Mobile App Development Coursework #01

Course: CSC 4360/6370 - Mobile App Development

Assignment: CW #01 - Counter & Image Toggle App

Due Date: 02/15/2026

Submission: APK File + GitHub Repository Link

Introduction

Welcome to your first major coursework assignment! The core objective of this task is to develop a functional Flutter application that demonstrates your understanding of basic state management, widget composition, and interactivity. You will be building an app that features a numeric counter and an interactive image toggle.

For graduate students (CSC 6370), this assignment includes an advanced component focusing on data persistence and complex state management.

CORE LEARNING OBJECTIVES

- **State Management:** Understand how to use `setState()` to update the UI dynamically.
- **Widget Composition:** Learn to combine `Column`, `Row`, `Text`, and `Button` widgets effectively.
- **Asset Management:** Practice importing and displaying local assets (images).
- **Theming:** Implement dynamic theme switching (Light/Dark mode).
- **Animation:** Use AnimationControllers for smooth visual transitions.

Task 1: The Counter Button

Objective

Create a simple interface that displays a number and increments it when a button is pressed. This is the "Hello World" of interactive apps, teaching you the fundamental cycle of Event → State Change → UI Rebuild.

Requirements

1. **Display:** Show an initial counter value (e.g., 0) prominently on the screen.
2. **Interaction:** Include a button labeled "Increment".
3. **Logic:** Increase the counter value by one every time the button is pressed.

Implementation Guide

Step 1: Project Setup

Initiate a new Flutter project using your terminal or IDE:

```
flutter create cw1_counter_app
```

Step 2: State Variable

Inside your `State` class (e.g., `_MyHomePageState`), declare an integer variable.

```
class _MyHomePageState extends State<MyHomePage> {
  int _counter = 0; // Initialize counter to 0

  void _incrementCounter() {
    setState(() {
      // This call to setState tells the Flutter framework that something has
      // changed in this State, which causes it to rerun the build method below.
      _counter++;
    });
  }
```

```
  // ... rest of the code
}
```

Step 3: UI Composition

Use a `Column` widget to center your content. Combine a `Text` widget for the display and an `ElevatedButton` for the interaction.

> **TIP: THE WIDGET TREE**
>
> Your widget tree structure should look roughly like this:
>
> `Scaffold -> Center -> Column -> [Text(Counter), ElevatedButton(Increment)]`

Task 1+ (Scale-Up Features)

Make Task 1 feel modern by implementing at least two of the enhancements below. These features showcase polish, usability, and thoughtful state control.

> **SCALE-UP FEATURE PACK**
>
> - **Multi-step controls:** Add +1, +5, +10 buttons (or a step selector) and display the current step.
> - **Decrement + Reset:** Add a decrement button and a reset button with disabled states at 0.
> - **Goal meter:** Let the user set a target and show a progress bar that celebrates at 100%.
> - **History + Undo:** Keep the last five actions and allow a one-tap undo.
> - **State persistence:** Save the counter value locally so it restores on app restart.

Task 2: Image Toggle & Animation

Objective

Enhance your application by adding visual interactivity. You will display an image that toggles between two states (e.g., a cat and a dog, or a sun and a moon) when pressed, accompanied by a smooth transition animation.

Requirements

1. **Asset Integration:** Display an initial image on the screen.
2. **Toggling:** Change the displayed image to a second image when a button is pressed.
3. **Theme Switching:** Add a button to toggle the entire app between Light and Dark modes.
4. **Animation:** The image transition must be animated (fade, scale, or rotation).

Detailed Implementation Guidelines

1. MANAGING ASSETS

First, create an `assets` folder in your project root and add two images. Don't forget to register them in your `pubspec.yaml` file:

```
flutter:
  assets:
    - assets/image1.png
    - assets/image2.png
```

2. ANIMATION LOGIC

To implement the animation requirements (AnimationController, CurvedAnimation, FadeTransition), you need to mix in `SingleTickerProviderStateMixin` to your State class.

> UNDERSTANDING ANIMATION COMPONENTS
>
> - **AnimationController:** The "engine" that drives the animation. It needs a duration (how long it takes).
> - **CurvedAnimation:** Defines the "feel" of the motion (e.g., bouncing, easing in/out).
> - **FadeTransition:** The widget that actually applies the visual effect based on the controller's value.

Code Snippet: The Toggle Logic

```
late AnimationController _controller;
late Animation<double> _animation;
bool _isFirstImage = true;

@override
void initState() {
  super.initState();
  _controller = AnimationController(
    duration: const Duration(milliseconds: 500),
    vsync: this,
  );
  _animation = CurvedAnimation(parent: _controller, curve: Curves.easeInOut);
}

void _toggleImage() {
  if (_isFirstImage) {
    _controller.forward(); // Play animation forward
  } else {
    _controller.reverse(); // Play animation backward
  }
  setState(() {
    _isFirstImage = !_isFirstImage;
  });
}
```

Graduate Task: Advanced State Persistence CSC 6370 ONLY

Activity: Robust State Management

This section is mandatory for graduate students and focuses on professional-grade application behavior. You must ensure the app "remembers" its state even after it is

closed and reopened.

Requirements

1. **Persistence:** Use `SharedPreferences` to save the counter value and the current image state.
2. **Reset Button:** Create a "Reset" button distinct from other controls.
3. **Safety:** Implement a Confirmation Dialog before resetting data.

Implementation Steps

1. ADDING DEPENDENCIES

Add the shared_preferences package to your `pubspec.yaml` .

2. SAVING AND LOADING DATA

You need to hook into the lifecycle of your app. Load data in `initState` and save data whenever it changes.

```
_loadState() async {
  final prefs = await SharedPreferences.getInstance();
  setState(() {
    _counter = prefs.getInt('counter') ?? 0;
    _isFirstImage = prefs.getBool('isFirstImage') ?? true;
  });
}

_saveState() async {
  final prefs = await SharedPreferences.getInstance();
  prefs.setInt('counter', _counter);
  prefs.setBool('isFirstImage', _isFirstImage);
}
```

3. THE CONFIRMATION DIALOG

Never delete user data without asking! Use the `showDialog` function.

```
Future<void> _showResetDialog() async {
  return showDialog<void>(
    context: context,
    barrierDismissible: false, // User must tap a button
    builder: (BuildContext context) {
      return AlertDialog(
        title: const Text('Confirm Reset'),
        content: const Text('Are you sure you want to clear all data?
This cannot be undone.'),
        actions: <Widget>[
          TextButton(
            child: const Text('Cancel'),
            onPressed: () => Navigator.of(context).pop(),
          ),
          TextButton(
            child: const Text('Reset'),
            onPressed: () {
              _resetApp(); // Your custom reset function
              Navigator.of(context).pop();
            },
          ),
        ],
      );
    },
  );
}
```

CRITICAL REQUIREMENT

When the user confirms the reset, you must:

1. Set the on-screen counter back to 0.
2. Revert the image to the initial asset.
3. Clear the SharedPreferences storage to ensure the reset is permanent.

Starter Code

Use this starter `main.dart` as a baseline for the counter, image toggle, animation, and theme switch. Place it in `lib/main.dart` after creating your Flutter project.

Starter Code (main.dart)

Copy

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

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

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'CW1 Counter & Toggle',
      theme: ThemeData.light(),
      darkTheme: ThemeData.dark(),
      home: const HomePage(),
    );
  }
}

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

  @override
  State<HomePage> createState() => _HomePageState();
}

class _HomePageState extends State<HomePage> with SingleTickerProviderStateMixin
{
  int _counter = 0;
  bool _isDark = false;
  bool _isFirstImage = true;
```

```
late final AnimationController _controller;
late final Animation<double> _fade;

@override
void initState() {
  super.initState();
  _controller = AnimationController(
    duration: const Duration(milliseconds: 500),
    vsync: this,
  );
  _fade = CurvedAnimation(parent: _controller, curve: Curves.easeInOut);
}

@override
void dispose() {
  _controller.dispose();
  super.dispose();
}

void _incrementCounter() {
  setState(() => _counter++);
}

void _toggleTheme() {
  setState(() => _isDark = !_isDark);
}

void _toggleImage() {
  if (_isFirstImage) {
    _controller.forward();
  } else {
    _controller.reverse();
  }
  setState(() => _isFirstImage = !_isFirstImage);
}

@override
Widget build(BuildContext context) {
  return MaterialApp(
    themeMode: _isDark ? ThemeMode.dark : ThemeMode.light,
    theme: ThemeData.light(),
```

```
      darkTheme: ThemeData.dark(),
      home: Scaffold(
        appBar: AppBar(
          title: const Text('CW1 Counter & Toggle'),
          actions: [
            IconButton(
              onPressed: _toggleTheme,
              icon: Icon(_isDark ? Icons.light_mode : Icons.dark_mode),
            ),
          ],
        ),
        body: Center(
          child: Column(
            mainAxisSize: MainAxisSize.min,
            children: [
              Text(
                'Counter: $_counter',
                style: Theme.of(context).textTheme.headlineMedium,
              ),
              const SizedBox(height: 12),
              ElevatedButton(
                onPressed: _incrementCounter,
                child: const Text('Increment'),
              ),
              const SizedBox(height: 24),
              FadeTransition(
                opacity: _fade,
                child: Image.asset(
                  _isFirstImage ? 'assets/image1.png' : 'assets/image2.png',
                  width: 180,
                  height: 180,
                  fit: BoxFit.cover,
                ),
              ),
              const SizedBox(height: 12),
              ElevatedButton(
                onPressed: _toggleImage,
                child: const Text('Toggle Image'),
              ),
            ],
          ),
        ),
```

```
        ),
      );
    }
  }
```

Submission Guidelines

SUBMISSION HUB

Everything you need to submit, in one place

Follow the checklist and steps below to finish strong.

STATE OF THE ART

WHAT TO SUBMIT

1. **GitHub Repository URL:** Make sure your partner is added as a collaborator!
2. **Text File:** Include your GitHub URL and both student names in a single text document.
3. **APK File:** Generate the build file for your Android app.

APK GENERATION STEPS

1. Open your project terminal and run `flutter pub get` to sync packages.
2. Update your app name, version, and build number in `pubspec.yaml` if needed.
3. Make sure an Android device or emulator is not running a debug build from this project.
4. Run `flutter build apk --release` to generate the release APK.
5. Find the file at `build/app/outputs/flutter-apk/app-release.apk` and submit that APK.

6. Optional: for smaller APKs by CPU type, run `flutter build apk --release --split-per-abi` and submit the appropriate APK.
7. If the build fails, run `flutter doctor` to confirm your Android toolchain is ready.

MOTIVATION

"Code is the language of logic, but creativity is the language of the heart."

Good luck, and have fun creating!

BONUS CHALLENGE

For students looking to go above and beyond: Try making the counter color change (e.g., from Green to Red) as the number gets higher, or add a sound effect when the image toggles!