# Flutter – First App – pt.3: Signup & Login

> ***Dopo avere realizzato lo scheletro della nostra prima applicazione Flutter, ed aver analizzato alcuni dei punti chiave, come le Stateless e le Stateful class, procediamo con lo sviluppo della nostra mini app!***

## Iniziamo

Ciò che andremo a sviluppare questa volta, saranno lo scheletro delle schermate di Login e di Registrazione.

**Step 1**: Riapriamo il nostro progetto ed eseguiamolo

Flutter run

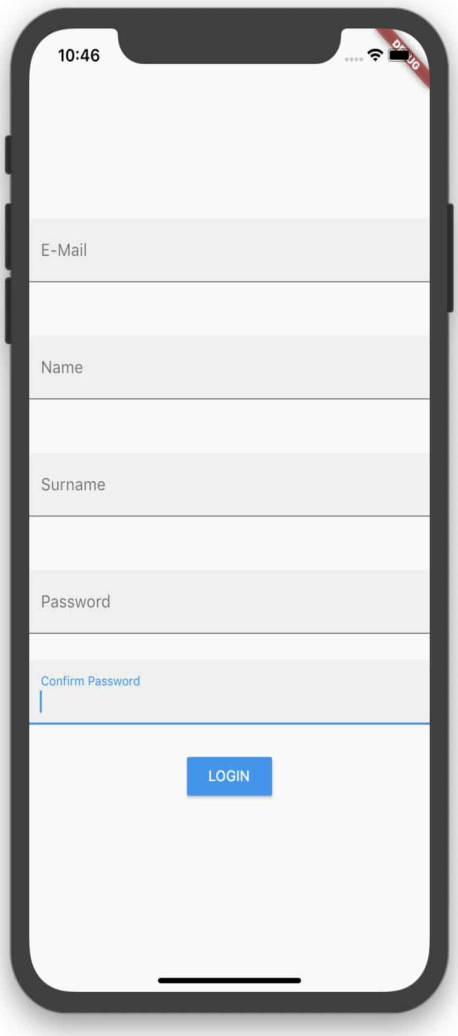
Una volta fatto andiamo a modificare il codice, aggiungendo altri campi di tipo **Textfield** che ci serviranno per la registrazione:

auth.dart

```
import 'package:flutter/material.dart'; class AuthPage extends StatefulWidget { @override State<StatefulWidget> createState() { return _AuthPageState(); }
} class _AuthPageState extends State<AuthPage> { Widget _buildEmailTextfield() { return TextFormField( decoration: InputDecoration(labelText: 'E-Mail', filled: true), keyboardType: TextInputType.emailAddress, ); } Widget _buildPasswordTextfield() { return TextFormField( decoration: InputDecoration(labelText: 'Password', filled: true), obscureText: true, ); } Widget _buildNameTextField() { return TextFormField( decoration: InputDecoration(labelText: 'Name', filled: true), keyboardType: TextInputType.text, ); } Widget _buildSurnameTextField() { return TextFormField( decoration: InputDecoration(labelText: 'Surname', filled: true), ); } Widget _buildPasswordConfirmTextField() { return
```

```
TextFormField( decoration: InputDecoration(labelText: 'Confirm Password', filled: true), obscureText: true, ); } Widget _buildButton() { return MaterialButton( color: Colors.blue, textColor: Colors.white, child: Text('LOGIN'), onPressed: () => {}); } @override Widget build(BuildContext context) { return Scaffold( body: Flex( direction: Axis.vertical, mainAxisAlignment: MainAxisAlignment.center, children: <Widget>[ _buildEmailTextfield(), SizedBox( height: 50.0, ), _buildNameTextField(), SizedBox( height: 50.0, ), _buildSurnameTextField(), SizedBox( height: 50.0, ), _buildPasswordTextfield(), SizedBox( height: 25.0, ), _buildPasswordConfirmTextField(), SizedBox( height: 25.0, ), _buildButton(), ], ), ); }
}
```

Ed ecco il risultato:




**Step 2**: fatto questo, come potrete notare non abbiamo alcun

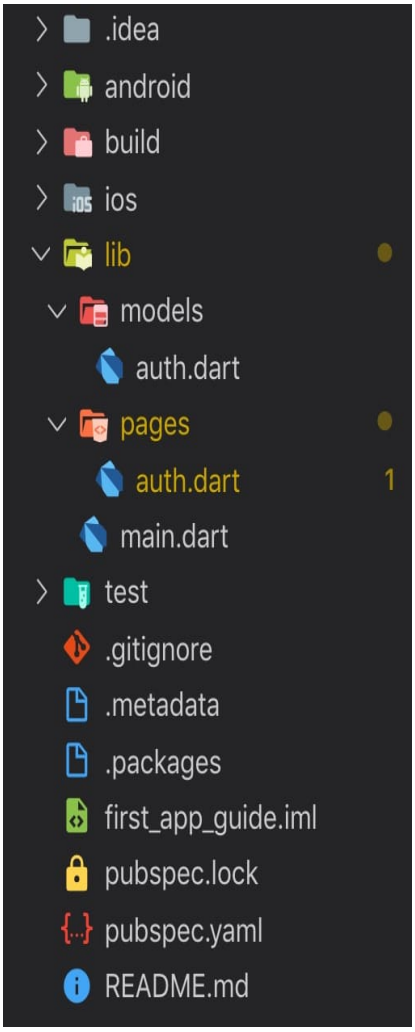
modo per distinguere i campi necessari per la login da quelli per la registrazione! Come possiamo risolvere? Semplicemente aggiungendo un **FlatButton** che al click ci permetterà di mostrare i rispettivi campi specifici per la **Login** e per la **Registrazione**! Procediamo:

auth.dart

```
....
.... FlatButton _buildFlatButton() { return FlatButton( child: Text('Switch Login / Signup'), ); } @override Widget build(BuildContext context) { return Scaffold( body: Flex( direction: Axis.vertical, mainAxisAlignment: MainAxisAlignment.center, children: <Widget>[ _buildEmailTextfield(), SizedBox( height: 50.0, ), _buildNameTextField(), SizedBox( height: 50.0, ), _buildSurnameTextField(), SizedBox( height: 50.0, ), _buildPasswordTextfield(), SizedBox( height: 25.0, ), _buildPasswordConfirmTextField(), SizedBox( height: 25.0, ), _buildFlatButton(), SizedBox( height: 25.0, ), _buildButton(), ], ), ); }
....
....
```

Fatto questo, aggiungiamo una cartella '**models**' in cui inseriremo i **Modelli** per i dati che andremo ad utilizzare. In questo primo caso, andremo ad inserire un file che chiameremo **auth.dart**, in cui dichiareremo un **Enumerativo** per switchare da Login a Registrazione al click del **button**.

La struttura del nostro progetto dovrebbe quindi assumere questa forma:




Ora aggiungiamo il seguente codice nel file auth.dart:

```
enum AuthMode { Signup, Login
}
```

**Step 3**: sfruttiamo l'operatore ternario per modificare la nostra pagina e rendere possibile la distinzione tra login e registrazione al click del button, ma per farlo avremo bisogno di introdurre un altro particolare costrutto Flutter: il **setState()**.

## Cos'è il setState()

Per dirla in due parole: **il setState() notifica al framework che lo stato interno di un oggetto è cambiato**.

```
setState(() { _myState = newValue });
```

Se si modificasse semplicemente lo stato di un oggetto senza

chiamare setState, il framework potrebbe non effettuare una build e l'interfaccia utente potrebbe non aggiornarsi.

Chiamando invece il setState, viene notificato al framework che lo stato interno di un oggetto è stato modificato. Ciò influisce sull'interfaccia utente e questo fa sì che il framework effettui nuovamente una **build**.

Generalmente si consiglia di utilizzare il metodo setState solo per racchiudere le modifiche allo stato e non i calcoli che potrebbero essere associati alla modifica.

**Step 4**: adesso che abbiamo chiaro cos'è il setState(), andiamo ad implementarlo nella nostra app:

pages/auth.dart

```
import 'package:flutter/material.dart'; import '../models/auth.dart';
import '../models/auth.dart'; class AuthPage extends StatefulWidget { @override State<StatefulWidget> createState() { return _AuthPageState(); }
} class _AuthPageState extends State<AuthPage> { AuthMode _authMode = AuthMode.Login; Widget _buildEmailTextfield() { return TextFormField( decoration: InputDecoration(labelText: 'E-Mail', filled: true), keyboardType: TextInputType.emailAddress, ); } Widget _buildPasswordTextfield() { return TextFormField( decoration: InputDecoration(labelText: 'Password', filled: true), obscureText: true, ); } Widget _buildNameTextField() { return TextFormField( decoration: InputDecoration(labelText: 'Name', filled: true), keyboardType: TextInputType.text, ); } Widget _buildSurnameTextField() { return TextFormField( decoration: InputDecoration(labelText: 'Surname', filled: true), ); } Widget _buildPasswordConfirmTextField() { return TextFormField( decoration: InputDecoration(labelText: 'Confirm Password', filled: true), obscureText: true, ); } Widget _buildButton() { return MaterialButton( color: Colors.blue, textColor: Colors.white, child: Text('LOGIN'), onPressed: () => {}); } FlatButton _buildFlatButton() { return FlatButton( child: Text('Switch to ${_authMode == AuthMode.Login ?
```

'Signup' : 'Login'}'), onPressed: () { setState(() { _authMode = _authMode == AuthMode.Login ? AuthMode.Signup : AuthMode.Login; }); }, ); } @override Widget build(BuildContext context) { return Scaffold( body: Flex( direction: Axis.vertical, mainAxisAlignment: MainAxisAlignment.center, children: <Widget>[ _buildEmailTextfield(), SizedBox( height: 50.0, ), _authMode == AuthMode.Signup ? _buildNameTextField() : Container(), _authMode == AuthMode.Signup ? SizedBox( height: 50.0, ) : Container(), _authMode == AuthMode.Signup ? _buildSurnameTextField() : Container(), _authMode == AuthMode.Signup ? SizedBox( height: 50.0, ) : Container(), _buildPasswordTextfield(), SizedBox( height: 25.0, ), _authMode == AuthMode.Signup ? _buildPasswordConfirmTextField() : Container(), _authMode == AuthMode.Signup ? SizedBox( height: 25.0, ) : Container(), _buildFlatButton(), SizedBox( height: 25.0, ), _buildButton(), ], ), ); }
}

Ed ecco che finalmente potremo utilizzare il button per switchare tra modalità login e registrazione! Semplice no?

Bene, per ora ci fermiamo qui. Nella prossima guida vedremo un approfondimento su come disegnare UI e UX per questa applicazione (dato che al momento è parecchio bruttina!) e successivamente le andremo ad implementare nel nostro codice Flutter. Esistono diversi tool che permettono di disegnare senza particolari competenze grafiche, personalmente mi trovo molto bene con **Figma**, un tool online per lo sviluppo in team di design Web & Mobile.

Per concludere, ecco il **LINK** del repository github di questo progetto, vi invito a seguirmi anche lì! Alla prossima, stay tuned!

https://italiancoders.it/flutter-first-app-pt-3-signup-login/