Developer NET task

Recruitment Task – .NET Developer (Online Coding + GitHub)

Task Overview

You are asked to build a **small, modular, thread-safe order-processing console application** in **C# (.NET 6 or later)**.
The solution must demonstrate **Dependency Injection**, **logging**, **error handling**, **thread safety**, and **good software design practices**.

You will complete the code **on the online IDE** and then **push the full solution to a new public GitHub repository**.

Online Coding Start Point

https://onlinegdb.com/wqIM5tlDsM
(Copy the code from the editor – it contains the skeleton with interfaces, `Program.cs`, and sample data.)

Required Tasks (Must-Have – 80%)

1. **Implement the following interfaces**

   ```
   public interface IOrderRepository
   {
       string GetOrder(int orderId);
   }

   public interface IOrderService
   {
       void ProcessOrder(int orderId);
   }
   ```

2. **Implement a console-based logger**
   - Create `ILogger` with:

```
void LogInfo(string message);
void LogError(string message, Exception ex);
```

- Implement `ConsoleLogger` that prints timestamped messages.

3. **Implement thread-safe in-memory repository**
   - Use `ConcurrentDictionary<int, Order>` or `lock` to ensure **thread safety**.
   - Pre-load at least two sample orders (e.g., ID 1 → "Laptop", ID 2 → "Phone").
   - Throw:
     - `ArgumentException` for `orderId <= 0`
     - `KeyNotFoundException` if order not found
4. **Implement `OrderService`**
   - Inject `IOrderRepository` and `ILogger` via constructor.
   - In `ProcessOrder`:
     - Log start of processing
     - Call repository
     - Log success or catch & log any exception
5. **Create a simple DI container (manual or lightweight)**
   - Example: `ServiceContainer.CreateServices()` → returns `(IOrderService, ILogger)`
   - All dependencies must be **injected**, not created with `new` inside classes.
6. **Complete `Main` method**
   - Use the DI container
   - Run **3 parallel tasks**:
     - Process order 1
     - Process order 2
     - Process invalid order (-1)
   - Use `Task.WaitAll` or `Task.WhenAll`
   - Log final message: `"All orders processed."`

Bonus Tasks (Optional – +20% Extra Credit)

> Complete **at least 2** of the following to stand out.

#	Bonus Task	Description
1	**Asynchronous Processing**	Change `ProcessOrder` → `Task ProcessOrderAsync(int orderId)`

#	Bonus Task	Description
		Simulate delay in repository with `Task.Delay(100)` Use `await` and `Task.WhenAll` in `Main`
2	**Add Order (CRUD)**	Add `void AddOrder(Order order)` to `IOrderRepository` Ensure thread-safe insert, reject duplicates Call it from a new `Task` in `Main`
3	**IOrderValidator**	Create interface: `bool IsValid(int orderId)` Inject into `OrderService`, validate before repo call
4	**Unit Tests**	Add `xUnit` or `NUnit` project/folder Write 3+ tests (happy path, invalid ID, not found) Mock `ILogger` and `IOrderRepository` using **Moq**
5	**Configuration via appsettings.json**	Add `appsettings.json` with `LogLevel` Read via `IConfiguration`, skip `Info` logs if level is `Error`
6	**Notification Service**	`INotificationService.Send(string message)` Inject and call after successful processing

---

GitHub Repository Requirements

1. Create a **new public GitHub repository**
2. Push **at least 3 meaningful commits** (e.g., "feat: add DI container", "fix: thread safety", "test: add unit tests")
3. Include:
    - `README.md` with:
        - Project overview
        - How to run
        - Architecture diagram (text or mermaid)
        - List of completed bonus tasks
    - `.gitignore` (exclude `bin/`, `obj/`)
    - All code + tests (if any)
4. **Paste the GitHub URL** in the online editor as a comment at the top:

```
// GitHub: https://github.com/yourname/order-processing-task
```

Evaluation Criteria

Category	Weight
Dependency Injection & Loose Coupling	30%
Logging & Error Handling	30%
Thread Safety	20%
Code Quality, SOLID, Naming, Comments	20%
Bonus Tasks & Tests	+20%

Submission

1. Complete the code in the online editor
2. Run it → verify output shows logs for success and error
3. Copy final code → create GitHub repo → push
4. Submit **GitHub link** via recruitment platform

Good luck! Show us clean, testable, and production-ready C# code.