Assessment Task

General Guidelines

- The problem is described in the form of business requirements, to ass your ability to transform it to a technical solution, so make sure you read carefully to fully understand the requirement, hence implement the right thing
- Feel free to utilize modern GenAI tools like ChatGPT or GitHub Copilot to get things done faster
- Implement ***only*** what you are already ***aware*** of
- Assessment is based on what you ***already*** have ***experience*** in
- Solve as ***correct*** as you can, rather than as ***much*** as you can
- Implement it on a ***public git hub repo(s)*** and provide URI(s)

Problem Statement

- This task involves simulating the implementation of a small part of an ERP system with a focus on the purchasing process. It includes handling the **Purchase Order (PO)**, a document commonly used in enterprises to formalize procurement.
- Each PO document has a unique number that identifies the specific document instance. Please note that users use that number to identify documents among them while using the system, which means the number should be readable
- Also, it has a date that represents when the document has been issued.
- It has a price where the total amount of money that will be paid by the enterprise to the supplying entity for buying the intended goods.
- The document is used to track the buying process which is a multi-phase process, where at its beginning the PO is in a **Created** state, indicating that the process has just been initiated, then it gets converted into **Approved** state, indicating that the process is confirmed to get executed, when a supplier attempts to ship the purchased goods the order state becomes **Shipped**. Finally, it gets converted into **Closed**, where at this stage, the purchasing should have been done, by having the goods received by the enterprise inventories and that the accounting department has paid the supplying party the agreed-on price.
- During that, at any point in time, the PO could be deactivated indicating that the buying process has been put on hold, without impacting the current PO state.
- The PO document has items, each item represents a good to be purchased, and each item has a serial number in the PO document that represents each order in the document.
- Each good has a code, identifies it and the users use to identify the good while using the system. Also, the item has a price of the purchased good.
- Goods cannot be repeated on the same PO.

- The fulfillment of the PO happens by having the suppliers submit a **Shipping Order (SHO)**, where that order indicates the attempt of the good suppliers to ship the purchased goods to the enterprise.
- Each SHO document has a unique number that identifies the specific document instance, that number is used by the system users to identify documents among them while using the system.
- Different document types e.g. PO & SHO have unique number scheme so that it stands out when it's seen that this number indicates either a PO or SHO.
- The SHO always refers to the PO document it fulfills, so it's obvious to the user which PO this SHO is fulfilling.
- Also, it has a date that represents when the goods will be delivered to the purchaser.
- It has a pallet count which indicates the number of pallets the goods will be grouped in (e.g. 100 pieces in 4 pallets).
- Each SHO has items (goods), which are the same as the referenced PO in terms of count, good code, and price

Implementation Requirements

1. Apply the solution for the above problem statement using DDD considering the different bounded contexts different
2. Each context should be in a separate repo, and each repo should have a single branch only for all work even for incomplete work, without that single branch getting damaged, or impacted by the incomplete work
3. Each context should be a standalone process
4. Each process should be containerized (Docker preferred) and own its data
5. Implement the business entities so that each entity and its members should be extensible and customizable
6. The states and items are related to the entities that they will be used only with and cannot be used with any other entity
7. Build the Transactional DB structure of the entities above using Microsoft SQL Server
8. DB entities, seeding, and deployment should be done by the DB Project.
9. Make sure that the operations are logged to any free online centralized logging platform (better to be Elastic Cloud {Elasticsearch and Kibana})
10. Communication among layers should be reactive
11. Changes in DB structure shouldn't impact the business entity
12. Data retrieval and DML should be done by Microsoft EF Core
13. Data retrieval should be delayed as much as possible
14. LINQ evaluation should be done only at the DB layer

15. The business entities should be the allowed entity for communication among the layers without leaking the business outside the business layer.
16. An endpoint is required to support the creation of multiple POs at once
17. Endpoints should prevent over-fetching
18. For multiple PO creation, the POs have to be validated in each layer with only one loop across all the layers
19. After finishing implementation, and entering few Pos, change the PO number generation logic so that the PO number should be PO`[Creation stamp till millisecond], without changing any old code
20. Old and new generation logic should available at run time as per need
21. Document entities (PO, SHO) should be responsible for the modification of the document-owned items
22. The DAL layer should have POCO entities similar to the business entities, to be used by the EF core. However, the navigation properties of the business entities should be working if they are POCOs and result in on-demand data retrieval
23. Any business scenario should be atomic, where either to succeed or fail as one unit
24. Business scenarios can be executed from any application (endpoint, background worker, desktop, ..., etc.) with 0 changes in the business scenario logic
25. Data retrieval of POs should support providing filtration specifications
26. The business layer operations should be designed so that any modification of the DB design or the required data model by the endpoints has zero impact on the business layer
27. Each notification should be sent once and only once.
28. Communication among layers should be functional-based
29. The POs states should reactively be changed to "Being shipped" once a related SHO is created, and once it's closed, the PO state automatically should be "Closed" with a single subscription/handler
30. Subscription should be decoupled from the handling
31. Run time errors should be handled
32. User should be informed about business errors
33. Logging should be done while DB operations are performed IO operations should efficiently utilize the infrastructural resources
34. Populate the DB with 1M PO, and Profile PO retrieval query and find the hottest PO retrieval code path (share the profiling artifact on source control)
35. Cache the last 7 POs for fast retrieval
36. Write unit test to you your code with 50%+ code coverage, and make sure all unit tests pass successfully