

Lab 3: Widget – Layout

In the previous exercises, we have been exposed the use of Container and Scaffold widgets as the core element for building the UI screen. Another important element in designing UI is the Layout which will define how the orientation of UI elements(widgets). From designing simple UI to complex UI, it is all start with designing the layout first.

In this lab session, we will try out to create and display certain widgets such icon, image and input text using layout elements - **List** and **ListView.Builder**

1) Basic ListView

Create a new Flutter Project >> name the project as Daily Expenses.

This section will display the Daily Expenses using standard ListView.

In **main.dart** file, write the source code as follow:

```
import 'package:flutter/material.dart';

void main() => runApp(const MyApp());

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    const title = 'Daily Expenses';

    return MaterialApp(
      title: title,
      home: Scaffold(
        appBar: AppBar(
          title: const Text(title),
        ),
      ),
    );
  }
}
```

- **import 'package:flutter/material.dart':** This line imports the Flutter Material library, which contains widgets and components for building Material Design-style applications.
- **const title = 'Daily Expenses':** This defines a constant variable **title** with the value "Daily Expenses.". It will set the app's title through the AppBar.
- **return MaterialApp(...):** This is the root of the widget tree. It creates the MaterialApp, which is a wrapper for the entire app and is responsible for managing the app's theme, navigation, and more.

Still in the main.dart file, continue the previous code with the source code as follow:

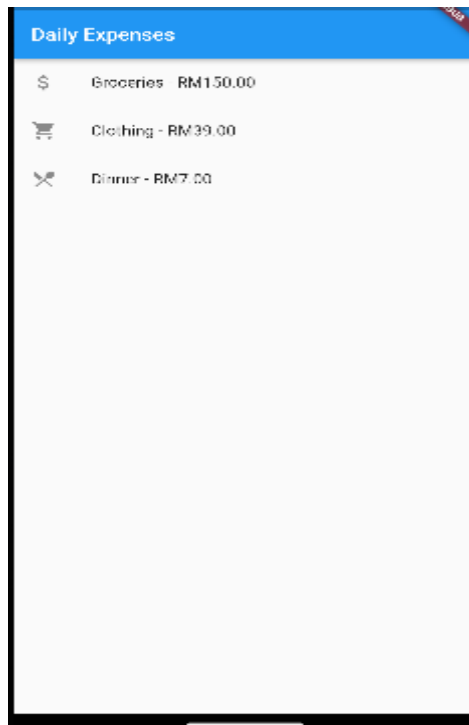
```

return MaterialApp(
  title: title,
  home: Scaffold(
    appBar: AppBar(
      title: const Text(title),
    ),
    body: ListView(
      children: <Widget>[
        ListTile(
          leading: Icon(Icons.attach_money),
          title: Text('Groceries - \RM150.00'),
        ),
        ListTile(
          leading: Icon(Icons.shopping_cart),
          title: Text('Clothing - \RM39.00'),
        ),
        ListTile(
          leading: Icon(Icons.local_dining),
          title: Text('Dinner - \RM7.00'),
        ),
      ],
    ),
  ),
);
}

```

- **body: ListView(...):** This is the body of the Scaffold and contains the main content of the app. It uses a ListView widget to display a scrollable list of items.
- **ListTile(...):** These are individual list items in the ListView. Each ListTile displays an icon (leading) and a title. In this example, three ListTiles are provided to represent daily expenses.
- **leading: Icon(Icons.attach_money):** This sets the icon for the ListTile. It uses the "attach_money" icon from the Material Icons library, which represents money or currency. You can choose other icon as shown in the source code to accommodate your code.
- **title: Text('Groceries - \RM150.00'):** This sets the title text for the ListTile. It displays the description of the expense ("Groceries") and the expense amount ("\RM150.00").

Run the code. The output will be as follow:



The above only display the static list of the `ListView` which are not dynamic. Which means, the items are defined in the code directly within the `ListView`. This step is suitable when you have a small, fixed, and known number of items, and you don't need to generate them dynamically.

We will proceed later to use **`ListView.builder`** which can build or generate items dynamically. We can display, add and remove the list of items.

2) `ListView.builder`

Create another dart file, right click on the **lib** folder >> New Dart file >> Name it as `dailyexpenses.dart`. Ensure that the method **`main()`** is defined as follow:

```
void main() {  
  runApp(DailyExpensesApp());  
}
```

Then, define a custom class called '`Expense`'. This class represents an expense and has two properties: *description* and *amount*. The definition of class is shown below:

```
class Expense {  
  final String description;  
  final String amount;  
  
  Expense(this.description, this.amount);  
}
```

Next, create the **DailyExpensesApp** class that extends `StatelessWidget` which returns `MaterialApp` widget.

```
class DailyExpensesApp extends StatelessWidget {  
  @override  
  Widget build(BuildContext context) {  
    return MaterialApp(  
      home: ExpenseList(),  
    );  
  }  
}
```

Define the class `ExpenseList` as `StatefulWidget` as the data are not constant. This class represents the main screen of the application.

```
class ExpenseList extends StatefulWidget {  
  @override  
  _ExpenseListState createState() => _ExpenseListState();  
}
```

Next, we will define the **_ExpenseListState** class which will manage the list of expenses, and handles user input. First, initialize the variables within the state classes which are the list of expenses and `TextEditingController` objects for description and amount.

Then, create method **_addExpense()**. This method will retrieve user inputs value, creates an `Expense` object and adds it to the list of expenses when the user clicks the *Add Expense* button.

The source code is shown below:

```
class _ExpenseListState extends State<ExpenseList> {  
  final List<Expense> expenses = [];  
  final TextEditingController descriptionController = TextEditingController();  
  final TextEditingController amountController = TextEditingController();  
  
  void _addExpense() {  
    String description = descriptionController.text.trim();  
    String amount = amountController.text.trim();  
    if (description.isNotEmpty && amount.isNotEmpty) {  
      setState(() {  
        expenses.add(Expense(description, amount));  
        descriptionController.clear();  
        amountController.clear();  
      });  
    }  
  }  
}
```

Still in the state class, add method **_removeExpense**. This method will remove an expense from the list based on its index.

```

void _removeExpense(int index) {
  setState(() {
    expenses.removeAt(index);
  });
}

```

Then we need to build the App's UI. The UI will consist of:

- AppBar with a title.
- Two input fields for entering the description and amount of an expense.
- An *Add Expense* button to add new expenses.
- A container to display the ListView.

The code still resides inside the state class.

```

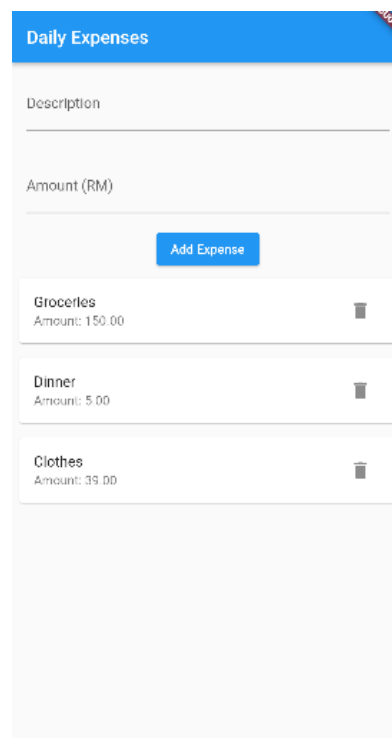
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text('Daily Expenses'),
    ),
    body: Column(
      children: [
        Padding(
          padding: const EdgeInsets.all(16.0),
          child: TextField(
            controller: descriptionController,
            decoration: InputDecoration(
              labelText: 'Description',
            ),
          ),
        ),
        Padding(
          padding: const EdgeInsets.all(16.0),
          child: TextField(
            controller: amountController,
            decoration: InputDecoration(
              labelText: 'Amount (RM)',
            ),
          ),
        ),
        ElevatedButton(
          onPressed: _addExpense,
          child: Text('Add Expense'),
        ),
        Container(
          child: _buildListView(),
        ),
      ],
    ),
  );
}

```

For the last step, implement the `_buildListView()` widget. This method will use the `ListView.builder`. They display the list of expenses as cards with descriptions, amounts, and delete buttons.

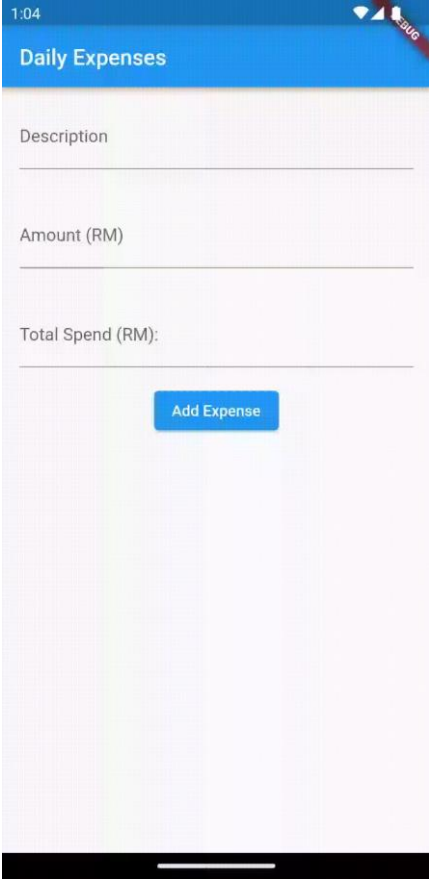
```
Widget _buildListView() {  
  return Expanded(  
    child: ListView.builder(  
      itemCount: expenses.length,  
      itemBuilder: (context, index) {  
        return Card(  
          margin: EdgeInsets.all(8.0),  
          child: ListTile(  
            title: Text(expenses[index].description),  
            subtitle: Text('Amount: ${expenses[index].amount}'),  
            trailing: IconButton(  
              icon: Icon(Icons.delete),  
              onPressed: () => _removeExpense(index),  
            ),  
          ),  
        );  
      },  
    ),  
  );  
}
```

Run the code. Input the field and click *Add Expense* button to add an expense in the list. You can also delete an expense by clicking the bin icon. The output should be similar as below:



Lab Task – Total amount spend

- 1) Add another Textfield to display the total amount of spending
- 2) The Total Spend should sum the total spending that user key in
- 3) Update the total spending whenever the user add or remove the spending list as shown in the output example below



The screenshot shows a mobile application interface for tracking daily expenses. At the top, there is a blue header bar with the text "Daily Expenses". Below the header, there are three text input fields stacked vertically. The first field is labeled "Description", the second is labeled "Amount (RM)", and the third is labeled "Total Spend (RM):". Below these fields is a blue button with the text "Add Expense". The status bar at the top of the screen shows the time "1:04" and some icons. A red "BUG" label is visible in the top right corner of the app's header area.