MPL Experiment 04

Prajjwal Pandey

D15A / 32

Aim: To Create an Interactive Form Using the Form Widget

Description of Form Pages:

1. Login Page:

- **Purpose:** Allows users to enter credentials and log in.
- **Widgets Used:**
  - `Form`: Wraps the input fields to enable validation.
  - `TextFormField`: For entering username and password.
  - `ElevatedButton`: For submitting the login form.
  - `InkWell`: For navigating to the sign-up page.
  - `Snackbar`: Displays validation messages.

2. Signup Page:

- **Purpose:** Allows new users to register by entering personal details.
- **Widgets Used:**
  - `Form`: Manages input validation.
  - `TextFormField`: Fields for username, email, password, and phone number.
  - `DropdownButtonFormField`: For selecting gender.
  - `CheckboxListTile`: For accepting terms and conditions.
  - `ElevatedButton`: Submits the form after validation.
  - `Navigator.push`: Redirects users after successful sign-up.

3. Profile Update Page:

- **Purpose:** Allows users to edit their profile details.
- **Widgets Used:**
  - `Form`: Ensures structured data entry.
  - `TextFormField`: For updating username, email, and phone number.
  - `ImagePicker`: Enables users to upload a profile picture.
  - `DropdownButtonFormField`: Lets users update their preferences.
  - `SwitchListTile`: Toggles notification preferences.

- `ElevatedButton`: Saves updates.

Key Features & Functionalities Implemented:

- **Validation:**
  - Used `validator` property in `TextFormField` to validate empty fields and input formats.
  - Displayed error messages using `SnackBar` or inline text.
- **State Management:**
  - Used `GlobalKey<FormState>` to validate and submit forms.
- **Navigation Between Forms:**
  - Utilized `Navigator.push` and `Navigator.pop` for seamless transitions.
- **User Experience Enhancements:**
  - Added proper padding, margins, and icons for better UI.
  - Included loaders and visual feedback on button presses.

CODE :

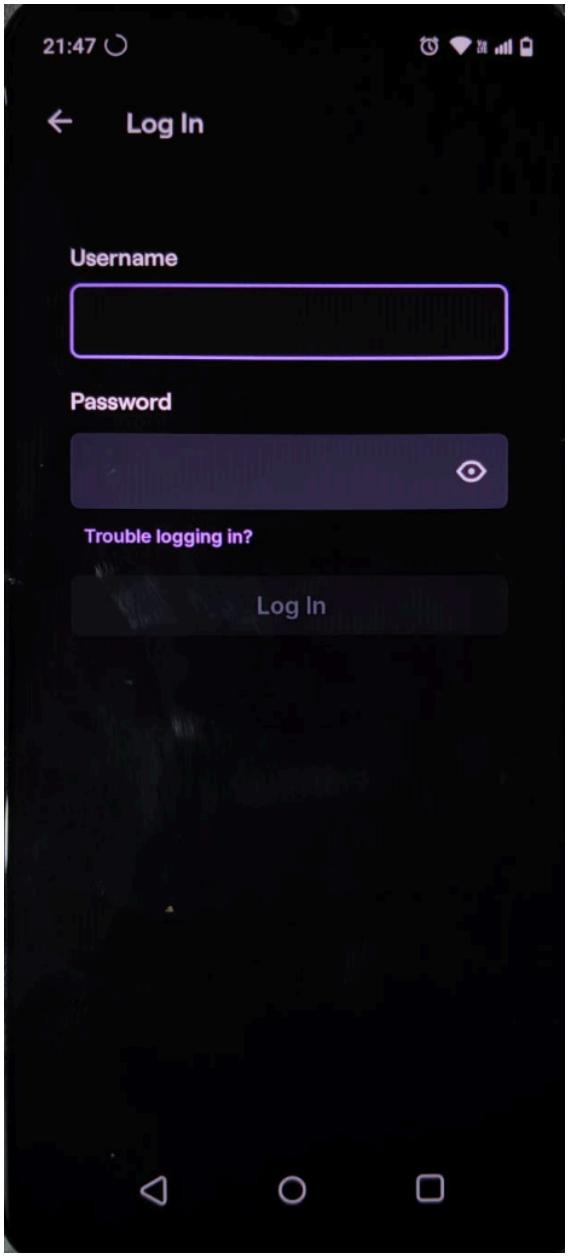
```
class LoginPage extends StatefulWidget {
  final VoidCallback onLoginSuccess;

  LoginPage({required this.onLoginSuccess});

  @override
  _LoginPageState createState() => _LoginPageState();
}

class _LoginPageState extends State<LoginPage> {
  final TextEditingController _emailController = TextEditingController();
  final TextEditingController _passwordController =
TextEditingController();
  final FirebaseAuth _auth = FirebaseAuth.instance;

  Future<void> _login() async {
    try {
      await _auth.signInWithEmailAndPassword(
        email: _emailController.text.trim(),
        password: _passwordController.text.trim(),
      );
      widget.onLoginSuccess();
```

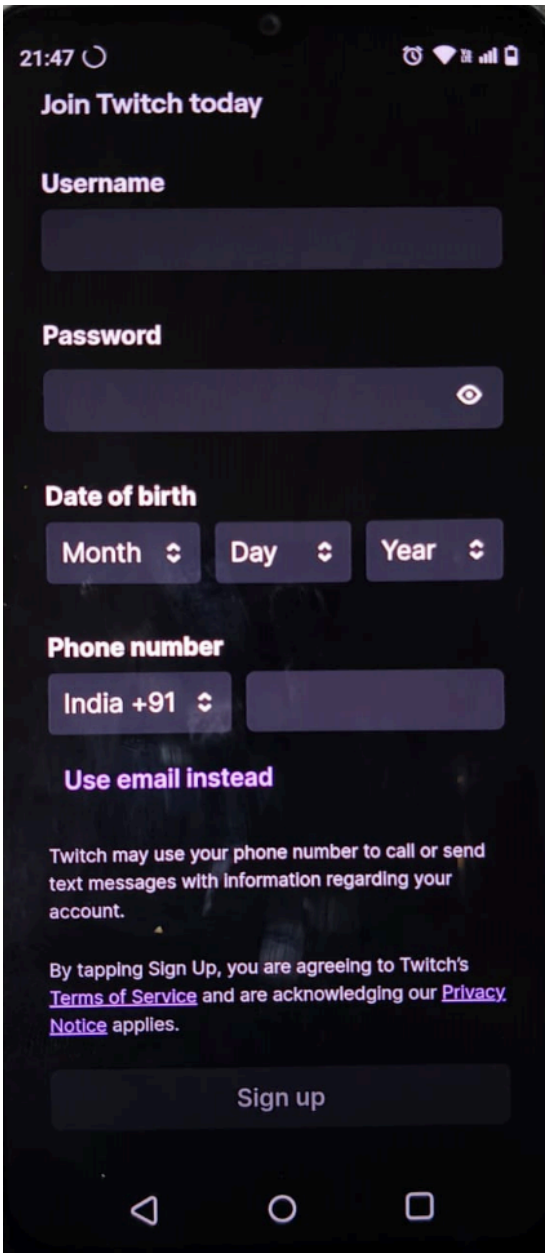
```
    } catch (e) {
      print("Login failed: $e");
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text("Login failed: ${e.toString()}")),
      );
    }
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text("Log In")),
      body: Padding(
        padding: EdgeInsets.all(20),
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            TextField(
              controller: _emailController,
              decoration: InputDecoration(labelText: "Email"),
            ),
            TextField(
              controller: _passwordController,
              decoration: InputDecoration(labelText: "Password"),
              obscureText: true,
            ),
            SizedBox(height: 20),
            ElevatedButton(
              onPressed: _login,
              child: Text("Log In"),
            ),
          ],
        ),
      ),
    );
  }
}

class SignupPage extends StatefulWidget {
  final VoidCallback onSignupSuccess;
```

```
  SignupPage({required this.onSignupSuccess});

  @override
  _SignupPageState createState() => _SignupPageState();
}

class _SignupPageState extends State<SignupPage> {
  final TextEditingController _emailController = TextEditingController();
  final TextEditingController _passwordController =
TextEditingController();
  final FirebaseAuth _auth = FirebaseAuth.instance;

  Future<void> _signup() async {
    try {
      await _auth.createUserWithEmailAndPassword(
        email: _emailController.text.trim(),
        password: _passwordController.text.trim(),
      );
      widget.onSignupSuccess();
    } catch (e) {
      print("Signup failed: $e");
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text("Signup failed: ${e.toString()}")),
      );
    }
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text("Sign Up")),
      body: Padding(
        padding: EdgeInsets.all(20),
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            TextField(
              controller: _emailController,
              decoration: InputDecoration(labelText: "Email"),
```

```
            ),
            TextField(
              controller: _passwordController,
              decoration: InputDecoration(labelText: "Password"),
              obscureText: true,
            ),
            SizedBox(height: 20),
            ElevatedButton(
              onPressed: _signup,
              child: Text("Sign Up"),
            ),
          ],
        ),
      ),
    );
  }
}
```

OUTPUT:


21:47
Log In
Username
Password
Trouble logging in?
Log In


21:47
Join Twitch today
Username
Password
Date of birth
Month
Day
Year
Phone number
India +91
Use email instead
Twitch may use your phone number to call or send text messages with information regarding your account.
By tapping Sign Up, you are agreeing to Twitch's Terms of Service and are acknowledging our Privacy Notice applies.
Sign up