Zadanie 1: Repository

Stworzyć repozytorium dla agregatów:
- pl.com.bottega.sales.domain.Order
- pl.com.bottega.sales.domain.Product

Interfejsy repozytoriów powinny znajdować się w pakiecie pl.com.bottega.sales.domain a implementacja w pakiecie pl.com.bottega.sales.infrastructure.repositories.jpa

Repozytoria powinny kasować logicznie obiekty – nie usuwamy danych. Repozytoria nie powinny zwracać usuniętych logicznie obiektów (w przyszłości będzie wprowadzony mechanizm dostępu do usuniętych danych dla użytkowników o odpowiednich uprawnieniach).

Wprowadzić klasę bazową dla Agregatów: pl.com.bottega.common.domain.BaseAggregateRoot, która będzie odpowiedzialna za:
- logiczne usuwanie
- identyfikację
- wersjonowanie (w jednym z kolejnych zadań)
- datę utworzenia (ustawiane w jednym z kolejnych zadań)
- datę ostatniej modyfikacji (ustawiane w jednym z kolejnych)

Stworzyć testy end-to-end komponentowe dla scenariuszy:
- dodanie i pobranie agregatu
- usunięcie agregatu i próba pobrania go

Zadanie 2: Agregat

Rozbudować agregat Order tak aby zawierał:

- informację o kliencie składającym zamówienie
- listę pozycji zamówienia
- całkowity koszt zamówienia
- koszt po rabacie
- status zamówienia

Agregat Order powinien oferować operacje dodania i usunięcia produktu, zmiany statusu.

Zastanowić się, które gettery i settery powinny znaleźć się w interfejsie (w

sensie publicznych metod) agregatu.

Stworzyć implementację serwisu aplikacyjnego pl.com.bottega.sales.application.services.PurchaseApplicationService, który będzie modelował Use Case'y dokonywania zakupów i operował na agregatach i repozytoriach.

Stworzyć testy end-to-end systemowe dla serwisu aplikacyjnego.

Zadanie 3: Kolekcje

Należy dopracować mapowanie kolekcji pozycji zamówienia:

- z uwagi na całkowitą agregację (kompozycję) nie potrzebujemy tabeli linkującej
- należy usuwać dane jeżeli pozycja zostanie usunięta z zamówienia
- chcemy uniknąć odtwarzania danych (usuwanie i ponowne wstawianie)
- zakładając, że granica agregatu jest poprawnie zakreślona decydujemy się na chciwe/łapczywe/gorliwe pobieranie pozycji

Zadanie 4: Dziedziczenie

Zamapować model **typów** użytkowników:

- Admin
- Supervisor
- Standard

oraz model ról:

- korektor faktur
- korektor zamówień (standardowy użytkownik nie może dokonywać korekty zamówień)

używając odpowiednich strategii dziedziczenia.

Zadanie 5: Wspólne operacje na encjach

Dodać ustawienie daty utworzenia i daty modyfikacji agregatu z wykorzystaniem callbacków.

Wstrzykąć podsystem zdarzeń i serwis zwracający bieżący czas do bazowej klasy agregatu poprzez mechanizm Listenerów. Wstrzykiwanie powinno się odbyć tylko jeśli agregat potrzebuje takiej funkcjonalności.

Zadanie 6: N+1 Select Problem

Stworzyć funkcjonalność eksportu zamówień danego użytkownika do zewnętrznego formatu (na potrzeby ćwiczenia wystarczający będzie String).

Dokonać obserwacji ilości wykonywanych zapytań dążąc do najmniejszej ich ilości.

Zadanie 7: Wyszukiwanie danych przekrojowych

Stworzyć implementację serwis pl.com.bottega.sales.read.OrderBrowser, który wyszukuje zamówienia danego klienta zwracając wynik w postaci dedykowanego Data Transfer Object

Zadanie 8: Cache drugiego poziomu i cache kwerend

Zakładając, że dane produktów i użytkowników zmieniają się relatywnie rzadko włączyć L2 cache dla encji Customer i Product.

Stworzyć serwis wyszukujący produkty ze specjalnej oferty. Zakładając, że specjalne oferty są relatywnie stabilne włączyć cache kwerend dla zapytania.

Zadanie 9: Wykorzystanie myBatis

Stworzyć alternatywną implementację serwisu pl.com.bottega.sales.read.OrderBrowser, która wykorzystuje bibliotekę myBatis.

Zadanie 10: Blokowanie

Wprowadzić optymistyczne zabezpieczenie zamówień i ich pozycji. Chcemy uniknąć sytuacji, w której jeden użytkownik zmieni adres dostawy, a inny doda do zamówienia produkt, którego nie wysyłamy do pewnych miejsc.

Zaobserwować na konsoli polecenia SQL, które pojawiają się po włączeniu blokowania.

Zadanie 11: Transakcje aplikacyjne

Pewne produkty są mocno pożądane (np. ze względu na wyprzedaż) i aplikacja musi jednocześnie obsłużyć żądanie zakupu tych produktów przez wielu różnych klientów. Zasoby w magazynie są ograniczone i firma chce uniknąć sytuacji, w której klient kupi produkt i nie będzie można zrealizować zamówienia z powodu braku towaru. Zaprojektuj rozwiązanie, które zapobiega nadsprzedaży produktów. Gdy wszystkie dostępne produkty danego typu zostaną sprzedane, aplikacja powinna odmawiać dalszej sprzedaży. Zwróć uwagę, że taki problem występuje stosunkowo rzadko, więc rozwiązanie nie powinno mieć znacząco negatywnego wpływu na wydajność obsługi żądań, których ten problem nie dotyczy.