Opgave: Lagersystem

Afleveringsbeskrivelse

Opgaven indebærer udvikling af et lagerstyringssystem som en full-stack applikation. Systemet skal bygges ved hjælp af objektorienteret programmering og designmønstrene Singleton og Object Factory. Yderligere så skal SOLID-principper anvendes.
Derudover inkluder gerne nogle af designmønstrene Observer, Strategy og Decorator hvis det giver mening.

Projektet udvikles i teams af 2-3 personer over en periode på to uger. Det forventes, at hver gruppe afleverer følgende:

- Kildekode til projektet organiseret i en GitHub-repository.
- Installationsvejledning
- En kort teknisk dokumentation med UML-diagrammer og beskrivelse af implementerede designmønstre og arkitektur.
- En kort præsentation af arkitekturvalg, teamwork-oplevelser og en live demonstration.

Projektet præsenteres i slutningen af fredag d. 11/04, hvor hvert team gennemgår deres arkitektur, brug af designmønstre og en demonstration af funktionaliteten. Det opfordres også til at præsentere nedslag i gruppe arbejdet, projektledelses værktøjer samt hvordan i har løst eller været udfordret på det at komme i mål som et team.

Scenarie beskrivelse

En mellemstor e-handelsvirksomhed, **StockFlow**, har brug for et lagerstyringssystem, der kan håndtere deres voksende lager af produkter. Systemet skal understøtte:

- Tilføjelse og fjernelse af produkter
- Oprettelse af ordrer, hvor produkter reserveres
- Håndtering af produkttransaktioner (salg, retur, overførsel mellem lagre)
- Integration med en RESTful API for at tillade dataudveksling med eksterne systemer
- En frontend til administration af lageret

Systemet skal designes med fleksibilitet og skalerbarhed for øje, så virksomheden nemt kan udvide funktionaliteten i fremtiden.

Data

Vi fanbefaler at I selv finder et eller flere datasæt at bruge i systemet eller at I genererer et datasæt med Faker. Som en backup plan kan I bruge datasættet fra Cereal opgaven.

Foreslået fremgangsmåde:

Planlægning

1. Lav en forventningsafstemning for at sætte rammerne for samarbejdet og hvad I forventer I kan nå med hensyn til omfang og kvalitet.
   Rammerne kan blandt andet være aftaler omkring:
   a. Hvilke(t) kodesprog i udvikler i og hvilke frameworks i gør brug af
   b. Struktur for opgaver/funktionaliteter
   c. Hvordan I håndterer branching
   d. Hvornår man skal pushe til github
   e. Om der skal være pull requests og code reviews.

2. Udarbejd et par grove overordnede UML-diagrammer til intern brug for forståelse af systemet.

3. Lav en overordnet planlægning for arbejdet via opsætning af et Kanban board med arbejdsopgaver.

OOP-design og database

1. Opret en database til lagring af produktinformation, ordrer og transaktioner.

2. Implementer en **Singleton**-klasse til databaseforbindelse for at sikre, at kun én instans eksisterer.
   Brug eventuelt en privat konstruktor og en statisk metode til at håndtere instansoprettelse (Java / C#) eller overskriv `def __new__(cls):` (Python)

3. Definere en baseklasse for varer med fælles egenskaber og metoder og opret specifikke underklasser for forskellige varer.

4. Implementer en **Factory**-klasse, der opretter forskellige typer af produkter baseret på inputparametre. Overvej at implementere en simpel id-generator for hver varetype, så hver ny vare får et unikt id, hvilket kan hjælpe med at spore varerne i systemet.

5. Overvej at implementere et lagerobjekt, der bruger **Observer-mønsteret** til at notificere om lagerændringer.

6. Overvej at anvende **Strategy-mønsteret** til at understøtte forskellige prisstrategier for produkter (f.eks. rabatstrategier).

7. Overvej at implementere **Decorator-mønsteret** til at tilføje ekstra egenskaber til produkter (f.eks. premium-levering eller forsikring).

API-udvikling

1. Implementer en RESTful API, der understøtter CRUD-operationer:
    a. **GET**: Hent produkter, ordrer og transaktioner.
    b. **POST**: Tilføj nye produkter og opret ordrer.
    c. **PUT**: Opdater produktinformation og ordrestatus.
    d. **DELETE**: Fjern produkter og annuller ordrer.

2. Implementer filtreringsmuligheder i API'et baseret på parametre som kategori, prisinterval og lagerstatus.

3. Sørg for, at API'et validerer data og returnerer passende HTTP-statuskoder.

4. Tilføj adgangskontrol, så kun autoriserede brugere kan oprette, opdatere og slette data.

Bonus (hvis tid):

Frontend

1. Implementer en webapplikation, der viser lagerstatus og ordrehistorik i et overskueligt format.

2. Implementer at webapplikationen henter data fra API'et via asynkrone GET requests.

3. Frontend'en skal kunne kommunikere med API'et og tillade brugere at oprette, redigere og slette produkter.

4. Implementer filtrerings- og sorteringsmuligheder for produktlisten.

5. Gør UI'et responsivt og brugervenligt.

6. Sørg for, at layoutet tilpasser sig forskellige skærmstørrelser, så det er brugervenligt både på desktop og mobile enheder.

Test suite

- Tilføj unittests.
- Tilføj integrationstests.

Multithreading

- Implementer en threadsafe database singleton

Læringsmål

De primære læringsmål som I skal fokusere mest på er:

- Forståelse og anvendelse af OOP-principper og SOLID.
- Implementering af designmønstre i praksis.
- SCRUM og teamwork.

De sekundære læringsmål som er knapt så vigtige at fokusere på er:

- Udvikling af en full-stack applikation.
- API-design og integration.

Kodeinspiration:

- **TypeScript basics:** TypeScript fundamentals
- **Web build med ASP.NET og React:** ASP.NET Core 6 og React
- **HTML, CSS, JavaScript:** Grundlæggende webudvikling
- **YouTube-guide:** Frontend udvikling med MVVM
- **Java:** Guide to Spring Boot
- **C#:** ASP.NET Core Web API
- **Python:** Introduction to Flask