# In-Class Activity 02: Creating Themes in Flutter (Part 1)

**Course:** *Mobile App Development* | **Due Date:** *02/09/206 @7:30 pm*

Today's task involves creating dynamic themes in Flutter. You will learn how to implement a Light and Dark mode system, customize UI components globally, and manage application state to switch between themes.

> ### 🎯 Learning Objectives
>
> By completing this activity, you will be able to:
>
> - Define custom `ThemeData` for both light and dark modes in a Flutter application.
> - Utilize `ThemeMode` to manage the state of the application's appearance.
> - Apply theme properties globally to widgets like Text, Containers, and Buttons.
> - Implement user interaction to toggle themes dynamically using `setState`.
> - Enhance User Experience (UX) with smooth transitions using `AnimatedContainer`.

## 1. Introduction to Theming

In Flutter, **theming** refers to the process of defining and customizing the visual appearance of your app's user interface elements—such as colors, typography, and shapes—in a consistent and cohesive way. Rather than styling every single widget individually, you define a "blueprint" that the entire app follows.

**Why it matters:** A unified theme ensures your app looks professional and polished. It also significantly reduces code duplication. When you want to change your app's primary color from Blue to Red, you only change it in one place (the Theme), and it updates everywhere.

### Key Components of a Flutter Theme

| Component | Description | Example Usage |
|---|---|---|
| **Colors** | Defines the palette (primary, background, accent colors). | `colorScheme: ColorScheme.light(...)` |

| Component | Description | Example Usage |
|---|---|---|
| **Typography** | Specifies font families, sizes, and weights for text (headings, body). | `textTheme: TextTheme(...)` |
| **Shapes** | Defines the corner radius and contours of cards, buttons, etc. | `shape: RoundedRectangleBorder(...)` |
| **Icons** | Customizes the default style of icons globally. | `iconTheme: IconThemeData(...)` |

## 2. Part 1: UI Design Requirements

Your objective is to build a single-screen application that toggles between two distinct themes.

### A. HomeScreen Requirements

The main screen must display a simple layout centered on the page.

- **Layout:** Use a `Column` inside a `Center` widget.
- **Text Widget:**
  - Content: "Mobile App Development Testing"
  - Font Size: 18
  - Style: Should inherit color from the current theme.
- **Container Widget:**
  - **Shape:** Circular (High border radius).
  - **Background Color:**
    - Light Mode: Grey
    - Dark Mode: White
  - **Dimensions:** Use reasonable width/height (e.g., 200x200 or 300 width).
  - **Margins/Padding:** Add spacing so elements aren't cramped.

> 💡 **Design Tip:** To make a container circular, use `BoxDecoration` with `borderRadius` or `shape: BoxShape.circle`. Remember that color must be defined inside the decoration if decoration is used.

### B. Settings/Control Requirements

- Provide a mechanism to toggle the theme.
- This can be a `Switch` widget or two separate buttons (as suggested in the template).

- **Interaction:** Tapping the control must rebuild the app with the new theme selected.

## 3. Step-by-Step Implementation Guide

### Step 1: Define Your Themes

In your `MaterialApp`, you will define two distinct theme properties: `theme` (for light mode) and `darkTheme` (for dark mode).

```
MaterialApp(
  theme: ThemeData(
    primarySwatch: Colors.blueGrey,
    scaffoldBackgroundColor: Colors.white, // Light background
  ),
  darkTheme: ThemeData.dark(), // Standard dark theme
  // ...
);
```

### Step 2: Accessing Theme Data

To use these colors in your widgets, do not hardcode colors (e.g., `Colors.red`). Instead, ask the context for the current theme's color.

```
Container(
  color: Theme.of(context).primaryColor, // Adapts to theme
  child: Text(
    "Hello",
    style: Theme.of(context).textTheme.bodyLarge,
  ),
)
```

### Step 3: Managing State

You need a variable to track the current mode, and a function to update it.

- **Variable:** `ThemeMode _themeMode = ThemeMode.system;`
- **Function:** Calls `setState` to update the variable.

## 4. Starter Code Template

Copy this code into your `lib/main.dart` file. Areas marked with comments require your implementation.

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

void main() {
  runApp(const RunMyApp());
```

Copy code

```
}

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

  @override
  State<RunMyApp> createState() => _RunMyAppState();
}

class _RunMyAppState extends State<RunMyApp> {
  // Variable to manage the current theme mode
  ThemeMode _themeMode = ThemeMode.system;

  // Method to toggle the theme
  void changeTheme(ThemeMode themeMode) {
    setState(() {
      _themeMode = themeMode;
    });
  }

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      title: 'Theme Demo',

      // TODO: Customize these themes further if desired
      theme: ThemeData(
        primarySwatch: Colors.blueGrey,
        scaffoldBackgroundColor: Colors.grey[200], // Light mode background
      ),
      darkTheme: ThemeData.dark(), // Dark mode configuration

      themeMode: _themeMode, // Connects the state to the app

      home: Scaffold(
        appBar: AppBar(
          title: const Text('Theme Demo'),
        ),
        body: Center(
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              // PART 1 TASK: Container and Text
              Container(
                width: 300,
                height: 200,
                margin: const EdgeInsets.all(20),
                decoration: BoxDecoration(
                  // Use a ternary operator to check theme brightness
                  color: Theme.of(context).brightness == Brightness.dark
                      ? Colors.white
                      : Colors.grey,
                  borderRadius: BorderRadius.circular(20),
```

```
                ),
                alignment: Alignment.center,
                child: const Text(
                  'Mobile App Development Testing',
                  style: TextStyle(fontSize: 18, color: Colors.black),
                  textAlign: TextAlign.center,
                ),
              ),

              const SizedBox(height: 20),

              const Text('Choose the Theme:', style: TextStyle(fontSize: 16)),

              const SizedBox(height: 10),

              // PART 1 TASK: Controls
              Row(
                mainAxisAlignment: MainAxisAlignment.spaceEvenly,
                children: [
                  ElevatedButton(
                    onPressed: () => changeTheme(ThemeMode.light),
                    child: const Text('Light Theme'),
                  ),
                  ElevatedButton(
                    onPressed: () => changeTheme(ThemeMode.dark),
                    child: const Text('Dark Theme'),
                  ),
                ],
              ),
            ],
          ),
        ),
      ),
    );
  }
}
```

## 5. Part 2: Additional Features (Enhanced)

> **Requirement:** You must complete **2 out of the 4** tasks listed below.

Standard transitions can feel abrupt. Adding animations improves the "feel" of your application.

### Task 1: AnimatedContainer

Replace your standard `Container` with an `AnimatedContainer`. This widget automatically animates changes to its properties (like color or size) over a set duration.

### Task 2: Theme Switch Logic with Switch Widget

Replace the two buttons in the template with a single `Switch` widget. This requires checking the current state (is it dark?) and passing a boolean value.

### Task 3: Custom Animation Duration

Set the `duration` property of your AnimatedContainer to **500 milliseconds**. This creates a smooth, half-second transition that is noticeable but not too slow.

### Task 4: Dynamic Icons

Display a Sun icon ( `Icons.wb_sunny` ) when in light mode and a Moon icon ( `Icons.nightlight_round` ) when in dark mode.

**Example Code for Part 2:**

```
// Example implementation merging Tasks 1, 2, and 3
AnimatedContainer(
  duration: const Duration(milliseconds: 500), // Task 3
  color: _themeMode == ThemeMode.dark ? Colors.white : Colors.grey, // Task 1
  padding: const EdgeInsets.all(20),
  child: Column(
    children: [
      const Text('Mobile App Development Testing', style: TextStyle(fontSize: 18)),

      // Task 2 & 4: Switch with Dynamic Icon
      Row(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          Icon(_themeMode == ThemeMode.dark ? Icons.nightlight_round :
Icons.wb_sunny),
          Switch(
            value: _themeMode == ThemeMode.dark,
            onChanged: (isDark) {
              setState(() {
                _themeMode = isDark ? ThemeMode.dark : ThemeMode.light;
              });
            },
          ),
        ],
      )
    ],
  ),
);
```

## 6. Submission Instructions

### 📌 Submission Requirements

**Due Date:** 02/09/206 @2:30 pm

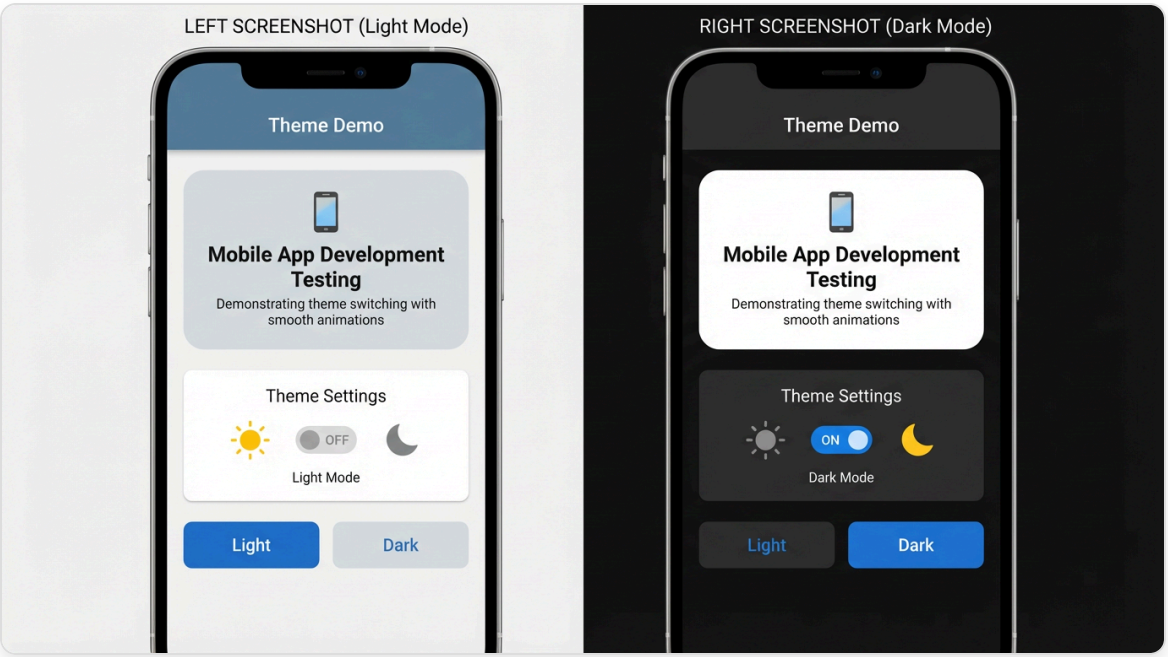
1. **Submit a GitHub Repository URL**
   Upload your project to GitHub and submit the direct link to your repository.
2. **Attach Your main.dart File**
   In addition to the GitHub link, you must also upload or attach your `main.dart` file separately.
3. **Upload Your Work to iCollege**
   Submit both required items (GitHub URL + `main.dart` file) in the In Class ACT 02 folder located in iCollege, making sure the submission is tied to the correct assignment and due date.

## Self-Check before Submitting:

1. Does the app compile without errors?
2. Does the container color change when theme is switched (Grey ↔ White)?
3. Did you implement at least 2 features from Part 2?
4. Is the code properly indented?

## 📱 Expected Visual Output:

Below is a visual representation of how your completed app should look in both Light and Dark modes:



🎨 **Visual Checklist:**

- **Left Side (Light Mode):** Light gray background, container with gray background, sun icon visible
- **Right Side (Dark Mode):** Dark background, container with WHITE background (important!), moon icon visible
- **Both Modes:** "Mobile App Development Testing" text clearly visible at 18pt
- **Switch/Controls:** Toggle or buttons to change between themes working correctly
- **Smooth Transition:** Colors should animate smoothly when switching (if Part 2 tasks completed)

## 7. References & Resources

- Flutter Themes Documentation - Official Guide
- Flutter Custom Theme with ThemeExtension - Advanced Reading

## 8. Complete Solution Code (Reference)

📘 **For Learning Reference:** This complete solution should demonstrate all features including ALL 4 optional tasks from Part 2.

### ✨ Features Implemented in This Solution:

- ✅ Custom Light and Dark themes with beautiful color schemes
- ✅ AnimatedContainer for smooth transitions (Task 1)
- ✅ Switch widget with proper state management (Task 2)
- ✅ Custom 500ms animation duration (Task 3)
- ✅ Dynamic icons based on theme mode (Task 4)
- ✅ Enhanced UI with cards, shadows, and professional styling
- ✅ Comprehensive comments explaining each section

🧮 **Grading Rubric (100 pts)**

| Criteria | Points |
|---|---|
| Custom Light and Dark themes with cohesive color schemes | 15 |
| AnimatedContainer for smooth transitions (Task 1) | 15 |
| Switch widget with proper state management (Task 2) | 15 |
| Custom 500ms animation duration (Task 3) | 10 |
| Dynamic icons based on theme mode (Task 4) | 10 |
| Enhanced UI with cards, shadows, and professional styling | 20 |
| Comprehensive comments explaining each section | 15 |
| **Total** | **100** |

📝 **Learning Exercise:**

After creating your solution, try these modifications to deepen your understanding:

1. Change the animation duration from 500ms to 1000ms - notice the difference?
2. Add a third theme (e.g., "System" mode that follows device settings)
3. Modify the container's border radius to make it more or less rounded
4. Add your own custom colors to the ColorScheme
5. Create an additional animated widget (try AnimatedOpacity or AnimatedSize)