

**SEB-312 Mobile Application Development**

**LAB # 04**

**LAB Title**

|  |
| --- |
| Exploring InkWell widgets and its usages. Explore ScrollView, ListView and ListTile widget of flutter. |

**Assessment of CLO: 03, PLO: 05**

|  |  |  |  |
| --- | --- | --- | --- |
| **Student Name:** |  | | |
| **Roll No.** |  | | |
| **Semester** |  | **Session** |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **S. No.** | **Perf. Level**  **Criteria** | **Excellent**  **(2.5)** | **Good**  **(2)** | **Satisfactory**  **(1.5)** | **Needs Improvement**  **(0 ~ 1)** | **Marks Obtained** |
| **1** | Project Execution & Implementation | Fully functional, optimized, and well-structured. | Minor errors, mostly functional. | Some errors, requires guidance. | Major errors, non-functional, or not Performed. |  |
| **2** | Results & Debugging  Or Troubleshooting | Accurate results with effective debugging  Or Troubleshooting. | Mostly correct, some debugging Or Troubleshooting needed. | Partial results, minimal debugging  Or Troubleshooting. | Incorrect results, no debugging Or Troubleshooting, or not attempted. |  |
| **3** | Problem-Solving & Adaptability  (VIVA) | Creative approach, efficiently solves challenges. | Adapts well, minor struggles. | Some adaptability, needs guidance. | Lacks innovation or no innovation, unable to solve problems. |  |
| **4** | Report Quality & Documentation | Clear, structured, with detailed visuals. | Mostly clear, minor gaps. | Some clarity issues, missing details. | Poorly structured, lacks clarity, or not submitted. |  |
| **Total Marks Obtained Out of 10** | | | | | |  |

**Experiment evaluated by**

|  |  |  |  |
| --- | --- | --- | --- |
| **Instructor’s Name** | **Sidra Khatoon** | | |
| **Date** |  | **Signature** |  |

**Objective**

The objective of lab is exploring inkwell widgets and its usages. Also learn how to use ScrollView, ListView and ListTile widget.

**Instructions**

You have to perform the following tasks yourselves. Raise your hand if you face any difficulty in understanding and solving these tasks. **Plagiarism** is an abhorrent practice and you should not engage in it.

**What is InkWell in Flutter?**

The InkWell is a rectangular area of a Material widget that responds to touch events by displaying a clipped splash. The Material widget is responsible for the ink effects that are displayed when a touch event occurs. The Material refers to the area where the ink reactions are painted.

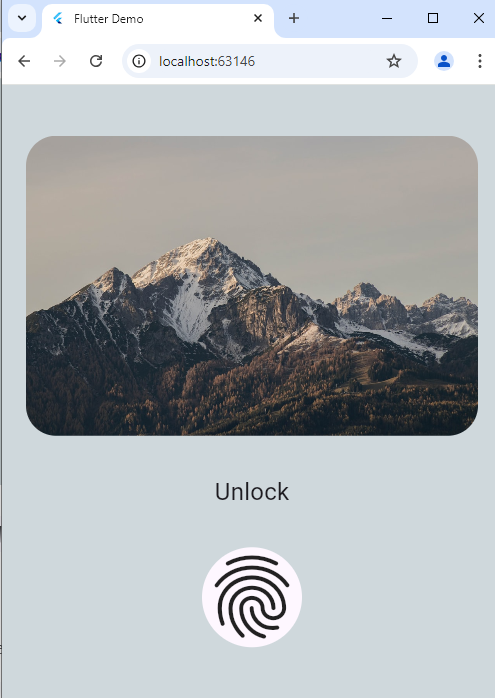
The InkWell needs a Material widget as an ancestor for the ink reactions to display correctly. The splash effect will not be visible if the InkWell class is used with opaque widgets, such as a Container with color. However, we can get around this by replacing an opaque widget with an Ink widget.

InkWell has several properties that can be used to display a customized response to user touch events.

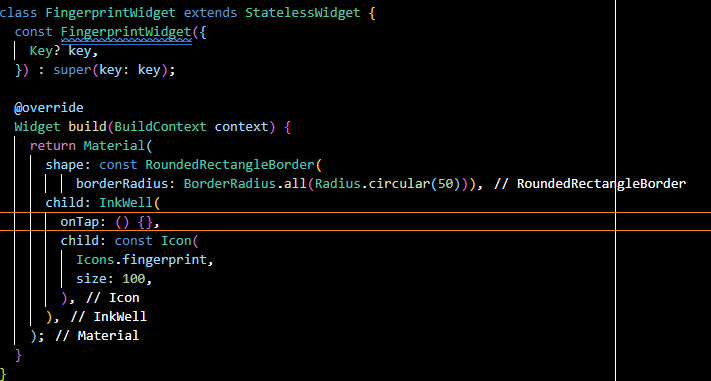
we’ll customize an app consisting of three widgets displayed in a vertical column:

image\_widget.dart: the image

* custom\_button\_widget.dart: the text
* fingerprint\_widget.dart: the fingerprint icon

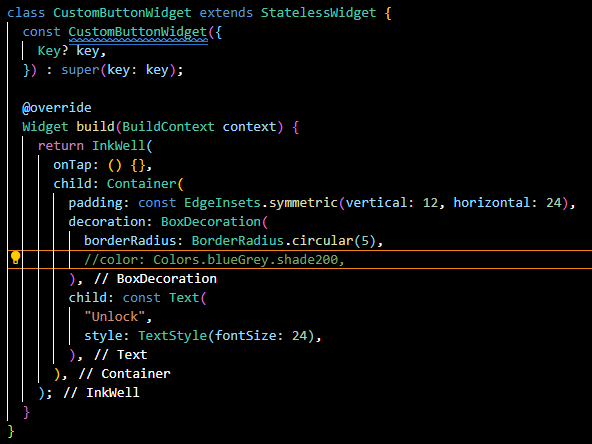


**Adding InkWell to an app’s button, text, and image**

To start, we’ll update the fingerprint\_widget.dart file:

We add an InkWell widget as the child of a Material widget. Then, we add an onTap handler. The InkWell splash effect is only visible when a gesture callback, onTap in this case, is added. With this code, the splash effect will be created when the fingerprint icon is tapped.

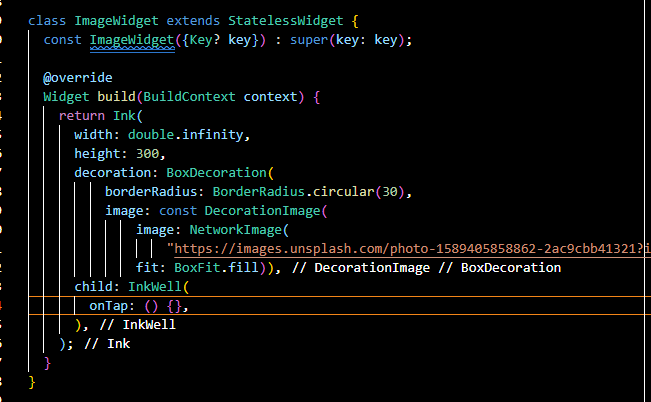
Next, we’ll update the custom\_button\_widget.dart file:



This widget contains an opaque Container widget. We wrap the Container widget with an InkWell widget and add the onTap handler. With this code, the splash effect will be created when the text is tapped. But, adding color to the Container will hide this effect, since the Container is opaque.

To address this issue, we replace the Container widget with an Ink widget. This ensures that the color or decoration applied to the Ink also paints in the InkWell and that the splash effect is visible.

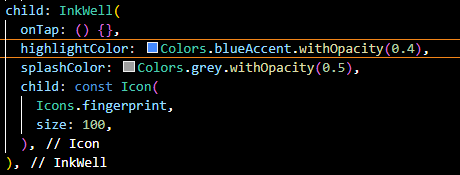
Now, we’ll update the image\_widget.dart file:



We use the Ink widget to display an image and add an InkWell widget as a child. The Ink has an image and decoration that paints without hiding the splash effect of the InkWell.

**Modifying the InkWell highlight and splash color**

When we tap the InkWell widget, the highlight color is immediately painted over the widget. Then, the splash color is painted on top of the highlight, creating a ripple effect.



We give the InkWell any color for highlightColor and splashColor. We also add opacity to the colors to ensure some transparency when the colors paint over the widgets. On a touch event, the ink paints a rectangular shape that does not match the shape of the image or fingerprint areas. So, we will modify the InkWell‘s rectangle’s shape.

**Customizing the InkWell border**

An InkWell has a rectangular shape. When tapped, the highlight color fills the rectangle. We use the borderRadius property or create a custom shape by using the customBorder property to clip the rectangle’s corners. The borderRadius property is effective only if the customBorder is not assigned. We’ll use the borderRadius property to modify the shape of the InkWell box.

First, we update the image\_widget.dart and custom\_button\_widget.dart files:



Then, we update the fingerprint\_widget.dart file:



These updates add a circular borderRadius to the InkWell widget. The borderRadius circularly clips the rectangle’s corners with the specified radius, (in this case, 30deg and 50deg).

**Responding to user gestures with InkWell**

Responding to user gestures is one way to make an app more interactive. InkWell offers properties that can be used to respond to user gestures.

Here are some common user gestures and the callback that is triggered when the user gesture occurs:

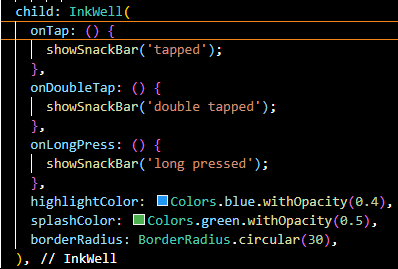
* Tap: a user touches the screen briefly with their fingertip. Tap callbacks include onTap, onTapDown, and onTapCancel
* Double-tap: a user taps the same location on the screen twice in quick succession. The onDoubleTap callback is used
* Long press: a user touches one location on the screen for an extended period. The onLongPress callback is used

The onTapDown callback is triggered when a user makes contact with the screen. When the contact results in a gesture that is not a tap, double-tap, or long press, the onTapCancel is triggered as a default.

We’ll create a method, showSnackBar, that takes in a String gesture. The function creates a SnackBar with Text to show the triggered gesture. It hides the previous displaying snack bar, if any, and displays a new snack bar:



We’ll handle the gesture callbacks in the image\_widget.dart file:



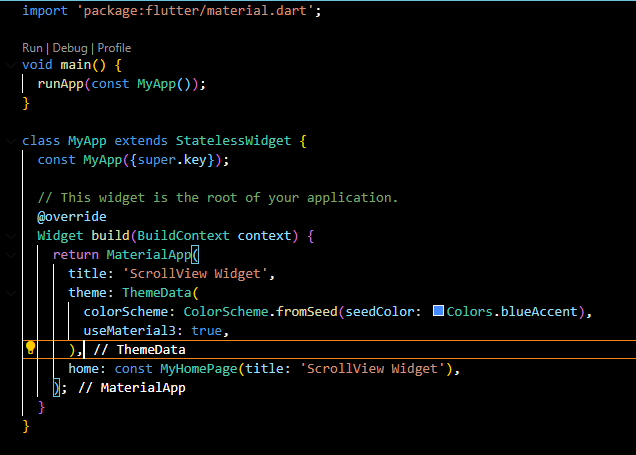
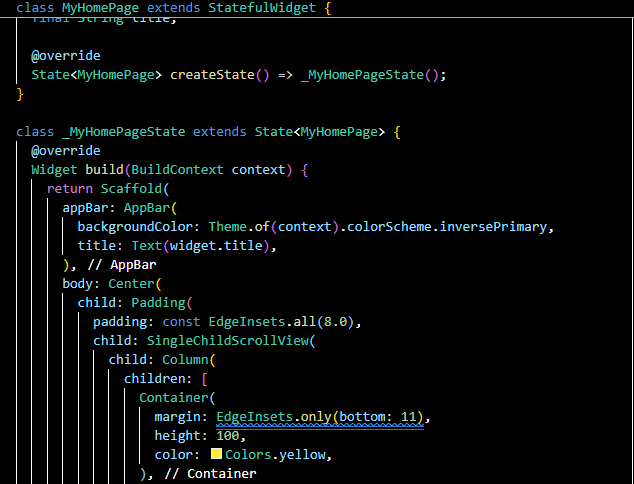
When a user triggers a gesture event, a snack bar is displayed with the corresponding string gesture.

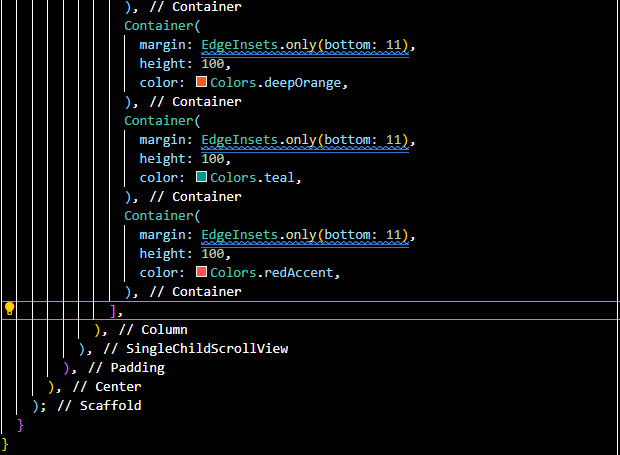
**Keypoints of using InkWell**

* InkWell is designed for rectangular areas.
* The InkWell widget must have a Material widget as its parent
* The InkWell’s splash will not automatically update to match changes in the size of its parent [Material](https://api.flutter.dev/flutter/material/Material-class.html) widget
* The InkWell widget’s color may be set via the color property of the Material widget
* Using an opaque widget with images or decorations between the Material widget and InkWell widget will hide the ripple effect of an InkWell
* The Ink widget can replace opaque widgets. It ensures that the images and decorations are displayed correctly while keeping the ripple effect visible
* The ripple effect on an InkWell will only be visible after a touch event handler, like onTap*,* has been added

**ScrollView Widget:**

A ScrollView widget in Flutter allows you to create complex scrolling effects and layouts. You can use it to create scrollable views with multiple slivers, each having its own behavior. In this lab, we are going to implement the ScrollView widget.

**** ****

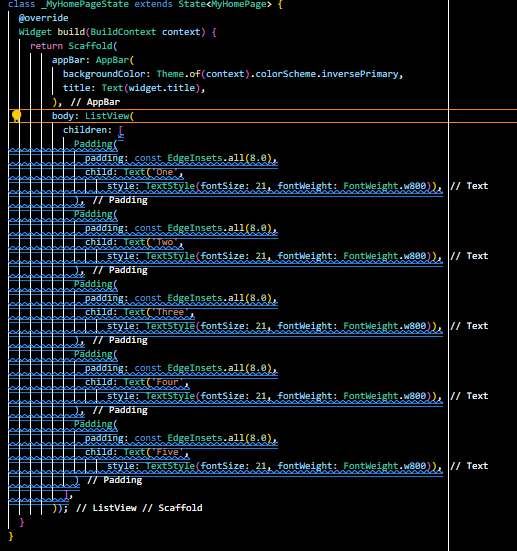
****

One of the key benefits of using a SingleChildScrollView is its ability to handle overflow. When content exceeds the screen's width or height, it can lead to an overflow error. The SingleChildScrollView automatically wraps its child in a scrollable interface, preventing overflow and allowing users to access all the content by scrolling.

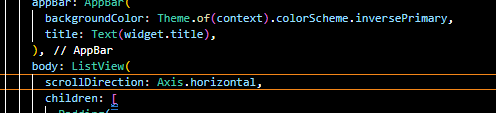
**ListView Widget:**

ListView in flutter is a widget used to display items in linear manner. For example, list view is used in apps like foodpanda to display a list of restaurants. Since it is a scrollable widget we can display multiple items on the same screen If scroll direction is vertical the children will be arranged one after another form top to the bottom. When the scroll direction is horizontal the children will be arranged from left to right.

This is the default constructor of the ListView class. A ListView simply takes a list of widgets and makes it scrollable. Usually, this is used with a few children as the List will also construct invisible elements in the list, so numerous widgets may render this inefficiently.



For row we have set scroll direction in horizontal axis.



**ListView.builder()**

The builder() constructor constructs a repeating list of widgets. The constructor takes two main parameters:

* An itemCount for the number of repetitions for the widget to be constructed (not compulsory).
* An itemBuilder for constructing the widget which will be generated ‘itemCount‘ times (compulsory).
* If the itemCount is not specified, infinite widgets will be constructed by default.



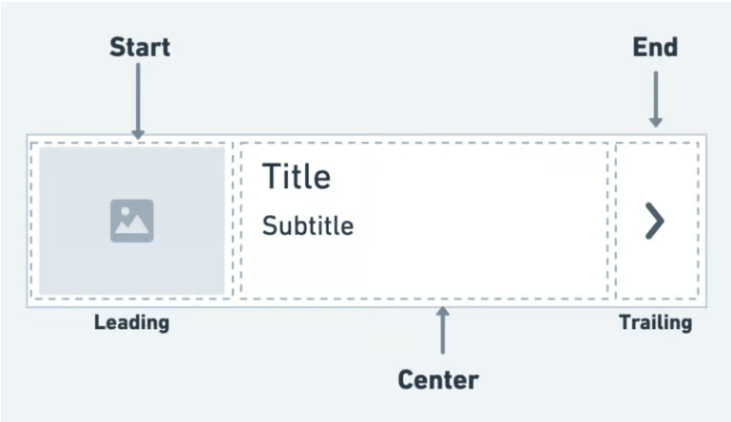
**ListView.separated ()**

The ListView.separated() constructor is used to generate a list of widgets, but in addition, a separator widget can also be generated to separate the widgets. In short, these are two intertwined list of widgets: the main list and the separator list. Unlike the builder() constructor, the itemCount parameter is compulsory here.



**ListTile Widget**

The ListTile widget in Flutter is a UI element that displays related information. It follows the List specification from Material Design. A typical ListTile is divided into three sections; Start, Center, and End. The Start section contains the leading widget; the Center section includes the title and subtitle, and the End section contains the trailing widget.



It is mainly used to populate the scrollable views such as ListView, Column, and Row. For example, you can use the ListTile to show a list of to-do items, emails, navigation options, and more. You can also receive the click event by tapping the ListTile.

**Adding ListTile**

Here is the minimal code to display ListTile inside the ListView widget:

import 'package:flutter/material.dart';

**void main() {**

**runApp(const MyApp());**

**}**

**class MyApp extends StatelessWidget {**

**const MyApp({super.key});**

**// This widget is the root of your application.**

**@override**

**Widget build(BuildContext context) {**

**return MaterialApp(**

**debugShowCheckedModeBanner: false,**

**title: 'Flutter Demo',**

**theme: ThemeData(**

**colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),**

**useMaterial3: true,**

**),**

**home: const MyHomePage(),**

**);**

**}**

**}**

**class MyHomePage extends StatelessWidget {**

**const MyHomePage({super.key});**

**@override**

**Widget build(BuildContext context) {**

**return Scaffold(**

**appBar: AppBar(**

**title: Text("ListTile in Flutter"),**

**backgroundColor: Theme.of(context).colorScheme.inversePrimary,**

**),**

**body: Center(**

**child: ListView(**

**children: [**

**ListTile(**

**leading: Icon(Icons.car\_rental),**

**title: Text('Car'),**

**trailing: Icon(Icons.more\_vert),**

**),**

**ListTile(**

**leading: Icon(Icons.flight),**

**title: Text('Flight'),**

**trailing: Icon(Icons.more\_vert),**

**),**

**ListTile(**

**leading: Icon(Icons.train),**

**title: Text('Train'),**

**trailing: Icon(Icons.more\_vert),**

**)**

**],**

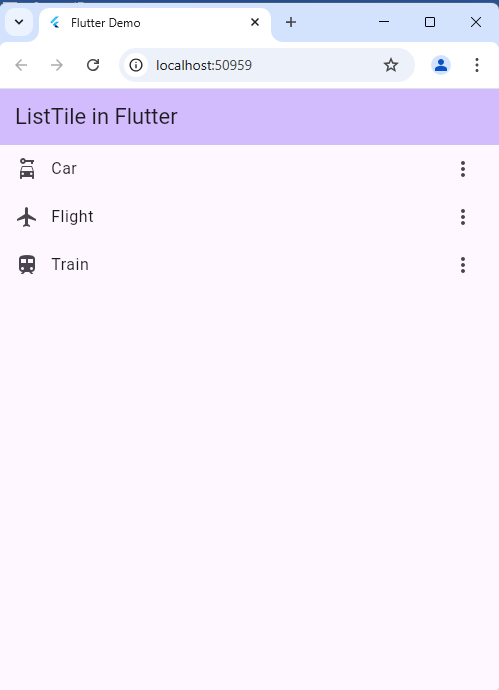
**),**

**),**

**);**

**}**

**}**



When you want to use the ListTile to populate the long list that might come from your backend, you can wrap the single ListTile widget inside the ListView.Builder and show the data inside the ListTile, as shown in the following code:

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

**void main() {**

**runApp(const MyApp());**

**}**

**class MyApp extends StatelessWidget {**

**const MyApp({super.key});**

**// This widget is the root of your application.**

**@override**

**Widget build(BuildContext context) {**

**return MaterialApp(**

**debugShowCheckedModeBanner: false,**

**title: 'Flutter Demo',**

**theme: ThemeData(**

**colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),**

**useMaterial3: true,**

**),**

**home: MyHomePage(),**

**);**

**}**

**}**

**class MyHomePage extends StatelessWidget {**

**MyHomePage({super.key});**

**final List<String> items = List<String>.generate(100, (i) => '$i');**

**@override**

**Widget build(BuildContext context) {**

**return Scaffold(**

**appBar: AppBar(**

**title: Text("ListTile in Flutter"),**

**backgroundColor: Theme.of(context).colorScheme.inversePrimary,**

**),**

**body: Center(**

**child: ListView.builder(**

**itemCount: items.length,**

**itemBuilder: (context, index) {**

**return ListTile(**

**leading: CircleAvatar(**

**backgroundColor: const Color(0xff764abc),**

**child: Text(items[index]),**

**),**

**title: Text('Item ${items[index]}'),**

**subtitle: Text('Item description'),**

**trailing: Icon(Icons.more\_vert),**

**);**

**},**

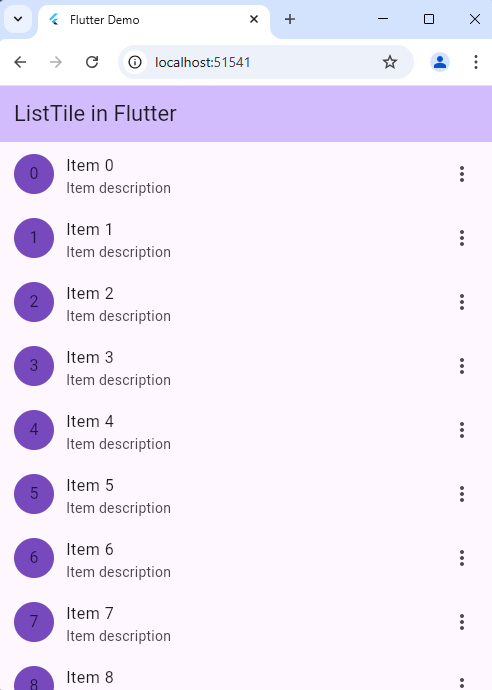
**),**

**),**

**);**

**}**

**}**



**ListTile variations**

There are also other types of ListTile that exist that allow you to perform a specific action on it.

These are:

1. CheckboxListTile
2. RadioListTile
3. SwitchListTile
4. **CheckboxListTile:**

The CheckboxListTile widget is a combination of ListTile and Checkbox widgets. We can use this widget to mark any item as complete — for example; to-do items. By default, the checkbox is displayed on the right side of the ListTile (for left-to-right language).

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

**void main() {**

**runApp(const MyApp());**

**}**

**class MyApp extends StatelessWidget {**

**const MyApp({super.key});**

**// This widget is the root of your application.**

**@override**

**Widget build(BuildContext context) {**

**return MaterialApp(**

**debugShowCheckedModeBanner: false,**

**title: 'Flutter Demo',**

**theme: ThemeData(**

**colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),**

**useMaterial3: true,**

**),**

**home: const MyHomePage(),**

**);**

**}**

**}**

**class Language {**

**String name;**

**bool isChecked;**

**Language(this.name, {this.isChecked = false});**

**}**

**class MyHomePage extends StatefulWidget {**

**const MyHomePage({super.key});**

**@override**

**State<MyHomePage> createState() => \_MyHomePageState();**

**}**

**class \_MyHomePageState extends State<MyHomePage> {**

**final List<Language> languages = [**

**Language('English'),**

**Language('French'),**

**Language('German')**

**];**

**@override**

**Widget build(BuildContext context) {**

**return Scaffold(**

**appBar: AppBar(**

**backgroundColor: Theme.of(context).colorScheme.inversePrimary,**

**title: Text("CheckBox ListTile"),**

**),**

**body: Center(**

**child: ListView.builder(**

**itemCount: languages.length,**

**itemBuilder: (context, index) {**

**return CheckboxListTile(**

**title: Text('${languages[index].name}'),**

**value: languages[index].isChecked,**

**onChanged: (bool? value) {**

**setState(() {**

**languages[index].isChecked = value!;**

**});**

**},**

**secondary: const Icon(Icons.language),**

**);**

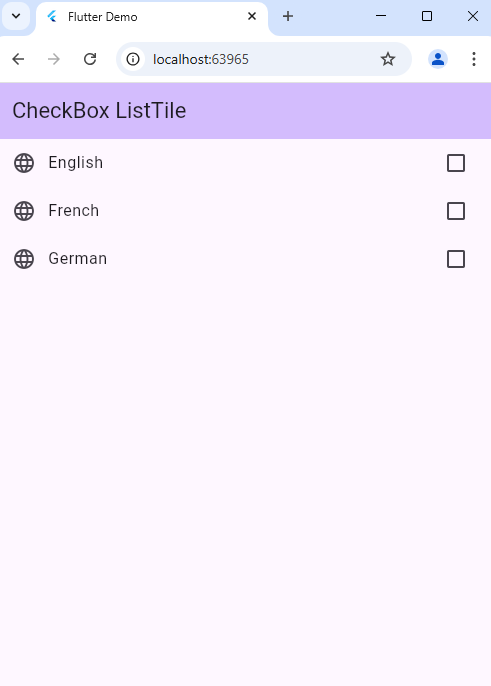
**},**

**)),**

**);**

**}**

**}**



1. **RadioListTile**

The RadioListTile widget is a combination of ListTile and RadioButton widgets — this widget is

used to select a single option from a list of items.

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

**void main() {**

**runApp(const MyApp());**

**}**

**class MyApp extends StatelessWidget {**

**const MyApp({super.key});**

**// This widget is the root of your application.**

**@override**

**Widget build(BuildContext context) {**

**return MaterialApp(**

**debugShowCheckedModeBanner: false,**

**title: 'Flutter Demo',**

**theme: ThemeData(**

**colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),**

**useMaterial3: true,**

**),**

**home: const MyHomePage(),**

**);**

**}**

**}**

**enum Gender { male, female }**

**Gender? \_gender = Gender.male;**

**class MyHomePage extends StatefulWidget {**

**const MyHomePage({super.key});**

**@override**

**State<MyHomePage> createState() => \_MyHomePageState();**

**}**

**class \_MyHomePageState extends State<MyHomePage> {**

**@override**

**Widget build(BuildContext context) {**

**return Scaffold(**

**appBar: AppBar(**

**backgroundColor: Theme.of(context).colorScheme.inversePrimary,**

**title: Text("RadioButton ListTile"),**

**),**

**body: Center(**

**child: ListView(**

**children: [**

**RadioListTile<Gender>(**

**secondary: Icon(Icons.male),**

**controlAffinity: ListTileControlAffinity.trailing,**

**title: const Text('Male'),**

**value: Gender.male,**

**groupValue: \_gender,**

**onChanged: (Gender? value) {**

**setState(() {**

**\_gender = value;**

**});**

**},**

**),**

**RadioListTile<Gender>(**

**secondary: Icon(Icons.female),**

**controlAffinity: ListTileControlAffinity.trailing,**

**title: const Text('Female'),**

**value: Gender.female,**

**groupValue: \_gender,**

**onChanged: (Gender? value) {**

**setState(() {**

**\_gender = value;**

**});**

**},**

**),**

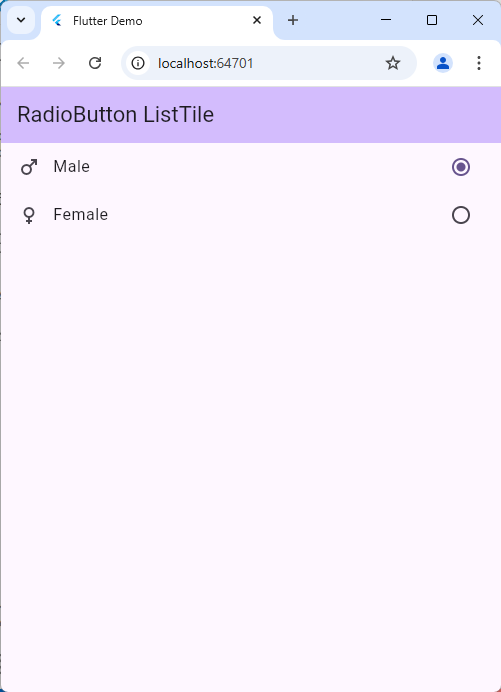
**],**

**)),**

**);**

**}**

**}**



1. **SwitchListTile**

The SwitchListTile widget is a combination of the ListTile and Switch widgets. We can use this widget to

build the UI interaction that allows users to switch on or off app settings.

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

**void main() {**

**runApp(const MyApp());**

**}**

**class MyApp extends StatelessWidget {**

**const MyApp({super.key});**

**// This widget is the root of your application.**

**@override**

**Widget build(BuildContext context) {**

**return MaterialApp(**

**debugShowCheckedModeBanner: false,**

**title: 'Flutter Demo',**

**theme: ThemeData(**

**colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),**

**useMaterial3: true,**

**),**

**home: const MyHomePage(),**

**);**

**}**

**}**

**class Appliance {**

**String name;**

**bool isOn;**

**Appliance(this.name, {this.isOn = false});**

**}**

**class MyHomePage extends StatefulWidget {**

**const MyHomePage({super.key});**

**@override**

**State<MyHomePage> createState() => \_MyHomePageState();**

**}**

**class \_MyHomePageState extends State<MyHomePage> {**

**final List<Appliance> appliances = [**

**Appliance('TV'),**

**Appliance('Fan'),**

**Appliance('Refrigerator'),**

**];**

**@override**

**Widget build(BuildContext context) {**

**return Scaffold(**

**appBar: AppBar(**

**backgroundColor: Theme.of(context).colorScheme.inversePrimary,**

**title: Text("Switch ListTile"),**

**),**

**body: Center(**

**child: ListView.builder(**

**itemCount: appliances.length,**

**itemBuilder: (context, index) {**

**return SwitchListTile(**

**title: Text('${appliances[index].name}'),**

**value: appliances[index].isOn,**

**onChanged: (bool value) {**

**setState(() {**

**appliances[index].isOn = value;**

**});**

**},**

**);**

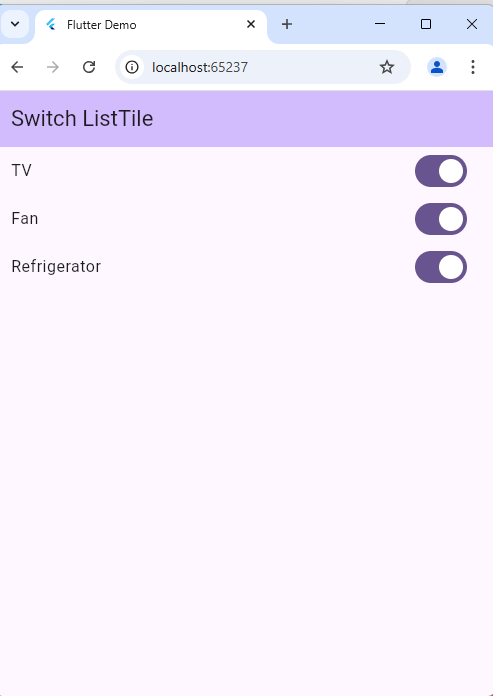
**},**

**)),**

**);**

**}**

**}**

****

**A screenshot of a phone

Description automatically generated**

**Assessment**

**Task 1:** Create a Flutter application that:

* Displays a column of widgets (e.g., Text, Image, Buttons) inside a SingleChildScrollView.
* The scrollable content should exceed the height of the screen, enabling vertical scrolling.
* Add padding and some decoration to make the items stand out.

**Task 2:** Modify the application to use ListView instead of SingleChildScrollView.

* Implement a simple ListView with 50 items, each displaying text or an image.
* Use ListView.builder to dynamically generate list items as the user scrolls.

**Task3:** Modify the ListView implementation to include:

* Different types of items (e.g., text, images, buttons).
* Add dividers between the list items.
* Implement scrolling in both directions (horizontal and vertical).

**Task 2:** Set up a new Flutter project named Contacts App using your preferred development

environment.

* Use ListView**.Builder** to dynamically generate a list of contacts.
* Implement a **ListTile** for each contact, displaying the contact’s name and phone number.
* Add an avatar (using **Icon**) and a trailing icon (phone icon) for each contact.