

Uniwersytet SWPS  
Wydział Projektowania w Warszawie  
Informatyka  
studia pierwszego stopnia

autor: Tetiana Borozdina  
nr albumu studenta: 73795

Praca licencjacka

**[TYTUŁ PRACY DYPLOMOWEJ]**

tytuł w języku polskim: [wpisać]

Praca napisana pod kierunkiem  
[tytuł/stopień naukowy, imię i nazwisko promotora]

Warszawa 2025

## Spis treści

Wprowadzenie .....	1
Metodologia .....	2
Projektowanie .....	2
Wyznaczenie wymagań do funkcjonalności aplikacji .....	2
Wyznaczenie przestrzeni projektowej .....	3
Organizacja bazy danych .....	4
Architektura serwisowa .....	8
Implementacja.....	11
Organizacja serwisów z dostarczenia danych strukturyzowanych .....	11
Organizacja serwisu z dostarczenia danych multimedialnych .....	12
Organizacja serwisu z przekazywania danych według logiki biznesowej .....	13
Organizacja serwisów z prezentacji widoków .....	13
Wyniki.....	14
Wnioski.....	15
Bibliografia .....	16

## Streszczenie

...

## Wprowadzenie

Wzrost ilości kontentu media do konsumencji w ostatnich czasach oznaczył wzrost liczby opcji do obejrzenia przez potencjalnych odbiorcy. Dla ostatnich to znaczyło rozszerzenie własnego zakresu preferowań oraz zyskanie możliwości dopasowania do już ustalonych interesów osobistych, nie uwzględniając jedynie dostępność kontentu. Według tego, wygodnie było by korzystać z wyszukiwania po pewnych kategoriach, gdyż dane o filmach są zebrane w jednym miejscu, tworząc niektórą bibliotekę.

Postawienie problemu: projektowanie aplikacji webowej, która: przedstawia pomóc użytkownikom w znajdowaniu, filtracji kina, uporządkowaniu informacji o nim, w podebraniu media kontentu według zainteresowań; wyświetla i kategoryzuje informacje powiązane, w tym informacje o podmiotach biorących udział w powstaniu kontentu, inne; pozwala dodawać i wyświetlać oceny użytkowników na pewne filmy.

Do rozwiązania danego problemu niezbędne jest:

- a) przemyślenie struktury modularnej projektu, na poziomie oddzielnych serwisów;
- b) przemyślenie struktury bazy danych, która będzie zachowała informacje o różnych encjach;
- c) organizowanie komunikacji pomiędzy serwisami;
- d) przemyślenie procesu projektowego z testowaniem, wdrażaniem aplikacji.

W każdym z powyższych punktów można przedstawić wynikające podproblemy.

Dla implementacji rozwiązania zdecydowane jest do użycia podejście mikroserwisowe, polegające na podziale oprogramowania na mniejsze niezależne części, komunikujące się między sobą. Dane podejście: a) pozwala na wydzieleniu i skupieniu się na poszczególnych częściach podczas pisania kodu z już gotowym rozkładem odpowiedzialności dla każdej z jednostek; b) umożliwia przeprowadzenie testów dla oddzielnych składników bez konieczności uruchamiania całego rozwiązania; c) wprowadza przejrzystość w strukturę oprogramowania, czym ułatwia znajdowanie oraz eliminację jednostek nadmiarowych.

## **Metodologia**

### **Projektowanie**

W danym rozdziale zostaną opisane szczegóły projektowe, dotyczące formalizacji zadania z przedstawieniem decyzji podjętych wobec obrania pewnych podejść projektowych.

#### **Wyznaczenie wymagań do funkcjonalności aplikacji**

Pierwszym etapem do formalizacji aplikacji webowej jest wyznaczenie wymagań do jej działania oraz logiki biznesowej.

#### ***Funkcjonalność użytkownika***

Podstawowa funkcjonalność danej aplikacji z punktu widzenia użytkownika będzie się różniła w zależności od trybu użytkownika – anonimowy „gościa” lub „zalogowany”, oraz jego roli w serwisie. Na razie przewidywane są dwie podstawowe role – zwykłego użytkownika oraz administratora.

W trybie anonimowym użytkownikowi dostępne są następujące działania:

- przeglądanie informacji o media, nawigacja po stronie internetowej według linków pod filmami z informacjami uzupełniającymi;
- wyszukiwanie, filtracja po kategoriach informacji o kinie, lub innych informacjach (np. artystach, kraju).
- założenie konta, lub logowanie się do serwisu.

Założenie konta lub same logowanie umożliwiają przejście do nadania innego statusu użytkownikowi. Po logowaniu wskazana wyżej funkcjonalność rozszerza się do następnych działań:

- ocenianie filmów po ustalonej skali ocen, od jednego do dziesięciu;
- markowanie filmu etykietami: „ulubiony”, „zaplanowany”, „w trakcie oglądania”, „obejrzany”;
- podstawowej kustomizacji własnego profilu z opcjonalnym umieszczeniem informacji o sobie i ulubionych filmach, gatunkach filmowych.

Do działań dodatkowych można odnieść wylogowanie, lub usunięcie konta. Tryb

administratora dodaje użytkownikowi dodatkowe uprawnienia na:

- dodanie, zmianę, lub usunięcie informacji o filmie, a także informacjach powiązanych;
- przeprowadzenie moderacji z usunięciem konta zwykłego użytkownika.

Dla kontroli roli administratora domyśla się o dodanie roli „super administrator”, który będzie w stanie przeznaczać rolę administratora zwykłym użytkownikom, lub je usuwać.

W taki sposób powstaje niektóra hierarchia użytkowników z różnym zakresem możliwych czynności, dla jednej części z których informacje przedstawiają się dopiero do czytania, a dla innej także do zapisu lub zmiany.

### ***Funkcjonalność aplikacji***

Z punktu widzenia działania samej aplikacji jej funkcjonalność polega na:

- wydaniu odpowiedzi na zapytania klienckie w wyglądzie stron internetowych;
- automatycznym formułowaniu widoków stron;
- wyciąganiu odpowiednich informacji z bazy danych poprzez podłączenie się do niej;
- filtracji, sortowaniu, wyszukiwaniu danych po słowom kluczowym;
- śledzeniu za spójnością nadchodzących danych, częściowo ich sensowością;
- opcjonalnym logowaniu, które zawiera uwierzytelnianie tożsamości użytkownika oraz autoryzację z nadaniem dostępu do pewnego zakresu działań;

### **Wyznaczenie przestrzeni projektowej**

W jakości podejścia do ułatwienia procesu przejścia od formalizacji do bezpośredniej implementacji kierowało się już przyjętymi powszechnie regułami podejścia Domain-Driven Design: podejście, które wymaga projektowania oprogramowania tak, aby niektóra jej część odzwierciedlała modele dziedziny w sposób, jak najwięcej zbliżony do rzeczywistych pojęć [1, s. 29]. Do tego, dziedzina staje centralnym obiektem w procesie projektowym, gdy każda z innych części systemu zależy na jej wyznaczeniu.

Przestrzeń projektową formuje zbiór pojęć którymi korzysta się w ciągu procesu projektowego. Jej ustalenie, inaczej definicja tych pojęć, sprzyja współrozumieniu, w pierwszym

rzędzie, pomiędzy stroną zamawiającą a stroną wykonującą, ale również ułatwia komunikację pracowników z różnych obszarów projektowych. Przestrzeń projektowa niesie nazwę, pod którym łączy podany zbiór pojęć, może dziedziczyć nazwę samej aplikacji.

W przestrzeni projektowej aplikacji, powiązanej ze zbieraniem materiału encyklopedycznego, większość pojęć jest powiązana same z treścią przedstawianej na stronie informacji. Wyodrębnia się w taki sposób dziedzina odpowiadająca tematyce, której w danym przypadku jest dziedzina informacji o kinie. Dziedzina ta formuje się z takich obiektów, jak „film“, „wytwórnia“, „artysta“, „reżyser“, „producent“, „scenarzysta“, które jednocześnie stanowią kategorie informacji wykazywanych na stronie.

Innymi pojęciami przestrzeni opisuje się obiekty wchodzące w interakcję z treściami na stronie, są to kategorię osób – „gość” (osoba niezalogowana), „użytkownik” (osoba zalogowana), „administrator”, „super administrator”.

### **Organizacja bazy danych**

Na poprzednim etapie zostały wyznaczone obiekty przestrzeni projektowej. W tej części odbywa się dalsza formalizacja problemu, ze skupieniem na strukturze wyznaczonych kategorii danych. Struktura zaprojektowanych poniżej modeli orientowana jest na ich przechowywanie w bazie typu NoSQL.

Organizacja lub projektowanie bazy danych pozwala na bardziej szczególne modelowanie struktury danych. Same pojęcie bazy danych oznacza zbiór połączonych informacji o wartościach bezwzględnych, które mogą być zapisane, oraz są spójne logicznie [2, s. 5]. Z kolei do jej projektowania wchodzi wyznaczenie encji, którymi są obiekty opisywane bazą danych, modelujące obiekty przestrzeni projektowej, oraz przeliczenie listy atrybutów, czyli cech, które będą opisywać te encje przez pewne przeznaczone wartości [2, s. 6].

Niekiedy do opisu encji używa się atrybutów o typach danych, odmiennych od podstawowych, które mogą logicznie łączyć cechy w inną grupę. W przełożeniu na bazy danych, dane opisujące encje pochodzą z innych tabel, a w tabeli głównej encji z kolei występują tylko posyłania na rekordy innych tabel – są to klucze obce. Pełny model, który powstaje przez połączenie informacji zawierającej się w innych tabelach jest nazywany agregatowym [1, s. 76]. Z kolei nie każda tabela, która niesie pewną informację, zasługuje na osobną reprezentację przez oddzielną encję; rekordy z tych tabeli rozstrzygają się jako informacje złożonego typu, gdyż sama unikatowość takich rekordów nie gra dużej roli. W takim razie rekord, który prezentuje jedynie

opis nie posiadając własnej tożsamości, nazywa się obiektem wartościowym (ang. „value object”) [2, s. 59].

Encje projektowanej aplikacji bazowane są na wyznaczonych pojęciach dziedziny projektowej. Modelowanie struktury danych złożonych rozpoczęto z formalizacji pojęcia użytkowników. Nie zważając na to, że wprowadzone na poprzednim etapie pojęcia są rozróżnialne według dostępnych im zasobów, jednak samych nazw tych ról jest całkiem wystarczająco do poznaczenia różnicy między nimi. Oprócz tego oczywiste jest, że tym pojęciom użytkowników można nadać podobne listy cech. Precyzowanie z dopasowywaniem zasobów nie będzie wchodziło do funkcjonalności bazy zawierającej modele programowe, jednak będzie odbywało się później. Z tego powodu przedstawione pojęcia zdecydowano jest połączyć pod jedną encją „użytkownik” z atrybutami nadanymi niżej w tabeli.

User		
Nazwa	Typ danych	Opis
Id	String	Identyfikator zapisu.
Username	String	Imię unikatowe użytkownika.
Birthdate	Date	Data urodzenia (opcjonalnie).
Picture	String	Nazwa zdjęcia profilu.
CinemaRecord		
Id	String	Identyfikator zapisu.
CinemaId	String	Identyfikator filmu.
UserId	String	Identyfikator użytkownika.
Label	Label	Etykieta, którą oznaczono film.
AddedAt	DateTime	Czas etykietowania filmu.
Label		
Value	Integer	Znaczenie przeznaczone etykietce.
Name	String	Nazwa etykiety.
CinemaRating		
Id	String	Identyfikator zapisu.
CinemaId	String	Identyfikator filmu.
UserId	String	Identyfikator użytkownika.
RatingScore	Double	Znaczenie oceny.



Tabela 1 – Model agregatowy encji „użytkownik” („user”).

Pierwsze z pojęć dziedziny tematycznej odpowiada obiektowi „film” i cechuje się atrybutami, opisanymi w następującej tabeli:

Cinema		
Nazwa	Typ danych	Opis
Id	String	Identyfikator zapisu.
Name	String	Nazwa filmu.
ReleaseDate	Date	Data wyjścia.
Genres	Genre	Gatunki filmowe.
Language	Language	Język oryginału filmu.
RatingScore	Double	Średnia z ocen użytkowników.
ProductionStudios	Array<StudioRecord>	Wytwórnie filmowe produkujące film.
Starrings	Array<Starring>	Osoby uczestniczące w produkcji.
Description	String	Opis filmu.
Picture	String	Nazwa zdjęcia plakatu.
Genre		
Value	Integer	Znaczenie przypisane gatunku.
Name	String	Nazwa gatunku.
Language		
Value	Integer	Znaczenie przypisane gatunku.
Name	String	Nazwa języka.
StudioRecord		
Id	String	Identyfikator wytwórni.
Name	String	Nazwa wytwórni.
Picture	String	Nazwa zdjęcia wytwórni.
Starring		
Id	String	Identyfikator osoby.
Name	String	Imię osoby.
Jobs	Job	Rola osoby w produkcji filmu.
ActorRole	ActorRole	Rola, którą gra artysta (opcjonalnie).
Picture	String	Nazwa zdjęcia osoby.

Job		
Value	Integer	Znaczenie przypisane roli w produkcji.
Name	String	Nazwa roli w produkcji.
ActorRole		
Id	String	Identyfikator roli.
Name	String	Nazwa roli w filmie.
Priority	RolePriority	Prioryteta roli.
RolePriority		
Value	Integer	Znaczenie przypisane priorytetowi.
Name	String	Nazwa poziomu prioryteta.

Tabela 2 – Model agregatowy encji „film” („cinema”).

Następną encją służy encja „osoba”, która reprezentuje jednocześnie modeli: „artysta“, „reżyser“, „producent“, „scenarzysta“ – ostatnie nazwy zdecydowano wykorzystać dopiero dla oznaczenia zawodu pewnej osoby.

Person		
Nazwa	Typ danych	Opis
Id	String	Identyfikator zapisu.
Name	String	Imię osoby.
BirthDate	Date	Data urodzenia.
Country	Country	Kraj urodzenia.
Jobs	Job	Zawody osoby.
Filmography	Array<CinemaRecord>	Filmy, w których osoba uczestniża.
Description	String	Informacja o życiu osoby.
Picture	String	Nazwa zdjęcia osoby.
Country		
Value	String	Znaczenie przypisane kraju.
Name	String	Nazwa kraju.
Job		
Value	Integer	Znaczenie przypisane roli w produkcji.
Name	String	Nazwa zawodu.
CinemaRecord		

Id	String	Identyfikator wytwórni.
Name	String	Nazwa filmu.
Year	Integer	Rok wyjścia filmu.
Picture	String	Nazwa zdjęcia plakatu.

Tabela 3 – Model agregatowy encji „osoba” („person”).

Ostatnią encją opisuje schemat reprezentujący wytwórnię filmową.

Studio		
Nazwa	Typ danych	Opis
Id	String	Identyfikator zapisu.
Name	String	Nazwa wytwórni.
FoundDate	DateTime	Data założenia.
Country	Country	Kraj operowania się.
Filmography	Array<CinemaRecord>	Filmy, produkowane przez wytwórnię.
PresidentName	String	Imię prezesa firmy.
Description	String	Informacja o wytwórni.
Picture	String	Nazwa zdjęcia wytwórni.
Country		
Value	Integer	Znaczenie przypisane kraju.
Name	String	Nazwa kraju.
CinemaRecord		
Id	String	Identyfikator wytwórni.
Name	String	Nazwa filmu.
Year	Integer	Rok wyjścia filmu.
Picture	String	Nazwa zdjęcia plakatu.

Tabela 4 – Model agregatowy encji „wytwórnia” („studio”).

### Architektura serwisowa

Zamiast pojedynczego złożonego systemu w jakości stylu projektowego zostało wybrane podejście mikroserwisowe. Aplikacja mikroserwisowa to zbiór autonomicznych usług, wykonujących proste pojedyncze polecenia, które współpracują ze sobą, aby wykonywać bardziej

złożone operacje [3, s. 4]. Usługi te komunikują ze sobą za pośrednictwem niezależnych od technologii protokołów przesyłania wiadomości, zarówno punkt-punkt, jak i asynchronicznie. Zaletami architektury mikroservisowej są łatwość w utrzymaniu, niezależność we wdrażaniu oraz w skalowalności oraz mały rozmiar każdej z komponenty systemu [4, s. 14]. Dana lista posłużyła motywacją do wybrania same danego typu architektury do aplikacji webowej.

Komunikacja między serwisami oraz ich klientami jest zorganizowana na podstawie regułach REST (ang. „representational state transfer”), który jest mechanizmem komunikacji międzyprocesowych. Definiuje się jego jak mechanizm zapewniający zestaw ograniczeń, które stosowane jako całość, umożliwiają skalowalność interakcji komponentów, ogólność interfejsów, niezależne wdrażanie komponentów pośredniczących w celu zmniejszenia opóźnień interakcji, wyegzekwowania bezpieczeństwa i hermetyzacji starszych systemów [4, s. 73]. W ten sposób nadaje się jego do implementacji niezależnych serwisów, które implementują i wystawiają pewny szereg poleceń, z którego mogą korzystać serwisy klienckie. Protokołem do przesyłania informacji o obiektach dziedziny służy HTTP, pod słowami kluczowymi którego (GET, POST, PUT, DELETE) za pewnymi ścieżkami URL wyznaczają się działania, odbywające się nad zasobami.

Ogólnym wzorem projektowym do tworzenia serwisów stał wzór Model-View-Controller (MVC), który wyznacza typową aplikację, złożoną z trzech warstw technicznych: warstwy danych, warstwy logicznej, warstwy prezentacyjnej [3, s. 52]. Dany wzór na początku został podzielony horyzontalnie na odpowiednie serwisy:

- a) serwis, zajmujący się bezpośrednim zwracaniem do bazy danych, który jednocześnie przedstawia interfejs dla zwrócenia się do danych według REST;
- b) serwis, przedstawiający jedynie interfejs do zwrócenia się do metod REST ze stosowaniem ograniczeń logiki biznesowej;
- c) serwis, zajmujący się jedynie przedstawieniem danych na stronie.

Nadal serwis a) został rozdzielony na dwa oddzielne serwisy z dostarczenia danych w skutku wydzielenia dwóch poddziedzin z ogólnej dziedziny projektowej. W ten sposób powstał interfejs danych o użytkownikach, oraz interfejs samego materiału encyklopedycznego, który tworzą informacje o filmach, osobach i wytwórniach. Aby zniwelować nacisk na jedną bazę, oraz możliwe zakłócenia przy poleceniach HTTP, została użyta reguła przeznaczenia każdemu serwisowi oddzielnych baz danych.

Ważnym momentem, zauważanym jeszcze w opisie każdego z pojęć dziedziny, jest

obecność pola „Picture”, które poznać jedynie nazwę zdjęcia. Aczkolwiek obiekty samych zdjęć w postaci dużych skupień danych binarnych mogą być przechowywane w bazach danych, jednak takie rozwiązanie jest zbyt nieoptymalne pod względem ilości czasu oraz pamięci operatywnej, które będą potrzebne do wykonania zapytań do bazy [source?]. Jednocześnie optymalnym a prostym wariantem jest przechowywanie tego typu danych na dysku twardym sprzętu hostującego serwer z organizacją hierarchii folderów ze zdjęciami. Alternatywą jest użycie serwisów dodatkowych, przedstawiających usługę przechowywania danych w chmurze. Przewagą takiego rozwiązania jest w już wbudowanych narzędziach, powiązanych ze sprawdzaniem całościowości oraz bezpieczeństwa ładowanych plików oraz w tym, że serwis staje mniej zależny od swojego hosta. ... Przyjmując do uwagi powody, przedstawione w tym akapicie, na potrzebę zarządzania jedynie tego typu danych z serwisu a) powstaje dodatkowy serwis.

Podobnie do poprzednich manipulacji z serwisem a), serwis c) rozdzielił się na dwie jednostki prezentujące dane z wyodrębnionych poddziedzin. Jedna z nich zwraca za poleceniem strony encyklopedyczne, inna zwraca strony powiązane z profilem pewnego użytkownika.

Serwis b), w odróżnieniu od poprzednich, nie został podzielony dla dostosowywania się do każdej pary jednostek jednak zamiast tego reprezentuje ogólny interfejs unifikowany do każdej kategorii danych. W nim że egzekwowane zostają prawa logiki poprzez sprawdzanie dostępu klienta do pewnego z zasobów zapytanych. Dane podejście z zapewnieniem pojedynczego punktu wejścia klienta zorientowanego na usługi z dostarczenia zasobów nazywa się wzorem bramy interfejsu oprogramowania (ang. „Gateway API”) [3, s. 68]. Brama przekazuje żądania do podstawowych usług i przekształca ich odpowiedzi a jednocześnie obsługuje połączone problemy klienta, takie jak uwierzytelnianie i podpisywanie żądań.

Dla danego projektu, podsumowując powody i decyzje wyłożone powyżej, przemyślano architekturę złożoną z następujących serwisów:

- “Encyclopedia Service” - główny serwis, częśćka, która rządzi główną stroną, widoczną dla wszystkich użytkowników, czyli wykazuje bezpośrednio informacje o kinie.
- “Cinema Data Service” - serwis, obsługujący zapis/szczytywanie danych z bazy danych o filmach;
- “Access Service” - serwis, zajmujący się uwierzytelnianiem użytkowników, czyli ich logowaniem, a także autoryzacją, czyli wydaje dostęp do zasobów.

- “Gateway API” - prosty serwis, który zawiera jedynie ubezpieczone polecenia do takich zasobów, wymagających praw dostępu po dokonaniu autoryzacji przez “Access service”. Jest także jednostką, odgrywającą rolę “proxy”.
- “Profile service” - serwis, który obsługuje zarządzanie kontem użytkownika w wyglądzie wydania odpowiednich widoków oraz możliwości do zapisu informacji.
- “User Data Service” - serwis, który obsługuje zapis/szczytywanie informacji o użytkownikach do/z odpowiedniej bazy danych. Baza danych, zarządzana przez niego, zawiera jak informacje autoryzacyjne, tak i ogólne dane profilu.
- „Image Service” – serwis, służący do obrabiania i przechowywania danych multimedialnych w postaci zdjęć.

## **Implementacja**

Zaimplementowane serwisy o wspólnym przeznaczeniu dzielą podobieństwa w swojej organizacji wewnętrznej, które są zaznaczane na początku każdego z podrozdziałów niżej. Implementację samego rozwiązania rozpoczęto z założenia jednostek z dostarczenia danych, które z jednej strony implementują standardowe metody do pobrania danych z bazy, a z innej strony same prezentują interfejs programowy, na których serwisy bazują zapytania do potrzebujących kategorii danych.

### **Organizacja serwisów z dostarczenia danych strukturyzowanych**

Serwisy „Cinema Data” oraz „User Data” służą jako warstwy komunikacji pomiędzy bazą danych oraz wewnętrznymi częściami aplikacji. Ich architektura składa się z podobnych decyzji bazujących się na wspomnianych technikach projektowych: podejściu DDD oraz wzorze projektowym MVC. Zgodnie z tymi podejściami wyznaczyło się trzy następujące składowe serwisy:

- a) „Domain” – warstwa definicji dziedziny projektowej;
- b) „Infrastructure” – warstwa pobrania i transferowania danych z bazy;
- c) „API” – warstwa definicji interfejsu z realizacją punktów końcowych.

W warstwie „Domain” pojęcia dziedziny zostają przenoszone w wygląd programowy. To oznacza, że ta warstwa zawiera wyznaczenia obiektów w wyglądzie klas, z wykorzystaniem paradygmatu programowania obiektowego. Odpowiednie klasy także zostają połączone pod jedną przestrzenią nazw dla wprowadzenia dodatkowej separacji pomiędzy nimi.

Warto zaznaczyć, że nazwa każdej z podprzestrzeni nazw pobiera, lub zawiera imię odpowiedniego folderu, w który jest umieszczony plik z wyznaczeniem klasy, lub innej struktury. W taki sposób, hierarchia przestrzeni nazw dziedziczy hierarchię ścieżkową projektu, co zapewnia dodatkową informację pro miejsce znajdowania obiektów przestrzeni przy dodaniu posyłania na inne przestrzeni nazw na początku pliku.

Warstwa „Domain” umieszcza tylko jedną przestrzeń „Aggregates” ze wszystkimi modelami agregatowymi, każdy z który wyznacza swoją własną podprzestrzeń. Wśród tych przestrzeni wydziela się „Base”, która mieści klasy rdzeniowe dla encji (obiektów głównych) – „Entity”, oraz obiektów wartościowych (pobocznych) – „Value”. Te klasy wyznaczają wspólne pola innych klas, które je dziedziczą. Tym samym także dodaje się poziom abstrakcji dla obiektów głównych i pobocznych. Wspólnymi dla głównych klas występują pola identyfikatora, imienia, nazwy zdjęcia, opis, a także dodatkowe służbowe pola: „IsDeleted” – oznacza obiekt jako „usunięty” oraz niedostępny do wysłania na zapytanie; „CreatedAt” – zawiera informację pro czas dodania obiektu, służy w jakości dodatkowego pola do sortowania obiektów. Poboczne klasy dziedziczą dopiero identyfikator (w tym na obiekt główny), imię oraz nazwę zdjęcia.

Każda z encji głównych organizuje się we własnej przestrzeni, gdy poboczne obiekty albo stanowią część przestrzeni pewnego modelu agregatowego, albo umieszczają się w przestrzeni wspólnej „Shared” w przypadku gdy dane pojęcie wiąże się z kilkoma encjami jednocześnie.

Do uwagi odnośnie układania serwisów z regułami DDD przyjmuje się niezależność warstwy dziedziny projektowej od innych składowych, które są opisane niżej.

...

... dalej: opis warstwy Infrastructure o przestrzeniach nazw Context (ze wspomnieniem używanego driveru bazy danych MongoDB), Repositories, Utils, z opisem wzoru projektowego Repository, oraz opis przestrzeni Models z opisem pojęcia DTO; opis warstwy Api o przestrzeniach nazw Commands, Queries (z opisem podejścia CQRS, użytego narzędzia MediatR), Mappings (opis narzędzia AutoMapper), Exceptions oraz Controllers (wspomnienie narzędzi ASP frameworku do organizacji mechanizmu routowania; w tym można skupić się na implementacjach endpointów, unikatowych dla każdego z serwisów)

...

### **Organizacja serwisu z dostarczenia danych multimedialnych**

...

... dalej: opis warstw, obecnych w serwisie Image Service: Infrastructure o przestrzeni nazw Models i Repositories, Api o przestrzeni nazw Requests, Handlers, Controllers, inne; skoro większość struktury dzieli podobieństwo z poprzednimi serwisami, opisanymi będą głównie różnice; opis przyczyn z pomijania implementacji niektórych części

...

### **Organizacja serwisu z przekazywania danych według logiki biznesowej**

...

... dalej: opis Gateway API Service z warstwą Infrastructure o przestrzeni nazw Services z implementacją klas i metodami, w których formują się zapytania do każdego z API z odpowiednim obrabianiem wyników tych zapytań (czyli odpowiedzi od innych serwisów); opis implementacji wzoru projektowego API Gateway, w czym się przejawia; opis warstwy Api, szczególnie o przestrzeni Controllers z opisem endpointów i szczególnym opisem kroków złożonych działań, logiki routowania, odpowiedzi na zapytania serwisów.

...

### **Organizacja serwisów z prezentacji widoków**

...

... dalej: opis warstw, obecnych w serwisach demonstracyjnych Encyclopedia Service, Profile Service, Access Service: Infrastructure o przestrzeni Services (opis klasy-narzędzia HttpClient); Api o przestrzeniach Models, Views, Extensions, opis wzoru projektowego MVVM (Model, View, View Model); ogólny opis narzędzia Razor pages, opis sposobu jego stosowania przy implementacji widoków ze wspomnieniem używanej css biblioteki; szczególny opis widoków (pewnie bez demonstracji całości, zamiast zrobić opis komponentów oraz powiązanych z nimi wydarzeń), opis różnicy pomiędzy widokami serwisów Encyclopedia Service, Profile Service, Access Service, w tym różnicy przy różnych rolach użytkownika;

Pewnie, potrzebuje rozdzielenia na podrozdziały, np. **Struktura serwisów** oraz **Komponenty stron i ich logika**

...



## Wyniki

...

## **Wnioski**

...

## Bibliografia

- [1] E. Evans *Domain Driven Design – Tackling Complexity in the Heart of Software*, wyd. 1. USA, MA, Westford: Addison-Wesley Professional, 2004.
- [2] R. Elmasri, Sh. B. Navathe, *Fundamentals of Database Systems*, wyd. 7. USA, NJ: Pearson Education Ltd, 2016.
- [3] M. Bruce, P.A. Pereira, *Microservices in Action*, USA, NY, Shelter Island, Manning Publications co., 2019.
- [4] C. Richardson, *Microservices Patterns With examples in Java*, USA, NY, Shelter Island, Manning Publications co., 2019.
- [5] I. Nadareishvili, R. Mitra, M. McLarty, M. Amundsen, *Microservice Architecture: Aligning Principles, Practices and Culture*, wyd. 1. Gravenstein Highway North, Sebastopol, CA: O'Reilly Media Inc., 2016
- [6]