Programming Assignment: Big JSON on Isolates

Flutter & Dart Asynchrony for Smooth UI

CS 1635/2035 Fall 2025

Overview

You are given a working Flutter app (uses `macos_ui` but will also run on Windows) that generates a *very large* JSON payload and then parses it to show a quick summary. Currently, both generation and parsing run on the main UI isolate. Your task is to **update the app so that both `generateJson` and `parseJson` run on Flutter's background isolates** when the switch is ON, using `compute` (from `package:flutter/foundation.dart`).

Why this matters

UI frameworks render on a single "main" thread (the UI isolate in Flutter). If you do CPU-intensive work (e.g., building a 1,000,000-item JSON string, or decoding and mapping it) on the main isolate, the UI stutters, animations freeze, and the app *feels* broken. Dart's **isolates** let you run CPU-bound work in parallel without blocking the UI. The `async/await` model keeps code readable, while `compute` spins up a background isolate to perform the heavy lifting.

Provided Code (high level tour)

All code is avilable on course GitHub:
`https://github.com/cs-1635/fall2025/tree/main/mps/mp1`

- **`BigJsonApp`** initializes a `MacosApp` and shows `HomePage`.
- **`HomePage`** state holds UI flags (`_busy`, `_status`, elapsed time, sample lines, etc.) and two key methods:
  - `generateJson()`: currently returns a huge JSON string by calling `fakeExternalGenerateBigJson(count)` on the main isolate.
  - `parseJson({String? overrideRaw})`: currently calls `fakeExternalParseJson(raw)` on the main isolate, then feeds the result to `summarize(...)` to produce a count and samples.
- **Helpers**:
  - `fakeExternalGenerateBigJson(int count)` builds a list of maps and `jsonEncode`s it.
  - `fakeExternalParseJson(String raw)` calls `jsonDecode` and projects fields.
  - `Summary summarize(dynamic parsed)` computes item count and a few sample lines.
- The UI button calls `generateJson()` then `parseJson()` in sequence. A switch toggles "Parse on isolate (good)" via `_parseGenUsingCompute`.

Your Tasks

Goal: When the user turns ON "Parse on isolate (good)", both large JSON generation *and* parsing must occur on background isolates, keeping the UI responsive. Concretely:

T1. Refactor `generateJson` to be async and, when `_parseGenUsingCompute` is true, run on an isolate via `compute`. When false, preserve current main-isolate behavior.

T2. Refactor `parseJson` to be async and, when `_parseGenUsingCompute` is true, run on an isolate via `compute`. When false, preserve current behavior.

T3. Update the button handler to be async and `await` the two steps in order: generate → parse.

T4. Maintain correct status text, progress indicator, elapsed timing, and error dialog behavior in both modes.

What to Submit

1. Updated source code with refactored `generateJson` and `parseJson` methods, showing async/await and isolate usage.
2. A short **write-up** that must include:
   - A description of how you moved work onto isolates and why this prevents UI freezing.
   - The observed performance difference: elapsed time with the switch OFF (main isolate) vs ON (background isolate).
   - An explanation of the **copying cost of isolates** — how data is passed between isolates by value rather than by reference, and how this might affect timing results for very large payloads.
   - Any bugs or edge cases you encountered and how you resolved them.

Grading Rubric (100 pts total)

1. Isolate Integration (30 pts)

- (15 pts) **generateJson** runs on isolate with `compute` when switch is ON.
  - 5 pts: Recognized need for `Future` return type.
  - 5 pts: Correct isolate wrapper and `compute` call.
  - 5 pts: UI state still updates after isolate completes.
- (15 pts) **parseJson** runs on isolate with `compute` when switch is ON.
  - 5 pts: Recognized need for `Future` return type.
  - 5 pts: Correct isolate wrapper and `compute` call.
  - 5 pts: Summary data still updates correctly after isolate completes.

2. Async/Await Flow (20 pts)

- (10 pts) Proper use of async/await in `generateJson`.
- (10 pts) Proper use of async/await in `parseJson`.

3. Button Handler & Sequencing (10 pts)

- (5 pts) Button handler made async.
- (5 pts) Correctly awaits generation before parsing.

4. UI Responsiveness & State Management (15 pts)

- (5 pts) Status text updates correctly ("Generating", "Parsing", "Done").
- (5 pts) ProgressCircle spinner shows/hides appropriately.
- (5 pts) Elapsed time and sample items displayed correctly.

5. Error Handling (10 pts)

- (5 pts) Exceptions update UI to "Error" state.
- (5 pts) Error dialog shown without crashing app.

6. Write-up Quality (15 pts)

- (5 pts) Clear explanation of isolate vs main isolate behavior and UI responsiveness.
- (5 pts) Records and compares timings for OFF vs ON modes.
- (5 pts) Explains **copy cost of isolates** and its impact on performance results.