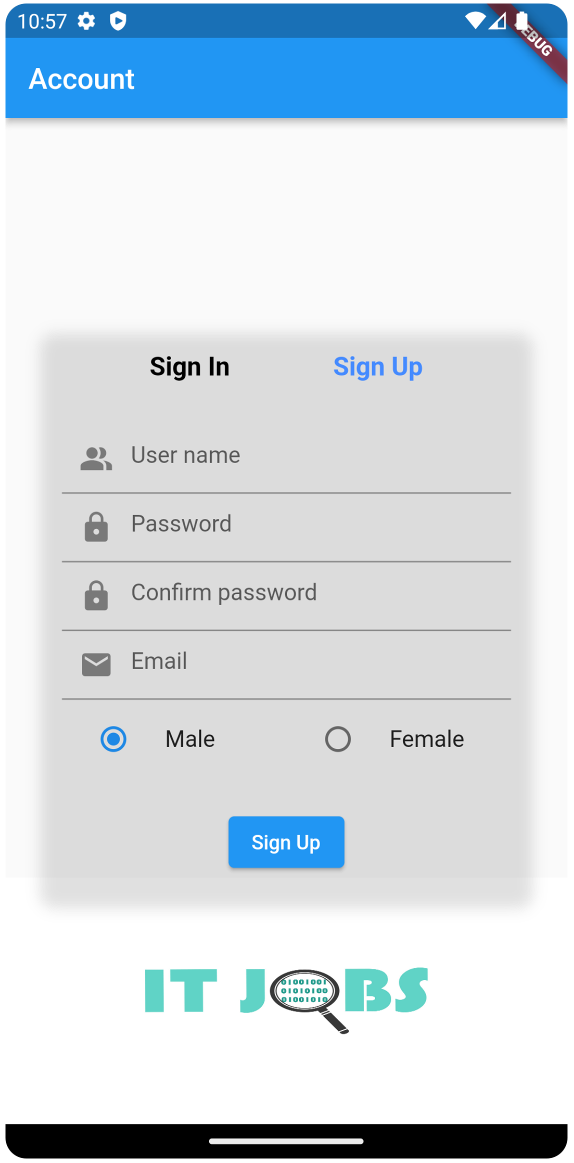
Resource Software Solution

Forms

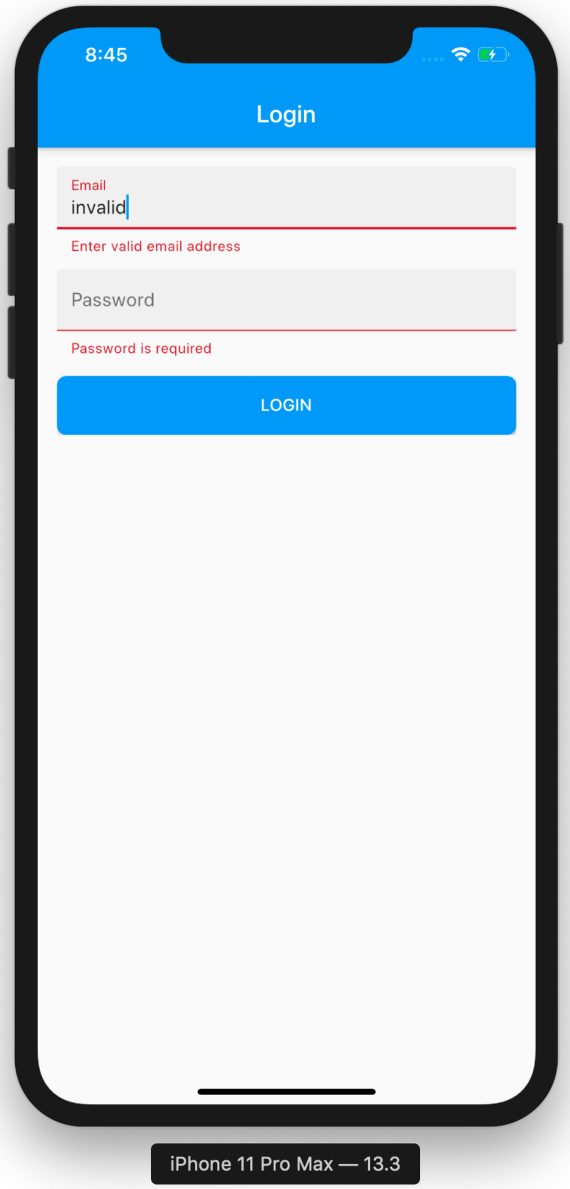
Building Forms in Flutter

1. Build a form with validation
2. Create and style a text field
3. Focus and text fields
4. Retrieve the value of a text field
5. CheckboxListTile class
6. RadioListTile class
7. DropdownButton class



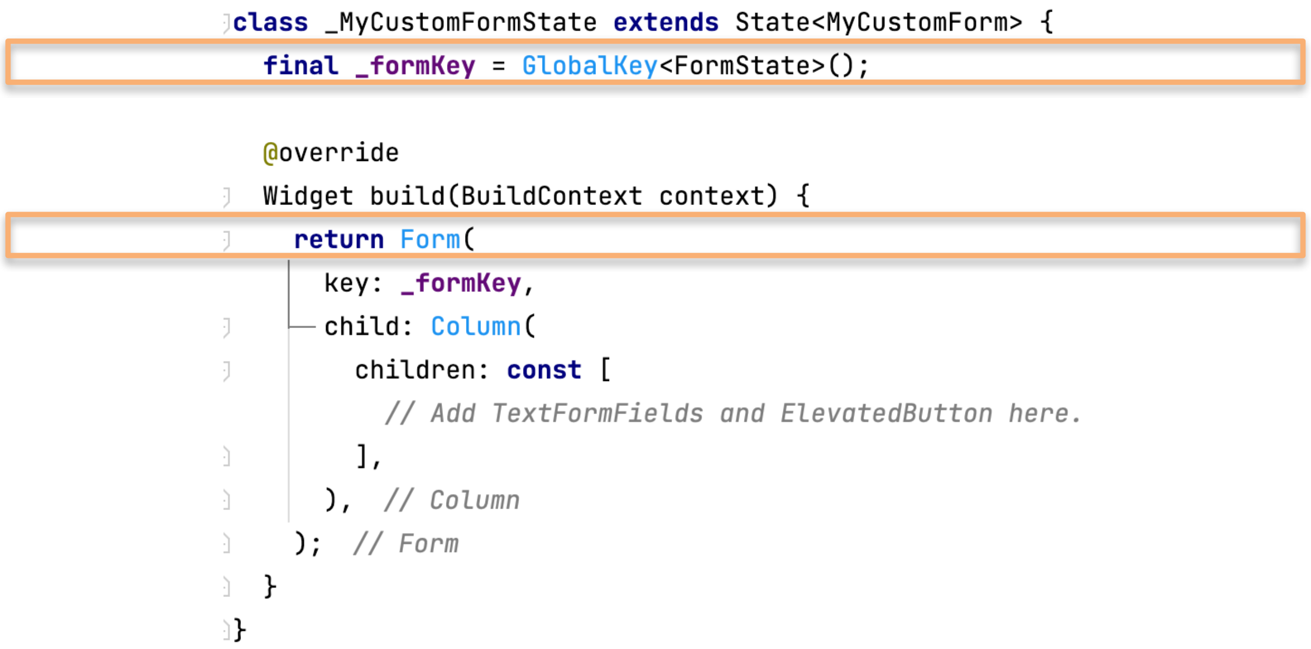
Build a form with validation

- Apps often require users to enter information into a text field.
- To check whether the information the user has provided is valid.
- If the user has correctly filled out the form, process the information.
- If the user submits incorrect information, display a friendly error message letting them know what went wrong.



Build a form with validation

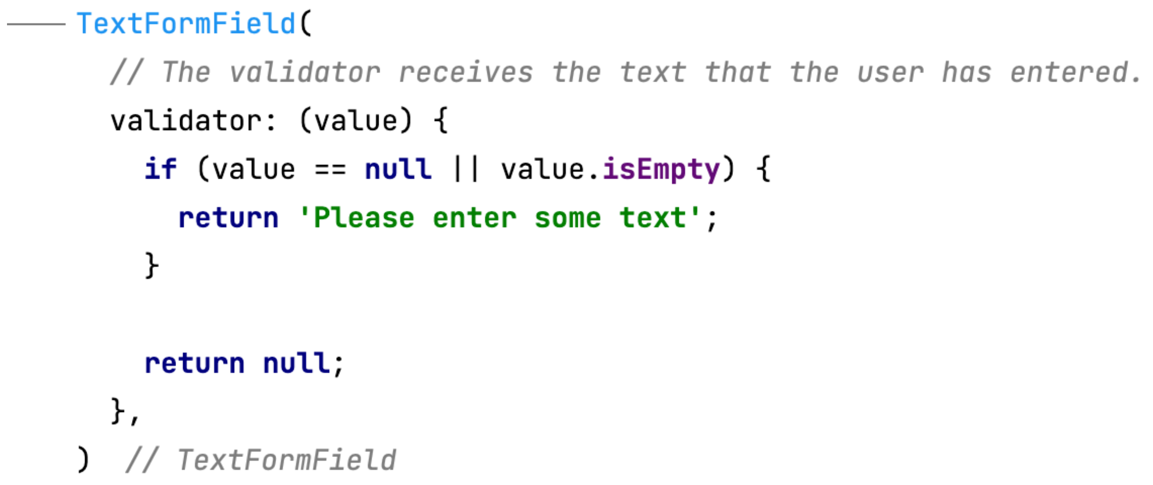
- Create a **`Form`** with a **`GlobalKey`**
  - The **`Form`** widget acts as a container for **grouping** and validating multiple **form fields**.
  - When creating the form, provide a **`GlobalKey`**. This uniquely identifies the `Form`, and allows **validation of the form**



```
class _MyCustomFormState extends State<MyCustomForm> {
  final _formKey = GlobalKey<FormState>();

  @override
  Widget build(BuildContext context) {
    return Form(
      key: _formKey,
      child: Column(
        children: const [
          // Add TextFormFields and ElevatedButton here.
        ],
      ), // Column
    ); // Form
  }
}
```

Build a form with validation

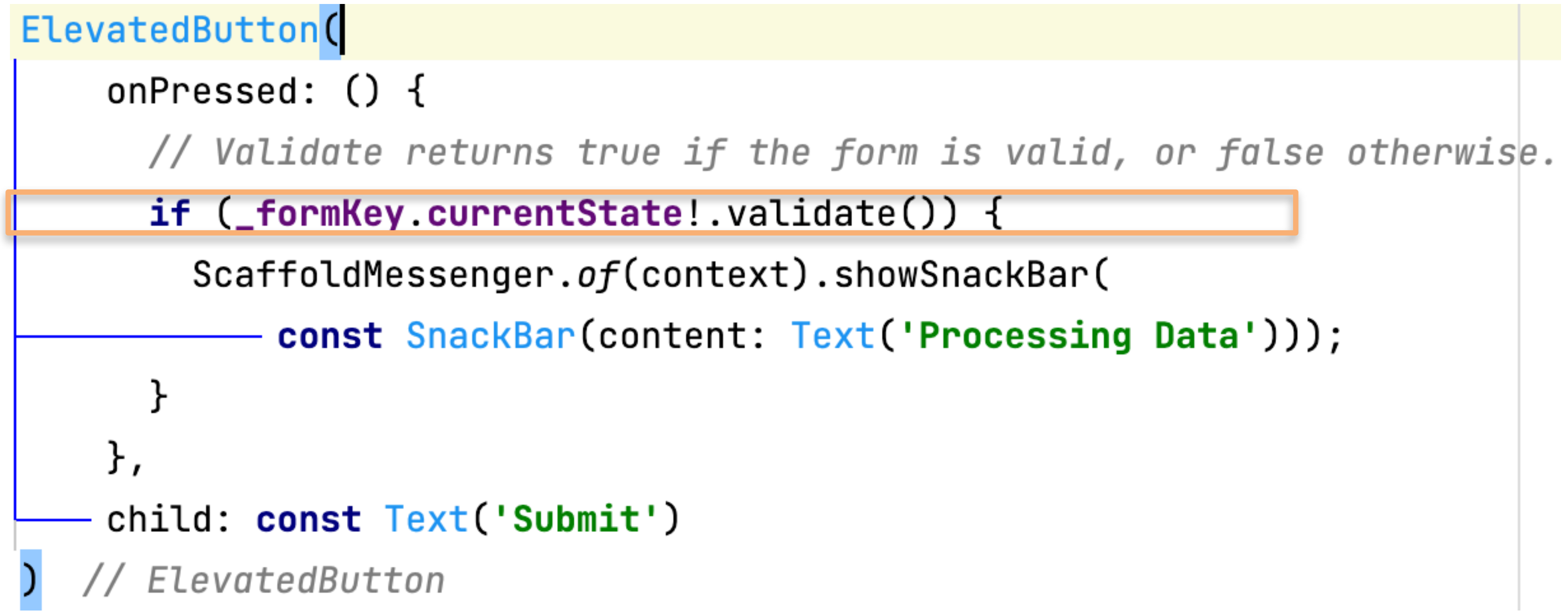
- Add a **`TextFormField`** with validation logic
  - Text fields allow users to type text into an app. They are used to build forms, send messages, create search experiences, and more.
  - The TextFormField widget can display validation errors when they occur. If the user’s input isn’t valid, the validator function returns a String containing an error message. If there are no errors, the validator must return null.



```
TextFormField(
  // The validator receives the text that the user has entered.
  validator: (value) {
    if (value == null || value.isEmpty) {
      return 'Please enter some text';
    }

    return null;
  },
)  // TextFormField
```

Build a form with validation

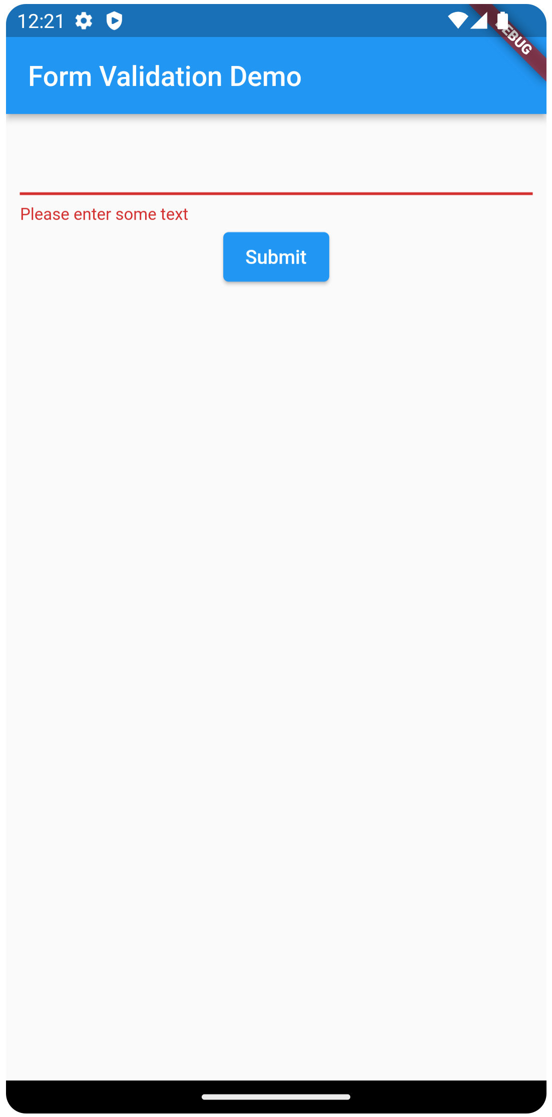
- Create a button to validate and submit the form
  - When the user attempts to submit the form, check if the form is valid.
  - If it is, display a success message. If it isn't (the text field has no content) display the error message.

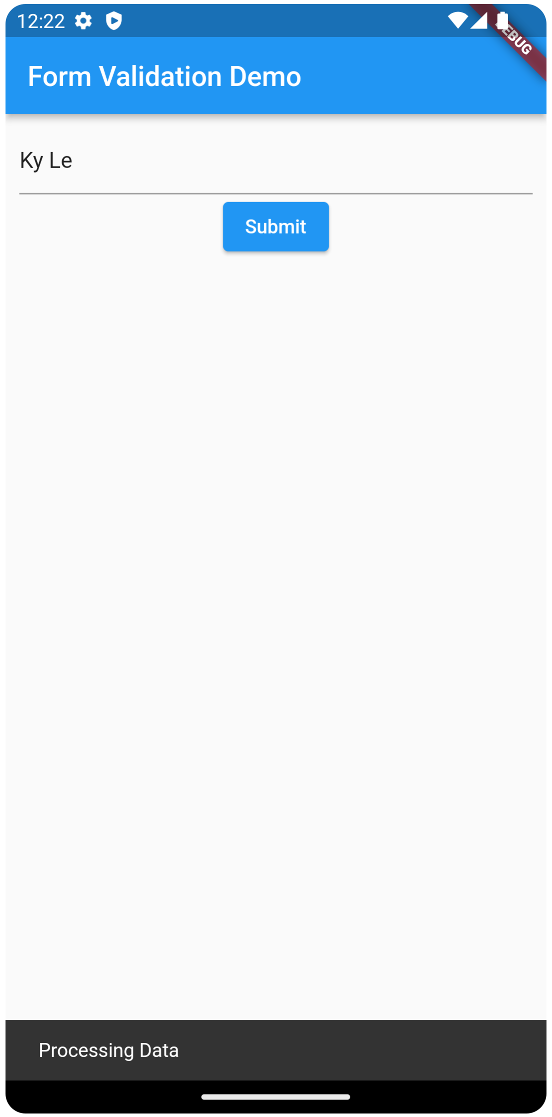


```
ElevatedButton(
    onPressed: () {
      // Validate returns true if the form is valid, or false otherwise.
      if (_formKey.currentState!.validate()) {
        ScaffoldMessenger.of(context).showSnackBar(
          const SnackBar(content: Text('Processing Data')));
      }
    },
    child: const Text('Submit')
) // ElevatedButton
```

Build a form with validation

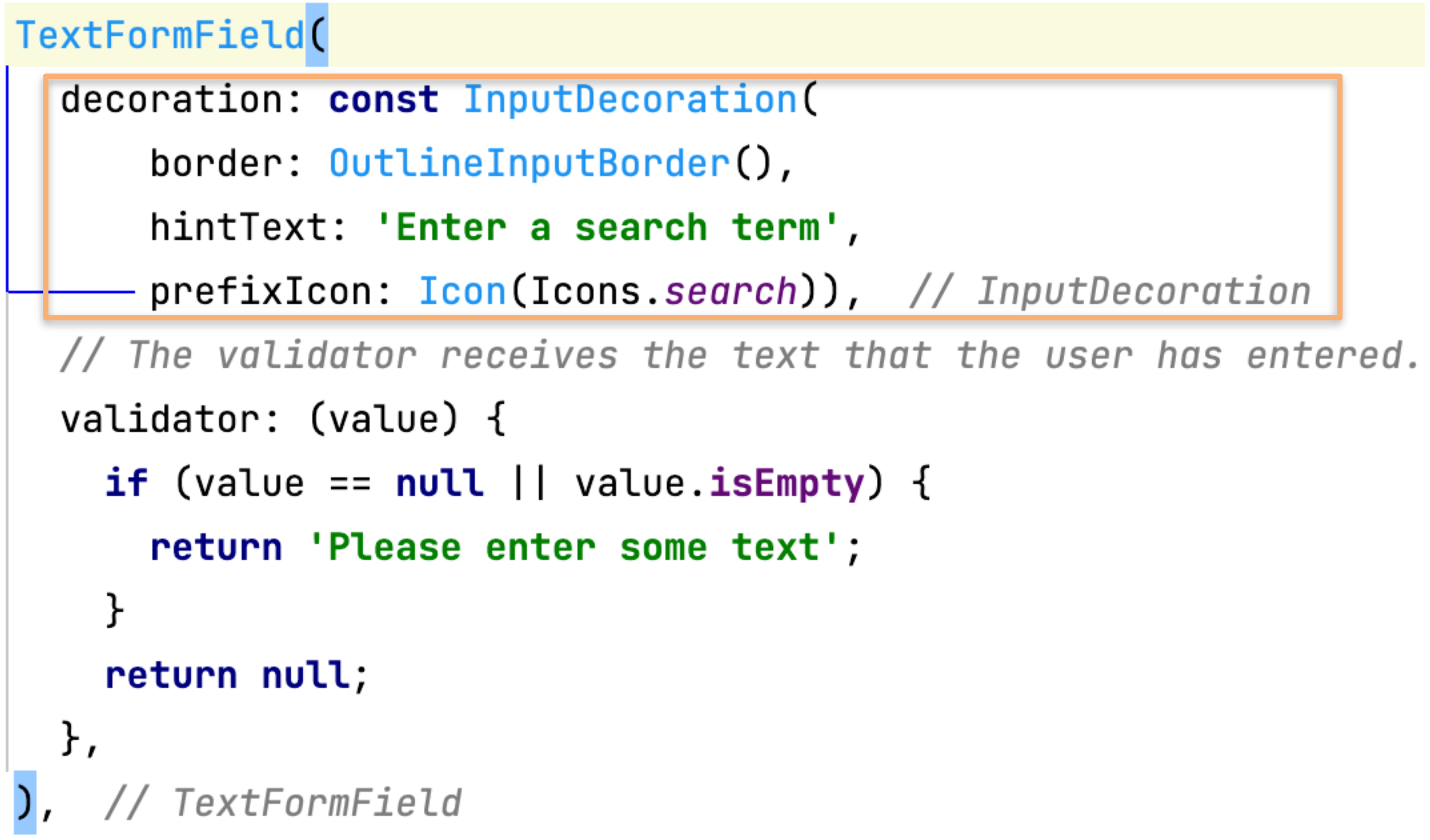
- Summary
  1. Create a `Form` with a `GlobalKey`
  2. Add a `TextFormField` with validation logic
  3. Create a button to validate and submit the form



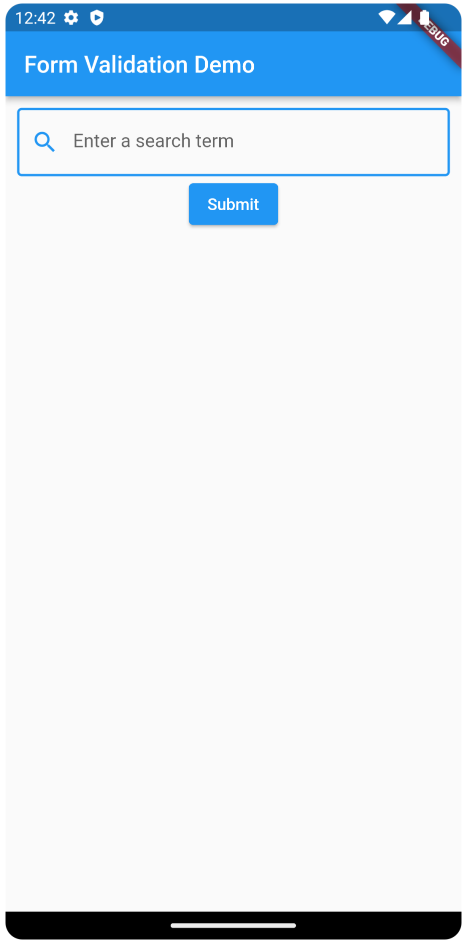


Create and style a text field

- By default, a TextField is decorated with an underline. You can add a **label**, **icon**, **inline hint text**, and **error text** by supplying an **InputDecoration** as the **decoration** property of the TextField.

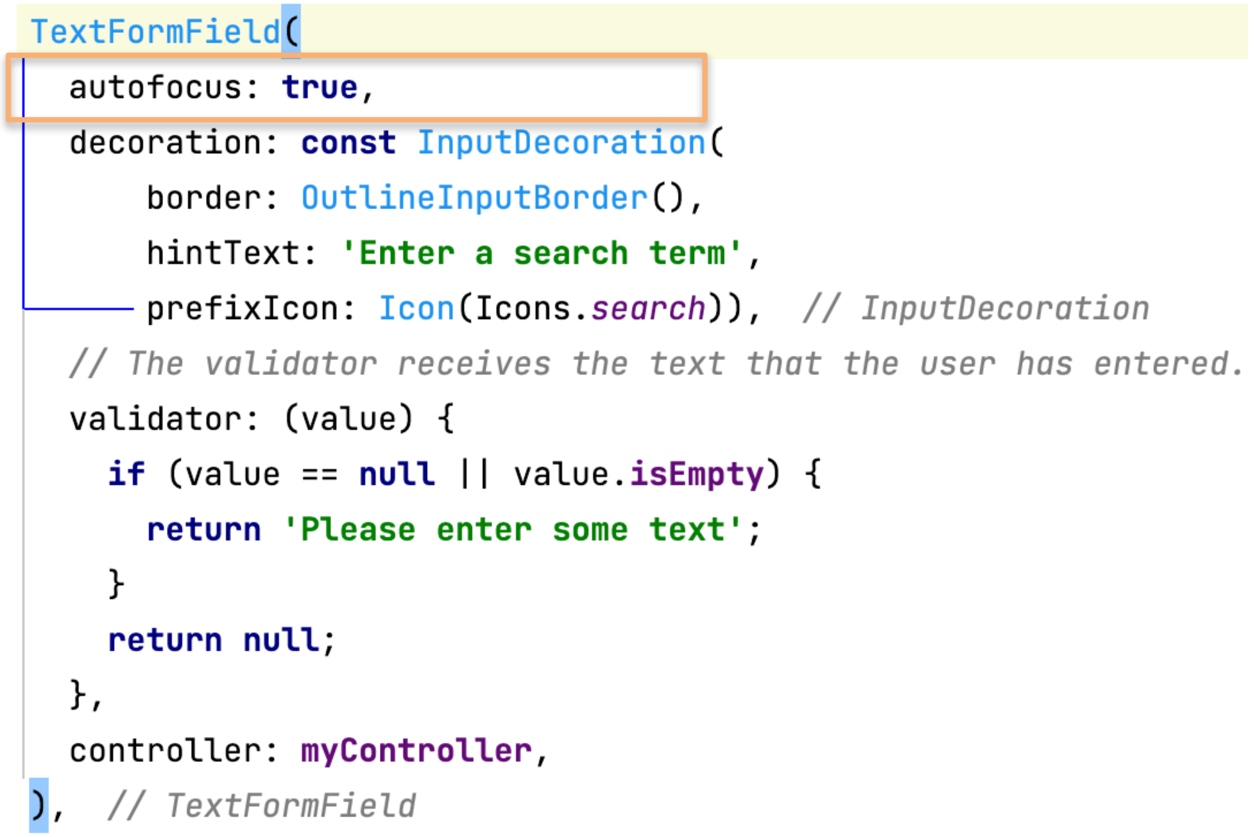


```
TextFormField(
  decoration: const InputDecoration(
      border: OutlineInputBorder(),
      hintText: 'Enter a search term',
      prefixIcon: Icon(Icons.search)),  // InputDecoration
  // The validator receives the text that the user has entered.
  validator: (value) {
    if (value == null || value.isEmpty) {
      return 'Please enter some text';
    }
    return null;
  },
),  // TextFormField
```



Focus and text fields

- When a text field is **selected** and **accepting input**, it is said to have “focus.”
- Focus a text field as soon as it’s visible



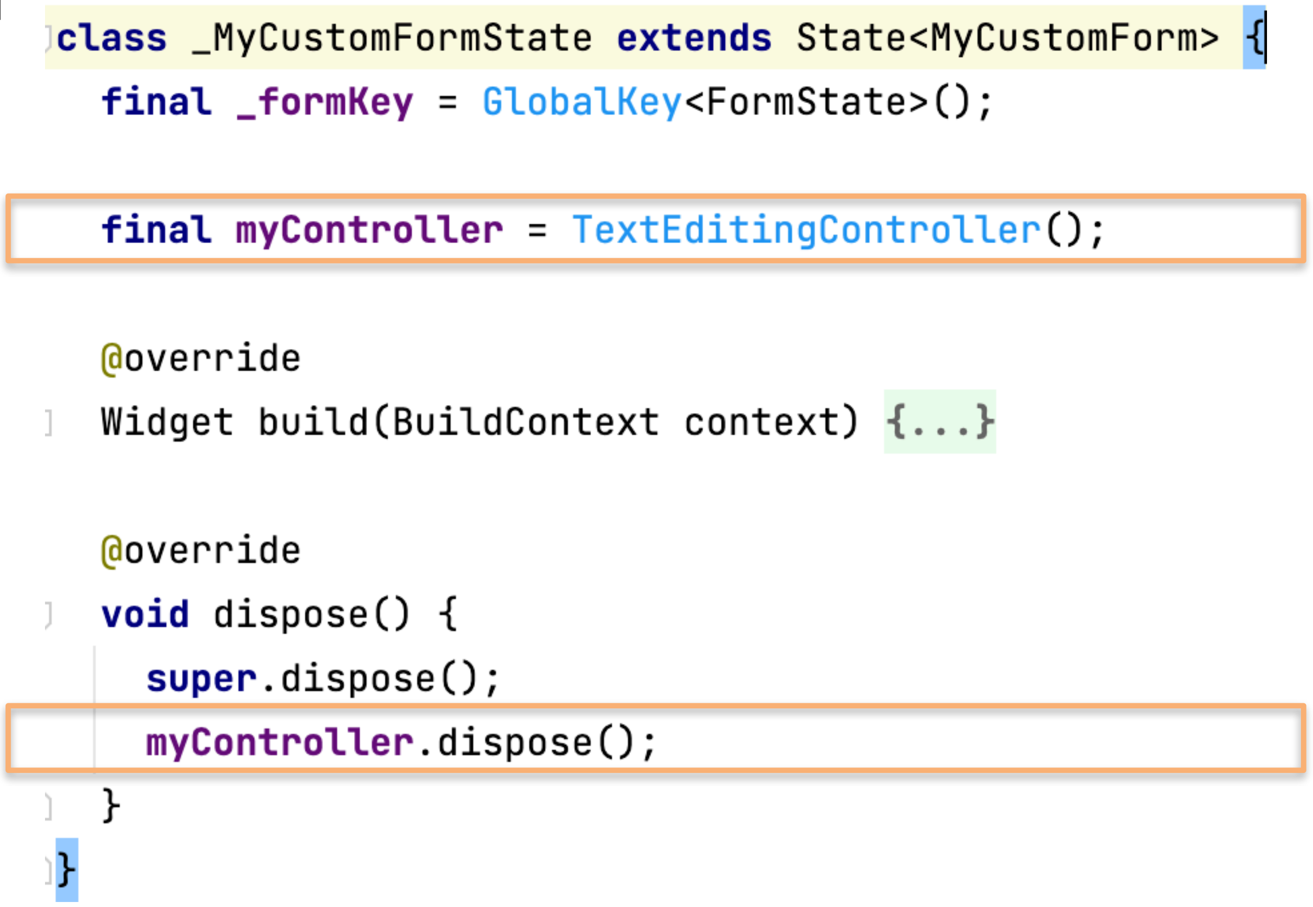
```
TextFormField(
  autofocus: true,
  decoration: const InputDecoration(
      border: OutlineInputBorder(),
      hintText: 'Enter a search term',
      prefixIcon: Icon(Icons.search)),  // InputDecoration
  // The validator receives the text that the user has entered.
  validator: (value) {
    if (value == null || value.isEmpty) {
      return 'Please enter some text';
    }
    return null;
  },
  controller: myController,
),  // TextFormField
```

Retrieve the value of a text field

- Use a TextEditingController
  1. Create a TextEditingController.
  2. Supply the TextEditingController to a text field.
  3. Display the current value of the text field.

Retrieve the value of a text field

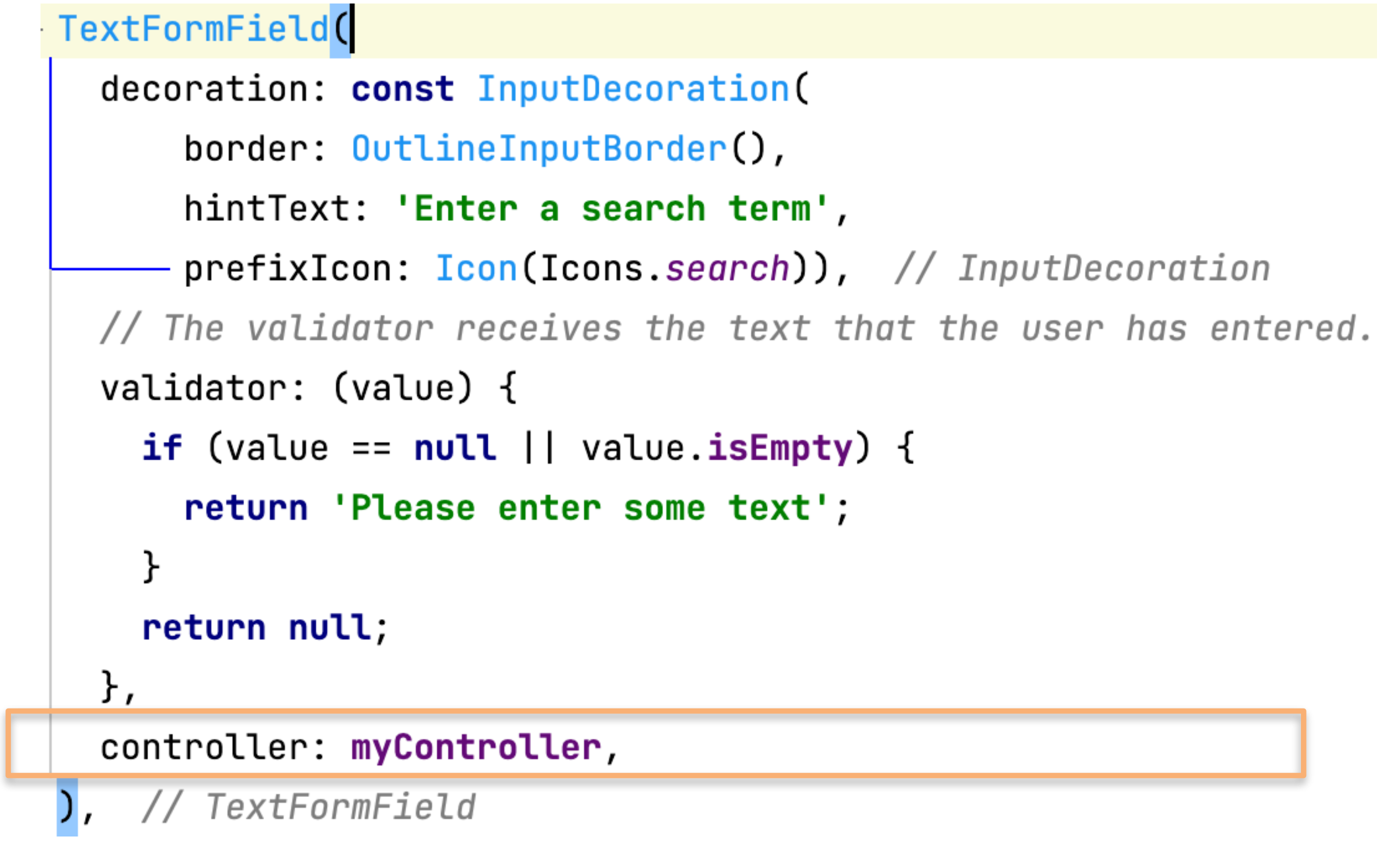
- Create a **TextEditingController**: To retrieve the text a user has entered into a text field, create a TextEditingController and supply it to text field



```
class _MyCustomFormState extends State<MyCustomForm> {
  final _formKey = GlobalKey<FormState>();

  final myController = TextEditingController();

  @override
  Widget build(BuildContext context) {...}

  @override
  void dispose() {
    super.dispose();
    myController.dispose();
  }
}
```

Retrieve the value of a text field

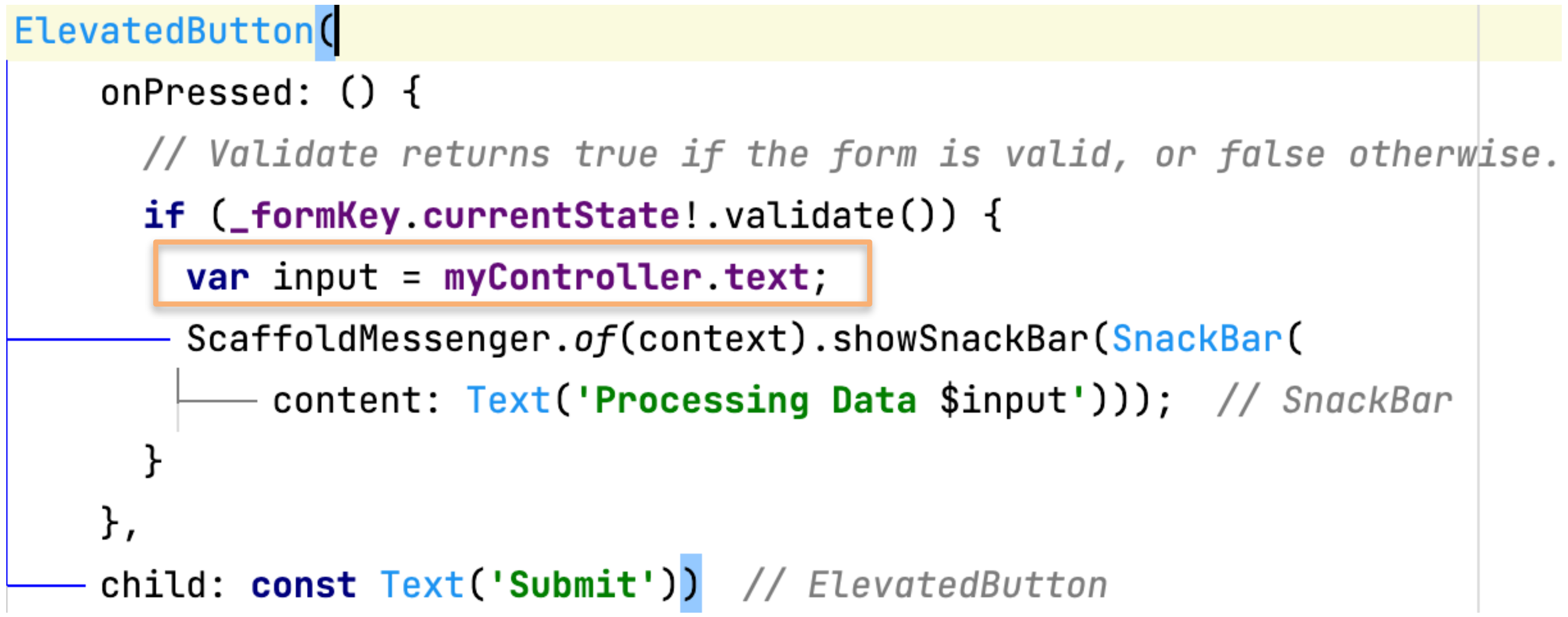
- Supply the TextEditingController to a text field: Using the **controller** property:



```
TextFormField(
  decoration: const InputDecoration(
      border: OutlineInputBorder(),
      hintText: 'Enter a search term',
      prefixIcon: Icon(Icons.search)),  // InputDecoration
  // The validator receives the text that the user has entered.
  validator: (value) {
    if (value == null || value.isEmpty) {
      return 'Please enter some text';
    }
    return null;
  },
  controller: myController,
),  // TextFormField
```

Retrieve the value of a text field

- Display the current value of the text field: Use the **text()** method provided by the TextEditingController to retrieve the String that the user has entered into the text field

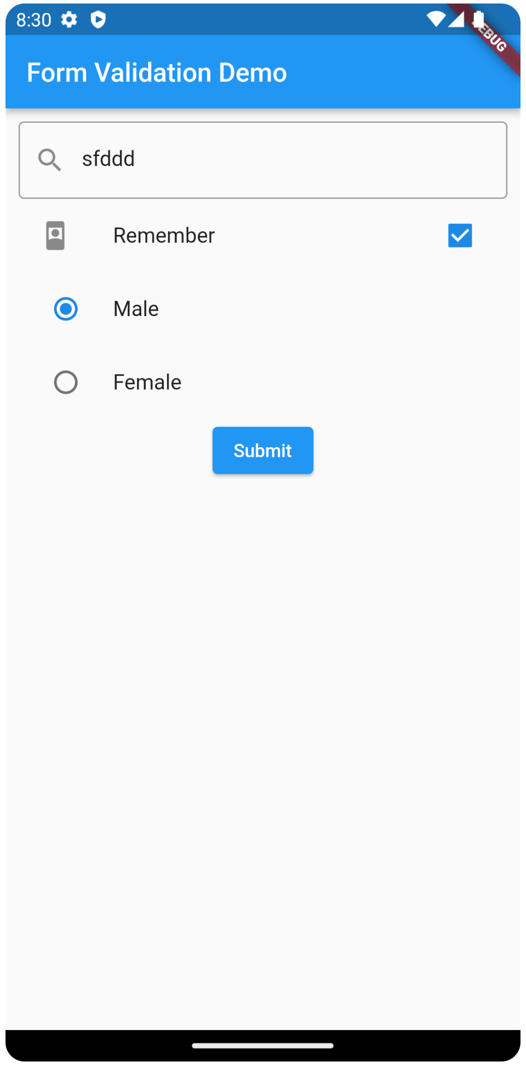


```
ElevatedButton(
    onPressed: () {
      // Validate returns true if the form is valid, or false otherwise.
      if (_formKey.currentState!.validate()) {
        var input = myController.text;
        ScaffoldMessenger.of(context).showSnackBar(SnackBar(
            content: Text('Processing Data $input')));  // SnackBar
      }
    },
    child: const Text('Submit'))  // ElevatedButton
```

CheckboxListTile class

- A **ListTile** with a **Checkbox**. In other words, a checkbox with a label.

```
CheckboxListTile(
      value: isChecked,
      title: const Text('Remember'),
      secondary: const Icon(Icons.remember_me),
      onChanged: (value) {
        setState(() {
          isChecked = value!;
        });
      }), // CheckboxListTile
```



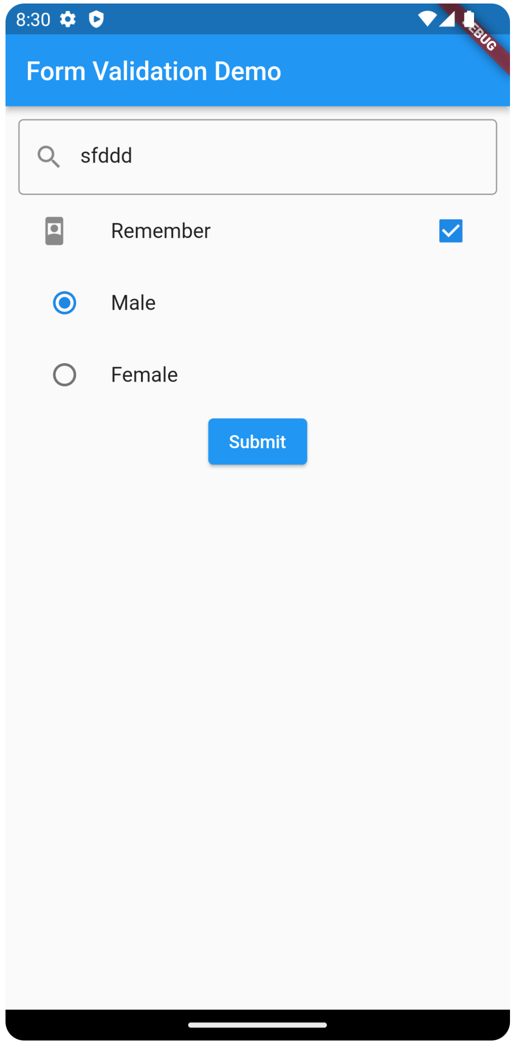
RadioListTile class

- A **ListTile** with a **Radio**. In other words, a radio button with a label.

```
enum GenderCharacter { male, female }
```

```
var gender = "";
GenderCharacter _character = GenderCharacter.male;
```

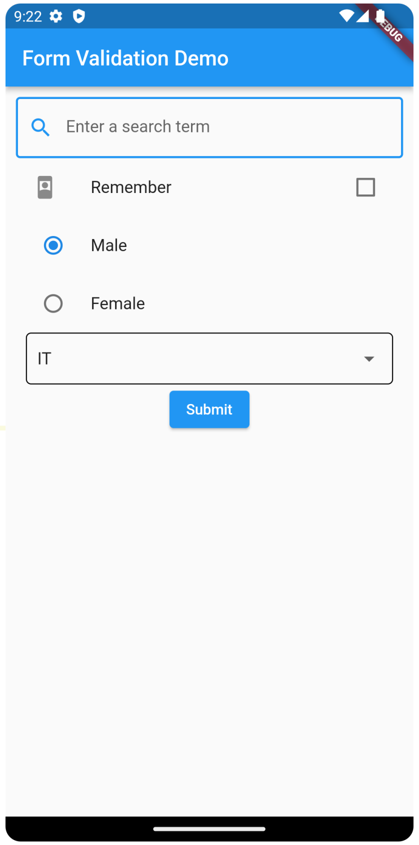
```
RadioListTile(
    title: const Text('Male'),
    value: GenderCharacter.male,
    groupValue: _character,
    onChanged: (GenderCharacter? value) {
      setState(() {
        _character = value!;
        gender = GenderCharacter.male.name;
      });
    }), // RadioListTile
RadioListTile(
    title: const Text('Female'),
    value: GenderCharacter.female,
    groupValue: _character,
    onChanged: (GenderCharacter? value) {
      setState(() {
        _character = value!;
        gender = GenderCharacter.female.name;
      });
    }), // RadioListTile
```



DropdownButton class

- A dropdown button lets the user select from a number of items. The button shows the currently selected item as well as an arrow that opens a menu for selecting another item.

```
List<String> list = ['IT', 'Marketing', 'Sales'];
late String dropdownValue;

@override
void initState() {
  super.initState();

  dropdownValue = list.first;
}
```



```
child: DropdownButtonHideUnderline(
  child: DropdownButton(
      borderRadius: const BorderRadius.all(Radius.circular(3)),
      items: list.map((value) {
        return DropdownMenuItem(value: value, child: Text(value));
      }).toList(),
      value: dropdownValue,
      onChanged: (String? value) {
        setState(() {
          dropdownValue = value!;
        });
      }),  // DropdownButton
), // DropdownButtonHideUnderline
```