

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування інституту, факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалавр

(ступінь вищої освіти)

на тему РОЗРОБКА МІКРОСЕРВІСНОГО ПРОЄКТУ ДЛЯ
ПЕРЕГЛЯДУ ФІЛЬМІВ ТА КЕРУВАННЯ ЇХ КОЛЕКЦІЯМИ
DEVELOPMENT OF MICROSERVICE PROJECT TO WATCH
MOVIES AND MANAGE THEIR COLLECTIONS

Виконав: студент(ка) 4 курсу, групи КНТ-119

Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

СПЕСИВЦЕВ О.К.

(ПРИЗВИЩЕ та ініціали)

Керівник КАПЛІЄНКО Т.І.

(ПРИЗВИЩЕ та ініціали)

Рецензент ШИТІКОВА О.В.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет ФКНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
« » 2023 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

СПЕСИВЦЕВА Олексія Костянтиновича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Розробка мікросервісного проєкту для перегляду фільмів та керування їх колекціями.

Development of Microservice Project to Watch Movies And Manage Their Collections

керівник проєкту (роботи) к.т.н., доцент, КАПЛІЄНКО Тетяна Ігорівна,
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від «04» квітня 2023 року № 87

2. Строк подання студентом проєкту (роботи) 07 червня 2022 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз проблеми та постановка завдань дослідження. 2. Матеріали і методи. 3. Опис програми. 4. Експлуатація, тестування та експериментальне дослідження програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	КАПЛІСНКО Т.І., доцент		
Нормоконтроль	КАМІНСЬКА Ж.К., асистент		

7. Дата видачі завдання « 06 » квітня 2023 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій розробки.	2 тиждень	Розділ 2
4	Розробка архітектури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тиждні	Розділ 3, 4
6	Тестування та експериментальне дослідження програмного забезпечення.	5 тиждень	Розділ 4
7	Оформлення пояснювальної записки та документів до неї.	6 тиждень	Додатки
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка)

(підпис) Олексій СПЕСИВЦЕВ
(Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

(підпис) Тетяна КАПЛІСНКО
(Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 189 с., 26 рис., 5 табл., 3 дод., 38 джерел.

МІКРОСЕРВІС, АРХІТЕКТУРА, САЙТ, СЕРВІС, ФІЛЬМ, КОЛЕКЦІЯ, РОЛЬ, ШАБЛОНІ ПРОГРАМУВАННЯ.

Об'єкт дослідження – програмні засоби для перегляду фільмів з використанням мікросервісів на Domain Driver Design (DDD).

Предмет дослідження – сайти (сервіси) з розвиненою системою безпеки для перегляду фільмів з подальшим менеджментом колекцій улюблених фільмів.

Мета роботи – створення програмної системи для зручного перегляду відео та швидкого менеджменту власних колекційних конфігурацій за умов універсальності і максимальної масштабованості системи.

Матеріали, методи та технічні засоби: об'єктно-орієнтоване програмування, чиста архітектура, DDD архітектурі, Firebase, PHP і Laravel, JavaScript і React, та HTML, CSS, MySQL.

Висновки. Розроблено мікросервісний проєкт для перегляду фільмів і створення улюблених колекцій, який максимально легко масштабувати, адаптувати, замінювати модульні частини під користувача з урахуванням особливостей місцезнаходження або вподобань клієнта.

Актуальність роботи пов'язана з відносно недавно випущеним гучним і привернувшем багато уваги сервісом Spotify для зручного прослуховування музики. Галузь використання – сервіси послуг з перегляду медіа файлів, а саме фільмів, відео, афіш, трейлерів. Окремо слід виділити можливість застосування такого сервісу професійними режисерами, операторами, працівниками телебачення, блогерами.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 189 pages, 26 figures, 5 tables, 3 appendixes, 38 sources.

MICROSERVICE, ARCHITECTURE, SITE, SERVICE, MOVIE, COLLECTION, ROLE, PROGRAMMING PATTERNS.

The object of the research is software tools for viewing movies using microservices on Domain Driver Design (DDD).

The subject of the research is websites (services) for watching movies with subsequent management of collections of favorite movies and sending emails.

The aim of the work is to create a software system for convenient video viewing and quick management of personal collection configurations under the conditions of universality and maximum scalability of the system.

Materials, methods, and technical means include object-oriented programming, clean architecture, DDD architecture, Firebase, PHP and Laravel, JavaScript and React, as well as HTML, CSS, MySQL.

The study concludes that a microservices project has been developed for viewing movies and creating favorite collections, which can be easily scaled, adapted, and modularized according to the user's location or preferences.

The relevance of the work is related to the relatively recently released and widely popular Spotify service for convenient music playback. The field of application is media file viewing services, specifically movies, videos, posters, and trailers. It is worth highlighting the possibility of using such a service by professional directors, cameramen, television workers, and bloggers.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК	8
ВСТУП.....	9
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ.....	10
1.1 Аналіз предметної області	10
1.2 Огляд існуючих систем, сервісів, сайтів, що надають змогу переглядати фільми і створювати власні колекції.....	13
1.2.1 Передмова	13
1.2.2 Сайт Rezka.ag.....	14
1.2.3 Сервіс Filmix.ac.....	16
1.2.4 Сервіс Sweet.tv	17
1.3 Висновки з розділу і постановка задачі	19
2 АНАЛІЗ ПРОГРАМНИХ ЗАСОБІВ.....	21
2.1 Огляд особливостей мов програмування	21
2.2 Огляд особливостей обраних платформ і бібліотек	26
2.3 Огляд особливостей обраних інтерпретаторів.....	27
2.4 Огляд архітектури DDD	29
2.4.1 Application рівень.....	29
2.4.2 Domain рівень	30
2.4.3 Infrastructure рівень.....	33
2.4.4 User Interface рівень.....	34
2.4.5 Компоненти присутні у декількох рівнях	36
2.4.6 Зв'язок одного MC з іншими	36
2.5 Огляд особливостей обраної СКБД.....	37
2.6 Висновки з розділу	38
3 ОСНОВНІ РІШЕННЯ З РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ	39
3.1 Основні рішення щодо розроблених MC і PCO	39
3.1.1 MC BestMovieClient.....	41
3.1.2 MC MovieMicroservice.....	45

3.1.3 MC EmailTemplateMicroservice	53
3.1.4 MC BestMovieStorage	53
3.1.5 PCO BestMovieCommon	54
3.2 Основні рішення щодо розробки інтерфейсу	57
3.2.1 Вибір додаткових бібліотек	58
3.2.2 Основні рішення щодо впорядкування інтерфейсу	59
3.2.3 Основні рішення з комунікації з користувачем	65
3.3 Основні рішення щодо розробки БД	66
3.3.1 Розроблена БД	67
3.4 Висновки з розділу	69
4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРЕМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ	71
4.1 Застосування системи	71
4.2 Розгортання і експлуатація системи мікросервісів	71
4.3 Тестування та дослідження сумісності проєкту	74
4.4 Висновки з розділу	81
ВИСНОВКИ	82
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ	84
Додаток А Текст програми	88
Додаток Б Конфігураційні файли Nginx	176
Додаток В Презентація	181

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

БД	– База Даних;
МС	– Мікросервіс;
ОС	– Операційна Система
СКБД	– Система Керування Бази Даних;
API	– Application Programming Interface;
AWS	– Amazon Web Services;
CMS	– Content Management System;
CQRS	– The command and Query Responsibility Segregation;
CRUD	– Create Read Update Delete;
DDD	– Domain Driver Design;
DDoS	– Distributed Denial of Service;
DOM	– Document Object Model;
DPO	– Domain Payload Object;
HTTP	– HyperText Transfer Protocol;
HTTPS	– HyperText Transfer Protocol Secure;
IDE	– Integrated Development Environment;
JS	– Java Script;
MVC	– Model View Controller
PCO	– Public Contract;
PHP	– Hypertext Preprocessor;
PHP-FPM	– PHP FastCGI Process Manager;
PSR-12	– PHP Standards Recommendations 12;
REST	– Representational State Transfer;
SEO	– Search Engine Optimization
UOW	– Unit Of Work.

ВСТУП

Завданням цієї роботи є створення мікросервісного проекту на архітектурі DDD для перегляду фільмів і керування їх колекціями, актуальність даної роботи полягає у відсутності аналогів у сфері перегляду фільмів. Окремо слід зазначити проблеми сучасного світу, такі як війна та глобальна пандемія, за яких люди все більше змушені значну частину доби проводити у ізольованих або захищених приміщеннях. У періоди ізольованості, люди зазвичай використовують сучасні пристрої для спілкування або обміну інформацією, але у зв'язку зі складністю розробки і захисту систем перегляду фільмів, у світі не існує систем, яка відповідала б усім сучасним показникам і задовольняла потреби користувачів.

Мета цієї роботи полягає у розробці інтуїтивно зрозумілої і автономної системи, яка буде надавати функції перегляду, створення, видалення і редагування фільмів і їх колекцій, що у разі потреби може замінити одним з важливих модулів системи, такі як сховище, сервіси авторизації.

Основна область використання такої системи - це звичайний користувач, який може вести облік своєї колекції фільмів, зберігати дані про кожен фільм (назва, рік випуску, режисер, актори, жанр, опис тощо), а також позначати фільми, які він переглянув або планує переглянути. Також, система керування фільмами може бути корисною для більших колекцій фільмів, які можуть зберігатися в бібліотеках, кінотеатрах, музеях кіно тощо. В таких випадках, система дозволяє легко знайти необхідний фільм, переглянути інформацію про нього та отримати доступ до нього в разі необхідності.

Отже, така система дозволить не тільки каталогізувати і надавати доступ до ресурсів, що містять фільм або відео, а ще й змогу кожній людині за власної логікою і бажанням відокремлювати від стандартизованих каталогів свої улюблені стрічки.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

У даному розділі розглянуті існуючі системи і сайти, що надають змогу переглядати, сортувати, редагувати інформацію про фільми та об'єднувати їх у колекції. Також у розділі буде визначена мета та задачі системи, що розроблюється.

1.1 Аналіз предметної області

Сучасні, великі, швидкі системи складаються з декількох сервісів, що надають послуги, які можуть бути побудовані на різних мовах програмування, так як кожна мова має свої переваги і недоліки у конкретних ситуаціях і сферах, тому для побудови такої системи доцільно використовувати декілька мов програмування.

Мета роботи створити сайт сервіс, що уникне типових помилок при розробці проекту і вдосконалисть існуючу функціональність створення колекцій та авторизації, а також загалом реалізує стандартні функції, такі як пагінація, фільтрація, додавання, редагування і видалення категорій, колекцій, фільмів, медіа файлів.

Найкращі сайти та сервіси, що побудовані на мікросервісній архітектурі складаються як мінімум з 4 сервісів, які взаємодіють між собою завдяки розроблену інтерфейсу спілкування. Визначимо перелік серверів і їх взаємодію:

- клієнт-сервер;
- головний сервер з архітектурою REST API (Representational State Transfer Application Programming Interface – це стиль архітектури програмного забезпечення вебслужб, що забезпечують доступ до даних та функцій інших програм або вебзастосункам [1]);
- поштовий сервіс;
- сховище даних медіа файлів.

Клієнт-сервер – це зручний інтуїтивно зрозумілий інтерфейс, що дозволяє керувати даними без поглиблення у зв'язки між даними у БД (БД - база даних). Завдяки набору функцій, клієнт серверна частина відсилає запити на один або декілька серверів з архітектурою REST API. Такі запити базуються на загальному протоколі HTTPS (HyperText Transfer Protocol Secure – це захищений протокол передачі даних через мережу Інтернет [2]) і розробленому методу обміну даних. Такі методи містять розроблену перевірку даних, відомості авторизації, специфічні хедери і структури даних, що притаманні саме розробленій системі. Реалізація такого інтерфейсу полягає у створенні PCS (PCS – public contract, публічний контракт, що реалізує інтерфейс взаємодії між декількома серверами).

PCS може бути специфічним пакетом даних або загальною частиною коду, що реалізує інтерфейс спілкування декількох серверів. Якщо реалізувати PCS у вигляді універсального пакету, який може бути завантажений улюбий час на будь-який носії, це призведе до покращення масштабованості і універсальності проекту. Слід зазначити, що розробка PCS – це довготривалий процес, який починається тільки після розробки двох або більше сервісів, що потрібно зв'язати.

Розглянемо головний сервер з архітектурою REST API. Такий сервер найнавантажениший сервіс з усіх, тому для його реалізації доцільно використовувати кеш, асинхронні процеси. Навантаженість цього сервісу полягає у тому, що він веде перевірку ролей і прав користувача, створює, видаляє, редагує дані про фільми, категорії, колекції, акторів, повідомлення і оцінки користувачів. Додатково до цього у сервісі роблять надбудови для поліпшення функцій і безпечності сховища, процесу авторизації.

У новітніх сервісах існують різні методи оптимізації навантажених сервісів, так як не усі мови програмування асинхронні, а кеш – це проста і не надійна система зберігання даних, тому що сама концепція кешу полягає у створенні ключ строки для доступу, який від способу створення буде мати колізії, ще і апаратно навантажувати сервер, що вплине на швидкодію

системи. Тому розробники приймають рішення на відокремлення частини сервісу на менші сервіси, такі як поштовий сервіс і сховище медіа файлів.

Переваги відокремлення функціональності відправлення листів на пошту полягає, по-перше, можна створити асинхронний процес черги, по-друге, знизити навантаження з одного апаратного сервісом шляхом розміщення розроблених логічних серверів на різних апаратних серверах. Остання перевага відокремлення функціональності відправлення листів на пошту притаманна і відокремленому сховищу, також перевагою такого сервісу сховища полягає у можливості використовувати незалежні системи зберігання даних, такі як AWS (Amazon Web Services – це хмарна платформа, що надає різноманітні сервіси та інфраструктуру для забезпечення обчислювальних, зберігання та інших потреб компаній та користувачів) і Firebase (Firebase – це платформа розробки мобільних та вебзастосунків, що належить компанії Google).

Таке дроблення на сервіси має власні методи і архітектурні підходи. Один з таких методів розвантаження сервісів описаний у книзі Еріка Евенса “Предметно-орієнтоване проєктування (DDD): структуризація складних програмних систем”.

DDD – це методологія до проєктування програмного забезпечення, який ставить доменну модель в центр розробки. Також цей підхід ставить на перший план бізнес логіку застосунку та його основні функції. Рівнева архітектура DDD складається з чотирьох рівнів – application, domain, Infrastructure та user interface. На рівні інтерфейсу розташовані компоненти, які взаємодіють з користувачами та зовнішніми системами. Рівень домену містить бізнес логіку, де визначаються сутності, агрегати та сервіси. Інфраструктурний рівень забезпечує збереження та обробку даних, а також комунікацію з зовнішніми системами. Взаємодія між рівнями відбувається за допомогою слабо зв'язаного архітектурного стилю, де кожен рівень може взаємодіяти з рівнями вище та нижче. Інтерфейс користувача це набір

інструментів, що надає користувачу змогу працювати з мікросервісним проєктом через вебзастосунок, десктопну програму або консоль [3] – [5].

Після роздроблення системи на багато МС (Мікросервіс – це найпростіша одиниця, сервіс, який приймає вхідні запити для здійснення дії), треба визначити, яку використовувати архітектуру і як зв'язати між собою розроблені сервіси. Визначимо основні мікросервіси, РСО, що будуть розроблені, а також допоміжні елементи та сервіси системи, що будуть використанні при побудові майбутньої система «BestMovie».

1.2 Огляд існуючих систем, сервісів, сайтів, що надають змогу переглядати фільми і створювати власні колекції

Перегляд фільмів є популярним видом розваг, і на сьогоднішній день існує безліч сервісів, систем та сайтів, що надають можливість перегляду фільмів онлайн. У цьому розділі розглянуті та проаналізуємо їх переваги та недоліки. Крім того, також розглянуті можливості створення власних колекцій фільмів на деяких з цих сервісів та систем.

1.2.1 Передмова

Перед розглядом аналогів, слід зазначити, що аналоги нашої системи мають безкоштовний контент і ряд функцій, але деякі потребують авторизації і підтвердження користувача. Завдяки таким умовам можна створити справедливую і критичну оцінку сайтів і сервісів. Основні сервіси і сайти, що розглядаються:

- сайт “Rezka.ag”;
- сервіс “Filmix.ac”;
- сервіс “Sweet.tv”.

Під час написання роботи було проведено аналіз основних сервісів та сайтів, які надають можливість перегляду фільмів та створення власних колекцій. Для проведення аналізу використовувалися такі критерії:

- наявність можливості завантаження файлів на пристрій;
- зручність перегляду;
- каталог доступних фільмів, серіалів та їх якість;
- основні принципи каталогізації;
- базові функції сортування, пагінації, групування, фільтрації;
- наявність колекцій та функцій їх редагування та створення;
- багатоплатформність;
- додаткові функції.

1.2.2 Сайт Rezka.ag

Вебсайт Hdrezka.me для перегляду фільмів онлайн в HD якості надає можливість перегляду різних фільмів, серіалів [6]. Сервіс пропонує обмежену якість відео, зручний пошук по жанрам, року випуску, акторам і режисерам. Всі матеріали на сайті розміщені безкоштовно, але їх неможливо завантажити. Також сервіс Hdrezka дозволяє користувачам створювати свої власні колекції фільмів та серіалів.

Переваги:

- наявність можливості створювати колекції;
- наявність колекцій за замовченням;
- кількість фільмів дорівнює 9306-и, кількість серіалів дорівнює 2836-и, кількість мультфільмів дорівнює 837-и;
- стрічка новин, анонсів, дат релізів серій і фільмів;
- адаптивність під браузері Opera, Google Chrome, Firefox, мобільні платформи Android, iPhone, iPad;
- можливість вибрати тему оформлення сайту.

Недоліки:

- не зручний інтерфейс додавання фільмів у власні колекції,
- порушений баланс уваги користувача, приклад інтерфейсу наведено на рисунку 1.1;
- більшість медіа доступні лише з низької якістю – 1080 full hd;
- низька швидкість роздачі медіа;
- мала кількість категорії і колекцій за замовченням;
- відсутність групування за студією видавцем;
- відсутність можливості групування за декількома акторами;
- відсутність адаптації для Smart TV;
- відсутність адаптації під декілька мов;
- проблеми зі стабільністю.

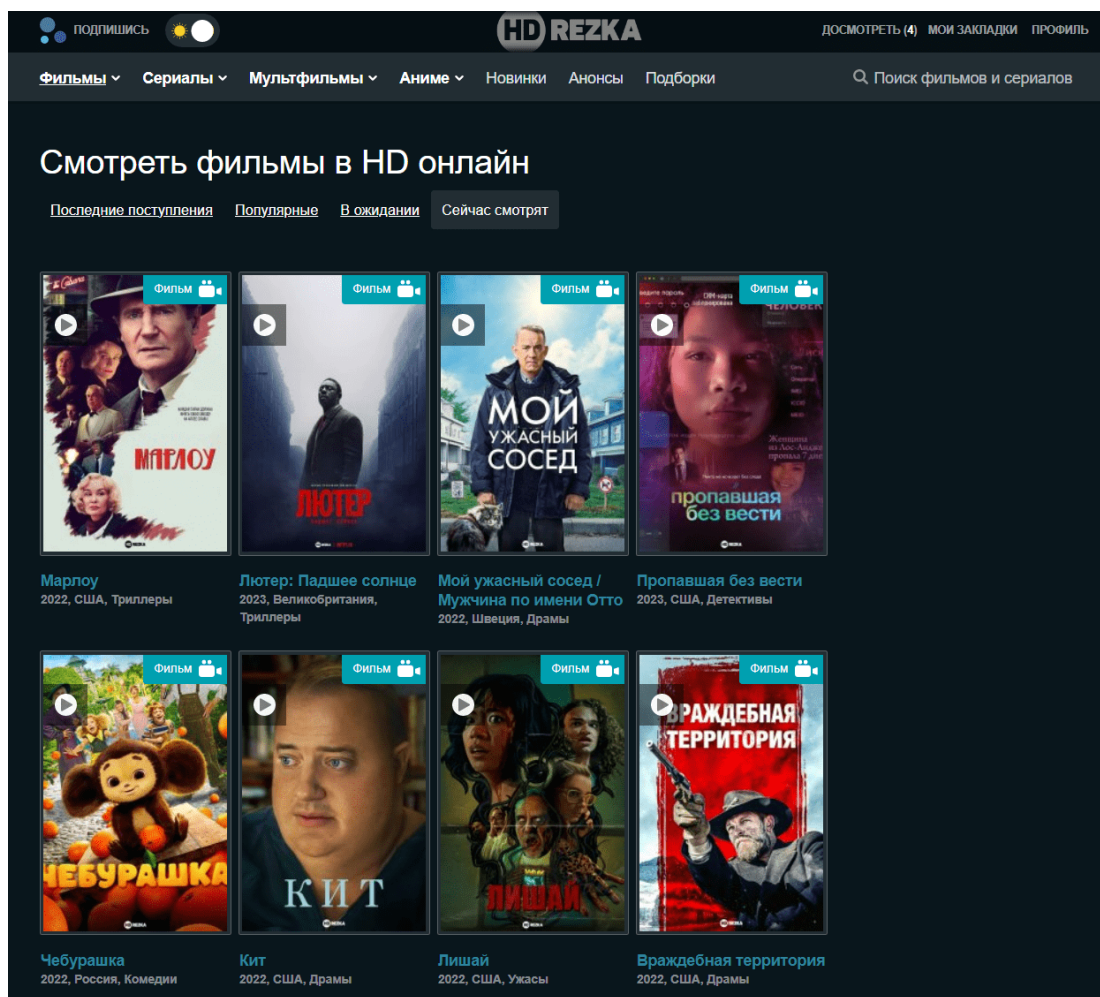


Рисунок 1.1 – Головна сторінка сайту Rezka.ag

1.2.3 Сервіс Filmix.ac

Сервіс Filmix надає можливість переглядати для перегляду фільмів, Мультфільмів, серіалів різних жанрів і форматів у чудовій 1080p [7]. На сайті доступні фільми і серіали різних років випуску, включаючи новинки кінематографа. Особливістю сайту є наявність можливості завантажити медіа файл. Також на сайті можна створити власну колекцію фільмів і серіалів, вести список переглянутих творів, а також зберігати улюблені фільми та серіали в обране. Filmix пропонує українське та російське озвучення для більшості фільмів і серіалів.

Переваги:

- наявність можливості створювати колекцій;
- наявність можливості завантажувати медіа файли;
- адаптивність під браузері Opera, Google Chrome, Firefox, мобільні платформи Android, iPhone, iPad;
- висока швидкість роздачі;
- медіа з надвисокою якістю 4K;
- наявність інтуїтивно зрозумілих фільтрів, це можна побачити у правому нижньому куті рисунку 1.2.

Недоліки:

- мала кількість категорії і колекцій за замовченням;
- незрозуміла стрічка новин;
- відсутність вибору дизайну сервісу;
- відсутність групування за студією видавцем;
- відсутність можливості групування за декількома акторами;
- відсутність адаптації для Smart TV;
- відсутність адаптації під декілька мов.

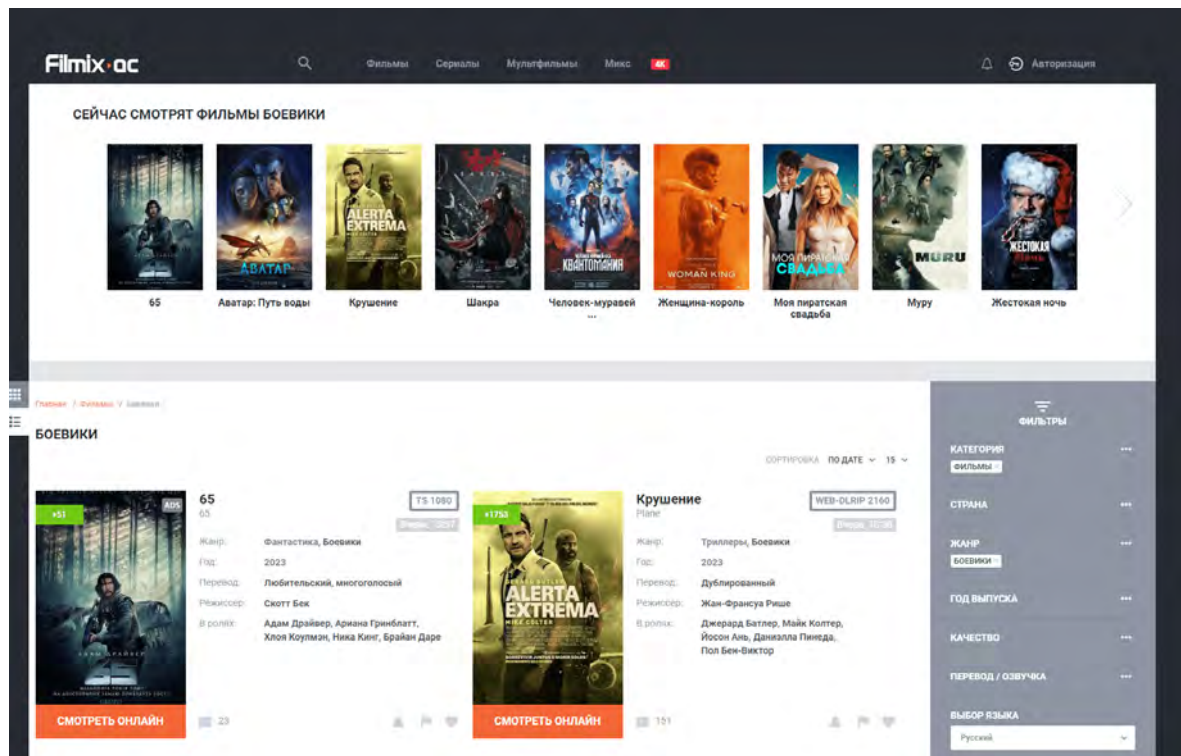


Рисунок 1.2 – Головна сторінка сервісу Filmix.ac

1.2.4 Сервіс Sweet.tv

Телебачення та фільми в ультрависокій якості доступні на Sweet TV [8]. Компанія-видавець: ТОВ "ОТТ Україна". Sweet TV це український інтернет сервіс, який надає можливість перегляду телеканалів та відео контенту онлайн. Сервіс пропонує широкий вибір телеканалів з різних країн світу, а також доступ до великої колекції фільмів, серіалів та іншого відео контенту, який можна дивитися в будь-який зручний для глядача час.

Переваги:

- наявність TV каналів;
- адаптація під декілька мов;
- наявність декількох колекцій за замовченням, приклад наведено на рисунку 1.3;
- групування за студією видавцем;
- висока швидкість роздачі;

- адаптивність під Smart TV, браузери Opera, Google Chrome, Firefox, мобільні платформи Android, iPhone, iPad;
- стрічка новин;
- додаткова функція навчання з відповідними медіа.

Недоліки:

- відсутність можливості групування за декількома акторами;
- мала кількість фільмів;
- відсутність можливості створювати колекцій;
- відсутність фільтрів;
- відсутність вибору головного дизайну сервісу.

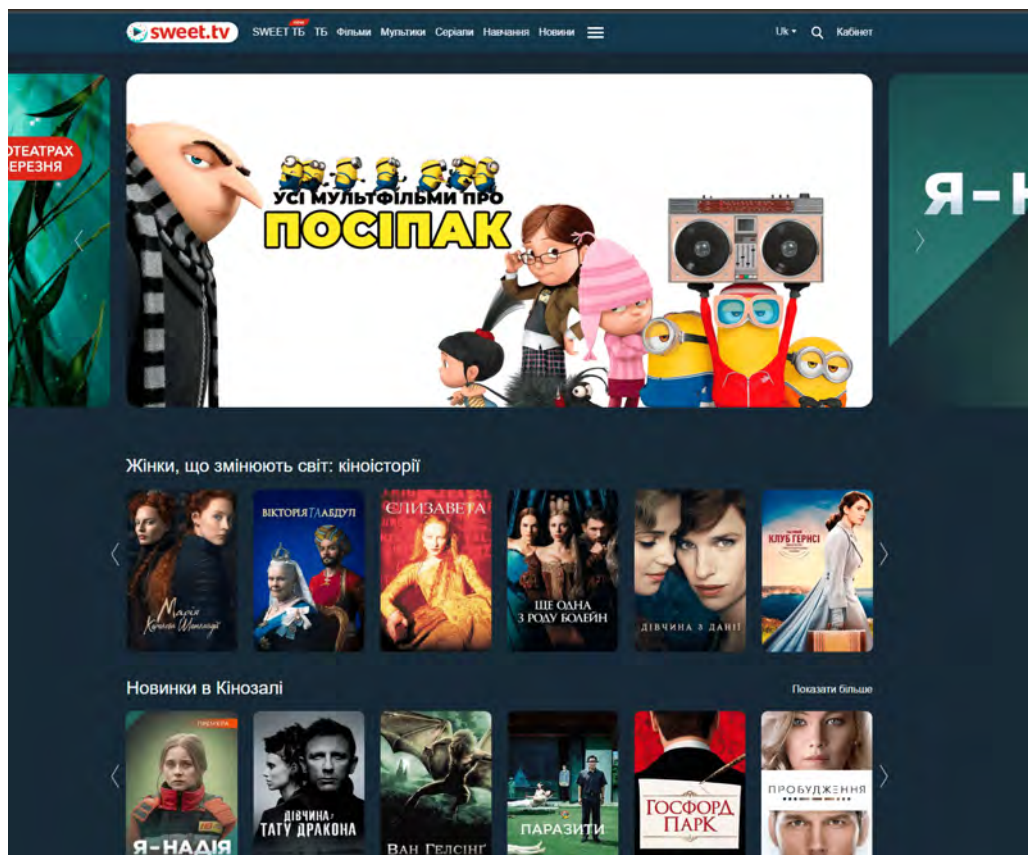


Рисунок 1.3 – Головна сторінка сервісу Sweet.tv

Загалом усі аналоги можна порівняти за рядом характеристик, порівняльну таблицю наведено у таблиці 1.1.

Таблиця 1.1 – Порівняння сайтів і сервісів аналогів

Порівняльні характеристики	Hdrezka.ag	Filmix.ac	Sweet.TV
Перегляд фільмів	Зручний	Зручний	Не зручний
Якість фільмів	Низька	Висока	Висока
Категорії і колекції	+	–	–
Адаптація під SmartTV, Android	–	Android	Smart TV
Стабільність	Низька	Низька	Висока
Локалізації	–	–	UA EN
Наявність вибору дизайну	+	–	–
Швидкодія	Низька	Висока	Низька
Безпечність	Низька	Низька	Висока

1.3 Висновки з розділу і постановка задачі

Під час детального аналізу було визначено основні характеристики, які повинен мати сайт (сервіс), що розроблюється, а саме: зручність у використанні і перегляді, масштабованість, швидкість завантаження і відображення елементів і ресурсів сайту.

Окремо необхідно створити систему авторизації для користувачів, розробити три ролі – адміністратор, зареєстрований користувач, звичайний переглядач.

Основні функції, що доступні усім трьом користувачам, які повинен мати сайт: можливість переглядати фільми, трейлери, афіши, детальну інформацію про медіа за умов пагінації. Можливість сортування за категоріями, колекціями. Адміністратор має додаткові можливості, такі як створення, видалення колекцій, категорія за замовчення, а також створення,

видалення і редагування інформації про фільми і афіши. Зареєстрований користувач має можливість створювати, видаляти колекції і додавати фільми до колекцій.

Усі розроблені класи, структури і логіка повинні розміщатися строго за рівнями DDD архітектури.

2 АНАЛІЗ ПРОГРАМНИХ ЗАСОБІВ

У даному розділі розглянуті основні особливості мов програмування, архітектури та обраних програмних засобів, що будуть використані при реалізації проєкту.

2.1 Огляд особливостей мов програмування

Перша обрана мова програмування це PHP (Hypertext Preprocessor – це скриптова мова програмування, яка використовується для розробки вебзастосунків та вебсайтів. PHP є динамічною мовою, що означає, що виконання програми здійснюється на стороні сервера) версії 8.0. PHP це доволі популярна і розвинута мова програмування, яка надає великий спектр інструментів для виконання різноманітних завдань. Детальніше про синтаксис, структурні одиниці і технічні характеристики мови:

- так як ця мова об'єктно-орієнтовна, то в неї є класи, інтерфейси, абстрактні класи, що дозволяє писати код дотримуючись основних принципів інкапсуляції, наслідування і поліморфізму [28];
- легкість у створенні і використанні масивів, мова надає можливість створювати масиви без ініціалізації покажчиків і ручного виділення пам'яті під дані, також є багато різноманітних функцій для контролювання даних масиву або декількох масивів;
- окремо слід виділити таку структуру, як trait, вона дозволяє розширити існуючий клас додатковим кодом без використання абстракції і наслідування, а потім використати цей trait у іншому класі без дублювання коду. Окремо слід зазначити, що за допомогою trait можна відокремити важкі залежності одного класу системи від інших, без підвищення складності тестування;
- є переліки, які з'явилися у версії 8.1 в 2021 році, що є свідченням про розвиток мови;

- підтримка слабкої динамічної і статичної типізації, або комбінування обох підходів;
- класові магічні методи, що допомагають вирішити відсутність стандартного перевантаження функцій, за рахунок створення динамічних властивостей і методів;
- достатньо швидка для виконання завдань пов'язаних з розробкою застосунків, сайтів, мікросервісних систем;
- мова програмування високого рівня.

Докладніше про інструменти:

- ядро мови має надбудови у вигляді розширень, що збільшують спектр функцій мови, такі як: `bcmath`, `bz2`, `ctype`, `mcrypt`, `fileinfo`, `curl`, `dom`;
- під мову розроблено багато платформ і CMS (Content Management System – це програма, яка використовується для керування вмістом, що дозволяє багатьом учасникам створювати, редагувати та публікувати), наприклад – `Laravel`, `Symphony`, `CodeIgniter`, `WordPress`, `Drupal`, `Joomla`;
- для регулювання версій пакетів та бібліотек розроблено `Composer`;
- для створення автоматичних тестів, що тестують розроблену систему, розроблено платформу `PHPUnit`;
- для покрокового тестування і налагодження коду – `Xdebug` і `Zend`;
- окрема документація з правилами стилю коду PSR-12 (PHP Standards Recommendations – це стандарт кодування на мові PHP, що визначає порядок використання методів або визначення конструкторів класів, загальні положення про форматування стилю коду);
- сумісність з `Git` для зручного контролю версій програмного коду;
- зручна IDE `PHPStorm`, що надає можливість слідкувати за виконання правил PSR-12 і має інтеграції для використання і слідкування за станом багатьох сучасних технологій;

– для реєстрації, контролю версій і завантаження пакетів існує онлайн бібліотека пакетів Packagist.

Докладніше про популярність мови:

- існує велика кількість детальної документації, відео і фото матеріалів з прикладами;
- 78.8% відсотків вебсайтів використовують PHP у ролі back-end серверної частини [9].

Слід зазначити, що обрана мова програмування має конкурентів, а саме C#, Java, Python. За допомогою цих мов також можна створювати складні мікросервісні системи, так як вони мають базові інструменти для створення вебзастосунків. Порівняльна характеристика наведена у таблиці 2.1.

Таблиця 2.1 – Порівняння мов програмування

Характеристики	PHP	C#	Java	Python
1	2	3	4	5
ООП, класи, інтерфейси, абстаркції	Присутні, але менш розвинуті	Присутні і добре розвинуті	Присутні і добре розвинені	Присутні, але менш розвинуті
Типізація	Слабка динамічна	Сильна статична	Сильна статична	Слабка динамічна
Система управління пам'яті	Немає	Є	Є	Немає
Платформи	Laravel, Symfony, Yii	.NET Framework, .NET Core	Spring, Struts	Django, Flask

Продовження табл. 2.1

1	2	3	4	5
Реєстрація пакетів онлайн	Packagist	NuGet	Maven Central	PyPI
Система відлагодження коду	Xdebug, Zend Debugger	Visual Studio Debugger, .NET Debugger	Java Debugger, Java Debug Wire Protocol	Python Debugger
Асинхронність	Присутня за допомогою ReactPHP, Swoole і інших бібліотек	Присутня за допомогою async/await	Присутня за допомогою CompletableFuture	Присутня за допомогою asyncio
Structure, Enum, Traits	Немає Structure	Немає Traits	Немає Traits	Немає Traits
Швидкість розробки	Висока	Середня	Середня	Висока
Гнучкість	Висока	Середня	Середня	Висока
Застосування	Вебзастосунки, десктоп застосунки	Вебзастосунки, десктопні застосунки, ігри, мобільні застосунки	Вебзастосунки, десктоп застосунки, ігри, мобільні застосунки	Штучний інтелект, вебзастосунки, десктоп застосунки, ігри.

Загалом мова PHP більше підходить для реалізації, так як її основний напрямок застосування створення вебзастосунків і вебсайтів. Обрана мова має набагато більшу кількість платформ і готових бібліотек ніж мови у порівняльній таблиці 2.1. PHP наздоганяє за можливостями у ООП і

асинхронних процесах C#, Java, а різниця у швидкості виконання коду не помітна при користуванні вебзастосунком, навіть якщо присутні складні структури на кшталт бінарних дерев або В-дерев, також швидкість у більшості випадків залежить від якості коду. Обрана мова схожа за синтаксисом з C#, тому документація з реалізації шаблонів програмування від компанії Microsoft доповнює і так досить велику кількість ресурсів з інформацією для програмування на PHP. Також на обраній мові програмування можна більш швидко створювати і розгортати вебзастосунки.

Для створення рівня інтерфейсу у обраній архітектури краще підходить JS (Java Script – це динамічна, об'єктно-орієнтована мова програмування). Одна з найпопулярніших мов програмування, що має ряд переваг над іншими мовами, а саме над PHP.

- об'єктно-орієнтовна мова;
- підтримка асинхронності, гнучка розробка і застосування асинхронних функцій;
- динамічна типізація;
- підтримка найпоширенішими браузерами Opera, Firefox, Chrome, Microsoft Edge і тд.

Додаткові бібліотеки і інструменти, які розширюють можливості написання коду на мові JS:

- потужні бібліотеки React.js, Vue.js, Angular.js;
- універсальна бібліотека Axios, що надає можливість обміну даними за допомогою HTTP (Hypertext Transfer Protocol – це протокол прикладного рівня для передачі медійних документів [2]) і HTTPs протоколу;
- легкість розгортання розробленого коду за допомогою платформи Node.js, що дозволяє швидко і надійно розгорнути вебсервер;
- сумісність с технологією git;
- велика кількість документацій, бібліотек з готовими гнучких компонентів, такі як Bootstrap, Bulma, Skeleton;

- для регулювання версій пакетів існують декілька менеджерів пакетів yarn і npm.

2.2 Огляд особливостей обраних платформ і бібліотек

MovieMicroservice, EmailTemplateMicroservice, BestMovieStorage мікросервіси розроблено на основі платформи Laravel версії 9.19, ця платформа створена для розробки, як монолітних сайтів, так і застосунків з інтерфейсом API. Переваги Laravel:

- наявність ServiceProvider класів, що допомагають у реалізації шаблону проєктування DependencyInjection, який представляє собою метод досягнення інверсії між класами;
- наявність готових класів для реєстрації API шляхів з однаковим інтерфейсом прийому запитів і відповіді на такі запити, приклад таких класів: Route, Request, Controller, Resource;
- наявність базового класу Model і конструктора запитів Eloquent, фасаду DB для роботи з сутностями у БД;
- наявність готових конструкторів для розробки міграції;
- наявність інтерфейсу Artisan для роботи через консоль, за допомогою нього можна контролювати міграції, створювати класи провайдерів, моделей, контролери. Контролювати сертифікати, ключі, різні кеши, у тому числі і кеш пов'язаний з файлом “.env”. Окремо у рамках взаємодії з консоллю через Artisan існує Laravel Tinker, за допомогою якого можна на основі конструкторів вбудованих в розроблені класи-моделі або фасаду DB роботи запити до активної БД;
- сумісність з платформою PHPUnit, існує ряд налаштувань, які дозволяють запускати розроблені інтеграційні тести на швидко створеній БД з СКБД (система керування бази даних) SqlLite;
- детальна документація з прикладами;
- найпопулярніша платформа з платформ PHP [10].

Для розробки вебсерверу було обрано бібліотеку React, це потужна бібліотека на мові JS, що дозволяє створювати швидкі і оптимізовані сайти, за допомогою використання різноманітних вбудованих хуків і створення станів. Слід зазначити, що бібліотека React має багато прихованих можливостей, які доволі важко дослідити з програмної точки зору, тому що відсутня система налагодження коду у реальному часі. Докладніше про переваги React:

- підтримка віртуальної DOM (Document Object Model – представлення даних об'єктів, які складають структуру та вміст документа в Інтернеті [11] – [12]) – React.js використовує віртуальну DOM, що дозволяє мінімізувати затрати на зміну стану компонентів та покращує продуктивність застосунків;
- підтримка серверного рендерінгу – React.js надає можливість рендерити компоненти на стороні сервера, що дозволяє покращити продуктивність застосунку та забезпечити SEO-оптимізацію (Search Engine Optmization – оптимізація під певні пошукові системи [13]);
- масштабованість, надає можливість легко масштабувати проекти, створюючи компоненти, які можна повторно використовувати і комбінувати між собою;
- модульність, дозволяє розділяти код на невеликі, легко керовані частини;
- декларативний підхід до розробки інтерфейсів, що забезпечує зручне керування станом компонентів і спрощує процес розробки.

2.3 Огляд особливостей обраних інтерпретаторів

Для запуску і розгортання багатьох МС на мові PHP було обрано сервер Nginx і інтерпретатор PHP-FPM (менеджер процесів PHP), декількох додаткових розширень. Nginx це вебсервер, який потребує встановлення інтерпретатору PHP-FPM і створення конфігураційного файлу у папці “sites-

available” та “sites-enabled”. Після налаштування доменного імені, шляху до інтерпретатору і проєкту, все що треба це запустити сервер, таким чином більше не треба перезапускати сервер або постійно використовувати кнопку запуску проєкту у обраній IDE або інші способи, які потребують систематичного ручного втручання.

Переваги Nginx:

- висока продуктивність, сервер здатен обробляти значні обсяги запитів і трафіку без втрати продуктивності;
- надійність, сервер забезпечує стабільну роботу сервера і захищає його від падінь в разі надмірного навантаження або атаки;
- гнучкість, Nginx можна налаштувати на виконання різноманітних завдань, таких як обробка запитів, кешування, балансування навантаження, захист від DDoS (Distributed Denial of Service Attack – це атака у якій зломисник переповнює сервер інтернет-трафіком, щоб запобігти доступу користувачів до підключення онлайн сервісів і сайтів) атак;
- підтримка протоколу HTTP/2, що забезпечує більш швидкий і ефективний обмін даними між клієнтом і сервером;
- підтримка захисту передачі даних за допомогою протоколу TLS/SSL, що забезпечує шифрування даних та захист від зломисників.

Для запуску сервісу на мові React послідовно поєднується ряд технологій для запуску і розгортання проєкту на сервері, деякі з технологій зазвичай вбудовані в браузер, тому вважається, що інтерпретатор проєкту на мові JS це браузер.

По-перше, код розгортається на вебсервері Node. По друге, до передачі коду у інтерпретатор, код на React перетворюється у JS за допомогою Babel і Webpack. Після перетворення коду його оброблює двигун, що вбудований у браузер. У якості основного браузера обрано Firefox 109.0, тому використаний двигун це SpiderMonkey. Слід зазначити, що для розгортання проєкту на сервері треба кожен раз викликати команду “npm

start”, npm це один з менеджерів пакетів, його було обрано основним менеджером для регулювання пакетами проєкту.

2.4 Огляд архітектури DDD

Проєкт на архітектурі DDD за одним з головних правил повинен відображати бізнес логіку процесів, тому зазвичай назви змінних і структур, максимально повинні відповідати бізнес визначенням, що використовує менеджер продукту, розробники дизайн і інтерфейсу, така особливість має термін – “Ubiquitous language”. Таким чином можна стандартизувати назви деяких класів, визначити стандарт щодо напрямків їх залежностей і способи взаємодії між такими класами у контексті проєкту на платформі Laravel і мові PHP.

Архітектура DDD поділяється на чотири рівні – Application, Domain, Infrastructure, деякі ресурси виділяють і четвертий рівень - User Interface, така рівнева система описана у “зеленій” книзі “Domain Driven Design Quickly” [14]. Рівень User Interface це спосіб звернення до мікросервісної системи з зовнішнього середовища, наприклад: консольна команда, десктопний застосунок, вебзастосунок, платформи для тестування API (Postman).

2.4.1 Application рівень

У Application рівні розташовані наступні компоненти системи: API контракти, додаткові сервіси, консольні команди, вбудовані у платформу класи і процеси, система безпеки. У платформі Laravel наявні такі класи: Route, Controller, Request, Resource, Middleware, Payload. Route це клас, що реєструє API контракт і визначає Controller, що буде реалізовувати цей контракт. Клас Request – це сформований клас запиту, що поступив на сервер, окрім стандартних даних запиту, а саме - data-row, json, query params, headers, він може мати дані про завантажені файли, браузер, додаткову

інформацію про реакцію сервера на дані в залежності від визначеної конфігурації сервера. Також клас Request має вбудований Validator, за допомогою якого можна обмежити вид, тип, зміст і розмір даних. Класи Middleware існують для перетворення окремих даних або для перевірки безпечності запиту. Клас Resource визначає загальний інтерфейс відповіді на запит, а саме відповідь у вигляді json. Resource може формувати відповідь з одного або декількох Агрегатів або класів Payload. Клас Payload має конкретно обрану і відвалідовану інформацію з запиту, такий клас буду відправлений на наступний рівень. Схема взаємодії класів у Application рівні наведена на рисунку 2.1.

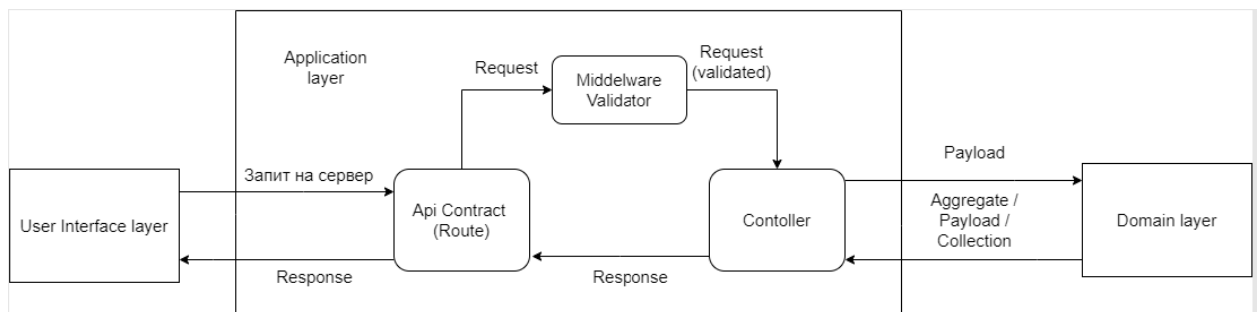


Рисунок 2.1 – Взаємодія класів application рівня

2.4.2 Domain рівень

У основі Domain рівня знаходиться декілька шаблонів проєктування агрегати і сутності, сервіси і репозитарії (з англ. Service and Repository), останній шаблон проєктування добре описаний у книзі “Clean Arhitecture” [20]. DDD архітектура наклала на поєднання цих шаблонів свої правила. Одне з яких полягає у відокремленні і визначенні меж узгодженості на Domain рівні між класами сервісами і агрегатами. Більша частина бізнес логіки перекладається з сервісів на класи агрегати (з англ. Aggregate). У Domain архітектурі багато уваги приділено саме класам агрегатам, ці класи контролюють стан сутності (з англ. Entity) [15] – [17]. Кожна таблиця у БД це сутність або підсуб’єктність (з англ. Subentity). Агрегат може контролювати

одну сутність і багато підсуб'єктностей. Для формування агрегату застосовується клас-сервіс. У контексті процесу виконання API контракту клас-сервіс приймає Payload, це скорочення для таких структур DPO (Domain Payload Objects – об'єкт корисного навантаження домену). Цей клас не є обов'язковим критерієм, так як присутність у архітектурі класів Payload це результат правильного об'єктно-орієнтовного стилю програмування і відокремлення зайвих елементів рівня Application від Domain рівня. Загалом шаблон сервіс створювався для CRUD логіки, але у DDD він може приймати участь у складних процесах перевірки інваріантів. Після отримання Payload сервіс має логіку зі створення, оновлення, вибору нової або нових сутностей шляхом створення класів агрегатів або фільтрів, що мають логіку перевірки правил і будуть передані на наступний рівень, а саме Infrastructure. Слід зазначити, що клас агрегат має недолік, для вирішення якого існує багато підходів, але немає універсального, цей недолік виражається у поступовому перетворенні агрегату у супер клас. Один з підходів, який можливо здійснити у PHP, це розбиття класу на більш менші структури, а саме на trait, які будуть відокремлювати по частинах логіку агрегату. Domain рівень не має реалізації класів репозитаріїв, але має їх інтерфейси. Також Domain рівень має інтерфейси сервісів. Для більш точного відображення процесів сервісів і агрегатів слід виділити п'ять сценаріїв:

- створення, наведено на рисунку 2.2;
- оновлення, наведено на рисунку 2.3;
- видалення, наведено на рисунку 2.4;
- вибірка, наведена на рисунку 2.5;
- процес з ускладненою бізнес логікою у якій пов'язано декілька агрегатів [25].

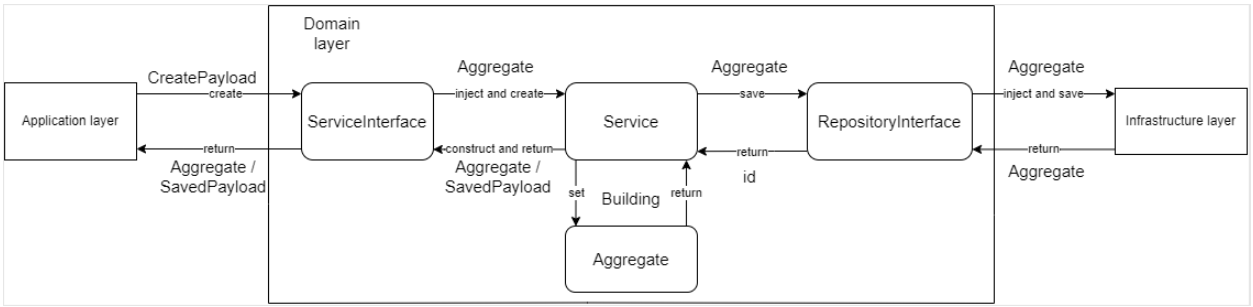


Рисунок 2.2 – Сценарій створення нової сутності у Domain рівні

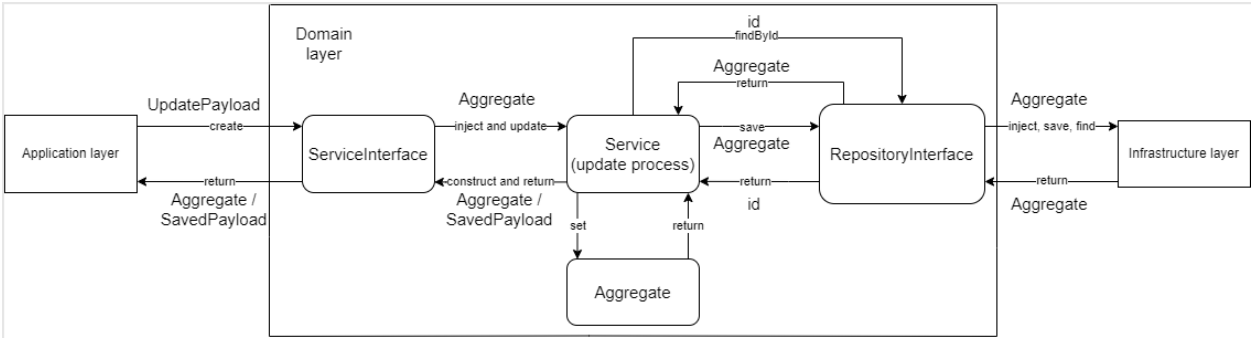


Рисунок 2.3 – Сценарій оновлення сутності у Domain рівні

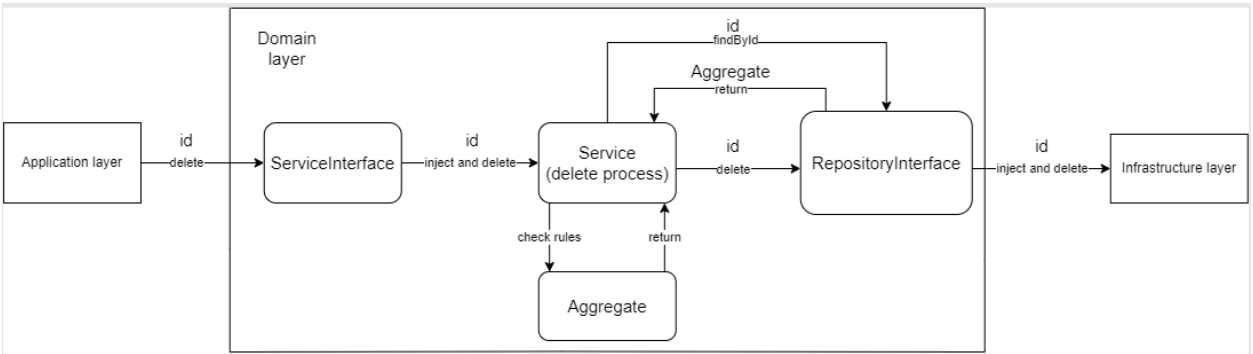


Рисунок 2.4 – Сценарій видалення сутності у Domain рівні

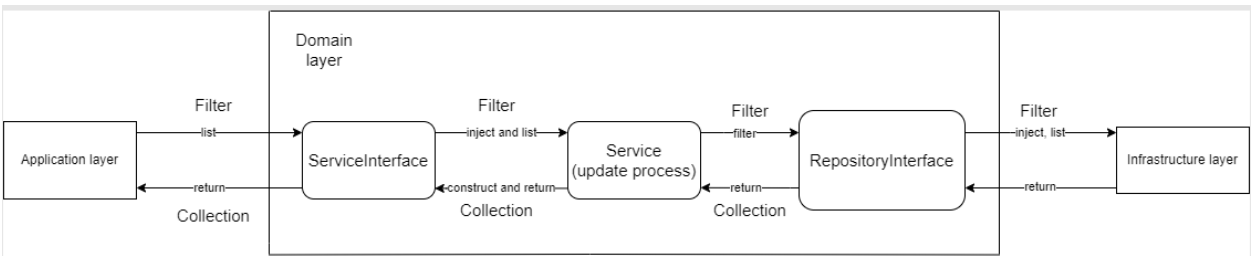


Рисунок 2.5 – Сценарій вибірки багатьох сутностей у Domain рівні

Архітектура самого класу-агрегату є сукупністю полів сутності, відповідних сетерів і гетерів, також клас, що використовує trait resolvers, такі структури надають можливість класу звертатися до сервісів, репозитаріїв,

диспетчеру шини подій, класів менеджерів, Validator, класу конфігурації. Перевага використання trait resolvers полягає в уникненні важких залежностей класу, що значно полегшує тестування і розвантажує агрегат завдяки використанню шаблону проєктування “Ін’єкція залежності” (з англ. Dependency Injection). Слід зазначити, що якщо сутність якій відповідає агрегат має підсуб’єктність, то у агрегаті з’являється нове поле, що теж є відповідним відображенням даних логіки підсуб’єктності. Агрегат може мати такі структури, як Value Object — це клас, що відображає дані сутності у агрегаті, які не мають унікального ідентифікатора і є незмінними, тобто якщо дані Value Object змінилися, то це кардинально новий об’єкт [24].

2.4.3 Infrastructure рівень

Третій рівень DDD це Infrastructure, він забезпечує зв’язок з БД, має конструктори запитів, моделі, класи репозитаріїв. Платформа Laravel має класи моделей, ці класи надають можливість описати поля і зв’язки сутності з іншими сутностями і без виклику фасаду DB створити запит з допомогою вбудованого конструктора sql-запитів. Завдяки платформі Laravel і класам моделям рівень Infrastructure є дуже простим, але реалізація роботи з БД через моделі також має недоліки, так як така модель є реалізацією шаблону “Активний Запис” (з англ. Active Record). Основним недоліком активного запису полягає у створенні декількох запитів під час складних операцій, яких можна було уникнути завдяки використанню нативних sql-запитів, це сповільнює систему. Також є два шаблони програмування, що можуть частково в залежності від логіки системи вирішити цю проблему, а саме: UOW (Unit of Work – призначений для створення рівня абстракції між рівнем доступу до даних і рівнем бізнес-логіки програми [18]), CQRS (The command and Query Responsibility Segregation – це шаблон, який розділяє операції читання та оновлення для сховища даних. Впровадження CQRS у вашу програму може максимізувати її продуктивність, масштабованість і безпеку

[19]). UOW рекомендовано використовувати у архітектурі DDD так як він підвищує швидкість обробки одного запиту шляхом збереження стану сутності при збереженні її у БД або після вибірки. Таким чином можна декілька разів оновлювати сутність у межах одного запиту, але під час вибірки не звертатися до самої БД, такий підхід зменшує кількість запитів. Також шаблон UOW має додаткову перевагу, але вона залежить від реалізації цього шаблону, вона полягає у зміні сутності у межах однієї бізнес транзакції, якщо зберігати усі зміни сутності у карті UOW, то треба буде лише два рази звернутися до БД, а саме під час фінального збереження сутності і під час її першої вибірки [21]. У залежності від реалізації такий шаблон програмування потребує багато часу для впровадження, якщо потрібно інтегрувати його у вже готову систему. Приклад UOW у DDD наведено на рисунку 2.6. Суть шаблон CQRS полягає у створенні двох БД, одна для запису інформації, друга для вибірки, БД у яку йде запис виконує асинхронну синхронізацію з БД для зберігання даних. БД для запису немає індексів, зовнішніх ключів і має спрощену структуру, навіть в залежності від реалізації синхронізатору може відрізнятися СКБД. Спрощена структура дозволяє більш швидко виконувати вставку великих даних, якщо вони мають достатньо велику кількість ключів і індексів, на основі яких БД створює бінарні дерева і хеш-карти для упорядкування, підрахування і швидкої вибірки даних. БД для вибірки навпроти має повноцінну структуру орієнтовану на швидку вибірку даних. Вигляд Infrastructure рівня наведено на схемі використанням на рисунку 2.6.

2.4.4 User Interface рівень

У DDD немає точних визначень архітектури рівня User Interface, основний критерій це надати користувачу можливості взаємодії з багатьма МС і участі у розроблених бізнес сценаріях. Загалом рівень повинен надавати необхідні інструменти для роботи з багатьма МС. Тому за основу архітектури

для цього рівня було взято шаблони програмування MVC (Model View Controller - це шаблон у розробці програмного забезпечення, який зазвичай використовується для реалізації інтерфейсів користувача, даних і логіки керування) і репозитарій. View представлена у вигляді HTML сторінки розробленої на основі компонентів бібліотеки React та компонентів Bootstrap. Model це звичайні класи-моделі, що зберігають дані отримані від МС за допомогою репозитарію, який звертається до API контракту. Controller це хуки, які відокремлюють логіку в HTML сторінки, наприклад логіку для перевірки форм або оновлення моделей за допомогою звертання до API.

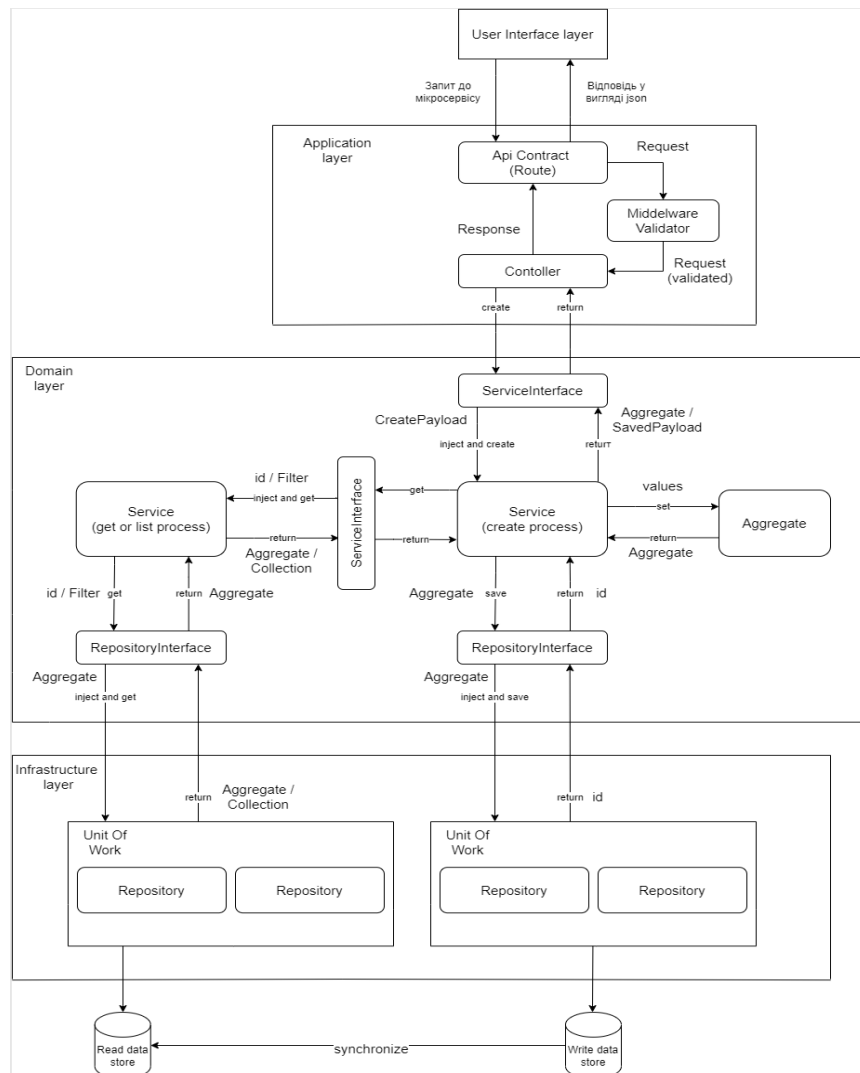


Рисунок 2.6 – Складний сценарій створення сутності

2.4.5 Компоненти присутні у декількох рівнях

Події і обробники, що є реалізацією шаблону “Спостерігач” (з англ. Observer), такі класи можуть бути присутні у всіх рівнях системи, так як вони по суті є реакцією на подію, але за логікою відокремленості рівнів DDD, не можна, щоб подія одного рівня мала обробник у іншому. Такий шаблон дозволяє прискорити систему і розбиття великих логічних транзакцій на менші, якщо бізнес логіка дозволяє це виконати. Події і обробники можуть працювати в асинхронному режимі.

Фабрики, як реалізація шаблону “Фабрика” (з англ. Factory), можуть бути використані у виборі і підключенні сервісів, валідаторів, менеджерів, інших фабрик в залежності від завдання.

Класи-валідатори перевіряють запит API контракту, а у domain можуть бути використані для перевірки інваріантів агрегату, якщо правила гнучкі і змінюються в залежності від даних агрегату. Слід зазначити, що фабрики і валідатори часто використовуються разом, наприклад якщо для однієї сутності відокремлюють декілька правил в залежності від типу або статусу. У Infrastructure рівні вони можуть застосовуватися у виборі підключення або контексту БД для запису або читання даних.

2.4.6 Зв'язок одного МС з іншими

Усі МС пов'язані між собою завдяки РСО. Існує два способи реалізації такого контракту. Перший – якщо МС розташовані у межах одного git репозитарію і мають спільний простір імен. Такий РСО представлений звичайним сервісом, який з'єднує сервіси першого і другого МС. Цей спосіб є простішим, оскільки потрібно розробити лише інтерфейс сервісу та клас-сервіс, що перевизначить повернутий агрегат або колекцію у новий вигляд відповіді та відокремить логіку Domain рівня другого МС від першого.

Другий спосіб застосовують у разі, коли два МС розташовані у різних git проєктах і не мають спільного простору імен. У цьому випадку всі сторонні МС підключені за допомогою класів сервісів і класів репозитаріїв, які звертаються до сторонньої API. Такий сервіс відноситься до рівня домену, а спілкування з API - до Infrastructure рівня. Спосіб спілкування відображений у вигляді інтерфейсу сервісу, сервіс та API повинні бути представлені у вигляді пакету, який можна завантажити на будь-який МС.

2.5 Огляд особливостей обраної СКБД

В ролі основної СКБД для виробничого середовища було обрано MySQL 8.0.32, так як це найновіша стабільна версія. MySQL є реляційною СКБД, це дозволить легко створювати зв'язки між таблицями за допомогою зовнішніх ключів. Обрана версія підтримує пошук за регулярними виразами і пошуком по полю з типом json, складні дати з секундами і мілісекундами. У MySQL є ряд переваг:

- безпечність, є можливість налаштувати права на перегляд і редагування для різних користувачів, також є можливість шифрувати паролі;
- масштабованість, якщо треба змінити тип поля або додати поле, таблицю, достатньо лише зробити нову міграцію, що розширить існуючу БД;
- оптимальні швидкість і прискорення за допомогою індексів, окремо налаштованих кешів;
- цілодобове безперебійне функціонування;
- набір функцій для реалізації конфігурації реплікації за принципом “майстер і робітники” (з англ. “master and slaves”);
- підтримка транзакцій, транзакції повністю атомарні, ізольовані, послідовні. СКБД забезпечує миттєву ідентифікацію блокування між серверами за допомогою контролювання посилованої цілісності, що підтримується сервером. Це дозволяє зменшити час на виявлення проблем з блокуванням і забезпечити більш ефективне управління СКБД. Таким чином,

забезпечується висока доступність і надійність системи за рахунок зменшення часу відмови і забезпечення швидкого відновлення послуг;

- надійність і легкість використання.

2.6 Висновки з розділу

У розділі визначені сукупності переваг мов, платформи і обраної архітектури, на їх основі можна визначити частини логіки шаблонів, що повинні буду закладені у фундамент програмного забезпечення на архітектурі DDD. Завдяки вивченню особливостей кожного аспекту архітектури можна закласти потужний фундамент для подальшого масштабування системи, попереднє вивчення архітектури у поєднанні з програмними засобами також дозволить корегувати процес розробки і чітко визначати орієнтири, які будуть вказувати на готовність системи для впровадження нового шаблону. Кожна обрана мова, платформа, бібліотека збільшують можливості масштабованості проєкту за рахунок більшого вибору доступних інструментів.

3 ОСНОВНІ РІШЕННЯ З РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ

У даному розділі розглянуті основні рішення, застосовані шаблони програмування, розроблені МС і БД, розроблені алгоритми в умовах їх комбінування і використання з іншими технологіями у цілісній системі.

3.1 Основні рішення щодо розроблених МС і РСО

Клієнт сервер має назву BestMovieClient і розроблений на основі мови JS і бібліотеки React. Головний MovieMicroservice і поштовий MovieMicroservice МС, сервіс-сховище написані на мові PHP і платформі Laravel. Поштовий МС використовує Mailgun, як незалежний сервіс для відправки поштових листів. Сервіс-сховище використовує Firebase, як незалежне сховище даних. Перший РСО буде представлений у розробленому пакеті BestMovieCommon на мові PHP, що буде зареєстрований на Packagist і буде доступний для будь-якого проєкту на PHP для встановлення за допомогою Composer. Другий РСО буде прикладом не пакетного підходу у вигляді загального коду на JS з використанням бібліотеки React і Axios [26] – [27].

Система і основні функції створені для управління центральними сутностями: користувач, фільм, колекція. У користувача є три ролі — адміністратор, звичайний користувач, зареєстрований користувач. Адміністратор має можливість додавати, редагувати і видаляти фільми, колекції за замовченням, категорії, завантажувати фільми і картинки до сховища. Звичайний користувач має можливість переглядати списки фільмів, фільтрувати їх за колекціями і категоріями, зареєструватися. Зареєстрований користувач, окрім можливостей звичайного користувача, може створювати власні колекції.

Основний процес реєстрації має підтвердження за електронною поштою по унікальному згенерованому коду. Під час реєстрації

BestMovieClnet ратифікує поля на стороні вебсерверу і відправляє запит на попередню реєстрацію до MovieMicroservice, він також ратифікує запит і дані, що надійшли. MovieMicroservice відправляє запит на EmailTemplateMicroservice, який згенерує код і вставить його у шаблон електронного листа, потім відправить його на електронну пошту користувача. У ролі mail-сервера використано Mailgun. Після отримання листа користувач вводить отриманий код і система підтверджує успішну реєстрацію. Для реалізації спілкування між BestMovieClnet і MovieMicroservice, перший МС має вбудований PCO, а другий має PCO BestMovieCommon, як встановлений пакет.

У системі наявно декілька процесів авторизації: між декількома МС, між користувачами і МС, останній процес ускладнений різними ролями користувачів і для реалізації такої авторизації теж існує декілька підходів. Перший полягає у створенні одного API контракту для декількох користувачів, а другий у створенні окремих API контрактів для різних користувачів. Перший підхід застосовується, якщо правила розгалуження досить прості і не мають складних багаторівневих вкладень, а другий навпроти, якщо бізнес логіка настільки відрізняється, що подальше розширення і тестування такого контракту непропорційно ускладнюється з кожним завданням. Авторизація між МС у багатьох випадках накладається на стандартний процес авторизації і додатково перевіряє, що МС не підмінили зломисники. Вона реалізована у вигляді перевірки відповідності хеш ключів.

Система завантаження файлів також має свої особливості, BestMovieClient при завантаженні відправляє запит с файлом саме на BestMovieStorage і підтримує асинхронний зв'язок, не зважаючи на ліміт часу відповіді від сервера. У ролі головного сховища використано Firebase (безкоштовне хмарне сховище). Відправки запита на пряму до сховища без використання центрального МС (центральним МС можна вважати MovieMicroservice) пришвидшує систему і зменшує навантаження на

центральний МС, який і так відповідає за авторизацію. У деяких випадках система авторизації взагалі відокремлюється на окремий МС, коли становиться занадто складною, наприклад якщо з'являються структури у вигляді дерев, що надають доступ до окремих сутностей. Схема завантаження файлів наведено на рис. 3.1. Слід зазначити, що розроблений код відповідає нормам PSR-12.

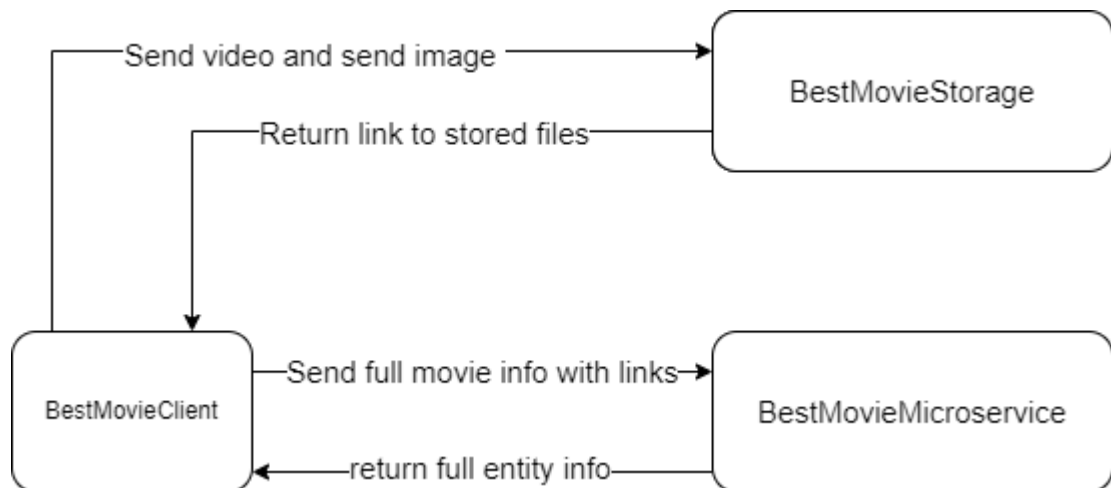


Рисунок 3.1 – Схема завантаження файлів і збереження посилань у БД

3.1.1 МС BestMovieClient

МС BestMovieClient представляє собою User Intreface рівень у архітектурі DDD, який побудований на шаблоні MVC з використанням репозитаріїв, класи репозитарії мають у назві частку Service, так як цей рівень звертається до мікросервісів, а не БД. Таким чином сервіси виконують роль вбудованого PCO, так як вони ґрунтуються на базовому коді JS і мають залежність тільки від бібліотеки Axios, яке є аналогом пакету Guzzle для PHP. Заради уникнення дублювання коду, бібліотеку Axios було перевизначено декілька разів для роботи через базове посилання з доменним ім'ям окремих МС, це класи: BestMovieAxios з доменним ім'ям MovieMicroservice і StorageMovieAxios з доменним ім'ям BestMovieStorage. У посиланні також вказаний протокол HTTP, як протокол спілкування. Для

перетворення відповіді від МС було розроблені методи перетворення даних у класи моделі. У МС присутні сервіси для роботи з BestMovieStorage і MovieMicroservice.

Для роботи з центральним МС було розроблено сервіс UserApiService для оновлення, реєстрації, попередньої реєстрації, авторизації користувача. У класі MovieApiService реалізовано методи для створення, видалення, вибірки одного і багатьох фільмів, створення відношення між рядом колекцій різних типів і фільмами. Для роботи з сутностями колекцій і категорій також створено окремі класи сервіси CollectionApiService і CategoryApiService, вони надають усі стандартні CRUD можливості, окрім оновлення. Так як тільки МС, що працює з БД або сховищем, може визначити кількість сторінок під час вибірки з фільтрами або без додаткових умов фільтрації, то на МС клієнту було створено окремі класи фільтри і класи для зберігання даних посторінкової навігації. Такі класи не мають логіки і є звичайними “валізами”. Завдяки даним з центрального МС можна легко створити посторінкову навігацію.

Для навігації на сайті і обмеженню доступу до сторінок було визначено перелік шляхів у класі App.js за допомогою React DOM Router. Звичайному користувачу доступні такі сторінки: авторизація, реєстрація, перегляд фільмів, перегляд інформації конкретного фільму, сторінка “Про нас”. Для адміністратора було розроблено сторінки з можливістю додавати і видаляти категорії, колекції, а на сторінці конкретного фільму можливість додавати його до колекції і видаляти фільм. Завдяки розробленому компоненту PersistLogin перед перевіркою ролей користувача оновлюються дані у локальному сховищі. Таким чином перевіряється чи існує такий користувач і чи має він доступ до окремих сторінок, такий підхід робить неможливим проникнення видаленого користувача до компонентів системи. Після оновлення даних, якщо сторінка має спеціальні правила для користувача, то вони перевіряються без другого запиту на оновлення даних користувача. Такі інваріанти реалізовані за допомогою компонентів

RequireAdminRoleAuth і RequiredViewerAuth. Перший компонент перевіряє чи є користувач адміністратором, а другий чи зареєстрований це користувач. Компонент перевірки прав адміністратора накладено на розроблені сторінки для адміністратора, а перевірка загальної реєстрації на сторінку з власними колекціями користувача.

Сторінка авторизації має поле для адресу електронної пошти і пароллю і одну кнопку для підтвердження Авторизації. Логіка ратифікації полів знаходиться у хуку useLogin, окрім ратифікації у розробленій структурі знаходиться логіка, яка ініціює запит на центральний МС для перевірки існування користувача, а перед запитом спрацьовує алгоритм ратифікації даних за допомогою схеми перевірки полів на формі, яка зроблена на основі компонентів бібліотеки Formik. Після ратифікації полів і успішного запиту на МС, хук визиває компонент з іншого розробленого компоненту Navigate і відправляє користувача на головну сторінку сайту. Непомітно для користувача хук зберігає згенерований авторизаційний токен у локальне сховище, який надалі використовується для перевірки при переході на різні сторінки, слід зазначити, що сама авторизація спрацьовує кожного разу, якщо на сторінку накладено авторизаційний компонент. Такий компонент знаходить у локальному сховищі токен і надсилає запит на центральний МС, отримує користувача, перевіряє права і починає будувати сторінку. Процес працює дуже швидко, тому користувач комфортно переходить зі сторінки на сторінку.

На головній сторінці сайту розміщено перелік фільмів, категорій та колекцій за замовченням, а також окремий компонент NavBar з різними вкладками в залежності від стану користувача. Логіка обробки натискань на вкладки проста, посилання виконуються за допомогою тегу "<a>", тому окремий хук для управління станом сторінки не потрібний. Перелік фільмів містить 10 пунктів з назвами і зображеннями, кожен фільм клікабельний. Логіка вибірки фільмів знаходиться у хуку useMovies, де реалізовані фільтри та посторінкова навігація за замовченням, обробка подій зміни номеру

сторінки та вибірка фільмів з використанням `MovieApiService`. Хуки `useViewerMovies`, `useCategories` і `useCollections` містять логіку отримання та управління категоріями та колекціями. Для отримання списку фільмів при ініціалізації сторінки використовується хук `useEffect`. Відображення списку фільмів на сторінці здійснюється за допомогою компонентів `MovieList` і `MovieListItem`. Всі обробники подій об'єднані у відповідні функціональні компоненти. Функціональний компонент `useCategories` об'єднує логіку вибірки, створення, видалення категорій і використовує `useEffect` для початкової вибірки даних. Хуки `useCollections` і `useCategories` використовують `useEffect` для вибірки початкових даних. Також вони використовують функціональні компоненти `useCategoryCheckedList` та `useCollectionsCheckedList`. Додаткові компоненти потрібні для відокремлення логіки вибору колекцій і категорій з відображеного списку на сторінці, так як логіка доволі складна і базовий хук становиться досить складним для читання, додатково вони використовуються у компонентах для адміністратора. Сторінка “Мої колекції” використовує логіку базового функціонального компоненту колекцій, оберненого у `useMyMovieCollections`, особливість даного компонента полягає у перевизначенні типу фільтру з типу для колекцій за замовченням на тип власних колекцій користувача, що дозволяє отримати відповідні колекції для сторінки з власними колекціями. Таким чином розроблені базові хуки обернуті у інші хуки, які легко перевизначають фільтри в залежності від сторінки. Сторінка конкретного обраного фільму має однаковий набір функцій для зареєстрованого і звичайного користувача, а саме перегляд інформації про фільм, рисунок, відео. Для перегляду відео використано відокремлений у компонент `MoviePlayer` звичайний тег “<source>”, так як готовий React компонент не підтримує транслювання відео зі сховища `Firebase`. Взагалі у документації `Firebase` написано, що сховище не підтримує транслювання відео, але на практиці все працює і тег “<source>” самостійно контролює процес

перегляду. Посилання на медіа файл приходить разом з відповіддю на отримання конкретного фільму за унікальним ідентифікатором.

Головна сторінка для адміністратора має окремий компонент AdminNavBar, так як адміністратор має більший спектр можливостей і неправильно робити великі гілки розгалуження у компонентах. Сама структура використання хуків і компонентів однакова з зареєстрованим користувачем, має однакові назви функціональних компонентів, але знаходиться у папках з файлами для адміністратора і у самих компонентах адміністратора є кнопки для видалення, додавання категорій, колекцій, фільмів. Додавання реалізовано за допомогою відображення модальних вікон MovieCollectionModal, MovieCategoryModal, MovieAddModal. Усі модальні вікна використовують бібліотеку Formik для ратифікації полів і виводу сповіщень про помилки. MC розташований у публічному репозитарії [29].

3.1.2 MC MovieMicroservice

MovieMicroservice є центральним MC, так як він має систему авторизації і БД. Для контролю над сутностями і підсуб'єктностями було розроблено п'ять контролерів: CategoryController, CollectionController, MovieController, UserController. Розроблені контролери є нащадками базового класу Controller і беруть участь у реалізації 19 API контрактів описаних у файлі "api.php". Більшість контрактів мають накладений захист за допомогою створених класів Middleware. Кожному контракту відповідає метод у класі контролера, кожен метод приймає клас нащадок класу Request і відповідає на запит даними у вигляді json за допомогою класів нащадків Resource. У клас контролер проводиться ін'єкція класів сервісів, у яких розташована бізнес логіка. Кожен сервіс має методи по управлінню класом агрегатом, який був створений щойно або витягнутий з БД. Методи сервісу повертають Агрегати або колекції з агрегатами. Агрегат має ряд trait resolvers для ін'єкції у нього інших сервісів і репозитаріїв. Клас сервіс також має ін'єкцію репозитарію і

інших сервісів. Реєстрація усіх ін'єкцій реалізовано за допомогою класів сервіс провайдерів нащадків класу `ServiceProvider` і файлу конфігурації середовища `“app.php”`. Для спілкування з іншими МС у систему встановлено `BestMovieCommon`.

У процесі було розроблено три класи `Middelware`: `AccessMicroserviceTokenCneck`, `AccessTokenCheck`, `AdminRoleValidator`. Для ратифікації спілкування з МС нашої системи порівнюються два хеш токени у класі `AccessMicroserviceTokenCneck`, перший токен визначений у файлі `“.env”` нашого середовища, а другий приходить разом з запитом у хедері `“MICROSERVICE_AUTH_TOKEN”`. Ратифікація зареєстрованого користувача здійснюється у `AccessTokenCheck`. За допомогою методу `“getUserByToken”` ін'єкції `UserTokenService` знаходить користувача по токenu з запиту, потім унікальний ідентифікатор користувача порівнюється з ідентифікатором у запиті. Такий підхід ускладнив можливість злому підбором, у такому виді злому можна підібрати токен або унікальний ідентифікатор, але разом це дуже складно, тому що складність підбору буде тільки збільшуватися, так як кількість користувачів збільшується. Клас `AdminRoleValidator` має перевірку доступу користувача за токеном, унікальним ідентифікатором, роллю, тільки якщо користувач має роль адміністратора він отримає відповідь з статусом у межах 200-299 кодів. Для того щоб відокремити однаковий код перевірки токен було розроблено `trait AuthMiddelwareTrait`.

`UserController` має п'ять методів для попередньої реєстрації користувача, повної реєстрації, авторизації, отримання даних про користувача і оновлення даних користувача. Для реалізації методу попередньої реєстрації контролер ратифікує дані у класі `UserPreRegisterRequest` і визиває метод `“perRegister”` у сервісу `UserService`, передає у нього отриманий адрес електронної пошти, далі сервіс визиває метод `“generateCode”` сервісу `EmailTemplateService` для роботи з `EmailTemplateMicroservice`, він у свою чергу відправляє запит для генерації і

відправки коду на адрес електронної пошти. Метод реєстрації ратифікує дані за допомогою `UserRegistrationRequest`, формує клас `UserCreatePayload` і передає його до методу “create”, що належить `UserService`. Сервіс перевіряє отриманий код за допомогою запиту на `EmailTemplateMicroservice` і якщо код не існує, то буде згенеровано помилку. Після перевірки коду, сервіс створює агрегат `User`, за допомогою репозитарію `UserRepository` зберігає його до БД, а потім додає роль звичайного користувача за допомогою `UserRoleService` [22] – [23]. Сервіс повертає агрегат у якості відповіді до контролеру, коли уся логіка виконана і відомі унікальні ідентифікатори створених сутностей і підсуб’єктностей. Метод авторизації досить простий, після перевірки полів у `UserGetRequest` він використовує сервіс сутності для перевірки її наявності, а сервіс у свою чергу метод репозитарію сутності користувача. У відповідь сервіс повертає агрегат `User`. Процес оновлення користувача дуже схожий на процес створення, але він використовує метод сервісу “update”, цей метод додатково перед оновлення витягує з БД сутність за допомогою методу репозитарію “findUser”. Метод збереження для процесу створення і оновлення у даному випадку є універсальним. Метод “refresh” існує для регулярного оновлення даних користувача на клієнті, який за допомогою методу “getUserByToken” сервісу `UserTokenService` і передачі до нього авторизаційного токена отримує користувача, сервіс у свою чергу використовує метод “getUserByToken” репозитарію `UserTokenRepository`. Усі методи контролера для формування відповіді на запит використовують клас `UserResource` і спеціальну функцію “response->json()”, яка перетворює масиви у json строку. `UserResource` перетворює агрегат у масив з даними за принципом ключ — значення (числове або строкове).

`CollectionController` має залежність від `CollectionService` і реалізовує 4 методи. Перший метод списку приймає запит з умовами фільтрації, на їх основі робить клас фільтр `CollectionFilter` і відправляє його у метод списку сервісу. Сервіс без перетворень надсилає його до методу списку репозитарію `CollectionRepository`, який повертає колекцію агрегатів. Агрегати

повертаються до контролеру, конвертується у потрібний формат завдяки класу ресурсу `CollectionListResource`. Додатково для вибору колекцій за замовченням було розроблено метод `“listOfDefaults”`, результат якого можна покласти до кешу, так як кількість колекцій за замовченням досить невелика, а частота оновлення замала, у разі появи проблем з невідповідністю даних, як механізм оновлення кешу можна розробити подію і обробник, що будуть оброблятися кожен раз, як оновлюється колекція за замовченням. Метод контролеру `“create”` викликає метод створення колекції у сервісу, у якому в залежності від ролі користувача визначається тип колекції. Таким чином, коли адміністратор додає колекцію, їй автоматично виставляється тип `“DEFAULT”`, а якщо це звичайний зареєстрований користувач то тип `“CUSTOM”`, для невизначеної ролі існує тип `“TEST”`. Після визначення типу колекції створюється агрегат і передається до методу `“save”` репозитарію. Останній метод це видалення, разом з запитом приходить унікальний ідентифікатор колекції і користувача, вони передаються до методу сервісу `“deleteWithPermissionCheck”`, де сервіс завдяки ін’єкціям `UserService` і `CollectionRepository` знаходить користувача і в залежності від ролі визначає чи може даний користувач видалити обрану колекцію. Якщо користувач є адміністратором, то сервіс за допомогою `MovieCollectionRepository` видаляє зв’язки між фільмами і колекціями, а потім за допомогою репозитарію сутності видаляє колекцію. Для звичайного користувача перевіряється чи належить колекція користувачу і чи має вона тип `“CUSTOM”`.

`CategoryController` має три методи: список категорії, створення і видалення категорії. Метод списку створює фільтр і передає до сервісу сутності `CategoryService`, який передає фільтр до репозитарію сутності `CategoryRepository`. Для методу створення категорії створюється клас `CategoryCreatePayload`, а для видалення класу `Payload` не існує, так як треба передати тільки унікальний ідентифікатор. Методи не мають складної логіки, окрім процесу видалення, він трохи ускладнений попереднім видаленням

зв'язків між фільмами і категоріями. Усі створені і видалені агрегати передаються на рівень Infrastructure для виконання результуючої дії у БД.

MovieController має 6 методів: списку фільмів, отримання одного фільму, створення, синхронізацію колекцій за замовченням, синхронізацію звичайних колекцій, оновлення категорій. Методи вибірки одного фільму і списку фільмів одноколовий за логічною структурної взаємодії класів з процесами вибірки колекцій або категорій. Процес створення окрім звичайної логіки створення агрегат і його збереження у репозитарії має додаткову перевірку наявності завантажених медіа файлів. Така ратифікація реалізована за допомогою використання методу “validatePath” ін'єкції сервісу BestMovieStorage з PCO BestMovieCommon. Синхронізація звичайних колекцій має складну логіку:

- знаходимо користувача, що синхронізує колекції;
- знаходимо фільм у якого синхронізується колекції;
- знаходимо існуючі колекції по унікальним ідентифікаторами і додатково встановленим фільтром на пошук за типом “CUSTOM”;
- знаходимо усі колекції, що і так зв'язані з фільмом, але не належать користувачу, який синхронізує;
- збираємо у один масив унікальні ідентифікатори з двох отриманих колекцій;
- відправляємо у репозитарій масив колекцій який повинен бути зв'язаний з фільмом і унікальні ідентифікатор фільм.

Результатом даного процесу буде додавання нових зв'язків між колекціями і фільмом, видалення старих зв'язків, що більше не використовуються. Процес синхронізації колекцій за замовченням відрізняється тим, що колекцій з типом “DEFAULT” можуть редагувати усі адміністратори, так само і зв'язок колекції за замовченням з фільмами. Під час оновлення категорій сервіс за допомогою репозитарію перевіряє наявність категорій і додає їх до БД.

Класи репозитарії в основному мають три основні методи: отримання списку за фільтром, отримання однієї сутності за унікальним ідентифікатором, збереження сутності за допомогою агрегату. Дані методи використовують розроблені класи моделі для створення sql запиту за допомогою вбудованого у базову модель конструктору запитів. Таким чином поля фільтра використовується для побудови sql запиту і звертання до БД. В усіх репозитаріях такий метод називається “applyToQuery”. У базову модель також вбудовані метод для отримання сутностей з умовами посторінкової навігації, завдяки цьому деякі поля фільтра використовуються для обмеження кількості і сутностей у списку. Для створення більш складних запитів з конструкціями вигляду “LEFT JOIN”, “RIGHT JOIN”, “JOIN” на базі моделі було розроблено методи, у яких вказують таблицю, і поле, за яким формується зв’язок. Розроблені методи можна використати для розширення sql запитів і оптимізації вибірки. Після проведення запиту до БД отримується колекція моделей, які завдяки розробленим алгоритмам конвертації перетворюються у агрегат. У MovieRepository розроблений метод перетворення фільтру використовує складні конструкції, такі як фільтрування за зв’язком з колекціями і категоріями. Усі репозитарії спілкуються з БД завдяки класам моделям: Category, Collection, Movie, MovieCategory, MovieCollection, Role, User, PersonalAccessToken. Моделі з подвійною назвою відображають таблиці для зв’язку “багато до багатьох”. Вони створені для оптимізації запитів, так як вбудований у платформу клас моделі потребує створення додаткових методів для отримання сутностей і підсуб’єктностей у одному запиті. Наприклад для моделі Movie було розроблено два методи: categories, collections.

Усього розроблено 5 класів агрегатів для роботи з БД: Category, Collection, Movie, User, Role. Агрегат User використовує два trait resolvers, для того щоб мати можливість звертатися до репозитарію сутності користувача і підсуб’єктності ролі. Загалом він має стандартний набір полів (відповідні поля сутності), сетерів, гетерів і спеціальні методи: hasRole,

isViewer, isAdmin, addRole, getRoles. Методи контролюють стан підсуб'єктностей і виконують пошук по цим підсуб'єктностям. Для того щоб отримати ролі, спочатку перевіряється їх наявність у полі і якщо їх немає, то за допомогою trait для підсуб'єктності ролі виконується пошук ролей за унікальним ідентифікатором користувача. Усі отримані ролі є об'єктами з типом Role, який також є агрегатом для підсуб'єктності, це агрегат не має особливих методів, так як бізнес логіка ролі є базовою. Агрегат Movie має три trait resolver, вони надають доступ до репозитарію колекцій, категорій і сервісу MC BestMovieStorage. Movie має декілька спеціальних методів для отримання повного шляху до різних медіа фільму, так як у БД зберігаються лише часові посилання на файли, це зроблено для того щоб можна було легко перейти на інше сховище без додаткових спеціально розроблених міграції для змінення домену у шляху до файлу. Таким чином у ресурсі MovieResource викликається метод для отримання повного шляху. Для оптимізації було розроблено кеш сервіс для сховища, завдяки такому сервісу підвищується швидкість для усіх запитів, пов'язаних з отриманням фільму. Повний шлях до медіа файлів фільму зберігається до кешу на одну годину. Агрегат фільму має метод на отримання власних категорій і колекцій, а також метод, що фільтрує колекції за ідентифікатором користувача. Агрегати колекцій і категорій не мають спеціальної логіки, окрім того, що колекція має методи isCustom, isDefault, isBelongToUser, які повертають відповідь з логічним типом.

Класи провайдери розташовані і папці "Providers", усього розроблено п'ять сервіс провайдерів для процесів основних сутностей. В усіх провайдерах описана реєстрація класів з реалізацією для відповідних інтерфейсів. Додатково в провайдерах використані статичні методи trait resolvers, які підключають класи агрегати до сервісів і репозитаріїв. MC розташований у публічному репозитарії [29].

Також за допомогою вбудованого класу Handler, завдяки якому усі помилки, що виникають у процесі проходять через метод handle. Цей метод

генерує остаточну відповідь на основі отриманої помилки. Метод відправляє у відповіді html сторінку з помилкою за стандартним визначенням, але так як МС реалізують правила REST API специфікації, то було розроблено інший спосіб конвертації помилки. Структура відповіді у випадку помилки представляє собою відповідь у вигляді json, який зберігає масив. У масиву є такі поля: “message”, “error_code”, “error_message”. Для створення кодів помилок було створено перелік ErrorCodes. Усього було визначено способи конвертації помилок для 7 помилок, одна з них є абстрактною для усіх сутностей, це зроблено для зменшення кількості коду. Дерево файлів наведено на рис. 3.2.



Рисунок 3.2 – Дерево файлів МС MovieMicroservice

3.1.3 MC EmailTemplateMicroservice

У даному MC розроблено два API контракти, які реалізовує один контролер CodeContoller і два сервіси: MailService, CodeService. Метод контролеру “generateCode” викликає метод з такою самою назвою у CodeService, цей метод генерує унікальний код на основі 4 випадкових байтів перекодованих у hex, згенерований код зберігається на три хвилини у кеші під ключем зі значенням, яке представляє собою адрес електронної пошти. Після генерації коду контролер викликає метод “sendCodeEmail” у MailService, який за допомогою фасаду Mail з конфігурацією Mailgun відправляє код на електронну пошту користувача. Конфігурація для Mailgun налаштована у файлі “.env”. Другий метод контролеру “getCode” викликає метод з такою самою назвою у сервісу CodeService, який повертає код за кешу. Приклад поштового листа наведено на рисунку 3.8. MC розташований у публічному репозитарії [30].

3.1.4 MC BestMovieStorage

Цей MC створений для завантаження медіа файлів до сховища Firebase. Для цього було розроблено три API контракти і один контролер для їх реалізації MediaContoller. Метод завантаження спочатку перевіряє файл і наявність користувача, його роль. Після перевірки викликає метод “upload” сервісу MediaService, який за допомогою фабрики MediaRestrictionFactory визначає клас валідатор, що треба використовувати для перевірки файлу. Для успішно перевіреного файлу створюється нова унікальна назва і шлях до папки у сховищі в залежності від випадкових бітів, поточної дати, типу файлу. Сервіс викликає MediaStorage для збереження файлу, коли файл перевірено, згенеровано нову назву і шлях у сховищі. Сховище за допомогою фасаду Firebase завантажує файл і повертає посилання на нього. Усього розроблено два класи валідатори: MovieImageRestriction,

MovieVideoRestriction. За допомог вбудованого у платформу валідатора виконуються прості перевірки на розмір, роздільну здатність, mime тип зображення. Додатково було розроблено правило, яке перевіряє саме внутрішній тип файлу, а не вказаний у назві. Таке правило підвищить безпечність звантажених файлів. MC розташований у публічному репозитарії [31].

3.1.5 PCO BestMovieCommon

PCO BestMovieCommon створений на мові PHP без застосування платформи Laravel або інших. Ціль даного пакету розширити наявний Domain рівень у розроблених MC, для того, щоб вони могли спілкуватися один з одним. Завдяки менеджеру Composer можна керувати залежностями даного пакету і надавати можливість вбудовувати його у інші платформи і швидко створити простір імен для пакету. Так як пакет повинен служити одній цілі, бути простим, універсальним і легко розширюватися, то у такому разі він також повинен мати мало залежностей від інших пакетів, щоб бути більш універсальним, тому у обов'язкових залежностях можна побачити лише декілька пакетів: php версії 8.0, який більше вказує на мінімальні потреби пакету, бібліотеку “guzzlehttp/guzzle”. Якщо провести аналіз останнього пакету, то можна побачити, що він оснований на базовому PHP і не має прихованих залежностей від складних пакетів, які можуть бути несумісними з існуючими популярними платформами. Бібліотека “guzzlehttp/guzzle” зроблена на основі базових функцій PHP зі стандартним синтаксисом без використання магічних методів і набором функцій “curl”.

У файлі “composer.json” наявна конфігурація для платформи Laravel, але за потреби туди можна додати конфігурацію для інших платформ. У конфігурації присутні класи сервіс провайдери, які притаманні саме платформі Laravel і допомагають реалізувати шаблон програмування “Ін'єкція залежності”. Для того щоб розробити сервіс провайдери саме для

Laravel на базі класу ServiceProvider було додано “laravel/framework” у залежності розробника. ServiceProvider це базовий і обов’язковий клас платформи, тому не треба додавати його до обов’язкових залежностей.

Додатково в залежностях для розробників вказана платформа PHPUnit це дуже важлива залежність, так як вона дозволяє тестувати пакет без використання платформи. Також для кращого тестування додані бібліотеки “mockery/mockery”, “fakerphp/faker”. Бібліотека “mockery/mockery” дозволяє імітувати запити і відповіді МС. Бібліотека “fakerphp/faker” зробить дані для тестування більш різноманітними і допоможе виявити не відповідності типів або помилки у складних підрахунках.

У PCO реалізована робота з деякими API контрактами EmailTemplateMicroservice, BestMovieMicroservice, BestMovieStorage. Для універсалізації пакету і легкого додавання до нього логіки роботи інших API контрактів було реалізовано клас BaseApi, який має залежність від фасаду конфігурації, класу HttpClient, який є нащадком класу Client з пакету “guzzlehttp/guzzle”. Фасад конфігурації надає доступ до змінних середовища платформи у яку встановлено пакет, таким чином базовий клас може самостійно додавати до запиту хедери потрібні для авторизації і доступу до інших МС проєкту. Клас також має приватне поле “\$baseUrlConfigKey”. Поле повинно мати назву ключа змінної за допомогою якої буде визначено посилання на МС, яке має протокол спілкування “http” або “https” і доменне ім’я МС. За замовченням базовий клас буде використовувати змінні “MS_ENV_KEY” для визначення ім’я масиву з потрібними змінними і “MS_AUTH_ENV_KEY”, яке визначить назву ключа змінної потрібної для авторизації під час спілкування з іншим МС. Завдяки використанню полів з модифікаторами “protected” і “private” така реалізація базового класу надає можливість нащадкам перевизначити усі наявні компоненти для спілкування з МС.

Спілкування по API контракту з EmailTemplateMicroservice реалізовану у класі EmailTemplatApi. Клас має два методи, перший для

створення кодів для підтвердження на пошті, а другий отримання наявних кодів за адресом електронної пошти користувача. Для кожного методу було розроблено класи нащадки `BaseResource`, який має базові методи декодування відповіді від МС у вигляді json. Класи нащадки мають уявлення про структуру відповіді, ключів і типів змінних, відповідні методи для отримання конкретних даних з відповіді МС. Архітектура спілкування для `BestMovieMicroservice`, `BestMovieStorage` реалізована так само, як для `EmailTemplateMicroservice`. Сервіс для `BestMovieStorage` має реалізацію спілкування по API контракту для отримання повного шляху до завантаженого файлу по його частковому посиланню і ратифікувати наявність файлу теж по частковому посиланню. Посилання містить шлях і назву до файл за ієрархію папок сховища. Для `BestMovieMicroservice` реалізовано лише один API контракт, він робить запит на отримання інформації про користувача.

Для того щоб автоматично підключити розроблені сервіси було створено сервіс провайдери на основі сервіс провайдеру платформи Laravel. Усі розроблені сервіс провайдери приймають участь у ін'єкціях розробленого класу API і сервісу, але з одною особливістю, при ін'єкції до класу с реалізацію API треба передати ключ змінної, за якою буде виконано пошук у даних середовища з конфігураційних файлів та файлу `“.env”`. Для `BestMovieApi`, `BestMovieStorageApi`, `EmailTemplateApi` було визначено наступні ключі `“BEST_MOVIE_MS”`, `“BEST_MOVIE_STORAGE_MS”`, `“EMAIL_TEMPLATE_MS”`. Слід зазначити що кожному сервісу не обов'язково визначати усі розроблені ключі, так як сервіс провайдер створює мапу ін'єкцій і тільки при спробі використати сервіс або клас з API реалізацію треба обов'язково визначити змінну, інакше є вірогідність отримати незрозумілу критичну помилку не пов'язану з сервіс провайдером або змінними середовища.

Даний PCO має 23 версії, остання версія `“v0.1.2”`, контроль версій здійснено за допомогою Packagist [32]. Для реєстрації пакету у бібліотеці

пакетів він має бути розташований у репозитарії для перевірки на безпечність [33]. Пакет встановлено у три МС: EmailTemplateMicroservice, BestMovieStorage, MovieMicroservice. Дерево файлів МС наведено на рис. 3.3.

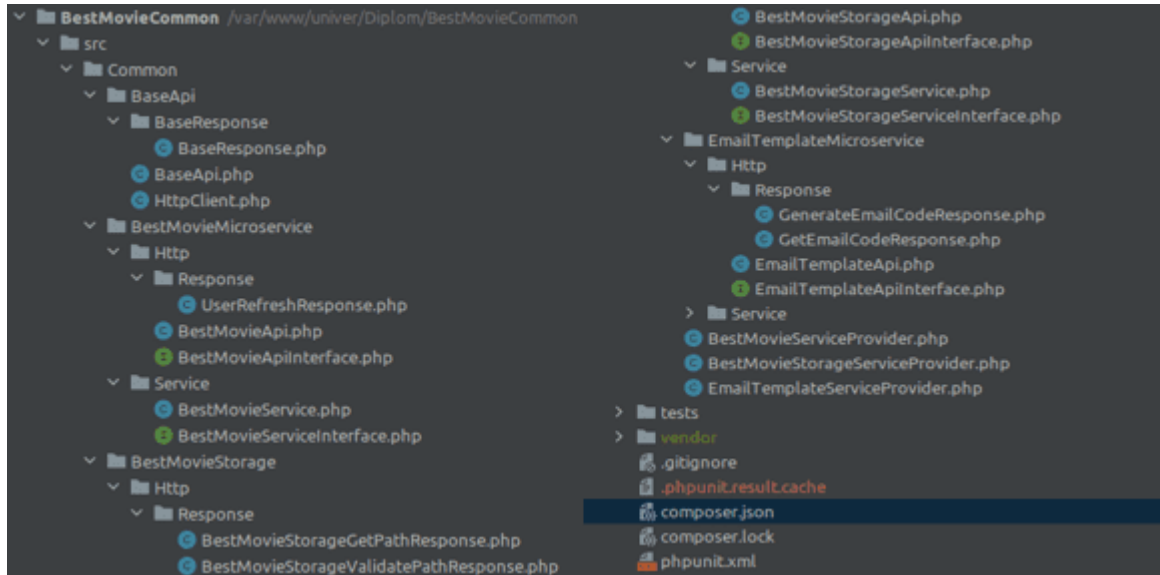


Рисунок 3.3 – Дерево файлів МС BestMovieCommon

3.2 Основні рішення щодо розробки інтерфейсу

За допомогою React, HTML і CSS було розроблено два окремі дерева сторінок для адміністратора і користувачів з іншими ролями. Головна сторінка користувача наведена на рис. 3.4., сторінка адміністратора на рис. 3.5. Для додавання нового фільму у панелі адміністратора було використано модальне вікно. Також модальні вікна було використано при додаванні і видаленні категорій і колекцій, при утворенні зв'язку між фільмом і колекцією, категорією. Слід зазначити, що по можливості було використано відсоткові значення для відступів, статичні значення мають лише невеликі об'єкти. Таким чином це позитивно вплинуло на можливості змінювати роздільну здатність екрану. Усі кнопки для створення або додавання мають зелений колір, а для видалення червоний. Загалом тема сайту темна і створена на основі цих кольорів: “#222d33”, “#0c1e25”, “#001219”.

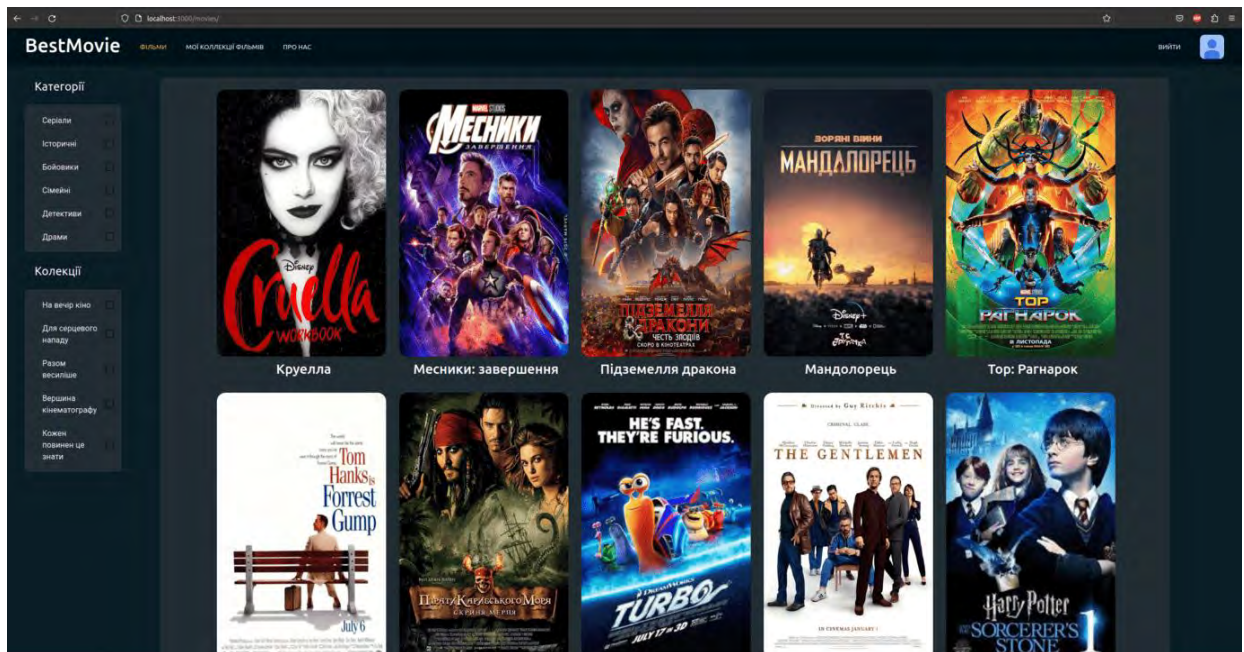


Рисунок 3.4 – Головний екран звичайного користувача

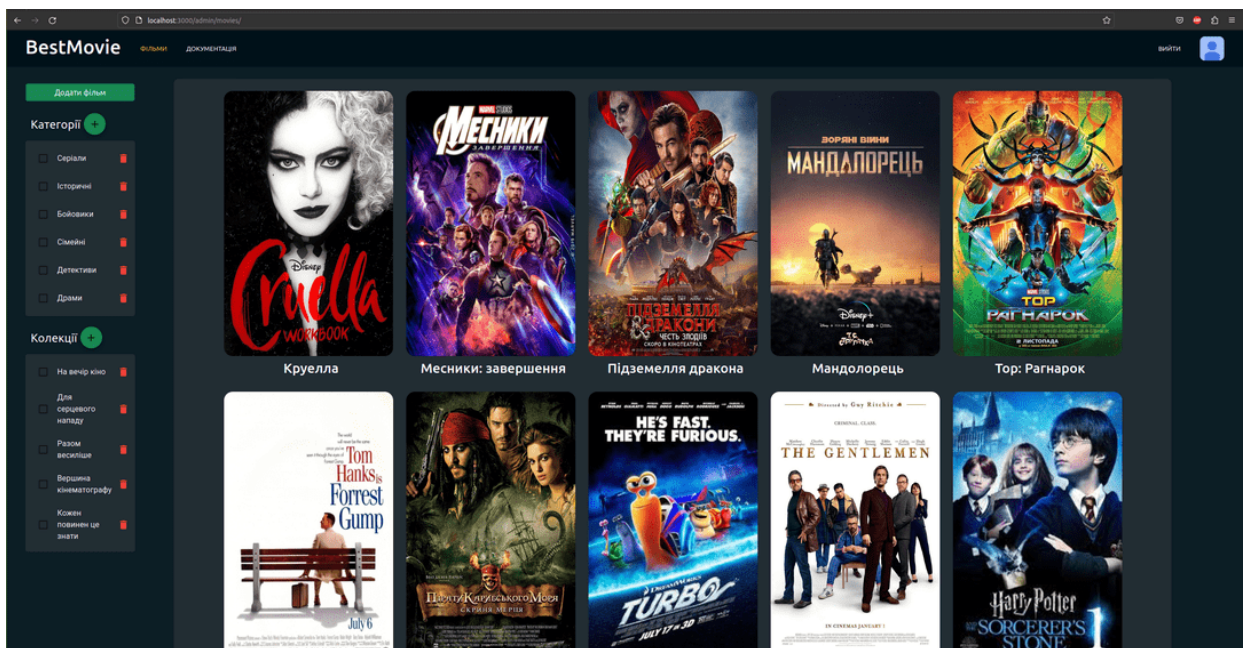


Рисунок 3.5 – Головний екран адміністратора

3.2.1 Вибір додаткових бібліотек

Було використано компоненти бібліотеки “react-bootstrap”, “mui/material”, Formik, Bootstrap. З “react-bootstrap” використано компоненти Row, Col, Modal для створення структури і спливаючих вікон, це дозволило

не вказувати точні розміри і стилі. З бібліотеки “mui/material” було використано компонент Pagination, для створення посторінкової навігації. Зарядки компонентам List, ListItem, Checkbox, ListItemButton було створено списки колекцій і категорій.

3.2.2 Основні рішення щодо впорядкування інтерфейсу

У шапці сайту три вкладки ліворуч: “Фільми”, “Мої колекції”, “Про нас”. Праворуч розташована вкладка “Зареєструватися” або “Увійти” (у залежності від стану користувача. Загалом шапка і контент сторінки вирівняно за допомогою звичайних файлів зі стилями.

На вкладці “Фільми” горизонтально у декілька рядів (кількість рядів залежить від розміру екрану) розташовано 10 фільмів, сторінку наведено на рис. 3.4. Фільми представлені афішою і назвою, що розташована під афішою. Фільтри для категорій і колекцій за замовчення розміщено ліворуч, фільтри не є взаємовиключними. У списках з фільтрами максимально розташовується від 10 до 15 категорій і колекцій в залежності від розмірів назв, усі що не попали у поле зору доступні за допомогою горизонтальної прокрутки. Списки відокремлені кольоровими блоками і назвами списків у вигляді заголовків. Посторінкова навігація розташована у низу сторінки і може мати до 10 номерів сторінок, для перемикання сторінки можна використовувати стрілки або номери.

При натисканні на афішу фільму, користувача переносить до сторінки з обраним фільмом, сторінку наведено на рис. 3.6. На ній розташована інформація про кіно, афішу і відео. Інформація представлена у вигляді таблиці і знаходиться праворуч від афіши. Відео розташовано під таблицею з інформацією і афішою. У правому куті відцентрованого блоку з інформацією знаходиться кнопка “Додати до колекції”. При натисканні на кнопку з’являється модальне вікно (вікно наведено на рис. 3.7) розташоване по правому краю, воно має список з власними колекціями користувача (старі

колекції у стані обраних) і стислу інформацію про фільм. Праворуч у горі є кнопка закриття вікна, а у низу дві кнопки “Зберегти зміни” і “Закрити”. У списку колекцій можна обирати декілька колекцій.

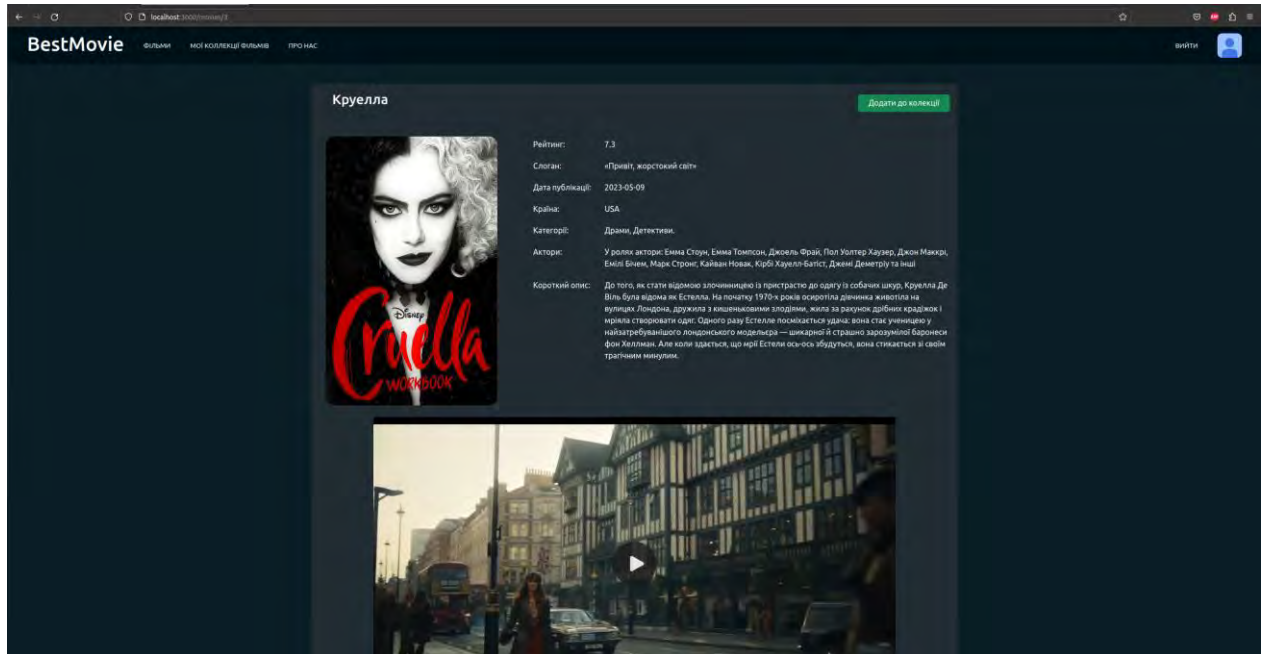


Рисунок 3.6 – Сторінка з фільмом

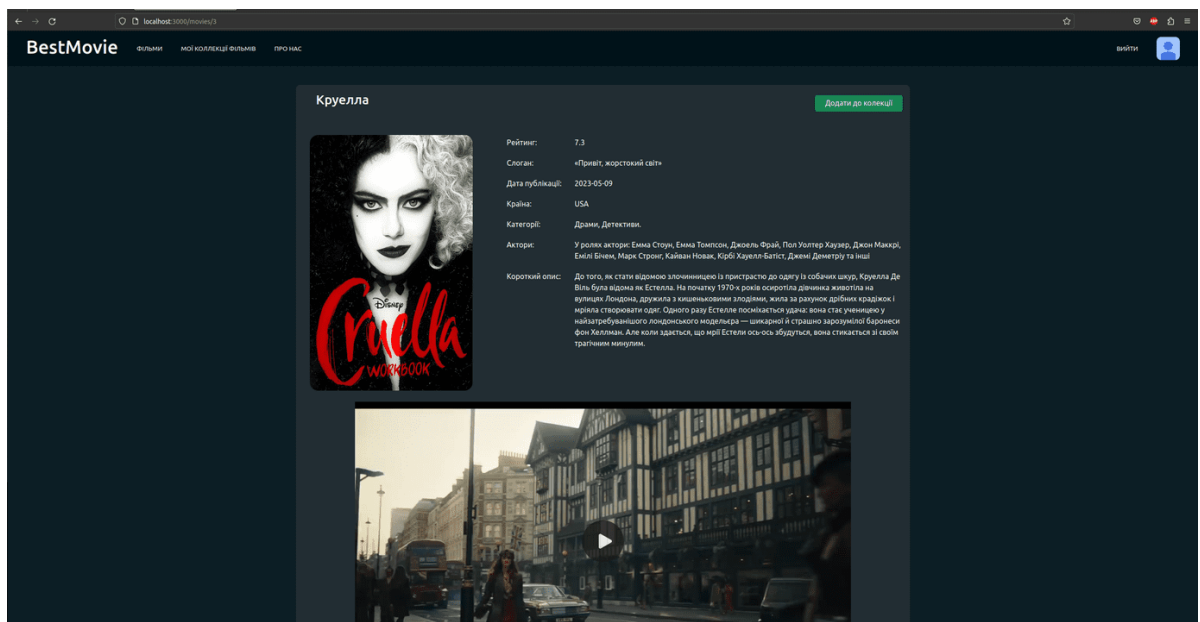


Рисунок 3.7 – Модальне вікно додавання колекції

Сторінка “Про нас” має опис, що це дипломний проект, розроблений виключно у дослідницьких цілях, сторінку наведено на рис. 3.8.

Форми для реєстрації і авторизації розташовані на окремих сторінках і мають великий логотип сайту і форми для введення даних. На сторінці реєстрації, яка наведена на рис. 3.9, розташовані поля для паролю, адресу електронної пошти, ім'я прізвища, у низу присутні дві кнопки: “Зареєструватися” і “Увійти”. Після натискання на кнопку “Зареєструватися” користувач повинен ввести спеціальний код, що було надіслано на пошту (електронний лист наведено на рис. 3.10), тільки після підтвердження за кодом, користувач остаточно зареєстрований. Якщо натиснути на кнопку “Увійти” то користувача перенесе на сторінку “Авторизації”, її наведено на рис. 3.11. На формі для авторизації розташовані поля для паролю, адресу електронної пошти і дві кнопки “Зареєструватися” і “Увійти”. Після натискання кнопки “Увійти” і успішної перевірки, користувача переносити на головну сторінку.

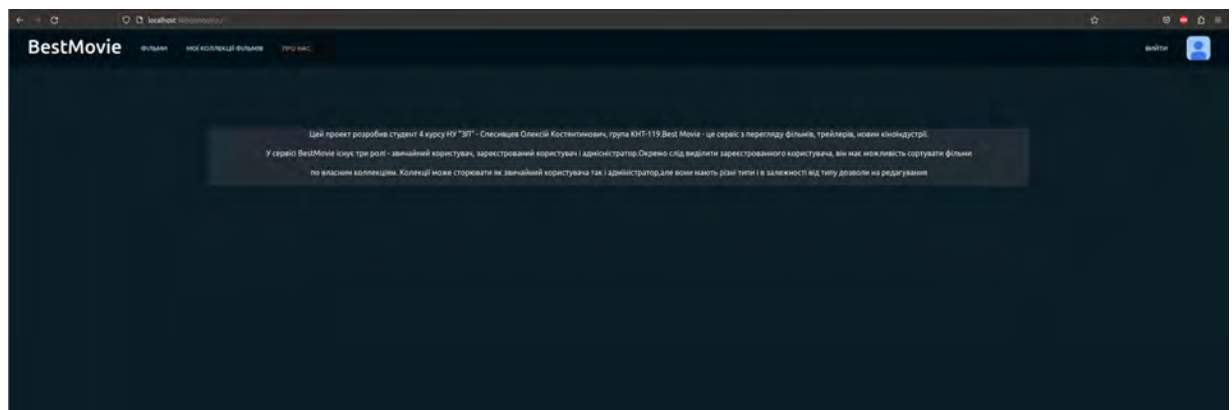


Рисунок 3.8 – Сторінка “Про нас”

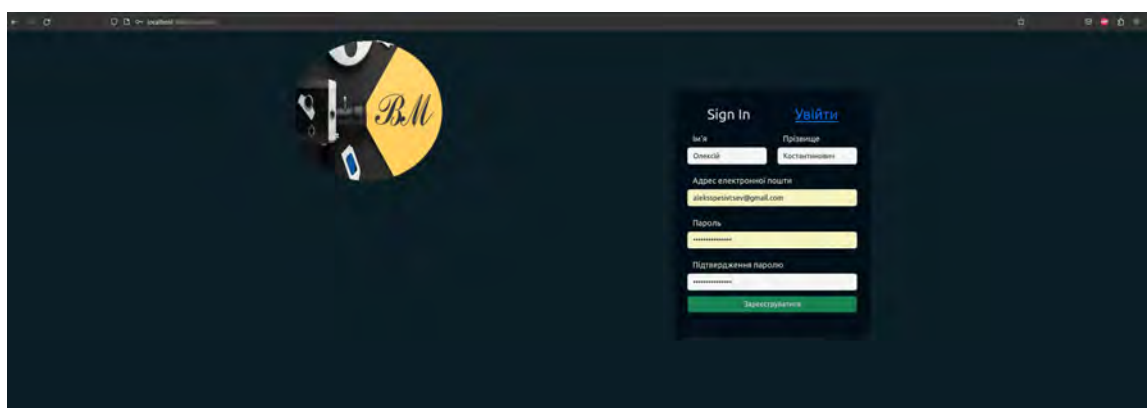


Рисунок 3.9 – Сторінка реєстрації



Рисунок 3.10 – Електронний лист з кодом

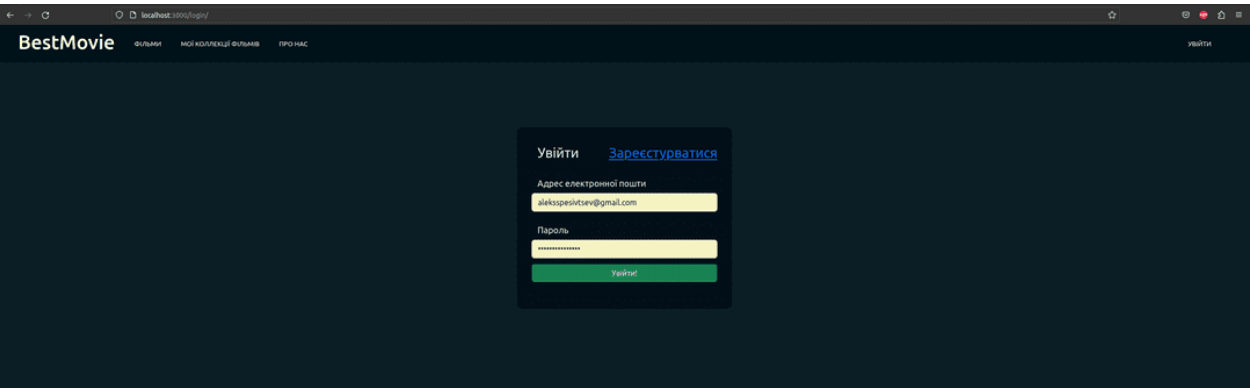


Рисунок 3.11 – Сторінка авторизації

Головна сторінка адміністратора наведена на рис. 3.5 і додатково до стандартних функцій має кнопки для додавання і видалення категорій, колекцій за замовченням. Модальні вікна, які з’являються при натисканні на кнопки для категорій і колекції наведені на рис. 3.12 і 3.13. На сторінці також присутня кнопка для додавання фільму - “Додати фільм”, при натисканні на кнопку з’являється вікно у якому є поля для додавання афіши, фільму, назви, рейтингу, країни, слогану, акторів, опису, категорій і дві кнопки “Додати” і “Видалити”, модальне вікно додавання фільму наведено на рис. 3.14.

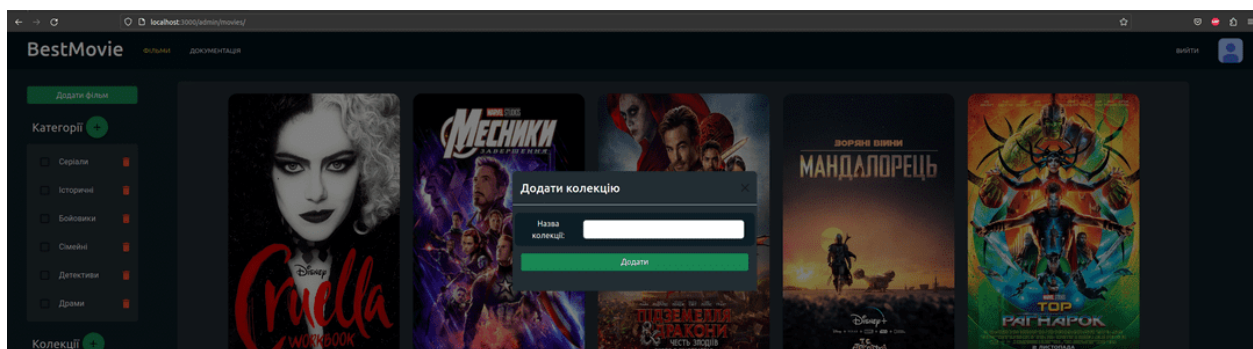


Рисунок 3.12 – Модальне вікно додавання колекції

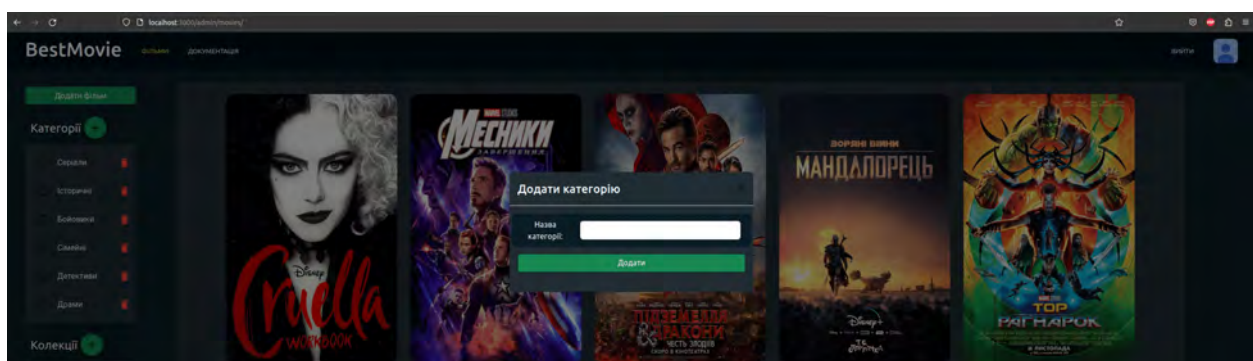


Рисунок 3.13 – Модальне вікно додавання категорії

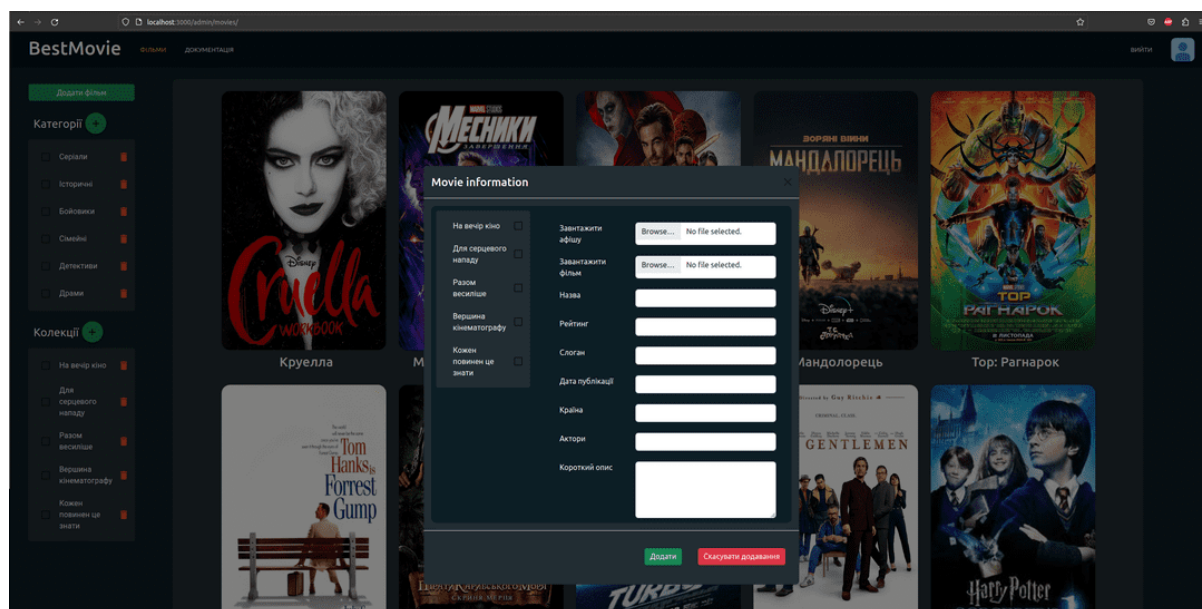


Рисунок 3.14 – Модальне вікно додавання фільму

На сторінці з конкретним фільмом є кнопки “Видалити”, “Додати до колекції” (наведено на рис. 3.15), “Додати до категорії” (наведено на рис. 3.16). Форма для додавання колекцій і категорій однакова з формою

додавання колекцій для звичайного користувача, але на ній розташовані лише колекції за замовченням. Окрім додавання є функція видалення колекцій і категорій, модальне вікно для підтвердження видалення наведено на рис. 3.17.

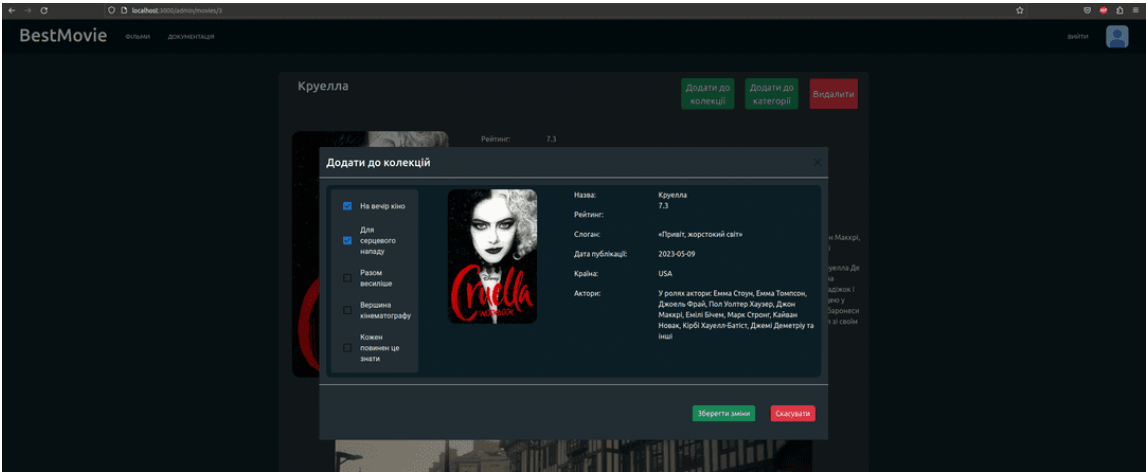


Рисунок 3.15 – Модальне вікно для додавання фільму у колекції

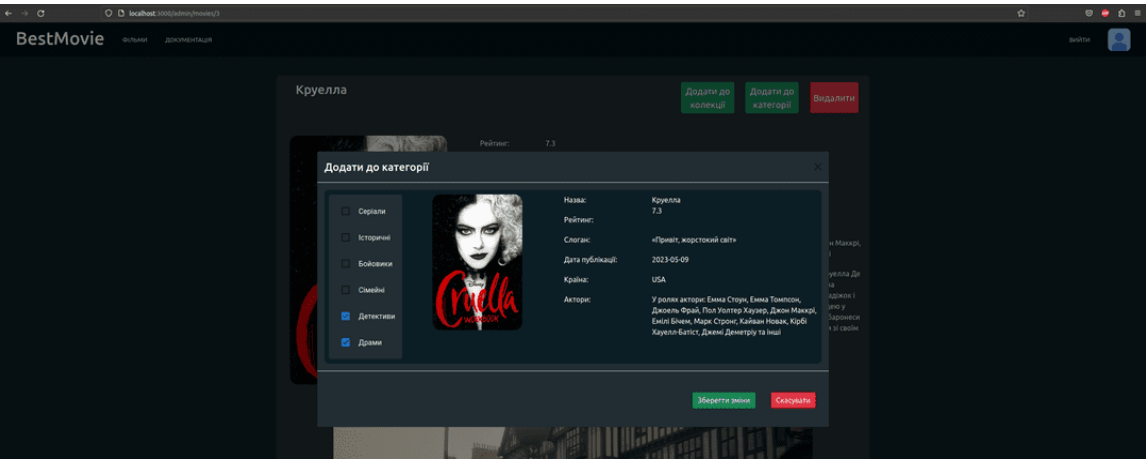


Рисунок 3.16 – Модальне вікно для додавання фільму у категорії

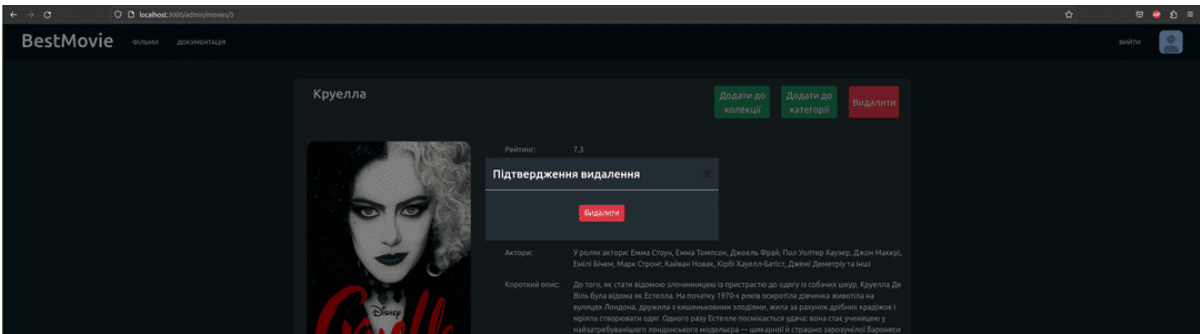


Рисунок 3.17 – Модальне вікно для підтвердження видалення

З ціллю абстрагування від розробленого інтерфейсу було створено схему з описом функцій різних користувачів (рис. 3.18).

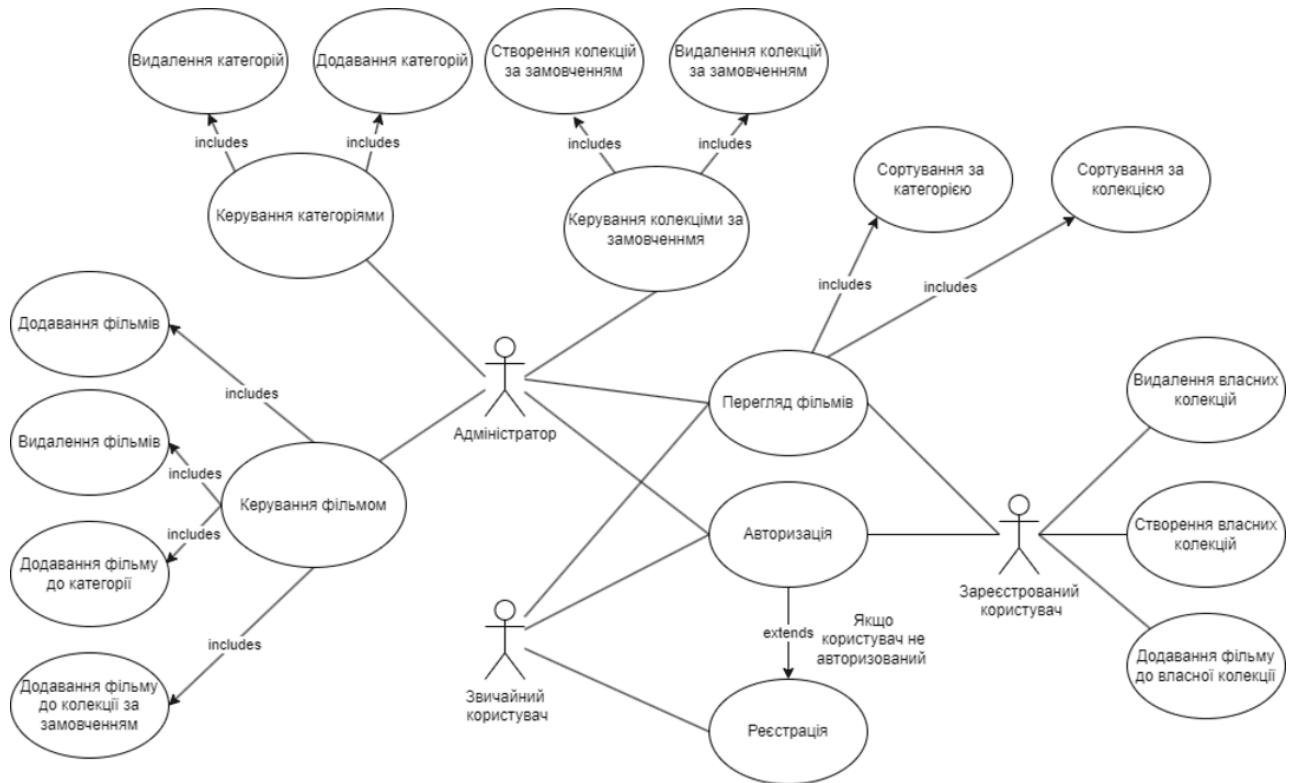


Рисунок 3.18 – Функції користувачів

3.2.3 Основні рішення з комунікації з користувачем

Кожна афіша має анімацію збільшення при наведенні на неї. Усі вкладки у шапці сайту мають специфічну анімацію при наведенні — спочатку назва піднімається до гори, підкреслюється знизу, міняє колір тексту на голубий. Якщо користувач знаходиться на вкладці фільми, то вкладка “Фільми” виділяється золотим коліром, таку саму логіку мають інші вкладки, приклад наведено на рис. 3.5. Усі кнопки мають анімацію освітлення і збільшення при наведенні. Також кнопки видалення мають вікно з підтвердженням видалення.

3.3 Основні рішення щодо розробки БД

Тільки у MovieMicroservice має БД, інші МС використовують тимчасові сховища, а саме Laravel кеш у EmailTemplateMicroservice і локальне сховище у BestMovieClient. У ролі основної СКБД було обрано MySQL 8.0.32.

Для створення таблиць було розроблено ряд міграції [34] – [35] (перелік міграції наведено на рис. 3.19.), зміст яких оснований на готових конструкторах таблиць у платформі Laravel. За допомогою класів Migration, Schema, Blueprint було розроблено прості міграції, що створюють поля з числовим і строковими типами, індекси, первинні, зовнішні і складові ключі. Кожна міграція може мати механізм знижування до попереднього рівня (з англ. rollback) міграції, тому додатково було визначено послідовність такого знижування до попереднього рівня.

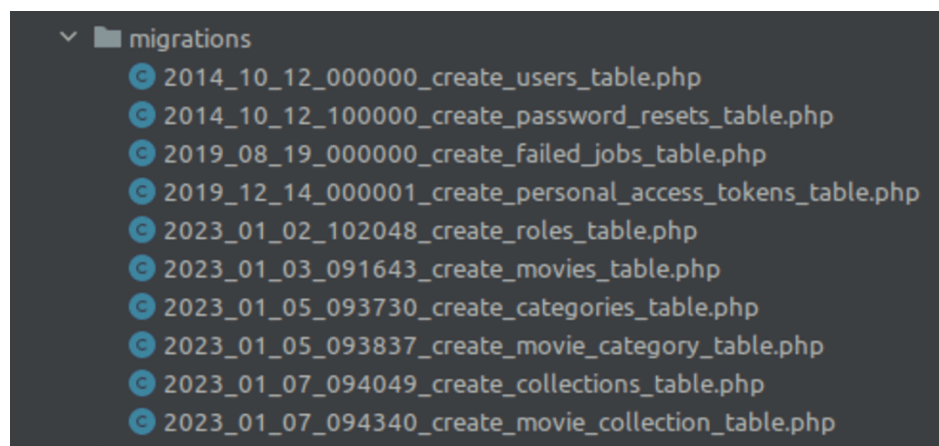


Рисунок 3.19 – Перелік розроблених міграцій

Завдяки наявним міграціям можна легко відслідковувати у якому стані перебуває наявна БД і легко контролювати її стан. Також наявність точних міграції надає можливість розробити інтеграційні тести, які будуть використовувати тимчасову БД для тестування з управлінням через СКБД SqlLite. Слід зазначити, що SqlLite немає багатьох можливостей MySQL, такі як пошук по json полям, складні нативні sql-запити у міграціях, регулярні

вирази, підтримка складних дат, утримання великих даних, якщо наприклад у БД зберігаються файли більш ніж 32 тис. kb.

Файл міграції можна зробити вручну, але це не рекомендується робити, так як при створення через команду `“php artisan make:migration --name”` сама платформа додає до назви точний час створення міграції і додатково перевіряє на унікальність імені такої міграції, для того щоб не було дублюванні міграції за смисловим змістом. Як результат створення міграції у такий спосіб є зручний перегляд папки з міграціями, так як вони впорядковані за датою створення. Слід зазначити, що якщо створити міграцію вручну і допустити помилку, то можна зіштовхнутися з такими проблемами: неможливо розгорнути проєкт, помилки під час ревізії пакетів, неможливо встановити, оновити пакети, створити кеш файлів, провести тести.

3.3.1 Розроблена БД

Розроблена БД має 11 таблиць і 2 види зв'язків між таблицями, один до багатьох і багато до багатьох. Дві таблиці створенні платформою за замовченням для внутрішніх процесів платформи. Таблиці за замовченням це `“failed_jobs”` і `“migrations”`, перша потрібна для черг, а друга для контролю проведення міграцій. Контроль проведення міграцій здійснюються шляхом додавання міграцій у таблицю під час проведення нової міграцій.

Головна сутність системи це користувач, тому йому відповідає центральна таблиця `“users”`. Таблиця має такі поля: унікальний ідентифікатор, ім'я, прізвище, адрес електронної пошти, час верифікації імейлу, пароль у вигляді хешу, резервний токен, час створення і оновлення. Таблиця `users` явно пов'язана зв'язком один до багатьох з таблицею `“user_roles”` і `“collections”`, і не явно з `“personal_access_token”` і `“password_resets”`.

Таблиця “user_roles” є підсуб’єктністю таблиці користувачів і пов’язана з користувачем зв’язком один до багатьох, вона зберігає усі ролі користувача і має такі поля: унікальний ідентифікатор, ідентифікатор користувача, роль, час створення і оновлення. Таблиці “personal_access_token” і “password_resets” є підсуб’єктностями сутності користувача за допомогою морфованого ключа, він формується на рівні платформи і пов’язує користувача з персональними токенами доступу і таблицею з попередньо використаним паролем. Такі таблиці мають не явний зв’язок один до багатьох і один до одного для старого паролю. Таблиця “personal_access_token” має такі поля: унікальний ідентифікатор, тип токена, унікальний ідентифікатор токена, можливості доступу, дату останнього використання, дату придатності, час створення і оновлення. Таблиця “password_resets” має поля: адрес електронної пошти, токен, дату оновлення, унікальним ідентифікатор виступає адрес електронної пошти. Слід зазначити, що ці таблиці створені з такою структурою для стабільної роботи бібліотеки Sunctum додатково встановленої до платформи.

Таблиця “collections” є окремою сутністю, хоча вона не може існувати без користувача, але за бізнес логікою вона може бути колекцією за замовченням доступ до якої буде у багатьох користувачів. Сутність колекції має наступні поля: унікальний ідентифікатор, ідентифікатор користувача, тип, ім’я, час створення і оновлення.

Друга центральна сутність системи представлена таблицею “movies”, яка пов’язана зв’язко багато до багатьох таблицями “categories” і “collections” за допомогою таблиць “movie_category” і “movie_collection”, категорії є підсуб’єктностями фільму. Сутність фільм має наступні поля: унікальний ідентифікатор, ім’я, опис, посилання на файл фільму, посилання на файл рисунку до фільму, час створення і оновлення. Таблиця “movie_collection” має такі поля: унікальний ідентифікатор, ідентифікатор фільму, ідентифікатор колекції, час створення і оновлення. Таблиця “movie_category” має наступні поля: унікальний ідентифікатор, ідентифікатор фільму,

ідентифікатор категорії, час створення і оновлення. Підсуб'єктність категорія має поля: унікальний ідентифікатор, ім'я, час створення і оновлення. Схема БД наведена на рис. 3.20.



Рисунок 3.20 – Схема таблиць БД і зв'язків між ними

3.4 Висновки з розділу

У даному розділі були розглянуті основні рішення, включаючи застосовані шаблони програмування, розроблені МС, БД, сторінки інтерфейсу користувача, а також розроблені алгоритми. При виборі рішень та шаблонів програмування було враховано цілі проекту, його масштаб, мету дослідження. Важливою складовою була інтеграція та взаємодія з іншими

технологіями. Було забезпечено сумісність та зручну інтеграцію між розробленими компонентами та іншими елементами системи, такими як інші сервіси, сторонні системи зберігання даних і відправки поштових листів. Це дозволило створити цілісну систему з високим рівнем функціональності та ефективною взаємодією.

При роботі над системою була приділена увага не лише розробці окремих елементів, але й їх взаємодії та оптимізації для досягнення більш ефективного функціонування системи в цілому.

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРЕМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

У даному розділі розглянутий алгоритм розгортання мікросервісів і основні умови їх експлуатації, а також проведене тестування і дослідження зручності інтерфейсу користувача, сумісність МС з розробленими РСО.

4.1 Застосування системи

Система призначена для перегляду, завантаження фільмів, афіш і зручного менеджменту власних колекцій. Основна направленість це звичайні користувачі і адміністратори, але систему можуть використовувати юридичні особи, такі як кінотеатри, для того щоб створювати власні колекції і у подальшому ділитися ними з користувачами з преміум пакетами або особливими ролями. Розвиток даної системи необмежений і неускладнений, так як у фундамент закладена архітектура DDD. Для розширення системи на мобільні застосунки або десктоп застосунки достатньо лише створити нові МС з рівнем “User interface”.

4.2 Розгортання і експлуатація системи мікросервісів

Система була розроблена і розгорнута у ОС (операційна система) Linux Mint, тому усі наведені команди і інструкції актуальні лише для неї. Для розгортання трьох МС (MovieMicroservice, EmailTemplateMicroservice, BestMovieStorage) потрібне встановлення і налаштування Nginx, PHP 8.0, PHP-FPM, MySQL, Composer [37]. Команда для встановлення Nginx – “sudo apt install nginx”, встановлення PHP – “sudo apt-get install php8.0”, встановлення PHP-FPM – “sudo apt-get install php8.0-fpm”, встановлення Composer – “sudo apt install composer”. Для встановлення MySQL треба виконати ряд команд:

- “sudo apt-get install mysql-server” (встановлення MySQL);
- “mysql -u root” (відкриття MySQL);
- у консоль MySQL ввести – “SET PASSWORD FOR 'root'@'localhost' = PASSWORD('4as6iKILL47<>and06052002alex');”;
- у консоль MySQL ввести – “CREATE DATABASE diplom_movie;”.

Після встановлення необхідним компонентів треба створити конфігураційні файли для серверів MC, зміст файлів наведено у додатку Б. Треба створити три файли у папці “/etc/nginx/sites-available” і вставити у них текст з додатку, після цього ввести у консоль три команди: “sudo ln -s /etc/nginx/sites-available/diplomEmailTemplate /etc/nginx/sites-enabled/”, “sudo ln -s /etc/nginx/sites-available/diplom /etc/nginx/sites-enabled/”, “sudo ln -s /etc/nginx/sites-available/diplomBestMovieStorage /etc/nginx/sites-enabled/”. Фінальна команда, яка розгорне сервери з MC це – “sudo service nginx restart”. Після треба перевірити стан Nginx за допомогою такої команди — “sudo service nginx status”, у консоль повинен з’явитися зелений текст “Active”. Після MC доступні за такими посиланнями у браузері: “http://diplomEmailTemplate”, “http://diplombestmoviestorage”, “http://movie/”. Якщо при звертанні до MC на головній сторінці з’явилася конфігурація Apache, то треба ввести в консоль такі команди – “sudo service apache2 stop”, “sudo service nginx restart” і перезавантажити сторінку. Якщо на сторінці помилка з текстом “Need to generate key”, необхідно відкрити у консолі корінну папку з MC і ввести команду – “php artisan key:generate”. Також перед користувачем може з’явитися пустий екран, у такому разі треба перевірити віртуальний сервер за замовченням за посиланням “127.0.0.1”, якщо за цим посиланням теж пустий екран, то треба по черзі відключати MC до моменту поки по посиланню за замовченням не з’явиться екран з успішним розгортанням віртуального серверу за замовченням. Після визначення проблемних MC треба відкрити у консолі корінну папку і ввести команду “composer dump-autoload”, якщо у консолі з’явиться одна з помилок – “Service Provider enabled”, “pdo exception”, необхідно перевірити наявність

файлу “.env” (вигляд файлу наведено у додатку Б) або встановити додаткове розширення за допомогою команди “sudo apt-get install php8.0-pdo”.

Для MC BestMovieClient треба встановити NodeJS і npm, команда для встановлення NodeJS – “sudo apt-get install nodejs”, для npm — “sudo apt get install npm”. Далі треба перейти до корінної папки BestMovieClient і ввести команду “npm start”. Розгортання інтерфейсу готове і доступне за посиланням – “http://localhost/movies/”. Слід зазначати, що при завантаженні проєкту з репозитаріїв, треба обов’язково прописати команду “composer install” у корінних папках MC (MovieMicroservice, EmailTemplateMicroservice, BestMovieStorage). А для MC BestMovieClient треба прописати команду “npm install”. Після виконання усіх інструкцій при перезавантаженні фізичного серверу, буде треба лише перейти до корінної папки BestMovieClient і ввести команду “npm start”. У реальному середовищі можна додатково налаштувати віртуальний сервер Node на автоматичний запуск після перезавантаження фізичного серверу. Таким чином система є автономною, так як не потребує постійних утручань у перезапуск серверів або інші вузли системи. У результаті розгортання системи є одий фізичний сервер, 4 логічних, які пов’язані з двома глобальними логічними серверами (Firebase, Mailgun). отримана мережа указана на рис. 4.1.

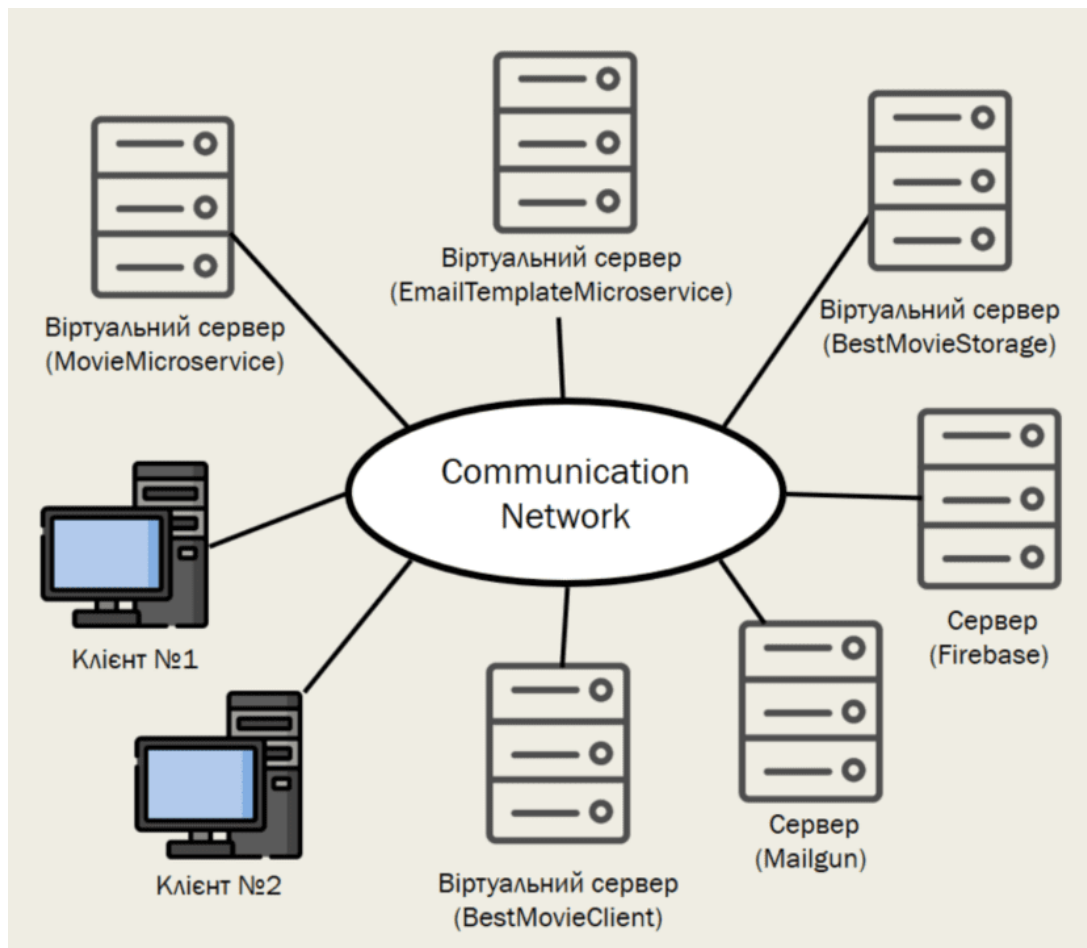


Рисунок 4.1 – Мережа мікросервісів

4.3 Тестування та дослідження сумісності проєкту

Так як система реалізовує інтерфейс спілкування REST API і усі MC є модулями зі своїми API контрактами, тому просту бізнес логіку можна тестувати окремо від клієнтської частини і інших MC. Таке тестування проведено за допомогою Postman, це платформа, яка надає можливості створювати і відсилати запит на сервер за допомогою зручного інтерфейсу, запити можуть мати параметри, тіло, заголовки. Також платформа надає можливість зробити запис у вигляді “curl”, що є дуже зручним інструментом при створенні документації, або тестуванні на системах Linux, так як з консолі даних ОС можна відсилати запити представлені структурою “curl” [36]. Протестовані API контракти і результати їх тестування наведено у таблиці 4.1.

Таблиця 4.1 – Тестування MC MovieMicroservice

API контракти ("baseUrl" - "http://movie")	Позитивний тест	Негативний тест	Результати
1	2	3	4
GET "{baseUrl}/user/" авторизація	Отримання даних користувача	Не існуючий користувач	+ / +
POST "{baseUrl}/user/"	Реєстрація. Створення користувача в БД з роллю глядача	Надіслані неправильні дані або EmailTemplateMicro- service не відповідає	+ / +
GET "{baseUrl}/movie/list"	Отримання списку фільмів	Надіслані неправильні дані для фільтрації	+ / +
GET "{baseUrl}/movie/ {id}"	Отримання даних про фільм	Не існуючий фільм	+ / +
PATCH "{baseUrl}/user/"	Оновлення даних користувача	Надіслані неправильні дані або користувача не існує	+ / +
POST "{baseUrl}/user/pre- regitser"	Відправка електронного листа на пошту користувача	Надіслані неправильні адрес електронної пошти	+ / +
PATCH "{baseUrl}/movie/ collections/{id}"	Оновлення списку колекції фільму	Користувача не авторизований	+ / +

Продовження таблиці 4.1

1	2	3	4
POST “{baseUrl}/movie/”	Оновлення даних фільму	Неправильні дані у запиті або фільму не існує	+ / +
PATCH “{baseUrl}/movie /collections /default/{id}”	Оновлення списку колекції за замовченням фільму	Користувача не має ролі адміністратора	+ / +
PATCH “{baseUrl}/movie/ categories/{id}”	Оновлення списку категорій фільму	Користувача не має ролі адміністратора	+ / +
DELETE “{baseUrl}/category/{ id}”	Видалення категорії	Категорії не існує	+ / +
POST “{baseUrl}/category/”	Оновлення категорії	Надіслані неправильні дані або категорії не існує	+ / +
GET “{baseUrl}/category/li st”	Отримання списку категорій	Надіслані неправильні дані для фільтрації	+ / +
DELETE “{baseUrl}/movie/ {id}”	Видалення фільму	Користувача не має ролі адміністратора або фільму не існує	+ / +
GET “{baseUrl}/collection/ list”	Отримання списку колекцій	Надіслані неправильні дані для фільтрації	+ / +

Продовження таблиці 4.1

1	2	3	4
GET “{baseUrl}/collection/list/ defaults”	Отримання списку колекцій за замовченням	Надіслані неправильні дані для фільтрації	+ / +
POST “{baseUrl}/collection/”	Оновлення колекції	Надіслані неправильні дані або колекції не існує	+ / +
DELETE “{baseUrl}/collection/{id}”	Видалення колекції	Колекції не існує	+ / +

За результатами тестування усі розроблені API контракти MC MovieMicroservice функціонують у межах визначеної логіки і мають обробку реакції на помилки під час перевірки запиту або у процесі виконання бізнес логіки. Результати тестування MC BestMovieStorage і MC EmailTempaleMicroservice наведено на таблиці 4.2. і 4.3.

Таблиця 4.2 – Тестування MC BestMovieStorage

API контракти (“baseUrl” - “http://movie”)	Позитивний тест	Негативний тест	Результати
1	2	3	4
POST “{baseUrl}/media- uploader/”	Завантаження фільму до Firebase	Фільм не відповідає вимогам	+ / +
GET “{baseUrl}/media- uploader/”	Отримання посилання на фільм	Такого фільму не існує у сховищі	+ / +

Продовження таблиці 4.2

1	2	3	4
GET “{baseUrl}/media-uploader/validate”	Перевірка існуючого фільму	Такого фільму не існує	+ / +

Таблиця 4.3 – Результати тестування MC EmailTempaleMicroservice

API контракти (“baseUrl” - “http://diplombest-moviestorage”)	Позитивний тест	Негативний тест	Результати
1	2	3	4
POST “{baseUrl}/email-codes/”	Генерація коду на три хвилини і відправка на електронну пошту користувача	Надісланий неправильний адрес електронної пошти	+ / +
GET “{baseUrl}/email-codes/”	Отримання коду з кешу	Авторизаційного коду немає	+ / +

Тестування MC BestMovieStorage і EmailTempaleMicroservice теж має задовільні результати, хоча основна частина перевірки припадає на непідконтрольні сервіси, такі як – Fireabse, Mailgun, а це робить логіку тестування розмитою. Розмиття логіки полягає у тому, що немає повної документації для виникаючих помилок під час спілкування з даними технологіями. Такі технологій, незважаючи на бракування документації, є дуже корисними, хоча це цілісний продукт, розробку якого не можливо контролювати, такий аспект накладає певні межі при його використанні. Такі межі можуть бути виражені у структурі або безпеці збереження даних. Слід зазначити, що компанія AWS частково вирішила це питання створенням консолі з власним програним синтаксисом для контролю безпечності і

доступу до функції редагування файлів, але це рішення не вирішило усі проблеми і є досить не зручним, навіть для розробників. Перелік розроблених тестів за допомогою Postman наведено на рис. 4.2. Тестування складної бізнес логіки було проведено за допомогою розробленого інтерфейсу, так як при користуванні системою визивається ряд запитів різних МС. Протестовані процеси і результати наведено у табл. 4.4.

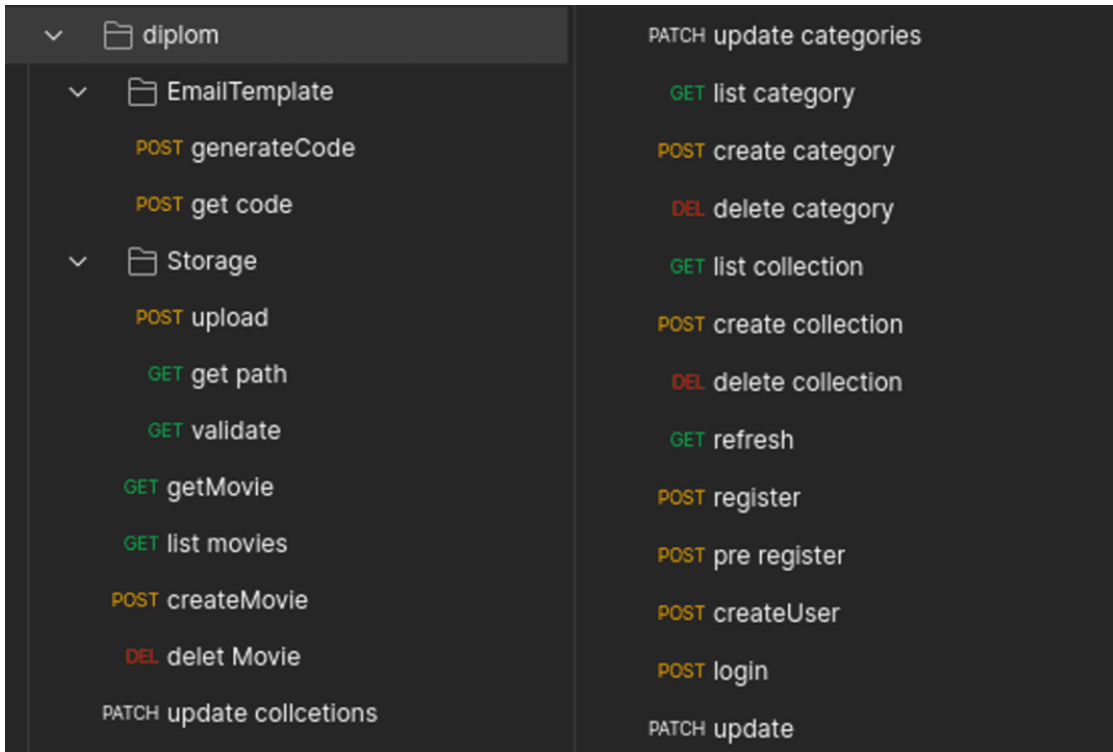


Рисунок 4.2 – Тести на Postman

Таблиця 4.4 – Протестовані складні бізнес процеси

Протестований процес	Результат
1	2
Відображення фільмів, фільтрів, шапки сайту	+
Функціонування навігації у шапці, фільтрації, посторінкової навігації для фільмів	+
Відображення і функціонування кнопок для додавання колекцій за замовченням і категорій, фільмів на головній сторінці адміністратора	+

Продовження таблиці 4.4

1	2
Відображення і функціонування кнопок для додавання власних колекцій на сторінці власних колекцій користувача	+
Відображення і функціонування полів пароль, ім'я, адрес електронної пошти, підтвердження паролю, реєстраційний код на формі реєстрації	+
Отримання електронного листа з реєстраційним кодом	+
Відображення повної інформації про фільм, відео, афіши на сторінці з конкретним фільмом	+
Відображення і функціонування кнопок для додавання фільму у колекцію за замовченням або у категорію на сторінці конкретного фільму адміністратора	+

Додатково була протестована сумісність PCO BestMovieCommon з розробленими MC. Під час встановлення і роботи не було знайдено проблем з сумісністю, так як пакет має лише дві обов'язкові залежності, але пакетний менеджер надавав перевагу більш новому пакету “faker/phpfaker: 2.x-dev” у необов'язкових залежностях PCO, що призводило до проблем під час запуску автоматичних тестів розроблених на платформі PHPUnit. Було прийнято рішення змінити версію з “2.x-dev” на версію “^1.9” у PCO для пакету “faker/phpfaker” і випустити нову версію BestMovieCommon – “v1.21”. Це дозволило без проблем запускати автоматичні тести. Успішне встановлення розробленого PCO у MC MovieMicroservice наведено на рис. 4.3.

```

dev@pc-640-php:/var/www/univer/Diplom/BestMovie/MovieMicroservice$ composer require bestmovie/common "4.1.2"
./composer.json has been updated
Running composer update bestmovie/common
Loading composer repositories with package information
Info from https://repo.packagist.org: #StandWithUkraine
Updating dependencies
Nothing to modify in lock file
Installing dependencies from lock file (including require-dev)
Nothing to install, update or remove
Generating optimized autoload files
Class App\MovieDomain\Movie\MovieCollection located in ./app/MovieDomain/Movie/MovieCollection.php does not comply with psr-4 autoloading standard. Skipping.
> Illuminate\Foundation\ComposerScripts::postAutoloadDump
> @php artisan package:discover --ansi

 INFO  Discovering packages.

bestmovie/common ..... DONE
laravel/sail ..... DONE
laravel/sanctum ..... DONE
laravel/tinker ..... DONE
nesbot/carbon ..... DONE
nunomaduro/collision ..... DONE
nunomaduro/termwind ..... DONE
spatie/laravel-enum ..... DONE
spatie/laravel-ignition ..... DONE

66 packages you are using are looking for funding.
Use the `composer fund` command to find out more!
> @php artisan vendor:publish --tag=laravel-assets --ansi --force

 INFO  No publishable resources for tag [laravel-assets].

No security vulnerability advisories found

```

Рисунок 4.3 – Успішне встановлення PCO у MC MovieMicroservice

4.4 Висновки з розділу

У даному розділі були визначено основні напрямки застосування розробленого проєкту, порядок розгортання і користування системою. У описі інструкції по розгортанню були зазначено обмежений перелік можливих проблем і їх вирішення. Описані проблеми є найчастішими, в основному вони залежать від кешу Composer, NodeJS, Bootstrap. Усі сценарії і функції системи були протестовані, як позитивні, так і негативні. Таке тестування визначило, що основні функції готові до використання і проєкт можна розширювати далі. Під час тестування сумності MC і PCO було знайдено пакет, який може вплинути на подальше розширення системи, тому його було змінено. Загалом тестування сумісності проводиться кожен раз при оновленні PCO, завдяки такому підходу 90-100% помилок виявляються на етапі закладання фундаменту, що дуже позитивно впливає у майбутньому, а саме економить час і гроші при розробці або змінінні коду.

ВИСНОВКИ

З метою створення системи для перегляду фільмів та керування їх колекціями з розвиненою системою безпеки за умов використання DDD було досліджено і реалізовано методи створення MC, їх багат шарову архітектуру, способи спілкування і шаблони програмування, які розглядаються у поєднанні з потужним фундаментом для подальшого розвитку процесів, що надає обрана архітектура.

У роботі розглянуті PCO, на перший погляд це звичайний інтерфейс спілкування MC між собою, але для реалізації існує декілька способів проєктування PCO, кожен з варіантів не є універсальним вирішенням, так як на вибір впливають бізнес процеси. При реалізації обраної теми було обрано найперспективніший варіант, що полягає у розробці окремого пакету. Перевага такого пакету полягає у тому, що система збереження і відображення фільмів може сильно ускладнитися при використанні декількох сховищ, при появі потреби конвертувати фільми у інше розширення або при використанні web-socket з'єднання. Велику частину дослідження було приділено Domain рівню, який в певний момент може потребувати біля сотні запитів до БД. У рамках роботи було проаналізовано багато підходів до оптимізації процесів і під час створення програмного продукту у повній мірі було досліджено найефективніший ряд рішень з точки зору часу і віддачі від реалізації, цей ряд полягає у оптимізації запитів і класів репозитаріїв за умов поєднання з шаблоном “Active Record” і кешем. Реалізація такого підходу дозволила пришвидшити запити на отримання фільмів і колекції. Хоча слід зазначити, що шаблон програмування UOW є більш досконалим вирішенням, але його ефективність можливо дослідити лише в системах з великою кількістю запитів, наприклад у проєктах з грошовими транзакціями. Також важливими елементами розробленої системи є Firebase і Mailgun, які надали можливість масштабувати систему для збільшення кількості програмних маневрів у контексті реалізації рівнів DDD. Кожна розглянута проблема і

відповідне вирішення має ряд переваг і недоліків, які створюються і вирішуються під час масштабування і вдосконалення проєкту, впровадження нових шаблонів.

Розробка проєкту у поєднанні з характерним для архітектури розмахом і популярними мовами програмування призвело до втілення вирішення поставленої проблеми, що полягає у відсутності зручних програмних засобів для керування колекціями, фільмами, медіа файлами. Також у роботі було визначено важність окремих шаблонів і аспектів DDD для сфери обраної теми, що підтвердило гнучкість дослідженої архітектури. Під час реверсної оцінки застосунку постали нові питання, а саме питання автоматичного тестування, надійність авторизації при збільшенні МС і інших вузлів, легке розгортання цілісної системи. Таким чином отримана робота може служити основою для подальшого дослідження легкого розгортання системи на Docker, створенні автоматичних тестів за допомогою платформи PHPUnit [38] або мови Python, поліпшенні авторизації.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. REST design principles [Electronic resource]. – Access mode: <https://www.ibm.com/topics/rest-apis#:~:text=the%20next%20step-,What%20is%20a%20REST%20API%3F,representational%20state%20transfer%20architectural%20style>.
2. Hypertext Transfer Protocol (HTTP) is an application-layer protocol for transmitting hypermedia [Electronic resource]. – Access mode: <https://developer.mozilla.org/en-US/docs/Web/HTTP>.
3. Проєктування мікросервісу, орієнтованого на DDD [Електронний ресурс]. – Режим доступу: <https://learn.microsoft.com/ru-ru/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/ddd-oriented-microservice>.
4. Best Practice - An Introduction To Domain-Driven Design [Electronic resource]. – Access mode: <https://learn.microsoft.com/en-us/archive/msdn-magazine/2009/february/best-practice-an-introduction-to-domain-driven-design>.
5. Evans, E. Domain-Driven Design: Tackling Complexity in the Heart of Software [Text] / Eric Evans. – Boston: Addison-Wesley, 2003. – 560 с.
6. Перегляд фільмів онлайн в HD якості на HdRezka [Електронний ресурс]. – Режим доступу: <https://rezka.ag/>.
7. Дивитися фільми, мультфільми та серіали у чудовій 1080p якості на Filmix [Електронний ресурс]. – Режим доступу: <https://filmix.ac/>.
8. Телебачення та фільми в ультрависокій якості на Sweet TV [Електронний ресурс]. – Режим доступу: <https://sweet.tv/>.
9. Are PHP Web Development Services Any Good for Building Websites in 2023? [Electronic resource]. – Access mode: <https://www.altamira.ai/blog/build-websites-with-php/#:~:text=PHP%20is%20known%20to%20be,PHP%20for%20their%20server%20side>.
10. Top Ranking PHP Frameworks [Electronic resource]. – Access mode:

<https://github.com/umpirsky/top-php-frameworks>.

11. The Document Object Model (DOM) is the data representation of the objects that comprise the structure and content of a document on the web [Electronic resource]. – Access mode: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction.

12. How To Handle DOM and Window Events with React [Electronic resource]. – Access mode: <https://www.digitalocean.com/community/tutorials/how-to-handle-dom-and-window-events-with-react>.

13. SEO оптимізація сайту [Електронний ресурс]. – Режим доступу: <https://www.taina.com.ua/shho-take-seo-optymizacija/>.

14. Abel, A. Domain-Driven Design Quickly [Text] / A. Abel, F. Marinescu. – Morrisville: Lulu Press, 2007. – 103 p.

15. James Hickey. Aggregates are one of the most misunderstood concepts in domain-driven design [Electronic resource]. – Access mode: <https://www.jamesmichaelhickey.com/domain-driven-design-aggregates/>.

16. James Hickey. DDD Aggregates: Consistency Boundary [Electronic resource]. – Access mode: <https://www.jamesmichaelhickey.com/consistency-boundary/>.

17. An In-Depth Understanding of Aggregation in Domain-Driven Design [Electronic resource]. – Access mode: https://www.alibabacloud.com/blog/an-in-depth-understanding-of-aggregation-in-domain-driven-design_59803.

18. Unit of Work Pattern [Electronic resource]. – Access mode: <https://ducmanhphan.github.io/2019-08-20-Unit-of-Work-pattern/>.

19. CQRS stands for Command and Query Responsibility Segregation [Electronic resource]. – Access mode: <https://learn.microsoft.com/en-us/azure/architecture/patterns/cqrs>.

20. Cecil, R.M. Clean Architecture: A Craftsman's Guide to Software Structure and Design [Text] / R.M. Cecil. – London: Pearson, 2017. – 432 p.

21. Implementing the Repository and Unit of Work Patterns [Electronic

resource]. – Access mode: <https://learn.microsoft.com/en-us/aspnet/mvc/overview/older-versions/getting-started-with-ef-5-using-mvc-4/implementing-the-repository-and-unit-of-work-patterns-in-an-asp-net-mvc-application>.

22. The Service / Repository Pattern [Electronic resource]. – Access mode: <https://pmichaels.net/service-repository-pattern/>.

23. The Repository-Service Pattern with DI [Electronic resource]. – Access mode: <https://exceptionnotfound.net/the-repository-service-pattern-with-dependency-injection-and-asp-net-core/>.

24. DDD Building Blocks: Value Object [Electronic resource]. – Access mode: <https://domaincentric.net/blog/ddd-building-blocks-in-php-value-object>.

25. Domain-Driven Design and the Hexagonal Architecture [Electronic resource]. – Access mode: <https://vaadin.com/blog/ddd-part-3-domain-driven-design-and-the-hexagonal-architecture>.

26. Scotts, J. Domain Driven Design : How to Easily Implement Domain Driven Design [Text] / J. Scotts. – Newark: Speedy Publishing LLC, 2015. – 32 p.

27. Hibbs, C. The Art of Lean Software Development [Text] / C. Hibbs. – Sebastopol: O'Reilly Media, 2009. – 142 p.

28. Object-Oriented PHP With Classes and Objects [Electronic resource]. – Access mode: <https://code.tutsplus.com/tutorials/basics-of-object-oriented-programming-in-php--cms-31910>.

29. BestMovie [Electronic resource]. – Access mode: <https://github.com/ALexxx598/BestMovie>.

30. BestMovie: EmailTemplateMicroservice [Electronic resource]. – Access mode: <https://github.com/ALexxx598/EmailTemplateMicroservice>.

31. BestMovie: BestMovieStorage [Electronic resource]. – Access mode: <https://github.com/ALexxx598/BestMovieStorage>.

32. Packagist: BestMovieCommon [Electronic resource]. – Access mode: <https://packagist.org/packages/bestmovie/common>.

33. BestMovie: BestMovieCommon [Electronic resource]. – Access

mode: <https://github.com/ALexxx598/BestMovieCommon>.

34. Laravel DB and Migrations [Electronic resource]. – Access mode: <https://laravel.com/docs/9.x#databases-and-migrations>.

35. Build a Laravel application with a MySQL database [Electronic resource]. – Access mode: <https://planetscale.com/blog/build-a-laravel-application-with-a-mysql-database>.

36. Postman Authorizing requests [Electronic resource]. – Access mode: <https://learning.postman.com/docs/sending-requests/authorization/>.

37. How to Configure PHP-FPM with NGINX. [Electronic resource]. – Access mode: <https://www.digitalocean.com/community/tutorials/php-fpm-nginx>.

38. Use Laravel Unit Testing to Avoid Project-Wrecking Mistakes. [Electronic resource]. – Access mode: <https://www.cloudways.com/blog/laravel-unit-testing/>.

ДОДАТОК А
Текст програми

A.1 MovieMicroservice

A.1.1 API контракты

```
use App\Http\Controllers\CategoryController;
use App\Http\Controllers\CollectionController;
use App\Http\Controllers\MovieController;
use App\Http\Controllers\UserController;
use Illuminate\Support\Facades\Route;
```

```
Route::controller(UserController::class)
->prefix('user')
->group(function () {
    Route::get('/', 'login');
    Route::post('/', 'register');
    Route::post('/pre-registration/', 'preRegister');

    Route::middleware('access_token')->group(function () {
        Route::patch('/', 'update');
        Route::get('/refresh/', 'refresh');
    });
});
```

```
Route::controller(MovieController::class)
->prefix('movie')
->group(function () {

    Route::get('/list', 'list');
    Route::get('/{id}', 'get');

    Route::middleware('access_token')->group(function () {
        Route::middleware('role.viewer')->group(function () {
            Route::patch('/collections/{movieId}', 'updateCollections');
        });

        Route::middleware('role.admin')->group(function () {
            Route::post('/', 'create');
            Route::patch('/collections/default/{movieId}', 'updateDefaultCollections');
            Route::patch('/categories/{movieId}', 'updateCategories');

            Route::group(['prefix' => '{movieId}'], function () {
                Route::delete('/', 'delete');
            });
        });
    });
});
```

```
Route::controller(CategoryController::class)
->prefix('category')
->group(function () {
    Route::get('/list', 'list');

    Route::middleware('access_token')->group(function () {
        Route::middleware('role.admin')->group(function () {
            Route::post('/', 'create');
            Route::delete('/{categoryId}', 'delete');
        });
    });
});
```

```
Route::controller(CollectionController::class)
```

```

->prefix('collection')
->group(function () {
    Route::get('/list', 'list');
    Route::get('/list/defaults/', 'listOfDefaults');

    Route::middleware('access_token')->group(function () {
        Route::post('/', 'create');

        Route::group(['prefix' => '{collectionId}'], function () {
            Route::delete('/', 'delete');
        });
    });
});

```

A.1.2 Middleware

A.1.2.1 AccessTokenCheck

```

namespace App\Http\Middleware;

use App\Common\AuthMiddlewareTrait;
use App\MovieDomain\User\Token\UserTokenServiceInterface;
use Closure;
use Illuminate\Http\Request;
use Symfony\Component\Finder\Exception\AccessDeniedException;

class AccessTokenCheck
{
    use AuthMiddlewareTrait;

    /**
     * @param UserTokenServiceInterface $userTokenService
     */
    public function __construct(
        private UserTokenServiceInterface $userTokenService
    ) {
    }

    /**
     * Handle an incoming request.
     *
     * @param \Illuminate\Http\Request $request
     * @param \Closure(\Illuminate\Http\Request): (\Illuminate\Http\Response|\Illuminate\Http\RedirectResponse)
     * $next
     * @return \Illuminate\Http\Response|\Illuminate\Http\RedirectResponse
     * @throws \App\MovieDomain\User\Exception\UserNotFoundException
     * @throws AccessDeniedException
     */
    public function handle(Request $request, Closure $next)
    {
        $user = $this->userTokenService->getUserByToken($this->checkAuthToken($request));

        if ($user->getId() !== (int)$request->input('user_id')) {
            throw new AccessDeniedException('You must be authorized !!!');
        }

        return $next($request);
    }
}

```

A.1.2.2 AdminRoleValidator

```

namespace App\Http\Middleware;

use App\Common\AuthMiddlewareTrait;
use App\MovieDomain\Role\RoleType;
use App\MovieDomain\User\Token\UserTokenServiceInterface;
use Closure;
use Illuminate\Http\Request;
use Symfony\Component\Finder\Exception\AccessDeniedException;

class AdminRoleValidator
{
    use AuthMiddlewareTrait;

    public function __construct(
        private UserTokenServiceInterface $userTokenService
    ) {
    }

    /**
     * Handle an incoming request.
     *
     * @param \Illuminate\Http\Request $request
     * @param \Closure(\Illuminate\Http\Request): (\Illuminate\Http\Response|\Illuminate\Http\RedirectResponse)
     * @return \Illuminate\Http\Response|\Illuminate\Http\RedirectResponse
     * @throws \App\MovieDomain\User\Exception\UserNotFoundException
     */
    public function handle(Request $request, Closure $next)
    {
        $user = $this->userTokenService->getUserByToken($this->checkAuthToken($request));

        if ($this->getUserId($request) !== $user->getId()) {
            throw new AccessDeniedException('You must be authorized !!!');
        }

        if (!$user->hasRole(RoleType::admin())) {
            throw new AccessDeniedException('You must have admin role');
        }

        return $next($request);
    }
}

```

A.1.2.3 ViwerRoleValidator

```

namespace App\Http\Middleware;

use App\Common\AuthMiddlewareTrait;
use App\MovieDomain\Role\RoleType;
use App\MovieDomain\User\Token\UserTokenServiceInterface;
use Closure;
use Illuminate\Http\Request;
use Symfony\Component\Finder\Exception\AccessDeniedException;

class ViewerRoleValidator
{
    use AuthMiddlewareTrait;

    public function __construct(
        private UserTokenServiceInterface $userTokenService
    ) {
    }
}

```

```

    }

    /**
     * Handle an incoming request.
     *
     * @param \Illuminate\Http\Request $request
     * @param \Closure(\Illuminate\Http\Request): (\Illuminate\Http\Response|\Illuminate\Http\RedirectResponse)
     * @return \Illuminate\Http\Response|\Illuminate\Http\RedirectResponse
     */
    public function handle(Request $request, Closure $next)
    {
        if (!$this->userService->getUserByToken($this->checkAuthToken($request))->hasRole(RoleType::viewer())) {
            throw new AccessDeniedException('You must have viewer role');
        }

        return $next($request);
    }
}

```

A.1.2.4 AccessMicroserviceTokenCheck

```

namespace App\Http\Middleware;

use Closure;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Config;
use Illuminate\Validation\UnauthorizedException;

class AccessMicroserviceTokenCheck
{
    public function __construct(
        private Config $config
    ) {
    }

    /**
     * Handle an incoming request.
     *
     * @param \Illuminate\Http\Request $request
     * @param \Closure(\Illuminate\Http\Request): (\Illuminate\Http\Response|\Illuminate\Http\RedirectResponse)
     * @return \Illuminate\Http\Response|\Illuminate\Http\RedirectResponse
     * @throws \App\MovieDomain\User\Exception\UserNotFoundException
     */
    public function handle(Request $request, Closure $next)
    {
        if ($this->config->get('MICROSERVICE_AUTH_TOKEN') !== $request->header('MICROSERVICE_AUTH')) {
            throw new UnauthorizedException();
        }

        return $next($request);
    }
}

```

A.1.3 Контролери

A.1.3.1 UserController

```

namespace App\Http\Controllers;

use App\Http\Request\User\UserGetRequest;
use App\Http\Request\User\UserPreRegisterRequest;
use App\Http\Request\User\UserRegistrationRequest;
use App\Http\Request\User\UserRequest;
use App\Http\Request\User\UserUpdateRequest;
use App\Http\Resource\User\UserResource;
use BestMovie\Common\EmailTemplateMicroservice\Service\EmailTemplateServiceInterface;
use GuzzleHttp\Exception\GuzzleException;
use Illuminate\Http\JsonResponse;
use App\MovieDomain\User\Payload\UserCreatePayload;
use App\MovieDomain\User\Payload\UserUpdatePayload;
use App\MovieDomain\User\Service\UserServiceInterface;
use App\MovieDomain\User\Token\UserTokenServiceInterface;
use Illuminate\Http\Response;
use Symfony\Component\Finder\Exception\AccessDeniedException;

class UserController
{
    /**
     * @param UserServiceInterface $userService
     * @param UserTokenServiceInterface $userTokenService
     */
    public function __construct(
        private UserServiceInterface $userService,
        private UserTokenServiceInterface $userTokenService,
    ) {
    }

    /**
     * @param UserPreRegisterRequest $request
     * @return JsonResponse
     * @throws GuzzleException
     */
    public function preRegister(UserPreRegisterRequest $request): JsonResponse
    {
        $this->userService->preRegister($request->getEmail());

        return response()->json([
            'status' => Response::HTTP_OK,
        ], Response::HTTP_OK);
    }

    /**
     * @param UserRegistrationRequest $request
     * @return JsonResponse
     * @throws GuzzleException
     */
    public function register(UserRegistrationRequest $request): JsonResponse
    {
        return response()->json(
            [
                'status' => Response::HTTP_CREATED,
                'data' => UserResource::make(
                    $this->userService->create(
                        UserCreatePayload::make(
                            name: $request->getName(),
                            surname: $request->getSurname(),
                            email: $request->getEmail(),
                            password: $request->getUserPassword(),
                            emailConfirmationCode: $request->getEmailConfirmationCode()
                        )
                    )
                )
            ]
        );
    }
}

```

```

        ),
    ], Response::HTTP_CREATED
);
}

/**
 * @param UserGetRequest $request
 * @return JsonResponse
 * @throws \App\MovieDomain\User\Exception\NonValidPasswordException
 * @throws \App\MovieDomain\User\Exception\UserNotFoundException
 */
public function login(UserGetRequest $request): JsonResponse
{
    return response()->json(
        [
            'data' => UserResource::make(
                $this->userService->getUserByEmailAndPassword(
                    $request->getEmail(),
                    $request->getUserPassword()
                )
            ),
        ],
    );
}

/**
 * @param UserRequest $request
 * @return JsonResponse
 * @throws \App\MovieDomain\User\Exception\UserNotFoundException
 */
public function refresh(UserRequest $request): JsonResponse
{
    $user = $this->userTokenService->getUserByToken($request->getAuthHeader());

    if ($user->getId() !== $request->getUserId()) {
        throw new AccessDeniedException('You must be authorized!!!');
    }

    return response()->json([
        'data' => UserResource::make($user)
    ]);
}

/**
 * @param UserUpdateRequest $request
 * @return JsonResponse
 * @throws \App\MovieDomain\User\Exception\UserNotFoundException
 */
public function update(UserUpdateRequest $request): JsonResponse
{
    return response()->json([
        'data' => UserResource::make(
            $this->userService->update(
                new UserUpdatePayload(
                    id: $request->getUserId(),
                    name: $request->getName(),
                    surname: $request->getSurname()
                )
            )
        ),
    ]);
}

```

```
}
}
```

A.1.3.2 MovieController

```
namespace App\Http\Controllers;

use App\Http\Request\Movie\MovieCategoriesRequest;
use App\Http\Request\Movie\MovieCollectionsRequest;
use App\Http\Request\Movie\MovieDeleteRequest;
use App\Http\Request\Movie\MovieListRequest;
use App\Http\Request\Movie\MovieCreateRequest;
use App\Http\Resource\Movie\MovieListResource;
use App\Http\Resource\Movie\MovieResource;
use App\MovieDomain\Movie\Filter\MovieFilter;
use App\MovieDomain\Movie\Payload\MovieCategoryPayload;
use App\MovieDomain\Movie\Payload\MovieCollectionPayload;
use App\MovieDomain\Movie\Payload\MovieCreatePayload;
use App\MovieDomain\Movie\Service\MovieServiceInterface;
use App\MovieDomain\User\Token\UserTokenServiceInterface;
use Illuminate\Http\JsonResponse;
use Illuminate\Http\Response;
use Symfony\Component\Finder\Exception\AccessDeniedException;

class MovieController extends Controller
{
    /**
     * @param MovieServiceInterface $movieService
     * @param UserTokenServiceInterface $userTokenService
     */
    public function __construct(
        private MovieServiceInterface $movieService,
        private UserTokenServiceInterface $userTokenService,
    ) {
    }

    /**
     * @param int $id
     * @return JsonResponse
     */
    public function get(int $id): JsonResponse
    {
        return response()->json([
            'status' => Response::HTTP_OK,
            'data' => MovieResource::make($this->movieService->findById($id))
        ], JsonResponse::HTTP_OK);
    }

    /**
     * @param MovieListRequest $request
     * @return JsonResponse
     * @throws \App\MovieDomain\User\Exception\UserNotFoundException
     */
    public function list(MovieListRequest $request): JsonResponse
    {
        if ($request->getUserId() !== null && $request->getAuthHeader() == null) {
            throw new AccessDeniedException('You must be authorized !!!');
        }

        $filter = MovieFilter::make(
            userId: $request->getUserId() !== null
                ? $this->userTokenService->getUserByToken($request->getAuthHeader())->getId()
        );
    }
}
```

```

        : null,
        categoryIds: $request->getCategoryIds() !== null ? collect($request->getCategoryIds()) : null,
        collectionIds: $request->getCollectionIds() !== null
            ? collect($request->getCollectionIds())
            : null
    )
    ->setPage($request->getPage())
    ->setPerPage($request->getPerPage());

return response()->json([
    'status' => Response::HTTP_OK,
    'data' => MovieListResource::make($this->movieService->list($filter))
], JsonResponse::HTTP_OK);
}

/**
 * @param MovieCreateRequest $request
 * @return JsonResponse
 * @throws \App\MovieDomain\Movie\Exception\InvalidImageUrl
 * @throws \App\MovieDomain\Movie\Exception\InvalidMovieUrl
 * @throws \GuzzleHttp\Exception\GuzzleException
 */
public function create(MovieCreateRequest $request): JsonResponse
{
    $payload = MovieCreatePayload::make(
        name: $request->getName(),
        descriptionRating: $request->getDescriptionRating(),
        descriptionSlogan: $request->getDescriptionSlogan(),
        descriptionScreeningDate: $request->getDescriptionScreeningDate(),
        descriptionCountry: $request->getDescriptionCountry(),
        descriptionActors: $request->getDescriptionActors(),
        shortDescription: $request->getShortDescription(),
        storageMovieUrl: $request->getStorageMovieLink(),
        storageImageUrl: $request->getStorageImageLink()
    );

    return response()->json([
        'status' => Response::HTTP_OK,
        'data' => MovieResource::make($this->movieService->create($payload))
    ], JsonResponse::HTTP_CREATED);
}

/**
 * @param MovieDeleteRequest $request
 * @return JsonResponse
 */
public function delete(MovieDeleteRequest $request, int $movieId): JsonResponse
{
    $this->movieService->delete($movieId);

    return response()->json([
        'status' => Response::HTTP_OK,
    ], JsonResponse::HTTP_CREATED);
}

/**
 * @param int $movieId
 * @param MovieCollectionsRequest $request
 * @return JsonResponse
 * @throws \App\MovieDomain\User\Exception\UserNotFoundException
 */
public function updateCollections(MovieCollectionsRequest $request, int $movieId): JsonResponse
{

```

```

        $payload = MovieCollectionPayload::make(
            userId: $request->getUserId(),
            movieId: $movieId,
            collectionIds: $request->getCollectionIds()
        );

        $this->movieService->syncCollections($payload);

        return response()->json([
            'status' => Response::HTTP_OK,
        ], Response::HTTP_OK);
    }

    /**
     * @param int $movieId
     * @param MovieCategoriesRequest $request
     * @return JsonResponse
     * @throws \App\MovieDomain\User\Exception\UserNotFoundException
     */
    public function updateCategories(MovieCategoriesRequest $request, int $movieId): JsonResponse
    {
        $payload = MovieCategoryPayload::make(
            userId: $request->getUserId(),
            movieId: $movieId,
            categoryIds: $request->getCategoryIds()
        );

        $this->movieService->syncCategories($payload);

        return response()->json([
            'status' => Response::HTTP_OK,
        ], Response::HTTP_OK);
    }

    /**
     * @param MovieCollectionsRequest $request
     * @param int $movieId
     * @return JsonResponse
     */
    public function updateDefaultCollections(MovieCollectionsRequest $request, int $movieId): JsonResponse
    {
        $payload = MovieCollectionPayload::make(
            userId: $request->getUserId(),
            movieId: $movieId,
            collectionIds: $request->getCollectionIds()
        );

        $this->movieService->syncDefaultCollections($payload);

        return response()->json([
            'status' => Response::HTTP_OK,
        ], Response::HTTP_OK);
    }
}

```

A.1.3.3 CollectionController

```

namespace App\Http\Controllers;

use App\Common\MovieMicroserviceRequest;
use App\Http\Request\Collection\CollectionCreateRequest;
use App\Http\Request\Collection\CollectionListRequest;

```

```

use App\Http\Request\Collection\CollectionRemoveRequest;
use App\Http\Resource\Collection\CollectionListResource;
use App\Http\Resource\Collection\CollectionResource;
use App\MovieDomain\Collection\CollectionType;
use App\MovieDomain\Collection\Filter\CollectionFilter;
use App\MovieDomain\Collection\Payload\CollectionCreatePayload;
use App\MovieDomain\Collection\Service\CollectionServiceInterface;
use Illuminate\Http\JsonResponse;
use Illuminate\Http\Response;

class CollectionController
{
    /**
     * @param CollectionServiceInterface $collectionService
     */
    public function __construct(
        private CollectionServiceInterface $collectionService,
    ) {
    }

    /**
     * @throws \App\MovieDomain\User\Exception\UserNotFoundException
     */
    public function list(CollectionListRequest $request): JsonResponse
    {
        $filter =
            CollectionFilter::make(
                userId: $request->getUserId(),
                type: CollectionType::tryFrom($request->getType()),
                collectionIds: $request->getCollectionIds() !== null
                    ? collect($request->getCollectionIds())
                    : null
            )
            ->setPage($request->getPage())
            ->setPerPage($request->getPerPage());

        return response()->json([
            'status' => Response::HTTP_OK,
            'data' => CollectionListResource::make($this->collectionService->list($filter)),
        ], Response::HTTP_OK);
    }

    /**
     * @param MovieMicroserviceRequest $request
     * @return JsonResponse
     */
    public function listOfDefaults(MovieMicroserviceRequest $request): JsonResponse
    {
        return response()->json([
            'status' => Response::HTTP_OK,
            'data' => CollectionListResource::make(
                $this->collectionService->listOfDefaults(CollectionFilter::make())
            )
        ], Response::HTTP_OK);
    }

    /**
     * @throws \App\MovieDomain\User\Exception\UserNotFoundException
     */
    public function create(CollectionCreateRequest $request): JsonResponse
    {
        $payload = CollectionCreatePayload::make(
            userId: $request->getUserId(),

```

```

        name: $request->getName()
    );

    return response()->json([
        'status' => Response::HTTP_OK,
        'data' => CollectionResource::make($this->collectionService->create($payload)),
    ], Response::HTTP_CREATED);
}

/**
 * @throws \App\MovieDomain\User\Exception\UserNotFoundException
 * @throws \App\MovieDomain\Collection\Exception\CollectionNotFound
 */
public function delete(CollectionRemoveRequest $request, int $collectionId): JsonResponse
{
    $this->collectionService->deleteWithPermissionCheck($request->getUserId(), $collectionId);

    return response()->json([
        'status' => Response::HTTP_OK,
    ], Response::HTTP_OK);
}
}

```

A.1.3.4 CategoryController

```

namespace App\Http\Controllers;

use App\Http\Request\Category\CategoryCreateRequest;
use App\Http\Request\Category\CategoryListRequest;
use App\Http\Request\Category\CategoryRemoveRequest;
use App\Http\Resource\Category\CategoryListResource;
use App\Http\Resource\Category\CategoryResource;
use App\MovieDomain\Category\Filter\CategoryFilter;
use App\MovieDomain\Category\Payload\CategoryCreatePayload;
use App\MovieDomain\Category\Service\CategoryServiceInterface;
use Illuminate\Http\JsonResponse;
use Illuminate\Http\Response;

class CategoryController extends Controller
{
    /**
     * @param CategoryServiceInterface $categoryService
     */
    public function __construct(
        private CategoryServiceInterface $categoryService,
    ) {
    }

    /**
     * @param CategoryListRequest $request
     * @return JsonResponse
     */
    public function list(CategoryListRequest $request): JsonResponse
    {
        $filter = CategoryFilter::make(
            categoryIds: $request->getCategoryIds(),
            name: $request->getName()
        )
        ->setPerPage($request->getPerPage())
        ->setPage($request->getPage());

        return response()->json([

```

```

        'status' => Response::HTTP_OK,
        'data' => CategoryListResource::make($this->categoryService->list($filter))
    ], Response::HTTP_OK);
}

/**
 * @param CategoryCreateRequest $request
 * @return JsonResponse
 */
public function create(CategoryCreateRequest $request): JsonResponse
{
    $payload = CategoryCreatePayload::make(
        $request->getName(),
    );

    return response()->json([
        'status' => Response::HTTP_OK,
        'data' => CategoryResource::make($this->categoryService->create($payload))
    ], Response::HTTP_CREATED);
}

/**
 * @param CategoryRemoveRequest $request
 * @param int $categoryId
 * @return JsonResponse
 */
public function delete(CategoryRemoveRequest $request, int $categoryId): JsonResponse
{
    $this->categoryService->delete($categoryId);

    return response()->json([
        'status' => Response::HTTP_OK,
    ], Response::HTTP_CREATED);
}
}

```

A.1.4 Агрегати

A.1.4.1 User

```

namespace App\MovieDomain\User;

use App\MovieDomain\Role\Repository\RoleRepositoryFinderTrait;
use App\MovieDomain\Role\Role;
use App\MovieDomain\Role\RoleType;
use App\MovieDomain\User\Repository\UserRepositoryFinderTrait;
use Carbon\Carbon;
use Illuminate\Support\Collection;

class User
{
    use UserRepositoryFinderTrait;
    use RoleRepositoryFinderTrait;

    /**
     * @param int|null $id
     * @param string $name
     * @param string $surname
     * @param string $email
     * @param string $password
     * @param Collection|null $roles

```

```

* @param string|null $accessToken
* @param Carbon|null $createDate
*/
public function __construct(
    private ?int $id = null,
    private string $name,
    private string $surname,
    private string $email,
    private string $password,
    private ?Collection $roles = null,
    private ?string $accessToken = null,
    private ?Carbon $createDate = null,
) {
}

/**
* @param string|null $accessToken
* @return $this
*/
public function setAccessToken(?string $accessToken): self
{
    $this->accessToken = $accessToken;

    return $this;
}

/**
* @param string $name
* @return $this
*/
public function setName(string $name): self
{
    $this->name = $name;

    return $this;
}

/**
* @param string $surname
* @return $this
*/
public function setSurname(string $surname): self
{
    $this->surname = $surname;

    return $this;
}

/**
* @param int|null $id
* @return $this
*/
public function setId(?int $id): self
{
    $this->id = $id;

    return $this;
}

/**
* @param Carbon|null $createDate
* @return $this
*/

```

```

public function setCreateDate(?Carbon $createDate): self
{
    $this->createDate = $createDate;

    return $this;
}

/**
 * @return int|null
 */
public function getId(): ?int
{
    return $this->id;
}

/**
 * @return Carbon|null
 */
public function getCreateDate(): ?Carbon
{
    return $this->createDate;
}

/**
 * @return string
 */
public function getName(): string
{
    return $this->name;
}

/**
 * @return string
 */
public function getSurname(): string
{
    return $this->surname;
}

/**
 * @return string
 */
public function getEmail(): string
{
    return $this->email;
}

/**
 * @return string
 */
public function getPassword(): string
{
    return $this->password;
}

/**
 * @return string|null
 */
public function getAccessToken(): ?string
{
    return $this->accessToken;
}

```

```

/**
 * @return Collection|null
 */
public function getRoles(): ?Collection
{
    return $this->roles ?? $this->roles = $this->getRepository()->getRolesByUserId($this->getId());
}

/**
 * @param RoleType $roleType
 * @return bool
 */
public function hasRole(RoleType $roleType): bool
{
    $roles = $this->getRoles();

    foreach ($roles as $role) {
        if ($roleType->equals($role->getRole())) {
            return true;
        }
    }

    return false;
}

/**
 * @return bool
 */
public function isAdmin(): bool
{
    return $this->hasRole(RoleType::ADMIN());
}

/**
 * @return bool
 */
public function isViewer(): bool
{
    return $this->hasRole(RoleType::VIEWER());
}

/**
 * @param Role $role
 */
public function addRole(Role $role): void
{
    if (!is_null($this->roles)) {
        $this->roles->add($role);
    } else {
        $this->roles = Collection::make([$role]);
    }
}
}

```

A.1.4.2 Movie

```

namespace App\MovieDomain\Movie;

use App\MovieDomain\Category\Category;
use App\MovieDomain\Category\FILTER\CategoryFilter;
use App\MovieDomain\Category\Repository\CategoryRepositoryFinderTrait;
use App\MovieDomain\Collection\Collection as MovieCollections;

```

```

use App\MovieDomain\Collection\Filter\CollectionFilter;
use App\MovieDomain\Collection\Repository\CollectionRepositoryFinderTrait;
use App\MovieDomain\Storage\BestMovieCachedStorageServiceServiceFinderTrait;
use Illuminate\Support\Collection;

```

```

class Movie
{
    use CategoryRepositoryFinderTrait;
    use CollectionRepositoryFinderTrait;
    use BestMovieCachedStorageServiceServiceFinderTrait;

    /**
     * @param int|null $id
     * @param string $name
     * @param MovieDescription $description
     * @param string $storageMovieUrl
     * @param string $storageImageUrl
     * @param Collection<Category>|null $categories
     * @param Collection<MovieCollections>|null $collections
     */
    public function __construct(
        private ?int $id = null,
        private string $name,
        private MovieDescription $description,
        private string $storageMovieUrl,
        private string $storageImageUrl,
        private ?Collection $categories = null,
        private ?Collection $collections = null,
    ) {
    }

    /**
     * @return int|null
     */
    public function getId(): ?int
    {
        return $this->id;
    }

    /**
     * @param int|null $id
     * @return self
     */
    public function setId(?int $id): self
    {
        $this->id = $id;

        return $this;
    }

    /**
     * @return string
     */
    public function getName(): string
    {
        return $this->name;
    }

    /**
     * @param string $name
     * @return self
     */
    public function setName(string $name): self

```

```

{
    $this->name = $name;

    return $this;
}

/**
 * @return MovieDescription
 */
public function getDescription(): MovieDescription
{
    return $this->description;
}

/**
 * @param MovieDescription $description
 * @return self
 */
public function setDescription(MovieDescription $description): self
{
    $this->description = $description;

    return $this;
}

/**
 * @return string
 */
public function getStorageMovieUrl(): string
{
    return $this->storageMovieUrl;
}

/**
 * @return string|null
 * @throws \GuzzleHttp\Exception\GuzzleException
 */
public function getFullMoviePath(): ?string
{
    return static::getBestMovieCachedStorageService()
        ->getPath(movieId: $this->getId(), path: $this->getStorageMovieUrl());
}

/**
 * @param string $storageMovieUrl
 * @return self
 */
public function setStorageMovieUrl(string $storageMovieUrl): self
{
    $this->storageMovieUrl = $storageMovieUrl;

    return $this;
}

/**
 * @return string
 */
public function getStorageImageUrl(): string
{
    return $this->storageImageUrl;
}

/**

```

```

* @return string|null
* @throws \GuzzleHttp\Exception\GuzzleException
*/
public function getFullImagePath(): ?string
{
    return static::getBestMovieCachedStorageService()
        ->getPath(movieId: $this->getId(), path: $this->getStorageImageUrl());
}

/**
* @param string $storageImageUrl
* @return self
*/
public function setStorageImageUrl(string $storageImageUrl): self
{
    $this->storageImageUrl = $storageImageUrl;

    return $this;
}

/**
* @return Collection<Category>
*/
public function getCategories(): Collection
{
    if ($this->getId() == null) {
        return new Collection([]);
    }

    return $this->categories
        ?? $this->categories = self::getCategoryRepository()->list(CategoryFilter::make(movieId: $this->getId()));
}

/**
* @param Collection<Category>|null $categories
* @return self
*/
public function setCategories(?Collection $categories): self
{
    $this->categories = $categories;

    return $this;
}

/**
* @return Collection<MovieCollections>|null
*/
public function getCollections(): ?Collection
{
    return $this->collections
        ?? $this->collections = $this::getCollectionRepository()->list(
            CollectionFilter::make(movieId: $this->getId())
        );
}

/**
* @param Collection<MovieCollections>|null $collections
* @return self
*/
public function setCollections(?Collection $collections): self
{
    $this->collections = $collections;
}

```

```

        return $this;
    }

    /**
     * @param int $userId
     * @return Collection<int>
     */
    public function getUserCollectionIds(int $userId): Collection
    {
        return $this
            ->getCollections()
            ->filter(fn (MovieCollections $collection) => $collection->isBelongToUser($userId))
            ->map(fn (MovieCollections $collection) => $collection->getId());
    }
}

```

A.1.4.3 Collection

```

namespace App\MovieDomain\Collection;

use App\MovieDomain\User\User;
use Illuminate\Support\Collection as IlluminateCollection;

class Collection
{
    /**
     * @param int|null $id
     * @param int $userId
     * @param CollectionType $type
     * @param string $name
     * @param IlluminateCollection|null $movieIds
     */
    public function __construct(
        private ?int $id = null,
        private int $userId,
        private CollectionType $type,
        private string $name,
        private ?IlluminateCollection $movieIds = null,
    ) {
    }

    /**
     * @return int|null
     */
    public function getId(): ?int
    {
        return $this->id;
    }

    /**
     * @param int|null $id
     * @return self
     */
    public function setId(?int $id): self
    {
        $this->id = $id;

        return $this;
    }

    /**
     * @return int
     */
}

```

```

*/
public function getUserId(): int
{
    return $this->userId;
}

/**
 * @param int $userId
 * @return self
 */
public function setUserId(int $userId): self
{
    $this->userId = $userId;

    return $this;
}

/**
 * @return CollectionType
 */
public function getType(): CollectionType
{
    return $this->type;
}

/**
 * @param CollectionType $type
 * @return self
 */
public function setType(CollectionType $type): self
{
    $this->type = $type;

    return $this;
}

/**
 * @return string
 */
public function getName(): string
{
    return $this->name;
}

/**
 * @param string $name
 * @return self
 */
public function setName(string $name): self
{
    $this->name = $name;

    return $this;
}

/**
 * @return IlluminateCollection|null
 */
public function getMovieIds(): ?IlluminateCollection
{
    return $this->movieIds;
}

```

```

/**
 * @param IlluminateCollection|null $movieIds
 * @return self
 */
public function setMovieIds(?IlluminateCollection $movieIds = null): self
{
    $this->movieIds = $movieIds;

    return $this;
}

/**
 * @return bool
 */
public function isDefault(): bool
{
    return $this->getType()->equals(CollectionType::DEFAULT());
}

/**
 * @return bool
 */
public function isCustom(): bool
{
    return $this->getType()->equals(CollectionType::CUSTOM());
}

/**
 * @param int $id
 * @return bool
 */
public function isBelongToUser(int $id): bool
{
    return $this->getUserId() === $id;
}
}

```

A.1.4.4 Category

```
namespace App\MovieDomain\Category;
```

```
use Illuminate\Support\Collection;
```

```

class Category
{
    /**
     * @param int|null $id
     * @param string $name
     * @param Collection|null $movieIds
     */
    public function __construct(
        private ?int $id = null,
        private string $name,
        private ?Collection $movieIds = null,
    ) {
    }

    /**
     * @return int|null
     */
    public function getId(): ?int
    {

```

```

        return $this->id;
    }

    /**
     * @param int|null $id
     * @return self
     */
    public function setId(?int $id): self
    {
        $this->id = $id;

        return $this;
    }

    /**
     * @return string
     */
    public function getName(): string
    {
        return $this->name;
    }

    /**
     * @param string $name
     * @return self
     */
    public function setName(string $name): self
    {
        $this->name = $name;

        return $this;
    }

    /**
     * @return Collection|null
     */
    public function getMovieIds(): ?Collection
    {
        return $this->movieIds;
    }

    /**
     * @param Collection|null $movieIds
     * @return self
     */
    public function setMovieIds(?Collection $movieIds = null): self
    {
        $this->movieIds = $movieIds;

        return $this;
    }
}

```

A.1.5 Моделі

A.1.5.1 User

```

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Relations\HasMany;

```

```

use Illuminate\Foundation\Auth\User as Authenticatable;
use Illuminate\Notifications\Notifiable;
use Laravel\Sanctum\HasApiTokens;

```

```

/**
 * @property int $id
 * @property string $name
 * @property string $surname
 * @property string $email
 * @property string $password
 * @property string $created_at
 */
class User extends Authenticatable
{
    use HasApiTokens;
    use HasFactory;
    use Notifiable;

    /**
     * @var string
     */
    protected $table = 'users';

    /**
     * The attributes that are mass assignable.
     *
     * @var array<int, string>
     */
    protected $fillable = [
        'name',
        'surname',
        'email',
        'password',
    ];

    /**
     * @return HasMany
     */
    public function roles(): HasMany
    {
        return $this->hasMany(Role::class);
    }
}

```

A.1.5.2 Role

```

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

/**
 * @property int $id
 * @property int $user_id
 * @property string $role
 */
class Role extends Model
{
    use HasFactory;

    /**
     * @var string
     */
}

```

```

    */
    protected $table = 'user_roles';

    /**
     * @var array<int, string>
     */
    protected $fillable = [
        'user_id',
        'role',
    ];
}

```

A.1.5.3 Movie

```

namespace App\Models;

use App\Models\Collection as CollectionModel;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\BelongsToMany;
use Illuminate\Support\Collection;

/**
 * @property int $id
 * @property int $name
 * @property string $description
 * @property string storage_image_link
 * @property string storage_movie_link
 *
 * @property-read Collection<Category> $categories
 * @property-read Collection<CollectionModel> $collections
 */
class Movie extends Model
{
    use HasFactory;

    /**
     * @var string
     */
    protected $table = 'movies';

    /**
     * @var array<int, string>
     */
    protected $fillable = [
        'name',
        'description',
    ];

    /**
     * @var string[]
     */
    protected $casts = [
        'description' => 'array',
    ];

    /**
     * @return BelongsToMany
     */
    public function categories(): BelongsToMany
    {
        return $this->belongsToMany(Category::class, MovieCategory::class);
    }
}

```

```

    }

    /**
     * @return BelongsToMany
     */
    public function collections(): BelongsToMany
    {
        return $this->belongsToMany(CollectionModel::class, MovieCollection::class);
    }
}

```

A.1.5.4 Collection

```

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\BelongsToMany;

/**
 * @property int $id
 * @property int $user_id
 * @property string name
 * @property string $type
 *
 * @property-read \Illuminate\Support\Collection<Movie> $movies
 */
class Collection extends Model
{
    use HasFactory;

    protected $table = 'collections';

    /**
     * @var array<int, string>
     */
    protected $fillable = [
        'user_id',
        'name',
        'type',
    ];

    /**
     * @return BelongsToMany
     */
    public function movies(): BelongsToMany
    {
        return $this->belongsToMany(Movie::class, MovieCollection::class);
    }
}

```

A.1.5.5 Category

```

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\BelongsToMany;

/**
 * @property int $id
 * @property string $name

```

```

*
* @property-read \Illuminate\Support\Collection<Movie> $movies
*/
class Category extends Model
{
    use HasFactory;

    /**
     * @var string
     */
    protected $table = 'categories';

    /**
     * @var array<int, string>
     */
    protected $fillable = [
        'name',
    ];

    /**
     * @return BelongsToMany
     */
    public function movies(): BelongsToMany
    {
        return $this->belongsToMany(Movie::class, MovieCategory::class);
    }
}

```

A.1.6 Репозитрії

A.1.6.1 UserRepositoryInterface

```

namespace App\MovieDomain\User\Repository;

use App\Models\User as UserModel;
use App\MovieDomain\User\Exception\UserNotFoundException;
use App\MovieDomain\User\User;

interface UserRepositoryInterface
{
    /**
     * @param int $id
     * @return User
     * @throws UserNotFoundException
     */
    public function findById(int $id): User;

    /**
     * @param string $email
     * @param string $password
     * @return User
     */
    public function findByEmailAndPassword(string $email, string $password): User;

    /**
     * @param User $user
     * @return int
     */
    public function save(User $user): int;

    /**
     * @param int $id

```

```

    * @return UserModel
    * @throws UserNotFoundException
    */
    public function getUserModel(int $id): UserModel;
}

```

A.1.6.2 UserRepository

```

namespace App\MovieDomain\User\Repository;

use App\Models\User as UserModel;
use App\MovieDomain\User\Exception\NonValidPasswordException;
use App\MovieDomain\User\Exception\UserNotFoundException;
use App\MovieDomain\User\Hash\HashServiceInterface;
use App\MovieDomain\User\User;
use Carbon\Carbon;

class UserRepository implements UserRepositoryInterface
{
    /**
     * @param UserModelMapper $userRepositoryMapper
     * @param HashServiceInterface $hashService
     */
    public function __construct(
        private UserModelMapper $userRepositoryMapper,
        private HashServiceInterface $hashService
    ) {
    }

    /**
     * @inheritDoc
     */
    public function findById(int $id): User
    {
        return $this->userRepositoryMapper->mapModelToEntity($this->getUserModel($id));
    }

    /**
     * @inheritDoc
     * @throws NonValidPasswordException
     * @throws UserNotFoundException
     */
    public function findByEmailAndPassword(string $email, string $password): User
    {
        $user = $this->findByEmail($email);

        if (is_null($user)) {
            throw new UserNotFoundException();
        }

        return $this->checkPassword($user, $password);
    }

    /**
     * @param string $email
     * @return UserModel|null
     */
    private function findByEmail(string $email): ?UserModel
    {
        return UserModel::query()->where('email', $email)->with('roles')->get()->first();
    }
}

```

```

/**
 * @param UserModel $user
 * @param string $password
 * @return User
 * @throws NonValidPasswordException
 */
private function checkPassword(UserModel $user, string $password): User
{
    if (!$this->hashService->check($password, $user->password)) {
        throw new NonValidPasswordException();
    }

    return $this->userRepositoryMapper->mapModelToEntity($user);
}

/**
 * @inheritDoc
 */
public function save(User $user): int
{
    $this->userRepositoryMapper->mapEntityToModel($user, $model = new UserModel());

    $model->exists = $model->id ?? false;
    $model->save();

    $user->setAccessToken($model->createToken('accessToken')->accessToken->token);
    $user->setCreateDate(Carbon::make($model->created_at));

    return $model->id;
}

/**
 * @inheritDoc
 */
public function getUserModel(int $id): UserModel
{
    if (is_null($user = UserModel::find($id))) {
        throw new UserNotFoundException();
    }

    return $user;
}
}

```

A.1.6.3 RoleRepositoryMapper

```

namespace App\MovieDomain\Role\Repository;

use App\Models\Role as RoleModel;
use App\MovieDomain\Role\Role;
use App\MovieDomain\Role\RoleType;
use Illuminate\Database\Eloquent\Collection;
use Illuminate\Support\Collection as IlluminateCollection;

class RoleRepositoryMapper
{
    /**
     * @param Collection<RoleModel> $rolesCollection
     * @return Collection<Role>
     */
    public function mapModelsToEntities(Collection $rolesCollection): IlluminateCollection
    {

```

```

        return $rolesCollection->map(function (RoleModel $role) {
            return $this->mapModelToEntity($role);
        });
    }

    /**
     * @param RoleModel $model
     * @return Role
     */
    public function mapModelToEntity(RoleModel $model): Role
    {
        return new Role(
            id: $model->id,
            userId: $model->user_id,
            role: RoleType::from($model->role)
        );
    }
}

```

A.1.6.4 RoleRepositoryInterface

```

namespace App\MovieDomain\Role\Repository;

use App\MovieDomain\Role\Role;
use App\MovieDomain\Role\RoleType;
use Illuminate\Support\Collection;

interface RoleRepositoryInterface
{
    /**
     * @param int $userId
     * @param RoleType $customer
     * @return Role
     */
    public function save(int $userId, RoleType $customer): Role;

    /**
     * @param int $userId
     * @return Collection<Role>
     */
    public function getRolesByUserId(int $userId): Collection;
}

```

A.1.6.5 RoleRepository

```

namespace App\MovieDomain\Role\Repository;

use App\MovieDomain\Role\Role;
use App\Models\Role as RoleModel;
use App\MovieDomain\Role\RoleType;
use Illuminate\Support\Collection;

class RoleRepository implements RoleRepositoryInterface
{
    /**
     * @param RoleRepositoryMapper $roleRepositoryMapper
     */
    public function __construct(
        private RoleRepositoryMapper $roleRepositoryMapper
    ) {
    }
}

```

```

/**
 * @param int $userId
 * @return Collection<Role>
 */
public function getRolesByUserId(int $userId): Collection
{
    $query = RoleModel::query();
    $query->where('user_id', $userId);

    return $this->roleRepositoryMapper->mapModelsToEntities($query->get());
}

/**
 * @param int $userId
 * @param RoleType $role
 * @return Role
 */
public function save(int $userId, RoleType $role): Role
{
    $model = new RoleModel();

    $model->user_id = $userId;
    $model->role = $role;
    $model->exists = false;

    $model->save();

    return $this->roleRepositoryMapper->mapModelToEntity($model);
}
}

```

A.1.6.6 MovieModelMapper

```

namespace App\MovieDomain\Movie\Repository;

use App\Models\Movie as MovieModel;
use App\MovieDomain\Category\Repository\CategoryModelMapper;
use App\MovieDomain\Collection\Repository\CollectionModelMapper;
use App\MovieDomain\Movie\Movie;
use App\MovieDomain\Movie\MovieDescription;
use Carbon\Carbon;
use Illuminate\Support\Collection;

class MovieModelMapper
{
    /**
     * @param CategoryModelMapper $categoryModelMapper
     * @param CollectionModelMapper $collectionModelMapper
     */
    public function __construct(
        private CategoryModelMapper $categoryModelMapper,
        private CollectionModelMapper $collectionModelMapper,
    ) {
    }

    /**
     * @param Movie $movie
     * @return MovieModel
     */
    public function mapEntityToModel(Movie $movie): MovieModel
    {
        $movieModel = new MovieModel();
    }
}

```

```

$movieModel->name = $movie->getName();

$movieModel->description = json_encode([
    'screeningDate' => $movie->getDescription()->getScreeningDate()->format('Y-m-d H:i:s'),
    'shortDescription' => $movie->getDescription()->getShortDescription(),
    'rating' => $movie->getDescription()->getRating(),
    'actors' => $movie->getDescription()->getActors(),
    'slogan' => $movie->getDescription()->getSlogan(),
    'country' => $movie->getDescription()->getCountry(),
]);

$movieModel->storage_image_link = $movie->getStorageImageUrl();
$movieModel->storage_movie_link = $movie->getStorageMovieUrl();

if ($movie->getId() !== null) {
    $movieModel->id = $movie->getId();
}

return $movieModel;
}

/**
 * @param MovieModel $movieModel
 * @return Movie
 */
public function mapModelToEntity(MovieModel $movieModel): Movie
{
    $description = json_decode($movieModel->description, true);

    $movie = new Movie(
        id: $movieModel->id,
        name: $movieModel->name,
        description: new MovieDescription(
            rating: $description['rating'] ?? null,
            slogan: $description['slogan'] ?? null,
            screeningDate: ($description['screeningDate'] ?? null) !== null
                ? Carbon::make($description['screeningDate'])
                : null,
            country: $description['country'] ?? null,
            actors: $description['actors'] ?? null,
            shortDescription: $description['shortDescription'] ?? null
        ),
        storageMovieUrl: $movieModel->storage_movie_link,
        storageImageUrl: $movieModel->storage_image_link,
    );

    if ($movieModel->relationLoaded('categories')) {
        $movie->setCategories($this->categoryModelMapper->mapModelsToEntities($movieModel->categories));
    }

    // if ($movieModel->relationLoaded('collections')) {
    //     $movie->setCollections($this->collectionModelMapper->mapModelsToEntities($movieModel->collections));
    // }

    return $movie;
}

/**
 * @param Collection<MovieModel> $models
 * @return Collection
 */

```

```

public function mapModelsToEntities(Collection $models): Collection
{
    return $models->map(fn (MovieModel $movie): Movie => $this->mapModelToEntity($movie));
}
}

```

A.1.6.7 MovieRepositoryInterface

```

namespace App\MovieDomain\Movie\Repository;

use App\MovieDomain\Movie\Exception\MovieNotFound;
use App\MovieDomain\Movie\FILTER\MovieFilter;
use App\MovieDomain\Movie\Movie;
use App\MovieDomain\Movie\MovieCollection;
use Illuminate\Support\Collection;

interface MovieRepositoryInterface
{
    /**
     * @param Movie $movie
     * @return int
     */
    public function save(Movie $movie): int;

    /**
     * @param MovieFilter $filter
     * @return MovieCollection
     * @throws MovieNotFound
     */
    public function list(MovieFilter $filter): MovieCollection;

    /**
     * @param int $id
     * @return Movie
     */
    public function findById(int $id): Movie;

    /**
     * @param int $movieId
     * @param int[]|null $collectionIds
     */
    public function syncCollections(int $movieId, ?array $collectionIds): void;

    /**
     * @param int $id
     */
    public function delete(int $id): void;

    /**
     * @param int[] $categoryIds
     */
    public function syncCategories(int $movieId, ?array $categoryIds): void;
}

```

A.1.6.8 MovieRepository

```

namespace App\MovieDomain\Movie\Repository;

use App\MovieDomain\Movie\Exception\MovieNotFound;
use App\MovieDomain\Movie\FILTER\MovieFilter;
use App\MovieDomain\Movie\Movie;
use App\Models\Movie as MovieModel;

```

```

use App\MovieDomain\Movie\MovieCollection;
use Illuminate\Database\Eloquent\Builder;

class MovieRepository implements MovieRepositoryInterface
{
    /**
     * @param MovieModelMapper $moveModelMapper
     */
    public function __construct(
        private MovieModelMapper $moveModelMapper,
    ) {
    }

    /**
     * @inheritDoc
     * @throws MovieNotFound
     */
    public function findById(int $id): Movie
    {
        return $this->moveModelMapper->mapModelToEntity($this->findModelById($id));
    }

    /**
     * @param MovieFilter $filter
     * @return MovieCollection
     */
    public function list(MovieFilter $filter): MovieCollection
    {
        $query = MovieModel::query();

        $query->with(['categories', 'collections']);
        $this->applyToQuery($filter, $query);

        $paginator = $query->paginate(perPage: $filter->getPerPage(), page: $filter->getPage());

        return MovieCollection::make($this->moveModelMapper->mapModelsToEntities(collect($paginator->items()))
            ->setPerPage($paginator->perPage())
            ->setPage($paginator->currentPage())
            ->setTotal($paginator->total())
            ->setLastPage($paginator->lastPage()));
    }

    /**
     * @inheritDoc
     */
    public function save(Movie $movie): int
    {
        $model = $this->moveModelMapper->mapEntityToModel($movie);

        $model->exists = $model->id ?? false;
        $model->save();

        return $model->id;
    }

    /**
     * @param MovieFilter $filter
     * @param Builder $query
     */
    private function applyToQuery(MovieFilter $filter, Builder $query): void
    {
        if ($filter->getCategoryIds() !== null) {

```

```

        $query->whereHas('categories', function (Builder $query) use ($filter) {
            $query->whereIn('category_id', $filter->getCategoryIds()->toArray());
        });
    }

    if ($filter->getUserId() !== null) {
        $query->whereHas('collections', function (Builder $query) use ($filter) {
            $query->where('user_id', $filter->getUserId());
        });
    }

    if ($filter->getCollectionType() !== null) {
        $query->whereHas('collections', function (Builder $query) use ($filter) {
            $query->where('type', $filter->getCollectionType()->value);
        });
    }

    if ($filter->getCollectionIds() !== null) {
        $query->whereHas('collections', function (Builder $query) use ($filter) {
            $query->whereIn('collection_id', $filter->getCollectionIds()->toArray());
        });
    }
}

/**
 * @param int $id
 * @return MovieModel
 * @throws MovieNotFound
 */
private function findModelById(int $id): MovieModel
{
    if (is_null($movie = MovieModel::find($id))) {
        throw new MovieNotFound();
    };

    return $movie;
}

/**
 * @param int $movieId
 * @param int[]|null $collectionIds
 * @throws MovieNotFound
 */
public function syncCollections(int $movieId, ?array $collectionIds): void
{
    $movie = $this->findModelById($movieId);

    if ($collectionIds !== null) {
        $movie->collections()->sync($collectionIds);
    }
}

public function delete(int $id): void
{
    if (is_null($movie = MovieModel::find($id))) {
        throw new MovieNotFound();
    };

    $movie->delete();
}

/**
 * @param int $movieId

```

```

    * @param array|null $categoryIds
    * @throws MovieNotFound
    */
    public function syncCategories(int $movieId, ?array $categoryIds): void
    {
        $movie = $this->findModelById($movieId);

        if ($categoryIds !== null) {
            $movie->categories()->sync($categoryIds);
        }
    }
}

```

A.1.6.9 CollectionRepositoryInterface

```

namespace App\MovieDomain\Collection\Repository;

use App\MovieDomain\Collection\Collection;
use App\MovieDomain\Collection\Exception\CollectionNotFound;
use App\MovieDomain\Collection\Filter\CollectionFilter;
use App\MovieDomain\Collection\MovieCollections;

interface CollectionRepositoryInterface
{
    /**
     * @param Collection $collection
     * @return int
     */
    public function save(Collection $collection): int;

    /**
     * @param CollectionFilter $filter
     * @return MovieCollections
     */
    public function list(CollectionFilter $filter): MovieCollections;

    /**
     * @param int $id
     * @return Collection
     * @throws CollectionNotFound
     */
    public function findById(int $id): Collection;

    /**
     * @param int $id
     */
    public function delete(int $id): void;
}

```

A.1.6.10 CollectionRepository

```

namespace App\MovieDomain\Collection\Repository;

use App\Models\Collection as CollectionModel;
use App\MovieDomain\Collection\Collection;
use App\MovieDomain\Collection\Exception\CollectionNotFound;
use App\MovieDomain\Collection\Filter\CollectionFilter;
use App\MovieDomain\Collection\MovieCollections;
use Illuminate\Database\Eloquent\Builder;

class CollectionRepository implements CollectionRepositoryInterface
{

```

```

/**
 * @param CollectionModelMapper $collectionModelMapper
 */
public function __construct(
    private CollectionModelMapper $collectionModelMapper,
) {
}

/**
 * @inheritDoc
 */
public function findById(int $id): Collection
{
    if (is_null($collection = CollectionModel::find($id))) {
        throw new CollectionNotFound();
    }

    return $this->collectionModelMapper->mapModelToEntity($collection);
}

/**
 * @inheritDoc
 */
public function save(Collection $collection): int
{
    $model = $this->collectionModelMapper->mapEntityToModel($collection);

    $model->exists = $model->id ?? false;
    $model->save();

    return $model->id;
}

/**
 * @inheritDoc
 */
public function list(CollectionFilter $filter): MovieCollections
{
    $query = CollectionModel::query();

    $query->with(['movies']);
    $this->applyToQuery($filter, $query);

    $paginator = $query->paginate(perPage: $filter->getPerPage(), page: $filter->getPage());

    return MovieCollections::make($this->collectionModelMapper->mapModelsToEntities(collect($paginator->items())))
        ->setPerPage($paginator->perPage())
        ->setPage($paginator->currentPage())
        ->setTotal($paginator->total())
        ->setLastPage($paginator->lastPage());
}

/**
 * @param CollectionFilter $filter
 * @param Builder $query
 */
private function applyToQuery(CollectionFilter $filter, Builder $query)
{
    if ($filter->getUserId() !== null) {
        $query->where('user_id', $filter->getUserId());
    }
}

```

```

        if ($filter->getWithoutUserId() !== null) {
            $query->whereNot('user_id', $filter->getWithoutUserId());
        }

        if ($filter->getType() !== null) {
            $query->where('type', $filter->getType()->value);
        }

        if ($filter->getTypes() !== null) {
            $query->whereIn('type', $filter->getTypesValuesArray());
        }

        if ($filter->getCollectionIds() !== null) {
            $query->whereIn('id', $filter->getCollectionIds()->toArray());
        }

        if ($filter->getMovieId() !== null) {
            $query->whereHas('movies', function (Builder $query) use ($filter) {
                $query->where('movie_id', $filter->getMovieId());
            });
        }
    }

    /**
     * @inheritDoc
     */
    public function delete(int $id): void
    {
        CollectionModel::query()->where('id', $id)->delete();
    }
}

```

A.1.6.11 CategoryRepositoryInterface

```

namespace App\MovieDomain\Category\Repository;

use App\MovieDomain\Category\Category;
use App\MovieDomain\Category\CategoryCollection;
use App\MovieDomain\Category\Filter\CategoryFilter;

interface CategoryRepositoryInterface
{
    /**
     * @param Category $category
     * @return int
     */
    public function save(Category $category): int;

    /**
     * @param CategoryFilter $filter
     * @return CategoryCollection
     */
    public function list(CategoryFilter $filter): CategoryCollection;

    /**
     * @param int $categoryId
     * @return void
     */
    public function deleteById(int $categoryId): void;
}

```

A.1.6.12 CategoryRepository

```

namespace App\MovieDomain\Category\Repository;

use App\Models\Category as CategoryModel;
use App\MovieDomain\Category\Category;
use App\MovieDomain\Category\CategoryCollection;
use App\MovieDomain\Category\FILTER\CategoryFilter;
use Illuminate\Database\Eloquent\Builder;

class CategoryRepository implements CategoryRepositoryInterface
{
    /**
     * @param CategoryModelMapper $categoryModelMapper
     */
    public function __construct(
        private CategoryModelMapper $categoryModelMapper,
    ) {
    }

    public function list(CategoryFilter $filter): CategoryCollection
    {
        $query = CategoryModel::query();

        $query->with(['movies']);
        $this->applyToQuery($filter, $query);

        $paginator = $query->paginate(perPage: $filter->getPerPage(), page: $filter->getPage());

        return CategoryCollection::make($this->categoryModelMapper->mapModelsToEntities(collect($paginator->items()))
            ->setPerPage($paginator->perPage())
            ->setPage($paginator->currentPage())
            ->setTotal($paginator->total())
            ->setLastPage($paginator->lastPage()));
    }

    /**
     * @param Category $category
     * @return int
     */
    public function save(Category $category): int
    {
        $model = $this->categoryModelMapper->mapEntityToModel($category);

        $model->exists = $model->id ?? false;
        $model->save();

        return $model->id;
    }

    /**
     * @param CategoryFilter $filter
     * @param Builder $query
     */
    private function applyToQuery(CategoryFilter $filter, Builder $query)
    {
        if ($filter->getMovieId() !== null) {
            $query->whereHas('movies', function (Builder $query) use ($filter) {
                return $query->where('movie_id', $filter->getMovieId());
            });
        }
    }
}

```

```

    if ($filter->getName() !== null) {
        $query->where('name', 'like', '%' . $filter->getName() . '%');
    }

    if ($filter->getCategoryIds() !== null) {
        $query->whereIn('id', $filter->getCategoryIds());
    }
}

/**
 * @inheritDoc
 */
public function deleteById(int $categoryId): void
{
    CategoryModel::query()->where('id', $categoryId)->delete();
}
}

```

A.1.7 Сервис

A.1.7.1 UserServiceInterface

```

namespace App\MovieDomain\User\Service;

use App\MovieDomain\Role\RoleType;
use App\MovieDomain\User\Exception\NonValidPasswordException;
use App\MovieDomain\User\Exception\UserNotFoundException;
use App\MovieDomain\User\Payload\UserCreatePayload;
use App\MovieDomain\User\Payload\UserUpdatePayload;
use App\MovieDomain\User\User;
use GuzzleHttp\Exception\GuzzleException;

interface UserServiceInterface
{
    /**
     * @param int $id
     * @return User
     * @throws UserNotFoundException
     */
    public function findUser(int $id): User;

    /**
     * @param UserCreatePayload $userPayload
     * @return User
     * @throws GuzzleException
     */
    public function create(UserCreatePayload $userPayload): User;

    /**
     * @param string $email
     * @param string $password
     * @return User
     * @throws NonValidPasswordException
     * @throws UserNotFoundException
     */
    public function getUserByEmailAndPassword(string $email, string $password): User;

    /**
     * @param int $id
     * @param RoleType $roleType
     * @return bool
     */
}

```

```

    * @throws UserNotFoundException
    */
    public function hasRole(int $id, RoleType $roleType): bool;

    /**
     * @param UserUpdatePayload $updatePayload
     * @return User
     * @throws UserNotFoundException
     */
    public function update(UserUpdatePayload $updatePayload): User;

    /**
     * @param string $email
     * @return void
     * @throws GuzzleException
     */
    public function preRegister(string $email): void;
}

```

A.1.7.2 UserService

```

namespace App\MovieDomain\User\Service;

use App\MovieDomain\Role\RoleType;
use App\MovieDomain\Role\Service\RoleServiceInterface;
use App\MovieDomain\User\Payload\UserCreatePayload;
use App\MovieDomain\User\Payload\UserPayloadToEntityMapper;
use App\MovieDomain\User\Payload\UserUpdatePayload;
use App\MovieDomain\User\Repository\UserRepositoryInterface;
use App\MovieDomain\User\User;
use BestMovie\Common\EmailTemplateMicroservice\Service\EmailTemplateServiceInterface;
use GuzzleHttp\Exception\GuzzleException;

class UserService implements UserServiceInterface
{
    /**
     * @param UserRepositoryInterface $userRepository
     * @param UserPayloadToEntityMapper $mapper
     * @param RoleServiceInterface $roleService
     * @param EmailTemplateServiceInterface $emailTemplateService
     */
    public function __construct(
        private UserRepositoryInterface $userRepository,
        private UserPayloadToEntityMapper $mapper,
        private RoleServiceInterface $roleService,
        private EmailTemplateServiceInterface $emailTemplateService,
    ) {
    }

    /**
     * @inheritDoc
     */
    public function findUser(int $id): User
    {
        return $this->userRepository->findById($id);
    }

    /**
     * @inheritDoc
     */
    public function getUserByEmailAndPassword(string $email, string $password): User
    {

```

```

        return $this->userRepository->findByEmailAndPassword($email, $password);
    }

    /**
     * @inheritDoc
     */
    public function hasRole(int $id, RoleType $roleType): bool
    {
        return $this->findUser($id)->hasRole($roleType);
    }

    /**
     * @inheritDoc
     */
    public function create(UserCreatePayload $userPayload): User
    {
        if ($this->emailTemplateService->getCode($userPayload->getEmail())->getCode()
            !== $userPayload->getEmailConfirmationCode()
        ) {
            //throw exception code non-valid
        }

        $user = $this->mapper->mapCreatePayloadToEntity($userPayload);

        $user->setId($this->userRepository->save($user));

        $user->addRole($this->roleService->addViewerRole($user->getId()));

        return $user;
    }

    /**
     * @inheritDoc
     */
    public function update(UserUpdatePayload $updatePayload): User
    {
        $user = $this->findUser($updatePayload->getId());

        if (!is_null($updatePayload->getName())) {
            $user->setName($updatePayload->getName());
        }

        if (!is_null($updatePayload->getSurname())) {
            $user->setSurname($updatePayload->getSurname());
        }

        return $user->setId($this->userRepository->save($user));
    }

    /**
     * @inheritDoc
     */
    public function preRegister(string $email): void
    {
        $this->emailTemplateService->generateCode($email);
    }
}

```

A.1.7.3 RoleServiceInterface

```
namespace App\MovieDomain\Role\Service;
```

```

use App\MovieDomain\Role\Role;

interface RoleServiceInterface
{
    /**
     * @param int $userId
     * @return Role
     */
    public function addAdminRole(int $userId): Role;

    /**
     * @param int $userId
     * @return Role
     */
    public function addViewerRole(int $userId): Role;
}

```

A.1.7.4 RoleService

```

namespace App\MovieDomain\Role\Service;

use App\MovieDomain\Role\Repository\RoleRepositoryInterface;
use App\MovieDomain\Role\Role;
use App\MovieDomain\Role\RoleType;

class RoleService implements RoleServiceInterface
{
    /**
     * @param RoleRepositoryInterface $roleRepository
     */
    public function __construct(
        private RoleRepositoryInterface $roleRepository
    ) {
    }

    /**
     * @param int $userId
     * @return Role
     */
    public function addAdminRole(int $userId): Role
    {
        return $this->roleRepository->save($userId, RoleType::admin());
    }

    /**
     * @param int $userId
     * @return Role
     */
    public function addViewerRole(int $userId): Role
    {
        return $this->roleRepository->save($userId, RoleType::viewer());
    }
}

```

A.1.7.5 MovieServiceInterface

```

namespace App\MovieDomain\Movie\Service;

use App\MovieDomain\Movie\Exception\InvalidImageUrl;
use App\MovieDomain\Movie\Exception\InvalidMovieUrl;
use App\MovieDomain\Movie\Footer\MovieFooter;
use App\MovieDomain\Movie\Movie;

```

```

use App\MovieDomain\Movie\Payload\MovieCategoryPayload;
use App\MovieDomain\Movie\Payload\MovieCollectionPayload;
use App\MovieDomain\Movie\Payload\MovieCreatePayload;
use App\MovieDomain\Movie\MovieCollection;

interface MovieServiceInterface
{
    /**
     * @param MovieCreatePayload $payload
     * @throws InvalidMovieUrl
     * @throws InvalidImageUrl
     * @throws \GuzzleHttp\Exception\GuzzleException
     * @return Movie
     */
    public function create(MovieCreatePayload $payload): Movie;

    /**
     * @param MovieFilter $filter
     * @return MovieCollection
     */
    public function list(MovieFilter $filter): MovieCollection;

    /**
     * @param int $id
     * @return Movie
     */
    public function findById(int $id): Movie;

    /**
     * @param MovieCollectionPayload $payload
     * @throws \App\MovieDomain\User\Exception\UserNotFoundException
     */
    public function syncCollections(MovieCollectionPayload $payload): void;

    /**
     * @param MovieCollectionPayload $payload
     */
    public function syncDefaultCollections(MovieCollectionPayload $payload): void;

    /**
     * @param int $getId
     */
    public function delete(int $id): void;

    /**
     * @param MovieCategoryPayload $payload
     */
    public function syncCategories(MovieCategoryPayload $payload): void;
}

```

A.1.7.6 MovieService

```

namespace App\MovieDomain\Movie\Service;

use App\MovieDomain\Category\Category;
use App\MovieDomain\Category\Filter\CategoryFilter;
use App\MovieDomain\Category\Service\CategoryServiceInterface;
use App\MovieDomain\Collection\Collection;
use App\MovieDomain\Collection\CollectionType;
use App\MovieDomain\Collection\Filter\CollectionFilter;
use App\MovieDomain\Collection\Service\CollectionServiceInterface;
use App\MovieDomain\Movie\Exception\InvalidImageUrl;

```

```

use App\MovieDomain\Movie\Exception\InvalidMovieUrl;
use App\MovieDomain\Movie\Filter\MovieFilter;
use App\MovieDomain\Movie\Movie;
use App\MovieDomain\Movie\MovieCollection;
use App\MovieDomain\Movie\MovieDescription;
use App\MovieDomain\Movie\Payload\MovieCategoryPayload;
use App\MovieDomain\Movie\Payload\MovieCollectionPayload;
use App\MovieDomain\Movie\Payload\MovieCreatePayload;
use App\MovieDomain\Movie\Repository\MovieRepositoryInterface;
use App\MovieDomain\User\Service\UserServiceInterface;
use BestMovie\Common\BestMovieStorage\Service\BestMovieStorageServiceInterface;

class MovieService implements MovieServiceInterface
{
    /**
     * @param MovieRepositoryInterface $movieRepository
     * @param UserServiceInterface $userService
     * @param CollectionServiceInterface $collectionService
     * @param CategoryServiceInterface $categoryService
     * @param BestMovieStorageServiceInterface $bestMovieStorageService
     */
    public function __construct(
        private MovieRepositoryInterface $movieRepository,
        private UserServiceInterface $userService,
        private CollectionServiceInterface $collectionService,
        private CategoryServiceInterface $categoryService,
        private BestMovieStorageServiceInterface $bestMovieStorageService,
    ) {
    }

    /**
     * @inheritdoc
     */
    public function findById(int $id): Movie
    {
        return $this->movieRepository->findById($id);
    }

    /**
     * @inheritdoc
     */
    public function list(MovieFilter $filter): MovieCollection
    {
        return $this->movieRepository->list($filter);
    }

    /**
     * @inheritdoc
     */
    public function create(MovieCreatePayload $payload): Movie
    {
        if (!$this->bestMovieStorageService->validatePath($payload->getStorageMovieUrl())) {
            throw new InvalidMovieUrl();
        }

        if (!$this->bestMovieStorageService->validatePath($payload->getStorageImageUrl())) {
            throw new InvalidImageUrl();
        };

        $movie = new Movie(
            name: $payload->getName(),
            description: new MovieDescription(
                rating: $payload->getDescriptionRating(),

```

```

        slogan: $payload->getDescriptionSlogan(),
        screeningDate: $payload->getDescriptionScreeningDate(),
        country: $payload->getDescriptionCountry(),
        actors: $payload->getDescriptionActors(),
        shortDescription: $payload->getShortDescription()
    ),
    storageMovieUrl: $payload->getStorageMovieUrl(),
    storageImageUrl: $payload->getStorageImageUrl()
);

return $movie->setId($this->movieRepository->save($movie));
}

/**
 * @inheritDoc
 */
public function syncCollections(MovieCollectionPayload $payload): void
{
    $user = $this->userService->findUser($payload->getUserId());

    $movie = $this->movieRepository->findById($payload->getMovieId());

    $collectionIds = $this
        ->collectionService
        ->list(
            CollectionFilter::make(
                userId: $user->getId(),
                type: CollectionType::CUSTOM(),
                collectionIds: $payload->getCollectionIds()
            )
        )
        ->map(fn (Collection $collection) => $collection->getId())
        ->toArray();

    $defaultCollectionIds = $this->collectionService
        ->list(
            CollectionFilter::make(
                movieId: $movie->getId(),
                withoutUserId: $user->getId(),
            ),
        )
        ->map(fn (Collection $collection): int => $collection->getId())
        ->toArray();

    $collectionIds = array_merge($collectionIds, $defaultCollectionIds);

    $this->movieRepository->syncCollections($movie->getId(), $collectionIds);
}

/**
 * @inheritDoc
 */
public function syncDefaultCollections(MovieCollectionPayload $payload): void
{
    $movie = $this->movieRepository->findById($payload->getMovieId());

    $collectionIds = $this
        ->collectionService
        ->list(
            CollectionFilter::make(
                type: CollectionType::DEFAULT(),
                collectionIds: $payload->getCollectionIds()
            )
        )

```

```

    )
    ->map(fn (Collection $collection) => $collection->getId())
    ->toArray();

    $customCollectionIds = $this->collectionService
        ->list(CollectionFilter::make(movieId: $movie->getId(), type: CollectionType::CUSTOM()))
        ->map(fn (Collection $collection): int => $collection->getId())
        ->toArray();

    $collectionIds = array_merge($collectionIds, $customCollectionIds);

    $this->movieRepository->syncCollections($movie->getId(), $collectionIds);
}

public function delete(int $id): void
{
    $this->movieRepository->delete($id);
}

public function syncCategories(MovieCategoryPayload $payload): void
{
    $movie = $this->movieRepository->findById($payload->getMovieId());

    $categoryIds = $this
        ->categoryService
        ->list(
            CategoryFilter::make(
                categoryIds: $payload->getCategoryIds()->toArray()
            )
        )
        ->map(fn (Category $category) => $category->getId())
        ->toArray();

    $this->movieRepository->syncCategories($movie->getId(), $categoryIds);
}
}

```

A.1.7.7 CollectionServiceInterface

```

namespace App\MovieDomain\Collection\Service;

use App\MovieDomain\Collection\Collection;
use App\MovieDomain\Collection\Filter\CollectionFilter;
use App\MovieDomain\Collection\MovieCollections;
use App\MovieDomain\Collection\Payload\CollectionCreatePayload;

interface CollectionServiceInterface
{
    /**
     * @param int $id
     * @return Collection
     * @throws \App\MovieDomain\Collection\Exception\CollectionNotFound
     */
    public function findById(int $id): Collection;

    /**
     * @param CollectionCreatePayload $payload
     * @return Collection
     */
    public function create(CollectionCreatePayload $payload): Collection;
}

```

```

/**
 * @param CollectionFilter $filter
 * @return MovieCollections
 */
public function list(CollectionFilter $filter): MovieCollections;

/**
 * @param CollectionFilter $filter
 * @return MovieCollections
 */
public function listOfDefaults(CollectionFilter $filter): MovieCollections;

/**
 * @param int $userId
 * @param int $collectionId
 * @throws \App\MovieDomain\User\Exception\UserNotFoundException
 * @throws \App\MovieDomain\Collection\Exception\CollectionNotFound
 */
public function deleteWithPermissionCheck(int $userId, int $collectionId): void;

/**
 * @param int $collectionId
 */
public function delete(int $collectionId): void;
}

```

A.1.7.8 CollectionService

```

namespace App\MovieDomain\Collection\Service;

use App\MovieDomain\Collection\Collection;
use App\MovieDomain\Collection\CollectionType;
use App\MovieDomain\Collection\Filter\CollectionFilter;
use App\MovieDomain\Collection\MovieCollections;
use App\MovieDomain\Collection\Payload\CollectionCreatePayload;
use App\MovieDomain\Collection\Repository\CollectionRepositoryInterface;
use App\MovieDomain\MovieCollection\Repository\MovieCollectionRepositoryInterface;
use App\MovieDomain\User\Service\UserServiceInterface;

class CollectionService implements CollectionServiceInterface
{
    /**
     * @param UserServiceInterface $userService
     * @param CollectionRepositoryInterface $collectionRepository
     * @param MovieCollectionRepositoryInterface $movieCollectionRepository
     */
    public function __construct(
        private UserServiceInterface $userService,
        private CollectionRepositoryInterface $collectionRepository,
        private MovieCollectionRepositoryInterface $movieCollectionRepository,
    ) {
    }

    /**
     * @inheritDoc
     */
    public function findById(int $id): Collection
    {
        return $this->collectionRepository->findById($id);
    }

    /**

```

```

* @param CollectionFilter $filter
* @return MovieCollections
* @throws \App\MovieDomain\User\Exception\UserNotFoundException
*/
public function list(CollectionFilter $filter): MovieCollections
{
    return $this->collectionRepository->list($filter);
}

/**
* @param CollectionFilter $filter
* @return MovieCollections
*/
public function listOfDefaults(CollectionFilter $filter): MovieCollections
{
    $filter->setType(CollectionType::DEFAULT());

    return $this->collectionRepository->list($filter);
}

/**
* @throws \App\MovieDomain\User\Exception\UserNotFoundException
*/
public function create(CollectionCreatePayload $payload): Collection
{
    $user = $this->userService->findUser($payload->getUserId());

    $type = match (true) {
//      $user->isAdmin() && $user->isViewer() => CollectionType::TEST(),
        $user->isAdmin() => CollectionType::DEFAULT(),
        $user->isViewer() => CollectionType::CUSTOM(),
        default => CollectionType::TEST(),
    };

    $collection = new Collection(
        userId: $payload->getUserId(),
        type: $type,
        name: $payload->getName()
    );

    return $collection->setId($this->collectionRepository->save($collection));
}

/**
* @inheritDoc
*/
public function deleteWithPermissionCheck(int $userId, int $collectionId): void
{
    $user = $this->userService->findUser($userId);
    $collection = $this->findById($collectionId);

    if ($user->isAdmin() && $collection->isDefault()) {
        $this->delete($collectionId);
    }

    if ($user->isViewer() && $collection->isCustom() && $collection->isBelongToUser($user->getId())) {
        $this->delete($collectionId);
    }
}

/**
* @inheritDoc
*/

```

```

public function delete(int $collectionId): void
{
    $this->movieCollectionRepository->deleteByCollectionId($collectionId);
    $this->collectionRepository->delete($collectionId);
}
}

```

A.1.7.9 CategoryServiceInterface

```

namespace App\MovieDomain\Category\Service;

use App\MovieDomain\Category\Category;
use App\MovieDomain\Category\CategoryCollection;
use App\MovieDomain\Category\Filter\CategoryFilter;
use App\MovieDomain\Category\Payload\CategoryCreatePayload;

interface CategoryServiceInterface
{
    /**
     * @param CategoryCreatePayload $payload
     * @return Category
     */
    public function create(CategoryCreatePayload $payload): Category;

    /**
     * @param CategoryFilter $filter
     * @return CategoryCollection
     */
    public function list(CategoryFilter $filter): CategoryCollection;

    /**
     * @param int $categoryId
     */
    public function delete(int $categoryId): void;
}

```

A.1.7.10 CategoryService

```

namespace App\MovieDomain\Category\Service;

use App\MovieDomain\Category\Category;
use App\MovieDomain\Category\CategoryCollection;
use App\MovieDomain\Category\Filter\CategoryFilter;
use App\MovieDomain\Category\Payload\CategoryCreatePayload;
use App\MovieDomain\Category\Repository\CategoryRepositoryInterface;
use App\MovieDomain\MovieCategory\Repository\MovieCategoryRepositoryInterface;

class CategoryService implements CategoryServiceInterface
{
    /**
     * @param CategoryRepositoryInterface $categoryRepository
     */
    public function __construct(
        private CategoryRepositoryInterface $categoryRepository,
        private MovieCategoryRepositoryInterface $movieCategoryRepository,
    ) {
    }

    /**
     * @param CategoryFilter $filter
     * @return CategoryCollection
     */
}

```

```

public function list(CategoryFilter $filter): CategoryCollection
{
    return $this->categoryRepository->list($filter);
}

/**
 * @param CategoryCreatePayload $payload
 * @return Category
 */
public function create(CategoryCreatePayload $payload): Category
{
    $category = new Category(
        name: $payload->getName(),
    );

    return $category->setId($this->categoryRepository->save($category));
}

public function delete(int $categoryId): void
{
    $this->movieCategoryRepository->deleteByCategoryId($categoryId);
    $this->categoryRepository->deleteById($categoryId);
}
}

```

A.1.7.11 HashService

```

namespace App\MovieDomain\User\Hash;

use Illuminate\Support\Facades\Hash;

class HashService implements HashServiceInterface
{
    /**
     * @param string $password
     * @return string
     */
    public function makeHash(string $password): string
    {
        return Hash::make($password);
    }

    /**
     * @param string $password
     * @param string $hash
     * @return bool
     */
    public function check(string $password, string $hash): bool
    {
        return Hash::check($password, $hash);
    }
}

```

A.1.7.12 HashServiceInterface

```

namespace App\MovieDomain\User\Hash;

interface HashServiceInterface
{
    /**
     * @param string $password
     * @return string
     */
}

```

```

    */
    public function makeHash(string $password): string;

    /**
     * @param string $password
     * @param string $hash
     * @return bool
     */
    public function check(string $password, string $hash): bool;
}

```

A.1.8 Traits

A.1.8.1 CategoryRepositoryFinderTrait

```

namespace App\MovieDomain\Category\Repository;

use Closure;

trait CategoryRepositoryFinderTrait
{
    /**
     * @var Closure|null
     */
    private static ?Closure $categoryResolver = null;

    /**
     * @var CategoryRepositoryInterface|null
     */
    private static ?CategoryRepositoryInterface $categoryRepository;

    /**
     * @param Closure|null $categoryResolver
     */
    public static function setCategoryRepositoryResolver(?Closure $categoryResolver = null): void
    {
        self::$categoryRepository = null;

        self::$categoryResolver = $categoryResolver;
    }

    /**
     * @return CategoryRepositoryInterface
     */
    public static function getCategoryRepository(): CategoryRepositoryInterface
    {
        return self::$categoryRepository ?? self::$categoryRepository = call_user_func(self::$categoryResolver);
    }
}

```

A.1.8.2 UserRepositoryFinderTrait

```

namespace App\MovieDomain\User\Repository;

use Closure;

trait UserRepositoryFinderTrait
{
    private static ?Closure $userRepositoryResolver;

    private static ?UserRepositoryInterface $userRepository;
}

```

```

/**
 * @param Closure|null $resolver
 */
public static function setUserRepositoryResolver(?Closure $resolver = null)
{
    static::$userRepository = null;

    static::$userRepositoryResolver = $resolver;
}

/**
 * @return UserRepositoryInterface
 */
public function getUserRepository(): UserRepositoryInterface
{
    return static::$userRepository ?? static::$userRepository = call_user_func(static::$userRepositoryResolver);
}

/**
 * @param UserRepositoryInterface $repository
 */
public static function setUserRepository(UserRepositoryInterface $repository)
{
    static::$userRepository = $repository;
}
}

```

A.1.8.3 RoleRepositoryFinderTrait

```

namespace App\MovieDomain\Role\Repository;

use Closure;

trait RoleRepositoryFinderTrait
{
    private static ?Closure $roleRepositoryResolver;

    private static ?RoleRepositoryInterface $roleRepository;

    /**
     * @param Closure|null $resolver
     */
    public static function setRoleRepositoryResolver(?Closure $resolver = null)
    {
        static::$roleRepository = null;

        static::$roleRepositoryResolver = $resolver;
    }

    /**
     * @return RoleRepositoryInterface
     */
    public function getRoleRepository(): RoleRepositoryInterface
    {
        return static::$roleRepository ?? static::$roleRepository = call_user_func(static::$roleRepositoryResolver);
    }

    /**
     * @param RoleRepositoryInterface $repository
     */
    public static function setRoleRepository(RoleRepositoryInterface $repository)

```

```

    {
        static::$roleRepository = $repository;
    }
}

```

A.1.9 Payloads

A.1.9.1 UserCreatePayload

```

namespace App\MovieDomain\User\Payload;

class UserCreatePayload
{
    /**
     * @param string $name
     * @param string $surname
     * @param string $email
     * @param string $password
     * @param string $emailConfirmationCode
     */
    private function __construct(
        private string $name,
        private string $surname,
        private string $email,
        private string $password,
        private string $emailConfirmationCode,
    ) {
    }

    /**
     * @param string $name
     * @param string $surname
     * @param string $email
     * @param string $password
     * @param string $emailConfirmationCode
     * @return static
     */
    public static function make(
        string $name,
        string $surname,
        string $email,
        string $password,
        string $emailConfirmationCode,
    ): self {
        return new self(
            name: $name,
            surname: $surname,
            email: $email,
            password: $password,
            emailConfirmationCode: $emailConfirmationCode
        );
    }

    /**
     * @return string
     */
    public function getName(): string
    {
        return $this->name;
    }

    /**

```

```

    * @return string
    */
    public function getSurname(): string
    {
        return $this->surname;
    }

    /**
     * @return string
     */
    public function getEmail(): string
    {
        return $this->email;
    }

    /**
     * @return string
     */
    public function getPassword(): string
    {
        return $this->password;
    }

    /**
     * @return string
     */
    public function getEmailConfirmationCode(): string
    {
        return $this->emailConfirmationCode;
    }
}

```

A.1.9.2 MovieCreatePayload

```

namespace App\MovieDomain\Movie\Payload;

use Carbon\Carbon;

class MovieCreatePayload
{
    /**
     * @param string $name
     * @param string|null $descriptionRating
     * @param string|null $descriptionSlogan
     * @param Carbon|null $descriptionScreeningDate
     * @param string|null $descriptionCountry
     * @param string|null $descriptionActors
     * @param string|null $shortDescription
     * @param string $storageMovieUrl
     * @param string $storageImageUrl
     */
    private function __construct(
        private string $name,
        private ?string $descriptionRating = null,
        private ?string $descriptionSlogan = null,
        private ?Carbon $descriptionScreeningDate = null,
        private ?string $descriptionCountry = null,
        private ?string $descriptionActors = null,
        private ?string $shortDescription = null,
        private string $storageMovieUrl,
        private string $storageImageUrl,
    ) {

```

```

}

/**
 * @param string $name
 * @param string|null $descriptionRating
 * @param string|null $descriptionSlogan
 * @param Carbon|null $descriptionScreeningDate
 * @param string|null $descriptionCountry
 * @param string|null $descriptionActors
 * @param string|null $shortDescription
 * @param string $storageMovieUrl
 * @param string $storageImageUrl
 * @return static
 */
public static function make(
    string $name,
    ?string $descriptionRating = null,
    ?string $descriptionSlogan = null,
    ?Carbon $descriptionScreeningDate = null,
    ?string $descriptionCountry = null,
    ?string $descriptionActors = null,
    ?string $shortDescription = null,
    string $storageMovieUrl,
    string $storageImageUrl,
): self {
    return new self(
        name: $name,
        descriptionRating: $descriptionRating,
        descriptionSlogan: $descriptionSlogan,
        descriptionScreeningDate: $descriptionScreeningDate,
        descriptionCountry: $descriptionCountry,
        descriptionActors: $descriptionActors,
        shortDescription: $shortDescription,
        storageMovieUrl: $storageMovieUrl,
        storageImageUrl: $storageImageUrl
    );
}

/**
 * @return string
 */
public function getName(): string
{
    return $this->name;
}

/**
 * @return string|null
 */
public function getDescriptionRating(): ?string
{
    return $this->descriptionRating;
}

/**
 * @return string|null
 */
public function getDescriptionSlogan(): ?string
{
    return $this->descriptionSlogan;
}

/**

```

```

    * @return Carbon|null
    */
    public function getDescriptionScreeningDate(): ?Carbon
    {
        return $this->descriptionScreeningDate;
    }

    /**
     * @return string|null
     */
    public function getDescriptionCountry(): ?string
    {
        return $this->descriptionCountry;
    }

    /**
     * @return string|null
     */
    public function getDescriptionActors(): ?string
    {
        return $this->descriptionActors;
    }

    /**
     * @return string|null
     */
    public function getShortDescription(): ?string
    {
        return $this->shortDescription;
    }

    /**
     * @return string
     */
    public function getStorageMovieUrl(): string
    {
        return $this->storageMovieUrl;
    }

    /**
     * @return string
     */
    public function getStorageImageUrl(): string
    {
        return $this->storageImageUrl;
    }
}

```

A.1.10 Handler

```

namespace App\Exceptions;

use App\Common\CanNotDeleteException;
use App\Common\NotFoundException;
use App\Exceptions\ErrorCodes;
use App\MovieDomain\Movie\Exception\InvalidImageUrl;
use App\MovieDomain\Movie\Exception\InvalidMovieUrl;
use App\MovieDomain\User\Exception\NonValidPasswordException;
use Exception;
use Illuminate\Foundation\Exceptions\Handler as ExceptionHandler;
use Illuminate\Http\JsonResponse;
use Illuminate\Http\Response;

```

use Throwable;

```
class Handler extends ExceptionHandler
{
  /**
   * @param \Illuminate\Http\Request $request
   * @param Throwable $e
   * @return JsonResponse
   */
  public function render($request, Throwable $e): JsonResponse
  {
    return match (true) {
      $e instanceof NonValidPasswordException => $this->mapNonValidPasswordException($e),
      $e instanceof NotFoundException => $this->mapNotFoundException($e),
      $e instanceof CanNotDeleteException => $this->mapCanNotDeleteException($e),
      $e instanceof InvalidMovieUrl => $this->mapInvalidMovieUrl($e),
      $e instanceof InvalidImageUrl => $this->mapInvalidImageUrl($e),
      default => parent::render($request, $e),
    };
  }

  /**
   * @param Exception|Throwable $e
   * @return JsonResponse
   */
  private function mapByDefault(Exception|Throwable $e): JsonResponse
  {
    return response()->json(
      [
        'message' => $e->getMessage(),
        'error_code' => $e->getCode(),
      ],
      Response::HTTP_INTERNAL_SERVER_ERROR
    );
  }

  /**
   * @param NonValidPasswordException $e
   * @return JsonResponse
   */
  private function mapNonValidPasswordException(NonValidPasswordException $e): JsonResponse
  {
    return response()->json([
      'message' => $e->getMessage(),
      'error' => ErrorCodes::non_valid_password()->value,
      'error_code' => ErrorCodes::NON_VALID_PASSWORD,
    ], Response::HTTP_NOT_ACCEPTABLE);
  }

  /**
   * @param NotFoundException $e
   * @return JsonResponse
   */
  private function mapNotFoundException(NotFoundException $e): JsonResponse
  {
    return response()->json([
      'message' => $e->getMessage(),
      'error' => ErrorCodes::not_found()->value,
      'error_code' => ErrorCodes::NOT_FOUND,
    ], Response::HTTP_BAD_REQUEST);
  }
}
```

```

* @param CanNotDeleteException $e
* @return JsonResponse
*/
private function mapCanNotDeleteException(CanNotDeleteException $e): JsonResponse
{
    return response()->json([
        'message' => $e->getMessage(),
        'error' => ErrorCodes::can_not_remove_model()->value,
        'error_code' => ErrorCodes::CAN_NOT_REMOVE_MODEL,
    ], Response::HTTP_BAD_REQUEST);
}

/**
* @param InvalidMovieUrl $e
* @return JsonResponse
*/
private function mapInvalidMovieUrl(InvalidMovieUrl $e): JsonResponse
{
    return response()->json([
        'message' => $e->getMessage(),
        'error' => ErrorCodes::invalid_movie_url()->value,
        'error_code' => ErrorCodes::INVALID_MOVIE_URL,
    ], Response::HTTP_BAD_REQUEST);
}

/**
* @param InvalidImageUrl $e
* @return JsonResponse
*/
private function mapInvalidImageUrl(InvalidImageUrl $e): JsonResponse
{
    return response()->json([
        'message' => $e->getMessage(),
        'error' => ErrorCodes::invalid_image_url()->value,
        'error_code' => ErrorCodes::INVALID_IMAGE_URL,
    ], Response::HTTP_BAD_REQUEST);
}
}

```

A.1.11 ErrorCodes

```

namespace App\Exceptions;

use Spatie\Enum\Enum;

/**
* @method static self NON_VALID_PASSWORD()
* @method static self NOT_FOUND()
* @method static self CAN_NOT_REMOVE_MODEL()
* @method static self INVALID_MOVIE_URL()
* @method static self INVALID_IMAGE_URL()
*/
class ErrorCodes extends Enum
{
    public const NON_VALID_PASSWORD = '1010';
    public const NOT_FOUND = '1011';
    public const CAN_NOT_REMOVE_MODEL = '1012';

    public const INVALID_MOVIE_URL = '1110';
    public const INVALID_IMAGE_URL = '1111';
}

```

A.1.12 Exceptions**A.1.12.1 CollectionNotFound**

```
namespace App\MovieDomain\Collection\Exception;

use App\Common\NotFoundException;

class CollectionNotFound extends NotFoundException
{
    protected $message = 'Not found movie collection.';
}
```

A.1.12.2 MovieNotFound

```
namespace App\MovieDomain\Movie\Exception;

use App\Common\NotFoundException;

class MovieNotFound extends NotFoundException
{
    protected $message = 'Not found movie';
}
```

A.1.12.3 InvalidImageUrl

```
namespace App\MovieDomain\Movie\Exception;

use Exception;

class InvalidImageUrl extends Exception
{
    protected $message = 'Invalid image url.';
}
```

A.1.12.4 InvalidMovieUrl

```
namespace App\MovieDomain\Movie\Exception;

use Exception;

class InvalidMovieUrl extends Exception
{
    protected $message = 'Invalid movie url';
}
```

A.1.12.5 NonValidPasswordException

```
namespace App\MovieDomain\User\Exception;

use Exception;

class NonValidPasswordException extends Exception
{
    protected $message = 'Non valid password';
}
```

A.1.12.6 UserNoFoundException

```

namespace App\MovieDomain\User\Exception;

use App\Common\NotFoundException;

class UserNotFoundException extends NotFoundException
{
    protected $message = 'Not found user';
}

```

A.1.13 Requests

A.1.13.1 CollectionCreateRequest

```

namespace App\Http\Request\Collection;

use App\Common\MovieMicroserviceRequest;

class CollectionCreateRequest extends MovieMicroserviceRequest
{
    /**
     * @inheritDoc
     */
    public function rules(): array
    {
        return [
            'user_id' => [
                'required',
                'int'
            ],
            'name' => [
                'required',
                'string',
            ],
        ];
    }

    /**
     * @return int
     */
    public function getUserId(): int
    {
        return $this->input('user_id');
    }

    /**
     * @return string
     */
    public function getName(): string
    {
        return $this->input('name');
    }
}

```

A.1.13.2 CollectionListRequest

```

namespace App\Http\Request\Collection;

use App\Common\MovieMicroserviceRequest;
use App\MovieDomain\Collection\CollectionType;
use Illuminate\Validation\Rule;
use Symfony\Component\Finder\Exception\AccessDeniedException;

```

```

class CollectionListRequest extends MovieMicroserviceRequest
{
    protected const PER_PAGE = 100;
    protected const PAGE = 1;

    public function rules(): array
    {
        return [
            'user_id' => [
                'nullable',
                'int',
            ],
            'type' => [
                'required',
                'string',
                Rule::in(CollectionType::toArray()),
            ],
            'collection_ids' => [
                'nullable',
                'array',
            ],
            'collection_ids.*' => [
                'int',
            ],
            'per_page' => [
                'nullable',
                'int',
            ],
            'page' => [
                'nullable',
                'int',
            ],
        ];
    }

    /**
     * @return int|null
     */
    public function getUserId(): ?int
    {
        return $this->input('user_id');
    }

    /**
     * @return string|null
     */
    public function getType(): ?string
    {
        return $this->input('type');
    }

    /**
     * @return array|null
     */
    public function getCollectionIds(): ?array
    {
        return $this->input('collection_ids');
    }

    /**
     * @return int
     */

```

```

public function getPage(): int
{
    return $this->input('page', self::PAGE);
}

/**
 * @return int
 */
public function getPerPage(): int
{
    return $this->input('per_page', self::PER_PAGE);
}
}

```

A.1.13.3 MovieCreateRequest

```

namespace App\Http\Request\Movie;

use App\Common\MovieMicroserviceRequest;
use Carbon\Carbon;

class MovieCreateRequest extends MovieMicroserviceRequest
{
    /**
     * @inheritDoc
     */
    public function rules(): array
    {
        return [
            'user_id' => [
                'required',
                'int',
            ],
            'name' => [
                'required',
                'string',
            ],
            'description' => [
                'required',
                'array',
            ],
            'description.rating' => [
                'numeric'
            ],
            'description.slogan' => [
                'string'
            ],
            'description.screening_date' => [
                'string',
            ],
            'description.country' => [
                'string', //Todo maybe add validation on country
            ],
            'description.actors' => [
                'string',
            ],
            'description.shortDescription' => [
                'string',
            ],
            'storage_movie_link' => [
                'required',
                'string',
            ],

```

```

        ],
        'storage_image_link' => [
            'required',
            'string',
        ]
    ];
}

/**
 * @return int
 */
public function getUserId(): int
{
    return $this->input('user_id');
}

/**
 * @return string
 */
public function getName(): string
{
    return $this->input('name');
}

/**
 * @return array
 */
public function getDescription(): array
{
    return $this->input('description');
}

/**
 * @return string|null
 */
public function getDescriptionRating(): ?string
{
    return $this->input('description.rating');
}

/**
 * @return string|null
 */
public function getDescriptionSlogan(): ?string
{
    return $this->input('description.slogan');
}

/**
 * @return Carbon|null
 */
public function getDescriptionScreeningDate(): ?Carbon
{
    $date = $this->input('description.screening_date');

    return $date !== null ? Carbon::make($date) : null;
}

/**
 * @return string|null
 */
public function getDescriptionCountry(): ?string
{

```

```

        return $this->input('description.country');
    }

    /**
     * @return string|null
     */
    public function getDescriptionActors(): ?string
    {
        return $this->input('description.actors');
    }

    /**
     * @return string|null
     */
    public function getShortDescription(): ?string
    {
        return $this->input('description.shortDescription');
    }

    /**
     * @return string
     */
    public function getStorageMovieLink(): string
    {
        return $this->input('storage_movie_link');
    }

    /**
     * @return string
     */
    public function getStorageImageLink(): string
    {
        return $this->input('storage_image_link');
    }
}

```

A.1.13.4 MovieCategoriesRequest

```

namespace App\Http\Request\Movie;

use App\Common\MovieMicroserviceRequest;
use Illuminate\Support\Collection;

class MovieCategoriesRequest extends MovieMicroserviceRequest
{
    /**
     * @inheritDoc
     */
    public function rules(): array
    {
        return [
            'user_id' => [
                'required',
                'int',
            ],
            'category_ids' => [
                'nullable',
                'array',
            ],
            'category_ids.*' => [
                'int',
            ],
        ],
    }
}

```

```

    ];
}

/**
 * @return int
 */
public function getUserId(): int
{
    return $this->input('user_id');
}

/**
 * @return Collection<int>
 */
public function getCategoryIds(): Collection
{
    return collect($this->input('category_ids', []));
}
}

```

A.1.13.5 UserGetRequest

```

namespace App\Http\Request\User;

use App\Common\MovieMicroserviceRequest;

class UserGetRequest extends MovieMicroserviceRequest
{
    /**
     * @inheritDoc
     */
    public function rules(): array
    {
        return [
            'password' => [
                'required',
                'string',
                'min:10',
            ],
            'email' => [
                'required',
                'email',
            ],
        ];
    }

    /**
     * @return string
     */
    public function getPassword(): string
    {
        return $this->get('password');
    }

    /**
     * @return string
     */
    public function getEmail(): string
    {
        return $this->get('email');
    }
}

```

A.1.14 Resources

A.1.14.1 UserResource

```
namespace App\Http\Resource\User;

use App\Common\MovieMicroserviceResource;
use App\MovieDomain\Role\Role;
use App\MovieDomain\User\User;

/**
 * @mixin User
 */
class UserResource extends MovieMicroserviceResource
{
    /**
     * @inheritDoc
     */
    public function toArray($request): array
    {
        return [
            'id' => $this->getId(),
            'name' => $this->getName(),
            'surname' => $this->getSurname(),
            'email' => $this->getEmail(),
            'roles' => $this
                ->getRoles()
                ?->map(function (Role $role) {
                    return $role->getRole()->value;
                })
                ->toArray(),
            'access_token' => $this->getAccessToken(),
            'create_date' => $this->getCreateDate()->format('Y-m-d H:i:s'),
        ];
    }
}
```

A.1.14.2 MovieResource

```
namespace App\Http\Resource\Movie;

use App\Common\MovieMicroserviceResource;
use App\Http\Resource\Category\CategoryResource;
use App\MovieDomain\Category\Category;
use App\MovieDomain\Movie\Movie;

/**
 * @mixin Movie
 */
class MovieResource extends MovieMicroserviceResource
{
    /**
     * @param \Illuminate\Http\Request $request
     * @return array
     * @throws \GuzzleHttp\Exception\GuzzleException
     */
    public function toArray($request): array
    {
        return [
            'id' => $this->getId(),
        ];
    }
}
```

```

        'name' => $this->getName(),
        'description' => [
            'screening_date' => $this->getDescription()->getScreeningDate()->format('Y-m-d'),
            'short_description' => $this->getDescription()->getShortDescription(),
            'rating' => $this->getDescription()->getRating(),
            'actors' => $this->getDescription()->getActors(),
            'slogan' => $this->getDescription()->getSlogan(),
            'country' => $this->getDescription()->getCountry(),
        ],
        'categories' => CategoryResource::collection($this->getCategories()),
        'storage_image_url' => $this->getFullImagePath(),
        'storage_movie_url' => $this->getFullMoviePath(),
    ];
}
}

```

A.1.14.3 RoleResource

```

namespace App\Http\Resource\User;

use App\Common\MovieMicroserviceResource;
use App\MovieDomain\Role\Role;

/**
 * @mixin Role
 */
class RoleResource extends MovieMicroserviceResource
{
    /**
     * @inheritDoc
     */
    public function toArray($request): array
    {
        return [
            'name' => $this->getRole()->value,
        ];
    }
}

```

A.1.14.4 CaetgoryResource

```

namespace App\Http\Resource\Category;

use App\Common\MovieMicroserviceResource;
use App\MovieDomain\Category\Category;

/**
 * @mixin Category
 */
class CategoryResource extends MovieMicroserviceResource
{
    /**
     * @inheritDoc
     */
    public function toArray($request): array
    {
        return [
            'id' => $this->getId(),
            'name' => $this->getName(),
            'movie_ids' => $this->getMovieIds()->toArray(),
        ];
    }
}

```

```

    }
}

```

A.1.14.5 CollectionResource

```

namespace App\Http\Resource\Collection;

use App\Common\MovieMicroserviceResource;
use App\MovieDomain\Collection\Collection;

/**
 * @mixin Collection
 */
class CollectionResource extends MovieMicroserviceResource
{
    public function toArray($request): array
    {
        return [
            'id' => $this->getId(),
            'user_id' => $this->getUserId(),
            'name' => $this->getName(),
            'type' => $this->getType(),
            'movie_ids' => $this->getMovieIds()?->toArray(),
        ];
    }
}

```

A.1.14.6 CategoryListResource

```

namespace App\Http\Resource\Category;

use App\Common\MovieMicroserviceResource;
use App\MovieDomain\Category\CategoryCollection;

/**
 * @mixin CategoryCollection
 */
class CategoryListResource extends MovieMicroserviceResource
{
    public function toArray($request): array
    {
        return [
            'items' => CategoryResource::collection($this->getItems()),
            'temp' => [
                'current_page' => $this->getPage(),
                'last_page' => $this->getLastPage(),
                'per_page' => $this->getPerPage(),
                'total' => $this->getTotal(),
            ],
        ];
    }
}

```

A.1.15 Filters

A.1.15.1 CollectionFilter

```

namespace App\MovieDomain\Collection\Filter;

use App\Common\Filter;
use App\MovieDomain\Collection\CollectionType;

```

```

use Illuminate\Support\Collection;

class CollectionFilter extends Filter
{
    /**
     * @param int|null $userId
     * @param int|null $movieId
     * @param int|null $withoutUserId
     * @param CollectionType|null $type
     * @param Collection|null $types
     * @param Collection<int>|null $collectionIds
     */
    private function __construct(
        private ?int $userId = null,
        private ?int $movieId = null,
        private ?int $withoutUserId = null,
        private ?CollectionType $type = null,
        private ?Collection $types = null,
        private ?Collection $collectionIds = null,
    ) {
    }

    /**
     * @param int|null $userId
     * @param int|null $movieId
     * @param CollectionType|null $type
     * @param Collection|null $types
     * @param Collection<int>|null $collectionIds
     * @return static
     */
    public static function make(
        ?int $userId = null,
        ?int $movieId = null,
        ?int $withoutUserId = null,
        ?CollectionType $type = null,
        ?Collection $types = null,
        ?Collection $collectionIds = null,
    ): self {
        return new self(
            userId: $userId,
            movieId: $movieId,
            withoutUserId: $withoutUserId,
            type: $type,
            types: $types,
            collectionIds: $collectionIds
        );
    }

    /**
     * @return int|null
     */
    public function getUserId(): ?int
    {
        return $this->userId;
    }

    /**
     * @return CollectionType|null
     */
    public function getType(): ?CollectionType
    {
        return $this->type;
    }
}

```

```

/**
 * @return Collection<int>|null
 */
public function getCollectionIds(): ?Collection
{
    return $this->collectionIds;
}

/**
 * @return int|null
 */
public function getMovieId(): ?int
{
    return $this->movieId;
}

/**
 * @param CollectionType|null $type
 * @return self
 */
public function setType(?CollectionType $type): self
{
    $this->type = $type;

    return $this;
}

/**
 * @param Collection|null $types
 * @return self
 */
public function setTypes(?Collection $types): self
{
    $this->types = $types;

    return $this;
}

/**
 * @return Collection|null
 */
public function getTypes(): ?Collection
{
    return $this->types;
}

/**
 * @return int|null
 */
public function getWithoutUserId(): ?int
{
    return $this->withoutUserId;
}

/**
 * @return array
 */
public function getTypesValuesArray(): array
{
    if (empty($this->getTypes())) {
        return [];
    }
}

```

```

    return $this->getTypes()->map(fn (CollectionType $type) => $type->value)->toArray();
  }
}

```

A.1.15.2 MovieFilter

```

namespace App\MovieDomain\Movie\Filter;

use App\Common\Filter;
use App\MovieDomain\Collection\CollectionType;
use Illuminate\Support\Collection;

class MovieFilter extends Filter
{
    /**
     * @param int|null $userId
     * @param CollectionType|null $collectionType
     * @param Collection|null $categoryIds
     * @param Collection|null $collectionIds
     */
    private function __construct(
        private ?int $userId = null,
        private ?CollectionType $collectionType = null,
        private ?Collection $categoryIds = null,
        private ?Collection $collectionIds = null,
    ) {
    }

    /**
     * @param int|null $userId
     * @param CollectionType|null $collectionType
     * @param Collection<int>|null $categoryIds
     * @param Collection|null $collectionIds
     * @return self
     */
    public static function make(
        ?int $userId = null,
        ?CollectionType $collectionType = null,
        ?Collection $categoryIds = null,
        ?Collection $collectionIds = null,
    ): self {
        return new self(
            userId: $userId,
            collectionType: $collectionType,
            categoryIds: $categoryIds,
            collectionIds: $collectionIds,
        );
    }

    /**
     * @return Collection<int>|null
     */
    public function getCategoryIds(): ?Collection
    {
        return $this->categoryIds;
    }

    /**
     * @return Collection<int>|null
     */
    public function getCollectionIds(): ?Collection

```

```

    {
        return $this->collectionIds;
    }

    /**
     * @return int|null
     */
    public function getUserId(): ?int
    {
        return $this->userId;
    }

    /**
     * @return CollectionType|null
     */
    public function getCollectionType(): ?CollectionType
    {
        return $this->collectionType;
    }
}

```

A.1.16 ServiceProviders

A.1.16.1 CategoryServiceProvider

```

namespace App\Providers;

use App\MovieDomain\Category\Repository\CategoryRepository;
use App\MovieDomain\Category\Repository\CategoryRepositoryInterface;
use App\MovieDomain\Category\Service\CategoryService;
use App\MovieDomain\Category\Service\CategoryServiceInterface;
use App\MovieDomain\MovieCategory\Repository\MovieCategoryRepository;
use App\MovieDomain\MovieCategory\Repository\MovieCategoryRepositoryInterface;
use Illuminate\Support\ServiceProvider;

class CategoryServiceProvider extends ServiceProvider
{
    public function register()
    {
        $this->registerCategoryService();
        $this->registerCategoryRepository();
        $this->registerMovieCategoryRepository();
    }

    private function registerCategoryService(): void
    {
        $this->app->singleton(CategoryServiceInterface::class, CategoryService::class);
    }

    private function registerCategoryRepository(): void
    {
        $this->app->singleton(CategoryRepositoryInterface::class, CategoryRepository::class);
    }

    private function registerMovieCategoryRepository(): void
    {
        $this->app->singleton(MovieCategoryRepositoryInterface::class, MovieCategoryRepository::class);
    }
}

```

A.1.16.2 CollectionServiceProvider

```

namespace App\Providers;

use App\MovieDomain\Collection\Repository\CollectionRepository;
use App\MovieDomain\Collection\Repository\CollectionRepositoryInterface;
use App\MovieDomain\Collection\Service\CollectionService;
use App\MovieDomain\Collection\Service\CollectionServiceInterface;
use App\MovieDomain\MovieCollection\Repository\MovieCollectionRepository;
use App\MovieDomain\MovieCollection\Repository\MovieCollectionRepositoryInterface;
use Illuminate\Support\ServiceProvider;

class CollectionServiceProvider extends ServiceProvider
{
    /**
     * @inheritDoc
     */
    public function register()
    {
        $this->registerCollectionService();

        $this->registerCollectionRepository();

        $this->registerMovieCollectionRepository();
    }

    private function registerCollectionService(): void
    {
        $this->app->singleton(CollectionServiceInterface::class, CollectionService::class);
    }

    private function registerCollectionRepository(): void
    {
        $this->app->singleton(CollectionRepositoryInterface::class, CollectionRepository::class);
    }

    private function registerMovieCollectionRepository(): void
    {
        $this->app->singleton(MovieCollectionRepositoryInterface::class, MovieCollectionRepository::class);
    }
}

```

A.1.16.3 MovieServiceProvider

```

namespace App\Providers;

use App\MovieDomain\Category\Repository\CategoryRepositoryInterface;
use App\MovieDomain\Collection\Repository\CollectionRepositoryInterface;
use App\MovieDomain\Movie\Movie;
use App\MovieDomain\Movie\Repository\MovieRepository;
use App\MovieDomain\Movie\Repository\MovieRepositoryInterface;
use App\MovieDomain\Movie\Service\MovieService;
use App\MovieDomain\Movie\Service\MovieServiceInterface;
use App\MovieDomain\Storage\BestMovieCachedStorageServiceInterface;
use App\MovieDomain\Storage\BestMovieCachedStorageService;
use BestMovie\Common\BestMovieStorage\Service\BestMovieStorageServiceInterface;
use Illuminate\Support\ServiceProvider;

class MovieServiceProvider extends ServiceProvider
{
    /**
     * @inheritDoc
     */
    public function register()
    {

```

```

{
    $this->registerMovieService();

    $this->registerRepository();

    $this->registerBestMovieCachedStorageServiceInterface();
}

/**
 * @inheritDoc
 */
public function boot()
{
    Movie::setCategoryRepositoryResolver(function (): CategoryRepositoryInterface {
        return $this->app[CategoryRepositoryInterface::class];
    });

    Movie::setCollectionRepositoryResolver(function (): CollectionRepositoryInterface {
        return $this->app[CollectionRepositoryInterface::class];
    });

    Movie::setBestMovieCachedStorageServiceResolver(function (): BestMovieCachedStorageServiceInterface {
        return $this->app[BestMovieCachedStorageServiceInterface::class];
    });
}

private function registerMovieService(): void
{
    $this->app->singleton(MovieServiceInterface::class, MovieService::class);
}

private function registerRepository()
{
    $this->app->singleton(MovieRepositoryInterface::class, MovieRepository::class);
}

private function registerBestMovieCachedStorageServiceInterface(): void
{
    $this->app->singleton(
        BestMovieCachedStorageServiceInterface::class,
        BestMovieCachedStorageService::class
    );
}
}

```

A.1.16.4 RoleServiceProvider

```

namespace App\Providers;

use App\MovieDomain\Role\Repository\RoleRepository;
use App\MovieDomain\Role\Repository\RoleRepositoryInterface;
use App\MovieDomain\Role\Service\RoleService;
use App\MovieDomain\Role\Service\RoleServiceInterface;
use Illuminate\Support\ServiceProvider;

class RoleServiceProvider extends ServiceProvider
{
    /**
     * @return void
     */
    public function register(): void

```

```

    {
        // TODO parse on methods
        $this->app->singleton(RoleServiceInterface::class, RoleService::class);
        $this->app->singleton(RoleRepositoryInterface::class, RoleRepository::class);
    }
}

```

A.2 BestMovieStorage

A.2.1 API контракти

```

use App\Http\Controllers\MediaController;
use Illuminate\Support\Facades\Route;

/*
|-----
| API Routes
|-----
|
| Here is where you can register API routes for your application. These
| routes are loaded by the RouteServiceProvider within a group which
| is assigned the "api" middleware group. Enjoy building your API!
|
*/

Route::controller(MediaController::class)
    ->prefix('media-uploader')
    ->group(function () {

        Route::middleware('access_token')->group(function () {
            Route::post('/', 'upload');
        });

        // TODO add microservices url check
        Route::get('/', 'getPath');
        Route::get('/validate', 'validatePath');
    });

```

A.2.2 MediaController

```

namespace App\Http\Controllers;

use App\Http\Request\Media\MediaGetPathRequest;
use App\Http\Request\Media\MediaUploadRequest;
use App\Http\Request\Media\MediaValidatePathRequest;
use App\Http\Resource\Media\MediaUploadResource;
use App\StorageDomain\Media\MediaTypeEnum;
use App\StorageDomain\Media\Payload\MediaUploadPayload;
use App\StorageDomain\Media\Service\MediaServiceInterface;
use BestMovie\Common\BestMovieMicroservice\Service\BestMovieServiceInterface;
use Illuminate\Auth\AuthenticationException;
use Illuminate\Http\JsonResponse;
use Illuminate\Http\Response;
use Mockery\Exception;
use Throwable;

class MediaController extends Controller
{
    /**
     * @param MediaServiceInterface $mediaService
     * @param BestMovieServiceInterface $bestMovieService
     */
}

```

```

*/
public function __construct(
    private MediaServiceInterface $mediaService,
    private BestMovieServiceInterface $bestMovieService,
) {
}

/**
 * @param MediaUploadRequest $request
 * @return JsonResponse
 * @throws \App\StorageDomain\Media\Exception\UnsupportedMediaType
 * @throws AuthenticationException
 */
public function upload(MediaUploadRequest $request): JsonResponse
{
    try {
        $user = $this->bestMovieService->refreshUser($request->getAuthHeader(), $request->getUserId());

        if (!$user->isOk() || !$user->isAdmin()) {
            throw new AuthenticationException();
        };
    } catch (Exception|Throwable $e) {
        throw new AuthenticationException();
    }

    $payload = MediaUploadPayload::make(
        userId: $request->getUserId(),
        type: MediaTypeEnum::tryFrom($request->getMediaType()),
        file: $request->getMediaFile()
    );

    return response()->json([
        'status' => Response::HTTP_CREATED,
        'data' => MediaUploadResource::make($this->mediaService->upload($payload))
    ], Response::HTTP_CREATED);
}

/**
 * @param MediaGetPathRequest $request
 * @return JsonResponse
 */
public function getPath(MediaGetPathRequest $request): JsonResponse
{
    return response()->json([
        'status' => Response::HTTP_CREATED,
        'data' => [
            'url' => $this->mediaService->getFileUrl($request->getPath()),
        ]
    ], Response::HTTP_CREATED);
}

/**
 * @param MediaValidatePathRequest $request
 * @return JsonResponse
 */
public function validatePath(MediaValidatePathRequest $request): JsonResponse
{
    return response()->json([
        'status' => Response::HTTP_OK,
        'result' => $this->mediaService->validatePath($request->getPath()),
    ], Response::HTTP_OK);
}

```

```
}
```

A.2.3 FirebaseMediaStorage

```
namespace App\StorageDomain\Media\Storage;

use Illuminate\Http\UploadedFile;

interface MediaStorageInterface
{
    /**
     * @param string $folderPath
     * @param UploadedFile $file
     * @param string $name
     * @return string
     */
    public function uploadFile(string $folderPath, UploadedFile $file, string $name): string;

    /**
     * @param string $path
     * @return string
     */
    public function getFileUrl(string $path): string;

    /**
     * @param string $path
     * @return bool
     */
    public function isPathValid(string $path): bool;
}

namespace App\StorageDomain\Media\Storage;

use Carbon\Carbon;
use Google\Cloud\Core\Exception\NotFoundException;
use GuzzleHttp\Exception\ClientException;
use Illuminate\Http\UploadedFile;
use Kreait\Laravel\Firebase\Facades\Firebase;

class FirebaseMediaStorage implements MediaStorageInterface
{
    /**
     * @param string $folderPath
     * @param UploadedFile $file
     * @param string $name
     * @return string
     */
    public function uploadFile(string $folderPath, UploadedFile $file, string $name): string
    {
        $image = Firebase::storage()
            ->getBucket()
            ->upload($file->getContent(), ['name' => $folderPath . $name]);

        return $image->info()['name'];
    }

    /**
     * @param string $path
     * @return string
     */
    public function getFileUrl(string $path): string
```

```

    {
        return Firebase::storage()->getBucket()->object($path)->signedUrl(Carbon::tomorrow());
    }

    /**
     * @param string $path
     * @return bool
     */
    public function isPathValid(string $path): bool
    {
        try {
            Firebase::storage()->getBucket()->exists();
        } catch (NotFoundException $e) {
            return false;
        }

        return true;
    }
}

```

A.2.4 FirebaseMediaService

```

namespace App\StorageDomain\Media\Service;

use App\StorageDomain\Media\Exception\UnsupportedMediaType;
use App\StorageDomain\Media\Payload\MediaSavedPayload;
use App\StorageDomain\Media\Payload\MediaUploadPayload;
use Exception;

interface MediaServiceInterface
{
    /**
     * @param MediaUploadPayload $payload
     * @return MediaSavedPayload
     * @throws UnsupportedMediaType
     * @throws Exception
     */
    public function upload(MediaUploadPayload $payload): MediaSavedPayload;

    /**
     * @param string $path
     * @return string
     */
    public function getFileUrl(string $path): string;

    /**
     * @param string $path
     * @return bool
     */
    public function validatePath(string $path): bool;
}

namespace App\StorageDomain\Media\Service;

use App\StorageDomain\Media\Exception\UnsupportedMediaType;
use App\StorageDomain\Media\MediaTypeEnum;
use App\StorageDomain\Media\Payload\MediaSavedPayload;
use App\StorageDomain\Media\Payload\MediaUploadPayload;
use App\StorageDomain\Media\RestrictionFactory\MediaRestrictionFactoryInterface;
use App\StorageDomain\Media\Storage\MediaStorageInterface;
use Carbon\Carbon;
use Exception;

```

```

class FirebaseMediaService implements MediaServiceInterface
{
    /**
     * @param MediaRestrictionFactoryInterface $mediaRestrictionFactory
     * @param MediaStorageInterface $mediaStorage
     */
    public function __construct(
        private MediaRestrictionFactoryInterface $mediaRestrictionFactory,
        private MediaStorageInterface $mediaStorage
    ) {
    }

    /**
     * @inheritdoc
     */
    public function upload(MediaUploadPayload $payload): MediaSavedPayload
    {
        $this->mediaRestrictionFactory
            ->make($payload->getType())
            ->validate($payload->getFile());

        $folderPath = $this->definePath($payload->getType());
        $name = $this->generateName($payload->getFile()->getClientOriginalExtension());

        $path = $this->mediaStorage->uploadFile($folderPath, $payload->getFile(), $name);

        return MediaSavedPayload::make($payload->getType(), $path);
    }

    /**
     * @inheritdoc
     */
    public function getFileUrl(string $path): string
    {
        return $this->mediaStorage->getFileUrl($path);
    }

    /**
     * @param MediaTypeEnum $type
     * @return string
     * @throws UnsupportedMediaType
     */
    private function definePath(MediaTypeEnum $type): string
    {
        return match ($type->value) {
            MediaTypeEnum::MOVIE_IMAGE => '/movie/images/',
            MediaTypeEnum::MOVIE_VIDEO => '/movie/video/',
            default => throw new UnsupportedMediaType()
        };
    }

    /**
     * @param string $fileExtension
     * @return string
     * @throws Exception
     */
    private function generateName(string $fileExtension): string
    {
        return sprintf(
            '%s%s%s.%s',
            Carbon::now()->format('Ymd'),
            intval(abs(time() / rand(1, 200)))
        );
    }
}

```

```

        bin2hex(random_bytes(4)),
        $fileExtension
    );
}

/**
 * @inheritDoc
 */
public function validatePath(string $path): bool
{
    return $this->mediaStorage->isPathValid($path);
}
}

```

A.2.5 Payloads

A.2.5.1 MediaSavedPayload

```

namespace App\StorageDomain\Media\Payload;

use App\StorageDomain\Media\MediaTypeEnum;

class MediaSavedPayload
{
    /**
     * @param MediaTypeEnum $mediaType
     * @param string $url
     */
    private function __construct(
        private MediaTypeEnum $mediaType,
        private string $url
    ) {
    }

    /**
     * @param MediaTypeEnum $mediaType
     * @param string $url
     * @return MediaSavedPayload
     */
    public static function make(MediaTypeEnum $mediaType, string $url): MediaSavedPayload
    {
        return new MediaSavedPayload($mediaType, $url);
    }

    /**
     * @return MediaTypeEnum
     */
    public function getMediaType(): MediaTypeEnum
    {
        return $this->mediaType;
    }

    /**
     * @return string
     */
    public function getUrl(): string
    {
        return $this->url;
    }
}

```

A.2.5.2 MediaUploadPayload

```

namespace App\StorageDomain\Media\Payload;

use App\StorageDomain\Media\MediaTypeEnum;
use Illuminate\Http\UploadedFile;

class MediaUploadPayload
{
    /**
     * @param int $userId
     * @param MediaTypeEnum $type
     * @param UploadedFile $file
     */
    private function __construct(
        private int $userId,
        private MediaTypeEnum $type,
        private UploadedFile $file,
    ) {
    }

    /**
     * @param int $userId
     * @param MediaTypeEnum $type
     * @param UploadedFile $file
     * @return static
     */
    public static function make(
        int $userId,
        MediaTypeEnum $type,
        UploadedFile $file,
    ): self {
        return new self(
            userId: $userId,
            type: $type,
            file: $file
        );
    }

    /**
     * @return int
     */
    public function getUserId(): int
    {
        return $this->userId;
    }

    /**
     * @return MediaTypeEnum
     */
    public function getType(): MediaTypeEnum
    {
        return $this->type;
    }

    /**
     * @return UploadedFile
     */
    public function getFile(): UploadedFile
    {
        return $this->file;
    }
}

```

```

    }
}

```

A.2.6 Factory

A.2.6.1 MediaRestrictionFactory

```

namespace App\StorageDomain\Media\RestrictionFactory;

use App\StorageDomain\Media\Exception\UnknownMediaRestrictionException;
use App\StorageDomain\Media\MediaTypeEnum;

class MediaRestrictionFactory implements MediaRestrictionFactoryInterface
{
    /**
     * @param MediaTypeEnum $type
     * @return MediaRestrictionInterface
     * @throws UnknownMediaRestrictionException
     */
    public function make(MediaTypeEnum $type): MediaRestrictionInterface
    {
        $restriction = 'make' . str_replace('_', ' ', $type) . 'restriction';

        if (!method_exists($this, $restriction)) {
            throw new UnknownMediaRestrictionException();
        }

        return $this->{$restriction}();
    }

    /**
     * @return MovieImageRestriction
     */
    private function makeMovieImageRestriction(): MovieImageRestriction
    {
        return new MovieImageRestriction();
    }

    /**
     * @return MovieVideoRestriction
     */
    private function makeMovieVideoRestriction(): MovieVideoRestriction
    {
        return new MovieVideoRestriction();
    }
}

```

A.2.6.2 MediaRestrictionFactoryInterface

```

namespace App\StorageDomain\Media\RestrictionFactory;

use App\StorageDomain\Media\MediaTypeEnum;

interface MediaRestrictionFactoryInterface
{
    /**
     * @param MediaTypeEnum $mediaType
     * @return MediaRestrictionInterface
     */
    public function make(MediaTypeEnum $mediaType): MediaRestrictionInterface;
}

```

```
}
```

A.2.6.3 MediaRestrictionInterface

```
namespace App\StorageDomain\Media\RestrictionFactory;

use App\StorageDomain\Media\Exception\ValidationMediaException;
use App\StorageDomain\Media\Exception\ValidatorCreationFailedException;
use Illuminate\Http\UploadedFile;

interface MediaRestrictionInterface
{
    /**
     * @param \Illuminate\Http\UploadedFile $file
     * @throws ValidationMediaException
     * @throws ValidatorCreationFailedException
     */
    public function validate(UploadedFile $file): void;
}
```

A.2.6.4 MovieImageRestriction

```
namespace App\StorageDomain\Media\RestrictionFactory;

use App\StorageDomain\Media\Exception\ValidationMediaException;
use App\StorageDomain\Media\Exception\ValidatorCreationFailedException;
use App\StorageDomain\Media\MediaUploadMimeType;
use Exception;
use Illuminate\Http\UploadedFile;
use Illuminate\Support\Facades\Validator;

class MovieImageRestriction implements MediaRestrictionInterface
{
    /**
     * @inheritDoc
     */
    public function validate(UploadedFile $file): void
    {
        try {
            Validator::extend(
                'file_extension',
                function($attribute, $file, $parameters) {
                    return in_array($file->getClientOriginalExtension(), $parameters);
                },
                sprintf(
                    'The file extension must be %s, but was - %s.',
                    MediaUploadMimeType::png(),
                    $file->getClientOriginalExtension()
                )
            );

            $validator = Validator::make(
                [
                    'file' => $file,
                ],
                [
                    'file' => [
                        sprintf(
                            'mimes:%s,%s',
                            MediaUploadMimeType::png(),
                            MediaUploadMimeType::jpeg(),
                        ),
                    ],
                ],
            );
        }
    }
}
```

```

        'dimensions:max_width=1920,max_height=1920',
        sprintf(
            'file_extension:%s,%s',
            MediaUploadMimeType::png(),
            MediaUploadMimeType::jpeg(),
        ),
    ],
],
[
    'file.mimes' => sprintf(
        'The file mime type must be - %s, but was %s.',
        MediaUploadMimeType::png(),
        $file->getMimeType()
    ),
    'file.dimensions' => 'The file has invalid image dimensions.',
]
);
} catch (Exception $exception) {
    throw new ValidatorCreationFailedException();
}

if ($validator->fails()) {
    throw new ValidationMediaException($validator->getMessageBag()->all());
}
}
}

```

A.2.6.5 MovieVideoRestriction

```

namespace App\StorageDomain\Media\RestrictionFactory;

use App\StorageDomain\Media\Exception\ValidationMediaException;
use App\StorageDomain\Media\Exception\ValidatorCreationFailedException;
use App\StorageDomain\Media\MediaUploadMimeType;
use Exception;
use Illuminate\Http\UploadedFile;
use Illuminate\Support\Facades\Validator;

class MovieVideoRestriction implements MediaRestrictionInterface
{
    /**
     * @inheritDoc
     */
    public function validate(UploadedFile $file): void
    {
        try {
            Validator::extend(
                'file_extension',
                function($attribute, $file, $parameters) {
                    return in_array($file->getClientOriginalExtension(), $parameters);
                },
                sprintf(
                    'The file extension must be %s, but was - %s.',
                    MediaUploadMimeType::mp4(),
                    $file->getClientOriginalExtension()
                )
            );

            $validator = Validator::make(
                [
                    'file' => $file,
                ],
            );
        }
    }
}

```

```

        [
            'file' => [
                sprintf(
                    'mimes:%s',
                    MediaUploadMimeType::mp4(),
                ),
                sprintf(
                    'file_extension:%s',
                    MediaUploadMimeType::mp4(),
                ),
            ],
        ],
        [
            'file.mimes' => sprintf(
                'The file mime type must be - %s, but was %s.',
                MediaUploadMimeType::mp4(),
                $file->getMimeType()
            ),
            'file.dimensions' => 'The file has invalid image dimensions.',
        ]
    );
} catch (Exception $exception) {
    throw new ValidatorCreationFailedException();
}

if ($validator->fails()) {
    throw new ValidationMediaException($validator->getMessageBag()->all());
}
}
}

```

A.2.7 Error Codes

```

namespace App\Exceptions;

use Spatie\Enum\Enum;

/**
 * @method static self MEDIA_UNKNOWN_RESTRICTION()
 * @method static self MEDIA_VALIDATION()
 * @method static self MEDIA_VALIDATION_CREATION()
 * @method static self MEDIA_UNSUPPORTED_TYPE()
 * @method static self UNAUTHORIZED()
 */
class ErrorCodes extends Enum
{
    public const MEDIA_UNKNOWN_RESTRICTION = '1100';
    public const MEDIA_VALIDATION = '1101';
    public const MEDIA_VALIDATION_CREATION = '1102';
    public const MEDIA_UNSUPPORTED_TYPE = '1103';

    public const UNAUTHORIZED = '2101';
}

```

A.2.8 Exceptions

A.2.8.1 UnknownMediaRestrictionException

```

namespace App\StorageDomain\Media\Exception;

use Exception;

```

```
class UnknownMediaRestrictionException extends Exception
{
    protected $message = 'Unknown media restriction';
}
```

A.2.8.2 UnsupportedMediaType

```
namespace App\StorageDomain\Media\Exception;

use Exception;

class UnsupportedMediaType extends Exception
{
    protected $message = 'Unsupported media type';
}
```

A.2.8.3 ValidationMediaException

```
namespace App\StorageDomain\Media\Exception;

use Mockery\Exception;
use Throwable;

class ValidationMediaException extends Exception
{
    /**
     * @param array $messages
     * @param int $code
     * @param Throwable|null $previous
     */
    public function __construct(array $messages = [], $code = 0, Throwable $previous = null)
    {
        parent::__construct(implode(' ', $messages), $code, $previous);
    }
}
```

A.2.8.4 ValidatorCreationFailedException

```
namespace App\StorageDomain\Media\Exception;

use Mockery\Exception;

class ValidatorCreationFailedException extends Exception
{
    protected $message = 'Validator creation failed.';
}
```

A.2.9 MediaServiceProvider

```
namespace App\Providers;

use App\StorageDomain\Media\RestrictionFactory\MediaRestrictionFactory;
use App\StorageDomain\Media\RestrictionFactory\MediaRestrictionFactoryInterface;
use App\StorageDomain\Media\Service\FirebaseMediaService;
use App\StorageDomain\Media\Service\MediaServiceInterface;
use App\StorageDomain\Media\Storage\FirebaseMediaStorage;
use App\StorageDomain\Media\Storage\MediaStorageInterface;
use Illuminate\Support\ServiceProvider;
```

```

class MediaServiceProvider extends ServiceProvider
{
    /**
     * @inheritDoc
     */
    public function register()
    {
        $this->registerMediaRestrictionFactory();

        $this->registerMediaStorage();

        $this->registerMediaService();
    }

    private function registerMediaService(): void
    {
        $this->app->singleton(MediaServiceInterface::class, FirebaseMediaService::class);
    }

    private function registerMediaStorage(): void
    {
        $this->app->singleton(MediaStorageInterface::class, FirebaseMediaStorage::class);
    }

    private function registerMediaRestrictionFactory()
    {
        $this->app->singleton(MediaRestrictionFactoryInterface::class, MediaRestrictionFactory::class);
    }
}

```

ДОДАТОК Б

Конфігураційні файли Nginx

Б.1 Конфігурація MovieMicroservice

```
server {
    listen 80;
    server_name movie;
    root /var/www/univer/Diplom/BestMovie/MovieMicroservice/public;

    add_header X-Frame-Options "SAMEORIGIN";
    add_header X-Content-Type-Options "nosniff";
    client_max_body_size 100M;
    index index.php;

    charset utf-8;

    location / {
        try_files $uri $uri/ /index.php?$query_string;
    }

    location = /favicon.ico { access_log off; log_not_found off; }
    location = /robots.txt { access_log off; log_not_found off; }

    error_page 404 /index.php;

    location ~ \.php$ {
        fastcgi_pass unix:/var/run/php/php8.0-fpm.sock;
        fastcgi_param SCRIPT_FILENAME $realpath_root$fastcgi_script_name;
        include fastcgi_params;
    }

    location ~ /\.(!well-known).* {
        deny all;
    }
}
```

Б.2 Конфігурація BestMovieStorage

```
server {
    listen 80;
    server_name diplomBestMovieStorage;
    root /var/www/univer/Diplom/BestMovieStorage/public;
```

```

add_header X-Frame-Options "SAMEORIGIN";
add_header X-Content-Type-Options "nosniff";
client_max_body_size 100M;
index index.php;

charset utf-8;

location / {
    try_files $uri $uri/ /index.php?$query_string;
}

location = /favicon.ico { access_log off; log_not_found off; }
location = /robots.txt { access_log off; log_not_found off; }

error_page 404 /index.php;

location ~ \.php$ {
    fastcgi_pass unix:/var/run/php/php8.0-fpm.sock;
    fastcgi_param SCRIPT_FILENAME $realpath_root$fastcgi_script_name;
    include fastcgi_params;
}

location ~ /\.(!well-known).* {
    deny all;
}
}

```

Б.3 Конфігурація EmailTemplateMicroservice

```

server {
    listen 80;
    server_name diplomEmailTemplate;
    root /var/www/univer/Diplom/EmailTemplateMicroservice/public;

    add_header X-Frame-Options "SAMEORIGIN";
    add_header X-Content-Type-Options "nosniff";
    client_max_body_size 100M;
    index index.php;

    charset utf-8;

```

```

location / {
    try_files $uri $uri/ /index.php?$query_string;
}

location = /favicon.ico { access_log off; log_not_found off; }
location = /robots.txt { access_log off; log_not_found off; }

error_page 404 /index.php;

location ~ \.php$ {
    fastcgi_pass unix:/var/run/php/php8.0-fpm.sock;
    fastcgi_param SCRIPT_FILENAME $realpath_root$fastcgi_script_name;
    include fastcgi_params;
}

location ~ /\.(!well-known).* {
    deny all;
}
}

```

Б.4 Файл “.env” MovieMicroservice

```

APP_NAME=MovieMicroservice

APP_ENV=local
APP_KEY=base64:ZZG3jhwO/QiZomx8kLi8SiV1RhtaWO8thecQMgY1RS4=
APP_DEBUG=true
APP_URL=http://localhost

LOG_CHANNEL=stack
LOG_DEPRECATIONS_CHANNEL=null
LOG_LEVEL=debug

DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=diplom_movie
DB_USERNAME=root
DB_PASSWORD=4as6iKILL47<>and06052002alex

BROADCAST_DRIVER=log
CACHE_DRIVER=file
FILESYSTEM_DISK=local
QUEUE_CONNECTION=sync
SESSION_DRIVER=file
SESSION_LIFETIME=120

MEMCACHED_HOST=127.0.0.1

REDIS_HOST=127.0.0.1

```

```
REDIS_PASSWORD=null
REDIS_PORT=6379
```

```
MAIL_MAILER=smtp
MAIL_HOST=mailhog
MAIL_PORT=1025
MAIL_USERNAME=null
MAIL_PASSWORD=null
MAIL_ENCRYPTION=null
MAIL_FROM_ADDRESS="hello@example.com"
MAIL_FROM_NAME="${APP_NAME}"
```

```
AWS_ACCESS_KEY_ID=
AWS_SECRET_ACCESS_KEY=
AWS_DEFAULT_REGION=us-east-1
AWS_BUCKET=
AWS_USE_PATH_STYLE_ENDPOINT=false
```

```
PUSHER_APP_ID=
PUSHER_APP_KEY=
PUSHER_APP_SECRET=
PUSHER_HOST=
PUSHER_PORT=443
PUSHER_SCHEME=https
PUSHER_APP_CLUSTER=mt1
```

```
VITE_PUSHER_APP_KEY="${PUSHER_APP_KEY}"
VITE_PUSHER_HOST="${PUSHER_HOST}"
VITE_PUSHER_PORT="${PUSHER_PORT}"
VITE_PUSHER_SCHEME="${PUSHER_SCHEME}"
VITE_PUSHER_APP_CLUSTER="${PUSHER_APP_CLUSTER}"
```

```
EMAIL_TEMPLATE_MS=http://diplomEmailTemplate
BEST_MOVIE_STORAGE_MS=http://diplombestmoviestorage
```

```
MICROSERVICE_AUTH_TOKEN=//token
```

ДОДАТОК В
Презентація

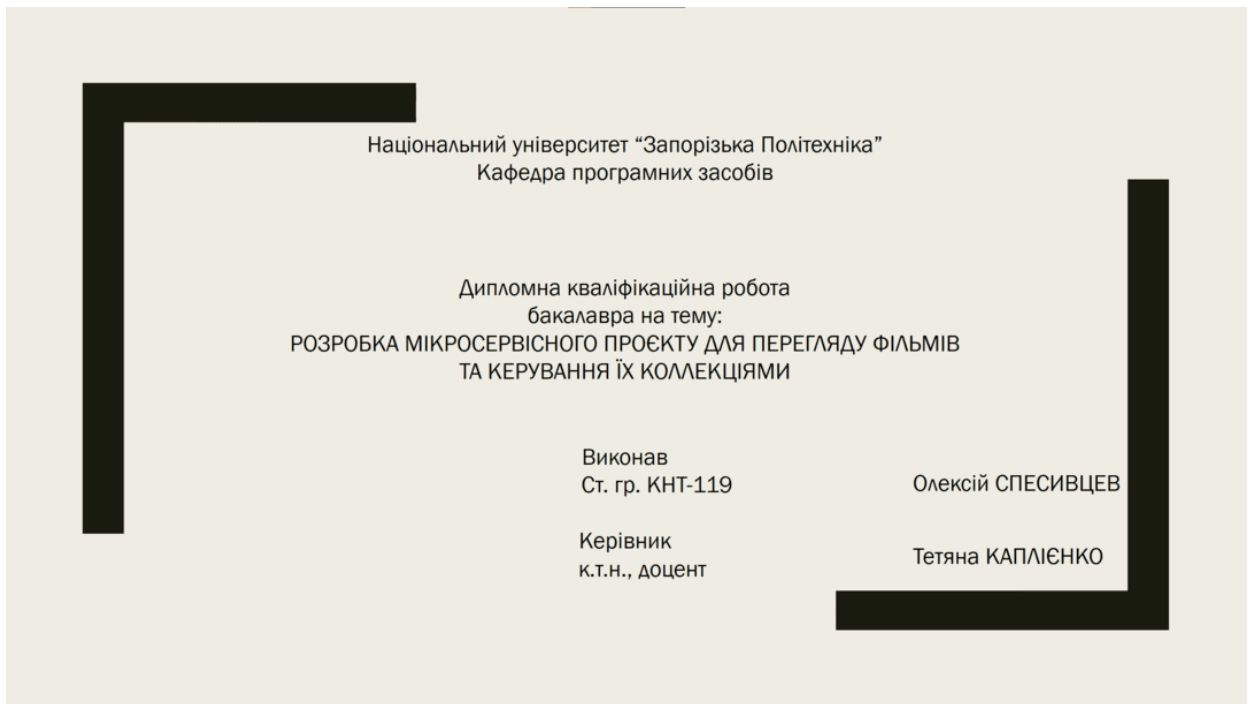


Рисунок В.1 – Слайд №1

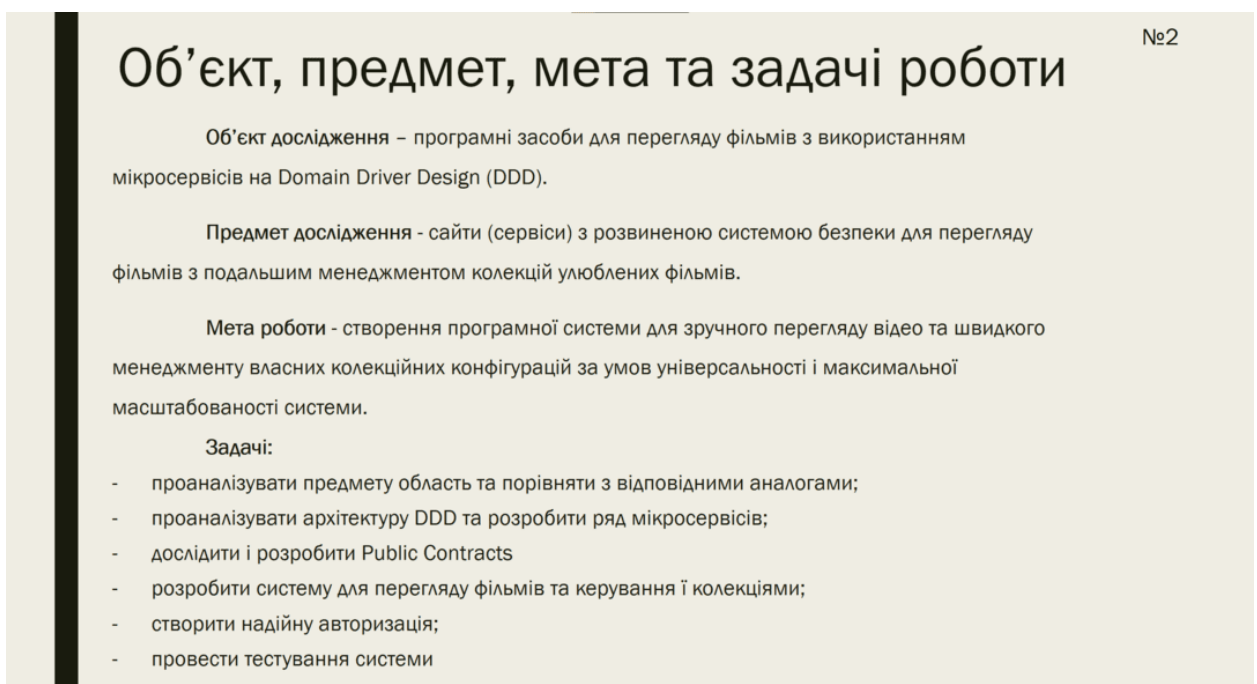


Рисунок В.2 – Слайд №2

Сервіси і сайти аналоги – Hdrezka.ag, Filmix.ac, Sweet.TV

№3

Порівняльні характеристики	Hdrezka.ag	Filmix.ac	Sweet.TV
Перегляд фільмів	Зручний	Зручний	Не зручний
Якість фільмів	Низька	Висока	Висока
Категорії і колекції	+	–	–
Адаптація під Smart TV, Android	–	Android	Smart TV
Стабільність	Низька	Низька	Висока
Локалізації	–	–	UA EN
Наявність вибору дизайну	+	–	–
Швидкодія	Низька	Висока	Низька
Безпечність	Низька	Низька	Висока

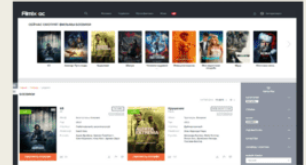
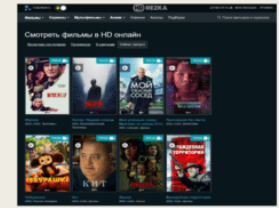


Рисунок В.3 – Слайд №3

Порівняння мов програмування

№4

Порівняльні характеристики	PHP	Java	C#	Python
ООП, класи, абстракції	Присутні, але менш розвинуті	Присутні і добре розвинуті	Присутні і добре розвинуті	Присутні і менш розвинуті
Типізація	Слабка динамічна	Сильна статична	Сильна статична	Слабка динамічна
Система управління пам'яті	–	+	+	–
Платформи	Laravel, Symfony, Yii	.NET Framework, .NET Core	Spring, Struts	Django, Flask
Реєстрація пакетів онлайн	Packagist	NuGet	Maven Central	PyPI
Асинхронність	ReactPHP, Swoole	async/await	CompletableFuture	asyncio
Structure, Enum, Traits	Немає Structure	Немає Traits	Немає Traits	Немає Traits
Швидкість розробки	Висока	Середня	Середня	Висока
Гнучкість	Висока	Середня	Середня	Висока
Застосування	Вебзастосунки, десктоп застосунки	Вебзастосунки, десктопні застосунки, ігри, мобільні застосунки	Вебзастосунки, десктоп застосунки, ігри, мобільні застосунки	Штучний інтелект, вебзастосунки, десктоп застосунки, ігри.

Рисунок В.4 – Слайд №4

Архітектура DDD

Чотири рівні:

- User Interface
- Application
- Domain
- Infrastructure

Шаблони програмування, що формують рівні:

- Service and Repository
- DPO (Domain Payload Object)
- Aggregate (Value objects, Traits Resolvers)
- Dependency Injection
- UOW (Unit of Work)
- CQRS (The command and Query Responsibility Segregation)

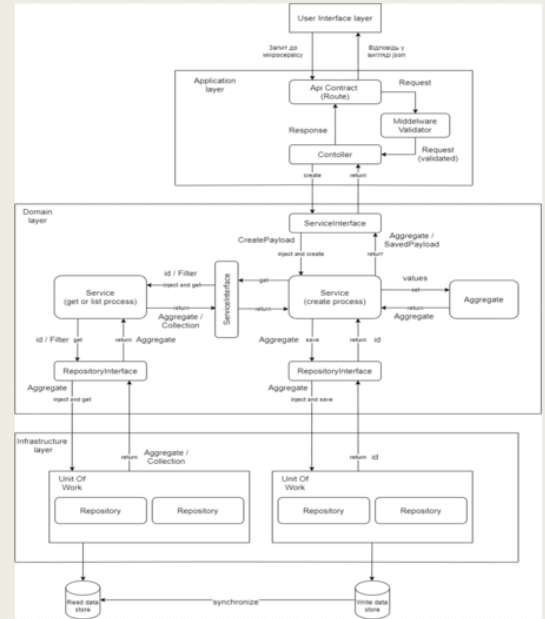
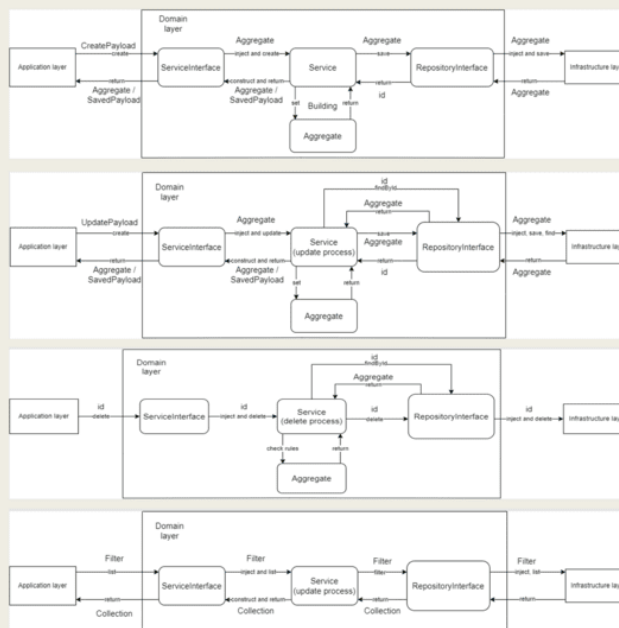


Рисунок В.5 – Слайд №5

Архітектура сценаріїв основних процесів Domain рівня



Створення нової сутності

Оновлення сутності

Сценарій видалення сутності у Domain рівні

Сценарій вибірки багатьох сутностей у Domain рівні

Рисунок В.6 – Слайд №6

Розроблені мікросервіси і використані технології

№7

Розроблено 4 мікросервіси:

- BestMovieClient
- MovieMicroservice
- BestMovieStorage
- EmailTemplateMicroservice

Використані технології

- Mailgun (поштовий сервіс)
- Firebase (сховище)
- Laravel (фреймворк)

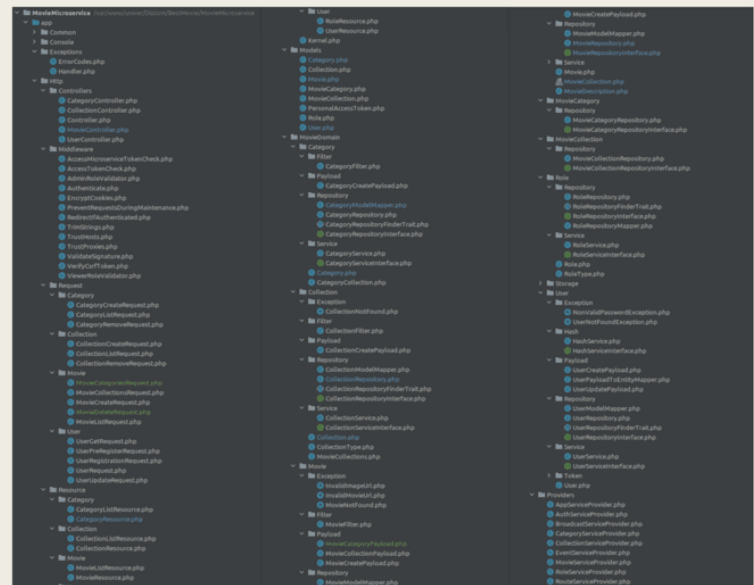


Рисунок В.7 – Слайд №7

Розроблені Public Contracts

№8

Розроблено два Public Contract:

- BestMovieCommon (пакетний)
- BestMovieClientApi (вбудований)

Пакет зареєстрований за допомогою

Packagist і доступний онлайн.

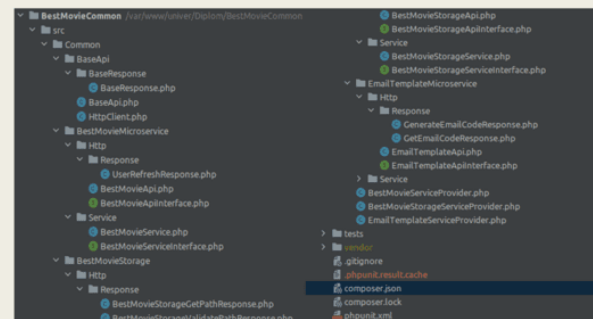


Рисунок В.8– Слайд №8

Розроблена БД і міграції

№9

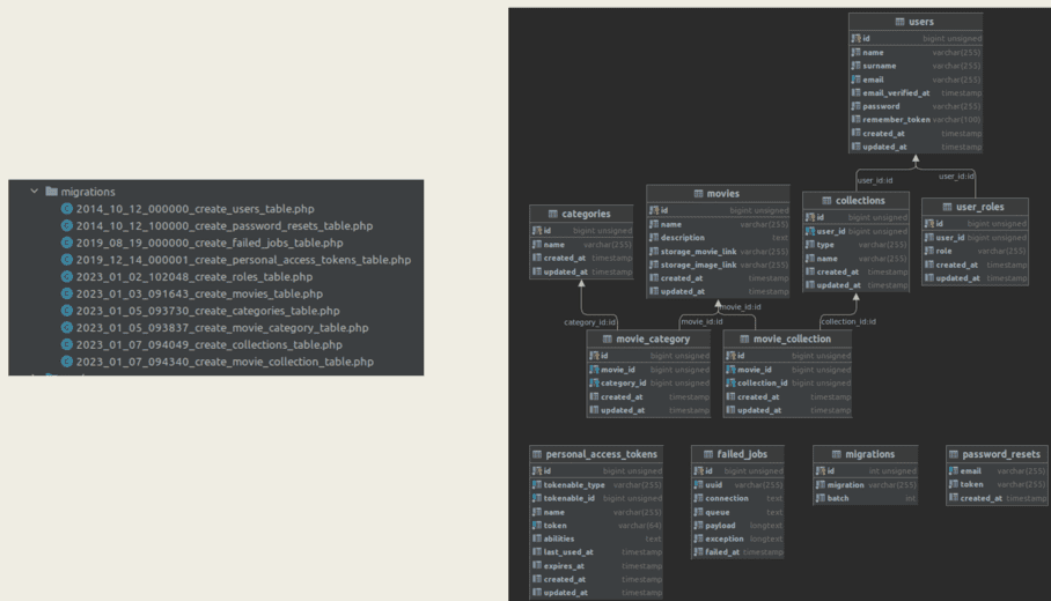


Рисунок В.9 – Слайд №9

Функції користувачів

№10

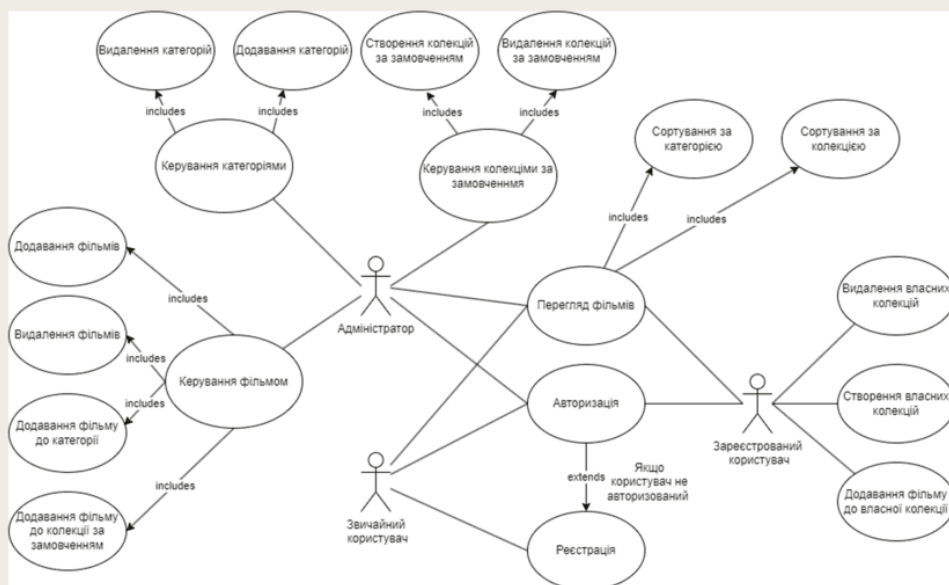
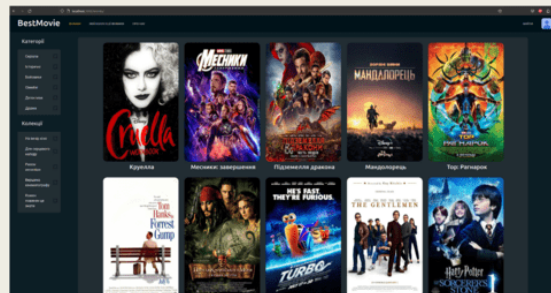


Рисунок В.10 – Слайд №10

Головні сторінки користувачів

№11



Сторінка звичайного і зареєстрованого користувача

Сторінка адміністратора

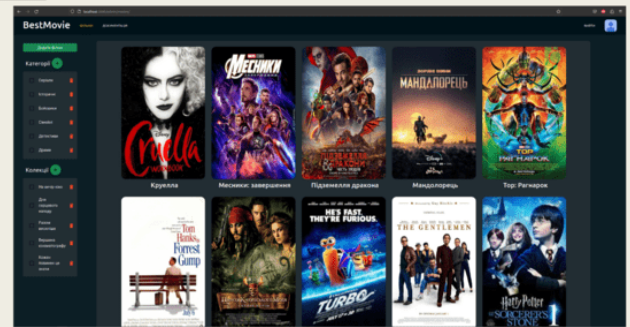


Рисунок В.11 – Слайд №11

Основні сторінки і модальні вікна

№12

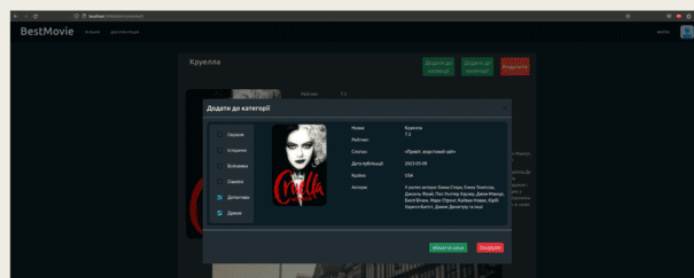
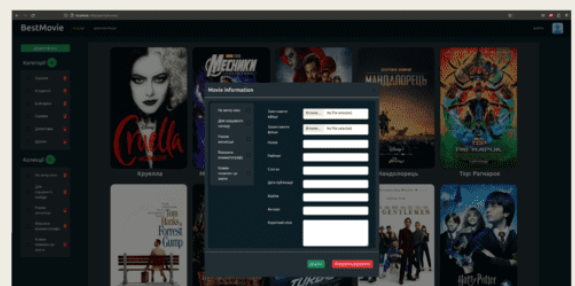
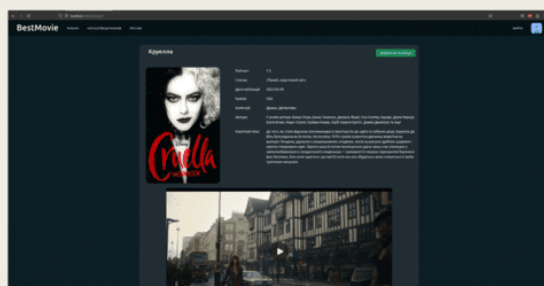


Рисунок В.12 – Слайд №12

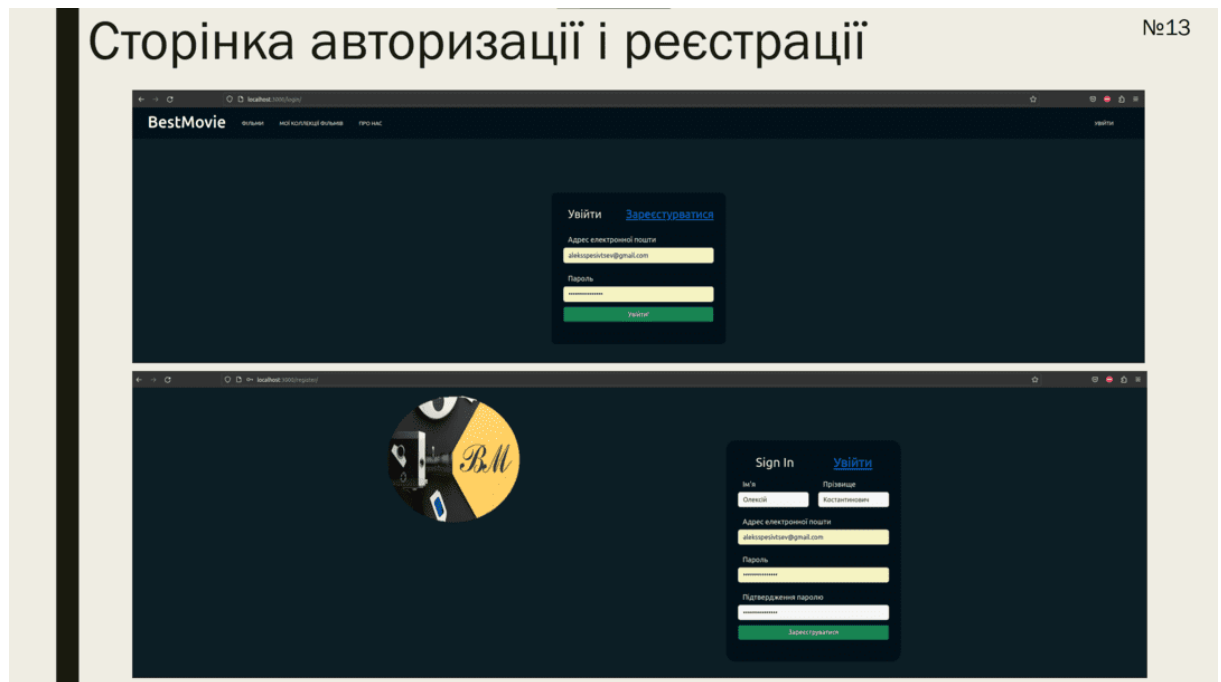


Рисунок В.13 – Слайд №13



Рисунок В.14 – Слайд №14

