Wymagania

System służy do zarządzania obiegiem dokumentów, które wspierają system jakości (np. ISO) w przedsiębiorstwie. System powinien byś otwarty na rozbudowę w kierunku wspierania innych systemów jakości.

Podstawowym artefaktem w systemie jest Dokument, który zawiera treść mającą na calu pomóc pracownikom pracować nad zwiększaniem jakości produkcji.

Są różne rodzaje dokumentów: Księga jakości, Księga procedur, Instrukcje,..

Generalny przepływ dokumentu wygląda tak:

- dokument jest tworzony przez managera jakości – jest wówczas w statusie Draft
- dokument jest weryfikowany przez innego managera jakości – Verified
- zmiana treści/tytułu powoduje powrót do statusu Draft
- dokument jest publikowany
    - podczas publikacji określamy działy w firmie, które muszą się zapoznać z treścią dokumentu,
    - wszyscy pracownicy z tych działów muszą zostać powiadomieni o ukazaniu się dokumentu,
    - niektórzy pracownicy nie potrafią korzystać z komputerów, dla nich są drukowane kopie, które muszą przeczytać, manager takiego pracownika odnotowuje w systemie fakt, że dokument został przez niego przeczytany
    - dokumenty powinny być podpisane cyfrowo, aby zabezpieczyć się przez zmianą treści na wypadek pozwów
- dokument wygasa (staje się nieaktualny) i trafia do archiwum
- jest tworzona nowa wersja dokumentu

System powinien być zintegrowany z:

- dowolnym system kadrowym przechowującym dane o pracownikach (np. adresy mail do wysyłki),
- dowolnym systemem do masowych wydruków
- dowolną biblioteką podpisów cyfrowych

System powinien być dostępny w formie:

- Aplikacji web (pełna funkcjonalność)
- Aplikacji mobilnej (czytanie dokumentów, potwierdzanie przeczytania przez managera w imieniu pracowników wykluczonych cyfrowo)
- API web serwisów do integracji w przyszłości z innymi systemami

Zadanie: Komponenty API

Stwórz serwisy (zaproponuj parametry metod) realizujące kompletny proces biznesowy:

- DocumentsCatalog – przegląd katalogu zdjęć
  - find – zwraca kolekcję dokumentów wg zadanych kryteriów
  - get – zwraca dokument wg numeru
- DocumentFlowProcess – proces obiegu dokumentów
  - create – wtorzy dokument danego typu, autorstwa danego użytkownia o zadanym tytule; zwraca numer dokumentu
  - change – zmienia treść dokument o zadanym numerze
  - verify – weryfikuje zadany dokument przez zdanego użytkownika
  - publish – publikuje zadany dokument
  - archive – archiwizuje zadany dokument
  - newVersion – tworzy nową wersję zadane dokumentu; zwraca numer nowej wersji
- ReadingConfirmator – zarządza potwierdzeniami przeczytania dokumentów
  - zaproponuj API – metody i ich parametry

Zadanie: Singleton

Stworzyć klasę globalnego zarządzania błędami: ErrorsRegistry
Obiekt tej klasy pozwala zarejestrować błąd metodą *log,* która przyjmuje komunikat i nazwę klasy zgłaszającej błąd.

Założenia:

- Obiekt tej klasy powinien być tworzony dopiero gdy będzie potrzebny
- Dostęp do obiektu może być współbieżny.

Zadanie: Strategy

Zmodyfikuj klasę reprezentującą Dokument tak aby realizowała wymagania:

- Dokument podczas tworzenia ma automatycznie generowany numer – zależnie od ustawień systemowych. System może działać w trybie obsługi dokumentów ISO, QEP, itp.
- Dokument podczas publikowania ma obliczany koszt wydruku jednej sztuki.
  - Wydruk czarno-biały, wydruk kolorowy.
  - typ wydruku zależy od ustawień systemowych.

Wskazówki:

- Klasa Document nie powinna wiedzieć o tym jaki jest aktualny system jakości – jest to poza jej odpowiedzialnością
  - Stwórz interfejs NumberGenerator z metodą generateNumber(DocumentType) zwracającą Number
    - Stwórz konkretne implementacje tego interfejsu: ISONumberGenerator i QEPNumberGenerator (tymczasowo mogą one generować losowy ciąg znaków)
  - Przekaż stworzony interfejs do konstruktora klasy Document, który wywoła metodę generateNumber aby ustawić pole number
    - uwaga: nie jest to rozwiązanie ostateczne
- Klasa Document nie powinna również wiedzieć o tym jaki jest aktualny sposób obliczania ceny wydruku dokumentu
  - Stwórz interfejs PriceCalculator z metodą calculatePrice(Document) zwracającą Money
    - Stwórz konkretne implementacje tego interfejsu: RGBPriceCalculator i BWPriceCalculator – każda z nich może tymczasowo mnożyć ilość stron dokumentu przez cenę aby wyliczyć koszt.
  - Przekaż interfejs PriceCalculator w parametrze metody publish klasy Dcument
    - obliczanie ceny następuje dopiero w momencie publikacji
    - przekazanie strategii w konstruktorze klasy Document jest zbyt wczesne – strategia może zostać zmieniona przez administratora systemu zanim dany dokument zostanie opublikowany

Przetestować:

- Tworzenie dokumentu
- Operację publish() na dokumencie (odpowiedni status dokumentu i odpowiednia cena)
- Polityki obliczania ceny
- Polityki generowanie numeru

Zadanie: Strategy – dyskusja

Jak sądzisz:

- Jak wprowadzenie strategii przływa na ilość przypadków testowych jakie należy zweryfikować?

- Jak wprowadzenie strategii wpływa na rozszerzalność projektu?
- Gdzie widzisz zastosowanie zasady SOLID: Open-Closed Principle?
- Podaj przykłady z projektów nad jakimi pracujesz/pracowałeś gdzie takie rozwiązanie byłoby korzystne i uzasadnione
- Jak przekazywanie strategii do konstruktora lub metody wpływa na testowalność kodu klasy Document?

Zadanie: Factory idiom – podejście 1

Tworzenie strategii należy enkapsulować w Fabrykach, jako szczegół implementacyjny.

Mamy tutaj do czynienia z dwoma podejściami:

- fabrykowanie strategii
- fabrykowanie obiektu, który zawiera strategie

Wskazówki:

- Stwórz klasę DocumentFactory, która przyjmuje Typ i Autora a zwraca dokument
  - Fabryka powinna „wstrzyknąć" odpowiednią implementację NumberGenerator przez konstruktor Document
  - Zakładamy, że fabryka pobrałaby informację o systemie jakości (ISO/QEP/inne) z konfiguracji – póki co możemy symulować to zachowanie przez stałą
  - Fabryka będzie wykorzystywana w wyższej warstwie w serwisie: DocumentFlowProcess->create

- Stwórz klasę PriceCalulatorFactory, która zwraca konkretną implementację sposobu obliczania cen wydruku
  - Zakładamy, że fabryka pobrałaby informację o sposobie wydruku z konfiguracji – póki co możemy symulować to zachowanie przez stałą
  - Fabryka będzie wykorzystywana w wyższej warstwie w serwisie: DocumentFlowProcess->publish:

    document→publish(priceCalculatorFactory->createPriceCalculator())

Zadanie: Factory idiom – podejście 2

Skoro klasa DocumentFactory buduje instancje Document, to możemy usunąć zależność pomiędzy Document a NumberGenerator.

Niech teraz Document przyjmuje w konstruktorze Number wygenerowany w fabryce przez NumberGenerator.

Dyskusja:

- Jak taka zmiana wpłynęła na testowalność klasy Document (ilość potrzebnych zaślepek)?
- Jak ochronić konstruktor klasy Document przez wywołaniem z niepoprawnym obiektem klasy Number?

Zadanie: Strategy i Decorator

Założenia dla cen:

- Sposób obliczania ceny wydruku może zależeć od wielu czynników (np.: typ dokumentu, kolor)
  - Typ MANUAL zwiększa całkowity koszt o 30%
  - Jeżeli ilość stron jest większa niż 100, wówczas zwiększa to koszt początkowy o 40.
- Czynniki te mogą zmieniać się dynamicznie, zatem struktura algorytmu obliczani powinna być „składana” również dynamicznie.

Założenia dla numerów

- Generowanie numerów dokumentu zależy od nakładających się czynników
- W wersji DEMO każdy numer ma przedrostek „demo”
- Jeżeli z systemem pracuje audytor, wówczas każdy numer ma przyrostek „audit”

Zadanie: Chain of Responsibility 1

Zaimplementować mechanizm walidacji dokumentu podczas zmiany jego stanów:

- weryfikacji
- publikacji

Zmiana stanu poprzez metodę biznesową.

W zależności od statusu w jakim znajduje się dokument należy zastosować inne kryteria walidacji:

W ISO:

- Dla dokumentu w stanie VERIFIED musi być ustawiony numer
- Dla dokumenty w stanie PUBLISHED musi być ustawiona data wygaśnięcia.

W QEP

- Dla dokumentu w stanie VERIFIED musi być ustawiony autor i data wygaśnięcia
- Dla dokumenty w stanie PUBLISHED musi istnieć treść dokumentu

Odpowiedni walidator dobrać w zależności od statusu dokumentu.

UWAGA: kryteria walidacji w danym statusie będą inne dla różnych systemów jakości (np. weryfikacja w ISO wymaga daty i numeru a w QEP numeru i nazwy), należy zaprojektować rozwiązanie, które będzie uwzględniało łatwe modyfikacje przy wprowadzaniu obsługi nowych systemów jakości.

Zadanie: Chain of Responsibility 2

Dokonać refaktoryzacji poprzedniego rozwiązania do postaci używającej Managera Walidacji.

Zadanie: Builder 1

Dodać funkcjonalność eksportu Dokumentu do strumienia. Zaimplementować jeden z możliwych strumieni wyjściowych: strumień CSV o następującej strukturze:

id, tytuł, status, typ dokumentu, data wygaśnięcia, osoba przypisana 1 | osoba przypisana N

Założenia:

- Może istnieć wiele możliwych form wynikowych
- Jedną z nich jest strumień bajtów
- Inne to np.: XML, JSON, PDF, itp.

Zadanie: Assembler 1

Dodać serwis aplikacyjny służący do wyszukiwania listy dokumentów spełniających pewne kryteria. Serwis jako argument przyjmuje obiekt DocumentSearchCriteria. Klasa DocumentSearchCriteria pozwala na ustawienie przy pomocy setetrów następujących kryteriów biznesowych:

- wygasły (data wygaśniecja < now, status != archived)
- opublikowany (data wygasniecia > now, status =- published)
- wygasający za tydzień (data wygasniecia < now + 7d)

- danego typu

Kryteria można łączyć.

Klasa udostępnia przy pomocy getterów pola modelu, które będą użyte w serwisie do złożenia odpowiedniego zapytania do bazy danych.

Zadanie: Assembler 2

Stworzyć test jednostkowy Dokumentu, który sprawdza:

- tworzenie nowego dokumentu
- publikowanie dokumentu
- niemożność dwukrotnego opublikowania dokumentu
- niemożność zmiany tytułu opublikowanego dokumentu

Zadanie: State

Zaimplementować maszynę stanów odpowiadającą przejściom statusów dokumentów wg schematu:

DRAFT->VERIFIED->PUBLISHED->ARCHIVE

Dodatkowe założenia:

- z każdego statusu można przejść do statusu ARCHIVE
- z ARCHIVE można przejść do DRAFT
- z PUBLISHED można przejść do VERIFIED i DRAFT
- z VERIFIED można przejść do DRAFT

Jeżeli zostanie wywołanie nielegalne przejście wówczas należy rzucić wyjątek.

Podczas przechodzenia pomiędzy statusami należy odnotować daty zajścia tych faktów

Refaktoryzacja:

Przenieść do maszyny stanów funkcjonalności:

- obliczania kosztu
- walidacji

Zadanie: Template Method

Zaimplementować funkcjonalność kopiowania dokumentu w osobnej klasie DocumentCopier.

Generalna zasada działania kopiowania polega na:

- przepisaniu kluczowych wartości: typ, tytuł, ilość stron, autor
- zainicjowaniu unikalnych wartości: numer
- wyliczeniu wartości: dataWygaśnięcia, dataUtworzenia

Założenie:

Kroki: generowanie numer i wyliczenie daty Wygaśnięcia są abstrakcyjne, ponieważ zależą od specyfiki konfiguracji itp., dlatego powinny znaleźć się poza ogólnym algorytmem.

Zadanie: Role Object

Zaimplementować model ról użytkownika:

- Korektor dokumentów
- Oceniający dokumenty

przy założeniu następujących typów użytkowników:

- Audytor
- Autor
- Manager jakości

Zadanie: Adapter

Dodać symulację integracji z systemem firmy trzeciej, odpowiedzialnym za generowanie numerów dokumentów. System ten zwraca trzy łańcuchy znaków: numer główny, numer drugiego rzędu, numer wersji.

Format po integracji: x-y-z

Założenie:

Chcemy zachować spójność corowych funkcjonalności.

Zadanie: Wrapper

Dokonać refaktoryzacji reprezentacji kosztów wydruku. Koszt powinien być przechowywany w specjalnej klasie Money, która enkapsuluje:

- walutę w jakiej jest przechowywana wartość (dodać metodę zwracającą wartość w zadanej walucie)
- sposób reprezentacji
- operacje na walutach

Uwagi:

W JPA posłużyć się adnotacją @Embedable aby uniknąć tworzenia dodatkowych tabel

Zadanie: Specification

Dodać serwis biznesowy oferujący funkcjonalność wyszukania na liście dokumentów tych z nich, które wymagają audytu.

Kryteria audytu mogą należeć do puli następujących kryteriów:

- pewien określony status
- pewien określony typ dokumentu
- określona grupa autorów
- dataUtworzenia mniejsza lub większa niż zadana
- tytuł zawierający określone słowa
- …

Założenie:

- Kryteria mogą być dowolnie kombinowane a pomiędzy nimi mogą zachodzić dowolne relacje logiczne.
- To z jakich kryteriów składa się aktualny warunek może zależeć od aktualnie zalogowanego użytkownika, konfiguracji systemu (wdrożenia) lub ew. może być określone na GUI

Wprowadzić dodatkową klasę Budowniczego kryteriów, która dostarcza wygodnego api, typu: ANY, ALL, PUBLISHED, ASSIGNED_TO_CURRENT_USER itp.

Zadanie: Command

Stworzyć API w warstwie aplikacji, które pozwala operować na dokumentach przy pomocy poleceń. Zaimplementować polecenia:

- Utwórz nowy dokument o zadanych parametrach, przez zadanego autora
- Przypisz dokument do zadanych osób i opublikuj go
- Przedłóż okres ważności dokumentu do zadanej daty

Dodatkowe wymagania:

- System powinien wspierać model Multitennancy i wersjowanie API per dzierżawca
- Niektórzy dzierżawcy mogą potrzebować dodatkowych usług, takich jak np. szpiegowanie swych pracowników
- System powinien odrzucać duplikaty poleceń
- System powinien wspierać polecenia asynchroniczne

Zadanie: Proxy

Założenie:

- Integracja z podsystemem generującym numery dokumentów sprawia problemy, ponieważ jest on często niedostępny.
- Nie mamy dostępu do kodu adaptera

Stworzyć klasę proxy, która próbuje łączyć się z serwisem. W razie niepowodzenia korzysta z własnego mechanizmu generowania numerów.

Zadanie: Observer

Dokument powinien publikować zdarzenia na temat zmiany swoich statusów.

Założenie:

- Słuchacze mogą dodawać nowe funkcjonalności do systemu (np. mailing)
- Zestaw wpiętych słuchaczy zależy do specyfiki wdrożenia systemu

Zadanie: Pub/Sub

Dokonać refaktoryzacji Observera do Pub/Sub opartego o broker zdarzeń.

Zadanie: Facade

Przykryć funkcjonalność filtrowania dokumentów wymagających audytu wygodną fasdą, która zwraca dokumenty opublikowane, które wygasają w ciągu tygodnia i są autorstwa aktualnie zalogowanego użytkownika.