _isLoading  
    ? null  
    : () {  
      setState(() {  
        isLogin = !isLogin;  
      });  
    },  
  child: Text(  
    isLogin  
      ? "Don't have an account? Sign Up"  
      : "Already have an account? Login",  
    style: TextStyle(  
      color: Colors.blue[800],  
      fontWeight: FontWeight.w600,  
    ),  
  ),  
),
```

The diagram illustrates the transition between two user interface forms. On the left is a login form with fields for 'Email' and 'Password', a blue 'LOGIN' button, and a link 'Don't have an account? Sign Up'. Below the login form is a link 'Forgot Password?'. A large black arrow points from the login form to the sign-up form on the right. The sign-up form includes fields for 'Full Name', 'Email', and 'Password', an optional 'Favorite Team' field with a team icon, a blue 'SIGN UP' button, and a link 'Already have an account? Login'.

Github link: <https://github.com/Rakshit5467/NBA-India>

Conclusion:

This experiment successfully demonstrated the creation of an interactive form in Flutter using the Form widget, form validation, and submission handling. By implementing TextFormField with validators, a toggle between login and sign-up modes, and Firebase authentication integration, we built a functional and user-friendly form that ensures data integrity and provides real-time feedback. This approach can be extended to more complex forms with additional input fields and advanced state management techniques.