



Empowering Digital Skills For The Jobs Of The Future



by



Form

Sommario

- Creare moduli in Flutter
- Input di testo, dropdown e altri elementi dei form
- Usare i tasti per gestire i form di Flutter

Docente



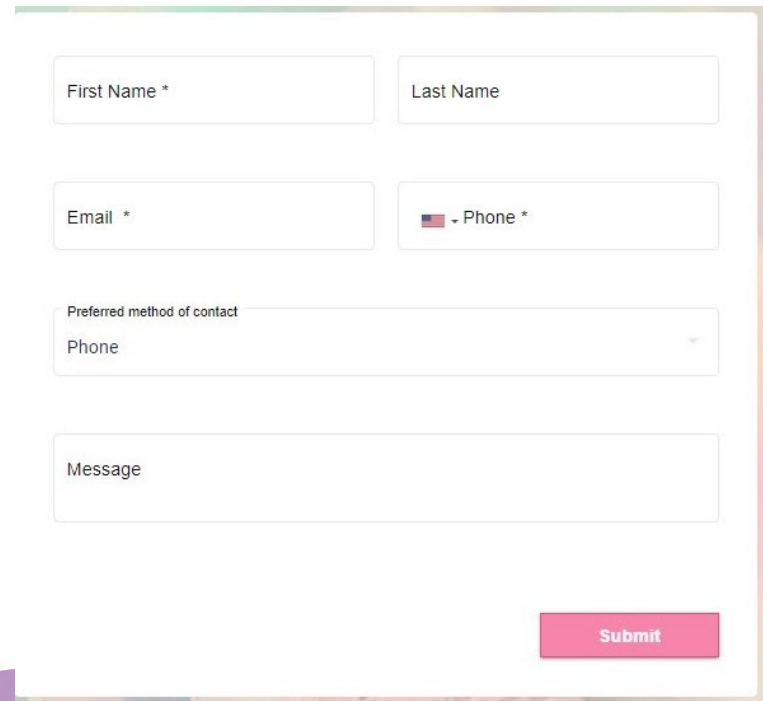
Claudia Infante



claudia.infante@bcsoft.net

Modulo web

Un modulo web è costituito da un insieme di campi di vario tipo, con cui l'utente può interagire per inserire delle informazioni.



A screenshot of a web form with the following fields:

- First Name *
- Last Name
- Email *
- Phone *
- Preferred method of contact (Dropdown menu with "Phone" selected)
- Message
- Submit button

Modulo web

I form di Flutter sono utilizzati per raccogliere e convalidare i dati inseriti dall'utente in un'applicazione. Un form in Flutter è costituito da un Widget di tipo Form che contiene una serie di campi di inserimento dati di tipo TextFormField, che rappresentano i campi in cui l'utente inserisce i propri dati.

Modulo web

Attraverso i `TextFormField` è possibile definire regole di validazione per i dati inseriti, come ad esempio la lunghezza minima o massima del testo, la presenza di determinati caratteri o il formato corretto per un'email.

Modulo web

Una volta che l'utente ha compilato tutti i campi del form, è possibile invocare il metodo `save()` del Form per recuperare i dati inseriti e processarli in base alle necessità dell'applicazione.

Modulo web

I form di Flutter offrono anche la possibilità di gestire la presentazione di messaggi di errore in caso di dati non validi inseriti dall'utente, rendendo l'esperienza utente più intuitiva e facile da utilizzare.

Modulo web

I form widget in flutter gestiscono la UI, inclusi i campi in cui l'utente inserisce i dati e implementano la loro collezione al verificarsi del submit.

Modulo web

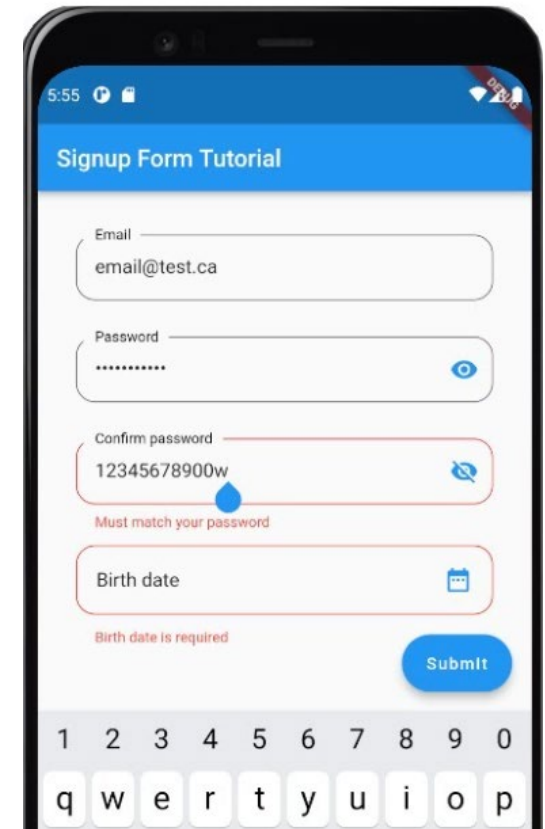
Si assicurano che i dati siano validi ed utilizzabili. Inoltre passano i dati alla logica di business.

Form widget

Un form widget è un contenitore che fornisce una serie di metodi utili e gestisce tutti i campi di input.

Form widget

Il modo in cui flutter consente l'interazione con i form è passando una chiave di tipo `FormState`. Il widget associato alla chiave globale dello stato del form dà accesso allo stato dell'oggetto.



GlobalKey<FormState>

L'uso di una chiave del form (che in realtà è una chiave globale del sottotipo FormState) è molto simile all'uso di un controllore su molti altri widget.

GlobalKey<FormState>

FormState è una classe incorporata che fornisce una serie di metodi utili per mantenere la logica del modulo come `FormState.reset`, `FormState.save` e `FormState.validate`.

GlobalKey<FormState>

Quando si lavora con i moduli, è comune utilizzare le chiavi, che possono fornire un riferimento all'oggetto `FormState`. Tutta la logica e le proprietà presenti nell'oggetto `FormState` sono accessibili tramite la chiave creata, il che significa che è possibile interagire con essa in tutti i widget presenti nel form, compresi i figli.

FormField widgets

Ci sono tre tipi di FormField widget:

- FormField: il campo standard, che può trasformare qualsiasi widget di input in un campo del modulo.
- TextFormField: un campo modulo specializzato che avvolge un campo di testo.
- DropdownButtonFormField: un widget pratico che racchiude un menu a discesa in un campo del modulo.

TextFormField

Si parte con la definizione della chiave del form e con la creazione di una variabile per valutare i dati inseriti:

```
main.dart X
lib > main.dart > main
1  import 'package:flutter/material.dart';
2
   Run | Debug | Profile
3  void main() {
4      runApp(
5          const MaterialApp(
6              home: MyPage(),
7          ); // MaterialApp
8      }
9
10 final _myKey = GlobalKey<FormState>();
11 final _textController = TextEditingController();
12
13 class MyPage extends StatelessWidget {
14     const MyPage({Key? key}) : super(key: key);
15 }
```

TextFormField

Si definisce la struttura del layout e si inserisce il widget form con la relativa chiave. Si definisce poi il widget TextFormField con il relativo controllo:

```
16  @override
17  Widget build(BuildContext context) {
18    return Scaffold(
19      appBar: AppBar(
20        title: const Text('Form Example'),
21      ), // AppBar
22      body: Center(
23        child: Padding(
24          padding: const EdgeInsets.all(16.0),
25          child: Form(
26            key: _myKey,
27            child: Column(
28              children: [
29                TextFormField(
30                  controller: _textController,
31                  validator: (value) {
32                    if (value == null || value.isEmpty) {
33                      return 'Campo obbligatorio';
34                    }
35                    return null;
36                  },
37                ), // TextFormField
```

TextFormField

Si definisce poi il bottone e la logica desiderata. In questo caso al submit corretto sarà mostrato un alert con i valori del campo :

```
38 ElevatedButton(  
39   onPressed: () {  
40     if (_myKey.currentState!.validate()) {  
41       _myKey.currentState!.save();  
42       showDialog(  
43         context: context,  
44         builder: (context) {  
45           return AlertDialog(  
46             title: const Text('Form inviato con successo'),  
47             content: Text('Value: ${_textController.text}'),  
48             actions: <Widget>[  
49               TextButton(  
50                 onPressed: () {  
51                   Navigator.of(context).pop();  
52                 },  
53                 child: const Text('OK'),  
54               ), // TextButton  
55             ], // <Widget>[]  
56           ); // AlertDialog  
57         },  
58       );  
59     },  
60   ),  
61   child: const Text('Invia'),  
62 ), // ElevatedButton  
63 ],
```

Flutter - C. Infante

DropDownButtonFormField

È possibile aggiungere anche altri campi, come il dropdown. Il campo leggerà dei valori da una variabile e l'utente avrà la possibilità di selezionare una tra le opzioni dal menù a tendina.

```
9   final _myKey = GlobalKey<FormState>();
10  final _textController = TextEditingController();
11  String _selectedItem = 'Opzione 1'; // Voce di default
12
13  class MyPage extends StatelessWidget {
14    const MyPage({Key? key}) : super(key: key);
15
16    @override
17    Widget build(BuildContext context) {
18      List<String> dropdownItems = ['Opzione 1', 'Opzione 2', 'Opzione 3'];
19    }
20  }
```

DropDownButtonFormField

```
DropDownButtonFormField<String>(  
  value: _selectedItem,  
  items: dropdownItems.map((String item) {  
    return DropdownMenuItem<String>(  
      value: item,  
      child: Text(item),  
    ); // DropdownMenuItem  
  }).toList(),  
  onChanged: (String? value) {  
    if (value != null) {  
      _selectedItem = value;  
    }  
  },  
) // DropDownButtonFormField
```

Utilizzo del map per iterare tra gli elementi, ognuno dei quali sarà una voce del Dropdown.

DropDownButtonFormField

```
ElevatedButton(  
  onPressed: () {  
    if (_myKey.currentState!.validate()) {  
      _myKey.currentState!.save();  
      showDialog(  
        context: context,  
        builder: (context) {  
          return AlertDialog(  
            title: const Text('Form inviato con successo'),  
            content: Column(  
              crossAxisAlignment: CrossAxisAlignment.start,  
              mainAxisAlignment: MainAxisAlignment.min,  
              children: [  
                Text(  
                  'Text Field Value: ${_textController.text}', // Text  
                ),  
                Text('Dropdown Value: $_selectedItem'),  
              ],  
            ), // Column  
            actions: <Widget>[  
              TextButton(  
                onPressed: () {  
                  Navigator.of(context).pop();  
                },  
                child: const Text('OK'),  
              ), // TextButton  
            ], // <Widget>[]  
          ); // AlertDialog  
        },  
      );  
    }  
  });  
);
```

Alert aggiornato

GenericFormField

Se si desidera utilizzare campi di input di altro tipo, come checkbox, radio button, input date picker, è possibile inserirli in qualsiasi FormField widget.

GenericFormField : checkbox

Ad esempio, è possibile inserire una checkbox. In questo caso, il widget creato dovrà essere di tipo `stateful` in modo da poter gestire lo stato della checkbox.

```
13
14 class MyPage extends StatefulWidget {
15   const MyPage({Key? key}) : super(key: key);
16   @override
17   MyPage createState() => _MyPage();
18 }
19 class _MyPage extends State<MyPage> {
20   bool _saveValues = false; //valore della checkbox
21
22   @override
```

GenericFormField : checkbox

```
Row(  
  children: [  
    Checkbox(  
      value: _saveValues,  
      onChanged: (value) {  
        setState(() {  
          _saveValues = value!;  
        });  
      },  
    ), // Checkbox  
    const Text('Salvare i dati?'),  
  ],  
) // Row
```

La checkbox richiamerà `setState()` per passare da uno stato all'altro e consentire l'interazione.

GenericFormField : radio

Si determina il valore iniziale del campo

```
18 }  
19 class _MyPage extends State<MyPage> {  
20   bool _saveValues = false; //valore della checkbox  
21   String _radioValue = ''; // Default radio value  
22 }
```

GenericFormField : radio

```
Row(  
  children: [  
    Radio<String>(  
      value: 'Yes',  
      groupValue: _radioValue,  
      onChanged: (value) {  
        setState(() {  
          _radioValue = value!;  
        });  
      },  
    ), // Radio  
    const Text('Yes'),  
  ],  
) // Row
```

```
Row(  
  children: [  
    Radio<String>(  
      value: 'No',  
      groupValue: _radioValue,  
      onChanged: (value) {  
        setState(() {  
          _radioValue = value!;  
        });  
      },  
    ), // Radio  
    const Text('No'),  
  ],  
) // Row
```

I campi sono legati dalla proprietà `groupValue` che ne identifica l'appartenenza.

GenericFormField : DatePicker

Il campo per la scelta della data è uno tra i campi più utilizzati nelle applicazioni mobile.

GenericFormField : DatePicker

Si parte dalla creazione di un TextField, cliccato il quale apparirà il calendario con la data selezionabile.

```
65     TextField(  
66         controller: _dateController,  
67         decoration: InputDecoration(  
68             labelText: 'Seleziona una data',  
69             prefixIcon: IconButton(  
70                 icon: const Icon(Icons.calendar_today),  
71                 onPressed: () => _selectDate(context),  
72             ), // IconButton  
73         ), // InputDecoration  
74         readOnly: true,  
75     ), // TextField
```

GenericFormField : DatePicker

Si definisce la variabile controller che gestisce il campo e una variabile per impostare una data di partenza corrispondente all'attuale.

```
20 class _MyPage extends State<MyPage> {  
21   bool _saveValues = false; //valore della checkbox  
22   String _radioValue = ''; // Default radio value  
23   final _dateController = TextEditingController(); //controller campo data  
24   DateTime selectDate = DateTime.now(); //recupero data attuale  
25 }
```

GenericFormField : DatePicker

Si andrà a definire un metodo asincrono che attenderà la scelta della data per la valorizzazione del campo.

```
Future<void> _selectDate(BuildContext context) async {  
  final DateTime? picked = await showDatePicker(  
    context: context,  
    initialDate: DateTime.now(),  
    firstDate: DateTime(2015),  
    lastDate: DateTime(2025),  
  );  
  if (picked != null) {  
    setState(() {  
      _dateController.text = picked.toString();  
    });  
  }  
}
```

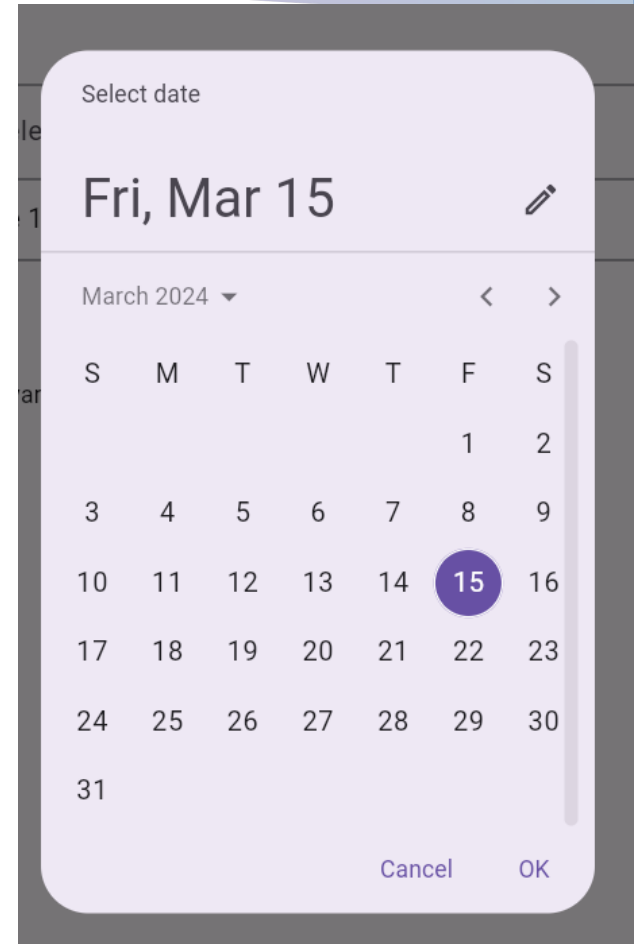

GenericFormField : DatePicker

Cliccando sull'icona del calendario apparirà un pop up con il calendario. Cliccando su ok, il giorno selezionato sarà poi mostrato nel campo di input.



Seleziona una data

2024-03-15 00:00:00.000



FormBuilder

FormBuilder è un pacchetto che facilita la creazione dei form consentendo la creazione di diversi campi di input, di ottenere i valori in modo semplice, di applicare dei validatori e di gestire il form in modo reattivo.

FormBuilder

I campi supportati sono :

- FormBuilderCheckbox
- FormBuilderCheckboxGroup
- FormBuilderChoiceChip
- FormBuilderDateTimePicker
- FormBuilderDropdown

FormBuilder

- FormBuilderFilterChip
- FormBuilderRadioGroup
- FormBuilderRangeSlider
- FormBuilderSlider
- FormBuilderSwitch
- FormBuilderTextField
- FormBuilderDateRangePicker

FormBuilder

Per utilizzarlo, installare le dipendenze

```
dependencies:  
  flutter:  
    sdk: flutter  
  cupertino_icons: ^1.0.2  
  intl: ^0.18.1  
  flutter_form_builder: ^9.2.1  
  form_builder_validators: ^9.1.0
```

FormBuilder

Da questo momento è possibile definire una chiave globale di tipo FormBuilderState

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

FormBuilder(
  key: _formKey,
  child: FormBuilderTextField(
    name: 'text',
    onChanged: (val) {
      print(val); // Print the text value write into TextField
    },
  ),
)
```

FormBuilder

Esempio di un campo di input gestito con FormBuilder e FormValidator :

```
FormBuilderTextField(  
  name: 'username',  
  controller: _textController,  
  validator: FormBuilderValidators.required(),  
  decoration: const InputDecoration(labelText: 'Username'),  
  autofocus: true,  
), // FormBuilderTextField
```

FormBuilder

FormValidator offre delle validazioni built in tra cui :

- `FormBuilderValidators.date()` - richiede che il valore del campo sia una stringa di data .
- `FormBuilderValidators.email()` - richiede che il valore del campo sia un indirizzo e-mail valido.

FormBuilder

- `FormBuilderValidators.integer()` - richiede che il valore del campo sia un numero intero.
- `FormBuilderValidators.match()` - richiede che il valore del campo corrisponda al modello regex fornito.
- `FormBuilderValidators.max()` - richiede che il valore del campo sia inferiore o uguale al numero fornito.
- `FormBuilderValidators.maxLength()` - richiede che la lunghezza del valore del campo sia inferiore o uguale alla lunghezza massima fornita.

FormBuilder

- `FormBuilderValidators.min()` - richiede che il valore del campo sia maggiore o uguale al numero fornito.
- `FormBuilderValidators.minLength()` - richiede che la lunghezza del valore del campo sia maggiore o uguale alla lunghezza minima fornita.
- `FormBuilderValidators.numeric()` - richiede che il valore del campo sia un numero valido.
- `FormBuilderValidators.required()` - richiede che il campo abbia un valore non vuoto.
- `FormBuilderValidators.url()` - richiede che il valore del campo sia un url valido.

FormBuilder

È possibile aggiungere anche un messaggio personalizzato quando si verificano degli errori :

```
FormBuilderTextField(  
  name: 'username',  
  controller: _textController,  
  validator: FormBuilderValidators.required(errorText: 'Campo obbligatorio'),  
  decoration: const InputDecoration(  
    labelText: 'Username',  
  ), // InputDecoration  
) // FormBuilderTextField
```

FormBuilder

Così come si possono gestire più validatori per uno stesso campo:

```
FormBuilderTextField(  
  name: 'username',  
  controller: _textController,  
  validator: FormBuilderValidators.compose([  
    FormBuilderValidators.required(errorText: 'Campo obbligatorio'),  
    (value) {  
      if (value!.isEmpty) {  
        return 'Campo obbligatorio';  
      } else if (!value.contains(RegExp(r'\d'))) {  
        return 'Deve contenere almeno un numero';  
      }  
      return null;  
    },  
  ]),  
  decoration: const InputDecoration(  
    labelText: 'Username',  
  ), // InputDecoration  
), // FormBuilderTextField
```

Form styling

Tutti i campi di input hanno una proprietà chiamata `decoration`. La classe `InputDecoration` accetta vari argomenti che è possibile utilizzare per personalizzare lo stile dei campi del form.

```
controller: _dateController,  
decoration: InputDecoration(  
  labelText: 'Seleziona una data',  
  labelStyle: TextStyle(color: Colors.purple.shade700),  
  border: OutlineInputBorder(  
    borderRadius: BorderRadius.circular(10.0),  
  ), // OutlineInputBorder  
  focusedBorder: OutlineInputBorder(  
    borderSide: const BorderSide(color: Colors.purple, width: 2.0),  
    borderRadius: BorderRadius.circular(10.0),  
  ), // OutlineInputBorder  
  prefixIcon: IconButton(  
    icon: const Icon(Icons.calendar_today),  
    onPressed: () => _selectDate(context),  
  ), // IconButton  
), // InputDecoration
```

The background of the slide features a series of overlapping, wavy bands in various shades of blue and purple. These bands flow horizontally across the frame, creating a sense of movement and depth. The colors range from deep, dark blues and purples to lighter, almost white tones, with the most intense colors concentrated at the top and bottom edges, leaving a central white space for the text.

Domande e approfondimenti

Attività

Si crei un form costituito dai campi username, password, conferma password e un bottone per l'invio.

- Tutti i campi sono obbligatori.
- Email dovrà rispettare il formato richiesto.
- Password deve essere costituito da almeno 8 caratteri, un simbolo e un numero.
- Conferma password deve avere gli stessi valori inseriti nel campo password.
- Entrambi i campi dovranno avere un'icona per mostrare su richiesta i dati inseriti in modo non decriptato.
- Mostrare un messaggio di errore a seconda dell'errore incontrato. Se non ci sono errori, mostrare una snackbar di colore diverso rispetto al tema utilizzato contenente un messaggio e visibile per un tempo limitato.

Attività

Registration Form

Email

Campo obbligatorio

Password



Campo obbligatorio

Conferma Password

Le password non corrispondono

Registrati

Registration Form

Email
ab@cd.it

Campo obbligatorio

Password
abc



Campo obbligatorio

Conferma Password
...

Le password non corrispondono

Registrati

Registration Form

Email
ab@cd.it

Password
.....



Conferma Password
.....

Registrati

Form inviato