

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 проекту (роботи)

магістр

(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ
МУЛЬТИПРОЦЕСНОГО МІКРОСЕРВІСНОГО ДОДАТКУ ДЛЯ
ПЕРЕГЛЯДУ ФІЛЬМІВ ТА КЕРУВАННЯ ЇХ КОЛЕКЦІЯМИ
RESEACRH AND SOFTWARE IMPLEMENTATION OF A MULTI-PROCESS
MICROSERVICE APPLICATION FOR WATCHING
MOVIES AND MANAGING THEIR COLLECTIONS

Виконав: студент(ка) 2 курсу, групи КНТ-113м
Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)
Інженерія програмного забезпечення

СПЕСИВЦЕВ О.К.

(ПРИЗВИЩЕ та ініціали)

Керівник ПОЛЯКОВ М.О.

(ПРИЗВИЩЕ та ініціали)

Рецензент ШИТІКОВА О.В.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти магістр
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
« » 2024 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

СПЕСИВЦЕВА Олексія Костянтиновича

(ПРІЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Дослідження та програмна реалізація
мультипроцесного мікросервісного додатку для перегляду фільмів та керування
їх колекціями.

Research and Software Implementation of a Multi-process Microservice Application
for Watching Movies and Managing their Collections
Collections

керівник проєкту (роботи) д.т.н., професор, Поляков Михайло Олексійович,
(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від «07» жовтня 2024 року № 398

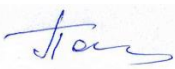
2. Строк подання студентом проєкту (роботи) 30 листопада 2024
року

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	ПОЛЯКОВ М.О., професор		
Нормоконтроль	КАМІНСЬКА Ж.К., асистент		

7. Дата видачі завдання « 15 » жовтня 2024 року.

КАЛЕНДАРНИЙ ПЛАН

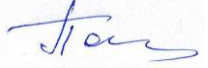
№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	2-3 тижні	Розділ 1
3	Вибір мови програмування та інших технологій розробки.	4-5 тижні	Розділ 2
4	Розробка архітектури програми.	6-7 тижні	Розділ 3
5	Розробка програми.	7-8 тижні	Розділ 3, 4
6	Тестування та експериментальне дослідження програмного забезпечення.	9 тиждень	Розділ 4
7	Оформлення пояснювальної записки та документів до неї.	10 тиждень	Додатки
8	Нормоконтроль та рецензування.	11 тиждень	
9	Захист роботи.	12 тиждень	

Студент(ка)


(підпис)

Олексій СПЕСИВЦЕВ
(Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)


(підпис)

Михайло ПОЛЯКОВ
(Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра:
123 с., 25 рис., 5 табл., 3 дод., 43 джерел.

АСИХРОННІСТЬ, МІКРОСЕРВІС, БАГАТОПОТІЧНІСТЬ,
АРХІТЕКТУРА, САЙТ, СЕРВІС, ФІЛЬМ, КОЛЕКЦІЯ, ШАБЛОНИ
ПРОГРАМУВАННЯ.

Об'єкт дослідження – мультипроцесні програмні засоби для перегляду фільмів та керування їх колекціями, реалізовані на основі мікросервісної архітектури з використанням підходу Domain Driver Design (DDD).

Предмет дослідження – сайти (сервіси) з високою швидкістю обробки великих даних для перегляду фільмів та менеджментом колекцій фільмів.

Мета роботи – створення програмної системи для зручного завантаження відео та швидкого менеджменту колекційних конфігурацій за умов високої продуктивності системи.

Матеріали, методи та технічні засоби: об'єктно-орієнтоване програмування, асинхронність і багатопоточність, Spatie-Async, AMPHP, DDD архітектура, Firebase, PHP, Laravel, React, MySQL.

Висновки. Розроблено мікросервісний проєкт для перегляду фільмів і створення улюблених колекцій з асинхронними і багатопоточними модулями, які максимально адаптовані під функціонування в незалежності від потреб користувача і обмежень HTTPS протоколу або конфігурацій серверу.

Актуальність роботи пов'язана з зростанням потреб у збільшенні комп'ютерних потужностей та подолання проблем зниження продуктивності під час пікових навантажень на фізичні сервери і мережи. Галузь використання – сервіси послуг з перегляду і меджементу медіа контенту, а саме фільмів, відео, афіш, трейлерів.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 123 pages, 25 figures, 5 tables, 3 appendixes, 43 sources.

ASYNCHRONOUSNESS, MICROSERVIS, MULTI-THREADING, ARCHITECTURE, SITE, SERVICE, MOVIE, COLLECTION, PROGRAMMING PATTERNS.

The subject of research is sites (services) with high speed of processing big data for watching movies and managing movie collections.

The purpose of the work is to create a software system for convenient video downloading and fast management of collection configurations under conditions of high system performance.

Materials, methods and technical means: object-oriented programming, asynchrony and multithreading, Spatie-Async, AMPHP, DDD architecture, Firebase, PHP, Laravel, React, MySql.

Conclusions. A microservice project has been developed for watching movies and creating favorite collections with asynchronous and multithreaded modules, which are maximally adapted to function independently of user needs and limitations of the HTTPS protocol or server configurations.

The relevance of the work is associated with the growing need to increase computer capacity and overcome the problems of reduced performance during peak loads on physical servers and networks. The field of use is services for viewing and managing media content, namely films, videos, posters, trailers.

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.....	14
1.2.1 Передмова	15
1.2.2 Сервіс Primevideo (Amazon).....	16
1.2.3 Сервіс Netflix	17
1.2.4 Сервіс Sweet.tv	19
1.3 Висновки з розділу і постановка задачі	21
2 АНАЛІЗ ПРОГРАМНИХ ЗАСОБІВ.....	23
2.1 Огляд особливостей мов програмування	23
2.2 Огляд особливостей обраних платформ і бібліотек	27
2.3 Огляд особливостей обраних інтерпретаторів.....	29
2.4 Огляд архітектури DDD	31
2.4.1 Application рівень.....	32
2.4.2 Domain рівень	33
2.4.3 Infrastructure рівень.....	35
2.4.4 User Interface рівень.....	36
2.4.5 Зв'язок мікросервісів	37
2.5 Огляд особливостей обраної СКБД.....	38
2.6 Огляд особливостей обраних бібліотек і розширень	39
2.6.1 Розширення pcntl	39
2.6.2 Бібліотека SpatieAsync	40
2.6.3 Бібліотека AMPHP	41
2.7 Висновки з розділу	42
3 ОСНОВНІ РІШЕННЯ З РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ	43

3.1 Основні рішення щодо розроблених MC і PCO	43
3.1.1 MC MasterClient	44
3.1.2 MC MovieMicroservice	47
3.1.3 MC EmailTemplateMicroservice	52
3.1.4 MC MasterMovieStorage	54
3.1.5 PCO BestMovieCommon	55
3.2 Основні рішення щодо розробки інтерфейсу	60
3.2.1 Основні рішення щодо впорядкування інтерфейсу	62
3.2.2 Основні рішення з комунікації з користувачем	69
3.3 Основні рішення щодо розробки БД	70
3.3.1 Розроблена БД	72
3.4 Висновки з розділу	74
4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРЕМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ	75
4.1 Застосування системи	75
4.2 Розгортання і експлуатація системи мікросервісів	75
4.3 Тестування та дослідження сумісності проєкту	77
4.4 Висновки з розділу	82
ВИСНОВКИ	83
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ	84
Додаток А Текст програми	89
Додаток Б Конфігураційні файли Docker	106
Додаток В Презентація	115

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

БД	– База Даних;
МС	– Мікросервіс;
ОС	– Операційна Система;
СКБД	– Система Керування Бази Даних;
ПК	– Персональний Комп'ютер;
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;
HTTP	– HyperText Transfer Protocol;
HTTPS	– HyperText Transfer Protocol Secure;
IDE	– Integrated Development Environment;
JS	– Java Script;
JSON	– JavaScript Object Notation;
MVC	– Model View Controller;
PCO	– Public Contract;
PHP	– Hypertext Preprocessor;
PHP-FPM	– PHP FastCGI Process Manager;
PSR	– PHP Standards Recommendations;
REST	– Representational State Transfer;
SEO	– Search Engine Optimization;
UOW	– Unit Of Work.

ВСТУП

Завданням цієї роботи є створення мікросервісного проєкту з застосуванням мультипроцесних і асинхронних операцій на архітектурі DDD для перегляду та адміністрування фільмів і їх колекцій.

Актуальність даної роботи полягає у щорічному зростанні навантаження на фізичні сервери і збільшенню потреб у енергії і складних системах охолодження. Відомо, що це пов'язано з великим попитом на штучний інтелект, але велика кількість програм, що створені ще до стрімкого росту нейромереж, погано оптимізовані або зовсім не використовують асинхронні і мультипроцесні операції для обробки даних. Сучасний світ потребує рішення, що зможе швидко та без зайвих перешкод або суттєвих фінансових вкладів покращити продуктивність і збільшити кількість доступних фізичних і програмних ресурсів [1] – [3].

Мета цієї роботи полягає у дослідженні і розробці системи, що зможе продемонструвати шляхи оптимізації і створення проєкту, який буде прикладом балансу обробки навантажених операцій, на простих прикладах функцій перегляду, створення, видалення і редагування фільмів і їх колекцій.

Проєкт має декілька областей використання, основна – це звичайний користувач, який може вести облік своєї колекції фільмів, зберігати дані про кожен фільм (назва, рік випуску, режисер, актори, жанр, опис тощо), додатковою областю можна назвати різноманітні електронні бібліотеки, кінотеатри, сховища медіа даних. Слід зазначити, що система керування фільмами може служити прикладом для розробників, як ресурс по тестуванню підходів і бібліотек.

Отже, проєкт дозволить дослідити і розробити рішення для оптимізації програмних процесів по каталогізації і звантаженню ресурсів, що містять фільм або відео.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

У цьому розділі розглядаються наявні системи та вебсайти, що забезпечують можливість перегляду, сортування, редагування інформації про фільми та їх об'єднання у колекції. Додатково аналізується використання мультипроцесингу та асинхронності в подібних системах для забезпечення ефективності обробки даних та масштабованості. У розділі також визначаються мета та завдання системи, що розробляється.

1.1 Аналіз предметної області

Сучасні, великі, швидкі системи складаються з декількох сервісів, що надають послуги і відповідають за контроль різних логічних процесів у системі, кожен з таких сервісів має свої особливості, що можуть впливати на енергоспоживання та вимоги до якості комплектуючих.

Мета роботи – створити сервіс, здатний балансувати між одночасним навантаженням на систему та швидкодією обробки запитів, забезпечуючи високоефективну обробку запитів навіть під значним навантаженням. Основною метою також є надати користувачеві функціональність для створення колекцій і реалізувати стандартні можливості, такі як пагінація, фільтрація, додавання, редагування та видалення категорій, колекцій, фільмів і медіафайлів.

Для досягнення цього необхідно впровадити черги завдань, де завдання будуть розподілені на спеціальні робочі процеси – воркери. Це дозволить обробляти ресурсомісткі операції в асинхронному режимі, знижуючи навантаження на основний потік і значно покращуючи загальну продуктивність системи. Зазвичай найкращі сайти та сервіси, що надають послуги по перегляду фільмів використовують дорогі і складні технології, які зв'язують систему жорсткою абстракцією або мають занадто слабку абстракцію, що у свою чергу призводить до повної переробки реалізації або

запобігає легкому розширенню систему. Визначимо перелік серверів і їх взаємодію:

- клієнт-сервер;
- брокер сервер з архітектурою REST API (Representational State Transfer Application Programming Interface – це архітектурний стиль для побудови вебсервісів, який дозволяє клієнтам взаємодіяти із сервером через стандартні HTTP-запити [4];
- сервіс інтеграції;
- високопродуктивне сховище даних, медіа файлів.

Клієнт-сервер – це інтуїтивний і простий для користувача інтерфейс, який дозволяє управляти даними, не заглиблюючись у деталі їх зв'язків у БД (БД – база даних) [5]. Завдяки спеціалізованому набору функцій клієнтська частина асинхронно надсилає запити на один або кілька серверів, що працюють на REST API, уникаючи затримок для користувача під час виконання процесу. Ці запити використовують загальноприйнятий захищений протокол HTTPS (HyperText Transfer Protocol Secure — протокол передачі даних в Інтернеті) та розроблену методологію обміну даними, включаючи перевірку даних, відомості про авторизацію, специфічні заголовки і структуровані дані, унікальні для системи [6]. Реалізація такого інтерфейсу полягає у створенні PCO (PCO – public contract, публічний контракт, що реалізує інтерфейс взаємодії між декількома серверами).

PCO може виступати у вигляді спеціального пакета даних або загальної частини коду, що формує інтерфейс для обміну даними між сервісами. Створення універсального PCO, здатного до завантаження в будь-який момент на різні платформи, дозволяє підвищити масштабованість і адаптивність проєкту. Така реалізація підходить для поліпшення взаємодії між МС, які можна налаштовувати та з'єднувати без змін у базовому коді кожного з них. Важливо зазначити, що розробка PCO – це тривалий процес, який має сенс після завершення роботи над основними сервісами, які цей PCO об'єднає.

Брокер сервер, побудований за архітектурою REST API, виконує найважливіші функції і є найбільш завантаженим серед усіх сервісів [7]. Завдяки публічному контракту (PCO) цей сервер або спеціальний брокер сервіс зможе керувати запитами, розподіляючи їх на обробку та, залежно від пріоритетів, визначати для них асинхронний чи синхронний метод виконання. У цьому контексті PCO дозволяє головному серверу ефективно здійснювати перевірку ролей і прав доступу користувачів, керувати процесами створення, редагування, видалення даних про фільми, категорії, колекції, акторів та оцінки. Для обробки таких запитів доцільно використовувати кешування та асинхронні процеси, зменшуючи навантаження і підвищуючи швидкодію. Крім того, у головному сервері можуть бути реалізовані додаткові функції для безпеки зберігання даних та авторизації.

Сучасні методи оптимізації навантажених сервісів враховують обмеження асинхронності в різних мовах програмування, адже не всі з них повноцінно підтримують асинхронність. Також кешування, яке часто використовується для зберігання даних, має свої недоліки, оскільки працює за принципом створення ключів для доступу до даних. Ці ключі, залежно від способу генерації, можуть призводити до колізій, додатково навантажуючи апаратні ресурси сервера та знижуючи загальну продуктивність системи. Щоб уникнути таких обмежень, розробники часто вирішують розділяти великий сервіс на спеціалізовані, менші, такі як окремий поштовий сервіс чи сховище медіа файлів. Важливо також розуміти, що кожна мова програмування має унікальні підходи до асинхронної та багатопотокової обробки завдань, зокрема щодо розподілу ресурсів, створення потоків і відокремлення їх у процеси. Ці особливості впливають на продуктивність і підходи до розробки сервісів. Розподіл завдань на окремі сервіси дає змогу не лише ефективніше управляти ресурсами, але й ізолювати ключові процеси, що спрощує їхнє обслуговування, масштабування та тестування. Завдяки

такій структурі, кожен сервіс може працювати незалежно, оптимізовано для своєї мети, та легко адаптуватися до нових вимог

Відокремлення функцій надсилання листів на окремий сервіс має численні переваги. Насамперед, це дозволяє організувати асинхронну чергу завдань, що знижує затримки у виконанні основних операцій. Використання черг з інструментами, такими як Laravel Horizon (інтерфейс моніторингу та керування чергами Laravel), Redis (система керування базами даних у пам'яті з високою швидкістю доступу) та Supervisor (утиліта для контролю за процесами на UNIX подібних системах), дає змогу ефективніше обробляти великий обсяг завдань, розподіляючи навантаження між воркерами і Horizon, зокрема, надає зручний інтерфейс для моніторингу черг і керування воркерами, а Redis забезпечує швидкий доступ до черг, що прискорює обробку асинхронних запитів [8].

Ще однією значною перевагою є зниження апаратного навантаження, оскільки функції можна розмістити на окремих фізичних серверах, що поліпшує продуктивність системи. Так само організація окремого сервісу для зберігання медіафайлів дозволяє використовувати хмарні платформи, такі як AWS (Amazon Web Services – хмарна платформа, що надає інфраструктуру для обчислень, зберігання даних та підтримки додатків) або Firebase (платформа від Google для створення мобільних і вебдодатків) забезпечуючи безпечне та масштабоване середовище для зберігання [9]. Це також дозволяє незалежно налаштовувати різні середовища для зберігання даних, обираючи найкращі варіанти для конкретних потреб, підвищуючи універсальність і масштабованість системи.

Ця фрагментація на системи має власні методи та інженерні принципи. Один із таких методів оптимізації роботи сервісів описано в книзі Еріка Евенса «Об'єктно-орієнтоване проектування (DDD): Організація складних програмних рішень» [10]. DDD – це підхід до створення програмного забезпечення, який ставить модель домену в центр уваги. Рівнева структура DDD складається з чотирьох компонентів – програми,

домену, інфраструктури та інтерфейсу користувача. Інтерфейсний рівень містить елементи, які працюють з користувачами та інтегруються із зовнішніми системами. Рівень домену включає бізнес логіку, яка описує сутності, агрегати та служби.

Рівень інфраструктури відповідає за зберігання та обробку інформації, а також за інтеграцію з іншими системами. Зв'язок між компонентами реалізується за допомогою слабозв'язаного архітектурного підходу, де кожен рівень може працювати з верхнім і нижнім шарами. Інтерфейс користувача – це набір інструментів, які дозволяють взаємодіяти з мікросервісною системою через вебдодаток, програму для ПК або командний рядок [11] – [12].

Після розподілу системи на декілька МС (Мікросервіс – це незалежний функціональний компонент, який приймає запити для виконання окремих операцій), необхідно визначити архітектуру взаємодії між сервісами та обрати оптимальний підхід для зв'язку. Визначимо ключові МС, РСО, що будуть створені, а також додаткові елементи та сервіси, які будуть використані для побудови системи «MovieMaster».

Для успішної інтеграції всіх МС, РСО (публічний контракт обміну) повинен підтримувати як синхронні, так і асинхронні запити, дозволяючи гнучко взаємодіяти з компонентами системи та забезпечуючи ефективний обмін даними. Кожен МС буде розроблено у двох варіантах: синхронному та асинхронному, що дозволить порівняти продуктивність і оцінити оптимізацію споживання ресурсів. Це забезпечить можливість оцінити, як зміни в обробці потоків впливають на швидкодію та ефективність, а також допоможе визначити найкращі підходи до розподілу навантаження між сервісами в реальних умовах.

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

Перегляд фільмів – один із найпопулярніших і безпечних способів відпочинку, і на сьогодні існує безліч сервісів, що дозволяють дивитися фільми онлайн.

У цьому розділі розглянуто та проаналізовано їхні переваги й недоліки. Також розглянуто можливості створення власних колекцій фільмів на деяких із цих платформ. Проведено дослідження швидкодії, асинхронної і мультипроцесної обробки запитів.

1.2.1 Передмова

Перед аналізом аналогічних систем важливо зазначити, що більшість із них надають безкоштовний контент та базові функції, проте деякі з них потребують авторизації та підтвердження користувача. Це дає змогу об'єктивно оцінити функціональні можливості сайтів і сервісів. Слід також врахувати, що процеси таких складних систем приховані від звичайного користувача, і повноцінно оцінити їхню продуктивність без доступу до серверного коду досить складно. Тому аналіз був проведений із застосуванням інструментів браузера, таких як інспектор елементів:

- сервіс “Primevideo” (Amazon);
- сервіс “Netflix”;
- 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:

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

- багатоплатформність;
- продуктивність під час асинхронної обробки;
- стійкість до високих навантажень.

1.2.2 Сервіс Primevideo (Amazon)

Сервіс Prime Video від Amazon пропонує онлайн-перегляд фільмів та серіалів у високій якості з можливістю завантаження контенту для перегляду офлайн. Сервіс має велику бібліотеку фільмів, серіалів, а також оригінальних програм від Amazon. Prime Video доступний як на вебплатформі, так і на різних мобільних та розумних пристроях, що забезпечує широкий доступ до контенту користувачам [13].

Переваги:

- наявність можливості створювати колекції;
- наявність колекцій за замовченням;
- адаптація буферизації і кешу в залежності від спроможності мережі і серверу;
- багатий каталог, містить понад 24000 фільмів, 2100 серіалів і оригінальних шоу від Amazon, доступні ексклюзивно;
- висока якість відео, контент доступну в 4K, HDR та Dolby Atmos, що забезпечує високу якість;
- функція завантаження, можливість завантажувати фільми та серіали на мобільні пристрої для офлайн перегляду;
- стрічка новин, анонсів, дат релізів серій і фільмів;
- інтерактивність, функція X-Ray показує інформацію про акторів, саундтреки, місця зйомки та інші деталі;
- багатоплатформеність, сумісний з iOS, Android, Fire TV, Apple TV, Smart TV;
- відсутність залежності від сторонніх сервісів;
- офлайн перегляд.

Недоліки:

- короткий безкоштовний період;
- недоступність у деяких регіонах;
- перенасичений інтерфейс, приклад наведено на рисунку 1.1;
- затримки оновлення даних, при завантаженні або оновленні контенту;
- відсутність персоналізованого алгоритму для нових колекцій.

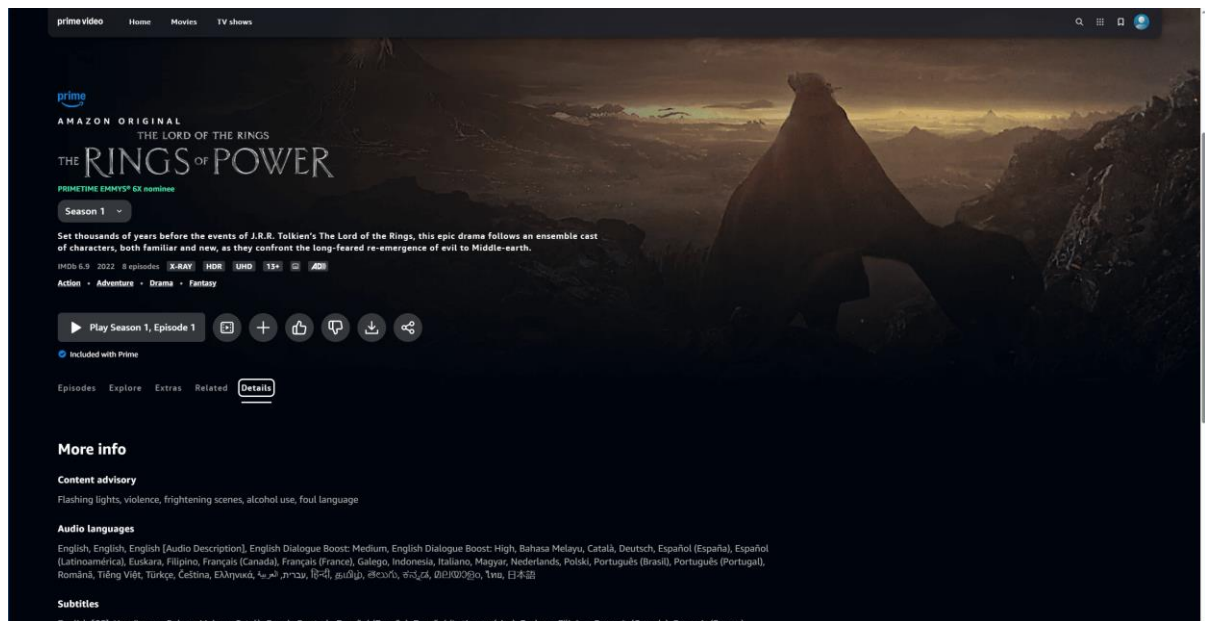


Рисунок 1.1 – Перенасичений інтерфейс сторінки Primevideo

1.2.3 Сервіс Netflix

Популярний сервіс для потокового перегляду фільмів та серіалів, що пропонує широкий каталог контенту, включаючи оригінальні шоу та фільми, доступні ексклюзивно на платформі. Netflix підтримує доступ із багатьох пристроїв та платформ, таких як веббраузери, мобільні додатки, смарт-телевізори, ігрові консолі та інші медіапристрої [14].

Переваги:

- широкий вибір контенту;
- висока якість відео;

- офлайн перегляд;
- персоналізація рекомендацій;
- оптимізація за рахунок використання локальних мереж і серверів;
- гнучкий інтерфейс для колекції;
- багатоплатформеність, підтримка всіх основних платформ, включаючи мобільні пристрої (IOS та Android), Smart TV, ігрові приставки, веббраузери та інші медіа пристрої, додатковий перелік можна побачити на рисунку 1.2;

- власна система кешування Open Connect;
- підтримка багатьох мов.

Недоліки:

- короткий безкоштовний період;
- ліцензійні обмеження контенту;
- обмеження для завантаження контенту;
- відсутність фільтрації за жанром або акторським складом;
- погана синхронізація даних при різних сесіях;
- висока залежність від сторонніх сервісів;
- зайнятість сервісів, погана оптимізація роздачі.

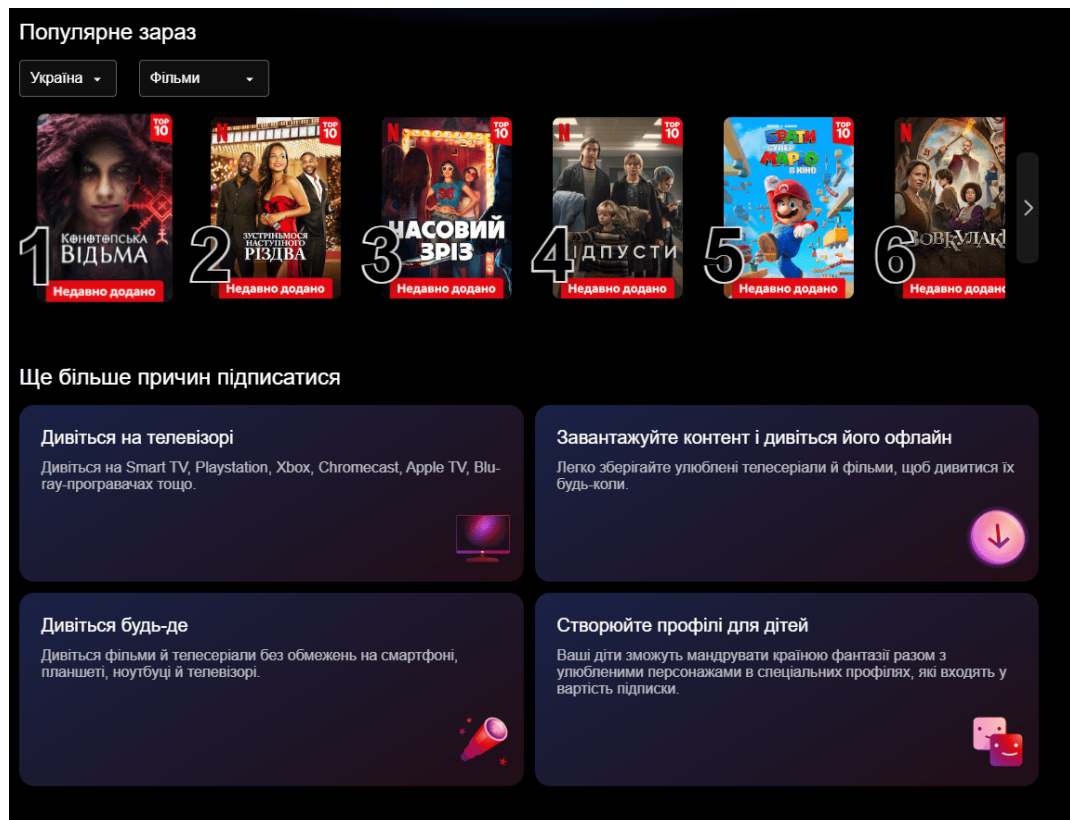


Рисунок 1.2 – Демонстрація багатоплатформеності сервісу Netflix

1.2.4 Сервіс Sweet.tv

Sweet TV є популярним українським сервісом, що дозволяє дивитися телевізійні канали та відеоконтент онлайн. Платформа пропонує широкий вибір телеканалів, включаючи місцеві та міжнародні, а також доступ до великої бібліотеки фільмів, серіалів та іншого медіаконтенту [15]. Завдяки підтримці високої якості трансляцій, Sweet TV надає користувачам можливість перегляду в ультрависокій роздільній здатності, що робить його конкурентоспроможним серед сучасних стрімінгових платформ. Приклад інтерфейсу наведено на рис. 1.3.

Переваги:

- широкий вибір TV каналів;
- адаптація під різні мови;
- колекції медіа контенту за замовченням, приклад наведено на рисунку 1.3;

- групування за студією видавцем;
- висока швидкість роздачі;
- оптимізація за рахунок використання CDN серверів-сховищ;
- адаптивність під Smart TV, Android, IOS, основні браузерери;
- інтегрована стрічка новин;
- функція навчання з медіа.

Недоліки:

- відсутність детальних фільтрів пошуку;
- обмежена кількість фільмів;
- немає можливості створювати власні колекції;
- обмежений вибір дизайну інтерфейсу;
- відсутність вибору головного дизайну сервісу.

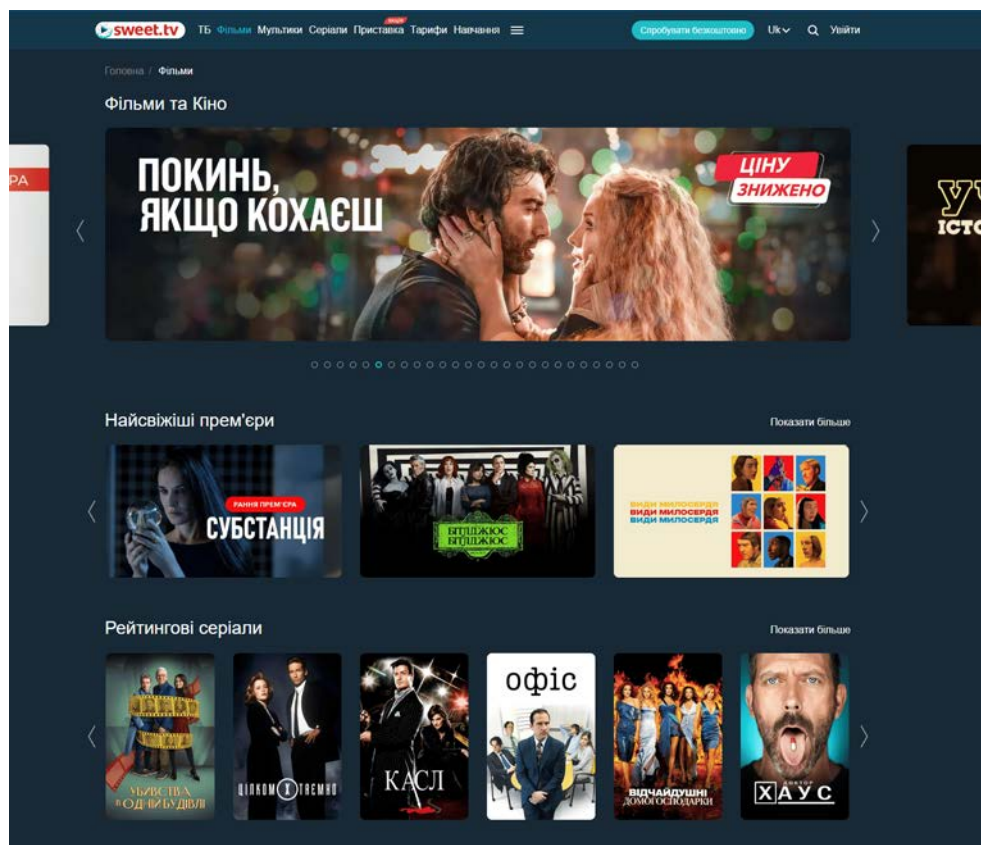


Рисунок 1.3 – Головна сторінка сервісу Sweet.tv

Загалом характеристики усіх розглянутих сайтів і сервісів можна порівняти за допомогою таблиці, яку наведено у таблиці 1.1.

Таблиця 1.1 – Порівняння сервісів аналогів

Порівняльні характеристики	Primevideo (Amazon)	Netflix	Sweet.TV
1	2	3	4
Перегляд фільмів	Зручний	Зручний	Не зручний
Технологія X-Ray	+	–	–
Якість фільмів	Висока	Висока	Надвисока
Розвинуті технології кешу	Відсутні	Open Connect	Відсутні
Залежність від сторонніх сервісів	Немає	Висока	Немає
Категорії і колекції	+	+	–
Адаптація під платформи і сервіси	Smart TV, Android, IOS, Blu-ray, Fire TV	Smart TV, Android, IOS, Xbox, PlayStation Blu-ray, Chromecast,	Smart TV, Android, IOS
Стабільність	Висока	Висока	Висока
Локалізації	Велика кількість	Велика кількість	UA EN
Наявність вибору дизайну	+	+	–
Швидкодія	Висока	Висока	Низька
Безпечність	Висока	Висока	Висока

1.3 Висновки з розділу і постановка задачі

Під час детального аналізу сервісів, визначено ключові характеристики, що забезпечують ефективне функціонування платформи для

потокowego перегляду: висока продуктивність, зручний інтерфейс, гнучка система авторизації користувачів та функціональність для перегляду та управління відеоконтентом. Пріоритетом стало створення масштабованої, асинхронної системи з багатопотоковою обробкою для швидкого завантаження контенту та забезпечення стабільної роботи незалежно від рівня навантаження.

Сервіс включає розробку трьох ролей користувачів із різними рівнями доступу: адміністратора, зареєстрованого користувача та звичайного переглядача. Кожна роль передбачає набір можливостей, що відповідають потребам різних типів користувачів, а також функції, доступні всім користувачам, як-от перегляд фільмів, трейлерів, детальної інформації та афіш із можливістю пагінації, сортування за категоріями, додавання фільмів у колекції. Адміністратори зможуть керувати категоріями, додавати та редагувати інформацію, тоді як зареєстровані користувачі отримають можливість створювати власні колекції.

Зважаючи на досліджені переваги та недоліки існуючих платформ, таких як Netflix, Amazon, Sweet TV, було зроблено акцент на адаптацію багатопотоковості й асинхронності для підвищення швидкодії саме серверної частини на рівні PHP коду та обробки запитів, що мінімізує затримки при роботі з великими обсягами даних. Використання кешування та оптимізації за допомогою системи багатопоточності допомагає забезпечити високу продуктивність незалежно від пропускну здатності мережі користувача.

Замість фокусування на технологіях адаптивного стрімінгу, що залежні від швидкості інтернет з'єднання користувача та браузерних рішень, увага була зосереджена на оптимізації передачі та обробки даних безпосередньо сервером. Це дозволяє стабільно й швидко надавати доступ до контенту, підвищуючи якість обслуговування й масштабованість системи, що важливо для забезпечення користувацького досвіду в реальних умовах високого навантаження.

2 АНАЛІЗ ПРОГРАМНИХ ЗАСОБІВ

У даному розділі розглянуті основні особливості мов програмування, бібліотек, розширень, технологій, сервісів, архітектур та обраних програмних засобів, що будуть застосовані при розробці проєкту.

2.1 Огляд особливостей мов програмування

Перша обрана мова програмування – PHP (Hypertext Preprocessor) версії 8.0, що є динамічною скриптовою мовою для розробки вебзастосунків та серверних сервісів. Вона залишається однією з найпопулярніших мов для створення вебплатформ завдяки своїй гнучкості, зручності та широкому спектру підтримуваних технологій:

- підтримка асинхронного програмування через файбери, розширення та бібліотеки, як от SpatieAsync, Amphp, розширення pcntl;
- зручна інтеграція з технологіями для роботи з чергами та відкладеними операціями (Horizon, Supervisord, Redis, RabbitMQ);
- легка адаптація до нових завдяки розширенням, що доповнюють стандартні функції ядра мови, як-от bcmath, curl, pcntl, fileinfo, bz2, ctype, mcrypt, dom;
- стабільна робота навіть зі старими серверами без зайвих інтеграцій, після встановлення потрібних php розширень;
- підтримка сучасних інструментів і штучного інтелекту у IDE PHPStorm (IDE – інтегрована середовище розробки), сумісність з Git, також слід виділити інтерфейс у IDE для зручного контролю версій програмного коду;
- класи, інтерфейси, переліки, абстрактні класи, traits дозволяють реалізувати принципи ООП: інкапсуляція, наслідування, поліморфізм [16];
- для тестування зроблено окремий фреймворк-розширення (пакет) PHPUnit для автоматичних тестів, Xdebug для налагодження коду, який неймовірно добре перехоплює асинхронні і мультипроцесні операції;

- контроль версій здійснюється за допомогою Composer, онлайн бібліотекою packagist;
- популярність і доступність навчальних матеріалів, велика кількість документації, прикладів, відео уроків, 77.5% сучасних вебсайтів використовують PHP як серверну мову програмування [17] – [19];
- простота у створенні масивів без ініціалізації покажчиків, відсутність потреби контролювати пам'ять;
- магічні методи для роботи з динамічними властивостями та методами, що вирішують проблему відсутності стандартних перевантажень методу;
- сумісність з сучасними платформами та CMS (Content Management System – це програмне забезпечення для керування вмістом, яке дає змогу користувачам створювати, редагувати та публікувати контент), Laravel, Symfony, WordPress, Drupal, Joomla;
- слабка динамічна типізація, статична типізація та їх комбінації дають розробнику гнучкість у виборі підходу;
- достатньо швидка для виконання завдань пов'язаних з розробкою застосунків, сайтів, мікросервісних систем;
- окрема документація з правилами стилю коду PSR (PHP Standards Recommendations – це стандарти кодування для PHP, що регламентують використання методів, визначення конструкторів класів та форматування коду);
- широка підтримка БД, таких як Mysql, PostgreSQL, MongoDB;
- підтримка вбудованого вебсервера для швидкого локального тестування;
- гнучкість конфігурації PHP через файл php.ini.

Слід зазначити, що обрана мова має конкурентні мови програмування, а саме C#, Go, Python. За допомогою перерахованих мов також можна створювати складні МС з асинхронними і мультипроцесними підходами, 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#	Go	Python
1	2	3	4	5
ООП, класи, інтерфейси, абстракції	Присутні, достатньо розвинуті	Присутні і добре розвинуті	Присутні, але дуже обмежені	Присутні, але менш розвинуті
Асинхронність	Присутня за допомогою, AMPHP, SpatieAsync	Присутня за допомогою async/await	Вбудована через Goroutines	Присутня за допомогою asyncio
Мультипроцесність	Розширення pcntl	Вбудована підтримка	Вбудована через Goroutines	Можлива через threading і concurrent.futures
Типізація	Слабка динамічна	Сильна статична	Сильна статична	Слабка динамічна
Система управління пам'яті	Обмежена	Є	Є	Обмежена
Платформи	Laravel, Symfony, Yii	.NET Framework, .NET Core	Go Standard Library, Gin, Echo	Django, Flask
Реєстрація пакетів онлайн	Packagist	NuGet	Go Modules	PyPI
Structure, Enum, Traits	Немає Structure	Немає Traits	Немає Traits	Немає Traits
Система відлагодження коду	Xdebug, Zend Debugger	Visual Studio Debugger, .NET Debugger	Delve	Python Debugger
Застосування	Вебзастосунки, десктоп застосунки, мікросервіси	Вебзастосунки, десктопні застосунки, ігри, мобільні застосунки	Сервери, мережеві сервіси, мікросервіси	Штучний інтелект, вебзастосунки, десктоп застосунки, ігри.

Продовження табл. 2.1

1	2	3	4	5
Гнучкість	Висока	Середня	Середня	Висока
Швидкість розробки	Висока	Середня	Висока	Висока

Загалом мова RНР попри свою традиційну орієнтацію на веброзробку, активно розширює свої можливості в напрямку мультипроцесного й асинхронного програмування. Це стало основною причиною обрати саме RНР для реалізації проєкту. Поява в RНР підтримки файберів (fibers) і таких бібліотек, як AMPНР та SpatieAsync, дозволяє створювати високопродуктивні, асинхронні вебзастосунки, хоча це досить свіжі функції, вони демонструють потенціал для спрощення роботи з багатопотоковими процесами. Інші мови, як-от Python і Go, мають велику кількість готових рішень, бібліотек і документації для асинхронного програмування. У RНР ця сфера тільки починає активно розвиватися, і тому вивчення й реалізація проєктів у цьому напрямку є інноваційною та актуальною задачею. Також RНР залишається простою для використання мовою з низьким порогом входу, що робить її ідеальною для швидкого створення прототипів. Слід зазначити, що унікальною перевагою RНР є підтримка проєктів із застарілим кодом і можливість поступового впровадження нових технологій без ризику втрати сумісності, що посилює важливість, так як завдяки легкій оптимізації можна зекономити купу часу і енергії.

Для створення рівня інтерфейсу у запропонованій архітектурі JavaScript (JS) є найкращим вибором завдяки його динамічності, гнучкості та підтримці сучасними вебтехнологіями. JavaScript, як одна з найпопулярніших мов програмування, має низку переваг, які роблять її ідеальною для побудови фронтенд рішень. Перелік особливостей:

- об'єктно-орієнтовна мова;
- підтримка асинхронності;

- присутні можливості багатопотоковості, завдяки використанню WebWorkers;
- динамічна типізація;
- підтримується всіма популярними браузерами, такими як Opera, Firefox, Chrome, Microsoft Edge;
- потужні бібліотеки React.js, Vue.js, Angular.js;
- універсальна бібліотека Axios, що надає можливість обміну даними за допомогою HTTP (Hypertext Transfer Protocol – це високорівневий протокол, що забезпечує обмін медійними даними між клієнтом і сервером. Його основна мета – передача гіпертекстових документів через мережу [20]) 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;
- розвинена спільнота, велика кількість форумів, навчальних матеріалів, відео уроків;
- для регулювання версій пакетів існують декілька менеджерів пакетів yarn і npm.

2.2 Огляд особливостей обраних платформ і бібліотек

MovieMicroservice, EmailTemplateMicroservice, MovieMasterStorage мікросервіси розроблено на основі платформи Laravel версії 9.19 і додатково встановлених пакетів SpatieAsync, AMPHP та розширення pcntl. Платформа Laravel платформа створена для розробки монолітних сайтів, застосунків з інтерфейсом API, проте останні версії, що підтримують сучасні функції останньої версії PHP не є безпечними в контексті контролю ресурсів і регулювання потоків, тому було використано додаткові бібліотеки.

Переваги Laravel для створення МС полягає у кількості готових рішень, які дозволяють створювати прототипи систем і прискіпливо досліджувати процеси. Переваги фреймворку:

- відсутність перешкод для інтеграція обраних пакетів та додаткових технологій для мультпроцесної обробки відкладених завдань;
- сучасна система логування;
- наявність ServiceProvider класів, що допомагають у реалізації шаблону проєктування DependencyInjection, який представляє собою метод досягнення інверсії між класами;
- наявність готової інфраструктури для реєстрації API шляхів, роботи з HTTPS протоколом, приклад готових класів: Route, Request, Controller, Resource;
- наявність базового класу Model і конструктора запитів Eloquent, фасаду DB для роботи з сутностями у БД, даний фасад надає можливість логувати тіло запиту, час і порядок його виконання системою;
- наявність конструктору таблиць для розробки міграцій БД;
- інтегрований консольний інтерфейс Artisan, який дозволяє виконувати різноманітні завдання: управління міграціями БД, створення провайдерів, моделей і контролерів, а також роботу з ключами, кешами. Крім того, artisan підтримує розширення Laravel Tinker, яке дозволяє взаємодіяти з базою даних через запити, створені за допомогою моделей або фасадів;
- забезпечує повну сумісність із платформою PHPUnit, що дозволяє створювати і виконувати інтеграційні тести. Зокрема, швидко налаштувати тимчасову БД на основі SQLite для тестування;
- Laravel найпопулярніша платформа з платформ PHP [21].

Для створення вебсервера було обрано бібліотеку React, яка є потужним інструментом на базі JavaScript для розробки швидких та ефективних вебсайтів. Вона дозволяє створювати динамічні інтерфейси за допомогою використання вбудованих хуків і керування станами. Однак варто відзначити, що React має деякі приховані можливості, які складно виявити з

технічної точки зору через відсутність вбудованої системи налагодження коду в реальному часі:

- підтримка віртуальної DOM (Document Object Model – це спосіб подання об'єктів, що утворюють структуру та вміст вебдокумента в Інтернеті [22] – [23]) – React.js використовує віртуальну DOM, що зменшує витрати на зміну стану компонентів, покращуючи продуктивність вебзастосунків;
- підтримка серверного рендерингу – React.js дозволяє виконувати рендеринг компонентів на сервері, що не тільки підвищує продуктивність, алей й забезпечує покращену SEO оптимізацію (Search Engine Optmization – процес оптимізації для конкретних пошукових систем [24]);
- масштабованість, бібліотека надає можливість легко масштабувати проекти, завдяки створенню повторно використовуваних компонентів, що можна комбінувати між собою;
- модульність, у React.js можна розділити код на невеликі керовані частини, що спрощує його підтримку та тестування;
- зручне керування станом компонентів, що дозволяє спростити розробку і зробити її більш зрозумілою;
- Підтримка інтеграції з іншими бібліотеками та фреймворками, такими як Redux, для управління станом, що дозволяє створювати більш складні та масштабовані вебзастосунки.

2.3 Огляд особливостей обраних інтерпретаторів

Для запуску і розгортання багатьох МС на мові PHP та JS було розроблено Docker файл, що створює окрему віртуальну машину для запуску серверу Nginx і інтерпретатор PHP-FPM (менеджер процесів PHP). Nginx це вебсервер, який потребує встановлення інтерпретатору PHP-FPM і створення конфігураційного файлу у папці “sites-available” та “sites-enabled”. Після створення конфігу з доменним іменем, обраним socket для обробки запитів, протоколу спілкування і перенаправлення запитів на виконуючий файл, все

що треба це запустити докер, який проініціалізує сервер з кешованих образів або створить новий. Таким чином було створену відокремлену середу з обраними інтерпретаторами, яку не треба пеерезапускати за допомогою купи консольних команд і або постійно використовувати кнопку запуску проєкту у обраній IDE або інші способи, які потребують систематичного ручного втручання.

Переваги Nginx:

- підтримка багатопоточності, завдяки можливості обробляти декілька запитів одночасно, сервер може ефективно підтримувати багатозадачність на сервері, що важливо для масштабуваних і всиконавантажених додатків, зокрема у контексті мікросервісної архітектури;
- можливість чітко визначити ресурси під час створення файлу конфігурацій логічного серверу, налаштувати кешування, балансування захист від DDoS (Distributed Denial of Service Attack – це атака, коли зловмисник навмисно перевантажує сервер трафіком, аби заблокувати доступ до онлайн-сервісів і сайтів.) атак;
- висока продуктивність, сервер здатен обробляти значні обсяги запитів і трафіку без втрати продуктивності, завдяки ефективному використанню паралельних запитів і невеликому споживанню пам'яті;
- надійність, сервер забезпечує стабільну роботу сервера, навіть у випадках надмірного навантаження або атак;
- підтримка автоматичне відновлення збоїв, забезпечує безперервність роботи;
- 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 зловмисників.

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. Проте головний МС розташований у Docker-контейнері поряд з проєктом для інтерфейсу, що дозволяє зручно організувати запуск і взаємодію між сервісами. При запуску Docker контейнера автоматично ініціалізується сервер, що використовує Node.js, який відповідає за обробку коду на React. Для цього у проєкті використовується комбінація кількох технологій, що спрощує процес розгортання. Код на React перед запуском обробляється за допомогою інструментів, таких як Babel і Webpack. Це необхідно для того, щоб перетворити JSX код у звичайний JavaScript, який може виконуватись у браузері. Після цього, готовий JavaScript код обробляється за допомогою вбудованого в браузерного движка, зокрема у Firefox використовується SpiderMonkey. Це дозволяє безперешкодно запускати React застосунок у браузері. Для розгортання проєкту докер автоматично кожного разу виконує команду “npm start”, тобто не можна сказати що браузер повністю обробляє.

2.4 Огляд архітектури DDD

Архітектура Domain Driven Design (DDD) орієнтована на відображення реальних бізнес процесів у програмному забезпеченні. Одним з основних принципів цієї архітектури є використання “Ubiquitous language” – універсальної мови, яка дозволяє всім учасникам проєкту (менеджерам продукту, розробникам, дизайнерам) спілкуватися за допомогою однакових термінів. З точки зору розробника це забезпечує чітке розуміння концепцій та спрощує стандартизацію назв класів, структур і їх взаємозв'язків, хоча деякі розробники намагаються скоротити назви класів або змінних, що порушує не тільки концепт описаний у DDD, а і у книзі “Clean Architecture” [25]. Проте, незважаючи на недоліки довгих назв, такий підхід дозволяє звести до мінімуму можливі непорозуміння та зберегти бізнес логіку проєкту на кожному етапі його розробки.

DDD структуровано на чотири основні рівні: Application, Domain, Infrastructure. Ці рівні допомагають організувати код таким чином, щоб бізнес логіка (Domain) була максимально ізольована від інших частин системи, таких як інфраструктура чи користувацький інтерфейс. Деякі джерела також виділяють окремо User Interface як рівень, який відповідає за взаємодію користувача з системою, будь то через консольні команди, вебзастосунки чи інші зовнішні інтерфейси. Ця концепція рівнів описана у книзі “Domain Driven Design Quickly”, що детально розглядає основні принципи та практики впровадження DDD. Рівень User Interface важливий тим, що забезпечує точку входу для зовнішнього середовища, дозволяючи системі взаємодіяти з користувачами та іншими сервісами [26] – [27].

2.4.1 Application рівень

У Application рівень це рівень який сприймає і повертає дані, делегує дані в обробку Domain рівню, завдяки його не великому об’єму функції і їх відносної простоти можна ефективно використовувати асинхронні запити, щоб оптимізувати обробку даних. Завдяки асинхронній реалізації теоретично класи Controller можуть більш гнучку взаємодіяти з Domain рівнем, також у цьому випадку, замість того щоб чекати на завершення обробки запиту, контролер передає його до домену, де обробка даних може здійснюватися без блокування основного потоку, що підтримує високий рівень продуктивності та швидку реакцію системи, оскільки контролер може продовжувати виконання інших завдань, поки очікується відповідь від домену. Що стосується класів Middleware, то вони також можуть працювати асинхронно, але це залежить від їх специфіки та завдань. Наприклад, якщо middleware виконує операції, які потребують обробки великих обсягів даних або взаємодії з зовнішніми сервісами, як перевірка доступу або обробка файлів, він може бути налаштований на асинхронну обробку, щоб не блокувати основний потік запиту, однак є операції на перевірку токена, які дещо

блокують виконання потоку і тому є прикладом операції, яку не можна занести у мультипроцесне або асинхронне виконання. Водночас, Domain рівень має більшу автономність у визначенні того, як і коли виконувати обробку, не залежачи від деталей реалізації Application рівня. Домени можуть вирішувати, коли і як виконати складні операції, обробку запитів або взаємодію з іншими компонентами, таким чином приховуючи ці деталі від інших рівнів. Завдяки такому підходу можна більш гнучко керувати процесами в рамках системи, що є однією з ключових переваг DDD. Схема взаємодії класів у Application рівні наведена на рисунку 2.1.

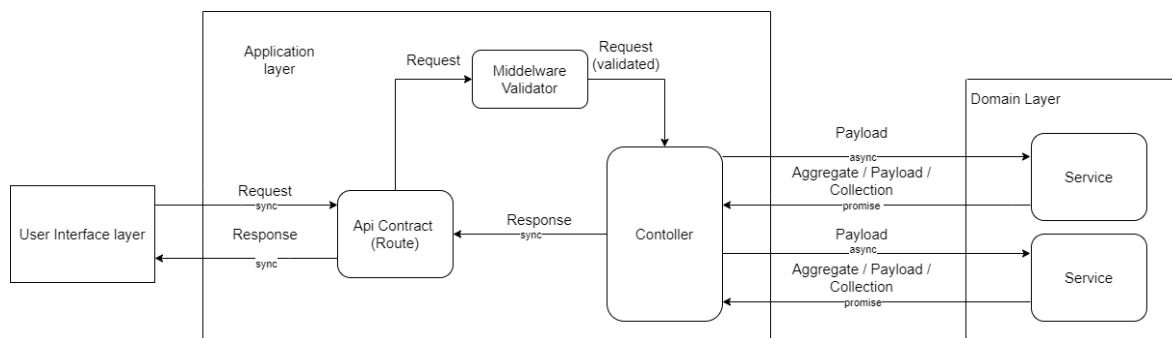


Рисунок 2.1 – Взаємодія класів application рівня

2.4.2 Domain рівень

Domain рівні архітектури DDD є найбільшим шаром за кількістю реалізованої логіки і логічному навантаженню. У реалізації таких процесів важливе місце займають шаблони проєктування, такі як агрегати, сутності, сервіси та репозитарії, що добре описано у книзі “Clean Architecture” [25]. Одним із ключових принципів DDD є чітке визначення меж узгодженості між класами на Domain рівні, що дозволяє організувати їх так, щоб кожен клас був відповідальним за певну частину бізнес логіки. Зокрема, більшість логіки переміщається із сервісів в агрегати, які стають центром управління станом сутностей.

Агрегати зазвичай включають одну сутність і багато підсуб'єктностей, що допомагає ефективно контролювати їхній стан [28] – [29]. У взаємодії з Application рівнем агрегати отримують дані у вигляді Payload, що є об'єктами корисного навантаження домену (Domain Payload Objects). Це дозволяє Application рівню передавати лише ті дані, які необхідні для виконання бізнес логіки, уникаючи надмірної взаємодії з деталями домену.

На рівні Domain також застосовується принцип асинхронності та багатопоточності для покращення продуктивності. Наприклад, створення, оновлення або вибірка даних в агрегатах може виконуватися асинхронно, що знижує час очікування та підвищує ефективність обробки запитів. Domain рівень залишається незалежним від реалізації Application рівня, дозволяючи йому самостійно визначати, коли і як виконувати асинхронні операції, не враховуючи специфіки зовнішньої логіки.

Клас агрегат може включати складні перевірки інваріантів, а сервіси підтримують виконання CRUD операцій (CRUD – це скорочення від англійських слів, а саме створення, оновлення, вибірки, видалення) і бізнес правил [30]. Для вирішення проблеми надмірної складності агрегатів, які можуть перетворитися у “суперкласи”, використовуються такі підходи, як розділення на trait. Це дозволяє розбивати логіку на дрібніші модулі. Domain рівень також визначає інтерфейси для репозитаріїв та сервісів, забезпечуючи високий рівень абстракції і гнучкості. Загалом, роль Domain рівня полягає у тому, щоб бути незалежним від інших рівнів, але забезпечувати їх необхідною бізнес логікою. Архітектура класу агрегату у DDD базується на чіткій структурі, яка складається з полів сутності, методів для роботи з ними, наприклад, сетерів і гетерів, а також додаткових компонентів, що забезпечують взаємодію з іншими частинами системи. Зокрема, агрегат може використовувати trait resolvers, які дають змогу звертатися до сервісів, репозитаріїв, шини подій, класів менеджерів, валідаторів, конфігурацій та інших структур. Цей підхід дозволяє уникнути важких залежностей у класі, що сприяє простішому тестуванню та підтримці коду. Також основна

перевага такого підходу полягає у застосуванні шаблону проєктування “Ін’єкція залежності” (Dependency Injection), який мінімізує жорстке зв’язування класів. Таким чином, якщо сутність, яку представляє агрегат, має підсуб’єктності, то в агрегаті додається нове поле для роботи з цими даними, а ін’єкція залежності надає можливості у будь який момент часу отримати дані за допомогою сервісу чи репозитарію. Така реалізація додавання залежностей дозволяє забезпечити цілісність логіки для підсуб’єктностей та їх зв’язок із головною сутністю. Агрегати також можуть включати структури, такі як Value Object. Це класи, які використовуються для відображення даних сутності в агрегаті. Особливістю Value Object є відсутність унікального ідентифікатора та незмінність [31].

2.4.3 Infrastructure рівень

Третій рівень архітектури DDD, Infrastructure, відповідає за інтеграцію бізнес логіки з БД та іншими зовнішніми системами. Цей рівень містить моделі, конструктори запитів і класи репозиторіїв, які реалізують взаємодію з даними. У Laravel реалізація цього рівня значно спрощується завдяки класам моделей, що дозволяють визначати структури даних, зв’язки між ними та виконувати запити через вбудований SQL-конструктор. Такий підхід забезпечує швидкий розвиток, проте його основою є шаблон “Активний Запис” (Active Record), що має обмеження, особливо при виконанні складних операцій, які можуть генерувати кілька зайвих запитів до бази даних. Це знижує продуктивність системи. Для вирішення цих проблем у контексті DDD часто застосовуються два основні шаблони: UOW, CQRS.

Unit of Work (UOW) – цей шаблон створює рівень абстракції між доступом до даних та бізнес логікою. Він дозволяє зберігати стан сутностей у межах однієї транзакції, зменшуючи кількість звернень до бази даних. Наприклад, у процесі зміни сутності всі її проміжні стани можуть зберігатися у спеціальній карті, а до бази даних здійснюється лише два запити: один для

початкової вибірки даних, а інший для фінального збереження змін. Такий підхід не лише підвищує продуктивність, але й забезпечує узгодженість даних у межах однієї бізнес транзакції. Водночас реалізація UOW у готовій системі може бути складною та вимагати додаткових ресурсів [32] – [33].

CQRS (Command and Query Responsibility Segregation) – це шаблон, що відокремлює операції читання і запису в сховищі даних, покращуючи продуктивність, масштабованість і безпеку. База даних для запису має спрощену структуру без індексів і зовнішніх ключів, що пришвидшує вставку великих обсягів даних. База для читання, навпаки, оптимізована для швидкої вибірки, маючи повноцінну структуру з індексами. Синхронізація між базами даних зазвичай виконується асинхронно. Використання CQRS підвищує масштабованість і продуктивність, особливо у високонавантажених системах [34] – [35].

Ці шаблони допомагають оптимізувати роботу Infrastructure рівня, дозволяючи зберігати баланс між продуктивністю та гнучкістю архітектури. Завдяки їх використанню можливо досягти ефективної реалізації складних сценаріїв взаємодії з даними, як це продемонстровано на схемі рисунку 2.6.

2.4.4 User Interface рівень

У Domain Driven Design (DDD) рівень User Interface (UI) не має чітко визначених архітектурних принципів, оскільки його основне завдання – це забезпечення ефективної взаємодії користувача з багатьма МС та забезпечення участі в розроблених бізнес сценаріях. На цьому рівні важливо зберігати гнучкість для інтеграції з різними МС, що потребує застосування таких архітектурних підходів, як шаблон Model-View-Controller (MVC) та використання репозиторіїв. Model складається з класів, що зберігають дані, отримані через API, що дозволяє системі бути масштабованою та зручною для подальшої інтеграції. Використання React і Bootstrap у View дозволяє створювати адаптивні інтерфейси, що можуть ефективно працювати на

різних пристроях, забезпечуючи відмінний досвід користувача. Controller в такій архітектурі є важливим елементом, оскільки він відповідає за логіку обробки запитів, таких як перевірка введених даних або оновлення моделей через API запити. Такий підхід дозволяє зберегти високий рівень відокремленості між бізнес-логікою і інтерфейсом користувача, що спрощує розробку та тестування. Використання Dependency Injection допомагає уникнути сильних залежностей між компонентами, підвищуючи гнучкість і підтримуваність коду. Окрім того, це дозволяє інтегрувати різні технології без необхідності переписувати великий обсяг коду. Користувацький інтерфейс при цьому залишається прозорим та модульним, що робить можливим його масштабування та оновлення без значних переробок у бізнес-логіці. Візуальна схема рівнів наведена на рис. 2.2.

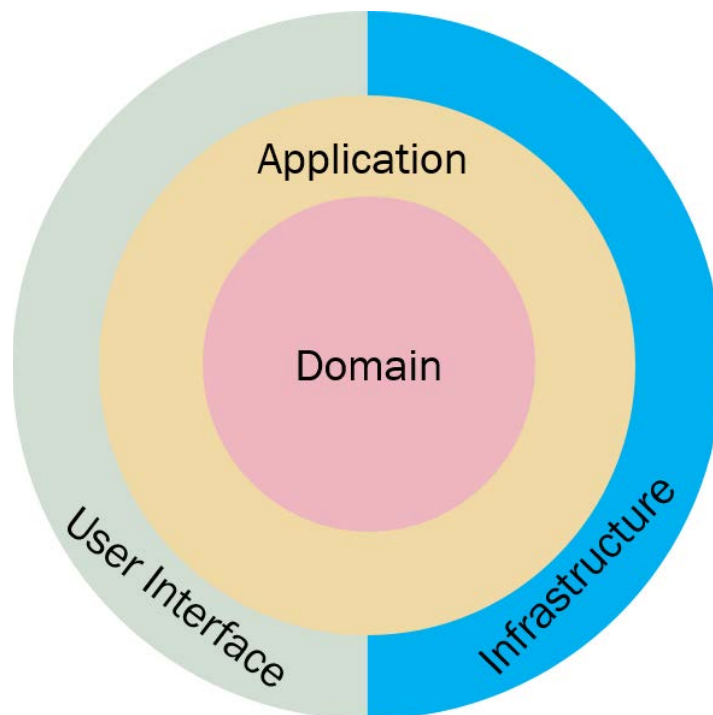


Рисунок 2.2 – Схема рівнів DDD

2.4.5 Зв'язок мікросервісів

Усі МС взаємодіють між собою через PCO. Існує два основні підходи до реалізації цього контракту. Перший полягає в тому, що МС розташовані в

межах одного git репозиторію та мають спільний простір імен. У цьому випадку PCO представлений простим сервісом, який з'єднує сервіси з двох MC, дозволяючи обробляти відповідні агрегати або колекції. Це спрощує процес, оскільки потрібно лише розробити інтерфейс та клас-сервіс, який забезпечує відокремлення логіки Domain рівня одного MC від іншого..

Другий підхід застосовується, коли MC розташовані в різних git проєктах і не мають спільного простору імен. У цьому випадку взаємодія з іншими мікросервісами здійснюється через сервіси та репозитарії, які підключають сторонні API. Тут сервіс належить до рівня домену, а взаємодія з API реалізується на рівні Infrastructure. Такий підхід потребує визначення інтерфейсу для сервісу, при цьому сам сервіс та API повинні бути упаковані у вигляді пакету, який можна підключити до будь-якого іншого MC.

2.5 Огляд особливостей обраної СКБД

Основною СКБД (СКБД – система керування баз даних) для виробничого середовища було обрано MySQL 8.0.32, оскільки це остання стабільна версія. MySQL є реляційною системою управління базами даних, яка дозволяє легко створювати зв'язки між таблицями за допомогою зовнішніх ключів. Ця версія підтримує пошук за регулярними виразами та пошук по полям типу JSON, а також роботу з датами, включаючи секунди та мілісекунди. MySQL підтримує кілька важливих механізмів для забезпечення ефективної роботи з даними. Це включає:

- двіжок InnoDB, який є основним механізмом зберігання даних у MySQL. InnoDB забезпечує високий рівень продуктивності, підтримку транзакцій, а також можливість робити ACID (атомарність, консистентність, ізольованість, довговічність). Це дозволяє СКБД виконувати складні операції з даними зберігаючи їх цілісність і стабільність.

- має механізм блокування для забезпечення синхронізації доступу до даних в умовах одночасних транзакції і забезпечує високу продуктивність при обробці великої кількості запитів.

Основні переваги MySQL включають

- 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”);
- надійність і легкість використання.

2.6 Огляд особливостей обраних бібліотек і розширень

У цьому розділі буде розглянуто три технології, що мають схожі можливості для асинхронної обробки та паралельного виконання задач у PHP, а саме: PCNTL, SpatieAsync та AMP. Незважаючи на подібність у їх призначенні, кожна з цих технологій має свої унікальні особливості та підходи до реалізації. Буде описано, як вони працюють, які функції надають для обробки паралельних завдань, і чим відрізняються одна від одної в контексті використання ресурсів, багатопоточності та мультипроцесності.

2.6.1 Розширення pcntl

PCNTL – це розширення PHP, яке надає низькорівневі інструменти для керування процесами, що робить його зручним для мультипроцесорних операцій у синхронному середовищі PHP. Воно дозволяє створювати,

контролювати та керувати процесами, надаючи базові функції для роботи з сигналами процесів, дочірніми процесами та керуванням потоком виконання. Основні функції включають `pcntl_fork()`, що створює новий дочірній процес, та `pcntl_wait()`, яка дозволяє батьківському процесу чекати завершення роботи дочірнього. Також є функція `pcntl_signal()`, яка дає змогу обробляти сигнали Unix, такі як зупинка або переривання процесу. PCNTL працює через використання можливостей операційної системи для керування процесами. Коли відбувається форк (створення нового процесу), цей процес майже ідентичний батьківському, і його можна використовувати для виконання різних завдань. Це є ефективним у випадках, коли задачі можуть бути виконані паралельно та незалежно. Проте PCNTL працює синхронно, що означає, що для обробки результатів потрібно використовувати техніки, як-от `pcntl_wait()`, що може бути не ідеальним для високопродуктивних завдань, де потрібна асинхронна обробка. З точки зору використання ресурсів, підхід PCNTL є відносно ресурсоемним, оскільки кожен процес, створений через `pcntl_fork()`, використовує додаткову пам'ять і процесорний час. Це обмежує кількість процесів, які можуть бути запущені одночасно на системах з обмеженими ресурсами. PCNTL добре підходить для сценаріїв, де потрібно створювати кілька процесів для паралельної обробки, але може мати проблеми зі масштабуванням, особливо в умовах високої навантаженості чи обмежених ресурсів [36].

2.6.2 Бібліотека SpatieAsync

SpatieAsync – це PHP бібліотека, яка забезпечує зручний спосіб роботи з багатопроектністю та асинхронним виконанням завдань. Вона дозволяє запускати задачі у фонових процесах, звільняючи основний потік для інших операцій. Структура базується на основі Symfony Process і надає зрозумілий API, що спрощує використання асинхронності навіть без глибоких знань про багатопоточність. Завдяки використанню окремих

процесів Spatie Async дозволяє ефективно розподіляти завдання між ядрами процесора, що особливо корисно для ресурсомістких операцій, таких як обробка великих масивів даних, робота з API або генерація звітів.

Ключовою особливістю є підтримка промісів, що забезпечує можливість відстеження результатів виконання задач і обробки помилок. Завдання додаються до пулу, виконуються у фоновому режимі, а після завершення їхні результати стають доступними для подальшої обробки. Крім того, Spatie Async дозволяє налаштувати кількість одночасно виконуваних процесів, оптимізуючи використання системних ресурсів. Такий підхід підходить для реалізації паралельних обчислень у мікросервісній архітектурі, де важливо забезпечити масштабованість і швидкість.

Хоча бібліотека спрощує роботу з асинхронністю, запуск окремих процесів може споживати більше пам'яті, ніж рішення, засновані на подієвих циклах, такі як ReactPHP або AMPHP. Проте Spatie Async є чудовим вибором для завдань, які не потребують високої продуктивності в реальному часі, але вимагають простоти реалізації та зручного управління паралельністю [37].

2.6.3 Бібліотека AMPHP

AMP – це фреймворк-бібліотека для мультипроцесного, асинхронного програмування і обробки паралельних завдань у PHP. Бібліотека наслідує деякі функції з бібліотеки ReactPHP і значно розширює можливості підтримкою мультипроцесорних і багатопоточних операцій завдяки застосуванню поєднання вбудованих у PHP або у розширення PCNTL функцій “stream_context” і “pcntl_fork”. AMP надає більш високу абстракцію для управління конкурентними задачами, дозволяє виконувати асинхронні операції, при цьому зберігаючи синхронний стиль коду. Це дає змогу писати асинхронний код, який виглядає та поводить ся як звичайний блокуючий код, але при цьому використовує всі переваги асинхронних операцій.

AMP підтримує асинхронне введення та виведення (I/O), а також надає інструменти для виконання CPU завантажених задач у багатопоточному режимі або через мультипроцесорність. Такі можливості управління реалізовані завдяки класу `amp\Promise`, що представляє майбутній результат, і класу `amp\Loop`, що управляє потоками виконання операцій, що у свою чергу налає змогу керування багатьма задачами одночасно. До однієї з важливих переваг відноситься можливість використовувати багатоядерні системи для паралельної обробки завдань, що навіть дає змогу масштабувати РНР додатки на кілька ядер або фізичних машин. Це робить AMP більш гнучким і потужним у порівнянні з `SpatieAsync`, який більше орієнтований на I/O завантажені задачі. З опису джерел було встановлено, що можливим недоліком такої розвиненої бібліотеки є надмірна витрата ресурсів порівняно з аналогами [38] – [39].

2.7 Висновки з розділу

У розділі розглянуті ключові аспекти вибору мов програмування, платформ, архітектури і бібліотек для реалізації принципів DDD в умовах застосування складних оптимізацій за рахунок мультипроцесних і багатопоточних процесів, реалізація яких базуються на сучасних провідних технологіях. Дослідження комбінації кожного з перерахованих аспектів дозволяє сформулювати теоретичну архітектуру і напрямок для початку реалізації модульних компонентів у масштабованій і ефективній системі. Завдяки вивченню аспектів технологій асинхронності і багатопоточності дає можливість адаптувати систему до різноманітних вимог і забезпечити її гнучкість у процесі розвитку. Розглянуті бібліотеки і розширення пропонують різні підходи до обробки паралельних та довготривалих задач, застосовуючи одночасно декілька підходів включаючи мультипроцесність, багатопоточність і асинхронність для підвищення продуктивності.

3 ОСНОВНІ РІШЕННЯ З РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ

У даному розділі розглянуті основні рішення щодо впровадження багатопоточних і асинхронних функцій у застосовані шаблони програмування, розроблені МС, БД, РСО, алгоритми в умовах їх взаємо впровадження з іншими технологіями у проєкті.

3.1 Основні рішення щодо розроблених МС і РСО

Мікросервісна система MasterMovie включає має клієнт-сервер MasterMovieClient, розроблений на JavaScript з використанням бібліотеки React, Він може взаємодіяти з головним і поштовим мікросервісами, які побудованими на мові PHP та фреймворці Laravel. Поштовий МС використовує Mailgun для надсилання електронних листів, а MovieMicroservice інтегрується з Firebase як сховищем даних. Для зручності, РСО для PHP буде доступний на Packagist як пакет BestMovieCommon, тоді як для JavaScript реалізовано спрощений підхід з React та Axios.

Система працює з основними сутностями – користувачем, фільмом, колекцією, та надає три ролі користувачів: адміністратор, звичайний користувач та зареєстрований користувач. Адміністратор має повні права на редагування фільмів і колекцій, а звичайний користувач може переглядати фільми та реєструватися. Зареєстровані користувачі мають можливість створювати персональні колекції. Реєстрація здійснюється лише за допомогою коду підтвердження, що надходить на електронною пошту. MasterMovieClient перевіряє поля на клієнтській стороні та відправляє запит до MovieMicroservice, який в свою чергу надсилає запит на EmailTemplateMicroservice для створення і відправлення листа. Зв'язок між MasterMovieClient і MovieMicroservice реалізовано через розроблений РСО пакет.

У системі існує кілька процесів авторизації: для взаємодії між МС і між користувачами та МС. Один із підходів передбачає створення загального API контракту для різних користувачів, інший – окремі контракти для кожного типу користувачів. Реалізація пакету залежить від бізнес логіки, яка може сильно впливати на підхід до вирішення задачі. Для забезпечення безпеки авторизація між МС передбачено перевірку хеш ключів, що дозволяє уникнути підміну запитів і загалом запобігає багатьом загрозам.

Особливістю системи є асинхронний процес завантаження файлів. При завантаженні MasterClient відправляє запит до MasterMovieStorage без блокування на час відповіді від сервера. Центральне сховище даних — Firebase, що дозволяє обробляти запити без навантаження на центральний MovieMicroservice, який відповідає за авторизацію. У разі потреби, авторизація може бути винесена на окремий МС для більш складних структур доступу. Всі процеси відповідають стандартам PSR.

3.1.1 МС MasterClient

На головній сторінці сайту користувач бачить перелік фільмів, категорій та колекцій за замовчуванням. Також є окремий компонент навігації, що є динамічним і змінюється в залежності від статусу користувача. Звичайний користувач має доступ до таких сторінок: авторизація, реєстрація, перегляд усіх фільмів, перегляд конкретного фільму, а також до сторінки “Про нас”. Адміністратор має доступ до базових сторінок, але з розширеними можливостями, що включають додавання і видалення категорії, колекції, а також керування фільмами. Для навігації по сайту та обмеження доступу до певних сторінок був створений перелік шляхів у класі App.js за допомогою компонентну React DOM Router. Для того щоб реалізувати перевірку прав доступу користувача було розроблено компонент PersistLogin. Компонент базується на маніпулюванні даними у локальному сховищі. Завдяки такому підходу спосіб перевірки доступу до сторінок

здійснюється без зайвих запитів до сервера, що покращує продуктивність за рахунок зменшення запитів і часу очікування на перевірку.

Сторінка авторизації містить два поля для введення і одну кнопку, а саме: електронна пошта та пароль, кнопка для підтвердження авторизації. Перевірка введених даних відбувається через хук `useLogin`, що забезпечує ініціацію запиту до центрального МС, що у свою чергу перевіряє існування користувача і його доступи. Після успішної перевірки та авторизації, користувач автоматично перенаправляється на головну сторінку. Як було зазначено раніше токен для авторизації зберігається в локальному сховищі і використовується для перевірки доступу до інших сторінок при кожному переході. Права доступу перевіряються у компонентах `RequireAdminRoleAuth` та `RequiredViewerAuth`. Перший компонент перевіряє, чи є користувач адміністратором, а другий чи є користувач зареєстрованим. Цей підхід запобігає несанкціонованому доступу до адміністративних функцій чи особистих колекцій користувачів. Цей процес не потребує додаткових запитів на кожну сторінку, що забезпечує швидкий доступ до ресурсів. Частково така реалізація легка у впровадженні через те, що сторінки у React це частина “single-application” підходу. Після успішної авторизації, користувач потрапляє на відповідну сторінку.

На головній сторінці сайту відображається перелік фільмів, категорій і колекцій за замовчуванням, а також окремий компонент навігації `NavBar`, який змінюється в залежності від стану користувача. Логіка навігації виконана досить просто – всі посилання для переходів на інші сторінки використовують тег “`<a>`” або код що перемикає вікно через об’єкт `window`, що дозволяє уникнути необхідності створення окремого хука для управління станом сторінки. Логіка вибірки фільмів організована через хук `useMovies`, який реалізує фільтри і посторінкову навігацію. Хук відповідає за обробку зміни номеру сторінки та за отримання фільмів через сервіс `MovieApiService`. Залишкова частина логіки по роботі з категоріями та колекціями організована через хуки `useViewerMovies`, `useCategories` і `useCollections`.

Компоненти `MovieList` і `MovieListItem` використовуються для представлення списку фільмів. Для адміністратора також реалізовані додаткові компоненти, які працюють з категоріями та колекціями, зокрема через компоненти `useCategoryCheckedList` та `useCollectionsCheckedList`. Ці компоненти надають змогу фільтрувати списки фільмів за категоріями і колекціями відповідно.

Сторінка власних колекцій користувача реалізована за допомогою компонент `useMyMovieCollections`, що надає змогу фільтрувати за обраними колекціями.

На сторінці конкретного фільму користувач має можливість переглядати детальну інформацію про фільм. Для перегляду відео на сайті було розроблено окремий компонент `MoviePlayer`, що базується на використанні стандартного тегу “`<source>`”, оскільки готові React компоненти не підтримують трансляцію відео з Firebase, хоча Firebase не підтримує пряме транслявання відео, на практиці все працює коректно і тег `<source>` самостійно справляється з контролем відтворення відео. На сторінці також є можливість додавати фільм до колекції або видаляти фільм. Усі ці функції реалізовано через окремі модальні вікна, що використовують бібліотеку `Formik`. Бібліотека `Formik` перевіряє введені дані та відображає сповіщення про помилки. Адміністратори мають доступ до кнопок управління фільмами, категоріями та колекціями, що дозволяє зручніше керувати контентом сайту. Загалом інтерфейс забезпечує багатофункціональність і зручність для звичайних користувачів і адміністраторів, надаючи доступ до перегляду контенту і управління ним, що робить використання сайту гнучким і зручним для різних типів користувачів.

Так як адміністратор має більше можливостей, то головна сторінка адміністратора включає окремий компонент `AdminNavBar`, також це зроблено з метою уникнення великих гілок розгалуження у компонентах. Логічні зв'язки і структура використання хуків та компонентів залишається аналогічною в незалежності від користувача, однак усі файли для

адміністратора зберігаються в окремих провідниках. У самих компонентах є додаткові кнопки для додавання та видалення категорій, колекцій та фільмів. Реалізація функцій винесена у модальні вікна, такі як `MovieCollectionModal`, `MovieCategoryModal` і `MovieAddModal`. Всі ці вікна базуються на бібліотеці `Formik`. Завдяки модальним вікнам користувач може додавати нові категорії, колекції або фільми, а також редагувати наявні дані, що значно покращує зручність адміністрування сайту.

Посилання на медіа файл передається разом з відповіддю при отриманні конкретного фільму за унікальним ідентифікатором. Таке рішення є оптимальним, так як дозволяє знизити навантаження на сервер завдяки використанню ресурсів `Firebase` для зберігання відео контенту. MC розташований у публічному репозитарії.

3.1.2 MC MovieMicroservice

`MovieMicroservice` виконує роль центрального MC, оскільки містить систему авторизації та БД. Його архітектура базується на п'яти контролерах: `CategoryController`, `CollectionController`, `MovieController`, `UserController`, які реалізують 19 API контрактів, визначених у файлі `api.php`. Кожен метод контролера відповідає окремому контракту, приймає нащадок класу `Request` і відповідає клієнту у форматі `JSON` за допомогою нащадків класу `Resource`.

Для обробки отриманих даних у рівні аплікації використовуються сервіси з рівню домену, ці сервіси обробники бізнес логіки що стосується отриманих і перевірених даних. Кожен сервіс взаємодіє з агрегатами, які або створюються, або витягуються з БД, і мають відомості про власний стан і обмеження по модифікації. Агрегати мають можливість інтегрують інші сервіси та репозиторії через трейти `Resolvers`, слід зазначити, що це не повноцінне створення нових сервісів чи репозитаріїв, а лише ін'єкція залежності через інструмент контейнера. Сервіси також отримують ін'єкції

залежностей через класи нащадки класу ServiceProvider та налаштування у файлі app.php.

У системі реалізовано комплексну архітектуру, що охоплює репозиторії, агрегати, сервіси, провайдери та механізми обробки помилок. Репозиторії містять базові методи для роботи з даними: отримання списку за фільтром, вибір однієї сутності за унікальним ідентифікатором та збереження сутностей. Для побудови SQL запитів використовуються методи моделей, зокрема applyToQuery, що дозволяє динамічно формувати запити, враховуючи посторінкову навігацію. Завдяки готовим метод Eloquent конструктора, що надають змогу легко створювати складні конструкції, такі як JOIN або інші підзапити для “lazy-loading” (lazy-loading – ліниве завантаження даних поєднаних таблиць), забезпечується оптимізація вибірки та зменшення кількості запитів до БД. Усі запити повертають моделі, які конвертуються в агрегати для подальшої обробки.

Взаємодія з іншими MC побудована завдяки інтегрованому пакету BestMovieCommon, який надає різні API контракти, в тому числі і контракти, що надають змогу обирати принцип обробки задачі, а саме обробляти задачу синхронно, асинхронно чи багатопоточно через відокремлені процеси по типу черг Laravel Horizon чи функції “pcntl_fork”. Також слід зазначити що не усі MC підтримують різні види обробки даних, це залежить від MC і контексту задачі, такий пакетний підхід забезпечує не тільки масштабованість і підвищення продуктивності, а ще запобігає хибному використанню розроблених контрактів або реалізованого інструментарію для пов’язання мікросервісів. Конкретний приклад можна побачити у реалізації публічних контрактів для EmailTemplateMicroservice, одна з імплементацій для генерації коду має 5 реалізаторів, а у MC MasterMovieStorage лише дві, так як завантаження файлу це синхронний процес, і лише окремі кроки обробки використовують або дозволяють використати асинхронність без технічних обмежень.

Наявна асинхронність у поточному МС реалізується через бібліотеку AMPHP, яка дозволяє оптимізувати роботу з БД, забезпечуючи високу продуктивність логічно навантажених процесів, як-от послідовна розсилка файлів налаштувань на різні МС або послідовне оновлення колекцій і категорій фільму.

Серед класів `middleware` реалізовано три класи: `AccessMicroserviceTokenCheck`, `AccessTokenCheck`, `AdminRoleValidator`. `AccessMicroserviceTokenCheck` перевіряє відповідність хешу токена в заголовку запиту та змінної середовища. `AccessTokenCheck` перевіряє токен користувача через `UserTokenService` і його ідентифікатор, ускладнюючи можливість злому, хоча такий спосіб сповільнює систему, так як має додатковий запит і відповідний час очікування. `AdminRoleValidator` дозволяє доступ до ресурсів тільки адміністраторам, перевіряючи роль користувача. Спільний функціонал токен перевірок винесено у трейт `AuthMiddlewareTrait`, що дозволяє уникати дублювання коду.

Контролер `UserController` реалізує п'ять ключових методів: попередня реєстрація `preRegister`, повна реєстрація `signUp`, авторизація `signIn`, оновлення даних `updateUser` і отримання інформації про користувача `getUser`. У методі попередньої реєстрації після валідації даних через `UserPreRegisterRequest` викликається метод `generateCode` сервісу `EmailTemplateService`, що забезпечує інтеграцію з `EmailTemplateMicroservice` для надсилання коду на пошту. Метод реєстрації виконує перевірку коду через той самий МС. Після успішної перевірки створюється агрегат `User`, який зберігається через `UserRepository` та отримує роль звичайного користувача через `UserRoleService`.

Метод авторизації використовує запит до `UserService`, який перевіряє існування користувача через метод репозиторію. Процес оновлення користувача схожий, але включає перевірку на існування сутності перед її зміною. Метод `refresh` оновлює дані користувача, викликаючи `getUserByToken` у `UserTokenService`.

У `CollectionController` реалізовано чотири методи: список колекцій “list”, отримання списку за замовченням “listOfDefaults”, створення “create” та видалення “deleteById”. Метод списку приймає фільтраційні параметри, формує `CollectionFilter` і передає його до `CollectionService`. Сервіс делегує обробку репозиторію `CollectionRepository`, а саме методу “save”, який повертає колекцію агрегатів. Агрегати перетворюються у формат JSON через `CollectionListResource`. Слід зазначити, що для обробки запитів у форматі JSON всі контролери використовують клас `UserResource`, який перетворює агрегати в асоціативний масив. Масив конвертується у JSON за допомогою функції “response->json()”. Для списку колекцій за замовченням використовується метод, що може кешувати результати для оптимізації. У разі зміни даних передбачено використання подій та обробників для автоматичного оновлення кешу. Метод створення визначає тип колекції залежно від ролі користувача: адміністратор – тип “DEFAULT”, звичайний користувач – “CUSTOM”, додаткова роль для тестування – “TEST”. Агрегат колекції зберігається через метод “save” у репозиторії. Метод видалення перевіряє доступ користувача до колекції за допомогою `deleteWithPermissionCheck`. Якщо користувач – адміністратор, то видаляються зв’язки між фільмами і колекціями через `MovieCollectionRepository`, а потім сама колекція. Для звичайного користувача перевіряється тип колекції “CUSTOM” та її приналежність до користувача що намагається її видалити.

У `CategoryController` реалізовано три методи: список категорій “list”, створення “create”, видалення “deleteById”. Список будується на основі `CategoryFilter`, що передається до `CategoryService`, а потім у `CategoryRepository`. Для створення категорії формується `CategoryCreatePayload`. Видалення відбувається після перевірки зв’язків між фільмами та категоріями, які видаляються перед операцією в БД.

`MovieController` має шість методів: список фільмів “list”, отримання одного фільму “getById”, створення “create”, синхронізація колекцій за

замовченням “applyDefaultCollection”, синхронізація звичайних колекцій “updateCollections”, оновлення категорій “updateCategories”. Методи вибірки фільмів реалізовані у MovieService, який звертається до MovieRepository. Створення фільмів включає перевірку медіа файлів через validatePath service MasterMovieStorage, через контракт у BestMovieCommon. Складна логіка синхронізації колекцій включає: знаходження користувача, який виконує синхронізацію, отримання і перевірки посилання на фільм у сховищі для синхронізації, пошук колекцій за унікальними ідентифікаторами з фільтром “CUSTOM”, звичайний пошук колекцій за ідентифікаторами, що вже прив’язані до фільму, але не пов’язаних з користувачем, формування масиву унікальних ідентифікаторів, збереження оновлених зв’язків через репозиторій, така кількість логіки дозволяє використати асинхронність, для пришвидшення виконання викликів у БД.

Процес синхронізації колекцій за замовченням допускає редагування усіма адміністраторами і передбачає аналогічний механізм для зв’язків між фільмами та колекціями.

Оновлення категорій перевіряє наявність категорій у БД і додає нові. Усі методи працюють із репозиторіями для виконання дій на рівні БД, забезпечуючи чітке розділення бізнес логіки та доступу до даних.

Агрегати, такі як Category, Collection, Movie, User і Role, містять базові поля сутностей, сетери, гетери та методи для бізнес логіки. Наприклад, User має методи “hasRole”, “isAdmin”, “addRole”, що динамічно керують станом підсуб’єктностей. Агрегат Movie через трейт Resolver взаємодіє з репозиторіями колекцій і категорій, а також сервісом медіа сховища. Його логіка включає побудову повного шляху до медіафайлів із використанням кешу, що значно прискорює роботу із запитамі. Агрегати колекцій і категорій додатково реалізують специфічні методи, наприклад, для перевірки типу колекції “isCustom”, “isDefault”.

Сервіс провайдери, розташовані у папці Providers, відповідають за реєстрацію класів, реалізацію інтерфейсів і підключення агрегатів до

репозиторіїв або сервісів за допомогою статичних методів резолверів. Це спрощує інтеграцію компонентів і підвищує модульність системи. Усі помилки обробляються через клас `Handler`, де стандартний механізм конвертації помилок адаптований під REST API і інші контракти. Відповіді формуються у форматі JSON із ключами “message”, “error_code”, “error_message”, а для уніфікації помилок створено перелік `ErrorCodes`, що охоплює сім основних типів. Система забезпечує ефективність через кешування запитів, оптимізовані зв’язкові таблиці для відносин багато до багатьох та використання агрегатів для зручного управління даними. Її структура дозволяє легко розширювати функціонал, мінімізуючи дублювання коду та забезпечуючи високу гнучкість у розробці. MC розташований у публічному репозитарії [40].

3.1.3 MC EmailTemplateMicroservice

У даному MC реалізовано два API контракти, які обробляються через контролер `CodeController` за допомогою двох сервісів: `MailService` та `CodeService`. Метод контролера `generateCode` викликає метод із такою ж назвою у `CodeService`, який відповідає за генерацію унікального коду, створеного на основі 4 випадкових байтів, перекодованих у формат hex (шістнадцятковий формат) за допомогою функції “bin2hex(random_bytes(4))”. Згенерований код зберігається в кеші на три хвилини під ключем, що містить адресу електронної пошти користувача. Після генерації код передається до сервісу `MailService`, який за допомогою методу `generateCode` і фасаду `Mail`, налаштованого для роботи з `Mailgun`, надсилає код на пошту користувача. Налаштування `Mailgun` визначені у файлі “.env”. Другий метод контролера `getCode` також викликає однойменний метод у `CodeService`, який витягує код із кешу за ключем.

Архітектура передбачає п'ять реалізацій методу `generateCode`, які відрізняються підходами до обробки запитів. У класі `SyncCodeService`

реалізовано послідовну синхронну обробку запитів без використання багатопоточності. У класі `PcntlForkCodeService` застосовується багатопроцесорний підхід за допомогою `"pcntl_fork"`, який створює окремі процеси для кожного запиту. Реалізація в `SpatieAsyncCodeService` використовує бібліотеку `SpatieAsync`, що забезпечує асинхронну обробку без блокування. `AmpPhpCodeService` обробляє запити через потоковий контекст та події, що реалізовані у бібліотеці `AmpPHP`, а `QueueCodeService` застосовує черги `Laravel Horizon` і `Redis` [41] – [42], де завдання додаються в чергу та обробляються у фоновому режимі. Останній реалізатор потребував додаткового встановлення бібліотек `"laravel/horizon"` та `"predis/predis"`, а також розширення `"php-relay"`. Також слід зазначити, що для функціонування асинхронних бібліотек було встановлено розширення `"php-pcntl"`. Загалом ці реалізації зберігають єдину сигнатуру методу `generateCode`, дозволяючи уніфікований виклик, але з використанням різних механізмів виконання.

Вибір потрібної реалізації здійснюється через `middleware DefineProcessServices`, яка визначає параметри запиту й обирає відповідний клас реалізатор із контейнера `Laravel`, керуючись переліком стратегій, визначених у переліку `ProcessHandleType`. Цей перелік містить можливі варіанти, такі як `SYNC`, `PCNTL_FORK`, `REACT_PHP`, `AMP_PHP`, `QUEUE`, і дозволяє легко розширювати набір сценаріїв обробки. `Middleware` отримує параметр реалізації з запиту та передає його в контейнер для ін'єкції відповідного класу. Усі реалізації інтегруються із фасадом `Cache`, який забезпечує збереження й отримання коду до відповідних сховищ, за замовчення кеш зберігається у вигляді файлів, але реалізація що застосовує черги використовує кеш `Redis`, і тому такий кеш налаштовано на збереження саме в оперативній пам'яті ПК. У реалізації `QueueMailerService` метод `"generateCode"` працює з чергами `Laravel` у реалізації через `QueueMailerService`, додаючи завдання на надсилання пошти до черги, що дозволяє оптимізувати ресурси та забезпечити швидкий відгук на запити. Слід зазначити що у реалізатори з чергами можна використати у якості

сховища БД або стандартний кеш, що надає варіант не встановлювати додатковий пакет і розширення на PHP для роботи з Redis. Завдяки цьому підходу кожна з реалізацій підтримує єдину сигнатуру методу, асинхронність і масштабованість, що дозволяє легко інтегрувати MC у різні сценарії використання. Дерево файлів наведено на рис. 3.1. MC розміщений у публічному репозиторії [43].

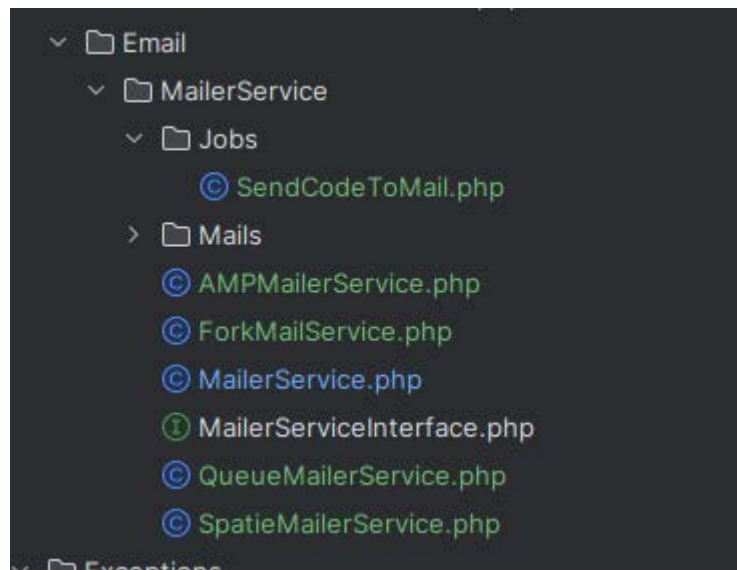


Рисунок 3.1 – Дерево файлів MC EmailTemplateMicroservice

3.1.4 MC MasterMovieStorage

MC MasterMovieStorage розроблено для завантаження медіафайлів до декількох сховищ, а саме локального і Firebase. Для цього створено три API контракти та один контролер MediaController для їх реалізації. Метод завантаження спочатку перевіряє файл і наявність користувача у системі, перевіряє його ключ токен доступу, а також його роль та дозволи на завантаження або редагування контенту. Після перевірки викликається метод upload сервісу MediaService, який за допомогою фабрики MediaRestrictionFactory визначає, який клас валідатор необхідно використовувати для перевірки файлу, а також який клас реалізатор

застосувати щоб завантажити файл, такі класи реалізатори мають спільний інтерфейс, але відповідають за збереження даних у різні сховища.

Для файлів, що успішно пройшли перевірку, створюється унікальна назва та шлях до папки у сховищі, які залежать від випадкових бітів, поточної дати та типу файлу. Сервіс викликає підібраний MediaStorage для збереження файлу, у випадку з реалізацією збереження до сховища Firebase забезпечується за допомогою фасаду Firebase, після завантаження у сховище, метод повертає посилання на завантажений медіа файл.

Усього було розроблено два класи валідаторів: `MovieImageRestriction` та `MovieVideoRestriction`. Реалізовані вони за допомогою вбудованого класу валідатору `Validator` платформи, у них виконуються базові перевірки на розмір, роздільну здатність та `mime` тип зображення. Додатково було розроблено правило для перевірки саме внутрішнього типу файлу, а не лише того, що вказано у назві, так як базовий клас валідатор перевіряє саме `mime` тип за назвою, все це підвищує безпеку завантажених файлів. MC розміщений у публічному репозиторії [44].

3.1.5 PCO BestMovieCommon

Розроблений пакет `PCO BestMovieCommon`, створений на мові PHP без використання платформи `Laravel` або інших фреймворків, спеціальних технологій і бібліотек. Такий пакет розширяє функціональність `Domain` рівня для інтеграції MC у єдину систему взаємодії. Головна ціль є забезпечення простого та ефективного способу обміну даними між MC, що досягається завдяки універсальності та низькому рівню залежностей. Одним з головних критеріїв розробки такого пакету є можливості гнучкого розширення.

Для керування залежностями використовується менеджер `Composer`, що дозволяє легко інтегрувати пакет в інші платформи та забезпечувати автоматичне створення простору імен. У складі обов'язкових залежностей

лише кілька базових пакетів: “php:^8.3”, що визначає мінімальні технічні вимоги, та бібліотека “guzzlehttp/guzzle”, яка забезпечує виконання HTTP запитів. Остання розроблена на основі стандартних вбудованих функцій PHP і не має прихованих залежностей від сторонніх бібліотек, ця бібліотека за допомогою функцій “curl” надає можливість легко посилати запити на сервери.

У файлі `composer.json` передбачено конфігурацію для фреймворків, а саме для Laravel, яка включає класи сервіс провайдери, що є нащадками класу `ServiceProvider`. Такий підхід дає змогу інтегрувати пакет з шаблоном програмування для ін’єкції залежностей, що реалізовані у застосованому фреймворку в інших MC. Для цього залежності розробника включають `laravel/framework`, а `ServiceProvider`, як базовий клас Laravel, дозволяє уникнути його додавання до обов’язкових залежностей пакету, таким чином ядро фреймворку Laravel і так буде мати клас провайдеру, а якщо пакет буде встановлений у інший фреймворк, то залежності не буде. Пакет також забезпечує підтримку тестування через PHPUnit [45]. Додаткові бібліотеки для тестування, такі як “mockery/mockery” та “fakerphp/faker”, додають можливість моделювання запитів, відповідей і створення різноманітних тестових даних. Це дозволяє перевіряти як базову функціональність, так і складні сценарії, зокрема виявлення типових помилок або невідповідностей у логіці. У рамках пакету реалізовано підтримку API контрактів для MC `emplateMicroservice`, `MasterMovieMicroservice` і `MasterMovieStorage`. Для цього створено базовий клас `BaseApi`, що відповідає за взаємодію з MC через HTTP протокол. Цей клас побудований на основі клієнта `Client` з пакету “guzzlehttp/guzzle” і має такі ключові функціональні особливості:

- фасад конфігурації, що забезпечує доступ до змінних середовища, наприклад через ключі `MS_ENV_KEY` і `MS_AUTH_ENV_KEY`, що дає змогу автоматично додавати заголовки авторизації до запитів;
- приватне поле “`$baseUrlConfigKey`”, яке визначає ключ змінної для отримання базової URL адреси MC. Це поле разом із механізмами

наслідування дає змогу адаптувати логіку взаємодії для кожного конкретного МС.

Для підвищення продуктивності взаємодії між МС було додано підтримку асинхронних викликів. Базовий клас `BaseApi` розширено методами, що дублюють стандартні методи, але працюють асинхронно або автоматично визначають деталь як саме МС до якого звертаються повинен обробити процес. У таких методах реалізовано магічний метод `__call`, який визначає необхідність виконання операції асинхронно. Було створено п'ять сервісів реалізаторів, кожен з яких відповідає за конкретний сценарій асинхронності:

- нативний (без асинхронності);
- сервіс із використанням `pcntl_fork`;
- сервіс на основі `SpatieAsync`;
- сервіс із підтримкою `AMPHP`;
- сервіс із чергами (`Redis`, `Horizon`);

Для зручного вибору сценарію використовується перелік `ProcessHandleType`, закладений у `PCO`, що дозволить визначити у пакеті перелік доступних операцій в залежності від специфіки МС і методу який визивається. Перемикання реалізаторів здійснюється через клас `middleware DefineProcessServices`, яка витягує параметри із запиту і за допомогою контейнеру `Laravel` ініціалізує “`Singelton`” патерном потрібний клас. У рамках роботи з колекціями додано асинхронні методи мапінгу, які перевантажують стандартні методи і дозволяють виконувати операції мапінгу сутностей а бо моделей без послідовних затримок.

Спілкування через `API` контракт із `EmailTemplateMicroservice` реалізовано у класі `EmailTemplateApi`, який забезпечує функціональність для роботи з кодами підтвердження електронної пошти. Клас має два основних методи, перший “`createEmailVerificationCode`”, що генерує коди підтвердження, та другий “`getEmailVerificationCodes`”, який повертає наявні коди за вказаним адресом електронної пошти користувача, що збережена у

кеші. Для кожного з цих методів створено окремі класи, що наслідують базовий ресурс `BaseResource`.

Базовий клас `BaseResource` забезпечує загальні методи для роботи з мапінгом отриманих ресурсів від запитів на API у форматі JSON, включаючи декодування, перевірку структури даних та обробку помилок. Нащадки цього класу мають чітке уявлення про структуру відповіді сервісу, включаючи ключі, типи змінних та специфічні методи для отримання конкретних даних з відповіді. Наприклад, у класах, що реалізують роботу з `EmailTemplateApi`, такі методи, як `“getVerificationCodeById”` і `“listAllVerificationCodes”` дозволяють витягувати точну інформацію для подальшої обробки. Аналогічна архітектура застосована і для `MasterMovieMicroservice` та `MasterMovieStorage`. Наприклад:

- `MasterMovieStorageApi` реалізує методи для отримання повного шляху до файлу через часткове посилання `“getFilePath”` та перевірки наявності файлу за частковим посиланням `“checkFileExistence”`;
- `MasterMovieApi` реалізує API контракт, що забезпечує отримання інформації про користувача за допомогою методу `“getUserInfo”`

Для автоматичного підключення та ін’єкції розроблених сервісів створено сервіс провайдери на основі базового провайдеру (клас `ServiceProvider`) фреймворку. Ці провайдери підтримують автоматичну ін’єкцію реалізацій класів API, однак мають особливість: під час ін’єкції вимагається передати ключ змінної середовища, за яким відбувається пошук необхідних налаштувань у конфігураційних файлах або у файлі `“.env”`. Для класів `MasterMovieApi`, `MasterMovieStorageApi` та `EmailTemplateApi` було визначено наступні ключі:

- `BEST_MOVIE_MS` для `Best MovieMicroservice`;
- `BEST_MOVIE_STORAGE_MS` для `MasterMovieStorage`;
- `EMAIL_TEMPLATE_MS` для `EmailTemplateMicroservice`.

Ці ключі відповідають унікальним ідентифікаторам кожного MS, забезпечуючи правильну маршрутизацію запитів. При спробі використання

класу або сервісу без визначення відповідного ключа виникає ризик отримання критичної помилки. Сервіси провайдери автоматично створюють мапу залежностей, але вимагають налаштувань і можливо перевантаження для уникнення непередбачуваних ситуацій. Схема взаємодії МС продемонстрована на рис 3.2

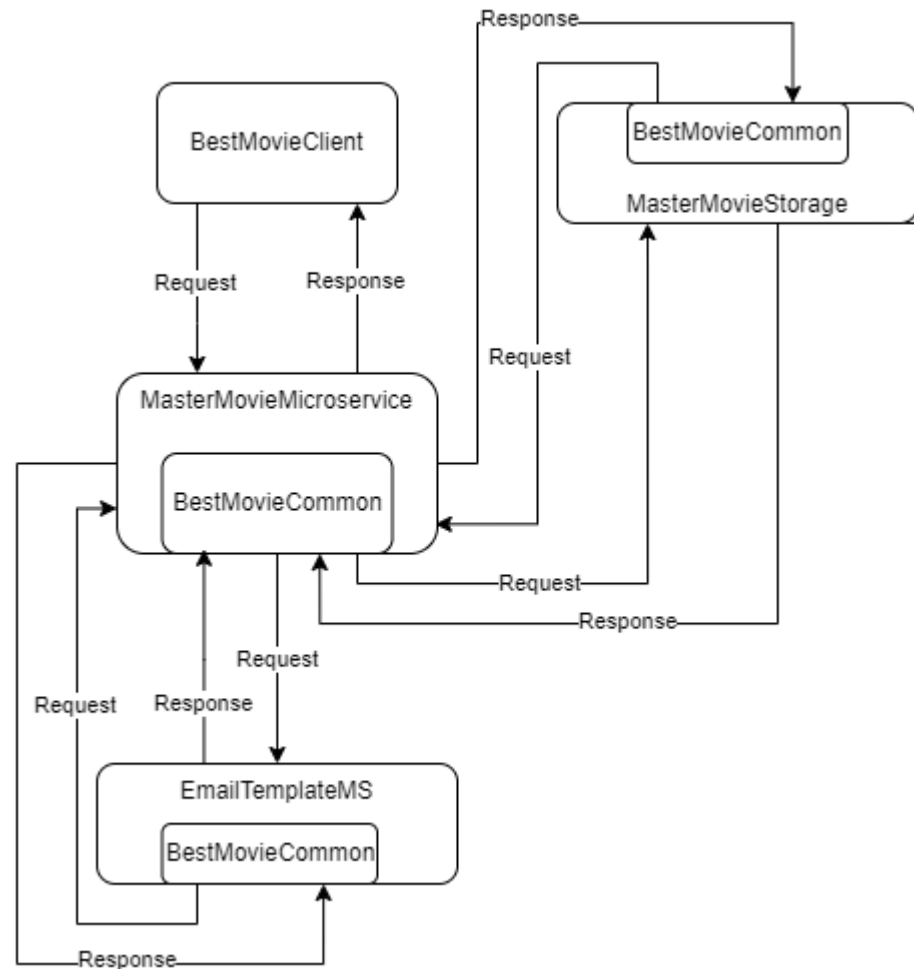


Рисунок 3.2 – Схема взаємодії МС

Для управління версіями пакету використовується Packagist. На момент написання документації існує 26 версії пакету, з яких актуальною є “v0.2.2-beta”. Контроль версій забезпечує відповідність стандартам безпеки та стабільності, перевіряючи код через репозиторій. PCO BestMovieCommon було інтегровано усі три МС. Структура PCO зазначена на рис. 3.3. PCO розміщений у публічному репозиторії [46] та на Packagist [47].

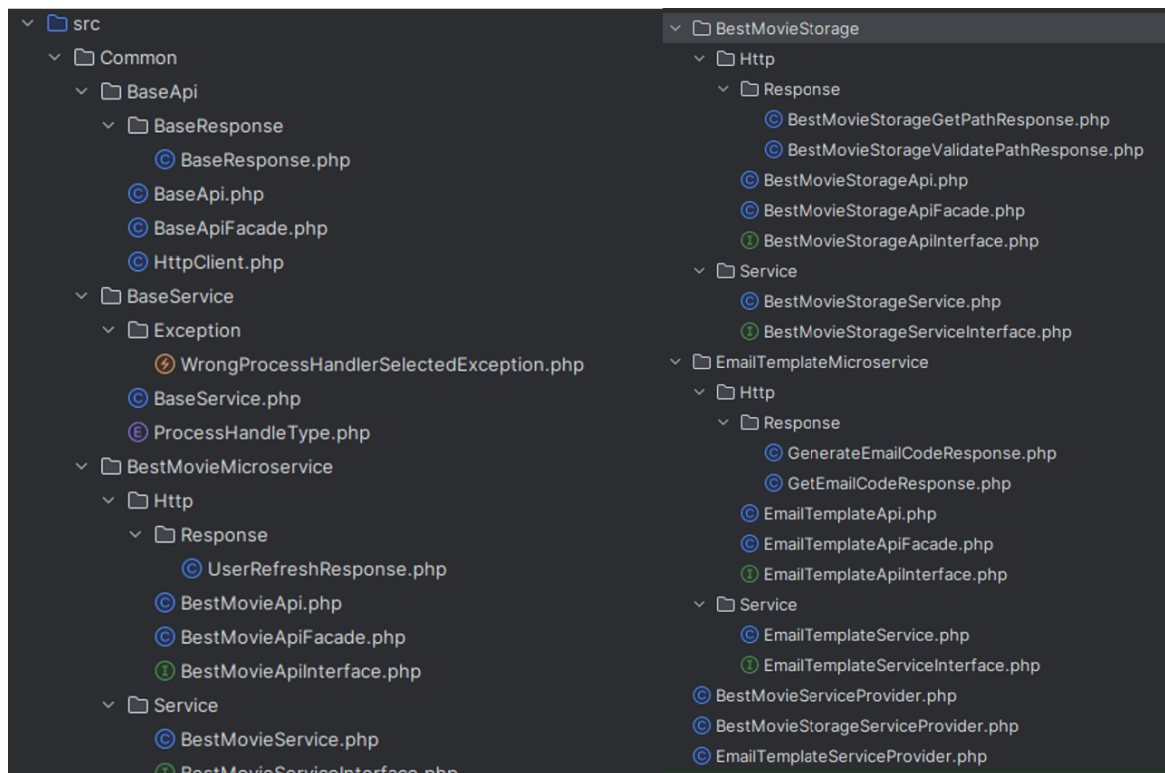


Рисунок 3.3 – Дерево файлів MC BestMovieCommon

3.2 Основні рішення щодо розробки інтерфейсу

Головна сторінка користувача, що представлена на рис. 3.3, демонструє інтерфейс із темним дизайном, на якому користувач може фільтрувати фільми за категоріями та переглядати їх у вигляді постерів. Ліве бічне меню дозволяє вибрати категорії, такі як “Екшн”, “Комедія”, “Драма”, що робить навігацію швидкою та зручною. Кнопки для додавання фільмів або категорій мають зелений колір, а для видалення — червоний. Всі функції сайту доступні для перегляду, але без можливості редагування чи додавання нових елементів, оскільки це обмежено правами користувача.

Сторінка адміністратора, що представлена на рис. 3.4, має додаткові функції для управління контентом. Адміністратор може додавати нові фільми, змінювати категорії та колекції, а також встановлювати зв'язки між фільмами та іншими об'єктами. Для цих операцій використовуються модальні вікна, що дозволяють зручно додавати нові елементи без перенавантаження інтерфейсу. Сторінка адміністратора містить більше

інтерфейсних елементів для управління контентом, що є відмінністю від користувацької версії сайту. Відмінність між сторінками адміністратора та користувача полягає в наявності розширених функцій для управління контентом на сторінці адміністратора, тоді як користувач має доступ лише до перегляду та фільтрації фільмів.

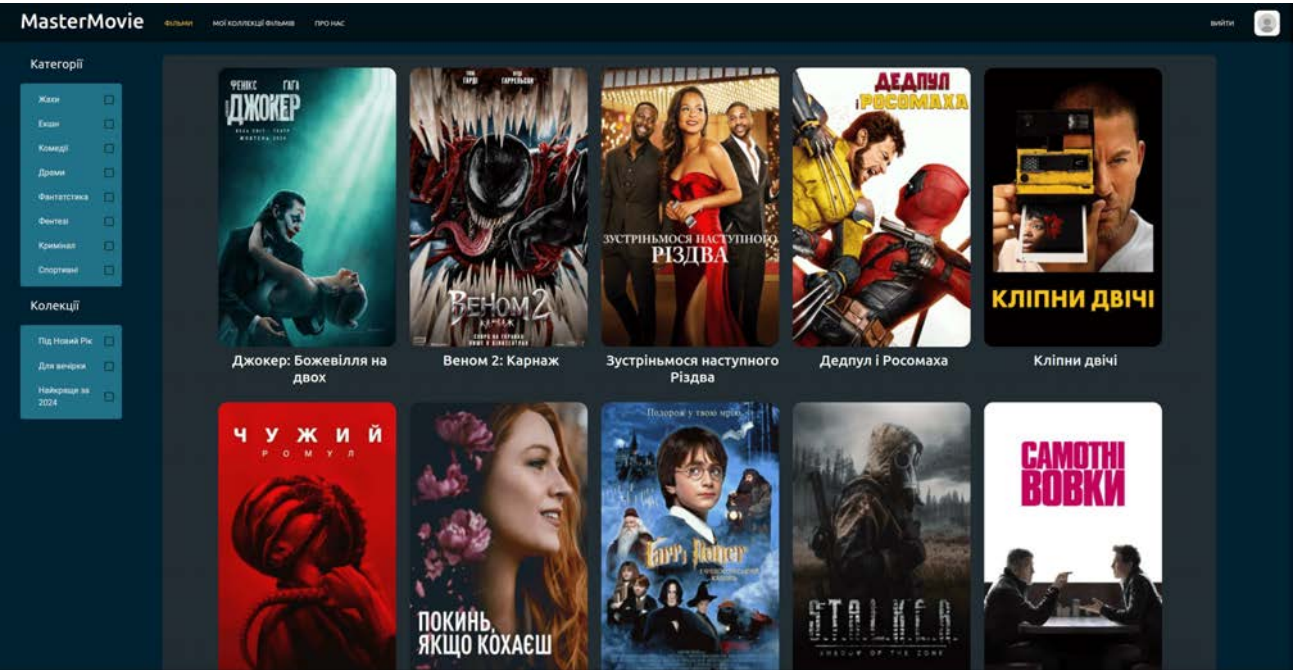


Рисунок 3.3 – Головний екран звичайного користувача

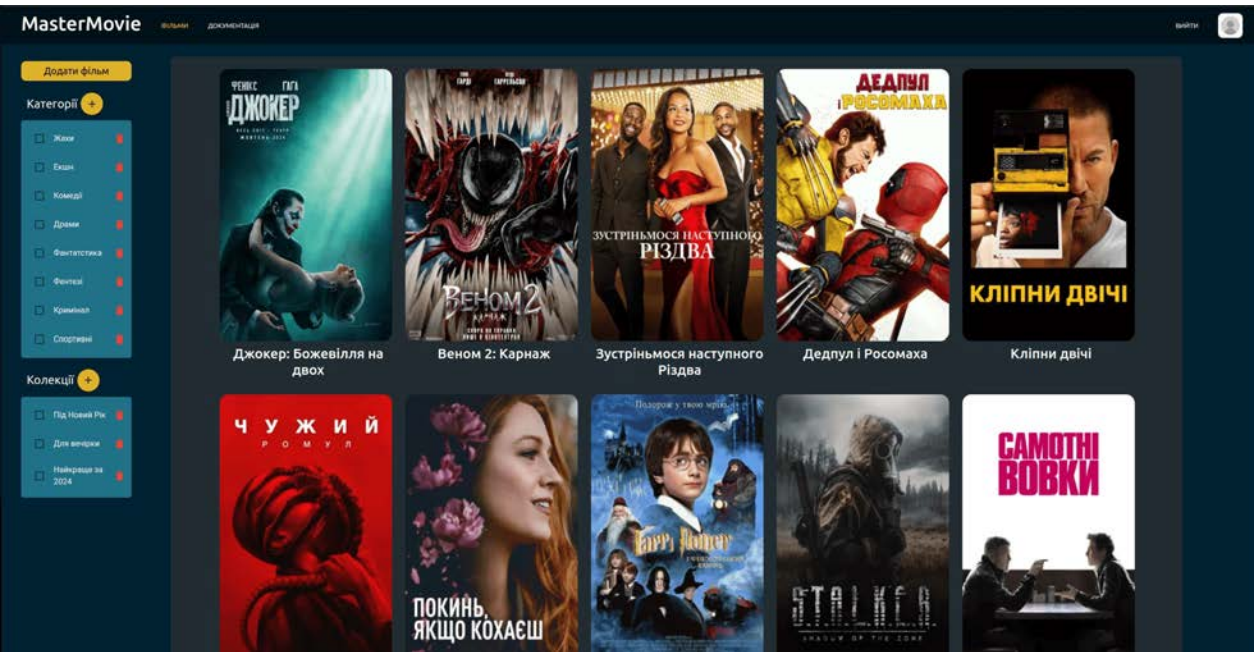


Рисунок 3.4 – Головний екран адміністратора

3.2.1 Основні рішення щодо впорядкування інтерфейсу

Сторінка сайту розбита на три компоненти, а саме: шапка, тіло, і футер. У шапці сайту розташовано три вкладки зліва: “Фільми”, “Мої колекції” та “Про нас”. Справа від них є кнопки “Зареєструватися” або “Увійти”, вона динамічна і присутня в залежності від статусу користувача.

На вкладці “Фільми” користувач може бачити колекції, категорії і фільми, що відображаються горизонтально у кількох рядах, кількість яких залежить від розміру екрану. Загалом на екрані який був використаний для тестування одночасно можна побачити до 10 фільмів, як це продемонстровано на рисунку 3.4. Кожен компонент, що виводить на інтерфейс фільм, представлений афішею та назвою під нею. Фільтри для категорій та колекцій знаходяться зліва й не є взаємовиключними. У фільтраційних списках можна побачити від 10 до 15 категорій і колекцій, кількість залежить від кількості записів у БД, якщо кількість більша за 15, то буде застосовуватися компонент горизонтальну прокрутки. Для поліпшення прийняття, списки відокремлені кольоровими блоками з назвами, що виконують функцію заголовків. Після того як користувач обирає потрібні фільтри, змінюється посторінкова навігація компоненту фільмів. Такі навігація по сторінках з фільмами розташована внизу і може містити до 10 номерів сторінок, причому для перемикання можна використовувати як стрілки, так і номери сторінок.

При натисканні на афішу фільму, користувач потрапляє на сторінку з детальною інформацією про фільм, що наведено на рисунку 3.5. На цій сторінці відображаються афіша, інформація про кіно в таблиці праворуч від неї та відео, яке знаходиться під інформаційною таблицею. У правому верхньому куті розташована кнопка “Додати до колекції”. При натисканні на неї з'являється модальне вікно, як це показано на рисунку 3.7. Це вікно розташоване праворуч і містить список колекцій користувача, у якому вже

обрані старі колекції. Нижче вікна є дві кнопки: “Зберегти зміни” та “Закрити”. Вибір кількох колекцій можливий одночасно.

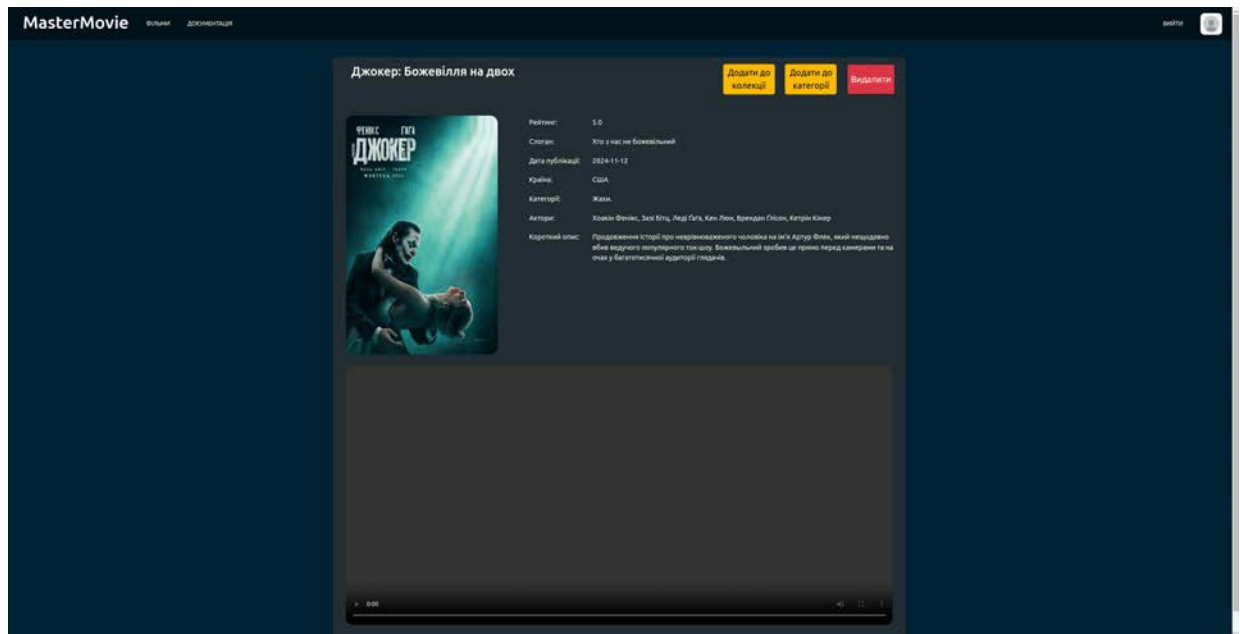


Рисунок 3.5 – Сторінка з фільмом

Сторінка “Про нас” має опис, що це дипломний проєкт, розроблений виключно у дослідницьких цілях, сторінку наведено на рис. 3.6.

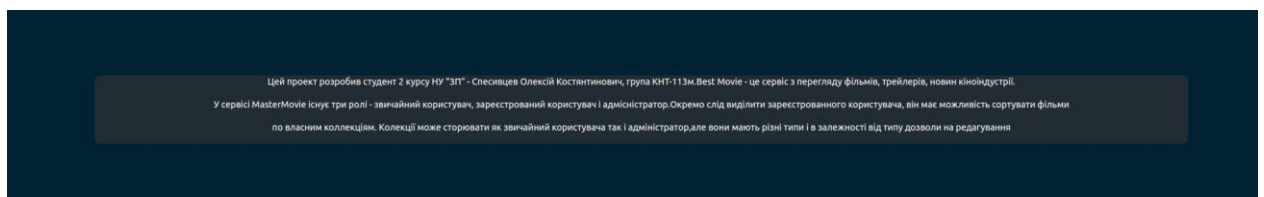


Рисунок 3.6 – Сторінка “Про нас”

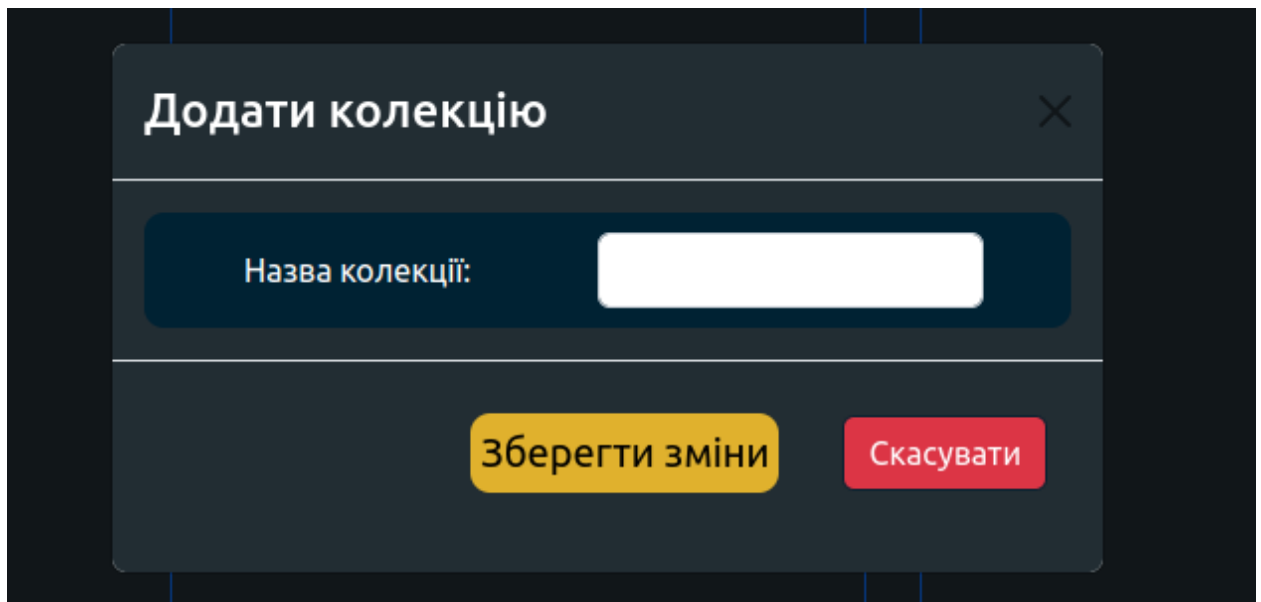
A dark-themed modal window titled "Додати колекцію" (Add collection) with a close button (X) in the top right corner. Below the title bar, there is a label "Назва колекції:" (Collection name:) followed by a white text input field. At the bottom of the modal, there are two buttons: a yellow button labeled "Зберегти зміни" (Save changes) and a red button labeled "Скасувати" (Cancel).

Рисунок 3.7 – Модальне вікно додавання колекції

Форми для реєстрації та авторизації представлені окремими сторінками з темним фоном, білими полями для введення даних і золотистими кнопками. На сторінці реєстрації, наведеної на рис. 3.8, розташовані поля для введення імені, прізвища, адреси електронної пошти, пароля та підтвердження пароля. У нижній частині форми знаходяться дві кнопки: “Зареєструватися”, яка є основною дією, та “Увійти”, яка дозволяє перейти на сторінку авторизації. Після натискання кнопки “Зареєструватися” користувачу надсилається електронний лист із кодом підтвердження.

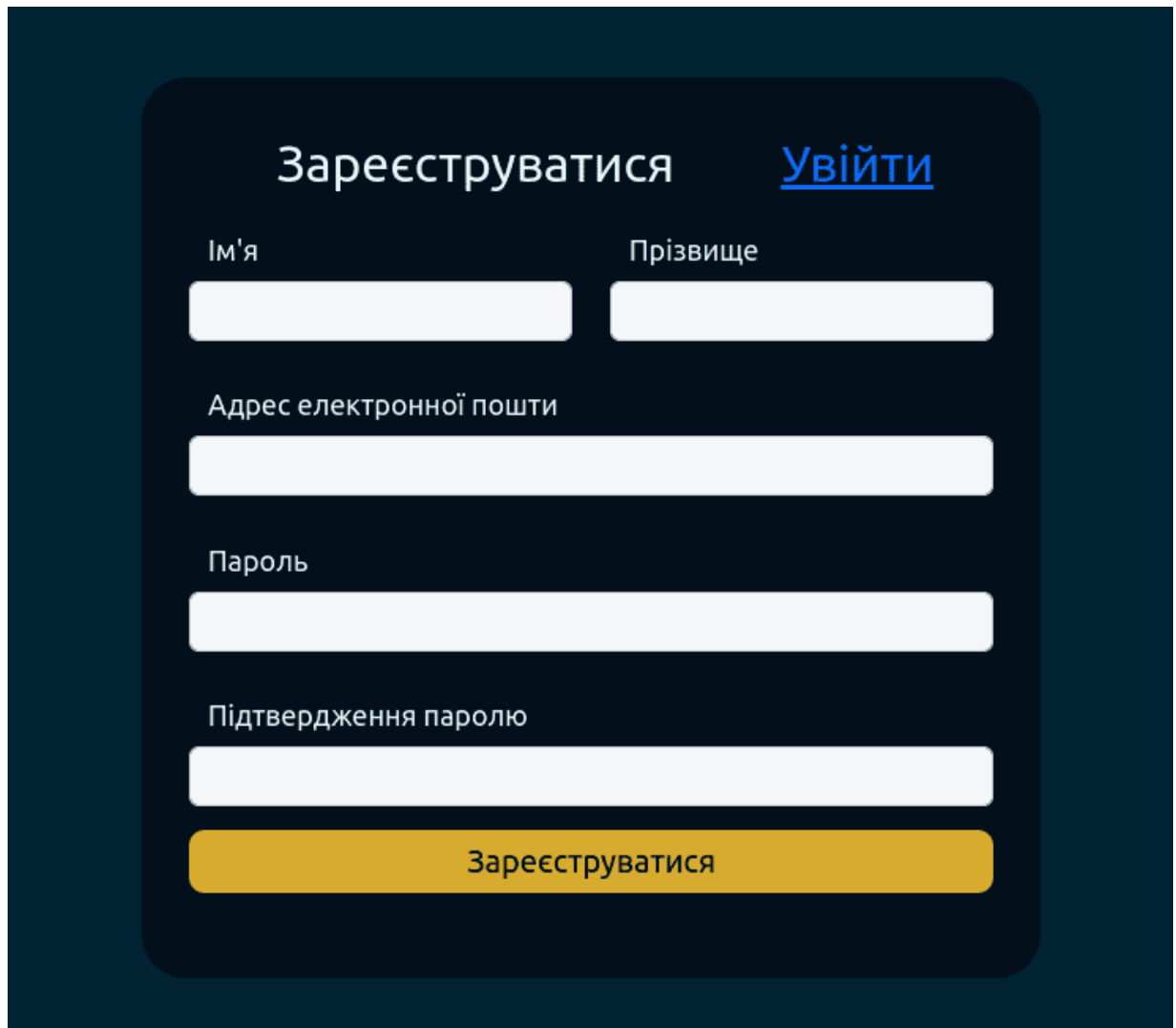
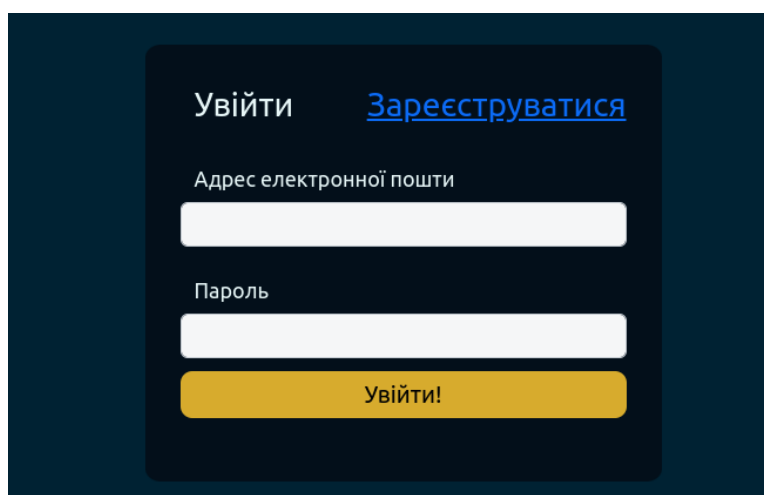
The image shows a registration form on a dark blue background. At the top, there are two links: "Зареєструватися" (Register) and "Увійти" (Login). The "Увійти" link is underlined and blue. Below the links are four input fields: "Ім'я" (Name), "Прізвище" (Surname), "Адрес електронної пошти" (Email address), and "Пароль" (Password). Each field is a white rectangle with rounded corners. Below the "Пароль" field is another white rectangle for "Підтвердження паролю" (Password confirmation). At the bottom of the form is a yellow button with the text "Зареєструватися" (Register).

Рисунок 3.8 – Сторінка реєстрації

Після введення цього коду користувач успішно завершує процес реєстрації і зможе користуватися сторінкою авторизації, яка зображена на рис. 3.9. Сторінка авторизації доволі проста і у ній знаходяться поля для введення адреси електронної пошти та пароля. У нижній частині також розташовані кнопки “Увійти”, яка є основною дією для входу в систему, та “Зареєструватися”, яка дозволяє перейти на сторінку реєстрації. Після введення даних і натискання кнопки “Увійти” система перевіряє їх, і в разі успіху користувач перенаправляється на головну сторінку сайту.

A dark-themed login and registration form. At the top, there are two links: 'Увійти' (Login) and 'Зареєструватися' (Register). Below these are two input fields: 'Адрес електронної пошти' (Email address) and 'Пароль' (Password). At the bottom is a yellow button labeled 'Увійти!' (Login!).

Увійти [Зареєструватися](#)

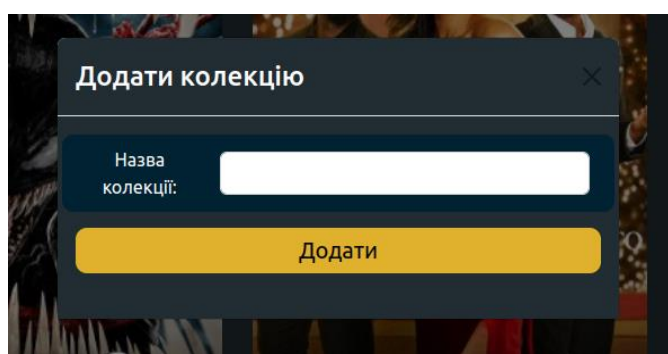
Адрес електронної пошти

Пароль

Увійти!

Рисунок 3.9 – Сторінка авторизації

Адміністративна головна сторінка, наведена на рис. 3.4, окрім основних функцій, містить додаткові можливості, такі як кнопки для додавання та видалення категорій і колекцій за замовченням. При натисканні на ці кнопки відкриваються відповідні модальні вікна, приклади яких показані на рис. 3.10 і 3.11.

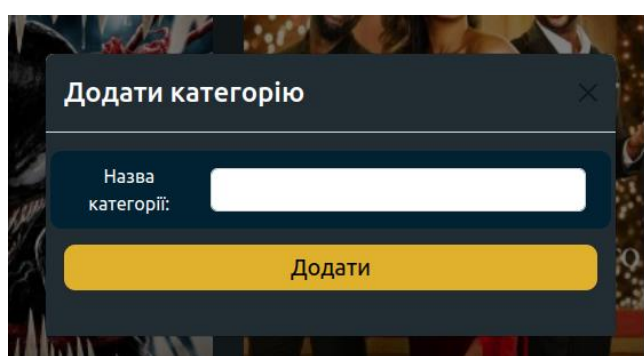
A modal window titled 'Додати колекцію' (Add collection). It has a close button (X) in the top right corner. Inside, there is a label 'Назва колекції:' (Collection name:) followed by a text input field. At the bottom is a yellow button labeled 'Додати' (Add).

Додати колекцію

Назва колекції:

Додати

Рисунок 3.10 – Модальне вікно додавання колекції

A modal window titled 'Додати категорію' (Add category). It has a close button (X) in the top right corner. Inside, there is a label 'Назва категорії:' (Category name:) followed by a text input field. At the bottom is a yellow button labeled 'Додати' (Add).

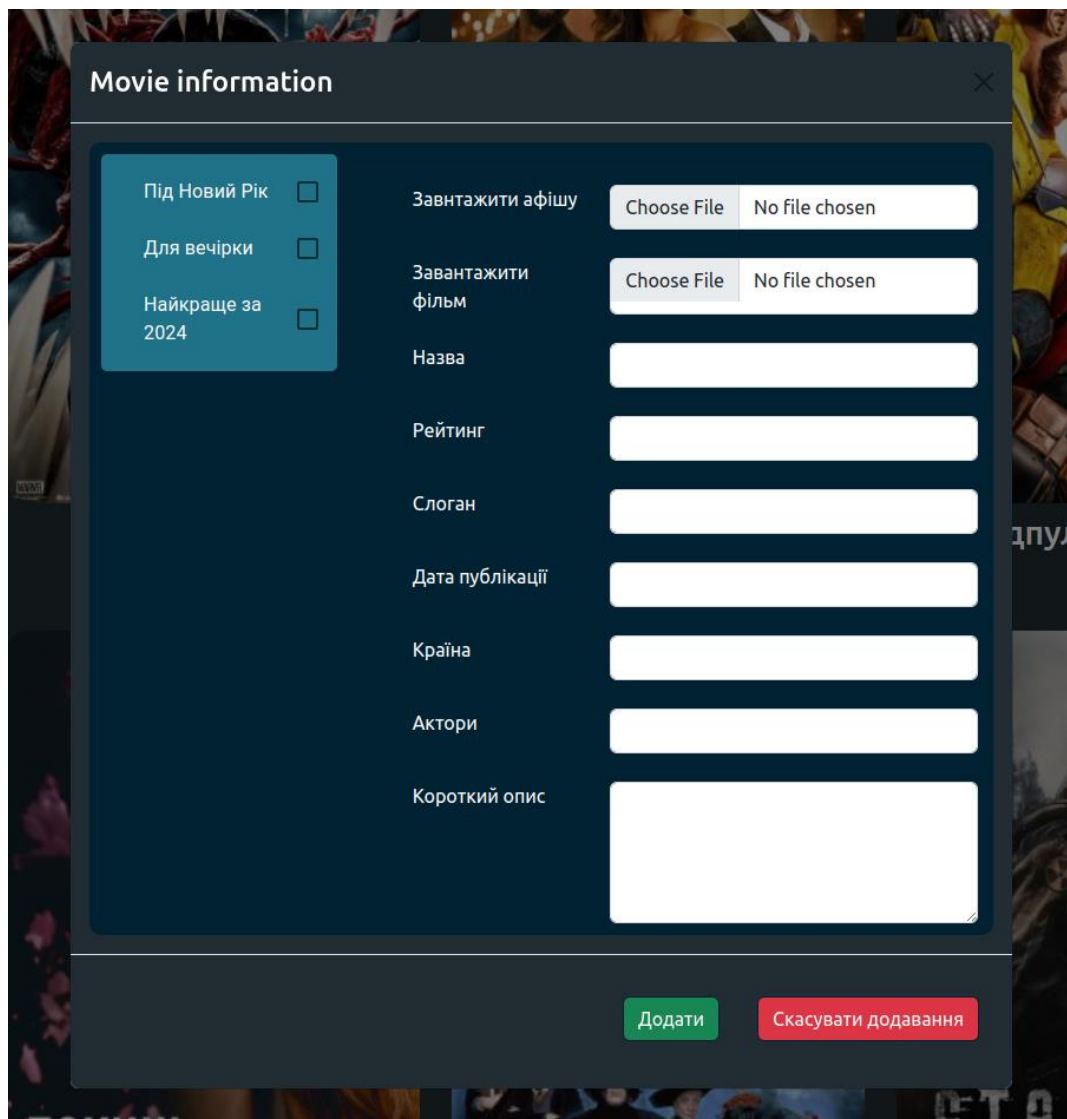
Додати категорію

Назва категорії:

Додати

Рисунок 3.11 – Модальне вікно додавання категорії

Також на сторінці є кнопка “Додати фільм”, яка активує спливаюче вікно для введення інформації. У цьому вікні можна вказати такі дані, як афіша, назва фільму, рейтинг, країна, слоган, актори, опис, категорії. У нижній частині модального вікна розташовані дві кнопки: “Додати” для збереження фільму та “Видалити” для скасування дій. Зовнішній вигляд модального вікна додавання фільму зображено на рис. 3.12.



Movie information

Під Новий Рік ☐

Для вечірки ☐

Найкраще за 2024 ☐

Завантажити афішу Choose File No file chosen

Завантажити фільм Choose File No file chosen

Назва

Рейтинг

Слоган

Дата публікації

Країна

Актори

Короткий опис

Додати Скасувати додавання

Рисунок 3.12 – Модальне вікно додавання фільму

На сторінці з конкретним фільмом розташовані кнопки для керування фільмом: “Видалити”, “Додати до колекції”, представлена на рис. 3.13, та

“Додати до категорії”, представлена на рис. 3.14. Натискання на кнопки “Додати до колекції” або “Додати до категорії” відкриває форму, яка схожа на форму додавання колекцій для звичайного користувача, але містить лише колекції за замовченням. Крім можливості додавання, доступна функція видалення колекцій і категорій. Для цього використовується модальне вікно підтвердження. У цьому вікні виводиться попередження про наслідки дії, список вибраних елементів для видалення, а також кнопки “Підтвердити” і “Скасувати”, що дозволяють користувачеві прийняти або відмовитися від видалення.

Для категорій доступна можливість додавання зображення, яке буде асоціюватися з категорією, а також зазначення пов’язаного жанру чи тематики. Це забезпечує зручну структуру для організації фільмів і полегшує їх пошук.

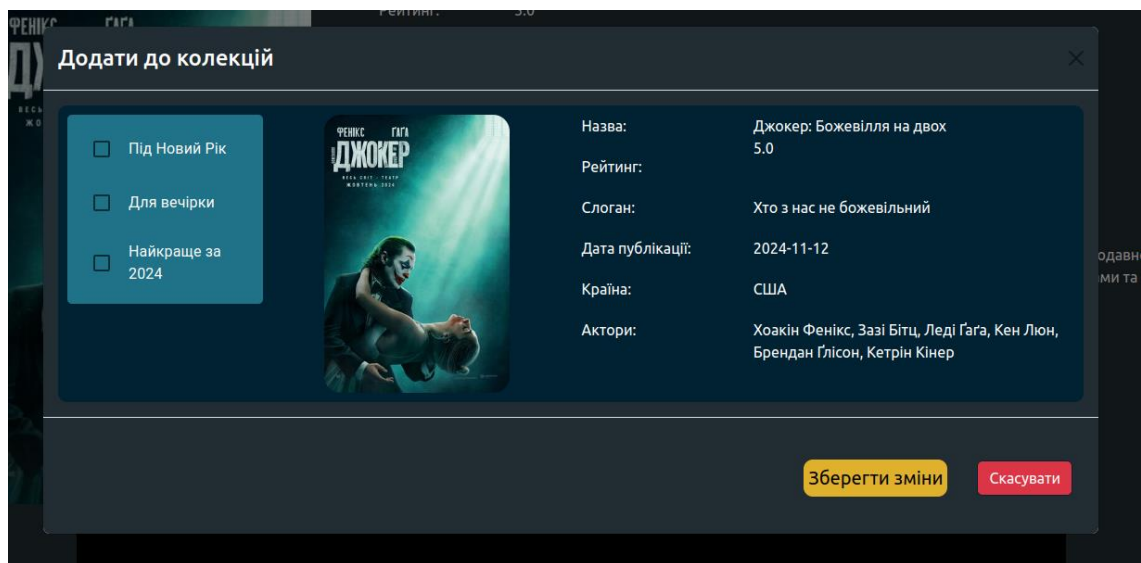


Рисунок 3.13 – Модальне вікно для додавання фільму у колекції



Рисунок 3.14 – Модальне вікно для додавання фільму у категорії

3.2.2 Основні рішення з комунікації з користувачем

Кожна афіша оснащена анімацією збільшення, яка активується при наведенні курсору. Вкладки у шапці сайту мають особливий візуальний ефект: при наведенні текст піднімається вгору, підкреслюється лінією знизу та змінює колір на блакитний. Якщо користувач перебуває на вкладці “Фільми”, ця вкладка виділяється золотим кольором. Аналогічна логіка застосовується до інших вкладок, як це показано на рис. 3.5.

Усі кнопки на сайті мають анімацію, яка створює ефект освітлення та збільшення при наведенні, забезпечуючи приємний користувацький досвід. Для кнопок видалення передбачене модальне вікно підтвердження дії, яке дозволяє уникнути випадкового видалення контенту.

Однак загальне сприйняття інтерфейсу дещо ускладнюється через насиченість кольорів, що може відволікати увагу користувача та знижувати концентрацію на основному контенті. Особливо сильно на це впливають яскраві афіші фільмів, які привертають забагато уваги, створюючи дисбаланс візуального сприйняття.

3.3 Основні рішення щодо розробки БД

Тільки у MovieMicroservice використовується повноцінна БД, тоді як інші МС застосовують тимчасові сховища. Зокрема, у EmailTemplateMicroservice використовується Redis, що забезпечує швидкий доступ до даних, а в MasterMovieClient застосовується комбінація локального та віддаленого сховищ для зберігання файлів. Такий підхід дозволяє ефективно розподілити ресурси, зменшити навантаження на основну базу даних та забезпечити швидкодію під час роботи з даними.

У ролі основної системи керування базами даних (СКБД) було обрано MySQL 8.0.32. Вибір MySQL обумовлений її стабільністю, широкими функціональними можливостями та зручністю інтеграції з фреймворком Laravel. Для налаштування структури БД було розроблено низку міграцій, перелік наведено на рис. 3.15, зміст яких базується на використанні готових інструментів Laravel, таких як класи Migration, Schema та Blueprint.

Міграції створюють структуру таблиць із полями числового та строкового типів, індексами, первинними, зовнішніми та складовими ключами. Крім того, кожна міграція має механізм зниження до попереднього стану (rollback), що забезпечує гнучкість і дозволяє легко відновлювати БД разі помилок або змін у проєкті, хоча слід зазначити, що дамп БД все ж таки є необхідним, незважаючи на переваги міграції.

Такий підхід до організації сховищ даних поєднує в собі надійність централізованої бази даних із швидкістю роботи Redis і зручністю локальних сховищ, що створює оптимальний баланс між продуктивністю та масштабованістю.

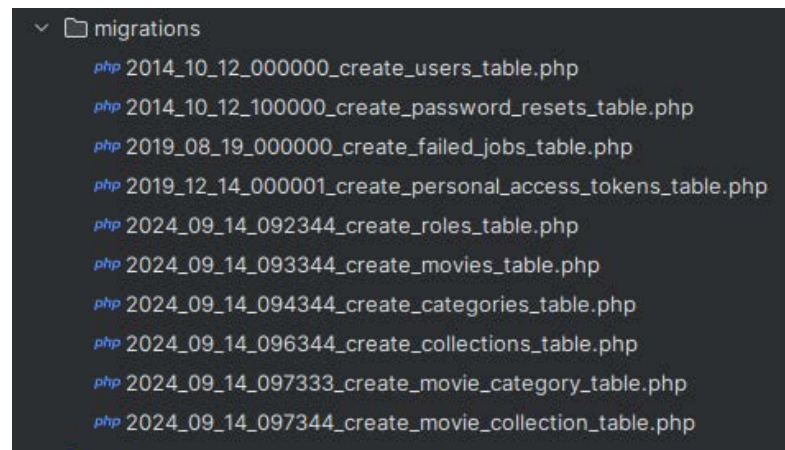


Рисунок 3.15 – Перелік розроблених міграцій

Додаткові переваги міграції:

- завдяки наявним міграціям можна без зусиль відслідковувати поточний стан БД та ефективно контролювати її структуру.
- створювати інтеграційні тести, які використовують тимчасову БД для тестування, керовану через СКБД SQLite. Однак варто зазначити, що SQLite має обмеження порівняно з MySQL: він не підтримує деякі можливості, такі як пошук по JSON полях, складні нативні SQL запити у міграціях, регулярні вирази, роботу з великими датами або зберігання великих файлів, наприклад, якщо БД містить файли, більші за 32 МБ.

Файли міграцій можна створювати вручну, однак цей підхід не рекомендується, оскільки за допомогою команди “php artisan make:migration -name” платформа автоматично додає до назви міграції точний час її створення, що дозволяє уникнути дублювання імен і гарантує унікальність. Цей процес також автоматично перевіряє правильність імені міграції та забезпечує її унікальність. Такий підхід забезпечує порядок в організації міграцій, оскільки вони впорядковані за датою створення, що полегшує перегляд і підтримку проєкту.

Слід зазначити, що створення міграцій вручну може призвести до проблем, таких як неможливість розгорнути проєкт, помилки при ревізії пакетів, проблеми з установкою та оновленням пакетів, а також із створенням кешу файлів і виконанням тестів. Тому рекомендується

використовувати автоматизовані інструменти для створення міграцій, щоб уникнути цих труднощів і забезпечити стабільність роботи проєкту.

3.3.1 Розроблена БД

Розроблена БД складається з одинадцяти таблиць і має два основних типи зв'язків між таблицями: один до багатьох і багато до багатьох. Для внутрішніх завдань платформа автоматично створює дві таблиці: “failed_jobs” і “migrations”. Перша використовується для роботи з чергами, а друга – для контролю за виконанням міграцій. Міграції реєструються шляхом додавання відповідних записів під час нової міграції до таблиці.

Основною сутністю в системі є користувач, для якого організована таблиця користувачів. Він містить такі поля, як унікальний ідентифікатор користувача, ім'я, прізвище, електронна пошта, час перевірки електронної пошти, хеш пароля, маркер для дій резервного копіювання, а також дати створення й оновлення. Таблиця “users” має прямий зв'язок із таблицями “user_roles” і “collections” за допомогою зв'язку «один до багатьох», а також непрямий зв'язок із таблицями “personal_access_token” і “password_resets”.

Таблиця “user_roles” є суб'єктом для таблиці “users” і записує всі ролі користувачів. Вона містить ідентифікатор користувача, роль та інформацію про час створення та оновлення. Таблиці “personal_access_token” і “password_resets” є суб'єктами для користувачів через змінені зв'язки. Вони підключаються до користувача через персональні маркери доступу та таблицю старих паролів. Ці таблиці мають непрямий зв'язок «один до багатьох» для маркерів доступу та старих паролів.

Таблиця “personal_access_token” містить такі поля, як ідентифікатор маркера, тип маркера, доступні можливості, дата останнього використання, термін дії, а також дати створення й оновлення. Таблиця “password_resets” містить адресу електронної пошти, маркер і дату останнього оновлення. Унікальним ідентифікатором тут є адреса електронної пошти. Варто

зазначити, що ці таблиці мають таку структуру для належної роботи бібліотеки “Sanctum”, яка була додана на платформу.

Таблиця “collections” є окремою сутністю, хоча вона не може існувати без прив’язки до користувача, але відповідно до бізнес-умов вона може бути колекцією за замовчуванням, доступною для кількох користувачів. Поля цієї таблиці включають унікальний ідентифікатор, ідентифікатор користувача, тип колекції, назву, а також час створення й оновлення.

Іншою важливою сутністю є таблиця “movies”, яка пов’язана з таблицями категорій і колекцій через проміжні таблиці “movie_category” і “movie_collection”. Категорії – це підсутності для фільмів. Таблиця фільмів містить такі дані, як унікальний ідентифікатор фільму, назва, опис, посилання на файл фільму, зображення фільму, а також час створення й оновлення. У таблиці “movie_collection” зберігаються ідентифікатор фільму, ідентифікатор колекції, а також дати створення й оновлення. Таблиця “movie_category” містить ідентифікатори фільму та категорії, а також дати створення та оновлення. Категорії мають власні атрибути: унікальний ідентифікатор, назву, а також час створення й оновлення. Схему бази даних, яка показує всі зв’язки між таблицями, можна побачити на рис. 3.18.

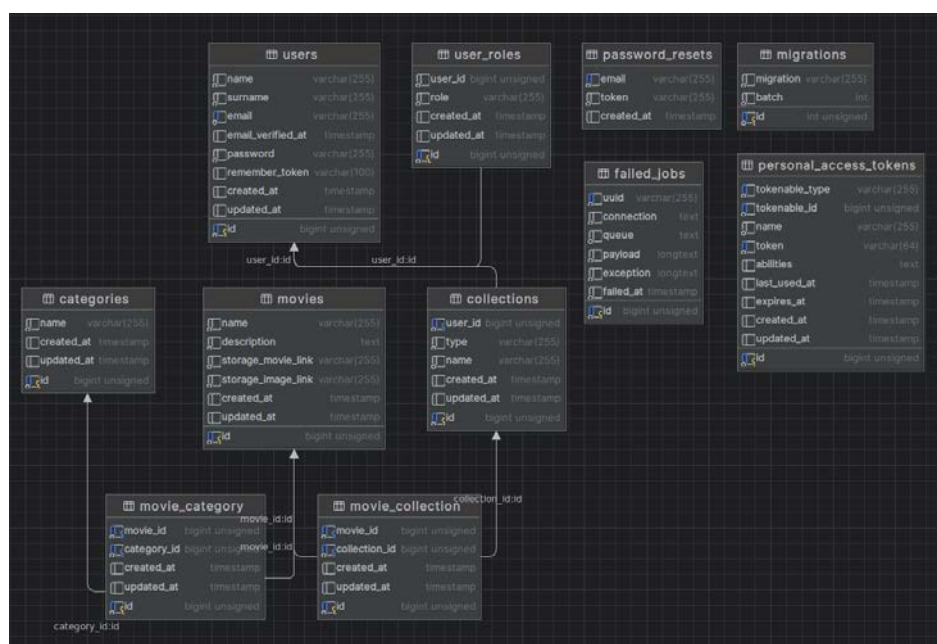


Рисунок 3.18 – Схема таблиць БД і зв’язків між ними

3.4 Висновки з розділу

У цьому розділі були детально розглянуті основні технічні рішення, включаючи вибір застосованих шаблонів програмування, розробку МС, структуру бази даних БД, сторінки інтерфейсу користувача, а також створені алгоритми. Під час вибору підходів і шаблонів програмування було враховано не лише цілі проєкту та його масштаб, а й специфіку мети дослідження, що дозволило підібрати найбільш ефективні методи реалізації. Важливим аспектом стало інтегрування та забезпечення взаємодії з іншими технологіями, що використовуються в системі.

Особлива увага була приділена забезпеченню сумісності між розробленими компонентами і сторонніми системами, такими як сервіси зберігання даних, системи відправки електронних листів та інші зовнішні сервіси. Такий підхід дозволив створити єдину, цілісну систему, яка ефективно взаємодіє з усіма елементами й має високий рівень функціональності.

Під час розробки системи також була здійснена значна увага на оптимізацію роботи окремих елементів і їх взаємодії, що дозволило досягти максимального ефекту від функціонування системи в цілому.

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРЕМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

У даному розділі розглянутий технологій що дозволяють розгорнути мереджу мікрсервісів і основні умови їх експлуатації, а також проведене тестування і дослідження швидкодій запитів, навантаження на віртуальну і фізичну машину, оброку процесів, зручності інтерфейсу, сумісність MS з розробленими PCO.

4.1 Застосування системи

Система призначена для перегляду, показу фільмів, афіш і керування публічними колекціями. Основна ідея полягає в тому, що система може використовуватися як звичайними корситувачами так і юридичними особами, такими як кінотеатри, для створення власних колекцій, а потім ділитися ними зі своїми клієнтами, що мають преміальними пакетами або спеціальні ролі. Щоб поширити систему на мобільні облікові записи або настільні облікові записи, достатньо створити нову MS на рівні «Інтерфейс користувача».

4.2 Розгортання і експлуатація системи мікросервісів

Система була розроблена і розгорнута у ОС (операційна система) Linux Mint 21 Cinnamon завдяки технологій Docker, створені докер файли теоретично можна застосувати будь якій Linux подібній системі, але однозначно будуть додаткові кроки у налаштуванні, так як усі ОС мають незначні особливості. Для розгортання трьох MS, а саме – MovieMicroservice, EmailTemplateMicroservice, MasterMovieStorage, потрібне звантажити проєкти з репозитарію в окремий провідник і створити “.env” файли

перераховані у додатку Б. Загалом Docker має встановити і налаштування Nginx, PHP 8.0, PHP-FPM, MySql та Composer [48] – [49].

Після встановлення і перед запуском Docker треба створити конфігураційні файли для серверів МС, а саме створити три файли “.env” у кожному МС і вставити у них текст з додатку Б. За необхідності видати права на папку “storage/” командою “sudo chmod -R 777 storage/”. Також перед командою запуску Docker треба повністю вимкнути локальні сервіси: Nginx, Mysql, Apache2, PHP-FPM, Redis та інші докери, що займають порти 9000, 8000, 6789, 3306. Перед фінальна команда, яка розгорне сервери з МС це – “sudo docker-compose up —build”, результат виконання продемонстровано на рис. 4.1.

```
Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them
Creating mysql_movie_microservice ... done
Creating movie_storage_redis ... done
Creating fpm_movie_microservice ... done
Creating nginx_movie_microservice ... done
Attaching to movie_storage_redis, mysql_movie_microservice, fpm_movie_microservice, nginx_movie_microservice
mysql_movie_microservice | 2024-11-26 10:32:08+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 8.0.2
movie_storage_redis | 1:C 26 Nov 2024 10:32:08.131 # WARNING Memory overcommit must be enabled! Without it, a back
fix this issue add 'vm.overcommit_memory = 1' to /etc/sysctl.conf and then reboot or run the command 'sysctl vm.ov
movie_storage_redis | 1:C 26 Nov 2024 10:32:08.131 * o000o000o000o Redis is starting o000o000o000o
movie_storage_redis | 1:C 26 Nov 2024 10:32:08.131 * Redis version=7.4.1, bits=64, commit=00000000, modified=0, pi
movie_storage_redis | 1:C 26 Nov 2024 10:32:08.131 # Warning: no config file specified, using the default config.
mysql_movie_microservice | 2024-11-26 10:32:08+00:00 [Note] [Entrypoint]: Switching to dedicated user 'mysql'
mysql_movie_microservice | 2024-11-26 10:32:08+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 8.0.2
mysql_movie_microservice | 2024-11-26T10:32:08.540754Z 0 [System] [MY-010116] [Server] /usr/sbin/mysqld (mysqld 8.
movie_storage_redis | 1:M 26 Nov 2024 10:32:08.131 * monotonic clock: POSIX clock_gettime
movie_storage_redis | 1:M 26 Nov 2024 10:32:08.132 * Running mode=standalone, port=6379.
movie_storage_redis | 1:M 26 Nov 2024 10:32:08.132 * Server initialized
movie_storage_redis | 1:M 26 Nov 2024 10:32:08.132 * Loading RDB produced by version 7.4.1
movie_storage_redis | 1:M 26 Nov 2024 10:32:08.132 * RDB age 91 seconds
movie_storage_redis | 1:M 26 Nov 2024 10:32:08.132 * RDB memory usage when created 0.90 Mb
movie_storage_redis | 1:M 26 Nov 2024 10:32:08.132 * Done loading RDB, keys loaded: 0, keys expired: 0.
fpm_movie_microservice | [26-Nov-2024 10:32:08] NOTICE: fpm is running, pid 1
mysql_movie_microservice | 2024-11-26T10:32:08.550504Z 1 [System] [MY-013576] [InnoDB] InnoDB initialization has s
nginx_movie_microservice | /docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform conf
nginx_movie_microservice | /docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
fpm_movie_microservice | [26-Nov-2024 10:32:08] NOTICE: ready to handle connections
mysql_movie_microservice | 2024-11-26T10:32:08.676396Z 1 [System] [MY-013577] [InnoDB] InnoDB initialization has e
movie_storage_redis | 1:M 26 Nov 2024 10:32:08.132 * DB loaded from disk: 0.000 seconds
movie_storage_redis | 1:M 26 Nov 2024 10:32:08.132 * Ready to accept connections tcp
nginx_movie_microservice | /docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default
```

Рисунок 4.1 – Успішний запуск віртуальної машини з мережею мікросервісів

Фінальна команди застосовуються безпосередньо у контейнері з PHP-FPM і в кожному МС, наступні команди треба ввести у консоль: “php artisan storage:link”, “composer install”. Після треба перевірити стан Docker за допомогою такої команди – “docker ps”, у консоль повинен з’явитися перелік активних контейнерів, наведений на рис. 4.2. Після розгортання МС доступні за такими посиланнями у браузері: “http://movieEmailTemplate”,

“http://movieStorage”, “http://movie/”. Якщо на сторінці помилка з текстом “Need to generate key”, необхідно відкрити у консолі контейнеру корінну папку MC і ввести команду – “php artisan key:generate”.

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
58b1343ae54e	nginx	"/docker-entrypoint..."	9 minutes ago	Up 9 minutes	127.0.0.1:80->80/tcp	nginx_movie_microservice
023d60e27238	univer_fpm	"docker-php-entrypoi..."	9 minutes ago	Up 9 minutes	127.0.0.1:9000->9000/tcp	fpm_movie_microservice
32d721465194	redis	"docker-entrypoint.s..."	9 minutes ago	Up 9 minutes	0.0.0.0:6380->6379/tcp, :::6380->6379/tcp	movie_storage_redis
765e560dfe83	mysql:8.0.26	"docker-entrypoint.s..."	9 minutes ago	Up 9 minutes	127.0.0.1:3306->3306/tcp, 33060/tcp	mysql_movie_microservice
20a6f35335c2	mongo	"docker-entrypoint.s..."	27 hours ago	Up 2 hours	0.0.0.0:27017->27017/tcp, :::27017->27017/tcp	mongo

Рисунок 4.2 – Віртуальні контейнери Docker

Для MC MasterMovieClient 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 MasterMovieClient і ввести дві команди “npm install” і “npm start” . Розгортання інтерфейсу готове і доступне за посиланням – “http://localhost/movies/”. У результаті розгортання системи є один фізичний сервер, один віртуальний контейнер, 4 логічних сервери, які пов’язані з двома глобальними логічними серверами (Firebase, Mailgun).

4.3 Тестування та дослідження сумісності проєкту

Так як система реалізовує інтерфейс спілкування REST API і унікальні контракти мають по 5 або менше класів реалізаторів, то доцільним було протестувати деякі контракти на прості позитивні і негативні тести, а інші, більш складні контракти протестовані з урахування статистичних даних навантаження на фізичний сервер, а точніше віртуальний докер з обмеженою кількістю ресурсів [50]. У якості фізичних обмежень, докер має обмеження по кількості фізичних і логічних ядер, а саме 10 логічних і 5 фізичних, також обмеження за кількістю оперативної пам’яті розміром 8GB. Додатково до попередніх параметрів будуть накладені деякі обмеження по розмірам статичної пам’яті, але з урахуванням тестування, кількість доступної

статичної пам'яті не повинно вплинути на швидкодію. Початковий стан контейнеру продемонстровано на рис. 4.3

CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O	PIDS
58b1343ae54e	nginx_movie_microservice	0.00%	17.43MiB / 31.26GiB	0.05%	27.6kB / 0B	8.92MB / 8.19kB	21
023d60e27238	fpm_movie_microservice	0.00%	9.215MiB / 31.26GiB	0.03%	27.9kB / 0B	12.9MB / 0B	3
32d721465194	movie_storage_redis	0.27%	6.555MiB / 31.26GiB	0.02%	28kB / 0B	15.6MB / 0B	6
765e560dfe83	mysql_movie_microservice	0.07%	350.8MiB / 31.26GiB	1.10%	27.9kB / 0B	16.4MB / 15.1MB	37

Рисунок 4.3 – Віртуальні контейнери Docker

Просте тестування проведено за допомогою Postman (Postman – це інструмент для створення і надсилання запитів до серверів через зручний інтерфейс), протестовані прості API контракти і результати їх тестування наведено у таблиці 4.1.

Таблиця 4.1 – Тестування Postman MC MovieMicroservice

API контракти	Позитивний тест	Негативний тест	Результати
1	2	3	4
GET “user/”	Отримання даних користувача	Не існуючий користувач	+ / +
GET “/movie/list”	Отримання списку фільмів	Надіслані неправильні дані для фільтрації	+ / +
GET “/movie/ {id}”	Отримання даних про фільм	Не існуючий фільм	+ / +
POST “/user/pre-regitser”	Відправка електронного листа на пошту користувача	Надіслані неправильні адрес електронної пошти	+ / +
PATCH “/movie /collections /default/{id}”	Оновлення списку колекції за замовченням фільму	Користувача не має ролі адміністратора	+ / +

Продовження таблиці 4.1

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

Продовження таблиці 4.1

1	2	3	4
DELETE “/collection/{id}”	Видалення колекції	Колекції не існує	+ / +

За результатами простого тестування розроблені API контракти MC MovieMicroservice функціонують у штатному режимі, а саме повертають, оновлюють і створюють записи у БД. Складні тести, що враховують специфіку асинхронних, мультипроцесних і асинхронних операцій MC MasterMovieStorage і MC EmailTempaleMicroservice наведені у таблиці 4.2 і 4.3 відповідно.

Таблиця 4.2 – Тестування MC MasterMovieStorage

API контракти	Реалізатор	Використан а пам'ять (мегабіти)	Навантаженн я процесора (відсотки)	Час виконання (секунди)
POST “/media- uploader/”	Native (синхронне виконання)	14 МБ	19 %	5 с.
POST “/media- uploader/”	AMPHP	12 МБ	19 %	5 с.
POST “/media- uploader/”	PCNTL	25 МБ	25 %	0.1 с + 5 с. (час обробки)
POST “/media- uploader/”	SpatieAsync	17 МБ	20 %	0.1 с + 5 с. (час обробки)

У отриманих результатах тестування MC MasterMovieStorage немає однозначного вирішення, з одного боку можна задовольнити потреби користувача і використати розширення PCNTL для створення швидкої відповіді, але це не забезпечить коректне завантаження медіа файлу у разі

обриву зв'язку з користувачем і до того ж ще майже в двічі навантажує сервер, що не можна назвати енергоефективним. З іншого боку, якщо не враховувати потреби користувача і звернути увагу на оптимізацію використання оперативної пам'яті, то бібліотека SpatieAsync посіла середню позицію і має більш збалансовані результати між навантаженням на сервер і час затримки відповіді, на перший погляд дана бібліотека потенційно є кращим підходом. Проте найкращий результат оптимізації відносно процесора і оперативної пам'яті показала бібліотека AMPHP.

Таблиця 4.3 – Результати тестування MC EmailTemplateMicroservice

API контракти	Реалізатор	Пам'ять (мегабіти)	Процесора (відсотки)	Час (секунди)
POST "/email-codes/"	Native	3.4 Мб	13 %	2.27 с.
POST "/email-codes/"	AMPHP	2.2 Мб	12 %	2.42 с.
POST "/email-codes/"	SpatieAsync	10 МБ	10%	0.1 с. + 2с. (час обробки)
POST "/email-codes/"	PCNTL	11 МБ	15 %	0.1 с + 2 с. (час обробки)
POST "/email-codes/"	Horizon and Redis	2 МБ + 0.5 МБ + 0.5 МБ (кеш)	10% + 3% (під час обробки)	0.1 с. + 2 с. (час обробки)

У результатах тестування MC EmailTemplateMicroservice також не так все однозначно. З одного боку результати поєднання Horizon і Redis показали найменше використання пам'яті, навіть при тому що використовується додатковий кеш, але головним недоліком поєднання є процес Horizon, який повинен постійно підтримуватися в системі, тобто це постійне статичне навантаження на сервер, незначне, але навантаження, якщо помножити час

підтримки процесу на час роботи серверу, то кінцева кількість використаних ресурсів буде значно більшою ніж у інших бібліотеках. Додатково слід порівняти результати бібліотеки SpatieAsync, яка показала гарні результати при тестуванні MC MasterMovieStorage. Знову бібліотека показала збалансований результат між часом затримки відповіді і навантаженням на сервер. Хоча найкращій результат з точки зору оптимізації навантаження на сервер все ж таки показала бібліотека AMPHP.

Додатково була протестована сумісність PCO BestMovieCommon з розробленими MC. Під час встановлення і роботи було знайдено ряд проблем з сумісністю, які були виправлені онволенням пакету.

4.4 Висновки з розділу

У цьому розділі визначено основні напрями застосування розробленого проєкту, а також описано порядок його розгортання та використання. У інструкції з розгортання зазначено обмежений перелік можливих проблем і способи їх вирішення. Розглянуті проблеми є найпоширенішими, здебільшого пов'язаними з ПК і попередньо встановленими сервісами.

Усі сценарії та функції системи були протестовані, включаючи як позитивні, так і негативні кейси, а також окремими тестами навантаження системи мультипроцесними і асинхронними процесами. Таке тестування виявило найкращі бібліотеки для підходу розширення і поліпшення наявних систем і функцій до використання, що дозволяє поетапно покращувати проєкти у майбутньому. Під час перевірки сумісності MC і PCO не було виявлено перешкоди, яка б заважала подальшому розширенню системи.

Під час аналізу було виявлено відносно збалансовану бібліотеку SpatieAsync, найкращу з точки зору продуктивності, а саме AMPHP. Також було виявлено рішення, які краще не застосовувати у системі в силу їх недоліків.

ВИСНОВКИ

У результаті дослідження і розробки проекту по перегляду і редагуванню фільмів і їх колекцій, в умовах застосування складних масштабованих оптимізацій, було виявлено найкращі засоби, що дозволяють знизити фізичне навантаження на сервер і покращити енергозбереження системи, за рахунок застосування мільтипроцесних, асинхронних і багатопотокових операції у конкретних прикладах.

На мою думку найбільш універсальною технологією серед чотирьох розглянутих засобів, які мають різні типи оптимізацій, функцій, результати виконання, навіть мають різний клас, від звичайного розширення до повноцінної бібліотеки, є AMPHP. Ця бібліотека змогла суттєво оптимізувати навантаження на оперативну пам'яті і процесор, порівняно з аналогами. Також завдяки використанню мікросервісної архітектури DDD і значно послабленої віртуальної машини, було доведено що навіть в системах, що розбиті на окремі сервіси і використовують особливі підходи для реалізації окремих рівнів системи, такі як Domain і Infrastructure, є простір для оптимізації. Якщо фундамент архітектури вибудовано згідно встановлених правил, то такий простір має необмежені можливості для впровадження бібліотек і технологій, таких як SpatieAsync і AMPHP, що надають змогу більш прискіпливо контролювати процеси для забезпечення швидкодії, не тільки в контексті перегляду і редагуванню фільмів. Також слід зазначити, що бібліотека фаворит на відміну від інших технологій має ряд додаткових функцій, а саме – блокування запитів до БД, балансування між ядрами процесору, розробка і застосування побітових операції, які не було дослідженні в силу специфіки застосування, що залишає певний простір для подальшого дослідження.

Отже, під час реверсної оцінки дослідження і реалізації проекту, було втілено поставлену мету, а саме виявлено і реалізовано найкращі способи оптимізації, які можна застосувати в умовах різноманітних завдань.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Strategies for Improving the Sustainability of Data Centers via Energy Mix, Energy Conservation, and Circular Energy [Electronic resource]. – Access mode: <https://www.mdpi.com/2071-1050/13/11/6114>
2. Percent of the Web Is Using Outdated Versions of PHP [Electronic resource]. – Access mode: <https://blog.wpsec.com/98-7-percent-of-web-using-outdated-versions-php/>.
3. Modernizing Legacy Applications in PHP Challenges Approaches PHP Application Modernization: Challenges and Approaches in 2024 [Electronic resource]. – Access mode: <https://mobidev.biz/blog/php-application-modernization-case-studies>
4. REST API як спосіб спілкування компонентів вебдодатків [Електронний ресурс]. – Режим доступу: <https://foxminded.ua/shcho-take-rest-api/>.
5. Розуміння Клієнт-Серверної Архітектури на прикладах [Електронний ресурс]. – Режим доступу: <https://dou.ua/forums/topic/44636/>.
6. Різниця між HTTP та HTTPS [Електронний ресурс]. – Режим доступу: <https://hostiq.ua/wiki/ukr/http-https/>.
7. Message Brokers: Pros, Cons, and Their Crucial Role in Microservices [Electronic resource]. – Access mode: <https://dev.to/jackynote/message-brokers-pros-cons-and-their-crucial-role-in-microservices-56pi>.
8. Introducing Laravel Horizon [Electronic resource]. – Access mode: <https://medium.com/@taylorotwell/introducing-laravel-horizon-4585f66e3e>.
9. Firebase vs AWS - Best Backend Solution for App Development in 2024 [Electronic resource]. – Access mode: <https://www.geeksforgeeks.org/firebase-vs-aws/>.
10. Evans, E. Domain-Driven Design: Tackling Complexity in the Heart of Software [Text] / Eric Evans. – Boston: Addison-Wesley, 2003. – 560 с.

11. Підхід до проектування ПЗ DDD [Електронний ресурс]. – Режим доступу: <https://foxminded.ua/ddd-shcho-tse/>.
12. Domain-Driven Design (DDD) [Electronic resource]. – Access mode: <https://redis.io/glossary/domain-driven-design-ddd/>.
13. Watch movies and TV shows on PrimeVideo [Electronic resource]. – Access mode: <https://www.primevideo.com/offers/nonprimehomepage>.
14. Netflix movies [Electronic resource]. – Access mode: <https://www.netflix.com/ua/>.
15. Телебачення та фільми в ультрависокій якості на Sweet TV [Електронний ресурс]. – Режим доступу: <https://sweet.tv/>.
16. Object-Oriented Programming Design Basics for PHP Apps [Electronic resource]. – Access mode: <https://www.zend.com/blog/object-oriented-programming-php>.
17. Percentages of Websites Using PHP [Electronic resource]. – Access mode: <https://kinsta.com/php-market-share/>.
18. Why PHP Remains the Powerhouse of Web Development in 2024 [Electronic resource]. – Access mode: <https://www.uplers.com/blog/why-php-remains-the-powerhouse-of-web-development-in-2024/>.
19. PHP Trends in 2024 [Electronic resource]. – Access mode: <https://daily.dev/blog/php-trends-in-2024/>.
20. An overview of HTTP [Electronic resource]. – Access mode: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview>
21. Top Ranking PHP Frameworks [Electronic resource]. – Access mode: <https://github.com/umpirsky/top-php-frameworks>.
22. DOM... what is it? [Electronic resource]. – Access mode: <https://dev.to/yhuteemoren/dom-what-is-it-3nop>.
23. ReactJS — How to Manipulate the DOM outside of the React Model — The Right Way (Details the ReactJS Docs Missed) [Electronic resource]. – Access mode: <https://medium.com/@garrettmac/reactjs-how-to-safely-manipulate-the-dom-when-reactjs-cant-the-right-way-8a20928e8a6>.

24. What Is SEO – Search Engine Optimization [Electronic resource]. – Access mode: <https://searchengineland.com/guide/what-is-seo>.
25. Cecil, R.M. Clean Architecture: A Craftsman's Guide to Software Structure and Design [Text] / R.M. Cecil. – London: Pearson, 2017. – 432 p.
26. Abel, A. Domain-Driven Design Quickly [Text] / A. Abel, F. Marinescu. – Morrisville: Lulu Press, 2007. – 103 p.
27. Scotts, J. Domain Driven Design : How to Easily Implement Domain Driven Design [Text] / J. Scotts. – Newark: Speedy Publishing LLC, 2015. – 32 p.
28. Domain-Driven Design: Aggregates in Practice [Electronic resource]. – Access mode: <https://medium.com/@aforank/domain-driven-design-aggregates-in-practice-bcced7d21ae5#:~:text=An%20Aggregate%20in%20DDD%20is,business%20rules%20within%20the%20domain>.
29. DDD Aggregates - Best Practices and Implementation Strategies [Electronic resource]. – Access mode: <https://alinabo.com/ddd-aggregates>.
30. Що таке CRUD простими словами: функції, переваги та приклади [Електронний ресурс]. – Режим доступу: <https://highload.today/uk/shho-take-crud-prostimi-slovami-funktsiyi-perevagi-ta-prikladi/>.
31. Value Objects in .NET (DDD Fundamentals) [Electronic resource]. – Access mode: <https://www.milanjovanovic.tech/blog/value-objects-in-dotnet-ddd-fundamentals>.
32. Understanding the Unit of Work Pattern [Electronic resource]. – Access mode: <https://code-maze.com/csharp-unit-of-work-pattern/>.
33. Unit of Work in Repository Pattern [Electronic resource]. – Access mode: <https://www.c-sharpcorner.com/UploadFile/b1df45/unit-of-work-in-repository-pattern/>.
34. Патерни CQRS і Event Sourcing [Електронний ресурс]. Режим доступу: <https://bool.dev/blog/detail/pattern-cqrs-i-event-sourcing>.
35. Controller-Service-Repository [Electronic resource]. – Access mode: <https://tom-collings.medium.com/controller-service-repository-16e29a4684e5>.

36. A Guide to the PHP PCNTL Library for Process Forking and Control [Electronic resource]. – Access mode: <https://reintech.io/blog/guide-to-php-pcntl-library-for-process-forking-and-control>.
37. How to Use async in PHP [Electronic resource]. – Access mode: <https://code.tutsplus.com/how-to-use-async-in-php--cms-39199t>.
38. Exploring Async PHP [Electronic resource]. – Access mode: <https://dev.to/jackmarchant/exploring-async-php-5b68>.
39. Getting started with asynchronous PHP using ReactPHP [Electronic resource]. – Access mode: <https://www.honeybadger.io/blog/getting-started-with-reactphp/>.
40. MasterMovie [Electronic resource]. – Access mode: <https://github.com/ALexxx598/MasterMovie>.
41. How to Use Redis with Laravel: A Comprehensive Guide [Electronic resource]. – Access mode: <https://medium.com/@mohammad.roshandelpoor/how-to-use-redis-with-laravel-a-comprehensive-guide-247cfcac0a68>.
42. How to Use Redis with Laravel: Unleashing the power of Laravel Horizon [Electronic resource]. – Access mode: <https://dev.to/devlopez/unleashing-the-power-of-laravel-horizon-16g9>.
43. EmailTemplateMicroservice [Electronic resource]. – Access mode: <https://github.com/ALexxx598/MasterMovieEmailTemplate>.
44. MasterMovieStorage [Electronic resource]. – Access mode: <https://github.com/ALexxx598/MasterMovieStorage>.
45. Пишемо Unit-тести на PHP: путівник PHPUnit та поради з досвіду [Електронний ресурс]. – Режим доступу: <https://dou.ua/lenta/articles/php-unit-testing/>.
46. BestMovieCommon [Electronic resource]. – Access mode: <https://github.com/ALexxx598/BestMovieCommon>.
47. Packagist: BestMovieCommon [Electronic resource]. – Access mode: <https://packagist.org/packages/BestMovie/common>.
48. PHP and Nginx on Docker [Electronic resource]. – Access mode:

<https://blog.devsense.com/2019/php-nginx-docker>.

49. How To Set Up Laravel, Nginx, and MySQL With Docker Compose on Ubuntu 20.04 [Electronic resource]. – Access mode: <https://www.digitalocean.com/community/tutorials/how-to-set-up-laravel-nginx-and-mysql-with-docker-compose-on-ubuntu-20-04>.

50. Docker Stats | Understand how to monitor Docker Metrics with docker stats [Electronic resource]. – Access mode: <https://signoz.io/blog/docker-stats/>.

ДОДАТОК А
Текст програми

A.1 MovieMicroservice

A.1.1 AsyncMovieService

```

namespace App\MovieLayers\Domain\Movie\Service;

use App\MovieLayers\Domain\Category\Category;
use App\MovieLayers\Domain\Category\Filter\CategoryFilter;
use App\MovieLayers\Domain\Collection\Collection;
use App\MovieLayers\Domain\Collection\CollectionTypeEnum;
use App\MovieLayers\Domain\Collection\Filter\CollectionFilter;
use App\MovieLayers\Domain\Movie\Exception\InvalidImageUrlException;
use App\MovieLayers\Domain\Movie\Exception\InvalidMovieUrlException;
use App\MovieLayers\Domain\Movie\Movie;
use App\MovieLayers\Domain\Movie\MovieDescription;
use App\MovieLayers\Domain\Movie\DPO\MovieCategoryDPO;
use App\MovieLayers\Domain\Movie\DPO\MovieCollectionDPO;
use App\MovieLayers\Domain\Movie\DPO\MovieCreatedPO;
use App\MovieLayers\Domain\User\User;

use function Amp\async;

class AsyncMovieService extends BaseMovieService implements MovieServiceInterface
{
    /**
     * @inheritDoc
     */
    public function create(MovieCreatedDPO $payload): Movie
    {
        $movieValidationPromise = async(function () use ($payload) {
            if (!$this->bestMovieStorageService->validatePath($payload->getStorageMovieUrl())) {
                throw new InvalidMovieUrlException();
            }
        });

        $imageValidationPromise = async(function () use ($payload) {
            if (!$this->bestMovieStorageService->validatePath($payload->getStorageImageUrl())) {
                throw new InvalidImageUrlException();
            }
        });

        $createMoviePromise = async(function () use ($payload) {
            return 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(),
            );
        });

        $movieValidationPromise->await();
        $imageValidationPromise->await();

        /** @var Movie $movie */
        $movie = $createMoviePromise->await();
    }
}

```

```

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

    /**
     * @inheritDoc
     */
    public function syncCollections(MovieCollectionDPO $payload): void
    {
        $userPromise = async(function () use ($payload): User {
            return $this->userService->findUser($payload->getUserId());
        });

        $moviePromise = async(function () use ($payload): Movie {
            return $this->movieRepository->findById($payload->getMovieId());
        });

        /** @var User $user */
        $user = $userPromise->await();

        /** @var Movie $movie */
        $movie = $moviePromise->await();

        $collectionIdsPromise = async(function () use ($payload, $movie, $user): array {
            return $this
                ->collectionService
                ->list(
                    CollectionFilter::make(
                        userId: $user->getId(),
                        type: CollectionTypeEnum::CUSTOM,
                        collectionIds: $payload->getCollectionIds()
                    )
                )
                ->map(fn(Collection $collection) => $collection->getId())
                ->toArray();
        });

        $defaultCollectionIdsPromise = async(function () use ($payload, $movie, $user): array {
            return $this->collectionService
                ->list(
                    CollectionFilter::make(
                        movieId: $movie->getId(),
                        withoutUserId: $user->getId(),
                    ),
                )
                ->map(fn (Collection $collection): int => $collection->getId())
                ->toArray();
        });

        $collectionIds = $collectionIdsPromise->await();
        $defaultCollectionIds = $defaultCollectionIdsPromise->await();

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

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

    /**
     * @inheritDoc
     */
    public function syncDefaultCollections(MovieCollectionDPO $payload): void
    {
        $collectionIdsPromise = async(function () use ($payload): array {

```

```

return $this
    ->collectionService
    ->list(
        CollectionFilter::make(
            type: CollectionTypeEnum::DEFAULT,
            collectionIds: $payload->getCollectionIds()
        )
    )
    ->map(fn (Collection $collection) => $collection->getId())
    ->toArray();
});

$moviePromise = async(function () use ($payload, $collectionIdsPromise): Movie {
    return $this->movieRepository->findById($payload->getMovieId());
});

/** @var Movie $movie */
$movie = $moviePromise->await();

$customCollectionIdsPromise = async(function () use ($payload, $movie): array {
    return $this->collectionService
        ->list(CollectionFilter::make(movieId: $movie->getId(), type: CollectionTypeEnum::CUSTOM))
        ->map(fn (Collection $collection): int => $collection->getId())
        ->toArray();
});

$customCollectionIds = $customCollectionIdsPromise->await();
$collectionIds = $collectionIdsPromise->await();

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

/**
 * @inheritDoc
 */
public function syncCategories(MovieCategoryDPO $payload): void
{
    $moviePromise = async(function () use ($payload): Movie {
        return $this->movieRepository->findById($payload->getMovieId());
    });

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

    /** @var Movie $movie */
    $movie = $moviePromise->await();

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

```

A.1.2 AsyncCollectionService

```
namespace App\MovieLayers\Domain\Collection\Service;
```

```

use App\MovieLayers\Domain\Collection\Collection;
use App\MovieLayers\Domain\Collection\CollectionTypeEnum;
use App\MovieLayers\Domain\Collection\DPO\CollectionCreateDPO;
use App\MovieLayers\Domain\User\User;

```

```
use function Amp\async;
```

```
class AsyncCollectionService extends BaseCollectionService implements CollectionServiceInterface
```

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

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

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

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

    /**
     * @inheritDoc
     */
    public function deleteWithPermissionCheck(int $userId, int $collectionId): void
    {
        $userPromise = async(function () use ($userId, $collectionId): User {
            return $this->userService->findUser($userId);
        });

        $collectionPromise = async(function () use ($userId, $collectionId): Collection {
            return $this->findById($collectionId);
        });

        /** @var Collection $collection */
        $collection = $collectionPromise->await();

        /** @var User $user */
        $user = $userPromise->await();

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

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

A.1.3 ProcessHandleTypeEnum

```
namespace App\Common;
```

```
enum ProcessHandleTypeEnum: string
{
```

```

    case ASYNC_AMPHP = 'ASYNC_AMPHP';
    case SYNC = 'SYNC';
}

```

A.1.4 DefineProcessHandler

```
<?php
```

```
namespace App\Http\Middleware;
```

```

use App\Common\ProcessHandleTypeEnum;
use App\Providers\CollectionServiceProvider;
use App\Providers\MovieServiceProvider;
use Closure;
use Illuminate\Http\Request;

```

```

class DefineProcessHandler
{
    /**
     * Handle an incoming request.
     *
     * @param \Illuminate\Http\Request $request
     * @param \Closure(\Illuminate\Http\Request): (\Illuminate\Http\Response|\Illuminate\Http\RedirectResponse)
     * @throws \App\MovieLayers\Domain\User\Exception\UserNotFoundException
     */
    $next
    public function handle(Request $request, Closure $next)
    {
        $requestHandleType = $request->get('handler_type') !== null
            ? ProcessHandleTypeEnum::tryFrom($request->get('handler_type'))
            : ProcessHandleTypeEnum::SYNC;

        CollectionServiceProvider::registerCollectionService($requestHandleType);
        MovieServiceProvider::registerMovieService($requestHandleType);

        return $next($request);
    }
}

```

A.1.5 AsyncUserService

```
namespace App\MovieLayers\Domain\User\Service;
```

```

use App\MovieLayers\Domain\User\DPO\UserCreatedDPO;
use App\MovieLayers\Domain\User\User;

```

```
use function Amp\async;
```

```

class AsyncUserService extends BaseService implements UserServiceInterface
{
    /**
     * @inheritDoc
     * @throws \Exception
     */
    public function create(UserCreatedDPO $userPayload): User
    {
        $emailTemplateMSCheck = async(function () use ($userPayload) {
            if ($this->emailTemplateService->getCode($userPayload->getEmail())->getCode()
                !== $userPayload->getEmailConfirmationCode()
            ) {

```

```

        throw new \Exception('Code not found.');
```

```

    }
});
```

```

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

```

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

```

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

```

    return $user;
```

```

}
```

```

}
```

A.1.6 MovieServiceProvider

```
namespace App\Providers;
```

```

use App\Common\ProcessHandleTypeEnum;
use App\MovieLayers\Domain\Category\Repository\CategoryRepositoryInterface;
use App\MovieLayers\Domain\Collection\Repository\CollectionRepositoryInterface;
use App\MovieLayers\Domain\Movie\Movie;
use App\MovieLayers\Domain\Movie\Repository\MovieRepositoryInterface;
use App\MovieLayers\Domain\Movie\Service\AsyncMovieService;
use App\MovieLayers\Infrastructure\Movie\Repository\MovieRepository;
use App\MovieLayers\Domain\Movie\Service\MovieService;
use App\MovieLayers\Domain\Movie\Service\MovieServiceInterface;
use App\MovieLayers\Infrastructure\Storage\CachedStorageServiceInterface;
use App\MovieLayers\Infrastructure\Storage\CachedStorageService;
use Illuminate\Support\Facades\App;
use Illuminate\SupportServiceProvider;
```

```
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 (): CachedStorageServiceInterface {
        return $this->app[CachedStorageServiceInterface::class];
    });
```

```

    });
}

/**
 * @param ProcessHandleTypeEnum|null $handleType
 * @return void
 */
public static function registerMovieService(?ProcessHandleTypeEnum $handleType =
ProcessHandleTypeEnum::SYNC): void
{
    match ($handleType->value) {
        ProcessHandleTypeEnum::ASync_AMPHP->value
        => APP::singleton(
            MovieServiceInterface::class,
            AsyncMovieService::class
        ),
        default => APP::singleton(
            MovieServiceInterface::class,
            MovieService::class
        ),
    };
}

/**
 * @return void
 */
public static function registerRepository()
{
    App::singleton(MovieRepositoryInterface::class, MovieRepository::class);
}

/**
 * @return void
 */
public static function registerBestMovieCachedStorageServiceInterface(): void
{
    App::singleton(
        CachedStorageServiceInterface::class,
        CachedStorageService::class
    );
}
}

```

A.1.7 CollectionServiceProvider

```

namespace App\Providers;

use App\Common\ProcessHandleTypeEnum;
use App\MovieLayers\Domain\Collection\Repository\CollectionRepositoryInterface;
use App\MovieLayers\Domain\Collection\Service\AsyncCollectionService;
use App\MovieLayers\Domain\Collection\Service\CollectionService;
use App\MovieLayers\Domain\Collection\Service\CollectionServiceInterface;
use App\MovieLayers\Domain\Movie\MovieCollection\Repository\MovieCollectionRepositoryInterface;
use App\MovieLayers\Infrastructure\Collection\Repository\CollectionRepository;
use App\MovieLayers\Infrastructure\MovieCollection\Repository\MovieCollectionRepository;
use Illuminate\Support\Facades\App;
use Illuminate\Support\ServiceProvider;

class CollectionServiceProvider extends ServiceProvider
{
    /**
     * @inheritDoc

```

```

    */
    public function register()
    {
        $this->registerCollectionService();

        $this->registerCollectionRepository();

        $this->registerMovieCollectionRepository();
    }

    public static function registerCollectionService(?ProcessHandleTypeEnum $handleType =
    ProcessHandleTypeEnum::SYNC): void
    {
        match ($handleType->value) {
            ProcessHandleTypeEnum::ASYNC_AMPHP => APP::singleton(
                CollectionServiceInterface::class,
                AsyncCollectionService::class
            ),
            default => APP::singleton(
                CollectionServiceInterface::class,
                CollectionService::class
            ),
        };
    }

    public static function registerCollectionRepository(): void
    {
        APP::singleton(CollectionRepositoryInterface::class, CollectionRepository::class);
    }

    public static function registerMovieCollectionRepository(): void
    {
        APP::singleton(MovieCollectionRepositoryInterface::class, MovieCollectionRepository::class);
    }
}

```

A.2 EmailTemplateMicroservice

A.2.1 AMPMailerService

```

namespace App\EmailDomain\Email\MailerService;

use App\EmailDomain\Email\MailerService\Mails\SendCodeEmail;
use App\Models\Subscriber;
use Illuminate\Support\Facades\Log;
use Illuminate\Support\Facades\Mail;

class AMPMailerService implements MailerServiceInterface
{
    /**
     * @inheritDoc
     */
    public function sendCodeEmail(string $email, string $code): void
    {
        $ampPool = \Amp\async(function () use ($email, $code) {
            $subscriber = Subscriber::create([
                'email' => $email
            ]);
        });

        if ($subscriber) {

```

```

        Mail::to($email)->send(new SendCodeEmail($email, $code));
    }
});

$ampPool->catch(function (\Exception $exception) {
    Log::error($exception->getMessage());
});

$ampPool->await();
}
}

```

A.2.2 ForkMailService

```

namespace App\EmailDomain\Email\MailerService;

use App\EmailDomain\Email\MailerService\Mails\SendCodeEmail;
use App\Models\Subscriber;
use Illuminate\Support\Facades\Mail;

class ForkMailService implements MailerServiceInterface
{
    public function sendCodeEmail(string $email, string $code): void
    {
        $pid = pcntl_fork();

        if ($pid == -1) {
            // Fork failed
            die('Could not fork the process.');
```

```

        } elseif ($pid) {
            // Parent process - continue executing the main script
            // echo "Parent process continuing. Child PID: $pid\n";
            return;
        } else {
            // Child process - execute the command asynchronously

            $subscriber = Subscriber::create([
                'email' => $email
            ]);

            if ($subscriber) {
                Mail::to($email)->send(new SendCodeEmail($email, $code));
            }

            exit(0);
        }
    }
}

```

A.2.3 QueueMailerService

A.2.4 SendCodeToMail

```

namespace App\EmailDomain\Email\MailerService\Jobs;

use App\EmailDomain\Email\MailerService\MailerService;
use Illuminate\Bus\Queueable;
use Illuminate\Contracts\Queue\ShouldQueue;
```

```

use Illuminate\Foundation\Bus\Dispatchable;
use Illuminate\Queue\InteractsWithQueue;
use Illuminate\Queue\SerializesModels;
use Illuminate\Support\Facades\App;

class SendCodeToMail implements ShouldQueue
{
    use Dispatchable, InteractsWithQueue, Queueable, SerializesModels;

    public function __construct(
        private string $email,
        private string $code
    ) {
    }

    public function handle()
    {
        /** @var MailerService $mailerService */
        $mailerService = App::make(MailerService::class);

        $mailerService->sendCodeEmail($this->email, $this->code);
    }
}

```

A.2.4 SpatieMailerService

```

namespace App\EmailDomain\Email\MailerService;

use App\EmailDomain\Email\MailerService\Mails\SendCodeEmail;
use App\Models\Subscriber;
use Illuminate\Support\Facades\Log;
use Illuminate\Support\Facades\Mail;
use Spatie\Async\Pool;

class SpatieMailerService implements MailerServiceInterface
{
    /**
     * @inheritDoc
     */
    public function sendCodeEmail(string $email, string $code): void
    {
        /**
         * Pool::create()->add(function () use ($email, $code) {
         *     $subscriber = Subscriber::create([
         *         'email' => $email
         *     ])
         * });
         *
         * if ($subscriber) {
         *     Mail::to($email)->send(new SendCodeEmail($email, $code));
         * }
         * }->then(function () {
         * }->catch(function (\Exception $exception) {
         *     Log::error($exception->getMessage());
         * });
         */
    }
}

```

A.3 BestMovieCommon

A.3.1 ProcessHandleType

```
namespace BestMovie\Common\BaseService;

enum ProcessHandleType: string
{
    case SPATIE_ASYNC = 'SPATIE_ASYNC';
    case AMPHP = 'AMPHP';
    case NATIVE_SYNC = 'NATIVE_SYNC';
    case PCNTL = 'PCNTL';
    case QUEUE = 'QUEUE';
}
```

A.3.2 BaseService

```
namespace BestMovie\Common\BaseService;

use BestMovie\Common\BaseService\Exception\WrongProcessHandlerSelectedException;

class BaseService
{
    /**
     * @param array $allowedHandlers
     * @param string $methodName
     * @param ProcessHandleType $type
     * @return void
     * @throws WrongProcessHandlerSelectedException
     */
    protected function validateAllowedProcessHandlers(
        array $allowedHandlers,
        string $methodName,
        ProcessHandleType $type
    ): void {
        if (!in_array($type->value, array_column($allowedHandlers[$methodName], 'value'))) {
            throw new WrongProcessHandlerSelectedException();
        }
    }

    /**
     * @param string $name
     * @param array $arguments
     * @return void
     * @throws WrongProcessHandlerSelectedException
     */
    public function __call(string $name, array $arguments)
    {
        $this->validateAllowedProcessHandlers(
            allowedHandlers: static::ALLOWED_PROCESS_HANDLERS,
            methodName: $name,
            type: $arguments['type']
        );

        call_user_func_array([$this, $name], $arguments);
    }
}
```

A.3.3 BaseApi

```
namespace BestMovie\Common\BaseApi;

use GuzzleHttp\Promise\PromiseInterface;
use Illuminate\Config\Repository as Config;
use Psr\Http\Message\ResponseInterface;
```

```

class BaseApi
{
    protected const MS_ENV_KEY = 'bestMovie';
    protected const MS_AUTH_ENV_KEY = 'MICROSERVICE_AUTH_TOKEN';

    /**
     * @param HttpClient $client
     * @param Config $config
     * @param string $baseUrlConfigKey
     */
    public function __construct(
        private HttpClient $client,
        private Config $config,
        private string $baseUrlConfigKey,
    ) {
    }

    /**
     * @return HttpClient
     */
    public function getClient(): HttpClient
    {
        return $this->client;
    }

    /**
     * @return Config
     */
    public function getConfig(): Config
    {
        return $this->config;
    }

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

    /**
     * @return string
     */
    public function getBaseUrl(): string
    {
        return $this->getConfig()->get(self::MS_ENV_KEY)[$this->getBaseUrlConfigKey()];
    }

    /**
     * @return string
     */
    public function getMSAuth(): string
    {
        return $this->getConfig()->get(self::MS_ENV_KEY)[self::MS_AUTH_ENV_KEY];
    }

    /**
     * @throws \GuzzleHttp\Exception\GuzzleException
     */
    public function post(string $url, array $data): ResponseInterface
    {
        if (array_key_exists('headers', $data)) {

```

```

        $data['headers']['MICROSERVICE_AUTH'] = $this->getMSAuth();

        return $this->getClient()->post($this->getBaseUrl() . $url, $data);
    }

    $data['headers'] = [
        'MICROSERVICE_AUTH' => $this->getMSAuth(),
    ];

    return $this->getClient()->post($this->getBaseUrl() . $url, $data);
}

/**
 * @param string $url
 * @param array $data
 * @return PromiseInterface
 */
public function asyncPost(string $url, array $data): PromiseInterface
{
    if (array_key_exists('headers', $data)) {
        $data['headers']['MICROSERVICE_AUTH'] = $this->getMSAuth();

        return $this->getClient()->postAsync($this->getBaseUrl() . $url, $data);
    }

    $data['headers'] = [
        'MICROSERVICE_AUTH' => $this->getMSAuth(),
    ];

    return $this->getClient()->postAsync($this->getBaseUrl() . $url, $data);
}

/**
 * @throws \GuzzleHttp\Exception\GuzzleException
 */
public function get($url, array $data): ResponseInterface
{
    if (array_key_exists('headers', $data)) {
        $data['headers']['MICROSERVICE_AUTH'] = $this->getMSAuth();

        return $this->getClient()->get($this->getBaseUrl() . $url, $data);
    }

    $data['headers'] = [
        'MICROSERVICE_AUTH' => $this->getMSAuth(),
    ];

    return $this->getClient()->get($this->getBaseUrl() . $url, $data);
}

/**
 * @param $url
 * @param array $data
 * @return PromiseInterface
 */
public function asyncGet($url, array $data): PromiseInterface
{
    if (array_key_exists('headers', $data)) {
        $data['headers']['MICROSERVICE_AUTH'] = $this->getMSAuth();

        return $this->getClient()->getAsync($this->getBaseUrl() . $url, $data);
    }
}

```

```

    $data['headers'] = [
        'MICROSERVICE_AUTH' => $this->getMSAuth(),
    ];

    return $this->getClient()->getAsync($this->getBaseUrl() . $url, $data);
}

/**
 * @param string $name
 * @param array $arguments
 * @return mixed
 */
public function __call(string $name, array $arguments): mixed
{
    if (method_exists($this, $name)) {
        return call_user_func_array([$this, $name], $arguments);
    }

    if (empty($arguments['type'])) {
        return call_user_func_array([$this, $name], $arguments);
    }

    $arguments['data']['handle_type'] = $arguments['type'];

    return call_user_func_array([$this, 'async' . $name], $arguments);
}
}

```

A.3.4 BaseApiFacade

```

namespace BestMovie\Common\BaseApi;

use BestMovie\Common\BaseService\ProcessHandleType;
use GuzzleHttp\Promise\PromiseInterface;

/**
 * @method PromiseInterface get($url, array $data, ProcessHandleType $type)
 * @method PromiseInterface post($url, array $data, ProcessHandleType $type)
 */
class BaseApiFacade extends BaseApi
{
}

```

A.3.4 EmailTemplateApi

```

namespace BestMovie\Common\EmailTemplateMicroservice\Http;

use BestMovie\Common\BaseApi\BaseApi;
use BestMovie\Common\BaseService\ProcessHandleType;
use BestMovie\Common\EmailTemplateMicroservice\Http\Response\GenerateEmailCodeResponse;
use BestMovie\Common\EmailTemplateMicroservice\Http\Response\GetEmailCodeResponse;
use GuzzleHttp\Promise\PromiseInterface;

class EmailTemplateApi extends BaseApi implements EmailTemplateApiInterface
{
    /**
     * @inheritDoc
     */
    public function generateCode(array $data): GenerateEmailCodeResponse|PromiseInterface
    {
        $result = $this->post('/api/email-codes/', $data);

        if ($result instanceof PromiseInterface) {

```

```

        return $result;
    }

    return GenerateEmailCodeResponse::make($result);
}

/**
 * @inheritDoc
 */
public function getCode(array $data): GetEmailCodeResponse|PromiseInterface
{
    $result = $this->get('/api/email-codes/', $data);

    if ($result instanceof PromiseInterface) {
        return $result;
    }

    return GetEmailCodeResponse::make($result);
}
}

```

A.3.4 EmailTemplateApiFacade

```

namespace BestMovie\Common\EmailTemplateMicroservice\Http;

use BestMovie\Common\BaseService\ProcessHandleType;
use GuzzleHttp\Promise\PromiseInterface;

/**
 * @method PromiseInterface generateCode(array $data, ProcessHandleType $type)
 * @method PromiseInterface getCode(array $data, ProcessHandleType $type)
 */
class EmailTemplateApiFacade extends EmailTemplateApi
{
}

```

A.3.4 EmailTemplateApiInterface

```

namespace BestMovie\Common\EmailTemplateMicroservice\Http;

use BestMovie\Common\EmailTemplateMicroservice\Http\Response\GenerateEmailCodeResponse;
use BestMovie\Common\EmailTemplateMicroservice\Http\Response\GetEmailCodeResponse;
use GuzzleHttp\Promise\PromiseInterface;

interface EmailTemplateApiInterface
{
    /**
     * @var array $data
     * @throws \GuzzleHttp\Exception\GuzzleException
     */
    public function generateCode(array $data): GenerateEmailCodeResponse|PromiseInterface;

    public function getCode(array $data): GetEmailCodeResponse|PromiseInterface;
}

```

A.3.4 EmailTemplateService

```

namespace BestMovie\Common\EmailTemplateMicroservice\Service;

```

```

use BestMovie\Common\BaseService\BaseService;
use BestMovie\Common\BaseService\ProcessHandleType;
use BestMovie\Common\EmailTemplateMicroservice\Http\EmailTemplateApiInterface;
use BestMovie\Common\EmailTemplateMicroservice\Http\Response\GenerateEmailCodeResponse;
use BestMovie\Common\EmailTemplateMicroservice\Http\Response\GetEmailCodeResponse;

```

class EmailTemplateService extends BaseService implements EmailTemplateServiceInterface

```

{
    public const array ALLOWED_PROCESS_HANDLERS = [
        'generateCode' => [
            ProcessHandleType::AMPHP,
            ProcessHandleType::NATIVE_SYNC,
            ProcessHandleType::SPATIE_ASYNC,
            ProcessHandleType::PCNTL,
            ProcessHandleType::QUEUE,
        ],
        'getCode' => [
            ProcessHandleType::NATIVE_SYNC,
        ],
    ];

    /**
     * @param EmailTemplateApiInterface $emailTemplateApi
     */
    public function __construct(
        private EmailTemplateApiInterface $emailTemplateApi,
    ) {
    }

    /**
     * @inheritDoc
     */
    public function generateCode(
        string $email,
        ?int $expireTime = null,
        ?ProcessHandleType $type = ProcessHandleType::NATIVE_SYNC
    ): GenerateEmailCodeResponse {
        return $this->emailTemplateApi->generateCode([
            'form_params' => [
                'email' => $email,
                'expire_time' => $expireTime,
            ],
        ], $type);
    }

    /**
     * @inheritDoc
     */
    public function getCode(
        string $email,
        ?int $expireTime = null,
        ?ProcessHandleType $type = ProcessHandleType::NATIVE_SYNC
    ): GetEmailCodeResponse {
        return $this->emailTemplateApi->getCode([
            'query' => [
                'email' => $email,
            ],
        ], $type);
    }
}

```

ДОДАТОК Б

Конфігураційні файли Docker

Б.1 Файл “docker-compose.yml”

version: “3.8”

services:

nginx:

container_name: nginx_movie_microservice

build:

context: .

dockerfile: Docker/Nginx.Dockerfile

image: nginx

volumes:

-

/var/www/Univer/MasterMovie/MovieMicroservice:/var/www/Univer/MasterMovie/MovieMicroservice

- /var/www/Univer/MasterMovieStorage:/var/www/Univer/MasterMovieStorage

- /var/www/Univer/EmailTemplateMicroservice:/var/www/Univer/EmailTemplateMicroservice

- /var/www/Univer/BestMovieCommon:/var/www/Univer/BestMovieCommon

ports:

- 127.0.0.1:80:80

links:

- fpm

- database

restart: unless-stopped

environment:

VIRTUAL_HOST: movie

APP_DEBUG: 1

PHP_IDE_CONFIG: “serverName=movieMicroserviceDocker”

extra_hosts:

- “host.docker.internal:host-gateway”

- “movieemailtemplate:127.0.0.1”

- “movieStorage:127.0.0.1”

networks:

- diplomNetwork

fpm:

container_name: fpm_movie_microservice

build:

context: .

dockerfile: Docker/Fpm.Dockerfile

volumes:

-

/var/www/Univer/MasterMovie/MovieMicroservice:/var/www/Univer/MasterMovie/MovieMicroservice

- /var/www/Univer/MasterMovieStorage:/var/www/Univer/MasterMovieStorage

- /var/www/Univer/EmailTemplateMicroservice:/var/www/Univer/EmailTemplateMicroservice

- /var/www/Univer/BestMovieCommon:/var/www/Univer/BestMovieCommon

ports:

- 127.0.0.1:9000:9000

links:

- database

networks:

- diplomNetwork

database:

container_name: mysql_movie_microservice

platform: linux/x86_64

image: mysql:8.0.26

volumes:

- /var/www/Univer/tmp/data:/var/lib/mysql

ports:

- 127.0.0.1:3306:3306

environment:

- MYSQL_ROOT_PASSWORD=4as6iKILL47<>and06052002alex

- MYSQL_PASSWORD=123

- MYSQL_DATABASE=diplom_movie

networks:

- diplomNetwork

redis:

image: redis

volumes:

- /var/www/Univer/tmp/data:/data

container_name: movie_storage_redis

ports:

- "6380:6379"

networks:

- diplomNetwork

networks:

diplomNetwork:

name: nginx-proxy

Б.2 Конфігурація PHP у "Fpm.dockerfile"

FROM php:8.3-fpm

RUN apt-get update && apt-get install -y \

libicu-dev \

libpq-dev \

```

git \
zip \
unzip \
&& docker-php-ext-install \
pdo_mysql \
bcmath \
mysqli \
pdo \
intl \
pcntl

```

```
RUN docker-php-ext-configure pcntl --enable-pcntl
```

```
#install mysql client for migrations
```

```
RUN apt-get -y --no-install-recommends install default-mysql-server
```

```
#install composer
```

```
RUN curl -sS https://getcomposer.org/installer | php -- --install-dir=/usr/bin/ --filename=composer
```

```
#install xdebug
```

```
COPY --from=mlocati/php-extension-installer /usr/bin/install-php-extensions /usr/bin/
```

```
RUN install-php-extensions xdebug && docker-php-ext-enable xdebug
```

```
#xdebug mappings
```

```
ENV PHP_IDE_CONFIG 'serverName=movieMicroserviceDocker'
```

```
RUN echo "xdebug.mode=debug" >> /usr/local/etc/php/conf.d/docker-php-ext-xdebug.ini
```

```
RUN echo "xdebug.start_with_request=yes" >> /usr/local/etc/php/conf.d/docker-php-ext-xdebug.ini
```

```
RUN echo "xdebug.client_host=192.168.0.109" >> /usr/local/etc/php/conf.d/docker-php-ext-xdebug.ini
```

```
RUN echo "xdebug.discover_client_host=1" >> /usr/local/etc/php/conf.d/docker-php-ext-xdebug.ini
```

```
RUN echo "xdebug.client_port=9001,9000,9003" >> /usr/local/etc/php/conf.d/docker-php-ext-xdebug.ini
```

```
RUN echo "xdebug.log=/var/log/xdebug.log" >> /usr/local/etc/php/conf.d/docker-php-ext-xdebug.ini
```

```
RUN echo "xdebug.idekey = PHPSTORM" >> /usr/local/etc/php/conf.d/docker-php-ext-xdebug.ini
```

```
RUN echo "xdebug.log_level=0" >> /usr/local/etc/php/conf.d/docker-php-ext-xdebug.ini
```

```
RUN echo "xdebug.remote_handler=dbgp" >> /usr/local/etc/php/conf.d/docker-php-ext-xdebug.ini
```

```
RUN echo "memory_limit=512M" >> /usr/local/etc/php/conf.d/docker-php-fpm.ini
```

```
RUN echo "upload_max_filesize=50M" >> /usr/local/etc/php/conf.d/docker-php-fpm.ini
```

```
RUN echo "post_max_size=50M" >> /usr/local/etc/php/conf.d/docker-php-fpm.ini
```

```
COPY . /var/www/Univer/
```

```
WORKDIR /var/www/Univer/
```

Б.3 Конфігурація Nginx у файлі “Nginx.dockerfile”

```
FROM nginx
```

```
ADD Docker/movieMS.conf /etc/nginx/conf.d/default.conf
```

```
COPY . /var/www/Univer/MasterMovie/MovieMicroservice
```

```
WORKDIR /var/www/Univer/MasterMovie/MovieMicroservice
```

```
COPY . /var/www/Univer/MasterMovieStorage
```

```
WORKDIR /var/www/Univer/MasterMovieStorage
```

```
COPY . /var/www/Univer/EmailTemplateMicroservice
```

```
WORKDIR /var/www/Univer/EmailTemplateMicroservice
```

Б.4 Файл “movieMs.conf”

```
server {
    listen 80;
    server_name movie;
    root /var/www/Univer/MasterMovie/MovieMicroservice/public;

    add_header X-Frame-Options "SAMEORIGIN";
    add_header X-Content-Type-Options "nosniff";

    index index.php;

    charset utf-8;

    location / {
        #proxy_pass http://movie;
        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 fpm:9000;
    fastcgi_param SCRIPT_FILENAME $realpath_root$fastcgi_script_name;
    include fastcgi_params;
}

location ~ /\.(!well-known).* {
    deny all;
}

}

server {
    listen 80;
    server_name movieStorage;
    root /var/www/Univer/MasterMovieStorage/public;

    add_header X-Frame-Options "SAMEORIGIN";
    add_header X-Content-Type-Options "nosniff";

    index index.php;

    charset utf-8;

    location / {
        #proxy_pass http://fpm_movie_microservice:9000;
        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 fpm:9000;
        fastcgi_param SCRIPT_FILENAME $realpath_root$fastcgi_script_name;
        include fastcgi_params;
    }

    location ~ /\.(!well-known).* {
        deny all;
    }
}

```

```

server {
    listen 80;
    server_name movieEmailTemplate;
    root /var/www/Univer/EmailTemplateMicroservice/public;

    add_header X-Frame-Options "SAMEORIGIN";
    add_header X-Content-Type-Options "nosniff";

    index index.php;

    charset utf-8;

    location / {
        #proxy_pass http://movieEmailTemplate;
        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 fpm:9000;
        fastcgi_param SCRIPT_FILENAME $realpath_root$fastcgi_script_name;
        include fastcgi_params;
    }

    location ~ /\.(!well-known).* {
        deny all;
    }
}

```

Б.5 Файл “.env” MasterMovieStorage

```

APP_ENV=local
APP_KEY=
APP_NAME=MovieMicroservice
APP_DEBUG=true
APP_URL=http://localhost

```

```

LOG_CHANNEL=stack
LOG_DEPRECATED_CHANNEL=null
LOG_LEVEL=debug

DB_CONNECTION=mysql
DB_HOST=database
DB_PORT=3306
DB_DATABASE=diplom_movie
DB_USERNAME=root
DB_PASSWORD=4as6iKILL47<>and06052002alex
#DB_SOCKET=/var/run/mysqld/mysqlx.sock

BROADCAST_DRIVER=log
CACHE_DRIVER=redis
FILESYSTEM_DISK=local
QUEUE_CONNECTION=redis
SESSION_DRIVER=file
SESSION_LIFETIME=120

MEMCACHED_HOST=127.0.0.1

REDIS_HOST=127.0.0.1
REDIS_PASSWORD=null
REDIS_PORT=6380

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://movieEmailTemplate:80  
BEST_MOVIE_STORAGE_MS=http://movieStorage
```

```
MICROSERVICE_AUTH_TOKEN=//token
```

ДОДАТОК В
Презентація

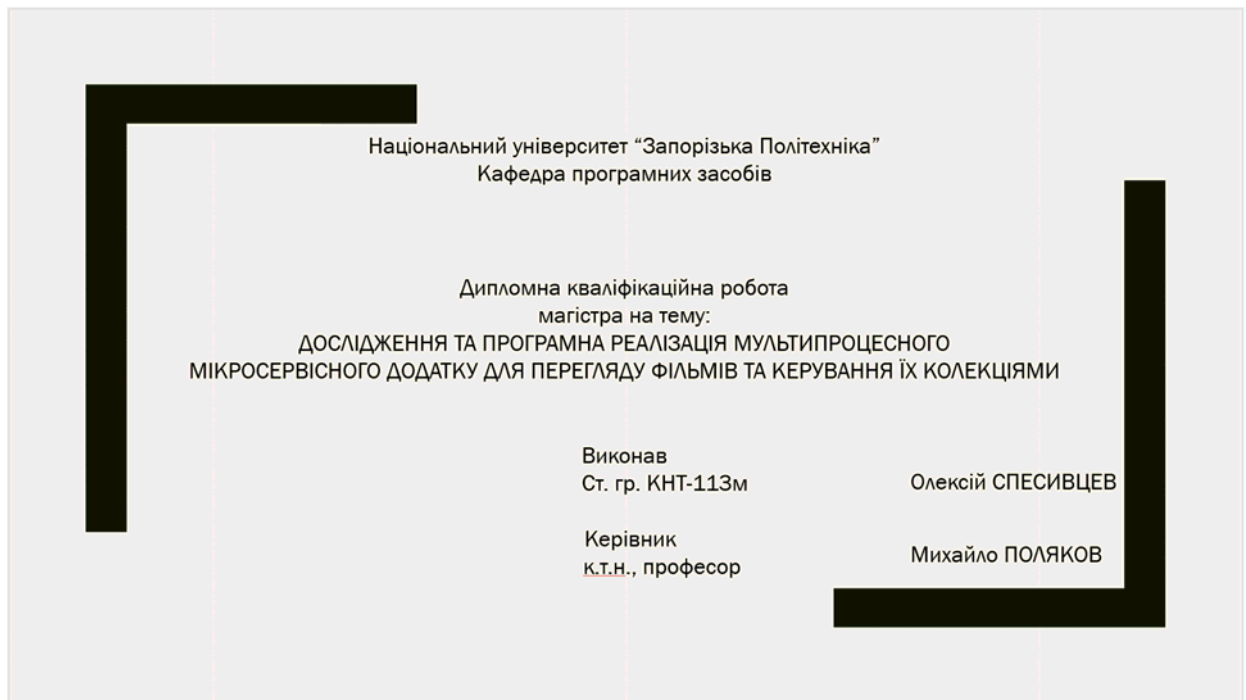


Рисунок В.1 – Слайд №1

№2

Об'єкт, предмет, мета та задачі роботи

Об'єкт дослідження – мультипроцесні програмні засоби для перегляду фільмів та керування їх колекціями, реалізовані на основі мікросервісної архітектури з використанням підходу Domain Driver Design (DDD).

Предмет дослідження - сайти (сервіси) з високою швидкістю обробки великих даних для перегляду фільмів та менеджментом колекцій фільмів.

Мета роботи – створення програмної системи для зручного завантаження відео та швидкого менеджменту колекційних конфігурацій за умов високої продуктивності системи.

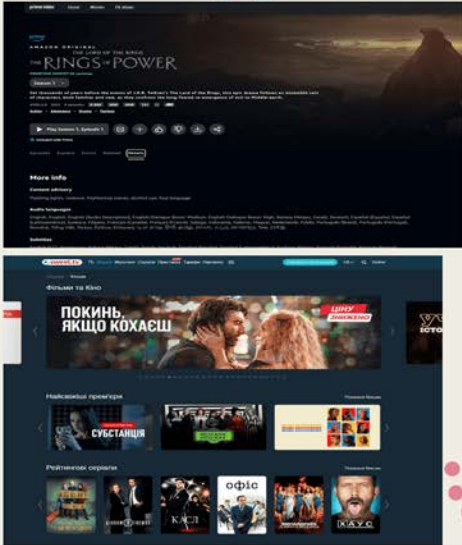
Задачі:

- дослідити предмету область, порівняти аналоги ;
- проаналізувати архітектуру DDD та розробити ряд мікросервісів;
- розробити асинхронний Public Contract
- розробити систему для перегляду фільмів та керування ї колекціями;
- створити 5 моделей обробки запитів;
- провести тестування системи

Рисунок В.2 – Слайд №2

№3

Сервіси і сайти аналоги



Порівняльні характеристики	PrimeVideo	Netflix	SweetTV
Якість фільмів	Висока	Висока	Надвисока
Розвинуті технології кешу	Відсутні	Open Connect	Відсутні
Залежність від сторонніх сервісів	Немає	Висока	Немає
Категорії і колекції	+	+	-
Адаптація під платформи і сервіси	Smart TV, Android, IOS, Blu-ray, Fire TV	Smart TV, Android, IOS, Xbox, PlayStation Blu-ray, Chromecast,	Smart TV, Android, IOS
Стабільність	Висока	Висока	Висока
Швидкодія	Висока	Висока	Низька

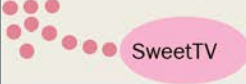


Рисунок В.3 – Слайд №3

№4

Порівняння мов програмування

Порівняльні характеристики	PHP	C#	Go	Python
ООП, класи, абстракції	Присутні, достатньо розвинуті	Присутні і добре розвинуті	Присутні, але дуже обмежені	Присутні, але менш розвинуті
Асинхронність	Присутня за допомогою, AMPHP, SpatieAsync	Присутня за допомогою async/await	Вбудована через Goroutines	Присутня за допомогою asyncio
Система управління пам'яті	Обмежена	Є	Є	Обмежена
Мультипроцесність	Розширення pcntl	Вбудована підтримка	Вбудована через Goroutines	Можлива через threading і concurrent.futures
Типізація	Слабка динамічна	Сильна статична	Сильна статична	Слабка динамічна
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, десктоп застосунки, ігри, мобільні застосунки	Вебзастосунки, десктопні застосунки, ігри, мобільні застосунки	Штучний інтелект, вебзастосунки, десктоп застосунки, ігри.
Реєстрація пакетів онлайн	Packagist	Maven Central	NuGet	PyPI

Рисунок В.4 – Слайд №4

Архітектура Domain Driven Design (DDD)

№5

Чотири рівні:

- User Interface
- Application
- Domain
- Infrastructure

Шаблони програмування, що формують рівні:

- Service and Repository
- DPO (Domain Payload Object)
- Aggregate (Value objects, Traits Resolvers)
- DI pattern
- UOW (Unit of Work)
- CQRS (The command and Query Responsibility Segregation)

Проблеми розробки:

- Порушення SOLID через використання DI
- Суперкласи
- Рекурсивні залежності

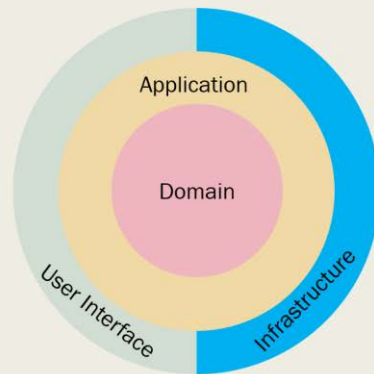


Рисунок В.5 – Слайд №5

Мультипроцесні бібліотеки, розширення, технології

№6

Розширення PCNTL

AMPHP

SpatieAsync

Horizon + Redis

Особливості:

1. Усі бібліотеки, а саме - AMPHP, SpatieAsync, базуються на базовому розширенні PCNTL



2. AMPHP додатково використовує PHP Fibers, і реалізує відразу асинхронні і мультипроцесні операції



4. Розширення PCNTL надає базовий набір недосконалих функцій, такі як pcntl_fork, stream_context

3. SpatieAsync реалізована на основі функції stream_context, і тому є багатопотоковою реалізацією



5. Horizon і Redis взагалі працюють без використання розширення PCNTL і застосовують підхід підтримки з'єднання з контейнером.



Рисунок В.6 – Слайд №6

Розроблена БД і міграції

- Основна СКБД MySQL
- Міграції адаптовані під тестову СКБД SQLite

```

migrations
  2014_10_12_000000_create_users_table.php
  2014_10_12_100000_create_password_resets_table.php
  2019_08_19_000000_create_failed_jobs_table.php
  2019_12_14_000001_create_personal_access_tokens_table.php
  2024_09_14_092344_create_roles_table.php
  2024_09_14_093344_create_movies_table.php
  2024_09_14_094344_create_categories_table.php
  2024_09_14_096344_create_collections_table.php
  2024_09_14_097333_create_movie_category_table.php
  2024_09_14_097344_create_movie_collection_table.php
  
```

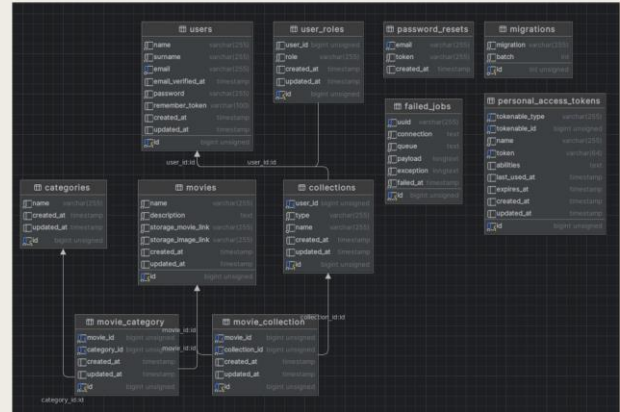


Рисунок В.9 – Слайд №9

Розроблені обробники процесів

Найбільша кількість реалізаторів процесів була розроблена у MC EmailTemplateMicroservice.

1. AMPHPMailerService
2. ForkMailerService (Pcntl)
3. MailerService (Native)
4. QueueMailerService
5. SpatieMailerService

У MC MasterMovieStorage було розроблено класи реалізатори для наступних обробників

1. AMPHP
2. SpatieAsync
3. Pcntl
4. Native
5. Horizon

У головний MC було встановлено бібліотеку AMPHP і на її основі розроблено ряд класів і методів для реалізації менеджменту фільмами і колекціями.

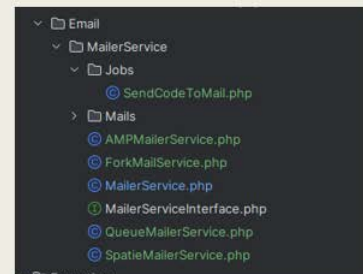
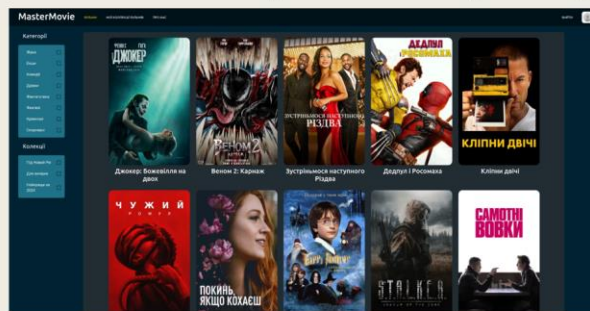


Рисунок В.10 – Слайд №10

Головні сторінки користувачів

№11



Сторінка звичайного і зареєстрованого користувача

Сторінка адміністратора

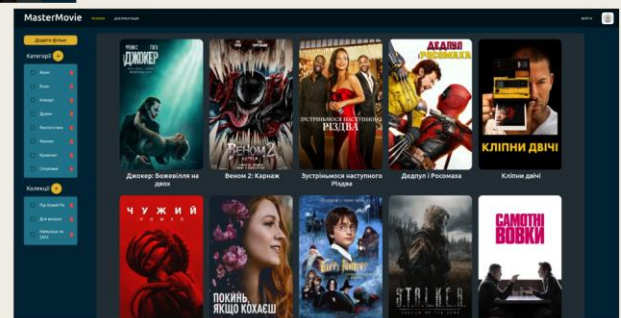


Рисунок В.11 – Слайд №11

Основні сторінки і модальні вікна

№12

Movie information

☐ Під Новий Рік
☐ Для вечірки
☐ Найкраще за 2024

Завантажити афішу: Choose File No file chosen
 Завантажити фільм: Choose File No file chosen

Назва:
 Рейтинг:
 Слоган:
 Дата публікації:
 Країна:
 Актори:
 Короткий опис:

Додати до категорії

☒ Жанр
☐ Екшн
☐ Комедія
☐ Драми
☐ Фантастика
☐ Фестиваль
☐ Кримінал
☐ Спортивна

Назва: Джонні Бомбейла на даху
 Рейтинг: 5.0
 Слоган: Хто з нас не Бомбейлний
 Дата публікації: 2024-11-12
 Країна: США
 Актори: Хоакін Фенікс, Зейд Бітц, Ред Гітс, Кен Лам, Брендан Гісон, Катрін Клар

Додати до колекції

☐ Під Новий Рік
☐ Для вечірки
☐ Найкраще за 2024

Назва: Джонні Бомбейла на даху
 Рейтинг: 5.0
 Слоган: Хто з нас не Бомбейлний
 Дата публікації: 2024-11-12
 Країна: США
 Актори: Хоакін Фенікс, Зейд Бітц, Ред Гітс, Кен Лам, Брендан Гісон, Катрін Клар

Рисунок В.12 – Слайд №12

Зареєструватися

[Увійти](#)

Ім'я

Прізвище

Адрес електронної пошти

Пароль

Підтвердження паролю

Зареєструватися

Увійти

[Зареєструватися](#)

Адрес електронної пошти

Пароль

Увійти!

Розгортання і використання мікросервісів у ОС Linux Mint

3. Поміняти шляхи у докер файлі на локальні
4. Запустити Docker командою `sudo docker-compose up --build`

[illegible]

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
3801343ae54e	nginx	"docker-entrypoint --"	9 minutes ago	Up 9 minutes	127.0.0.1:80->80/tcp	nginx_movie_microservice
02306e27238	univer_fm	"docker-php-entrypoint --"	9 minutes ago	Up 9 minutes	127.0.0.1:9000->9000/tcp	fm_movie_microservice
32a79770303	redis	"docker-entrypoint --"	9 minutes ago	Up 9 minutes	127.0.0.1:6379->6379/tcp, 113300->6379/tcp	movie_storage_redis
76525a6d3f83	mysql:8.0.26	"docker-entrypoint --"	9 minutes ago	Up 9 minutes	127.0.0.1:3306->3306/tcp, 33060->3306/tcp	movie_storage_mysql
7925539370c3	nginx	"docker-entrypoint --"	71 hours ago	Up 2 hours	0.0.0.0:72913->72913/tcp, 127.0.0.1:72913->72913/tcp	nginx

Рисунок В.14 – Слайд №14

Тестування

№15

Було проведено чотири етапи тестування

1. Візуальне, за допомогою інтерфейсу
2. Просте за допомогою Postman
3. Тест на сумісність PCO з мікросервісами
4. Тестування на продуктивність

CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O	PID
88b13430e54e	nginx-microservice	0.68%	17.43MiB / 31.26GiB	0.05%	27.8kB / 0B	0.92MB / 8.19kB	21
6226d6d72238	api-microservice	0.94%	9.21MiB / 31.26GiB	0.03%	27.8kB / 0B	12.9MB / 0B	3
32d721465194	movie-storage-redis	0.27%	6.55MiB / 31.26GiB	0.02%	28kB / 0B	15.0MB / 0B	6
70ce5d8f1e03	mysql-microservice	0.87%	358.0MiB / 31.26GiB	1.10%	27.9kB / 0B	10.4MB / 15.1MB	37

API контракти	Реалізатор	Пам'ять (мегабіти)	Процесора (відсотки)	Час (секунди)
POST "/email-codes/"	Native	3.4 МБ	13 %	2.27 с.
POST "/email-codes/"	AMPHP	2.2 МБ	12 %	2.42 с.
POST "/email-codes/"	SpatieAsync	10 МБ	10%	0.1 с. + 2с. (час обробки)
POST "/email-codes/"	PCNTL	11 МБ	15 %	0.1 с + 2 с. (час обробки)
POST "/email-codes/"	Horizon and Redis	0.5 МБ + 0.5 МБ (кеш)	10% + 3% (під час обробки)	0.1 с. + 2 с. (час обробки)

Рисунок В.15 – Слайд №15

Висновки

№16

- було досліджено і реалізовано метод створення мікросервісів, їх багатoshарову архітектуру, способи спілкування і шаблони програмування, для збільшення можливих сценаріїв поведінки системи;
- у роботі були створено PCO видів Public Contract;
- розроблено систему авто підняття мікросервісної мережі, сервісів і засобів
- велику частину дослідження було приділено мультипроцесним, багатопотоковим і асинхронним операціям. У результаті чого було досліджено бібліотеки і розширення: PCNTL, AMPHP, SpatieAsync;
- у рамках системи було використано Firebase, Mailgun, Horizon, Redis, які надали можливість навантажити систему процесами з'єднання з третіми сервісами, що дозволило симулювати навантаження мікросервісної системи наближеною до реального середовища.

Рисунок В.16 – Слайд №16