Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка інформаційної платформи оглядів японських медіапродуктів з використанням DJango фреймворку та MySQL

Виконав(ла): студент(ка) 4 курсу, групи СП-41
спеціальності 121 інженерії програмного забезпечення
(шифр і назва спеціальності)

(підпис) Занчук Т.А. (прізвище та ініціали)

Керівник (підпис) Петрик М.Р. (прізвище та ініціали)

Нормоконтроль (підпис) Стоянов Ю.М. (прізвище та ініціали)

Завідувач кафедри (підпис) Петрик М.Р. (прізвище та ініціали)

Рецензент (підпис) Липак Г.І. (прізвище та ініціали)

Тернопіль 2025

АНОТАЦІЯ

Занчук Тарас, Розробка інформаційної платформи оглядів японських медіапродуктів з використанням DJango фреймворку та MySQL. сторінок – 90, джерел – 23, рисунків – 40, додатків – 5.

Ключові слова: інформаційна система, платформа оглядів, медіапродукти.

У даній кваліфікаційній роботі проведено розробку та реалізацію інформаційної платформи для огляду японських медіапродуктів: манги, новели та аніме. Метою роботи є створення зручної та інтуїтивно зрозумілої системи, що відповідає сучасним вимогам, та надає користувачам можливість переглядати інформацію про цікаві їм медіапродукти, з можливістю їх відстежування, коментування та стежити за новинами. Реалізація даної роботи супроводжувалась з використанням сучасних технологій: фреймворком Django, СУБД MySQL, HTML/CSS (SCSS).

У процесі розробки було проведено аналіз вимог, визначено користувацькі, функціональні та нефункціональні вимоги, спроектовано архітектуру, розділивши систему на модулі, використано патери проектування «MVT» і «Фасад», а також спроектовано відповідні UML-діаграми, результати яких було опубліковано у тезах до наукової конференції, що наведені у додатку А. Було створено інтерфейс користувача, реалізовано URL-маршрути та представлення (Views) до них, щоб забезпечити можливість взаємодії користувача з системою. Перевірка функціональності супроводжувалась модульним та автоматизованим тестуванням.

Розроблена система демонструє доцільність створення локалізованих інформаційних платформ у сфері культури та розваг. Вона є перспективною для подальшого розвитку: впровадження нових функцій, інтеграції з іншими сервісами тощо. Проект виконано з дотриманням сучасних стандартів програмної інженерії, що забезпечує його актуальність та конкурентоспроможність.

ABSTRACT

Zanchuk Taras, Development of an information platform for reviews of Japanese media products using DJango framework and MySQL. pages - 90, sources - 23, figures - 40, appendices - 5.

Keywords: information system, review platform, media products.

In this qualification work, the development and implementation of an information platform for reviewing Japanese media products: manga, novels and anime. The aim of the work is to create a convenient and intuitive system that meets modern requirements and provides users with the opportunity to view information about media products of interest to them, with the ability to track, comment and follow the news. This work was implemented using modern technologies: Django framework, MySQL database, HTML/CSS (SCSS).

In the course of development, we analyzed the requirements, identified user, functional and non-functional requirements, designed the architecture, dividing the system into modules, used the MVT and Facade design patterns, and designed the corresponding UML diagrams, the results of which were published in the abstracts for the scientific conference, which are given in Appendix A. We created a user interface, implemented URLs and views to them to enable user interaction with the system. Functionality verification was accompanied by modular and automated testing.

The developed system demonstrates the feasibility of creating localized information platforms in the field of culture and entertainment. It is promising for further development: introduction of new functions, integration with other services, etc. The project was implemented in compliance with modern software engineering standards, which ensures its relevance and competitiveness.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ І ТЕРМІНІВ

Манга – японський комікс, переважно чорно-білий, іноді кольоровий.

Аніме – японська анімація, особлива тим, що розрахована не тільки на дитячу аудиторію, а на різні вікові групи.

Новела (японська) – роман, відомий також під назвами «ранобе» або «light novel», орієнтований переважно на підлітковий та юнацький віки.

Том – одиниця публікації, що уособлює розділ для манги або новели.

Сезон – одиниця публікації для аніме.

Django – веб-фреймворк для розробки веб-сайтів, сервісів.

MySQL – це система управління реляційними базами даних. Термін «реляційна» (від латинського relatio — відношення) вказує, перш за все, на те, що така модель зберігання даних побудована на взаємовідношенні складових її частин. Реляційна база даних, як правило, складається з багатьох відношень. Відношення — це таблиця даних.

СУБД – програмне забезпечення для керування базою даних.

HTML – мова програмування для реалізації веб-сторінок.

SCSS – розширена технологія для покращення рівня абстракції для CSS.

CSS – спеціалізована мова для стилів веб-сторінок.

Міксини – це шаблони коду, які дозволяють повторно використовувати стилі з можливістю передачі параметрів для гнучкості та зменшення дублювання.

ВВ – варіанти використання до даної системи.

URL-маршрути – це система, яка визначає, яка частина коду (представлення) обробляє запит за конкретною веб-адресою.

Представлення (Views) – це функції або класи, які обробляють запити користувачів і повертають відповіді, зазвичай у вигляді HTML-сторінок або даних.

ВСТУП

На сьогоднішній день, у світі інформація розповсюджується досить високою швидкістю та доступна усім у цифровому форматі. Для різних сфер діяльності: науки, освіти, культури, розваг, послуг та інших, є необхідність розробки різних видів інформаційних систем.

Оскільки, у наш час сфера культури та розваг різних країн світу розширилась та стала доступна для всіх охочих через інтернет, постає завдання у пошуку зручних інформаційних систем, які надають доступ до перегляду інформації про різні цікаві людям продукти. Особливою увагою останнім часом серед підлітків та молоді вирізняється попит на японські медіапродукти, а саме: манги, новели та аніме. Тож попит на доступні, а особливо локалізовані інформаційні платформи, де користувач може швидко знайти та отримати інформацію про необхідний продукт, швидко зростає. Метою даної кваліфікаційної роботи є розробка веб-платформи для огляду японських медіапродуктів.

Завданням цієї роботи є спроектувати та розробити зручну для користувача інформаційну систему для платформи огляду медіапродуктів використовуючи сучасні технології, інструменти та підходи до розробки програмного забезпечення.

У проекті планується реалізувати зручний та інтуїтивно зрозумілий інтерфейс користувача. Також платформа міститиме можливість для користувача відслідковувати цікаву йому роботу, можливість спілкуватися з іншими користувачами через коментарі, а також бути в курсі останніх новин платформи та індустрії.

Також, результати розробки цієї інформаційної системи було опубліковано у вигляді тез до конференції «ПРИРОДНИЧІ ТА ГУМАНІТАРНІ НАУКИ. АКТУАЛЬНІ ПИТАННЯ», які можна переглянути у додатку А.

Реалізувавши даний проект у межах цієї бакалаврської роботи, очікується підкреслити важливість веб-платформ для огляду медіапродуктів у сучасному світі, а також надати користувачам більш ширший вибір зручних інформаційних систем.

РОЗДІЛ 1 АНАЛІЗ ВИМОГ ДО ІНФОРМАЦІЙНОЇ ПЛАТФОРМИ

1.1 Огляд конкурентів

Одним з найпопулярніших та найбільших інформаційних веб-платформ для продуктів з сфери розваг є «Fandom».

Рисунок 1.1 – Логотип веб-платформи «Fandom» [1]

Заснована у 2004 році Джиммі Вейлзом та Анджелою Біслі Старлінг (британською веб-підприємницею), Fandom (спочатку Wikicities, а пізніше Wikia) була новою платформою, яка розвивалася на основі основної технології, що забезпечує роботу Вікіпедії, щоб задовольнити потреби неймовірно захопленої групи клієнтів: шанувальників [1].

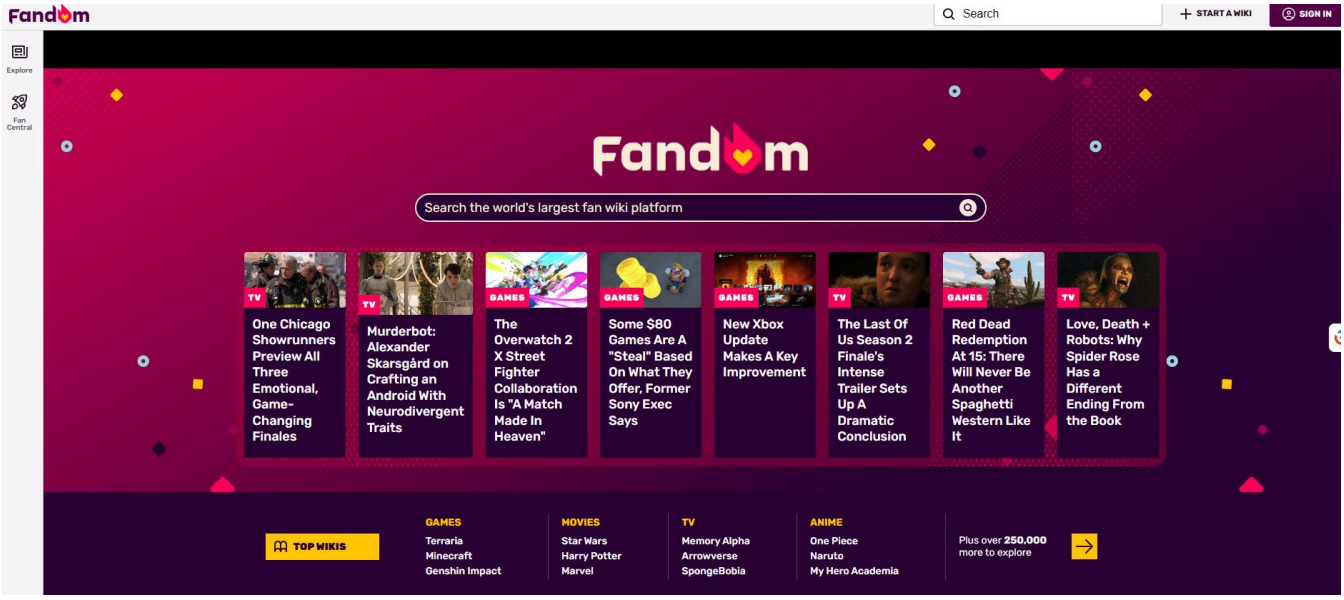

Рисунок 1.2 – Головна сторінка веб-платформи «Fandom» [1]

ЇЇ особливістю є те, що крім уже наявної інформації про продукти, будь який користувач може створити власну інформаційну базу про необхідний йому продукт безплатно.

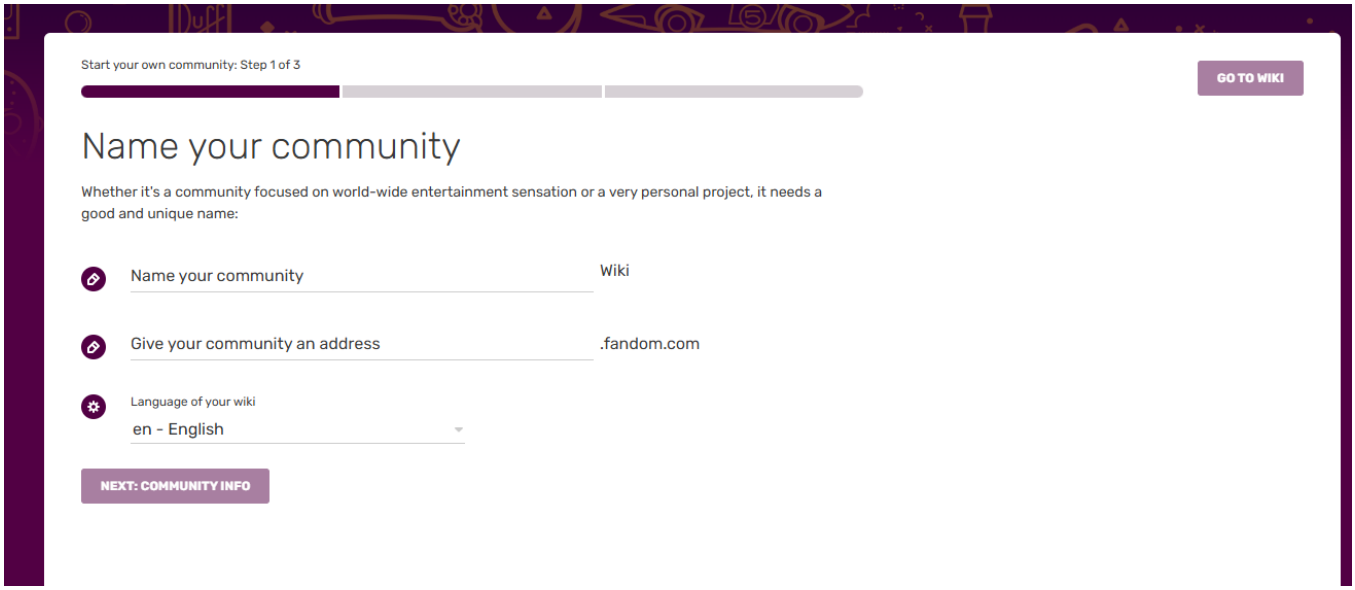

Рисунок 1.3 – Сторінка створення власної інформаційної бази [1]

Дана платформа є досить сильним конкурентом не тільки для розроблювальної системи, а й для інших конкурентів, оскільки вона має зручний, широкий та дещо унікальний функціонал, досить велику аудиторію та інформаційну базу даних для різних продуктів, що не обмежується лише японськими медіапродуктами.

Іншим конкурентом, уже на українському ринку, може бути веб-сайт «НекоТека», що виконує роль українізованої інформаційної платформи японських медіапродуктів.

Рисунок 1.4 – Логотип веб-сайту «НекоТека» [2]

Також, на даному веб-сайті можна переглянути інформацію про цікаві медіародукти відповідно до вибраного користувачем сезону.

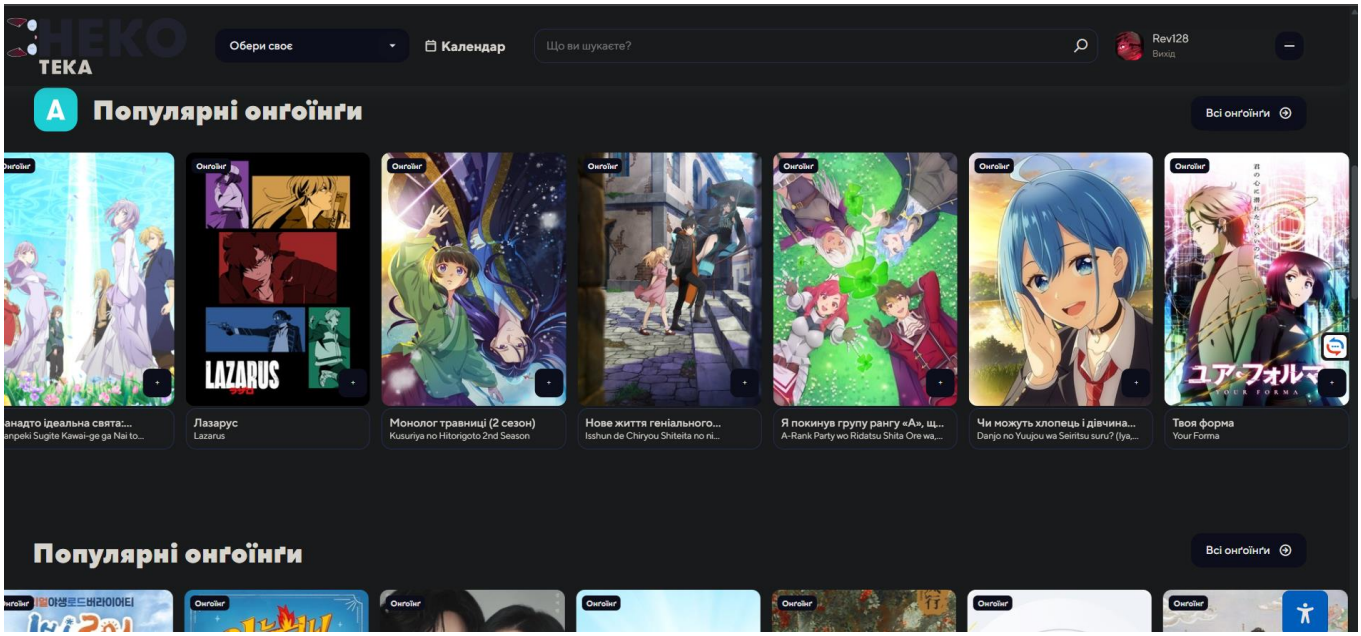

Рисунок 1.5 – Головна сторінка веб-сайту «НекоТека» [2]

Оскільки даний веб-сайт має обширний вибір україномовної інформації про медіапродукти у каталозі, що оновлюється у реальному часі, можливість створювати власні списки перегляду та прочитаного, що допомагає організувати та відстежувати їхні уподобання та прогрес, а також елементи інтерактивності [2], що робить його конкурентом на українському ринку.

Отже, було проаналізовано функціонал, можливості та переваги двох конкурентів до розроблювальної платформи: веб-сайт «НекоТека» на українському ринку та веб-платформа «Fandom», що є сильним конкурентом не тільки на українському, а й на глобальному ринку.

1.2 Визначення вимог до проекту

Після огляду конкурентів на різних ринках, далі слід проаналізувати вимоги та сформувати специфікацію до розроблювального проекту.

Аналіз вимог – один з основних робочих потоків (workflow) програмної інженерії, поряд, з такими потоками, як проектування інтерфейсу користувача, або програмування [3].

Для його позначення в англомовній літературі, як правило, використовується поняття "Requirement Process". У вітчизняній практиці, разом з узагальнювальним терміном "аналіз вимог", зустрічаються також такі терміни, як "потік робіт "вимоги", "робота з вимогами", "визначення вимог" і так далі [3].

Отже, провівши огляд та аналіз конкурентів на ринку, можна виділити функціональні та нефункціональні вимоги до розроблювальної інформаційної платформи.

Перш ніж почати аналізувати та виявляти функціональні вимоги до програмного забезпечення, визначимо склад акторів у системі.

Актор - це хтось або щось, що володіє активністю по відношенню до програмної системи [4].

Дана інформаційна система буде передбачати ролі таких користувачів (акторів):

– Користувач – виконує роль звичайного користувача;

– Адміністратор – має ті самі можливості, що й звичайний користувач, а також додаткові (розширені) права.

Визначивши акторів системи можемо приступити до формування їх вимог користувачів.

Вимоги для користувача:

– Користувач повинен мати змогу авторизуватись або зареєструватися у системі;

– Користувач може переглядати основну інформацію про певну роботу (назва, жанри, автор, опис, наявні публікації);

– Користувач може переглядати додаткову інформацію про певну роботу (інформація про певну публікацію (манга, новела, аніме) або одиницю публікації (том, сезон));

– Користувач повинен мати доступ до додавання коментарів до певної одиниці публікації (тому, сезону);

– Користувач повинен мати можливість відстежувати обрані ним роботи;

– Користувач може редагувати основну інформацію у своєму профілі (ім’я користувача, електронна адреса, аватар);

– Користувач має мати доступ до перегляду серверних новин та новин індустрії (новини про роботи);

Вимоги для адміністратора:

a) Адміністратор повинен мати змогу вносити нову інформації у систему:

1) Додавання нової роботи;

2) Додавання інформації про публікації роботи (манга, новела, аніме);

3) Додавання інформації про одиницю публікації (том, сезон);

4) Додавання серверних новин;

5) Додавання новин індустрії (новини про роботи);

b) Адміністратор повинен мати доступ редагування інформації у системі:

1) Перевірка коментарів користувачів;

2) Надання прав новому адміністратору;

3) Редагування уже наявної інформації.

Отже, визначивши вимоги для користувачів, складемо ряд функціональних вимог для системи:

– Система повинна передбачати можливість авторизації та реєстрації користувачів;

– Повинен бути реалізований функціонал доступу користувача до перегляду основної та додаткової інформації про роботи, серверних новин та новин індустрії;

– На платформі повинен бути реалізований та доступний функціонал для коментування користувачем одиниць публікації;

– Повинен бути реалізований функціонал для відстежування робіт користувачем;

– Система має передбачати можливість редагування користувачем власної основної інформації;

– Система повинна надавати можливість внесення змін інформації у системі: додавання нової, перегляд та зміна уже наявної інформації у системі.

Далі перейдемо до формування нефункціональних вимог.

Нефункціональні вимоги до проекту:

– Зручність використання: користувачеві повинен надаватись інтуїтивно зрозумілий та зручний інтерфейс;

– Оновлення версій: користувач повинен бути проінформований про зміни у системі;

– Допомога в режимі онлайн: користувач повинен мати змогу зв’язатися з адміністратором сайту;

Також сформуємо вимоги до інтерфейсу користувача:

– Інтерфейс повинен бути зручним та зрозумілим для користувача у користуванні;

– Повинна бути можливість адаптування елементів сторінки платформи під різні види розмірів екрану;

– Необхідність присутності системи навігації по платформі (навігаційних кнопок, системи пошуку).

Отже, було визначено ролі користувачів та їх можливості у системі, функціональні та нефункціональні вимоги до проекту, та вимоги до користувацького інтерфейсу.

1.3 Визначення технологій розробки, інструментів, методології та архітектури системи

Після аналізу та виявлення вимог до проекту, визначимо архітектуру системи.

Вибір архітектури ПЗ — це етап проектування, що виконується після етапу аналізу і формулювання вимог. Задача такого проектування — перетворення вимог до системи у вимоги до ПЗ і побудова на їх основі архітектури системи. Побудова архітектури системи здійснюється шляхом визначення цілей системи, її вхідних і вихідних даних, декомпозиції системи на підсистеми, компоненти або модулі та розроблення її загальної структури. Проектування архітектури системи може проводитися різними методами (стандартизованим, об’єктно-орієнтованим, компонентним і ін.), кожний з яких пропонує свій шлях побудови архітектури, а саме, визначення концептуальної, об’єктної й інших моделей за допомогою відповідних конструктивних елементів (блок-схем, графів, структурних діаграм тощо) [9].

Архітектура програмного забезпечення (англ. software architecture) – це структура програми або обчислювальної системи, яка включає програмні компоненти, видимі зовні властивості цих компонентів, а також відносини між ними [9].

Оскільки метою даної кваліфікаційної роботи є розробка веб-платформи, було вирішено вибрати 3-х рівневу клієнт-серверну архітектуру.

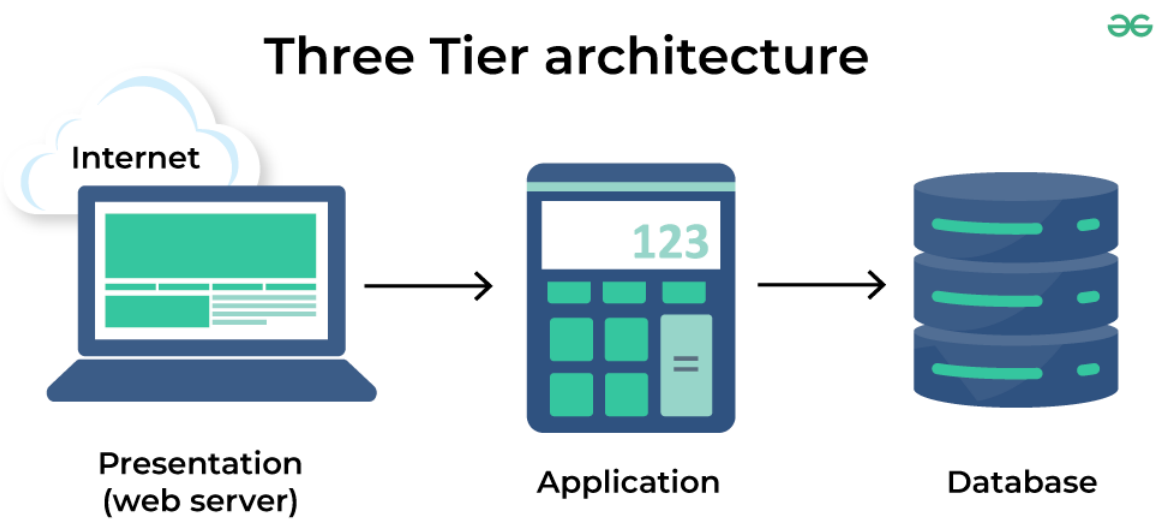

Рисунок 1.6 – Приклад роботи 3-х рівневої архітектури [6]

Трирівнева архітектура — це добре налагоджена архітектура програмних застосунків, яка організовує застосунки на три логічні та фізичні обчислювальні рівні: рівень представлення або інтерфейс користувача; рівень застосунків, де обробляються дані; та рівень даних, де зберігаються та управляються дані застосунків [5].

Рівень презентації : Рівень інтерфейсу користувача, де відбуваються взаємодії. Він обробляє відображення даних та введення користувачем [6].

Рівень застосунку : Рівень бізнес-логіки, який обробляє запити користувачів, виконує обчислення та приймає рішення. Він діє як посередник між рівнями представлення та даних [6].

Рівень даних : Рівень зберігання відповідає за керування даними та їх зберігання. Він обробляє операції з базою даних та їх пошук [6].

Головна перевага трирівневої архітектури полягає в тому, що оскільки кожен рівень працює на власній інфраструктурі, кожен рівень може розроблятися одночасно окремою командою розробників. І його можна оновлювати або масштабувати за потреби, не впливаючи на інші рівні [5].

Далі перейдемо до вибору методології.

Методологія - це система принципів, а також сукупність ідей, понять, методів, способів і засобів, які визначають стиль розробки програмного забезпечення. Це – реалізація стандарту. Самі стандарти лише вказують на те, що

повинно бути, залишаючи свободу вибору і адаптації. Методології являють собою ядро теорії управління розробкою програмного забезпечення [9].

Для полегшення розподілення розробки інформаційної платформи було обрано методологію Agile, а саме: Scrum.

Agile – це підхід, який поділяє роботу на фази, з акцентом на безперервному виконанні та вдосконаленні. Методологія Agile приносить користь командам, дозволяючи адаптивне планування, швидке виконання та постійну оцінку, що призводить до більш оперативних та успішних результатів [7].

Scrum — це гнучка структура управління проектами , яка допомагає командам структурувати та керувати своєю роботою за допомогою набору цінностей, принципів та практик [7].

У scrum розробка продукту розділяється на ітерації (спринти), що розділяє розробку на невеликі частини.

Спринт — це короткий, обмежений у часі період, протягом якого scrum-команда працює над виконанням певного обсягу роботи [7].

Визначивши архітектуру проекту та методологію розробки, починається вибір технологій та інструментів для реалізації цієї інформаційної системи.

Для реалізації 3-х рівневої архітектури було обрано веб-фреймворк Django на мові програмування Python, що забезпечує спрощення та сприяє швидкій розробки веб-додатків. Щодо користувацького інтерфейсу, за який у даній кваліфікаційній роботі відповідають веб-сторінки розроблювальної інформаційної платформи, було обрано стандартизовану мову розмітки для веб-сторінок, а також використання технології SCSS, що є розширенням для CSS. Для зберігання інформації системи було обрано реляційну базу даних MySQL.

Щоб забезпечити зручність розробки та управління проекту було обрано інструменти, що перелічені у таблиці 1.1.

Таблиця 1.1 – Перелік та опис призначення інструментів розробки

№	Назва	Опис призначення
1	Trello	Використовується для управління проектом.
2	GitHub	Система для керування та контрою версіями, використовується для зберігання коду та відстежування результатів розробки.
3	PyCharm	Використовується як середовище програмування.
4	MySQL Workbench	СУБД, що забезпечує керування для бази даних MySQL.

Отже, було визначено архітектуру розроблювальної системи, методологію розробки, а також технології та інструменти для спрощення розробки інформаційної платформи.

1.4 Підсумки розділу 1

У розділі 1 «Аналіз вимог до інформаційної платформи» було розглянуто таких конкурентів до розроблювальної платформи: веб-платформа «Fandom» та веб-сайт «НекоТека», що, завдяки своєму функціоналу, представляють конкуренцію не тільки на українському, а й на глобальному ринку.

Далі було виявлено акторів системи, ряд функціональних та нефункціональних вимог, вимог до інтерфейсу користувача.

Вибрано 3-х рівневу клієнт-серверну архітектуру системи, гнучку методологію Agile та її фреймворк Scrum, що полегшує розробку додатку. Після цього було вибрано технології такі, як Django, MySQL, HTML та SCSS, що допомагають розробити інформаційну платформу відповідно до виявлених вимог, а також ряд інструментів, що забезпечують реалізацію обраної архітектури, методології та технологій.

РОЗДІЛ 2 ПРОЕКТУВАННЯ СИСТЕМИ

2.1 Вибір інструментів проектування

Мова UML призначена для опису моделей і кардинально відрізняється від таких мов програмування як Visual C ++ чи Java. Для роботи з UML використовується спеціальні CASE-засоби проектування ПЗ, що включають редактори діаграм [10].

UML не залежить від об'єктно-орієнтованих мов програмування і може підтримувати будь-яку з них. Ця мова також не залежить від використовуваної методології розробки проекту. Створені на UML діаграми виразні і зрозумілі для всіх розробників, залучених до проекту, причому, що важливо, не тільки в момент розробки, а й багато місяців по тому [10].

UML є відкритим і володіє засобами розширення базового ядра. На UML можна змістовно описувати класи, об'єкти і компоненти в різних предметних областях, що часто сильно відрізняються одна від одної [10].

Отже, для правильного проектування розроблювальної інформаційної системи, потрібно обрати відповідний інструмент. Одним з таких інструментів є IBM Rational Software Architect Designer, що є одним з лідерів серед інструментів проектування на глобальному ринку.

Головна відмінність IBM Rational Software Architect від інших CASE-засобів в тому, що він корисний не тільки проектувальнику систем, але і розробнику програмного коду. Для проектувальника переваги використання CASE-засобів не викликають сумнівів, але для розробника коду використання IBM Rational Software Architect дає такі ж переваги, як заміна велосипеда на літак. Використовуючи такий інструмент, досягнення цілей стає простіше і швидше, а вся робота може бути розглянута одним поглядом. Якщо в проект приходить нова людина, то ознайомившись з діаграмами IBM Rational Software Architect, без складностей увійде в курс справи [10].

2.2 Моделювання варіантів використання системи на основі вимог

Для створення ефективної програмної системи необхідно зануритися в предметну область, щоб чітко уявити роботу всього того механізму, з яким матиме справу програмна система [10].

У попередньому розділі «Аналіз вимог до інформаційної платформи», було виявлено ряд користувацьких та функціональних вимог, які система повинна забезпечити для акторів системи. Щоб краще виділити та наглядно показати, які саме система повинна надавати користувачеві функції, була розроблена діаграма варіантів використання.

Розробка діаграми варіантів використання переслідує мету [11]:

– Визначити загальні межі і контекст модельованої предметної області на початкових етапах проектування системи;

– Сформулювати загальні вимоги до функціональної поведінки проектованої системи;

– Розробити вихідну концептуальну модель системи для її подальшої деталізації у формі логічних і фізичних моделей;

– Підготувати вихідну документацію для взаємодії розробників системи з її замовниками і користувачами.

Суть даної діаграми полягає в наступному: проектована система представляється у вигляді множини сутностей або акторів, що взаємодіють з системою за допомогою так званих варіантів використання [11].

У загальному випадку, діаграма варіантів використання – це граф спеціального виду, який є графічною нотацією для представлення конкретних варіантів використання, акторів, можливо деяких інтерфейсів, і відносин між цими елементами [11].

Отже, проаналізувавши вимоги користувача та функціональні умови системи, що були визначені у попередньому розділі, було визначено ряд варіантів використання для акторів системи, що описані у таблиці 2.1.

Таблиця 2.1 – Варіанти використання для акторів системи

Актор	Варіанти використання
Користувач	Додавання коментаря до певного тому або сезону роботи.
	Додання певної роботи до списку відстежуваних робіт користувача.
	Редагування профілю користувача.
	Перегляд основної інформації про роботу. Включає: – Перегляд додаткової інформації про роботу.
	Авторизація користувача. Включає: – Реєстрація нового користувача.
	Перегляд новин. Включає: – Перегляд новин індустрії; – Перегляд серверних новин.
Адміністратор	Додавання коментаря до певного тому або сезону роботи.
	Додання певної роботи до списку відстежуваних робіт користувача.
	Редагування профілю користувача.
	Перегляд основної інформації про роботу. Включає: – Перегляд додаткової інформації про роботу.
	Авторизація користувача. Включає: – Реєстрація нового користувача.
	Перегляд новин. Включає: – Перегляд новин індустрії; – Перегляд серверних новин.
	Редагування інформації про роботи.
	Додавання нових робіт.
	Перегляд інформації у системі.
	Додавання нових новин

Сформувавши варіанти використання для визначених ролей користувачів, на їх основі було спроектовано діаграму варіантів використання.

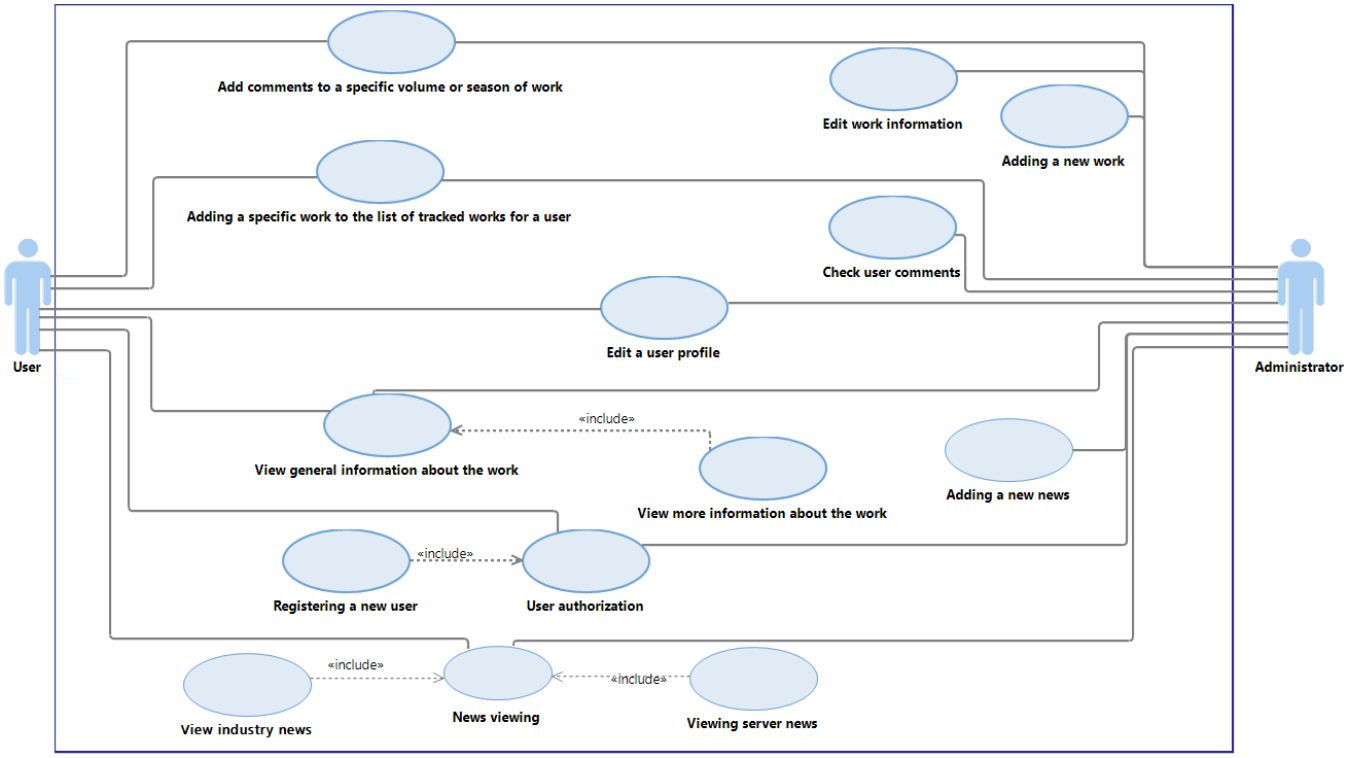

Рисунок 2.1 – Діаграма варіантів використання для авторів системи

У таблиці 2.2 наведено опис до кожного варіанту використання.

Таблиця 2.2 – Опис варіантів використання

Варіанти використання	Опис варіантів використання
Додавання коментаря до певного тому або сезону роботи.	Користувач може на сторінці одиниці публікації залишити власний коментар.
Додання певної роботи до списку відстежуваних робіт користувача.	Користувач може відстежувати цікаві роботи, що будуть відображатись у профілі.
Редагування профілю користувача.	Користувач може змінювати свої ім’я, пошту та аватар.

Продовження таблиці 2.2

Перегляд основної інформації про роботу. Включає: – Перегляд додаткової інформації про роботу.	Користувач може переглянути усю наявну інформацію про медіапродукт на відповідних сторінках платформи.
Авторизація користувача. Включає: – Реєстрація нового користувача.	Користувач може зайти на свій акаунт або створити новий.
Перегляд новин. Включає: – Перегляд новин індустрії; – Перегляд серверних новин.	Користувач може переглядати вміст серверних та індустріальних новин на головній сторінці платформи.
Редагування інформації про роботи.	Адміністратор може редагувати уже наявну інформацію у системі.
Додавання нових робіт.	Адміністратор може додавати інформацію про нові медіапродукти.
Перегляд інформації у системі.	Адміністратор може переглядати усю наявну інформацію у системі.
Додавання нових новин	Адміністратор може додавати індустріальні та системні новини.

Отже, на даному етапі проектування, для акторів розроблювальної інформаційної системи, що були визначені у минулому розділі «Аналіз вимог до інформаційної платформи», виконано моделювання їх варіантів використання з використанням UML-діаграми варіантів використання. Також до кожного варіанту використання було додано їх опис.

2.3 Архітектурне проектування системи

2.3.1 Вибір архітектурної моделі системи

Після того як було визначено варіанти використання, потрібно розділити функціонал на різні частини для кращої організації та поліпшення керування системою. Для вирішення цієї проблеми було застосовано модульну архітектуру.

Модульна архітектура — це саме те, чим вона здається: метод розбиття складнощів проблеми на менші, більш керовані частини. Ключова відмінність полягає в тому, що вона містить певні правила, концепції та шаблони як тип архітектури програмного забезпечення [12].

Один модуль містить усі функції програмного забезпечення, а не лише код. Гнучка розробка є головною перевагою модульної архітектури, і це перший крок у контролі складності програмного забезпечення. Модулі виконують спеціалізовані завдання, які зазвичай потребують можливостей інших модулів для завершення потрібного бізнес-процесу. Оскільки залежності не можуть бути циклічними, один модуль залежить від кількох інших [12].

Щодо цього, було вирішено поділити систему на п’ять окремих модулів, де кожен модуль відповідає за реалізацію конкретного функціоналу платформи:

– Модуль infoPlatform: центральний модуль системи, що відповідає за опис налаштування системи: підключення необхідних бібліотек, інших модулів, інструментів та їх конфігурацію.

– Модуль infoPlatformMainApp: модуль, що описує та надає основний функціонал платформи: загальний макет сторінки, головну сторінку, загальні стилі, новини платформи та їх вивід у адміністративну панель.

– Модуль apps: складається з двох інших модулів, що описують функціонал для роботи з медіапродуктами та користувачами.

– Модуль mediaProducts: один з модулів з якого складається модуль apps. Описує та надає функціонал роботи з інформацією про роботи, публікації та

одиниць публікацій: сторінки для відображення інформації про медіапродукт, додаткові стилі для даних сторінок, вивід медіапродуктів у адміністративну панель.

– Модуль user: один з модулів з якого складається модуль apps. Описує функціонал роботи з користувачами та їх коментарями: сторінка профілю та її додаткові стилі, вивід користувачів та коментарів у адміністративну панель.

Щоб наглядно продемонструвати описані вище модулі та зв'язки між ними, було складено діаграму пакетів системи, що зображена на рисунку 2.2.

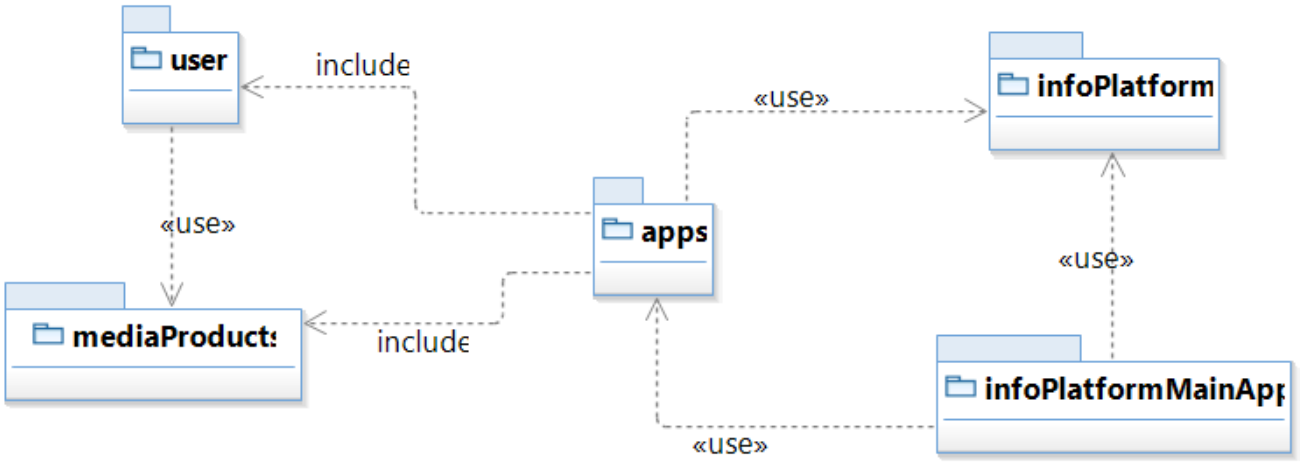

Рисунок 2.2 – Діаграма пакетів для модулів системи

Отже, для кращої організації та управління системою, було застосовано модульну архітектуру, що розподілило функціонал додатку на більш керовані частини.

2.3.2 Побудова UML-діаграм ієрархії класів

Оскільки, як було вище згадано, розроблювальну інформаційну систему розподіллено систему на декілька керованих частин (модулів), далі було

спроектовано основні елементи системи, що утворюють її загальну структуру – її класи. Для цього використали такий UML-елемент, як діаграму класів.

Діаграма класів – основна діаграма для створення коду програми. За допомогою діаграми класів створюється внутрішня структура системи, описується спадкування і взаємне положення класів один щодо одного. Тут описується логічне представлення системи. Логічні структури, такі як класи – це лише заготовки, на основі яких потім будуть визначені фізичні об’єкти [10].

Діаграма класів містить значки, що представляють класи, інтерфейси і їх зв’язки [10].

Можливе створення однієї або декількох діаграм класів, які описують класи верхнього рівня в поточній моделі. Також можливе створення однієї або більше діаграм класів, які описують класи, що містяться в пакетах. Так, діаграма класів сама по собі є пакетом для класів моделі, але можна виділити додаткові пакети для логічної угруповання класів [10].

За допомогою діаграми класів можлива зміна в будь-який момент властивостей будь-якого класу або його зв’язків, і при цьому діаграми або специфікації, пов’язані із змінним класом, будуть автоматично оновлені. Діаграма класів може бути використана як при аналізі готової системи, так і при розробці нової [10].

Перш ніж перейти до проектування діаграми класу системи, слід обрати шаблони (патерни) проектування, що будуть застосовуватись для покращення та спрощення управління функціоналом системи.

Патерн проектування — це типовий спосіб вирішення певної проблеми, що часто зустрічається при проектуванні архітектури програм [13].

На відміну від готових функцій чи бібліотек, патерн не можна просто взяти й скопіювати в програму. Патерн являє собою не якийсь конкретний код, а загальний принцип вирішення певної проблеми, який майже завжди треба підлаштовувати для потреб тієї чи іншої програми [13].

Також шаблони проектування мають власну кваліфікацію.

Патерни відрізняються за рівнем складності, деталізації та охоплення проектованої системи [13].

Найбільш низькорівневі та прості патерни — ідіоми. Вони не дуже універсальні, позаяк мають сенс лише в рамках однієї мови програмування [13].

Найбільш універсальні — архітектурні патерни, які можна реалізувати практично будь-якою мовою. Вони потрібні для проектування всієї програми, а не окремих її елементів [13].

Для даної розроблювальної системи було обрано та реалізовано такі два шаблони проектування: MVT (Model-View-Template) та Фасад.

Оскільки однією з технологій обраних для розробки інформаційної платформи було обрано фреймворк Django, до даного проекту застосовується архітектурний шаблон проектування MVT, що є одним з варіантів традиційного шаблону проектування MVC (Model-View-Controller), який застосовується у сфері веб розробки. Цей шаблон розділяє застосунок на три основні компоненти [14]:

1. Model: Модель виступає в ролі рівня даних вашої програми. Вона визначає структуру вашої бази даних та обробляє логіку, пов'язану з даними. Зазвичай вона представляє таблиці вашої бази даних і відповідає за запити, вставку, оновлення та видалення даних. Моделі Django зазвичай підтримуються реляційними базами даних, такими як MySQL, PostgreSQL, SQLite тощо.

2. View: Представлення обробляє бізнес-логіку та рендеринг інтерфейсу користувача. Воно обробляє запити користувачів, взаємодіє з моделями для отримання даних та передає ці дані шаблонам для відображення. У Django представлення (Views) – це функції або класи Python, які повертають HTTP-відповіді.

3. Template: Шаблон відповідає за представлення даних користувачеві. Він містить статичні частини HTML та спеціальний синтаксис шаблону (наприклад, мову шаблонів Django) для динамічної вставки даних. Шаблони зазвичай складаються з HTML, CSS та JavaScript.

Розглянувши структуру шаблону MVT, було спроектовано діаграму для відображення взаємодії елементів між собою:

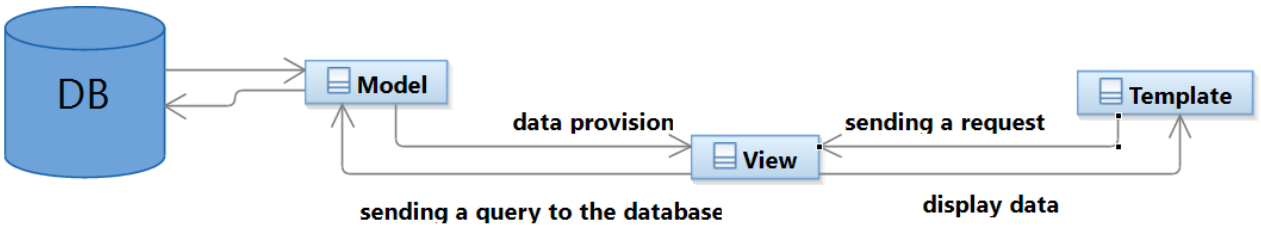

Рисунок 2.3 – UML-діаграма для відображення взаємодії елементів архітектурного шаблону проектування MVT між собою

Інший шаблон проектування, який був використаний у системі був Фасад.

Фасад — це структурний патерн проектування, який надає простий інтерфейс до складної системи класів, бібліотеки або фреймворку. Фасад може бути спрощеним відображенням системи, що не має 100%% тієї функціональності, якої можна було б досягти, використовуючи складну підсистему безпосередньо. Разом з тим, він надає саме ті «фічі», які потрібні клієнтові, і приховує все інше [9].

Фасад корисний у тому випадку, якщо ви використовуєте якусь складну бібліотеку з безліччю рухомих частин, з яких вам потрібна тільки частина [9].

Даний шаблон проектування було застосовано для трьох модулів системи mediaProducts, user та infoPlatformMainApp.

– Для mediaProducts: використовується для надання доступу до класів моделей модуля mediaProducts та опису основної бізнес-логіки роботи з інформацією про медіапродукти.

– Для user: використовується для надання доступу до класів моделей модуля user та опису основної бізнес-логіки роботи з інформацією профіля користувача та коментарями.

– Для infoPlatformMainApp: використовується для надання доступу до класів моделей модуля infoPlatformMainApp та опису основної бізнес-логіки роботи з індустріальними та серверними новинами.

Щоб краще наглядно продемонструвати зв’язки між класами, було спроектовані окремі діаграми класів для вище перелічених модулів системи та більш детально описано основні класи у даних модулях.

У модулі mediaProducts було спроектовано класи, що зберігають інформацію про медіапродукти інформаційної системи та надають до неї доступ, а також клас фасаду, що є централізованим класом та описує основну бізнес-логіку модуля.

Перелік класів для даного модуля та їх опис наведено у таблиці 2.3.

Таблиця 2.3 – Опис класів модуля mediaProducts

Назва класу	Опис (Дієслова і дієслівні конструкції)
MediaProductsViewer	Фасад для модуля mediaProducts (централізований клас для модуля mediaProducts), забезпечує доступ до інформації про твори, публікації, персонажів тощо.
Work	Модель, що представляє загальний твір (manga, аніме, новела), пов'язаний з жанрами, авторами та персонажами.
Genre	Модель, що зберігає жанри творів: визначає назву та опис жанру.
Author	Модель, що зберігає інформацію про авторів: додає ім'я, опис та роки творчості.
Publication	Абстрактний клас, що є базовою моделлю для всіх типів публікацій: визначає назву, обкладинку, посилання та зв'язок із твором.
Manga	Модель, що наслідується від класу Publication, та зберігає інформацію про мангу (японський комікс): додає художника до базової публікації.
Novel	Модель, що наслідується від класу Publication, та зберігає інформацію про новели: додає ілюстратора та видавництво.
Anime	Модель, що наслідується від класу Publication, та зберігає інформацію про аніме (японська анімація): додає режисера та студію.

Продовження таблиці 2.3

PublicationUnit	Абстрактний клас, що є базовою моделллю для одиниці публікації (окремий том або сезон), яку можна коментувати.
Volumes	Модель, що наслідується від класу PublicationUnit, та описує томи (для манги та новел).
Seasons	Модель, що наслідується від класу PublicationUnit, та описує сезони (для аніме).
Characters	Модель, що містить інформацію про персонажів медіа-продуктів.
CharacterPublication	Модель, що зберігає для персонажа відповідні описи в залежності від типу публікації.
Status	Перелічує можливі статуси публікацій: виходить, завершений, закинуто, анонсовано, призупинено.
PuplicationType	Перелічує типи публікації творів: манга, новела, аніме.

Також, було спроектовано діаграму класів для архітектурної моделі шаблону «Фасад» з описами зв’язків між класами модуля mediaProducts, що наведена на рисунку 2.4.

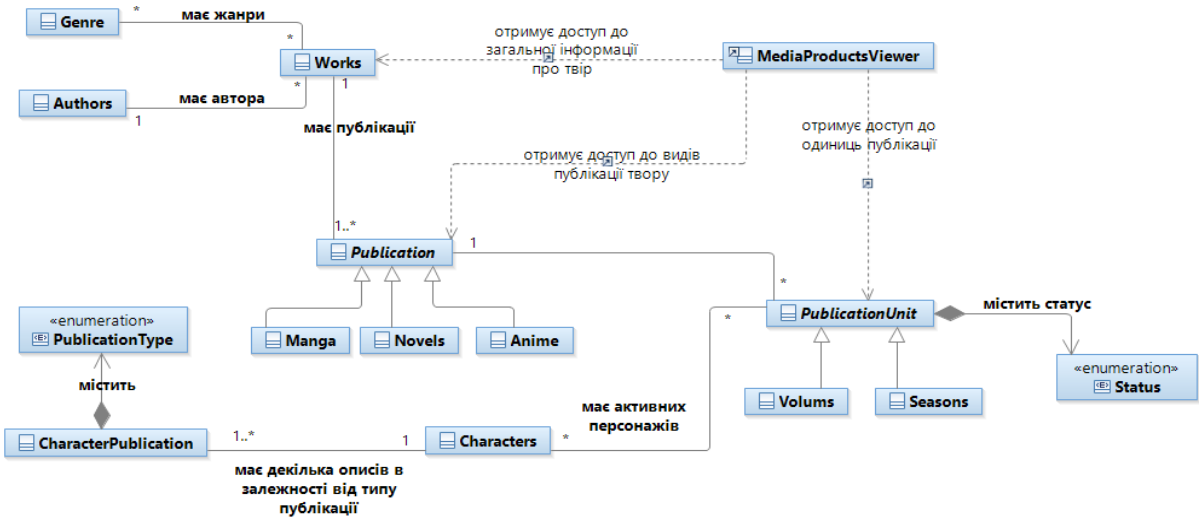

Рисунок 2.4 – UML-діаграма ієрархії класів архітектурної моделі (шаблона «Фасад») для модуля mediaProducts

У модулі user було спроектовано класи, що зберігають інформацію про користувачів інформаційної системи та їх коментарів, надаючи до них доступ, а також клас фасаду, що є централізованим класом та описує основну бізнес-логіку модуля.

Перелік класів для даного модуля та їх опис наведено у таблиці 2.4.

Таблиця 2.4 – Опис класів для модуля user

Назва класу	Опис (Дієслова і дієслівні конструкції)
UserViewer	фасад для модуля user (централізований клас модуля user), що надає доступ до користувацьких даних і коменталів користувача.
Profile	Розширює модель користувача: додає аватар, групи, дозволи.
Comment	Модель, що зберігає коментарі користувачів до одиниць публікацій (томів, сесонів).

Також, було спроектовано діаграму класів для шаблона Фасад з описами зв’язків між класами модуля, що наведена на рисунку 2.5.

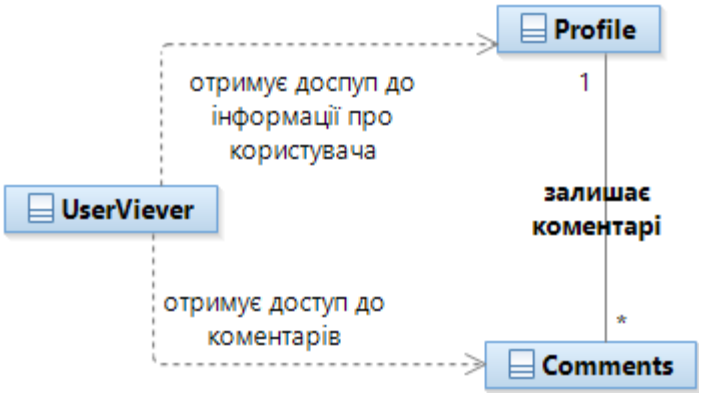

Рисунок 2.5 – UML-діаграма ієрархії класів архітектурної моделі (шаблона «Фасад») для модуля user

У модулі infoPlatformMainApp було спроектовано класи, що зберігають інформацію про індустріальні та серверні новини системи, надаючи до них доступ,

а також клас фасаду, що є централізованим класом та описує основну бізнес-логіку модуля.

Перелік класів для даного модуля та їх опис наведено у таблиці 2.5.

Таблиця 2.5 – Опис класів модуля infoPlatformMainApp.

NewsViewer	фасад для модуля infoPlatformMainApp (централізований клас для модуля infoPlatformMainApp), що надає доступ до новин про медіа-продукти та сервері новини.
BasicNews	Абстрактний клас, що є базовою моделлю для новин: зберігає автора, контент, дату створення.
ServerNews	Модель, що наслідується від класу BasicNews, та зберігає серверні новини: додає тип новини (інформація, помилка, попередження тощо).
WorkNews	Модель, що наслідується від класу BasicNews, та зберігає новини, пов’язані з творами: додає зображення, посилання, заголовок.
NewsType	Перелічує типи новин: інформація, попередження, помилка, оновлення.

Також, було спроектовано діаграму класів для шаблона Фасад з описами зв’язків між класами модуля, яка наведена на рисунку 2.6.

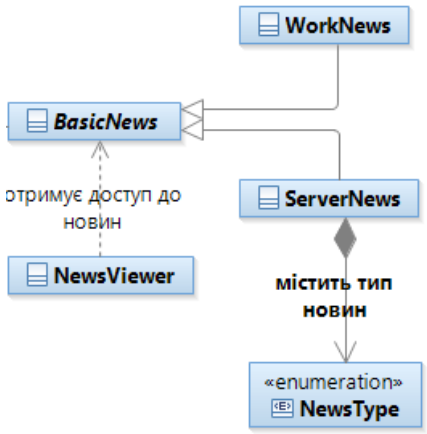

Рисунок 2.6 – UML-діаграма ієрархії класів архітектурної моделі (шаблона «Фасад») для модуля infoPlatformMainApp

Після того, як було описано шаблони, які використані при розробці даної інформаційної платформи та описом класів, що спроектували у модулях системи, було спроектовано загальну діаграму класів, що демонструє зв’язки між усіма класами інформаційної системи. Дана діаграма наведена на рисунку 2.7.

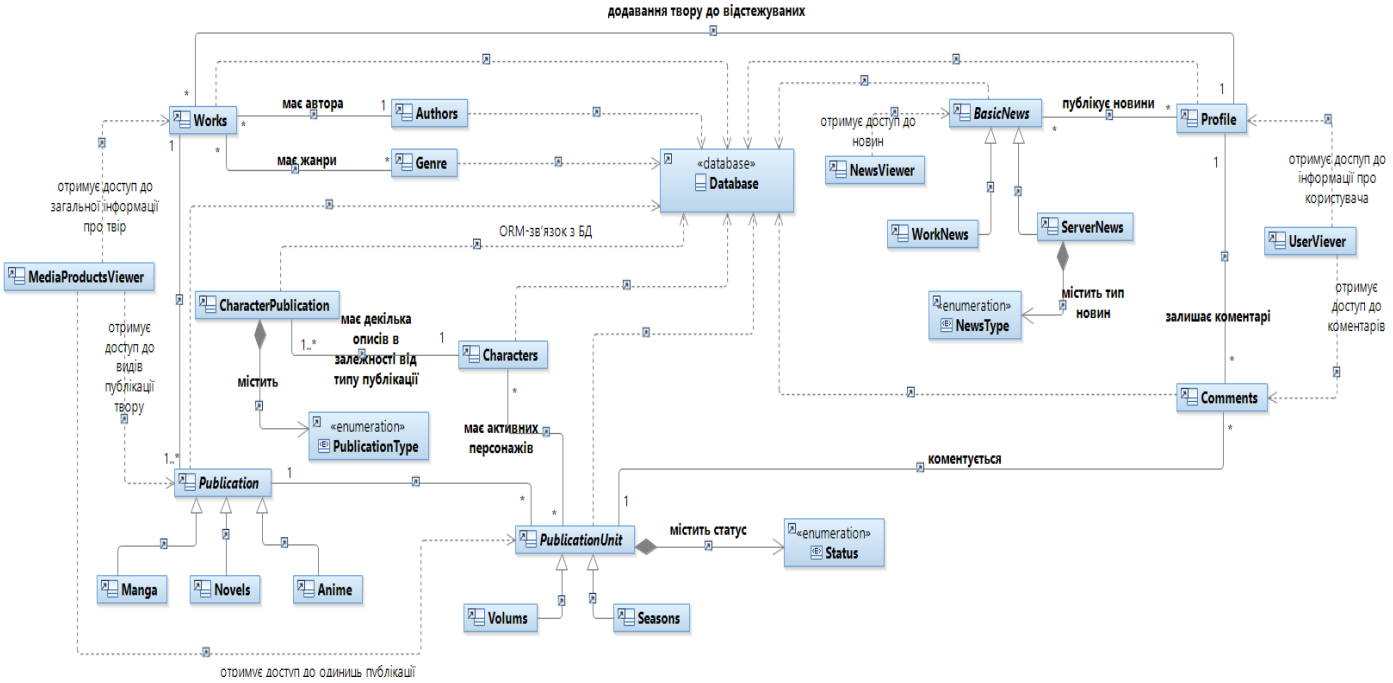

Рисунок 2.7 – Загальна UML-діаграма ієрархії класів інформаційної системи

Отже, у даному підрозділі було спроектовано детальні UML-діаграми ієрархії класів. Це включає як загальну діаграму класів системи в цілому, що відображає архітектуру та взаємозв'язки основних компонентів, так і окремі діаграми для кожного модуля. Такий підхід дозволив забезпечити чітку структуру, розподіл функціоналу та ефективну взаємодію між частинами системи, що є важливим для її подальшої розробки.

2.4 Детальне проектування класів підсистем

Після того, як були спроектовані UML-діаграми ієрархії класів системи, слід описати більш детальніше основні класи підсистеми платформи.

Для модуля mediaProducts, основними класами є Works, Publication, PublicationUnit та MediaProductsViewer, що наведені на рисунку 2.8.

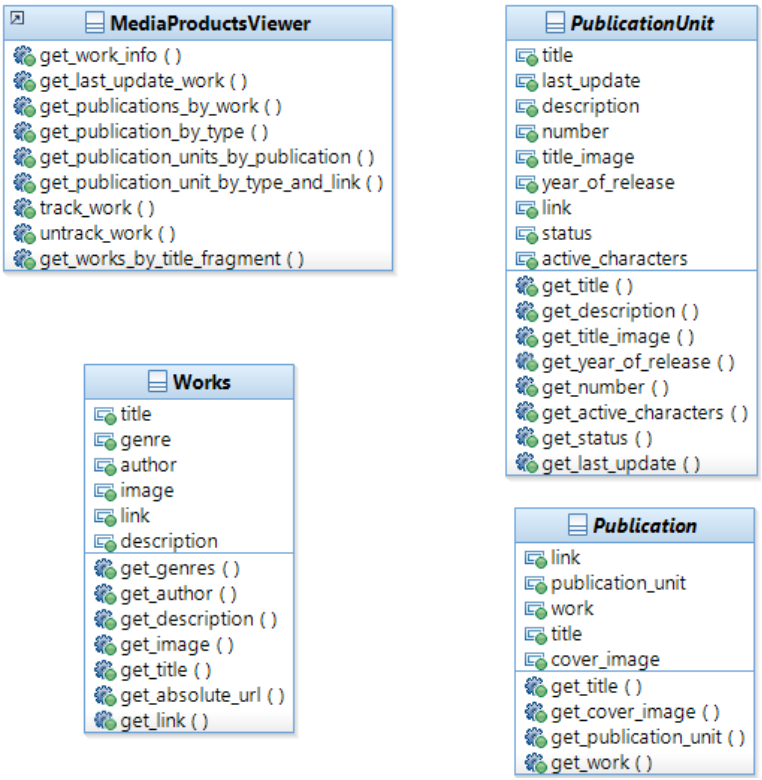

Рисунок 2.8 – Основні класи модуля mediaProducts

Детальний опис основних класів модуля mediaProducts:

1. Клас Works представляє собою певну роботу та зберігає основну інформацію про медіапродукт таку, як назву, зображення, опис, жанри, автор, а також за допомогою методів при потребі надає до неї доступ.

Атрибути класу:

– title – призначений для зберігання назви роботи;

– image – призначений для зберігання посилання на зображення роботи;

– description – призначений для зберігання опису роботи;

– genre – зв’язок багато до багатьох до моделі Genre, та містить перелік жанрів роботи;

– author – вказує на автора роботи;

– link – призначений для зберігання посилання, за яким буде здійснюватися перехід на сторінку медіапродукту.

Методи класу:

– get_genres – метод, що забезпечує надання жанрів медіапродукту;

– get_author – метод, що забезпечує надання автора медіапродукту;

– get_descripttion – метод, що забезпечує надання опису роботи;

– get_image – метод, що забезпечує надання посилання на зображення роботи;

– get_title – метод, що забезпечує надання назви роботи;

– get_absolute_url – метод, що надає повне посилання для переходу на сторінку роботи.

– get_link – метод, що забезпечує надання посилання на роботу.

2. Клас Publication є абстрактним класом, що представляє собою певну публікацію та успадковується класами Manga, Novel та Anime. Описує поля, що використовуються в успадкованих класах для зберігання інформації про публікацію медіапродукту таку, як назву та зображення, а також за допомогою методів при потребі надає до неї доступ.

Атрибути класу:

– title – призначений для зберігання назви публікації;

– cover_image – призначений для зберігання посилання на зображення роботи;

– work – вказує на роботу, якій належить публікація;

– publication_unit – має зв’язок багато до багатьох до абстрактної моделі PublicationUnit та містить перелік одиниць даної публікації;

– link – призначений для зберігання посилання, за яким буде здійснюватися перехід на сторінку публікації медіапродукту.

Методи класу:

– get_cover_image – метод, що забезпечує надання посилання на зображення публікації;

– get_title – метод, що забезпечує надання назви роботи;

– get_publication_unit – метод, що надає одиниці публікації;

– get_work – метод, що надає роботу, з якою зв’язана публікація;

– get_link – метод, що забезпечує надання посилання на публікаію.

3. Клас PublicationUnit є абстрактним класом, що представляє собою певну одиницю публікації та успадковується класами Volumes та Seasons. Описує поля, що використовуються в успадкованих класах для зберігання інформації про одиницю публікації медіапродукту таку, як назву та зображення, опис, номер одиниці та рік виходу, а також за допомогою методів при потребі надає до неї доступ.

Атрибути класу:

– title – призначений для зберігання назви роботи;

– title_image – призначений для зберігання посилання на зображення роботи;

– description – призначений для зберігання опису одиниці публікації;

– year_of_release – призначений для зберігання року випуску одиниці публікації;

– number – призначений для зберігання номеру одиниці публікації;

– last_update – зберігає дату останнього оновлення;

– status – зберігає статус одиниці публікації;

– active_characters – має зв’язок багато бо багатьох до моделі Characters, та містить перелік активних персонажів у даній одиниці публікації;

– link – призначений для зберігання посилання, за яким буде здійснюватися перехід на сторінку медіапродукту.

Методи класу:

– get_descripttion – метод, що забезпечує надання опису роботи;

– get_title_image – метод, що забезпечує надання посилання на зображення роботи;

– get_title – метод, що забезпечує надання назви роботи;

– get_number – метод, що надає номер одиниці публікації.

– get_active_characters – метод, що надає активних персонажів одиниці публікації;

– get_status – метод, що надає статус одиниці публікації;

– get_link – метод, що забезпечує надання посилання на роботу.

4. Клас MediaProductsViewer є централізованим класом модуля mediaProducts, що виконує роль класу Фасаду, та описує бізнес-логіку для роботи з медіапродуктами.

Методи класу:

– get_work_info – повертає детальну інформацію про твір (роботу);

– get_last_update_work – метод, що повертає список творів, відсортованих за датою останнього оновлення серед їхніх публікацій (манґа, аніме, новел);

– get_publications_by_work – метод, що повертає всі публікації (манґа, аніме, новела), пов’язані з конкретним твором;

– get_publication_by_type – метод, що повертає публікацію певного типу;

– get_publication_units_by_publication – метод, що повертає одиниці публікації, пов’язані з конкретною публікацією;

– get_publication_unit_by_type_and_link – метод, що повертає одиницю публікації за типом та посиланням;

– track_work – метод, що додає роботу до списку відстежуваних користувачем, якщо він ще не відстежується;

– untrack_work – метод, що видаляє твір зі списку відстежуваних користувачем, якщо він там є;

– get_works_by_title_fragment – метод, що повертає список творів, назви яких містять переданий фрагмент тексту.

Для модуля user, основними класами є Profile та UserViewer, що наведені на рисунку 2.9.

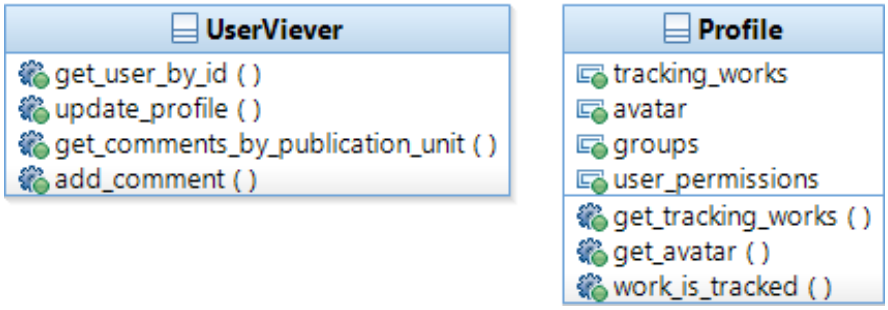

Рисунок 2.9 – Основні класи модуля user

Детальний опис основник класів модуля user:

1. Клас Profile є класом, що представляє профіль користувача, та зберігає розширену інформацію про користувача: відстежуванні роботи, аватар користувача, група до якої належить користувач та його дозволи.

Атрибути касу:

– avatar – зберігає посилання на зображення аватару користувача;

– groups – зберігає інформацію про групи користувачів, в яких перебуває користувач;

– user_permision – зберігає дозволи користувача;

– tracking_works – зберігає перелік творів (робіт), які користувач відстежує.

Методи класу:

– get_tracking_works – метод, що повертає всі відстежувані користувачем роботи;

– get_avatar – метод, що повертає посилання на аватара;

– work_is_tracked – метод, що перевіряє, чи відстежується користувачем вказаний твір.

2. Клас UserViewer є централізованим класом модуля user, та є класом фасаду, що описує бізнес-логіку роботи з користувачами та їх коментарями: надання певного користувача, оновлення інформації профілю та робота з коментарями.

Методи класу:

– get_user_by_id – метод, що повертає певного користувача за його id;

– update_profile – метод, що оновлює інформацію профіля, а саме: аватар користувача, його ім’я та пошту;

– get_comments_by_publication_unit – метод, що повертає всі коментарі, прив'язані до конкретної одиниці публікації;

– add_comment – метод, що додає коментар користувача до одиниці публікації.

Для модуля infoPlatformMainApp, основними класами є BasicNews та NewsViewer, що наведені на рисунку 2.10.

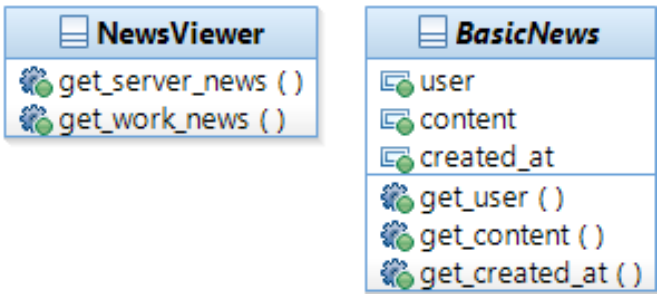

Рисунок 2.10 – Основні класи модуля infoPlatformMainApp

Детальний опис класів для модуля infoPlatformMainApp:

1. Клас BasicNews є абстрактним класом, якого успадковують каси WorksNews та ServerNews, та представляє новини розроблювальної платформи, та зберігає загальну інформацію про новини: користувача (адміністратора), що створив дану новину, вміст новини та дату створенння, а також описує методи, що надають доступ да даної інформації.

Атрибути касу:

– user – зберігає користувача, який створив дану новину;

– content – зберігає вміст новини;

– created_at – зберігає дату створення новини.

Методи класу:

– get_user – метод, що повертає користувача, який створив дану новину;

– get_content – метод, що повертає вміст новини;

– get_created_at – метод, що повертає дату створення новини.

2. Клас NewsViewer є централізованим класом модуля infoPlatformMainApp, та є класом фасаду, що описує бізнес-логіку роботи новинами системи: надає індустріальні та серверні новини.

Методи класу:

– get_server_news – метод, що повертає ряд серверних новин;

– get_work_news – метод, що повертає ряд новин індустрії.

Також, на загальній діаграмі класів присутній клас Database.

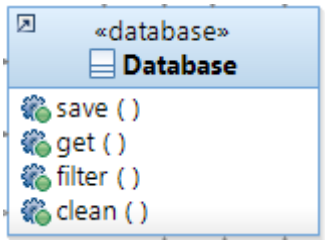

Рисунок 2.11 – Умовний клас Database

Клас Database є умовним (абстрактним) представленням бази даних , що взаємодіє з іншими ORM-моделями системи та забезпечує основну бізнес-логіку доступу до даних. Даний клас наведено на рисунку 2.11.

Методи класу:

– save – метод, що описує функцію зберігання об'єкта моделі до бази даних;

– clean – метод, що описує перевірку на коректність даних перед збереженням (валідація);

– filter – метод, що здійснює фільтрацію об'єктів за переданими параметрами;

– get – метод, що повертає один об'єкт, що відповідає критеріям пошуку.

Отже, у даному підрозділі було описано, що таке діаграма класів та для чого вона використовується. Вибрано шаблони проектування, що використанні при розробці даної архітектури системи, а також описано загальну діаграму ієрархії класів, окремі діаграми для модулів та спроектовані класи підсистем.

2.5 Проектування сценаріїв ВВ на основі UML-діаграм послідовності

Після того, як було описано класи системи та спроектовані загальна UML- діаграма ієрархії класів, а також діаграм для обраних шаблонів проектування, наступним кроком слід описати їх взаємодію при виконуванні описаних варіантів використання для користувачів, тобто сценаріїв роботи системи. Для вирішення цієї задачі було використано діаграми послідовностей, які дозволяють чітко простежити послідовність викликів методів між об'єктами та обмін повідомленнями під час виконання ключових варіантів використання.

Діаграми послідовностей UML – це діаграми взаємодії, які детально описують, як виконуються операції. Вони фіксують взаємодію між об'єктами в контексті співпраці. Діаграми послідовностей орієнтовані на час і візуально показують порядок взаємодії за допомогою вертикальної осі діаграми для представлення часу, часу надсилання повідомлень і часу їх надсилання [15].

Діаграми послідовностей фіксують [15]:

– взаємодія, що відбувається у співпраці, яка реалізує або варіант використання, або операцію (діаграми екземплярів або загальні діаграми);

– взаємодії високого рівня між користувачем системи та системою, між системою та іншими системами або між підсистемами (іноді відомі як діаграми послідовності систем).

Для даної розроблювальної інформаційної системи було вирішено спроектувати дві діаграми послідовності, що описують сценарії роботи основного функціоналу платформи.

Першим сценарієм, який було вирішено продемонструвати на діаграмі послідовності, є отримання основної інформації про твір (роботу): дані що зберігає модель Work та наявні публікації роботи. Даний сценарій є описом варіанту використання користувача «Перегляд основної інформації про роботу».

Пояснення послідовних кроків виконання сценарію:

– Спочатку представлення (Views) звертається через метод get_work_info(link) до класу фасаду MediaProductsViewer, подаючи посилання на потрібну роботу;

– Далі фасад MediaProductsViewer звертається до моделі Works через метод get(link), щоб отримати об’єкт необхідної роботи, що відповідає переданому посиланню;

– Після отримання необхідного твору (роботи), представлення знову звертається до фасаду MediaProductsViewer через метод get_publication_by_work(link), передаючи посилання на твір, щоб отримати усі наявні публікації даної роботи;

– Далі фасад MediaProductsViewer послідовно звертається до моделей публікацій Manga, Novel та Anime через метод filter(link), що фільтрує публікації за посилання на роботу, та повертає наявні публікації роботи.

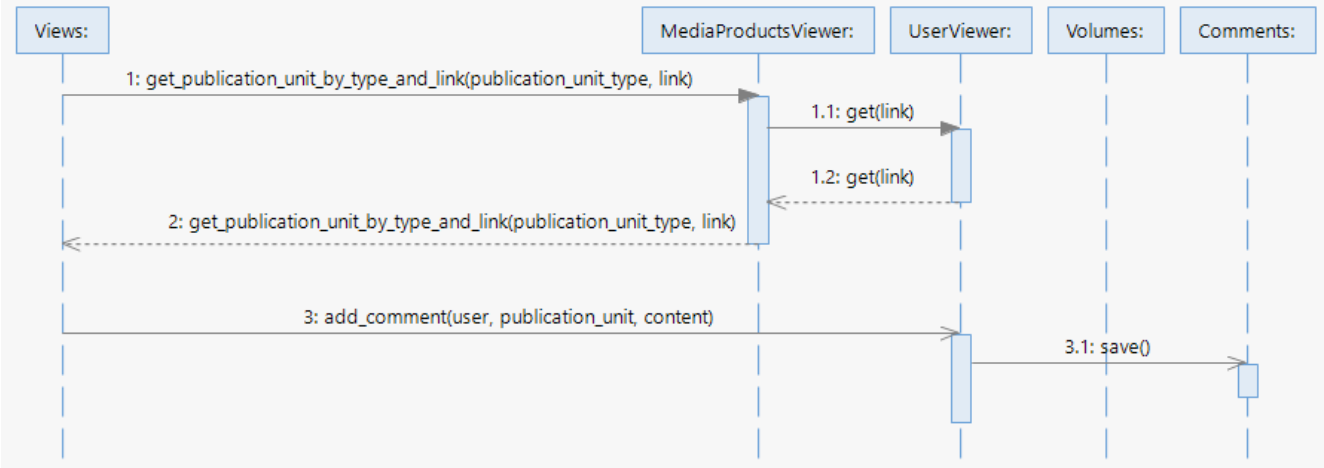

Рисунок 2.12 – Діаграма послідовності для сценарію отримання основної інформації про роботу

Наступним сценарієм, який вирішено продемонструвати, обрано додавання користувачем коментаря до одиниці публікації. Даний сценарій є основним елементом інтерактивності розроблювальної інформаційної платформи та описує варіант використання користувача «Додавання коментаря до певного тому або сезону роботи».

Опис послідовних кроків виконання сценарію:

– Спочатку представлення (Views) звертається через метод get_publication_unit_by_type_and_link(publication_unit_type, link) до класу фасаду MediaProductsViewer, подаючи посилання на необхідну одиницю публікації та тип одиниці публікації (у даному випадку це тип для тому (volume));

– Далі фасад MediaProductsViewer звертається до моделі Volumes через метод get(link), щоб отримати об’єкт необхідного тому публікації, що відповідає переданому посиланню;

– Після отримання необхідного тому публікації, представлення звертається до фасаду UserViewer через метод додавання коментаря add_comment(user, publication_unit, content), передаючи користувача, що додає коментар, том, до якого додається коментар, та вміст коментаря;

– Далі фасад UserViewer через метод save моделі Comments зберігає новий коментар у базі даних.

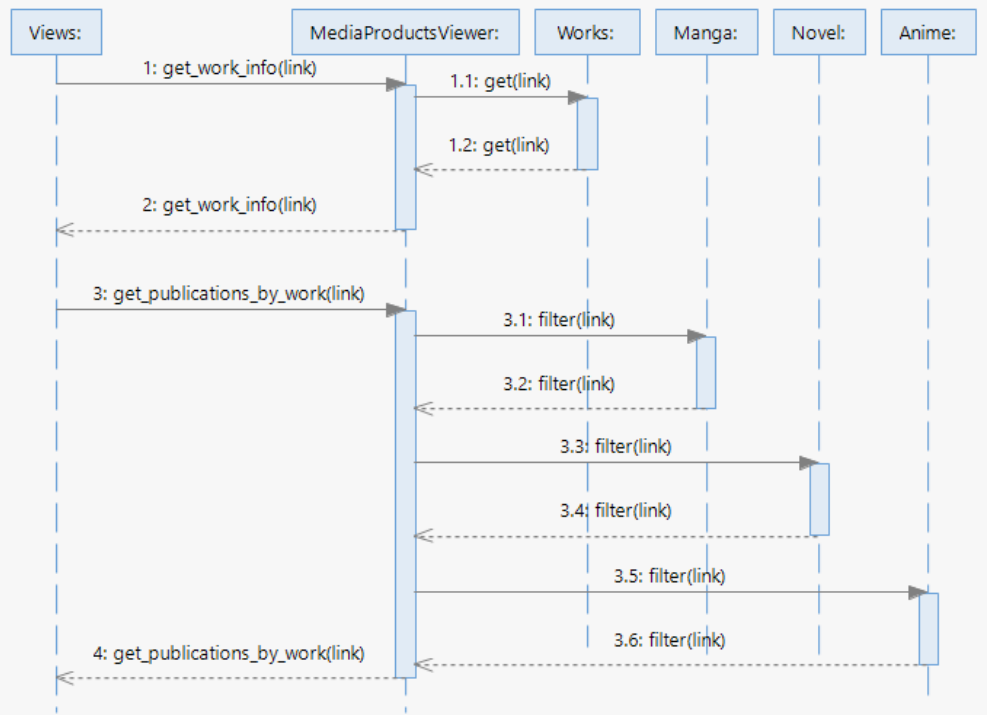

Рисунок 2.13 – Діаграма послідовності сценарію додавання користувачем коментаря до одиниці публікації

Отже, у даному підрозділі «Детальне проектування класів підсистем» було розглянуто що таке діаграма послідовності, для чого вона використовується, а також розглянуто два основні сценарії роботи системи: отримання основної інформації про твір (роботу) та додавання користувачем коментаря до одиниці публікації, що реалізують варіанти використання користувача «Перегляд основної інформації про роботу» та «Додавання коментаря до певного тому або сезону роботи», що наведені у таблиці 2.1. До даних сценаріїв було спроектовано відповідні UML-діаграм послідовності та описано їх послідовність кроків.

2.6 Підсумки розділу 2

Отже, у розділі 2 «Проектування системи» було обрано інструмент для проектування системи IBM RSAD, що є одним з найкращих рішень на глобальному ринку інструментів проектування для моделювання систем.

Далі було виділено та описано варіанти використання, пакети (модулі), на які було розділено систему та функціонал інформаційної платформи, обрано шаблони проектування «MVT» та «Фасад» для модулів mediaProducts, user і infoPlatformMainApp, що допомагають краще розділити та реалізувати бізнес-логіку системи та спроектовано і описано відповідні класи та UML-діаграми. Також, для кращого розуміння спроектованої системи, до кожного вище переліченого модуля, було детальніше описано їх основні класи.

У кінці розділу розглянуто два основних сценарії роботи системи на основі UML-діаграм послідовності: отримання основної інформації про роботу та додавання користувачем коментаря до одиниці публікації, що відповідають варіантам використання користувача «Перегляд основної інформації про роботу» та «Додавання коментаря до певного тому або сезону роботи» відповідно.

РОЗДІЛ 3 КОНСТРУЮВАННЯ ТА ТЕСТУВАННЯ ПЛАТФОРМИ

3.1 Базова структура, налаштування проекту та опис моделей і фасадів системи

Після завершення етапу проектування інформаційної системи розроблювальної платформи огляду японських медіапродуктів було розпочато етап розробки. Початком розробки було створення базової структури та опис налаштувань системи засобами фреймворку Django.

Розгортання проекту розпочалося з ініціалізації проекту за допомогою команди «django-admin startproject», що створило основний модуль infoPlatform та файл manage.py [16]:

– manage.py: Утиліта командного рядка, яка дозволяє вам взаємодіяти з цим проектом Django різними способами.

– infoPlatform/: Каталог, який є фактичним пакетом Python для вашого проєкту. Його назва — це назва пакета Python, яку вам потрібно буде використовувати для імпорту будь-чого всередині нього (наприклад, infoPlatform.urls);

– infoPlatform /__init__.py: Порожній файл, який повідомляє Python, що цей каталог слід вважати пакетом Python;

– infoPlatform/settings.py: Налаштування/конфігурація для цього проєкту Django;

– infoPlatform/urls.py: Оголошення URL-адрес для цього проєкту Django; «зміст» вашого сайту на базі Django;

– infoPlatform/asgi.py: Точка входу для веб-серверів, сумісних з ASGI, для обслуговування вашого проєкту;

– infoPlatform/wsgi.py: Точка входу для веб-серверів, сумісних із WSGI, для обслуговування вашого проєкту;

Після того, як було створено основний модуль, за допомогою наступної команди «startapp» було створено необхідні додатки, що представляють модулі

(пакети) і містять бізнес-логіку платформи, а саме: mediaProducts, user та infoPlatformMainApp. Дані додатки мають наступну структуру:

– __init__.py: файл, що позначає каталог як Python-пакет, необхідний для імпорту додатку;

– admin.py: файл, що використовується для реєстрації описаних моделей у адміністративній панелі Django, що дозволяє зручно керувати вмістом бази даних через інтерфейс адміністратора;

– apps.py: містить конфігурацію додатку, який Django використовує для реєстрації додатку в системі;

– facade.py: файл, що реалізує шаблон проектування «Фасад», забезпечуючи централізований інтерфейс для взаємодії з кількома моделями чи сервісами всередині додатку;

– models.py: файл, що містить описані моделі бази даних додатку;

– urls.py: файл, що містить маршрути (URL-шляхи), які відповідають функціоналу цього додатку.

– views.py: файл, що містить логіку обробки запитів.

– migrations/: директорія, шо містить міграції, які автоматично генеруються для створення та зміни структури бази даних відповідно до описаних моделей додатку.

– static/: каталог для статичних файлів (зображень, стилів, скриптів), що, пов’язані із даним додатком.

– templates/: директорія для HTML-шаблонів веб-сторінок платформи.

– tests/: каталог для тестів, у якому реалізуються модульні тести бізнес-логіки додатку.

Після того, як було створено базову структуру проекту, потрібно перейти до її конфігурації у файлі infoPlatform/settings.py, та описати основні налаштування системи.

З самого початку потрібно описати INSTALLED_APPS, що містить назви всіх програм Django, активованих у цьому екземплярі Django [17].

За замовчуванням INSTALLED_APPS містить такі програми, всі з яких постачаються з Django [17]:

– django.contrib.admin – Сайт адміністратора. Ви скоро ним користуватиметесь;

– django.contrib.auth – Система автентифікації;

– django.contrib.contenttypes – Фреймворк для типів контенту;

– django.contrib.sessions – Структура сесії;

– django.contrib.messages – Структура обміну повідомленнями;

– django.contrib.staticfiles – Фреймворк для керування статичними файлами.

Ці програми включені за замовчуванням для зручності у поширеному випадку [17].

Крім даних, уже включених за замовченням програм, додатково потрібно підключити необхідні інструменти (filebrowser, ckeditor, ckeditor_uploader, mptt і compressor для зручного керування файлами в адмінці, візуального редагування контенту, завантаження файлів через редактор, роботи з ієрархічними структурами (деревами) даних та оптимізації та збирання CSS/JS ресурсів) та створені попередньо додатки.

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'infoPlatformMainApp',
    'apps.mediaProducts',
    'apps.user',
    'filebrowser',
    'ckeditor',
    'ckeditor_uploader',
    'mptt',
    'compressor',
]
```

Рисунок 3.1 – Опис конфігурації INSTALLED_APPS для розроблювальної системи

Після цього, було описано налаштування підключення до бази даних, що залежить від середовища запуску:

– У локальному режимі використовується вбудована база даних SQLite, яка зручна для тестування системи, оскільки забезпечує ізольоване середовище для тестування функціональності;

```
if LOCAL:
    DATABASES = {
        'default': {
            'CONN_MAX_AGE': None,
            'ENGINE': 'django.db.backends.sqlite3',
            'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
            # 'NAME': BASE_DIR / 'db.sqlite3',
            'OPTIONS': {
                'timeout': 20,
            }
        }
    }
```

Рисунок 3.2 – Налаштування бази даних для локального середовища

– У виробничому середовищі застосовується MySQL, параметри підключення до якої зчитуються з змінних середовища, що зазначені у файлі «.env».

```
else:
    DATABASES = {
        'default': {
            'ENGINE': 'django.db.backends.mysql',
            'NAME': os.environ.get('MYSQL_DATABASE', 'database'),
            'USER': os.environ.get('MYSQL_USER', 'root'),
            'PASSWORD': os.environ.get('MYSQL_PASSWORD', 'password'),
            'HOST': os.environ.get('MYSQL_HOST', 'localhost'),
            # 'PORT': os.environ.get('MYSQL_PORT', '3306'),
            'OPTIONS': {
                'charset': 'utf8',
            },
        }
    }
```

Рисунок 3.3 – Налаштування бази даних для виробничого середовища

Оскільки було описано підключення до бази даних, потрібно описати файл «.env», що містить основні змінні середовища проекту для середовища виконання, підключення бази даних та створення основного адміністратора системи.

```
LOCAL=0
DEBUG=1

COMPRESS_ENABLED=1

MYSQL_HOST=localhost
MYSQL_DATABASE=JapanInfoMediaPlatform
MYSQL_USER=Rev128
MYSQL_PASSWORD=ZanchukBD24

DJANGO_SUPERUSER_USERNAME=admin
DJANGO_SUPERUSER_PASSWORD=admin
DJANGO_SUPERUSER_EMAIL="admin@admin.com"
```

Рисунок 3.4 – Опис основних змінних середовища проекту

Також було додано налаштування локалізації та часової зони проекту.

```
LANGUAGE_CODE = 'uk'
DEFAULT_CHARSET = 'utf-8'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_TZ = True
```

Рисунок 3.5 – Налаштування локалізації та часової зони

Далі, у файлі urls.py модуля infoPlatform слід описати основні URL-маршрути системи: адміністративна панель, медіасховище та URL-маршрути інших модулів.

```
urlpatterns = [
    path('admin/filebrowser/', site.urls),
    path('admin/', admin.site.urls),
    re_path(r'^media/(?P<path>.*)$', serve, {'document_root': settings.MEDIA_ROOT}),
    path("", include("infoPlatformMainApp.urls")),
    path("", include("apps.mediaProducts.urls")),
    path("", include("apps.user.urls")),

] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

Рисунок 3.6 – Опис основних URL-маршрутів системи

Після того, як було створено базову структуру та описано налаштування системи, наступним етапом є опис раніше спроектованих класів моделей та фасадів платформи:

i. У файлі models.py для кожного модуля описано всі необхідні класи моделей, основні з яких є Work, Publication, PublictionUnit, Profile та BasicNews;

– У файлі facade.py для кожного модуля описано класи фасадів MediaProductsViewer, UserViewer та NewsViewer, що реалізують основну бізнес-логіку системи.

Реалізацію коду основних класів наведено у додатку Б.

Окрему увагу можна приділити класу Profile, що розширює функціонал користувача системи, оскільки фреймворк Django надає власний клас користувача під назвою AbstractUser при підключенні пакету django.contrib.auth.models.

```
class Profile(AbstractUser):  Zanchuk Taras +1 *
    avatar = FileBrowseField(verbose_name='Аватар', max_length=255,
                             directory='user_avatar/', extensions=['.jpg', '.jpeg', '.png'], blank=True, null=True)
    groups = models.ManyToManyField(Group, related_name="user_profiles",
                                    blank=True, verbose_name='Групи')
    user_permissions = models.ManyToManyField(Permission, related_name="user_permissions_set",
                                              blank=True, verbose_name='Дозволи')
    tracking_works = models.ManyToManyField(Work, verbose_name="Відстежувані твори",
                                            blank=True, related_name='tracking_works')

    def get_tracking_works(self):  Zanchuk Taras
        return self.tracking_works.all()

    def get_avatar(self):  1 usage (1 dynamic)  Zanchuk Taras
        return self.avatar.url if self.avatar else None

    def work_is_tracked(self, work):  3 usages (3 dynamic)  Zanchuk Taras
        return self.tracking_works.filter(id=work.id).exists()
```

Рисунок 3.7 – Опис класу моделі Profile

Після опису класів моделей, їх слід реалізувати у базі даних. Для цього відкривши командного рядка і виконавши команду «python manage.py makemigrations» буде автоматично створено необхідні файли міграцій до кожного модуля, де було описано моделі бази даних. Далі, виконавши команду «python manage.py migrate», за інструкціями, що раніше згенерувались у файлах міграцій, у базі даних було створено необхідні таблиці, які відповідають описаним вище моделям системи.

Описавши моделі, їх далі слід винести у адміністративну панель. Для цього у файлі admin.py для кожного модуля системи слід імпортувати модуль адмін-панелі Django django.contrib.admin, що надає наступний функціонал [18]:

– Клас ModelAdmin – це представлення моделі в інтерфейсі адміністратора. Зазвичай вони зберігаються у файлі з іменем admin.py у вашій програмі.

– Декоратор register – декоратор для реєстрації ваших ModelAdmin класів.

Як приклад, на рисунку 3.7 було продемонстровано опис додавання класу Work у адміністративну панель.

```
@admin.register(Work)  Zanchuk Taras
class WorkAdmin(admin.ModelAdmin):
    list_display = ['title', 'author', 'get_genre']
    search_fields = ['title', 'author']
    list_display_links = ['title']
    autocomplete_fields = ['author']
    list_filter = ['genre']
    filter_horizontal = ['genre']
    ordering = ('title',)

    def get_genre(self, obj):  1 usage  Zanchuk Taras
        return ", ".join([genre.get_name() for genre in obj.get_genres()])
    get_genre.short_description='Жанри'
```

Рисунок 3.8 – Опис додавання класу Work у адміністративну панель

Після того, як усі необхідні моделі додали до адміністративної панелі, за допомогою команди «python manage.py createsuperuser --noinput» створюється супер користувач (адміністратор) системи, використовуючи раніше описані у фалі «.evn» дані для адміністратора.

Далі запустивши платформу за допомогою команди «python manage.py runserver» та перейшовши за шляхом /admin/, можна перейти на адміністративну панель, де будуть відображатись виведені моделі системи.

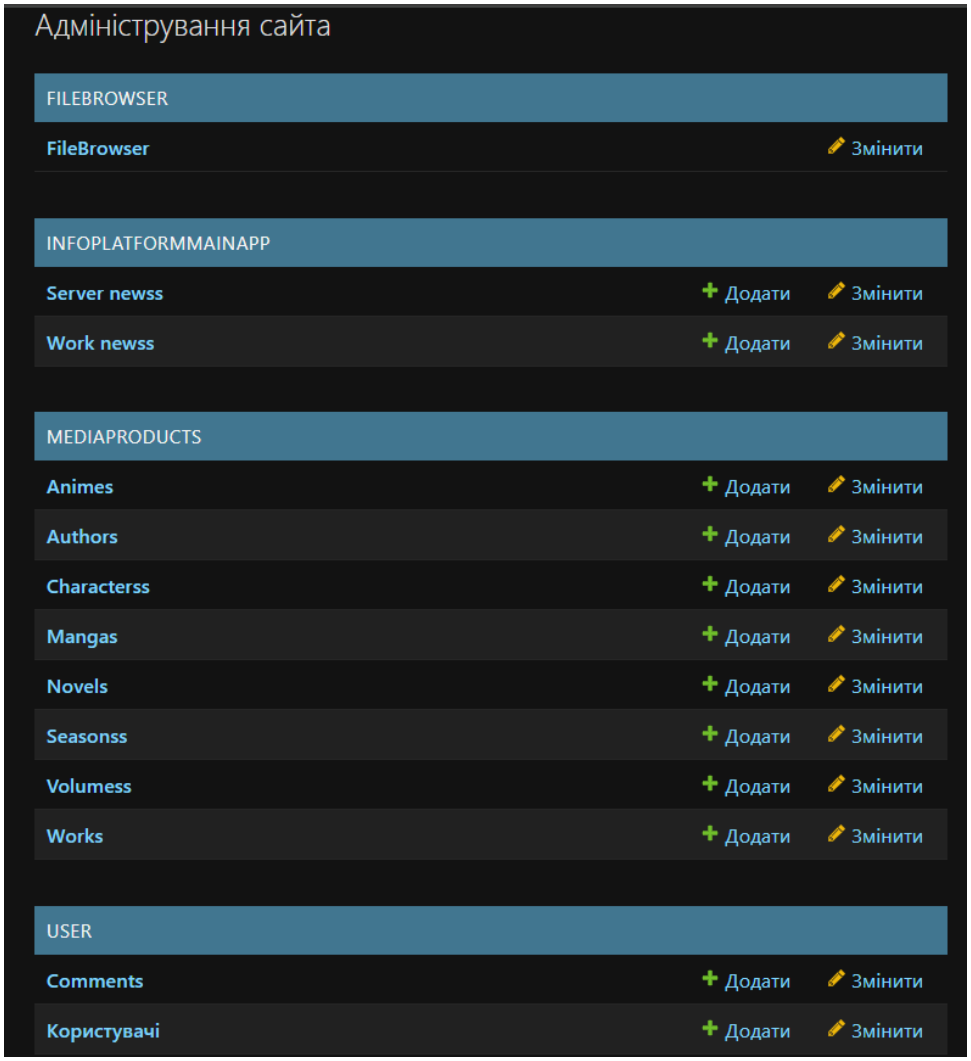

Рисунок 3.9 – Адміністративна панель

Отже, у даному підрозділі було описано базову структуру системи, що включає усі модулі платформи, проведено базову конфігурацію, описано вище спроектовані моделі та додано їх до адміністративної панелі.

3.2 Створення та реалізація користувацького інтерфейсу

Після того, як було описано моделі системи та фасади модулів, що описують основну бізнес-логіку системи, можна перейти до наступного, одного з ключових етапів розробки програмного забезпечення – створення та реалізація користувацького інтерфейсу, що є головним елементом взаємодії користувача з системою.

Оскільки дана розроблювальна інформаційна платформа є веб-додатком, об’єктами користувацького інтерфейсу є веб-сторінки. Для їх розробки було вибрано онлайн-сервіс Figma.

Кожна сторінка платформи (крім сторінок реєстрації та авторизації) містить header, у якому міститься функціонал навігації платформи, та footer, де зберігається інформація про платформу та контактні дані. Шаблони для header та footer наведено на рисунках 3.10 та 3.11.

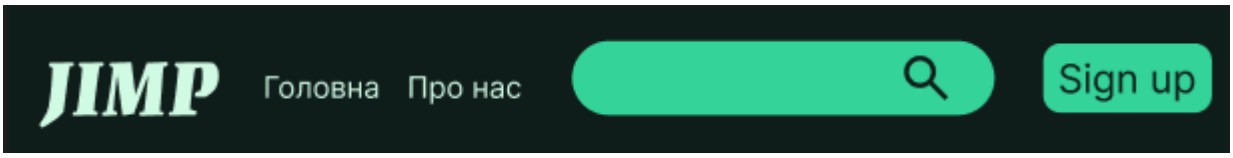

Рисунок 3.10 – Шаблон для header

Рисунок 3.11 – Шаблон для footer

Для кожного модуля системи були розроблені відповідні сторінки:

– Для модуля infoPlatformMainApp була розроблена головна сторінка платформи, що відображає новини індустрії, серверні новини та останні оновлені роботи. Шаблон головної сторінки наведено на рисунку 3.12;

Рисунок 3.12 – Шаблон головної сторінки

– Для модуля user було розроблено сторінку профілю, що містить основну інформацію користувача, форму для оновлення інформації користувача (зміна ім’я, пошти та аватару), а також відстежуванні користувачем роботи. Крім того, було розроблено сторінки для авторизації та реєстрації користувача. Шаблони веб-сторінок модуля user наведено на рисунку 3.13;

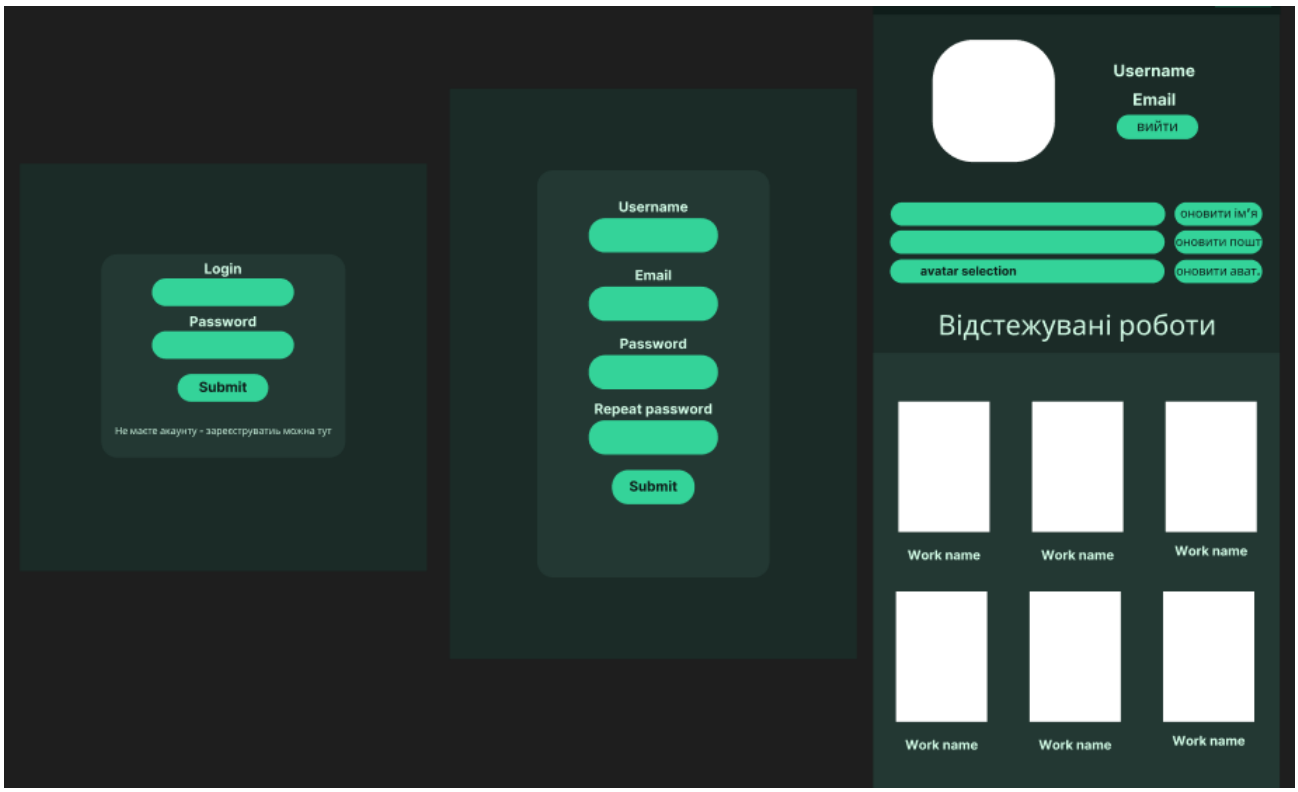

Рисунок 3.13 – Шаблони сторінок профілю, авторизації та реєстрації користувача

– Для модуля mediaProducts були розроблені сторінки для відображення інформації про медіапродукти, а саме: роботи, публікації та одиниць публікації. Шаблони до розроблених веб-сторінок наведено на рисунку 3.14.

Рисунок 3.14 – Шаблони сторінок модуля mediaProducts

Після створення шаблонів веб-сторінок платформи, їх потрібно реалізувати в системі у відповідних модулях. Оскільки усі сторінки мають одну й ту саму загальну структуру, у модулі infoPlatformMainApp головного додатку, де описується головна сторінка платформи, було створено загальний html шаблон у файлі «wrapper.html» каталогу infoPlatformMainApp/templates/ та основні стилі платформи у файлі «main_style.scss» каталогу infoPlatformMainApp/static/mainapp/scss/. Також оскільки для опису стилів використовується інструмент SCSS, у каталозі, де пописані головні стилі, був добавлений новий каталог units, у якому описані загальні змінні та міксини, для спрощення реалізації стилів. Опис загального шаблону сторінок, змінні та міксини стилів наведено у додатку В. Далі у каталогах templates та static кожного модуля були описані відповідні html сторінки та стилі до них. Також, оскільки деякі елементи та сторінках мають унікальний функціонал (наприклад, вивід модального вікна, зміна тексту при виборі зображення і т.д.), до них були описані відповідні скрипти у каталозі static відповідного модуля.

Отже, реалізувавши сторінки платформи, наступним заключним етапом конструювання є опис отримання доступу користувачеві до них та наповнення сторінок відповідною інформацією. Для цього до кожного модуля були описані URL-марнрути та відповідні до них представлення:

– Для модуля infoPlatformMainApp описані URL-маршрути для головної сторінки платформи та функції пошуку роботи по назві, а також відповідні до них представлення;

```
urlpatterns = [
    path("", index, name="index"),
    path('search/', ajax_search, name='search')
]
```

Рисунок 3.15 - URL-маршрути модуля infoPlatformMainApp

– Для модуля user описані URL-маршрути та відповідні представлення для сторінки профіля та системи авторизації та реєстрації користувача;

```
urlpatterns = [
    path("login/", LoginView.as_view(), name="login"),
    path("register/", RegisterView.as_view(), name="register"),
    path("logout/", LogoutView.as_view(), name="logout"),
    path("profile/", profile_view, name="profile"),
]
```

Рисунок 3.16 - URL-маршрути модуля user

– Для модуля mediaProducts описані URL-маршрути та відповідні представлення для сторінок відображення інформації про роботу, публікацію та одиницю публікації.

```
urlpatterns = [
    path('work/<str:link>/', work_view, name='work'),
    path('publication/<str:publication_type>/<str:link>/', publication_view, name='publication'),
    path('publication-unit/<str:publication_unit_type>/<str:link>/',
         publication_unit_view, name='publication_unit'),
]
```

Рисунок 3.17 - URL-маршрути модуля mediaProducts

Отже, у даному підрозділі було розроблено шаблони та реалізовано користувацький інтерфейс до розроблювальної інформаційної платформи у вигляді веб-сторінок, описано URL-маршрути та представлення які наповнюють сторінки інформацією і поєднують функціонал системи з графічним інтерфейсом.

3.3 Тестування системи

Одним із важливих етапів розробки програмного забезпечення є його тестування. Да даному етапі, за допомогою різних видів та інструментів тестування, проводиться перевірка описаної бізнес-логіки та функціональності елементів додатку.

Для даної розроблювальної інформаційної платформи було проведено два види тестування: модульне та автоматизоване.

3.3.1 Модульне тестування

До модульного тестування відносять написання ряду тестів, для перевірки бізнес-логіки та функціональності окремого модуля системи. У даному випадку, було написано ряд модульних тестів у каталозі tests для модулів infoPlatformMainApp, mediaPlatform та user. Даний каталог складається з 4 файлів:

– test_facade.py: у даному файлі описуються тести для перевірки бізнес-логіки функцій фасаду модуля;

– test_models.py: у даному файлі описуються тести для усіх моделей модуля;

– test_urls.py: у даному файлі описуються тести для перевірки усіх URL-маршрутів модуля;

– test_views.py: у даному файлі описуються тести для перевірки усіх описаних представлень модуля.

Після того, як для кожного вище переліченого модуля описали відповідні тести, у PyCharm було створені та налаштовано відповідні конфігурації для запуску тестів для кожного модуля за допомогою команди «python manage.py test». Як

приклад, на рисунку 3.18 було наведено приклад на основі конфігурації запуску тестів для модуля infoPlatformMainApp.

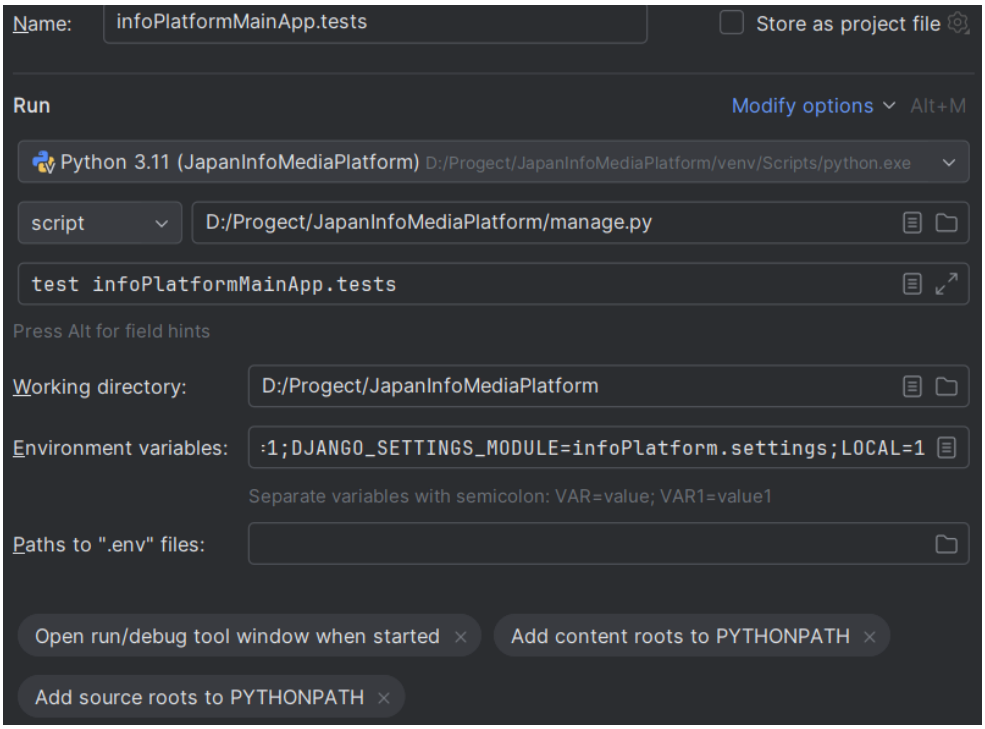

Рисунок 3.18 – Конфігурація запуску тестів для модуля infoPlatformMainApp

На рисунку 3.18 можна розглянути приклад конфігурації, де:

– Спочатку вибираємо як налаштування запуску скрипт, та вказуємо шлях до файлу «manage.py» та додатково в полі аргументів запуску вписуємо команду запуску тестів відповідного модуля;

– Далі вказуємо, як робочу директорію, кореневу директорію розроблювального проекту;

– На завершення додатково вписуємо до змінних середовища змінні, що відповідають за використання файлу налаштувань та використання внутрішньої бази даних для тестування.

Виконавши конфігурацію для запуску тестів для модулів infoPlatformMainApp, mediaPlatform та user, почергово було запущено тести, результат яких наведено на рисунках 3.19 – 3.21 відповідно.

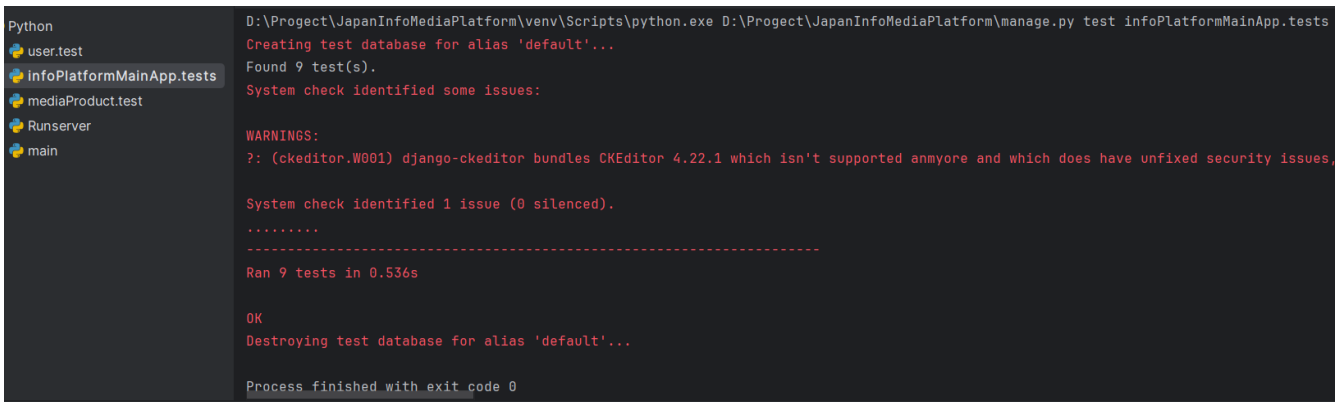

Рисунок 3.19 – Результати виконання модульних тестів для модуля infoPlatformMainApp

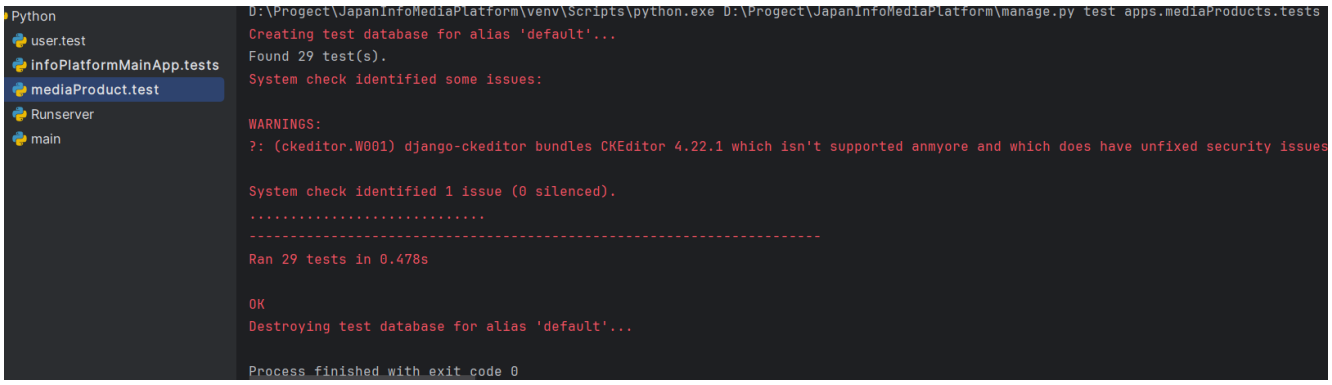

Рисунок 3.20 – Результати виконання модульних тестів для модуля mediaPlatform

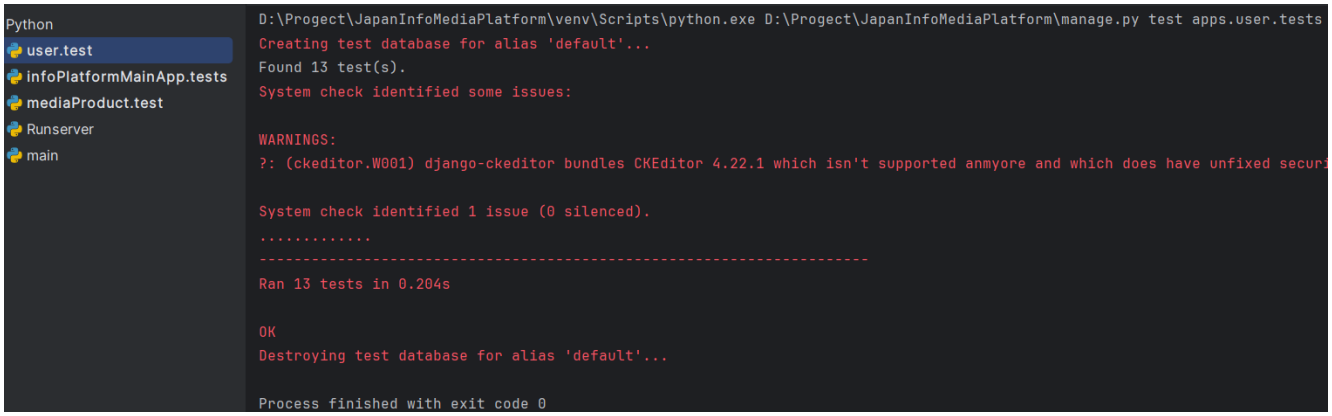

Рисунок 3.21 – Результати виконання модульних тестів для модуля user

Отже, по результатам, що наведені на рисунках 3.19 – 3.21, усі описані модульні тести для модулів infoPlatformMainApp, mediaPlatform та user пройшли успішно.

3.3.2 Автоматизоване тестування

Автоматизоване тестування програмного забезпечення — частина процесу тестування на етапі контролю якості в процесі розробки програмного забезпечення. Воно використовує програмні засоби для виконання тестів і перевірки результатів виконання, що допомагає скоротити час тестування і спростити його процес [19].

Selenium IDE - це інтегроване середовище розробки скриптів Selenium. IDE Selenium - це не тільки інструмент запису: це повна IDE. Можна вибрати, чи використовувати його можливості запису, або ви можете редагувати свої скрипти вручну [19].

Для розроблювальної інформаційної платформи було розроблено 6 тест-кейсів, що перевіряють основну функціональність системи:

1) Перевірка функціоналу перегляду інформації про медіапродукт.

В даному тесті описались ряд послідовних команд, що відтворюють кроки виконання функціоналу перегляду інформації про медіапродукт: перехід з головної сторінки на сторінку роботи, перехід на сторінку публікації та одиниці публікації, перегляд інформації про активних персонажів.

2) Перевірка функціоналу перегляду серверних та індустріальних новин.

В даному тесті описались ряд послідовних команд, що відтворюють кроки виконання функціоналу перегляду на головній сторінці індустріальних та серверних новин.

3) Перевірка системи реєстрації та авторизації.

В даному тесті описались ряд послідовних команд, що відтворюють кроки виконання функціоналу системи авторизації користувача, а також реєстрації нового користувача.

4) Перевірка функціоналу додавання коментаря.

В даному тесті описались ряд послідовних команд, що відтворюють кроки виконання функціоналу додавання користувачем нового коментаря до одиниці публікації медіапродукту.

5) Перевірка функціоналу відстежування роботи.

В даному тесті описались ряд послідовних команд, що відтворюють кроки виконання функціоналу відстежування роботи користувачем: додавання користувачем роботи до списку відстежування, перехід на сторінку профілю та з неї на сторінку роботи, а також вилучення користувачем роботи з відстежуваних.

6) Перевірка функціоналу редагування профілю користувача

В даному тесті описались ряд послідовних команд, що відтворюють кроки виконання функціоналу редагування профілю користувача: перехід на сторінку профілю та виконання зміни ім’я та пошти користувача.

Результати виконання тестів наведені у додатку Д.

3.4 Підсумки розділу 3

Отже, у розділі 3 «Конструювання та тестування платформи» було розписано про етап конструювання та тестування розроблювальної інформаційної платформи:

– Створення базової структури проекту та його налаштувань;

– Реалізація класів моделей та фасадів системи, додавання їх у адміністративну панель;

– Розробка користувацького інтерфейсу та його реалізація;

– Реалізація і опис URL-маршрутів та представлень у модулях системи;

– Опис модульних тестів та проведення автоматизованого тестування.

Результати та лістинги коду були наведені у відповідних додатках.

РОЗДІЛ 4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Актуальність безпеки життєдіяльності людини.

У сучасному світі, де промисловість та технології безупинно розвиваються та покращуються, виникає питання в актуальності безпеки життєдіяльності людини. Протягом останніх років 20-го століття та перших років 21-го століття відбулася ціла низка катастроф технологічного та природного походження, зросла кількість соціальних небезпек [20].

Щодня частіше порушуються механізми взаємодії людини та природи, людини та техніки, індивіда та середовища. Це зумовлює появу небезпек, які стають на перешкоді людській життєдіяльності. Проявом цього є значні втрати у вигляді людських жертв, збитків від аварій, катастроф, стихійних лих тощо [20].

Тому нагальною є потреба створення безпечних умов життя, формування якісних заходів у галузі безпеки життєдіяльності всіх верств населення, що є запорукою сталого та безпечного життя. Засобом досягнення цієї мети є організація системи безперервного навчання з безпеки життєдіяльності [20].

Рівень безпеки людини в міру розвитку цивілізації постійно зростає. Людство перемогло епідемії холери, віспи, чуми, тифу. Середня тривалість життя людини у найрозвиненіше них країнах світу становить майже 78 років і продовжує зростати. (у нас - середня - 71,3 р.; чоловіки - 67 р., жінки - 77 р.) [20].

Розвиток науки і техніки, в цілому збільшуючи безпеку життєдіяльності людини, призвів до появи цілого ряду нових проблем [20].

Перш за все - надзвичайне зростання ступеня ризику травматизму та загибелі людей при взаємодії зі складними технічними системами на виробництві, транспорті, в побуті [20].

За даними Всесвітньої організації охорони здоров’я (ВООЗ), смертність від нещасних випадків в наш час займає 3-є місце після серцево-судинних і онкологічних захворювань, причому переважно гинуть працездатні люди віком до 40 років. В цілому світі, враховуючи усі нещасні випадки, пов’язані з

використанням машин, обладнання, технічних пристроїв, кожного року страждає понад 10 млн. людей, а близько 0,5 млн. – гине [20].

По друге - зростання числа випадків технологічних катастроф (аварії на АЕС, на хімічних та інших небезпечних виробництвах, транспортні нещасні випадки та ін.) зумовлене зниженням реальної надійності пристроїв, зроблених людиною, та помилками персоналу під час їх експлуатації. З'явився страх втратити контроль над технікою [20].

По третє - істотне збільшення антропогенного навантаження на навколишнє середовище від життєдіяльності людини досягло граничного рівня, що викликає загрозу існуванню людини, як біологічного виду. Безглузде виробництво зброї, зростання кількості атомних електростанцій, гіпертрофований розвиток автомобільного транспорту, хімічних та шкідливих підприємств, інші антропогенні впливи істотно змінили якість води, повітря та інших характеристик навколишнього природного середовища, зробили життя людини значно не безпечнішим. Зростання кількості стихійних лих пов'язується з діяльністю людини. Потрібно задуматись над таким фактом: з 1960 року по 2000 рік кількість стихійних лих на землі зросла у 5 разів і продовжує зростати. Особлива небезпека життєдіяльності людей виникла в результаті найбільшої на нашій планеті техногенної катастрофи - аварія на Чорнобильській АЕС [20].

По четверте - досягнення потенційної ефективності технічних систем неможливе з багатьох причин: неузгодженість рівня розвитку та підготовки людини з особливостями техніки; неузгодженість можливостей людини з параметрами обладнання, що особливо проявляється за умов дефіциту часу, інформації та дії зовнішніх факторів; низький рівень відповідальності людей [20].

Вирішення цих проблем потребує комплексного вивчення принципів взаємодії людини з технікою (машиною) та навколишнім середовищем. Потрібно змінити психологію людини та рівень її навчання для дій в складних, надзвичайних обставинах, які виникають внаслідок аварій, катастроф. Психологи зазначають, що сучасна людина більше боїться корови чи коня на пасовищі, ніж деталей машин, що швидко обертаються, чи автомобіля, який мчить назустріч. У той же час,

опинившись у нестандартній ситуації, яка вимагає негайних та поточних рішень, люди раптом стають безпорадними перед обставинами, неспроможними вирішувати найпростіші, але часто важливі для їх життя та здоров'я (і для оточуючих) проблеми [20].

Досягнувши значного прогресу в боротьбі з хворобами, у розвитку науки і техніки, людство ще не спроможне ефективно боротись з нещасними випадками та їх наслідками - травмами, виникає поширення явищ наркоманії, СНІДУ [20].

Переважно це зумовлено значними прогалинами у вихованні та освіті людини. Недоліки існуючої системи освіти та підготовка кадрів зумовлюють високі показники побутового та виробничого травматизму і смертність. Існує думка, що інфальтильна щодо власної та суспільної безпеки ментальність людини не формує адекватної небезпечним життєвим ситуаціям лінії поведінки. Щоб зберегти життя і здоров'я, людина повинна знати про небезпеку у навколишньому середовищі, вміти їх виявляти, знати засоби захисту [20].

4.2 Заходи щодо запобігання розповсюдження пожежі в приміщенні

Пожежа в приміщенні є однією з найбільш небезпечних надзвичайних ситуацій, що являє собою загрозу не тільки для матеріальних цінностей, а й для життя та здоров'я людей. Запобігання її поширенню є ключовим складником ефективної протипожежної безпеки.

Універсальним методом зниження передачі енергії є зниження температури горіння до значення, нижчого ніж температура погасання. Досягається це на основі чотирьох відомих принципів припинення горіння [23]:

– охолодження зони горіння або речовини, що горить до певних температур;

– розбавлення речовин – учасників реакції горіння, тобто зниження їх концентрації (як горючої речовини, так і окисника);

– ізоляція реагуючих речовин (горючого або окисника) від зони горіння;

– хімічне гальмування (інгібування) швидкості хімічних реакцій горіння;

– механічний зрив полум’я сильним струменем газу або води;

– створення умов вогнеперешкодження, тобто таких умов, при яких полум’я розповсюджується через вузькі канали і при зменшенні перетину останніх до встановленої величини розповсюдження полум’я припиняється.

Вогнегасна речовина – це речовина або однорідна суміш, за своїми фізико-хімічними властивостями придатна до застосування в технічних засобах задля припинення горіння [23].

Разом з тим, для гасіння пожеж застосовують не всі речовини, а лише ті, що відповідають певним вимогам [23]:

– володіти високим ефектом гасіння при порівняно малій витраті;

– бути доступними, дешевими і простими у застосуванні;

– не проявляти шкідливого впливу на людей, матеріали та довкілля.

У якості вогнегасних речовин застосовують: воду, що подають розпиленими або суцільними струменями; воду з добавками (змочувачі, добавки проти замерзання тощо); піну (хімічну, повітряно-механічну різної кратності); інертні газові розріджувачі (діоксид вуглецю), азот, аргон, відпрацьовані гази двигунів, водяна пара); галогенопохідні вуглеводнів (хладони 13В1, 12В1, 114В2); порошки; комбіновані склади [23].

Надзвичайно обережно необхідно підходити при використанні води для гасіння [23]:

– рідин, що мають питому густину, меншу за питому густину води – ці рідини водою розносяться на більшу площу, продовжуючи горіти, (гасити їх можна лише тонкорозпиленою водою);

– пожеж пилоподібних матеріалів (їх теж можна гасити лише тонкорозпиленою водою, так як суцільний струмінь може підняти аерогель у повітря, перетворивши його на аерозоль і викликати вибух);

– тліючих мас у закритих приміщеннях (внаслідок раптового випаровування великої кількості води і утворення великої кількості водяної пари можна отримати опіки);

– рідин у резервуарах (суцільний струмінь води, потрапивши у розпечене середовище, випаровується з вибухом);

– особливо обережно слід підходити до гасіння водою електричних мереж і електрообладнання. Краще всього їх знеструмити або вимкнути їх живлення.

Кожному способу припинення горіння відповідає конкретний вид вогнегасних засобів, які можна поділити на [21]:

– охолоджуючі (вода, водні розчини, снігоподібна вуглекислота та інші);

– розбавляючі (вуглекислий газ, водяна пара, інертні гази та інші);

– ізолюючі (хімічна та повітряно-механічна піна, вогнегасні порошки, пісок та інші);

– засоби хімічного гальмування горіння (брометил, хладон, склад 3,5 та інші).

До первинних засобів гасіння пожежі відносяться: вогнегасники - переносні (не більше 20 кг) вогнегасники (типу ВВП-10; ВВК-2; ВП-2; ВАХ-0,5) та пересувні (не більше 450 кг), інвентар (покривала з негорючого теплоізоляційного полотна, ящики з піском (об’ємом не менше 0,5 м^3), бочки з водою, пожежні відра (об’ємом не менше 0,08 м^3), совкові лопати та пожежний інструмент (гаки, ломи, тощо). Первинні засоби пожежогасіння застосовують при ліквідації невеликих загорань до приведення в дію стаціонарних та напів стаціонарних засобів гасіння пожежі або до прибуття пожежної команди [21].

Кожне приміщення, відділення, цех, рухомий склад повинні бути забезпечені такими засобами відповідно нормам оснащення протипожежним обладнанням та інвентарем [21].

Одним із головних критеріїв обґрунтування потрібної кількості вогнегасників є клас потенційно можливої пожежі горючих речовин і матеріалів у захищуваному приміщенні [21].

Громадські будівлі та споруди промислових підприємств повинні мати на кожному поверсі не менше двох ручних вогнегасників [21].

При захисті приміщень, в яких знаходяться електронно-обчислювальні машини, копіювальна та інша оргтехніка,а також телефонних станцій, архів тощо, необхідно враховувати специфіку вогнегасних речовин у вогнегасниках, що можуть призвести під час гасіння пожежі до псування обладнання. Такі приміщення рекомендується забезпечувати вуглекислотними вогнегасниками з урахуванням гранично допустимої концентрації вогнегасної речовини [21].

Для гасіння великих загорань у приміщеннях застосовують стаціонарні установки хімічного та повітряно-пінного гасіння [21].

До розповсюджених автоматичних стаціонарних засобів гасіння пожежі відносять установки водяного та пінного пожежогасіння, а також установки газового та порошкового пожежогасіння. Спринклерні та дренчерні установки являють собою розгалужену мережу трубопроводів зі спринклерними або дренчерними головками і розташовуються під стелею приміщення, яке потрібно захистити [21].

Також, для швидкого запобігання розповсюдженню пожежі та забезпечення власної безпеки необхідно знати, як правильно користуватися засобами пожежогасіння відповідно.

Правильно використовуйте засоби гасіння пожеж [22]:

– переносні вогнегасники (порошкові, вуглекислотні, водопінні) використовують згідно з інструкцією, наведеною у вигляді малюнків і тексту на корпусі вогнегасників;

– для приведення в дію пожежних кранів, що знаходяться в будинку (споруді), необхідно відкрити дверцята шафи і розкласти в напрямку осередку пожежі рукав, з’єднаний з краном і стволом. Відкрити вентиль поворотом маховика проти ходу годинникової стрілки і спрямувати струмінь води із ствола на осередок горіння.

ВИСНОВКИ

При виконанні даної кваліфікаційної роботи було описано повний життєвий цикл розробки інформаційної платформи оглядів японських медіапродуктів, що відповідає вимогам користувачів.

На етапі аналізу вимог було проведено дослідження існуючих конкурентів на глобальному та українському ринках, а саме: веб-платформа «Fandom» та веб-сайт «НекоТека», визначено акторів системи, ряд користувацьких, функціональних та нефункціональних вимог. Також описано обрану архітектуру, методологію розробки (Agile/Scrum) і технологій (Django, MySQL, HTML, SCSS).

При проектуванні системи, для моделювання архітектури було використано інструмент IBM RSAD. Виявлено необхідні варіанти використання, описано модулі системи, їх призначення, обрано шаблони проектування (MVT та Фасад) та створено відповідні UML-діаграми ієрархії класів і UML-діаграми послідовностей для ключових сценаріїв взаємодії користувачів із платформою.

На етапі конструювання здійснено саму реалізацію розроблювальної платформи: створено та описано базову структуру проекту, її налаштування, реалізовано спроектовані класи моделей та фасадів, розроблено користувацький інтерфейс. Крім того, описано URL-маршрути та представлення (Views), що забезпечують взаємодію інтерфейсу з системою.

Також було проведено тестування модульне та автоматизоване тестування, що забезпечило перевірку надійності та стабільності роботи платформи.

Загалом, виконаний проект підтверджує доцільність та ефективність створення різноманітних інформаційних платформ для огляду медіапродуктів. Реалізоване рішення не лише задовольняє вимоги користувачів, а й демонструє потенціал подальшого розвитку у напрямі розширення функціональності, локалізації та інтеграції з іншими сервісами. Платформа створена з урахуванням сучасних підходів до розробки програмного забезпечення, що робить її актуальною та конкурентоспроможною в сучасному цифровому середовищі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Fandom URL: https://www.fandom.com/ (Дата звернення: 18.05.2025).
2. НекоТека | Найбільша бібліотека по східній культурі URL: https://nekoteka.com/ (Дата звернення: 20.05.2025).
3. Процес аналізу вимог URL: https://dl.tntu.edu.ua/content.php?cid=98816 (Дата звернення: 20.05.2025).
4. Класифікація і специфікація вимог URL: https://dl.tntu.edu.ua/content.php?cid=99015 (Дата звернення: 20.05.2025).
5. What Is Three-Tier Architecture? URL: https://www.ibm.com/think/topics/three-tier-architecture (Дата звернення: 20.05.2025).
6. Three-Tier Client Server Architecture in Distributed System URL: https://www.geeksforgeeks.org/three-tier-client-server-architecture-in-distributed-system/ (Дата звернення: 21.05.2025).
7. What is Agile? | Atlassian URL: https://www.atlassian.com/agile (Дата звернення: 21.05.2025).
8. Балик Н.Р., Мандзюк В.І. Б20 Бази даних MySQL: навчальний посібник. — Тернопіль: Навчальна книга – Богдан, 2010.— 160 с.
9. Петрик М., Михалик Д., Мудрик І., Стоянов Ю. Лабораторний практикум з розділу «Шаблони проектування» дисципліни «Архітектура та проектування програмного забезпечення: навчальний посібник — Тернопіль: ТНТУ імені Івана Пулюя, 2016. — 36 с.
10. М.Р. Петрик, Ф.Я. Мудрик Архітектура програмного забезпечення (на базі використання CASE-засобів IBM(sad)) навчальний посібник, Тернопіль: ТНТУ імені Івана Пулюя, 2017. — 100с.
11. Моделювання та аналіз програмного забезпечення URL: https://dl.tntu.edu.ua/index.php (Дата звернення 23.05.2025).
12. Modular Architecture Software Development URL: https://triare.net/insights/modular-architecture-software/ (Дата звернення 26.05.2025).

13. Патерни/шаблони проектування URL: https://refactoring.guru/uk/design-patterns (Дата звернення 28.05.2025).

14. Django Project MVT Structure URL: https://www.geeksforgeeks.org/django-project-mvt-structure/ (Дата звернення 28.05.2025).

15. What is Sequence Diagram? URL: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-sequence-diagram/ (Дата звернення 30.05.2025).

16. Writing your first Django app, part 1 URL: https://docs.djangoproject.com/en/5.2/intro/tutorial01/ (Дата звернення 03.06.2025).

17. Writing your first Django app, part 2 URL: https://docs.djangoproject.com/en/5.2/intro/tutorial02/ (Дата звернення 03.06.2025).

18. The Django admin site URL: https://docs.djangoproject.com/en/5.2/ref/contrib/admin/ (Дата звернення 03.06.2025).

19. Якість програмного забезпечення та тестування (SE321) URL: https://dl.tntu.edu.ua/content.php?cid=279258 (Дата звернення 07.06.2025).

20. Потенційна безпека життєдіяльності людини. Основні поняття URL: https://dl.tntu.edu.ua/content.php?cid=299141. (Дата звернення 02.06.2025).

21. Гасіння пожежі, вогнегасні речовини та первинні засоби пожежогасіння URL: https://dl.tntu.edu.ua/content.php?cid=289214. (Дата звернення 07.06.2025).

22. Пожежа у приміщенні та на відкритій місцевості URL: https://dsns.gov.ua/abetka-bezpeki-1/pozezna-nebezpeka/pozeza. (Дата звернення 31.05.2025).

23. Визначення параметрів розвитку пожежі та необхідних витрат вогнегасних речовин URL: https://elib.lntu.edu.ua/sites/default/files/elib_upload/www/page20.html (Дата звернення 10.06.2025).

ДОДАТКИ

Додаток А. Тези конференції

УДК 004.4
Занчук Т. – студент групи СП-41
Тернопільський національний технічний університет імені Івана Пулюя

Розробка інформаційної платформи для медіапродуктів з використанням сучасних технологій

Науковий керівник: д.ф.-м.н., професор Петрик М.Р.

Zanchuk T.
Ternopil Ivan Puluj National Technical University

Development of an information platform for media products using modern technologies

Supervisor: Petryk M.R.

Ключові слова: інформаційна платформа, медіа продукти
Keywords: information platform, media products

Інтерес до японської поп-культури, зокрема аніме (японської анімації), манги (коміксів) та новел, стрімко зростає в усьому світі. Однак користувачі часто стикаються з проблемою пошуку зручних платформ для перегляду, каталогізації та обговорення улюблених медіапродуктів. Створення спеціалізованої інформаційної платформи дозволяє централізувати інформацію про твори, їхніх творців, персонажів та новини індустрії, надаючи користувачам швидкий доступ до систематизованої інформації.

Метою дипломної роботи є розробка веб-платформи для зберігання та перегляду інформації про японські медіапродукти з використанням сучасних технологій Django та MySQL.

Django — це високорівневий веб-фреймворк Python, який заохочує швидку розробку та чистий, прагматичний дизайн [1]. MySQL — це СУБД з відкритим вихідним кодом, яка використовує SQL для створення та керування базами даних [2].

Для проектування системи було вибрано середовище IBM RSAD, що спрощує архітектурне моделювання [3].

Щодо структури системи, то вона реалізована за допомогою модульної архітектури, де кожен функціональний блок винесений в окремий додаток:

- **mediaProducts:** зберігання та перегляд творів, публікацій (манга, новели, аніме), персонажів.
- **infoPlatformMainApp:** містить систему новин (про твори, сервер).
- **user:** містить профілі користувачів, коментарі.

У дипломній роботі була спроектована UML-діаграма класів, яка відображає основну логіку взаємозв’язків між елементами інформаційної платформи. Основним об'єктом є клас Works, що поєднаний із жанрами, авторами та публікаціями. Публікації розділено на три типи — манга, новели та аніме, кожна з яких включає в себе одиниці публікації (PublicationUnit), представлені у вигляді томів або сезонів. Також реалізовано механізм зв’язку між творами та персонажами, що дозволяє гнучко структурувати контент. Користувачі мають змогу залишати коментарі до публікацій, що забезпечує інтерактивність платформи.

Окрему роль у системі відіграють класи MediaProductsViewer, UserViewer та NewsViever, які реалізують шаблон фасаду і забезпечують централізований доступ до функцій перегляду медіаконтенту, профілів користувачі, коментарів та новин. Для забезпечення взаємодії з базою даних у діаграму було додано умовний клас Database, що символізує доступ через Django ORM до СУБД MySQL. Це демонструє, як усі моделі здійснюють читання та запис даних, а також підкреслює архітектурну прозорість та масштабованість розробленої системи.

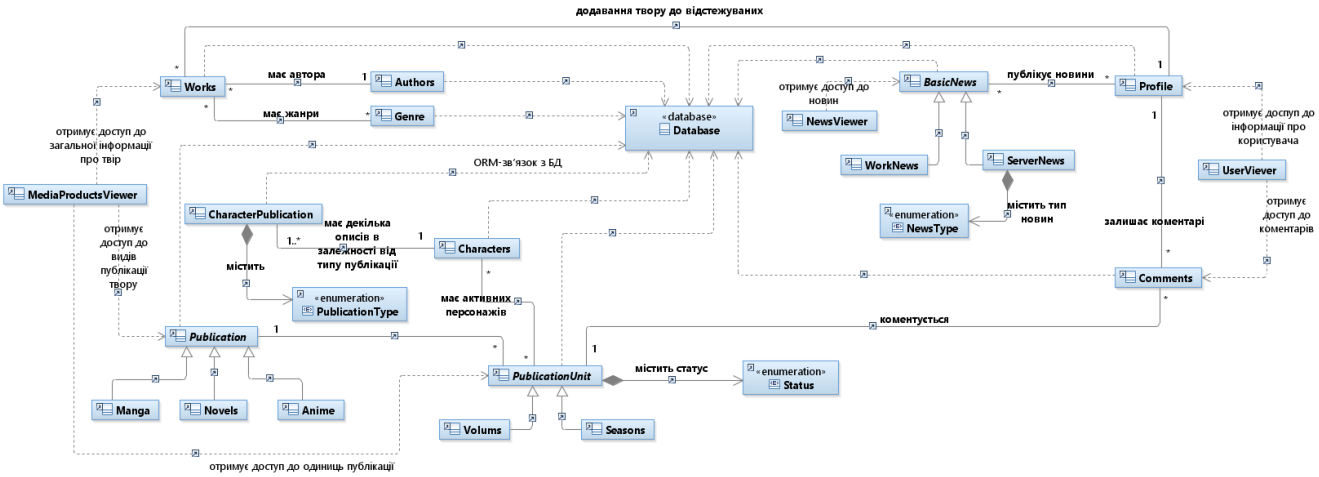

Рисунок 1 – UML діаграма класів системи

Висновок

Розроблена платформа є гнучкою та легкою у підтримці завдяки модульному підходу та використанню фреймворку Django. Робота демонструє практичне застосування об'єктно-орієнтованого проектування та роботу з БД.

Список використаних джерел

1. Django: The web framework for perfectionists with deadlines. URL: https://www.djangoproject.com/.
2. MySQL: Understanding What It Is and How It's Used – Oracle. URL: https://www.oracle.com/mysql/what-is-mysql/.
3. Rational Software Architect Designer – IBM. URL: https://www.ibm.com/products/rational-software-architect-designer./.

Додаток Б. Опис основних класів моделей та фасадів

Опис основних класів моделей та фасадів у модулі mediaProducts:

– Клас моделі Works:

```
class Work(models.Model):
    title = models.CharField(max_length=255)
    description = RichTextField(blank=True, null=True, verbose_name='Опис')
    image = FileBrowseField(verbose_name='Зображення твору',
max_length=255, directory='work_image/', extensions=['.jpg', '.jpeg',
'.png'], blank=True, null=True)
    genre = models.ManyToManyField('Genre', related_name='works',
verbose_name='Жанр')
    author = models.ForeignKey('Author', on_delete=models.CASCADE,
related_name='works', verbose_name='Автор')
    link = models.CharField(max_length=255, blank=True, null=True,
verbose_name='Посилання')

    def __str__(self):
        return self.get_title()

    def save(self, *args, **kwargs):
        if not self.link:
            self.link = slugify(self.title, to_lower=True)
        super().save(*args, **kwargs)

    def get_genres(self):
        return self.genre.all()

    def get_author(self):
        return self.author

    def get_description(self):
        return self.description

    def get_image(self):
        return self.image.url if self.image else None

    def get_title(self):
        return self.title

    def get_absolute_url(self):
        return f"/work/{self.link}/"

    def get_link(self):
        return self.link
```

– Клас моделі Publication:

```
class Publication(models.Model):
    title = models.CharField(max_length=255, verbose_name='Назва')
    cover_image = FileBrowseField(verbose_name='Обкладинка',
max_length=255, directory='publication_image/', extensions=['.jpg',
'.jpeg', '.png'], blank=True, null=True)
    link = models.CharField(max_length=255, blank=True, null=True,
verbose_name='Посилання')
```

```
    work = models.ForeignKey(Work, on_delete=models.CASCADE,
related_name='%(class)s_publications', verbose_name='Твір')
    publication_unit = models.ManyToManyField('PublicationUnit',
blank=True, related_name='%(class)s_publications',
verbose_name='Публікаційна одиниця')

    class Meta:
        abstract = True

    def __str__(self):
        return self.get_title()

    def get_title(self):
        return self.title

    def get_cover_image(self):
        return self.cover_image.url if self.cover_image else None

    def get_publication_unit(self):
        return self.publication_unit.all()

    def get_work(self):
        return self.work

    def get_link(self):
        return self.link

    def save(self, *args, **kwargs):
        if not self.link:
            self.link = slugify(self.title, to_lower=True)
        super().save(*args, **kwargs)
```

– Клас моделі PublicationUnit:

```
class PublicationUnit(models.Model):
    title = models.CharField(max_length=255, verbose_name='Назва')
    description = RichTextField(blank=True, null=True, verbose_name='Опис')
    title_image = FileBrowseField(verbose_name='Зображення обкладинки',
max_length=255, directory='publication_unit_image/', extensions=['.jpg',
'.jpeg', '.png'], blank=True, null=True)
    link = models.CharField(max_length=255, blank=True, null=True,
verbose_name='Посилання')
    year_of_release = models.PositiveIntegerField(default=0, blank=True,
null=True, verbose_name='Рік випуску')
    number = models.PositiveIntegerField(default=0, blank=True, null=True,
verbose_name='Номер')
    active_characters = models.ManyToManyField('Characters',
related_name='active_characters', verbose_name='Активні персонажі')
    last_update = models.DateTimeField(auto_now=True, verbose_name='Останнє
оновлення')

    class Status(models.TextChoices):
        ONGOING = 'ongoing', 'Ongoing'
        COMPLETED = 'completed', 'Completed'
        DROPPED = 'dropped', 'Dropped'
        Announced = 'announced', 'Announced'
        Suspended = 'suspended', 'Suspended'
```

```
    status = models.CharField(max_length=50, choices=Status.choices,
verbose_name='Статус')

    def __str__(self):
        return self.get_title()

    def get_title(self):
        return self.title

    def get_description(self):
        return self.description

    def get_title_image(self):
        return self.title_image.url if self.title_image else None

    def get_year_of_release(self):
        return self.year_of_release

    def get_number(self):
        return self.number

    def get_active_characters(self):
        return self.active_characters.all()

    def get_status(self):
        return self.status

# Removed the get_publication method as it references an undefined
attribute.

    def save(self, *args, **kwargs):
        if not self.link:
            self.link = slugify(self.title, to_lower=True)
        super().save(*args, **kwargs)
```

– Клас фасаду MediaProductsViewer:

```
class MediaProductsViewer:
    @staticmethod
    def get_work_info(link):
        """Повертає інформацію про твір за ID"""
        try:
            return
Work.objects.prefetch_related('genre').select_related('author').get(link=li
nk)
        except Work.DoesNotExist:
            return None

    @staticmethod
    def get_last_update_work():
        """Повертає список робіт, відсортованих за останнім оновленням"""
        try:
            return Work.objects.prefetch_related(
                'manga_publications__publication_unit',
                'anime_publications__publication_unit',
                'novel_publications__publication_unit'
```

```
            ).annotate(
last_update=Max('manga_publications__publication_unit__last_update')
            ).order_by('-last_update')
        except Work.DoesNotExist:
            return None

    @staticmethod
    def get_publications_by_work(link):
        """Повертає публікації, пов'язані з твором"""
        try:
            manga = Manga.objects.filter(work__link=link)
            anime = Anime.objects.filter(work__link=link)
            novel = Novel.objects.filter(work__link=link)
            return {
                'manga': manga,
                'anime': anime,
                'novel': novel
            }
        except Work.DoesNotExist:
            return None

    @staticmethod
    def get_publication_by_type(publication_type, link):
       """Повертає публікацію за типом"""
       try:
            if publication_type == 'manga':
                return Manga.objects.filter(link=link).first()
            elif publication_type == 'anime':
                return Anime.objects.filter(link=link).first()
            elif publication_type == 'novel':
                return Novel.objects.filter(link=link).first()
            else:
                return None
       except Exception:
            return None

    @staticmethod
    def get_publication_units_by_publication(publication):
        """Повертає публікаційні одиниці, пов'язані з публікацією"""
        try:
            return publication.publication_unit.all()
        except Exception as e:
            return None

    @staticmethod
    def get_publication_unit_by_type_and_link(publication_unit_type,
publication_unit_link):
        """Повертає публікаційну одиницю за посиланням"""
        try:
            if publication_unit_type == 'volume':
                return Volumes.objects.get(link=publication_unit_link)
            elif publication_unit_type == 'season':
                return Seasons.objects.get(link=publication_unit_link)
            else:
                return None
        except Exception:
            return None
```

```
    @staticmethod
    def track_work(user, work):
        """Додає твір до відстежуваних"""
        if not user.work_is_tracked(work):
            user.tracking_works.add(work)
            return True
        return False

    @staticmethod
    def untrack_work(user, work):
        """Видаляє твір з відстежуваних"""
        if user.work_is_tracked(work):
            user.tracking_works.remove(work)
            return True
        return False

    @staticmethod
    def get_works_by_title_fragment(title):
        """Повертає твір за назвою"""
        try:
            return Work.objects.filter(title__icontains=title)
        except Work.DoesNotExist:
            return None
```

Опис основних класів моделей та фасадів у модулі user:

– Клас моделі Profile:

```
class Profile(AbstractUser):
    avatar = FileBrowseField(verbose_name='Аватар', max_length=255,
                             directory='user_avatar/', extensions=['.jpg',
'.jpeg', '.png'], blank=True, null=True)
    groups = models.ManyToManyField(Group, related_name="user_profiles",
                                    blank=True, verbose_name='Групи')
    user_permissions = models.ManyToManyField(Permission,
related_name="user_permissions_set",
                                              blank=True,
verbose_name='Дозволи')
    tracking_works = models.ManyToManyField(Work,
verbose_name="Відстежувані твори",
                                            blank=True,
related_name='tracking_works')

    def get_tracking_works(self):
        return self.tracking_works.all()

    def get_avatar(self):
        return self.avatar.url if self.avatar else None

    def work_is_tracked(self, work):
        return self.tracking_works.filter(id=work.id).exists()

    def __str__(self):
        return self.username
```

– Клас моделі UserViewer:

```
class UserViewer:
    @staticmethod
    def get_user_by_id(user_id):
        """Повертає користувача за ID"""
        try:
            return Profile.objects.get(id=user_id)
        except Profile.DoesNotExist:
            return None

    @staticmethod
    def update_profile(action, update_value, user):
        """Оновлює профіль користувача"""
        if action == 'username_update':
            if update_value and update_value != user.username:
                if not
Profile.objects.filter(username=update_value).exists():
                    user.username = update_value
                    user.save()
                    return True
                else:
                    return False

        elif action == 'email_update':
            if update_value and update_value != user.email:
                if not Profile.objects.filter(email=update_value).exists():
                    user.email = update_value
                    user.save()
                    return True
                else:
                    return False

        elif action == 'avatar_update':
            if update_value:
                # Видалення старої аватарки
                if user.avatar:
                    old_avatar_path = os.path.join(settings.MEDIA_ROOT,
user.avatar.name)
                    if os.path.exists(old_avatar_path):
                        os.remove(old_avatar_path)

                # Збереження нової аватарки
                avatar_name =
default_storage.save(f"user_avatar/{user.username}_avatar_{update_value.nam
e}", update_value)
                user.avatar = avatar_name
                user.save()
                return True

        return False

    @staticmethod
    def get_comments_by_publication_unit(publication_unit_id):
        """Повертає коментарі до публікаційної одиниці"""
        return
Comment.objects.filter(publication_unit_id=publication_unit_id)
```

```
    @staticmethod
    def add_comment(user, publication_unit, content):
        """Додає коментар до публікаційної одиниці"""
        comment = Comment(user=user, publication_unit=publication_unit,
content=content)
        comment.save()
        return comment
```

Опис основних класів моделей та фасадів у модулі infoPlatformMainApp:

– Клас моделі BasicNews:

```
class BasicNews(models.Model):
    user = models.ForeignKey(Profile, on_delete=models.CASCADE,
verbose_name='Користувач')
    content = models.TextField(max_length=255, verbose_name='Контент')
    created_at = models.DateTimeField(auto_now_add=True, verbose_name='Дата
створення')

    def get_user(self):
        return self.user

    def get_content(self):
        return self.content

    def get_created_at(self):
        return self.created_at

    class Meta:
        abstract = True
```

– Клас фасаду NewsViewer:

```
class NewsViewer:
    @staticmethod
    def get_server_news():
        """Повертає список серверних новин."""
        return ServerNews.objects.all().order_by('-created_at')

    @staticmethod
    def get_work_news():
        """Повертає список новин, що стосуються творів."""
        return WorkNews.objects.all().order_by('-created_at')
```

Додаток В. Опис загального шаблону сторінок, змінні та міксини стилів

Опис загального шаблону сторінок у файлі «wrapper.html»:

```
{% load static %}
{% load compress %}
{% spaceless %}
    <!DOCTYPE html>
    <html lang="uk">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-
scale=1.0">
        <title>{% spaceless %}{% block title %}{% endblock %}{%
endspaceless %}</title>
        <meta name="description" content="{% spaceless %}{% block
description %}{% endblock %}{% endspaceless %}">
        <link
href="https://fonts.googleapis.com/css2?family=Konkhmer+Sleokchher&display=
swap" rel="stylesheet">
        <link
href="https://fonts.googleapis.com/css2?family=Sansita+Swashed:wght@400;600
;700&display=swap" rel="stylesheet">

        {% compress css %}
            <link rel="stylesheet" href="{% static
'mainapp/scss/main_style.scss' %}" type="text/x-scss">
            {# Блок призначений для внутрішніх scss стилів, які потрібно
вставити в head #}
            {% block head_custom_styles %}

            {% endblock %}
        {% endcompress %}

        {# Блок призначений для зовнішніх (cdn) css стилів, які потрібно
вставити в head #}
        {% block head_external_styles %}
        {% endblock %}

        {% compress js %}
            <script src="{% static 'mainapp/js/script.js' %}"></script>
            {# Блок призначений для внутрішніх js скриптів, які потрібно
вставити в head #}
            {% block head_custom_scripts %}
            {% endblock %}
        {% endcompress %}
        {# Блок призначений для зовнішніх (cdn) js скриптів, які потрібно
вставити в head #}
        {% block head_external_scripts %}
        {% endblock %}
    </head>
    <body>
    <input type="hidden" name="csrfmiddlewaretoken" value="{{ csrf_token
}}">
    <div class="wrapper">
    <header class="header">
        {% include 'mainapp/header.html' %}
    </header>
```

```
    <main class="content">
        {% block body %}
        {% endblock %}
    </main>
    <footer class="footer">
        {% include 'mainapp/footer.html' %}
    </footer>
    {% compress js %}
        <script src="{% static 'mainapp/js/script.js' %}"></script>
        {# Блок призначений для внутрішніх js скриптів, які потрібно
вставити в body #}
        {% block extra_js %}
        {% endblock %}
    {% endcompress %}
    {# Блок призначений для зовнішніх (cdn) js скриптів, які потрібно
вставити в body #}
    {% block external_extra_js %}
    {% endblock %}
    </div>
    </body>
    </html>
{% endspaceless %}
```

Опис загальних змінних для стилів у файлі «_variables.scss»:

```
// _variables.scss
$font-main: 'Konkhmer Sleokchher', sans-serif;
$font-secondary: 'Inter', sans-serif;

$color-bg: #1B2B27;
$color-header-bg: #0F1D1A;
$color-footer-bg: #0F1D1A;
$color-section-bg: #233833;
$color-card-bg: #2E534A;
$color-image-bg: #ccc;
$color-success: #34D399;
$color-text: #D1FAE5;
$color-text-secondary: #0F1D1A;
$color-placeholder: #0F1D1A;
$color-hover: #f0f0f0;
$color-border: #333;
$color-image-placeholder: #777;
$color-a: inherit;
$color-element-bg: #34D399;
```

Опис загальних міксинів для стилів у файлі «_mixins.scss»:

```
// _mixins.scss
@mixin flex-custom($direction: row, $align_items: center, $justify_content:
center, $gap: 0, $padding: 0) {
  display: flex;
  flex-direction: $direction;
  align-items: $align_items;
  justify-content: $justify_content;
  gap: $gap;
  padding: $padding;
```

```
}

@mixin rounded($radius: 20px) {
  border-radius: $radius;
}

@mixin reference() {
    a{
        color: inherit;
        text-decoration: none;
    }
}

@mixin placeholder($color: $color-text) {
  &::placeholder {
    color: $color;
    opacity: 0.7;
  }
}

@mixin hover-underline($color: $color-success) {
  &:hover {
    color: $color;
    text-decoration: underline;
  }
}

@mixin tooltip($bg: $color-section-bg, $border: $color-header-bg) {
  content: attr(data-description);
  position: absolute;
  top: 100%;
  right: 0%;
  transform: translateX(10px);
  background-color: $bg;
  color: $color-text;
  padding: 5px 10px;
  border-radius: 5px;
  border: 3px solid $border;
  font-size: 12px;
  white-space: nowrap;
  opacity: 0;
  visibility: hidden;
  transition: opacity 0.2s ease, visibility 0.2s ease;
  z-index: 10;
}
```

Додаток Д. Результати виконання описаних тест-кейсів для автоматизованого тестування

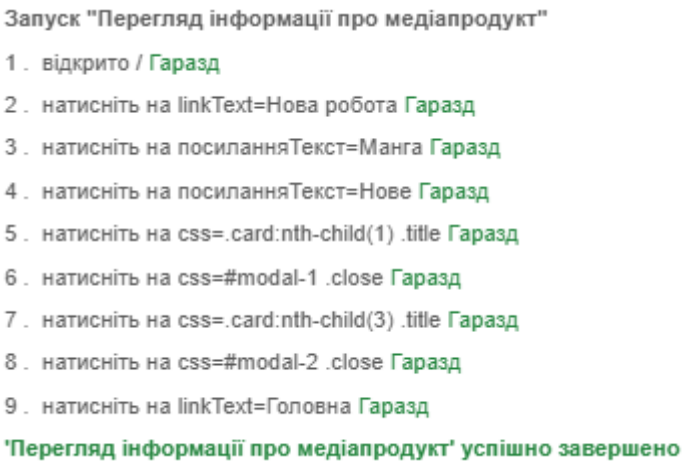

Рисунок 1 – Результати виконання тест-кейсу для перевірки функціоналу перегляду інформації про медіапродукт

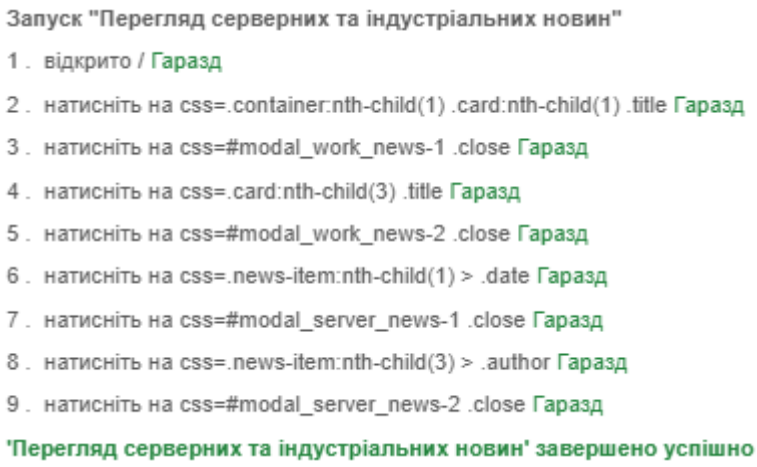

Рисунок 2 – Результати виконання тест-кейсу для перевірки функціоналу перегляду серверних та індустріальних новин

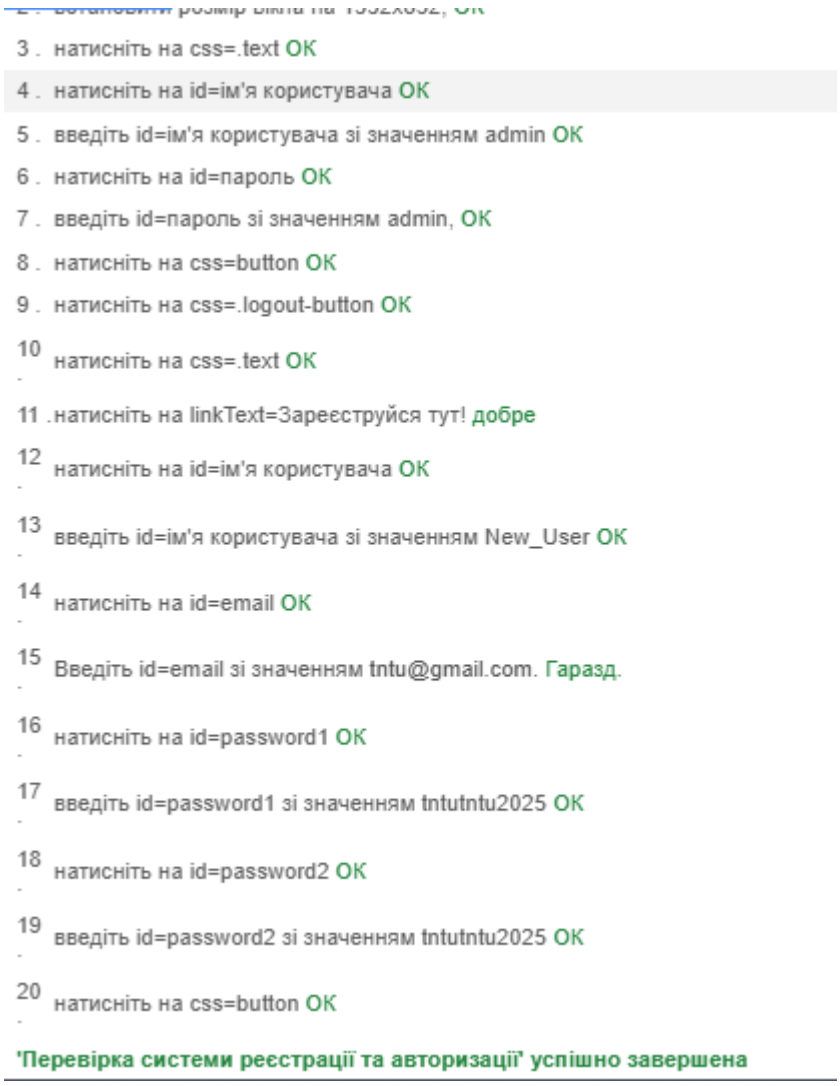

Рисунок 3 – Результати виконання тест-кейсу для перевірки системи реєстрації та авторизації

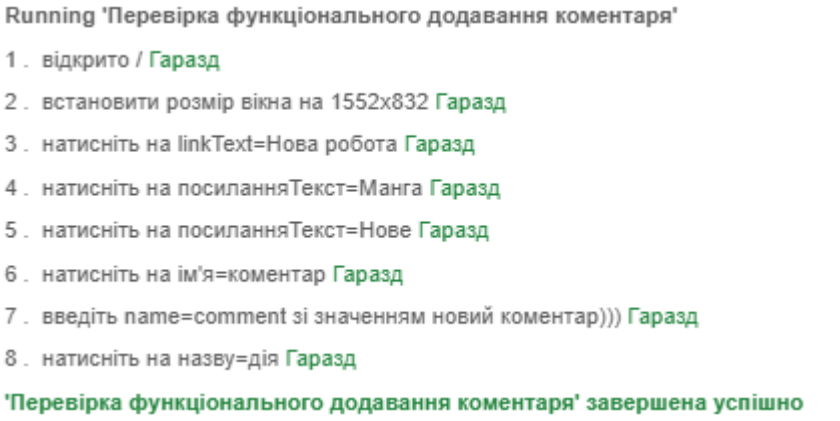

Рисунок 4 – Результати виконання тест-кейсу для перевірки функціоналу додавання коментаря

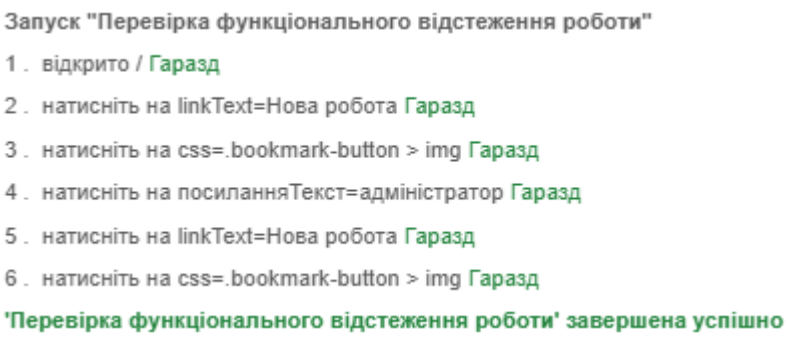

Рисунок 5 – Результати виконання тест-кейсу для перевірки функціоналу відстежування роботи

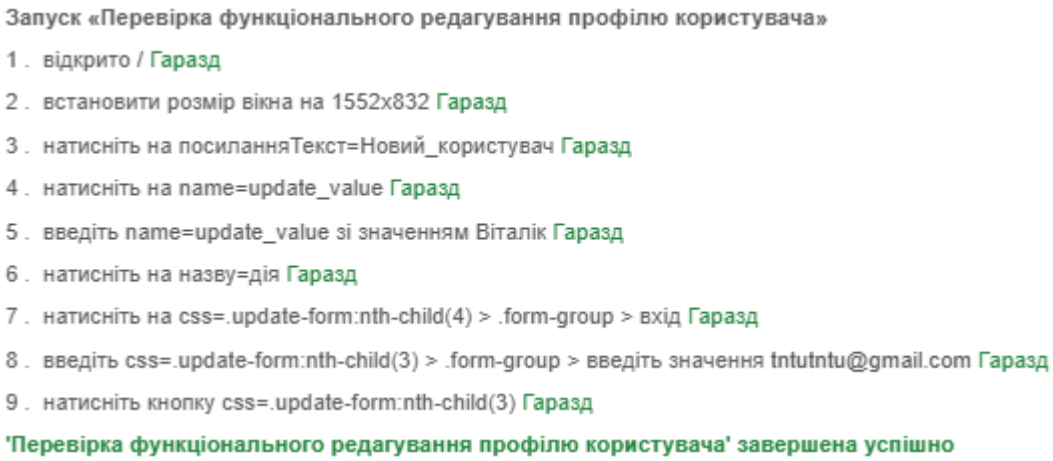

Рисунок 6 – Результати виконання тест-кейсу для функціоналу редагування профілю користувача

Додаток Ж. Диск з роботою