04. Buttons and Texts

The most basic input/outputs in a GUI

- Buttons
  - ElevatedButton (elevatedbutton1.dart)
  - IconButton (iconbutton1.dart)
  - Checkbox/Switch (checkbox1.dart)
  - DropdownButton (dropdownbutton1.dart)
  - RadioButton
- Text
  - Icon (icon1.dart)
  - Text (text1.dart)
  - TextField
  - TextFormField (textformfield1.dart)
  - TextForm Validation (textformfield2.dart)
  - TextButton (textbutton1.dart)

Buttons

- Flutter supports a variety of input/output widgets.
- Buttons are the most widely used input widget.

ElevatedButton (elevatedbutton1.dart)

```
ElevatedButton(
  child: const Text('ElevatedButton'),
  onPressed: () {},
),
```

- ElevatedButton widget is a simple button that reacts to the user's press action.

IconButton (iconbutton1.dart)

```
IconButton(
  icon: const Icon(Icons.add),
  color: Colors.red,
  iconSize: 100.0,
  onPressed: () {},
),
```

- The IconButton widget is an icon-type button.

Checkbox/Switch (checkbox1.dart)

```
Checkbox(
  value: _isChecked,
  onChanged: (value) {
    ...
  },
),
```

- A checkbox is an on/off switch.
- The state variable is _isChecked.

```
onChanged: (value) {
  setState(() {
    _isChecked = value ?? false;
  });
```

- When users change the checkbox, the `setState()` is called to redraw widgets when users change the value of _isChecked.

```
Switch(
  value: _isChecked,
  onChanged: (value) {
    setState(() {
      _isChecked = value;
    });
  },
),
```

- The Switch widget is the same as the Checkbox with a different shape.

DropdownButton (dropdownbutton1.dart)

```
final _valueList = ['First', 'Second', 'Third'];
var _selectedValue = 'First';
Text(this._selectedValue), // <- display the selection
```

- When users need to choose between these three items, the DropdownButton widget enables them to select an item from multiple choices.

DropdownButton

```
DropdownButton(
  value: _selectedValue,
  items: _valueList.map(...).toList(),
  onChanged: (value) {...},
),
```

- The selected value is stored in the `value` property.
- The items to be selected are stored in the `items` parameter.

DropdownMenuItem

```
items: _valueList.map(
  (value) {
    return DropdownMenuItem(
      value: value,
      child: Text(value),
    );
  },
```

- From the _valueList, the map function makes a list of DropdownMenuItem.

```
onChanged: (value) {
  setState(() {
    _selectedValue = value as String;
  });
},
```

- From the DropdownMenuItem, users select an item to trigger the onChange property.
- The setState() function is called to change the selection and redraw widgets.

RadioButton

```
enum Gender { MAN, WOMEN }
Gender _gender = Gender.MAN;
```

- Radio button is the same as the DropdownBox in that they create exclusive-choice lists.
- Users can choose between MAN and WOMEN using a radio button.

ListTile + Radio (radioButton1.dart)

```
ListTile(title: Text('Gentleman'), leading: Radio(...),),
ListTile(title: Text('Lady'), leading: Radio(...),),
```

- The ListTile widget hosts the Radio Button and its Text.
- The ListTile uses a Text widget to display the title.

```
ListTile(title: Text('Gentleman'),
  leading: Radio(
    value: Gender.MAN,
    groupValue: _gender,
    onChanged: (value) {_update(value);},
  ),
```

- `value` defines what the radio button represents.
- `groupValue` tells Flutter the currently selected value in the group of radio buttons.

```dart
bool _isSelected = value == groupValue;
```

- The `Radio` widget needs to know whether it should appear selected or unselected.
- It does this by comparing `value` and `groupValue`.
- If `value == groupValue`, the radio button shows as selected (filled circle).

```
value: ...
onChanged: (value) {_update(value);},
```

- The `value` is used when a user selects the Radio button to call the `_update(value)`.

```
_update(value) {
  setState(() {
    _gender = value as Gender;
    if (_gender == Gender.MAN) {
      _text = 'Gentleman was Chosen';
    }
    else {
     _text = 'Lady was Chosen';
    }
```

- In the _update function, setState() is called to update the GUI variable and redraw widgets.

RadioListTile (radiobutton2.dart)

```
RadioListTile(
  title: Text('Gentleman'),
  value: Gender.MAN,
  groupValue: _gender,
  onChanged: (value) {
    setState(() {_gender = value as Gender;});
  },
),
```

- `RadioListTile` is a widget to combine `Radio` and `ListTile`.

```
// ListTile and Radio should implement this code
onTap: () {
  setState(() {
    _delivery = Delivery.express;
  });
},
```

- On Top of that, tapping anywhere on a `RadioListTile` selects the radio button, not just the radio button itself.
- We don't need to implement the onTap property.

Text

- The Text widgets display information using an image (icon) or text.

Icon (icon1.dart)

```
Icon(
  Icons.favorite,
  color: Colors.pink,
  size: 24.0,
),
```

- An icon is used to display information using an image.

Text (text1.dart)

```
return const Text(
  "16 size, 3 spacing, and blue color",
  style: TextStyle(
    color: Colors.blue,
    fontSize: 16,
    decoration: TextDecoration.none,
  ),
);
```

- We use the style property and the TextStyle widget to decorate the text.

text_themeof1.dart

```
Widget build(BuildContext context) {
  return Scaffold(
    body:Text(
      "From TextTheme: labelLarge ",
      style: Theme.of(context).textTheme.labelLarge,
    ),
  );
}
```

- We can decorate texts using a pre-existing theme.

```
style: Theme.of(context).textTheme.labelLarge,
```

- `Theme.of(context)` is a static method that takes the widget's `BuildContext` to find the `ThemeData`.
- `.textTheme` is a property of `ThemeData` that contains a collection of standard text styles (like headline, body, label, etc.).

TextField

- The TextField widget in Flutter is the primary tool for user text input (e.g., usernames, emails, messages).
- It’s highly customizable for styling, input handling, and validation.

textfield1.dart

```
TextField(),
TextField(
  decoration: InputDecoration(
    border: OutlineInputBorder(),   // bordered outline
    labelText: 'Input Anything',
  ),
),
```

- TextField is a widget to get the user's input.
- We can decorate it with the InputDecoration widget.

textfield2.dart

```
void _updateString(String newString) {
  setState(() {_string = newString;});
}
TextField(
  onChanged: (text) {_updateString(text);},
),
```

- When users give any input, the onChanged property is triggered to run the `_updateString` function.

textfield3.dart

```
Text(_string),
TextField(
  onSubmitted:
   (value) => _updateString(value),
),
```

- When users finish giving input, the onSubmitted property is triggered.

textfield4.dart

```dart
final myController = TextEditingController();

void _printLatestValue() {
  final text = myController.text;
  print(...);
}
```

- We can use `TextEditingController` to manage users.

```
void initState() {
  super.initState();
  // setup observer
  myController.addListener(_printLatestValue);
}
```

- The service function (_printLatestValue) is invoked whenever users give inputs to the text field.
- This is called the `observer design pattern`.

```
TextField(controller: myController,),
```

- In the TextField widget, we can connect the controller with the TextEditingController widget.

```
// with a controller
TextField(controller: myController,),
// without a controller
TextField(
  onChanged: (text) {
    print('${text.characters.length})');
  },
),
```

- Compared to the code without a controller, using the controller makes the program simple and easy to read.

TextFormField (textformfield1.dart)

```
return TextFormField(
  decoration: const InputDecoration(
    icon: Icon(Icons.person),
    hintText: 'What do people call you?',
    labelText: 'Name *',
  ),
);
```

- For more complex input, we can use TextFormField.

TextForm Validation (textformfield2.dart)

```
final _formKey = GlobalKey<FormState>();
var _controller = TextEditingController();
```

- TextFormField is a Flutter widget that supports form validation.
- It works with a Form and GlobalKey to validate user input using custom logic.

```
TextFormField(controller: _controller,
  validator: (value) => validateRequired(value),
),
ElevatedButton(
  onPressed: () { if (_formKey.currentState!.validate()) {
```

- The `validator` function (e.g., `validateRequired`) is assigned to each field.
- When the user taps the Submit button, you call `validate()` on the form's state.

The validator at Each Field

```
TextFormField(controller: _controller,
  validator:
    (value) => validateRequired(value),
),
```

- The TextFormField uses the controller to get a textual input.
- We use the `validator` property to check the validity of the input.

```
static String? validateRequired(String? value) {
  if (value == null || value.isEmpty) {
    return 'Please enter some text';
  }
  return null;
}
```

- In this example, the validation utility checks if the input is not empty.
- It returns `null` when the form is valid.

The validate() function

```
final _formKey = GlobalKey<FormState>();
ElevatedButton(
  onPressed: () {
    if (_formKey.currentState!.validate()) { ... }
  }
```

- We need the GlobalKey<FormState> to access the form's state.
- This method goes through every field with a validator in the form and calls its `validator` function.

Be careful with memory leakage!

```
void dispose() {
  // Clean up the controller when the widget is disposed.
  _controller.dispose();
  super.dispose();
}
```

- When the widget is disposed of from memory, the TextEditingController should also be disposed of because it listens to users' input.
- This can lead to memory leakage.

TextButton (textbutton1.dart)

```
child: TextButton(
  child: const Text('TextButton'),
  onPressed: () {},
),
```

- TextButton is a Text that can react to the user's input.