

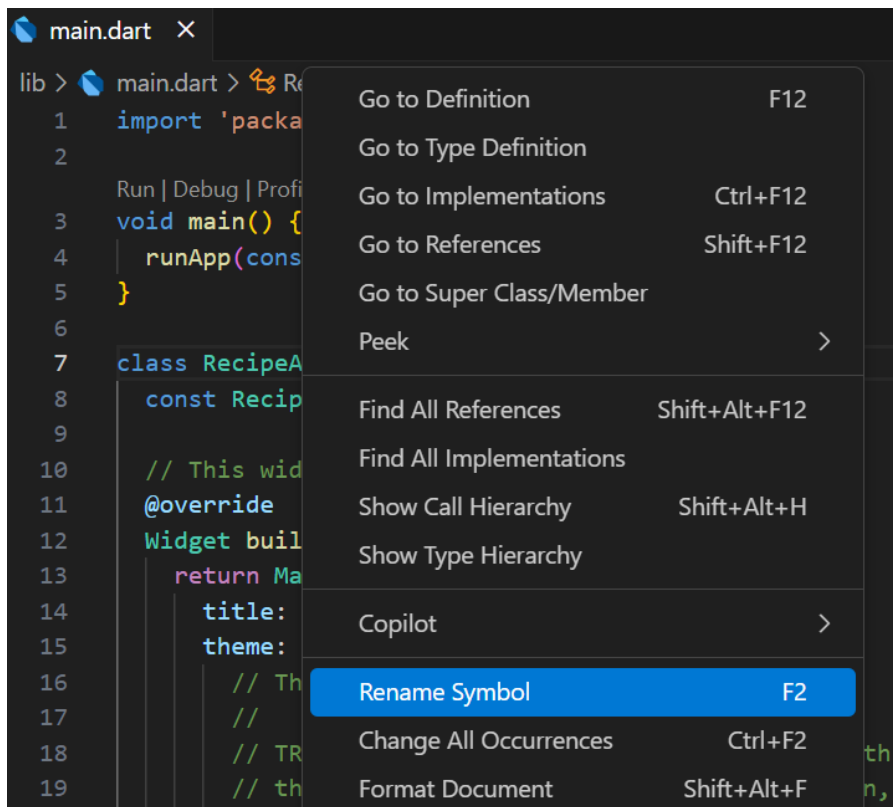


Pre-Condition:

You need to create and run default Flutter App first!

1. Rename MyApp to RecipeApp

Right click on MyApp and choose **Rename Symbol**



2. Styling the app – Customize widget appearance

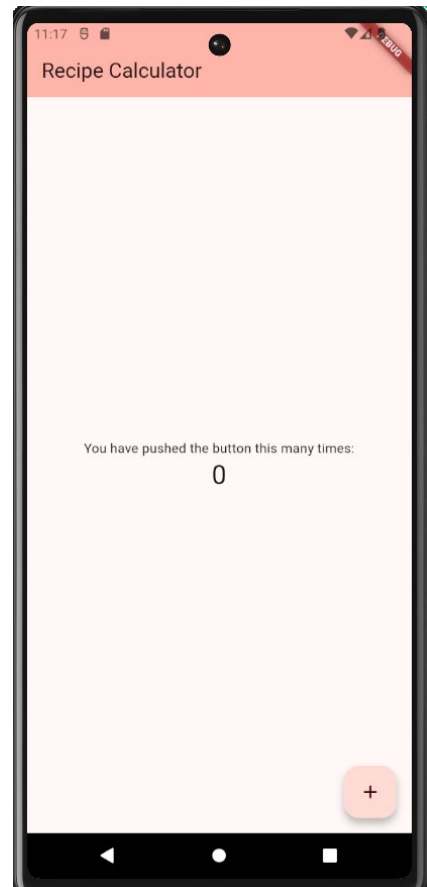
a) Replace class RecipeApp's **build ()** with:

```
Widget build(BuildContext context) {  
  return MaterialApp(  
    title: 'Recipe Calculator',  
    theme: ThemeData(  
      colorScheme: ColorScheme.fromSeed(  
        seedColor: Colors.red, // Use red as the seed color  
        secondary: Colors.grey, // you can still define secondary.  
      ),  
      useMaterial3: true, // Enable Material 3  
    ),  
    home: const MyHomePage(title: 'Recipe Calculator'),  
  );  
}
```

b) After editing, click on save

c) If your emulator did not reload by itself, go to your terminal, and hit Shift + R

d) You'll get the display as shown in the figure.





Notes:

Let's break down this Flutter build method step by step:

1. Widget build(BuildContext context) { ... }

- This is the standard build method required for all Flutter widgets.
- BuildContext context provides information about the widget's position within the widget tree, allowing access to themes, media queries, and other context-dependent data.
- The method returns a Widget, which defines the UI that this part of the widget tree will display.

2. return MaterialApp(...)

- MaterialApp is the root widget for a Material Design application in Flutter.
- It sets up the basic structure and configuration for a Material app, including theming, routing, and localization.

3. title: 'Recipe Calculator'

- This sets the title of the app, which is typically displayed in the operating system's task switcher or app manager.

4. theme: ThemeData(...)

- This defines the overall theme for the app using a ThemeData object.
- ThemeData allows you to customize the visual appearance of your app's widgets.

5. colorScheme: ColorScheme.fromSeed(...)

- ColorScheme.fromSeed is a way to generate a complete ColorScheme based on a single "seed" color.
- seedColor: Colors.red tells Flutter to generate a color palette based on the red color. Material 3 then uses algorithms to create a full set of colors that work well together.
- secondary: Colors.grey this allows you to override certain colors that the seed color method generated. If you remove this line, then the seed color will generate all of the colors.
- This is the preferred way to handle colors in Material 3, ensuring consistency and accessibility.

6. useMaterial3: true

- This enables Material 3 design for the app.
- Material 3 is Google's latest design system, and setting this to true ensures that your app uses the latest visual guidelines and features.

7. home: const MyHomePage(title: 'Recipe Calculator')

- This sets the home screen of the app to an instance of the MyHomePage widget.
- MyHomePage is likely a custom widget defined elsewhere in the code.
- const MyHomePage(...) creates a constant instance of MyHomePage, which can improve performance.
- title: 'Recipe Calculator' passes the title string to the MyHomePage widget.

In summary, this code creates a Flutter Material Design application with:

- A title of "Recipe Calculator."
- A Material 3 theme based on the colour red, with the secondary colour set to grey.
- The MyHomePage widget as the home screen.

3. Clearing the app

a) Replace **class _MyHomePageState** with:

```
class _MyHomePageState extends State<MyHomePage> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        backgroundColor: Theme.of(context).colorScheme.inversePrimary,
        title: Text(widget.title),
      ),
      body: Container());
  }
}
```

- b) After editing, click on save
- c) If your emulator did not reload by itself, go to your terminal, and hit **Shift + R**
- d) You'll get the display as shown in the figure.



Notes:

This code snippet defines a stateful widget's state class in Flutter, specifically for a page called MyHomePage. Let's break it down step by step:

1. `_MyHomePageState` extends `State<MyHomePage>`

- **`_MyHomePageState`**: This declares a class named `_MyHomePageState`. The underscore prefix (`_`) makes it a **private** class within the current Dart file, meaning it can only be used within that file.
- **`extends State<MyHomePage>`**: This indicates that `_MyHomePageState` inherits from the `State` class. The `State` class is a core part of Flutter's stateful widget mechanism.
- **`<MyHomePage>`**: This is a generic type parameter. It specifies that this `State` class is specifically associated with the `MyHomePage` widget. This means that `_MyHomePageState` is the state that manages the mutable data and behavior of the `MyHomePage` widget.

2. `@override`

- This is an annotation that tells the Dart compiler that the following method (`build`) is overriding a method from its superclass (`State`). It's good practice to use `@override` to catch potential errors if you accidentally misspell a method name.

3. Widget build(BuildContext context)

- **Widget build(BuildContext context):** This is the essential method in the State class. It's called every time Flutter needs to rebuild the widget.
- **Widget:** The build method must return a Widget. This widget represents the user interface that this state class will render.
- **BuildContext context:** This is a special object that provides information about the widget's location in the widget tree. It's used to access things like the theme, media queries, and other context-related data.

4. return Scaffold(...)

- **return Scaffold(...):** The build method returns a Scaffold widget. The Scaffold is a fundamental layout widget in Flutter that provides a basic app structure, including an AppBar, body, and other common UI elements.

5. AppBar(...)

- **AppBar(...):** The AppBar widget creates the app's top bar.
- **backgroundColor: Theme.of(context).colorScheme.inversePrimary:** This sets the background color of the AppBar.
 - Theme.of(context) gets the current theme of the app.
 - colorScheme accesses the app's color scheme, which defines various colors used in the app.
 - inversePrimary is a specific color from the color scheme, often used for elements that should stand out against the primary background.
- **title: Text(widget.title):** This sets the title of the AppBar.
 - widget is a property of the State class that provides access to the associated MyHomePage widget.
 - widget.title accesses the title property of the MyHomePage widget, assuming it has a title property defined.
 - Text(...) creates a text widget to display the title.

6. body: Container()

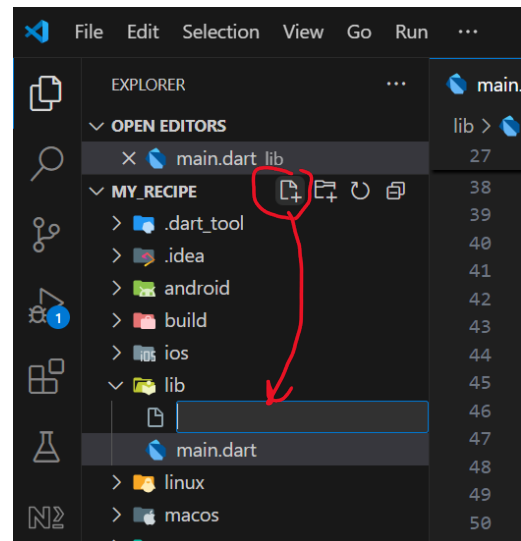
- **body: Container():** This sets the main content area of the Scaffold to a Container widget.
- **Container():** In this case, the Container is empty, meaning it doesn't display anything. This is likely a placeholder that will be replaced with other widgets to build the actual content of the page.

In summary, this code creates a basic Flutter page with an AppBar and an empty Container as its body. The AppBar's title is taken from the MyHomePage widget's title property, and its background color is determined by the app's theme.

4. Building a recipe list

- Recipe main data structure will be used for this app.
- Create new file name **recipe.dart** in the lib folder.
- Add the following class to the file:

```
class Recipe {
  String label;
  String imageUrl;
  Recipe(this.label, this.imageUrl);
}
```



Notes:

This code defines a simple **Dart class** called Recipe. Let's break it down:

1. class Recipe { ... }

- class Recipe:** This line declares a new class named Recipe. Classes are blueprints for creating objects (instances) that have specific properties (data) and behaviors (methods).

2. String label;

- String label;:** This declares a **property** (also called a field or instance variable) named label of type String.
 - String specifies that this property will hold text values.
 - label is the name of the property. It's intended to store the name or title of a recipe.

3. String imageUrl;

- String imageUrl;:** This declares another property named imageUrl, also of type String.
 - String indicates that this property will hold text values, specifically a URL (Uniform Resource Locator).
 - imageUrl is designed to store the web address of an image associated with the recipe.

4. Recipe(this.label, this.imageUrl);

- **Recipe(...):** This is the **constructor** of the Recipe class. A constructor is a special method that's called when you create a new object (instance) of the class.
- **(this.label, this.imageUrl):** This part defines the constructor's parameters.
 - this.label and this.imageUrl are parameters that receive values when a new Recipe object is created.
 - this. is used to refer to the instance variables of the class. It assigns the values passed in as parameters to the respective label and imageUrl properties of the newly created Recipe object.

In summary, this Recipe class creates a data structure that can hold information about a recipe, specifically its label (name) and an image URL. It allows you to create Recipe objects.

d) Add the following method into the **Recipe** class:

```
static List<Recipe> samples = [
  Recipe('Spaghetti and Meatballs',
    'assets/2126711929_ef763de2b3_w.jpg'),
  Recipe('Tomato Soup',
    'assets/27729023535_a57606c1be.jpg'),
  Recipe('Grilled Cheese',
    'assets/3187380632_5056654a19_b.jpg'),
  Recipe('Chocolate Chip Cookies',
    'assets/15992102771_b92f4cc00a_b.jpg'),
  Recipe('Taco Salad',
    'assets/8533381643_a31a99e8a6_c.jpg'),
  Recipe('Hawaiian Pizza',
    'assets/15452035777_294cefced5_c.jpg'),
];
```




Notes:

This Dart code defines a **static list** named `samples` that contains `Recipe` objects. Let's break it down:

1. `static List<Recipe> samples = [...]`

- **static:** This keyword indicates that the `samples` list is a **class-level variable**. This means:
 - It belongs to the `Recipe` class itself, not to any specific instance (object) of the `Recipe` class.
 - You can access it directly using `Recipe.samples` without needing to create a `Recipe` object.
- **List<Recipe>:** This declares a list that can only hold objects of type `Recipe`.
 - `List` is a built-in Dart data structure for storing ordered collections of elements.
 - `<Recipe>` specifies that the list is a **generic list**, meaning it's specialized to hold `Recipe` objects.
- **samples:** This is the name of the list.
- **[...]:** This is the **list literal** syntax, which is used to create and initialize a list with elements.

2. `Recipe('Spaghetti and Meatballs', 'assets/2126711929_ef763de2b3_w.jpg'),`

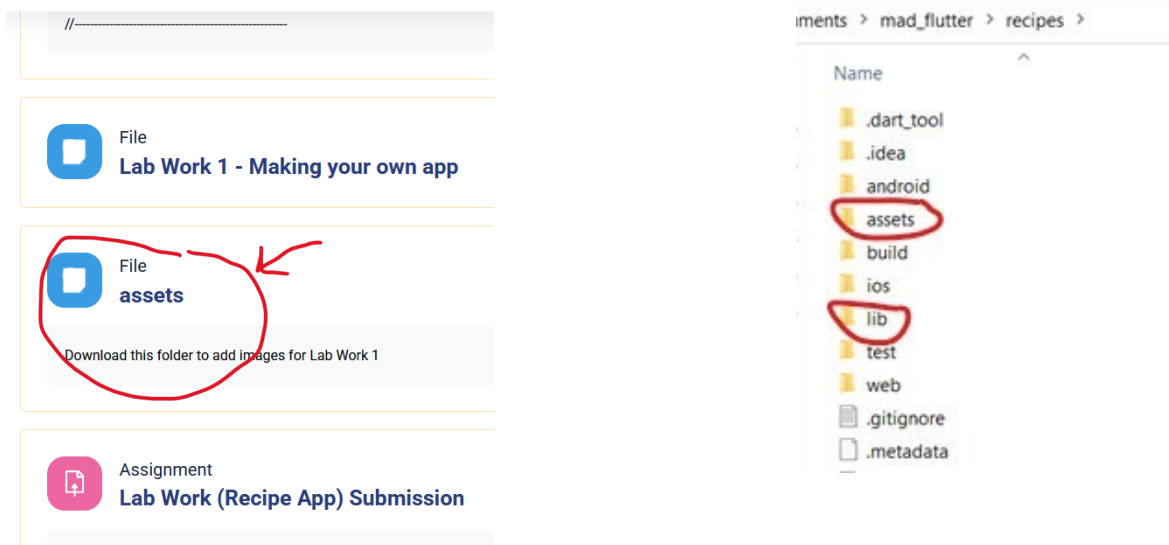
- **Recipe(...):** This calls the constructor of the `Recipe` class to create a new `Recipe` object.
- **'Spaghetti and Meatballs':** This is the first argument passed to the constructor, which represents the label of the recipe.
- **'assets/2126711929_ef763de2b3_w.jpg':** This is the second argument passed to the constructor, which represents the `imageUrl` of the recipe.
- **,:** This comma separates the elements within the list.

3. The rest of the `Recipe(...)` calls follow the same pattern:

- Each `Recipe(...)` call creates a new `Recipe` object with a specific label and image URL.
- These `Recipe` objects are added as elements to the `samples` list.

In summary, this code creates a list called `samples` that contains six `Recipe` objects. Each `Recipe` object represents a different recipe with its name and an image file path (presumably stored in the `assets` folder of the project).

e) To add images on the list that you've created, copy folder **assets** from your **VLE**. Paste the folder same level as your **lib** folder. Later, the app will be able to find the images when you run it.



f) Open **pupspec.yaml** (It's located in your **Recipe** project folder). Under the line **# To add assets to your application** then add the given code:

```
assets:
- assets/
```

Notes:

This code snippet is from a Flutter project's `pubspec.yaml` file, specifically the `assets` section. Let's break it down:

assets:

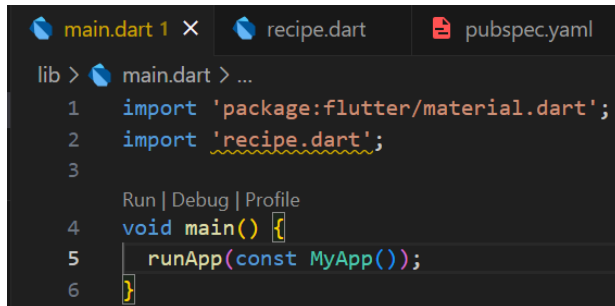
- This is a section in the `pubspec.yaml` file that tells Flutter where to find assets (like images, fonts, or other files) that your app needs to use.

- assets/

- **-:** This dash indicates that `assets/` is an item in a list. In YAML, dashes are used to define list items.
- **assets/:** This specifies a **directory** named `assets` in your project's root directory. The trailing slash `/` is crucial; it tells Flutter to include *all* files and subdirectories within the `assets` folder.

g) To display the list go to file **main.dart** and add

```
import 'recipe.dart';
```



The screenshot shows an IDE with three tabs: main.dart 1, recipe.dart, and pubspec.yaml. The main.dart file is open, showing the following code:

```
lib > main.dart > ...  
1 import 'package:flutter/material.dart';  
2 import 'recipe.dart';  
3  
Run | Debug | Profile  
4 void main() {  
5   runApp(const MyApp());  
6 }
```

Notes:

The line `import 'recipe.dart';` in a Dart file is a directive that tells the Dart compiler to include the code from another Dart file named `recipe.dart` into the current file.

Here's a breakdown:

- **import:** This is a keyword in Dart that is used to bring in code from other libraries or files.
- **'recipe.dart':** This is a string literal that specifies the path to the Dart file you want to import. In this case, it means:
 - The file is named `recipe.dart`.
 - The file is located in the same directory as the current Dart file (because it's a relative path without any directory prefixes).

What This Does:

When you import `recipe.dart`, you make all the public declarations (classes, functions, variables, etc.) from `recipe.dart` available for use in your current Dart file.

h) In **main.dart** file, in **class _MyHomePageState**, replace the **child: Container()**, with:

```
body: ListView.builder(
  itemCount: Recipe.samples.length,
  itemBuilder: (BuildContext context, int index) {
    return Text(Recipe.samples[index].label);
  },
),
```

- i) After editing, click on save
- j) If your emulator did not reload by itself, go to your terminal, and hit **Shift + R**
- k) You'll get the display as shown in the figure below.



Notes:

This code snippet is part of a Flutter build method, likely within a Scaffold's body property. It creates a scrollable list of text items using `ListView.builder`. Let's break it down:

1. **body: ListView.builder(...)**

- **body::** This specifies the main content area of the Scaffold.
- **ListView.builder(...):** This is a Flutter widget that efficiently builds a scrollable list of items on demand. It's particularly useful for lists with a large number of

items because it only creates the widgets that are currently visible on the screen.

2. `itemCount: Recipe.samples.length`

- **`itemCount::`** This named parameter specifies the total number of items in the list.
- **`Recipe.samples.length`:** This gets the length (number of elements) of the `Recipe.samples` list. We've previously discussed that `Recipe.samples` is a static list of `Recipe` objects. This means the list will display one text item for each recipe in the samples list.

3. `itemBuilder: (BuildContext context, int index) { ... }`

- **`itemBuilder::`** This named parameter takes a function that is called for each item in the list to build the corresponding widget.
- **`(BuildContext context, int index) { ... }`:** This is an anonymous function (lambda expression) that takes two arguments:
 - **`BuildContext context`:** The build context, which provides information about the widget's location in the widget tree.
 - **`int index`:** The index of the current item being built. It starts from 0 and goes up to `itemCount - 1`.
- **`return Text(Recipe.samples[index].label);`:** This is the body of the `itemBuilder` function.
 - **`Recipe.samples[index]`:** This accesses the `Recipe` object at the specified index in the `Recipe.samples` list.
 - **`.label`:** This accesses the `label` property of the `Recipe` object, which is a `String` representing the recipe's name.
 - **`Text(...)`:** This creates a `Text` widget to display the recipe's label.
 - **`return`:** This returns the `Text` widget, which will be displayed as an item in the `ListView`.

In summary, this code creates a vertical, scrollable list where each item displays the label (name) of a recipe from the `Recipe.samples` list.

5. Putting the list on the card

- a) Open **main.dart** file
- b) At the bottom of **class _MyHomePageState** add the following code:

```
Widget buildRecipeCard(Recipe recipe) {
  return Card(
    child: Column(
      children: <Widget>[
        Image(image: AssetImage(recipe.imageUrl)),
        Text(recipe.label),
      ],
    ),
  );
}
```

Notes:

This Dart code defines a function called `buildRecipeCard` that takes a `Recipe` object as input and returns a `Widget` representing a recipe card. Let's break it down:

1. Widget buildRecipeCard(Recipe recipe) { ... }

- **Widget buildRecipeCard(Recipe recipe):** This declares a function named `buildRecipeCard` that:
 - Takes a single parameter `recipe` of type `Recipe`.
 - Returns a `Widget`. This means the function will create and return a Flutter UI element.

2. return Card(...)

- **return Card(...):** The function returns a `Card` widget. A `Card` is a Flutter widget that displays content with rounded corners and a slight elevation, giving it a card-like appearance.

3. child: Column(...)

- **child: Column(...):** The `Card` widget contains a `Column` widget as its child. A `Column` widget arranges its children vertically, one below the other.

4. children: <Widget>[...]

- **children: <Widget>[...]:** The `Column` widget has a `children` property, which is a list of `Widget` objects that will be displayed inside the column.
- **<Widget>[...]:** This specifies that the list will contain widgets.

5. Image(image: AssetImage(recipe.imageUrl))

- **Image(...):** This creates an Image widget, which is used to display images.
- **image: AssetImage(recipe.imageUrl):** This sets the image property of the Image widget to an AssetImage.
 - **AssetImage(...):** This loads an image from the project's assets folder.
 - **recipe.imageUrl:** This retrieves the image URL from the recipe object passed to the function. It assumes that the imageUrl property of the Recipe object contains the path to the image in the assets folder.

6. Text(recipe.label)

- **Text(...):** This creates a Text widget, which is used to display text.
- **recipe.label:** This retrieves the label (name) of the recipe from the recipe object passed to the function.

In summary, this buildRecipeCard function takes a Recipe object and creates a Card widget that displays the recipe's image and label in a vertical column.

c) Go to **class _MyHomePageState** and update the **ListView** itemBuilder's **return** statement to this:

```
return buildRecipeCard(Recipe.samples[index]);
```

Notes:

This line of code, `return buildRecipeCard(Recipe.samples[index]);`, is typically found within the `itemBuilder` function of a `ListView.builder` in Flutter. Let's break it down:

1. **return**

- This keyword indicates that the function (in this case, the `itemBuilder` function) will return a value.

2. **buildRecipeCard(Recipe.samples[index])**

- **buildRecipeCard(...):** This calls the `buildRecipeCard` function that we discussed earlier. Remember, this function takes a `Recipe` object and returns a `Widget` representing a recipe card.
- **Recipe.samples[index]:** This is the argument passed to the `buildRecipeCard` function.
 - **Recipe.samples:** This accesses the static `samples` list from the `Recipe` class. As we've seen, this list contains `Recipe` objects.
 - **[index]:** This accesses the `Recipe` object at the specified index within the `Recipe.samples` list.

- d) After editing, click on save
- e) If your emulator did not reload by itself, go to your terminal, and hit **Shift + r**
- f) You'll get the display as shown in the figure given.

