

Requirements for Homework Assignment 4

JSON file

 [ExerciseJSON.json](#)

Overview

In this assignment, you will create a restaurant-like simulator. The simulation is supposed to show the preparation of different recipes using different ingredients provided by a JSON file. Each recipe has a step-by-step guide and durations, which must be followed to make the meal. The student is supposed to take the JSON file and parse it and then simulate the meal preparation process using multithreading logic and then using Avalonia UI to display the progression of the process. The UI should also include animations (such as an animated progress bar or visual cues to explain the progression).

Objectives

- **Data Parsing:** Take the provided JSON file, read it, and parse it
- **Multithreading:** Use multithreading logic to concurrently process different threads that simulate separate kitchen stations, which then prepare different recipes concurrently
- **UI with Animations:** Build an Avalonia UI that visualizes a real-time dashboard of the meal preparations. Include animations that will display the progress of each meal (progress bar, percentage, etc).
- **Extensibility:** This assignment also encourages students to have different approaches and be creative. There is no singular solution to this assignment, there are many.
- **Implementation:** This assignment must be implemented using C# and Avalonia UI. Additionally, document the making of the assignment (comments in code, separate Word document, etc).

Requirements

1. JSON File handling

Input Data:

- Parse the provided JSON file that contains the ingredients, recipes, and steps that include durations (in seconds or minutes). An example of a step could be: preheat the oven, wash the salad, etc.

Data Integrity:

- Validate the JSON file and ensure that any potential parsing errors are handled.

2. Meal preparation simulation:

Concurrent Processing:

- Use multithreading or tasks to simulate different kitchen stations preparing meals concurrently. Each station can pick up a different recipe and process its steps sequentially.

Step Simulation:

- For each step in each recipe, simulate the delay between the steps according to the given duration. Additionally, the simulation should not block the UI thread.

Thread-Safe Communication:

- Implement thread safe communication (locks, concurrentQueue<T> or SemaphoreSlim) to manage shared resources.

3. Avalonia UI Dashboard:

Live Updates:

- Display a grid or a list that will show current meal preparation. This should include the recipe name, current step, and the progress (percentage or time

left).

Animation Requirement:

- Include animations to visualize the progress of different processes. This could be bars that fill out, visual transitions between steps, or animated icons, etc.

User Interaction:

- Also include user interaction. This could be starting or stopping a simulation via a button, adjusting parameters like speed or frequency of new orders, or filtering active recipes.

4. **Responsiveness:**

- The UI should be responsive at all times. Additionally, all the background processes should be handled in separate threads.

5. **Scalability:**

- Make the design scalable. This could be achieved by adding additional recipes or by increasing the order frequency. Also, the performance should not be affected too significantly.

6. **Robust Error Handling:**

- Any exception errors, like JSON parsing errors or multithreading exceptions, etc., should be handled accordingly, and the simulation could have a “recovery” functionality from such errors.

7. **Maintainability and Extensibility:**

- Make the codebase modular. This means that concerns like data parsing, simulation logic, UI updates, etc., should be put into different classes or modules.

8. **Thread Safety:**

- Ensure that the shared data between concurrently running threads is accessed in a thread-safe manner to avoid potential race conditions or data corruption.

Task Description

1. JSON Data integration:

- The provided JSON file should be loaded at the start of the application.
- The data from the JSON file should be validated and parsed into appropriate data structures. Meaning you separate the data into different classes (Ingredients, Recipes, RecipeStep, etc.).

2. Simulating Meal Preparation:

- Create different threads or tasks that simulate multiple kitchen stations. Additionally, each kitchen station should be able to pick up and process a recipe.
- For the recipes, go through the current step and simulate the duration of each step using asynchronous delays. Also, update the progress of the current step.
- Use synchronization to update the shared data.

3. Avalonia UI development:

- Use Avalonia's data binding framework to refresh the UI automatically when the data changes.
- Implement animations that display the progress (animated progress bars, transitions between different steps, visual cues when a recipe has finished, etc.).
- Include buttons to give the user control over the application (start or pause simulation, adjust the speed or the order frequency, view the details of a specific recipe, etc.)

Creative Extensions

1. Role-Based feature

- Introduce additional roles such as multiple chefs, head chef (could be twice as fast), multiple waiters etc.

2. Priority Orders

- Add priority orders, such as VIP customers that can interrupt the normal flow.

3. Status Report

- Implement a status report that summarizes completed orders so the user can view the history of orders.

Grading

The bonus points will be awarded in accordance to the following requirements:

- **For one bonus point**, you must make an application that runs without any additional setup and that covers all of the *Requirements*. For this, follow the *Task Description*.
- **For two bonus points**, you must implement one of the *Creative Extensions*

Implementation Hints

- For the JSON parsing, consider using the built-in System.Text.Json libraries for parsing.
- task.run could be used for the asynchronous or await pattern or explicit threading.
- Use thread-safe collections like ConcurrentQueue<T> or any other safe thread collection for managing the shared data like active recipes.
- For the animations in Avalonia, you can use built-in animations that support progress indicators like progress bars, etc. If necessary, look up Avalonia documentation.
- Separate your concerns into different modules for parsing, simulation, and UI logic to make the structure of your application easier to understand and maintain.

- Try to include documentation. This could be any comments in your code, summaries of methods, or even a separate word document that displays your thought process.