

Duhok Polytechnic University

Technical College of Zakho

Department of CIS



LAB 2:

Actions & UI Recap

MOBILE APPLICATION II

Objectives

Media Integration

Event Handling

Modal Overlays

Constraint Management

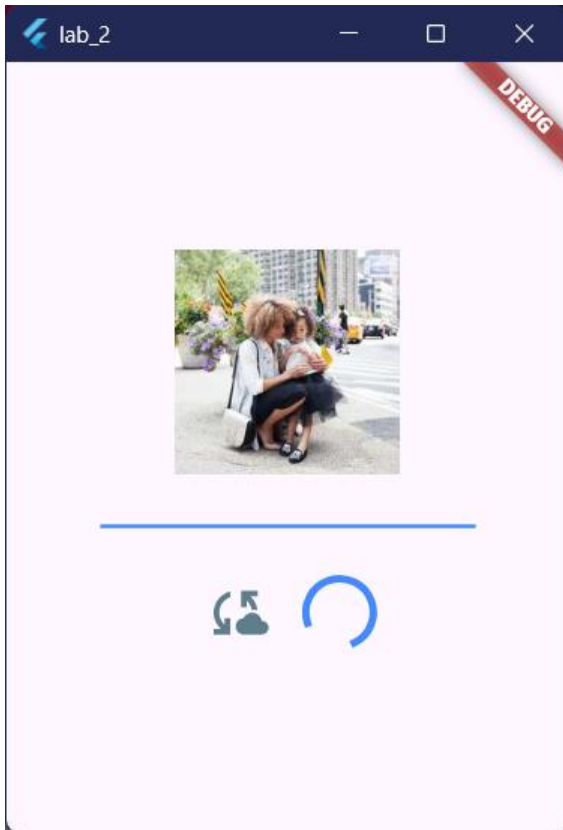
Media Integration & Visual Feedback

A professional application requires visual polish to feel complete. This section reviews how to integrate imagery, utilize vector iconography, draw structural separators, and provide visual feedback so the user knows the app is actively processing data.

Key Widgets & Attributes:

- Images (Image.network vs Image.asset): Use .network for dynamic web images and .asset for bundled local files. Remember that Image.asset requires strict YAML indentation in the pubspec.yaml file to register the folder path. Control visual scaling using fit: BoxFit.cover (fill and crop) or BoxFit.contain (shrink to fit without cropping).
- Iconography (Icon): Uses the Icons class (e.g., Icons.shopping_cart) to render lightweight, scalable vector graphics. Because they are rendered like fonts, they scale perfectly without pixelating and are customized using size and color.
- Visual Indicators (CircularProgressIndicator & LinearProgressIndicator): Crucial for preventing users from thinking the app has frozen. If the value attribute is left as null, the indicator animates infinitely (Indeterminate). If value is set between 0.0 and 1.0, it acts as a precise mathematical completion bar (Determinate).
- Separators (Divider & VerticalDivider): Draws physical lines to logically group UI elements. Customized using thickness, color, indent (start spacing), and endIndent (end spacing)

Media Example



```
1 Column(  
2   mainAxisAlignment: MainAxisAlignment.center,  
3   children: [  
4     Image.network(  
5       'https://picsum.photos/200',  
6       width: 120,  
7       height: 120,  
8       fit: BoxFit.cover,  
9     ),  
10    SizedBox(height: 20),  
11    Divider(  
12      thickness: 2,  
13      indent: 50,  
14      endIndent: 50,  
15      color: Colors.blueAccent,  
16    ),  
17    SizedBox(height: 20),  
18    Row(  
19      mainAxisAlignment: MainAxisAlignment.center,  
20      children: [  
21        Icon(Icons.cloud_sync, size: 35, color: Colors.blueGrey),  
22        SizedBox(width: 20),  
23        CircularProgressIndicator(  
24          value: null, // Spins infinitely  
25          color: Colors.blueAccent,  
26          strokeWidth: 4,  
27        ),  
28      ],  
29    ),  
30  ],  
31 )
```

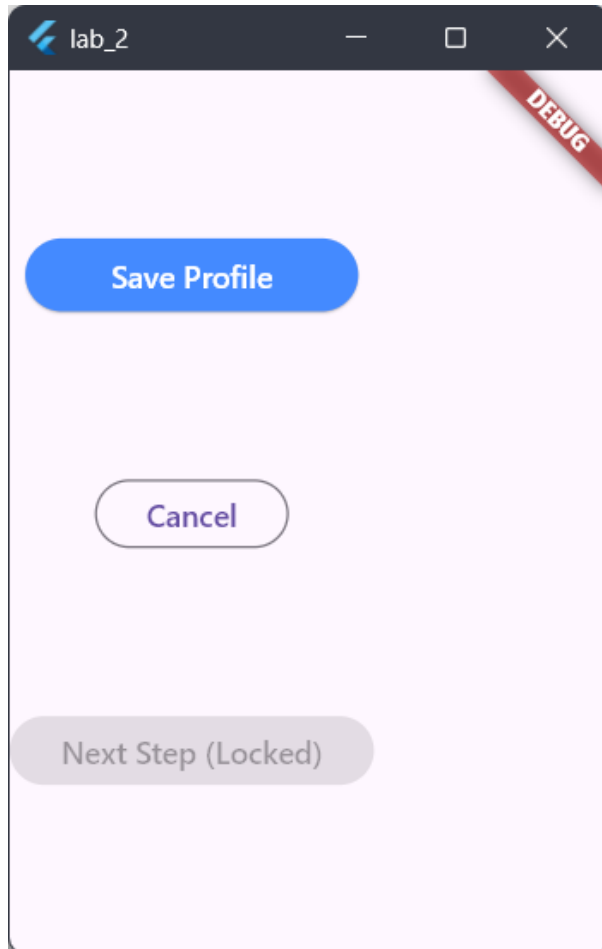
Event Handling & Material Controls

Buttons are what transform a static layout into an interactive application. In Flutter, buttons handle touch events, trigger callback functions, and automatically provide visual feedback to the user, such as the Material "ripple" ink animation when tapped.

Key Widgets & Attributes:

- The Callback (onPressed): The most critical attribute. It tells the app what code to execute the exact moment the button is tapped. If you set onPressed: null, the button automatically grays out and becomes disabled.
- ElevatedButton: Solid background and shadow for primary actions (e.g., "Submit").
- OutlinedButton: Transparent background with a visible border for secondary actions (e.g., "Cancel").
- TextButton: No background or border until tapped, used for tertiary links.
- Icon Buttons: used when space is tight (taking an icon instead of a child), while the FloatingActionButton (FAB) is a specialized, highly important circular button that hovers over the UI and has its own dedicated slot in the Scaffold.
- Button Styling: The style property and the .styleFrom() method, allow you to easily change the backgroundColor (fill color), foregroundColor (text/icon color), and shape

Event Handling Example



```
1 Column(  
2   mainAxisAlignment: MainAxisAlignment.spaceEvenly,  
3   children: [  
4     ElevatedButton(  
5       style: ElevatedButton.styleFrom(  
6         backgroundColor: Colors.blueAccent,  
7         foregroundColor: Colors.white,  
8         padding: const EdgeInsets.symmetric(  
9           horizontal: 40,  
10          vertical: 15,  
11        ),  
12      ),  
13      onPressed: () {  
14        print("Data Saved to Database!");  
15      },  
16      child: const Text('Save Profile'),  
17    ),  
18    OutlinedButton(  
19      onPressed: () {  
20        print("Action Cancelled");  
21      },  
22      child: const Text('Cancel'),  
23    ),  
24    const ElevatedButton(  
25      onPressed: null,  
26      child: Text('Next Step (Locked)'),  
27    ),  
28  ],  
29 ),  
30 )
```

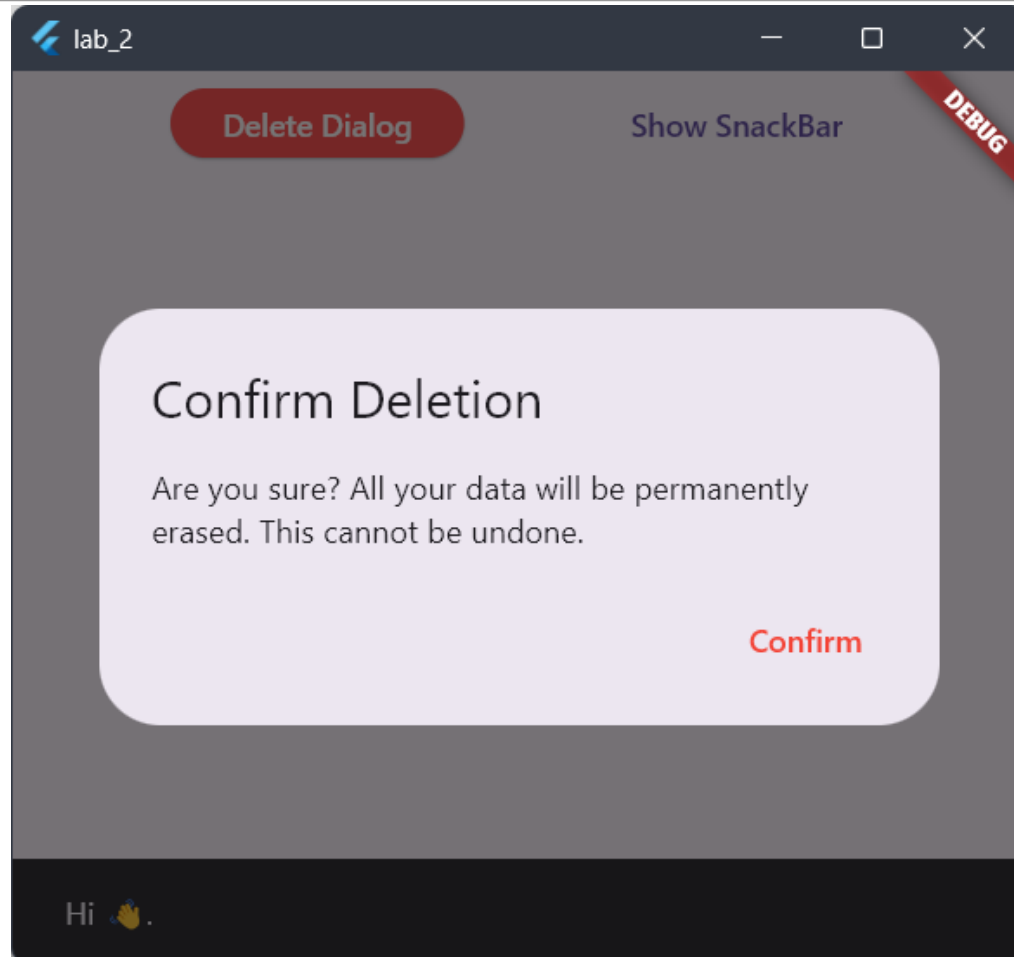
Modal Overlays & User Prompts

When a critical action is taken (like purchasing an item, deleting data, or logging out), the app must interrupt the normal user flow to confirm the choice. Flutter handles this by pushing a temporary, floating widget over the current screen, dimming the background to force the user's attention.

Key Attributes, Widgets, & Functions:

- `showDialog`: This is not a widget; it is a built-in Flutter function that triggers the popup. It requires the current `BuildContext` (so it knows where you are in the app) and a builder function to construct the popup UI.
- `AlertDialog`: The standard Material Design popup widget returned inside the `showDialog` builder. It is structured into three main slots:
 - `title`: The bold headline at the top (usually a `Text` widget).
 - `content`: The main body text explaining the situation.
 - `actions`: A list of widgets (almost always `TextButtons`) placed at the bottom right for choices like "Cancel" or "Confirm".

Modal Preview



Modal Example



```
1  Builder(  
2    builder: (context) {  
3    return Scaffold(  
4      body: Padding(  
5        padding: const EdgeInsets.all(8.0),  
6        child: Row(  
7          mainAxisAlignment: MainAxisAlignment.spaceEvenly,  
8          children: [  
9            ElevatedButton(  
10              style: ElevatedButton.styleFrom(  
11                backgroundColor: Colors.redAccent,  
12              ),  
13              child: Text(  
14                "Delete Dialog",  
15                style: TextStyle(color: Colors.white),  
16              ),  
17              onPressed: () {  
18                showDialog(  
19                  context: context,  
20                  builder: (BuildContext context) {  
21                    return AlertDialog(  
22                      title: Text("Confirm Deletion"),  
23                      content: Text(  
24                        "Are you sure? All your data will be permanently erased. This cannot be undone.",  
25                      ),
```

Modal Example (cont.)

```
26         actions: [
27             TextButton(
28                 onPressed: () {
29                     print("Deletion confirmed.");
30                 },
31                 child: Text(
32                     "Confirm",
33                     style: TextStyle(color: Colors.red),
34                 ),
35             ),
36         ],
37     );
38 },
39 );
40 },
41 ),
42 TextButton(
43     onPressed: () {
44         ScaffoldMessenger.of(
45             context,
46         ).showSnackBar(SnackBar(content: Text("Hi 🙋🏻.")));
47     },
48     child: Text("Show SnackBar"),
49 ),
50 ],
51 ),
52 ),
53 );
54 },
55 )
```

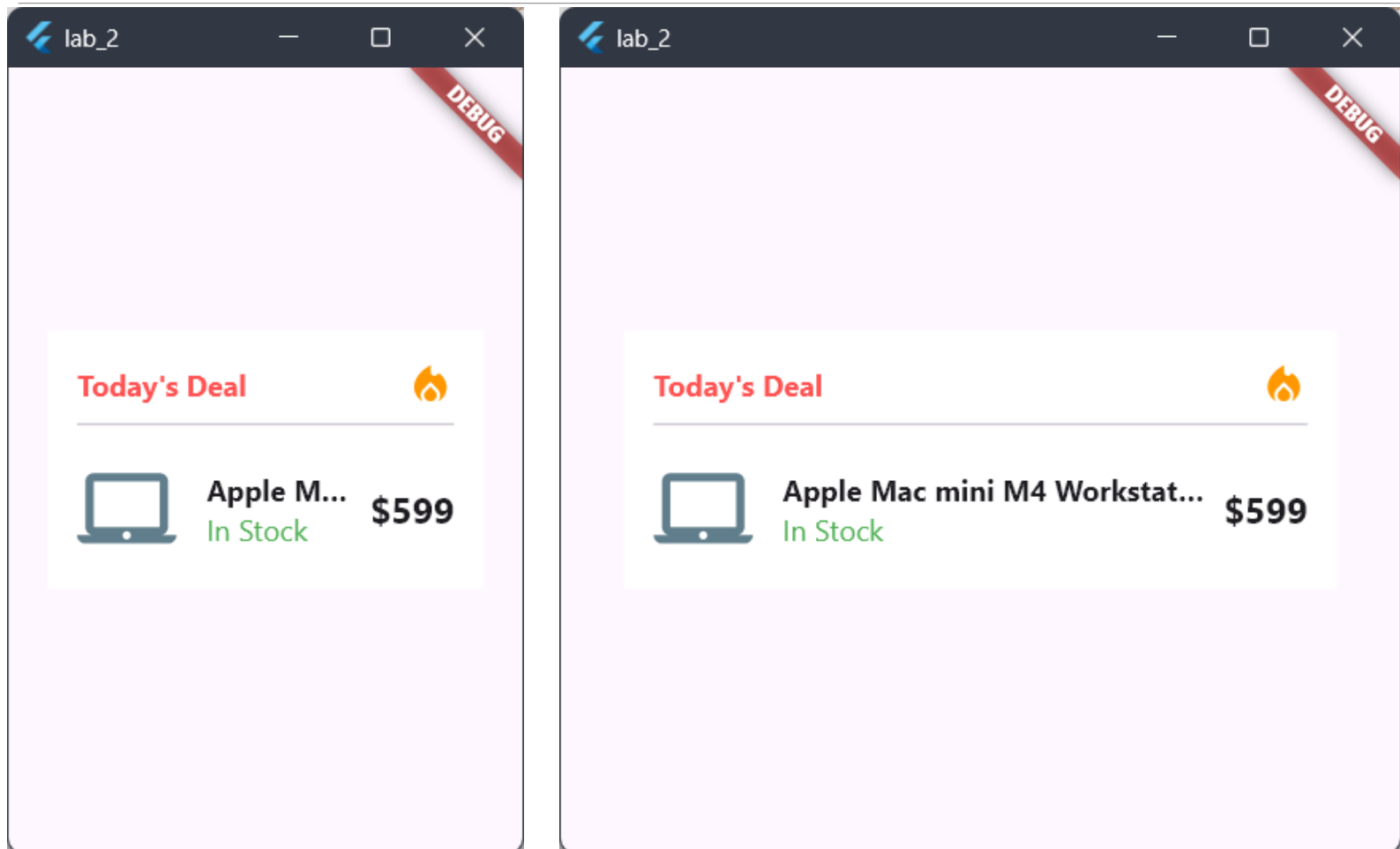
Responsive UI & Constraint Management

Mobile devices come in thousands of different screen sizes. If you hardcode a container to be 400 pixels wide, it might look perfect on an iPhone 17 Pro Max but will instantly crash with an Overflow Error on an older, narrower device. Responsive design ensures your UI adapts dynamically to any screen using flexible constraints.

Key Widgets & Concepts:

- Expanded Widget: The ultimate fix for text/child overflow. Wrapping a child widget in Expanded tells Flutter to dynamically stretch or shrink that specific widget to fill the remaining available space on the axis.
- Spacer Widget: An invisible, flexible widget used inside a Row or Column to take up all empty space between other widgets. It acts like an inflating balloon, pushing the widgets on either side of it to the far edges of the screen.
- MediaQuery: Allows your code to "ask" the actual device for its exact dimensions at runtime.
 - Syntax: `double screenWidth = MediaQuery.of(context).size.width;`

Responsive UI Preview



Responsive UI Code

```
1 // right under the build method
2 double screenWidth = MediaQuery.of(context).size.width;
3
4 // inside the material app and scaffold widgets
5 Center(
6   child: Container(
7     width: screenWidth * 0.85,
8     padding: EdgeInsets.all(15),
9     color: Colors.white,
10    child: Column(
11      mainAxisAlignment: MainAxisAlignment.min,
12      children: [
13        Row(
14          children: [
15            Text(
16              "Today's Deal",
17              style: TextStyle(
18                color: Colors.redAccent,
19                fontWeight: FontWeight.bold,
20              ),
21            ),
22            Spacer(),
23            Icon(Icons.local_fire_department, color: Colors.orange),
24          ],
25        ),
26        Divider(),
27        SizedBox(height: 10),
28        Row(
29          children: [
30            Icon(
31              Icons.laptop_mac,
32              size: 50,
33              color: Colors.blueGrey,
34            ),
```

```
35      SizedBox(width: 15),
36      Expanded(
37        child: Column(
38          crossAxisAlignment: CrossAxisAlignment.start,
39          children: [
40            Text(
41              "Apple Mac mini M4 Workstation - 16GB",
42              style: TextStyle(fontWeight: FontWeight.bold),
43              overflow: TextOverflow
44                .ellipsis,
45            ),
46            Text(
47              "In Stock",
48              style: TextStyle(color: Colors.green),
49            ),
50          ],
51        ),
52      ),
53      SizedBox(width: 10),
54      Text(
55        "\\$599",
56        style: TextStyle(
57          fontSize: 18,
58          fontWeight: FontWeight.bold,
59        ),
60      ),
61    ],
62  ),
63 ],
64 ),
65 ),
66 )
```

Any Questions?



Lab Work

