КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра інтелектуальних програмних систем

Кваліфікаційна робота

на здобуття освітнього рівня бакалавра

за спеціальністю 121 Інженерія програмного забезпечення

на тему:

РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ ДЛЯ ПОШУКУ ІНФОРМАЦІЇ ПРО ФІЛЬМИ

Виконав студент 4-го курсу
Єгор ПОЛІЩУК
Науковий керівник:
доцент, кандидат фіз.-мат. наук
Євгеній ІВАНОВ

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.
Поліщук Є.Д.

Роботу розглянуто й допущено до захисту на засіданні кафедри інтелектуальних програмних систем
дата
протокол №
Завідувач кафедри
Олександр ПРОВОТАР

Київ – 2021

РЕФЕРАТ

Обсяг роботи 56 сторінок, 20 ілюстрацій, 1 таблиця, 24 джерела посилання, 2 додатки.

API, LARAVEL, АДМІН-ПАНЕЛЬ, БЕКЕНД, ВЕБ-СКРЕЙПІНГ, ІНФОРМАЦІЙНА СИСТЕМА, ОПТИМІЗАЦІЯ ФРЕЙМВОРКУ, ПАРСИНГ, ПОШУК, ФІЛЬМ.

Об’єктом розробки є інформаційна система з текстовою інформацією про кінофільми та додатковими можливостями для користувачів у вигляді рейтингової системи, системи рекомендацій тощо. Розробка передбачає створення інтерфейсу для адміністратора та API для клієнтських застосунків.

Метою роботи є створення програмного продукту для його подальшого розгортання у мережі Інтернет та надання доступу до нього широку колу користувачів. Також метою розробки є закріплення знань з розробки усіх необхідних компонентів повноцінної веб-орієнтованої системи.

Основна задача полягає у написанні програмного коду та тестуванні системи, тому використовуються відповідні програмні інструменти, у тому числі – середовища розробки та текстові редактори («PHPStorm», «VisualCode»), середовище виконання серверного ПЗ («Open Server Panel», «Node.js»), СУБД «MySQL» та клієнт до нього «MySQL Workbench» та «DBeaver», мова програмування PHP, фреймворк «Laravel», пакетні менеджери «Composer» та «npm»; браузери «Mozilla Firefox» та «Google Chrome», «Postman» для тестування API. Процес розробки системи відповідає відповідає еволюційній моделі розробки ПЗ [1].

У ході виконання роботи розглянуто підходи до розробки веб-орієнтованих інформаційних систем, їх необхідні компоненти та доцільність їх використання. Проведено загальний огляд схожих систем та описано процес розробки даної. Розроблено серверний (багатокористувацький) додаток на базі фреймворку «Laravel» для зберігання та керування

інформацією про фільми, у тому числі розроблено відповідні кінцеві точки (endpoints, API) для з’єднання з додатком.

Система працює в тестовому режимі в локальному середовищі – на персональному комп’ютері.

Робота виконана як підсумок знань та досвіду з розробки програм, отриманих на курсах «Програмна інженерія», «Інформаційні системи», «Розробка WEB-орієнтованих систем», «Груповий проект», «Організація баз даних та знань» та інших. Ідея створення даної системи роботи є ініціативою виконавця.

Система може бути впроваджена для використання в мережі Інтернет за призначенням для любителів та дослідників кіно, зацікавлених в отриманні інформації про фільми, написанні відгуків, нотуванні для себе записів щодо них та для формування списків улюблених композицій.

Функціонал програмного продукту може бути розширений шляхом надання списків посилань на сайти, де фільми можна переглянути чи завантажити, придбати квитки на перегляд у кінозалі тощо. Програма може надавати систему мотивацій для користувачів, наприклад, у вигляді балів за певну роботу, спрямовану на зростання популярності сайту та покращення його наповнення. Система відкрита до пропозицій з боку кінцевих користувачів.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

API – application programming interface, прикладний програмний інтерфейс;

CDN – content delivery network, мережа доставки контенту;

CI/CD – continuous integration / continuous delivery, постійна інтеграція / безперервна доставка;

CLI – command line interface, інтерфейс командного рядка;

CRUD – create-read-update-delete;

DOM – document object model;

IDE – integrated development environment, інтегроване середовище розробки;

SEO – search engine optimization, пошукова оптимізація;

SPA – single page application, односторінковий застосунок;

SSR – server side rendering, рендеринг на стороні сервера;

БД – база даних;

ІС – інформаційна система;

ПЗ – програмне забезпечення;

СУБД – система управління базами даних.

ВСТУП

Оцінка сучасного стану об’єкта розробки. У сучасному світі більшість сфер з надання послуг переходять в онлайн за можливості повністю або принаймні частково, зокрема будуються довідкові сайти, інтернет-магазини, сайти з бронювання та замовлення продуктів та послуг (квитки, готелі, купівля нерухомості) з одного боку, а з іншого створюються онлайн-кінотеатри, мережеві ігри, бібліотеки тощо, тобто розважально-пізнавальні ресурси. Такі технології сприяють росту бізнесу та задоволенню потреб споживачів.

Тому побудова веб-орієнтованих інформаційних систем є невід’ємною частиною розвитку індустрії інформаційних технологій.

Одним з векторів розробки є створення онлайн-бібліотек, у тому числі й фільмів. Такі сайти постійно створюються та конкурують на підставах зручності чи доступності клієнтам.

Актуальність роботи та підстави її виконання. Часто в шанувальників кіно постає дилема вибору фільму до перегляду. Причому є багато критеріїв, за якими відбувається вибір. Серед них – жанр кіно, акторський склад, тематика та сюжет, оцінка інших глядачів – рейтинг, відгуки та рецензії. Щоб зробити сукупний аналіз цих факторів та сформувати свою оцінку, користувачі вимушені блукати сайтами, що забирає чимало часу. Тож зібравши цю та додаткову інформацію в одному місці, можна отримати ресурс вартий уваги мільйонів поціновувачів кіно.

Звичайно, у світовій павутині є чимало аналогів, проте дійсно корисні та цікаві ресурси платні («Netflix») або ж суто англомовні («IMDb»), закриті для користувачів з України («Кинопоиск») або ж містять малу вибірку фільмів чи їх атрибутів. Тому створення та розгортання даної системи може виявитись корисним для прихильників кіномистецтва.

Мета й завдання роботи. Метою даної роботи є створення бекенду програмної системи «All films» для пошуку інформації про фільми.

Для цього необхідно систематизувати досвід розробки інформаційних систем, виявити атрибути фільмів та інтерактивні складові веб-орієнтованих систем, визначити користувацькі вимоги до системи, проаналізувати можливості реалізації цих вимог, спроектувати систему та запрограмувати її складові.

Об’єкт, методи та засоби розробки. Об’єктом розробки є інформаційна система, створення якої передбачає вирішення комплексу задач, пов'язаних з організацією даних та доступу до них в цій системі.

Процес виконання роботи відповідає еволюційній моделі розробки [1], тобто постійно супроводжується уточненням вимог до системи від більш абстрактних до конкретних.

Основна програма побудована на базі фреймворку «Laravel» та написана мовою PHP. Програма використовує хмарні сервіси «Pusher», «Algolia», «Scraperapi», «Recombee» та відповідні клієнти для них, встановлені з репозиторію «packagist.org».

Як інструменти для написання коду були використані IDE PHPStorm та текстовий редактор «Visual Studio Code»; для тестування API – «Postman», «Google Chrome», «Mozilla Firefox», утиліта «cURL»; пакетний менеджер «Composer» для встановлення пакетів, розв’язування залежностей та оптимізації коду; «npm» та «Node.js» для встановлення та запуску клієнта для бази даних Redis; MySQL у якості основної СУБД та клієнти «MySQL Workbench» та «DBeaver» для відображення структури БД; програма «Open Server Panel» для забезпечення середовища розгортання проекту.

Можливі сфери застосування. Робота орієнтована на задоволення потреб повсякденного життя та художніх інтересів користувачів. Дана програмна система може бути використана для розважально-пізнавальних цілей шляхом її розгортання у мережі Інтернет.

Взаємозв’язок з іншими роботами. Система створена повністю за ініціативою автора в межах даної роботи. У розробці використані знання та досвід отриманий переважно з курсів «Груповий проект з технології програмування», «Програмна інженерія», «Інформаційні системи» та інших дисциплін, пройдених у рамках навчальної програми спеціальності.

1 СУЧАСНІ ВЕБ-ОРІЄНТОВАНІ ІНФОРМАЦІЙНІ СИСТЕМИ

1.1 Структура сучасного веб-застосунку

Сучасні інформаційні системи з великою завантаженістю потребують властивості масштабованості. Для цього необхідно забезпечити максимальну незалежність та одночасно узгодженість її основних компонентів, звернути увагу на блокування операцій читання-запису, складні та ресурсномісткі процеси тощо.

Більшість старих веб-орієнтованих систем написані монолітно та були розраховані для запуску на одному сервері в єдиному екземплярі. У той час як вимоги до швидкодії застосунку чи швидкого розширення його функціоналу зростали, система залишалась складною для внесення правок, а її вузьким місцем ставала спроможність сервера до обслуговування запитів. Громіздким системам доводилося проводити рефакторинг, у ході якого виділялись окремі, більшою мірою незалежні підсистеми. Узагальнюючи такий досвід, розглянемо деякий абстрактний клієнт-серверний застосунок, що обслуговує вхідні запити, працюючи на сервері.

Сервер має обслуговувати вхідні запити максимально швидко, тож його потрібно якомога більше розвантажити від супутніх задач, до того ж розглядаючи виділені сервери та загальні практики впровадження систем, бажано відділити файлове сховище від даного сервера. Отож, першим кроком необхідно відокремити дисковий простір системи. Для цього необхідно запустити спеціальне програмне забезпечення на окремому сервері або ж використати хостингові сервіси, наприклад від Google (Google Cloud Storage) чи Amazon (Amazon S3).

Надалі, зважаючи на потребу в бекапах бази даних, перевірці даних у ній на цілісність, видалення застарілих та неактуальних даних або інші маніпуляції з ними, розвантаження HTTP-сервера, варто виокремити базу

даних для розгортання в окремому середовищі (іншому сервері або контейнері).

Для обробки складних запитів та таких, що не потребують миттєвої відповіді, варто використати асинхронні процеси, а тому відповідно задіяти черги завдань [2], брокери повідомлень та обробники черг. Таке завдання вирішується шляхом використання сторонніх сервісів обробки або рукописним рішенням та його розгортанням на окремому сервері. Функції брокера повідомлень може виконувати, наприклад, сервіс «RabbitMQ» або «Apache Kafka», «Apache Pulsar». Процесори (обробники) задач бажано також відокремити від основного веб-застосунку. Такий набір перелічених технологій використовується переважно для надсилання повідомлень зі звітністю про роботу системи, про реєстрацію користувачів у системі, відновлення паролів, так виконується розсилка за підпискою тощо; також це використовується для формування або обробки великих файлів, наприклад в сервісах обробки відео це можуть бути задачі з конвертування відео файлів, покращення їх якості тощо.

Після розвантаження сервера веб-застосунку (web application) вузьким місцем системи залишається база даних, котра є спільною навіть для кількох запущених екземплярів (instance) сервера веб-додатку. Щоб зменшити навантаження на неї, варто використати сервіси кешування та перехоплювати запити до бази даних.

Це має бути більш швидка база даних, тож серед таких – бази даних, що працюють в оперативній пам’яті, серед таких – NoSQL бази даних, зокрема часто використовується Redis. Для цього потрібний сервер з великою кількістю оперативної пам’яті та відповідне програмне забезпечення.

На додачу, дуже важливим для великих систем з точки зору користувачів є пошук в ній. Це можна забезпечити надавши системі можливість повнотекстового пошуку. Для невеликих систем він може бути забезпечений безпосередньо рушієм (engine) бази даних, однак це поганий варіант для навантаженої системи. Отож, варто використати пошуковий

двигун, серед таких – програмне забезпечення «Elastic Search», «Apache Solr», «Lucene», «Sphinx».

Таким чином, інфраструктура системи може набувати вигляду схеми [3] на рисунку 1.1. Програми застосунків можуть бути розгорнуті в кількох екземплярах, зокрема сервер додатків (1), який зазвичай містить мінімум логіки та обслуговує фронтенд; веб-сервіс, де зосереджена бізнес-логіка системи (2). Окремим класом компонентів виділено файлові сховища (3), куди належать бази даних, файловий хостинг, сервери потокової обробки даних.

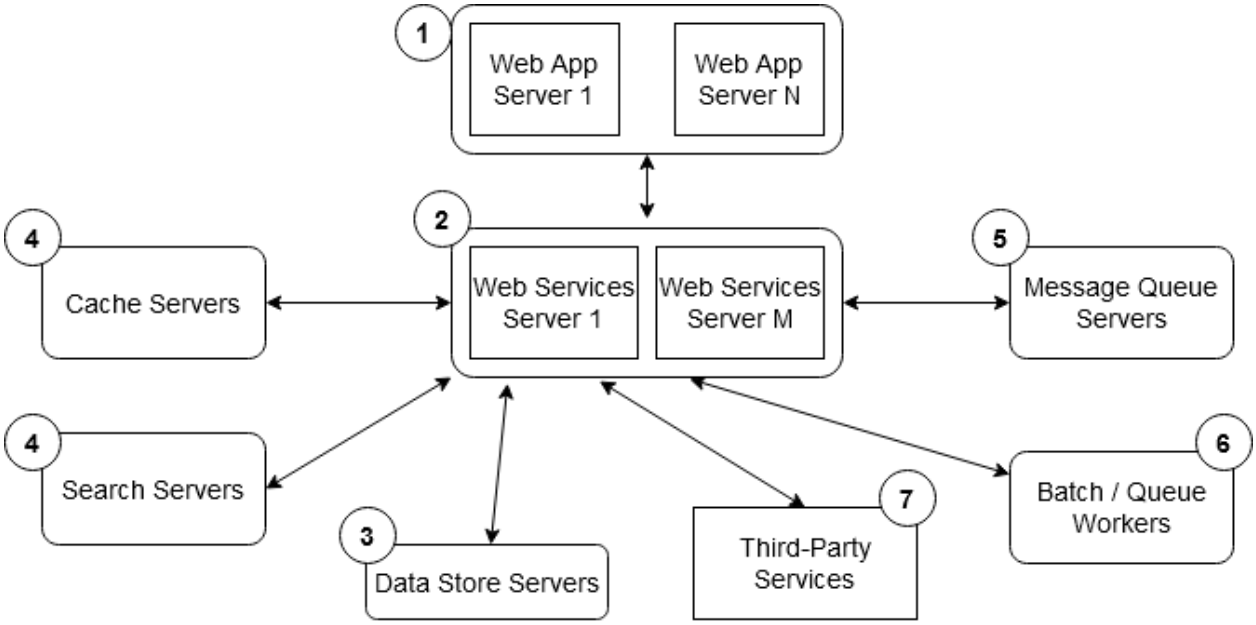

Рисунок 1.1 – Інфраструктура сучасної веб-орієнтованої ІС

Отже, будь-який сучасний веб-застосунок має містити наведені компоненти, причому бажано у відокремленому фізично вигляді.

1.2 Етапи розробки ПЗ

Створення будь-якої програмної системи без попередніх заготовок явно чи неявно складається з етапів розробки специфікації, проектування та реалізації, атестації та еволюції [1].

Процес розробки може базуватися на моделях каскадній, еволюційній, формальній чи інших [1], однак кожна з них міститиме вказані основні кроки.

Розробка специфікації зазвичай включає формування ідеї продукту, аналіз здійсненності, формування та аналіз вимог до системи та їх документування. У свою чергу формування й аналіз вимог може включати аналіз предметної області, у тому числі й розгляд подібних продуктів.

Етап проектування полягає у визначенні особливостей системи з технічної точки зору, зокрема відбувається окреслення основних сутностей системи, проектування компонентів та їх зв'язків, створення специфікації інтерфейсів, у тому числі макетів. На даному етапі виконується поділ системи на підсистеми та модулі, моделюються зв'язки між цими компонентами та управління ними.

Процес реалізації системи полягає в її кодуванні, причому робота може поділятися на завдання з визначеними часовими рамками їх виконання.

На етапі верифікації та атестації проводиться планування тестування, збірка системи, її аналіз та саме тестування на відповідність продукту вимогам. Процес еволюції передбачає підтримку продукту та нові ітерації його розробки з урахуванням змін у вимогах.

З урахуванням особливостей умов для створення даної системи була обрана за основу еволюційна модель розробки [1], у якій етапи специфікації, реалізації та атестації змішані та можуть проходити паралельно. Кінцеві вимоги до системи будуть зазначені у відповідному розділі.

1.3 Інструменти розробки ІС

З кожним роком складність систем збільшується, а тому процес розробки та підтримки застосунків стає важчим. У процесі розробки на всіх його етапах застосовують різного роду інструменти для спрощення та пришвидшення створення системи.

Серед переваг використання спеціалізованих засобів розробки: стандартизація, доступність, безпечність, контроль змін та контроль версій,

підтримка абсолютних та перехресних посилань, візуальне моделювання, управління конфігурацією, генерація документації тощо.

Відповідно до сфер інструменти можна використовувати для [4]:

а) бізнес-моделювання;

б) проектного менеджменту;

в) розробки вимог;

г) системного моделювання;

д) розробки ПЗ;

е) аналізу та керування тестуванням;

ж) реліз-менеджменту;

з) сервіс-менеджменту.

Більшість з інструментів надають функціонал для створення діаграм, таблиць, інтеграції їх даних у систему та забезпечують колаборативну роботу різних команд.

Для проектного менеджменту корисними можуть бути системи, що підтримують конкретні моделі розробки – skrum та canban-дошки («Jira», «Trello», «Asana», «Bitrix24»), тайм трекери різної складності та додатки для оформлення графіку робіт («Toggl Track», «Clockify», «Teamwork»).

Наразі для різних виробничих процесів компанії намагаються використовувати одну платформу, яка б забезпечувала потреби кожного з етапів розробки для кожної команди.

Особливої уваги для сучасних систем варті інструменти автоматизації тестування, збірки та розгортання проектів. Це забезпечують системи неперервної інтеграції CI/CD, наприклад «Jenkins» [5].

У межах даної роботи до більшості інструментів надає доступ програма «OpenServer» [6]. Вона містить графічні клієнт до БД: «phpAdminer» для MySQL та «PHPRedisAdmin» для Redis. Програма фактично надає адмін-панель для налаштувань сервера, дає можливість обирати версії мови PHP, HTTP-сервера, SQL та NoSQL баз даних, налаштовувати FTP, SMTP, та DNS.

Таким чином, «OpenServer» забезпечує усіма необхідними інструментами для реалізації програми.

Безпосередньо для розробки використано «git» та «GitHub» для забезпечення керуванням версіями; виявлення змін не потребується, оскільки мова є інтерпретованою; середовища PHPStrom та VisualCode для виявлення посилань, рефакторингу, забезпечення підсвітки синтаксису та автодоповнення.

Система на даному етапі не інтегрована з CI-сервісами, тому не використовує інструментів автоматичного тестування та збірки. Головним чином виконується тестування сценаріїв та API за допомогою браузера та програми «Postman» [7].

Інструменти кодогенерації, створення документації, виявлення залежностей надає сам фреймворк «Laravel» (інтерфейс «artisan»), пакетний менеджер «Composer» та самі пакети. Для відслідковування поточних задач використано дошку з сервісу «Trello» [8].

2 КОНЦЕПЦІЯ СИСТЕМИ

2.1 Загальний огляд та особливості системи

Система є веб-орієнтованою програмою для серверного ПЗ, яка агрегує інформацію про фільми. Надалі «All films» вживатиметься у якості назви даної системи. Система має надавати публічне API для клієнтських застосунків.

Структурно система належить до розподілених інформаційно-довідкових систем з клієнт-серверною архітектурою [9].

Програмна система передбачає 5 типів користувачів з відповідними правами:

а) гість (Г);

б) споживач (С);

в) адміністратор (А);

г) модератор (М);

д) контент-менеджер (КМ).

Система надає гостям та споживачам доступ до перегляду каталогу фільмів у системі, відповідну навігацію між ними, перегляд детального опису кожного фільму, його акторського складу, знімальної групи, перегляду рейтингу фільму, коментарів , залишених споживачами до нього. Гість також може зареєструватись у системі у якості споживача, а також може надіслати скаргу на споживача чи на сторінку відповідного фронтенду, що використовує дані. Зареєстрований користувач має «кабінет користувача», у якому може змінювати свої облікові дані, може залишати персональні нотатки до фільмів чи без прив’язки до них, наприклад, просто нагадування, може залишати коментарі до фільмів, оцінювати фільми.

Адміністратор має повний доступ до системи, окрім паролів та нотаток користувачів, не може без їх відома читати чи змінювати їх.

Контент-менеджер може змінювати інформацію, що стосується фільмів, акторського складу тощо. Система має модуль, що відповідає за парсинг стороннього сайту з метою отримання з нього інформації про фільми, цей процес працює самостійно без сторонніх налаштувань, його може запустити контент-менеджер.

Модератор може оцінювати правомірність дій користувачів у системі та впливати на них. Модератор може видалити коментар, заблокувати користувачу доступ до коментування на деякий час чи заблокувати обліковий запис повністю, може надіслати користувачу попередження. Також модератор розглядає скарги на користувачів, контент сторінок тощо та може або повідомити контент-менеджерів про необхідність коригування контенту, або розробників про несправність функціоналу системи.

Система в автоматичному режимі проводить збір інформації про оцінки користувачами фільмів та формує відповідні рейтинги. Система може бути розширена та містити додаткові атрибути фільмів.

Основний сценарій, виконання якого повинна забезпечувати система для споживача такий:

1) Споживач реєструється в системі.
2) Споживач підтверджує пошту.
3) Споживач входить у систему (login).
4) Споживач налаштовує персональні дані.
5) Споживач шукає фільми.
6) Споживач отримує рекомендації.
7) Споживач отримує деталі фільму.
8) Споживач ставить оцінку фільму.
9) Споживач залишає коментар фільму.
10) Споживач робить нотатки в системі.
11) Споживач виходить з системи.

У підрозділі 2.3 будуть сформовані користувацькі вимоги до системи.

2.2 Огляд аналогів

Серед систем, які реалізують схожу ідею, можна назвати багато як вітчизняних, так і закордонних аналогів. Оскільки мета даної системи – створення API, то здійснити порівняння можна тільки з точки зору функціоналу: з однієї сторони відштовхуючись від вимог, а з іншого – від наявних графічних інтерфейсів.

Популярними в користуванні є системи: прості каталоги фільмів, кіноафіші, стримінгові платформи (для трансляції кіно), онлайн-каси, фільм-орієнтовані файлообмінники або їх комбінації тощо.

Стримінговий сервіс «Netflix» дозволяє шукати та переглядати фільми, однак без реєстрації та оплати підписки неможливо отримати ніякі дані, окрім показаних на головній сторінці.

Найбільш популярний серед шукачів фільмів до перегляду є сервіс «Киного». Він пропонує зручний інтерфейс для пошуку та фільтрації фільмів, однак не надає персоналізованих рекомендацій, не відображає кількість оцінок за фільм та показує мало атрибутів об’єктів. Персональний кабінет не надає користувачеві ніяких цікавих можливостей, окрім встановлення оцінки фільму та його коментування.

Також мають чимало користувачів сервіси «Киноафиша», «ivi.ru», «kinonews.ru», «Кіноріум», «КіноБаза», «Кіно-театр.ua», «TMDB», «Cinemate». За підсумками проведеного аналізу, вони надають менше атрибутів фільмів, ніж пропонує система «All-films». «Кіноріум» надає багато графічних матеріалів, зокрема кадрів, трейлерів, відеорецензій, однак мало текстових. Система пропонує посилання на інші онлайн-кінотеатри – як платні, так і безкоштовні.

Популярний сервіс «Megogo» орієнтований саме на стримінг відео, є платним та взагалі не надає фільтрів при пошуку, що є незручним.

Менш відомими є рекомендаційні платформи «Джарвіс» та «movielens». Після реєстрації користувача вони надають можливість навігації

між фільмами, їх оцінювання, надають невелику кількість деталей, але саме про фільм, без додаткових статей чи деталей про акторів або інше.

Загальна тенденція систем описаного типу – це реалізація базового функціоналу з пошуку фільмів за невеликою кількістю атрибутів, тобто реалізація каталогу, та деяка спеціалізована логіка, зокрема для стримінгу чи продажу білетів, агрегації специфічної інформації, як-от рейтингів чи посилань на онлайн-кінотеатри.

Найбільш популярними серед схожих систем за кількістю користувачів та за наповненістю є два дуже схожі за функціоналом сайти – «IMDb» на англомовному ринку та «Кинопоиск» на російськомовному. Вони забезпечують добре структурований пошук кіно з різними фільтрами, надають деталі як про саме кіно, так і його атрибути, в тому числі кіностудії, склад знімальної групи, акторів, жанри; відображають новини індустрії; здатні надавати рекомендації до перегляду, містять рейтинги та надають можливості для оцінювання та коментування фільмів, створення різних списків, зокрема списків бажань (wishlist, watchlist) та вибраного, відслідковувати свою активність тощо.

Система «All films» також є гібридною, адже окрім каталогу, може надавати рекомендації фільмів до перегляду, надає можливості оцінювання та рецензування фільмів, створення персональних нотаток та нагадувань. У систему закладено можливості до збереження різного роду атрибутів та пошуку за ними. Додатково «All films» впроваджує контроль за контентом та поведінкою користувачів за допомогою модераторів.

Отже, Інтернет пропонує користувачам багато схожих систем. Основними критеріями при порівнянні можуть бути зручність пошуку в системі та кількість атрибутів кіно. У цілому ж, такі системи є досить динамічними, адже потребують постійної підтримки, а тому їх не варто порівнювати жорстко.

2.3 Вимоги до системи та варіанти її використання

Першим завданням на шляху створення системи є визначення її цілей, загальна характеристика концепції системи, формулювання вимог до неї, які й визначать її функціональність.

Наведемо уточнені користувацькі вимоги до API системи у таблиці 2.1.

Таблиця 2.1 – Функціональні користувацькі вимоги до API системи

Код	Опис вимоги	Користувач
К-1	Може зареєструватись у системі в якості споживача.	Г
К-2	Може змінювати свої облікові дані та пароль	С, А, М, КМ
К-3	Може відновити пароль до свого облікового запису в разі його втрати.	Усі
К-4	Здійснює пошук та фільтрацію фільмів за назвою, жанром, роком виготовлення, продюсером, режисером, акторами, рейтингом	Усі
К-5	Може створити персональну записку або нагадування до фільму. Доступ до перегляду записок має тільки автор записки. Може додати нагадування до записки.	С
К-6	Переглядає детальну інформацію про фільм.	Усі
К-7	Переглядає детальну інформацію про діяча кіноіндустрії (актора, продюсера, режисера тощо)	Усі
К-8	Може залишити коментар до фільму просто або у відповідь на інший коментар.	С

Продовження таблиці 2.1

Код	Опис вимоги	Користувач
К-9	Може видалити свій коментар.	С
К-10	Може поставити оцінку фільму за 10-бальною шкалою або показати свою реакцію за допомогою спеціальних позначок, наприклад, у вигляді смайлів «Emoji».	С
К-11	Може забрати змінити свою оцінку фільму або ж відмінити її.	С
К-12	Може змінити реакцію до фільму або відмінити її.	С
К-13	Може надіслати скаргу на контент сторінки з відповідним посиланням на сторінку та повідомленням.	Г, С
К-14	Може надіслати скаргу на «споживача» чи на його коментар.	Г, С
К-15	Розглядає скарги та реагує на них.	М
К-16	Може заблокувати обліковий запис «споживача» на деякий строк або назавжди.	М
К-17	Може заблокувати «споживачу» можливість залишати коментарі на деякий строк або назавжди.	М
К-18	Може розблокувати обліковий запис споживача та його можливість залишати коментарі.	М
К-19	Може відправити запит на перегляд блокування його можливостей в системі.	С
К-20	Може завантажити дані в систему з файлу.	КМ

Кінець таблиці 2.1

К-21	Додає та редагує інформацію про фільми.	КМ
К-22	Додає та редагує інформацію про діячів кіноіндустрії.	КМ
К-23	Може в автоматичному режимі запустити додавання фільмів в систему (запустити парсинг іншого сайту або імпортувати дані).	КМ
К-24	Додає та видаляє облікові записи контент-менеджерів, модераторів.	А

Відповідно до користувацьких вимог можна побудувати діаграми варіантів використання (use case) системи для наочного відображення її можливостей (рисунки 2.1, 2.2)

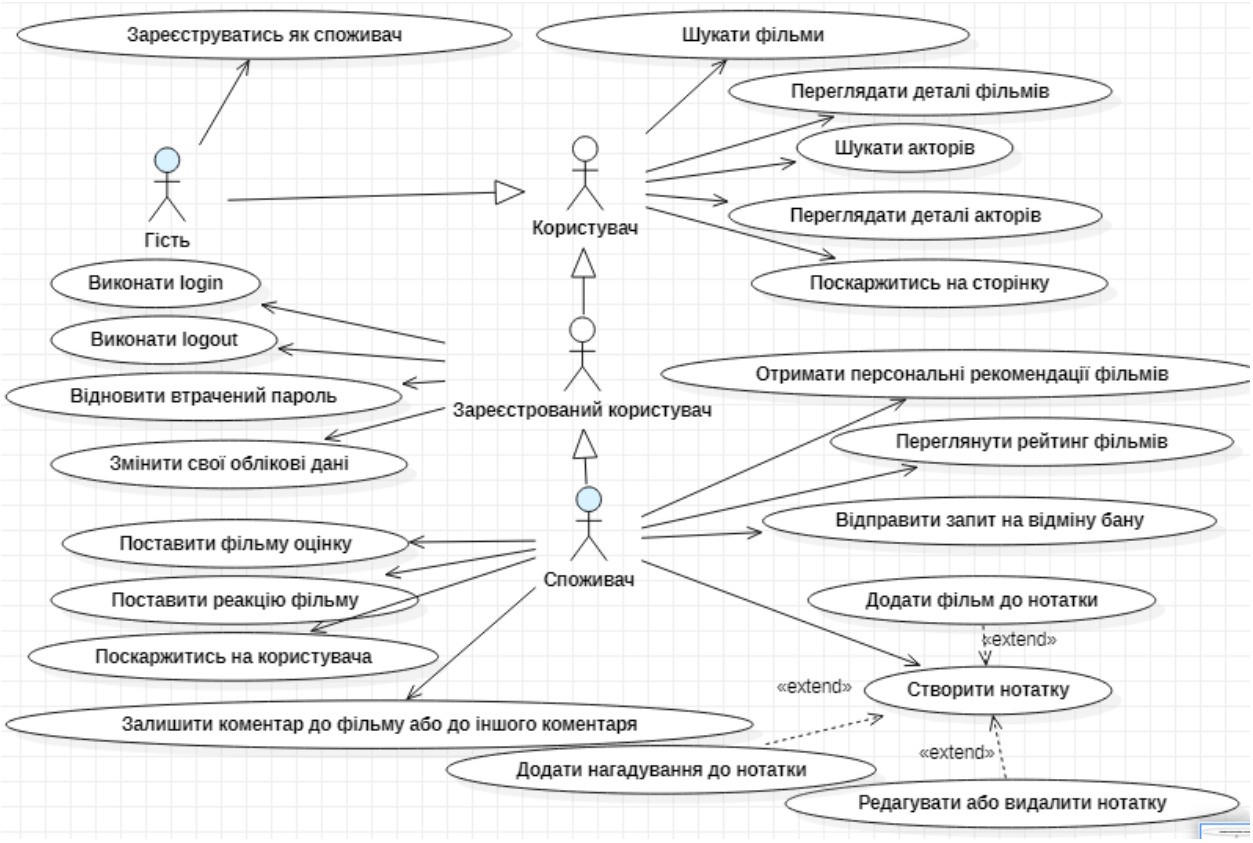

Рисунок 2.1 – Діаграма варіантів використання для гостя та споживача

Рисунок 2.2 – Діаграма варіантів використання системи для контент-менеджера, модератора та адміністратора

3 РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1 Архітектура системи

Проектування системи є важливим етапом у розробці програмної системи. Маючи наочне зображення архітектури, по-перше, новому розробнику можна відносно швидко зрозуміти використані абстракції та зрозуміти логіку компонентів системи; по-друге, організувати процес розробки більш точно в плані розбиття його на задачі, і по-третє, бачити взаємодію складових та недоліки системи.

За типом система відповідає моделі репозиторію [1]. Тобто підсистеми даної системи звертаються до спільного сховища даних для обміну інформацією. Як пише Соммервіл: «Більшість систем, що оброблюють великі об’єми даних, організовані навколо спільного сховища даних, або ж репозиторію» [1]. У даному випадку основним репозиторієм є база даних, однак не тільки, оскільки застосунок також використовує файлову систему хоста як сховище фотографій.

Архітектуру системи можна показати через схему бази даних (або ER-діаграму) та діаграми підсистем. Звичайно, у процесі розробки схема БД може змінюватись, а підсистеми доповнюватись, тому наведені діаграми можуть не збігатись з кінцевим результатом.

На рисунках 3.1, 3.2 можна бачити частини спрощеної ER-діаграми основної бази даних системи. Вони згенеровані з використанням функції

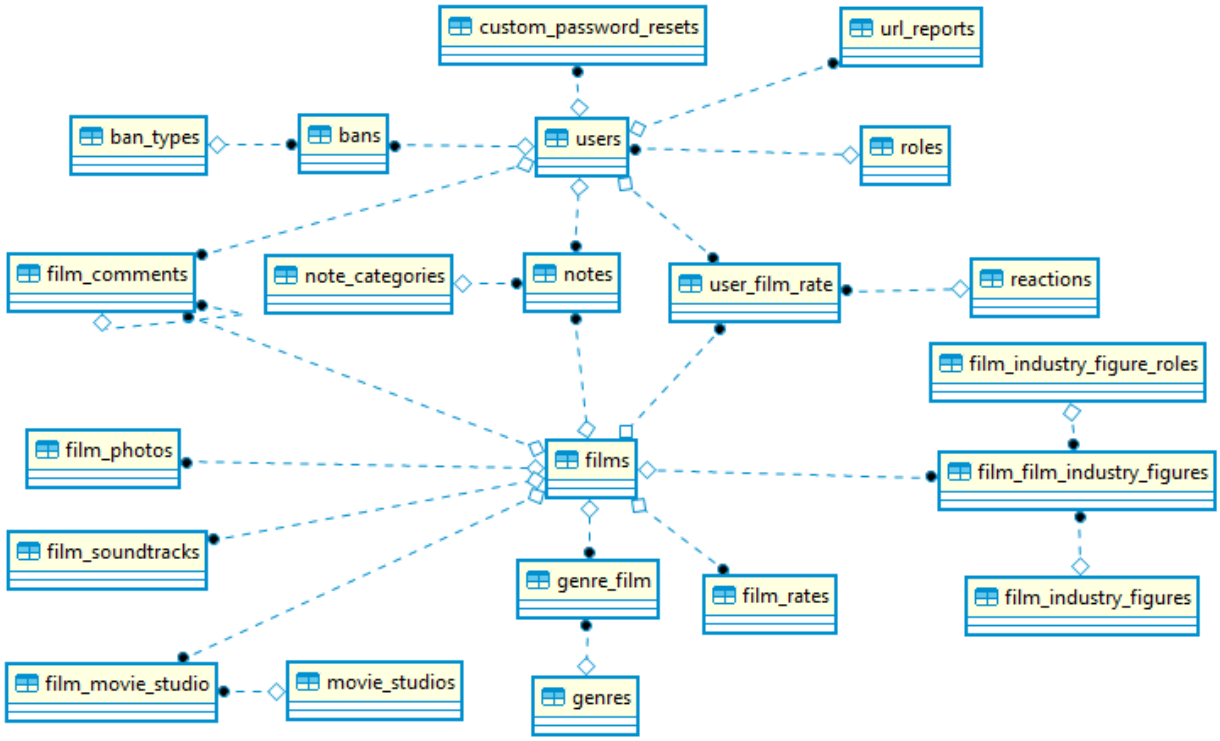

Рисунок 3.1 – ER-діаграма бази даних системи. Частина 1

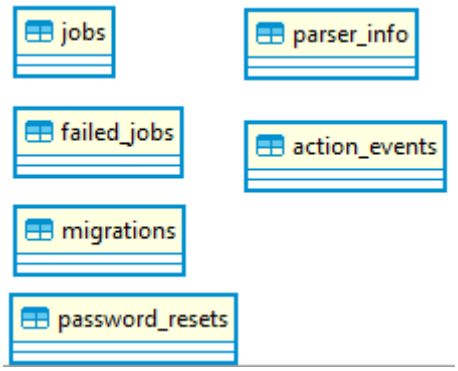

Рисунок 3.2 – ER-діаграма бази даних системи. Частина 2

клієнта для БД «DBeaver». Зв’язки агрегації відображають відношення належності, що встановлюється з використанням зовнішнього ключа.

З діаграм видно, що основними сутностями є фільм та користувач. Користувач пов'язаний з ролями, персональними нотатками, банами (bans), відновленнями пароля, скаргами, коментарями та рейтингом фільму. Фільм має зв’язки відповідно з учасниками зйомок, музичними композиціями, жанром, рейтингом та коментарями.

Інша частина діаграми (рисунок 2.2) містить незалежні сутності (кожну сутність асоціюємо з таблицею в БД). Діаграма показує 5 службових таблиць, створених фреймворком та пакетом «Laravel Nova» та 1 користувацьку («parser_info»). Таблиця «migrations» відповідає за міграції БД, «password_resets» – за створення токенів для відновлення пароля до облікового запису користувача, «jobs» та «failed_jobs» за заплановані завдання, які обслуговує queue-worker, «action_events» – таблиця «Laravel Nova», «parer_info» – за зберігання інформації для парсингу сайтів.

Міграції Laravel автоматично додають до кожної таблиці поля «created_at» та «updated_at», вони можуть знадобитися в процесі доробки та використання системи для відслідковування змін, внесених у дані, тому я залишаю їх. Більшість рядкових типів даних залишаємо довжини за замовчуванням, оскільки виявлення оптимальних розмірів полів є окремою проблематикою, а зміна типів у крайньому разі може бути досягнута написанням скрипта для переносу (відображення) БД на іншу схему.

Поділ системи на підсистеми відображено на рисунку 3.3.

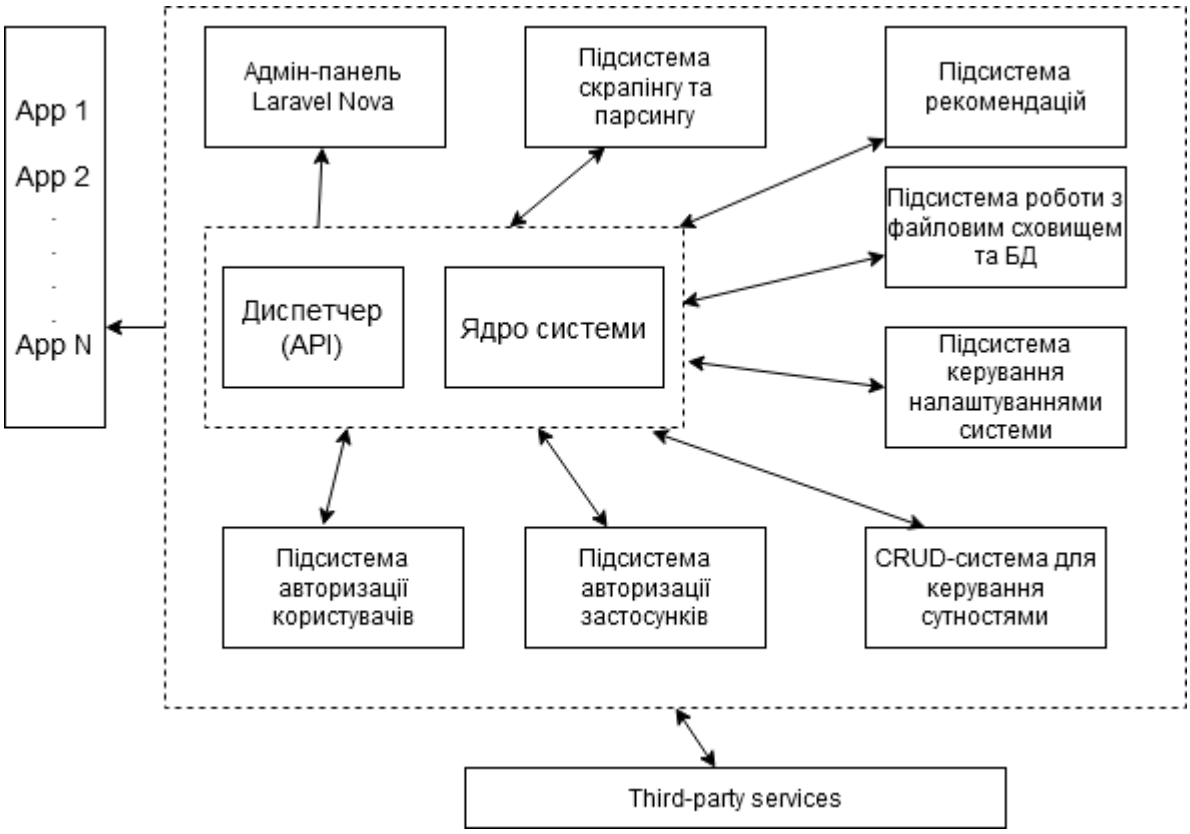

Рисунок 3.3 – Діаграма підсистем «All films»

Діаграма відображає основні програмні складові системи. Такими є диспетчер, який відповідає за маршрутизацію запитів, перенаправлення їх на відповідні контролери; ядро, яке відповідає за зчитування та кешування конфігурації, забезпечення роботи ORM, міграцій, впровадження залежностей, реєстрацією компонентів системи, подій та слухачів, каналів, автозавантаження пакетів тощо; підсистеми авторизації відповідають за організацію керуванням доступу до ресурсів системи; та інші наведені в діаграмі. Окремо винесено сторонні сервіси («third-party services»), оскільки вони не є частиною коду, що розробляється, а тільки надають відповідні інтерфейси та бібліотеки.

3.2 Фронтенд для адмін-панелі

Для адміністрування система потребує спеціального інтерфейсу з доступом до моделей та, можливо, окремих налаштувань. Такі інтерфейси носять назву «адмін-панелі».

Найбільшою популярністю користуються CRUD-панелі, тобто такі, що надають примітивний інтерфейс для створення, редагування, читання та видалення записів, причому зазвичай кожен запис асоційований з сутністю бази даних.

Для прискорення розробки в екосистемі «Laravel» [2] для цього часто використовують пакети, які надають стандартний графічний інтерфейс. Серед таких панелей можуть бути досить об’ємні рішення, які загромаджують код та які при рефактоорингу складно відокремити від бізнес-логіки застосунку, серед таких кодогенератори за заготовки проектів такі як «Voyager», «October CMS», «Craftable», «Backpack» та інші.

Також використовуються SPA-рішення, які є найбільш гнучкою альтернативою, однак вони сильно не прискорюють розробку.

У системі використано пакет «Laravel Nova» [10], який надає гарний CRUD-інтерфейс, не загромаджує БД, а створює усього 2 таблиці в ній; код,

що використовує адмін-панель зручно зберігається в одній директорії, а не розкиданий по усьому проекту. До того ж дану панель можна вбудувати в проект на будь-якому етапі його розробки. «Laravel Nova» дозволяє також змінювати фронтенд панелі, додавати користувацькі дії (actions), фільтри (filters, lenses), панелі індикаторів (dashboards), метрики, сортування та інше, налаштовувати різний доступ користувачам з різними ролями.

З використанням «Laravel Nova» у даній роботі створено інтерфейси для адміністратора, контент-менеджера та модератора. Пакет надає можливість локалізації інтерфесу, однак для більшої гнучкості використано мову за замовчуванням – англійську.

Для користувачів системи надано ресурсні панелі («resources», рисунки А.1, А.2), додано пошук користувачів, фільмів, акторів, коментарів та інших моделей за атрибутами, створено фільтри (рисунок А.3). Створено та розміщено на основному дашборді метрики (рисунок А.1) для демонстрації приросту контенту та користувачів системи, у тому числі метрики значень (value), трендів та розподілу (partition).

Також для відповідних ресурсів додано кнопки дій (actions), які передбачені у вимогах до системи, зокрема кнопки для завантаження контенту в систему, для запуску парсингу (рисунки А.4, А.5), кнопки для модератора для надсилання повідомлень споживачам та інші. Ресурси надають можливості для введення як текстових даних, так і для пошуку та вибору зв’язаних сутностей, зокрема й зв’язків «багато до багатьох» (рисунки А.6, А.7).

3.3 Автоматизація збору інформації та наповнення її контентом

Як вже було сказано, для додавання та редагування контенту системи використовується графічний інтерфейс адмін-панелі. Проте в деякій мірі це наповнення можна автоматизувати, використавши спеціальні інструменти. Це може бути імпортування підготованих даних з excel-таблиць, csv-файлів,

xml, json тощо. Такий підхід вимагає наявності вже підготованих даних. Наступний підхід – отримання даних з інших систем, однієї чи багатьох. При такому підході можливі два варіанти: звернення до інших систем з використанням наданого публічного API (безкоштовного чи на платній основі) або ж використання веб-скрейпінгу.

Використання підготованих даних розглядати не будемо, оскільки процедура досить проста: файл завантажується на сервер, де описана процедура парсингу файлу та вставки в базу даних. Процедура запускається, наприклад, обробником черги завдань. Також можна завантажувати sql-файли, але такий підхід має недолік, оскільки адміну чи контент-менеджеру фактично надається прямий доступ до бази даних, а ті самі адмін-панелі надають зазвичай більш абстраговані від низькорівневої архітектури системи інтерфейс.

Часто веб-системи або не надають стороннім застосункам API або надають його на платній основі, тому популярно для отримання даних, наприклад, погоди, цін на біржах, курсу валют, цін на товари в інтернет-магазинах, використовувати саме веб-скрейпінг [11]. По-суті, процес полягає в імітації входу користувача через браузер на сайт та пошуку потрібних елементів сторінки. HTTP-клієнт виконує GET запит на URL сторінки та отримує дані у вигляді html-розмітки сторінки. Після цього шляхом парсингу попередньо обраних елементів з розмітки виділяються потрібні дані: картинки, посилання, текстова інформація.

Отже, для виконання скрейпінгу необхідні:

1) HTTP-клієнт (бібліотека, пакет);
2) парсер (DOM-парсер) [11];
3) сайт-ціль;
4) сховище даних;
5) проксі-сервер [12].

У 5-му пункті сказано про проксі-сервер. Він потрібен, якщо клієнт виконує багато запитів до сайту-цілі. При дуже частих зверненнях до сайту,

його http-сервер чи інший сервер на шляху до системи (firewall, balance-loader) може виявити IP-адресу запиту, що надходить, та заблокувати його (додати в чорний список). Після цього запити з даної IP-адреси не будуть обслуговуватися і, як наслідок, парсинг не буде виконано.

Інша проблема, яку вирішує використання proxy-сервера – це блокування (деяких) сайтів деякими провайдерами, наприклад, в межах міста чи країни. Проксі-сервер маскує IP-адресу клієнта, з якого був надісланий запит, таким чином надаючи доступ до закритого ресурсу. Точніше кажучи, попередьо потрібно знати, чи відкритий доступ для IP-адрес, що використовуються проксі, до потрібного нам ресурсу. Цю інформацію можна дізнатися прямо у провайдера проксі. В залежності від затребуваних ресурсів та цілей використання власники проксі-серверів надають відповідні тарифи.

У моєму випадку я скористався безкоштовним проксі «Scraperapi» [13]. В подальшому в коді це не буде грати великої ролі, адже провайдера можна змінювати, а для конфігурації зазвичай потрібні лише домен проксі та приватний API-ключ. Ці дані можна змінювати у конфігураційному файлі проекту («.env»-файл).

При підготовці до парсингу потрібно дослідити структуру сайту (його верстку), та виявити елементи DOM-дерева, з яких дані будуть вибиратися парсером. Потрібно звернути також увагу на динамічні елементи сайту, час завантаження сторінок.

Саму задачу скрейпінгу бажано відокремити від усього застосунку, проте надати адміністраторам доступ до запуску цих задач та їх зупинки. При виконанні парсингу потрібно вирішити, чи дані будуть одразу зберігатись в основну БД, чи для пришвидшення зберігатимуться в іншій, можливо NoSQL базі даних, або взагалі будуть записуватися відразу в файл. Кращим підходом вважається другий варіант, а саме збереження в NoSQL БД, зокрема у «MongoDB».

Великою проблемою збору контенту шляхом скрейпінгу є його динамічність. Тобто потрібно наперед вирішити, чи виконувати його один

раз, а іншу роботу покладати на контент-менеджерів, чи час від часу оновлювати інформацію, коли вона оновлюється в джерелі (сайті-цілі).

У даній роботі запропоновано перший варіант, тобто скрейпінг виконується одноразово для спрощення роботи контент-менеджерів. Запит на виконання збору інформації про кілька фільмів помістимо в чергу, а сам скрейпінг виконуватиме обробник черги.

На цьому етапі робота не потребує великої продуктивності, тому отримані дані після відповідної обробки відразу зберігатимуться в основну БД.

При скрейпінгу названим вище чином виникають також чотири проблеми, пов’язані з контентом сторінки: проблема поганого (poor) HTML, проблема розподіленості наповнення сторінки, нездатність HTTP-клієнта до інтерпретації JavaScript; асинхронність завантаження, ліниве завантаження тощо [14].

Перша проблема полягає у тому, що HTML-розмітка сторінки, яка досліджується, може бути написана з помилками, тобто мати незакриті теги, неправильну вкладеність елементів, помилки в написанні атрибутів, незакриті лапки. Це є проблемою для парсера, який намагається обійти DOM-модель. Цю проблему звичайний користувач може не помічати, адже більшість браузерів автоматично виправляють розмітку, а отже генерують коректний для відображення HTML-документ.

Друга проблема пов’язана з тим, що звичайний http-клієнт, який використовується при скрейпінгу, робить тільки один GET-запит, на отримання HTML-документа сторінки, та автоматично не завантажує частини розмітки, що визначені через посилання. Таким чином зазвичай вставляються скрипти, таблиці стилів, зображення. Перші два зазвичай не грають суттєвої ролі при розборі сторінки, а для завантаження зображень, доведеться робити окремі запити.

Третьою проблемою на шляху скрейпінгу є SPA-застосунки. Спрощено, структура отримуваного HTML-документу містить один

кастомний тег або атрибут, наприклад , у який повинен динамінчо рендеритися контент за допомогою JavaScript, та підключення js-файлу, так званого bundle-файлу. JavaScript файл після завантаження його браузером знаходить той самий тег та будує в ньому DOM-модель застосунку. Цього не відбувається при скрейпінгу, адже js-файл не завантажується та не інтерпретується, на відміну від браузера, який є середовищем виконання програмного коду на JavaScript.

Четверта проблема – це асинхронний контент уцілому. Зазвичай, це називають лінивим завантаженням (lazy loading), коли при прокручуванні сторінки кількість товарів на сторінці зростає, тобто фактично відбуваються виклики до бекенду на отримання інформації про ці товари. Звичайний http-клієнт не може імітувати таку поведінку, тому будуть завантажені тільки перші товари.

Останні дві проблеми можна вирішити лише за допомогою драйвера браузера, тобто деякого виконуваного коду, який імітує поведінку браузера та надає інтерфейси для керування імітованою поведінкою користувача в ньому. Найбільш відомим з таких надбудов є Selenium, проте він використовується у більшій мірі для тестування. Для ж Laravel можна використати пакет «Panter». Підхід з використанням драйвера у цілому вимагає тонкого налаштування.

У даній роботі як джерело інформації використано сайт «КиноПоиск» (kinopoisk.ru), який очевидно є SSR, тобто при виконанні GET-запиту клієнт отримує вже відмальовану (rendered) сторінку, тому робота не потребує використання додаткових драйверів.

3.4 Показники системи

Будь-яка інформаційна система генерує звіти з деякими статистичними даними або використовує узагальнюючі показники у своїй роботі.

Проведення аналізу даних для великої вибірки може виявитися часозатратною задачею, тому її варто оптимізувати. Зокрема підтримувати у БД таблиці хоча б з частковою інформацією, особливо коли запити на отримання статистики надходять часто.

Дана система, наприклад, повинна відображати рейтинг фільму,тобто його середню оцінку за всіма користувачами, причому очікується, що цей запит буде не одиничним. Для підрахунку оцінки фільму, потрібно як мінімум продивитись всю таблицю з користувацькими оцінками фільмів лінійно. Якщо б ця дія виконувалась, наприклад, раз у день, то зробити це можна було б звичайним запитом, однак якщо в систему надходять кілька таких запитів в секунду, час відповіді стрімко зростає.

Отож, для рейтингу виділимо спеціальну таблицю, а збір результатів проводитимемо раз у день, тобто оновлюватимемо таблицю раз у день, або за потребою адміністратора.

Для підрахунку рейтингу кожного фільму використана формула [15]:

$$W = \frac{R * v + C * m}{v + m},$$

де W – показник рейтингу; R – середня оцінка за даний фільм; v – кількість оцінок за фільм; m – мінімальна кількість оцінок для включення до рейтингу; C – середня оцінка у системі (по всіх фільмах).

Система також будує метрики трендів, значень та розподілу користувачів та фільмів за допомогою пакету «Laravel Nova». Це надає можливість спостерігати тенденції користувачів системи. Частина метрик відображена на рисунку А.1.

3.5 Оптимізація роботи проекту

Для будь-якої інформаційної системи швидкість роботи є критичною. Особливо для веб-орієнтованих систем, які орієнтовані на зовнішнього споживача та великі навантаження.

Швидкість роботи впливає на користувацький досвід (UX), який визначає чи буде система сприйнята користувачем та мати популярність, що в даному випадку важливо. Сучасний показник часу завантаження сторінки повинен бути не більшим за 1.5 с. Це також означає, що для більшості простих запитів, відповідь повинна бути отримана протягом цього часу. Для цього необхідно правильно розробити архітектуру проекту, а також оптимізувати роботу системи вже після запуску.

Можна розглядати кілька рівнів оптимізації: 1) рівня мови, 2) рівня фреймворку, 3) рівня коду розробника , 4) оптимізацію на рівні заліза та програмного забезпечення, на якому запущений проект.

Для розробника може бути невідомою кінцева платформа (середовище) розгортання, тому в першу чергу важливо звернути увагу на оптимізацію на етапі проектування системи та написання коду.

Головним чином оптимізація виконується шляхом зменшення розмірів файлів та перенесенням їх у швидку пам’ять – таким чином, кешування є основною процедурою оптимізації.

В Laravel при написанні коду виділимо такі позиції для оптимізації:

а) проблема N+1 запиту;
б) уникнення використання ORM за можливості;
в) кешування конфігурації та маршрутів;
г) більше кешування, використання резидентних БД (in-memory);
д) оптимізація автозавантажувача;
е) зменшення сервісів автозавантаження та посередників (middleware) [16];
ж) додавання асинхронної обробки (черг завдань);
з) оптимізація зображень, файлів, допоміжних файлів (assets);
и) використання CDN.

Проблема N+1 запиту полягає в тому, що при отриманні деякої вибірки з однієї таблиці БД робиться 1 запит, а при ітерації вибірки, якщо ми захочемо отримати дані зв’язані деяким відношенням з кожним записом

отриманої вибірки, то буде виконано ще n запитів до бази даних. Ця задача при використанні ORM (Eloquent ORM) вирішується шляхом переходу до eager loading (коли одним запитом з БД разом з сутністю з однієї таблиці отримуються також зв’язані з нею сутності з інших таблиць) на противагу lazy loading.

В цілому ж на великій кількості даних будемо уникати використання ORM та користуватися «сирими» (raw) запитами. Конструювання об’єктів моделей, займає час та затримує звернення до бази даних або ж навпаки повернення результатів користувачеві. Важливо зауважити, що така оптимізація коштуватиме наочності коду, тому її варто використовувати тільки на великих вибірках.

Для зменшення кроків завантаження фреймворком конфігураційних файлів та файлів маршрутів варто використати командний інтерфейс artisan: виконати команди «php artisan route:cache» та «php artisan config:cache» для генерації кешів з відповідних файлів.

Варто використати кешування на всіх можливих рівнях системи. В тому числі кешувати запити до бази даних. У моєму випадку, наприклад, використаємо кешування для запитів на популярні фільми. Очевидно, що таких запитів за визначенням буде виконуватись більше ніж інші. Тому система заздалегідь буде тримати значення вибірки популярних фільмів, а не звертатись кожного разу до БД.

Зауважимо, що кеші, засновані на файловому сховищі, є повільними, а найбільш швидкою альтернативою в цьому випадку слугують резидентні БД, зокрема NoSQL бази даних. Я використаю Redis.

При запуску проекту застосуємо команду пакетного менеджера «Composer» [17]: «composer install --optimize-autoloader --no-dev». Вона завантажить тільки пакети потрібні для безпосередньої роботи застосунку, пропустивши пакети, що використовувались виключно для цілей розробки та згенерує карту класів (classmap) для більш швидкого виявлення

розташування php-класів у файловій системі, без обходження директорій, як це робиться без використання оптимізації автозавантажувача.

Для великих задач, таких як надсилання листів, форматування зображень або інших, що потребують довгих ланцюгів запитів, складних запитів до БД застосуємо асинхронну обробку. Реалізуємо це за допомогою асинхронних черг завдань, які будуть складатися в базу даних та оброблюватись не в основному потоці (flow) застосунку, а окремим процесом, концептуально названим «queue worker»-ом. Такий механізм забезпечується самим фреймворком, однак він реалізує концепцію брокера повідомлень (message broker), а тому для реалізації черг можна використати один з відомих брокерів RabbitMQ, а реалізацію обробників винести в окремий сервіс з каркасом на Laravel.

Для прискорення роботи системи при розгортанні також можна буде запустити проект в контейнерах в кількох екземплярах. Тоді вузьким місцем системи стане СУБД. Оптимізація роботи СУБД потребує тонкого її налаштування та не буде розглядатись в межах роботи. Варто також зазначити, що зменшити навантаження допоможуть техніки вертикального та горизонтального масштабування бази даних.

В даному проекті розглядається загалом розробка бекенду, проте можна зазначити й про оптимізацію фронтенду. Вона полягає у зменшенні часу доставки контенту (завантаження сторінки) – у загальному випадку це досягається шляхом розвантаження http-сервера. Це може бути досягнуто при зменшенні розмірів файлів розмітки, стилів, скриптів та додатків. Для цього варто використати мініфікатори та зв’язувачі js-файлів, зменшені зображення, мініфіковані та зв’язані файли стилів, замість підключення локальних бібліотек користуватися CDN.

3.6 Відкладені завдання та черги

Для прискорення відповідей застосунку на користувацькі запити та розвантаження сервера ресурсно та часозатратні задачі можна виконувати асинхронно. В Laravel це реалізується використанням черг (queues) та задач (jobs).

Дана система використовує їх для надсилання листа підтвердження реєстрації в системі, при надсиланні повідомлень (сповіщень) модераторами чи системою, для скрейпінгу сайту та парсингу файлів з даними (при завантаженні), для формування метрик, зокрема рейтингів фільмів.

Система на час розробки використовує у якості драйвера черг [2] базу даних («database»). «Laravel» також підтримує драйвери для роботи з «Redis» та «Amazon SQS». Підхід з використанням БД створює таблицю для запису завдань в неї, причому завдання класифіковані за назвами черг. Обробником завдань слугує процес, який пропонує сам фреймворк (запуск «php artisan queue:work»). Обробник черг працює незалежно від циклу обробки запитів та може бути запущений як демон для окремих черг. Після виконання завдання, воно видаляється з бази даних, причому при запуску обробник конфігурується кількістю спроб виконати завдання (tries) та часом його виконання (timeout).

Система надає інтерфейс для планування скрейпінгу на певний час або проведення його з вказаною частотою та інтерфейс для налаштовування частоти виконання бекапів. Хоча зазвичай для відкладених завдань використовувались cron-завдання, у даній системі за це відповідає планувальник «Laravel» [18].

За допогою планувальника зокрема проводиться підрахунок рейтингів фільмів та виконання бекапів. При жорсткому кодуванні планування операції резервного копіювання виглядає так (рисунок 3.4).

```
protected function schedule(Schedule $schedule)
{
    $schedule->exec('php artisan backup:run')->dailyAt('03:30')->runInBackground();
}
```

Рисунок 3.4 – Запланована команда резервного копіювання

Фреймворк надає можливості до планування завдань та команд «artisan» або shell-команд та запуску їх обробників. Важливо пам’ятати про запуск обробника планувальника, у локальному середовищі це робиться командою «php artisan schedule:work».

3.7 Сповіщення системи та push-повідомлення

Для забезпечення хорошого користувацього досвіду інформаційна система має забезпечувати real-time повідомлення. Для цього використовується push-технологія, або ж так звані пуш-повідомлення.

Для забезпечення роботи цієї технології сервер повинен постійно підтримувати зв’язок через сокети з клієнтами, що знаходяться у мережі. Така технологія вимагає від сервера багатьох ресурсів та особливого підходу для підтримки сокетів.

Тому не варто реалізовувати пуш-повідомлення на головному сервері застосунку. Для цього бажано використати окремий оптимізований для цього веб-застосунок, що реалізує даний функціонал.

Я обираю хмарне рішення, сервіс «Pusher» («Pusher Channels») [19]. Він дозволяє легко інтегруватися з різними фронтендовими та бекендовими фреймворками.

Після реєстрації на сервісі обираємо якою мовою програмування або з використанням якого фреймворку написана програма, що забезпечує бекенд, а також окремо фронтенд. Сервіс генерує два скрипти, які потрібно влаштувати в потрібні місця програми та надає підказку, які пакети необхідно додатково встановити.

Сервіс використовує концепцію каналів (channels) та подій (event) та фактично реалізує шаблон проектування «публікація-підписка»; архітектурно ж сервіс є вузько спеціалізованим брокером повідомлень.

На серверній частині ініціюється подія, наприклад «BanReviewed», з нею зв’язуються деякі дані, зокрема ідентифікатор користувача, для якого призначене повідомлення, та назва каналу. Ця подія у вигляді повідомлення відправляється звичайним http-запитом на сервіс Pusher, де зберігається в базі даних. Якщо користувач онлайн, то він отримує повідомлення, а з бази даних у сервісі воно видаляється.

Для того, щоб сервіс знав чи в мережі користувач та які події йому потрібно відправляти, в коді фронтенду розробник має забезпечити логіку виконання скрипта підписки після авторизації користувача (відправка в Pusher запиту). Підписка виконується на конкретний канал та конкретну подію. Фактично, часто один канал асоційований з одним користувачем. Таким чином, на фронтенді користувач отримує «через канали» деякі події, на які встановлюються відповідні обробники.

Після реєстрації у сервісі Pusher було отримано ключі та скрипт для бекенду. Для застосунку, який буде отримувати сповіщення, генеровані системою, можна також отримати заготовки під відповідну мову чи фреймворк (рисунок 3.5).

```php
class MyEvent implements ShouldBroadcast
{
  use Dispatchable, InteractsWithSockets, SerializesModels;

  public $message;

  public function __construct($message)
  {
      $this->message = $message;
  }

  public function broadcastOn()
  {
      return ['my-channel'];
  }

  public function broadcastAs()
  {
      return 'my-event';
  }
}
```

Рисунок 3.5 – Шаблон скрипта генерації push-повідомлень від «Pusher» для «Laravel»

Наразі система генерує сповіщення для споживачів при відповідях на їх коментарі, при діях модераторів відносно їх облікових записів або при нагадуваннях, які встановлюються в нотатках.

3.8 Пошуковий двигун

Як вже було сказано, для повноцінної інформаційної системи необхідний пошуковий двигун (search engine). Звичайна СУБД буде оброблювати пошукові запити набагато довше та вимагатиме тонкого налаштування, яке або не буде відображене в коді, або заплутає його, після чого проект стане складним для підтримки.

Пошукові двигуни можна налаштовувати локально, або використати готове хмарне рішення, що я і зробив в рамках проекту.

«Laravel» має хорошу інтеграцію з сервісом «Algolia» [20] та надає для цього інтерфейс (фасад) «Laravel Scout» з відповідними консольними командами. «Algolia» попонує повтотекстовий пошук за моделями та надає веб-інтерфейс для відслідковування та налаштування індексів.

У коді моделей для забезпечення роботи індексу додаємо трейт [20, 21] Searchable (рисунок 3.6). Після чого для кожної такої моделі визначено назву індексу та можна виконати імпорт моделей. Будь-які зміни в моделях автоматично змінюють індекси. Це можна робити синхронно та асинхронно. Для більшої продуктивності налаштовуємо черги задач та користуємося перевагами асинхронної обробки запитів на зміну індексу.

Рисунок 3.6 – Фрагмент коду з визначення моделі фільму «Film»

Надалі для пошуку моделей запити надсилаємо вже не до бази даних, а до «Algolia», використовуючи метод «search()».

Важливо пам’ятати про синхронізацію індексу: по-перше, час від часу перебудовувати його (використовуючи команди фасаду «Scout» та налаштувати автоматичну перебудову індексу); по-друге, знати, що функція зворотнього виклику (callback) оновлення індексу спрацьовує тільки при роботі з моделями, а при сирих запитах індекс буде залишатись

неактуальним. Варто зазначити, що індекс та дані в самій базі даних можуть бути не цілісними і це допустимо для некритичних даних.

У моїй системі індекси використано для пошуку фільмів та акторів, оскільки користувачі будуть часто їх запитувати та шукати за входженням слів у назву або інше поля об’єкта (атрибути), що також налаштовується в інтерфейсі «Algolia» (рисунок 3.7). З рисунку видно, що при пошуку враховуються можливі помилки введеного рядка, та що в індексі знайдено один запис, що підходить до запиту. Надалі важливо правильно налаштувати сервіс, зокрема сприйняття помилок та пошукові атрибути моделей (рисунок 3.8).

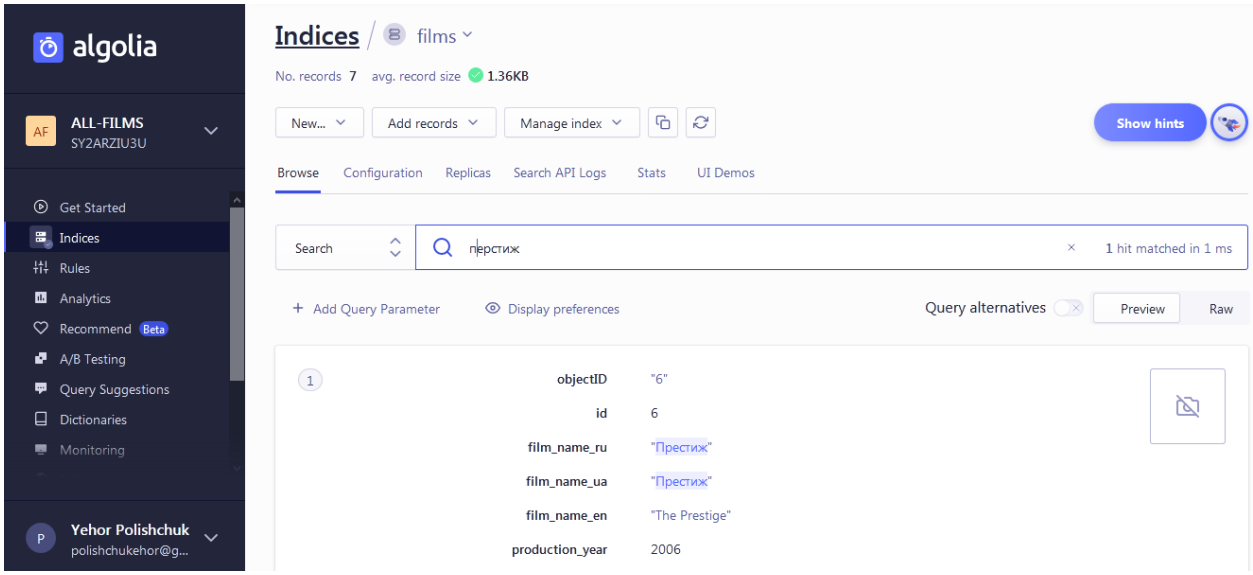

Рисунок 3.7 – Інтерфейс «Algolia». Індекс фільмів. Пошук з помилкою

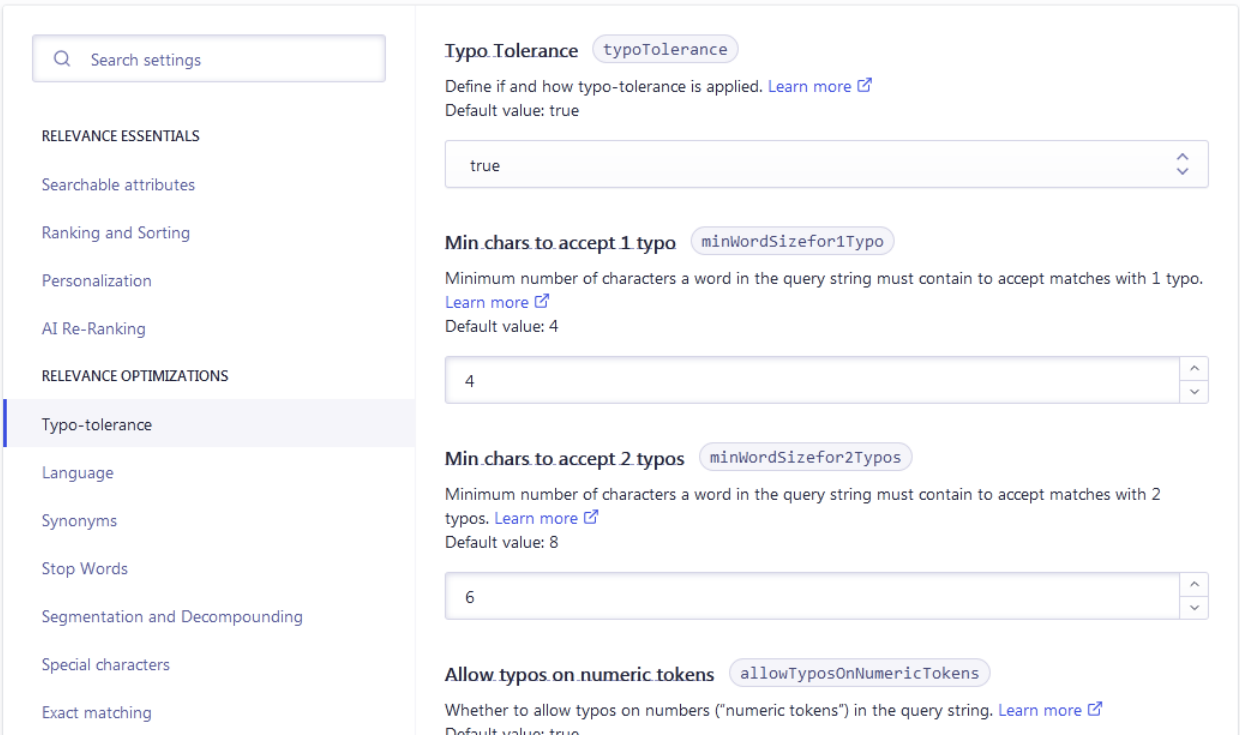

Рисунок 3.8 – Розділи конфігурації пошукового сервісу «Algolia». Толерантність до помилок вводу

3.9 Безпека та цілісність системи

Говорячи про безпеку потрібно розуміти безпеку роботи системи, тобто її стійкість, надійність та процеси доступу до даних системи, а тому процеси аутентифікації та авторизації дій користувачів в ній.

Важливо також забезпечувати цілісність даних у системі та їх відновлюваність, що ускладнюється в розподілених системах.

Для відновлення даних у разі втрати БД дана система використовує пакет «spatie/laravel-backup». Він використаний у сукупності з запланованими завданнями. Бекап виконується автоматично кожний визначений проміжок часу, який можна змінювати. У моєму випадку це 1 день. Також після виконання бекапу адміністратор системи отримує повідомлення на пошту або в Slack з його деталями. Пакет можна налаштувати для збереження резервної копії на інший диск (не тільки на локальний). Потрібно завжди перевіряти якість бекапів, тобто чи можна з них й справді відновити базу

даних. Досвідчені програмісти не покладаються на випадок, а обов’язково тестують бекапи баз даних.

Так як застосунок, фактично, є монолітним і не містить розподіленої бази даних, авторизацію можна виконувати на місці з використанням цієї ж бази даних. Я реалізував просту аутентифікацію за допомогою токена типу «Bearer» [22]. Вона працює таким чином, що при запиті на «вхід» в обліковий запис, користувачу виділяється унікальний токен. Токен не містить ніякої додаткової інформації. Він є аналогом логіну та паролю. Фронтенд отримує токен та зберігає, наприклад, у localStorage [22], надалі при кожному запиті до бекенду фронтенд додає цей токен у поле заголовку авторизації: Authorization: Bearer <token>. Токен однозначно визначає користувача та його статус авторизації в системі. Такий підхід аналогічний авторизації Basic [22], але там користувач надсилає логін та пароль кожного разу, а тому така система має більше поле для атак. При використанні токена зміст пароля та логіну не розкривається.

Для генерації відповідних токенів та виконання процесів перевірки авторизованості користувача система використовує пакет «laravel/sanctum», який реалізує описану вище логіку.

3.10 Доступ до системи сторонніх застосунків

Відповідно до опису система має надавати доступ до API стороннім застосункам. Для цього було розроблене публічне API. Однак це вважається небезпечним, тому варто обмежити можливість до віддалених викликів програмами функцій системи.

Основними проблемами є неавторизовані дії в системі та DDoS-атаки. Другий пункт налаштовується не в межах програми, а на рівні конфігурації сервера застосунку (або файрволу). Говорячи про авторизацію можна розглядати як агента, тобто як кінцевого користувача, людину або ж інший додаток.

Головним чином для безпеки системи, запити які модифікують дані, можуть робити тільки авторизовані користувачі, що пояснено у попередньому розділі.

Іншим рівнем захисту є робота посередників (middleware) фреймворку, де за допомогою перевірки CORS-заголовків запиту обмежується набір доменів, з яких запити будуть оброблюватись. Таким чином, для використання API на своєму сайті, його власник звертається до власника системи для оформлення реєстрації домену як дозволеного для викликів.

Для розширення чи співпраці системи з іншою в майбутньому можливо буде потрібно надати доступ іншому модулю системи як цієї, працюючи за http-протоколом. Тобто інший модуль може робити виклики API без ініціативи людини та її авторизаційних даних, але фактично з тими ж правами. Для цього додаток реєструється у якості клієнта системи. Для цього можна використовувати API-ключі, однак у даній системі «сторонній» додаток реєструється як звичайний користувач системи та використовує для авторизації bearer-токен, додаючи його в заголовках запитів.

3.11 Документування системи

Будь-яка веб-орієнтована інформаційна система вимагає підтримки. В іншому разі в системі можуть почати виникати помилки різного роду: порушитися цілісність даних, втрата даних, сторонній доступ до системи, перевантаженість серверів зумовлює зупинку роботи застосунку або велику затримку у відповідях на запити. Також у час стрімкого розвиту IT-індустрії система швидко втрачає актуальність, її ресурсів починає не вистачати, а схема бази даних не відповідати потребам даних. Тому робоча система потребує постійного моніторингу, аналізу, рефакторингу, масштабування тощо.

Час, витрачений, на виправлення помилок може бути критичним або ж ні, однак розуміння роботи системи принаймні розробником є життєво-

необхідним для неї. Тому модулі системи, її концепція, авторизаційні дані (credentials), API мають бути обов'язково задокументовані.

Система, що розробляється, орієнтована на використання сторонніми застосунками, тому ключовою є наявність задокументованого API.

Це можна робити вручну, однак при розростанні проекту відслідковування змін з часом стає важким, тому документація може ставати неактуально. Автоматизація документування системи є бажаною.

У даній системі було використано пакет «knuckleswtf/scribe». Генерація виконується командою «php artisan scribe:generate». На виході отримуємо HTML-файл (рисунок 3.9). Для генерації документації пакет використовує докблоки контролерів, файл маршрутів та за потреби файли посередників (middleware). Докблоки мають відповідати правилам документування, що використовуються пакетом [23]. Пакет дозволяє описати статуси та тіло відповідей, параметри та тіло запиту, згенерувати приклади та колекції запитів, групувати ендпойнти та давати їм опис, що і було виконано.

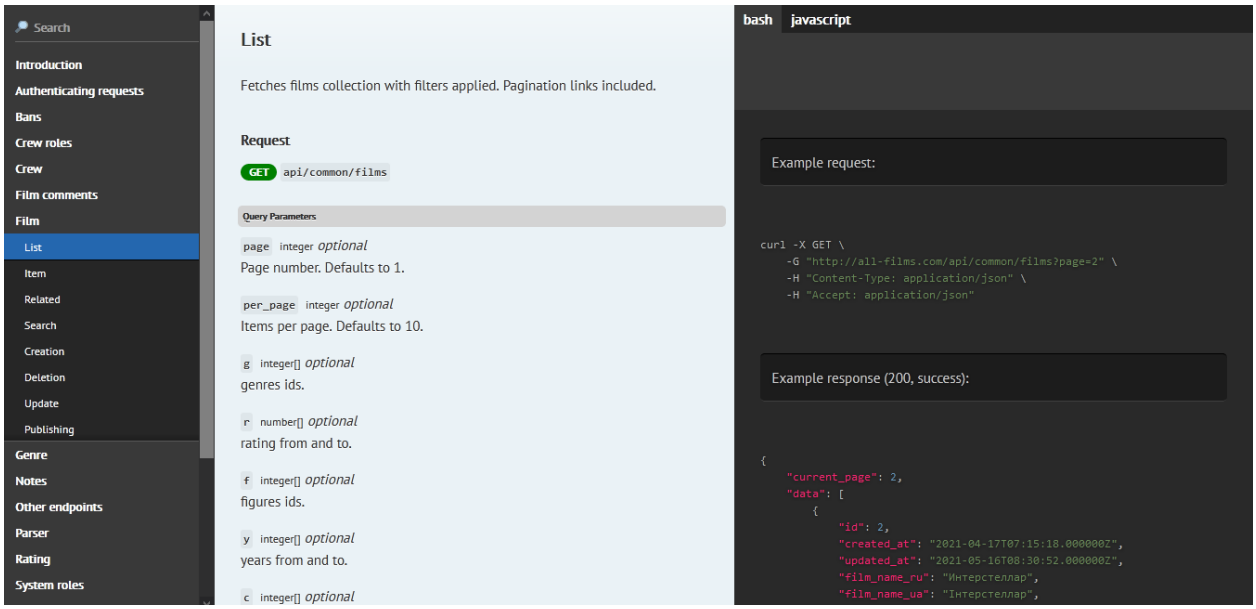

Рисунок 3.9 – Згенерована документація API системи «All films»

3.12 Рекомендаційна підсистема

Видача користувачам рекомендацій може бути зручною як для користувачів, так і для власників системи. Зокрема, це використовується на платформах з продажу товарів чи послуг з метою запропонувати потенційному покупцю можливо кращий товар або додатково зацікавити його у покупці іншого. За це відповідає рекомендаційна система.

Вона будується на даних, що відповідають об’єктам зацікавленості користувача. У цілому початкові дані генеруються активністю користувача: його переходами за посиланнями для переглядів деяких одиниць сайту, його позначок про вподобання, коментарів, інших оцінок.

Такі системи можуть реалізовуватися як сторонніми сервісами («Recombee») [24], так і бути частиною основної системи. Кінцеві дані будуються на основі деякої аналітики, яка може генеруватися самою системою або ж сторонніми сервісами («Яндекс.Метрика», «Facebook Pixel», «Google Analytics»).

Рекомендаційні системи бувають двох видів: на основі контенту та на основі колаборативної фільтрації. Використаємо сервіс «Recombee», який імплементує перший («content-based») підхід.

«Recombee» оперує поняттями одиниць (items), користувачів (users) та взаємодій (interactions). На основі взаємодій система здатна будувати рекомендації з одиниць для користувача або іншого товару (зазвичай секція сайту «Related items»). Взаємодії класифіковані як:

а) перегляд деталей (detail views);
б) додавання до кошику (cart additions);
в) оцінки (ratings);
г) закладинки (bookmarks);
д) покупки (purchases).

Кожний тип взаємодії можна співставити з наявними у системі, але це не є обов'язковим. Зокрема, в «All films» наразі зіставлено перегляд деталей фільму та виставлення йому оцінки.

Система підтримує три моделі роботи. Перший варіант – це інтеграція на стороні сервера, коли фронтенд напряму не звертається до сервісу, а логіка взаємодії з ним зосереджена на сервері застосунку. Інших два варіанти інтегруються і на фронтенді, і на бекенді, в цих моделях клієнтський застосунок звертається безпосередньо до «Recombee». У реалізації «All films» використано перший варіант.

Конфігурувати рекомендаційну систему можна як з коду через API-виклики, так і через графічний інтерфейс (рисунок 3.10).

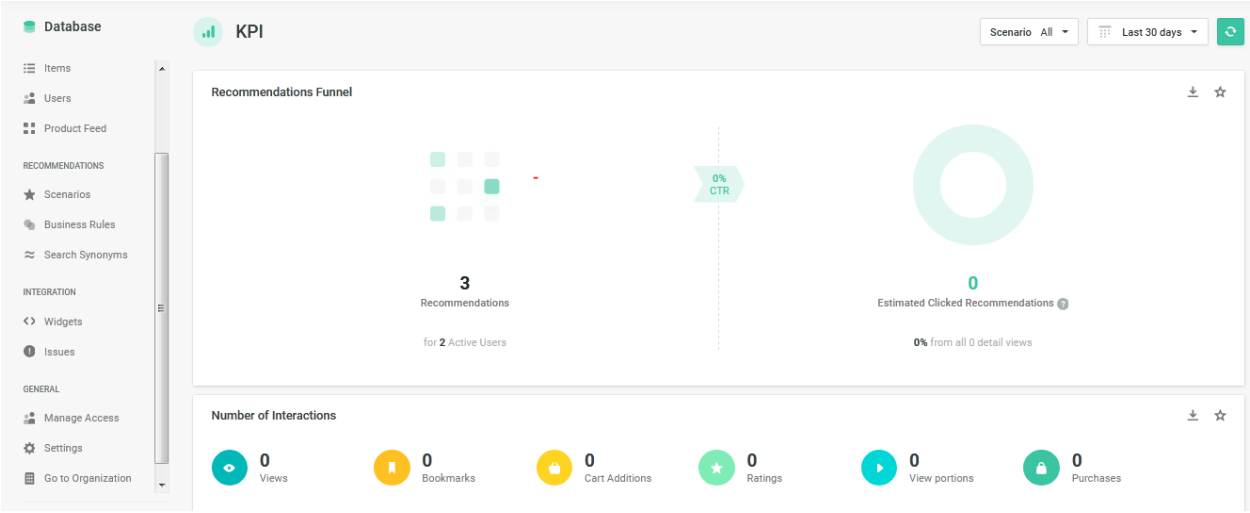

Рисунок 3.10 – Інтерфейс (дашборд) сервісу «Recombee»

Логіка роботи на стороні застосунку полягає у реєстрації дій користувача відносно сутностей в системі «Recombee». Застосунок «All films» наразі підтримує надання рекомендацій з фільмів.

Наприклад, коли споживач ставить оцінку фільму, система генерує подію «FilmRated». Відповідний слухач (listener) перехоплює подію та обробляє її, відправляючи запит до сервісу «Recombee» з ідентифікатором фільму, користувача, часовою міткою та маркером події «оцінка» (rating). При запиті користувачем фільмів до рекомендації, система звертається до «Recombee», передаючи ідентифікатор користувача та опціональні дані, зокрема кількість фільмів (limit). Сервіс генерує відповідь у вигляді масиву

ідентифікаторів фільмів. Після цього «All films» може звернутися до сховища даних про фільми, дістати їх та відправити кінцевому користувачеві.

3.13 Напрямки подальшого розвитку системи

В результаті виконаної роботи був отриманий монолітний застосунок, однак його можна декомпозувати на самостійні сервіси, таким чином переходячи до мікросервісної архітектури.

Як було сказано раніше, для великої системи необхідний пошуковий двигун, тобто можна самостійно розгорнути та підключити один з пошукових двигунів, наприклад, «ElasticSearch», який на даний момент вважається найкращим серед інших.

Бажано розробити власну рекомендаційну систему, або принаймні ізолювати її, розгорнувши на власному сервері.

Великим вкладом до такої системи може слугувати впровадження адаптивних автоматичних парсерів для пошуку та збору інформації з інших сайтів, наприклад, про сайти, де фільми можна скачати або переглянути, платно чи безкоштовно, кінотеатри, у яких іде показ кіно та про онлайн-каси цих кінотеатрів.

До сервісу можна розробити ботів, що будуть використовуватись у соцмережах та месенджерах, зокрема для Telegram. Боти можуть містити функції простого пошуку, а також надсилати персональні рекомендації чи новинки, нагадування тощо.

У системі можна зробити сервіс підписок, можливо, на платній основі, оформлення підписок дозволити у тому числі через ботів у месенджерах.

ВИСНОВКИ

При виконанні роботи розглянуто структуру сучасних веб-застосунків та процес розробки програмних систем, зроблено огляд концептуально схожих систем та вивчено фреймворк «Laravel» і супутні інструменти розробки, розібрано та використано компоненти сучасних веб-орієнтованих додатків.

Результатом проведеної роботи є створений бекенд інформаційної системи, тобто розроблена бізнес-логіка застосунку та API. Для системи також створено адмін-панель.

Програмний застосунок розроблено з використанням сучасних технологій та актуальних хмарних сервісів, з дотриманням принципів SOLID, KISS, DRY та патернів проектування. Рішення працює в тестовому режимі на локальній машині та готове до впровадження.

Процес створення системи пройшов етапи розробки специфікації, проектування, реалізації та атестації (тестування виконано в ручному режимі). Розроблена система надає всі можливості, передбачені вимогами до неї, у тому числі надає API для: доступу користувача до персонального кабінету, пошуку та фільтрації фільмів та їх атрибутів, надання рекомендацій користувачам, можливостей оцінювання фільмів, створення дискусії (коментування), скарг на контент, функціонал для створення персональних нотаток та нагадувань. Для розробників та партнерів система передбачає документацію та можливість реєстрації у сервісі сторонніх застосунків.

Система може бути використана цільовою аудиторією для пізнавально-розважальних та художніх цілей при розгортанні у мережі Інтернет.

Подальший розвиток системи може представити власну реалізацію фронтенду для споживачів, розширити можливості рекомендаційної системи, надати API для розробки або повністю розробити бота для «Telegram», впровадити механізми підписки, у тому числі платної.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Соммервил И. Инженерия программного обеспечения / Иан Соммервилл., 2002. – 624 с. – (6-ое издание).
2. Laravel Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://laravel.com/docs/8.x.
3. Ejsmont A. Web Scalabillity for Startup Engineers / Artur Ejsmont. – New York: McGraw-Hill Education, 2015. – 417 с.
4. Developing Web Information Systems / R.Vidgen, D. Avison, B. Wood, T. Wood-Harper. – London: Elsevier Inc., 2002.
5. Jenkins Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://www.jenkins.io/.
6. Open Server. Документация [Електронний ресурс] – Режим доступу до ресурсу: https://ospanel.io/docs/.
7. Download Postman [Електронний ресурс] – Режим доступу до ресурсу: https://www.postman.com/downloads/.
8. Trello [Електронний ресурс] – Режим доступу до ресурсу: https://trello.com/home.
9. Davis W. The Information System Consultant's Handbook. Systems Analysis and Design / W. Davis, D. Yen., 1998. – 800 с.
10. Laravel Nova – beautifully-designed administration panel for Laravel [Електронний ресурс] – Режим доступу до ресурсу: https://nova.laravel.com/.
11. Turland M. php|architect's Guide to Web Scraping with PHP / Matthew Turland. – Toronto: Marco Tabini & Associates, Inc., 2010. – 192 с. – (2nd edition).
12. Smith V. Go Web Scrapping Quick Start Guide / Vincent Smith. – Birmingham: Packt Publishing Ltd., 2019.

13. Proxy API for Web Scraping [Електронний ресурс] – Режим доступу до ресурсу: https://www.scraperapi.com.
14. Schrenk M. Webbots, Spiders, and Screen Scrapers: A Guide to Developing Internet Agents with PHP/CURL / Michael Schrenk. – San Francisco: No Starch Press, Inc, 2012. – 396 с. – (2nd edition).
15. Of bayesian average and star ratings [Електронний ресурс]. – 2013. – Режим доступу до ресурсу: http://fulmicoton.com/posts/bayesian_rating/.
16. Pecoraro C. Mastering Laravel / Christopher John Pecoraro. – Birmingham: Packt Publishing Ltd., 2015. – 204 с.
17. Composer Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://getcomposer.org/doc/.
18. Stauffer M. Laravel: Up & Running / Matt Stauffer. – Sebastopol: O'Reilly Media, Inc., 2020. – 512 с. – (2nd edition).
19. Pusher | Leader In Realtime Technologies [Електронний ресурс] – Режим доступу до ресурсу: https://pusher.com/.
20. Site Search & Discovery powered by AI | Algolia [Електронний ресурс] – Режим доступу до ресурсу: https://www.algolia.com/.
21. PHP Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://www.php.net/manual/en/.
22. Madden N. API Securiy in Action / Neil Madden. – Shelter Island, NY: Manning Publications Co., 2020. – 575 с.
23. Scribe Docs [Електронний ресурс] – Режим доступу до ресурсу: https://scribe.readthedocs.io/en/latest/.
24. AI-Powered Real-Time Recommender | Recombee [Електронний ресурс] – Режим доступу до ресурсу: https://www.recombee.com/.

ДОДАТОК А Інтерфейси системи

Адміністративна панель системи, розроблена з використанням пакету «Laravel Nova», складається з панелі інструментів (та ресурсів) зліва, блоку для відображення деталей по центру, пошукового рядка та індикатора поточного користувача системи справа вгорі (рисунок А.1).

На головній сторінці відображається дашборд, що відображає метрики системи. Дашбордів може бути додана необхідна кількість, один з них буде головною сторінкою. Наразі дашборд один.

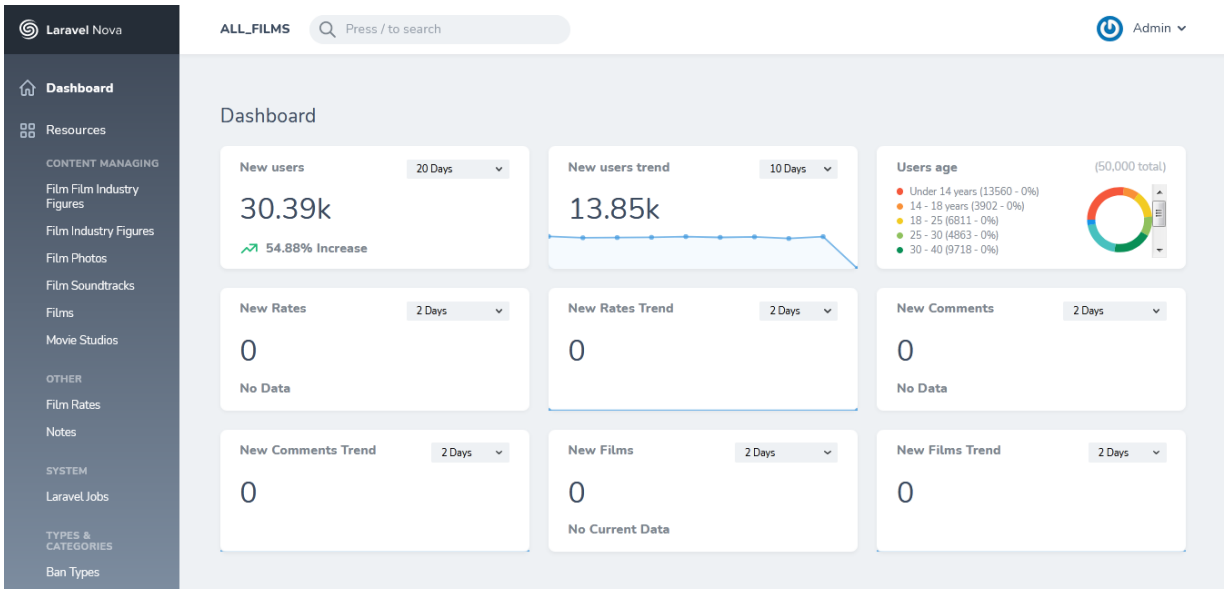

Рисунок А.1 – Загальний вигляд інтерфейсу «Laravel Nova»

Сайдбар зліва містить посилання на ресурси або інструменти системи. Для зручності навігації ресурси групуються (рисунок А.2). Кожен ресурс є наочним відображенням моделі (Eloquent-моделі [2]), яка у свою чергу відображає сутність БД.

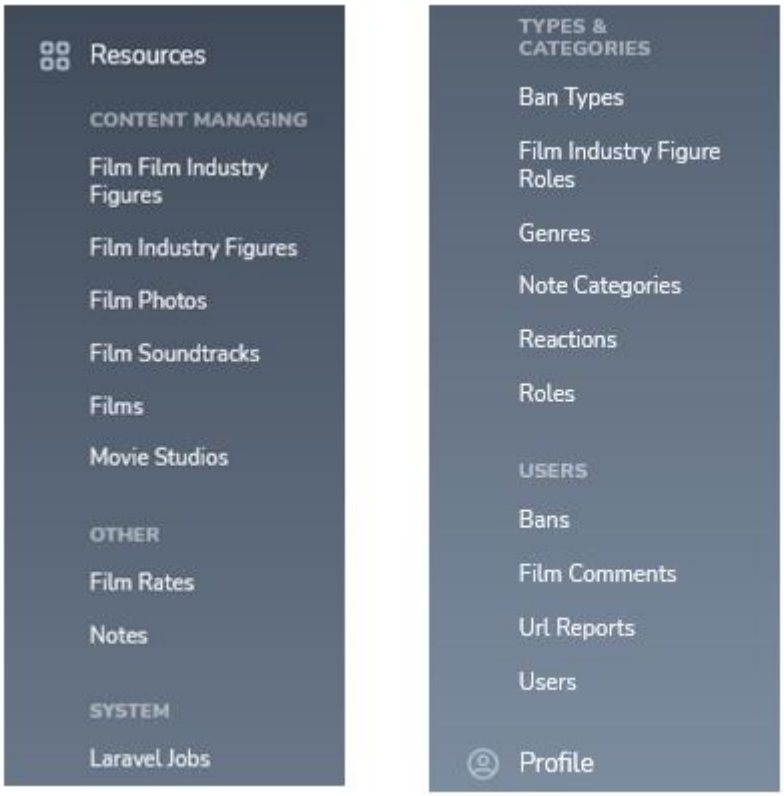

Рисунок А.2 – Ресурсна панель інтерфейсу адміністратора

Перехід за посиланням ресурсу «Users» (рисунок А.3), як і інших ресурсів, спочатку відображає список («індекс») усіх ресурсів цього типу (з пагінацією). Надається рядок для пошуку ресурсів за searchable-атрибутами [10]. Справа відкрита панель фільтрів, зокрема для типу користувача, дати реєстрації, статусу блокування (ban) тощо. Кожен зі списку ресурсів можна редагувати після переходу на його сторінку (здійснюється натисканням на ідентифікатор запису).

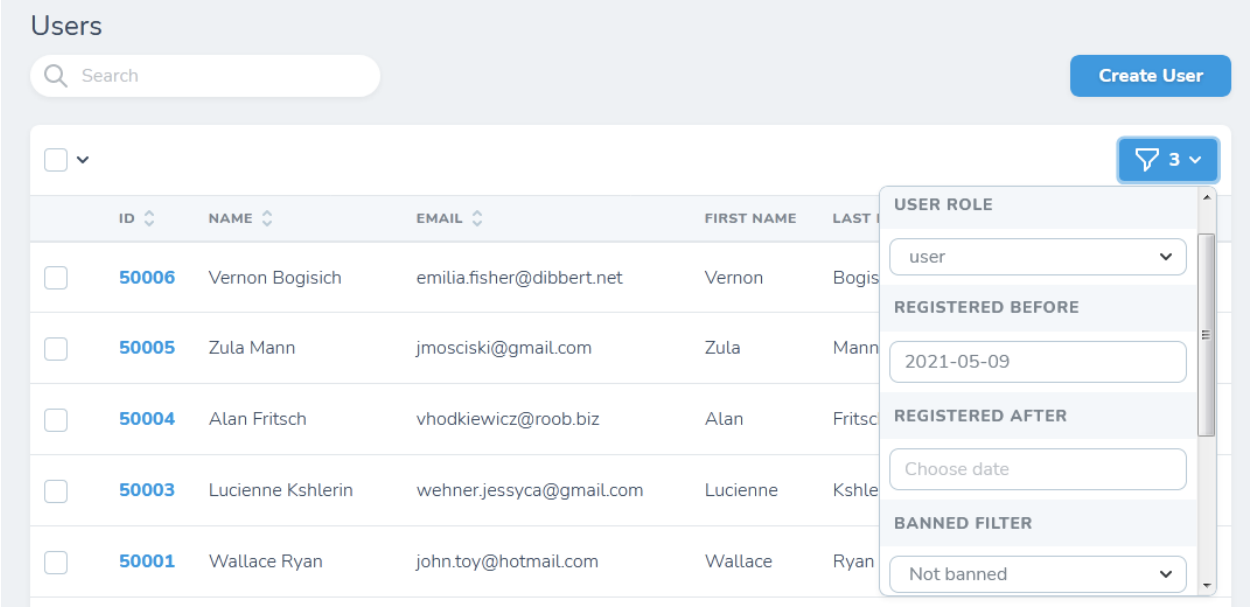

Рисунок А.3 – Інтерфейс пошуку користувачів у системі та панель фільтрів

Інтерфейси керування фільмами показано на рисунках А.4 – А.7. Кнопки для ініціації дій (actions) знаходяться справа зверху (рисунки А.4, А.5). Поля для встановлення значень атрибутів ресурсу наведено на рисунках А.6, А.7.

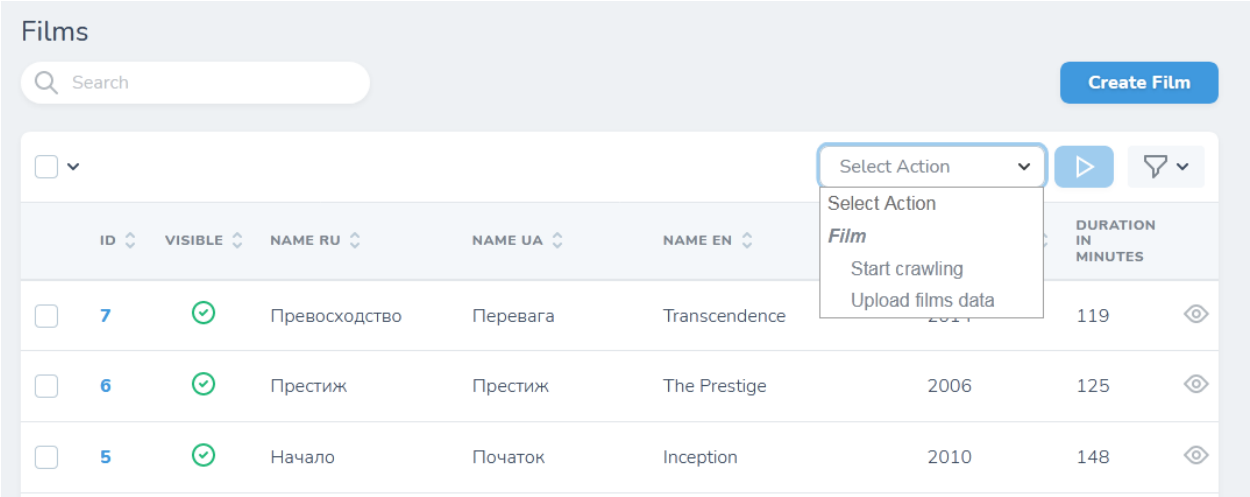

Рисунок А.4 – Адмін-панель. Кнопки для ініціювання завантаження файлів та запуску скрейпінгу

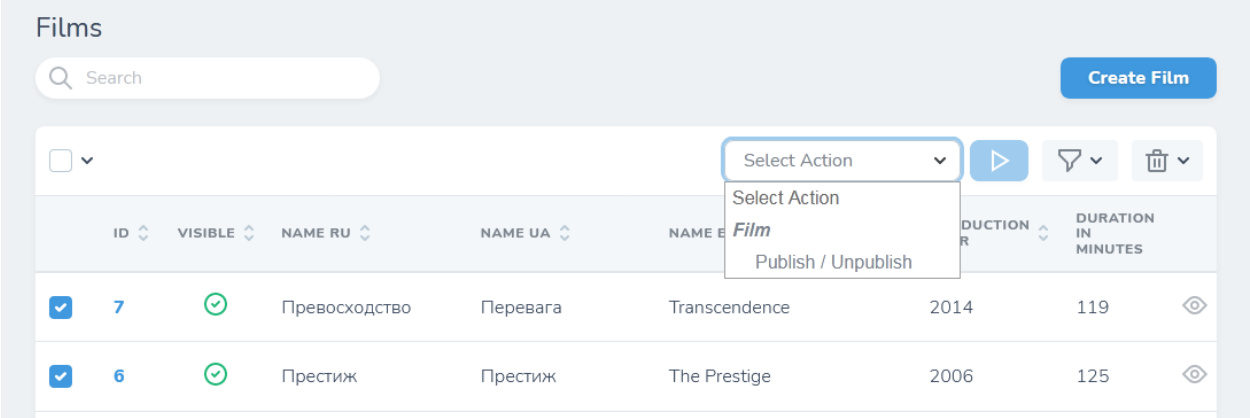

Рисунок А.5 – Адмін-панель. Перелік фільмів та кнопки дій

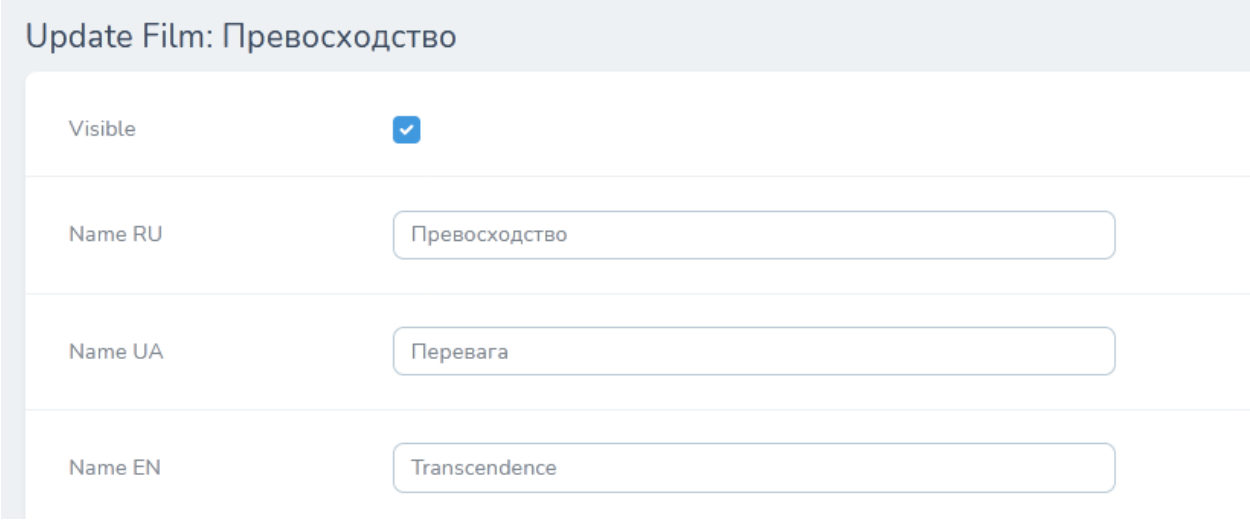

Рисунок А.6 – Форма створення та оновлення фільму. Текстові поля

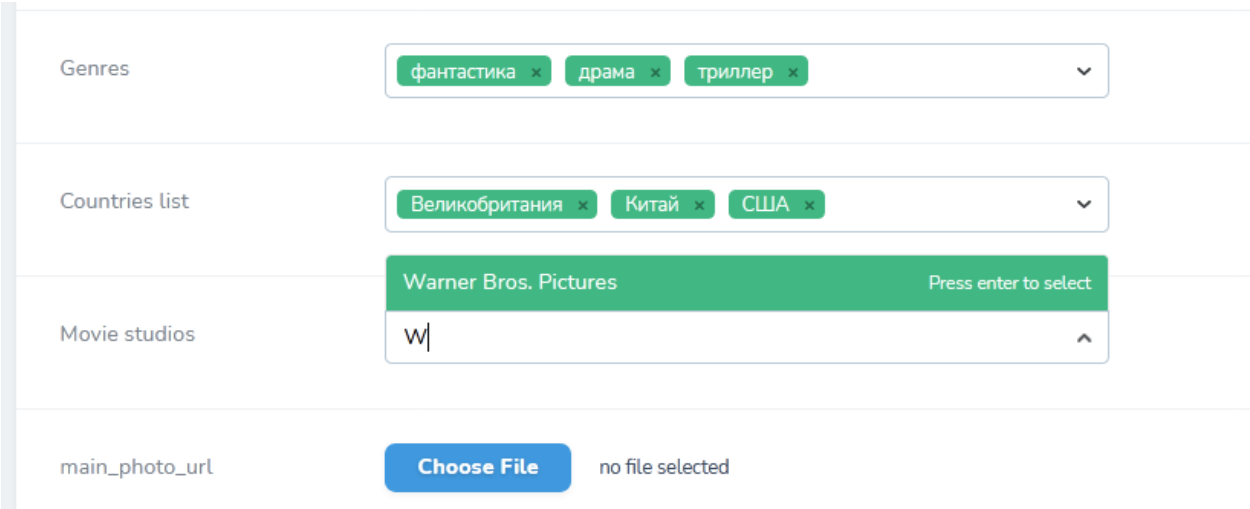

Рисунок А.7 – Форма створення та оновлення фільму. Поля для встановлення зв’язків з іншими ресурсами

ДОДАТОК Б ІНСТРУКЦІЯ З РОЗГОРТАННЯ СИСТЕМИ

Система вимагає наявності:

- встановленого та налаштованого веб-сервера («Apache HTTP Server», «nginx» або іншого);
- СУБД MySQL;
- PHP ~7.3;
- пакетного менеджера Composer;
- інші компоненти в конфігураційному файлі «.env.example».

Виконати:

1) скопіювати файл «.env.example» в «.env» та вказати зазначені в ньому конфігураційні змінні;
2) встановити пакети з файлу «composer.json» (команда «composer install»);
3) запустити міграції та сідер («php artisan migrate --seed»);
4) запустити обробники черг (черги: reset_requests, reset_mails, parsing_queue, update_rating, default).

Застосунок працює на вказаному у конфігуції сервера порті (зазвичай 80). Інші налаштування вказані у файлі «README.md» та документації фреймворку «Laravel» [2].