Step 3 – Add A Todo

Pop-up's, cleaner code and form/text inputs.

Floating Action Button

- For this Todo app lets add a floating action button to handle the adding a new Todo.
- This button will present the user with options to add a new Todo.
- In the scaffold, after the body, add a floatingActionButton: parameter.
- Create a new FloatingActionButton() with three parameters:
  - onPressed:
  - tooltip:
  - child:
- The on pressed is like an event handler you may have seen from JavaScript or other event driven programming styles.

Floating Action Button - onPressed

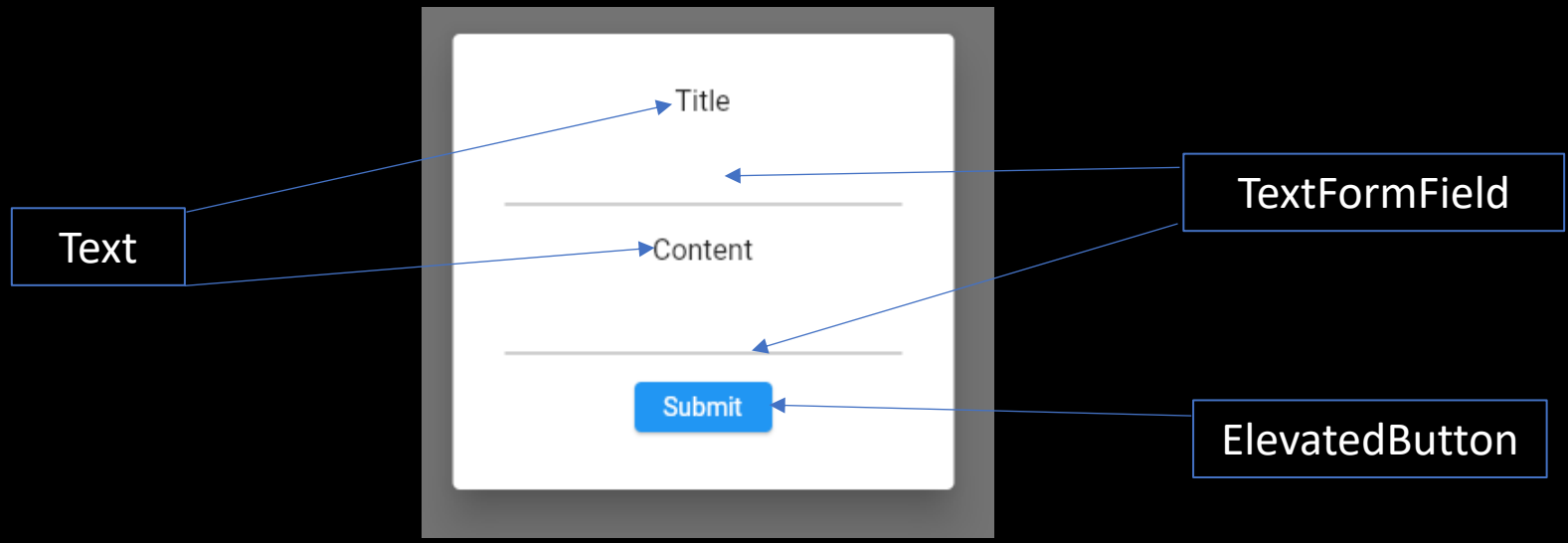
- For the onPressed parameter we must pass it a function that will be called (the callback) for the function when it is pressed.
- Anywhere inside your state object, create a new function with no return type called _openAddTodo.
- Remember the underscore means this function is private.
- It requires no input parameters because we're still working in the stateful widget.
- Add this function to the onPressed parameter as a reference not by calling the function. I.e. without the parentheses.
- Calling the function: _openAddTodo()
- Passing Reference to function: _openAddTodo
- Fill the tooltip and add a child of a const icon with the Icons.add in it.

ShowDialog()

- To easily present a pop up we can use the ShowDialog() function.
- It takes a context and a builder (like our list builder) which expects a widget to be returned.
- For the popup we'll return an AlertDialog widget.
- Create this widget and in its content create a column with children that represent the fields for the Todo data.
- To position this Column, set its mainAxisSize to MainAxisSize.min.
- In the children we can lay out our visual elements from top to bottom.

ShowDialog() UI Exercise

- We're aiming to create this UI:



- To space Widgets out wrap them in a Padding widget with EdgeInsets.

TextFormField

- This widget handles inputs.
- However, it requires a controller to handle the text editing functionality the user can perform.
- At the top of the _TodoHomePageState class near your Todo's add two TextEditingController's and instantiate them.
- Give them appropriate names.

```
final TextEditingController _controlName = TextEditingController();
final TextEditingController _controlDescription = TextEditingController();
```

TextFormField

- Now we can add these to the controller parameter of each TextFormField.
- Make sure you match the field with the correct controller.
- We will use these later to fetch the data and create the Todo object.
- You should end up with something like this for both:

```
const Padding(
  padding: EdgeInsets.fromLTRB(5, 8, 5, 0),
  child: Text("Name"),
), // Padding
Padding(
  padding: const EdgeInsets.fromLTRB(5, 0, 5, 8),
  child: TextFormField(
    controller: _controlName,
  ), // TextFormField
), // Padding
```

TextEditingController

- Finally, we can use the TextEditingController to fetch the text the user has entered when the submit button is pressed.
- Inside the onPressed: of the elevated button create an anonymous function with a body.
- () {}
- Inside call setState(){}
- Inside that call todos.add();
- Pass in the todo object by calling the constructor passing through the controller.text property.
- Todo(name: <controller>.text, description: <controller>.text)
- Finally, call Navigator.pop(context) to close the alert.

Final Adding of Todo

- You should end up with something like this, test it works:

```
child: ElevatedButton(
    child: const Text("Submit"),
    onPressed: () {
      setState(() {
        todos.add(Todo(
            name: _controlName.text,
            description: _controlDescription.text)); // Todo
      });
      Navigator.pop(context);
    }), // ElevatedButton
```