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 9

Implementacja 11

Organizacja serwisów z dostarczenia danych strukturyzowanych 11

Organizacja serwisu z dostarczenia danych multimedialnych 16

Organizacja serwisu z przekazywania danych według logiki biznesowej 18

Organizacja serwisów z prezentacji widoków 20

Wyniki 24

Wnioski 25

Bibliografia 26

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 roli – zwykłego użytkownika oraz administratora.

W trybie anonimowym użytkownikowi dostępne są następne 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 kastomizacji 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 administratora”, 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 informacje, zasługuje na osobną reprezentacje przez oddzielną encje; 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 roli 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 poznaczono film.
AddedAt	DateTime	Czas etykietowania filmu.
Label		
Value	Integer	Znaczenie przeznaczone etykiecie.
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 obiektu „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	RatingScore	Ocena 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.
RatingScore		
N	Integer	Ogólna ilość ocen od użytkowników.
Score	Double	Średnia z ocen użytkowników.
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	Priorytet roli.
RolePriority		
Value	Integer	Znaczenie przypisane priorytetowi.
Name	String	Nazwa poziomu priorytetu.

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 poznaczenia 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 uczestniła.
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 wytórni.
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 mikroserwisowej 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 sługuje 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 każdych z pojęć dziedziny, jest obecność pola „Picture", które poznacza 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 **[ref]**. 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 także zostają egzekwowane 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 następnych serwisów:

- "Encyclopedia Service" - główny serwis, cząstka, która rządzi główną stroną, widoczną dla wszystkich użytkowników, czyli wykazuje bezpośrednie informacje o kinie.

- “Cinema Data Service” - serwis, obsługujący zapis/sczytywanie 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/sczytywanie 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, sługujący 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.

Używanym systemem do przechowywania danych jest MongoDB. Serwer bazy danych jest wspólny pomiędzy serwisami, na każdy serwis przychodzi się po jednej bazie danych do przechowywania obiektów dziedziny, a same obiekty pewnego pojęcia w tych bazach łączą się w kolekcje. Pojedynczy obiekt dziedziny przechowujący się w kolekcji nazywany jest dokumentem.

Organizacja serwisów z dostarczenia danych strukturyzowanych

Serwisy „Cinema Data” oraz „User Data” sługują jako warstwy komunikacji pomiędzy bazą danych oraz wewnętrznymi częściami aplikacji. Ich architektura została złożona przez podobne decyzje projektowe bazujące się na wspomnianych technikach projektowych: podejściu

DDD oraz wzorze projektowym MVC. Zgodnie z tymi podejściami wyznaczyło się trzy następne składowe serwisu:

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. Wydziela się kilka przestrzeni nazw pod którymi zostają połączone odpowiednie grupy klas dla wprowadzenia dodatkowej separacji pomiędzy nimi.

Warto zaznaczyć, że nazwa każdej z podprzestrzeni nazw 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”. Tę klasy wyznaczają wspólne pola dla dziedziczących je innych klas. 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 użykownika; „CreatedAt” – zawiera informację pro czas dodania obiektu, służuje 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.

W pozostałym klasy odzwierciedlają struktury pojęć, opisanych w tabelach części *Organizacja bazy danych*.

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

Drugą warstwą jest warstwa „Infrastructure”, która opisuje trzy przestrzeni: kontekstu bazy danych „Context”, repozytoriów danych „Repositories” oraz modeli danych do transferowania „Models”.

Przestrzeń kontekstu zawiera jedną klasę, która wyznacza konkretnie jaki system zarządzania bazą danych jest używany do przechowywania danych. Ta klasa wprowadza zależność od zewnętrznego pakietu „MongoDB Driver”, który ze swojej strony zawiera narzędzia do zarządzania serwerem bazy danych MongoDB. Takimi narzędziami występują klasy, obiekty których reprezentują bazy danych oraz zawierające się w nich kolekcje. Kolekcje wytwarza się na każdą z pojęć dziedziny oraz niektóre rekordy, opisujące relacje „Wiele-do-Wielu”. Kontekst klasy, w taki sposób, zawiera jeden obiekt bazy danych zewnętrznej klasy MongoDatabase, oraz kilka obiektów reprezentujących kolekcje tej bazy klasy zewnętrznej MongoCollection. Obiekty kolekcji tworzy się za pośrednictwem obiektu bazy danych.

Następną przestrzenią logiczną jest przestrzeń repozytoriów. Pod „repozytorium” w tym kontekście oznacza się obiekt, implementujący wzór projektowy „Repository”. Sens tego wzoru polega na oddzieleniu standardowych metod operujących się bezpośrednio nad obiektami bazy danych od logiki zewnętrznej z wyznaczeniem dodatkowych metod, właściwych dopiero dla konkretnego pojęcia. Standardowymi, w tym także i wspólnymi, metodami wszystkich klas są CRUD („Create-Read-Update-Delete”), metody tworzenia, odczytu po identyfikatorze, aktualizacji oraz usunięcia obiektu encji pewnej kolekcji. Dane metody tworzą w serwisie wspólny interfejs, nazywany IEntityRepository<T>, który dziedziczy abstrakcyjna klasa EntityRepository<T>. Ten zapis w „<>” poznacza klasę jako generyczną, czyli zaimplementowaną bez specyfikacji konkretnej klasy – litera „T” przy nazwie interfejsu uogólnia obiekt encji, którym operuje się odnośnie bazy. W ten sposób metody o podobnych implementacji według funkcjonalności nie duplikuje się dla każdej z encji. Jest to także sposób użycia właściwości polimorfizmu.

W klasie abstrakcyjnej EntityRepository implementują się wszystkie metody CRUD, oprócz metody aktualizacji obiektu, skoro wymaga ona indywidualnego podejścia do każdej z encji. Dodatkowo abstraktyzuje się metoda znalezienia mnóstwa obiektów zgodnie z przedstawionymi warunkami w jakości warunku do filtracji, warunku sortowania, ilość obiektów

do przeskoczenia przez metodę pobrania, ograniczenia ilości obiektów do pobrania. Abstraktyzuje się także wariant dla pobrania pojedynczego obiektu, który przyjmuje warunek filtracji. Od tych dwóch metod dziedziczą wszystkie inne specyficzne metody do odczytu.

Do każdego z repozytorium encji także przedstawia się własny interfejs metod, dziedziczący główny interfejs IEntityRepository. Odpowiednie klasy repozytoriów encji oprócz EntityRepository dziedziczą także i tę specyficzne interfejsy. Odróżnienia pomiędzy nimi przejawiają się głównie w metodach odczytu danych, filtracja w których odbywa się odnośnie pól, unikatowych dla klas obiektów.

Przy odczytywaniu dane z bazy automatycznie zostają przetransformowane na obiekty odpowiednich klas programowych należących przestrzeni „Domain", co także odbywa się i w odwrotny sposób przy zapisywaniu informacji do bazy.

Ostatnią część podziału „Infrastructure" formują modele obiektów do transferowania (dalej DTO, „Data Transfer Object"). Umieszcza ona definicje klas, obiekty których są wysyłane do innych serwisów w jakości odpowiedzi na ich zapytania do danych. W większości powtarzają one pola klas, opisujących obiekty dziedziny, z transformacją typów tych pól na bardziej proste, lub skróceniem ilości tych pól. Skrócenie odbywa się w celu dostosowywania do specyficznego zapytania, dla którego odpowiedź w wyglądzie informacji po wszystkim znaczeniom pól może być zbytnia. Rozdziela się modele według tego, czy idzie zapytanie na pojedynczy obiekt, lub mnóstwo obiektów, co sprawia, że inne serwisy potrafią otrzymywać dopiero potrzebujące im dane, robiąc zapytanie na obiekty pewnego modelu. Z tego powodu ilość modeli potrafi właściwie przewyższać kilkakrotnie ilość obiektów dziedziny.

Głównym odróżnieniem warstwy klas modeli DTO od wszelkich innych warstw serwisów jest to, że tę klasy zostają wyniesione do oddzielnej przestrzeni poza serwisami, do której zwracają się serwisy, zapytujące po dane. Skoro obiekty tych klas uczestniczą w wymianie danych pomiędzy serwisami, same serwisy mają posiadać informację o strukturze tych obiektów. Wyniesienie obiektów w oddzielną przestrzeń pozwala uniknąć zależności międzyserwisowej, gdyż proste wykorzystanie obiektów DTO nie zyskuje takiej zależności. Wszystkie dostosowywania modeli odbywają się na potrzebę innych jednostek, lecz nie tej, prezentującej API do zapytań.

Ostatnią warstwą serwisów „Cinema Data" oraz „User Data" jest „Api", która łączy wszystko powiązane z zewnętrzną logiką dostępu do danych. Dla formalizacji i strukturyzacji

pojęcia „zapytanie” używa się wzorca projektowego CQRS („Command Query Responsibility Segregation”), który cechuje się podziałem ogólnej ilości zapytań na komendy powiązane ze zmianą stanu obiektu, oraz zapytania z prostym odczytem obiektu. Sens tego wzoru polega na optymizacji i dostosowywaniu się do potrzeb różnych modeli zapytań, jako tych powiązanych z wydajnością, skalowalnością, lub bezpieczeństwem **[ref]**. Według tego klasy typu Request zostają transformowane do odpowiednich obiektów CQRS zanim odbywa się użycie metod repozytoriów dla dostępu do danych. Klasy zdefiniowane każdego z podziałów umieszczają się w odpowiednich przestrzeniach nazw Commands oraz Queries, które dzielą się na podprzestrzeni według celowych encji, a w przestrzeni Commands jeszcze odbywa się dodatkowe rozdzielenie według typu działania celowego („Create”, „Update”, lub „Delete”).

Najbardziej ciekawymi jednostkami tych serwisów są jednostki bezpośrednio otrzymujące zapytania i wysyłające odpowiedzi na nich. Same tę obiekty przedstawiają interfejs programowy „na zewnątrz” do innych serwisów, z którego one korzystają. Dla ich implementacji użyto narzędzia frameworku ASP, który przedstawia narzędzia do implementacji reguł REST API z wykorzystaniem protokołu wymianą danych HTTP. Jednostka wymieniająca się danymi jest reprezentowana przez klasę Controller, który zostaje dziedziczony dla tworzenia API dla różnych typów obiektów dziedziny. Przedstawia on kierowanie metodami Get, Post, Put, Delete, odpowiednio dla każdej z podstawowych zmiennych HTTP.

Warto zaznaczyć, że transferowanie danych z przedstawieniem API zostaje jedyną funkcją klas dziedziczących Controller. Logika przedstawienia ograniczeń na dane, oraz śledzenie za ich spójnością odbywa się w metodach, nazywanych „handlerami”, które są wyniesione poza nimi. Dla przeniesienia danych między tymi częściami używa się klasy, implementującej wzorzec projektowy „Mediator”, który polega na tworzeniu takiej centralizowanej jednostki, którym celem jest przesyłanie danych pomiędzy kilkoma niezależnymi warstwami oprogramowania. W tym celu skorzystano z pakietu pod nazwą „MediatR”, który dodatkowo zawiera narzędzia do napisania handlerów przetwarzających zapytania. Obiektem do wysyłania służy pewny obiekt CQRS, gdy możliwym obiektem zwróconym po procedurze jest DTO, którego od razu wysyła się z obiektu Controller w jakości odpowiedzi z wydaniem statusu sukcesu HTTP zapytania.

Każdy z handlerów jest przedstawiony przez swoją klasę Handler, dziedziczący interfejs IRequestHandler pakietu „MediatR”. Te klasy używają obiektów, dziedziczących interfejsy repozytoriów dla pobierania i dostarczenia danych. W ciągu wykonania metody sprawdzają

pewne warunki, na podstawie których metoda może wydawać wyjątki z informacjami o wynikniętym problemie podczas przetwarzania danych. Przy sukcesie operacji metoda zwraca sformowany obiekt DTO.

Zanim obiekty DTO zostają zwrócone w jakości odpowiedzi, one muszą być sformowane na podstawie obiektów, wydanych przez repozytorium po działaniu nad bazą, który ze swojej strony operuje wyłącznie na obiektach dziedziny projektowej. Z tego powodu obiekty komend CQRS muszą także zostać przetransformowane do odpowiednich obiektów dziedziny. Aby scentralizować konwertowanie obiektów do innych typów, także sięga się po wzorcu „Mediator", implementację którego przedstawia pakiet „AutoMapper". Opis szczegółów konwertowania jest wyniesiony do przestrzeni Mappings.

W taki sposób, dzięki powyższym podejściom projektowym, celowe serwisy z dostarczenia danych „Cinema Data Service" oraz „User Data Service" przyjmują strukturyzowany wygląd. Wydzielenie logicznych części pozwala na większą kontrolę procesu implementacji kluczowych warstw, które zapewniają dane dziedziny projektowej i przebieg operacji nad nimi przewidywane logiką biznesową.

Organizacja serwisu z dostarczenia danych multimedialnych

Zadaniem serwisu „Image Service" jest zapewnienie zachowania oraz ładowania danych w postaci zdjęć. Serwis do przechowywania danych tego typu przedstawia platforma Azure z usługą Blob Storage z możliwością ładowania danych z i do chmury. Do zarządzania metodami kontenerów chmurowych użyto odpowiedniego API, przedstawionego przez pakiet „Azure.Storage.Blobs".

Serwis bazuje się na podejściach, rozpisanych dla serwisów „Cinema Data Service" oraz „User Data Service", ale w znacznej mierze ich redukuje. Wywodzi się taka decyzja z tego, że przedstawiane dane binarne nie są strukturyzowane, jednocześnie nie potrzebuje się przechowywania żadnych innych informacji o tych obiektach, oprócz identyfikatorów (czyli, nazw zdjęć), oraz grup bajtów, które je tworzą. To pozbawia konieczności dodawania warstwy dziedziny „Domain", oraz implementacji wzoru CQRS – dla tego serwisu wszystkie zapytania tworzą wspólną przestrzeń „Requests".

Jedyny typ modeli przedstawiony w serwisie jest model DTO (transferowania danych), znajdujący się w warstwie „Infrastructure" z modelami zapytań na dodawanie, zamianę, usunięcie

oraz pobranie zdjęć. Oprócz tego w warstwie znajduje się implementacja repozytorium jako warstwa abstrakcji nad obiektem klasy BlobContainerClient do zarządzania zdjęciami w chmurze.

Modele zapytań obiektów dostarczają dane o zdjęciach w postaci wierszy – przenoszą one albo nazwę zdjęcia, albo sam obiekt zdjęcia, lub i to, i to razem. Przed wysłaniem do danego serwisu zdjęcie znajduję się w postaci obiektu klasy Stream, reprezentującą potok ładowanego pliku. Jednak przy tworzeniu modeli ten obiekt musi zostać konwertowany do postaci, bardziej nadającej się do transferowania – w tym celu wybrano stosować format Base64. Przekształtowanie do i z wierszu o tej postaci odbywa się przez klasę Convert ze standardowej przestrzeni języka C#. W ten sposób, serwis otrzymuje zdjęcia w postaci Base64, konwertuje na zwykły format bajtowy i przeprowadza do zachowania w chmurę.

Aby lepiej dostosować się do potrzeb serwisów z prezentacji danych, które mogą potrzebować zdjęcia różnej skali i rozmiarów, serwis tworzy i przechowuje kilka kopii jednocześnie dla jednego zdjęcia. Dla obsługi możliwości precyzowania potrzebujących rozmiarów dodaje się typ przeliczenia ImageSize ze znaczeniami „Tiny”, „Small”, „Medium”, „Big”, „Large”. Przeliczone znaczenia nie są niczym innym, jak flagami, czyli wskazane poznaczenia mogą być kombinowane między sobą. Odpowiednie pole reprezentujące rozmiar zdjęcia dodaje się dla wszystkich modeli zapytań. Z tego powodu typ flag dzieli z nimi jedną przestrzeń. Dla uproszczenia wszystkie kopii jednego zdjęcia dzielą jedno imię, ale mają różną lokalizację w kontenerze Azure Blob Storage, gdzie jest możliwość tworzenia struktury złożonej z folderów i podfolderów. Odpowiednio na każdy typ zdjęcia wydziela się jeden folder.

Do tworzenia kopii zdjęcia o innym rozmiarze stosuje się skalowania. Narzędzia do skalowania przedstawia pakiet „Magick.NET”, z którego użyto metodę skalowania interpolacyjnego z użyciem algorytmu interpolacji dwuliniowej, który różni się swoją szybkością.

Konkretne znaczenia parametrów, według których zdjęcia podpadają w jedną z kategorii typu przeliczonego ImageSize nie są zadane wprost w kodzie, jednak są wyniesione do pliku „appsettings.json”, w którym, do tego, zadaje się parametr maksymalnego rozmiaru pliku oraz ścieżki do każdego typu zdjęć w kontenerze chmurowym. Parametrem, decydującym o rozmiarze jest górna granica rozmiaru mniejszej strony. Przy ładowaniu serwisu informacja wyciąga się z pliku i jest przechowywana w obiekcie programowym w postaci słownika.

W tym serwisie obiekt typu Controller także jedynie otrzymuje zapytania i wysyła odpowiedzi. Logika przekształcenia zdjęć w tym serwisie także zostaje skierowana do handlerów,

w których odbywa się sprawdzenie obecności pewnych flag, skalowanie, formowanie ścieżek do zdjęć, wywołanie metod repozytorium do zachowania danych po tym ścieżkom w kontenerze oraz sprawdzenie wyników ze zgłoszeniem problemów w postaci wyjątków. Przy sukcesie handler tworzy obiekt odpowiedzi, który składa się z imienia zdjęcia, przeliczenia obrobionych rozmiarów, oraz słownika w postaci klucz-znaczenie, gdzie kluczem występuje rozmiar typu ImageSize, a znaczeniem – odpowiadający jemu wiersz z linkiem do zdjęcia pewnego rozmiaru. Czyli same zdjęcia nie są wysyłane w jakości odpowiedzi, ale sformowanych linków jest wystarczająco dla dalszej demonstracji zdjęć na stronach serwisów które organizują widoki stron.

Organizacja serwisu z przekazywania danych według logiki biznesowej

Następną jednostką jest serwis „Gateway API", który zabezpiecza wymianę danych nawzajem pomiędzy serwisami. Posiada ten serwis bardziej prostą strukturą w porównaniu do innych serwisów, lecz ma złożoną implementację metod same z wymiany danymi.

Serwis posiada dwie warstwy – warstwę „Infrastructure" oraz „Api". Do „Infrastructure" wchodzi jedna przestrzeń „Services", która kształci modele klientów HTTP. Celem klientów jest wysyłanie zapytań wobec danych dziedziny z otrzymywaniem i transformacją odpowiedzi do formatu celowego. Faktycznie klasy z klientem HTTP reprezentują modele samych serwisów, kopiując strukturę ich API. Implementacje tych klas w „Gateway Service" używają obiekty klasy HttpClient frameworku ASP. Zapytania oraz odpowiedzi HTTP formują się ze wszystkich klas DTO, umieszczonych w przestrzeni Shared. W ten sposób formują się klasy opakowujące klient.

Wspomniane klasy formują się dopiero dla serwisów, które wysyłają danę i przyjmują zapytania. Czyli do serwisów, przedstawiających widoki, one nie są potrzebne. W taki sposób, implementacje HTTP klientów powstają dla serwisów „Cinema Data Service", „User Data Service" oraz „Image Service". Dla uproszczenia podjęto decyzję o implementacji klientów dla każdej z API ogólnych encji – w ten sposób klient serwisu „Cinema Data" rozbija się na „Cinema HTTP Client", „Person HTTP Client" oraz „Studio HTTP Client". W ten sposób także dodaje się poziom abstrakcji – nie specyfikuje się w jawny sposób, jaki serwis odpowiada za dane pewnej encji projektowej.

W odpowiedni sposób w warstwie „Api" zorganizowano klasy-kontrolery zarządzające przesyłaniem danych, w tym sensie, że napisane one pod każdą encję projektową. W danej części formują się własne dla serwisu API encji „Cinema", „Person", „Studio" oraz „User".

Sens serwisu „GatewayAPI” polega na implementacji logiki dziedziny projektowej przy wymianie danymi, dlatego prawa logiki zdecydowano dopisywać od razu w kontrolerach, w których odbywa się otrzymywanie i wysyłanie danych. Dlatego implementacja oddzielnych funkcji-handlerów jest opuszczona. W ten sposób przestrzeń klas-kontrolerów jest jedyną przestrzenią warstwy „Api” tego serwisu.

Jeżeli serwisy z dostarczenia danych operują na obiektach dopiero jednego pojęcia projektowego, serwis „Gateway” zarządza procedurami, dotyczącymi kilka pojęć jednocześnie. Łączenie odbywa się na poziomie jak zwykłych zapytań na dane, tak i zapytań, skutkiem których jest zmiana stanu obiektu – w pierwszym przypadku obiekt z transferowania danych jest układany z danych pobranych z różnych odpowiedzi, kiedy w drugim przypadku celowa zmiana dopiero jednego obiektu może powodować poboczne działania nad kilkoma obiektami pochodzącymi z różnych dziedzin.

Podstawowym działaniem na poziomie procedury „GET” jest robienie zapytania dodatkowego w celu zwrócenia ścieżki do zdjęcia po sprawdzeniu, że odpowiednie pole z nazwy zdjęcia nie jest puste. Innym jest otrzymanie niedostających informacji, które w pewnym obiekcie przedstawiają się dopiero jako identyfikatory. Wtedy robi się zapytanie za tym identyfikatorem na informację powiązanego obiektu. Takie podejście obowiązuje przy formowaniu i zwróceniu listy etykietowanych użytkownikiem filmów, gdy z początku idzie zapytanie na listę identyfikatorów filmów za identyfikatorem użytkownika, a dalej idzie zapytanie do API „Cinema” na informacje o wszystkich filmach za wydaną listą identyfikatorów, która i jest zwracana w jakości odpowiedzi.

Innym rodzajem bardziej złożonych działań są procedury z dodawania, aktualizacji stanu, lub usunięcia obiektów. Dla listy etykietowanych filmów lub ocen filmów użytkownika tę działania cechują się w większości sprawdzeniem obecności obiektów za wskazanymi identyfikatorami, gdy same działania ze zmiany informacji przedstawiają się dopiero jednym zapytaniem. W przypadku tych obiektów, które zawierają w sobie informację o powiązanych obiektach jako część własnych informacji, oprócz wspomnianych sprawdzeń wymaga się podejmowanie odpowiednich działań dla zapewnienia logicznej integralności danych. Taką złożoną procedurą występuje aktualizacja odpowiednich list – jeżeli obiekt dodaje posyłanie w liście na powiązany obiekt innego pojęcia, powiązany obiekt także musi dodać posyłanie na ten bieżący obiekt, do listy którego on został dodany. W podobny sposób musi przebiegać usunięcie powiązanych obiektów z listy, a w przypadku, gdy pewnego obiektu usuwa się kompletnie, musi

on także zostać usunięty z list wszystkich obiektów, które jego zawierały. Także odbywa się i aktualizacja, zmiany przy której muszą zostać odbijane i w listach, w której zaktualizowany obiekt się pojawia.

Przykładem mogą służyć działania powiązane z encją „Cinema". Przy aktualizacji obiektu danej encji, zapis o nim aktualizuje się i dla list encji „Person" oraz „Studio", gdzie rozpatrywany obiekt jest zawierany w listach „Filmography". Skoro te pojęcia są powiązane ze sobą nawzajem, wyszukiwanie dokładnych obiektów, listy filmografii których zawierają potoczny film, nie potrzebuje się, skoro odpowiednie identyfikatory takich obiektów już wchodzą do zapisów odpowiednich list „Starrings" oraz „ProductionStudios" samego filmu. Więc jest możliwe robienie aktualizacji od razu dokładnego wejścia w liście bez poprzedniego wyszukiwania. Przy dokonaniu zapytania na usunięcie filmu sprawdza się, czy film zawiera posyłania na zapisy innych encji, a w przypadku zawierania wejście „Filmography" usuwa się z listy odpowiednich encji.

W podobny sposób odbywają się przebiegi procedur nad encjami „Person" i „Studio" z odwróceniem ról obiektów celowych i zapisów: przy działaniach nad obiektami osób zmiany odbijają się na listach „Starrings", przy działaniach na wytwórniach zmiany można prześledzić na listach „ProductionStudios".

Dodatkowe aktualizacje obiektu „Cinema" przedstawiają się przy działaniach zmiany lub dodawania oceny użytkownika na film. Wtedy przed zaniesieniem zapisu o ocenie użytkownika z początku idzie próba aktualizacji tej oceny dla samego filmu.

W ten sposób danym serwisem zorganizowano warstwę logiki całej aplikacji.

Organizacja serwisów z prezentacji widoków

Dana część opisuje te serwisy, w których implementują się widoki i powiązana z nimi funkcjonalność, skierowana na obsługiwanie interaktywności tych widoków. Przedstawiona jest zarówno struktura jak samego serwisu, tak i widoków z opisem ich naznaczenia.

Dla formowania komponentów stron użyta statyczną CSS bibliotekę „bootstrap.css" wersji 5.1.3, lecz niektóre style w niej zostają nadpisane we własnym pliku CSS. Oprócz tego wyznaczają się i własne style prezentacji komponentów.

Wygląd widoków oczekuje się różnicować w zależności od pewnych warunków. Z tej przyczyny zdecydowano sięgać po mechanizmu generacji widoków dla generacji napełnienia stron w ciągu ich ładowania. W jakości mechanizmu generacji widoków obrano natywny dla

frameworku .NET mechanizm Razor Pages, który umożliwia renderowanie HTML znaczników z użyciem języka C# do kierowania logiką renderowania. Dany mechanizm jest pomocny w generacji dopiero statycznych, ale nie dynamicznych stron. Interaktywność stron z dynamiczną zmianą ich napełnienia wspiera się przez język JavaScript (lub ECMAScript), głównie przez odpowiednie biblioteki, napisane w tym języku. Do takich bibliotek wchodzą „bootstrap.js", „jquery.js", oraz inne rozszerzające funkcjonalność przeliczonych. Oprócz tego przedstawiają się pliki z własnymi skryptami, dostosowanymi do każdej z obecnych stron.

Struktura serwisów

Serwisy dzielą się na warstwy „Api" oraz „Infrastructure", jednak główne napełnienie serwisów jest zlokalizowane same w warstwie „Api".

Warstwa „Infrastructure" wyznacza dopiero jedną przestrzeń imion „Services" w której znajduje się wyznaczenie klienta do wymiany danych. Klient powstaje w celu reprezentacji serwisu „GatewayAPI", w podobny sposób, w który przykładowy klient jest wyznaczony w samym „GatewayAPI". Poprzez same ten klient odbywa się posyłanie zapytań na dane oraz polecenia na działania nad nimi wychodzące bezpośrednio ze strony użytkownika, podejmującego działania na stronach. Klient przedstawia uogólniony interfejs, bez podziału odnośnie encji poddziedzin. Przy tym polecenia interfejsów różnią się dla każdego z serwisu widoków.

Organizacja warstwy „Api" postępuje zgodnie ze wzorcem projektowym „MVVM" („Model, View, View Model"). Sens tego wzorcu jest podobny do wzorca „MVC" z tą różnicą, że jednostka modeli widoków („View Model") odpowiadająca za dostarczenie danych skupia się dopiero na logice poszczególnych stron, a nie na logice operowanych danych, przez co dzieli wspólną logiczną przestrzeń z widokami części „View" **[ref?]**. Odpowiednio do tego podejścia w warstwie powstaje dwie przestrzeni „Views" oraz „Models". Część „Views" wyznacza strony o rozszerzeniu „.cshtml" napisane z użyciem rozwiązania Razor oraz modeli widoków kierujące stronami i robiące zapytania do danych przez klient HTTP serwisu „GatewayAPI". W części „Models" znajdują się wyznaczone modeli do reprezentacji danych, dostosowane dla kategorii przedstawianych na stronach informacji. Po otrzymaniu odpowiedzi w wyglądzie obiektów modeli transferowania danych, ostatnie zostają przekonwertowane do odpowiednich obiektów modeli widoków, dopiero po czym używają się przy generacji stron.

Ostatnią niemałoważną część stanowią elementy folderu „wwwroot", w którym znajdują się biblioteki statyczne języka JS – główne z nich są „jquery.js", „jquery.validate.js" oraz

„bootstrap.js” i „bootstrap.min.js”. Dotąd wchodzi także i biblioteka stylów „bootstrap.css”, a także inne spersonalizowane pod strony pliki ze skryptami oraz stylami. Mieści ten folder także inne elementy statyczne – są to ikony oraz zdjęcia.

Komponenty stron i ich logika

Już wspomniane jest iż część widoków „Views” formują jednocześnie pliki widoków oraz ich modeli. Dla uproszczenia wyglądu i strukturyzacji plików ze znacznikami HTML niektóre strony rozbijają się na oddzielne komponenty – widoki częściowe, – które układają się w jedną stronę podczas renderowania. Same strony można rozbić na kategorie w zależności od przeznaczenia: strony z mnóstwem obiektów pewnej encji, strony z szczegółowym wyglądem obiektu oraz inne.

Oddzielnym odróżniającym się komponentem jest „Layout”, czym jest widok częściowy, który tworzy wygląd ramki nad wszystkimi stronami serwisu. Do danej ramki wchodzi górna nawigacja z nazwą serwisu w lewym rogu, przycisku „Search” i polem do wyszukiwania po środku oraz opcją „Account” w prawym rogu. Przy naciskaniu na ostatnią wykazuje się lista możliwych działań w koncie w zależności od trybu użytkownika („zalogowany”, lub „niezalogowany”) oraz jego roli w serwisie.

Strony z mnóstwem obiektów formują się osobiście dla każdej encji encyklopedii – dla filmów, osób oraz wytwórni. Widoki demonstrują obiekty podobnych typów na stronie w wyglądzie „siatki” kartek, sformowanej z kilku kolumn i wierszy. Każda kartka zawiera skróconą informację oraz pomniejszone zdjęcie. Kartki zawierają niejawne linki do przejścia do detalicznego widoku obiektu. Oprócz tego dane strony posiadają opcje do sortowania oraz filtracji, wyłożone po prawej stronie od widoku „siatki”. Na dole strony wykazują listę numeracji stron obiektów, czyli numer potocznej strony oraz numery stron sąsiadujących z nią.

Nawigacja do tych widoków może odbywać się przez pole do wyszukiwania, które nadaje możliwość znalezienia obiektów za imieniem. Przy polu „Search” znajduje się lista do wybrania celowego typu pojęcia do wyszukiwania („Cinema”, „Person”, lub „Studio”). Nadanie pustego znaczenia wyszukuje wszystkie obiekty wskazanego typu w dowolnej kolejności.

Szczegółowe widoki obiektów także są napisane pod każdą z encji projektowych. Mają one bardziej złożony wygląd przez możliwość wykonania poleceń nad danymi w przypadku, gdy użytkownik jest zalogowany i posiada na to odpowiednie prawa. Decyzja o nadaniu praw podejmuje się w modelach widoku. Jeżeli użytkownik nie jest zalogowany, lub jest zalogowany

jak zwykły użytkownik, przedstawia się jemu możliwość w większości dopiero do przejrzenia się danych obecnych na stronach. W przypadku, gdy użytkownik posiada rolę przywilejowaną, na odpowiednich stronach wykazują się opcję do aktualizacji lub usunięcia strony z obiektem, a w opcjach profilu z nawigacji w prawym rogu pojawia się opcja dodania nowej strony z wybraniem typu obiektu, dla którego strona jest tworzona. Przy tym opcję nie są po prostu zachowane za pomocą stylizacji w innym przypadku – zamiast nie są one renderowane dla mniej przywilejowanych użytkowników w ogóle. Wyjątkiem przywilejowania jest szczegółowy widok profilu osoby – najwięcej przywilej jest same przy posiadaczu konta tego profilu niezależnie od jego roli, gdy najbardziej przywilejowany typ użytkowników potrafi dopiero zmieniać rolę, lub usuwać konto użytkownika. Wyjątek dotyczy list etykietowanych użytkownikiem filmów – dodanie nowych zapisów, zmiana etykiet zapisów oraz usunięcie także przedstawia się dopiero do posiadacza tych list.

Strony szczegółowych widoków zbierają wokół siebie kilka widoków częściowych. Są to widoki z formami do dodania lub zmiany obiektu, oraz widoki z zapisami o obiektach powiązanych (przedstawionymi przez listy „Starrings”, „ProductionStudios” oraz „Filmography”). Niektóre z tych zapisów na szczególnych widokach obiektu głównego są układane w element „siatki”. Dla tych zapisów także są dodane widoki częściowe do ich zmiany lub dodania, przy czym dodanie odbywa się za pomocą wyszukiwania odpowiedniej encji z wybraniem opcji z ograniczonej listy z zaznaczeniem dodatkowych informacji o zapisie.

Operacja zgłaszania formy z odpowiednimi opcjami wywołuje wydarzenie ze skierowaniem informacji formy do pewnej metody modelu widoku, która wysyła zapytanie na przetwarzanie nadanych informacji za pomocą klienta „GatewayAPI”. Po tym metoda wysyła odpowiedź odwrotną z którą za pomocą skryptów języka JS opcjonalnie zmienia się stan potocznej strony bez jej kompletnego przeładowania.

…

Wyniki

…

Wnioski

…

Bibliografia

[1] E. Evans. 2004. *Domain Driven Design – Tackling Complexity in the Heart of Software*, wyd. 1. MA, Westford: Addison-Wesley Professional.

[2] R. Elmasri, Sh. B. Navathe. 2016. *Fundamentals of Database Systems*, wyd. 7. NJ: Pearson Education Ltd.

[3] M. Bruce, P.A. Pereira. 2019. *Microservices in Action*, NY, Shelter Island: Manning Publications co.

[4] C. Richardson. 2019. *Microservices Patterns With examples in Java*. NY, Shelter Island: Manning Publications co.

[5] I. Nadareishvili, R. Mitra. M. McLarty, M. Amundsen. 2016. *Microservice Architecture: Aligning Principles, Practices and Culture*, wyd. 1. Gravenstein Highway North, Sebastopol, CA: O'Reilly Media Inc.

[6]