Tips & Tricks for Creating a Flutter App

Overview

This guide is designed to help students create the first phase of their Flutter app, which will be **1-2 pages for now**, with the option to expand into a fully functional app later. Students can choose to either:

- Develop a simple **standalone** 1-2 page app.
- Start with **1-2 pages as the foundation** for a larger project built over time.

Regardless of your choice, these tips will help you effectively brainstorm, plan, and develop your Flutter app.

---

1. Brainstorming Your App Idea

Before you start coding, take some time to plan your app:

- **Keep it Simple**: Since this is a 1-2 page app, pick a small but creative idea.
- **Think About Your Interests**: What would you enjoy building? Some ideas include:
    - A to-do list app
    - A digital business card
    - A simple weather app
    - A mood tracker
    - A quiz game
- **Sketch Your App**: Draw a rough wireframe of how your app should look. Think about:
    - What screens will it have?
    - How will users interact with it?
    - What UI elements will be needed?

---

2. Planning Before Coding

- **Write Down the Features**: List the main features of your app to stay focused.

- **Decide on Navigation**:
    - If your app has multiple pages, use **Navigator.push()** to move between screens.
    - For bottom navigation, use **BottomNavigationBar**.
- **Think About State Management**:
    - If your app has dynamic content (e.g., a counter or form), use **StatefulWidget**.
    - If everything stays the same, use **StatelessWidget**.

---

3. Looking at Common App Setups & Learning from Existing Apps

Before designing your app, consider how other similar apps are structured. Even though you are only creating a **1-2 page app for now**, looking at **real-world applications** can help you understand what works best.

Common Setups for Different Apps:

App Type	Common Layouts Used
Social Media	BottomNavigationBar + Feed (ListView)
E-Commerce	GridView for products + Search Bar
Portfolio	Scrollable Single Page + Contact Button
Notes App	ListView + Floating Action Button (FAB)

Why Should You Look at Existing Apps?

- **Get inspiration**: Look at well-designed apps and identify **what works well**.
- **Learn UI/UX best practices**: Compare different apps and see how they structure their screens.
- **Improve your own design**: If an app has a good user flow, think about implementing similar patterns in your own app.

Example: Creating a To-Do App

- Search for popular to-do list apps.
- Notice how tasks are structured (list vs. grid).

- Identify key features (checklists, reminders, categories).
- Pick elements you like and adapt them to your Flutter design.

---

4. Understanding Widget Structure & Indentation

One of the most confusing parts of Flutter for beginners is the **nested widget structure**. Every widget is wrapped inside another widget, which creates a hierarchy.

Why Are Widgets Nested Like This?

Flutter uses a **tree-like structure** because every UI element is a widget, and widgets can contain other widgets to build complex layouts. This allows flexibility and customization while keeping each widget's role clear and separate.

For example:

- **The `Scaffold` widget** provides a structured page layout.
- **Inside the `Scaffold`, we use a `Container`** because it helps with styling, like padding or background color.
- **The `Column` widget** is used inside the `Container` to stack items vertically.
- **A `Row` widget** inside the `Column` helps align elements side by side.

Understanding `child` vs. `children`

- **Widgets like `Container`, `Padding`, and `Expanded` can only have a single child.** This means that when you place a widget inside them, it must be wrapped as `child: SomeWidget()`.
- **Widgets like `Column`, `Row`, and `Stack` can have multiple children.** These widgets require a list of widgets, so you must use `children: [Widget1(), Widget2()]`.
- **Example:** If you want multiple widgets inside a `Container`, you must first add a `Column` or `Row` because `Container` can only take a single widget as its `child`.

Example Structure:

```
Scaffold(
  body: Container(
    padding: EdgeInsets.all(16.0),
    child: Column(
```

```
      children: [
        Row(
          children: [
            Image.asset('assets/example.png', width: 100),
            SizedBox(width: 10), // Adds space between widgets
            Text("Hello, Flutter!"),
          ],
        ),
      ],
    ),
  ),
);
```

Understanding the Indentation:

- **`Scaffold`**: The root layout structure of a page. It holds everything on the screen.
- **`Container`**: A flexible box that contains padding and background properties. It's used to group widgets together and apply styling.
- **`Column`**: Arranges widgets vertically. We use this when we need multiple elements stacked on top of each other.
- **`Row`**: Arranges widgets horizontally inside the column. This allows side-by-side elements.
- **`Image.asset`**: Displays an image. Inside the `Row`, it appears next to the text.
- **`SizedBox`**: Adds space between widgets. Here, it prevents the text from touching the image.
- **`Text`**: Displays text content.

Tips for Managing Indentation:

- Use **proper indentation** to visually separate parent and child widgets.
- **Remember that `Container` can only have one child, but `Column` or `Row` can have multiple children.**
- If your code gets too nested, break it into **separate methods or widgets**.
- Use **comments** to clarify different widget sections.
- If confused, imagine your app's structure like a family tree—each widget is a parent or child to another widget.

4. UI Design Tips

- **Use Common Layout Widgets**:

    - **Column & Row**: Use **Column** to arrange elements vertically and **Row** to arrange them horizontally. These are essential for structuring your layout.
    - **Container**: Acts as a flexible box that can hold padding, borders, and background colors. It's one of the most commonly used widgets.
    - **Stack**: Allows widgets to overlap each other. Useful for placing text over images or creating layered UI elements.
- **Spacing & Padding**:
    - Use **SizedBox** to add empty space between widgets.
    - Use **Padding** to control the space around a widget, preventing the UI from looking cramped.
    - Maintain a consistent spacing style throughout your app to ensure a clean layout.
- **Text Styling**:
    - Use **TextStyle** to change fonts, colors, and sizes.
    - Ensure readability by maintaining good contrast between text and background.
    - Consider using different font weights (bold, light) to create a visual hierarchy.

---

5. Navigation: Moving Between Screens

Choosing the Right Navigation Type

Navigation Type	When to Use
`Navigator.push()`	When moving between different full-screen pages in your app. Most common for simple navigation.
`BottomNavigationBar`	When your app has 2-5 main sections that users need to switch between easily. Great for apps with a home, profile, and settings section.
`Drawer`	When you have many sections or settings that you don’t want to take up screen space. A slide-out menu is ideal for apps with multiple categories.
`TabBar`	When switching between closely related content, such as different product categories in a shopping app.

---

6. Debugging & Common Fixes

- **Use `flutter doctor`**: Ensures that Flutter and dependencies are correctly set up.
- **Hot Reload vs. Hot Restart**:
    - *Hot Reload* → Only updates changed code while keeping the app running. Faster for UI tweaks.
    - *Hot Restart* → Restarts the entire app, clearing states. Use this when debugging app behavior.
- **Check the Debug Console**: When errors occur, the debug console provides details on what went wrong.
- **Wrap Widgets in `SafeArea`**: Ensures that content does not get hidden behind notches or system UI elements.

7. Conclusion

- When planning your app, imagine what you want it to look like. This will help you when you start writing the code. Imagine where a row is going or how you want the app bar or body to look.
    - If you can't think of any ideas, look at other apps for motivation
- If you're having trouble with navigation, I would look at chapter 7 in the book, which goes up to page 99. It details how that works and how to create simple navigation routes. Everything past page 99 will provide different ways to do navigation routes like tabs or drawers.
- If you're wondering how you can style your widgets and what can go in them, I will look at chapters 5 and 4, as they go a lot more into value widgets and their functions, as well as gesture widgets and their functions.
    - Specifically, if you're looking for buttons, gestures, or dismissibles, I recommend Chapter 5.
    - I recommend Chapter 4 if you're interested in creating forms, text fields, input, and sizing images.

- If you’re looking at how to structure your app regarding the page, I would look at Chapter 6. This is also where you can find different ways to position the strings or images that appear in your widgets or deal with extra spaces.