**Assessment:**

**Lab Task 1: Build a Shopping List App Using Flutter**

**Objective**

Create a simple Shopping List app where users can:

* Add items to the list
* Mark items as purchased
* Delete items from the list

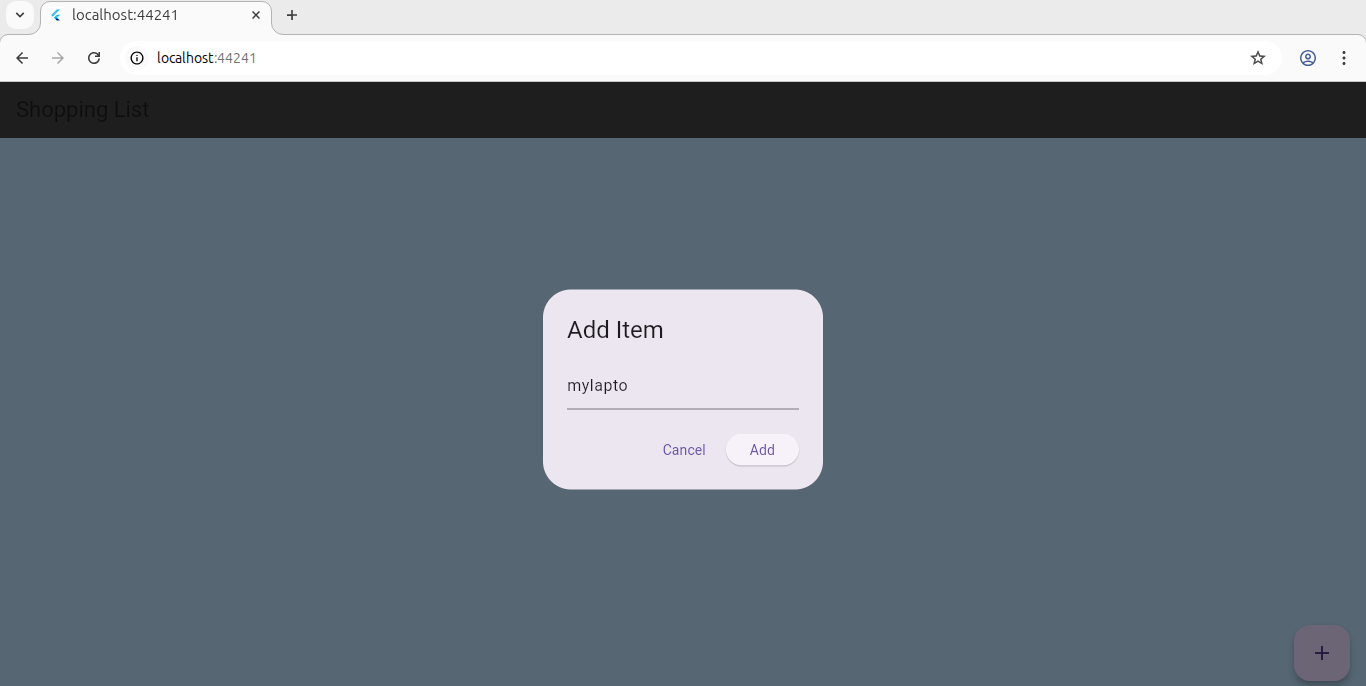
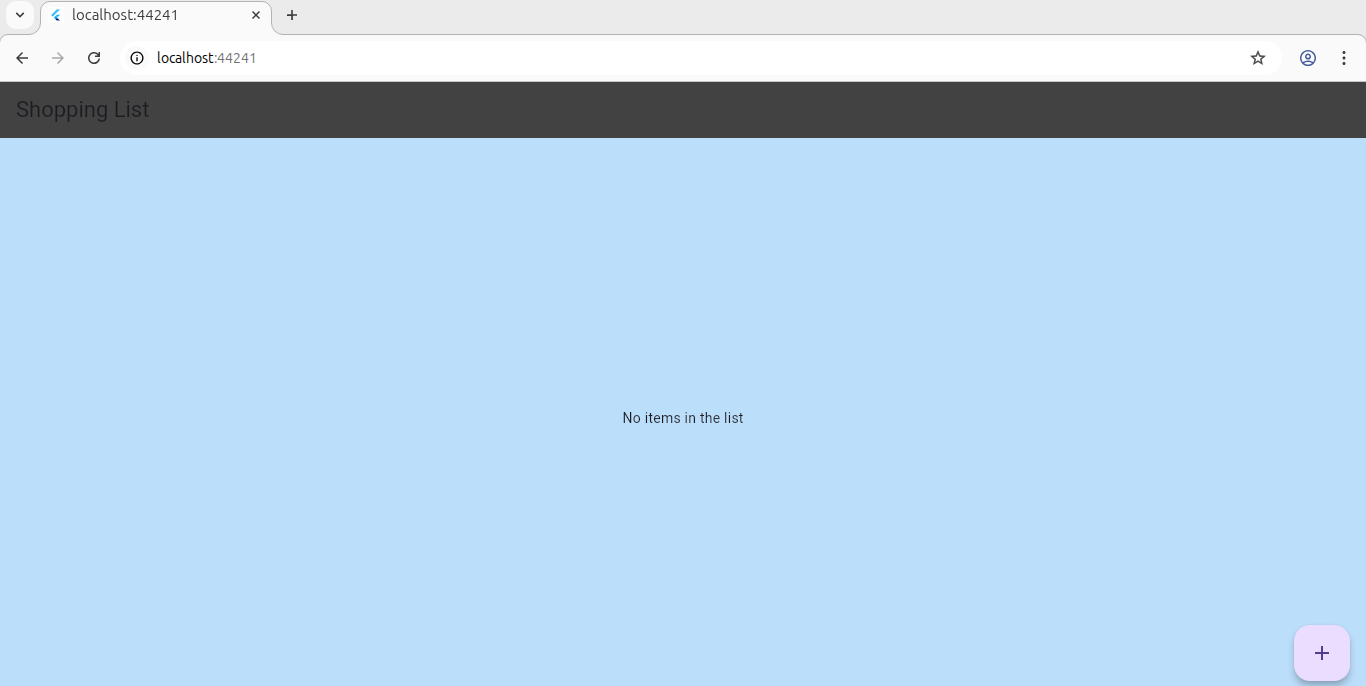
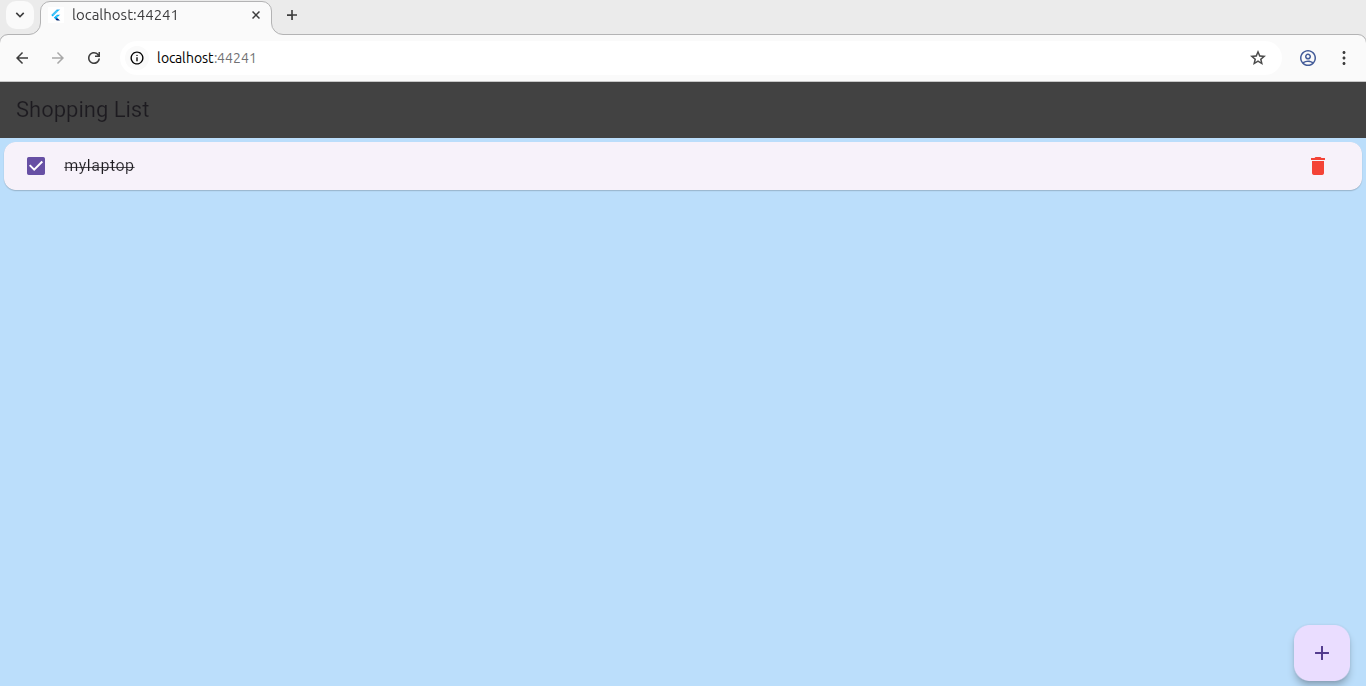
**Requirements**

1. Use StatefulWidget to manage the list.
2. Each item should have a name and a purchased status.
3. Use a TextField in a dialog to add new items.
4. Display items in a ListView.
5. Tap an item or use a checkbox to mark it as purchased (with strikethrough).
6. Include a delete icon to remove items.
7. Use a FloatingActionButton to add items.

Main.dart:

|  |
| --- |
| import 'package:flutter/material.dart';  void main() {  runApp(const MaterialApp(  home: ShoppingListApp(),  debugShowCheckedModeBanner: false,  ));  }  class ShoppingListApp extends StatefulWidget {  const ShoppingListApp({super.key});  @override  State<ShoppingListApp> createState() => \_ShoppingListAppState();  }  class \_ShoppingListAppState extends State<ShoppingListApp> {  final List<Map<String, dynamic>> \_items = [];  final TextEditingController \_textController = TextEditingController();  void \_addItem() {  showDialog(  context: context,  builder: (context) {  return AlertDialog(  title: const Text('Add Item'),  content: TextField(  controller: \_textController,  decoration: const InputDecoration(hintText: 'Enter item name'),  autofocus: true,  ),  actions: [  TextButton(  onPressed: () {  \_textController.clear();  Navigator.of(context).pop();  },  child: const Text('Cancel'),  ),  ElevatedButton(  onPressed: () {  if (\_textController.text.trim().isNotEmpty) {  setState(() {  \_items.add({  'name': \_textController.text.trim(),  'isPurchased': false,  });  });  \_textController.clear();  Navigator.of(context).pop();  }  },  child: const Text('Add'),  ),  ],  );  },  );  }  void \_togglePurchased(int index) {  setState(() {  \_items[index]['isPurchased'] = !\_items[index]['isPurchased'];  });  }    void \_deleteItem(int index) {  setState(() {  \_items.removeAt(index);  });  }    @override  void dispose() {  \_textController.dispose();  super.dispose();  }  @override  Widget build(BuildContext context) {  return Scaffold(  backgroundColor: Colors.blue[100],  appBar: AppBar(  title: const Text('Shopping List'),  backgroundColor: Colors.grey[800],  ),  body: \_items.isEmpty  ? const Center(child: Text('No items in the list'))  : ListView.builder(  itemCount: \_items.length,  itemBuilder: (context, index) {  final item = \_items[index];  return Card(  child: ListTile(  leading: Checkbox(  value: item['isPurchased'],  onChanged: (value) => \_togglePurchased(index),  ),  title: Text(  item['name'],  style: TextStyle(  decoration: item['isPurchased']  ? TextDecoration.lineThrough  : TextDecoration.none,  ),  ),  trailing: IconButton(  icon: const Icon(Icons.delete, color: Colors.red),  onPressed: () => \_deleteItem(index),),),);}, ),  floatingActionButton: FloatingActionButton(  backgroundColor: Colors.lightBlueAccent[800],  onPressed: \_addItem,  child: const Icon(Icons.add),),);}} |

**OUTPUT:**



**Lab Task 2: Answer the following questions.**

Q1: What is the purpose of the Provider package in Flutter?

Ans:Providers allow you to not only expose a value, but also create, listen, and dispose of it.To expose a newly created object, use the default constructor of a provider. Do not use the .value constructor if you want to create an object, or you may otherwise have undesired side effects.

Q2: What is the difference between ChangeNotifierProvider and Provider?

Ans:ChangeNotifierProvider (flutter\_riverpod/hooks\_riverpod only) is a provider that is used to listen to and expose a ChangeNotifier from Flutter itself. While In Flutter, “Provider” refers to a design pattern and a package that helps manage the state of your application. It's commonly used to efficiently share and update data between different parts of your app, such as widgets, without the need for prop drilling (passing data through multiple widget layers).

Q3: How do you update and listen to changes in state using Provider?

Ans:Use a class with ChangeNotifier, call notifyListeners() after updates, and listen using Consumer or context.watch().

Q4: Explain the role of Consumer in the Provider package.

Ans:Consumer rebuilds only the widget that depends on the provided data when it changes, optimizing performance.