

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр» за спеціальністю 121 «Інженерія програмного забезпечення» (освітня програма «Інженерія програмного забезпечення») на тему:

«Соціальна мережа читачів книг»

Виконала студентка групи ІПЗ-20-4

КОЛІСНИК Ольга Володимирівна

Керівник роботи:

КОНЦЕДАЙЛО Валерій Валерійович

Консультант:

ЧИЖМОТРЯ Олексій Володимирович

Рецензент:

МОРОЗОВ Андрій Васильович

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Завідувач кафедри інженерії
програмного забезпечення

Тетяна ВАКАЛЮК

«26» лютого 2024 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **КОЛІСНИК Ольга Володимирівна**

Керівник роботи: **КОНЦЕДАЙЛО Валерій Валерійович**

Тема роботи: **«Соціальна мережа читачів книг»,**

затверджена Наказом закладу вищої освіти від **«23» лютого 2024 р., № №74/с**

Вихідні дані для роботи: розробка соціальної мережі читачів книг

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів,
що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Концедайло В. В., Чижмотря О. В.	27.02.2024р.	27.02.2024р.
2	Концедайло В. В., Чижмотря О. В.	27.02.2024р.	27.02.2024р.
3	Концедайло В. В., Чижмотря О. В.	27.02.2024р.	27.02.2024р.

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається з програмного комплексу веб - сайту соціальної мережі читачів книг та пояснювальної записки. Пояснювальна записка до випускної роботи містить текстову частину, викладену на 65 сторінках друкованого тексту, має 37 сторінок додатків. Список використаних джерел містить 30 найменувань і займає 3 сторінки. В роботі наведено 27 рисунків та 14 таблиць. Загальний обсяг роботи – 102 сторінки.

Метою роботи є розробка веб-сайту соціальної мережі читачів книг є створення платформи, де користувачі зможуть ділитися своїми думками щодо прочитаних книг, знаходити нову книгу для прочитання та збереження даних про прочитані книжки та заплановані книжки.

У роботі було визначено вимоги користувача та обґрунтовано створення концептуальної моделі. Також було проаналізовано аналоги програмних продуктів та обрано засоби реалізації, проведено розробку структури бази даних, а також було розроблено алгоритм роботи, обрано архітектуру та розроблено об'єктно-орієнтовану структуру системи. Також описано програмну реалізацію сайту.

КЛЮЧОВІ СЛОВА: АДМІНІСТРУВАННЯ, БАЗА ДАНИХ, ІНТЕРНЕТ, КНИГИ, ЖАНРИ, АВТОРИ, КОРИСТУВАЧІ, СОЦІАЛЬНА МЕРЕЖА, САЙТ.

					ІПЗ.КР.Б – 121 – 24 – ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата	Соціальна мережа читачів книг Пояснювальна записка	Літ.	Арк.	Аркушів
Розробник	Колісник О.В.						3	102
Керівник	Концедайло В. В.							
Рецензент	Морозов А. В.					Житомирська політехніка, група ІПЗ-20-4		
Зав. каф.	Вакалюк Т. А.							

ABSTRACT

The bachelor's thesis consists of a complex software implementation of a graphic editor, as well as an explanatory note. The explanatory note to the thesis contains 65 pages, 37 pages of appendices, 27 illustrations, 14 tables and 30 references. The total volume of bachelor's thesis is 102 pages.

The main purpose of this work is to develop social network of book readers web-site with using of HTML, CSS, JavaScript and also developing of the administration system with using of Python's framework Django.

The paper defined user requirements and substantiated the creation of a conceptual model. Analogues of software products were also analyzed and means of implementation were selected, the database structure was developed, and the work algorithm was developed, the architecture was chosen, and the object-oriented structure of the system was developed. The software implementation of the site is also described.

KEYWORDS: ADMINISTRATION, DATABASE, INTERNET, BOOKS, GENRES, AUTHORS, USERS, SOCIAL NETWORK, WEB-SITE.

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		4

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	6
ВСТУП.....	7
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ СИСТЕМИ	9
1.1. Опис проекту та постановка задачі.....	9
1.2. Аналіз аналогів програмного продукту.....	10
1.3. Вибір архітектури веб-орієнтованої системи	12
1.4. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення	14
Висновки до розділу 1	17
РОЗДІЛ 2. ПРОЕКТУВАННЯ СОЦІАЛЬНО МЕРЕЖІ ЧИТАЧІВ КНИГ	19
2.1. Визначення варіантів використання та об'єктно-орієнтованої структури системи	19
2.2. Розробка бази даних системи	23
2.3. Проектування та реалізація алгоритмів роботи системи.....	31
2.4. Реалізація функціоналу системи соціальної мережі читачів книг	33
Висновки до розділу 2	42
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ПЛАТФОРМОЮ СОЦІАЛЬНОЇ МЕРЕЖІ ЧИТАЧІВ КНИГ	44
3.1. Порядок встановлення та налаштування параметрів платформи.....	44
3.2. Структура інтерфейсу соціальної мережі читачів книг.....	46
Висновки до розділу 3	58
ВИСНОВКИ.....	60
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	62
ДОДАТКИ.....	65

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

CMS – Content Management System.

CRUD – основні операції, що використовуються для управління даними (Create , Read , Update , Delete).

MVT – схема розділення даних додатків і керуючої логіки на три окремі компоненти Model-View-Template.

ORM – Object-Relational Mapping.

SQL – Structured Query Language.

БД – база даних.

СУБД – система управління базами даних.

ПЗ – програмне забезпечення.

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				6
Змн.	Арк.	№ докум.	Підпис	Дата		

ВСТУП

Актуальність теми. Використання веб-технологій помітно зростає в усіх сферах людського життя. Веб-сайт – це своєрідний інтерфейс між підприємством та його оточенням. Соціальна мережа у свою чергу це веб-сайт або інша служба у Веб, яка дозволяє користувачам створювати публічну або приватну анкету, скласти список користувачів, з якими вони мають зв'язок та переглядати власний список зв'язків і списки інших користувачів.

Тому створення сайту є одним з головних завдань підприємницької діяльності, в тому числі і в мережі Інтернет. Нині все більша кількість компаній починає використовувати Інтернет для обміну інформацією та оптимізації роботи з клієнтами. Найпоширенішою формою присутності у мережі є створення сайту. Структурована інформація сайту дає змогу користувачам отримати повне уявлення про підприємство і його діяльність.

Мета даної роботи полягає у розробці веб-сайту соціальної мережі читачів книг засобами HTML, CSS, JavaScript, а також розроблення системи адміністрування за допомогою мови програмування Java із використанням бази даних MySQL. Сервіс має забезпечувати функціонал для повної і безперебійної роботи порталу та взаємодіяти як з комп'ютерними платформами, так і мобільними. Також метою є поглиблення теоретичних та практичних знань та сформуванні вмінь і навиків, необхідних для розроблення веб-системи сайту.

Поставлена мета обумовлює необхідність виконання наступних завдань:

- аналіз теоретичних та функціональних потреб проектування та реалізації сайту;
- аналіз існуючих веб-сайтів на обрану тематику, визначення унікальності;
- визначення технологій для реалізації веб-сервісу;
- визначення інформаційних потреб предметної області дослідження;
- обрання оптимальної в сучасних умовах архітектури функціонування програмного комплексу;
- виконання опису методів для взаємодії з веб-системою;

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				7
Змн.	Арк.	№ докум.	Підпис	Дата		

- проектування бази даних, для збереження інформації сервісу та користувачів;
- визначення об'єктної структури веб-сервісу кінотеатру;
- розробка адаптивного інтерфейсу веб-сайту;
- розробка математичної та алгоритмічної моделі функціонування системи на основі БД [1];
- написання серверної частини;
- реалізація CMS [2].

Об'єктом дослідження є методи та засоби розробки веб-сайту за визначеними предметними областями.

Предметом дослідження є моделі, технології, структури, засоби та методи розробки та взаємодії веб-сайтів та модулів управління сайту. А також використанні необхідні веб-орієнтовані технології. В роботі було використано методи об'єктно орієнтованого проектування, веб-програмування та дизайну користувацьких інтерфейсів.

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				8
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ СИСТЕМИ

1.1. Опис проекту та постановка задачі

Метою кваліфікаційною роботи бакалавра є створення соціальної мережі читачів книг, використовуючи сучасні технології, що дозволить легко користуватись та масштабувати її у майбутньому.

На сьогоднішній день практично кожна комерційна організація має сторінку у соціальній мережі. В умовах використання сучасних інформаційних технологій – це необхідний чинник існування, що дозволяє розширити поле рекламної діяльності та залучити тим самим додаткових клієнтів, які зможуть реєструватися на сайті, додавати обрані книжки та залишати коментарі.

Основними етапами розробки платформи є:

1. Розробка дизайну вебсайту:

- створення загальної концепції дизайну додатку;
- створення UI компонентів [3];
- розробка дизайну сторінок.

2. Побудова архітектури клієнтської частини додатку [4].

3. Побудова архітектури серверної частини додатку.

4. Розробка системи авторизації та ролей користувачів.

5. Розробка основного функціоналу:

- створення логіки авторизації користувачів з відповідними ролями [5];
- створення віддаленої бази даних для зберігання інформації;
- імплементація бекенду на клієнтській стороні;
- організація спілкування серверної частини з клієнтською;

6. Тестування додатку:

- функціональне тестування додатку [6];
- тестування графічного інтерфейсу.

Результатом виконання плану дій є створення соціальної мережі, за допомогою якої користувачі зможуть взаємодіяти з контентом сайту.

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				9
Змн.	Арк.	№ докум.	Підпис	Дата		

1.2. Аналіз аналогів програмного продукту

Під час аналізу існуючих веб-сайтів за тематикою кваліфікаційної роботи було знайдено багато тематично схожих прикладів, проте усі вони мають дещо інший функціонал.

На рисунку 1.1 представлено веб-сайт «Goodreads» [7]. Дана платформа є лідером серед соціальних мереж читачів книг, має гарний дизайн, який є приємним користувачу, будь-який бажаючий може зареєструватися та створити збір.

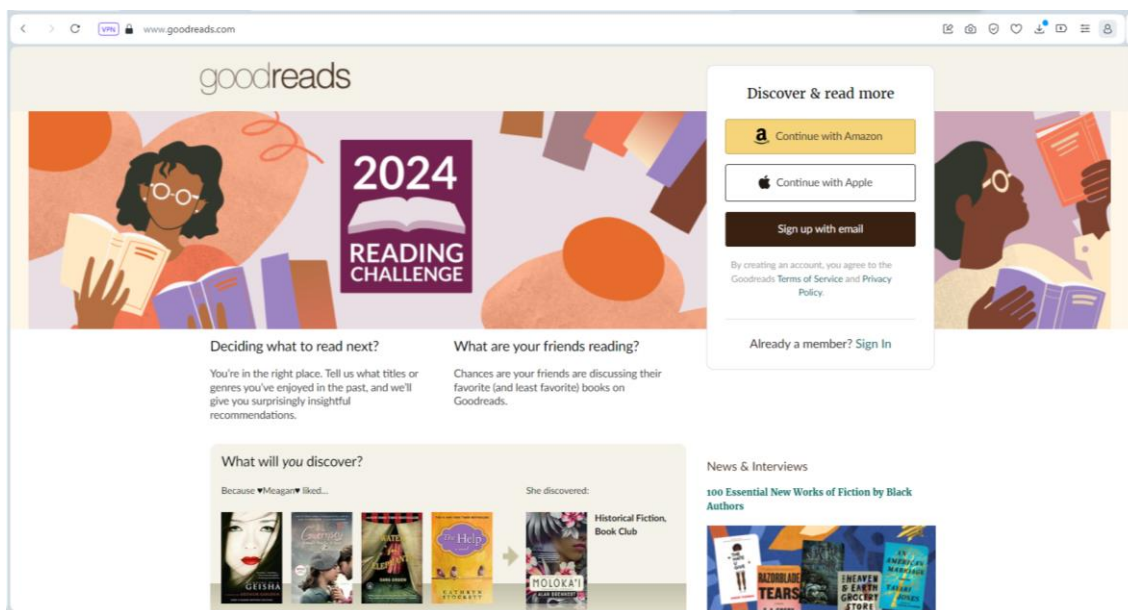


Рис. 1.1. Приклад одного із існуючих веб-сайтів за тематикою курсового проекту
Основний перевагами Goodreads є:

- інтуїтивно зрозуміла система у користуванні;
- має мобільні додатки для IOS і для Android.

Недоліки:

- відсутня українська локалізація;
- низька кількість можливостей для зареєстрованого користувача;
- немає можливості написати статтю;
- відсутній форум.

На рисунку 1.2 представлений наступний приклад, що вартий уваги, це український застосунок «RORK» [8]. Ця платформа більше зорієнтована на відслідковування темпу читання книжок. Вона надає багато можливостей,

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				10
Змн.	Арк.	№ докум.	Підпис	Дата		

включаючи: трекер, календар, нотатки, роботу офлайн, спільноту, колекції, оцінки, підсумки року, темну тему, пошук книжок, сканер штрихкоду, відгуки.

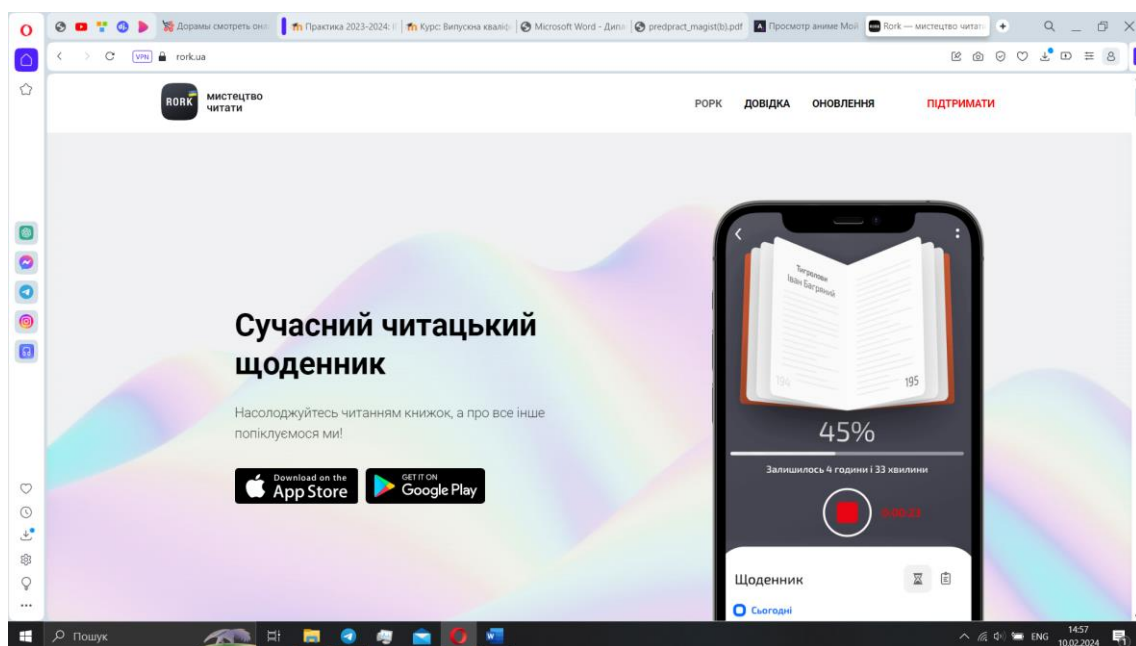


Рис. 1.2. Приклад одного із існуючих веб-сайтів за тематикою курсового проекту
Основний перевагами Goodreads є:

- проста авторизація;
- трекер читання;
- інтуїтивно зрозуміла система у користуванні;
- українська локалізація

Недоліками є:

- відсутній веб-сайт;
- немає можливості переглянути книжки інших користувачів;
- відсутній форум.

Виходячи з наявних переваг та недоліків, складемо порівняльну таблицю вищезгаданих програмних аналогів (табл. 1.1).

Таблиця 1.1

Таблиця порівняння існуючих соціальних мереж

Характеристика	Goodreads	RORR	Book Library
1.	2.	3.	4.
Проста авторизація			+

1.	2.	3.	4.
Зручність використання	+	-	+
Багатоплатформеність	+	-	-
Українська локалізація	-	+	+
Додавання відгуків	+	+	+
Наявність форуму	-	-	+

Обидві платформи, Goodreads та RORK, мають свої сильні сторони і слабкі місця. Goodreads виділяється своєю популярністю та наявністю мобільних додатків, але має обмежену функціональність для активної взаємодії користувачів і відсутність української локалізації. RORK, навпаки, пропонує корисні функції для відстеження прогресу читання та підтримує українську мову, але обмежений відсутністю веб-версії та соціальних функцій, таких як форуми.

Для створення нової соціальної мережі для читачів книг варто врахувати ці аспекти та об'єднати найкращі риси обох платформ, додавши при цьому відсутні функції. Зокрема, важливо забезпечити підтримку різних мов, зручний інтерфейс, мобільні додатки, а також можливість створення спільнот, обговорення книг, написання статей та перегляду бібліотек інших користувачів. Це дозволить створити багатофункціональну платформу, яка задовольнить потреби сучасних читачів.

1.3. Вибір архітектури веб-орієнтованої системи

Для того щоб розроблювана інформаційна система могла добре функціонувати довгий час, вона має забезпечувати поставленим вимогам, мати високу якість та бути доступною на легкому у використанні для звичайних користувачів. На всі ці параметри впливає архітектура програмного забезпечення.

Архітектура – це базова організація системи, втілена в її компонентах, зовнішніми та внутрішніми відношеннями у системі, а також принципи, що визначають проектування та розвиток системи [9].

Правильно та своєчасно спроектована архітектура полегшить подальшу розробку, впровадження, користування, модифікацію інформаційної системи.

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				12
Змн.	Арк.	№ докум.	Підпис	Дата		

Для створення подібних систем доцільно використовувати клієнтсерверну архітектуру програмного забезпечення.

Патерн проектування – це загальне рішення певної проблеми в дизайні архітектури [10]. Патерн являє собою не конкретний код, а загальну концепцію або приклад рішення яке має бути налаштоване під потреби вашої програми.

Використання перевірених патернів при проектуванні архітектури інформаційної системи полегшить подальший розвиток системи. Завдяки можливості повторного використання коду, яку дають нам патерни проектування, майбутнє розширення системи або внесення змін до неї будуть відбуватися швидко та з мінімальними витратами.

Клієнтська частина інформаційної системи має відповідати за обробку дій користувача. У відповідь на введену користувачем інформацію, формується та надсилається запит до серверної частини, відповідь сервера оброблюється та надається користувачу. Тобто клієнтська частина взаємодіє з клієнтом, відображаючи йому необхідну інформацію у зручному та зрозумілому для нього вигляді.

Цей шаблон передбачає поділ системи на три взаємопов'язані частини: модель даних, вигляд (інтерфейс користувача) та модуль керування. Застосовується для відокремлення даних (моделі) від інтерфейсу користувача (вигляду) так, щоб зміни інтерфейсу користувача мінімально впливали на роботу з даними, а зміни в моделі даних могли здійснюватися без змін інтерфейсу користувача.

Мета шаблону – гнучкий дизайн програмного забезпечення, який повинен полегшувати подальші зміни чи розширення програм, а також надавати можливість повторного використання окремих компонентів програми. Крім того використання цього шаблону у великих системах сприяє впорядкованості їхньої структури і робить їх більш зрозумілими за рахунок зменшення складності.

Для виконання етапу кодування було обрано середовище програмування PyCharm [11], у якому буде тривати подальша розробка додатку за допомогою засобів Python, HTML та CSS. PyCharm являє собою інтелектуальний редактор з

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				13
Змн.	Арк.	№ докум.	Підпис	Дата		

можливостями аналізу коду, запобігання помилок у початковому коді і автоматизованими засобами рефакторинга [12].

1.4. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

Веб-додаток буде побудований на клієнт-серверній архітектурі [13]. Серверна частина буде представлена у вигляді API та буде написана з використанням сучасного бекенд фреймворку Django [14]. Для клієнтської частини буде використано HTML, CSS та JavaScript, а як сховище буде використовуватись база даних SQL3.

Django — це високорівневий веб-фреймворк на мові програмування Python, який дозволяє швидко створювати надійні і масштабовані веб-застосунки. Django розроблявся з акцентом на зручність використання, безпеку та масштабованість, що робить його популярним вибором серед розробників веб-застосунків.

Django використовує патерн MVC (Model-View-Controller), але з певними модифікаціями, що більше відповідають його архітектурі. У контексті Django цей патерн називається MTV (Model-Template-View) [15]:

- **Model (Модель):** відповідає за структуру даних та взаємодію з базою даних.
- **Template (Шаблон):** відповідає за презентацію даних, тобто за те, як вони будуть відображатися у вигляді HTML.
- **View (Вид):** обробляє логіку бізнес-процесів та відправляє відповідні дані в шаблони для відображення.

Переваги Django:

- *Швидка розробка:* Django дозволяє розробляти веб-застосунки швидко завдяки своїй структурі та вбудованим інструментам, таким як адміністраторська панель, ORM [16] та багато іншого.
- *Безпека [17]:* Django надає високий рівень безпеки з вбудованим захистом від багатьох поширених атак, таких як SQL-ін'єкції, XSS, CSRF тощо.

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				14
Змн.	Арк.	№ докум.	Підпис	Дата		

- *Масштабованість*: Django легко масштабується для роботи з великими об'ємами даних та високим трафіком завдяки своєму ефективному управлінню ресурсами.
- *Документація та спільнота*: Django має відмінну документацію і велику спільноту розробників, що полегшує вирішення проблем та впровадження нових функцій.
- *Модульність [18]*: Django побудований з використанням модулів, що дозволяє легко розширювати функціональність через підключення сторонніх бібліотек і пакетів.

Недоліки Django:

- *Складність для новачків*: Django може бути складним для новачків через свою комплексність та велику кількість вбудованих функцій.
- *Важкий для невеликих проектів*: використання Django для невеликих проектів може бути надмірним через його розмір та функціональні можливості.
- *Шаблонізація [19]*: Django використовує власний шаблонний мову, яка може бути менш гнучкою порівняно з деякими іншими підходами, такими як React чи Angular.

База даних відіграє важливу роль для кожного сучасного веб-додатку. Завдяки динамічній природі веб-застосунків зараз, навіть найпростіші програми вимагають деяких механізмів зберігання, доступу та зміни даних. СУБД — це програмні та мовні засоби, які виступають посередником між БД та її численними користувачами. Завдяки сукупності мовних та програмних засобів, СУБД сприяють створенню, ведення та спільного використання БД різними користувачами. Сучасна програма СУБД складаються з ядра, процесору мови БД, підсистеми підтримки часу виконання та сервісних програм, які надають додаткові можливості для обслуговування інформаційних систем. Правильний вибір СУБД є одним з важливих етапів при розробці сайту чи додатку баз даних, а також він має задовольняти потреби як теперішні, так і в майбутньому приносячи вигоду.

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				15
Змн.	Арк.	№ докум.	Підпис	Дата		

Важливо також звертати увагу яку структуру (реляційну чи нереляційну) [20] містить обрана СУБД, адже вони мають ряд своїх переваг та особливостей.

У таблиці 1.2 проведено порівняння характеристик БД, що розглядалися.

Таблиця 1.2

Характеристика існуючих СУБД

Характеристика	MS SQL Server	MongoDB	SQLite3
1.	2.	3.	4.
Модель даних	Реляційна	Документно-орієнтована	Реляційна
Адміністративне керування	Добре	Відмінно	Відмінно
Графічні інструменти	Відмінно	Відмінно	Добре
Простота обслуговування	Відмінно	Відмінно	Відмінно
Робота з декількома ЦП	Задовільно	Добре	Відмінно
Підключення до Web	Задовільно	Добре	Відмінно
Одночасний доступ декількох користувачів	Добре	Відмінно	Відмінно
Побудова БД	Добре	Добре	Відмінно
Мова SQL	Добре	Відсутня (нереляційнаСУБД)	Добре
Вбудована мова програмування	Задовільно	Задовільно	Відмінно
Підтримка об'єктно-орієнтованих парадигм	Задовільно	Задовільно	Відмінно

1.	2.	3.	4.
Інтеграція з іншими СУБД	Добре	Добре	Добре
Масштаб застосування	Підприємства малих, середніх та великих розмірів	Підприємства малих, середніх та великих розмірів	Підприємства малих та середніх розмірів

Оскільки важливість баз даних стрімко зростає, реляційні системи управління базами даних або реляційні СУБД набирають своєї популярності. Дві з них є MS SQL Server і SQLite3. Обидві виконують однакову функцію, хоча мають різні варіанти використання. Вони відрізняються деякими особливостями, але обидві системи базуються на SQL або Structured Query Language (структурована мова запитів). У зв'язку з цим, можуть виявити кілька схожостей між ними, таких як використання таблиць для збереження даних, посилання на первинні та зовнішні ключі, так само як кілька баз даних в одному середовищі або на одному сервері. MS SQL Server і SQLite3 базуються на SQL, відмінності синтаксису все ж таки є.

Отже, після порівняння трьох СУБД я обрала SQLite3. Вона зручна для створення динамічних веб-сторінок, оскільки має чудову підтримку з боку різноманітних мов програмування. Характеризується високою швидкістю, стійкістю і простотою використання.

Висновки до розділу 1

В першому підрозділі було розглянуто особливості веб-сайтів, визначено мету та завдання на кваліфікаційну роботу бакалавра. Також було вирішено в якому середовищі програмного забезпечення буде розроблятися сайт та за допомогою яких засобів.

В другому підрозділі було розглянуто то проаналізовано існуючі сайти за тематикою курсового проекту. А саме веб-сайт «Goodreads» та застосунок «Rork». Після чого було визначено особливості та функції власного веб додатку.

У третьому підрозділі було визначено архітектуру веб-орієнтованої системи соціальної мережі та обрано паттерн проектування MTV.

У четвертому підрозділі було сформоване технічне завдання веб додатку, яким вимогам має відповідати функціонал та інтерфейс, а також розглянуто мінімальні системні вимоги для справної роботи сайту.

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				18
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 2. ПРОЕКТУВАННЯ СОЦІАЛЬНО МЕРЕЖІ ЧИТАЧІВ КНИГ

2.1. Визначення варіантів використання та об'єктно-орієнтованої структури системи

У цьому розділі представлено аналіз вимог до програмного продукту з акцентом на виявленні функціональних та нефункціональних вимог. Крім того, визначено вимоги до часових характеристик реакції додатку на дії користувача, а також окреслено різні варіанти використання. Для забезпечення чіткого розуміння структури додатково включено узагальнену діаграму елементів (сутностей системи), їх структури та методів з описом.

Аналіз вимог передбачає докладне вивчення потреб та очікувань користувачів щодо функціональності та характеристик соціальної мережі читачів книг. Основною метою аналізу вимог є зрозуміти, що потрібно втілити в програмному забезпеченні для задоволення потреб користувачів. За допомогою інтерв'ю з користувачами, опитувань та зворотного зв'язку були визначені функціональні вимоги, які окреслюють специфічні особливості та можливості, якими повинен володіти програмний продукт. На додаток, були зібрані нефункціональні вимоги, які стосуються таких аспектів, як продуктивність, стабільність роботи і зручність використання.

Функціональні вимоги [21]:

- Керування профілем користувача: можливість створювати та редагувати профілі користувачів з інформацією про них, зображеннями та іншими персональними даними.
- Додавання та оцінка книг: користувачі можуть додавати книги до своєї бібліотеки, оцінювати їх, писати рецензії та ділитися ними з іншими користувачами.
- Коментування статей та рецензій: можливість коментувати статті, книги та рецензії, що сприяє активній взаємодії між користувачами.
- Створення та управління статтями: користувачі можуть створювати та редагувати статті про книги, авторів та жанри.

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		19

- Форум та обговорення: інтеграція форуму для обговорення книг, авторів та жанрів.
- Пошук та фільтрація: зручні інструменти для пошуку книг, користувачів, рецензій та статей за різними критеріями.
- Сповіщення: система сповіщень про нові коментарі, оцінки, відповіді на коментарі тощо.
- Соціальні функції: можливість підписуватися, слідкувати за їх активністю та ділитися книгами.

Нефункціональні вимоги [22]:

- **Продуктивність:** система повинна швидко реагувати на дії користувача, забезпечуючи безперебійну та чуйну роботу.
- **Ефективне використання ресурсів:** додаток повинен ефективно використовувати пам'ять та процесор, щоб забезпечити швидку та безперебійну роботу.
- **Відновлення стану:** при падінні або несподіваному закритті додатку важливо мати можливість відновити збережені дані та відновити останні дії користувача.
- **Безпека даних:** система повинна забезпечувати безпеку та конфіденційність збережених даних користувачів.
- **Інтуїтивний інтерфейс:** додаток повинен мати зрозумілий та легкий у використанні інтерфейс.

Часові характеристики реакції системи:

- **Час відгуку на дії користувача:** швидкий час відгуку забезпечує миттєву відповідь на дії користувача, що поліпшує загальний досвід роботи з додатком.
- **Час завантаження та збереження даних:** додаток повинен ефективно завантажувати великі обсяги даних та зберігати їх після змін; оптимізація цих процесів допомагає забезпечити швидку обробку даних.

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		20

- Час виконання операцій: деякі операції, такі як фільтри або пошук, можуть займати багато часу, тому швидкість виконання цих операцій впливає на плавність та продуктивність роботи з додатком.
- Відкриття та закриття програми: швидке відкриття програми дозволяє користувачеві швидко почати працювати, а швидке закриття забезпечує збереження часу та енергії користувача.

За допомогою тестування продуктивності визначаються прийнятні часові межі для різних дій та процесів у системі. Ці часові характеристики є важливими для задоволення очікувань користувачів та уникнення розчарування, спричиненого затримками або відсутністю реакції.

Варіанти використання:

Щоб задовольнити різноманітні потреби користувачів, система пропонує різні варіанти використання. Забезпечуючи гнучкість у використанні системи, користувачі можуть адаптувати програмний продукт до своїх конкретних вимог і робочих процесів, підвищуючи свій загальний досвід і продуктивність.

Діаграма використання [23] соціальної мережі читачів книг відображає основні функціональні можливості та взаємодію користувача з системою. Основний актор на діаграмі – зареєстрований користувач, який взаємодіє з системою через різні використовувані випадки використання.

Було створено діаграму варіантів використання (рис. 2.1) - візуальне представлення взаємодії між користувачем системи (актором) і самою системою та специфікацію прецедентів (дод.А).

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		21



Рис. 2.1. Діаграма варіантів використання

Користувач:

- автентифікація у системі;
- вихід з системи;
- перегляд головної сторінки;
- перегляд авторів;
- перегляд жанрів;
- перегляд категорій;
- перегляд статей;
- коментування статей;

		Колісник О.В.		
		Концедайло В. В.		
Змн.	Арк.	№ докум.	Підпис	Дата

- реагування на статтю;
- редагування статей;
- видалення статей;
- перегляд книжок;
- додавання книжок;
- редагування книжок;
- видалення книжок;
- додавання книжок до власної бібліотеки;
- додавання відгуків на книжку;
- редагування своїх відгуків на книжку;
- видалення своїх відгуків на книжку;
- коментування відгуків на книжку;
- налаштовувати свій профіль;
- переглядати профілі інших користувачів;
- підписуватися і відписуватися від інших користувачів;
- переглядати форум;
- переглядати обговорення;
- створювати обговорення;
- редагувати своє обговорення;
- видаляти своє обговорення;
- коментувати обговорення;
- редагувати свої коментарі до обговорення;
- видаляти свої коментарі до обговорення;

2.2. Розробка бази даних системи

Для збереження файлів та інформації у додатку використовуються SQLite, яка базується на реляційній моделі. SQLite [24] - це компактна, самодостатня, вбудовувана система керування базами даних (СКБД), яка дозволяє створювати, управляти та використовувати локальні бази даних без необхідності сервера чи складних конфігурацій.

База даних системи складається з дванадцяти таблиць:

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		23

1. app_users_customuser – таблиця, яка наслідує стандартну модель AbstractUser і зберігає дані користувачів соціальної мережі читачів книг.
2. app_articles_categories – таблиця, яка зберігає дані про категорії статей.
3. app_articles_articles – таблиця, яка зберігає дані про статті.
4. app_articles_commentarticle – таблиця, яка зберігає дані про коментарі.
5. app_books_authors – таблиця, яка зберігає дані про авторів книжок.
6. app_books_genres – таблиця, яка зберігає дані про жанри книжок.
7. app_books_books – таблиця, яка зберігає дані про книжки.
8. app_books_reviews – таблиця, яка зберігає дані про відгуки на книжки.
9. app_books_commentreview – таблиця, яка зберігає дані про коментарі.
10. app_forum_forum – таблиця, яка зберігає дані про форум.
11. app_forum_topic – таблиця, яка зберігає дані про обговорення.
12. app_forum_comment – таблиця, яка зберігає дані про коментарі до обговорень на форумі.

Опис кожної з таблиць бази даних наведений у таблицях 2.1 – 2.12.

Таблиця 2.1

Опис таблиці app_users_customuser

Поле	Тип	Призначення
1.	2.	3.
id	int	Ідентифікатор (первинний ключ)
password	varchar	Пароль
last_login	datetime	Дата останнього входу в систему
is_superuser	boolean	Суперкористувач (адмін)
username	varchar	Ім'я користувача
first_name	varchar	Ім'я
last_name	varchar	Прізвище
email	varchar	Електронна адреса

Продовження таблиці 2.1

1.	2.	3.
is_staff	boolean	Персонал (модератор)
is_active	boolean	Є активним
date_joined	datetime	Дата реєстрації
img	varchar	Посилання на зображення
city	varchar	Місто
date_of_birth	datetime	Дата народження
subscribers	int	Підписники (зовнішній ключ)

Таблиця 2.2

Опис таблиці app_articles_categories

Поле	Тип	Призначення
id	int	Ідентифікатор (первинний ключ)
title	varchar	Назва
img	varchar	Посилання на зображення

Таблиця 2.3

Опис таблиці app_articles_articles

Поле	Тип	Призначення
1.	2.	3.
id	int	Ідентифікатор (первинний ключ)
title	varchar	Назва
description	varchar	Опис
quote	varchar	Цитата
biography	varchar	Біографія
full_name	varchar	Повна назва/ім'я
img	varchar	Посилання на зображення
date_created	datetime	Дата створення

Продовження таблиці 2.3

1.	2.	3.
date_updated	datetime	Дата оновлення
viewscout	int	Кількість переглядів
connected_books	int	Книжка з якою пов'язана стаття (зовнішній ключ)
author_article	int	Автор статті (зовнішній ключ)
category	int	Категорія (зовнішній ключ)

Таблиця 2.4

Опис таблиці app_articles_commentarticle

Поле	Тип	Призначення
id	int	Ідентифікатор (первинний ключ)
text	varchar	Текст коментаря
article	int	Стаття (зовнішній ключ)
created	datetime	Дата створення
updated	datetime	Дата оновлення
likes	int	Користувачі, що вподобали (зовнішній ключ)
dislikes	int	Користувачі, що не вподобали (зовнішній ключ)

Таблиця 2.5

Опис таблиці app_books_authors

Поле	Тип	Призначення
id	int	Ідентифікатор (первинний ключ)
name	varchar	Ім'я автора
surname	varchar	Прізвище автора
biography	varchar	Біографія
img	varchar	Посилання на зображення

Таблиця 2.6

Опис таблиці app_books_genres

Поле	Тип	Призначення
id	int	Ідентифікатор (первинний ключ)
title	varchar	Назва
description	varchar	Опис
img	varchar	Посилання на зображення

Таблиця 2.7

Опис таблиці app_books_books

Поле	Тип	Призначення
1.	2.	3.
id	int	Ідентифікатор (первинний ключ)
title	varchar	Назва
description	varchar	Опис
isbn	varchar	ISBN (International Standard Book Number)
rate	int	Оцінка
img	varchar	Посилання на зображення
img_name	varchar	
date_created	datetime	Дата створення
date_updated	datetime	Дата оновлення
creator	int	Користувач, що додав (зовнішній ключ)
book_author	int	Автор (зовнішній ключ)
genre	int	Жанр (зовнішній ключ)
read_now	int	Користувач читає зараз(зовнішній ключ)
want_to_read	int	Користувач планує прочитати (зовнішній ключ)

Продовження таблиці 2.7

1.	2.	3.
read_books	int	Прочитано користувачем (зовнішній ключ)

Таблиця 2.8

Опис таблиці app_books_reviews

Поле	Тип	Призначення
id	int	Ідентифікатор (первинний ключ)
description	varchar	Текст
created	datetime	Дата створення
updated	datetime	Дата оновлення
book	int	Книжка (зовнішній ключ)
creator	int	користувач, що додав (зовнішній ключ)
likes	int	Користувачі, що вподобали (зовнішній ключ)
dislikes	int	Користувачі, що не вподобали (зовнішній ключ)

Таблиця 2.9

Опис таблиці app_books_commentreview

Поле	Тип	Призначення
1.	2.	3.
id	int	Ідентифікатор (первинний ключ)
body	varchar	Опис
created	datetime	Дата створення
updated	datetime	Дата оновлення

Продовження таблиці 2.9

1.	2.	3.
creator	int	Користувач, що додав (зовнішній ключ)
review	int	Рецензія (зовнішній ключ)

Таблиця 2.10

Опис таблиці app_forum_forum

Поле	Тип	Призначення
id	int	Ідентифікатор (первинний ключ)
title	varchar	Назва
description	varchar	Опис
created	datetime	Дата створення
updated	datetime	Дата оновлення
creator	int	Користувач, що додав (зовнішній ключ)

Таблиця 2.11

Опис таблиці app_forum_topic

Поле	Тип	Призначення
1.	2.	3.
id	int	Ідентифікатор (первинний ключ)
title	varchar	Назва
description	varchar	Опис
created	datetime	Дата створення
updated	datetime	Дата оновлення
is_privat	boolean	Є приватним
closed	boolean	Обговорення закрите

Продовження таблиці 2.11

1.	2.	3.
creator	int	Користувач, що додав (зовнішній ключ)
forum	int	Форум (зовнішній ключ)
allowed_users	int	Допущені користувачі (зовнішній ключ)

Таблиця 2.12

Опис таблиці app_forum_comment

Поле	Тип	Призначення
id	int	Ідентифікатор (первинний ключ)
body	varchar	Опис
created	datetime	Дата створення
updated	datetime	Дата оновлення
creator	int	Користувач, що додав (зовнішній ключ)
topic	int	Обговорення (зовнішній ключ)

Структура бази даних для соціальної мережі читачів книг є добре продуманою та організованою, забезпечуючи функціональність і гнучкість, необхідні для задоволення потреб користувачів. Структура бази даних підтримує різноманітні аспекти соціальної мережі, включаючи статті, книги, відгуки, коментарі та обговорення на форумах, забезпечуючи широкий спектр можливостей для взаємодії користувачів. БД складається з кількох таблиць, кожна з яких відповідає за зберігання специфічних даних, що дозволяє ефективно керувати інформацією та забезпечувати взаємодію між різними аспектами платформи. Поля для збереження зображень дозволяють користувачам завантажувати та зберігати зображення профілів, обкладинки книг та інші графічні елементи, що підвищує візуальну привабливість платформи. Розширена модель користувача

app_users_customuser [25] забезпечує збереження додаткової інформації про користувачів та реалізує функціонал підписки, що є ключовою особливістю соціальних мереж.

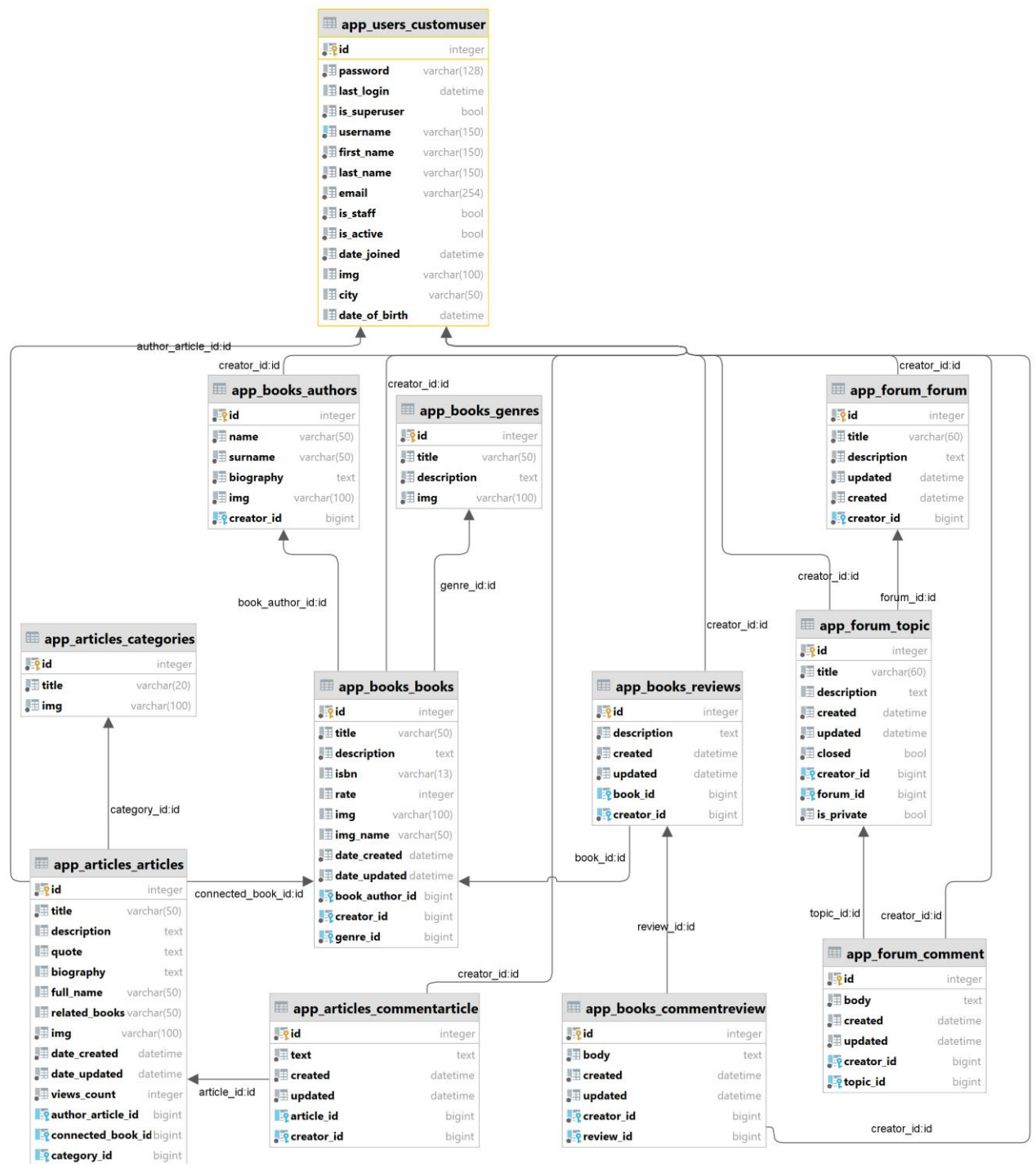


Рис. 2.2. Діаграма БД

2.3. Проектування та реалізація алгоритмів роботи системи

Додаток є простим у використанні, має всі основні функції платформи, що дозволяє користувачам легко взаємодіяти з контентом. Алгоритм роботи з

платформою зображено на рисунку 2.3, також нижче описано детально його пункти.

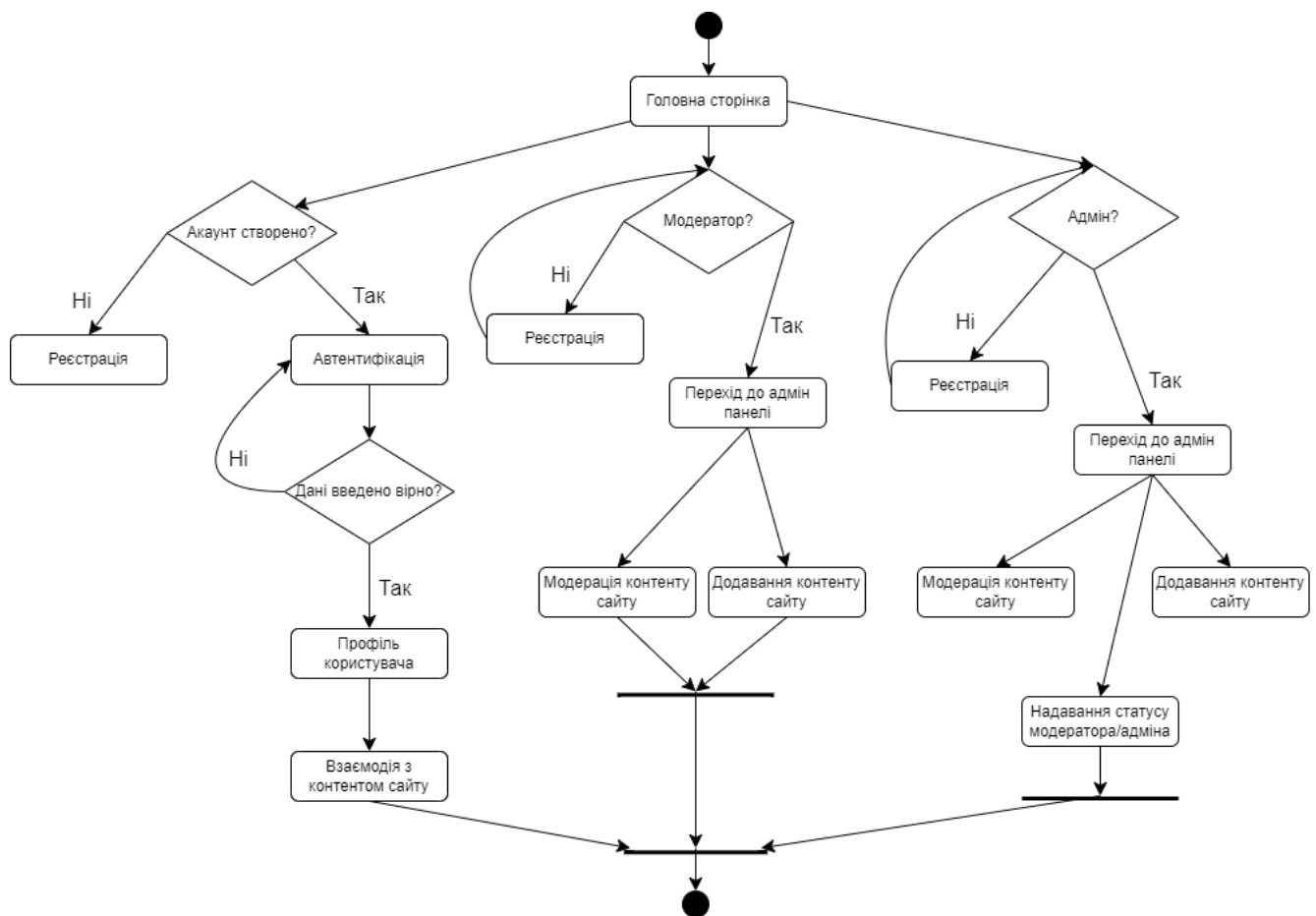


Рис. 2.3. Діаграма активності [26]

Якщо авторизувався адміністратор:

- потрапляє на головну сторінку;
- переходить в панель адміністратора;
- надає користувачам статус модератора або адміністратора.

Якщо авторизувався модератор:

- потрапляє на головну сторінку;
- може додати контент на сайт (категорії статей, жанри книжок, форум);
- може відредагувати контент (категорії статей, жанри книжок, книжки, форум);
- може видалити контент з сайту;

Якщо авторизувався користувач:

- потрапляє на головну сторінку;

- може додати контент на сайт (статті, коментарі до статей, книжки, рецензії, коментарі);
- може поставити реакцію на коментарі та рецензії;
- може відредагувати лише той контент що додав/додала сам/сама.

2.4. Реалізація функціоналу системи соціальної мережі читачів книг

Розглянемо основні етапи додавання контенту на сайт. Першочергово адміністрацією сайту повинно бути додано авторів та жанри книжок. Нижче наведено відповідні методи і класи. Цей блок коду виконується тільки якщо метод HTTP-запиту — POST. Це означає, що дані були відправлені з форми і їх потрібно обробити. Створюється екземпляр форми `AuthorForm`, використовуючи дані, отримані з `request.POST` та `request.FILES`. `request.POST` містить дані, відправлені з форми, а `request.FILES` — файли, відправлені через форму (якщо є). Цей блок коду `if form.is_valid()`: перевіряє, чи є дані форми правильними та відповідають усім вимогам, визначеним у формі. Метод `form.save(commit=False)` зберігає дані форми у змінну `post`, але не зберігає їх у базі даних поки що. Це дозволяє внести додаткові зміни перед остаточним збереженням. Встановлюється поле `creator` для об'єкта `post` як поточний користувач, що виконав запит (`request.user`). Зберігаються дані об'єкта `post` в базу даних. Після успішного збереження об'єкта користувача перенаправляють на сторінку з авторами. Якщо форма не є валідною, у змінну `error` записується повідомлення про помилку, яке потім буде відображено користувачеві. Якщо метод запиту не є POST (наприклад, якщо користувач просто відкриває сторінку додавання автора), створюється новий, порожній екземпляр форми `AuthorForm`. Створюється словник `context`, який містить дані, що будуть передані в шаблон. Він включає форму, поточного користувача, повідомлення про помилку та заголовок сторінки. Нарешті, функція `render` використовується для відображення шаблону `add_book.html` з переданими даними з `context`.

Метод, який відповідає за додавання автора до БД:

```
def add_author(request):
    error = ''
```

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		33

```

if request.method == 'POST':
    form = AuthorForm(request.POST, request.FILES)
    if form.is_valid():
        post = form.save(commit=False)
        post.creator = request.user

        post.save()
        return redirect('authors')
    else:
        print("ERROR!!!\nCAN NOT ADD BOOK!")
        error = 'ERROR!!!\nCAN NOT ADD BOOK!'

form = AuthorForm()
context = {
    'form': form,
    'author': request.user,
    'error': error,
    'title': 'Додати автора'
}
return render(request, 'app_books/add_book.html', context)

```

Метод додавання жанру до БД:

```

def add_genre(request):
    if request.method == 'POST':
        form = GenreForm(request.POST, request.FILES)
        if form.is_valid():
            post = form.save(commit=False)

            post.save()
            return redirect('home')

form = GenreForm()
context = {
    'form': form,
    'title': 'Додати жанр'
}
return render(request, 'app_books/add_book.html', context)

```

Після того як адміністрацією було додано жанри і авторів книжок користувачі при додаванні книжки на сайт можуть зазначити автора і жанр. Нижче наведено лістинг коду відповідного методу. Створюється екземпляр форми BookForm, використовуючи дані, отримані з request.POST та request.FILES.

Після успішного збереження об'єкта користувача перенаправляють на сторінку з книжками. Якщо форма не є валідною, у змінну error записується повідомлення про помилку, яке потім буде відображено користувачеві. Якщо метод запиту не є POST (наприклад, якщо користувач просто відкриває сторінку додавання книжки), створюється новий, порожній екземпляр форми BookForm.

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		34

```

def add_book(request):
    error = ''

    if request.method == 'POST':
        form = BookForm(request.POST, request.FILES)
        if form.is_valid():
            post = form.save(commit=False)
            post.creator = request.user

            post.save()
            return redirect('books')
        else:
            print("ERROR!!!\nCAN NOT ADD BOOK!")
            error = 'ERROR!!!\nCAN NOT ADD BOOK!'

    form = BookForm()
    context = {
        'form': form,
        'author': request.user,
        'error': error,
        'title': 'Додати книгу'
    }
    return render(request, 'app_books/add_book.html', context)

```

Користувачі сайту можуть додати книжки у свою бібліотеку на полиці «Хочу прочитати», «Читаю зараз» та «Прочитано». Цей функціонал реалізований у методах `want_to_read`, `read_now` і `read_books`. Функції приймають два параметри: `request` (об'єкт HTTP-запиту) і `pk` (первинний ключ книги, яку користувач хоче додати до списку). Використовується функція `get_object_or_404` для отримання об'єкта книги з моделі `Books` за заданим первинним ключем (`pk`). Якщо книга з таким первинним ключем не знайдена, повертається сторінка помилки 404. Метод `add` додає поточного користувача (`request.user`) до списку `want_to_read` книги. Передбачається, що `want_to_read` є полем типу `Many-to-Many` у моделі `Books`, яке зберігає користувачів, що бажають прочитати цю книгу. Викликається метод `save`, щоб зберегти зміни у базі даних. Хоча додавання користувача до `Many-to-Many` поля зазвичай автоматично зберігається, це є гарантією збереження змін. Після успішного додавання користувача до списку "Хочу прочитати" функція перенаправляє користувача на сторінку детальної інформації про книгу. Вказаний шлях `'book-detail'` використовує `pk` книги для перенаправлення на відповідну сторінку. Отже, функції забезпечують можливість користувачу додати книгу до

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		35

свого списку "Хочу прочитати". Вона отримує книгу за її первинним ключем, додає поточного користувача до списку бажаючих прочитати, зберігає зміни у базі даних і перенаправляє користувача на сторінку з детальною інформацією про книгу.

Лістинг коду:

```
def want_to_read(request, pk):
    book = get_object_or_404(Books, pk=pk)
    book.want_to_read.add(request.user)
    book.save()
    return redirect('book-detail', pk=pk)

def read_now(request, pk):
    book = get_object_or_404(Books, pk=pk)
    book.read_now.add(request.user)
    book.save()
    return redirect('book-detail', pk=pk)

def read_books(request, user_id):
    books_read =
Books.objects.filter(read_books=user_id).order_by("-id")
    books_read = mk_paginator(request, books_read,
BOOKS_PER_PAGE)
    user = User.objects.get(pk=user_id)

    return render(request,
"app_books/statistics/read_books.html", context={
        'read_books': books_read,
        'user': user,
    })
```

До створеної книжки зареєстрований користувач має змогу додати рецензію. if form.is_valid(): перевіряє, чи є дані форми правильними та відповідають усім вимогам, визначеним у формі. Метод form.save(commit=False) зберігає дані форми у змінну post, але не зберігає їх у базі даних поки що. Це дозволяє внести додаткові зміни перед остаточним збереженням. Встановлюються поля creator та book для об'єкта post. creator визначається як поточний користувач, що виконав запит (request.user), а book встановлюється як поточна книга. Після успішного збереження об'єкта користувача перенаправляють на сторінку детальної інформації про книгу. Використовується функція reverse для отримання URL детальної сторінки книги з передачею параметра book_id. Створюється словник context, який містить дані, що будуть передані в шаблон. Він включає форму та інформацію про

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		36

книгу. Нарешті, функція `render` використовується для відображення шаблону `new_review.html` з переданими даними з `context`.

Відповідний програмний код:

```
def new_review(request, book_id):
    form = ReviewForm()
    book = get_object_or_404(Books, pk=book_id)

    if request.method == 'POST':
        form = ReviewForm(request.POST)

        if form.is_valid():
            post = form.save(commit=False)
            post.creator = request.user
            post.book = book
            post.save()

            return HttpResponseRedirect(reverse('book-detail',
args=(book_id,)))
        context = {
            'form': form,
            'book': book,
        }
    return render(request, 'app_books/new_review.html',
context)
```

Створену рецензію інші користувачі мають змогу вподобати або не вподобати. Цей функціонал реалізований у методах `like_review` та `dislike_review`.

`@login_required` – декоратор забезпечує, що доступ до функції мають лише автентифіковані користувачі. Якщо користувач не увійшов у систему, його буде перенаправлено на сторінку входу. Якщо поточний користувач вже присутній у списку дизлайків (`dislikes`) рецензії, його буде видалено з цього списку. Якщо поточний користувач вже присутній у списку лайків (`likes`) рецензії, його буде видалено з цього списку. Якщо користувач не був у списку лайків, його буде додано до цього списку. Це реалізує механізм перемикавання лайка: якщо лайк вже існує, він знімається, якщо ні — додається. Після виконання всіх змін, користувач перенаправляється на сторінку рецензій. Використовується функція `redirect`, яка перенаправляє користувача на вказану URL-позицію, передаючи параметр `pk` рецензії. Отже, функції забезпечують можливість користувачам лайкати або знімати лайки з рецензій. Спочатку вона перевіряє, чи користувач увійшов у

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		37

систему. Потім, якщо користувач вже поставив дизлайк рецензії, він видаляється зі списку дизлайків. Якщо користувач вже поставив лайк, він видаляється зі списку лайків, інакше додається до нього. Після цього користувач перенаправляється на сторінку з рецензіями.

Лістинг коду:

```
@login_required
def like_review(request, pk):
    review = get_object_or_404(Reviews, pk=pk)
    if request.user in review.dislikes.all():
        review.dislikes.remove(request.user)
    if request.user in review.likes.all():
        review.likes.remove(request.user)
    else:
        review.likes.add(request.user)
    return redirect('reviews', pk=review.pk)
```

```
@login_required
def dislike_review(request, pk):
    review = get_object_or_404(Reviews, pk=pk)
    if request.user in review.likes.all():
        review.likes.remove(request.user)
    if request.user in review.dislikes.all():
        review.dislikes.remove(request.user)
    else:
        review.dislikes.add(request.user)
    return redirect('reviews', pk=review.pk)
```

До створеної рецензії можна додавати коментарі, відповідний метод виводить детальнішу інформацію про відгук на книжку та форму додавання коментаря. У цьому блоці коду використовується фільтрація для отримання рецензій з моделі Reviews, які відповідають заданому ідентифікатору pk. Ініціалізується порожня форма CommentForm для введення коментаря. Отримуються всі коментарі до рецензії, відсортовані за спаданням (нові коментарі першими). Коментарі розбиваються на сторінки за допомогою функції `mk_paginator`, щоб обмежити кількість коментарів, що відображаються на одній сторінці. Якщо метод запиту POST, створюється об'єкт форми CommentForm з даними POST-запиту. Якщо форма дійсна (`form.is_valid()`), створюється новий коментар до рецензії з текстом, взятим з поля `body`, поточним користувачем як творцем і поточною рецензією як зв'язаним об'єктом. Після успішного створення

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		38

коментаря користувач перенаправляється на ту ж сторінку рецензій. Якщо метод запиту не POST, або якщо форма недійсна, відбувається рендеринг сторінки рецензій з використанням шаблону "app_books/review.html".

Програмний код:

```
def review(request, pk):
    reviews = Reviews.objects.filter(pk=pk)
    form = CommentForm()
    comments =
    CommentReview.objects.filter(review_id=pk).order_by("-id")
    comments = mk_paginator(request, comments,
    REVIEWS_COMMENTS_PER_PAGE)

    if request.method == 'POST':
        form = CommentForm(request.POST)
        if form.is_valid():
            CommentReview.objects.create(
                body=request.POST.get('body'),
                creator=request.user,
                review=Reviews.objects.get(id=pk)
            )
            return HttpResponseRedirect(reverse('reviews',
args=(pk,)))

    return render(request, "app_books/review.html", context={
        'form': form,
        'reviews': reviews,
        'comments': comments,
        'user': request.user,
    })
```

Також до створеної книжки можна зареєстрований користувач може додавати статтю свого авторства. Реалізація програмного коду створення статті не має суттєвих відмінностей від реалізації додавання книжки.

Лістинг коду:

```
def create(request):
    error = ''

    if request.method == 'POST':
        form = ArticleForm(request.POST, request.FILES)
        if form.is_valid():
            post = form.save(commit=False)
            post.author_article = request.user

            post.save()
            return redirect('articles')
        else:
            print("ERROR!!!\nCAN NOT CREATE ARTICLE!")
```

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				39
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        error = 'ERROR!!!\nCAN NOT CREATE ARTICLE!'

    form = ArticleForm()
    context = {
        'form': form,
        'author': request.user,
        'error': error,
        'title': 'Створити статтю'
    }
    return render(request, 'app_articles/create.html', context)

```

Статті, додані до бібліотеки книжки користувач може переглянути у своєму профілі. У наведеному нижче методі у змінні записуються дані з відповідних таблиць, а потім передаються у шаблон, де стануть доступні до перегляду на веб-сайті.

Використовуючи модель Articles, отримуються останні 5 статей, написаних користувачем з ідентифікатором pk, сортування відбувається за спаданням (новіші статті першими). Використовуючи модель Books, отримуються останні 5 книг, які користувач читає зараз, а також рахується загальна кількість таких книг. Отримуються останні 5 книг, які користувач хоче прочитати, а також рахується їх загальна кількість. Використовується функція `mk_paginator` для розбиття статей на сторінки відповідно до визначеного параметра `MAIN_TASKS_FOR_PROFILE_PER_PAGE`. Отримуємо об'єкт користувача з моделі User за ідентифікатором pk.

Відбувається рендеринг шаблону "profile.html", де передаються наступні контекстні дані:

- `articles`: статті користувача, розбиті на сторінки.
- `books_read_now`: книги, які користувач читає зараз.
- `books_want_to_read`: книги, які користувач хоче прочитати.
- `read_books`: книги, які користувач вже прочитав.
- `top_5_read_books`: топ-5 прочитаних книг за рейтингом.
- `user`: об'єкт користувача.
- `books_read_now_count`: загальна кількість книг, які користувач читає зараз.

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		40

- books_want_to_read_count: загальна кількість книг, які користувач хоче прочитати.
- read_books_count: загальна кількість прочитаних книг.

Функція UserDetailView обробляє перегляд профілю користувача. Вона отримує останні статті, книги, які користувач читає зараз, хоче прочитати та вже прочитав, а також топ-5 прочитаних книг за рейтингом. Дані розбиваються на сторінки, і всі необхідні контекстні дані передаються в шаблон для рендерингу профілю користувача.

```
def UserDetailView(request, pk):
    articles =
Articles.objects.filter(author_article=pk).order_by("-id")[:5]

    books_read_now =
Books.objects.filter(read_now=pk).order_by("-id")[:5]
    books_read_now_count =
Books.objects.filter(read_now=pk).count()

    books_want_to_read =
Books.objects.filter(want_to_read=pk).order_by("-id")[:5]
    books_want_to_read_count =
Books.objects.filter(want_to_read=pk).count()

    read_books =
Books.objects.filter(read_books=pk).order_by("-id")[:5]
    read_books_count =
Books.objects.filter(read_books=pk).count()

    top_5_read_books =
Books.objects.filter(read_books=pk).order_by("rate")[:5]

    articles = mk_paginator(request, articles,
MAIN_TASKS_FOR_PROFILE_PER_PAGE)
    user = User.objects.get(pk=pk)

    return render(request, "profile.html", context={
        'articles': articles,
        'books_read_now': books_read_now,
        'books_want_to_read': books_want_to_read,
        'read_books': read_books,
        'top_5_read_books': top_5_read_books,
        'user': user,

        'books_read_now_count': books_read_now_count,
        'books_want_to_read_count': books_want_to_read_count,
        'read_books_count': read_books_count
    })
```

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		41

У методі `subscribe_view` реалізовано функціонал підписки на інших користувачів, відписки у `unsubscribe_view`.

Використовуючи функцію `get_object_or_404`, отримується об'єкт користувача з моделі `CustomUser` за ідентифікатором `pk`. Якщо користувача з таким ідентифікатором не знайдено, буде повернута помилка 404 (об'єкт не знайдено). До списку підписників користувача `user` додається поточний користувач, який здійснив запит (отриманий через `request.user`). Зміни (додавання нового підписника) зберігаються в базі даних. Після успішного додавання підписника користувач перенаправляється на сторінку профілю користувача, до якого він підписався. Використовується функція `redirect` для перенаправлення на URL з іменем `'profile'` і переданим параметром `pk`.

Функція `subscribe_view` обробляє підписку поточного користувача на іншого користувача, а `unsubscribe_view` відписку. Вона отримує користувача за переданим ідентифікатором, додає або видаляє поточного користувача до списку підписників цього користувача, зберігає зміни і перенаправляє на сторінку профілю цього користувача.

```
def subscribe_view(request, pk):
    user = get_object_or_404(CustomUser, pk=pk)
    user.subscribers.add(request.user)
    user.save()
    return redirect('profile', pk=pk)

def unsubscribe_view(request, pk):
    user = get_object_or_404(CustomUser, pk=pk)
    user.subscribers.remove(request.user)
    user.save()
    return redirect('profile', pk=pk)
```

Висновки до розділу 2

У розділі 2 було розглянуто та виконано проектування соціальної мережі читачів книг. Було проведено детальний аналіз варіантів використання системи, що дозволило визначити основні функціональні вимоги та сценарії взаємодії користувачів із системою. На основі цього аналізу була розроблена об'єктно-

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				42
Змн.	Арк.	№ докум.	Підпис	Дата		

орієнтована структура системи, що включає ключові класи та їхні взаємозв'язки, забезпечуючи гнучкість та масштабованість системи.

Спроековано та реалізовано структуру бази даних, яка відповідає вимогам зберігання інформації про користувачів, книги, рецензії та взаємодії між користувачами. Забезпечено нормалізацію даних та оптимізацію запитів для забезпечення ефективного та швидкого доступу до інформації.

Розроблено та впроваджено алгоритми, що забезпечують основний функціонал системи, включаючи алгоритми пошуку книг, рекомендацій, обробки рецензій та управління користувачами. Особливу увагу приділено оптимізації алгоритмів для підвищення продуктивності системи та забезпечення комфортного користувацького досвіду.

Виконано інтеграцію розроблених компонентів у єдину систему, що забезпечує повний набір функцій соціальної мережі, включаючи створення та управління профілями користувачів, пошук та додавання друзів, обмін повідомленнями, рецензування книг та рекомендації.

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		43

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ПЛАТФОРМОЮ СОЦІАЛЬНОЇ МЕРЕЖІ ЧИТАЧІВ КНИГ

3.1. Порядок встановлення та налаштування параметрів платформи

Встановлення та налаштування платформи вимагає розгортання всіх його компонентів та налаштування як кожного елементу окремо, так і загальної системи в цілому. Для опису архітектури платформи було створено діаграму розгортання [27] (рис. 3.1).

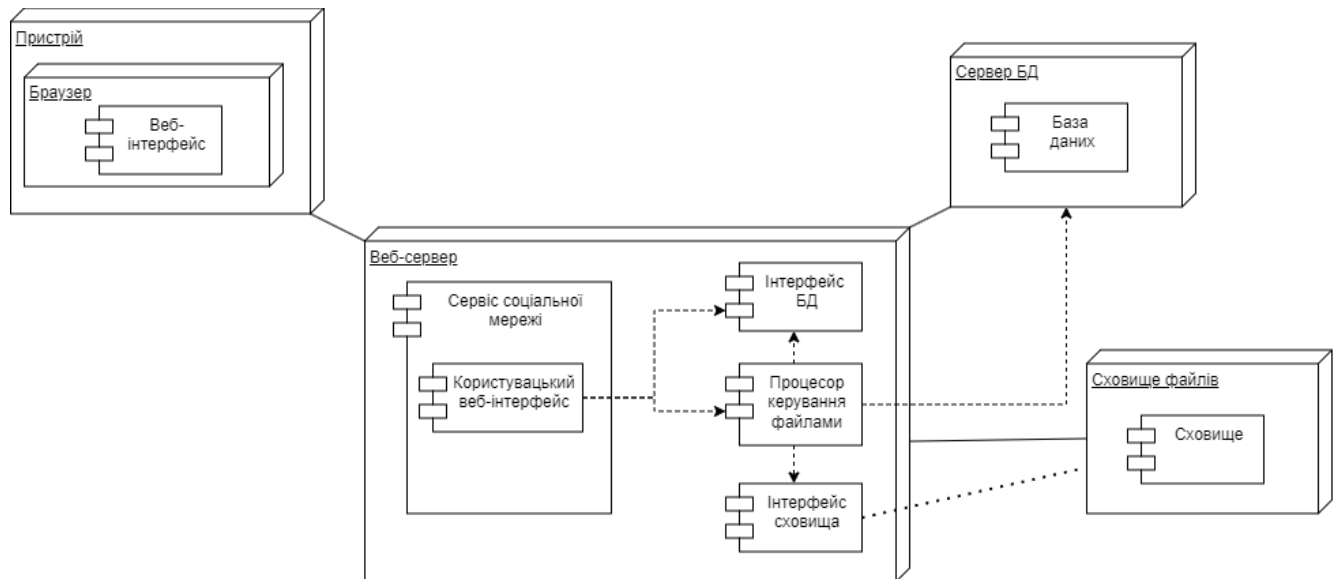


Рис. 3.1. Діаграма розгортання платформи соціальної мережі

Опис процесу розгортання:

1. Пристрій:

– Браузер:

- Веб-інтерфейс: інтерфейс користувача, який забезпечує доступ до сервісу соціальної мережі через браузер. Користувачі взаємодіють із системою через веб-інтерфейс.

2. Веб-сервер:

- Сервіс соціальної мережі: основний компонент, який забезпечує функціонування соціальної мережі, обробку запитів користувачів та управління контентом.

- Користувацький веб-інтерфейс: інтерфейс, через який користувачі взаємодіють із сервісом соціальної мережі. Відповідає за відображення інформації та прийом користувацьких запитів.

3. Сервер БД (База даних):

- База даних: містить всі дані, що зберігаються та використовуються системою, включаючи інформацію про користувачів, книги, рецензії, взаємодії тощо.
- Інтерфейс БД: компонент, який забезпечує зв'язок між веб-сервером та базою даних. Обробляє запити на отримання та зміну даних.

4. Сховище файлів [28]:

- Сховище: компонент, де зберігаються всі файли, які завантажують користувачі (наприклад, зображення обкладинок книг, аватари користувачів тощо).
- Інтерфейс сховища: забезпечує зв'язок між веб-сервером та сховищем файлів, обробляючи запити на зберігання та отримання файлів.

5. Процесор керування файлами: компонент, який відповідає за обробку файлів у системі, включаючи завантаження, зберігання та видачу файлів користувачам. Забезпечує взаємодію між веб-сервером, базою даних та сховищем файлів.

Взаємодії між компонентами:

- Взаємодія між браузером та веб-сервером [29]: користувачі взаємодіють із сервісом соціальної мережі через браузер, надсилаючи запити до веб-сервера, який обробляє ці запити та надсилає відповіді назад у вигляді веб-сторінок.
- Взаємодія між веб-сервером та базою даних [30]: веб-сервер надсилає запити до інтерфейсу БД для отримання або оновлення даних, які зберігаються в базі даних.
- Взаємодія між інтерфейсом БД та процесором керування файлами: інтерфейс БД та процесор керування файлами співпрацюють для

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		45

забезпечення синхронізації даних між базою даних та файлами, які зберігаються в сховищі.

Ця діаграма ілюструє, як компоненти платформи соціальної мережі читачів книг взаємодіють між собою для забезпечення функціональності та обробки запитів користувачів.

3.2. Структура інтерфейсу соціальної мережі читачів книг

Перейдемо до огляду інтерфейсу та робочого процесу соціальної мережі читачів книг. У цьому підрозділі розглянемо різні скріншоти і надамо короткі описи до кожного з них, підкреслюючи ключові особливості та функціональні можливості програмного продукту. Ці скріншоти демонструють елементи користувацького інтерфейсу, з якими користувачі взаємодітимуть під час роботи.

Взаємодія користувачів з платформою відбувається через веб-додаток. На головній сторінці незареєстрований користувач може побачити останні додані книжки, жанри книжок, останні додані статті та категорії статей. Також через навігаційну панель може перейти до усіх авторів, жанрів, книжок, категорій, статей та форуму. Незареєстрований користувач може здійснити пошук контенту на сторінці. На рисунку 3.2 можна побачити вигляд головної сторінки для незареєстрованого користувача.

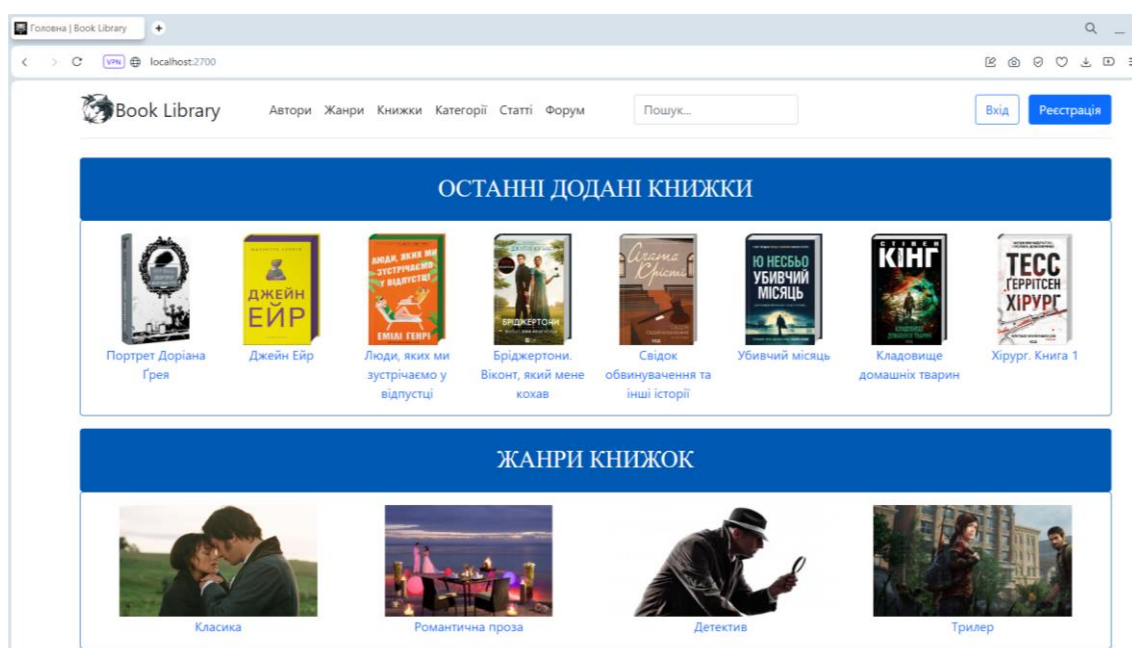


Рис. 3.2. Головна сторінка

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				46
Змн.	Арк.	№ докум.	Підпис	Дата		

Після того як незареєстрований користувач ознайомився з контентом сайту і бажає отримати більше можливостей для взаємодії з наповненням він може перейти до реєстрації (рис. 3.3). На цій сторінці відбувається автоматична валідація форми. Якщо користувача вводять логін або адресу електронної скриньки, що уже наявні у БД, буде виведене відповідне повідомлення, що реєстрація неможлива, оберіть інший логін або адресу електронної скриньки. Якщо пароль, що ввів користувач виявиться ідентичним логіну, або матиме менше 8 символів, або не відповідати значенню введеному у полі «Повторіть пароль», виведеться відповідне повідомлення.

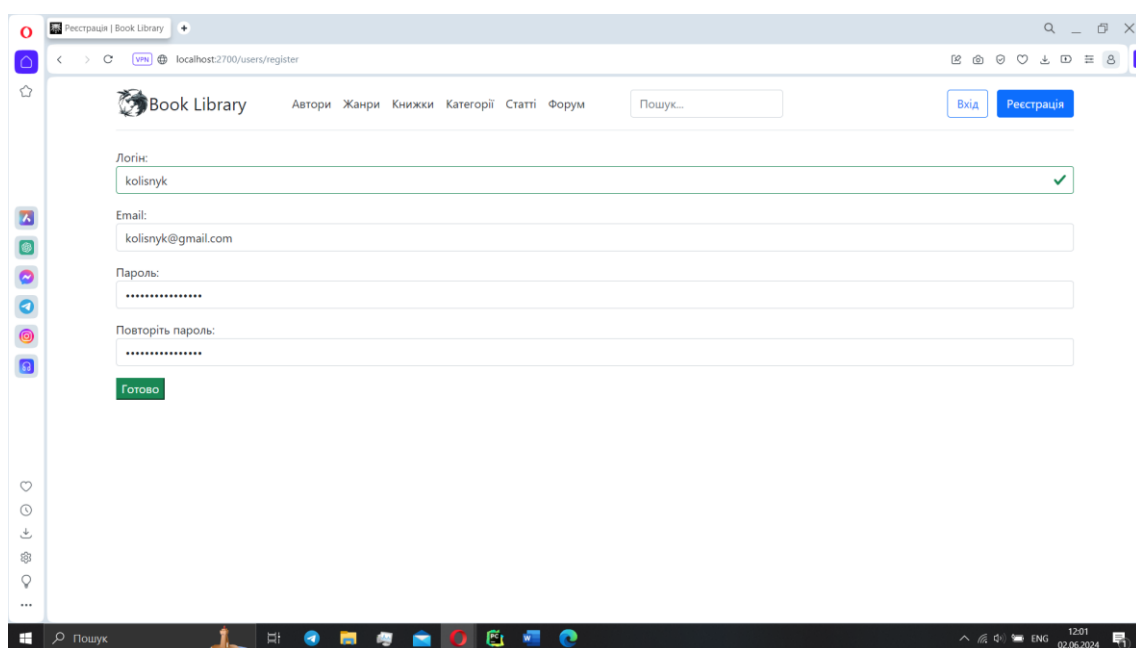


Рис. 3.3. Сторінка реєстрації

Для зареєстрованого користувача доступно набагато більше можливостей у порівнянні з незареєстрованим користувачем. Перш за все автентифікований користувач може перейти до власного профілю (рис 3.4), де виводиться логін користувача, зображення за замовчуванням, дата останнього входу в систему та дата реєстрації на сайті.

Також, можна, натиснувши на кнопку «мої підписки» перейти до сторінки підписок.

На цій сторінці користувач може переглянути написані ним/нею статті, а також книжки, які були додані до своєї бібліотеки та користувачів, на яких було підписано.

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		47

Більш того саме з цієї сторінки можна перейти до редагування профілю. На сторінці редагування профілю відображатиметься форма, де можна змінити особисті дані користувача.

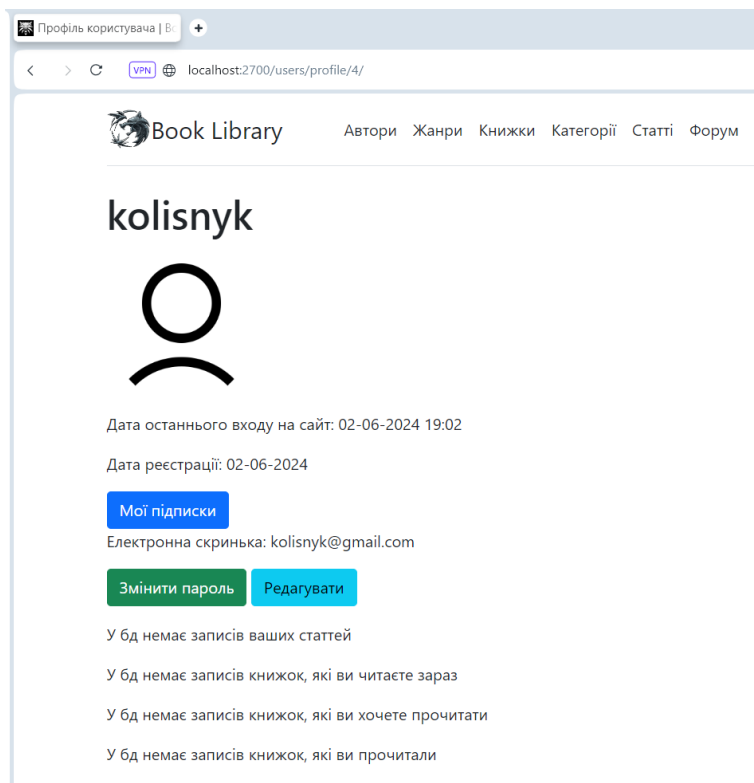


Рис. 3.4. Сторінка профілю користувача

На сторінці перегляду усіх книжок (рис. 3.5) у користувача з'явилася можливість додати свою книжку, але як бачимо можливості редагувати чи видаляти немає. Доступне сортування наявних у БД книжок за такими параметрами:

- новіші (значення за замовчуванням);
- старіші;
- за назвою;
- за автором;
- за оцінкою.

Відповідно до обраного параметру сортування буде викликатися метод, який відповідає за виведення у правильному порядку.

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		48

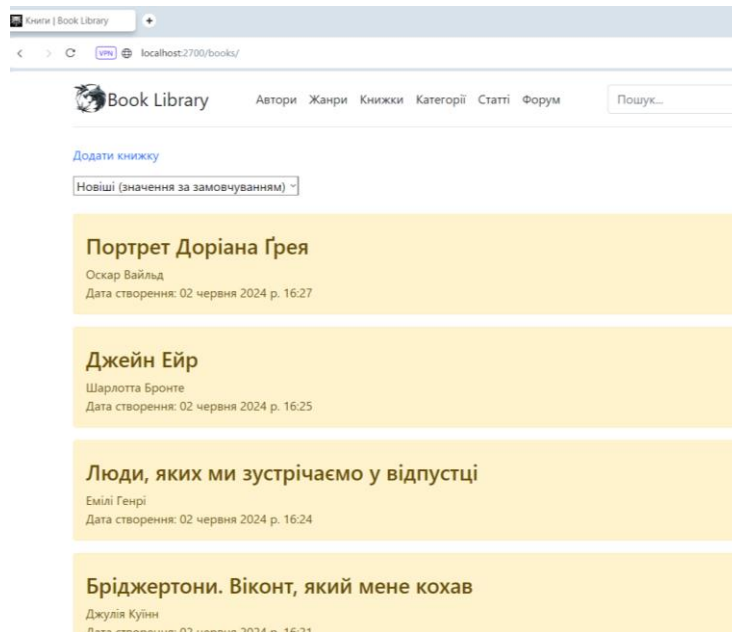


Рис. 3.5. Сторінка перегляду усіх книжок

Для автентифікованого користувача доступний функціонал додавання рецензій та книжок у власну бібліотеку на сторінці деталей книжки (рис. 3.6). На цій сторінці можна переглянути деталі книжки, ознайомитися з профілем користувача, що оформив сторінку, натиснувши на відповідне посилання. Перейти до автора книжки і переглянути усі написані ним книжки та ознайомитися з його біографією. Перейшовши на сторінку перегляду книжки у користувача з'явилася можливість додати книжку до своїх книжкових полиць «Читаю зараз», «Хочу прочитати», «Прочитано». Виконаємо додавання книжки до полиці «Читаю зараз» та поглянемо на результат (рис. 3.7).

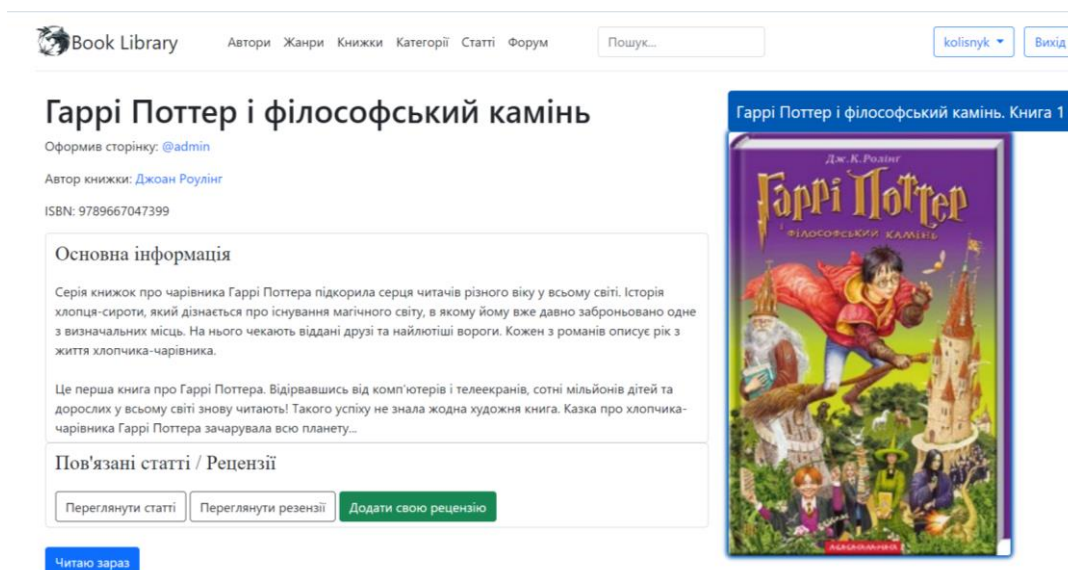


Рис. 3.6. Сторінка перегляду книжки

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				49
Змн.	Арк.	№ докум.	Підпис	Дата		

Переходимо на оновлену сторінку профіля автентифікованого користувача. Можемо відстежити зміни, які відбулися, а саме оновлене фото користувача, повне ім'я, а також додана книжка до полиці «Читаю зараз».

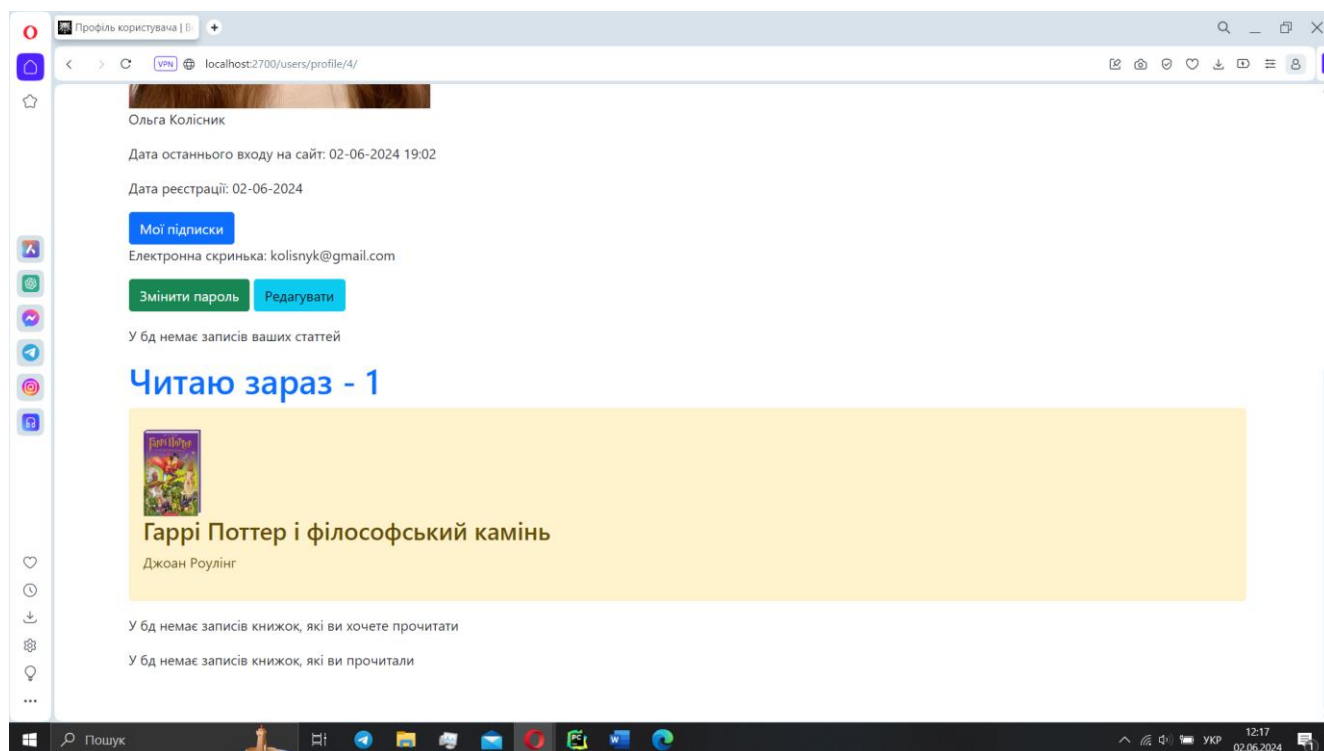


Рис. 3.7. Оновлений профіль користувача

Продовжуємо взаємодіяти з книжками. Тепер користувач може писати відгуки на книжку, заповнивши відповідну форму, і після цього рецензія з'явиться у переліку усіх відгуків, що були написані для цієї книги.

Як бачимо на рисунку 3.8 у користувача є можливість відредагувати і видалити свою статтю, але такої можливості немає для статей іншого автора.

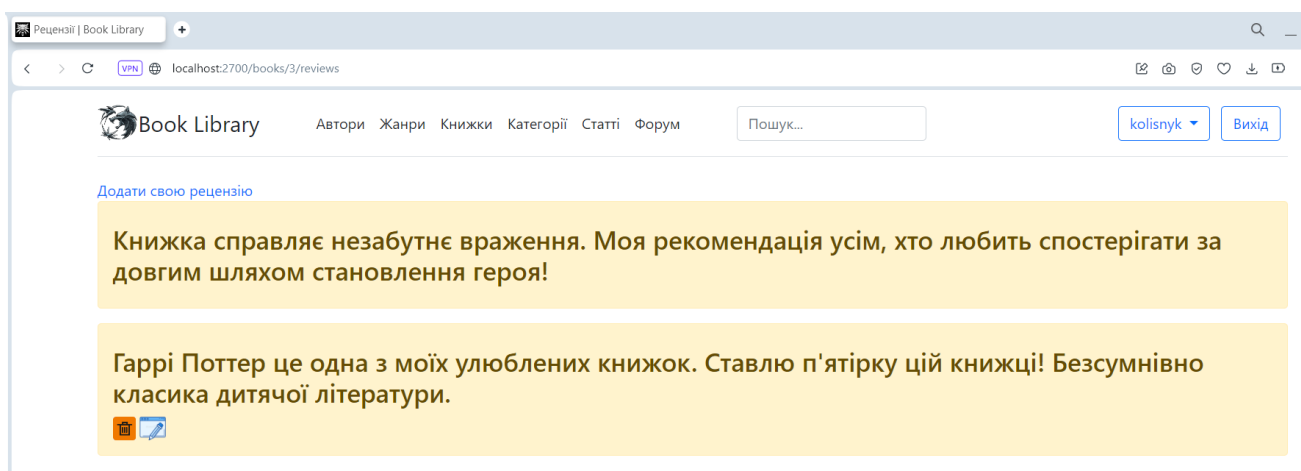


Рис. 3.8. Сторінка перегляду усіх написаних відгуків до книжки

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		50

На рисунку 3.9 показано як виглядає стаття іншого користувача. На цій сторінці авторизований користувач може відреагувати на рецензію і написати свій коментар. Як бачимо коментарі виводять нище поля введення коментаря. Вивід відбувається за спаданням (новіший коментар зноходиться уверху сторінки). Тут можна перейти на профіль користувача – автора коментаря, а також відредагувати та видалити свій коментар за допомогою відповідних кнопок.

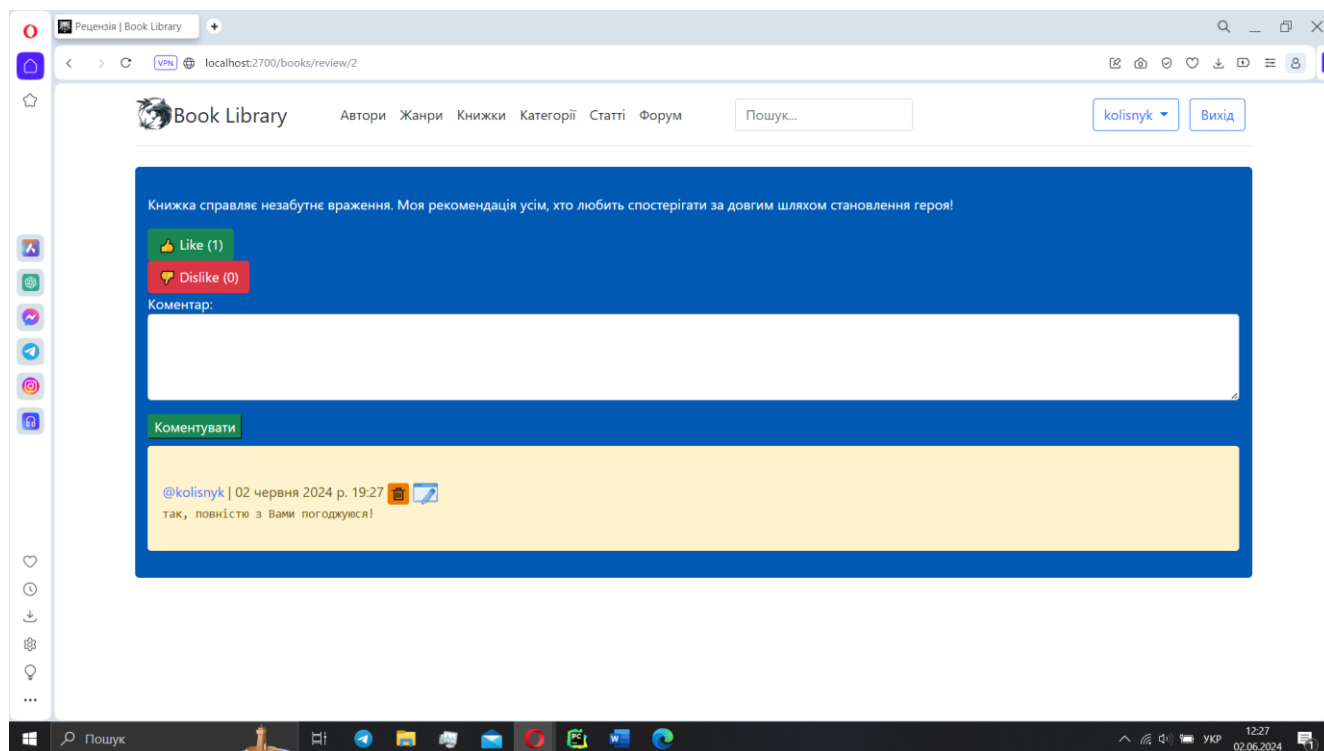


Рис. 3.9. Відображення поставленого «лайку» і щойно написаного коментаря

Повернемося на сторінку, де виведено детальну інформацію про книжку і переглянемо усі написані до неї статті (рис. 3.10). Як бачимо на цій сторінці є можливість переглянути написані іншими користувачами статті та написати свою.

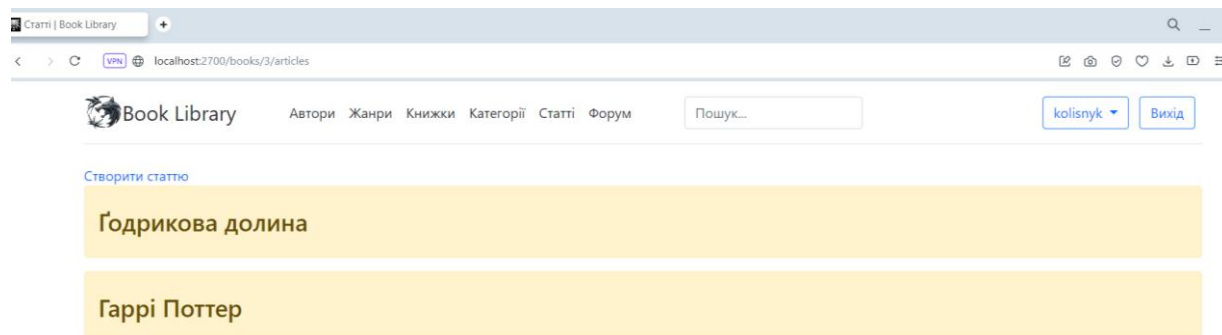


Рис. 3.10. Сторінка написаних статей до книжки «Гаррі Поттер і філософський камінь»

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				51
Змн.	Арк.	№ докум.	Підпис	Дата		

Перейдемо до додавання нової статті. На сторінці, що відкрилася відображаються поля для заповнення відповідно до полів у таблиці БД, яка створювалася. Виконаємо заповнення форми відповідними даними, у полі завантаження картинки, виконаємо завантаження з фізичного носія.

На рисунку 3.11 бачимо у переліку всіх статей до книжки «Гаррі Поттер і філософський камінь», щойно написану статтю, є можливість відредагувати і видалити її, але немає такої можливості до статей іншого авторства.

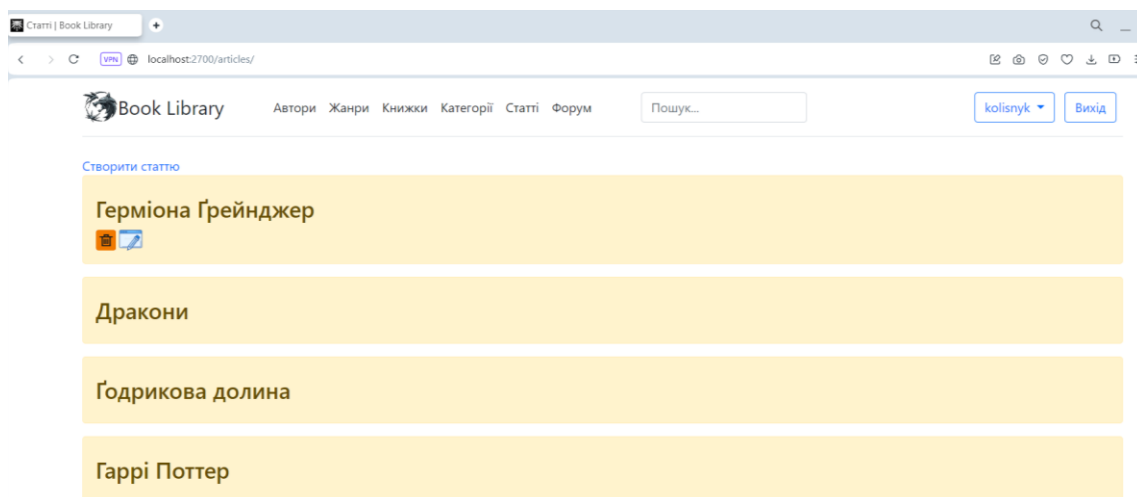


Рис. 3.11. Сторінка перегляду усіх статей до книжки

Переглянемо детальніше як виглядає сторінка статті (рис. 3.12). Як бачимо, вірно відображається автор статті, посилання працює правильно, при натисканні на нього відбувається перехід на профіль автора, і посилання до пов'язаної книжки теж працює правильно.

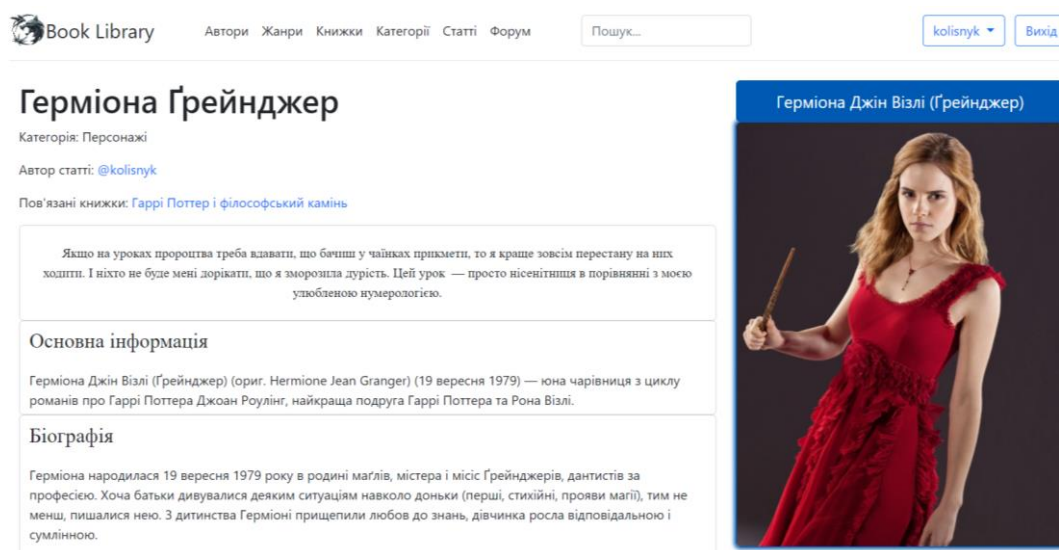


Рис. 3.12. Сторінка написаної статті

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				52
Змн.	Арк.	№ докум.	Підпис	Дата		

На сторінці, що зображена на рисунку 3.12 є можливість додати коментар (рис. 3.13). Як бачимо коментарі виводять нище поля введення коментаря. Вивід відбувається за спаданням (новіший коментар зноходиться уверху сторінки). Тут можна перейти на профіль користувача – автора коментаря, а також відредагувати та видалити свій коментар за допомогою відповідних кнопок. А також на коментар можна поставити реакцію у вигляді кнопок «Лайк» та «Дизлайк».

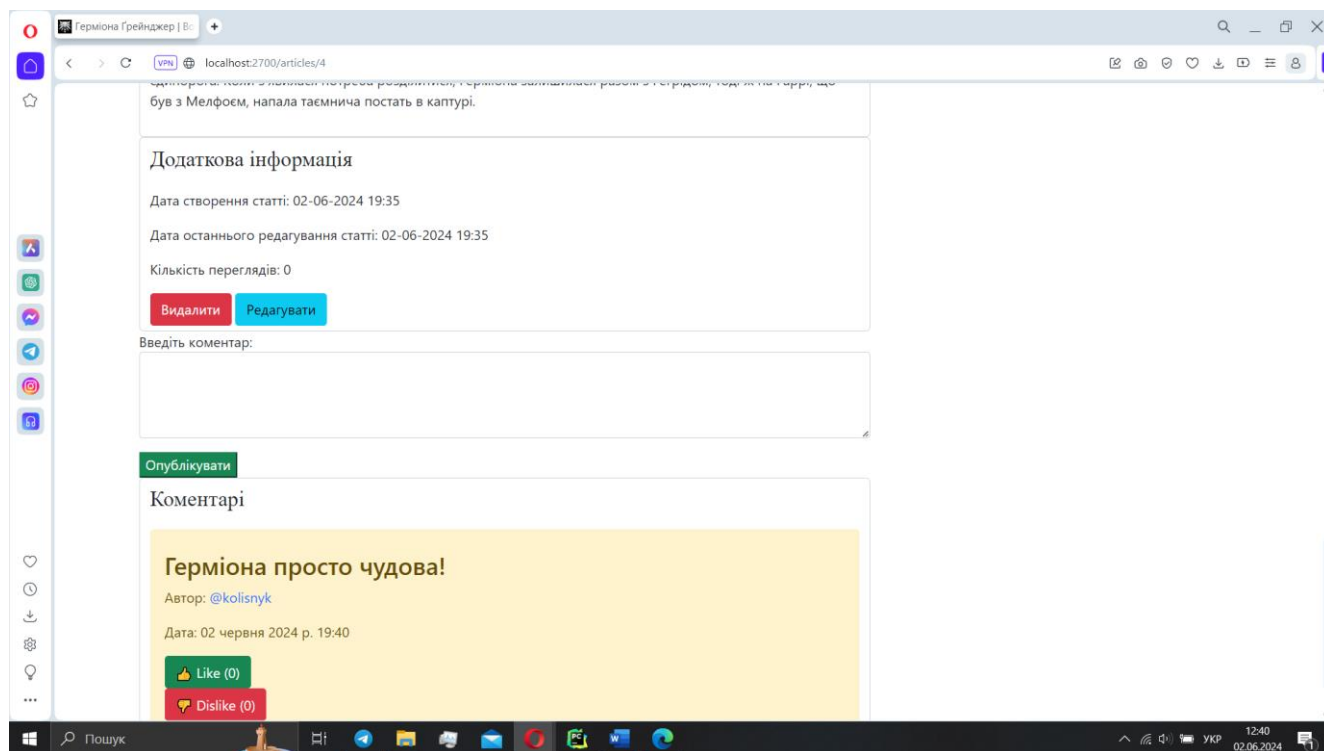


Рис. 3.13. Сторінка написаної статті (доданий коментар)

Тепер розглянемо сторінку іншого користувача (рис. 3.14). Як бачимо там є кнопка «Підписатися», виконаємо підписку, звернемо увагу на зміни на сторінці іншого користувача і переглянемо результат доступний після переходу за посиланням на кнопці «Мої підписки» у профілі користувача kolisnyk. Там відображається актуальна кількість підписників та список користувачів, що підписалися. Кожне ім'я користувача є посиланням, що веде на профіль обраного користувача.

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				53
Змн.	Арк.	№ докум.	Підпис	Дата		

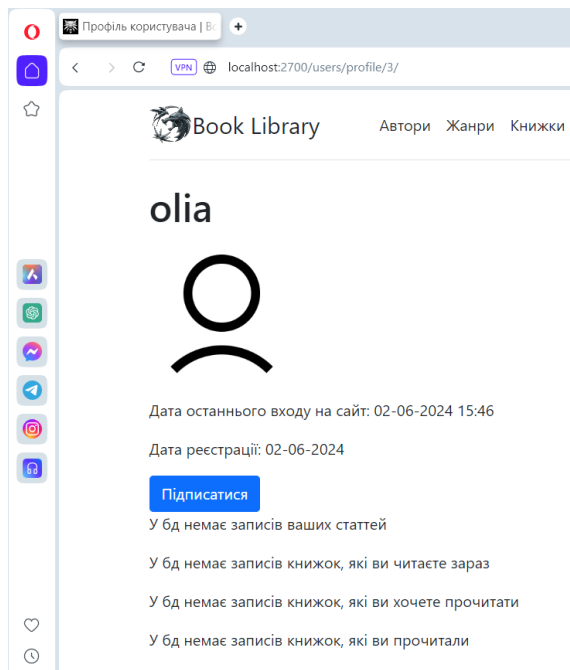


Рис. 3.14. Сторінка іншого користувача

Виконаємо вхід на уже створений акаунт модератора. На рисунку 3.15 у меню бачимо перелік дій які може виконати модератор. А також те, що з'явилися посилання над блоками для додавання контенту.

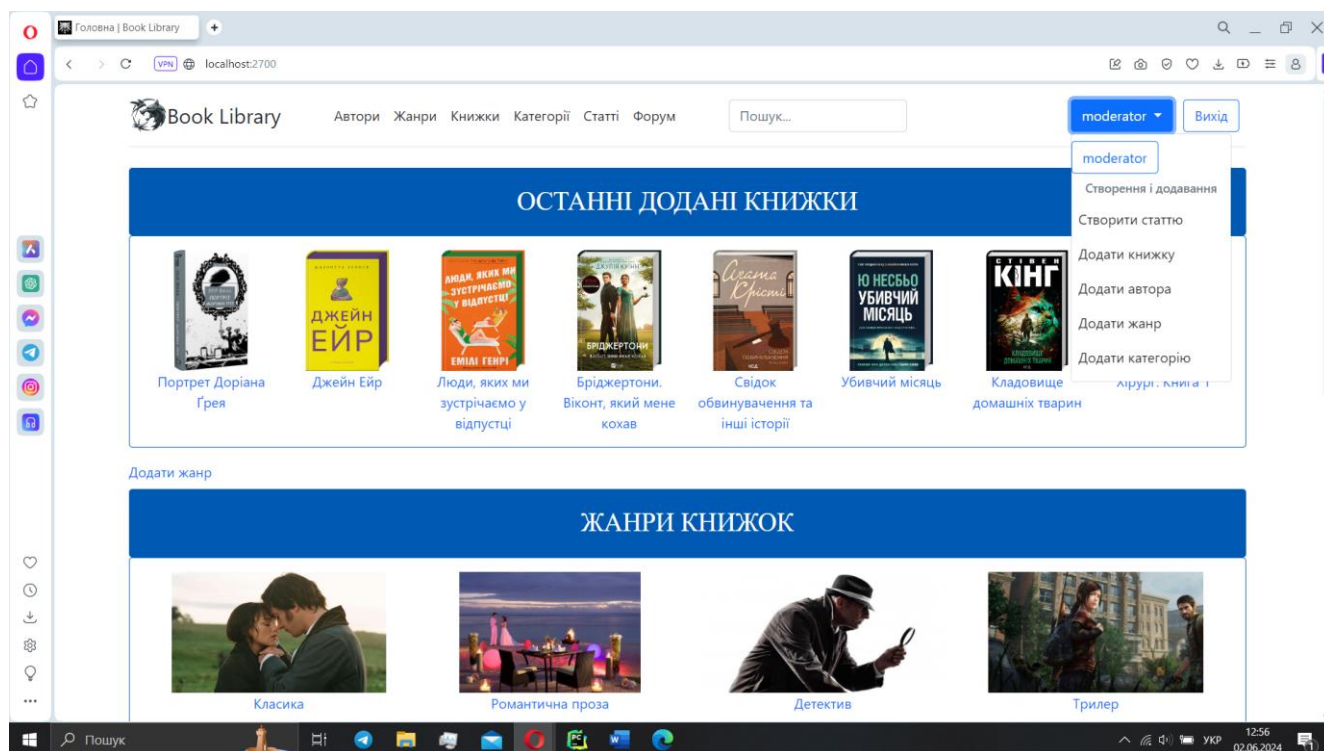


Рис. 3.15. Головна сторінка очима модератора

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				54
Змн.	Арк.	№ докум.	Підпис	Дата		

На сторінці авторів у модератора є можливість додати, відредагувати і видалити автора раніше створеного іншим користувачем з роллю модератора або адміністратора. Виконаємо додавання автора, заповнивши відповідну форму

Переглянемо усіх авторів. Кожне поле є доступним для редагування і видалення автентифікованим користувачем з роллю модератор. Виконаємо редагування даних автора, заповнивши форму відповідними даними та переглянемо результат дій користувача з роллю модератор.

Переглянемо як виглядає оновлена сторінка (рис. 3.16).

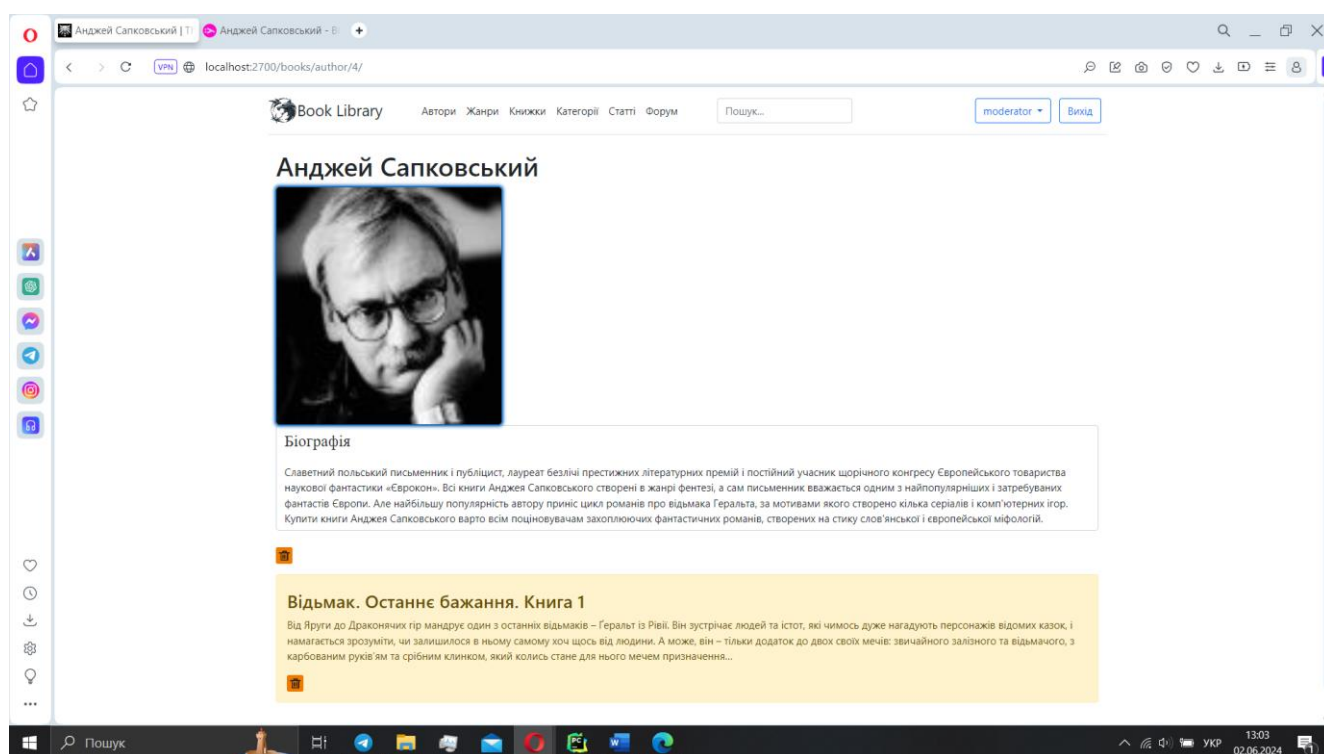


Рис. 3.16. Оновлена сторінка інформації про автора

Аналогічно до функціоналу взаємодії з інформацією про авторів реалізовано можливості додавання, редагування та видалення контенту жанрів (рис. 3.17), книжок, категорій, і форуму. Проте, існують обмеження стосовно взаємодії з контентом статей, оскільки вони є предметом інтелектуальної власності у модератора є можливість лише видалити цей допис, але не редагувати.

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				55
Змн.	Арк.	№ докум.	Підпис	Дата		

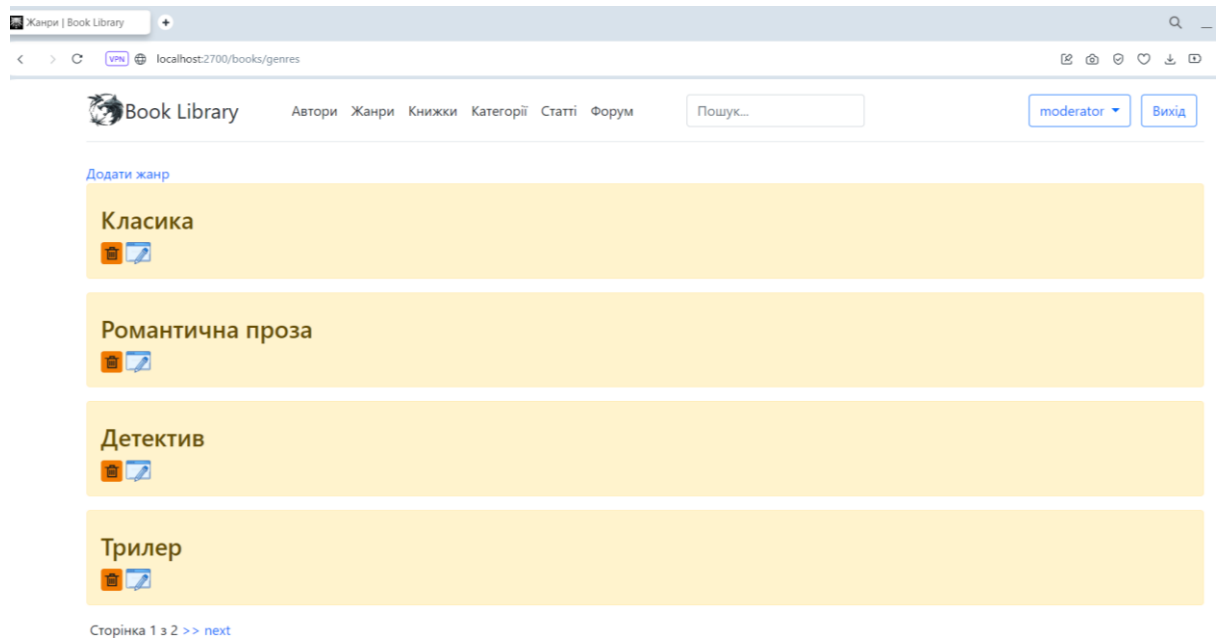


Рис. 3.17. Сторінка перегляду усіх жанрів

Виконаємо вхід в уже створений акаунт адміністратора (рис. 3.18).

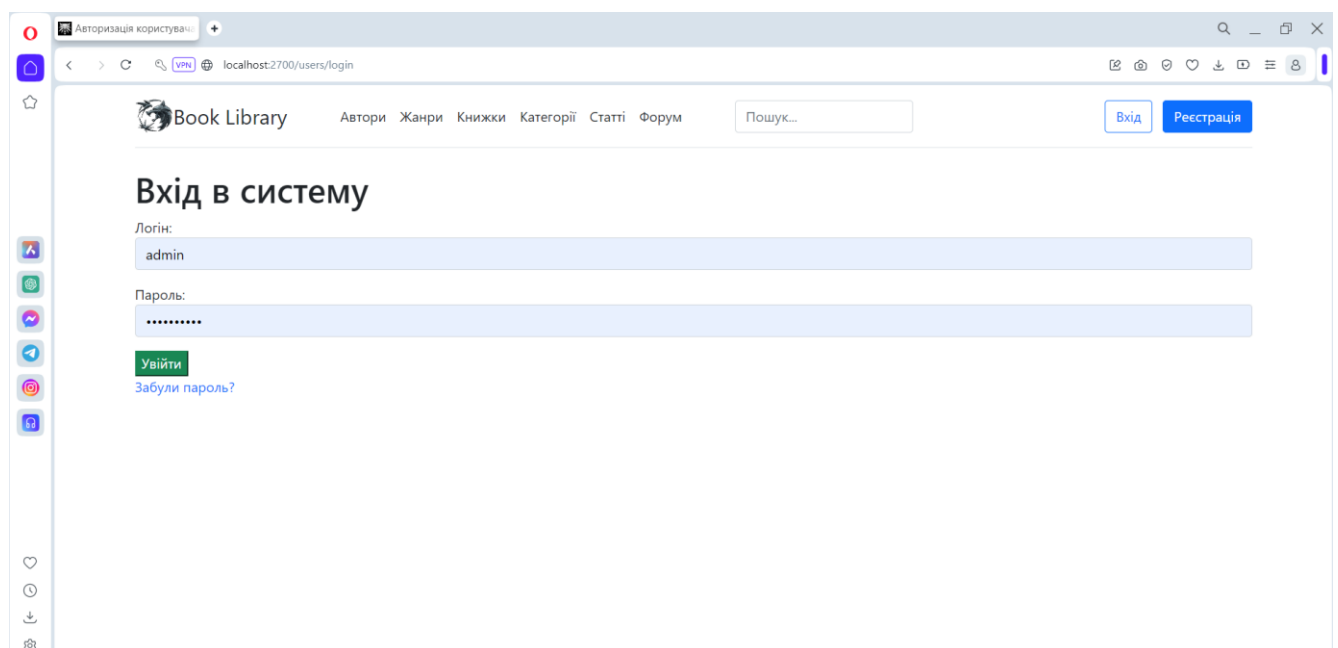


Рис. 3.18. Автентифікація в системі

На сайті адміністратор має ті ж можливості, що і зареєстрований користувач і модератор. Відмінністю можливостей адміністратора є саме перехід до панелі адміністратора (рис. 3.19), де є можливість переглянути останній дії на сайті і безпосередньо керувати контентом сайту, та переглядати статистику по сайті.

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				56
Змн.	Арк.	№ докум.	Підпис	Дата		

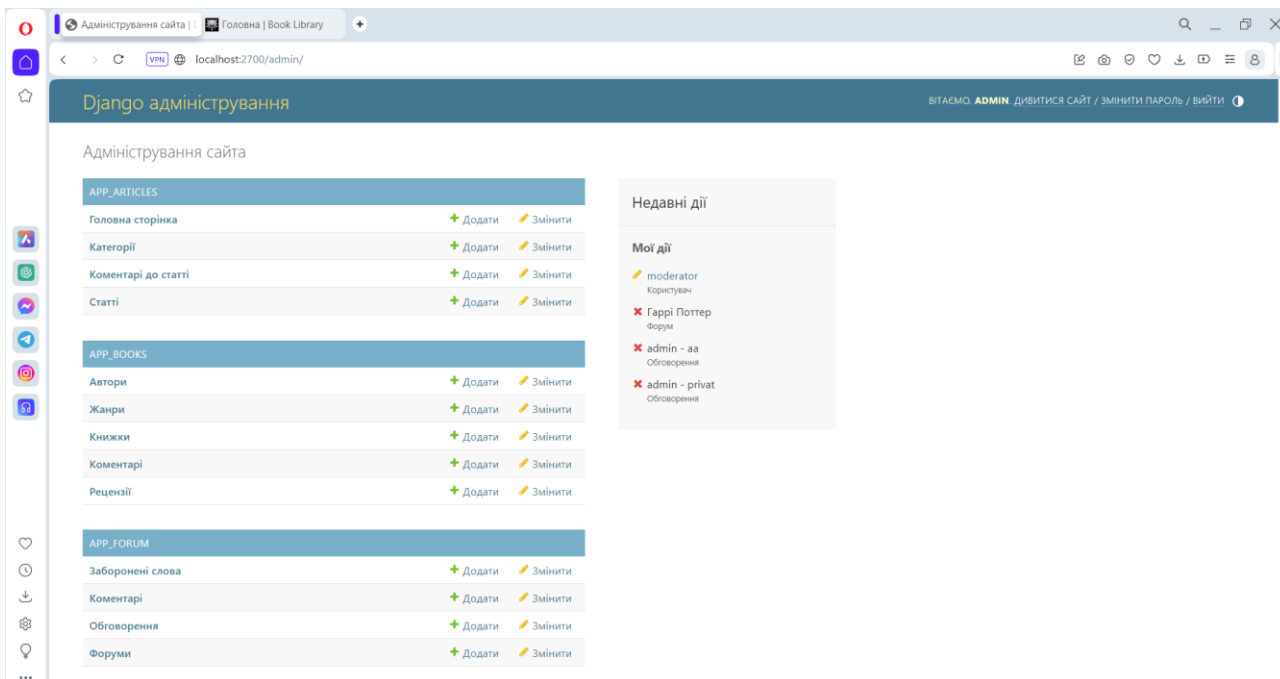


Рис. 3.19. Панель адміністратора

Перейдемо до вкладки користувачі та надамо користувачу kolisnyk роль модератора (рис. 3.20).

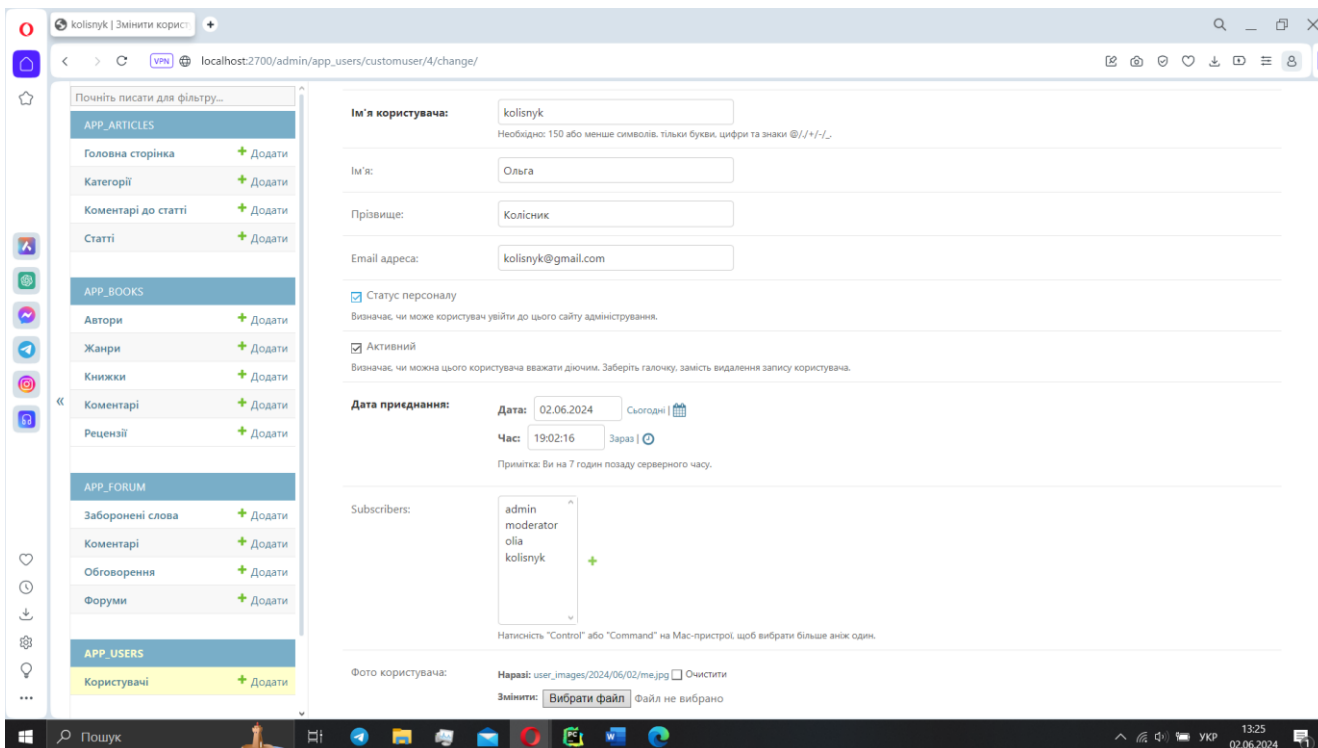


Рис. 3.20. Панель адміністратора. Редагування інформації про користувача kolisnyk

На приклад сторінки авторів бачимо, що у користувача kolisnyk з'явилася можливість редагувати і видаляти контент (рис. 3.21)

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		57

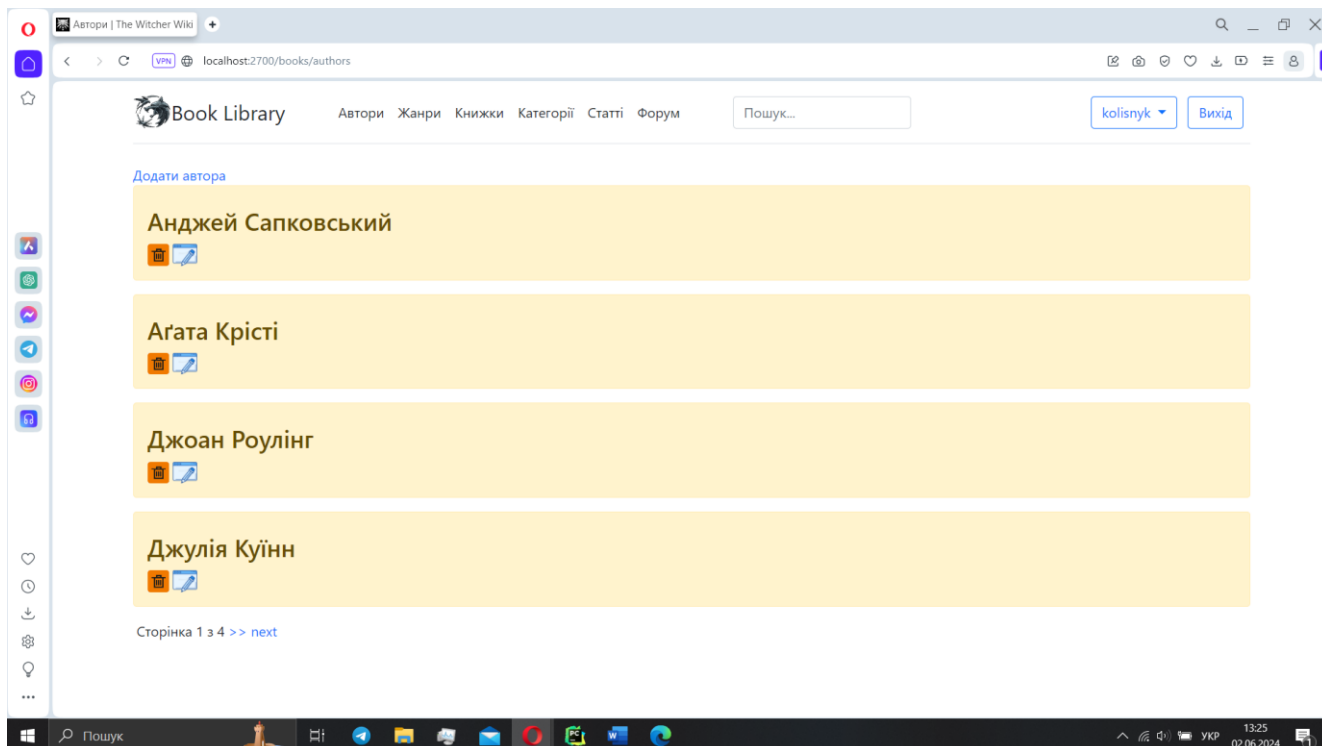


Рис. 3.21. Сторінка авторів

Висновки до розділу 3

У розділі 3 було детально розглянуто інтерфейс та порядок роботи з платформою соціальної мережі читачів книг. Описано покроковий процес встановлення платформи на різні операційні системи. Цей процес включає в себе налаштування необхідних параметрів для забезпечення оптимальної роботи системи. Розглянуто базові конфігурації та специфічні налаштування, що дозволяють адаптувати платформу під конкретні потреби користувачів та адміністраторів.

Представлено детальний опис структури інтерфейсу користувача, включаючи основні елементи, такі як головна сторінка, профілі користувачів, бібліотека книг, система рекомендацій, функції спілкування та інші інтерактивні компоненти.

Пояснено принципи дизайну та ергономіки, які використовуються для забезпечення зручності та інтуїтивної зрозумілості інтерфейсу, що сприяє позитивному користувацькому досвіду.

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		58

Проведено всебічне тестування системи, включаючи функціональне, інтеграційне та користувацьке тестування. Це дозволило виявити та виправити можливі помилки та недоліки в роботі платформи.

Результати тестування підтвердили стабільність та надійність роботи системи, що є критично важливим для забезпечення безперервного та комфортного користування платформою.

Результати розділу 3 підтверджують, що платформа соціальної мережі читачів книг є функціональною, зручною для користувачів та готовою до широкого впровадження. Встановлення та налаштування платформи є простим та зрозумілим процесом, а інтерфейс забезпечує інтуїтивну взаємодію з користувачем. Проведене тестування гарантує надійність та ефективність роботи системи, що сприяє залученню нових користувачів та розширенню спільноти читачів книг.

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		59

ВИСНОВКИ

У ході виконання кваліфікаційної роботи бакалавра було проведено всебічне дослідження, проектування та розробку платформи соціальної мережі читачів книг.

У першому розділі було розглянуто особливості веб-сайтів, визначено мету та завдання роботи. Було обрано середовище програмного забезпечення для розробки сайту та засоби реалізації. В другому підрозділі проаналізовано існуючі веб-сайти, такі як «Goodreads» та застосунок «Rork», що дозволило визначити особливості та функції власного веб-додатку. У третьому підрозділі визначено архітектуру веб-орієнтованої системи соціальної мережі та обрано паттерн проектування MTV. У четвертому підрозділі сформовано технічне завдання веб-додатку, окреслено вимоги до функціоналу та інтерфейсу, а також розглянуто мінімальні системні вимоги для справної роботи сайту.

Другий розділ присвячений детальному проектуванню соціальної мережі читачів книг. Було проведено аналіз варіантів використання системи, що дозволило визначити основні функціональні вимоги та сценарії взаємодії користувачів із системою. Розроблено об'єктно-орієнтовану структуру системи, включаючи ключові класи та їхні взаємозв'язки. Спроектовано та реалізовано структуру бази даних для зберігання інформації про користувачів, книги, рецензії та взаємодії між користувачами, забезпечивши нормалізацію даних та оптимізацію запитів. Розроблено та впроваджено алгоритми для основного функціоналу системи, зокрема пошуку книг, рекомендацій, обробки рецензій та управління користувачами. Проведено інтеграцію розроблених компонентів у єдину систему, забезпечуючи створення та управління профілями користувачів, пошук та додавання друзів, обмін повідомленнями, рецензування книг та рекомендації.

У розділі 3 було детально розглянуто інтерфейс та порядок роботи з платформою соціальної мережі читачів книг. Описано покроковий процес встановлення платформи на різні операційні системи, включаючи налаштування необхідних параметрів. Розглянуто базові конфігурації та специфічні налаштування для адаптації платформи під конкретні потреби користувачів та адміністраторів. Представлено структуру інтерфейсу користувача, включаючи

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		60

головну сторінку, профілі користувачів, бібліотеку книг, систему рекомендацій, функції спілкування та інші інтерактивні компоненти. Проведено всебічне тестування системи, включаючи функціональне, інтеграційне та користувацьке тестування, що дозволило виявити та виправити можливі помилки та недоліки. Результати тестування підтвердили стабільність та надійність роботи системи, що є критично важливим для забезпечення безперервного та комфортного користування платформою.

Загалом, виконана робота підтверджує, що платформа соціальної мережі читачів книг є функціональною, зручною для користувачів та готовою до широкого впровадження. Встановлення та налаштування платформи є простим та зрозумілим процесом, а інтерфейс забезпечує інтуїтивну взаємодію з користувачем. Проведене тестування гарантує надійність та ефективність роботи системи, що сприяє залученню нових користувачів та розширенню спільноти читачів книг.

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		61

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Setting Up a Database in Django [Електронний ресурс] – Режим доступу до ресурсу: <https://reintech.io/blog/setting-up-a-database-in-django>
2. What is a content management system (CMS)? [Електронний ресурс] – Режим доступу до ресурсу: <https://kinsta.com/knowledgebase/content-management-system/>
3. UI/UX дизайн: принципи та методи створення зручного інтерфейсу користувача. [Електронний ресурс] – Режим доступу до ресурсу: <https://whileweb.com/uk/blog/uiux-dizajn-principi-ta-metodi-stvorenniya-zruchnogo-interfejsu-koristuvacha/>
4. Архітектура веб додатків. [Електронний ресурс] – Режим доступу до ресурсу: <https://medium.com/@IvanZmerzlyi/apxitektyra-veb-dodatki-4c82f75bcf>
5. Django Tutorial Part 8: User authentication and permissions. [Електронний ресурс] – Режим доступу до ресурсу: <https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Authentication>
6. Види функціонального та нефункціонального тестування. [Електронний ресурс] – Режим доступу до ресурсу: <https://dan-it.com.ua/uk/blog/vidy-funkcionalnogo-i-nefunkcionalnogo-testirovaniya/>
7. Goodreads. [Електронний ресурс] – Режим доступу до ресурсу: <https://www.goodreads.com>
8. Rork — мистецтво читати. [Електронний ресурс] – Режим доступу до ресурсу: <https://rork.ua>
9. Django Architecture – Detailed Explanation. [Електронний ресурс] – Режим доступу до ресурсу: <https://www.interviewbit.com/blog/django-architecture/>
10. Каталог патернів проектування. [Електронний ресурс] – Режим доступу до ресурсу: <https://refactoring.guru/uk/design-patterns/catalog>
11. PyCharm Professional for Web Development. [Електронний ресурс] – Режим доступу до ресурсу: <https://www.jetbrains.com/pycharm/web-development/>

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		62

- 12.Code Refactoring Best Practices – with Python Examples. [Електронний ресурс] – Режим доступу до ресурсу: <https://www.freecodecamp.org/news/best-practices-for-refactoring-code/>
- 13.Клієнт-серверна архітектура. [Електронний ресурс] – Режим доступу до ресурсу: <https://training.qatestlab.com/blog/technical-articles/client-server-architecture/>
- 14.Django Documentation. [Електронний ресурс] – Режим доступу до ресурсу: <https://www.djangoproject.com>
- 15.Working Structure of Django MTV Architecture. [Електронний ресурс] – Режим доступу до ресурсу: <https://towardsdatascience.com/working-structure-of-django-mtv-architecture-a741c8c64082>
16. Оптимізація роботи з базами даних за допомогою ORM (Object-Relational Mapping). [Електронний ресурс] – Режим доступу до ресурсу: <https://it-rating.ua/optimizatsiya-roboti-z-bazami-danih-za-dopomogoyu-orm-object-relational-mapping>
- 17.Web Security in Django – How to Build a Secure Web Application. [Електронний ресурс] – Режим доступу до ресурсу: <https://www.freecodecamp.org/news/how-to-build-a-secure-django-web-app/>
- 18.Introduction to Django ORM Module. [Електронний ресурс] – Режим доступу до ресурсу: <https://www.almabetter.com/bytes/tutorials/django/django-orm>
- 19.Django Template Tags. [Електронний ресурс] – Режим доступу до ресурсу: https://www.w3schools.com/django/django_template_tags.php
- 20.SQL чи NoSQL – ось в чому питання. [Електронний ресурс] – Режим доступу до ресурсу: <https://alternativescience.net/programming/242-sql-chy-nosql-os-v-chomu-pytannya/>
- 21.Що таке функціональні вимоги: приклади, визначення, повний посібник. [Електронний ресурс] – Режим доступу до ресурсу: <https://visuresolutions.com/uk/blog/functional-requirements/>

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		63

- 22.Бізнес-аналітики і нефункціональні вимоги в agile розробці. [Електронний ресурс] – Режим доступу до ресурсу: <https://www.ba.in.ua/2023/08/18/biznes-analityky-i-nefunkczionalni-vymogy-v-agile-rozrobchi/>
- 23.Use Case Diagrams | Unified Modeling Language (UML). [Електронний ресурс] – Режим доступу до ресурсу: <https://www.geeksforgeeks.org/use-case-diagram/>
- 24.SQLite Documentation. [Електронний ресурс] – Режим доступу до ресурсу: <https://www.sqlite.org/docs.html>
- 25.Creating a Custom User Model in Django. [Електронний ресурс] – Режим доступу до ресурсу: <https://testdriven.io/blog/django-custom-user-model/>
- 26.Activity Diagram. [Електронний ресурс] – Режим доступу до ресурсу: https://sparxsystems.com/enterprise_architect_user_guide/16.1/modeling_languages/activitydiagram.html
27. Діаграма розгортання: Підручник з UML із ПРИКЛАДОМ. [Електронний ресурс] – Режим доступу до ресурсу: <https://www.guru99.com/uk/deployment-diagram-uml-example.html>
28. Що таке хмарні сховища та навіщо вони потрібні? [Електронний ресурс] – Режим доступу до ресурсу: <https://vistplus.com/novini/shho-take-hmarni-shovishha-ta-navishho-voni-potribni/>
- 29.Що таке веб-сервер і для чого він потрібний? [Електронний ресурс] – Режим доступу до ресурсу: <https://introserv.com/ua/blog/scho-take-veb-server-i-dlya-chogo-vin-potribnij/>
- 30.Веб -сервер та база даних. [Електронний ресурс] – Режим доступу до ресурсу: <https://sites.google.com/view/onlayn-konspekt-z-informatiku/11-klas/urok-26-veb-server-ta-baza-danih>

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		64

ДОДАТКИ

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		65

DjangoProject

settings.py

```
import os
from pathlib import Path

BASE_DIR = Path(__file__).resolve().parent.parent

SECRET_KEY = '1111'

DEBUG = True

ALLOWED_HOSTS = []

# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'app_articles',
    'app_users',
    'app_forum',
    'app_books',
]

AUTH_USER_MODEL = 'app_users.CustomUser'

DBBACKUP_STORAGE = 'django.core.files.storage.FileSystemStorage'
DBBACKUP_STORAGE_OPTIONS = {'location': BASE_DIR / 'backup'}
```

```
CRONJOBS = [
    ('*/60 * * * *', 'app_articles.cron.my_scheduled_job'), # 60хв
    ('*/60 * * * *', 'app_books.cron.my_scheduled_job'),
    ('*/60 * * * *', 'app_forum.cron.my_scheduled_job'),
    ('*/60 * * * *', 'app_users.cron.my_scheduled_job')
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'DjangoProject.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [BASE_DIR / 'templates']
    },
    {
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
            ]
        }
    }
]
```

```

        'django.contrib.auth.context_processors.auth',
        'django.contrib.messages.context_processors.messages',
    ],
},
]

WSGI_APPLICATION = 'DjangoProject.wsgi.application'

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME':
        'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME':
        'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]

LANGUAGE_CODE = 'uk'

TIME_ZONE = 'Europe/Kiev'

USE_I18N = True

USE_TZ = True

STATIC_URL = 'static/'

STATICFILES_DIRS = [
    os.path.join(BASE_DIR, "static"),
]

DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'

LOGIN_REDIRECT_URL = '/'

SESSION_COOKIE_AGE = 6 * 60 * 60 # 6 ГОДИН

MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
MEDIA_URL = '/media/'

AUTHENTICATION_BACKENDS = ('django.contrib.auth.backends.ModelBackend',
)

EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
EMAIL_HOST = 'smtp.gmail.com'
EMAIL_PORT = 587

```

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		67

```
EMAIL_USE_TLS = True
EMAIL_HOST_USER = 'email@gmail.com'
DEFAULT_FROM_EMAIL = EMAIL_HOST_USER
EMAIL_HOST_PASSWORD = '1111'
```

urls.py

```
from django.conf.urls.static import static
from django.contrib import admin
from django.urls import path, include

from app_articles import views
from app_articles.views import *

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.home, name='home'),
    path('gallery', views.gallery, name='gallery'),
    path('statistics', views.statistics, name='statistics'),

    path('articles/', include('app_articles.urls')),
    path('users/', include('app_users.urls')),
    path('forum/', include('app_forum.urls')),
    path('books/', include('app_books.urls')),

    path('search', SearchResultsView.as_view(), name='search'),
    path('update_home_page/<int:pk>', UpdateHomePageView.as_view(),
name='update_home_page'),
]
if settings.DEBUG:
    urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

app_articles

admin.py

```
from django.contrib import admin

from .models import *

admin.site.register(Articles)
admin.site.register(Categories)
admin.site.register(HomePage)
admin.site.register(CommentArticle)
```

forms.py

```
from django import forms
from django.forms import.ModelForm, Textarea

from .models import *

class ArticleForm(ModelForm):
    title = forms.CharField(label='Введіть назву (ім`я)',
widget=forms.TextInput(attrs={'class': 'form-control'}))
    full_name = forms.CharField(label='Введіть повну назву (ім`я)',
required=False,
widget=forms.TextInput(attrs={'class': 'form-control'}))
    description = forms.CharField(label='Введіть опис', required=False,
widget=forms.Textarea(attrs={'class': 'form-
```

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		68

```

control'}}))
    quote = forms.CharField(label='Введіть цитату', required=False,
                             widget=forms.Textarea(attrs={'class': 'form-
control'}}))
    biography = forms.CharField(label='Введіть біографію', required=False,
                                 widget=forms.Textarea(attrs={'class': 'form-
control'}}))

    class Meta:
        model = Articles
        fields = ('title', 'full_name', 'description', 'quote', 'biography',
'img', 'category', 'connected_book')

class UpdateHomePageForm(ModelForm):
    class Meta:
        model = HomePage
        fields = ['welcome_text', 'help_text']
        widgets = {
            'welcome_text': Textarea(attrs={
                'class': 'form-control',
            }),
            'help_text': Textarea(attrs={
                'class': 'form-control',
            })
        }

class CategoryForm(ModelForm):
    title = forms.CharField(label='Введіть назву',
                             widget=forms.TextInput(attrs={'class': 'form-control'}))

    class Meta:
        model = Categories
        fields = ('title', 'img')

class CommentForm(ModelForm):
    text = forms.CharField(label='Введіть коментар',
                             widget=forms.Textarea(attrs={'class': 'form-control',
'style': 'height: 100px'}))

    class Meta:
        model = CommentArticle
        fields = ('text',)

```

models.py

```

from django.contrib.auth.models import User
from django.db import models

from DjangoProject import settings
from app_books.models import Books
from .validators import validate_file_extension

class Categories(models.Model):
    title = models.CharField(verbose_name='Назва', max_length=20)
    # description = models.TextField(verbose_name='Опис', blank=True)
    img = models.FileField(verbose_name='Картинка', default=None,
upload_to="category_images/%Y/%m/%d/",
                        validators=[validate_file_extension])

```

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В.В.				69
Змн.	Арк.	№ докум.	Підпис	Дата		

```

def __str__(self):
    return self.title

def get_absolute_url(self):
    return f'/tasks/category/{self.id}/'

class Meta:
    verbose_name = 'Категорія'
    verbose_name_plural = 'Категорії'

class Articles(models.Model):
    title = models.CharField(verbose_name='Назва', max_length=50)
    description = models.TextField(verbose_name='Опис', blank=True, null=True)
    quote = models.TextField(verbose_name='Цитата', default='', blank=True,
null=True)
    biography = models.TextField(verbose_name='Біографія', default='', blank=True,
null=True)
    full_name = models.CharField(verbose_name='Повне ім'я', default='',
max_length=50, blank=True, null=True)
    related_books = models.CharField(verbose_name='Назва', max_length=50,
default='', blank=True, null=True)
    img = models.FileField(verbose_name='Картинка', default=None,
upload_to="articles_images/%Y/%m/%d/",
validators=[validate_file_extension])
    date_created = models.DateTimeField(auto_now_add=True)
    date_updated = models.DateTimeField(auto_now=True)
    views_count = models.IntegerField(verbose_name='к-сть переглядів', default=0)
    connected_book = models.ForeignKey(Books, verbose_name='Пов'язана книжка',
blank=True, default=None, null=True,
on_delete=models.CASCADE,
related_name='book')
    author_article = models.ForeignKey(settings.AUTH_USER_MODEL,
verbose_name='Автор статті', default=None, null=True,
on_delete=models.CASCADE,
related_name='author')
    category = models.ForeignKey(Categories, verbose_name="Категорія", default='',
on_delete=models.CASCADE, blank=True,
null=True)

def __str__(self):
    return self.title

def get_absolute_url(self):
    return f'/articles/{self.id}/'

class Meta:
    ordering = ['-id']
    verbose_name = 'Стаття'
    verbose_name_plural = 'Статті'

class CommentArticle(models.Model):
    text = models.TextField(verbose_name='Коментар')
    article = models.ForeignKey(Articles, verbose_name='Пов'язана стаття',
blank=True, default=None, null=True,
on_delete=models.CASCADE,
related_name='article_comment')
    created = models.DateTimeField(auto_now_add=True)
    creator = models.ForeignKey(settings.AUTH_USER_MODEL, verbose_name="Автор
коментаря", null=False, on_delete=models.CASCADE, default=None)
    updated = models.DateTimeField(auto_now=True)
    likes = models.ManyToManyField(settings.AUTH_USER_MODEL,

```

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		70

```

related_name='article_likes', blank=True)
dislikes = models.ManyToManyField(settings.AUTH_USER_MODEL,
related_name='article_dislikes', blank=True)

def __str__(self):
    return u"%s - %s" % (self.creator, self.article.title)

def short(self):
    return u"%s - %s" % (self.creator, self.created.strftime("%b %d, %I:%M
%p"))

def total_likes(self):
    return self.likes.all().count()

def total_dislikes(self):
    return self.dislikes.all().count()

class Meta:
    ordering = ['-id']
    verbose_name = 'Коментар до статті'
    verbose_name_plural = 'Коментарі до статті'

class HomePage(models.Model):
    welcome_text = models.TextField(verbose_name='welcome_text')
    help_text = models.TextField(verbose_name='Допомога')

    def __str__(self):
        return self.welcome_text

    def get_absolute_url(self):
        return '/'

    class Meta:
        verbose_name = 'Головна сторінка'
        verbose_name_plural = 'Головна сторінка'

```

settings.py

```
from django.conf import settings
```

```

MAIN_TASKS_PER_PAGE = getattr(settings, 'MAIN_TASKS_PER_PAGE', 10)
MAIN_TASKS_FOR_PROFILE_PER_PAGE = getattr(settings,
'MAIN_TASKS_FOR_PROFILE_PER_PAGE', 3)

```

```

BOOKS_PER_PAGE = getattr(settings, 'ARTICLES_PER_PAGE', 4)
REVIEWS_COMMENTS_PER_PAGE = getattr(settings, 'REVIEWS_COMMENTS_PER_PAGE', 3)
REVIEWS_PER_PAGE = getattr(settings, 'REVIEWS_PER_PAGE', 4)

```

urls.py

```
from django.urls import path
```

```
from app_articles import views
```

```

urlpatterns = [
    path('', views.index, name='articles'),

    path('create', views.create, name='create'),
    # path('<int:pk>', views.ArticleDetailView.as_view(), name='article-detail'),

```

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В.В.				71
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    path('<int:pk>', views.ArticleDetailView, name='article-detail'),
    path('<int:pk>/update', views.ArticleUpdateView.as_view(), name='article-
update'),
    path('<int:pk>/delete', views.ArticleDeleteView.as_view(), name='article-
delete'),

    path('categories', views.all_categories, name='all-categories'),
    path('create-category', views.create_category, name='create-category'),
    path('category/<int:pk>/update', views.CategoryUpdateView.as_view(),
name='update-category'),
    path('category/<int:pk>/delete', views.CategoryDeleteView.as_view(),
name='delete-category'),
    path('category/<int:pk>/', views.CategoryDetailView, name='category-detail'),

    path('<int:pk>/like', views.like_comment, name='like_comment'),
    path('<int:pk>/dislike', views.dislike_comment, name='dislike_comment'),
]

```

validators.py

```

def validate_file_extension(value):
    import os
    from django.core.exceptions import ValidationError
    ext = os.path.splitext(value.name)[1] # [0] returns path+filename
    valid_extensions = ['.jpg', '.png', '.jpeg']
    if not ext.lower() in valid_extensions:
        raise ValidationError('Цей тип файлу не підтримується')

```

views.py

```

from django.contrib.auth.decorators import login_required
from django.core.paginator import Paginator, InvalidPage, EmptyPage
from django.db.models import Q
from django.http import HttpResponseRedirect
from django.shortcuts import render, redirect, get_object_or_404
from django.urls import reverse
from django.views.generic import UpdateView, DeleteView, ListView

from app_articles.settings import *
from app_books.models import *
from app_forum.forms import CommentForm
from app_users.models import CustomUser
from .forms import *
from .utils import *

def home(request):
    categories = Categories.objects.order_by('-id')[:4]
    articles = Articles.objects.order_by('-id')[:8]
    genres = Genres.objects.order_by('-id')[:4]
    books = Books.objects.order_by('-id')[:8]

    return render(request, 'home.html',
                  {'home': home,
                   'articles': articles,
                   'categories': categories,
                   'genres': genres,
                   'books': books
                  })

def gallery(request):
    articles = Articles.objects.order_by('-id')[:3] # [:1]

```

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				72
Змн.	Арк.	№ докум.	Підпис	Дата		

```

return render(request, 'gallery.html', {'articles': articles})

def statistics(request):
    return render(request, 'statistics.html')

def index(request):
    contact_list = Articles.objects.order_by('-id').all()
    articles = mk_paginator(request, contact_list, MAIN_TASKS_PER_PAGE)
    return render(request, 'app_articles/index.html', {'articles': articles})

def mk_paginator(request, items, num_items):
    """Create and return a paginator."""
    paginator = Paginator(items, num_items)
    try:
        page = int(request.GET.get("page", '1'))
    except ValueError:
        page = 1

    try:
        items = paginator.page(page)
    except (InvalidPage, EmptyPage):
        items = paginator.page(paginator.num_pages)
    return items

def create(request):
    error = ''

    if request.method == 'POST':
        form = ArticleForm(request.POST, request.FILES)
        if form.is_valid():
            post = form.save(commit=False)
            post.author_article = request.user

            post.save()

            return redirect('articles')
        else:
            print("ERROR!!!\nCAN NOT CREATE ARTICLE!")
            error = 'ERROR!!!\nCAN NOT CREATE ARTICLE!'

    form = ArticleForm()
    context = {
        'form': form,
        'author': request.user,
        'error': error,
        'title': 'Створити статтю'
    }
    return render(request, 'app_articles/create.html', context)

def ArticleDetailView(request, pk):
    article = get_object_or_404(Articles, pk=pk)
    form = CommentForm()
    comments = CommentArticle.objects.filter(article=article).order_by("-id")
    comments = mk_paginator(request, comments, REVIEWS_COMMENTS_PER_PAGE)

    if request.method == 'POST':
        form = CommentForm(request.POST)

```

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В.В.				73
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        if form.is_valid():
            CommentArticle.objects.create(
                text=request.POST.get('text'),
                creator=request.user,
                article=article
            )
            return HttpResponseRedirect(reverse('article-detail', args=(pk,)))

    return render(request, "app_articles/article_details.html", context={
        'form': form,
        'article': article,
        'comments': comments,
        'user': request.user,
    })

@login_required
def like_comment(request, pk):
    comment = get_object_or_404(CommentArticle, pk=pk)
    if request.user in comment.dislikes.all():
        comment.dislikes.remove(request.user)
    if request.user in comment.likes.all():
        comment.likes.remove(request.user)
    else:
        comment.likes.add(request.user)
    return redirect('article-detail', pk=comment.article.pk)

@login_required
def dislike_comment(request, pk):
    comment = get_object_or_404(CommentArticle, pk=pk)
    if request.user in comment.likes.all():
        comment.likes.remove(request.user)
    if request.user in comment.dislikes.all():
        comment.dislikes.remove(request.user)
    else:
        comment.dislikes.add(request.user)
    return redirect('article-detail', pk=comment.article.pk)

class ArticleUpdateView(UpdateView):
    model = Articles
    template_name = 'app_articles/create.html'
    form_class = ArticleForm

class ArticleDeleteView(DeleteView):
    model = Articles
    template_name = 'app_articles/delete.html'
    success_url = '/articles/'

class SearchResultsView(ListView):
    template_name = 'app_articles/search.html'

    def get_queryset(self):
        query = self.request.GET.get('q', '')
        articles = Articles.objects.none()
        books = Books.objects.none()
        authors = Authors.objects.none()
        users = CustomUser.objects.none()
        if query:
            articles = Articles.objects.filter(

```

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В.В.				74
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        Q(title__icontains=query) |
        Q(description__icontains=query) |
        Q(quote__icontains=query) |
        Q(biography__icontains=query) |
        Q(full_name__icontains=query)
    )
    books = Books.objects.filter(
        Q(title__icontains=query) |
        Q(isbn__icontains=query)
    )
    authors = Authors.objects.filter(
        Q(name__icontains=query) |
        Q(surname__icontains=query)
    )
    users = CustomUser.objects.filter(
        Q(username__icontains=query)
    )
    return articles, books, authors, users

def get_context_data(self, **kwargs):
    context = super().get_context_data(**kwargs)
    articles, books, authors, users = self.get_queryset()
    context['articles'] = articles
    context['books'] = books
    context['authors'] = authors
    context['users'] = users
    return context

class UpdateHomePageView(UpdateView):
    model = HomePage
    template_name = 'update_home_page.html'
    form_class = UpdateHomePageForm

def all_categories(request):
    contact_list = Categories.objects.order_by('-id').all()
    categories = mk_paginator(request, contact_list, MAIN_TASKS_PER_PAGE)
    return render(request, 'app_articles/all_categories.html', {'categories':
categories})

def create_category(request):
    if request.method == 'POST':
        form = CategoryForm(request.POST, request.FILES)
        if form.is_valid():
            post = form.save(commit=False)
            post.save()
            return redirect('home')

    form = CategoryForm()
    context = {
        'form': form,
        'title': 'Створити категорію'
    }
    return render(request, 'app_articles/create.html', context)

def CategoryDetailView(request, pk):
    articles = Articles.objects.filter(category=pk).order_by("-id")
    articles = mk_paginator(request, articles, MAIN_TASKS_PER_PAGE)

```

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				75
Змн.	Арк.	№ докум.	Підпис	Дата		

```

category = get_object_or_404(Categories, pk=pk)

return render(request, "app_articles/category_details.html", context={
    'articles': articles,
    'category': category,
})

class CategoryUpdateView(UpdateView):
    model = Categories
    template_name = 'app_articles/create.html'
    form_class = CategoryForm
    success_url = '/articles/categories'

class CategoryDeleteView(DeleteView):
    model = Categories
    template_name = 'app_articles/delete.html'
    success_url = '/articles/categories'

```

app_books

admin.py

```

from django.contrib import admin

from app_books.models import *

admin.site.register(Books)
admin.site.register(Genres)
admin.site.register(Authors)
admin.site.register(Reviews)
admin.site.register(CommentReview)

```

forms.py

```

from django import forms
from django.forms import ModelForm

from .models import *

class AuthorForm(ModelForm):
    name = forms.CharField(label='Введіть ім'я',
        widget=forms.TextInput(attrs={'class': 'form-control'}))
    surname = forms.CharField(label='Введіть прізвище',
        widget=forms.TextInput(attrs={'class': 'form-control'}))
    biography = forms.CharField(label='Введіть біографію', required=False,
        widget=forms.Textarea(attrs={'class': 'form-control'}))
    img = forms.FileField(label='Фото', required=False,
        widget=forms.FileInput(attrs={'class': 'form-control'}))

    class Meta:
        model = Authors
        fields = ('name', 'surname', 'biography', 'img')

class BookForm(ModelForm):
    title = forms.CharField(label='Введіть назву (ім'я)',
        widget=forms.TextInput(attrs={'class': 'form-control'}))
    description = forms.CharField(label='Введіть анотацію',
        widget=forms.Textarea(attrs={'class': 'form-control'}))

```

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				76
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        isbn = forms.CharField(label='Введіть ISBN',
                                widget=forms.TextInput(attrs={'class': 'form-control'}))
        img_name = forms.CharField(label='Заголовок зображення', required=False,
                                    widget=forms.TextInput(attrs={'class': 'form-control'}))

    class Meta:
        model = Books
        fields = ('title', 'description', 'isbn', 'book_author', 'img',
                  'img_name', 'genre')

class GenreForm(ModelForm):
    title = forms.CharField(label='Введіть назву жанру',
                             widget=forms.TextInput(attrs={'class': 'form-control'}))
    description = forms.CharField(label='Введіть опис жанру', required=False,
                                  widget=forms.Textarea(attrs={'class': 'form-control'}))
    img = forms.FileField(label='Фото', widget=forms.FileInput(attrs={'class': 'form-control'}))

    class Meta:
        model = Genres
        fields = ('title', 'description', 'img')

class TypeBooksForm(ModelForm):
    rate = forms.IntegerField(label='Оцінка',
                              widget=forms.NumberInput(attrs={'class': 'form-control', 'min': '0', 'max': '5'}))

    class Meta:
        model = Books
        fields = ('rate', 'read_now', 'want_to_read', 'read_books')

class SortForm(forms.Form):
    sort = forms.ModelChoiceField(queryset=Books.objects.all())

class ReviewForm(ModelForm):
    description = forms.CharField(label='Введіть анотацію',
                                   widget=forms.Textarea(attrs={'class': 'form-control'}))

    class Meta:
        model = Reviews
        fields = ('description',)

```

models.py

```

from django.contrib.auth.models import User
from django.db import models

from DjangoProject import settings
from app_articles.validators import validate_file_extension

class Authors(models.Model):
    name = models.CharField(verbose_name='Ім'я', max_length=50)
    surname = models.CharField(verbose_name='Прізвище', max_length=50)
    biography = models.TextField(verbose_name='Біографія', blank=True)

```

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		77

```

        creator = models.ForeignKey(settings.AUTH_USER_MODEL,
verbose_name="Створювач", null=False,
                                on_delete=models.CASCADE, default=None)
        img = models.FileField(verbose_name='Фото автора',
                                default=None,
                                upload_to="author_images/%Y/%m/%d/",
                                validators=[validate_file_extension])

    def __str__(self):
        return self.name + " " + self.surname

    def get_absolute_url(self):
        return f'/books/author/{self.id}/'

    class Meta:
        verbose_name = 'Автор'
        verbose_name_plural = 'Автори'

class Genres(models.Model):
    title = models.CharField(verbose_name='Назва', max_length=50)
    description = models.TextField(verbose_name='Опис', blank=True)
    img = models.FileField(verbose_name='Картинка', default=None,
upload_to="genres_images/%Y/%m/%d/",
                                validators=[validate_file_extension])

    def __str__(self):
        return self.title

    def get_absolute_url(self):
        return f'/books/genre/{self.id}'

    class Meta:
        verbose_name = 'Жанр'
        verbose_name_plural = 'Жанри'

class Books(models.Model):
    title = models.CharField(verbose_name='Назва', max_length=50)
    description = models.TextField(verbose_name='Опис')
    isbn = models.CharField(verbose_name='ІSBN', max_length=13, blank=True,
null=True)
    rate = models.IntegerField(verbose_name='Оцінка', default=0, blank=True,
null=True)
    img = models.FileField(verbose_name='Картинка',
default="![ ](..static/img/book_default.png)",
                                upload_to="books_images/%Y/%m/%d/", blank=True,
null=True)
    img_name = models.CharField(verbose_name='Назва картинки', default=" ",
max_length=50, blank=True, null=True)
    date_created = models.DateTimeField(auto_now_add=True)
    date_updated = models.DateTimeField(auto_now=True)
    # views_count = models.IntegerField(verbose_name='к-сть переглядів',
default=0, blank=True, null=True)
    creator = models.ForeignKey(settings.AUTH_USER_MODEL, verbose_name="Додав
книжку", default='',
                                on_delete=models.PROTECT)
    book_author = models.ForeignKey(Authors, verbose_name="Автор", default='',
on_delete=models.CASCADE)
    genre = models.ForeignKey(Genres, verbose_name="Жанр", default='',
on_delete=models.CASCADE)

    read_now = models.ManyToManyField(settings.AUTH_USER_MODEL,

```

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				78
Змн.	Арк.	№ докум.	Підпис	Дата		

```

verbose_name="Читаю зараз", blank=True,
                                related_name='read_now')
    want_to_read = models.ManyToManyField(settings.AUTH_USER_MODEL,
verbose_name="Хочу прочитати", blank=True,
                                related_name='want_to_read')
    read_books = models.ManyToManyField(settings.AUTH_USER_MODEL,
verbose_name="Прочитано", blank=True,
                                related_name='read_books')

    def __str__(self):
        return self.title

    def get_absolute_url(self):
        return f'/books/{self.id}'

    def short(self):
        return u"%s" % self.title

short.allow_tags = True

class Meta:
    ordering = ['-id']
    verbose_name = 'Книжка'
    verbose_name_plural = 'Книжки'

class Reviews(models.Model):
    description = models.TextField(verbose_name='Основна частина')
    book = models.ForeignKey('Books', verbose_name='Пов'язана книжка', blank=True,
default=None, null=True,
                                on_delete=models.CASCADE, related_name='book_review')
    created = models.DateTimeField(auto_now_add=True)
    creator = models.ForeignKey(settings.AUTH_USER_MODEL, verbose_name="Автор
рецензії", null=False,
                                on_delete=models.CASCADE, default=None)
    updated = models.DateTimeField(auto_now=True)
    likes = models.ManyToManyField(settings.AUTH_USER_MODEL,
related_name='review_likes', blank=True)
    dislikes = models.ManyToManyField(settings.AUTH_USER_MODEL,
related_name='review_dislikes', blank=True)

    def __str__(self):
        return u"%s - %s" % (self.creator, self.book.title)

    def short(self):
        return u"%s - %s" % (self.creator, self.created.strftime("%b %d, %I:%M
%p"))

    def total_likes(self):
        return self.likes.all().count()

    def total_dislikes(self):
        return self.dislikes.all().count()

    class Meta:
        ordering = ['-id']
        verbose_name = 'Рецензія'
        verbose_name_plural = 'Рецензії'

class CommentReview(models.Model):

```

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				79
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    body = models.TextField(verbose_name="Коментар до рецензії", max_length=1000,
blank=False)
    created = models.DateTimeField(auto_now_add=True)
    creator = models.ForeignKey(settings.AUTH_USER_MODEL, verbose_name="Автор
коментаря до рецензії", null=False,
                                on_delete=models.CASCADE,
                                default=None)
    updated = models.DateTimeField(auto_now=True)
    review = models.ForeignKey(Reviews, verbose_name="Рецензія",
on_delete=models.CASCADE)

    def __str__(self):
        return u"%s - %s - %s" % (self.creator, self.body, self.review.title)

    def short(self):
        return u"%s - %s" % (self.creator, self.created.strftime("%b %d, %I:%M
%p"))

    short.allow_tags = True

    class Meta:
        verbose_name = 'Коментар'
        verbose_name_plural = 'Коментарі'

    def get_absolute_url(self):
        return f'/books/review/{self.review.id}'

```

settings.py

```
from django.conf import settings
```

```

BOOKS_PER_PAGE = getattr(settings, 'BOOKS_PER_PAGE', 4)
REVIEWS_COMMENTS_PER_PAGE = getattr(settings, 'REVIEWS_COMMENTS_PER_PAGE', 3)
REVIEWS_PER_PAGE = getattr(settings, 'REVIEWS_PER_PAGE', 4)

```

urls.py

```
from django.urls import path, re_path
```

```

from app_books import views
from app_books.views import *

```

```

urlpatterns = [
    path('', views.index_books, name='books'),
    path('authors', views.index_authors, name='authors'),
    path('add_author', views.add_author, name='add-author'),
    re_path(r'^author/(\d+)/$', views.author_p, name='author-detail'),
    path('author/<int:pk>/update', views.AuthorUpdateView.as_view(), name='author-
update'),
    path('author/<int:pk>/delete', views.AuthorDeleteView.as_view(), name='author-
delete'),

    # STATISTICS
    path('old', views.index_books_old, name='books_old'),
    path('title', views.index_books_title, name='books_title'),
    path('book_author', views.index_books_book_author, name='books_book_author'),
    path('rate', views.index_books_rate, name='books_rate'),
    # STATISTICS

    path('add', views.add_book, name='add-book'),
    path('<int:pk>', views.BookDetailView.as_view(), name='book-detail'),

```

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		80

```

re_path(r'^(\d+)/articles$', connected_book, name='connected_book'),
path('<int:pk>/update', views.BookUpdateView.as_view(), name='book-update'),
path('<int:pk>/delete', views.BookDeleteView.as_view(), name='book-delete'),

# STATISTICS
re_path(r'^read_now/(\d+)/$', read_now, name='read_now'),
re_path(r'^want_to_read/(\d+)/$', want_to_read, name='want_to_read'),
re_path(r'^read_books/(\d+)/$', read_books, name='read_books'),

re_path(r'^read_books/(\d+)/old$', read_books_old, name='read_books_old'),
re_path(r'^read_books/(\d+)/title$', read_books_title,
name='read_books_title'),
re_path(r'^read_books/(\d+)/book_author$', read_books_book_author,
name='read_books_book_author'),
re_path(r'^read_books/(\d+)/rate$', read_books_rate, name='read_books_rate'),

re_path(r'^read_now/(\d+)/old$', read_now_old, name='read_now_old'),
re_path(r'^read_now/(\d+)/title$', read_now_title, name='read_now_title'),
re_path(r'^read_now/(\d+)/book_author$', read_now_book_author,
name='read_now_book_author'),
re_path(r'^read_now/(\d+)/rate$', read_now_rate, name='read_now_rate'),

re_path(r'^want_to_read/(\d+)/old$', want_to_read_old,
name='want_to_read_old'),
re_path(r'^want_to_read/(\d+)/title$', want_to_read_title,
name='want_to_read_title'),
re_path(r'^want_to_read/(\d+)/book_author$', want_to_read_book_author,
name='want_to_read_book_author'),
re_path(r'^want_to_read/(\d+)/rate$', want_to_read_rate,
name='want_to_read_rate'),

# STATISTICS

path('genres', views.index_genres, name='genres'),
path('create-genre', views.add_genre, name='create-genre'),
path('genre/<int:pk>/update', views.GenreUpdateView.as_view(), name='update-
genre'),
path('genre/<int:pk>/delete', views.GenreDeleteView.as_view(), name='delete-
genre'),
re_path(r'^genre/(\d+)/$', views.GenreDetailView, name='genre-detail'),
re_path(r'^(\d+)/reviews$', all_reviews, name='all_reviews'),
path('review/<int:pk>', views.review, name='reviews'),
re_path(r'^newreview/(\d+)/$', views.new_review, name='new_review'),
path('review/<int:pk>/update', views.ReviewUpdateView.as_view(), name='review-
update'),
path('review/<int:pk>/delete', views.ReviewDeleteView.as_view(), name='review-
delete'),
path('review/<int:pk>/like', like_review, name='like_review'),
path('review/<int:pk>/dislike', dislike_review, name='dislike_review'),

path('comment/<int:pk>/update/', views.CommentUpdateView.as_view(),
name='comment-update'),
path('comment/<int:pk>/delete/', views.CommentDeleteView.as_view(),
name='comment-delete'),
]

```

views.py

```

from django.contrib.auth import get_user_model
from django.contrib.auth.decorators import login_required
from django.core.paginator import Paginator, InvalidPage, EmptyPage
from django.db.models import Q
from django.http import HttpResponseRedirect

```

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		81

```

from django.shortcuts import render, redirect, get_object_or_404
from django.urls import reverse
from django.views.generic import UpdateView, DeleteView, ListView

from app_articles.models import Articles
from app_books.settings import *
from app_forum.forms import CommentForm
from app_forum.views import add_csrf
from .forms import *
from .utils import *

User = get_user_model()

def index_genres(request):
    contact_list = Genres.objects.order_by('-id').all()
    genres = mk_paginator(request, contact_list, BOOKS_PER_PAGE)
    return render(request, 'app_books/all_genres.html', {'genres': genres})

def index_authors(request):
    contact_list = Authors.objects.order_by('name', 'surname').all()
    authors = mk_paginator(request, contact_list, BOOKS_PER_PAGE)
    return render(request, 'app_books/author/all_authors.html', {'authors':
authors})

# Author
def author_p(request, author_id):
    author = Authors.objects.get(pk=author_id)
    books = Books.objects.filter(book_author=author_id).order_by("-id")

    return render(request, "app_books/author/author_profile.html", context={
        'author': author,
        'books': books
    })

def add_author(request):
    error = ''

    if request.method == 'POST':
        form = AuthorForm(request.POST, request.FILES)
        if form.is_valid():
            post = form.save(commit=False)
            post.creator = request.user

            post.save()
            return redirect('authors')
        else:
            print("ERROR!!!\nCAN NOT ADD BOOK!")
            error = 'ERROR!!!\nCAN NOT ADD BOOK!'

    form = AuthorForm()
    context = {
        'form': form,
        'author': request.user,
        'error': error,
        'title': 'Додати автора'
    }
    return render(request, 'app_books/add_book.html', context)

```

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		82

```

class AuthorUpdateView(UpdateView):
    model = Authors
    template_name = 'app_books/add_book.html'
    form_class = AuthorForm

class AuthorDeleteView(DeleteView):
    model = Authors
    template_name = 'app_books/delete.html'
    success_url = '/books/authors'

# STATISTICS
def index_books(request):
    # contact_list = Books.objects.order_by('-id').all()
    # books = mk_paginator(request, contact_list, MAIN_TASKS_PER_PAGE)

    books = Books.objects.order_by('-date_created').all()
    books = mk_paginator(request, books, BOOKS_PER_PAGE)
    return render(request, 'app_books/index.html', {'books': books})

def index_books_old(request):
    # contact_list = Books.objects.order_by('-id').all()
    # books = mk_paginator(request, contact_list, MAIN_TASKS_PER_PAGE)
    books = Books.objects.order_by('date_created').all()
    books = mk_paginator(request, books, BOOKS_PER_PAGE)
    return render(request, 'app_books/index.html', {'books': books})

def index_books_title(request):
    # contact_list = Books.objects.order_by('-id').all()
    # books = mk_paginator(request, contact_list, MAIN_TASKS_PER_PAGE)
    books = Books.objects.order_by('title').all()
    books = mk_paginator(request, books, BOOKS_PER_PAGE)
    return render(request, 'app_books/index.html', {'books': books})

def index_books_book_author(request):
    # contact_list = Books.objects.order_by('-id').all()
    # books = mk_paginator(request, contact_list, MAIN_TASKS_PER_PAGE)
    books = Books.objects.order_by('book_author').all()
    books = mk_paginator(request, books, BOOKS_PER_PAGE)
    return render(request, 'app_books/index.html', {'books': books})

def index_books_rate(request):
    # contact_list = Books.objects.order_by('-id').all()
    # books = mk_paginator(request, contact_list, MAIN_TASKS_PER_PAGE)
    books = Books.objects.order_by('-rate').all()
    books = mk_paginator(request, books, BOOKS_PER_PAGE)
    return render(request, 'app_books/index.html', {'books': books})

def read_now(request, pk):
    book = get_object_or_404(Books, pk=pk)
    book.read_now.add(request.user)
    book.save()
    return redirect('book-detail', pk=pk)

def read_now_old(request, user_id):
    books_read_now = Books.objects.filter(read_now=user_id).order_by("id")

```

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ІЗ	Арк.
		Концедайло В.В.				83
Змн.	Арк.	№ докум.	Підпис	Дата		

```

books_read_now = mk_paginator(request, books_read_now, BOOKS_PER_PAGE)
user = User.objects.get(pk=user_id)

return render(request, "app_books/statistics/read_now.html", context={
    'books_read_now': books_read_now,
    'user': user,
})

def read_now_title(request, user_id):
    books_read_now = Books.objects.filter(read_now=user_id).order_by("title")
    books_read_now = mk_paginator(request, books_read_now, BOOKS_PER_PAGE)
    user = User.objects.get(pk=user_id)

    return render(request, "app_books/statistics/read_now.html", context={
        'books_read_now': books_read_now,
        'user': user,
    })

def read_now_book_author(request, user_id):
    books_read_now =
Books.objects.filter(read_now=user_id).order_by("book_author")
    books_read_now = mk_paginator(request, books_read_now, BOOKS_PER_PAGE)
    user = User.objects.get(pk=user_id)

    return render(request, "app_books/statistics/read_now.html", context={
        'books_read_now': books_read_now,
        'user': user,
    })

def read_now_rate(request, user_id):
    books_read_now = Books.objects.filter(read_now=user_id).order_by("-rate")
    books_read_now = mk_paginator(request, books_read_now, BOOKS_PER_PAGE)
    user = User.objects.get(pk=user_id)

    return render(request, "app_books/statistics/read_books.html", context={
        'books_read_now': books_read_now,
        'user': user,
    })

def want_to_read(request, pk):
    book = get_object_or_404(Books, pk=pk)
    book.want_to_read.add(request.user)
    book.save()
    return redirect('book-detail', pk=pk)

def want_to_read_old(request, user_id):
    books_want_to_read = Books.objects.filter(want_to_read=user_id).order_by("-
id")
    books_want_to_read = mk_paginator(request, books_want_to_read, BOOKS_PER_PAGE)
    user = User.objects.get(pk=user_id)

    return render(request, "app_books/statistics/want_to_read.html", context={
        'books_want_to_read': books_want_to_read,
        'user': user,
    })

def want_to_read_title(request, user_id):

```

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		84

```

books_want_to_read =
Books.objects.filter(want_to_read=user_id).order_by("title")
books_want_to_read = mk_paginator(request, books_want_to_read, BOOKS_PER_PAGE)
user = User.objects.get(pk=user_id)

return render(request, "app_books/statistics/want_to_read.html", context={
    'books_want_to_read': books_want_to_read,
    'user': user,
})

def want_to_read_book_author(request, user_id):
    books_want_to_read =
Books.objects.filter(want_to_read=user_id).order_by("book_author")
books_want_to_read = mk_paginator(request, books_want_to_read, BOOKS_PER_PAGE)
user = User.objects.get(pk=user_id)

return render(request, "app_books/statistics/want_to_read.html", context={
    'books_want_to_read': books_want_to_read,
    'user': user,
})

def want_to_read_rate(request, user_id):
    books_want_to_read = Books.objects.filter(want_to_read=user_id).order_by("-
rate")
books_want_to_read = mk_paginator(request, books_want_to_read, BOOKS_PER_PAGE)
user = User.objects.get(pk=user_id)

return render(request, "app_books/statistics/want_to_read.html", context={
    'books_want_to_read': books_want_to_read,
    'user': user,
})

def read_books(request, user_id):
    books_read = Books.objects.filter(read_books=user_id).order_by("-id")
books_read = mk_paginator(request, books_read, BOOKS_PER_PAGE)
user = User.objects.get(pk=user_id)

return render(request, "app_books/statistics/read_books.html", context={
    'read_books': books_read,
    'user': user,
})

def read_books_old(request, pk):
    book = get_object_or_404(Books, pk=pk)
    book.read_books.add(request.user)
    book.save()
    return redirect('book-detail', pk=pk)

def read_books_title(request, user_id):
    books_read = Books.objects.filter(read_books=user_id).order_by("title")
books_read = mk_paginator(request, books_read, BOOKS_PER_PAGE)
user = User.objects.get(pk=user_id)

return render(request, "app_books/statistics/read_books.html", context={
    'read_books': books_read,
    'user': user,
})

```

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		85

```

def read_books_book_author(request, user_id):
    books_read = Books.objects.filter(read_books=user_id).order_by("book_author")
    books_read = mk_paginator(request, books_read, BOOKS_PER_PAGE)
    user = User.objects.get(pk=user_id)

    return render(request, "app_books/statistics/read_books.html", context={
        'read_books': books_read,
        'user': user,
    })

def read_books_rate(request, user_id):
    books_read = Books.objects.filter(read_books=user_id).order_by("-rate")
    books_read = mk_paginator(request, books_read, BOOKS_PER_PAGE)
    user = User.objects.get(pk=user_id)

    return render(request, "app_books/statistics/read_books.html", context={
        'read_books': books_read,
        'user': user,
    })

def mk_paginator(request, items, num_items):
    paginator = Paginator(items, num_items)
    try:
        page = int(request.GET.get("page", '1'))
    except ValueError:
        page = 1

    try:
        items = paginator.page(page)
    except (InvalidPage, EmptyPage):
        items = paginator.page(paginator.num_pages)
    return items

def add_book(request):
    error = ''

    if request.method == 'POST':
        form = BookForm(request.POST, request.FILES)
        if form.is_valid():
            post = form.save(commit=False)
            post.creator = request.user

            post.save()
            return redirect('books')
        else:
            print("ERROR!!!\nCAN NOT ADD BOOK!")
            error = 'ERROR!!!\nCAN NOT ADD BOOK!'

    form = BookForm()
    context = {
        'form': form,
        'author': request.user,
        'error': error,
        'title': 'Додати книгу'
    }
    return render(request, 'app_books/add_book.html', context)

class BookDetailView(UpdateView):

```

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		86

```

model = Books
template_name = 'app_books/book_details.html'
context_object_name = 'book'
form_class = TypeBooksForm

def connected_book(request, book_id):
    articles = Articles.objects.filter(connected_book=book_id).order_by('-
id').all()
    return render(request, 'app_articles/index.html', {'articles': articles})

class BookUpdateView(UpdateView):
    model = Books
    template_name = 'app_books/add_book.html'
    form_class = BookForm

    def get_object(self, queryset=None):
        return self.request.user

class BookDeleteView(DeleteView):
    model = Books
    template_name = 'app_books/delete.html'
    success_url = '/books/'

class SearchResultsView(ListView):
    model = Books
    template_name = 'app_books/search_book.html'

    def get_queryset(self): # НОВЫЙ
        query = self.request.GET.get('q', '')
        if query:
            books = Books.objects.filter(
                Q(title__icontains=query) |
                Q(quote__icontains=query) |
                Q(biography__icontains=query) |
                Q(full_name__icontains=query)
            )
            return books
        else:
            books = Books.objects.all()
            return books

def add_genre(request):
    if request.method == 'POST':
        form = GenreForm(request.POST, request.FILES)
        if form.is_valid():
            post = form.save(commit=False)

            post.save()
            return redirect('home')

    form = GenreForm()
    context = {
        'form': form,
        'title': 'Додати жанр'
    }
    return render(request, 'app_books/add_book.html', context)

```

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				87
Змн.	Арк.	№ докум.	Підпис	Дата		

```

def GenreDetailView(request, genre_id):
    books = Books.objects.filter(genre=genre_id).order_by("-id")
    books = mk_paginator(request, books, BOOKS_PER_PAGE)
    genre = Genres.objects.get(pk=genre_id)

    return render(request, "app_books/genre_details.html", context={
        'books': books,
        'genre': genre,
    })

class GenreUpdateView(UpdateView):
    model = Genres
    template_name = 'app_books/add_book.html'
    form_class = GenreForm

class GenreDeleteView(DeleteView):
    model = Genres
    template_name = 'app_books/delete.html'
    success_url = '/books/genres'

def all_reviews(request, book_id):
    """Listing of topics in a forum."""
    reviews = Reviews.objects.filter(book=book_id).order_by("-created")
    reviews = mk_paginator(request, reviews, REVIEWS_PER_PAGE)

    book = get_object_or_404(Books, pk=book_id)

    return render(request, "app_books/all_reviews.html",
        add_csrf(request, reviews=reviews, pk=book_id, book=book))

def review(request, pk):
    reviews = Reviews.objects.filter(pk=pk)
    form = CommentForm()
    comments = CommentReview.objects.filter(review_id=pk).order_by("-id")
    comments = mk_paginator(request, comments, REVIEWS_COMMENTS_PER_PAGE)

    if request.method == 'POST':
        form = CommentForm(request.POST)
        if form.is_valid():
            CommentReview.objects.create(
                body=request.POST.get('body'),
                creator=request.user,
                review=Reviews.objects.get(id=pk)
            )
            return HttpResponseRedirect(reverse('reviews', args=(pk,)))

    return render(request, "app_books/review.html", context={
        'form': form,
        'reviews': reviews,
        'comments': comments,
        'user': request.user,
    })

@login_required
def new_review(request, book_id):
    form = ReviewForm()
    book = get_object_or_404(Books, pk=book_id)

```

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		88

```

if request.method == 'POST':
    form = ReviewForm(request.POST)

    if form.is_valid():
        post = form.save(commit=False)
        post.creator = request.user
        post.book = book
        post.save()

        return HttpResponseRedirect(reverse('book-detail', args=(book_id,)))
context = {
    'form': form,
    'book': book,
}
return render(request, 'app_books/new_review.html', context)

@login_required
def like_review(request, pk):
    review = get_object_or_404(Reviews, pk=pk)
    if request.user in review.dislikes.all():
        review.dislikes.remove(request.user)
    if request.user in review.likes.all():
        review.likes.remove(request.user)
    else:
        review.likes.add(request.user)
    return redirect('reviews', pk=review.pk)

@login_required
def dislike_review(request, pk):
    review = get_object_or_404(Reviews, pk=pk)
    if request.user in review.likes.all():
        review.likes.remove(request.user)
    if request.user in review.dislikes.all():
        review.dislikes.remove(request.user)
    else:
        review.dislikes.add(request.user)
    return redirect('reviews', pk=review.pk)

class ReviewUpdateView(UpdateView):
    model = Reviews
    template_name = 'app_books/add_book.html'
    form_class = ReviewForm

class ReviewDeleteView(DeleteView):
    model = Reviews
    template_name = 'app_books/delete.html'
    success_url = '/books/genres'

class CommentUpdateView(UpdateView):
    model = CommentReview
    template_name = 'app_forum/update-comment.html'
    form_class = CommentForm

class CommentDeleteView(DeleteView):
    model = CommentReview

```

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				89
Змн.	Арк.	№ докум.	Підпис	Дата		

```
template_name = 'app_forum/delete.html'
success_url = '/books'
```

app_forum

admin.py

```
from django.contrib import admin

from app_forum.models import *

class ForumAdmin(admin.ModelAdmin):
    pass

class TopicAdmin(admin.ModelAdmin):
    list_display = ["title", "forum", "creator", "created"]
    list_filter = ["forum", "creator"]

class CommentAdmin(admin.ModelAdmin):
    search_fields = ["creator"]
    list_display = ["topic", "creator", "created"]

class ReplyForCommentAdmin(admin.ModelAdmin):
    search_fields = ["text", "creator"]
    list_display = ["text", "comment", "creator", "created"]

class ProfaneWordAdmin(admin.ModelAdmin):
    pass

admin.site.register(Forum, ForumAdmin)
admin.site.register(Topic, TopicAdmin)
admin.site.register(Comment, CommentAdmin)
admin.site.register(ProfaneWord, ProfaneWordAdmin)
```

forms.py

```
from functools import reduce

from django import forms
from django.forms import TextInput, Textarea, ModelForm

from app_forum.models import *
from app_forum.settings import *
from app_users.models import CustomUser

class ForumForm(ModelForm):
    class Meta:
        model = Topic
        fields = ['title', 'description']
        widgets = {
            'title': TextInput(attrs={
                'class': 'form-control',
                'placeholder': 'Введіть назву'
            }),
            'description': Textarea(attrs={
                'class': 'form-control'
            })
        }
```

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				90
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    }

class TopicForm(ModelForm):
    allowed_users = forms.ModelMultipleChoiceField(
        queryset=CustomUser.objects.all(), # Використовуємо кастомну модель
користувача
        required=False,
        widget=forms.CheckboxSelectMultiple,
        label='Допущені користувачі'
    )

    class Meta:
        model = Topic
        fields = ['title', 'description',] # Видалить allowed_users з полів форми
        widgets = {
            'title': TextInput(attrs={
                'class': 'form-control',
                'placeholder': 'Введіть назву'
            }),
            'description': Textarea(attrs={
                'class': 'form-control'
            }),
        }

class AddUserForm(forms.Form):
    allowed_users = forms.ModelMultipleChoiceField(
        queryset=CustomUser.objects.all(),
        widget=forms.CheckboxSelectMultiple,
        label='Додати користувачів',
        required=False,
    )

class CommentForm(ModelForm):
    class Meta:
        model = Comment
        fields = ['body']
        widgets = {
            'body': Textarea(attrs={
                'class': 'form-control',
                'style': 'height: 100px',
            })
        }

    def clean_body(self):
        body = self.cleaned_data["body"]

        if FORUM_FILTER_PROFANE_WORDS:
            profane_words = ProfaneWord.objects.all()
            bad_words = [w for w in profane_words if w.word in body.lower()]

            if bad_words:
                raise forms.ValidationError("Такі слова як '%s' заборонені." % (
                    reduce(lambda x, y: "%s,%s" % (x, y), bad_words)))

        return body

```

models.py

```

from django.contrib.auth.models import User
from django.db import models
from DjangoProject import settings

```

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		91

```

class Forum(models.Model):
    title = models.CharField(verbose_name="Заголовок", max_length=60, blank=False)
    description = models.TextField(verbose_name="Опис", blank=False)
    updated = models.DateTimeField(auto_now=True)
    created = models.DateTimeField(auto_now_add=True)
    creator = models.ForeignKey(settings.AUTH_USER_MODEL, verbose_name="Автор",
null=False, on_delete=models.CASCADE,
                                default=None)

    def __str__(self):
        return self.title

    def get_absolute_url(self):
        return f'/forum/'

class Meta:
    verbose_name = 'Форум'
    verbose_name_plural = 'Форуми'

    def num_comments(self):
        return sum([t.num_comments() for t in self.topic_set.all()])

    def last_comment(self):
        if self.topic_set.count():
            last = None
            for t in self.topic_set.all():
                l = t.last_comment()
                if l:
                    if not last:
                        last = l
                    elif l.created > last.created:
                        last = l
            return last

class Topic(models.Model):
    title = models.CharField(verbose_name="Заголовок", max_length=60, blank=False)
    description = models.TextField(verbose_name="Опис", max_length=10000,
blank=False, null=True)
    forum = models.ForeignKey(Forum, verbose_name="Форум",
on_delete=models.CASCADE)
    created = models.DateTimeField(auto_now_add=True)
    creator = models.ForeignKey(settings.AUTH_USER_MODEL, verbose_name="Автор",
null=False, on_delete=models.PROTECT,
                                default=None)
    is_private = models.BooleanField(verbose_name="Приватне обговорення",
default=False)
    allowed_users = models.ManyToManyField(settings.AUTH_USER_MODEL,
verbose_name="Допущені користувачі",
                                related_name='allowed_topics',
blank=True)
    updated = models.DateTimeField(auto_now=True)
    closed = models.BooleanField(default=False)

class Meta:
    verbose_name = 'Обговорення'
    verbose_name_plural = 'Обговорення'

    def num_comments(self):
        return self.comment_set.count()

```

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		92

```

def num_comms(self):
    return max(0, self.comment_set.count())

def last_comment(self):
    if self.comment_set.count():
        return self.comment_set.order_by("created")[0]

def __str__(self):
    return str(self.creator) + " - " + self.title

def get_absolute_url(self):
    return f'/forum/topic/{self.id}/'

class Comment(models.Model):
    body = models.TextField(verbose_name="Коментар", max_length=1000, blank=False)
    created = models.DateTimeField(auto_now_add=True)
    creator = models.ForeignKey(settings.AUTH_USER_MODEL, verbose_name="Автор",
                                null=False, on_delete=models.CASCADE,
                                default=None)
    updated = models.DateTimeField(auto_now=True)
    topic = models.ForeignKey(Topic, verbose_name="Обговорення",
                              on_delete=models.CASCADE)

    def __str__(self):
        return u"%s - %s - %s" % (self.creator, self.body, self.topic.title)

    def short(self):
        return u"%s - %s" % (self.creator, self.created.strftime("%b %d, %I:%M
%p"))

    short.allow_tags = True

class Meta:
    verbose_name = 'Коментар'
    verbose_name_plural = 'Коментарі'

    def get_absolute_url(self):
        return f'/forum/topic/{self.topic.id}'

class ProfaneWord(models.Model):
    word = models.CharField(verbose_name="", max_length=60)

    def __str__(self):
        return self.word

    class Meta:
        verbose_name = 'Заборонене слово'
        verbose_name_plural = 'Заборонені слова'

```

settings.py

```
from django.conf import settings
```

```

FORUM_PER_PAGE = getattr(settings, 'FORUM_TOPICS_PER_PAGE', 10)
FORUM_COMMENTS_PER_PAGE = getattr(settings, 'FORUM_COMMENTS_PER_PAGE', 3)
FORUM_FILTER_PROFANE_WORDS = getattr(settings, 'FORUM_FILTER_PROFANE_WORDS', True)

```

urls.py

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				93
Змн.	Арк.	№ докум.	Підпис	Дата		

```

from django.urls import path, re_path

from app_forum import views

urlpatterns = [

    path('', views.index, name='forum-index'),

    re_path(r'^(\d+)/$', views.forum, name='forum-detail'),
    path('newforum/', views.new_forum, name='new-forum'),
    path('<int:pk>/update/', views.ForumUpdateView.as_view(), name='forum-
update'),
    path('<int:pk>/delete/', views.ForumDeleteView.as_view(), name='forum-
delete'),

    re_path(r'^topic/(\d+)/$', views.topic, name='topic-detail'),
    # path('topic/<int:pk>/', views.topic, name='topic-detail'),
    path('topic/<int:pk>/update/', views.TopicUpdateView.as_view(), name='topic-
update'),
    path('topic/<int:pk>/delete/', views.TopicDeleteView.as_view(), name='topic-
delete'),
    re_path(r'^newtopic/(?P<forum_id>\d+)/$', views.new_topic, name='new-topic'),

    re_path(r'^comment/(\d+)/$', views.topic, name='comment'),
    path('comment/<int:pk>/update/', views.CommentUpdateView.as_view(),
name='comment-update'),
    path('comment/<int:pk>/delete/', views.CommentDeleteView.as_view(),
name='comment-delete'),
    path('comment/<int:id>/', views.topic, name='comment'),
]

```

views.py

```

from django.contrib.auth.decorators import login_required
# from django.core.urlresolvers import reverse
from django.core.paginator import Paginator, InvalidPage, EmptyPage
from django.http import HttpResponseRedirect
from django.shortcuts import get_object_or_404
from django.shortcuts import render, redirect
# from django.core.context_processors import csrf
from django.template.context_processors import csrf
from django.urls import reverse
from django.views.generic import UpdateView, DeleteView

from app_forum.forms import *
from app_forum.settings import *

def index(request):
    """Main listing."""
    forums = Forum.objects.all()
    forums = mk_paginator(request, forums, FORUM_PER_PAGE)
    context = {
        'forums': forums,
        'user': request.user
    }
    return render(request, 'app_forum/list.html', context)

def add_csrf(request, **kwargs):
    d = dict(user=request.user, **kwargs)
    d.update(csrf(request))
    return d

```

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				94
Змн.	Арк.	№ докум.	Підпис	Дата		

```

def mk_paginator(request, items, num_items):
    """Create and return a paginator."""
    paginator = Paginator(items, num_items)
    try:
        page = int(request.GET.get("page", '1'))
    except ValueError:
        page = 1

    try:
        items = paginator.page(page)
    except (InvalidPage, EmptyPage):
        items = paginator.page(paginator.num_pages)
    return items

def forum(request, forum_id):
    """Listing of topics in a forum."""
    topics = Topic.objects.filter(forum=forum_id).order_by("-created")
    topics = mk_paginator(request, topics, FORUM_PER_PAGE)

    forum = get_object_or_404(Forum, pk=forum_id)

    return render(request, "app_forum/forum.html",
                  add_csrf(request, topics=topics, pk=forum_id, forum=forum))

def topic(request, topic_id):
    topic_instance = get_object_or_404(Topic, pk=topic_id)
    form = CommentForm()
    add_user_form = AddUserForm()
    comments = Comment.objects.filter(topic_id=topic_id).order_by("-id")
    comments = mk_paginator(request, comments, FORUM_COMMENTS_PER_PAGE)

    if request.method == 'POST':
        if 'comment_submit' in request.POST:
            form = CommentForm(request.POST)
            if form.is_valid():
                Comment.objects.create(body=form.cleaned_data['body'],
creator=request.user, topic=topic_instance)
                return HttpResponseRedirect(reverse('topic-detail',
args=[topic_id]))

            elif 'add_user_submit' in request.POST:
                add_user_form = AddUserForm(request.POST)
                if add_user_form.is_valid():
                    username = add_user_form.cleaned_data['username']
                    try:
                        user = User.objects.get(username=username)
                        topic_instance.allowed_users.add(user)
                        return HttpResponseRedirect(reverse('topic-detail',
args=[topic_id]))
                    except User.DoesNotExist:
                        add_user_form.add_error('username', 'Користувача з таким
іменем не знайдено.')

    context = {
        'form': form,
        'add_user_form': add_user_form,
        'topic': topic_instance,
        'comments': comments,
        'user': request.user,

```

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		95

```

    }
    return render(request, "app_forum/topic.html", context)

@login_required
def new_topic(request, forum_id):
    form = TopicForm()
    forum = get_object_or_404(Forum, pk=forum_id)

    if request.method == 'POST':
        form = TopicForm(request.POST)

        if form.is_valid():
            topic = Topic()
            topic.title = form.cleaned_data['title']
            topic.description = form.cleaned_data['description']
            topic.is_private = form.cleaned_data['is_private']
            topic.forum = forum
            topic.creator = request.user

            topic.save()
            topic.allowed_users.set(form.cleaned_data['allowed_users'])

            return HttpResponseRedirect(reverse('forum-detail', args=(forum_id,)))

    context = {
        'form': form,
        'forum': forum,
    }
    return render(request, 'app_forum/new-topic.html', context)

class TopicUpdateView(UpdateView):
    model = Topic
    template_name = 'app_forum/update-topic.html'
    form_class = TopicForm

class TopicDeleteView>DeleteView):
    model = Topic
    template_name = 'app_forum/delete.html'
    success_url = '/forum/'

@login_required
def new_forum(request):
    error = ''
    if request.method == 'POST':
        form = ForumForm(request.POST)

        if form.is_valid():
            Forum.objects.create(title=request.POST.get('title'),
                                description=request.POST.get('description'),
                                creator=request.user)
            return redirect('forum-index')

    form = ForumForm()
    context = {
        'form': form,
        'author': request.user,
        'error': error
    }

```

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				96
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        return render(request, 'app_forum/new-forum.html', context)

class ForumUpdateView(UpdateView):
    model = Forum
    template_name = 'app_forum/update-topic.html'
    form_class = TopicForm

class ForumDeleteView(DeleteView):
    model = Forum
    template_name = 'app_forum/delete.html'
    success_url = '/forum/'

class CommentUpdateView(UpdateView):
    model = Comment
    template_name = 'app_forum/update-comment.html'
    form_class = CommentForm

class CommentDeleteView(DeleteView):
    model = Comment
    template_name = 'app_forum/delete.html'
    success_url = '/forum/'

```

app_users

admin.py

```

from django.contrib import admin
from app_users.models import *

admin.site.register(CustomUser)

```

forms.py

```

from django import forms
from django.contrib.auth import get_user_model
from django.contrib.auth.forms import UserCreationForm, AuthenticationForm
from django.core.exceptions import ValidationError
from django.forms import ModelForm

User = get_user_model()

class AuthForm(AuthenticationForm):
    username = forms.CharField(label='Логін',
                               widget=forms.TextInput(attrs={'class': 'form-control'}))
    password = forms.CharField(label='Пароль',
                               widget=forms.PasswordInput(attrs={'class': 'form-control'}))

    def clean_username(self):
        username = self.cleaned_data.get("username")
        user_qs = User.objects.filter(username=username)
        if user_qs.count() == 0:
            raise forms.ValidationError("Користувача не знайдено")
        return username

class RegisterUserForm(UserCreationForm):
    username = forms.CharField(label='Логін',
                               widget=forms.TextInput(attrs={'class': 'form-control'}))
    email = forms.EmailField(label='Email',

```

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				97
Змн.	Арк.	№ докум.	Підпис	Дата		

```

widget=forms.EmailInput(attrs={'class': 'form-control'}))
password1 = forms.CharField(label='Пароль',
widget=forms.PasswordInput(attrs={'class': 'form-control'}))
password2 = forms.CharField(label='Повторіть пароль',
widget=forms.PasswordInput(attrs={'class': 'form-control'}))

class Meta:
    model = User
    fields = ("username", "email", "password1", )

def clean_email(self):
    email = self.cleaned_data['email'].strip()
    if User.objects.filter(email__iexact=email).exists():
        raise ValidationError('Ця електронна скринька вже зареєстрована!')
    return email

class ChangePassUserForm(UserCreationForm):
    password1 = forms.CharField(label='Пароль',
widget=forms.PasswordInput(attrs={'class': 'form-control'}))
password2 = forms.CharField(label='Повторіть пароль',
widget=forms.PasswordInput(attrs={'class': 'form-control'}))

class Meta:
    model = User
    fields = ('password1',)

class ChangeUserStatusForm(ModelForm):
    class Meta:
        model = User
        fields = ('is_active', 'is_staff', 'is_superuser')

class UpdateUserForm(forms.ModelForm):
    username = forms.CharField(label='Логін', required=False,
widget=forms.TextInput(attrs={'class': 'form-control'}))
    email = forms.EmailField(label='Email', required=False,
widget=forms.EmailInput(attrs={'class': 'form-control'}))
    first_name = forms.CharField(label='Ім'я', required=False,
widget=forms.TextInput(attrs={'class': 'form-control'}))
    last_name = forms.CharField(label='Прізвище', required=False,
widget=forms.TextInput(attrs={'class': 'form-control'}))
    img = forms.FileField(label='Фото', required=False,
widget=forms.FileInput(attrs={'class': 'form-control'}))
    city = forms.CharField(label='Місто', required=False,
widget=forms.TextInput(attrs={'class': 'form-control'}))
    date_of_birth = forms.DateField(label='Дата народження', required=False,
widget=forms.TextInput(attrs={'class': 'form-control'}))

class Meta:
    model = User
    fields = ("username", "email", "first_name", "last_name", "img")

```

models.py

```

from django.contrib.auth.models import AbstractUser
from django.db import models

from app_articles.validators import validate_file_extension

```

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				98
Змн.	Арк.	№ докум.	Підпис	Дата		

```

class CustomUser(AbstractUser):
    subscribers = models.ManyToManyField('self', symmetrical=False, blank=True,
related_name='subscribed_to')
    img = models.FileField(verbose_name='Фото користувача',
                           default=None,
                           upload_to="user_images/%Y/%m/%d/",
                           validators=[validate_file_extension], blank=True,
null=True)
    city = models.CharField(verbose_name='Місто', max_length=50, blank=True,
null=True)
    date_of_birth = models.DateTimeField(verbose_name='Дата народження',
blank=True, null=True)

    def __str__(self):
        return self.username

```

urls.py

```

from django.contrib.auth import views as auth_views
from django.urls import path, re_path

from app_users import views
from app_users.views import *

urlpatterns = [
    path('register', RegisterUser.as_view(), name='register'),
    path('validate-username', views.ajax_username, name='ajax-username'),

    path('login', LoginUser.as_view(), name='login'),
    path('validate-login', views.ajax_login, name='ajax-login'),

    path('logout', logout_view, name='logout'),
    path('profile/<int:pk>/', UserDetailsView, name='profile'),
    # re_path(r'^profile/(\d+)/$', UserDetailsView, name='profile'),
    re_path(r'^profile/(\d+)/statistics$', profile_statistics,
name='profile_statistics'),

    path('profile-update/<int:pk>', UserUpdateView.as_view(), name='profile-
update'),
    path('change-pass/<int:pk>', ChangePassView.as_view(), name='change-pass'),

    path('status-update/<int:pk>', StatusUserUpdateView.as_view(), name='status-
update'),

    path('reset_password/',
auth_views.PasswordResetView.as_view(template_name='password_reset_form.html'),
    name='reset_password'),

    path('reset_password_sent/',
auth_views.PasswordResetDoneView.as_view(template_name='password_reset_done.html')
,
    name='password_reset_done'),

    path('reset/<uidb64>/<token>/',

auth_views.PasswordResetConfirmView.as_view(template_name='password_reset_confirm.
html'),
    name='password_reset_confirm'),

    path('reset_password_complete/',

auth_views.PasswordResetCompleteView.as_view(template_name='password_reset_complet
e.html'),

```

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				99
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        name='password_reset_complete')),

    re_path(r'^subscribe/(\d+)/$', views.subscribe_view, name='subscribe'),
    re_path(r'^unsubscribe/(\d+)/$', views.unsubscribe_view, name='unsubscribe'),
    # re_path(r'^profile/(\d+)/subscriptions$', views.subscriptions_view,
name='subscriptions'),
    path('profile/<int:pk>/subscriptions/', views.subscriptions_view,
name='subscriptions'),
]

```

views.py

```

from django.contrib.auth import logout, login
from django.contrib.auth.views import LoginView
from django.http import JsonResponse
from django.urls import reverse_lazy
from django.views.generic import CreateView

from app_articles.models import *
from app_articles.settings import *
from app_articles.utils import DataMixin
from app_books.models import *
from app_forum.views import *
from .forms import *
from .models import CustomUser

User = get_user_model()

class LoginUser(DataMixin, LoginView):
    form_class = AuthForm
    template_name = 'login.html'

    def get_context_data(self, *, object_list=None, **kwargs):
        context = super().get_context_data(**kwargs)
        c_def = self.get_user_context(title="Авторизация")
        return dict(list(context.items()) + list(c_def.items()))

    def get_success_url(self):
        return reverse_lazy('home')

class RegisterUser(DataMixin, CreateView):
    form_class = RegisterUserForm
    template_name = 'register.html'
    success_url = reverse_lazy('login')

    def get_context_data(self, *, object_list=None, **kwargs):
        context = super().get_context_data(**kwargs)
        c_def = self.get_user_context(title="Реєстрація")
        return dict(list(context.items()) + list(c_def.items()))

    def form_valid(self, form):
        user = form.save()
        login(self.request, user,
backend='django.contrib.auth.backends.ModelBackend')
        return redirect('home')

    def ajax_username(request):
        username = request.GET.get('username', None)
        response = {
            'is_taken': User.objects.filter(username__iexact=username).exists()

```

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В.В.				00
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    }
    return JsonResponse(response)

def ajax_login(request):
    username = request.GET.get('username', None)
    password = request.GET.get('password', None)
    response = {
        'is_taken': {User.objects.filter(username__iexact=username).exists(),
                     User.objects.filter(password__iexact=password).exists()}
    }
    return JsonResponse(response)

def logout_view(request):
    logout(request)
    return redirect('login')

def UserDetailsView(request, pk):
    articles = Articles.objects.filter(author_article=pk).order_by("-id")[:5]

    books_read_now = Books.objects.filter(read_now=pk).order_by("-id")[:5]
    books_read_now_count = Books.objects.filter(read_now=pk).count()

    books_want_to_read = Books.objects.filter(want_to_read=pk).order_by("-id")[:5]
    books_want_to_read_count = Books.objects.filter(want_to_read=pk).count()

    read_books = Books.objects.filter(read_books=pk).order_by("-id")[:5]
    read_books_count = Books.objects.filter(read_books=pk).count()

    top_5_read_books = Books.objects.filter(read_books=pk).order_by("rate")[:5]

    articles = mk_paginator(request, articles, MAIN_TASKS_FOR_PROFILE_PER_PAGE)
    user = User.objects.get(pk=pk)

    return render(request, "profile.html", context={
        # 'form': form,
        'articles': articles,
        'books_read_now': books_read_now,
        'books_want_to_read': books_want_to_read,
        'read_books': read_books,
        'top_5_read_books': top_5_read_books,
        'user': user,

        'books_read_now_count': books_read_now_count,
        'books_want_to_read_count': books_want_to_read_count,
        'read_books_count': read_books_count
    })

def profile_statistics(request, user_id):
    read_books_count = Books.objects.filter(read_books=user_id).count()
    read_books_date =
    Books.objects.filter(read_books=user_id).order_by("date_updated")

    return render(request, "profile_statistics.html", context={
        'read_books_date': read_books_date,
        'read_books_count': read_books_count,
    })

class StatusUserUpdateView(UpdateView):

```

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		01

```

model = User
template_name = 'status.html'
form_class = ChangeUserStatusForm
success_url = '/'

def get_context_data(self, **kwargs):
    context = super().get_context_data(**kwargs)
    context['title'] = 'Status'
    return context

# PROFILE UPDATE
class UserUpdateView(UpdateView):
    model = User
    template_name = 'update.html'
    form_class = UpdateUserForm
    success_url = "/"

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context['title'] = 'Редагування даних'
        return context

    def get_object(self, queryset=None):
        return self.request.user

class ChangePassView(UpdateView):
    model = User
    template_name = 'change-pass.html'
    form_class = ChangePassUserForm
    success_url = '/'

    def form_valid(self, form):
        user = form.save()
        login(self.request, user,
backend='django.contrib.auth.backends.ModelBackend')
        return redirect('home')

def subscribe_view(request, pk):
    user = get_object_or_404(CustomUser, pk=pk)
    user.subscribers.add(request.user)
    user.save()
    return redirect('profile', pk=pk)

def unsubscribe_view(request, pk):
    user = get_object_or_404(CustomUser, pk=pk)
    user.subscribers.remove(request.user)
    user.save()
    return redirect('profile', pk=pk)

def subscriptions_view(request, pk):
    user = get_object_or_404(CustomUser, pk=pk)
    subscribers = user.subscribers.all()
    subscriptions_count = user.subscribers.all().count()
    return render(request, "subscriptions.html", context={
        'user': user,
        'subscribers': subscribers,
        'subscriptions_count': subscriptions_count
    })

```

		Колісник О.В.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		02