05. Dialogs

Giving inputs or getting outputs in standalone widgets

- Dialogs
    - alertdialog.dart
    - alertdialog_arguments_return.dart
- Dialog widgets
    - dialog1.dart
    - dialog2.dart
- Pickers
    - Date Picker (datepicker1.dart)
    - Time Picker (timepicker1.dart)

Dialogs

- Flutter supports dialog widgets for giving inputs or getting outputs in standalone widgets.
- Using dialog widgets, we can open a new window for users.
- We can show information using dialog widgets.

alertdialog.dart

```
body: ElevatedButton(
  onPressed: () { _openDialog(); },
  child: const Text('Alert Dialog'),
),
_openDialog() {
  return showDialog(...);
}
```

- The `_openDialog` is a placeholder to run the showDialog function.

showDialog function

```
return showDialog(
  builder: (BuildContext context) {
    return AlertDialog(...)
  }
)
```

- showDialog displays an AlertDialog in a modal overlay, with focus and dismissal.
- It blocks interaction with the underlying UI and returns values when closed.

AlertDialog Widget

```
return AlertDialog(
  title: const Text('Title'),
  content: ...
  actions: <Widget>[
    TextButton(...), ...
  ],
);
```

- AlertDialog is just a widget—without showDialog, it won’t appear as a modal or block interaction.

showDialog as a Middleman

- In Flutter, showDialog acts as a middleman, managing how dialogs are displayed.
- Calling AlertDialog directly is simpler, but removes flexibility.
- Using showDialog maximizes flexibility, letting Flutter handle dialogs in a consistent, customizable way.

Return from Dialog

```
Navigator.of(context).pop();
```

- Showing a dialog pushes a new route onto the navigation stack.
- You can access the stack with Navigator.of(context).
- Pop the stack to return to the previous widget.

alertdialog_arguments_return.dart

```
onPressed: () async {
  var result = await _openDialog('hello');
  _update(result);
},
...
_openDialog(String info) { ... }
```

- We can give arguments to the `_openDialog` service function.

Usage of the Arguments

```
_openDialog(String info) {
  return showDialog(
    builder: (BuildContext context) {
      return AlertDialog(
        title: Text('Title + ${info}'),
```

- The arguments can be used to make an AlertDialog widget.

Return from the AlertDialog

```
onPressed: () {
  Navigator.of(context).pop(_controller.text);
},
...
var result = await _openDialog('hello'); //
```

- The dialog returns a value using Navigator.pop.
- The result variable stores the value, such as _controller.text.

Dialog widgets

- AlertDialog is a simple, ready-to-use dialog for alerts, confirmations, and messages with up to three action buttons.
- Dialog is a flexible base widget for building fully custom dialogs with any layout or content.

dialog1.dart

```
builder: (context) {
  return const Dialog(
    child: Text('Dialog Title'),
  );
}
```

- Dialog is a general-purpose container for customization.
- You have full control over layout and appearance.

dialog2.dart

```dart
floatingActionButton: FloatingActionButton(
  onPressed: () {
    showDialog(
      context: context,
      builder: (context) {
        // Customize Dialog
        return DialogUI(input:info, function:_updateString);
      }
    );
  },
  child: const Text('Dialog'),
),
```

- In this example, we customize a Dialog widget.

DialogUI Stateless Widget

```
class DialogUI extends StatelessWidget {
  final String input;
  final function;
  DialogUI({required this.input, required this.function});

  String returnValue = "";
  @override
  Widget build(BuildContext context) {
    return Dialog(
```

- The DialogUI is a stateless widget to host the Dialog.

```
DialogUI({required this.input, required this.function});
```

- The DialogUI constructor has two arguments:
  - The first one is the information displayed on the Dialgo.
  - The second one is the function to be invoked inside the Dialog.

```
String info = 'No information yet';
_updateString(result) {
  setState(() {
    info = result;
  });
}
return DialogUI(input: info, function: _updateString);
```

- We give the `String info` to the first argument.
- We give the _updateString function to the second argument.

Dialog widget

```
return Dialog(
  child: Container(
    child: Column(
      children: [TextField(...),TextButton(...),TextButton(...)]
```

- We need multiple widgets, so we use the Container.
- The Container has a Column to host three widgets

```
Text(this.input),
TextField(
  onChanged: (val) {returnValue = val;},
),
TextButton(onPressed:(){
    function(returnValue); Navigator.pop(context);
  }
),
TextButton(child: Text('Cancel'),
  onPressed:(){ Navigator.pop(context);
  }
)
```

- The Dialog has one text, a textfield, and two buttons.

1. Text

```
return DialogUI(input: info, function: _updateString);
...
Text(this.input),
```

- The Text displays the string from the first argument of DialogUI.

2. TextField

```
String returnValue = "";
...
TextField(
  onChanged: (val) {returnValue = val;},
),
```

- This is for users' input.
- When users give inputs, the input is stored in the `returnValue`.

3. Done TextButton

```
DialogUI({required this.input, required this.function});
...
TextButton(
  child: Text('Done'),
  onPressed:(){
    function(returnValue);
    Navigator.pop(context);
  }
),
```

- When users press this button, the given function is called with the returnValue.

```
_updateString(result) {
  setState(() {
    info = result;
  });
}
```

- The function is `_updateString` that updates the `info` and redraws widgets with `setState`.
- The returnValue is stored in this.info.

4. Cancel TextButton

```
TextButton(
  child: Text('Cancel'),
  onPressed:(){ Navigator.pop(context);
  }
)
```

- When users press the `Cancel` button, the Dialog returns to its caller using the `Navigator.pop(context)`.

DialogPage and State<DialogPage>

```
class _DialogPageState extends State<DialogPage> {
...
floatingActionButton: FloatingActionButton(
  onPressed: () {
    showDialog(
      builder: (context) {
        return DialogUI(input:info,
...
```

- To use the DialogUI, we make a Stateful widget and its State<T>.
- This widget calls the DialogUI when the FAB is pressed.

Pickers

- A Picker Dialog in Flutter is a pop-up used to select values like dates or times.
- It provides a styled, interactive interface for user-friendly input.
- After a selection, the result can be handled (e.g., with setState) to update the UI.

Date Picker (datepicker1.dart)

```
var selectedDate = showDatePicker(
  context: context,
  initialDate: DateTime.now(),
  firstDate: DateTime(2025),
  lastDate: DateTime(2030),
); // showDatePicker
```

- To select a date using a dialog, we can use the `showDatePicker` function.

```
selectedDate.then((dateTime) {
  setState(() {
    _selectedTime = dateTime as DateTime;
  });
  ..
}
...
Text('$_selectedTime'),
```

- After users select the date, the chosen date is given to the `setState` to redraw widgets.

Time Picker (timepicker1.dart)

```dart
var selectedTime = showTimePicker(
  initialTime: TimeOfDay.now(),
  context: context,
);
```

- We can use the `showTimePicker` to pick a time.

```
selectedTime.then((timeOfDay) {
  setState(() {
    _selectedTime =
      '${timeOfDay?.hour ?? 0}:
       ${timeOfDay?.minute ?? 0}';
  });
...
Text('$_selectedTime'),
```

- Users select a time, and the chosen time is given to the `setState` to redraw widgets.