Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматики
Кафедра комп'ютерних наук

Бакалаврська дипломна робота на тему:

РОЗРОБКА WEB-ПЛАТФОРМИ ДЛЯ ПУБЛІКАЦІЇ КОМІКСІВ

Виконала: студентка 4 курсу, групи 1КН-21б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Сичова М. С.
(прізвище та ініціали)
Керівник: к.т.н., доцент каф. КН
Сілагін О. В.
(прізвище та ініціали)
«04» червня 2025р.
Рецензент: к.т.н., доцент каф. САІТ
Горячев Г. В.
(прізвище та ініціали)
«05» червня 2025р.

Допущено до захисту
Зав. кафедри Яровий А.А.
«06» червня 2025 р.

Вінниця ВНТУ – 2025 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра Комп'ютерних наук
Рівень вищої освіти перший (бакалаврський)
Галузь знань – 12 «Інформаційні технології»
Спеціальність – 122 «Комп'ютерні науки»
Освітньо-професійна програма – «Комп'ютерні науки»

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
проф., д.т.н. Яровий А.А.
«05» травня 2025 року

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТЦІ
Сичовій Марині Сергіївні

1. Тема роботи: Розробка WEB-платформи для публікації коміксів.

Керівник роботи: Сілагін Олексій Віталійович, к.т.н., доцент кафедри КН, затверджені наказом вищого навчального закладу від «20» 03. 2025 року №97

2. Термін подання студентом роботи "30" травня 2025р.

3. Вихідні дані до роботи: кількість сутностей бази даних – не менше 4; кількість вебсторінок – не менше 4; база даних – реляційна; середовище реалізації програмного продукту – браузер; дизайн користувацького інтерфейсу має бути мінімалістичним та висококонтрастним; наявність інтерфейсу адміністрування.

4. Зміст текстової частини БКР (перелік питань, які потрібно розробити): вступ; обґрунтування доцільності створення платформи для публікації коміксів; аналіз відомих рішень онлайн ресурсів для публікації коміксів; проектування платформи для публікації коміксів; реалізація платформи для публікації коміксів; тестування роботи платформи та аналіз результатів; висновки; список використаних джерел; додатки.

5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень): UML-діаграма класів онлайн платформи для публікації коміксів; UML-діаграма прецедентів онлайн платформи для публікації коміксів; UML-діаграма послідовності платформи для публікації коміксів; ER-модель платформи для публікації коміксів; схема алгоритму роботи платформи для публікації коміксів.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1-3	Сілагін О. В., к.т.н., доцент каф. КН		

7. Дата видачі завдання "05" травня 2025р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1	Обґрунтування доцільності створення платформи для публікації коміксів	05.05.25	07.05.25	
2	Аналіз відомих рішень онлайн ресурсів для публікації коміксів	08.05.25	11.05.25	
3	Проектування платформи для публікації коміксів	12.05.25	15.05.25	
4	Реалізація платформи для публікації коміксів	16.05.25	26.05.25	
5	Тестування роботи платформи та аналіз результатів	27.05.25	29.05.25	
6	Розробка інструкції користувача	30.05.25	01.06.25	
7	Оформлення матеріалів до захисту БКР	02.06.25	04.06.25	

Студент _____ (підпис) **Сичова М. С.** (прізвище та ініціали)

Керівник роботи _____ (підпис) **Сілагін О.В.** (прізвище та ініціали)

АНОТАЦІЯ

Сичова М. С. Розробка WEB-платформи для публікації коміксів. Бакалаврська дипломна робота зі спеціальності 122 – комп’ютерні науки, освітня програма – комп’ютерні науки. Вінниця: ВНТУ, 2025.

Бакалаврська дипломна робота складається з 92 сторінок формату А4, на яких є 45 рисунків, одну таблицю, список використаних джерел містить 27 найменувань.

В даній бакалаврській дипломній роботі було здійснено розробку онлайн платформи для публікації коміксів. Було досліджено наявні Інтернет-ресурси для публікації коміксів, їхні переваги, недоліки та причини популярності. Спроектовано структуру та функціонал платформи. Для розробки було обрано середовище програмування Visual Studio 2022, мову програмування C#, фреймворк ASP.NET Core, систему управління базами даних Miscrosoft SQL Server. Тестування програмного продукту довело, що основні поставленні цілі були досягнені, є можливість розширювати можливості платформи у майбутньому.

Ключові слова: WEB-платформа, комікси, клієнт-серверна архітектура, реляційна база даних, UML-діаграма.

ABSTRACT

Sychova M. S. Development of the web platform for comics publishing. Bachelor's thesis in specialty 122 – computer science, educational program – computer science. Vinnytsia: VNTU, 2025.

The bachelor's thesis consists of 92 pages of A4 format, which contain 45 figures, one tables, the list of sources used contains 27 sources.

In this bachelor's thesis, an online platform for publishing comics was developed. Existing Internet resources for publishing comics were studied – their advantages, disadvantages, and reasons for popularity. The structure and functionality of the platform were designed.

The Visual Studio 2022 programming environment, C# programming language, ASP.NET Core framework, and Microsoft SQL Server database management system were chosen for the development. Testing of the software product proved that the main goals were achieved, and there is an opportunity to expand the platform's capabilities in the future.

Keywords: web platform, comics, client-server architecture, relational database, UML diagram.

ЗМІСТ

ВСТУП

Актуальність. Формат розповіді історії через зображення зародився більше 50000 років тому, що підтверджуються малюнками у печерах. Згодом вони переросли у єгипетські ієрогліфи, грецькі фризи, середньовічні гобелени, ілюстровані священні писання і т. д.

З розвитком друку у XVII та XVIII столітті гравюри почали використовуватись для критики різних аспектів соціального та політичного життя. У XIX столітті такі зображення мали назву «карикатури» та з наростанням популярності газет самі почали ставати більш популярними. У кінці XIX століття, карикатури почали серіалізуватися, доки у XX столітті вони не переросли у самостійний вид літератури.

З розвиток технологій та збільшенням вартості друку, комікси знайшли своє місце в Інтернет-середовищі, залучаючи до себе ще більше людей, у яких були власні ідеї для сюжетів, але не володіли достатнім словниковим запасом та майстерністю письма, щоб виразити їх у текстовій формі.

Сучасні Інтернет-ресурси дозволяють художникам легко публікувати свої роботи та за бажанням отримувати з них прибуток. Отже, розробка зручної платформи для публікації коміксів, яка повинна виділятись серед інших своєю зручністю використання та політикою роботи, є досі актуальної задачею, рішення якої може принести її власникам значний прибуток з відсотків від заробітку художників.

Метою бакалаврської кваліфікаційної роботи є розширення функціональних можливостей WEB-платформи для публікації коміксів.

Об’єктом дослідження бакалаврської кваліфікаційної роботи є процес створення WEB-платформи для публікації коміксів.

Предметом дослідження бакалаврської кваліфікаційної роботи є алгоритми та програмні засоби для реалізації WEB-платформи для публікації коміксів.

Задачі дослідження:

- Аналіз доцільності та відомих рішень створення ресурсів для публікації коміксів;
- Проектування онлайн платформи для публікації коміксів;
- Реалізація клієнтської та серверної частини онлайн платформи для публікації коміксів;
- Тестування та аналіз отриманих результатів.

Апробація роботи. Результати досліджень було апробовано на LIV Всеукраїнській науково-технічній конференції факультету інтелектуальних інформаційних технологій та автоматизації Вінницького національного технічного університету (НТКП ВНТУ - 2025) м. Вінниці у 2025 р.

Публікації. За основними результатами досліджень бакалаврської дипломної роботи було опубліковано тези доповіді на науково-технічній конференції [1].

Результати, одержані в процесі виконання бакалаврської кваліфікаційної роботи, а саме алгоритми та програмні засоби, планується використати в розробках МНВП ТОВ «ІТІ», про що у додатку Д є відповідна довідка про впровадження.

1 ОБҐРУНТУВАННЯ ДОЦІЛЬНОСТІ СТВОРЕННЯ ОНЛАЙН ПЛАТФОРМИ ДЛЯ ПУБЛІКАЦІЇ КОМІКСІВ

1.1 Проблеми у проектуванні, експлуатації вебресурсів з публікації графічного артдизайну

Публікація коміксів в Інтернеті почалася майже в той самий час, що й онлайн-передача файлів. Американська Інтернет-компанія CompuServe, яка була заснована у 1969 році та припинила свою роботу під цим іменем у 2009 році, славиться не лише статусом першого провайдера Інтернет-послуг, а й статусом хостингу першого онлайн-коміксу «Witches and Stitches», який було оприлюднено у 1985 році Еріком Міллікіном. Ця робота стала поштовхом до публікації власної творчості для тої невеликої на той час кількості користувачів Інтернету, яка переважно складалась із студентів університету [2].

Іншими поштовхами для карикатуристів-початківців почати свою кар’єру з цифрового формату були немала ціна друку стрипів та прискіпливість газетних синдикатів [3]. Останні часто були консервативними до нових ідей, тому художники публікували свої роботи в Інтернеті, не надаючи уваги тому, хто буде їхньою аудиторією. Але ця свобода часто не давала можливості карикатуристам перетворити свій талант у щось більше, ніж хобі. Художники публікували свої комікси на власних сайтах та поширювали посилання на них серед друзів чи через пости на інших сайтах. Хостинг вимагав значну суму грошей, які ці художники часто заробляли з продажу мерчандайзу, як-от футболок із зображеннями персонажів своїх коміксів.

У період з 90-х років і далі завдяки появі більшої кількості соціальних мереж, вебсайтів для публікації коміксів та/чи краудсорсингу, отримувати гроші за власну працю стало набагато легше і цифровий формат коміксів почав розвиватися швидше.

Автори веб-коміксів почали створювати колективи, щоб підтримувати один одного та ділитися аудиторією. У 1997 році Брайан МакНет заснував хостинг для вебкоміксів, назвавши його Big Panda. Це був перший портал для вебкоміксів, який був закритий у 2000 році через малий до нього інтерес Інтернет-користувачів. Скоро після цього було створено новий портал вебкоміксів, який мав назву Keenspot (рис. 1.1). Цей новий портал став великим успіхом.

Рисунок 1.1 — Головна сторінка Keenspot

У 2002 році у Keenspot з’явилась платформа-конкурент Modern Tales, яка стала однією з перших прибуткових моделей підписки на вебкомікси. У Modern Tales до 2005 року було 2000 учасників, кожен з яких платив 3 долари США на місяць. У тому ж році Keenspot залучав близько 125 000 читачів на день, заробляючи понад 200 000 доларів США на рік через рекламу. Відомі художники коміксів, такі як Карла Спід Макнейл і Ліа Ернандес, виявили, що рухаються до Інтернету, щоб охопити більшу аудиторію та створити «онлайн-портфоліо». З 2012 року платформа Modern Tales більше не активна.

Популяризація коміксів досягла кульмінації з публікацією книги Скотта Макклауда «Перевинахід коміксів» у 2000 році. Захоплений можливостями

Інтернету, Макклауд присвятив значну частину своєї книги трьом сферам, де комп'ютерні технології та комікси можуть перетинатися: цифрове виробництво (виробництво коміксів на комп'ютері), цифрова доставка (публікація коміксів онлайн) і цифрові комікси (створення коміксів, спеціально розроблених для Інтернету). Реакція на «Перевинахід коміксів» була неоднозначною, але книга все одно дала початок розквіту експериментів із коміксами у форматі WEB. Деяких вебкарикатуристів надихнула книга Макклауда; інші просто користувалися перевагами нових цифрових інструментів, доступних для художників. У 2000 році Патрік Фарлі запустив Electric Sheep Comics, веб-сайт для своїх численних експериментальних комікс-проектів. У 2002 році Кіт Гарза започаткував Cuentos de la Frontera, колекцію іспаномовних народних казок і міських легенд, адаптовану для дослідження ідеї Макклауда про формат коміксів типу «нескінченне полотно». У 2001 році швейцарський карикатурист Demian5 опублікував свою безмовну нескінченну стрічку «When I Am King» (рис. 1.2).

Рисунок 1.2 – Головна сторінка сайту коміксу «When I Am King»

У цей період Інтернет і друк почали підживлювати одне одного. Оскільки прямий ринок ставав дедалі ворожішим до коміксів малої преси, багато самостійних художників перемістили свої роботи в Інтернет. Комікс «Girl

Genius» Філа та Кайї Фоліо (рис. 1.3), який почав своє життя як друкований комікс у 2001 році, став феноменом після того, як у 2005 році вийшов у мережу, привернувши величезну кількість читачів і вигравши Фоліос дві нагороди Х'юго (єдині дві нагороди Г'юго за графічні історії). Інші комікси були запущені в Інтернеті, а потім перейшли до друку. Графічний роман Джина Янга, номінований на національну книжкову премію «American Born Chinese» (2006), спочатку був опублікований на сайті Modern Tales.

Рисунок 1.3 – Головна сторінка сайту коміксу «Girl Genius»

З розвитком цифрових пристроїв з'явився новий макет коміксів. У Південній Кореї на початку XXI століття почала набирати популярність модель розташування панелей коміксу одна під одною, перетворюючи послідовність книжкових сторінок одного розділу на одну довгу смугу (рис. 1.4). Такий формат є набагато зручнішим для смартфонів, формат типу «нескінченне полотно» нарешті отримав назву – «вебтун» (англ. Webtoon) [4]. У 2010-ті роки вебсайти для публікації вебтунів почали набирати популярність за межами Південної Кореї, зокрема в Китаї, Тайвані, США та інших країнах Азії та англомовних країнах. станом на 2024 рік платформа WEBTOON, найбільша та

найпопулярніша платформа для публікації коміксів налічувала близько 166 мільйонів щомісячно-активних користувачів по всьому світу [5].

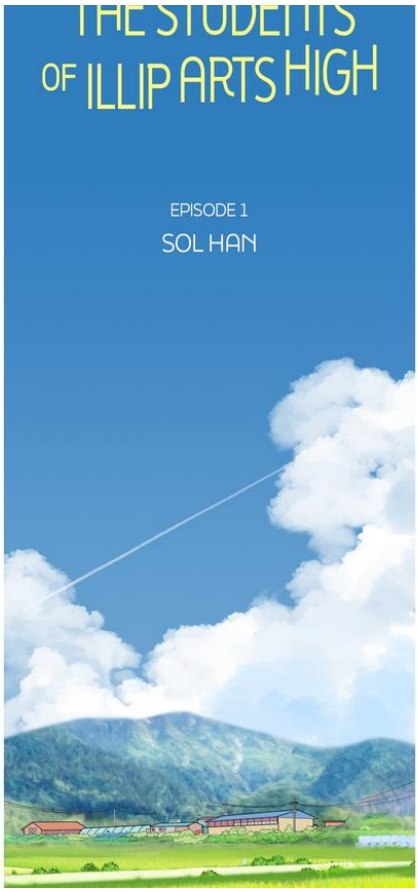

Рисунок 1.4 — Приклад вебтуну

На сьогодні, художники часто використовують соціальні мережі для публікації своїх робіт. Таким чином зручніше залучати більшу аудиторію. В більшості випадків першими «хостингами» стають Twitter/X, Tumblr та Instagram [6-8]. Twitter/X є соціальною мережею, створеною у 2006 році та орієнтованою на мікроблоги, дозволяє публікувати відносно короткі текстові повідомлення разом з прикріпленими фото. Цю соціальну мережу зручно використовувати для публікації коротких коміксів, кількість сторінок одного випуску яких знаходиться в межах чотирьох сторінок. Коли користувачі хочуть включити більшу кількість сторінок, їм доводиться створювати кілька постів, що є не дуже зручним для читачів. Тому вони найчастіше використовують ці

соціальні мережі для публікації фрагментів нових випусків разом із посиланнями на повні варіанти, які розміщуються на інших вебсайтах. Tumblr, який було створено у 2007 році, є більш адаптованою для поширення графічного контенту, не має таких строгих обмежень в розмірах постів у порівнянні з Twitter/X, дозволяє налаштовувати зовнішній вигляд блогів. Instagram, який було створено у 2010 році, імпонує більше художникам, які займаються своїм ремеслом професійно. Подібно до Twitter/X, він має обмеження до розміру постів, але є більш зручнішим засобом для самореклами.

Серед художників, які користуються платформою Twitter/X, користуються популярністю такі платформи як Pixiv (рис. 1.5) та Poipiku [9-10]. Обидві платформи є онлайн галереями, які дозволяють публікувати повноцінні випуски, перша навіть дозволяє отримувати пожертви від підписників. Але вони є не дуже зручними для публікації кількох серій. Саме тому платформи, розраховані для публікації повноцінних серій коміксів мають право на існування.

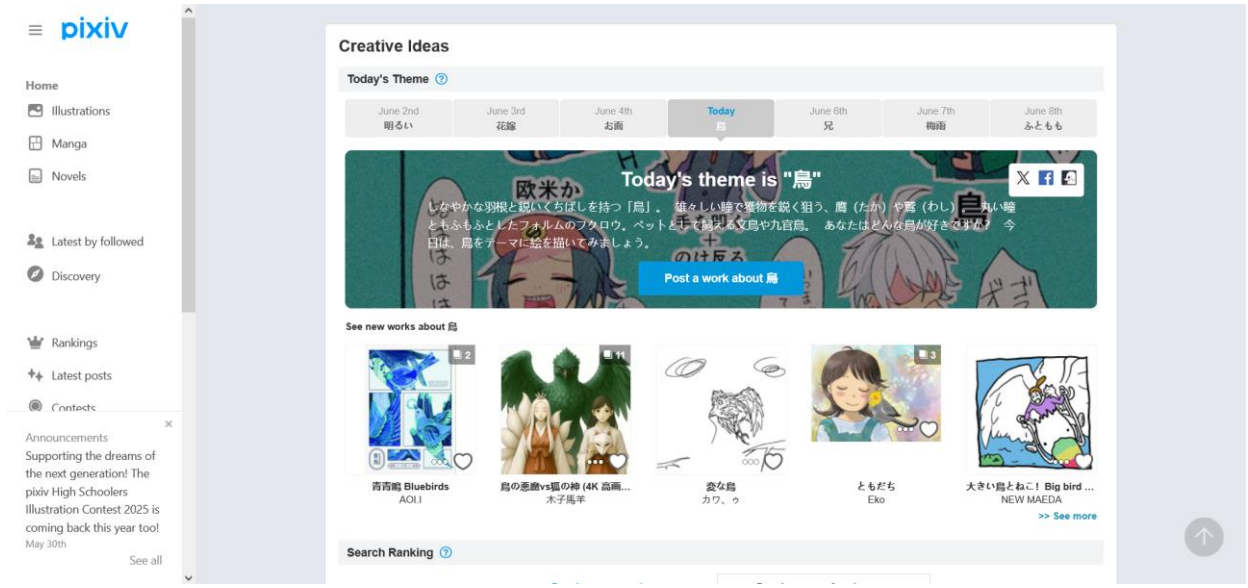

Рисунок 1.5 – Pixiv, сторінка «Creative ideas»

1.2 Аналіз відомих рішень онлайн ресурсів для публікації коміксів

WEBTOON – найбільша та одна із найперших платформ для вебтунів, встановлена в 2004 році (рис. 1.6) [11]. Вона дозволяє художникам виставляти розділи серій згідно розкладу з можливістю збереження чорновикових варіантів цих розділів та їх редагування як до так і після їх публікації.

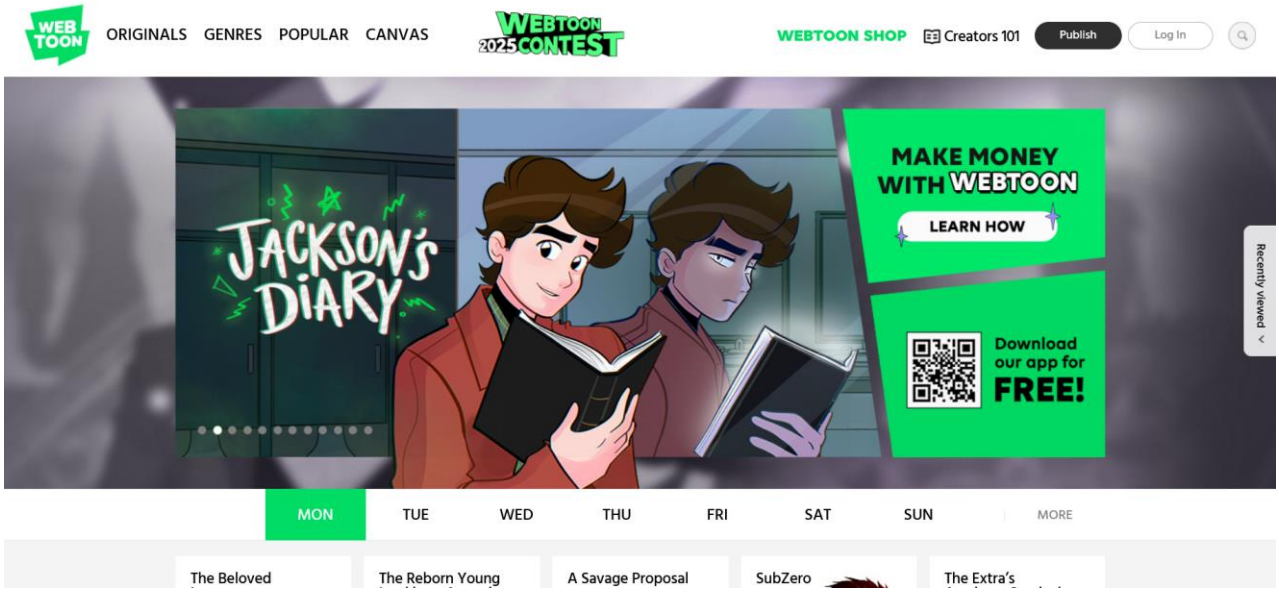

Рисунок 1.6 – Головна сторінка WEBTOON

Автори можуть монетизувати свої комікси, беручи участь у програмах «Super Like Program» чи «Ad Revenue Sharing Program». Серія може отримати статус «Super Like» якщо її автору вже виповнилось вісімнадцять років, він подав заявку для участі в програмі, серія відповідає правилам спільности та вона набрала всього 500 чи більше лайків. Після отримання серією статусу «Super Like», читачі можуть купувати для неї лайки і залишити коментар, який буде поміщено в секцію «Super Fan» для коментарів. Чим більше лайків було куплено, тим довше коментар буде розміщуватися в цій секції, а після закінчення терміну його буде переміщено в звичайну секцію для коментарів. Автор серії отримує 70% з купівлі читачами лайків. «Ad Revenue Sharing Program» дозволяє авторам серій отримувати 50% прибутку від реклами на їхніх серіях. Для участі серії в програмі, вона повинна мати 1000 підписників та

100000 щомісячних переглядів. Крім участі в програмах, автори можуть налаштувати інтеграцію з Patreon.

Художники можуть переглядати такі дані як кількість опублікованих робіт в місяць, кількість переглядів в місяць, прибуток з реклами і т. д.

Читачі мають можливість оцінювати серії, залишати коментарі та модифікувати їх пізніше, оцінювати коментарі інших читачів. Система пошуку оснащена штучним інтелектом, що дозволяє читачам знаходити нові твори, найбільш подібні до їх вподобань.

Платформа отримала свою популярність завдяки успішній рекламі та можливості читати і публікувати вебтуни безкоштовно. Читачі загалом позитивно оцінюють свій досвід користування вебсайту, але автори коміксів часто критикують програми через часто недосяжні вимоги вступу та занадто малий заробіток після їх дотримання. Через важкі умови праці автори часто додатково створюють акаунт на Tapas і не рідко потім зовсім покидають WEBTOON.

Платформа Tapas була встановлена в 2012 році та подібна до WEBTOON в плані функціоналу (рис. 1.7) [12]. Вебсайт є більш адаптованим для читання як на вертикальних, так і на горизонтальних екранах в порівнянні з WEBTOON. На вибір читачів є безкоштовні серії та серії, розділи яких потрібно купувати.

Tapas також пропонує свої програми для заробітку: «Ad Revenue Program», «Support Program» та «Merch Shop». «Ad Revenue Program» пропонує 70% прибутку від реклами, для участі в ній серія повина мати лише 100 підписників. Серія може брати участь у програмі «Support Program» після досягнення 250 підписників. Читачі можуть надсилати авторам валюту платформи «Ink». Автори отримують 100% від ціни отриманої валюти, не враховуючи комісій. Програма «Merch Shop» дозволяє авторам продавати футболки із дизайном, пов’язаним із серіями, які досягли 500 підписників.

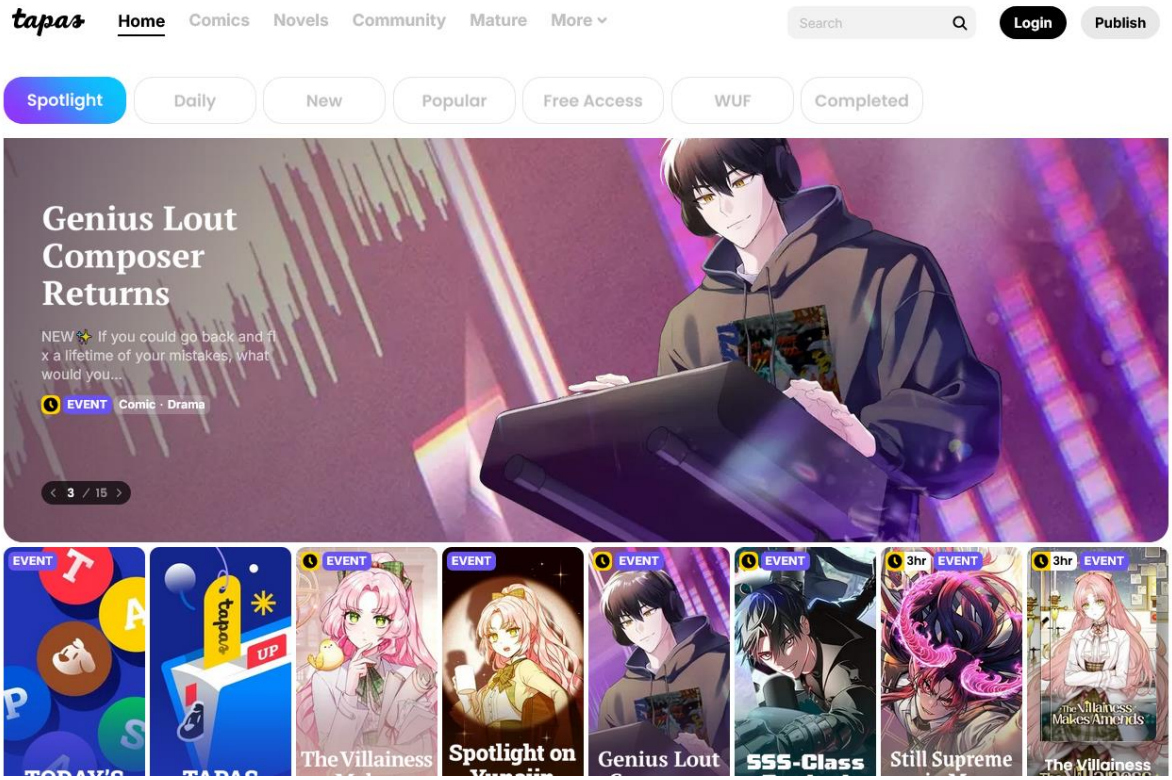

Рисунок 1.7 – Головна сторінка Tapas

Платформа GlobalComix була запущена з 2017 року та є однією із провідних для любителів як класичного сторінкового формату коміксів, так і формату вебтунів (рис. 1.8) [13]. Вона заповнена незалежними художниками, які можуть працювати у власному темпі та отримувати заробіток із випусків. Навіть є можливість продавати випуски у паперовому форматі, від автора вимагається лише встановити ціну. Користувацький інтерфейс є одним із найзручніших, з можливістю налаштовування розмірів та кількості одночасно відображених панелей.

Через відносну новизну платформи та відсутності великий рекламних кампаній, платформа не привертає до себе уваги користувачів більш популярних альтернатив. На теперішній час GlobalComix збирає фанатів класичних коміксів та авторів, яких не влаштовує політика функціонування цих самих більш популярних альтернатив.

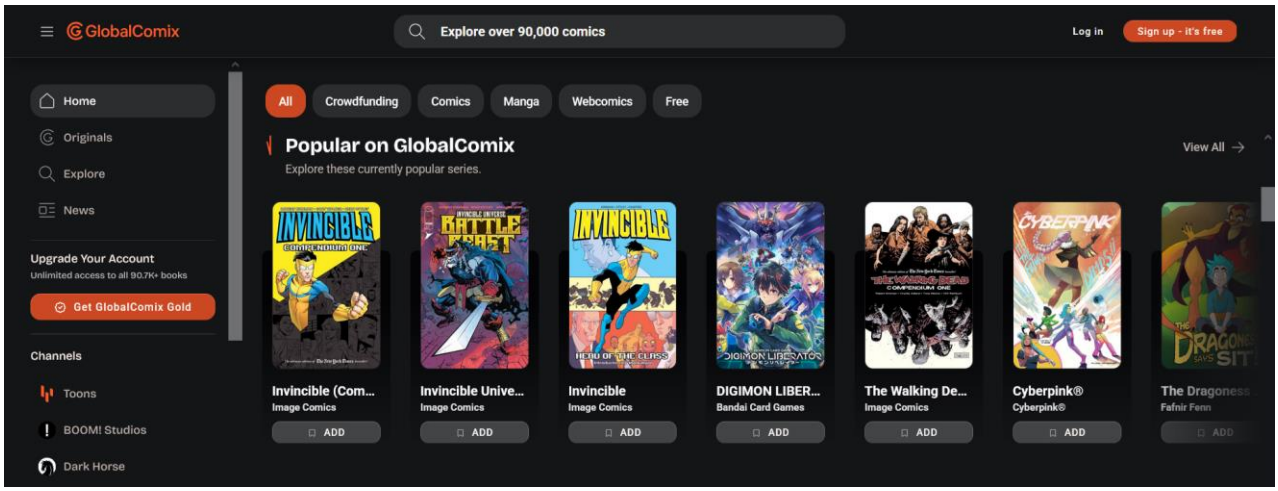

Рисунок 1.8 – Головна сторінка GlobalComix

Отже, аналіз наведених платформ показує, що на позитивний досвід з платформою для коміксів впливає не лише зручний інтерфейс, а й її політика роботи та наявність безкоштовних робіт, які стають візитними картками платформ.

1.3 Формулювання вимог та постановка задачі для створення платформи для публікації коміксів

Основна ціль розробки онлайн платформи є створення універсального ресурсу в плані підтримуваних форматів коміксів та способах заробітку для їх авторів.

Для кожного зареєстрованого користувача має бути наданий такий асортимент функцій:

- налаштування профілю з аватаром, можливістю зміни електронну пошту та паролю та портфоліо робіт;
- завантаження та публікація власних коміксів з можливістю їх редагування;
- контроль видимості коміксів (загальнодоступні, приватні);
- аналітика та статистика («перегляди», «оцінки», «коментарі»);
- можливість залишати коментарі до епізодів, оцінки;

- закладки для улюблених серій.

Упродовж розробки потрібно виконати:

1. проаналізувати технології та середовища розробки для платформи;
2. визначитись з архітектурою платформи;
3. розробити загальний алгоритм роботи платформи;
4. розробити структуру та дизайн інтерфейсу користувача;
5. провести тестування реалізованої платформи.

1.4 Висновки

В першому розділі був проведений короткий екскурс в історію встановлення онлайн коміксів та деякі проблеми з використанням наявних платформ для їх публікації та інших Інтернет-ресурсів для оприлюднення зображень. Результати аналізу показали, що на вибір користувачів впливає як зручність читання та великий набір платних і безкоштовних серій, так і адекватність політики роботи для авторів цих серій.

2 ПРОЕКТУВАННЯ ВЕБПЛАТФОРМИ ДЛЯ ПУБЛІКАЦІЇ КОМІКСІВ

2.1 Вибір та обґрунтування архітектурних та структурних рішень

Архітектура програми визначає як її внутрішню організацію, так і засоби її розробки, тому над вибором оптимальної потрібно замислитись на ранній етапах. Розглянемо найпопулярніші архітектури для сучасний вебресурсів.

Вся програма з монолітною архітектурою побудована як єдине ціле зі спільною кодовою базою, базою даних та інтерфейсом користувача. Цей шаблон простий у розробці, тестуванні та розгортанні, він може запропонувати високу продуктивність. Має труднощі з масштабуванням, оновленням і налагодженням, а також підвищений ризик збою та простою. Придатна для невеликих або простих додатків.

У мікросервісній архітектурі, програма складається з кількох незалежних і слабко пов’язаних служб, кожна з яких має власну кодову базу, базу даних і інтерфейс користувача [14]. Ці служби спілкуються одна з одною через чітко визначені API, і їх можна розгортати, оновлювати та масштабувати незалежно. Цей шаблон пропонує багато переваг, таких як гнучкість, модульність, стійкість і гнучкість.

Клієнт-серверна модель ділить систему на дві частини: клієнтську та серверну (рис. 2.1) [15]. Клієнт запитує послуги від сервера, такі як пошук даних, зберігання, обчислення або інші функції. Сервер обробляє клієнтські запити, надсилає відповіді або виконує визначені дії. Сервер і клієнт можуть знаходитися на одній машині або на різних пристроях у мережі. Архітектура характеризується централізованістю, економічною ефективністю, високою продуктивністю і низькою затримкою.

Мережа доправлення контенту (англ. Content Delivery Network, CDN) будується поверх клієнт-серверної архітектури та покращує її ефективність, вводячи проміжні сервери між клієнтом і сервером вебсайту (рис. 2.2) [16]. Ці

CDN-сервери керують деякими комунікаціями між клієнтом і сервером. Вони зменшують вебтрафік до вебсервера, зменшують споживання пропускної здатності та покращують взаємодію з клієнтами.

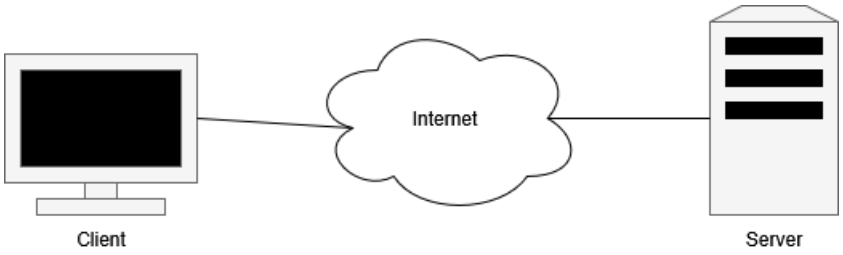

Рисунок 2.1 – Клієнт-серверна архітектура

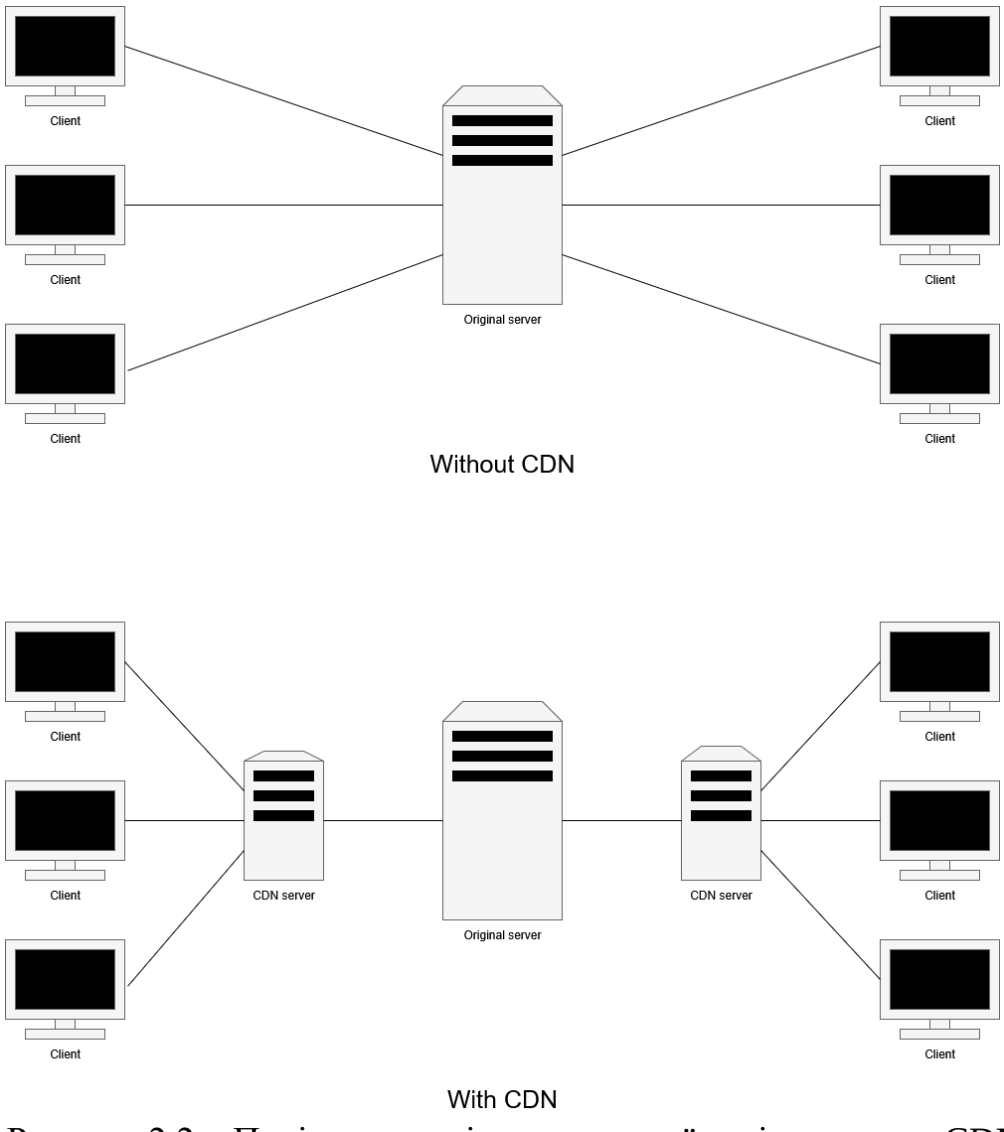

Рисунок 2.2 – Порівняння клієнт-серверної архітектури та CDN

Логіка програми з безсерверною архітектурою виконується хмарним провайдером, а не виділеним сервером [17]. Хмарний постачальник займається наданням, масштабуванням і обслуговуванням обчислювальних ресурсів і стягує плату лише за фактичне використання.

Зважаючи на всі переваги, найбільш підхожим рішенням в даній роботі буде використання клієнт-серверної моделі, яка в майбутньому може перерости у CDN. У контролера також буде зв’язок із базою даних та файловою системою. В базі даних буде міститись загальна інформація про серії та користувачів, файлова система міститиме файли панелей коміксів (рис. 2.3).

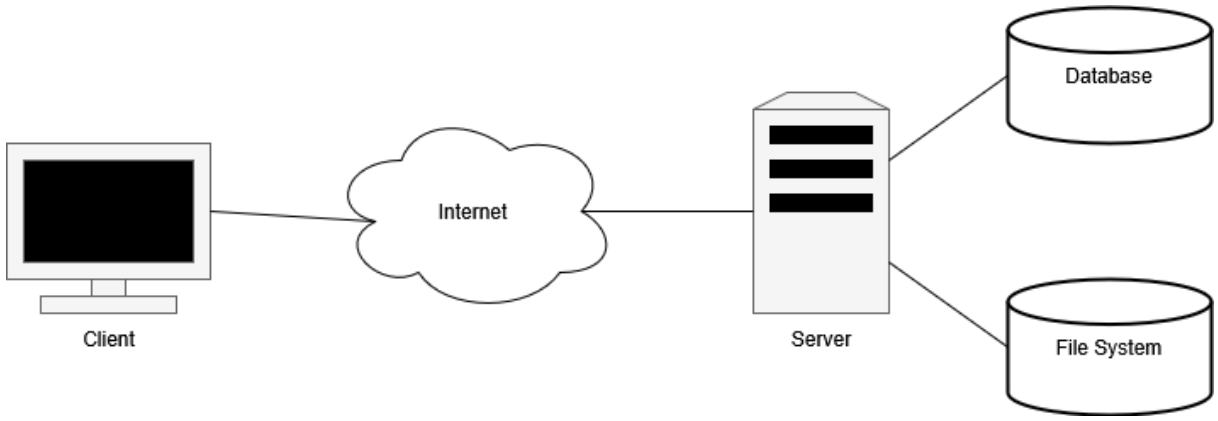

Рисунок 2.3 – Архітектура платформи для публікації коміксів

Вибір типу бази даних на ранніх також має велике значення для вибору засобів розробки. Існує великий вибір варіацій баз даних [18].

Ієрархічні бази даних організовують дані у вигляді дерева, де кожен батьківський запис може мати декілька дочірніх записів (рис. 2.4). Цю структуру також називають батьківсько-дочірніми зв’язками, де кожен батьківський запис може мати кілька дочірніх записів, але дочірній запис може мати лише одного батьківський. Ця модель добре працює для випадків, коли дані дотримуються попередньо визначених ієрархічних зв’язків.

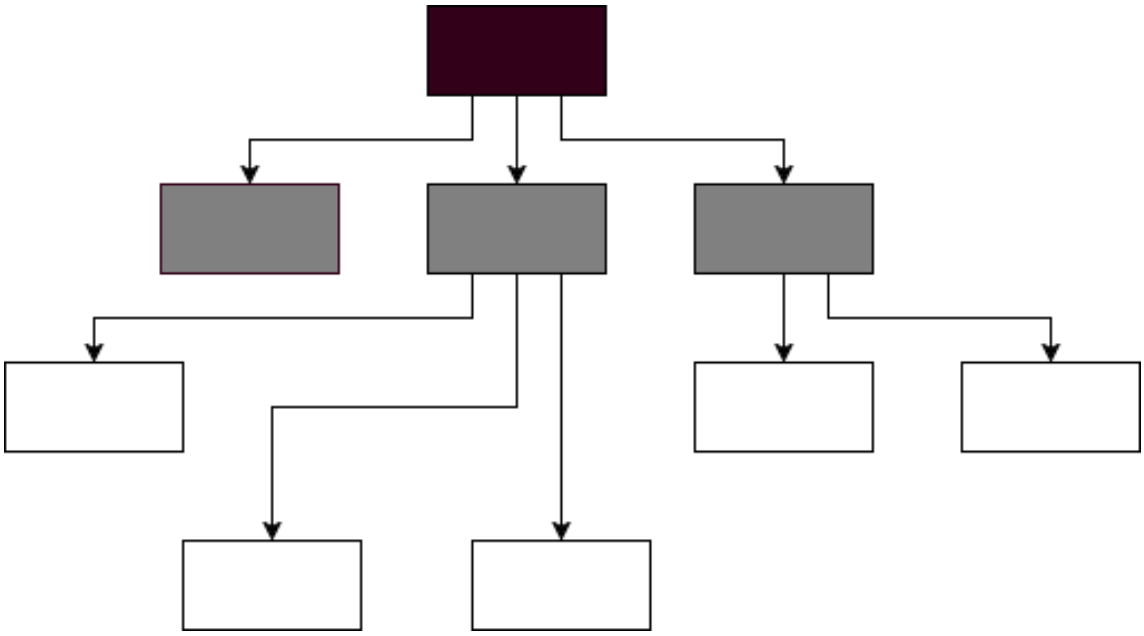

Рисунок 2.4 — Ієрархічна база даних

Мережні бази даних побудовані на основі ієрархічної моделі, але дозволяють дочірнім записам зв’язуватися з декількома батьківськими записами, створюючи мережеву структуру взаємопов’язаних даних (рис. 2.5). Це призводить до більш гнучкої структури, де сутності можна з’єднувати різними способами.

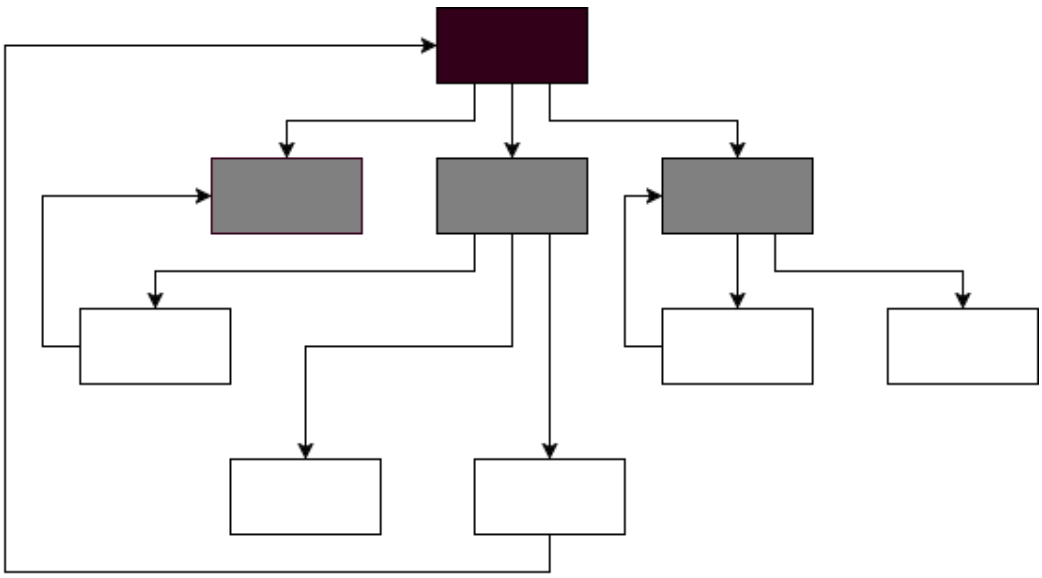

Рисунок 2.5 — Мережева база даних

Об’єктно-орієнтовані бази даних використовують об’єктно-орієнтовану модель даних для зберігання та керування даними (рис. 2.6). Це дозволяє розробникам працювати з даними більш природним способом, оскільки об’єкти схожі на об’єкти мови програмування, яку вони використовують.

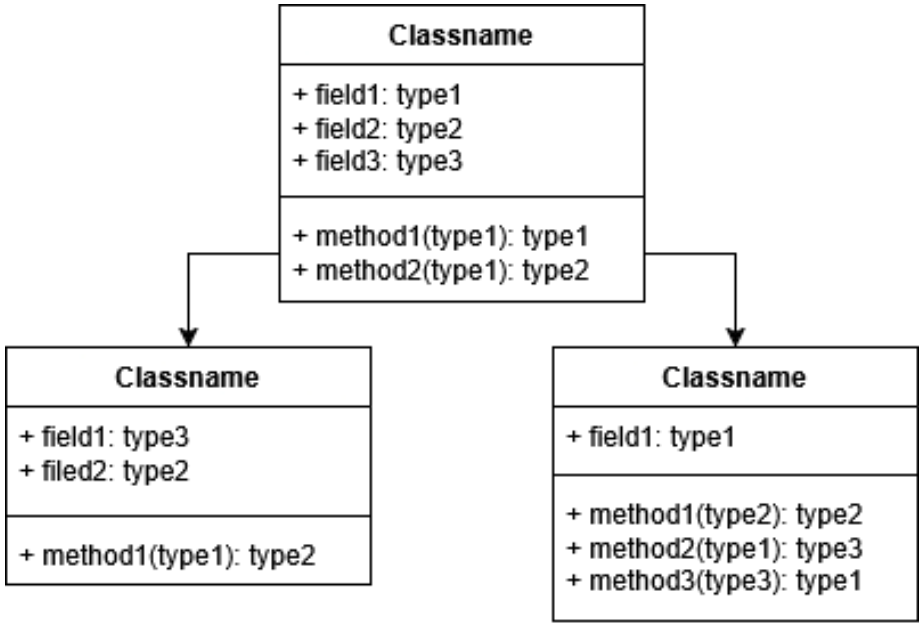

Рисунок 2.6 — Об’єктно-орієнтована база даних

Реляційні бази даних зберігають дані в таблицях, у яких рядки представляють записи, а стовпці — атрибути записів (рис. 2.7). У цій базі даних кожна частина інформації пов’язана з будь-якою іншою інформацією. Це завдяки тому, що кожне значення даних у базі даних має унікальну ідентифікацію у формі запису.

Нереляційна база даних працює зовсім інакше, ніж реляційна база даних. Замість таблиці використовуються структури пар ключів або простих значень — як, наприклад, у файлах JSON. На даний момент існує 4 типи баз даних, які працюють за моделлю NoSQL: колонкова, графова, типу ключ-значення, шаблон документа.

Колонкова NoSQL подібна до реляційної моделі, але тип стовпців NoSQL не працює з таблицями (рис. 2.8). Тут інформація має власні стовпці, які можна групувати в одне сімейство, але це групування не є обов’язковим.

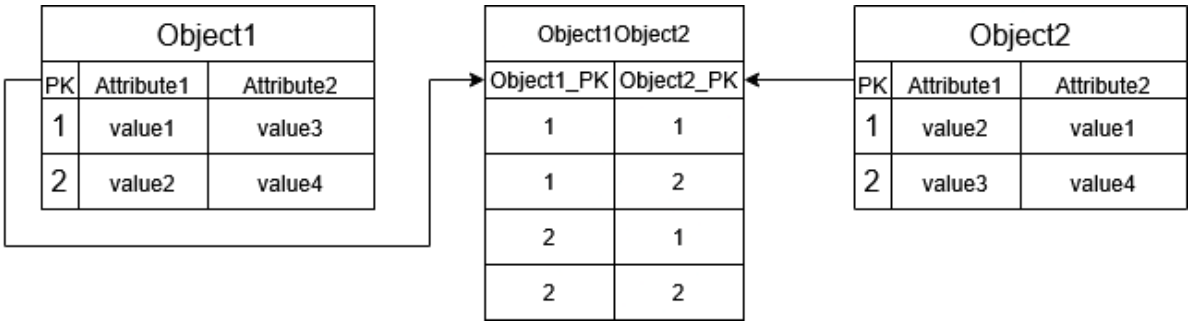

Рисунок 2.7 — Реляційна база даних

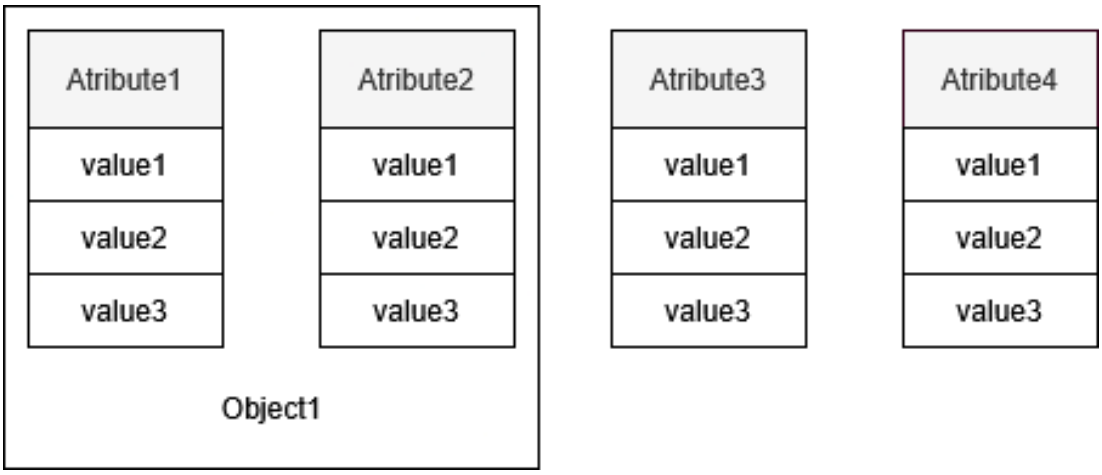

Рисунок 2.8 — Колонкова NoSQL

Графова NoSQL характеризується зберіганням даних у вузлах, з’єднаних ребрами — і з унікальними властивостями. Це забезпечує більшу швидкість під час складних пошуків, де потрібна низька затримка для отримання інформації.

NoSQL типу ключ-значення є одним із найпопулярніших типів NoSQL, саме через її легкість управління та гнучкості в роботі. Для цього його робота проста: дані реєструються в ключі, який, у свою чергу, зберігає певне значення. Таким чином, для запиту даних необхідно лише знати ключ, який також розглядається як властивість, щоб отримати пов’язане значення, яке може мати різні формати.

У шаблоній моделі немає необхідності мати будь-яку попередньо визначену структуру, оскільки вона встановлюється відповідно до потреб програми. Шаблон документа може навіть мати групи документів. Таким чином, можна працювати з окремими документами або навіть набором документів, залежно від потреб програми.

Зважаючи на всі характеристики моделей, було прийнято рішення використовувати реляційну модель. Вона має найзручнішу для розуміння структуру та має велику підтримку користувачів.

Модель «сутність-зв’язок» (ER-модель) — це концептуальна модель для проектування баз даних [19]. Ця модель представляє логічну структуру бази даних, включаючи сутності, їхні атрибути та зв’язки між ними. Сутності зображуються прямокутниками, асоціації — ромбами, зв'язки — ребрами, над якими може проставлятися ступінь зв'язку.

На рисунку 2.9 зображено ER-модель платформи для публікації коміксів. База даних міститиме такі сутності: Користувач (User), Серія (Series), Випуск (Chapter), Сторінка (Page), Жанр (Genre), Коментар (Comment), Закладка (Fave). В одного Користувача може бути багато Серій і багато Коментарів; в одній Серії може бути багато Випусків і багато Жанрів; в одного випуску може бути багато Сторінок і Коментарів; у багатьох Жанрів може бути багато Серій. Наведені сутності мають такі характеристики:

- Користувач: ID, Ім’я, Електронна_пошта, Пароль, Посилання_на_фото_профілю;
- Серія: ID, Назва, Мова, Рік, Вікове_обмеження, Формат, Напрям_читання, Монетизація, Видимість, Опис, Теги, Посилання_на_обкладинку, Посилання_на_папку, Користувач, Жанр, Розділи, Кількість_переглядів;
- Випуск: ID, Назва, Дата_публікації, Посилання_на_папку, Перша_сторінка, Сторінки, Коментарі;

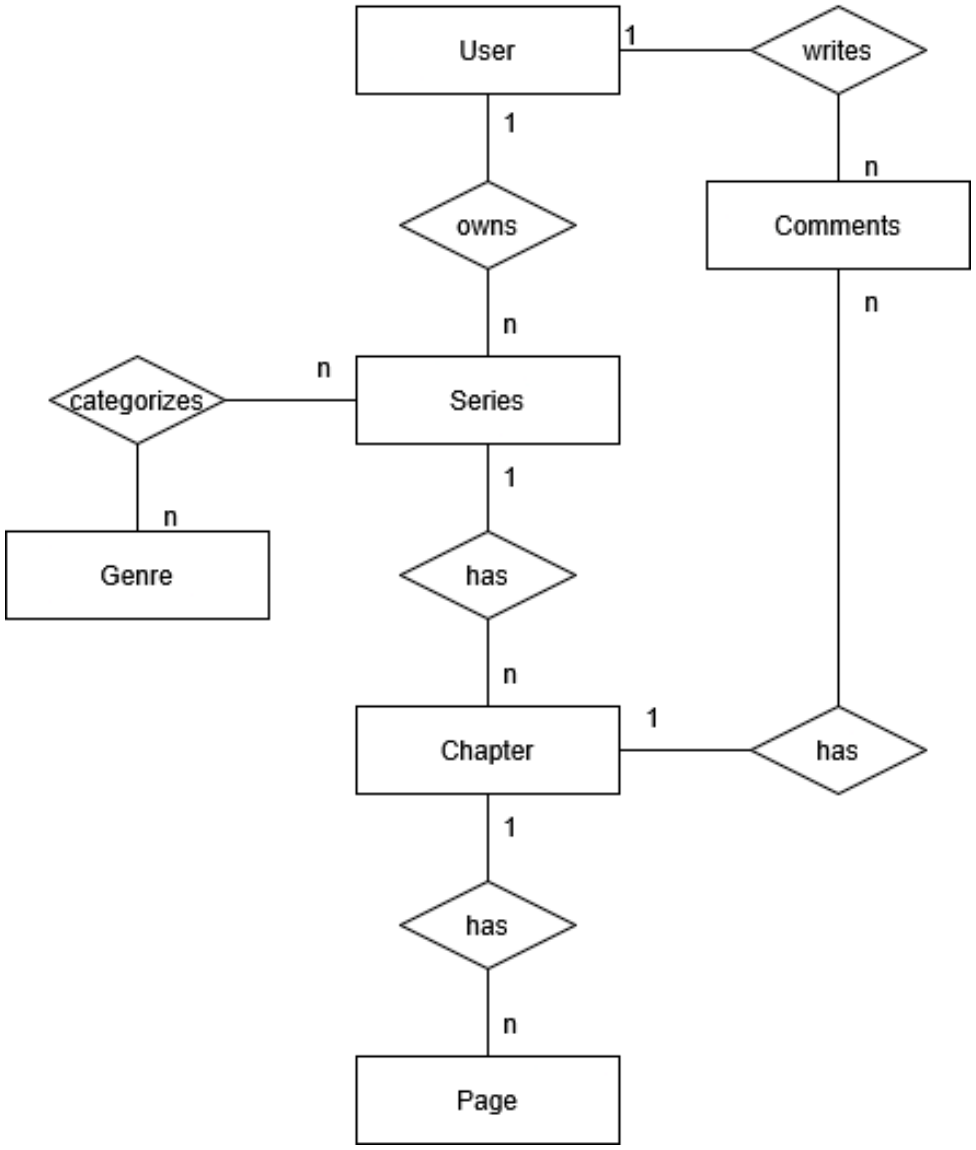

Рисунок 2.9 – ER-модель платформи для публікації коміксів

- Сторінка: ID, Посилання_на_файл, Наступна_сторінка, Розділ;
- Жанр: ID, Назва, Серії;
- Коментар: ID, Дата_публікації, Текст, Користувач, Розділ.
- Закладка: ID, Користувач, Серія.

Універсальне відношення для даної бази даних буде мати наступний вигляд: R (Користувач.ID, Користувач.Ім’я, Користувач.Електронна_пошта, Користувач.Пароль, Користувач.Посилання_на_фото_профілю, Серія.ID, Серія.Назва, Серія.Мова, Серія.Рік, Серія.Вікове_обмеження, Серія.Формат, Серія.Напрям_читання, Серія.Монетизація, Серія.Видимість, Серія.Опис,

Серія.Теги, Серія.Посилання_на_обкладинку, Серія.Посилання_на_папку, Серія.Користувач, Серія.Жанр, Серія.Розділи, Серія.Кількість_переглядів Випуск.ID, Випуск.Назва, Випуск.Дата_публікації, Випуск.Посилання_на_папку, Випуск.Перша_сторінка, Випуск.Сторінки, Випуск.Коментарі, Сторінка.ID, Сторінка.Посилання_на_файл, Сторінка.Наступна_сторінка, Сторінка.Розділ, Жанр.ID, Жанр.Назва, Жанр.Серії, Коментар.ID, Коментар.Дата_публікації, Коментар.Текст, Коментар.Користувач, Коментар.Розділ, Закладка.ID, Закладка.Користувач, Закладка.Серія). Ступінь універсального відношення – 44.

2.2 UML-проектування вебплатформи

Уніфікована мова моделювання (англ. Unified Modeling Language, UML) — це стандартизована мова візуального моделювання дизайну системи [20]. Заощаджується багато часу, коли команди можуть візуалізувати процеси, взаємодію користувачів і статичну структуру системи.

Найпоширенішою діаграмою UML є діаграма класів (Class Diagram). Діаграми класів використовуються для зображення статичної структури системи, показуючи класи системи, їхні методи та атрибути. На рисунку 2.10 зображено діаграму класів платформи для публікації коміксів. Кожен користувач (User) може публікувати нуль і більше серій (Series), яка може мати нуль і більше випусків (Chapter), які у свою чергу можуть мати нуль і більше сторінок (Page). Одна серія може мати кілька жанрів (Genre), один випуск має кілька коментарів (Comment). Багато користувачів можуть закладати багато серій (Faves).

Діаграма прецедентів (Use Case Diagram) використовується для візуалізації взаємодії між учасниками і системою, що розглядається. Для цієї візуалізації вона використовує позначення актора, варіантів використання, та зв’язки включення та розширення. Актори (Actors) – це зовнішні суб'єкти, які

взаємодіють із системою. Це можуть бути користувачі, інші системи або апаратні пристрої. У контексті діаграми варіантів використання актори ініціюють варіанти використання та отримують результати. Варіанти використання (User Cases) представляють конкретні речі, які може робити ваша система. Зв'язок включення (<<include>>) вказує на те, що варіант використання включає функціональність іншого варіанту використання. Зв'язок розширення (<<extend>>) ілюструє, що варіант використання може бути розширений іншим варіантом використання за певних умов. Цей зв’язок корисний для обробки необов’язкової чи виняткової поведінки. Фрагмент діаграми прецедентів платформи для публікації коміксів зображено на рисунку 2.11.

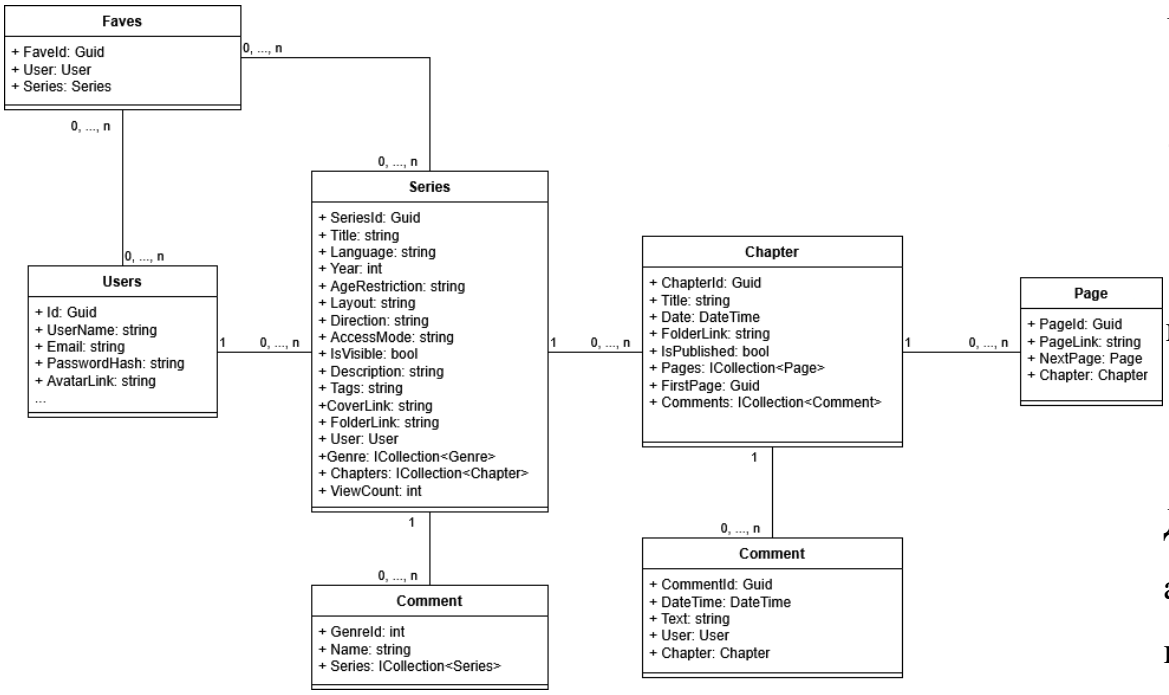

Рисунок 2.10 – Діаграма класів

Діаграма послідовності (Sequence Diagram) зображує взаємодію між об’єктами в послідовному порядку, тобто порядок, у якому відбуваються ці взаємодії. Діаграму послідовності платформи для публікації коміксів зображено на рисунку 2.12. Архітектура платформи побудована на основі архітектури модель-вигляд-контролер (англ. Model-View-Controller, MVC), варіація клієнт-серверної архітектури. Модель визначає формат даних, які використовуються контролерами і виглядами. Вигляд визначає формат відображення даних із

моделі, тобто вебсторінки. Контролери містять логіку, яка змінює вміст моделей і виглядів в залежності від дій користувача. Інтерфейс на рис. 2.12 представляє собою вигляди, сервер — контролери, зв’язки — моделі.

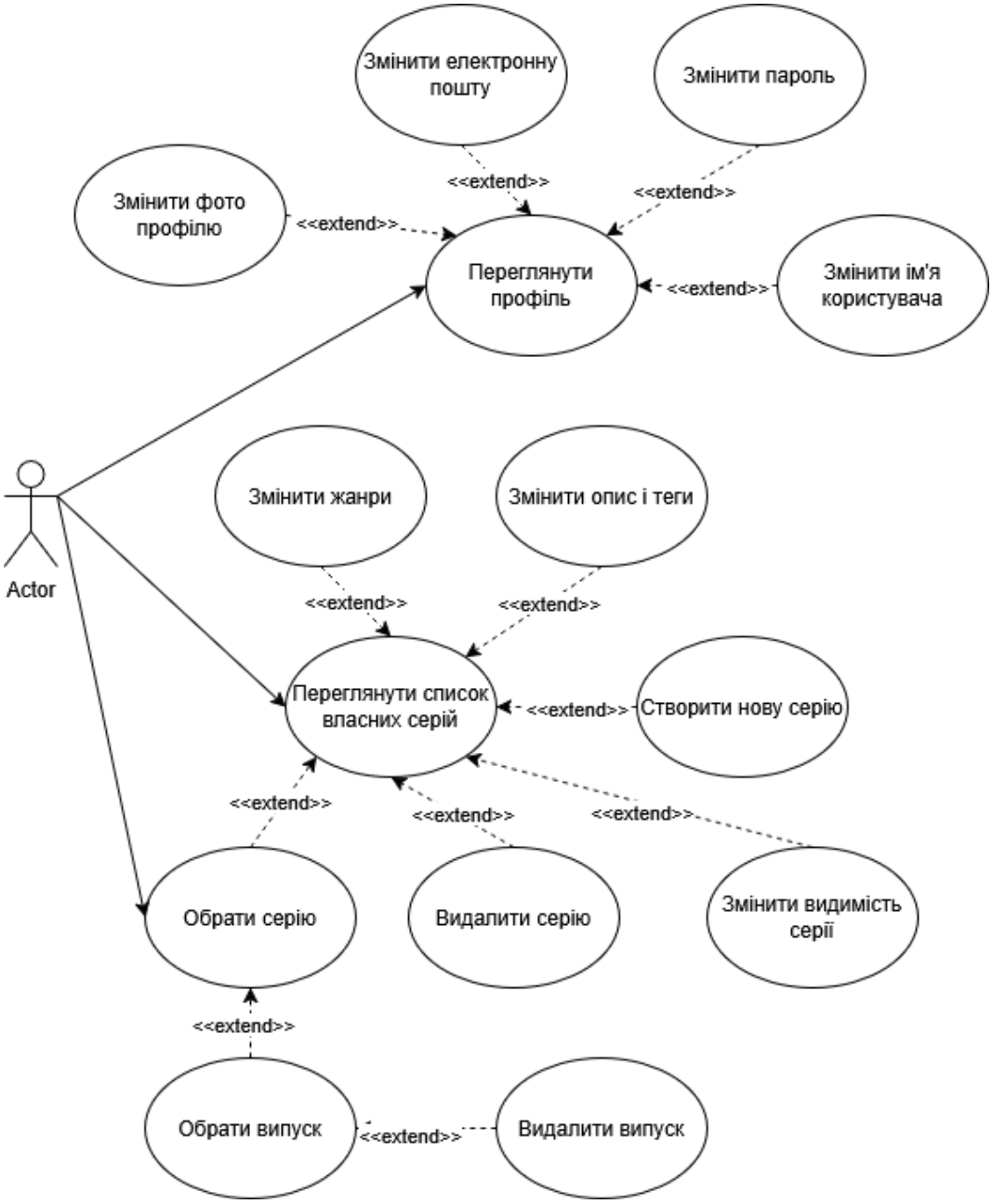

Рисунок 2.11 – Діаграма прецедентів

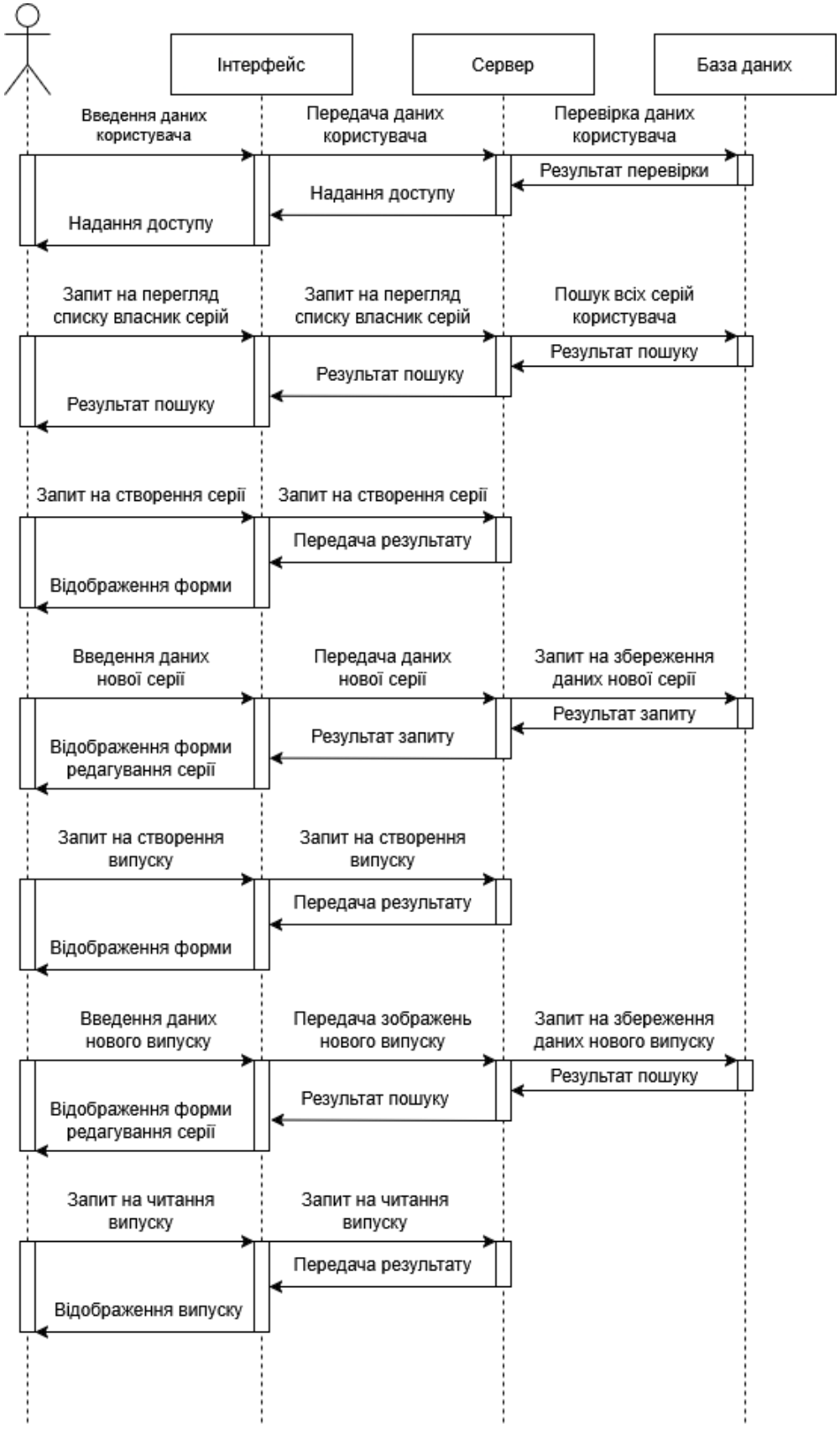

Рисунок 2.12 – Діаграма послідовності

2.3 Розробка алгоритмів роботи платформи для публікації коміксів

Загальний алгоритм роботи платформи для публікації коміксів зображено на рисунку 2.13.

I випадок

Алгоритм реєстрації користувача:

Крок 1: Запуск головної сторінки.

Крок 2: Користувач натискає на посилання «Create account».

Крок 3: Відображення форми реєстрації.

Крок 4: Користувач вводить особисту інформацію: ім’я користувача, електронна пошта, пароль, підтвердження пароля.

Крок 5: Користувач натискає на кнопку «Register».

Крок 6: Якщо реєстрація проведена успішно — перехід до кроку 1, якщо ні — перехід до кроку 3.

II випадок

Алгоритм входу зареєстрованого користувача

Крок 1: Запуск головної сторінки.

Крок 2: Користувач вводить особисту інформацію: електронну пошту, пароль.

Крок 3: Користувач натискає на кнопку «Login».

Крок 4: Якщо вхід успішний — перехід до кроку 1, якщо ні — відображається форма реєстрації.

III випадок

Алгоритм зміни фото профілю користувача:

Крок 1: Виконання дій для II випадку.

Крок 2: Користувач натискає на кнопку «My account».

Крок 3: Відображення форми зміни даних профілю.

Крок 4: Користувач натискає на кнопку «Огляд».

Крок 5: Користувач обирає зображення.

Крок 6: Користувач натискає на кнопку « Change avatar».

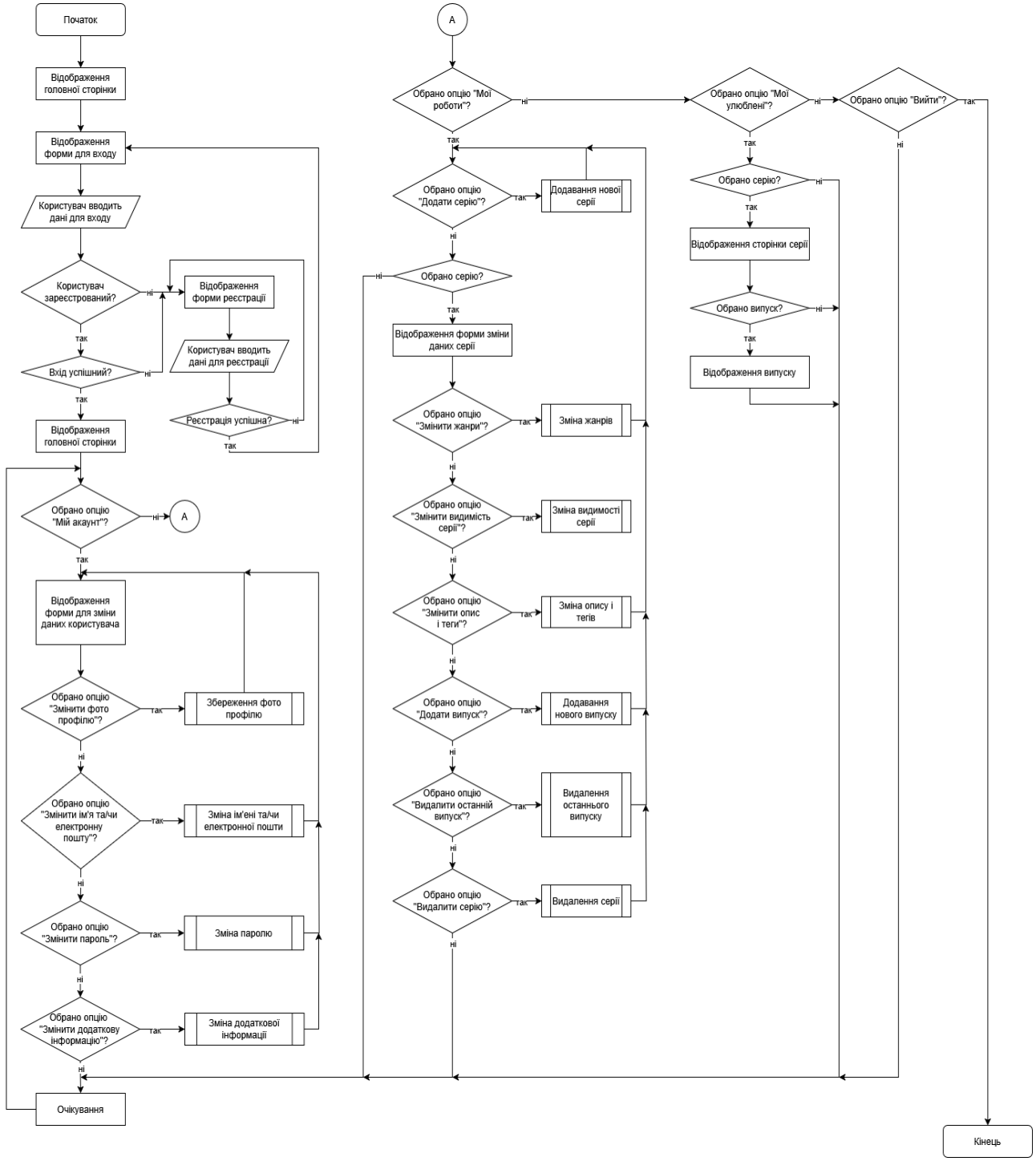

Рисунок 2.13 – Загальний алгоритм роботи платформи для публікації коміксів

Крок 7: Відображення нового зображення профілю.

IV випадок

Алгоритм зміни імені користувача та/чи електронної пошти:

Крок 1: Виконання дій для II випадку.

Крок 2: Користувач натискає на кнопку «My account».

Крок 3: Відображення форми зміни даних профілю.

Крок 4: Користувач вводить нову особисту інформацію: нове ім’я користувача та/чи нову електронну пошту.

Кроку 5: Користувач натискає на кнопку «Save changes».

Крок 6: Відображення новий особистих даних.

V випадок

Алгоритм зміни паролю:

Крок 1: Виконання дій для II випадку.

Крок 2: Користувач натискає на кнопку «My account».

Крок 3: Відображення форми зміни даних профілю.

Крок 4: Користувач вводить теперішній пароль, новий пароль та підтвердження нового паролю.

Кроку 5: Користувач натискає на кнопку «Change password».

Крок 6: Пароль змінено.

VI випадок

Алгоритм додавання нової серії:

Крок 1: Виконання дій для II випадку.

Крок 2: Користувач натискає на кнопку «My works».

Крок 3: Відображення списку серій, опублікованих користувачем.

Крок 4: Користувач натискає на кнопку «Add Another Series».

Крок 5: Відображення форми створення нової серії.

Крок 6: Користувач вводить інформацію про нову серію: назва, жанр, мова, вікове обмеження, формат коміксу, напрямок читання, монетизація, видимість.

Крок 7: Користувач натискає на кнопку «Create series».

Крок 8: Серія створена, перехід до кроку 3.

VII випадок

Алгоритм зміни жанрів серії:

Крок 1: Виконання дій для II випадку.

Крок 2: Користувач натискає на кнопку «My works».

Крок 3: Відображення списку серій, опублікованих користувачем.

Крок 4: Користувач обирає серію із списку.

Крок 5: Відображення інформації про серію.

Крок 6: Користувач обирає жанри.

Крок 7: Користувач натискає на кнопку «Save genres».

Крок 8: Жанри змінено, перехід до кроку 3.

VIII випадок

Алгоритм зміни опису і тегів:

Крок1: Виконання дій для II випадку.

Крок 2: Користувач натискає на кнопку «My works».

Крок 3: Відображення списку серій, опублікованих користувачем.

Крок 4: Користувач обирає серію із списку.

Крок 5: Відображення інформації про серію.

Крок 6: Користувач вводить опис та теги серії.

Крок 7: Користувач натискає на кнопку «Save the description and tags».

Крок 8: Опис та теги збережено, перехід до кроку 3.

IX випадок

Алгоритм зміни видимості:

Крок1: Виконання дій для II випадку.

Крок 2: Користувач натискає на кнопку «My works».

Крок 3: Відображення списку серій, опублікованих користувачем.

Крок 4: Користувач обирає серію із списку.

Крок 5: Відображення інформації про серію.

Крок 6: Користувач натискає на кнопку «Make series visible».

Крок 7: Серія видима для всіх користувачів, перехід до кроку 3.

X випадок

Алгоритм додавання нового випуску:

Крок 1: Виконання дій для II випадку.

Крок 2: Користувач натискає на кнопку «My works».

Крок 3: Відображення списку серій, опублікованих користувачем.

Крок 4: Користувач обирає серію із списку.

Крок 5: Відображення інформації про серію.

Крок 6: Користувач натискає на кнопку «Add a new chapter».

Крок 7: Відображення конструктора створення випуску.

Крок 8: Користувач натискає на кнопку «Add a Page» стільки разів, скільки сторінок містить випуск.

Крок 9: Користувач обирає файли панелей.

Крок 10: Користувач натискає на кнопку «Save a Chapter».

Крок 11: Випуск додано, перехід до кроку 3.

2.4 Висновки

У даному розділі було проведено проектування онлайн платформи для публікації коміксів. Було обґрунтовано вибір клієнт-серверної архітектури, яка найкраще пасує для онлайн платформи.

Було проаналізовано концепції декількох моделей баз даних: ієрархічної, мережевої, об’єктно-орієнтованої, реляційної та NoSQL. Було обґрунтовано вибір реляційної бази даних зважаючи на її простоту реалізації та великою підтримкою її користувачів.

В ході проектування було розроблено UML-діаграми класів, прецедентів і послідовності. За допомогою цих діаграм можна краще зрозуміти структуру та функціонал платформи.

До того ж, було розроблено загальний алгоритм роботи платформи для публікації коміксів та докладніше словесно описано його фрагменти. Ці описи та схема допоможе з розробкою програмного коду.

3 РЕАЛІЗАЦІЯ ВЕБПЛАТФОРМИ ДЛЯ ПУБЛІКАЦІЇ КОМІКСІВ

3.1 Вибір та обґрунтування мови програмування, середовищ розробки, бібліотек та технологій

На вибір мови серверної частини WEB-платформи було взято Python, C#, та Java.

Python – мова, яка підтримує об’єктно-орієнтоване програмування та процедурне програмування [21]. Python використовується для розробки широкого спектру проектів, пов’язаних з наукою про дані, машинним навчанням та навіть веброзробки. Ця мова набула свою популярність за рахунок своєї простоти в розумінні та написанні коду. Але ця простота створює великі недоліки. Python є інтерпретованою, слабо типізованою мовою програмування, тобто код виконується команда за командою без попередньої його компіляції, а змінні протягом свого життя можуть змінювати свій тип. Це робить роботу програми повільнішою та підвищує вірогідність появи помилок.

C# – об’єктно-орієнтована мова програмування загального призначення середнього рівня [22]. Вона була розроблена Андерсом Хейлсбергом у 2000 році як частина технології .NET. Вона має спільні риси з іншими мовами, включаючи C та C++. Як статично типізовану мову, C# легко читати та розуміти, що полегшує пошук помилок у коді як для компілятора, так і для розробників. Це також полегшує написання коду в цілому, причому багаторазовий код є основною функцією C#, що робить її високоефективною, гнучкою, масштабованою, а написані нею програми простими в обслуговуванні. Вона оснащена збирачем сміття, який вилучає з пам’яті змінні, які більше не використовуються програмою, що попереджує витік пам’яті. C# також дозволяє асинхронне програмування для обробки кількох процесів без зупинки основного потоку, а компонент LINQ (англ. Language Integrated Query) дозволяє

створювати запити до бази даних за допомогою синтаксису, який попереджує SQL-ін’єкції.

Java, яка була випущена 1995 року компанією «Sun Microsystems», багато в чому подібна до C# [23]. Так само як і C#, вона є строго типізованою, об’єктно-орієнтованою мовою програмування з можливістю створення асинхронних процесів. Вона також оснащена збирачем сміття та компонентом Hibernate, який аналогічно LINQ, дозволяє створювати запити до баз даних. Але Java не рідко програє C# у швидкості як у плані роботи програм, так і у плані виходу оновлень. Java Virtual Machine, віртуальна машина для інтерпретації байт коду Java, додає більше навантаження на компоненти системи, ніж компоненти середовищ для C#. До того ж, середовища розробки для Java є менш зручними у порівнянні із середовищами для C#.

У таблиці 3.1 наведено основні характеристики, за якими оцінюються переваги розглянутих мов. Найбільше для розробки WEB-платформи для публікації коміксів пасує C#.

C# часто використовується разом з Visual Studio чи Visual Studio Code. Обидва середовища є розробкою Microsoft для роботи з технологіями .NET. Visual Studio орієнтується на задоволення потреб якомога більшої кількості користувачів, які працюють з різними мовами програмування та бібліотеками (рис. 3.1) [24]. Visual Studio надає компілятори, інструменти завершення коду, керування версіями, розширення, методи розробки за допомогою штучного інтелекту і т. ін. Середовище дозволяє завантажувати лише ті бібліотеки та технології, які потрібні розробнику, та створювати програми, які можуть працювати на багатьох платформах. На виріб розробників надаються пакети для роботи з такими мовами програмування як C#, Visual Basic, C++, F#, JavaScript, TypeScript, Python, HTML, CSS, and XML.

Visual Studio Code є легшою версією Visual Studio, їй зазвичай вистачає 300 МБ оперативної пам’яті, в порівнянні Visual Studio, який займає місце до 1 ГБ оперативної пам’яті під час запуску та поступово розширює його [25].

Інтерфейс користувача також є порівняно простим з великою кількістю бокових панелей Visual Studio (рис. 3.2). Але Visual Studio Code орієнтується на скрипти та невеликі проекти та не дозволяє розробникам так само зручно завантажувати повні потрібні пакети засобів, вони повинні самостійно завантажувати всі плагіни. Саме тому для розробки WEB-платформи для публікації коміксів було обрано Visual Studio.

Таблиця 3.1 – Порівняння Python, C#, Java

	Python	C#	Java
Типізація	Слабка	Строга	Строга
Продуктивність	Низка	Висока	Менш висока (JVM додає навантаження)
Підтримувані платформи	Багато-платформна	Багато-платформна	Багато-платформна
Підтримка багатопотоковості та асинхронного програмування	Підтримує	Підтримує	Підтримує
Безпека	Наявність методів об’єктно-реляційного відображення	Наявність методів об’єктно-реляційного відображення, перевірка коду на помилки під час компіляції	Наявність методів об’єктно-реляційного відображення, перевірка коду на помилки під час компіляції
Використання	Наука про дані, машинне навчання, веброзробка	Настільні програми, ігри, хмарні обчислення, веброзробка	Вбудовані системи, ігри, хмарні обчислення, веброзробка

Однією з останніх технологій для .NET-веброзробки є фреймворк ASP.NET Core [26]. ASP.NET Core використовується у вебдодатках в основному

через те, що основна увага приділяється back-end розробці. Тим не менш, він здатний забезпечити і front-end розробку.

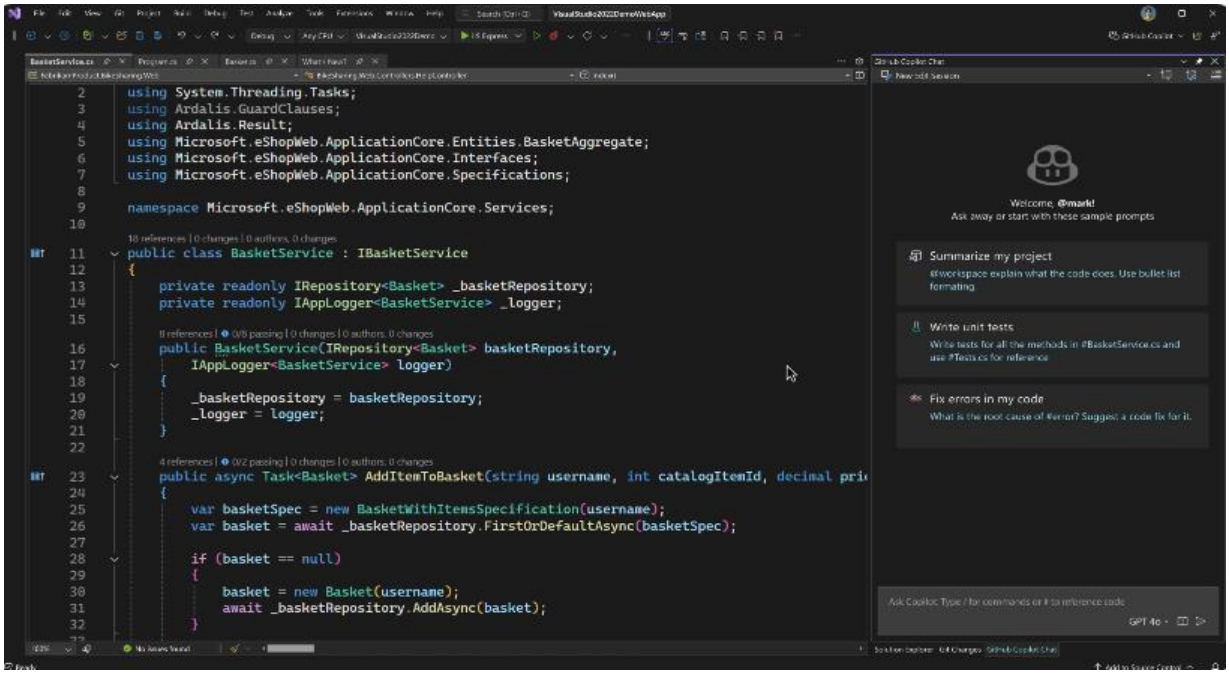

Рисунок 3.1 — Робоче вікно Visual Studio 2022

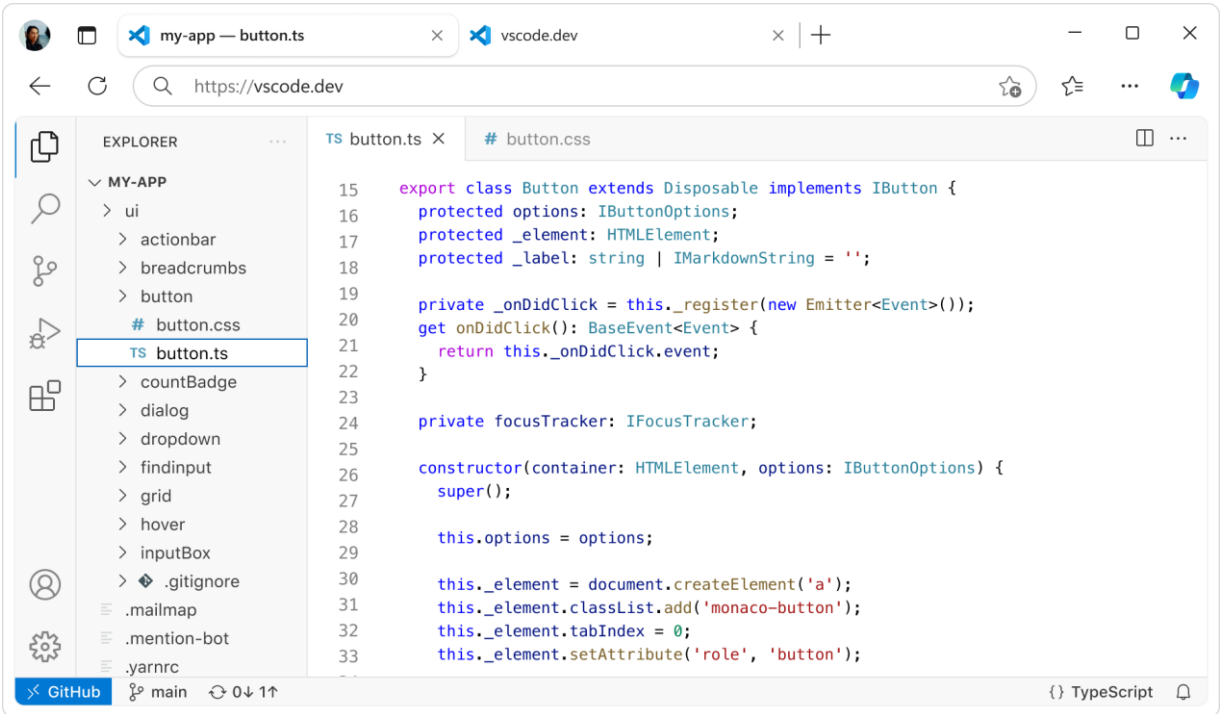

Рисунок 3.2 — Робоче вікно Visual Studio Code

В цій роботі будуть використовуватись такі його компоненти: ASP.NET Core MVC, Entity Framework Core та ASP.NET Core Identity. Як вже було згадано в попередньому розділі, структура WEB-платформи базується на моделі MVC. З реалізацією цієї структури допоможе ASP.NET Core MVC. Технологія доступу до даних Entity Framework Core використовується для роботи з базами даних за допомогою об’єктів .NET. ASP.NET Core Identity є системою автентифікації та авторизації. Вона використовуватись для надання таких функцій як реєстрація користувача, вхід, вихід, відновлення пароля, блокування облікового запису, перевірка облікового запису за допомогою SMS і електронної пошти, автентифікація на основі ролей, автентифікація на основі вимог, автентифікація третьої сторони, зовнішня автентифікація, двофакторна автентифікація тощо.

3.2 Реалізація оригінальних програмних рішень

Головна задача платформи полягає у додаванні до неї коміксів та можливість їх перегляду. Першим процесом, який потрібно реалізувати, є функція додавання нової серії (рис. 3.3). Функція отримує від форми значення введеної назви серії, обраного жанру, мови, вікового обмеження, тип макету панелей коміксів (традиційна, вертикальна), напрям читання і т. д. За допомогою цих даних створюється новий запис серії та папку для його розділів.

Після створення серії користувачу буде надана можливість змінити видимість роботи, опис, теги, обкладинку та жанр. Функція зміни жанрів дозволить додавати та віднімати жанри серій (рис. 3.4).

Користувачу надається можливість створення нового випуску ввівши його назву та завантаживши сторінки (рис. 3.5). Кожна сторінка має посилання на наступну сторінку, остання сторінка має значення null. Отже, зображення додаються у базу даних в порядку, оберненому до того, який був відображений у формі. Окрема змінна зберігає значення наступної сторінки. Після того як сторінку було додано до бази даних, значення окремої змінної набуває значення

збереженої сторінки, на яку буде посилатись наступна сторінка у списку на збереження.

Рисунок 3.3 – Алгоритм процесу додавання нової серії

Після створення нового випуску, користувач зможе редагувати його: додавати нові сторінки, видаляти старі, міняти їх місцями. Новий порядок сторінок обробляється наступним чином:

1. Видалення сторінок із бази даних, які більше не присутні у новому порядку.

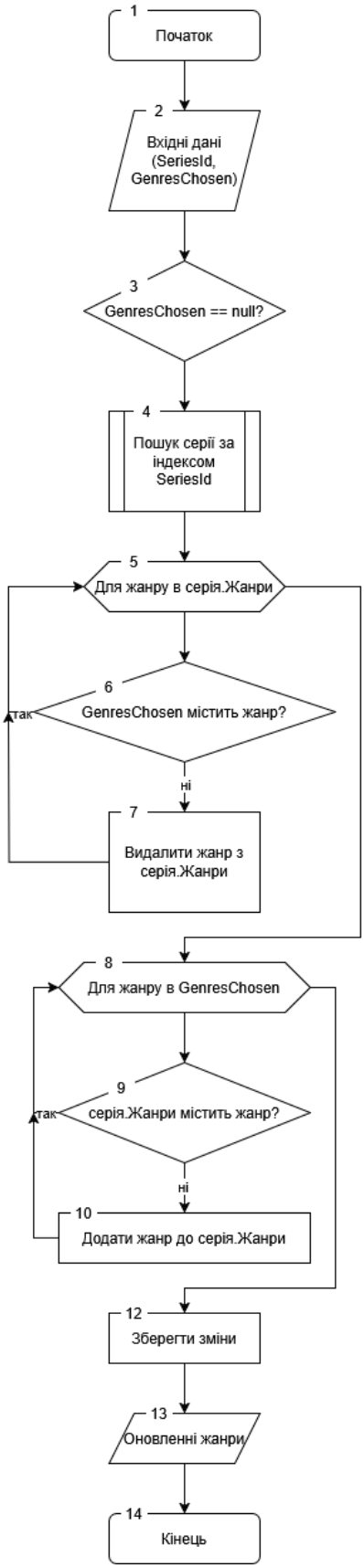

Рисунок 3.4 – Алгоритм процесу зміни жанрів серії

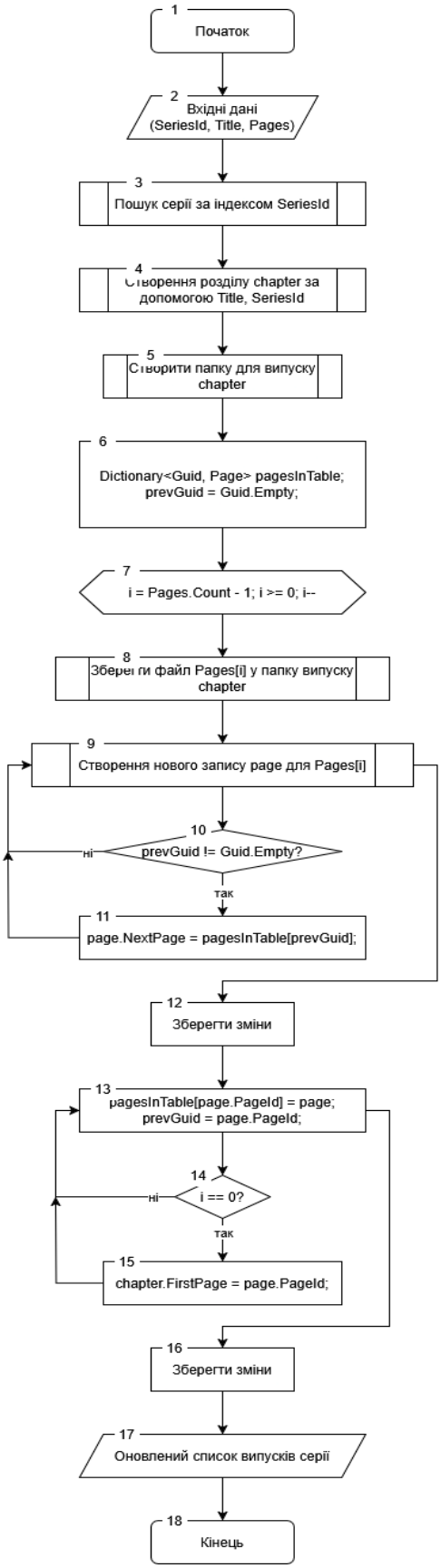

Рисунок 3.5 – Алгоритм процесу додавання нового випуску

2. Додавання нових сторінок до бази даних без визначення їх наступних сусідів.
3. Зміна значень наступної сторінки для кожної сторінки, відповідно новому порядку.

Відображення сторінок коміксу на вебсторінці починається з пошуку першої сторінки. Індекс першої сторінки кожного випуску зберігається у таблиці випусків. Далі за посиланням на наступну сторінку відображається ця наступна сторінка. Коли значення наступної сторінки останньої відображеної сторінки дорівнює null, цикл проходження по списку сторінок випуску зупиняється.

Користувачу також надається можливість видалення останнього випуску та всієї серії. Видалення випуску призводить до видалення сторінок також, як метаданих із бази даних, так і файлів із файлового провайдера. При видаленні серії також видаляються дані випусків та їх папки.

3.3 Тестування розробленого продукту

Для реєстрації користувача потрібно ввести ім’я користувача, електронну пошту, пароль та підтвердити пароль (рис. 3.6). Після успішної реєстрації, користувач повертається до головного вікна програми (рис. 3.7). Якщо він зареєстрований, то він може увійти за допомогою бокової панелі «Log In» зліва. Для входу потрібно лише ввести електронну пошту і пароль.

Для переходу на сторінку профілю (рис. 3.8) потрібно натиснути на кнопку «My account» на панелі зліва. Щоб змінити фото профілю, потрібно натиснути на кнопку над кнопкою «Change avatar», обрати фото та натиснути на кнопку «Change avatar». Після цього зображення буде змінено (рис. 3.9).

У панелі «Change password» можна змінити власний пароль. Для цього потрібно ввести теперішній пароль, новий пароль і підтвердити новий пароль. Після натискання на кнопку «Change Password», пароль буде змінено.

Рисунок 3.6 – Сторінка реєстрації користувача

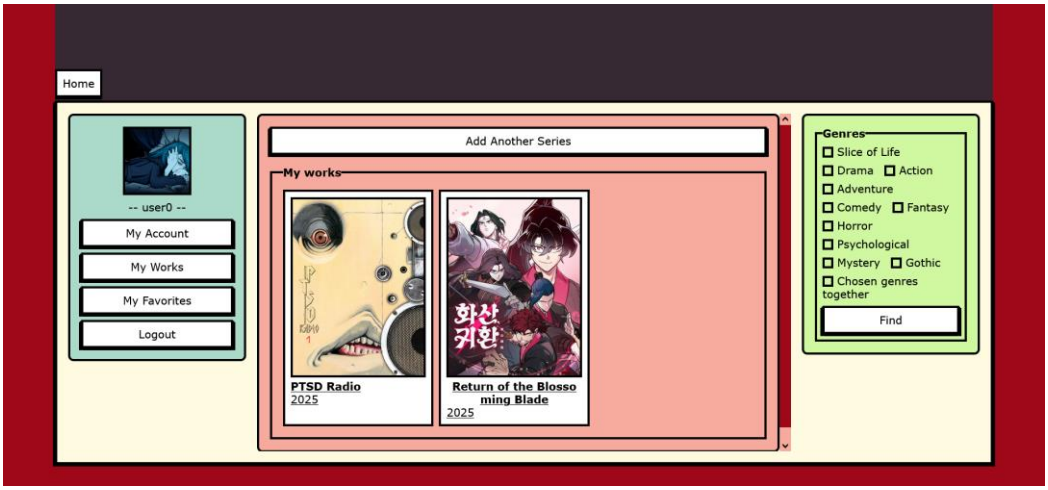

Рисунок 3.7 – Головна сторінка

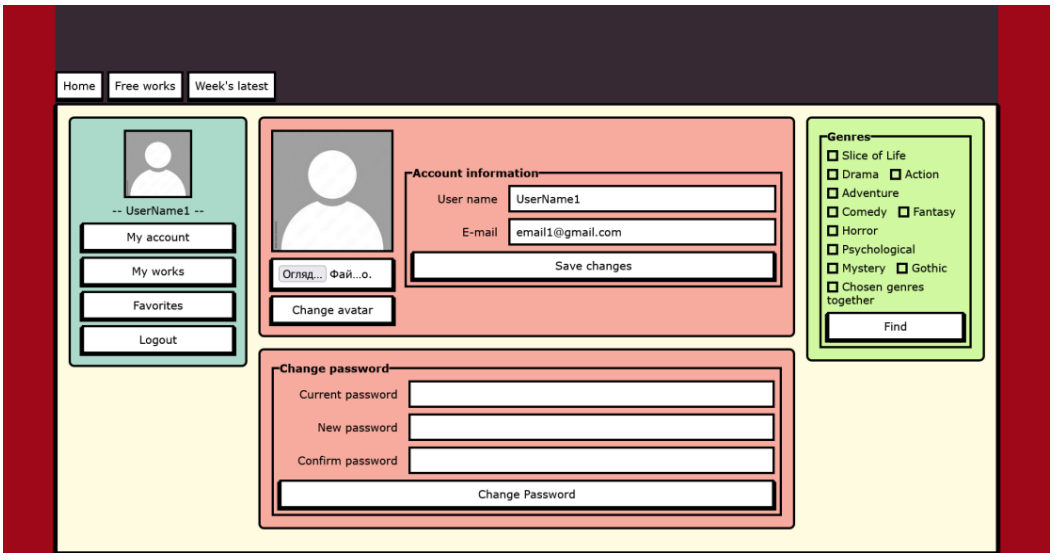

Рисунок 3.8 – Профіль користувача

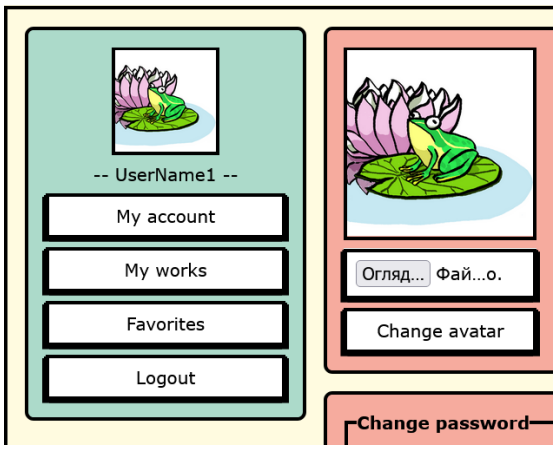

Рисунок 3.9 – Зміна фото профілю

Після натискання на кнопку «My works», користувач переходить на сторінку власних робіт. Щоб додати нову роботу, потрібно натиснути на кнопку «Add Another Series». Користувача буде переведено до форми створення серії (рис. 3.10). Потрібно обрати назву, мову, жанр, вікове обмеження, тип макету панелей, напрям читання і т. д. В цьому прикладі буде продемонстровано публікацію коміксу «The Students of Illip Arts High» (кор. «일립예고 학생들»). Після натискання на кнопку «Create the Series», знову відображається сторінка власних робіт користувача, але на цей раз із новою серією (рис. 3.11).

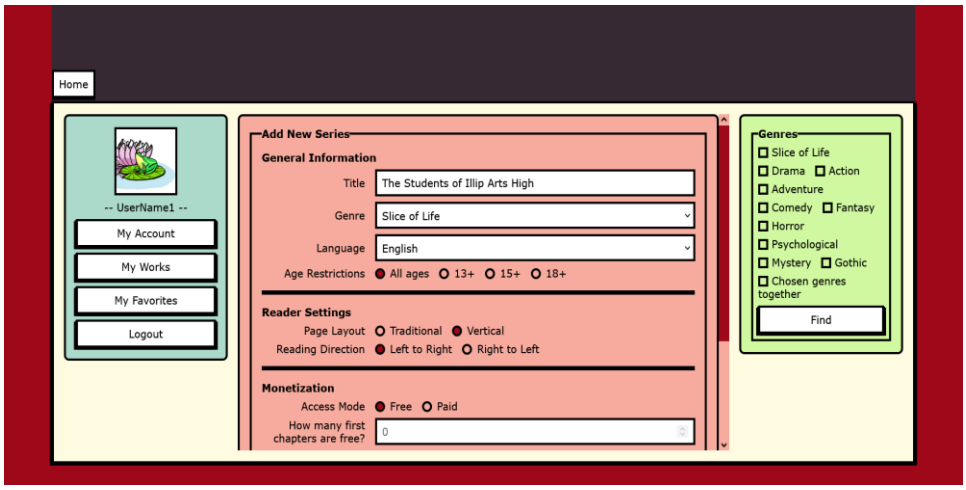

Рисунок 3.10 – Форма створення нової серії

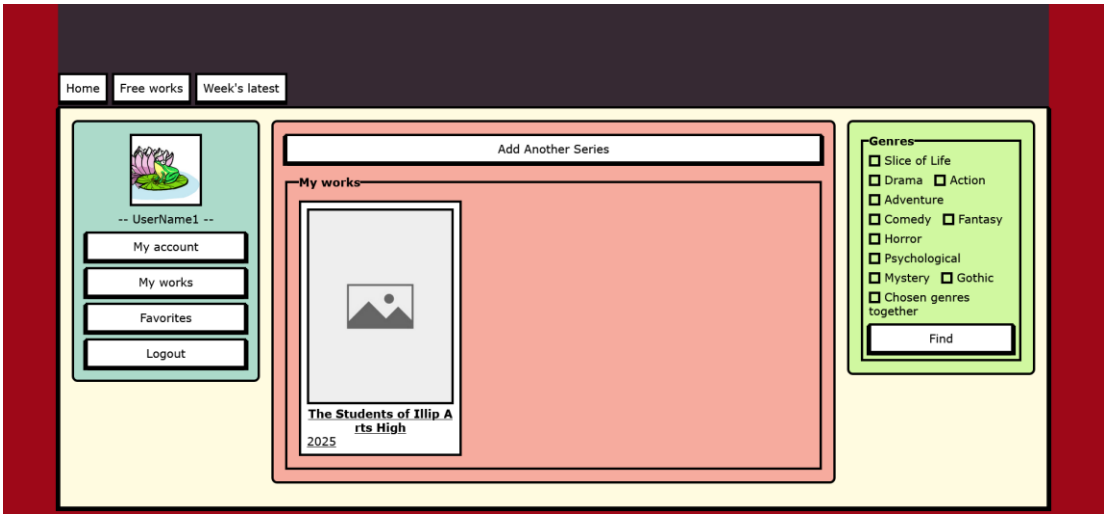

Рисунок 3.11 – Сторінка власних робіт користувача

Якщо користувач натисне на власну роботу, його буде перенесено на сторінку редагування серії (рис. 3.12). Подібно до зміни фото профілю, можна так само змінити обкладинку серії. Якщо обрати інший набір жанрів та натиснути на кнопку «Save genres», буде відображено новий набір жанрів (рис. 3.13).

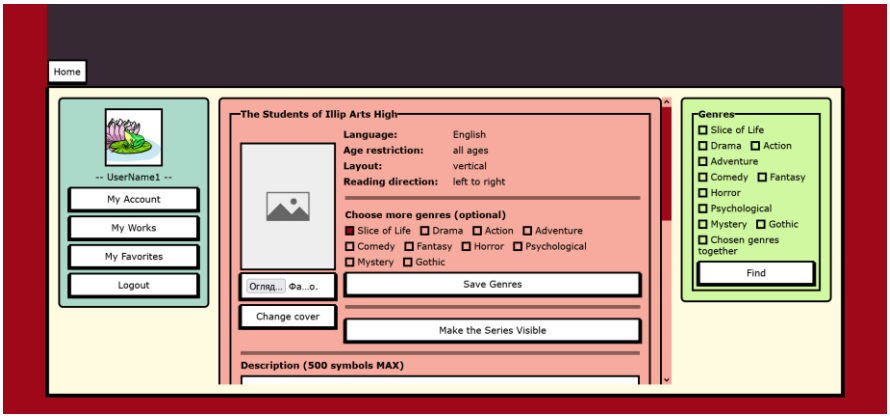

Рисунок 3.12 – Сторінка редагування серії

Якщо натиснути на кнопку «Make the Series visible», серія буде видима для всіх користувачів. Далі користувач не зможе видалити серію. Якщо ввести

опис серії, теги та натиснути на кнопку «Save the Description and Tags», от їх буде збережено.

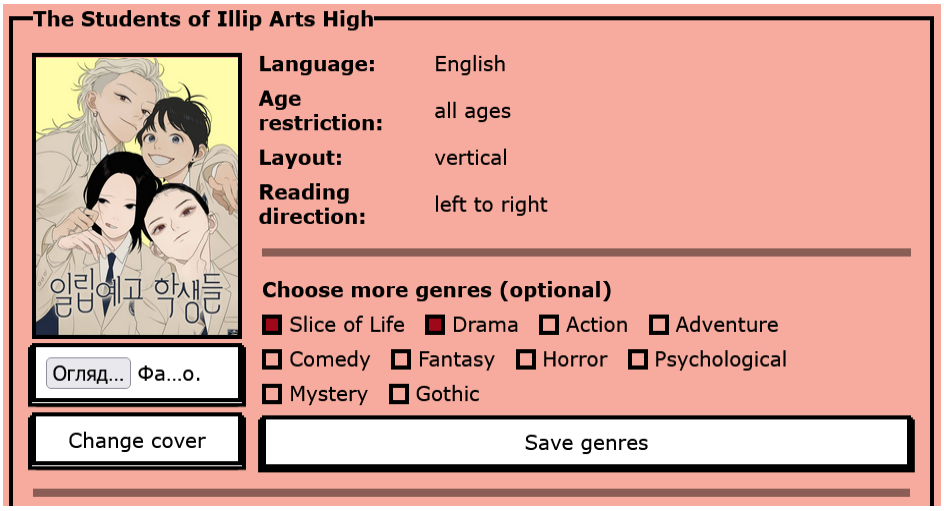

Рисунок 3.13 – Зміна обкладинки та жанрів

Після натискання на кнопку «Add a new Chapter», користувача буде направлено на сторінку створення випуску (рис. 3.14). Потрібно ввести назву випуску та додати сторінки. За допомогою кнопок «Up» і «Down» можна переміщати сторінки вверх-вниз. Після натискання на кнопку «Save the Chapter», користувача буде переведено до сторінки редагування серії. Далі випуск можна редагувати (додавати панелі, видаляти панелі, міняти їх місцями) та переглядати.

Рисунок 3.14 – Створення нового випуску

Для читача, головна сторінка серії виглядає як на рисунку 3.15. В залежності від формату панелей, сторінки можуть бути представленні або у горизонтальному рядку (рис. 3.16), або у вертикальному рядку (рис. 3.17).

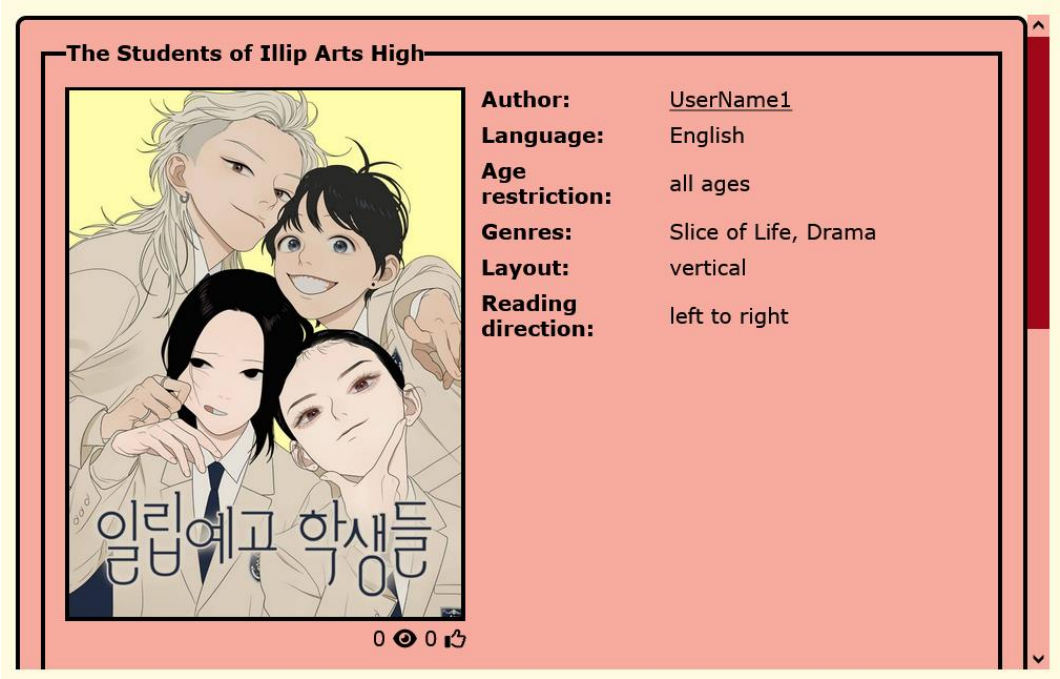

Рисунок 3.15 – Сторінка читача

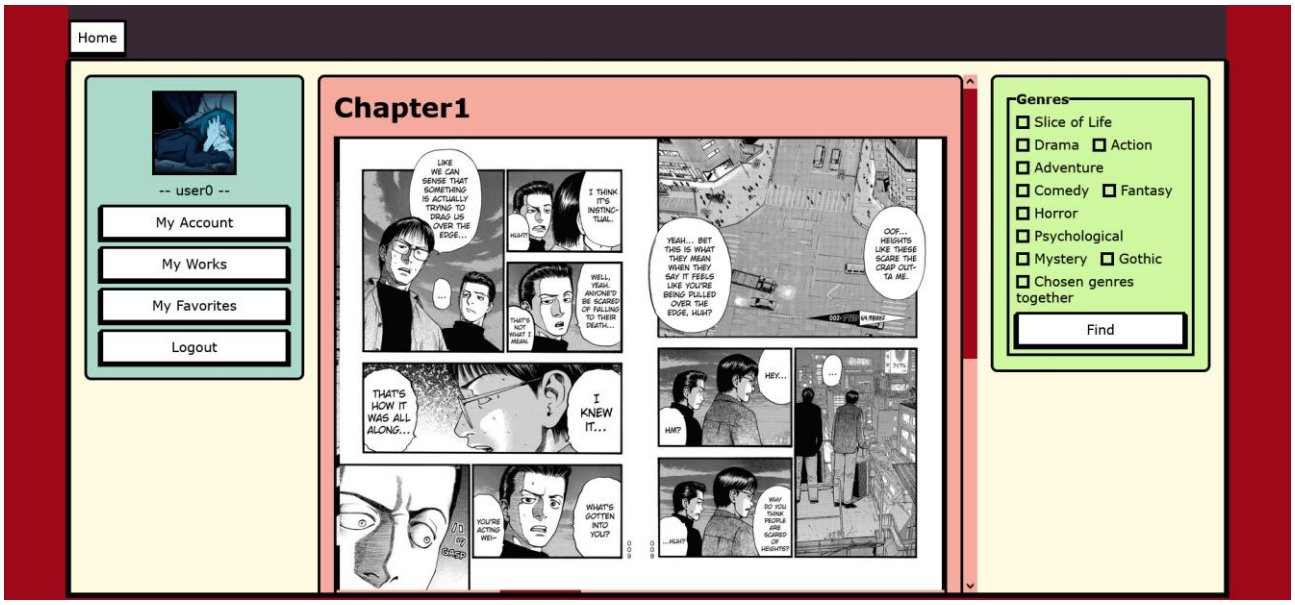

Рисунок 3.16 – Відображення випуску з традиційним форматом панелей

Рисунок 3.17 – Відображення випуску з вертикальним форматом панелей

Під кожним розділом можна додати коментарі. Для цього потрібно ввести його у виділену форму та натиснути на кнопку «Post Comment» (рис. 3.18). Після цього коментар буде опубліковано під цією формою.

Рисунок 3.18 – Додавання коментарів

Користувачі також можуть закладати улюблені роботи, натиснувши на кнопку «Вподобайка». Після вибору читачем розділу для читання, кількість переглядів серії збільшується на одиницю. Такий механізм показує якість історії не лише за рахунок того як багато користувачів читають серію, а й як часто читачі повертаються до своїх улюблених серій, наскільки невеликою є кількість фанатів цих серій в порівнянні з іншими.

Для адміністраторів інтерфейс виглядає дещо інакше. На рисунку 3.19 зображено бокове меню для адміністраторів. Адміністратори можуть переглядати список користувачів, інших адміністраторів та серій.

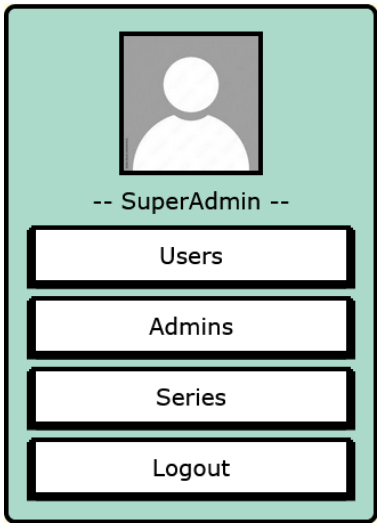

Рисунок 3.19 – Бокове меню адміністратора

Сторінка списку користувачів (рис. 3.20) дозволяє адміністратору переглядати список користувачів та шукати потрібних користувачів за допомогою або ідентифікатора, або імені, або електронної пошти. Для кожного користувача призначено три кнопки: «View Profile», «Set as Admin», «Delete». Кнопка «View Profile» переносить адміністратора на сторінку профілю користувача, де відображено ідентифікатор, ім’я, електронну пошту, фото профілю та список робіт користувача (рис. 3.21). На сторінці профілю користувача також є кнопки «Set as Admin» та «Delete». Кнопка «Set as Admin» встановлює користувача як нового адміністратора, кнопка «Delete» – видаляє користувача, зберігаючи його роботи.

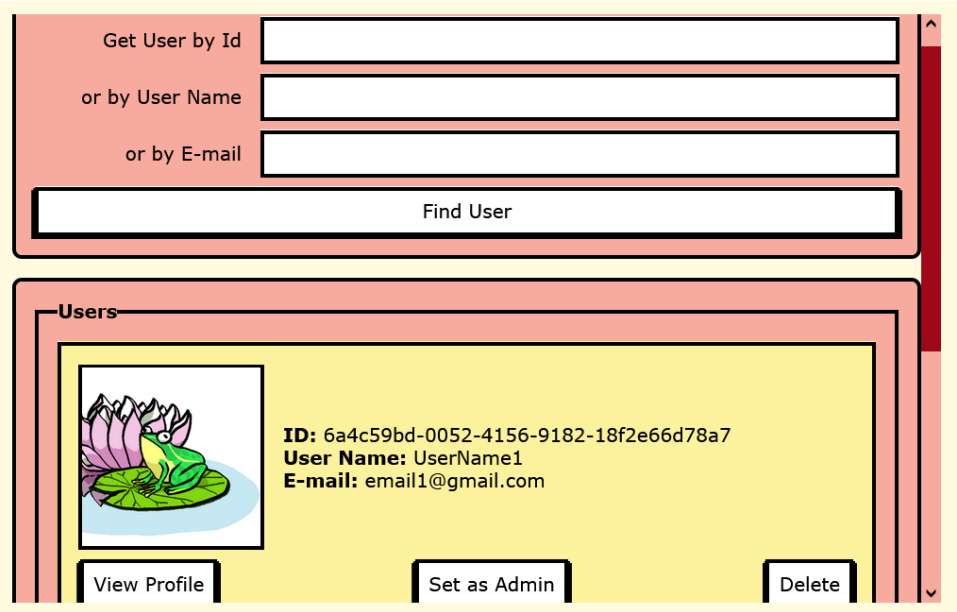

Рисунок 3.20 – Сторінка списку користувачів

Рисунок 3.21 – Сторінка профілю користувача

Якщо натиснути на кнопку «Set as Admin», користувача буде вилучено із списку користувачів та перенесено у список адміністраторів (рис. 3.22). Сторінка списку адміністраторів також дозволяє шукати адміністраторів за допомогою або ідентифікатора, або імені, або електронної пошти.

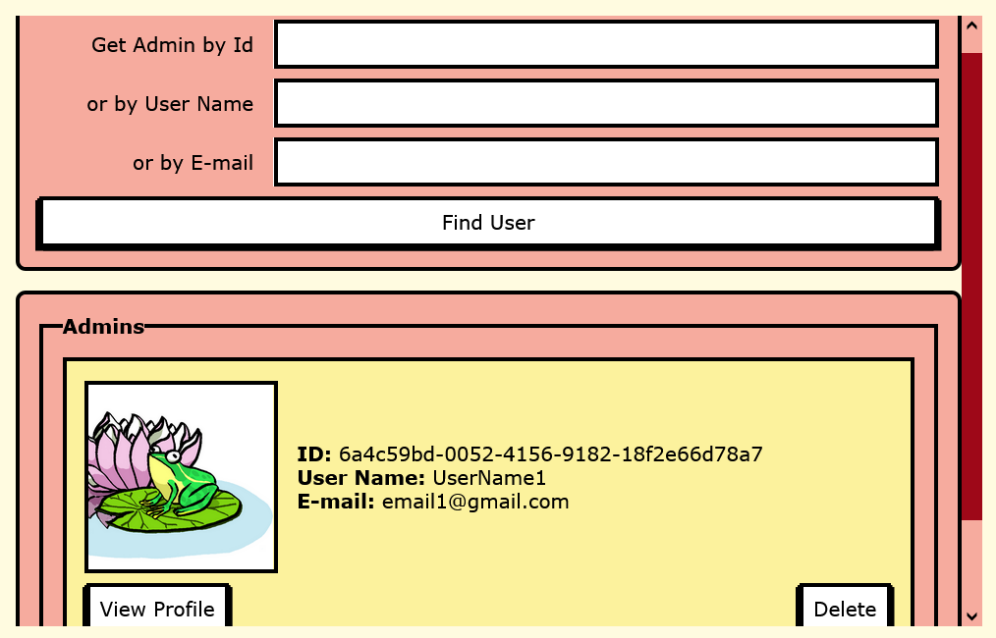

Рисунок 3.22 – Сторінка списку адміністраторів

Сторінка списку робіт (рис. 3.23) дозволяє шукати роботи за допомогою або ідентифікатора, або назви, або року випуску. Для кожної роботи призначено дві кнопки: «Edit», «Delete». При натисканні на кнопку «Edit» адміністратора переносить на сторінку відповідної серії (рис. 3.24), де він може побачити загальну інформацію про серію, список випусків, опубліковані та збереженні в чернетках, які можна прочитати, та кнопку «Delete the Series», яка видаляє серію.

Таким чином, розроблена платформа для публікації коміксів вирізняється від своїх аналогів тим, що поєднує окремі переваги кожної з них.

По-перше, платформа дозволяє обирати між класичним книжковим форматом та вертикальним форматом поєднання сторінок, між напрямком читання зліва направо та справа наліво. Це дозволяє авторам не обмежувати себе одним шаблоном та публікувати роботи, які попередньо були у продажі у книжковому форматі.

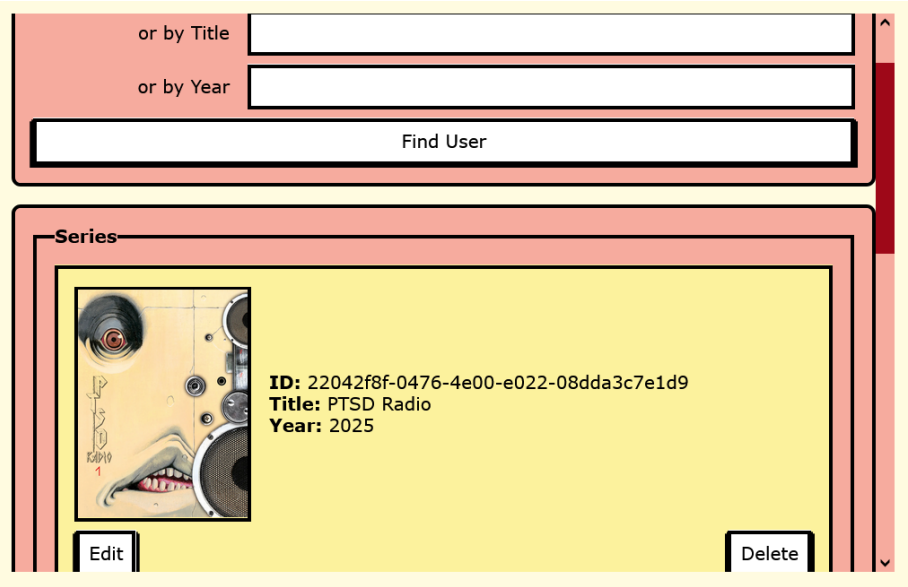

Рисунок 3.23 – Сторінка списку робіт

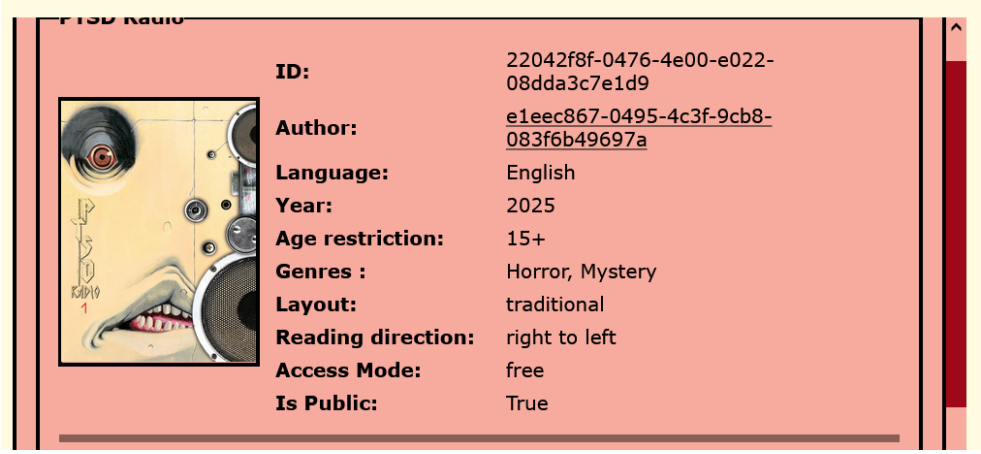

Рисунок 3.24 – Сторінка серії

По-друге, платформа підтримує анімовані зображення, зокрема у форматі .gif. Це є значною перевагою перед платформами WEBTOON, яка обмежує анімації на сторінках для художників, які працюють для неї, та GlobalComix, яка зовсім не підтримує анімації.

По-третє, дизайн користувацького інтерфейсу налаштований для тих, хто віддають більшу перевагу функціоналу ресурсу, ніж естетиці. Ресурси-аналоги дозволяють переглядати випуски лише у повноекранному режимі, приховуючи всі засоби навігації до кінця випусків. Розроблена WEB-платформа пропонує

зручність для користувачів, які вважають такий вид сторінок незручним та люблять, коли «все є під рукою».

3.4 Висновки

У цьому розділі було детально розглянуто основні моменти програмної реалізації WEB-платформи для публікації коміксів.

Було поріняно такі мови програмування: Python, C#, Java. З огляду на переваги кожної із них та потреби проекту, було обрано C#. Як середовище розробки було використано Visual Studio. Серед бібліотек для веброзробки було обрано засоби ASP.NET Core.

Було описано алгоритми основних функцій програми: створення серій, створення випусків, редагування серій, редагування випусків.

Під час тестування здійснювалася перевірка коректності обробки даних, введених користувачем, а також виявлення та усунення можливих помилок чи некоректних дій.

ВИСНОВКИ

Бакалаврська кваліфікаційна робота присвячена створенню Інтернет-ресурсу для публікації коміксів. Всі задачі поставлені перед БКР виконані в повному об’ємі, а саме:

- проаналізовано доцільність та відомі рішення створення ресурсів для публікації коміксів;
- спроектовано онлайн платформу для публікації коміксів;
- реалізовано клієнтську та серверну частини онлайн платформи для публікації коміксів;
- отриманий продукт протестовано та отримані результати проаналізовано.

Аналіз трьох із найпопулярніших Інтернет-ресурсів для публікації коміксів, WEBTOON, Tapas та GlobalComix, показав, що на позитивний досвід користувачів впливає як доступні їм послуги, так і політика використання цих ресурсів. Читачів приваблює часта реклама платформ, а широкий вибір безкоштовних робіт, які зручно читати на будь-якому екрані змушує залишитись. Ті, хто хоче оприлюднити власні роботи, користуються тими платформами, які дозволяють їм отримувати прибуток з них: частки з рекламних інтеграцій, донат чи плата за випуски.

На основі результатів аналізу, було спроектовано структуру WEB-платформи для публікації коміксів: було обґрунтовано вибір архітектури; розроблено ER-модель бази даних, UML-діаграм алгоритмів основних функцій платформи.

Було проаналізовано три найпопулярніші мови для веброзробки: Python, C#, Java. Для потреб проекту було обрано C#. Між Visual Studio та Visual Studio Core перевагу було віддано Visual Studio як середовищу розробки, яке краще адаптоване під великі проекти. Серед фрейморків було обрано ASP.NET Core та його компоненти для реалізації клієнт-серверної структури програми.

Було описано алгоритми основних функцій платформи для роботи із публікаціями. Тестування фінального продукту показало, що поставлені задачі було виконано.

Мета БКР, яка полягала у розширенні функціоналу WEB-платформи для публікації коміксів була досягнута за рахунок того, що у порівнянні з аналогами, які орієнтуються на повноекранне читання, розроблена платформа віддає перевагу функціоналу, не приховує елементи інтерфейсу та дозволяє повну свободу в форматах оповідання, підтримуючи як анімовані сторінки (GIF), так і різні напрямки читання [27].

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Сичова М. С., Сілагін О. В. Аналіз підходів до розробки вебплатформи для публікації коміксів. LIV Всеукраїнська науково-технічна конференція факультету інтелектуальних інформаційних технологій та автоматизації (2025), Вінниця, 2025. [Електронний ресурс]. – Режим доступу: https://conferences.vntu.edu.ua/index.php/all-fksa/all-fksa-2025/paper/view/24044

2. The History of Webcomics [Електронний ресурс] – Режим доступу: https://www.tcj.com/the-history-of-webcomics/

3. The History and Future of Webcomics [Електронний ресурс] – https://cryptocomics.com/blog/C/2023/05/17/The-History-and-Future-of-Webcomics-1584

4. A brief history of webtoons [Електронний ресурс] – https://www.vam.ac.uk/articles/a-brief-history-of-webtoons?srsltid=AfmBOoqUre0CABSyat8W1-SNEz2SA-gZuU69cjflLWF6HHN4Zb8RxWZF

5.WEBTOON: Our Brands [Електронний ресурс] – https://about.webtoon.com/our-brands

6. Twitter/X [Електронний ресурс]. – https://x.com/

7. Tumblr [Електронний ресурс] – https://www.tumblr.com/

8. Instagram [Електронний ресурс] – https://www.instagram.com/

9. Pixiv [Електронний ресурс] – https://www.pixiv.net/en/

10. Poipiku [Електронний ресурс] – https://poipiku.com/

11. WEBTOON [Електронний ресурс] – https://www.webtoons.com/en/

12. Tapas [Електронний ресурс] – https://tapas.io/

13. GlobalComix [Електронний ресурс] – https://globalcomix.com/

14. What are Microservices? [Електронний ресурс] – https://aws.amazon.com/microservices/

15. Client-server architecture [Електронний ресурс] – https://en.training.qatestlab.com/blog/technical-articles/client-server-architecture/

16. What Is a CDN (Content Delivery Network)? [Електронний ресурс] – https://www.akamai.com/glossary/what-is-a-cdn

17. Building Applications with Serverless Architectures [Електронний ресурс] – https://aws.amazon.com/lambda/serverless-architectures-learn-more/

18. Types of Databases [Електронний ресурс] – https://www.geeksforgeeks.org/types-of-databases/

19. Introduction of ER Model [Електронний ресурс] – https://www.geeksforgeeks.org/introduction-of-er-model/

20. Unified Modeling Language (UML) Diagrams Introduction of ER Model [Електронний ресурс] – https://www.geeksforgeeks.org/unified-modeling-language-uml-introduction/

21. Python Features [Електронний ресурс] – https://www.geeksforgeeks.org/python-features/

22. Top 10 Features of C#.NET Every Beginner Should Know [Електронний ресурс] – https://dev.to/vandana_babshetti_91df8eb/top-10-features-of-cnet-every-beginner-should-know-1i21

23. Java Features [Електронний ресурс] – https://www.geeksforgeeks.org/java-features/

24. Visual Studio 2022 [Електронний ресурс] – https://visualstudio.microsoft.com/

25. Visual Studio Core [Електронний ресурс] – https://code.visualstudio.com/

26. ASP.NET Core Tutorials For Beginners and Professionals [Електронний ресурс] – https://dotnettutorials.net/course/asp-net-core-tutorials/

27. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 122 «Комп’ютерні науки» [Електронний ресурс] / уклад.: А. А. Яровий, О. В. Сілагін, І. В. Богач. – Вінниця : ВНТУ, 2023. – (PDF, 54 с.).

ДОДАТКИ

ДОДАТОК А (обов'язковий)

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Розробка WEB-платформи для публікації коміксів

Тип роботи: бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра комп'ютерних наук
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 4,97 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

✓ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту.

□ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

□ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

Яровий А.А., зав. каф. КН _____
(прізвище, ініціали, посада) (підпис)

Колесницький О.К., проф. каф КН _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Озеранський В.С.
(підпис) (прізвище, ініціали)

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Сілагін О.В.
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Сичова М.С.
(підпис) (прізвище, ініціали)

ДОДАТОК Б (обов'язковий)

ЛІСТИНГ ПРОГРАМИ

AccountController.cs

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;

namespace ComiBerry.Controllers
{
   public class AccountController(SignInManager<User> signInManager, UserManager<User>
userManager, EncryptionService encryptionService) : Controller
   {
      private readonly SignInManager<User> _signInManager = signInManager;
      private readonly UserManager<User> _userManager = userManager;
      private readonly EncryptionService _encryptionService = encryptionService;

      [HttpPost]
      public async Task<IActionResult> Login(LoginPartialViewModel model)
      {
         if (ModelState.IsValid)
         {
            var user = await _userManager.FindByEmailAsync(model.Email!);
            if (user is not null)
            {
               var singInResult = await _signInManager.PasswordSignInAsync(user.UserName,
model.Password!, false, false);
               if (singInResult.Succeeded)
               {
                  Response.Cookies.Append("UserId", _encryptionService.EncryptData(user.Id));
                  Response.Cookies.Append("UserName",
_encryptionService.EncryptData(user.UserName));
                  Response.Cookies.Append("UserEmail",
_encryptionService.EncryptData(user.Email));
                  ViewData["PageName"] = "Explore";
                  return RedirectToAction("Home", "Navigation");
               }
            }
         }
         return RedirectToAction("Register", "Account");
      }

      [HttpGet]
      public ActionResult Register()
      {
         return View();
      }

      [HttpPost]
```

```
        public async Task<IActionResult> Register(RegisterViewModel model)
        {
            if (ModelState.IsValid)
            {
                var user = await _userManager.FindByEmailAsync(model.Email!);
                if (user is null)
                {
                    User newUser = new()
                    {
                        UserName = model.Name!,
                        Email = model.Email!
                    };

                    var registerResult = await _userManager.CreateAsync(newUser, model.Password!);

                    if (registerResult.Succeeded)
                    {
                        await _userManager.AddToRoleAsync(newUser, "user");
                        var singInResult = await
_signInManager.PasswordSignInAsync(newUser.UserName, model.Password!, false, false);
                        if (singInResult.Succeeded)
                        {
                            Response.Cookies.Append("UserId",
_encryptionService.EncryptData(newUser.Id));
                            Response.Cookies.Append("UserName",
_encryptionService.EncryptData(newUser.UserName));
                            Response.Cookies.Append("UserEmail",
_encryptionService.EncryptData(newUser.Email));
                            ViewData["PageName"] = "Explore";
                            return RedirectToAction("Home", "Navigation");
                        }
                    }
                }
            }
            return View(model);
        }

        [Authorize]
        public async Task<IActionResult> Logout()
        {
            Response.Cookies.Delete("UserId");
            Response.Cookies.Delete("UserName");
            Response.Cookies.Delete("UserEmail");
            await _signInManager.SignOutAsync();
            ViewData["PageName"] = "Explore";
            return RedirectToAction("Home", "Navigation");
        }
    }
}
```

UserMenuController.cs

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;

namespace ComiBerry.Controllers
{
   [Authorize]
   public class UserMenuController(IConfiguration configuration, SignInManager<User>
signInManager, UserManager<User> userManager, EncryptionService encryptionService,
ApplicationDbContext context) : Controller
   {
      public readonly IConfiguration _configuration = configuration;
      private readonly SignInManager<User> _signInManager = signInManager;
      private readonly UserManager<User> _userManager = userManager;
      private readonly EncryptionService _encryptionService = encryptionService;
      private readonly ApplicationDbContext _context = context;

      // MY ACCOUNT
      [HttpGet]
      public ActionResult MyAccount()
      {
         return View();
      }

      [HttpPost]
      public async Task<IActionResult> ChangeAvatar(IFormFile newAvatar)
      {
         if (ModelState.IsValid)
         {
            var permittedExtensions = new[] { ".jpg", ".png", ".gif" };
            var extension = Path.GetExtension(newAvatar.FileName).ToLowerInvariant();
            if (string.IsNullOrEmpty(extension) || !permittedExtensions.Contains(extension)) { return
RedirectToAction("MyAccount", "UserMenu"); }

            var mimeType = newAvatar.ContentType;
            var permittedMimeTypes = new[] { "image/jpeg", "image/png", "image/gif" };
            if (!permittedMimeTypes.Contains(mimeType)) { return RedirectToAction("MyAccount",
"UserMenu"); }

            var user = HttpContext.Items["CurrentUser"] as User;
            var fileName = user!.Id + extension;
            var filePath = Path.Combine(_configuration.GetConnectionString("Avatars")!, fileName);
            using (var stream = new FileStream(filePath, FileMode.Create, FileAccess.Write))
            {
               await newAvatar.CopyToAsync(stream);
            }
            user.AvatarLink = filePath ;
            await _userManager.UpdateAsync(user);
         }
```

```
            return RedirectToAction("MyAccount", "UserMenu");
        }

        [Authorize]
        [HttpPost]
        public async Task<IActionResult> SaveChanges(MyAccountViewModel model) {
            if (ModelState.IsValid)
            {
                var user = HttpContext.Items["CurrentUser"] as User;
                user!.UserName = model.Name;
                user.Email = model.Email;
                var result = await _userManager.UpdateAsync(user);
                if (result.Succeeded)
                {
                    Response.Cookies.Append("UserName",
_encryptionService.EncryptData(model.Name));
                    Response.Cookies.Append("UserEmail",
_encryptionService.EncryptData(model.Email));
                }
            }
            return RedirectToAction("MyAccount", "UserMenu");
        }

        [HttpPost]
        public async Task<IActionResult> ChangePassword(MyAccountPasswordViewModel model)
        {
            if (ModelState.IsValid)
            {
                var user = HttpContext.Items["CurrentUser"] as User;
                var changeResult = await _userManager.ChangePasswordAsync(user!,
model.CurrentPassword!, model.NewPassword!);
                if (changeResult.Succeeded)
                {
                    Response.Cookies.Delete("UserId");
                    Response.Cookies.Delete("UserName");
                    Response.Cookies.Delete("UserEmail");
                    await _signInManager.SignOutAsync();
                    var singInResult = await _signInManager.PasswordSignInAsync(user!.UserName,
model.NewPassword!, false, false);
                    if (singInResult.Succeeded)
                    {
                        Response.Cookies.Append("UserId", _encryptionService.EncryptData(user.Id));
                        Response.Cookies.Append("UserName",
_encryptionService.EncryptData(user.UserName));
                        Response.Cookies.Append("UserEmail",
_encryptionService.EncryptData(user.Email));
                    }
                }
            }
            return RedirectToAction("MyAccount", "UserMenu");
```

```
        }

        [HttpPost]
        public async Task<IActionResult> ChangeBio(MyAccountBioViewModel model)
        {
            if (ModelState.IsValid)
            {
                var user = HttpContext.Items["CurrentUser"] as User;
                user!.Bio = model.Bio;
                await _userManager.UpdateAsync(user);
            }
            return RedirectToAction("MyAccount", "UserMenu");
        }

        //MY WORKS
        [HttpGet]
        public ActionResult MyWorks()
        {
            var user = HttpContext.Items["CurrentUser"] as User;
            var worksModel = new WorksViewModel
            {
                Series = [.. _context.Series.Where(s => s.User == user)]
            };
            return View(worksModel);
        }

        [HttpGet]
        public IActionResult MyFaves()
        {
            var user = HttpContext.Items["CurrentUser"] as User;
            var faves = _context.Faves.Where(f => f.User == user).Include(f => f.Series).Select(f =>
f.Series).ToList();
            var favesModel = new WorksViewModel
            {
                Series = faves
            };
            ViewData["PageName"] = "My Favorites";
            return View("~/Views/Navigation/Home.cshtml", favesModel);
        }
    }
}
```

AuthorController.cs

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;

namespace ComiBerry.Controllers
{
```

```
    [Authorize]
    public class AuthorController(IConfiguration configuration, ApplicationDbContext context) :
Controller
    {
        private readonly IConfiguration _configuration = configuration;
        private readonly ApplicationDbContext _context = context;

        [HttpGet]
        public IActionResult CreateSeries()
        {
            var model = new AUTHOR_CreateSeriesViewModel
            {
                Genres = [.. _context.Genres]
            };
            return View(model);
        }

        [HttpPost]
        public async Task<ActionResult> CreateSeries(AUTHOR_CreateSeriesViewModel model)
        {
            if (ModelState.IsValid)
            {
                var user = HttpContext.Items["CurrentUser"] as User;

                var series = new Series
                {
                    Title = model.Title!,
                    Genre = [_context.Genres.First(g => g.GenreId == model.Genre!)],
                    Language = model.Language!,
                    Year = DateTime.Now.Year,
                    AgeRestriction = model.AgeRestriction!,
                    Layout = model.Layout!,
                    Direction = model.Direction!,
                    AccessMode = model.AccessMode!,
                    FreeChapters = model.FreeChapters,
                    IsVisible = model.IsVisible,
                    User = user!,
                    FolderLink = ""
                };

                await _context.Series.AddAsync(series);

                string folderLink = Path.Combine(_configuration.GetConnectionString("Series")!,
series.SeriesId.ToString());
                Directory.CreateDirectory(folderLink);
                series.FolderLink = folderLink;
                await _context.SaveChangesAsync();
            }

            return RedirectToAction("MyWorks", "UserMenu");
```

```
    }

    [HttpGet]
    public IActionResult EditSeries(Guid seriesId)
    {
        var series = _context.Series.Include(s => s.Genre).Include(s => s.Chapters).Single(x =>
x.SeriesId == seriesId);
        var genres = _context.Genres.ToList();
        var model = new AUTHOR_EditSeriesViewModel
        {
            Series = series,
            Genres = genres
        };
        return View(model);
    }

    [HttpPost]
    public async Task<IActionResult> ChangeCover(string seriesId, IFormFile newCover)
    {
        if (ModelState.IsValid)
        {
            var permittedExtensions = new[] { ".jpg", ".png", ".gif" };
            var extension = Path.GetExtension(newCover.FileName).ToLowerInvariant();
            if (string.IsNullOrEmpty(extension) || !permittedExtensions.Contains(extension)) { return
RedirectToAction("MyWorks", "UserMenu"); }

            var mimeType = newCover.ContentType;
            var permittedMimeTypes = new[] { "image/jpeg", "image/png", "image/gif" };
            if (!permittedMimeTypes.Contains(mimeType)) { return RedirectToAction("MyWorks",
"UserMenu"); }

            var series = _context.Series.Single(x => x.SeriesId == Guid.Parse(seriesId));
            if (series is not null)
            {
                var fileName = series.SeriesId + extension;
                var directory = _configuration.GetConnectionString("Covers");
                var filePath = Path.Combine(directory!, fileName);
                using (var stream = new FileStream(filePath, FileMode.Create, FileAccess.Write))
                {
                    await newCover.CopyToAsync(stream);
                }

                series.CoverLink = filePath;
                await _context.SaveChangesAsync();
            }
        }
        return RedirectToRoute(new { controller = "Author", action = "EditSeries", seriesId });
    }

    [HttpPost]
```

```
        public async Task<IActionResult> SaveGenres(string seriesId, List<string> GenresChosen)
        {
            if (GenresChosen is null)
            {
                return RedirectToRoute(new { controller = "Author", action = "EditSeries", seriesId });
            }

            var series = _context.Series.Include(s => s.Genre).Single(x => x.SeriesId ==
Guid.Parse(seriesId));
            foreach (var genre in series.Genre)
            {
                if (!GenresChosen.Contains(genre.GenreId.ToString()))
                {
                    series.Genre.Remove(genre);
                }
            }

            var allGenres = _context.Genres.ToList();
            foreach (var genreId in GenresChosen)
            {
                if (!series.Genre.Contains(allGenres.First(s => s.GenreId == Int32.Parse(genreId))))
                {
                    series.Genre.Add(allGenres.First(g => g.GenreId == Int32.Parse(genreId)));
                }
            }
            await _context.SaveChangesAsync();
            return RedirectToRoute(new { controller = "Author", action = "EditSeries", seriesId });
        }

        [HttpGet]
        public async Task<IActionResult> MakeVisible(Guid seriesId)
        {
            var series = _context.Series.Include(s => s.Chapters).Single(x => x.SeriesId == seriesId);
            if (!series.IsVisible)
            {
                series.IsVisible = true;
                if (series.Chapters is not null)
                {
                    foreach (var chapter in series.Chapters)
                    {
                        chapter.IsPublished = true;
                    }
                }
                await _context.SaveChangesAsync();
            }
            return RedirectToRoute(new { controller = "Author", action = "EditSeries", seriesId =
seriesId.ToString() });
        }

        [HttpPost]
```

```
        public async Task<IActionResult> SaveDescAndTags(string seriesId,
AUTHOR_EditSeriesViewModel model)
        {
            if (ModelState.IsValid)
            {
                var series = _context.Series.Single(x => x.SeriesId == Guid.Parse(seriesId));
                series.Description = model.Description;
                series.Tags = model.Tags;
                await _context.SaveChangesAsync();
            }
            return RedirectToRoute(new { controller = "Author", action = "EditSeries", seriesId });
        }

        [HttpGet]
        public async Task<IActionResult> DeleteSeries(Guid seriesId)
        {
            var series = _context.Series.Single(s => s.SeriesId == seriesId);
            if ((series is not null) && !series.IsVisible)
            {
                if ((series.FolderLink is not null) && Directory.Exists(series.FolderLink))
                {
                    Directory.Delete(series.FolderLink, recursive: true);
                }
                if (series.CoverLink is not null)
                {
                    System.IO.File.Delete(series.CoverLink);
                }
                _context.Series.Remove(series);
                await _context.SaveChangesAsync();
            }
            return RedirectToAction("MyWorks", "UserMenu");
        }

        [HttpGet]
        public IActionResult AddChapter(Guid seriesId)
        {
            if (ModelState.IsValid)
            {
                var addChapterModel = new AUTHOR_AddChapterViewModel
                {
                    SeriesId = seriesId
                };
                return View(addChapterModel);
            }
            return RedirectToRoute(new { controller = "Author", action = "EditSeries", seriesId });
        }

        [HttpPost]
        public async Task<IActionResult> AddChapter(AUTHOR_AddChapterViewModel model)
        {
```

```
        if (ModelState.IsValid)
        {
            var series = _context.Series.Include(s => s.Genre).Include(s => s.Chapters).Single(x =>
x.SeriesId == model.SeriesId);
            if (series is not null)
            {
                var chapter = new Chapter
                {
                    Title = model.Title!,
                    Date = DateTime.Now,
                    FolderLink = "",
                    Series = series
                };

                await _context.Chapters.AddAsync(chapter);
                string folderLink = Path.Combine(series.FolderLink, chapter.ChapterId.ToString());
                Directory.CreateDirectory(folderLink);
                chapter.FolderLink = folderLink;

                Dictionary<Guid, Page> pagesInTable = [];
                Guid prevGuid = Guid.Empty;

                for (int i = model.Pages!.Count - 1; i >= 0; i--)
                {
                    var filePath = Path.Combine(chapter.FolderLink,
model.Pages.ElementAt(i).FileName);
                    using (var stream = new FileStream(filePath, FileMode.Create, FileAccess.Write))
                    {
                        await model.Pages.ElementAt(i).CopyToAsync(stream);
                    }

                    var page = new Page
                    {
                        PageLink = filePath,
                        Chapter = chapter
                    };

                    if (prevGuid != Guid.Empty)
                    {
                        page.NextPage = pagesInTable[prevGuid];
                    }

                    await _context.Pages.AddAsync(page);
                    pagesInTable[page.PageId] = page;
                    prevGuid = page.PageId;

                    if (i == 0)
                    {
                        chapter.FirstPage = page.PageId;
                    }
```

```
                await _context.SaveChangesAsync();
            }
        }
    }
    return RedirectToRoute(new { controller = "Author", action = "EditSeries", seriesId =
model.SeriesId });
}

[HttpGet]
public IActionResult EditChapter(Guid seriesId, Guid chapterId)
{
    var chapter = _context.Chapters.Include(c => c.Pages).Single(x => x.ChapterId ==
chapterId);
    var pages = _context.Pages.Include(p => p.NextPage).Where(p => p.Chapter ==
chapter).ToList();

    var model = new AUTHOR_EditChapterViewModel
    {
        SeriesId = seriesId,
        ChapterId = chapterId,
        Title = chapter.Title,
        Pages = pages,
        FirstPageId = chapter.FirstPage
    };
    return View(model);
}

[HttpPost]
public async Task<IActionResult> EditChapter(AUTHOR_EditChapterViewModel model)
{
    if (ModelState.IsValid)
    {
        int newPageCount = model.PagesData!.Count(p => p.PageId == Guid.Empty);
        int newPageFileCount = model.NewPages is null ? 0 : model.NewPages.Count;

        if (newPageCount == newPageFileCount)
        {
            var chapter = _context.Chapters.Single(x => x.ChapterId == model.ChapterId);

            //Delete pages if needed
            for (int i = 0; i < model.PagesData!.Count; i++)
            {
                var currentPageNextPageId = model.PagesData[i].NextPageId;
                if (currentPageNextPageId is null)
                {
                    continue;
                }
                if (!model.PagesData.Any(p => p.PageId == currentPageNextPageId))
                {
```

```
                    var pageToDelete = _context.Pages.First(p => p.PageId ==
currentPageNextPageId);
                    _context.Pages.Remove(pageToDelete);
                }
            }

            if (!model.PagesData.Any(p => p.PageId == chapter.FirstPage))
            {
                var pageToDelete = _context.Pages.First(p => p.PageId == chapter.FirstPage);
                _context.Pages.Remove(pageToDelete);
            }

            // Add new pages if needed
            if (model.NewPages is not null)
            {
                for (int i = 0; i < model.PagesData.Count; i++)
                {
                    var currentPage = model.PagesData[i];
                    if (currentPage.PageId == Guid.Empty)
                    {
                        var filePath = Path.Combine(chapter.FolderLink,
model.NewPages.ElementAt(0).FileName);
                        using (var stream = new FileStream(filePath, FileMode.Create,
FileAccess.Write))
                        {
                            await model.NewPages.ElementAt(0).CopyToAsync(stream);
                        }
                        model.NewPages.RemoveAt(0);

                        var page = new Page
                        {
                            PageLink = filePath,
                            Chapter = chapter
                        };
                        await _context.Pages.AddAsync(page);

                        model.PagesData[i].PageId = page.PageId;
                    }
                }
            }
            await _context.SaveChangesAsync();

            // Switch pages' places if needed
            for (int i = 0; i < model.PagesData.Count - 1; i++)
            {
                var currentPage = model.PagesData[i];
                var nextPage = model.PagesData[i + 1];
                if (currentPage.NextPageId != nextPage.PageId)
                {
```

```
                    var contextCurrentPage = _context.Pages.First(p => p.PageId ==
currentPage.PageId);
                    var contextNextPage = _context.Pages.First(p => p.PageId == nextPage.PageId);
                    contextCurrentPage.NextPage = contextNextPage;
                }
            }

            var lastPage = _context.Pages.First(p => p.PageId == model.PagesData.Last().PageId);
            lastPage.NextPage = null;

            chapter.FirstPage = model.PagesData[0].PageId;
            await _context.SaveChangesAsync();
        }
      }
      return RedirectToRoute(new { controller = "Author", action = "EditSeries", seriesId =
model.SeriesId });
    }

    [HttpGet]
    public IActionResult ReadChapter(Guid chapterId, string layout, string direction)
    {
      var chapter = _context.Chapters.Include(c => c.Pages).Single(x => x.ChapterId ==
chapterId);
      var model = new ReadChapterViewModel
      {
        Chapter = chapter,
        Layout = layout,
        Direction = direction,
        FirstPageId = (Guid)chapter.FirstPage!
      };
      return View(model);
    }

    [HttpGet]
    public async Task<IActionResult> PublishLastChapter(Guid seriesId)
    {
      var series = _context.Series.Include(s => s.Chapters).Single(x => x.SeriesId == seriesId);
      if ((series.Chapters is not null) && !series.Chapters.Last().IsPublished)
      {
        series.Chapters.Last().IsPublished = true;
        await _context.SaveChangesAsync();
      }
      return RedirectToRoute(new { controller = "Author", action = "EditSeries", seriesId });
    }

    [HttpGet]
    public async Task<IActionResult> DeleteLastChapter(Guid seriesId)
    {
      var series = _context.Series.Include(s => s.Chapters).Single(x => x.SeriesId == seriesId);
      if ((series.Chapters is not null) && !series.Chapters.Last().IsPublished)
```

```
        {
            if ((series.Chapters.Last().FolderLink is not null) &&
Directory.Exists(series.Chapters.Last().FolderLink))
            {
                Directory.Delete(series.Chapters.Last().FolderLink, recursive: true);
            }
            _context.Chapters.Remove(series.Chapters.Last());
            await _context.SaveChangesAsync();
        }
        return RedirectToRoute(new { controller = "Author", action = "EditSeries", seriesId });
    }
  }
}
```

ReaderController.cs

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Mono.TextTemplating;

namespace ComiBerry.Controllers
{
  public class ReaderController(ApplicationDbContext context) : Controller
  {
    private readonly ApplicationDbContext _context = context;

    [HttpGet]
    public IActionResult ViewSeries(Guid seriesId)
    {
        var series = _context.Series.Include(s => s.Genre).Include(s => s.User).Include(s =>
s.Chapters).First(s => s.SeriesId == seriesId);
        if (series is not null)
        {
            var likes = _context.Faves.Where(l => l.Series == series);
            var model = new READER_ViewSeriesViewModel
            {
                Series = series,
                LikeCount = likes.Count()
            };
            return View(model);
        }
        ViewData["PageName"] = "Explore";
        return RedirectToAction("Home", "Navigation");
    }

    [HttpGet]
    public IActionResult ViewAuthor(string authorId)
    {
```

```
        var author = _context.Users.Include(a => a.Series).First(a => a.Id == authorId);
        if (author is not null)
        {
            var model = new READER_ViewAuthorViewModel
            {
                UserId = author.Id,
                UserName = author.UserName,
                Bio = author.Bio,
                Series = author.Series
            };
            return View(model);
        }
        ViewData["PageName"] = "Explore";
        return RedirectToAction("Home", "Navigation");
    }

    [HttpGet]
    public async Task<IActionResult> ViewChapter(Guid chapterId, string layout, string direction)
    {
        var chapter = _context.Chapters.Include(c => c.Pages).Include(c => c.Series).Include(c =>
c.Series.User).Single(x => x.ChapterId == chapterId);
        var comments = _context.Comments.Include(c => c.User).Where(c => c.Chapter ==
chapter).ToList();
        if ((HttpContext.Items["CurrentUser"] is not null) &&
(HttpContext.Items["CurrentUser"] != chapter.Series.User))
        {
            chapter.Series.ViewCount += 1;
            await _context.SaveChangesAsync();
        }

        var model = new READER_ViewChapterViewModel
        {
            Chapter = chapter,
            Title = chapter.Title,
            Layout = layout,
            Direction = direction,
            FirstPageId = chapter.FirstPage,
            Comments = comments
        };
        return View(model);
    }

    public class LikeRequest
    {
        public required string SeriesId { get; set; }
    }

    [Authorize]
    public async Task<IActionResult> LikeSeries([FromBody] LikeRequest request)
    {
```

```
        if (HttpContext.Items["CurrentUser"] is not User user) { return Json(new { success =
false }); }

        var series = _context.Series.First(s => s.SeriesId == Guid.Parse(request.SeriesId));
        if (series.User == user) { return Json(new { success = false }); }

        var existingFave = _context.Faves.Where(f => (f.User == user) && (f.Series ==
series)).ToList();
        if ((existingFave is null) || (existingFave.Count == 0))
        {
            var newFave = new Fave
            {
                User = user,
                Series = series
            };
            _context.Faves.Add(newFave);
            await _context.SaveChangesAsync();
            var likes = _context.Faves.Where(l => l.Series == series).Count();
            return Json(new { success = true, likeCount = likes });
        }
        return Json(new { success = false });
    }

    [Authorize]
    public async Task<IActionResult> AddComment(READER_AddCommentViewModel model)
    {
        var user = HttpContext.Items["CurrentUser"] as User;
        if (user is not null)
        {
            var comment = new Comment
            {
                Text = model.Text!,
                DateTime = DateTime.Now,
                Chapter = _context.Chapters.First(c => c.ChapterId == model.ChapterId),
                User = user
            };
            _context.Comments.Add(comment);
            await _context.SaveChangesAsync();
        }
        return RedirectToAction("ViewChapter", "Reader", new { chapterId = model.ChapterId,
layout = model.Layout, direction = model.Direction });
    }
  }
}
```

AdminController.cs

```
using System.Data;
using Microsoft.AspNetCore.Authorization;
```

```
using Microsoft.AspNetCore.Mvc;

namespace ComiBerry.Controllers
{
   [Authorize(Roles = "superadmin, admin")]
   public class AdminController(UserManager<User> userManager, ApplicationDbContext
context) : Controller
   {
      private readonly UserManager<User> _userManager = userManager;
      private readonly ApplicationDbContext _context = context;

      // FIND USERS
      [HttpGet]
      public IActionResult GetUsers(string currentRole)
      {
         var users = _userManager.GetUsersInRoleAsync(currentRole).Result;
         var model = new ADMIN_GetUsersViewModel()
         {
            Users = [.. users],
            Role = currentRole
         };
         return View(model);
      }

      [HttpPost]
      public IActionResult GetUserByParameters(ADMIN_GetUsersViewModel model)
      {
         if (ModelState.IsValid)
         {
            List<User>? users = null;
            if (model.Id is not null)
            {
               users = [.. _userManager.GetUsersInRoleAsync(model.Role).Result.Where(u => u.Id
== model.Id)];
            }
            else if (model.UserName is not null)
            {
               users = [.. _userManager.GetUsersInRoleAsync(model.Role).Result.Where(u =>
u.UserName == model.UserName)];
            }
            else if (model.Email is not null)
            {
               users = [.. _userManager.GetUsersInRoleAsync(model.Role).Result.Where(u =>
u.Email == model.Email)];
            }
            var newModel = new ADMIN_GetUsersViewModel()
            {
               Users = users,
               Role = model.Role
            };
```

```
            return View("~/Views/Admin/GetUsers.cshtml", newModel);
        }
        var referer = Request.Headers.Referer.ToString();
        if (referer is null)
        {
            ViewData["PageName"] = "Explore";
            return RedirectToAction("Home", "Navigation");
        }
        return Redirect(referer);
    }

    // EDIT USER
    [HttpGet]
    public async Task<IActionResult> SetAsAdmin(string userId)
    {
        var user = await _userManager.FindByIdAsync(userId);
        if (user is not null)
        {
            var userRole = _userManager.GetRolesAsync(user).Result;
            if ((userRole is not null) && !userRole.Contains("admin"))
            {
                await _userManager.RemoveFromRolesAsync(user, userRole);
                await _userManager.AddToRoleAsync(user, "admin");
            }
        }
        var referer = Request.Headers.Referer.ToString();
        if (referer is null)
        {
            ViewData["PageName"] = "Explore";
            return RedirectToAction("Home", "Navigation");
        }
        return Redirect(referer);
    }

    [HttpGet]
    public async Task<IActionResult> DeleteUser(string userId, string currentRole)
    {
        var user = _context.Users.Include(u => u.Series).First(u => u.Id == userId);
        if (user is not null)
        {
            if (user.Series is not null)
            {
                foreach (var work in user.Series)
                {
                    work.User = null!;
                }
            }
            var result = await _userManager.DeleteAsync(user);
        }
        return RedirectToRoute(new { controller = "Admin", action = "GetUsers", currentRole });
```

```
}

//VIEW SERIES
[HttpGet]
public IActionResult GetSeries()
{
    var series = _context.Series.ToList();
    var model = new ADMIN_GetSeriesViewModel
    {
        Series = series
    };
    return View(model);
}

public IActionResult GetSeriesByParameters(ADMIN_GetSeriesViewModel model)
{
    if (ModelState.IsValid)
    {
        List<Series>? series = null;
        if (model.Id is not null)
        {
            series = [.. _context.Series.Where(u => u.SeriesId == Guid.Parse(model.Id))];
        }
        else if (model.Title is not null)
        {
            series = [.. _context.Series.Where(u => u.Title == model.Title)];
        }
        else if (model.Year is not null)
        {
            series = [.. _context.Series.Where(u => u.Year == model.Year)];
        }
        var newModel = new ADMIN_GetSeriesViewModel()
        {
            Series = series
        };
        return View("~/Views/Admin/GetSeries.cshtml", newModel);
    }
    var referer = Request.Headers.Referer.ToString();
    if (referer is null)
    {
        ViewData["PageName"] = "Explore";
        return RedirectToAction("Home", "Navigation");
    }
    return Redirect(referer);
}

//EDIT SERIES
[HttpGet]
public IActionResult ViewProfile(string userId)
{
```

```
            var author = _context.Users.Include(a => a.Series).First(a => a.Id == userId);
            var authorRole = _userManager.GetRolesAsync(author).Result;
            if (author is null)
            {
                var referer = Request.Headers.Referer.ToString();
                if (referer is null)
                {
                    ViewData["PageName"] = "Explore";
                    return RedirectToAction("Home", "Navigation");
                }
                return Redirect(referer);
            }
            var model = new ADMIN_ViewProfileViewModel
            {
                Author = author,
                Role = authorRole[0],
                Series = author.Series
            };
            return View(model);
        }

        [HttpGet]
        public IActionResult EditSeries(Guid seriesId)
        {
            var series = _context.Series.Include(s => s.Genre).Include(s => s.User).Include(s =>
s.Chapters).First(s => s.SeriesId == seriesId);
            if (series is null)
            {
                var referer = Request.Headers.Referer.ToString();
                if (referer is null)
                {
                    ViewData["PageName"] = "Explore";
                    return RedirectToAction("Home", "Navigation");
                }
                return Redirect(referer);
            }
            var model = new ADMIN_EditSeriesViewModel
            {
                Series = series
            };
            return View(model);
        }

        [HttpGet]
        public async Task<ActionResult> DeleteSeries(Guid seriesId)
        {
            var series = _context.Series.First(s => s.SeriesId == seriesId);
            if (series is not null)
            {
                var faves = _context.Faves.Where(f => f.Series == series).ToList();
```

```
            if (faves.Count > 0)
            {
                foreach (var fave in faves)
                {
                    _context.Faves.Remove(fave);
                }
            }
            if ((series.FolderLink is not null) && Directory.Exists(series.FolderLink))
            {
                Directory.Delete(series.FolderLink, recursive: true);
            }
            if (series.CoverLink is not null)
            {
                System.IO.File.Delete(series.CoverLink);
            }
            _context.Series.Remove(series);
            await _context.SaveChangesAsync();
        }
        return RedirectToRoute(new { controller = "Admin", action = "GetSeries" });
    }
  }
}
```

ДОДАТОК В (обов'язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

РОЗРОБКА WEB-ПЛАТФОРМИ ДЛЯ ПУБЛІКАЦІЇ КОМІКСІВ

Виконала: студентка 4 курсу, групи 1КН-21б
спеціальності 122 Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Сичова М.С.
(прізвище та ініціали)

Керівник: к.т.н., доцент каф. КН
Сілагін О. В.
(прізвище та ініціали)

«03» червня 2025 р.

Вінниця ВНТУ – 2025 рік

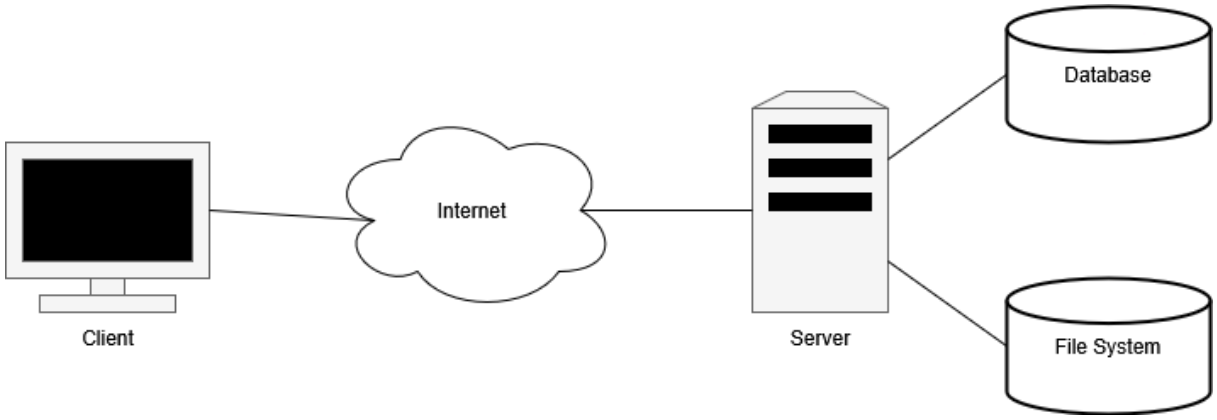

Рисунок В.1 – Архітектура платформи для публікації коміксів

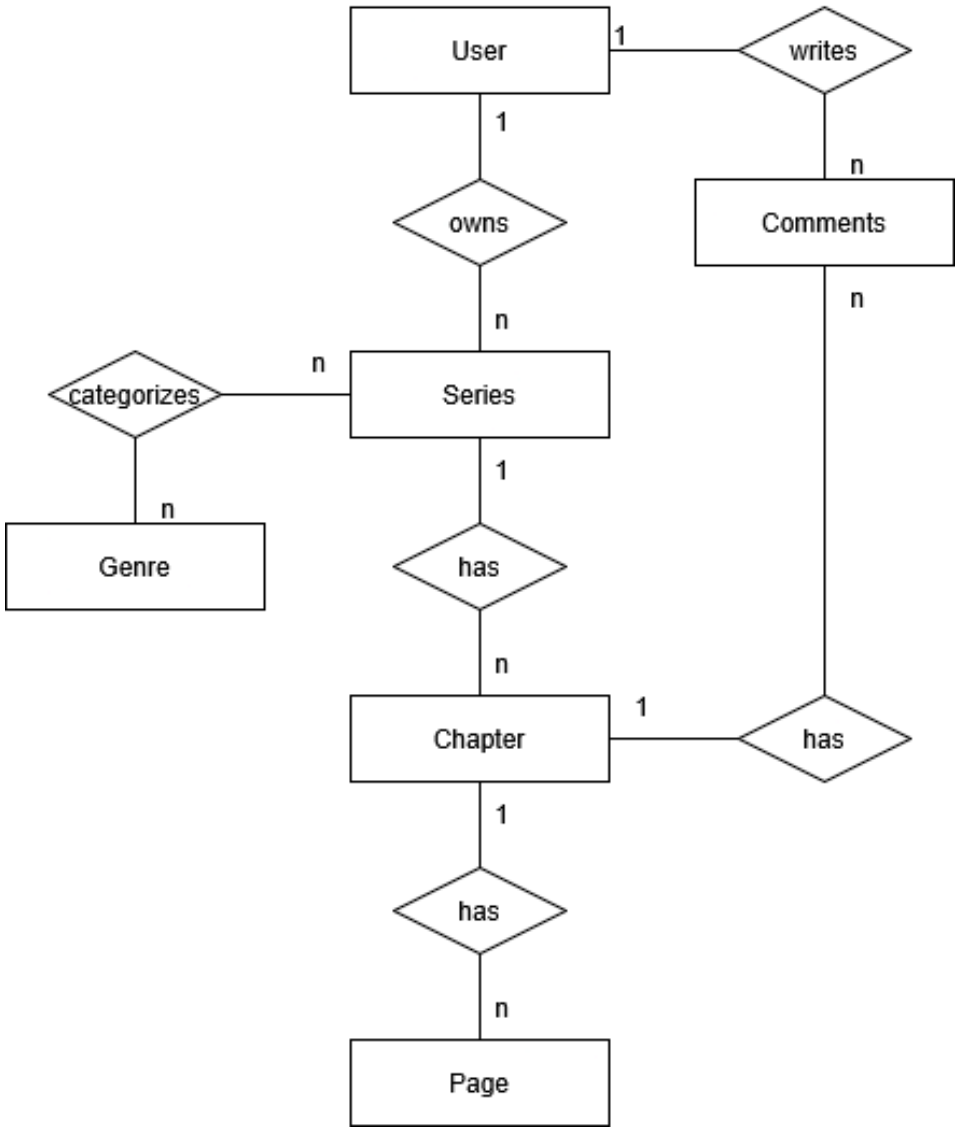

Рисунок В.2 – ER-модель платформи для публікації коміксів

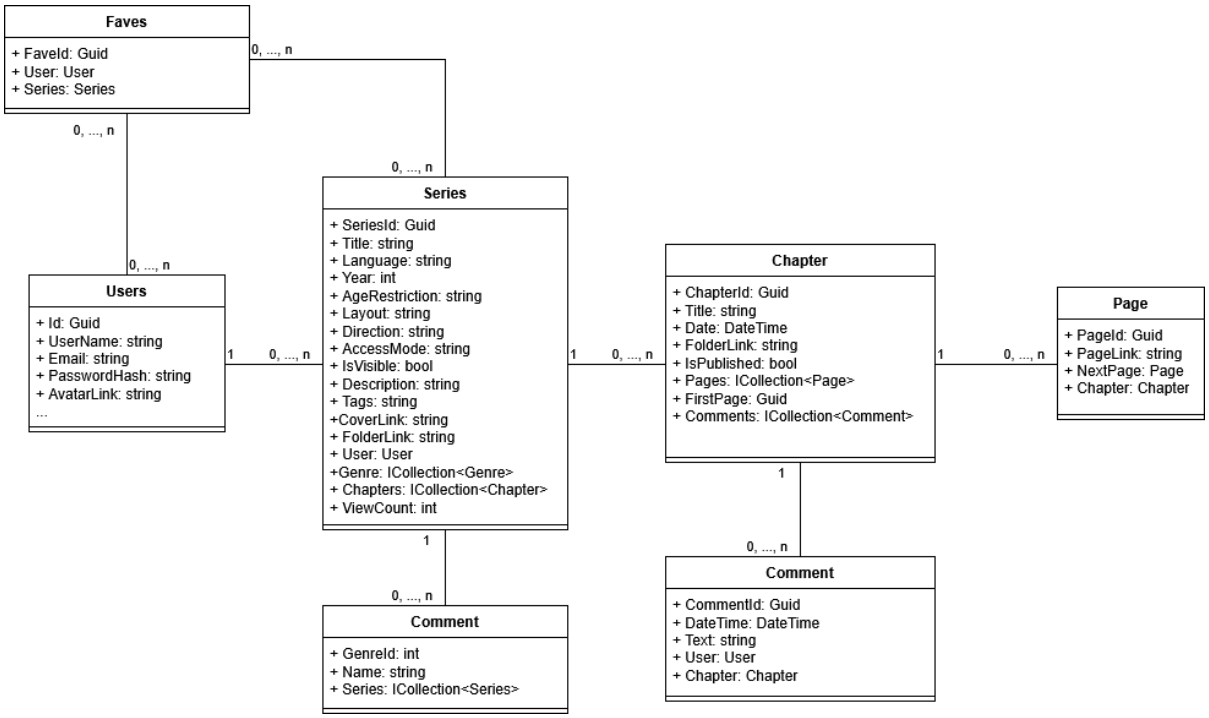

Рисунок В.3 – Діаграма класів

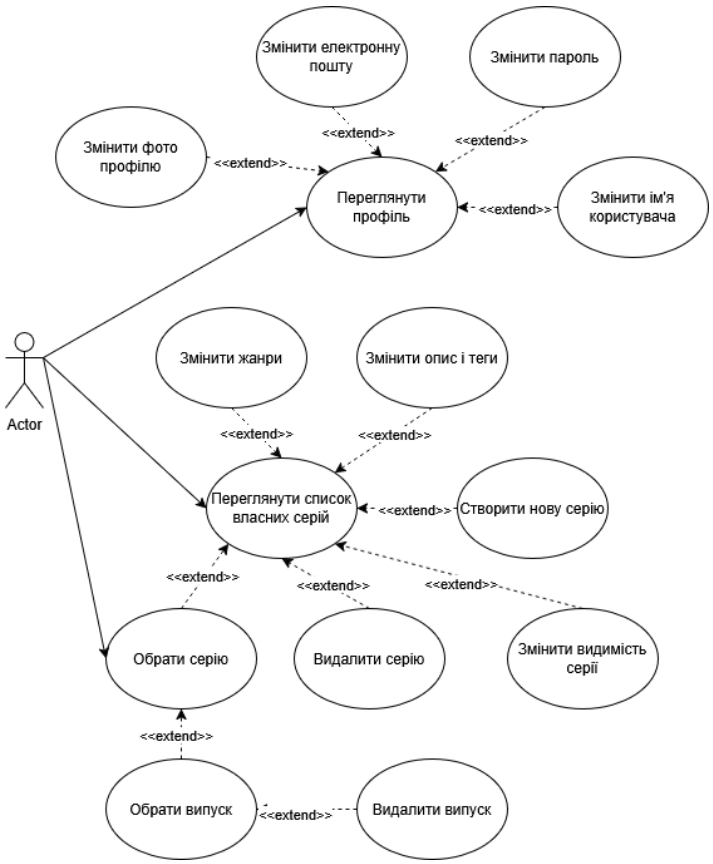

Рисунок В.4 – Діаграма прецедентів

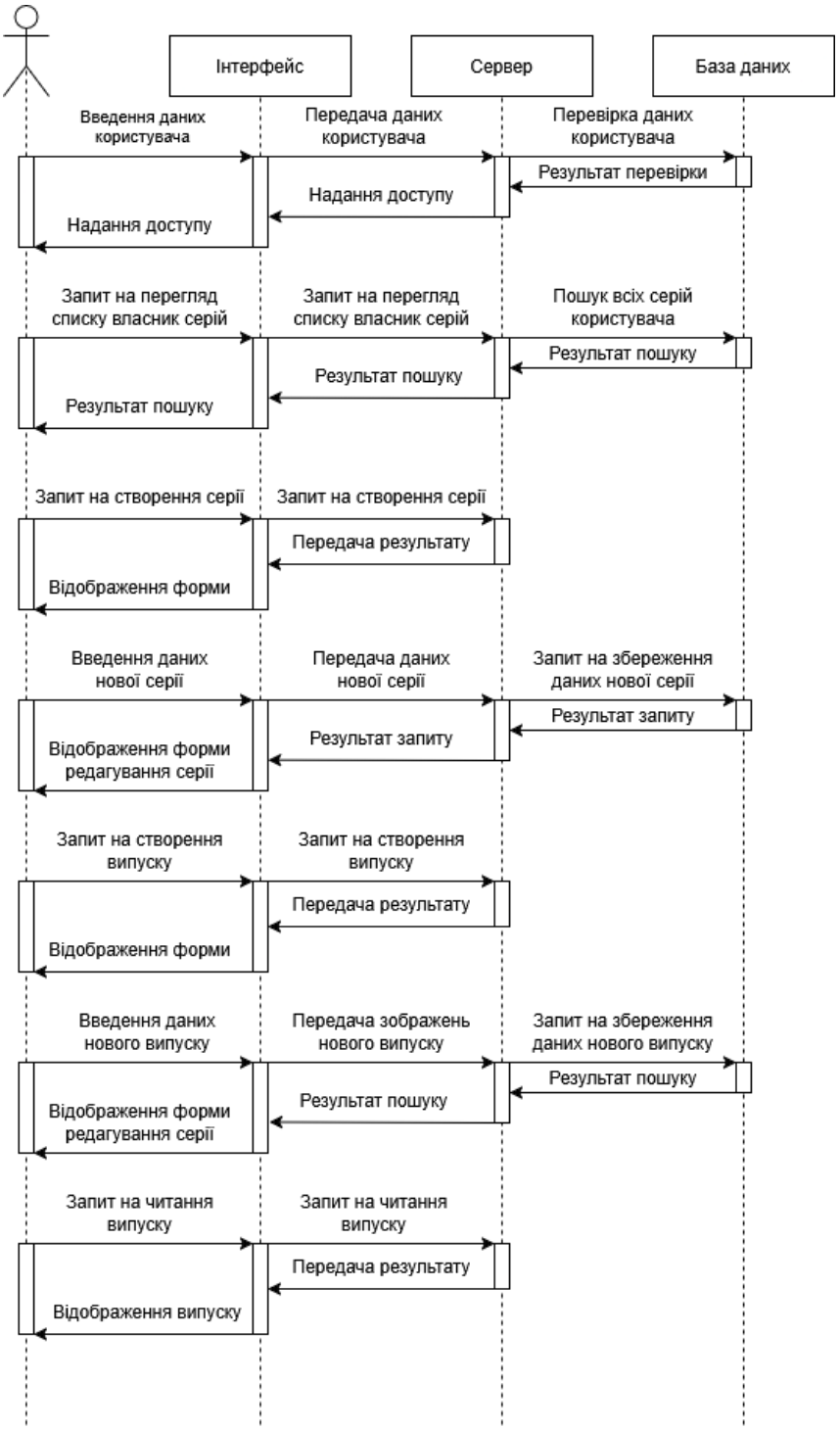

Рисунок В.5 – Діаграма послідовності

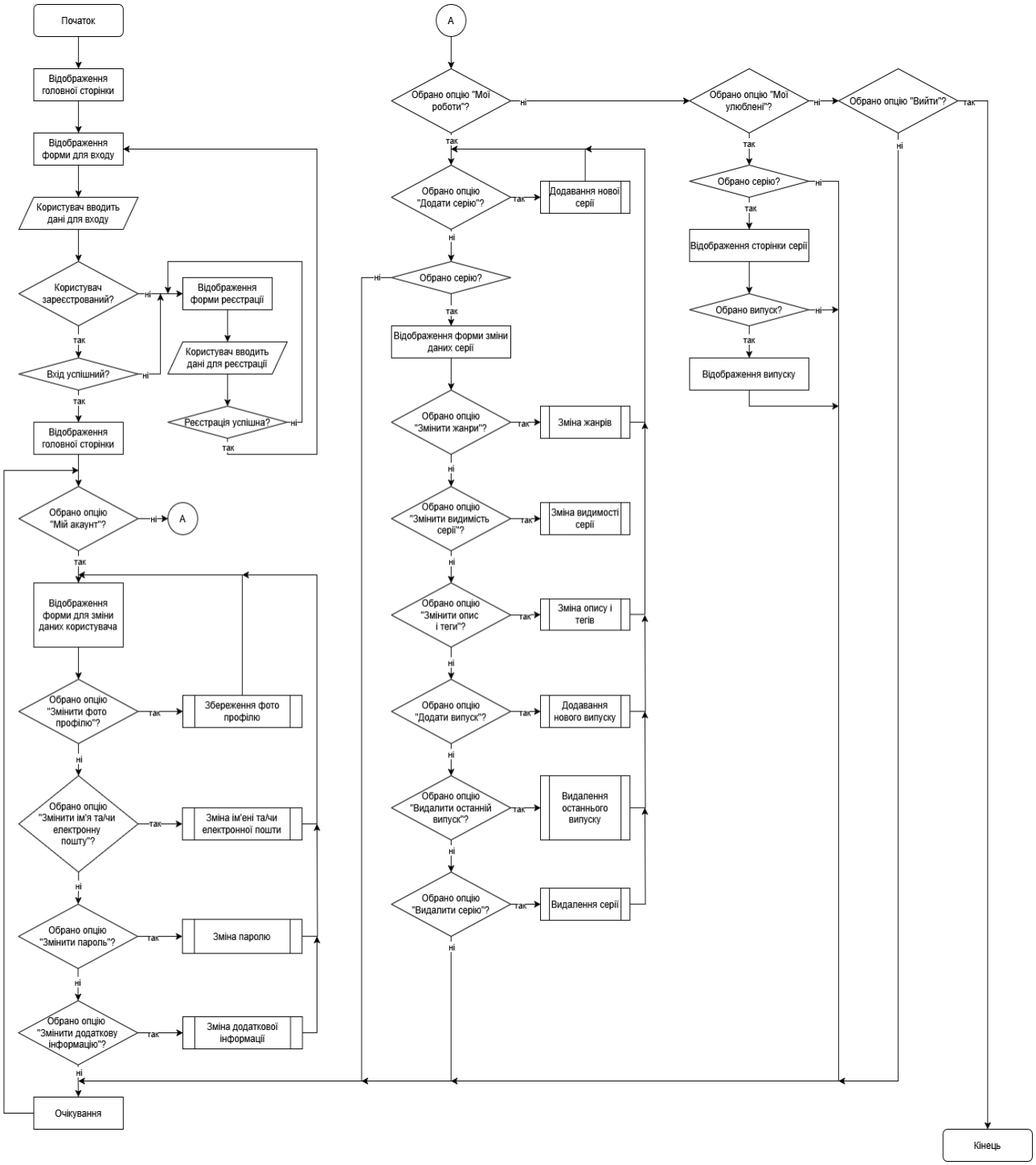

Рисунок В.6 – Загальний алгоритм роботи платформи для публікації коміксів

ДОДАТОК Г (довідниковий)
ІНСТРУКЦІЯ КОРИСТУВАЧА

Для реєстрації користувача потрібно ввести ім’я користувача, електронну пошту, пароль та підтвердити пароль. Сторінка реєстрації разом з панеллю для входу зображено на рисунку Г.1.

Рисунок Г.1 – Сторінка реєстрації користувача

Після успішної реєстрації/входу, користувач повертається до головного вікна програми (рис. Г.2).

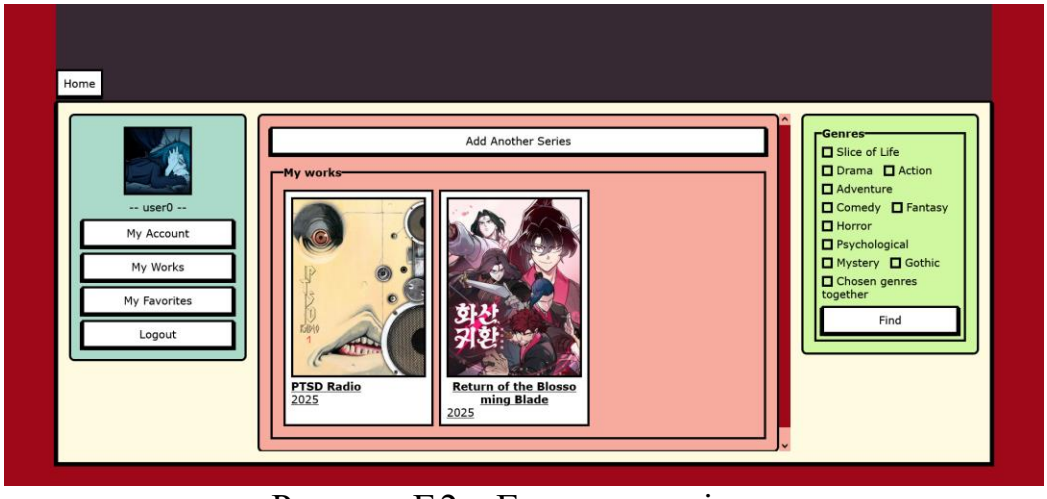

Рисунок Г.2 – Головна сторінка

На лівій боковій панелі відображається меню користувача з такими опціями: «My Account», «My Works», «My Favorites», «Logout».

При натисканні на кнопку «My Account», користувача буде перенесено на сторінку налаштування профілю користувача (рис. Г.3).

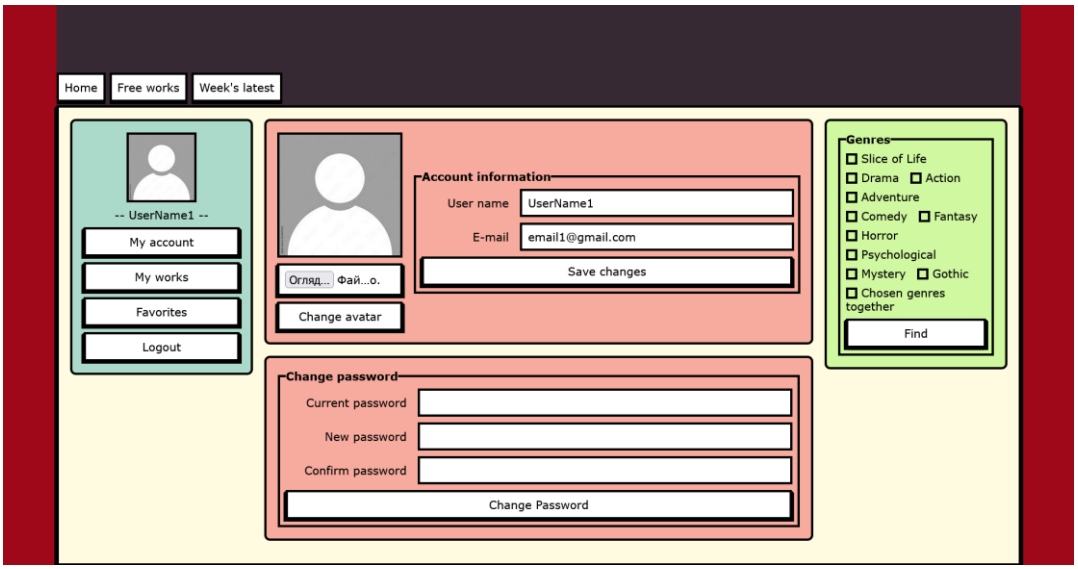

Рисунок Г.3 – Профіль користувача

Щоб змінити фото профілю, потрібно натиснути на кнопку над кнопкою «Change avatar», обрати фото та натиснути на кнопку «Change avatar». Після цього зображення буде змінено. У панелі «Change password» можна змінити власний пароль. Для цього потрібно ввести теперішній пароль, новий пароль і підтвердити новий пароль. Після натискання на кнопку «Change Password», пароль буде змінено. У панелі «Change Bio» можна змінити додаткову інформацію про користувача. Для цього потрібно ввести нову інформацію та натиснути на кнопку «Change Password», додаткову інформацію буде змінено.

Після натискання на кнопку «My works», користувач переходить на сторінку власних робіт (рис. Г.4). Щоб додати нову роботу, потрібно натиснути на кнопку «Add Another Series». Користувача буде переведено до форми створення серії (рис. Г.5). Потрібно обрати назву, мову, жанр, вікове обмеження, тип макету панелей, напрям читання форму монетизації. Після натискання на

кнопку «Create the Series», знову відображається сторінка власних робіт користувача, але на цей раз із новою серією.

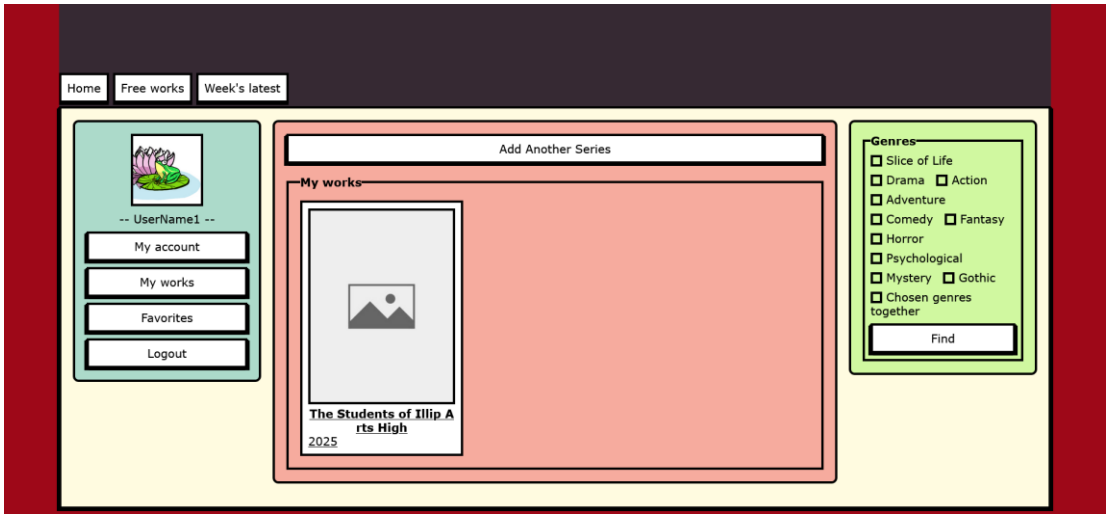

Рисунок Г.4 – Сторінка власних робіт користувача

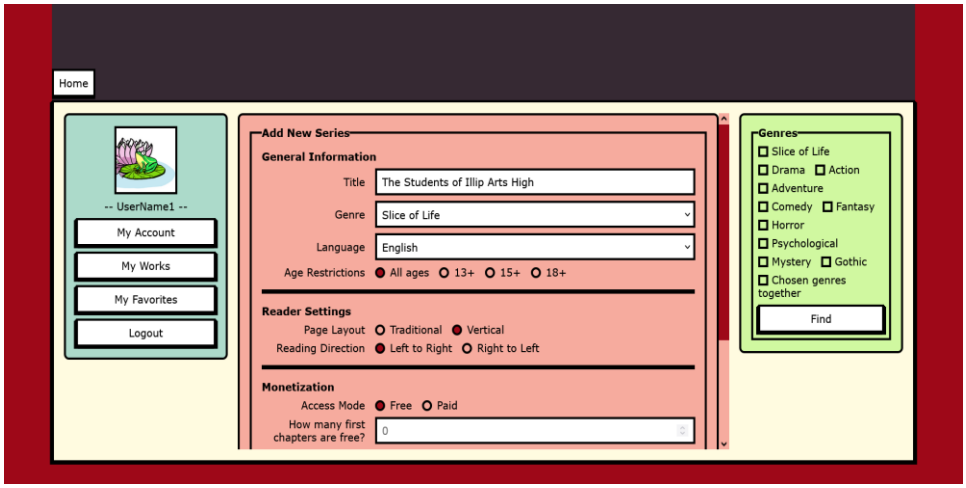

Рисунок Г.5 – Форма створення нової серії

Якщо користувач натисне на власну роботу, його буде перенесено на сторінку редагування серії (рис. Г.6). Подібно до зміни фото профілю, можна так само змінити обкладинку серії. Якщо обрати інший набір жанрів та натиснути на кнопку «Save genres», буде відображено новий набір жанрів. Якщо натиснути на кнопку «Make the Series visible», серія буде видима для всіх користувачів. Далі користувач не зможе видалити серію. Якщо ввести опис серії,

теги та натиснути на кнопку «Save the Description and Tags», то їх буде збережено.

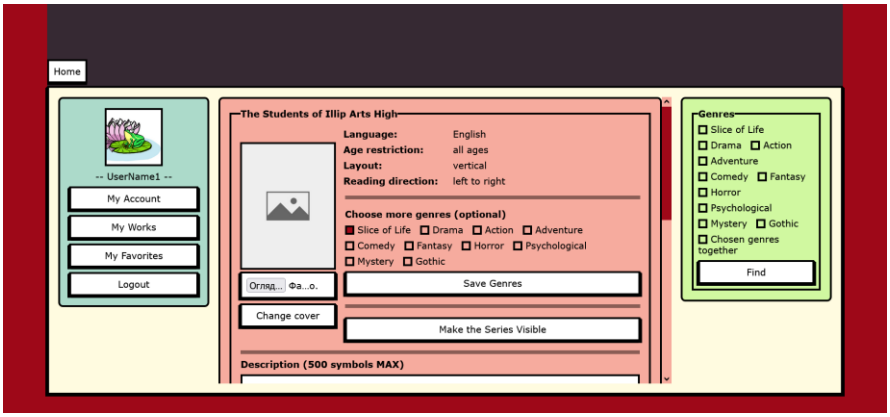

Рисунок Г.6 – Сторінка редагування серії

Після натискання на кнопку «Add a new Chapter», користувача буде направлено на сторінку створення випуску (рис. Г.7). Потрібно ввести назву випуску та додати сторінки. За допомогою кнопок «Up» і «Down» можна переміщати сторінки вверх-вниз. Після натискання на кнопку «Save the Chapter», користувача буде переведено до сторінки редагування серії. Далі випуск можна редагувати (додавати панелі, видаляти панелі, міняти їх місцями) та переглядати.

Рисунок Г.7 – Створення нового випуску

Для читача, головна сторінка серії виглядає як на рисунку Г.8. В залежності від формату панелей, сторінки можуть бути представленні або у горизонтальному рядку (рис. Г.9), або у вертикальному рядку (рис. Г.10).

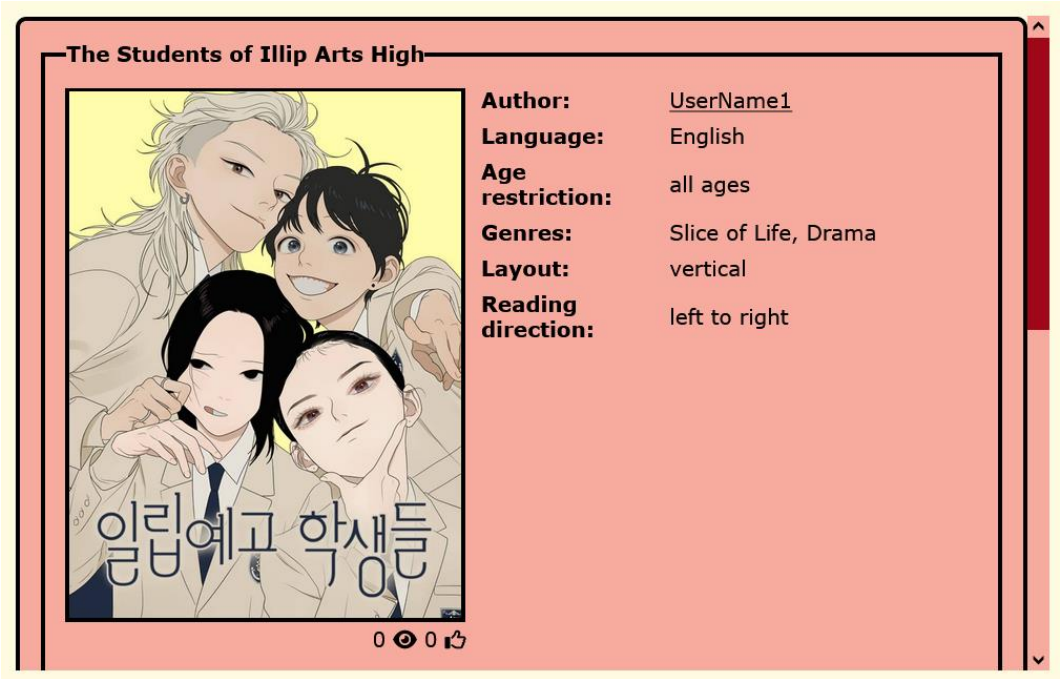

Рисунок Г.8 – Сторінка читача

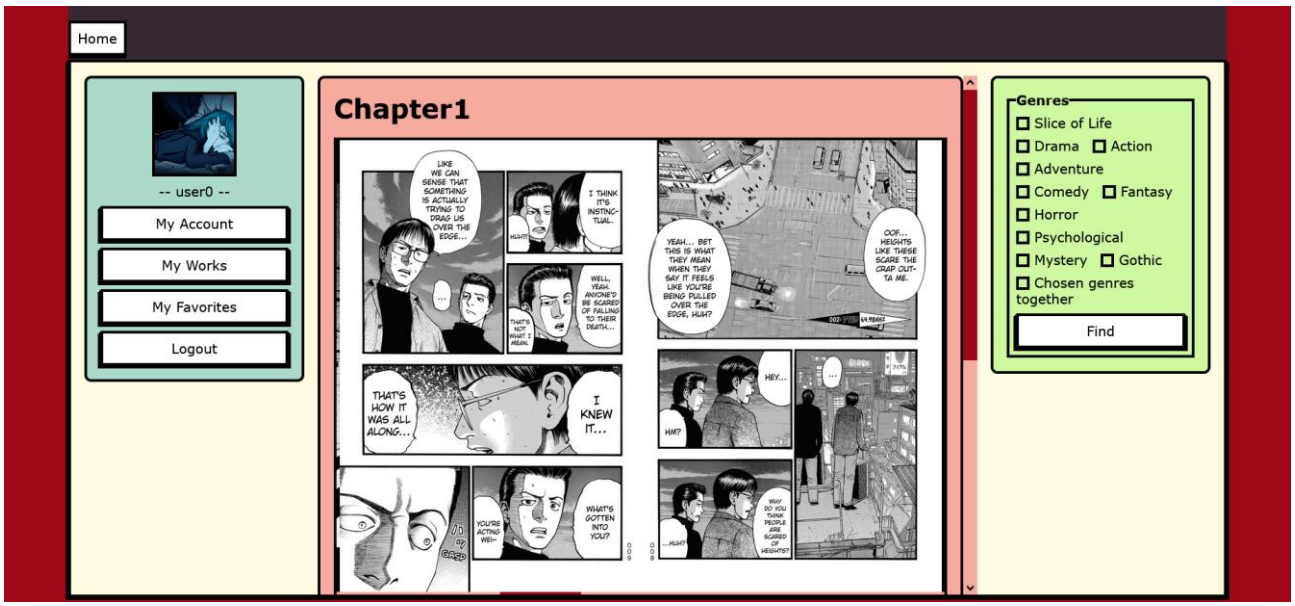

Рисунок Г.9 – Відображення випуску з традиційним форматом панелей

Рисунок Г.10 – Відображення випуску з вертикальним форматом панелей

Під кожним розділом можна додати коментарі. Для цього потрібно ввести його у виділену форму та натиснути на кнопку «Post Comment» (рис. Г.11). Після цього коментар буде опубліковано під цією формою.

Рисунок Г.11 – Додавання коментарів

Користувачі також можуть закладати улюблені роботи, натиснувши на кнопку «Вподобайка». Після вибору читачем розділу для читання, кількість переглядів серії збільшується на одиницю.

Додаток Д (довідниковий). Довідка про впровадження

МАЛЕ НАУКОВО-ВИРОБНИЧЕ ПІДПРИЄМСТВО "ТОВ "ІТІ"
Україна, 21021, м.Вінниця, вул. Келецька, 56

№ 39 від "6" 06 2025р.

ДОВІДКА

Дана студентці групи 1КН-21б Сичовій Марині Сергіївні в тому що результати, одержані нею в процесі виконання бакалаврської дипломної роботи «Розробка WEB-платформи для публікації коміксів», а саме: алгоритми, комунікації та форми, планується використати в розробках ТОВ «ІТІ».

Зам. Директора ТОВ «ІТІ» Бодяк В.Н.