Part 1: Setting up your flutter enviornment

a. Install Flutter SDK

- Visit: https://flutter.dev/docs/get-started/install
- Download the SDK for your operating system.
- Extract it to a location like C:\src\flutter.
- Add C:\src\flutter\bin to your system PATH.

b. Check Your Installation

- Open terminal or command prompt and run:

flutter doctor

- This tool checks all required components, like Android Studio, device emulators, and Dart SDK.

c. Install Android Studio

- Download: https://developer.android.com/studio
- During installation, make sure to install:
  - Android SDK
  - Android Virtual Device (AVD) Manager
- In Android Studio, go to Preferences > Plugins, search for and install:
  - Flutter plugin
  - Dart plugin

d. Install Visual Studio Code (Optional but Recommended)

- Download from: https://code.visualstudio.com/
- Add Flutter and Dart extensions from the Extensions Marketplace.

e. Create and Run Your First Project

- Open terminal or VS Code terminal:
- Click create a new flutter project
- Under the “lib” , open it and it should have some prebuilt code

Code

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

void main() {
  runApp(const themes());
}

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      title: 'Flutter Demo',
      theme: ThemeData(
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.red),
      ),
      home: landingPage()
    );
  }
}
```

1. import 'package:flutter/material.dart';
This imports the core Flutter Material library, giving access to UI components.

2. import 'landingPage.dart';
This imports the landingPage widget so it can be used as the home screen.

3. void main() { runApp(const themes()); }

- The app’s entry point.
- runApp() initializes the app with the themes widget.
- const makes the widget immutable and improves performance.

4. class themes extends StatelessWidget

- Defines a widget that does not hold state.
- Responsible for returning the main MaterialApp.

5. @override Widget build(BuildContext context)

- The build method describes how to display the widget in the UI.

6. MaterialApp(...)

- The root widget that sets the app's theme and main configurations.
- debugShowCheckedModeBanner: false hides the debug label.
- title defines the app title.
- theme provides consistent styling and colors.
- home defines the first page that appears (your landingPage).

7. ColorScheme.fromSeed(seedColor: Colors.red)

- Automatically generates a full-color palette from a single red seed color.

3. Main Page Purpose Recap

- It serves as the app entry point.
- Defines theming, app title, and removes the debug banner.
- Sets the first visible page of the app (landingPage).

Part 2: Creating your new landing page!

1. Importing Dependencies

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

- flutter/material.dart → Provides UI components.
- secondPage.dart → The target page to navigate to.

2. Defining the Stateful Widget (LandingPage)

```
class LandingPage extends StatefulWidget {
  const LandingPage({super.key});
```

```
  @override
  State<LandingPage> createState() => _LandingPageState();
}
```

- LandingPage is a stateful widget because we may update UI dynamically.
- _LandingPageState holds the logic and UI structure.

3. Building the UI (build method)

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    body: Stack(
      children: [
```

- Scaffold provides the base layout for the screen.
- Stack allows layering widgets on top of each other (background image + button).

4. Adding the Background Image

```
Image.asset(
  'assets/landingpage.jpg',
  fit: BoxFit.cover,
  width: double.infinity,
  height: double.infinity,
),
```

- Loads an image from the assets/ folder.
- BoxFit.cover ensures the image covers the full screen.
- double.infinity makes it expand to fill the screen.

5. Centering the Button

```
Center(
  child: Container(
    width: 200.0,
    height: 50.0,
    decoration: BoxDecoration(
      color: Colors.grey,
      borderRadius: BorderRadius.circular(50),
    ),
```

- Center aligns the button to the middle of the screen.
- Container gives the button size and rounded corners.

6. Adding a Button

```
child: TextButton(
  onPressed: () {
    Navigator.of(context).push(_secondRoute());
  },
  child: const Text(
    "Welcome",
    style: TextStyle(color: Colors.white, fontSize: 18),
  ),
),
```

- TextButton creates a tappable button.
- onPressed triggers navigation using Navigator.push().
- Text styling makes it white and readable.

7. Navigation with Slide Animation

```
Route _secondRoute() {
```

```
  return PageRouteBuilder(
    pageBuilder: (context, animation, secondaryAnimation) =>
        const SecondPage(title: "Second"),
    transitionsBuilder: (context, animation, secondaryAnimation, child) {
      const begin = Offset(0.0, 1.0);
      const end = Offset.zero;
      const curve = Curves.ease;

      var tween = Tween(begin: begin, end: end).chain(CurveTween(curve: curve));
      return SlideTransition(position: animation.drive(tween), child: child);
    },
  );
}
```

- PageRouteBuilder creates a custom animation.
- Slide Transition moves the page from bottom to top (Offset(0.0, 1.0)).
- Curves.ease makes the animation smooth.

Landing Page Purpose Recap

- It serves as the first screen of the app.
- Displays a background image and a Welcome button.
-Navigates to the Second Page when the button is clicked.
- Uses a smooth slide transition for navigation.

Part 3: Creating your main page

1. Importing Dependencies

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

```
import 'landingPage.dart';
```

- flutter/material.dart → Provides UI components.
- landingPage.dart → Allows navigation back to the first screen.

2. Defining the Stateful Widget (secondPage)

```
class secondPage extends StatefulWidget {
  const secondPage({super.key, required this.title});

  final String title;

  @override
  State<secondPage> createState() => _secondPage();
}
```

- secondPage is a stateful widget because it dynamically updates the task list.
- title is a required parameter (set to "Second" in the first page).
- _secondPage handles the UI and logic.

3. the Task List

```
final List<Widget> _tasks = [];
```

- _tasks stores a list of task cards.
- Tasks are displayed using a ListView.

Adding a New Task (_createCard())

```
void _createCard() {
  setState(() {
    _tasks.add(
      Card(
        color: Colors.grey,
```

```
        elevation: 10,
        margin: EdgeInsets.all(10),
        child: Padding(
          padding: const EdgeInsets.all(8.0),
          child: TextField(
            style: TextStyle(color: Colors.white),
            decoration: InputDecoration(
              hintText: "Enter Task",
              hintStyle: TextStyle(color: Colors.white),
            ),
          ),
        ),
      ),
    );
  });
}
```

- setState() updates _tasks to include a new card.
- Card has a Text Field where users can type tasks.
- TextField text is styled in white for readability.

Deleting a Task (_deleteCard(index))

```
void _deleteCard(int index) {
  setState(() {
    _tasks.removeAt(index);
  });
}
```

- Removes a task at the given index.
- setState() updates the UI immediately.

4. **Building the UI (build method)**

AppBar with Add Button

```
appBar: AppBar(
  actions: [
    IconButton(
      onPressed: () {
        _createCard();
      },
      icon: const Icon(Icons.add),
      color: Colors.white,
    ),
  ],
  backgroundColor: Colors.black87,
  title: Text("Todo List", style: TextStyle(color: Colors.white)),
),
```

- AppBar has a title "Todo List".
- Add button (+) calls _createCard().
- Background color is set to dark mode (black87).

```
body: Container(
  child: _tasks.isEmpty
      ? Center(child: Text("No current tasks!"))
      : ListView.builder(
          itemCount: _tasks.length,
          itemBuilder: (context, index) {
            return Dismissible(
              key: Key('$index'),
```

```
            onDismissed: (direction) {
              _deleteCard(index);
            },
            direction: DismissDirection.startToEnd,
            child: _tasks[index],
          );
        },
      ),
),
```

- If _tasks is empty, show "No current tasks!".
- If tasks exist, display them using ListView.builder().
- Swipe left to delete tasks using Dismissible.

5. Navigation Back to First Page

```
Route _firstRoute() {
  return PageRouteBuilder(
    pageBuilder: (context, animation, secondaryAnimation) =>
       const landingPage(),
    transitionsBuilder: (context, animation, secondaryAnimation, child) {
      const begin = Offset(0.0, 1.0);
      const end = Offset.zero;
      const curve = Curves.ease;

      var tween = Tween(begin: begin, end: end).chain(CurveTween(curve: curve));

      return SlideTransition(position: animation.drive(tween), child: child);
    },
```

);

}

- Defines a slide transition back to LandingPage.
- Moves the page from bottom to top.
-

Second Page Purpose Recap

- It serves as a simple To-Do List.
- Allows users to add tasks dynamically using a button.
-Users can delete tasks by swiping them away.
- Uses cards to display each task with a text field.
-Implements a slide transition for navigation back to the Landing Page.

Using SingleChildScrollView vs. ListView.builder

I first tried using SingleChildScrollView to display a list of cards. However, I ran into problems when trying to delete cards from the list. Since SingleChildScrollView loads all the cards at once, removing a card required rebuilding the entire list, which was inefficient and caused performance issues.

Why I Switched to ListView.builder

ListView.builder only builds the visible items on the screen, making it much more efficient. It also allowed me to delete cards smoothly without reloading the entire list, improving performance. With ListView.builder, I could manage a large number of items with faster scrolling.