Галицький фаховий коледж імені В’ячеслава Чорновола
відділення комп’ютерних технологій
циклова комісія інформатики та комп’ютерних дисциплін

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач відділення
комп’ютерних технологій
Наталія СТЕФУРАК /_____________/
(підпис)
«___» ________________ 2025 р.

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи
освітньо-професійного ступеня «фаховий молодший бакалавр»
зі спеціальності 122 «Комп’ютерні науки»
на тему: «Платформа для публікації літературної творчості»

Студентка групи КН-41	Оксана ГЕЙНА	________________________ (підпис)
Керівник роботи	Наталя КУЛЬЧИНСЬКА	________________________ (підпис)
Консультанти:		
з техніко-економічного обґрунтування	Любов МЕЛЕНЧУК	________________________ (підпис)
нормоконтролер	Надія ГАВРИШКІВ	________________________ (підпис)

Тернопіль – 2025

Галицький фаховий коледж імені В'ячеслава Чорновола
відділення комп'ютерних технологій
циклова комісія інформатики та комп'ютерних дисциплін

ЗАТВЕРДЖУЮ
Завідувач відділення
комп'ютерних технологій
Наталія СТЕФУРАК /____________/
(підпис)
«___» ________________ 2024 р.

ЗАВДАННЯ
на кваліфікаційну роботу
на здобуття освітньо-професійного ступеня «фаховий молодший бакалавр»
_____студентці Гейні Оксані Іванівні___ ____________
(прізвище, ім'я та по-батькові студента)

1. Тема роботи __Платформа для публікації літературної творчості___
затверджена наказом по коледжу від "25"_листопада_2024 р., № 253а-н
2. Термін здачі студентом завершеної роботи "___" ____________202_ р.
3. Вихідні дані до роботи ___аналіз існуючих платформ для публікації літературної творчості, дослідження технологій розробки, вимоги до застосунків такого типу.
4. Перелік питань, які повинні бути розроблені:
а) основна частина ___аналіз предметної області та огляд існуючих рішень, постановка завдання, проєктування, реалізація і тестування платформи.
б) техніко-економічне обгрунтування___аналіз ринку, розрахунок витрат та обґрунтування необхідності розробки
5. Перелік графічного матеріалу __UML-діаграма, DFD-діаграма, IDEF3-діаграма, ER-діаграми, схема навігації, макети сторінок___

Розділ	Консультанти	Підпис, дата	
		Завдання видано	Завдання прийнято
З техніко-економічного обґрунтування	Меленчук Л. І. (вчена ступінь, звання П.І.Б. консультанта)		

КАЛЕНДАРНИЙ ПЛАН

виконання кваліфікаційної роботи

№ п/п	Найменування етапу	Терміни	
		початку	завершення
1	Вибір теми кваліфікаційної роботи, ознайомлення з вимогами.	20.11.2024	29.12.2024
2	Аналіз існуючих рішень. Написання відповідного розділу	30.12.2024	27.02.2025
3	Дослідження технологій реалізації.	28.02.2025	04.04.2025
4	Розробка функціональних вимог до платформи та робота над її структурою.	05.04.2025	11.04.2025
5	Проєктування системи, її функціоналу та інтерфейсу. Написання відповідного розділу.	12.04.2025	20.04.2025
6	Встановлення та налаштування середовища розробки.	21.04.2025	22.04.2025
7	Реалізація платформи. Написання відповідного розділу.	23.04.2025	10.05.2025
8	Тестування та налагодження системи	11.05.2025	17.05.2025
9	Опрацювання економічної частини. Написання відповідного розділу.	18.05.2025	28.05.2025
10	Оформлення пояснювальної записки.	29.05.2025	15.06.2025
11	Попередній захист кваліфікаційної роботи.	13.06.2025	13.06.2025
12	Підготовка до захисту кваліфікаційної роботи.	14.06.2025	23.06.2025
13	Захист кваліфікаційної роботи.	24.06.2025	24.06.2025

Дата видачі 25 листопада 2024 р. Керівник ____________/ Кульчинська Н.З.

Завдання прийняла до виконання ________________/ Гейна О.І.

Реферат

Платформа для публікації літературної творчості. Кваліфікаційна робота. Гейна Оксана Іванівна. Галицький фаховий коледж імені В’ячеслава Чорновола, відділення комп’ютерних технологій. Спеціальність 122 «Комп’ютерні науки», 2025. Сторінок – 89, рисунків – 30, додатків – 17, таблиць – 19, джерел – 10.

Об’єкт дослідження – платформи для публікації і прочитання літературних творів.

Метою роботи є розробка функціональної кросплатформної системи, що забезпечує публікацію, читання, зберігання і оцінювання творів. У дослідженні застосовано методи аналізу предметної області, контент-аналізу, моделювання, а також елементи машинного навчання для реалізації гібридної системи рекомендацій.

Платформа для публікації літературної творчості повинна забезпечити можливість читання, публікації, редагування, зберігання та оцінювання творів, підтримку розділення тексту на розділи, фільтрацію за жанрами, тегами та статусами, а також пошук за ключовими словами. Крім того, необхідно спроєктувати та розробити інтерфейс для забезпечення можливості перегляду користувачами наявної літератури.

Для реалізації поставленої задачі було використано велику кількість різноманітних інструментів і шаблонів, доступних у середовищах розробки Flutter та Python, а також мови Dart і SQL.

Результатом розробки стала платформа рекомендована для використання авторами та читачами художніх творів.

ЛІТЕРАТУРНА ПЛАТФОРМА, ПУБЛІКАЦІЯ ТВОРІВ, СИСТЕМА РЕКОМЕНДАЦІЙ, FLUTTER, БАЗА ДАНИХ, МАШИННЕ НАВЧАННЯ, FAST API, FLUTTER, КЛІЄНТ-СЕРВЕРНА АРХІТЕКТУРА.

Abstract

A platform for publishing literary works. Qualification work. Heina Oksana Ivanivna. Halytsky applied college named after Vyacheslav Chornovil, Department of Computer Technologies. Speciality 122 «Computer Science», 2025. Pages – 89, figures – 30, appendices – 17, tables – 19, sources – 10.

The object of research is platforms for publishing and reading literary works.

The purpose of the study is to develop a functional cross-platform system that provides publication, reading, storage and evaluation of works. The study applied the methods of domain analysis, content analysis, modelling, and machine learning elements to implement a hybrid recommendation system.

The platform for publishing literary works should provide the ability to read, publish, edit, store and evaluate works, support text division into sections, filtering by genre, tag and status, and keyword search. In addition, it was necessary to design and develop an interface to enable users to browse the available literature.

To achieve this task, we used a large number of different tools and templates available in the Flutter and Python development environments, as well as Dart and SQL.

The result of the development is a platform recommended for use by authors and readers of literary works.

LITERARY PLATFORM, PUBLISHING WORKS, RECOMMENDATION SYSTEM, FLUTTER, DATABASE, MACHINE LEARNING, FAST API, FLUTTER, CLIENT.

ЗМІСТ

Вступ 7
1 Аналіз існуючих рішень і постановка завдання 8
1.1 Опис предметної області 8
1.2 Аналіз наявних рішень 10
1.3 Аналіз вимог до програмного засобу та постановка завдання 18
2 Проєктування системи 21
2.1 Модель системи 21
2.2 Проєктування бази даних 25
2.3 Проєктування інтерфейсу 33
2.4 Дослідження технологій та засобів реалізації застосунку 38
3 Реалізація та тестування системи 40
3.1 Вибір та обґрунтування засобів реалізації застосунку 40
3.2 Реалізація бази даних 42
3.3 Розробка платформи 46
3.3 Тестування програмного засобу 67
4 Техніко-економічне обґрунтування 78
4.1 Аналіз ринку 78
4.2 Економічне обґрунтування розробки 79
4.3 Обґрунтування необхідності розробки 86
Висновки 88
Перелік джерел посилання 89
Додатки 90

Змн.	Арк.	№ докум.	Підпис	Дата	КР.КН 25.588.07.000 ПЗ			
Розроб.		Гейна О.І.			Платформа для публікації літературної творчості	Літ.	Арк.	Акрушів
Перевірка		Кульчинська Н.З					5	89
Рецензент		Сиротюк О.Б.				ГФК.ВКТ. КН-41		
Норм. контр.		Гавришків Н.Г.						
Зав. від.		Стефурак Н.А.						

СКОРОЧЕННЯ І УМОВНІ ПОЗНАКИ

БД – База даних

ПДВ – Податок на додану вартість

ПДФО – Податок на доходи фізичних осіб

СУБД – Система управління базами даних

ЄСВ – Єдиний соціальний внесок

API – Application Programming Interface

AUR – Arch User Repository

AWS – Amazon Web Services

CORS – Cross-Origin Resource Sharing

HTTP – Hypertext Transfer Protocol

HTTPS – HyperText Transfer Protocol Secure

JSON – JavaScript Object Notation

IDF – Inverse Document Frequency

PHP – Hypertext Preprocess

REST – REpresentational State Transfer

TF – Term Frequency

UML – Unified Modeling Language

UI – User Interface

UUID – Universally Unique Identifier

ВСТУП

Розробка платформи для публікації літературної творчості є дуже важливою в сучасному цифровому світі. Цифрові технології кардинально змінили світ літератури, відкривши багато нових шансів для письменників і читачів, але разом з тим принесли й нові труднощі.

Зараз, коли інформаційні технології швидко розвиваються і самвидав стає все популярнішим, створення спеціального онлайн-простору для авторів набуває особливого значення. Сучасний читач очікує миттєвого доступу до контенту, можливості взаємодії з авторами та участі в літературному процесі.

Традиційні шляхи публікації через видавництва часто забирають багато часу і є складними, що обмежує можливості багатьох письменників-аматорів поділитися своїми творами з читачами. Класична модель видавничої індустрії передбачає багаторівневу систему відбору, редагування та просування, яка може тривати місяці або навіть роки. Для молодих авторів це створює значні бар'єри входу в літературний простір. Крім того, видавництва часто орієнтуються на комерційно успішні жанри, залишаючи поза увагою експериментальні або нішеві твори.

Наявні рішення часто не враховують потреби авторів-початківців, мають обмежені можливості або недостатньо адаптовані для україномовної аудиторії. Створення платформи, яка поєднує функції публікації, редагування та отримання відгуків, зробить літературну діяльність доступнішою для багатьох авторів і допоможе розвитку сучасної української літератури.

Метою роботи є розробка та впровадження веб-платформи для публікації літературної творчості, яка забезпечує повну взаємодію між авторами та читачами, від створення та публікації творів до отримання відгуків.

Створення такої платформи допоможе зробити літературний процес доступнішим, знизити перешкоди для нових авторів та створити активну спільноту читачів і письменників в українському інтернеті.

1 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ І ПОСТАНОВКА ЗАВДАННЯ

1.1 Опис предметної області

Важливою частиною культурного розвитку суспільства є літературна творчість. Створення нової платформи для читання і розміщення аматорських творів є доцільним з огляду на зростаючий попит на такі послуги та наявні недоліки існуючих рішень. Сучасний літературний ринок демонструє стійку тенденцію до збільшення кількості авторів-аматорів, які прагнуть поділитися своєю творчістю та отримати визнання. Водночас, читацька аудиторія виявляє інтерес до нового, оригінального контенту, який часто не потрапляє до традиційних видавництв.

Наявні платформи не повністю задовольняють потреби як авторів, так і читачів. Зокрема, автори-початківці часто стикаються з труднощами у просуванні своїх творів та отриманні конструктивного зворотного зв'язку. Читачі, у свою чергу, мають проблеми з пошуком якісного контенту серед великої кількості творів. Створення нової системи дозволить вирішити ці проблеми, запропонувавши інноваційні підходи до організації літературної спільноти, навчання авторів та взаємодії між учасниками платформи.

Немає такого автора, який би не мріяв побачити свій витвір на полицях книгарень. У зв'язку з цим багато молодих письменників задаються питанням, як видати власну книгу. Перше, що спадає на думку, відправити рукопис до авторитетного видавництва. Однак, як показує практика, далеко не всі видавництва охоче співпрацюють із новачками. Крім того, не кожен автор може презентувати своє творіння так, щоби видавець зацікавився ним. Тим не менш, це не привід опускати руки.

В умовах стрімкого розвитку цифрових технологій та зростання популярності самвидаву, існує стійкий попит на платформи для публікації літературної творчості. Цифровий формат значно розширює можливості літературного процесу, усуваючи традиційні бар'єри між авторами та читачами.

Розвиток інтернет-технологій призвів до формування нової культури споживання літературного контенту. Сучасні читачі все частіше надають перевагу онлайн-форматам, що забезпечують миттєвий доступ до творів та можливість інтерактивної взаємодії з авторами.

Кількість українців, які читають щодня, за три роки зросла удвічі – до 17%. Однак популярність друкованих книжок значно знизилася, з 39% до 21%. Хоча українці все ще часто обирають друковані книги, їхня популярність поступово зменшується [1].

Особливо актуальною є потреба у створенні платформи для україномовного сегменту літературного ринку. Розвиток національної літератури потребує сучасних інструментів для публікації та просування творів українських авторів, що сприятиме збагаченню культурного простору та підтримці молодих талантів.

До повномасштабного вторгнення велика кількість українців користувались російськими платформами для читання чи розміщення творів. Однак, у зв'язку з відмовою багатьох українців від російського контенту та зростанням суспільного інтересу до розвитку україномовного контенту, виникла необхідність у створенні власних альтернатив [2]. Відсутність якісних національних платформ стримує розвиток літературної спільноти, змушуючи авторів шукати інші шляхи для публікації та просування своїх творів.

Технологічний аспект створення такої платформи підкріплюється наявністю сучасних інструментів розробки, що дозволяють забезпечити надійне функціонування системи та її подальше масштабування. Впровадження інноваційних технологій сприятиме покращенню користувацького досвіду та підвищенню ефективності взаємодії між учасниками платформи.

Соціальна значимість проєкту полягає у створенні середовища для розвитку літературної творчості, підтримки культурного різноманіття та формування активних читацьких спільнот. Платформа може стати важливим інструментом для популяризації читання та підтримки літературної освіти.

Створення подібної системи є своєчасним та обґрунтованим кроком, що відповідає сучасним тенденціям розвитку цифрового контенту та потребам літературної спільноти. Реалізація проєкту сприятиме розвитку цифрової культури та створенню нових можливостей для творчої самореалізації.

1.2 Аналіз наявних рішень

В Інтернеті вже існує низка платформ, які пропонують подібний функціонал та можливості для публікації літературної творчості. Ці платформи мають різні особливості, зокрема різний рівень зручності користування, можливості для авторів, систему монетизації та модерації контенту.

Щоб створити власний проєкт, важливо провести детальний аналіз існуючих рішень, оцінити їх переваги та недоліки, а також зрозуміти, які функції є найбільш затребуваними серед користувачів. Такий підхід дозволить визначити, які рішення можна використати при створенні нової платформи, щоб зробити її більш конкурентоспроможною і привабливою для потенційних користувачів.

Для аналізу обрано найбільш популярні платформи для публікації літературної творчості, що представлені на ринку: міжнародну платформу Wattpad, що має найбільшу користувацьку базу та різноманітний функціонал, українські платформи Arkush.net та Гоголівська академія.

Аналіз цих платформ дозволить визначити їхні сильні та слабкі сторони, а також виявити потенційні напрямки для вдосконалення при розробці нової системи.

Wattpad є однією з найбільших міжнародних платформ для публікації літературної творчості, що має як мобільний застосунок, так і веб-версію [3]. Попри широку популярність, платформа має низку суттєвих недоліків у користувацькому досвіді.

Головною проблемою платформи є складність пошуку якісного контенту. Система рекомендацій працює недостатньо ефективно, а можливості сортування дуже обмежені – доступні лише фільтри «популярне» та «нове» (рисунок 1.1).

Після вибору жанру функціонал пошуку та фільтрації залишається мінімальним, користувач може лише додати запропоновані системою теги. Алгоритм пошуку часто не дає релевантних результатів.

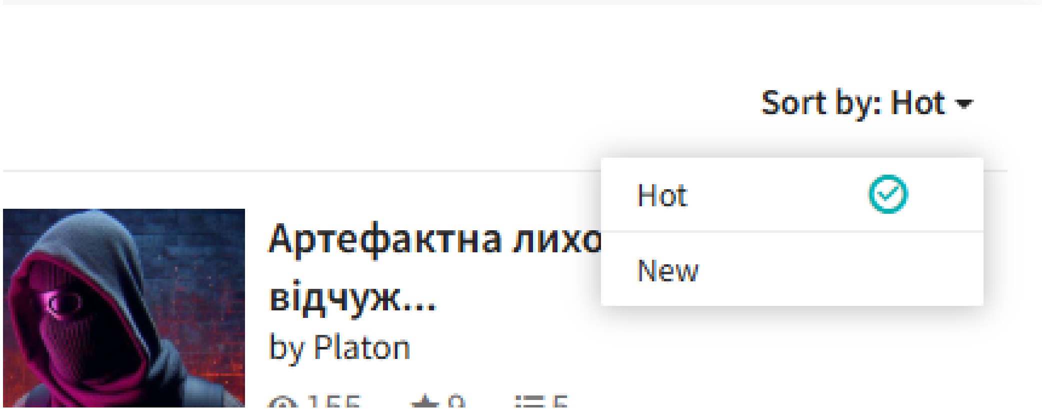

Рисунок 1.1 – Сортування у Wattpad

Літературна платформа здебільшого спрямована на підліткову аудиторію та включає переважно твори з простими сюжетами. Значну частину контенту складають неякісні роботи, а також неправомірно розміщені копії опублікованих книг, які модерація не видаляє.

На платформі також представлені твори українських авторів. Україномовний сегмент також страждає від низької якості більшості представлених творів.

Щодо функціоналу для авторів, платформа надає базові інструменти форматування тексту (жирний, курсив, підкреслення, вирівнювання), можливість додавання медіа-контенту та налаштування додаткових даних твору (назва, опис, персонажі, теги, цільова аудиторія, мова).

Наявна функція відкладеної публікації та можливість редагування опублікованого контенту (рисунок 1.2). Однак у мобільній версії сайту є проблеми з доступністю функції створення творів – відсутня кнопка публікації.

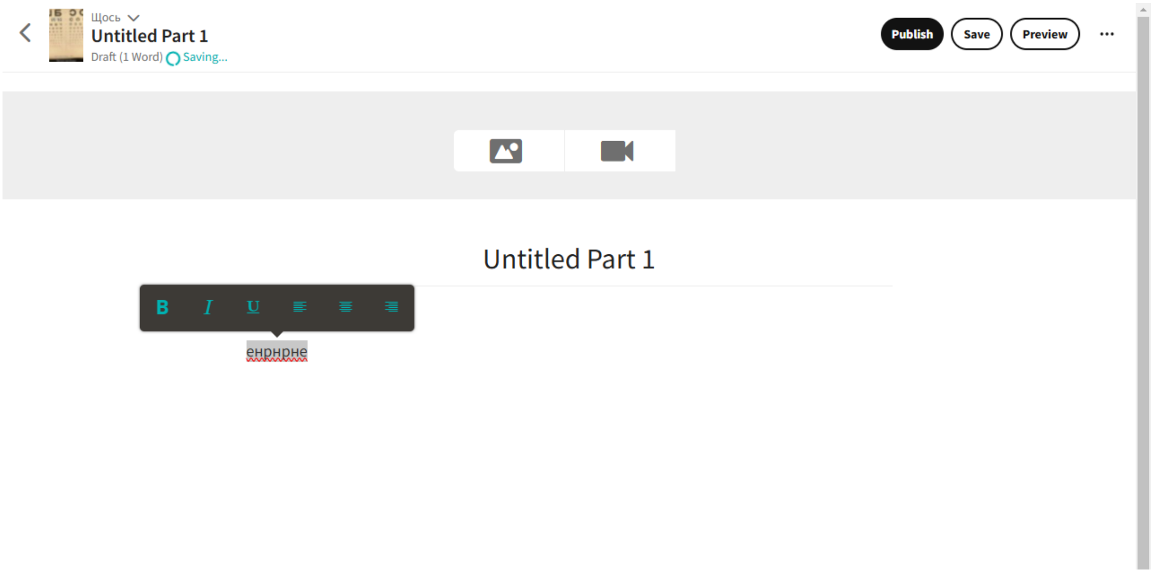

Рисунок 1.2 – Текстовий редактор у Wattpad

Платформа пропонує систему налаштувань приватності, що дозволяє блокувати небажані теги та контент для дорослих, а також окремі акаунти. Доступна функція збереження творів для офлайн-читання в мобільному застосунку, хоча з обмеженням кількості творів.

Монетизація реалізована через систему преміум-підписок з різними рівнями. Базовий преміум надає доступ до версії без реклами та розширює ліміт офлайн-завантажень. Існує також розширений преміум+ з системою внутрішньої валюти, проте її функція не очевидна для нових користувачів.

Технічні аспекти платформи демонструють якісну адаптивність під різні пристрої, проте мають певні недоліки. Зокрема, функція завантаження тексту у файл відсутня або важкодоступна для користувачів. Крім того, система іноді некоректно визначає мову публікації, що призводить до додаткових незручностей для авторів і може впливати на правильність відображення контенту. Ці технічні недоліки створюють проблеми для тих, хто хоче зручно працювати з платформою та забезпечити належний рівень представленості своїх творів.

Попри наявність успішних випадків співпраці з видавництвами та екранізацій окремих творів, загальна якість контенту та користувацький досвід на платформі залишаються на низькому рівні, особливо для користувачів, які шукають якісну літературу поза підлітковим сегментом.

Загалом, попри популярність Wattpad, платформа потребує суттєвих покращень у функціональності та якості контенту для забезпечення кращого користувацького досвіду та більш широкої аудиторії, включаючи тих, хто шукає літературу поза підлітковим сегментом.

Платформа «Гоголівська академія», заснована братами Капрановими, є онлайн майданчиком для письменників, але має низку суттєвих обмежень та проблемних моментів [4].

Дизайн сайту справляє украй негативне враження. Застарілий інтерфейс ускладнює навігацію та сприйняття контенту. Проблеми включають нечитабельний шрифт, некомфортний інтервал між літерами, незручне розташування елементів та відсутність сучасних дизайнерських рішень. Світла тема без можливості її зміни додатково погіршує досвід користування додатком (рисунок 1.3).

Рисунок 1.3 – Головна сторінка "Гоголівська академія"

Функціональні обмеження платформи «Гоголівська академія» є серйозною проблемою для користувачів. Вони виглядають надзвичайно обмеженими і застарілими в порівнянні з сучасними вимогами до літературних майданчиків. Система публікації творів не має базових інструментів, які стали стандартом на інших платформах. Наприклад, відсутність опису творів позбавляє можливості для авторів надавати більш детальну інформацію про свої роботи, що призводить до зниження зацікавленості читачів. Теги, які допомагають класифікувати твори, також відсутні, що ускладнює пошук і вибір літератури.

Додатково, відсутність вікових обмежень на твори може призвести до того, що непідготовлені читачі натраплять на контент, який не відповідає їхнім віковим категоріям або емоційним потребам. Ще одним великим недоліком є майже повна відсутність можливостей сортування та фільтрації. Користувачі не можуть вибирати твори за жанром, рейтингом або іншими важливими критеріями. Це створює відчуття хаосу, коли користувачі не можуть швидко знайти те, що їх цікавить, і змушує витрачати значну кількість часу на перегляд величезної кількості контенту, що не відповідає їхнім уподобанням. Всі ці обмеження роблять платформу значно менш зручною та конкурентоспроможною на фоні більш розвинених ресурсів.

Форма публікації на платформі «Гоголівська академія» є надзвичайно спрощеною і зведена до мінімуму. Для того щоб опублікувати твір, автору достатньо вказати лише назву, жанр і сам текст. Такий підхід абсолютно не враховує сучасні вимоги до публікацій на літературних платформах, де зазвичай є можливість додавати опис твору, вказувати теги для кращої класифікації, вікові обмеження. Відсутність таких функцій обмежує можливості авторів для кращого представлення свого контенту та робить твори менш доступними для потенційних читачів.

Крім того, під час спроби публікації на платформі інколи виникають необроблені помилки, які ускладнюють процес. На рисунку 1.4 видно приклад такої помилки, що ще більше підкреслює недоліки платформи в її технічній реалізації.

Рисунок 1.4 – Помилка при спробі публікації у «Гоголівська академія»

Платформа «Гоголівська академія» також має серйозні проблеми з безпекою, які можуть ставити під загрозу користувачів. Один із найбільших недоліків – використання протоколу HTTP замість безпечного HTTPS. Це робить платформу вразливою до різноманітних атак, зокрема до перехоплення даних, що передаються між користувачами та сервером.

Додатково, реєстраційна форма на платформі має суттєві вади безпеки: відсутні будь-які обмеження на складність паролів. Це дозволяє користувачам створювати надто прості паролі, що робить облікові записи вразливими до злому.

Позитивним моментом платформи є наявність спільнот, що дозволяє користувачам взаємодіяти, обмінюватися ідеями та отримувати зворотний зв'язок. Спільноти створюють можливість для авторів і читачів бути частиною активної літературної спільноти, де кожен може висловлювати свою думку про твори, ставити запитання та ділитися власними поглядами.

Платформа «Гоголівська академія» має досить багато важливих недоліків, зокрема обмежену функціональність, проблеми з безпекою та стабільністю роботи системи.

Однак є й позитивні моменти, такі як наявність спільнот, що дають можливість отримати зворотний зв'язок від інших користувачів і сприяють розвитку творчого потенціалу.

Загалом, хоча платформа має певні переваги, вона потребує значних покращень у функціональності та безпеці, щоб відповідати вимогам сучасних літературних ресурсів.

Arkush.net є українською літературною платформою, яка вирізняється чіткою проукраїнською позицією та сприяє розвитку української літератури [5]. Незважаючи на те, що вона знаходиться в альфа-версії, платформа вже зараз демонструє високий рівень функціональності та зручності використання, що робить її привабливою для користувачів. Платформа пропонує різноманітний контент, включаючи книги, вірші та аудіокниги, що дозволяє задовольнити потреби різних категорій читачів.

Система навігації та читання реалізована на достатньо високому рівні, забезпечуючи комфортний користувацький досвід. Особливо варто відзначити зручність інтерфейсу для читання творів, що є важливою перевагою порівняно з конкурентними платформами. Соціальна складова включає розвинену систему взаємодії з можливістю ставити лайки як цілим творам, так і окремим розділам, додавати книги до бібліотеки, отримувати сповіщення та спілкуватися у приватних чатах. Реалізована можливість прямої комунікації між читачами та авторами через особисті повідомлення сприяє розвитку активної спільноти.

Платформа пропонує зручну систему відстеження читацької активності, що є корисним інструментом як для читачів, так і для авторів. Користувачі мають змогу переглядати історію прочитаних книг, що дозволяє зручно повернутися до раніше прочитаного контенту або швидко знайти твори, які вони хочуть перечитати. Крім того, є можливість створювати списки бажаних для прочитання творів, що допомагає організувати свої майбутні літературні плани та легше групувати цікаві книги серед великої кількості контенту на платформі.

Щодо організації контенту, Arkush.net використовує базові інструменти категоризації, поділяючи твори за жанрами та віковими обмеженнями. Доступне сортування за оновленнями та вподобаннями, але система фільтрації обмежується лише жанровим поділом, що може ускладнювати пошук конкретних творів за іншими критеріями.

Серед технічних аспектів, які потребують вдосконалення, варто відзначити кілька важливих недоліків, що ускладнюють користування платформою. Зокрема, відсутність можливості авторизації через соціальні мережі є значним обмеженням для користувачів, особливо тих, хто звик швидко реєструватися та входити через популярні платформи, такі як Facebook, Google та інші.

Важливою перевагою платформи Arkush.net є її активна модерація контенту, що гарантує високу якість публікацій та відповідність правилам спільноти. Це забезпечує довіру з боку користувачів, оскільки вони можуть бути впевнені в тому, що контент, який вони знаходять, відповідає вимогам і не містить нецензурної та тієї, яка порушує правила платформи інформації або матеріалів, які не відповідають етичним стандартам. Така модерація сприяє створенню здорової атмосфери в спільноті та підтримці високого рівня творчості.

Наявність великої кількості безкоштовних книг різних жанрів робить платформу особливо привабливою для широкої читацької аудиторії. Користувачі мають змогу безкоштовно ознайомитися з різноманітним контентом, що дозволяє знайти твори на будь-який смак.

У порівнянні з іншими платформами в даному сегменті Arkush.net демонструє кращу реалізацію пошуку та адаптивність для мобільних пристроїв, що є суттєвою перевагою в сучасних умовах споживання контенту.

Таким чином, всі три платформи мають свої сильні та слабкі сторони. Wattpad і Arkush.net пропонують потужні можливості для соціальної взаємодії та надають великий обсяг контенту, проте страждають від певних технічних та функціональних обмежень. Платформа «Гоголівська академія», з огляду на свою примітивну структуру та низьку якість функціональності, потребує значних удосконалень для того, щоб стати конкурентоспроможною на ринку літературних платформ.

У таблиці 1.1 наведено порівняння раніше проаналізованих платформ для публікації літературної творчості, що дозволяє оцінити ключові аспекти кожної з них. Це порівняння дає змогу виділити сильні та слабкі сторони кожної платформи.

Таблиця 1.1 – Перелік недоліків та переваг аналогів

Характеристика	Wattpad	Arkush.net	Гоголівська академія
Текстовий редактор	Базове редагування	Розширені функції	Дуже обмежено
Система рекомендацій	Присутня, неефективна	Присутня	Відсутня
Модерація контенту	Присутня, недієва	Присутня	Відсутня
Мобільна версія	Присутня	Присутня	Відсутня
UI/UX	Сучасний, зручний	Сучасний, зручний	Застарілий, незручний

Аналіз наявних платформ для публікації літературної творчості дозволив визначити їхні ключові переваги та недоліки. Він дозволив виділити напрямки для покращення власної платформи, зокрема ефективність пошуку та рекомендацій, а також можливості взаємодії між авторами та читачами.

1.3 Аналіз вимог до програмного засобу та постановка завдання

На основі аналізу наявних платформ для публікації літературної творчості сформовано вимоги до розроблюваної системи. Головна мета програмного засобу – створення зручної та функціональної платформи для авторів і читачів, яка забезпечуватиме ефективний пошук контенту, взаємодію між користувачами та гнучкі інструменти для роботи з текстами.

Платформа має надавати зручний механізм пошуку творів за різними параметрами, такими як назва, жанр, автор та теги. Додатково повинна бути реалізована система фільтрації за жанрами, тегами, попередженнями, віковими категоріями, видом стосунків, станом твору та сортування за кількістю переглядів, вподобаних, рейтингом, датою оновлення та завершення, кількістю сторінок, що забезпечить користувачам швидкий доступ до релевантного контенту. Важливим аспектом є впровадження персоналізованих рекомендацій на основі аналізу читацьких уподобань.

Для підвищення взаємодії між користувачами має бути передбачена можливість ділитися творами у соціальних мережах та отримувати сповіщення про нові публікації, коментарі чи оновлення у творах. Інтерфейс системи має бути адаптивним, підтримувати зміну мови та теми для зручності користувачів.

Платформа повинна відповідати віковим обмеженням, допускаючи реєстрацію користувачів від 13 років, а також містити механізм попередження перед відкриттям контенту з позначкою «18+». Реєстрація має здійснюватися також через Google, що спрощує доступ до сервісу.

Для авторів має бути передбачена система керування творами з обов'язковими параметрами, такими як назва, опис, жанри та мова, а також можливість додавання необов'язкових параметрів, таких як обкладинка, вікове обмеження та тропи. Необхідно реалізувати мітки статусу твору (завершено/незавершено) та список оновлень. Також важливо передбачити систему чернеток, онлайн-редактор тексту та статистику переглядів.

Функціонал для читачів повинен включати можливість збереження творів у закладках («Улюблене», «Прочитати пізніше»), перегляд історії читання та систему досягнень, яка мотивуватиме активність на платформі. Важливим елементом є впровадження механізмів модерації контенту, що сприятиме підтримці якості та дотриманню правил спільноти.

Безпека користувачів буде забезпечуватись політикою конфіденційності, механізмами контролю доступу до творів та можливістю видалення облікового запису з подальшим експортом даних.

Технічні вимоги включають підтримку офлайн-режиму читання, що забезпечить доступ до завантажених творів без підключення до інтернету.

На основі проведеного аналізу було визначено ключові вимоги до програмного засобу для публікації літературної творчості. Основний акцент робиться на зручності користування, розширених можливостях пошуку та персоналізації контенту, інтерактивності та забезпеченні високого рівня безпеки.

Для ефективної навігації платформа повинна мати розширену систему пошуку та фільтрації за жанрами, тегами, віковими категоріями, станом твору та іншими параметрами. Це дозволить користувачам швидко знаходити релевантний контент. Додатково має бути впроваджена система рекомендацій, яка буде аналізувати вподобання користувачів і пропонувати їм відповідні твори.

З метою підвищення взаємодії між користувачами планується реалізація можливості коментування, сповіщень про оновлення творів, а також інтеграції з соціальними мережами для поширення контенту. Інтерфейс має бути адаптивним та налаштованим, з підтримкою зміни мови та теми оформлення.

Для авторів повинна бути передбачена система керування творами, що включає можливість додавання опису, обкладинок, тегів, жанрів та інших параметрів. Важливою функцією є підтримка чернеток, онлайн-редагування тексту та перегляд статистики переглядів.

Значну увагу приділено безпеці: повинні бути впроваджені механізми конфіденційності, контролю доступу до творів та можливість повного видалення облікового запису разом із даними.

Таким чином, розроблювана платформа покликана усунути недоліки наявних рішень та запропонувати зручний, безпечний і функціональний сервіс для авторів та читачів.

2 ПРОЄКТУВАННЯ СИСТЕМИ

2.1 Модель системи

Архітектура системи визначає загальну структуру програмного засобу, його основні компоненти, взаємозв’язки між ними та спосіб взаємодії користувачів із платформою. Вона розроблена з урахуванням функціональних вимог до системи та забезпечує її масштабованість, продуктивність і зручність використання.

Система передбачає три основні категорії користувачів: читач, автор та модератор. Читач має доступ до базового функціоналу платформи, зокрема перегляду творів, коментування, збереження у вибране та взаємодії з іншими користувачами. У будь-який момент читач може стати автором, опублікувавши свій перший твір. Таким чином, роль автора є розширенням ролі читача, що дозволяє користувачам створювати, редагувати та керувати власними творами.

Модератор виконує функції, які включають управління контентом, модерацію користувачів і коментарів, а також забезпечення дотримання правил платформи. На рисунку 2.1 представлено взаємодію користувачів із системою, де зображено основні сценарії використання: перегляд творів, коментування, публікація, модерація та інші.

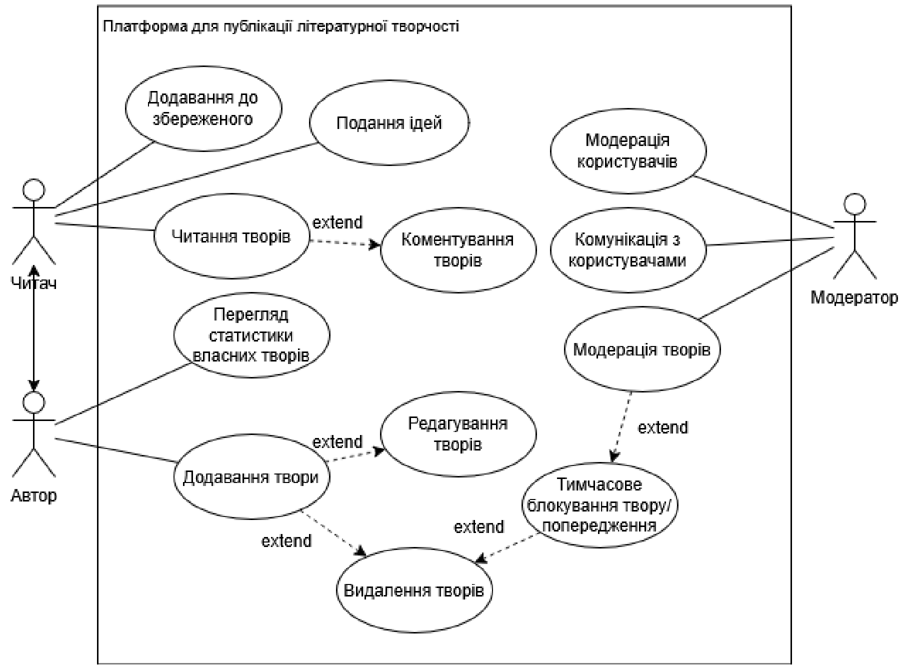

Рисунок 2.1 – UML-діаграма варіантів використання

Функції системи зображені у вигляді функціональних блоків, кожен із яких виконує окремі завдання. На рисунку 2.2 представлена контекстна діаграма IDEF0, яка включає основні завдання системи. Вхідні дані формуються з рукописів та користувацьких даних, що надходять у систему, а також регулюються правилами публікації та політикою безпеки. На виході отримуємо опубліковані твори, відгуки та статистичні дані, що забезпечують зворотний зв'язок для авторів та модераторів. Додатковий контроль здійснюється модераторами, а якість контенту та його популярність впливають на систему рейтингу.

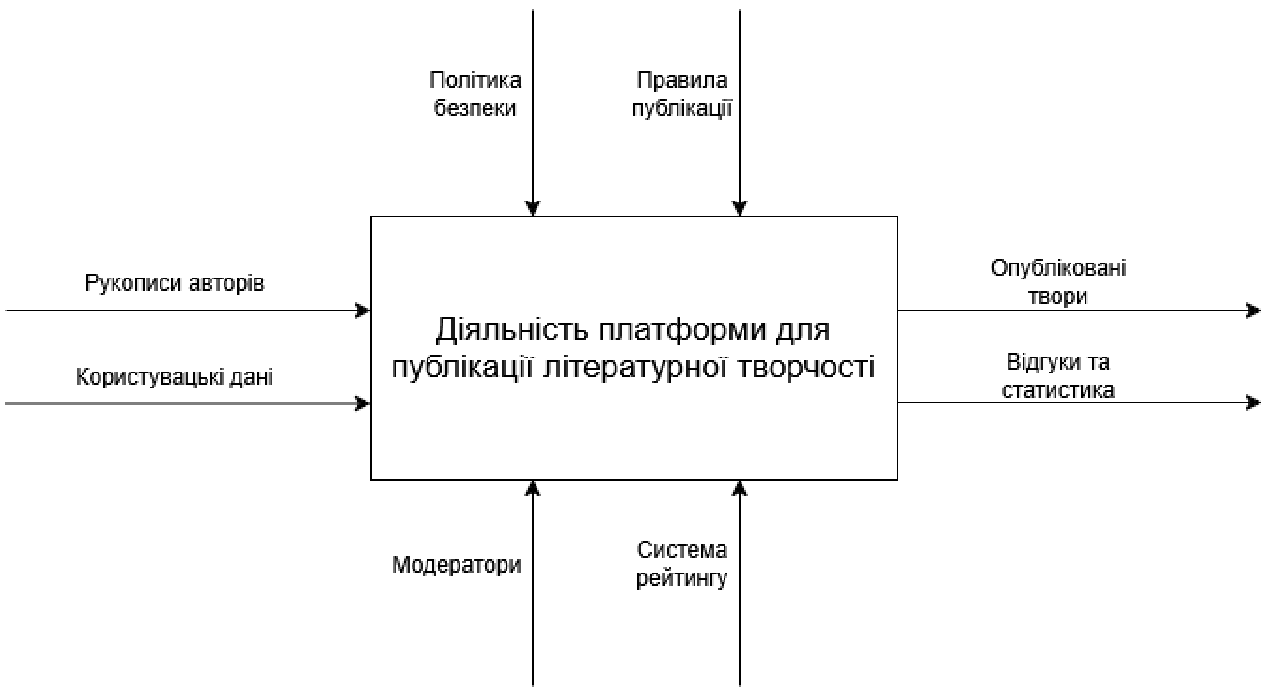

Рисунок 2.2 – Контекстна діаграма

На рисунку 2.3 деталізується процес публікації твору шляхом його декомпозиції. Взаємодія починається з реєстрації користувача та надсилання рукопису, після чого проводиться перевірка відповідності правилам публікації та політиці безпеки. Далі відбувається етап редагування та додавання супровідної інформації, після чого твір проходить через систему модерації. У разі схвалення він стає доступним для читачів, а його популярність впливає на статистику та рейтинг автора, який відображається на платформі. Таким чином, кожен етап дозволяє контролювати якість контенту та підтримувати відповідність стандартам платформи.

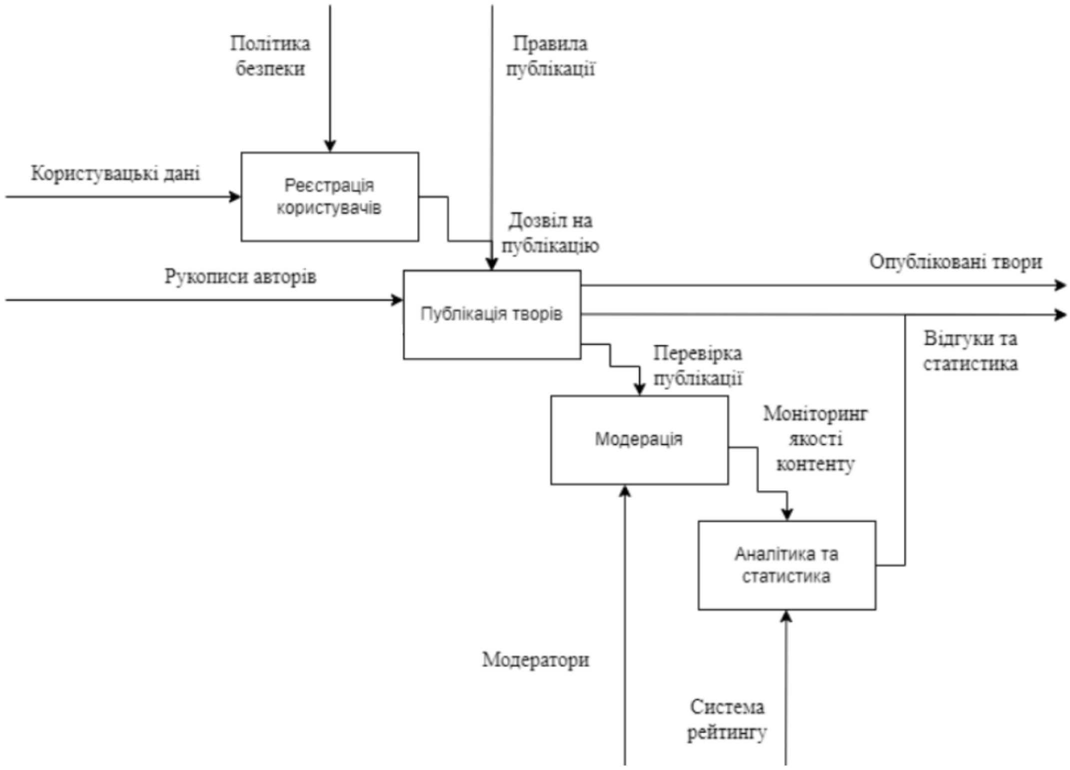

Рисунок 2.3 – Діаграма декомпозиції

Для опису послідовності виконання операцій у системі створено діаграму IDEF3, яка зображує інформаційні потоки, взаємини між процесами обробки інформації та об’єктів, що є частиною цих процесів. Вона охоплює такі ключові події, як реєстрація користувача, публікація твору, прочитання та коментування, а також модерація контенту. IDEF3-діаграма зображена на рисунку 2.4.

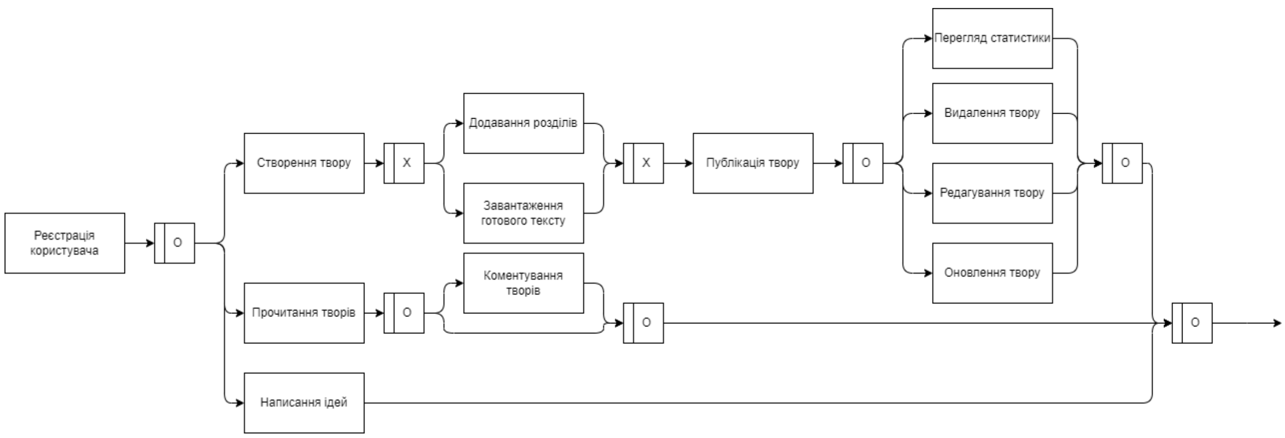

Рисунок 2.4 – IDEF3-діаграма

Реєстрація користувача передбачає створення облікового запису, авторизацію та налаштування профілю. Процес створення та публікації твору містить написання чи завантаження розділу та збереження у базі даних. Читачі мають змогу читати твори та при бажанні коментувати їх.

Також на рисунку 2.5 представлено діаграму потоків даних (DFD), яка описує процес реєстрації користувачів на платформі. На діаграмі показано, як користувачі взаємодіють з системою під час створення облікового запису, включаючи введення персональних даних, валідацію введеної інформації та збереження даних у базі даних. Потоки даних демонструють важливі етапи перевірки та підтвердження інформації. Цей процес забезпечує коректну реєстрацію користувачів і підготовку до подальшої взаємодії з платформою.

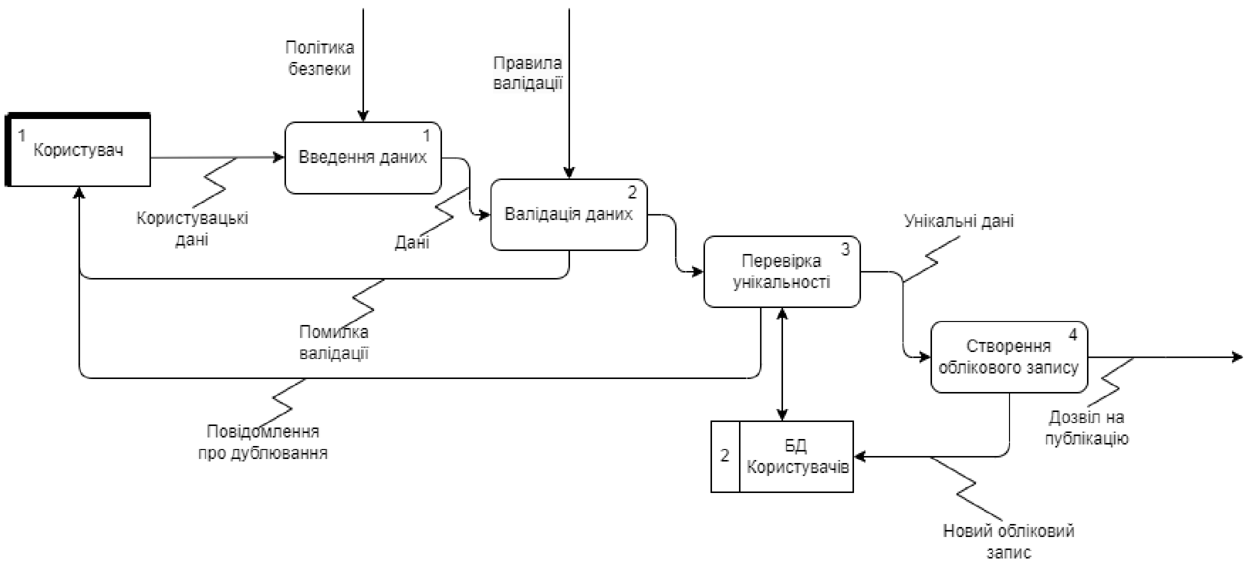

Рисунок 2.5 – DFD-діаграма потоків даних

Загальна архітектура системи спрямована на забезпечення ефективної взаємодії між користувачами, зручності використання та можливості масштабування. Використані методи проєктування дозволяють створити гнучку та функціональну платформу для публікації літературної творчості. Структура системи організована у вигляді взаємопов’язаних модулів, кожен з яких відповідає за певні аспекти її роботи: обробку запитів користувачів, управління даними, авторизацію, модерацію, аналітику та інші функціональні напрямки. Це дозволяє не лише розмежувати логіку платформи, а й забезпечити її розширюваність у майбутньому, залежно від потреб користувачів або технічних вимог.

2.2 Проєктування бази даних

Для забезпечення надійного зберігання та ефективної обробки даних у системі було здійснено проєктування бази даних. База даних є основним компонентом, який відповідає за збереження інформації про користувачів, твори, розділи, взаємодії та інші об'єкти платформи.

У структурі бази даних системи публікації літературної творчості однією з головних сутностей є користувачі. Сутність «Користувачі» зберігає персональні дані користувачів, такі як ім’я, прізвище, унікальне ім’я користувача, електронну адресу, номер телефону, шлях до аватару, рік народження та біографічну інформацію. Для автентифікації також використовується поле пароля. Це відображено в таблиці 2.1.

Таблиця 2.1 – Атрибути сутності users

Назва поля	Тип даних	Опис
id	UUID [pk]	Унікальний ідентифікатор
name	VARCHAR(50)	Ім’я користувача
last_name	VARCHAR(50)	Прізвище
username	VARCHAR(50)	Унікальний логін
email	VARCHAR(100)	Унікальна пошта
password	VARCHAR(50)	Пароль
phone_number	VARCHAR(15)	Номер телефону
avatar_path	VARCHAR(150)	Шлях до аватара
birth	INT	Рік народження
bio	TEXT	Біографія

Сутність works відповідає за представлення літературних творів, кожен з яких має назву, опис, обкладинку, шлях до файлів, дату створення та оновлення, а також належить до певної категорії. Також зберігається вікове обмеження та статус роботи, для чого використовується зовнішній ключ status_id, який посилається на сутність work_statuses. Ця сутність наведена в таблиці 2.2.

Таблиця 2.2 – Атрибути сутності works

Назва поля	Тип даних	Опис
id	UUID [pk]	Ідентифікатор твору
title	VARCHAR(100)	Назва твору
author	UUID	Ідентифікатор автора (користувача)
description	TEXT	Опис твору
cover_path	VARCHAR(150)	Шлях до обкладинки
file_path	VARCHAR(150)	Шлях до основного файлу
created_at	DATETIME	Дата створення
updated_at	DATETIME	Дата останнього оновлення
category_id	INT	Ідентифікатор категорії
age_limit	INT	Вікове обмеження
status_id	INT	Ідентифікатор статусу твору

У таблиці 2.3 представлено сутність work_statuses, яка містить перелік можливих статусів творів на платформі. Ця сутність відіграє важливу роль у системі, оскільки дозволяє класифікувати твори за їхнім поточним станом або етапом готовності. Завдяки цьому користувачі можуть легко фільтрувати та знаходити твори, які знаходяться, наприклад, у процесі написання, вже завершені або знаходяться на стадії редагування.

Таблиця 2.3 – Атрибути сутності work_statuses

Назва поля	Тип даних	Опис
id	INT [pk]	Ідентифікатор статусу
name	VARCHAR(20)	Назва статусу

Сутність chapters деталізує структуру творів, оскільки кожен твір може складатися з одного або кількох розділів. Для кожного розділу зберігається його номер, назва та шлях до відповідного файлу. Атрибути даної сутності відображено в таблиці 2.4.

Таблиця 2.4 – Атрибути сутності chapters

Назва поля	Тип даних	Опис
id	INT [pk]	Ідентифікатор категорії
name	VARCHAR(100)	Назва категорії
description	TEXT	Опис категорії

Таблиця categories, яка наведена в таблиці 2.5, містить категорії творів, які можуть виступати як жанрові класифікації (наприклад, фантастика, драма, поезія). Вона використовується для організації й фільтрації творів.

Таблиця 2.5 – Атрибути сутності categories

Назва поля	Тип даних	Опис
id	INT [pk]	Ідентифікатор категорії
name	VARCHAR(100)	Назва категорії
description	TEXT	Опис категорії

Сутність tags використовується для позначення творів ключовими словами або тематичними мітками, що допомагає краще описати зміст твору і полегшує його пошук за конкретними темами чи жанрами. Для реалізації складної структури, де один твір може мати багато тегів, а один тег може бути прив'язаний до багатьох творів, застосовується таблиця work_tags, яка забезпечує зв'язок типу «багато-до-багатьох» між сутностями творів і тегів. Такий підхід дозволяє гнучко організовувати пошук і фільтрацію контенту на платформі, роблячи процес знаходження творів за тематикою швидким і зручним. У таблиці 2.6 наведено основні атрибути сутності tags, які описують властивості кожного тегу.

Таблиця 2.6 – Атрибути сутності tags

Назва поля	Тип даних	Опис
Id	INT [pk]	Ідентифікатор
Name	VARCHAR(50)	Назва тегу

Атрибути сутності work_tags зображено в таблиці 2.7.

Таблиця 2.7 – Атрибути сутності work_tags

Назва поля	Тип даних	Опис
work_id	INT	Ідентифікатор твору
tag_id	INT	Ідентифікатор тегу

Сутність user_interactions, яка наведена в таблиці 2.8, зберігає дані про взаємодію користувача з творами: чи збережено, вподобано, переглянуто чи прочитано твір. Це дозволяє реалізувати персоналізовану стрічку або рекомендаційні системи.

Таблиця 2.8 – Атрибути сутності user_interactions

Назва поля	Тип даних	Опис
id	INT [pk]	Ідентифікатор взаємодії
work_id	INT	Ідентифікатор твору
user_id	INT	Ідентифікатор користувача
is_saved	BOOLEAN	Чи збережено
is_liked	BOOLEAN	Чи вподобано
is_viewed	BOOLEAN	Чи переглянуто
is_read	BOOLEAN	Чи прочитано

Сутність comments призначена для збереження коментарів, які залишають користувачі до окремих розділів творів на платформі. Вона дозволяє фіксувати відгуки, думки або обговорення, що допомагає створити інтерактивну спільноту навколо літературного контенту. Завдяки цій сутності користувачі можуть висловлювати свою думку щодо конкретного розділу твору.

Таблиця 2.9 – Атрибути сутності comments

Назва поля	Тип даних	Опис
id	INT [pk]	Ідентифікатор коментаря
user_id	INT	Ідентифікатор користувача
chapter_id	UUID	Ідентифікатор розділу
text	TEXT	Текст коментаря
created_at	DATETIME	Дата створення

Сутність ratings призначена для збереження оцінок, які користувачі ставлять різним творам на платформі. Вона дозволяє збирати зворотній зв'язок від аудиторії, що допомагає формувати рейтинги творів за популярністю або якістю.

Таблиця 2.10 – Атрибути сутності ratings

Назва поля	Тип даних	Опис
id	INT [pk]	Ідентифікатор оцінки
work_id	INT	Ідентифікатор твору
user_id	INT	Ідентифікатор користувача
rating	INT	Значення оцінки

В таблиці 2.11 відображена сутність subscriptions використовується для реалізації підписки на твори. Користувач може підписатися на твір і отримувати сповіщення про нові розділи.

Таблиця 2.11 – Атрибутит сутності subscriptions

Назва поля	Тип даних	Опис
id	UUID [pk]	Ідентифікатор підписки
work_id	UUID	Ідентифікатор твору
user_id	UUID	Ідентифікатор користувача
created_at	DATETIME	Дата підписки

І також сутність sessions дозволяє управляти сесіями автентифікованих користувачів. Тут зберігається інформація про активні сесії, що наведено в таблиці 2.12.

Таблиця 2.12 – Атрибути сутності sessions

Назва поля	Тип даних	Опис
id	UUID [pk]	Ідентифікатор сесії
user_id	UUID	Ідентифікатор користувача
created_at	DATETIME	Дата створення сесії
expires_at	DATETIME	Час завершення сесії
ip_address	VARCHAR(45)	IP-адреса користувача

Також важливою частиною бази даних платформи є сутність призначена для збереження скарг користувачів на конкретні коментарі. Вона допомагає модераторам швидко виявляти небажаний чи недоречний контент та оперативно реагувати на порушення. Кожен запис містить інформацію про конкретний коментар, який було оскаржено, користувача, що подав скаргу, причину звернення, а також дату й час створення звіту, що наведено в таблиці 2.13.

Таблиця 2.13 – Атрибути сутності comment_reports

Назва поля	Тип даних	Опис
id	UUID [pk]	Унікальний ідентифікатор звіту
comment_id	UUID	Ідентифікатор коментаря, на який подана скарга
user_id	UUID	Ідентифікатор користувача, який подав скаргу
reason	TEXT	Текстова причина подання скарги
created_at	TIMESTAMP	Дата і час створення звіту (за замовчуванням поточний)

Сутність ideas зберігає ідеї творів, які користувачі можуть пропонувати для подальшої творчої розробки, що представлено в таблиці 2.14. Кожна ідея пов’язана з користувачем, містить заголовок і детальний опис, дату створення, а також може бути пов’язана з конкретним твором, якщо ідея вже була реалізована у вигляді твору або пов’язана з ним.

Таблиця 2.14 – Атрибути сутності ideas

Назва поля	Тип даних	Опис
id	UUID [pk]	Унікальний ідентифікатор ідеї
user_id	UUID	Ідентифікатор користувача, що створив ідею
title	VARCHAR(100)	Заголовок ідеї
description	TEXT	Детальний опис ідеї
created_at	TIMESTAMP	Дата та час створення ідеї (за замовчуванням поточний)
linked_work_id	UUID	Ідентифікатор твору, пов’язаного з ідеєю (може бути NULL)

Для реалізації платформи було спроєктовано базу даних, яка відображає основні сутності предметної області, їх атрибути та логічні зв’язки між ними. Кожна таблиця відповідає певному об’єкту системи, наприклад, користувач, твір, розділ, коментар, оцінка тощо.

Зв’язки між таблицями реалізовано за допомогою зовнішніх ключів, що дозволяє забезпечити цілісність даних і гнучкість у побудові запитів. На основі аналізу функціональних вимог до системи була створена ER-діаграма, яка наочно демонструє взаємозв’язки між сутностями та структуру атрибутів (рисунок 2.6).

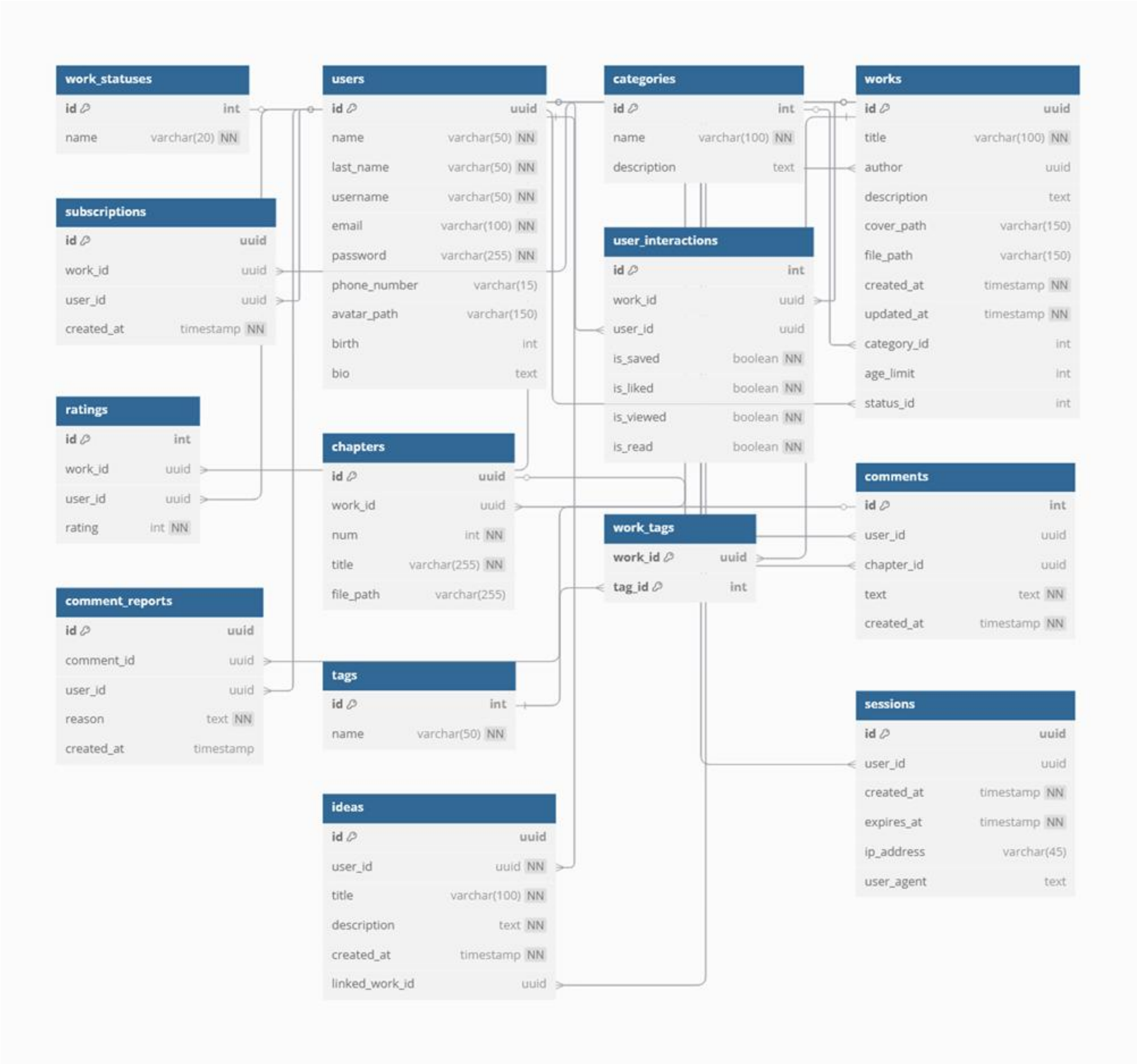

Рисунок 2.6 – ER-діаграма

В додатку А відображено таблицю всіх сутностей бази даних та їх опис. Кожна сутність відображає певну реальну або логічну одиницю, з якою взаємодіє користувач або сама система.

Здійснено проєктування бази даних, що є важливою складовою під час створення інформаційної системи. Спроєктована структура враховує всі основні аспекти функціонування платформи, зокрема зберігання інформації про користувачів, твори, розділи, оцінки, коментарі та взаємодії. Було визначено необхідні сутності, встановлено зв’язки між ними, а також розроблено ER-діаграму, що наочно відображає логічну модель бази даних.

2.3 Проєктування інтерфейсу

Інтерфейс користувача є однією з ключових складових будь-якої інформаційної системи, адже саме через нього здійснюється взаємодія між користувачем і функціоналом платформи. У цьому підрозділі буде наведено приклади макетів основних екранів.

При проєктуванні інтерфейсу було враховано потреби різних категорій користувачів: читачів, авторів і модераторів. Застосунок розробляється з урахуванням принципів кросплатформності, тому інтерфейс має бути зручним як для мобільних, так і для десктопних пристроїв.

Основний акцент зроблено на простоту навігації, доступність функцій та естетичну привабливість.

Усі екрани умовно поділяються на публічні (доступні всім користувачам) та приватні (доступні після авторизації). Передбачено також адаптивність елементів інтерфейсу залежно від типу пристрою користувача.

Для більш детального розуміння структури переходів між екранами і навігації по платформі, на рисунку 2.7 зображено схему навігації авторизованого користувача. Це дозволить чітко побачити, як користувач взаємодіє з платформою через різні сторінки та функції.

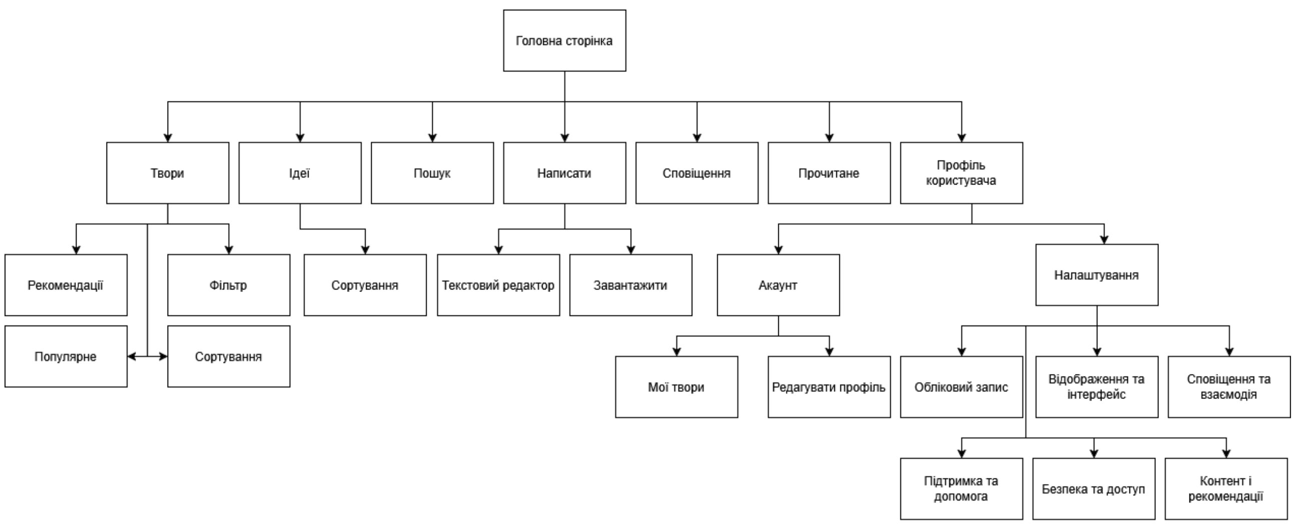

Рисунок 2.7 – Схема навігації

На головній сторінці відображаються розділи «Популярне» та «Рекомендації», що дозволяють користувачеві швидко ознайомитися з актуальними та рекомендованими творами, що відповідають його інтересам. Для зручності пошуку доступний фільтр і сортування, що дають змогу знаходити твори за різними критеріями, такими як жанр, автор, рейтинг тощо. Макет даної сторінки наведено в рисунку 2.8.

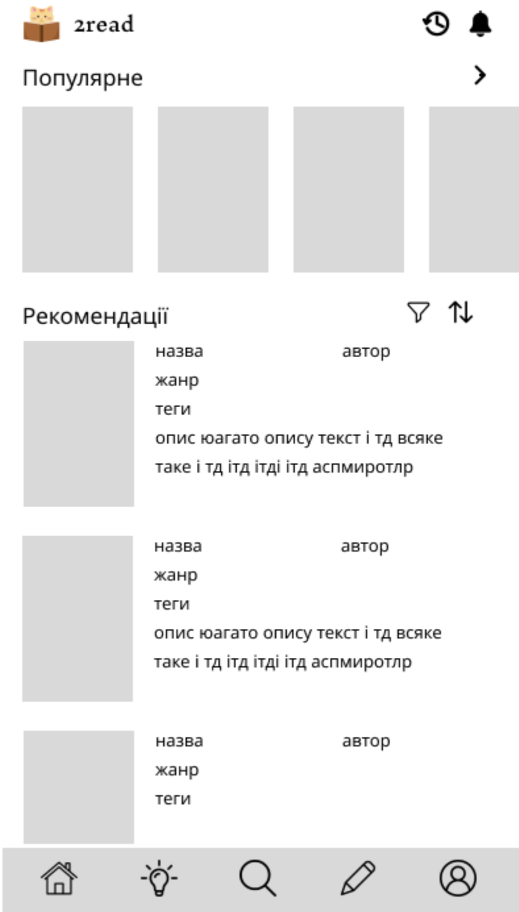

Рисунок 2.8 – Макет головної сторінки

Знизу сторінки розташована навігаційна панель, яка дозволяє з легкістю переміщатися між головною сторінкою, сторінкою ідей, пошуком, написанням власних творів та акаунтом. Ця панель завжди під рукою, щоб користувач міг швидко здійснити необхідний перехід.

Крім того, на верхній частині сторінки є блок сповіщень, де користувач може побачити нові повідомлення, такі як коментарі до його творів, нові лайки або оновлення. Також в цьому блоці знаходиться розділ з останніми прочитаними творами, що дозволяє зручно повернутися до попередньо відкритих матеріалів.

На сторінці «Ідеї» авторизований користувач може ознайомитися з наявними ідеями, які є в базі даних платформи. Ідеї представлені без обкладинок, на відміну від творів, адже в цьому немає потреби, але кожна з них має заголовок, короткий опис, а також вказано автора. Це дає можливість зрозуміти суть кожної ідеї без необхідності відкривати додаткові сторінки. Макет сторінки зображено на рисунку 2.9.

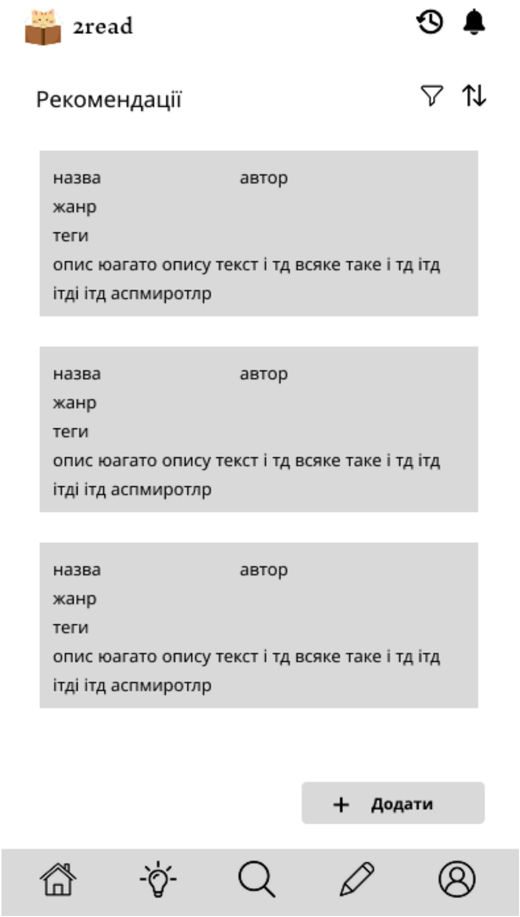

Рисунок 2.9 – Макет сторінки ідей

Сторінка «Пошук» дозволяє користувачеві ефективно знаходити твори або ідеї на платформі, що зображено на рисунку 2.10. Головним елементом цієї сторінки є поле для пошуку, в яке користувач може ввести ключові слова, такі як назва твору, автор, жанр або будь-яка інша інформація, що допоможе знайти потрібний контент.

Нижня частина сторінки містить блок із різними жанрами, що дозволяє користувачеві швидко фільтрувати результати пошуку за певними категоріями.

Така структура сторінки надає користувачеві зручний і інтуїтивно зрозумілий інтерфейс для пошуку контенту на платформі, дозволяючи йому легко знаходити цікаві твори або ідеї за допомогою простого введення запитів та використання жанрових фільтрів, що зображено на рисунку 2.10.

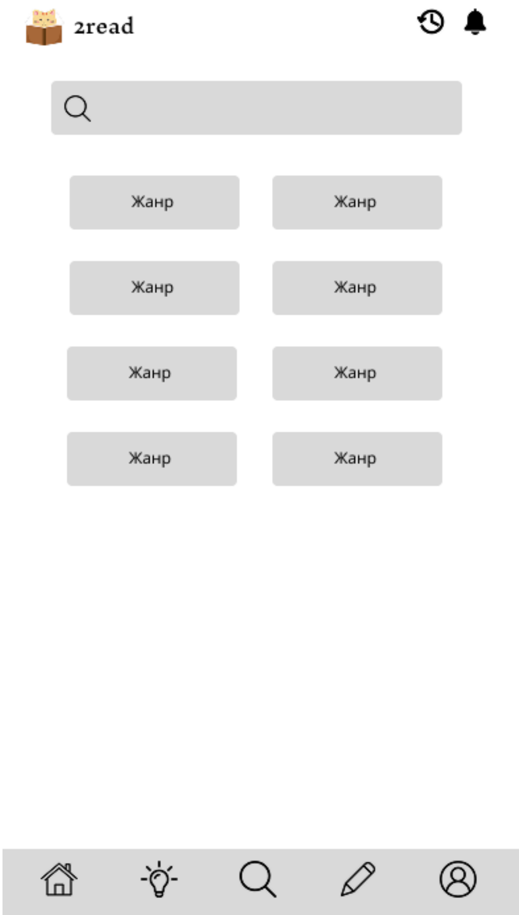

Рисунок 2.10 – Макет сторінки пошуку

Сторінка для створення нового твору з'являється, коли авторизований користувач натискає на кнопку «Написати твір» на панелі навігації, як зображено на рисунку 2.11. Відразу після цього користувач бачить форму, у якій потрібно заповнити кілька основних полів для створення нового твору.

Рисунок 2.11 – Макет сторінки створення твору

У формі є такі обов'язкові поля: назва твору, опис, жанр та тег. Користувач може вибрати один або кілька жанрів, що найкраще описують твір, а також додати відповідні теги, щоб полегшити пошук твору на платформі. Поле для вікових обмежень дає змогу вказати, які вікові категорії можуть читати твір, що допомагає зорієнтуватися, для якої аудиторії він призначений. Крім того, користувач може завантажити обкладинку для свого твору, що зробить його вигляд більш привабливим та інформативним для інших користувачів.

Загалом, проєктування системи охопило всі ключові аспекти, починаючи з розробки архітектури, яка забезпечує надійність та масштабованість, і закінчуючи детальним проектуванням бази даних для ефективного зберігання та обробки інформації. Особлива увага була приділена інтерфейсу користувача, який зроблено інтуїтивно зрозумілим та зручним для різних типів пристроїв, що сприяє комфортній взаємодії як авторів, так і читачів. Завдяки продуманій структурі та використанню сучасних технологій платформа забезпечує високу продуктивність, безпеку та можливість подальшого розвитку, що дозволить створити стабільний і гнучкий сервіс для публікації, збереження, пошуку та читання літературних творів.

2.4 Дослідження технологій та засобів реалізації застосунку

Здійснено дослідження сучасних технологій та засобів, які можуть бути використані для реалізації платформи для публікації літературної творчості. Основними критеріями вибору стали кросплатформеність, простота в розгортанні, активна підтримка спільнотою та зручність подальшого супроводу проєкту.

Для реалізації клієнтської частини розглядалися такі варіанти, як Flutter, React Native та Kotlin Multiplatform [6].

У сучасній розробці кросплатформених застосунків особливо популярним є Flutter від Google, який дозволяє створювати додатки для Android, iOS, вебу та десктопу, використовуючи єдиний код. Цей фреймворк відзначається високою швидкістю роботи, привабливою графікою та підтримкою великої спільноти, однак розмір застосунків може бути доволі великим.

Ще одним розповсюдженим підходом є React Native, що базується на JavaScript і надає розробникам широкий вибір готових рішень [7]. Водночас для досягнення повної функціональності на різних платформах іноді необхідно писати нативний код окремо для Android та iOS, а продуктивність може знижуватися у складних проєктах.

Kotlin Multiplatform – це відносно нова технологія, яка дозволяє створювати застосунки для різних пристроїв, повторно використовуючи спільну логіку. Найкраще підходить для Android, але підтримує також iOS, Windows, macOS та інші. Перевага – це можливість писати частину коду один раз. Але вона ще не така стабільна, як Flutter, і не має повної підтримки веберсії.

Для створення серверної частини розглядалися такі варіанти, як Node.js, Django, Laravel та Serverpod.

Node.js є асинхронною серверною платформою, яка базується на JavaScript. Вона широко використовується та має безліч бібліотек, однак для якісної реалізації REST API потребує ретельного проєктування та знання додаткових фреймворків, таких як Express. До того ж, JavaScript має свої недоліки, пов'язані з типізацією [8].

Django і Laravel є потужними MVC-фреймворками, які дозволяють швидко створювати серверну логіку та працювати з базами даних. Вони добре підходять для великих проєктів, однак створення окремої серверної частини на іншій мові, ніж Dart, ускладнює інтеграцію з Flutter.

Serverpod – це серверний фреймворк, створений спеціально для проєктів на Flutter. Він написаний на мові Dart і дозволяє створювати REST API, керувати базою даних, проводити валідацію та логувати події. Його перевагою є повна сумісність із Flutter, автоматична генерація клієнтського коду та зручна інтеграція з базою даних. Це дозволяє суттєво скоротити час розробки та уникнути конфліктів між клієнтом і сервером.

У питанні збереження даних порівнювались PostgreSQL та MariaDB.

PostgreSQL використовується як об'єктно-реляційна система управління базами даних, що підтримує роботу зі складними типами даних, транзакціями, а також різноманітними функціями і розширеннями [9]. Вона особливо підходить для масштабних проєктів, які вимагають складної логіки, хоча її налаштування і експлуатація можуть потребувати більше зусиль.

MariaDB є реляційною базою даних з відкритим вихідним кодом, сумісною з MySQL. Вона характеризується високою продуктивністю, дотриманням стандартів SQL і хорошою інтеграцією з інструментами адміністрування, наприклад, phpMyAdmin, що робить її зручною для розробників.

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ

3.1 Вибір та обґрунтування засобів реалізації застосунку

Розробка платформи «ToRead» здійснюється в середовищі операційної системи Arch Linux, яка забезпечує стабільну та гнучку основу для програмування. Вибір цієї системи зумовлений її високою продуктивністю, актуальністю пакетів та можливістю глибокого налаштування під потреби розробника. Arch Linux дозволяє створити максимально оптимізоване робоче середовище без зайвих компонентів, які могли б негативно впливати на швидкодію системи під час компіляції та тестування застосунку.

Для встановлення необхідних пакетів та залежностей використовується пакетний менеджер yay, який забезпечує доступ до офіційних репозиторіїв та AUR, що значно розширює можливості встановлення спеціалізованого програмного забезпечення. Використання yay замість стандартного pacman обумовлено необхідністю встановлення деяких специфічних інструментів розробки, які доступні лише через AUR репозиторій.

Реалізація клієнтської частини здійснюється за допомогою фреймворку Flutter, що дозволяє створювати кросплатформні застосунки з єдиною базою коду. Вибір Flutter зумовлений кількома ключовими факторами, серед яких найважливішими є можливість одночасної розробки для Android та iOS платформ, що значно скорочує час розробки проєкту.

Основною мовою програмування для фронтенду є Dart, яка пропонує сучасні можливості, такі як строга типізація, підтримка асинхронного програмування та ефективне керування пам’яттю. Dart забезпечує швидку компіляцію та гарячу перезагрузку, що значно прискорює цикл розробки та тестування нових функцій.

Розробка інтерфейсу та логіки користувацької взаємодії здійснюється у середовищі Visual Studio Code з відповідними розширеннями для підтримки Flutter, Dart та REST API, оскільки цей редактор забезпечує найкращу підтримку

Flutter екосистеми з можливістю налагодження, автодоповнення коду та інтеграції з емуляторами.

Серверна частина платформи розробляється з використанням фреймворку FastAPI, який забезпечує швидке створення та підтримку асинхронного виконання. Вибір FastAPI був обумовлений його виключною швидкодією серед Python веб-фреймворків, що критично важливо для обробки великої кількості запитів від користувачів та реалізації складних алгоритмів рекомендацій у реальному часі.

Автоматична генерація документації API значно спрощує процес тестування та інтеграції з клієнтською частиною, а також полегшує подальшу підтримку та розширення функціональності. Вбудована підтримка валідації даних через Pydantic моделі забезпечує надійність обробки вхідних параметрів та зменшує ймовірність помилок під час виконання. API відповідає за обробку запитів з клієнтського застосунку, взаємодію з базою даних та реалізацію логіки, зокрема системи рекомендацій. У проєкті також використовуються такі бібліотеки, як pandas для ефективної обробки структурованих даних про книги та користувачів, scikit-learn та spaCy для реалізації алгоритмів машинного навчання в системі рекомендацій [10]. Python було обрано як основну мову серверної розробки через широкий набір бібліотек для машинного навчання та обробки даних, простоту синтаксису, що пришвидшує розробку, а також завдяки високій ефективності у роботі з великими обсягами текстової інформації.

Для контролю версій та управління кодом використовується система Git з хостингом на платформі GitHub. Вибір Git був очевидним через його статус загальновизнаного стандарту в індустрії розробки програмного забезпечення, забезпечуючи розподілену архітектуру, яка дозволяє працювати над проєктом навіть без постійного підключення до мережі. Це забезпечує надійне збереження історії змін, можливість відкату до попередніх версій коду, а також створення окремих гілок для розробки різних функціональностей без впливу на основну версію проєкту. Використання системи гілок дозволяє паралельно розробляти клієнтську та серверну частини, а також експериментувати з різними алгоритмами рекомендацій без ризику порушення стабільності основної версії.

Для зберігання даних використовується система управління базами даних MariaDB, яка забезпечує надійне та ефективне збереження інформації. Даний вибір був спричинений її повною сумісністю з MySQL при кращій продуктивності та відкритістю коду, що гарантує довгострокову підтримку та відсутність ліцензійних обмежень.

MariaDB демонструє кращі показники швидкодії при роботі з великими обсягами даних, що важливо для зберігання інформації про книги, користувачів та їхні взаємодії. Вбудовані механізми індексації та оптимізації запитів забезпечують швидкий доступ до даних, необхідний для роботи системи рекомендацій у реальному часі.

Створення та адміністрування бази даних виконується через інструмент DBeaver, який забезпечує зручну візуалізацію, модифікацію таблиць і виконання SQL-запитів. DBeaver підтримує роботу з кількома підключеннями, що спрощує тестування та налагодження під час розробки, а також забезпечує можливості експорту та імпорту даних, критично важливі для створення тестових наборів даних та резервного копіювання.

3.2 Реалізація бази даних

Розробка починається зі створення базової таблиці користувачів, яка включає всю необхідну персональну інформацію, контактні дані та налаштування профілю. Після створення основної структури таблиці додаються обмеження для забезпечення унікальності критично важливих полів. Реалізацію обмеження унікальності для електронної адреси та імені користувача наведено в лістингу 3.1.

Лістинг 3.1 – Додавання обмеження унікальності для таблиці users

```
ALTER TABLE users
ADD CONSTRAINT unique_email UNIQUE (email);
ALTER TABLE users
ADD CONSTRAINT unique_username UNIQUE (username);
```

Особливу увагу приділено валідації віку користувачів при реєстрації. Для забезпечення відповідності вимогам безпеки неповнолітніх створено тригер перевірки віку, представлений у лістингу 3.2.

Лістинг 3.2 – Тригер перевірки віку користувача

```
DELIMITER $
CREATE TRIGGER check_user_age
BEFORE INSERT ON users
FOR EACH ROW
BEGIN
     IF (YEAR(CURRENT_DATE) - NEW.birth) < 13 THEN
              SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'User
must be at least 13 years old';
     END IF;
END$
DELIMITER ;
```

Паралельно проводилась розробка системи управління сеансами для забезпечення безпечної автентифікації та відстеження активних підключень.

Наступним кроком стало створення центральної таблиці творів, яка зберігає основну інформацію про публікації. Кожен твір пов'язано з автором через зовнішній ключ, що забезпечує цілісність даних. Передбачено зберігання метаданих про файли обкладинок та контенту, дати створення та останнього оновлення, а також додаткову інформацію про вікові обмеження. Для автоматичного оновлення часу модифікації записів реалізовано спеціальний тригер, наведений у лістингу 3.3.

Лістинг 3.3 – Тригер автоматичного оновлення часу модифікації

```
CREATE TRIGGER update_timestamp
BEFORE UPDATE ON works
FOR EACH ROW
SET NEW.updated_at = NOW();
```

Для забезпечення коректності даних про вікові обмеження творів додано перевірочне обмеження, представлене в лістингу 3.4.

Лістинг 3.4 – Обмеження на вікові рамки творів

```
ALTER TABLE works
ADD CONSTRAINT chk_age_limit CHECK (age_limit >= 0);
```

Для забезпечення гнучкої організації контенту розроблено подвійну систему класифікації. Таблиця категорій дозволяє організувати твори за основними жанрами, тоді як система тегів через проміжну таблицю забезпечує більш детальну та багатовимірну класифікацію. Такий підхід дозволяє користувачам легко знаходити контент за різними критеріями.

Особливу увагу приділено розробці системи відстеження взаємодій користувачів. Централізована таблиця взаємодій зберігає інформацію про всі дії користувачів: збереження в закладки, вподобання, перегляди та статус прочитання. Це рішення спрощує аналіз поведінки користувачів та формування персоналізованих рекомендацій.

Система коментарів спроєктована з прив'язкою до конкретних розділів творів, що дозволяє користувачам обговорювати окремі частини контенту. Кожен коментар містить мітку часу створення та зв'язок з автором для забезпечення модерації та відповідальності.

Рейтингова система реалізовується через окрему таблицю, що дозволяє користувачам оцінювати твори за числовою шкалою. Такий підхід забезпечує можливість формування рейтингів популярності та якості контенту. Щоб забезпечити коректність рейтингових оцінок, застосовано обмеження, показане в лістингу 3.5.

Лістинг 3.5 – Обмеження діапазону рейтингових оцінок

```
ALTER TABLE ratings
ADD CONSTRAINT rating_range CHECK (rating BETWEEN 1 AND 5);
```

Також створено SQL-запит, наведений у лістингу 3.6, який додає зовнішній ключ до таблиці ratings. Цей ключ забезпечує зв’язок між полем user_id таблиці ratings та полем id таблиці users. Таким чином гарантується цілісність даних, оскільки не може бути запису з оцінкою, який посилається на неіснуючого користувача.

Лістинг 3.6 – SQL-запит, який додає нове обмеження зовнішнього ключа

```
ALTER TABLE ratings
ADD CONSTRAINT ratings_ibfk_2
FOREIGN KEY (user_id) REFERENCES users(id)
ON DELETE CASCADE;
```

Крім того, у запиті використовується опція ON DELETE CASCADE. Вона означає, що при видаленні користувача з таблиці users автоматично буде видалено всі пов’язані з ним записи в таблиці ratings. Це дозволяє уникнути «висячих» записів у базі даних і полегшує її супровід.

Система підписок дозволяє користувачам отримувати сповіщення про оновлення улюблених творів або публікації нових робіт від конкретних авторів. Кожна підписка містить мітку часу створення для відстеження активності.

На завершальному етапі розробки було ретельно перевірено всі зв'язки між таблицями та налаштовано обмеження зовнішніх ключів для забезпечення цілісності. Проведено оптимізацію типів даних для ефективного використання дискового простору та підвищення продуктивності запитів.

Результатом розробки стала нормалізована база даних, що складається з дванадцяти взаємопов'язаних таблиць, які забезпечують повний функціонал літературної платформи (рисунок 3.1). Повний код створення бази даних наведено в додатку Б.

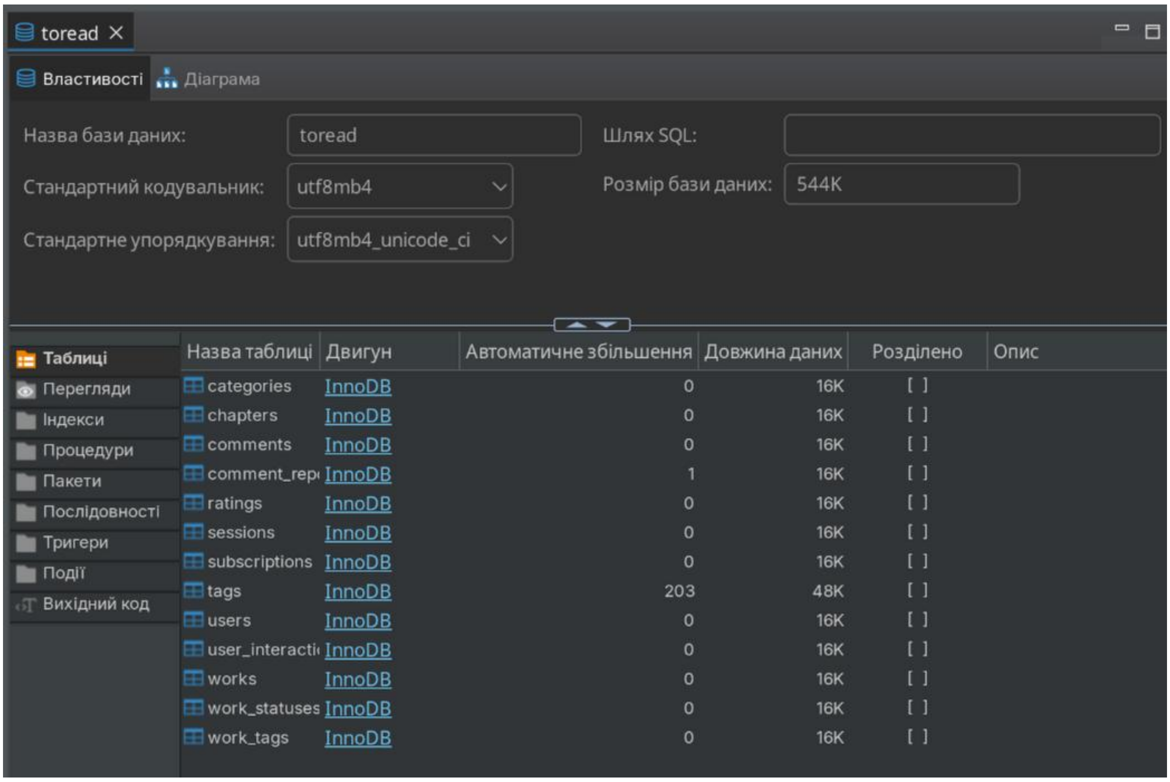

Рисунок 3.1 – Створена база даних

У результаті реалізації було створено повноцінну, нормалізовану базу даних, яка охоплює всі ключові аспекти функціонування літературної платформи. Особливу увагу приділено забезпеченню цілісності та безпеки даних, зокрема через використання зовнішніх ключів, обмежень і тригерів.

3.3 Розробка платформи

Основою серверної частини є створений екземпляр FastAPI застосунку, який слугує центральною точкою для всіх шляхів та конфігурацій.

Застосунок організовано за принципом модульності з окремими роутерами для різних функціональних блоків. Підключення до бази даних реалізовується через SQLAlchemy Engine, який забезпечує ефективне управління з'єднаннями. Використовується базові метадані для автоматичного створення необхідних таблиць в базі даних при запуску додатку. Всі таблиці бази даних описано у вигляді моделей у відповідному модулі і наведені в додатку В.

Для забезпечення взаємодії з фронтендом налаштовується CORS. Конфігурація дозволяє запити з усіх джерел, увімкнено підтримку облікових даних та дозволено використання всіх HTTP методів. Це забезпечує гнучкість у роботі з різними клієнтськими застосунками.

Модуль авторизації є ключовою частиною застосунку, що забезпечує безпечну реєстрацію нових користувачів та автентифікацію існуючих. Для реєстрації призначений метод обробляє POST запити на маршрут '/register' та приймає дані через валідовану схему RegisterRequest. Система виконує перевірку унікальності електронної пошти та імені користувача у базі даних, запобігаючи створенню дублікатів. При спробі реєстрації з існуючими даними користувач отримує відповідну помилку з HTTP статусом 400.

Особливістю реалізації є обробка дати народження, яка парситься з ISO формату та зберігається як рік народження для подальшого використання в системі рекомендацій. При створенні нового користувача генерується унікальний UUID ідентифікатор, а пароль хешується перед збереженням у базу даних.

Захист паролів користувачів реалізовується за допомогою криптографічної системи на основі бібліотеки Passlib з використанням алгоритму bcrypt. Функція hash_password перетворює звичайні паролі в безпечний хеш, який зберігається в базі даних, тоді як verify_password дозволяє перевіряти відповідність введеного пароля збереженому хешу без необхідності зберігання оригінального пароля.

Система входу реалізована через кінцевий вузол '/login', який приймає дані LoginRequest та виконує перевірку облікових даних. Процес автентифікації включає пошук користувача за електронною адресою та верифікацію пароля через bcrypt алгоритм. У разі невдалої автентифікації повертається HTTP статус 401 з відповідним повідомленням.

Повний код реалізації модуля авторизації наведено в додатку Г.

Створено функцію, яка реалізує механізм визначення популярних літературних творів на платформі на основі аналітики взаємодії користувачів. Ця функція використовує SQL-запит для агрегації статистичних даних про різні типи користувацьких дій, включаючи перегляди, вподобання, прочитання та збереження творів.

Алгоритм роботи функції базується на двоетапному підході до обробки даних. На першому етапі формується підзапит, який здійснює підрахунок кожного типу взаємодії користувачів з творами через аналіз таблиці «UserInteraction». Для кожного твору визначається кількість переглядів, вподобань, повних прочитань та збережень.

Другий етап включає основний запит, який об'єднує інформацію про твори з попередньо розрахованою статистикою взаємодій. Запит використовує операцію join для зв'язування таблиці творів з результатами підзапиту, а також застосовує оптимізацію завантаження пов'язаних даних через joinedload для авторів, категорій та тегів, що дозволяє уникнути проблеми N+1 запитів.

Сортування результатів здійснюється за кількістю переглядів у спадаючому порядку, що дозволяє отримати найбільш переглянуті твори на початку списку. Архітектура функції передбачає можливість легкої зміни критерію сортування на будь-який інший показник популярності.

Вихідні дані формуються у вигляді структурованого JSON-об'єкта, який містить всю необхідну інформацію про твір, включаючи його ідентифікатор, назву, опис, інформацію про автора, жанрову категорію, теги та статистичні показники взаємодії. Такий формат даних забезпечує зручність використання результатів на фронтенді платформи для відображення рейтингів популярності та рекомендацій користувачам.

Детальна реалізація цієї функції представлена в додатку Ґ, де наведено повний програмний код.

Створено функцію «get_chapters_for_work», яка відповідає за завантаження та повернення всіх розділів конкретного літературного твору. Як показано в лістингу 3.7, функція приймає унікальний ідентифікатор твору через параметр work_id типу UUID та сесію бази даних через dependency injection механізм FastAPI. Основною особливістю цієї функції є те, що вона автоматично сортує розділи за полем order, забезпечуючи правильну послідовність відображення розділів читачеві.

Функція використовує SQLAlchemy ORM для виконання запиту до бази даних та повертає список об'єктів типу ChapterOut, що містять всю необхідну інформацію про розділи твору.

Лістинг 3.7 – Функція отримання списку розділів твору

```
def get_chapters_for_work(work_id: UUID, db: Session =
Depends(get_db)):
     return db.query(Chapter).filter(Chapter.work_id ==
work_id).order_by(Chapter.order).all()
```

Також прописано функцію, яка призначена для отримання детальної інформації про конкретний розділ твору за його унікальним ідентифікатором. Як демонструє лістинг 3.8, функція приймає chapter_id типу UUID та повертає повний вміст розділу.

Важливою особливістю реалізації є обробка випадку, коли розділ з вказаним ідентифікатором не існує - в такому разі функція генерує HTTP exception з кодом 404 та відповідним повідомленням українською мовою. Ця функція є критично важливою для відображення контенту читачам.

Лістинг 3.8 - Функція завантаження окремого розділу

```
def get_chapter(chapter_id: UUID, db: Session =
Depends(get_db)):
     chapter = db.query(Chapter).get(chapter_id)
     if not chapter:
     raise HTTPException(status_code=404, detail="Розділ не
знайдено")
     return chapter
```

Додавання нових розділів до літературних творів відбувається з урахуванням авторизації та перевірки прав доступу. Згідно з лістингом 3.9, спочатку здійснюється перевірка наявності відповідного твору, після чого підтверджується, що поточний користувач є його автором. Лише за умови успішної верифікації створюється новий розділ, який зберігається у базі даних і повертається користувачу зі статусом HTTP 201 Created.

Лістинг 3.9 – Функція створення нового розділу

```
def create_chapter(
     chapter_data: ChapterCreate,
     db: Session = Depends(get_db),
     current_user: User = Depends(get_current_user)):
     work = db.query(Work).get(chapter_data.work_id)
     if not work or work.author_id != current_user.id:
     raise HTTPException(status_code=403, detail="Ви не маєте
прав на додавання розділу")
     new_chapter = Chapter(**chapter_data.dict())
     db.add(new_chapter)
     db.commit()
     db.refresh(new_chapter)
     return new_chapter
```

Оновлення існуючих розділів виконується з урахуванням безпеки та контролю доступу. Для цього приймаються ідентифікатор розділу, дані для оновлення, сесія бази даних і інформація про поточного користувача. Процес включає двоетапну перевірку: спочатку підтверджується наявність розділу, а потім перевіряється, чи має користувач право на редагування через авторство твору. Код реалізації наведено в лістингу 3.10.

Лістинг 3.10 - Функція редагування розділу

```
def update_chapter(
     chapter_id: UUID,
     chapter_data: ChapterUpdate,
     db: Session = Depends(get_db),
     current_user: User = Depends(get_current_user)):
     chapter = db.query(Chapter).get(chapter_id)
     if not chapter:
     raise HTTPException(status_code=404, detail="Розділ не
знайдено")
     work = db.query(Work).get(chapter.work_id)
     if work.author_id != current_user.id:
     raise HTTPException(status_code=403, detail="Ви не маєте
прав на редагування")
     for key, value in
chapter_data.dict(exclude_unset=True).items():
     setattr(chapter, key, value)
     db.commit()
     db.refresh(chapter)
     return chapter
```

Безпечне видалення розділів здійснюється з урахуванням авторизації та прав доступу. Для цього приймаються ідентифікатор розділу, сесія бази даних та інформація про поточного користувача. Спочатку перевіряється наявність розділу, після чого підтверджується право користувача на видалення через перевірку авторства твору. Після успішної верифікації розділ видаляється з бази даних, а користувачу повертається HTTP статус 204 No Content, що відповідає стандартам REST API для операцій видалення. Програмний код наведено в лістингу 3.11.

Лістинг 3.11 - Функція видалення розділу

```
def delete_chapter(
     chapter_id: UUID,
     db: Session = Depends(get_db),
     current_user: User = Depends(get_current_user)
):
     chapter = db.query(Chapter).get(chapter_id)
     if not chapter:
     raise HTTPException(status_code=404, detail="Розділ не
знайдено")

     work = db.query(Work).get(chapter.work_id)
     if work.author_id != current_user.id:
     raise HTTPException(status_code=403, detail="Ви не маєте
прав на видалення")
```

```
        db.delete(chapter)
        db.commit()
        return
```

Центральною функцією системи коментування є add_comment, яка забезпечує універсальний механізм додавання коментарів як до цілих творів, так і до окремих розділів. Особливістю архітектури цієї функції є гнучка валідація вхідних даних – система перевіряє, що користувач обов'язково вказав принаймні один з двох можливих ідентифікаторів: або work_id для коментування всього твору, або chapter_id для коментування конкретного розділу. Після первинної валідації функція виконує перевірку існування відповідного об'єкта в базі даних, щоб уникнути створення «висячих» коментарів. Процес створення коментаря завершується автоматичним прив'язуванням до поточного авторизованого користувача та поверненням створеного об'єкта з HTTP статусом 201 Created. Програмний код даної функції наведено в додатку Д.

Створено функцію для отримання коментарів з бази даних, яка наведена в лістингу 3.12.

Лістинг 3.12 - Функція отримання коментарів з фільтрацією

```
def get_comments(
        work_id: Optional[UUID] = Query(default=None),
        chapter_id: Optional[UUID] = Query(default=None),
        db: Session = Depends(get_db)
):
        query = db.query(Comment)
        if work_id:
        query = query.filter(Comment.work_id == work_id)
        if chapter_id:
        query = query.filter(Comment.chapter_id == chapter_id)

        return query.order_by(Comment.created_at).all()
```

Функція get_comments реалізує підхід до побудови динамічних запитів з опціональною фільтрацією. На відміну від статичних кінцевих вузлів, ця функція приймає опціональні параметри work_id та chapter_id, що дозволяє клієнтській частині гнучко налаштовувати вибірку коментарів. Спочатку створюється базовий запит до таблиці коментарів, а потім поступово додаються фільтри залежно від наданих параметрів. Результати автоматично сортуються за часом створення, забезпечуючи хронологічний порядок відображення коментарів.

Операція видалення коментарів реалізована через функцію, яка впроваджує багаторівневу систему авторизації. Унікальною особливістю цієї функції є підтримка двох типів користувачів, які мають право видаляти коментарі: безпосередньо автор коментаря та користувачі з модераторськими правами. Така архітектура дозволяє забезпечити баланс між правами користувачів на контроль свого контенту та необхідністю модерації спільноти. Логіка перевірки прав, реалізована у функції, поданій у лістингу 3.13, побудована на комбінації умов – вона спершу порівнює ідентифікатор автора коментаря з поточним користувачем, а у випадку невідповідності перевіряє наявність модераторських прав.

Лістинг 3.13 – Функція видалення

```
def delete_comment(
     comment_id: UUID,
     db: Session = Depends(get_db),
     current_user: User = Depends(get_current_user)
):
     comment = db.query(Comment).get(comment_id)
     if not comment:
     raise HTTPException(status_code=404, detail="Коментар не
знайдено")
     if comment.user_id != current_user.id and not
current_user.is_moderator:
     raise HTTPException(status_code=403, detail="Немає прав
для видалення")
     db.delete(comment)
     db.commit()
     return
```

Завершує функціональність модуля система скарг, реалізована через функцію для оброблення повідомлень про порушення. Її код наведено в лістингу 3.14. Ця функція демонструє підхід до обробки користувацьких скарг з використанням прямих SQL-запитів для максимальної продуктивності. Архітектурне рішення полягає у створенні окремої таблиці звітів, що дозволяє зберігати історію всіх скарг та їх подальшу обробку модераторами. Функція повертає HTTP статус 202 Accepted, що сигналізує про прийняття запиту до обробки, але не гарантує миттєвого результату – це відповідає асинхронній природі процесу модерації контенту в реальних системах.

Лістинг 3.14 – Функція подачі скарг з асинхронною обробкою

```
def report_comment(
     comment_id: UUID,
     report: ReportCommentRequest,
     db: Session = Depends(get_db),
     current_user: User = Depends(get_current_user)):
     comment = db.query(Comment).get(comment_id)
     if not comment:
     raise HTTPException(status_code=404, detail="Коментар не
знайдено")
     db.execute(
     """
     INSERT INTO comment_reports (comment_id, user_id, reason)
     VALUES (:comment_id, :user_id, :reason)
     """,
     {"comment_id": str(comment_id), "user_id":
str(current_user.id), "reason": report.reason})
     db.commit()
     return {"message": "Скарга прийнята"}
```

Система оцінювання творів працює через функцію встановлення рейтингу, код якої представлено в лістингу 3.15. Дана функція забезпечує універсальний механізм додавання нових оцінок та оновлення існуючих в рамках однієї операції.

Архітектурною особливістю реалізації є валідація діапазону оцінок – система автоматично перевіряє, що надана оцінка знаходиться в межах від 1 до 5 балів. Логіка роботи побудована на принципі «upsert» операції: функція спочатку намагається знайти існуючу оцінку від конкретного користувача для вказаного твору, і якщо така знаходиться, то оновлює її значення, в іншому випадку створює новий запис в базі даних.

Лістинг 3.15 – Функція встановлення та оновлення оцінок

```
def set_rating(db: Session, work_id: uuid.UUID, user_id:
uuid.UUID, rating: int):
     if not (1 <= rating <= 5):
     raise ValueError("Rating must be between 1 and 5")
     existing_rating = (
     db.query(Rating)
     .filter(Rating.work_id == work_id, Rating.user_id ==
user_id)
```

```
        .first()
        )
        if existing_rating:
        existing_rating.rating = rating
        else:
        new_rating = Rating(work_id=work_id, user_id=user_id,

rating=rating)
        db.add(new_rating)
        db.commit()
```

Для відображення загальної популярності творів використовується функція обчислення середнього рейтингу, представлена в лістингу 3.16. Ця функція демонструє ефективне використання агрегатних функцій SQLAlchemy для виконання математичних обчислень безпосередньо на рівні бази даних. Особливістю реалізації є обробка граничного випадку, коли твір ще не має жодних оцінок – в такому разі функція повертає 0.0 замість None, що спрощує подальшу обробку результатів в користувацькому інтерфейсі.

Лістинг 3.16 – Функція обчислення середнього рейтингу

```
def get_average_rating(db: Session, work_id: uuid.UUID) ->
float:
     avg =
db.query(func.avg(Rating.rating)).filter(Rating.work_id ==
work_id).scalar()
     return float(avg) if avg is not None else 0.0
```

Завершує функціональність рейтингової системи функція отримання персональної оцінки користувача, код якої показано в лістингу 3.17. Дана функція критично важлива для користувацького досвіду, оскільки дозволяє відображати поточну оцінку користувача в інтерфейсі та запобігати повторному оцінюванню того самого твору.

Мінімалістичний підхід до реалізації забезпечує високу швидкість виконання завдяки використанню методу, який повертає тільки значення оцінки без створення повноцінного об'єкта моделі. Повернення None у випадку відсутності оцінки дозволяє клієнтському коду легко розрізняти між відсутністю оцінки та оцінкою зі значенням 0.

Лістинг 3.17 – Функція отримання індивідуальної оцінки користувача

```
def get_user_rating(db: Session, work_id: uuid.UUID, user_id: uuid.UUID) -> int | None:
     rating = (
     db.query(Rating.rating)
     .filter(Rating.work_id == work_id, Rating.user_id == user_id)
     .scalar())
     return rating
```

Базовим елементом системи профілів є функція отримання даних користувача, реалізація якої продемонстрована в лістингу 3.18. Дана функція забезпечує простий та ефективний доступ до повної інформації про користувача за його унікальним ідентифікатором. Мінімалістичний підхід до реалізації гарантує швидке виконання запиту та дозволяє отримати всі поля користувача одним зверненням до бази даних. Важливою особливістю є обробка ситуації відсутності користувача – функція коректно повертає None замість генерування винятку, що спрощує логіку обробки помилок в вищих рівнях додатку.

Лістинг 3.18 – Функція отримання інформації про користувача

```
def get_user_profile(db: Session, user_id: uuid.UUID) -> Optional[User]:
     return db.query(User).filter(User.id == user_id).first()
```

Забезпечується механізм обліку взаємодій користувача з творами, що включає фіксацію переглядів. Для цього використовується таблиця user_interactions, яка містить відповідні позначки для кожного користувача щодо конкретного твору. Щоб отримати список творів, які були переглянуті певним користувачем, створено функцію get_viewed_works, код якої наведено в лістингу 3.19.

Лістинг 3.19 – Функція отримання переглянутих творів користувача

```
def get_viewed_works(db: Session, user_id: uuid.UUID) -> List[Work]:
     return (
     db.query(Work)
     .join(UserInteraction, UserInteraction.work_id == Work.id)
     .filter(UserInteraction.user_id == user_id, UserInteraction.is_viewed == True)      .all()    )
```

Функція приймає ідентифікатор користувача та об’єкт сесії бази даних SQLAlchemy. Вона здійснює запит до таблиці творів, приєднуючи до неї таблицю «UserInteraction» через поле з id роботи, і відбирає лише ті записи, де id збігається з переданим параметром, а поле, яке відповідає за перегляд роботи, має значення True. Таким чином, формується повний список творів, які були переглянуті зазначеним користувачем.

Система редагування профілю організовується через універсальну функцію оновлення, код якої представлено в додатку Е. Архітектурною перевагою цієї функції є підтримка часткового оновлення профілю – користувач може змінити будь-яку комбінацію полів без необхідності передавати всі дані. Реалізація використовує паттерн явної перевірки кожного параметру на None, що забезпечує точний контроль над тим, які поля будуть оновлені. Така логіка особливо важлива для збереження існуючих даних користувача, які не потребують змін, та дозволяє клієнтським додаткам реалізовувати як повне, так і часткове редагування профілю.

Для відображення творчого доробку користувача використовується спеціалізована функція, яка показана в лістингу 3.20. Ця функція демонструє прямолінійний підхід до вибірки творів за авторством, використовуючи просту фільтрацію по полю автора. Простота реалізації забезпечує високу швидкість виконання та легкість розуміння логіки, що критично важливо для часто використовуваних функцій профілю. Результат функції може бути легко інтегрований з системами пагінації та сортування для оптимального відображення великих колекцій творів.

Лістинг 3.20 – Функція отримання власних творів автора

```
def get_own_works(db: Session, user_id: uuid.UUID) ->
List[Work]:
     return db.query(Work).filter(Work.author ==
user_id).all()
```

Система закладок впроваджується через функцію отримання збережених творів, код якої представлено в лістингу 3.21. Дана функція демонструє використання JOIN операцій для зв'язування таблиць творів та користувацьких взаємодій. Фільтрація здійснюється за допомогою комбінованих умов – одночасно перевіряється ідентифікатор користувача та статус збереження твору. Такий підхід забезпечує точну вибірку тільки тих творів, які користувач явно позначив для збереження, виключаючи випадкові або застарілі записи взаємодій.

Лістинг 3.21 – Функція отримання збережених творів

```
def get_saved_works(db: Session, user_id: uuid.UUID) ->
List[Work]:
     return (
     db.query(Work)
     .join(UserInteraction, UserInteraction.work_id ==
Work.id)
     .filter(UserInteraction.user_id == user_id,
UserInteraction.is_saved == True)
     .all()
     )
     .all())
```

Аналогічно до системи закладок працює функція отримання вподобаних творів, реалізація якої показана в лістингу 3.22. Структурно ця функція повторює логіку збережених творів, але фільтрує записи за полем лайків замість збережень. Використання однакового патерну для різних типів користувацьких взаємодій забезпечує узгодженість коду та спрощує підтримку системи. Функція ефективно обробляє ситуації, коли користувач має велику кількість вподобань, завдяки оптимізованим запитам з JOIN операціями.

Лістинг 3.22 – Функція отримання вподобань користувача

```
def get_liked_works(db: Session, user_id: uuid.UUID) ->
List[Work]:
     return (
     db.query(Work)
     .join(UserInteraction, UserInteraction.work_id ==
Work.id)
     .filter(UserInteraction.user_id == user_id,
UserInteraction.is_liked == True)
     .all()
     )
```

Система відстеження коментарів користувача, код якої представлено в лістингу 3.23, має найбільш складну структуру запиту, оскільки потребує зв'язування трьох таблиць: творів, коментарів та розділів. Хоча в коментарі до коду зазначено про потенційну проблему з логікою JOIN операцій, базова ідея функції полягає у відстеженні активності користувача через його коментарі.

Реалізація дозволяє користувачам легко знаходити твори, в обговоренні яких вони брали участь, що покращує навігацію по платформі та заохочує до повторних відвідувань.

Лістинг 3.23 – Функція отримання прокоментованих творів

```
def get_commented_works(db: Session, user_id: uuid.UUID) ->
List[Work]:
    return (
         db.query(Work)
         .join(Comment, Comment.chapter_id == Work.id)
         .join(Chapter, Chapter.id == Comment.chapter_id)
         .filter(Comment.user_id == user_id)
         .all())
```

Важливою частиною системи пошуку платформи літературної творчості є комплексна функція з багатьма критеріями пошуку, повна реалізація якої представлена в додатку Є. Ця функція демонструє прогресивний підхід до побудови складних запитів, коли базовий запит поступово доповнюється фільтрами залежно від переданих параметрів. Рішення полягає в тому, що кожен фільтр є опціональним і не впливає на роботу інших, дозволяючи користувачам комбінувати будь-які критерії пошуку. Особливо важливою є реалізація пошуку за автором, яка одночасно перевіряє як повне ім'я, так і псевдонім письменника, використовуючи логічний оператор OR для максимального охоплення результатів.

Фільтрація за тегами демонструє складніший підхід із використанням вкладених JOIN операцій та ітеративного додавання умов для кожного тега окремо. Такий метод забезпечує пошук творів, які містять всі вказані теги одночасно, а не будь-який з них. Сортування за популярністю реалізовано через обчислення середнього рейтингу з використанням агрегатної функції та LEFT JOIN для включення творів без оцінок. Використання GROUP BY у поєднанні з ORDER BY дозволяє ефективно сортувати результати за рейтингом від найвищого до найнижчого.

Фундаментом системи автентифікації є функція створення сесій, детальна реалізація якої показана в лістингу 3.24. Ця функція відповідає за генерацію унікальних сесійних токенів та їх збереження в базі даних з відповідними метаданими. Архітектурною особливістю є автоматичне обчислення часу закінчення сесії на основі поточного моменту та заздалегідь визначеної константи тривалості. Система також збирає додаткову інформацію про контекст входу користувача, включаючи IP-адресу, що може бути корисним для аналізу безпеки та відстеження підозрілої активності.

Лістинг 3.24 – Функція створення нових користувацьких сесій

```
def create_session(db: Session, user_id: uuid.UUID,
ip_address: str = None, user_agent: str = None) ->
SessionModel:
    now = datetime.utcnow()
    session = SessionModel(
         id=uuid.uuid4(),
         user_id=user_id,
         created_at=now,
         expires_at=now +
timedelta(hours=SESSION_DURATION_HOURS),
         ip_address=ip_address,
         user_agent=user_agent
    )
    db.add(session)
    db.commit()
    db.refresh(session)
    return session
```

Система підтримки тривалих сесій реалізована через механізм оновлення термінів дії, код якого представлено в лістингу 3.25. Дана функція забезпечує безперервний користувацький досвід, автоматично продовжуючи сесії активних користувачів без необхідності повторної автентифікації. Логіка роботи побудована на принципі «sliding expiration» – кожне звернення до системи скидає лічильник часу до початкового значення. Повернення булевого результату дозволяє легко визначити успішність операції та відповідно відреагувати на ситуації з неіснуючими сесіями.

Лістинг 3.25 – Функція продовження активних сесій

```
def refresh_session(db: Session, session_id: uuid.UUID) ->
bool:
    session = db.query(SessionModel).filter(SessionModel.id
== session_id).first()
    if not session:
         return False
```

```
    now = datetime.utcnow()
    session.expires_at = now + 
timedelta(hours=SESSION_DURATION_HOURS)
    db.commit()
    return True
```

Критично важливою для безпеки системи є функція перевірки дійсності сесій, реалізація якої показана в лістингу 3.26. Ця функція виконує комплексну валідацію, одночасно перевіряючи як існування сесії в базі даних, так і актуальність її терміну дії. Використання комбінованого запиту з двома умовами фільтрації забезпечує високу продуктивність, оскільки база даних може використовувати індекси для швидкого знаходження відповідних записів.

Лістинг 3.26 – Функція валідації існуючих сесій

```
def validate_session(db: Session, session_id: uuid.UUID) -> 
bool:
    now = datetime.utcnow()
    session = db.query(SessionModel).filter(
         SessionModel.id == session_id,
         SessionModel.expires_at > now
    ).first()
    return session is not None
```

Для управління сесіями функціонує механізм логауту, який є важливою частиною системи та код якого представлено в лістингу 3.27. Ця функція забезпечує безпечне завершення користувацьких сесій шляхом їх повного видалення з бази даних. На відміну від пасивного закінчення терміну дії, активне видалення сесії гарантує, що токен не може бути використаний повторно навіть у випадку його компрометації. Функція також включає перевірку існування сесії перед спробою видалення, що запобігає виникненню помилок та дозволяє коректно обробляти ситуації з вже неіснуючими або застарілими токенами.

Лістинг 3.27 – Функція завершення користувацьких сесій

```
def logout_session(db: Session, session_id: uuid.UUID) -> 
bool:
    session = db.query(SessionModel).filter(SessionModel.id 
== session_id).first()
    if not session:
         return False
    db.delete(session)
    db.commit()
    return True
```

На платформі застосовується гібридна система рекомендацій, що поєднує методи контентної та колаборативної фільтрації. Її основна мета – надати користувачеві добірку актуальних творів, які можуть його зацікавити на основі історії оцінювання, схожості описів та активності інших користувачів.

Для побудови рекомендаційної системи спочатку створюється матриця «користувач – твір», де фіксуються оцінки, які кожен користувач поставив окремим творам. Ці дані використовуються для аналізу уподобань і визначення схожості між користувачами шляхом обчислення косинусної подібності. Це дозволяє виявити групи користувачів з подібними смаками, що є основою колаборативного підходу.

Паралельно застосовується контентний підхід, який ґрунтується на описах творів. Вони обробляються із застосуванням TF-IDF векторизації, після чого також обчислюється косинусна подібність між векторами описів. Таким чином, для кожного користувача можна знайти твори, схожі на ті, які він вже читав або оцінював.

Для досягнення найвищої релевантності результатів обидва підходи поєднуються. Якщо ж користувач новий або не має достатньо активності, система переходить до рекомендацій за популярністю. У цьому випадку аналізується кількість оцінок і середній рейтинг творів, що дозволяє рекомендувати найпопулярніший і найкраще оцінений контент на платформі.

Результат формується у вигляді добірки творів із включенням основної інформації: назви, опису, автора, жанрів і тегів, щоб користувач міг одразу зорієнтуватися в запропонованому контенті.

Вся логіка побудови персоналізованих рекомендацій доступна в додатку Ж.

Клієнтська частина платформи для літературної творчості реалізовується за допомогою фреймворку Flutter, який дає змогу створювати кросплатформні застосунки з єдиною кодовою базою. Проєкт має типову структуру для Flutter-застосунків, яка зберігається в окремій папці, що містить підкаталоги та файли для керування залежностями, логікою інтерфейсу, стилями та ресурсами.

У процесі розробки клієнтської частини платформи для літературної творчості застосовується модульний підхід, відповідно до якого кожен функціональний елемент застосунку реалізується в окремому файлі. Така структура підвищує зручність у розробці, тестуванні та підтримці коду. Всі екрани, сервіси, допоміжні компоненти та бізнес-логіка розміщуються у відповідних директоріях, що забезпечує логічне розділення відповідальностей у програмі.

У клієнтській частині застосунку здійснюється розробка кількох ключових Dart-файлів, які забезпечують запуск застосунку, взаємодію з користувачем та підключення до серверної частини через HTTP-запити.

Файл main.dart містить точку входу в застосунок. Саме звідси розпочинається виконання коду після запуску програми. У ньому створено клас MyApp, який є спадкоємцем StatelessWidget і відповідає за базове налаштування інтерфейсу. Вказується назва застосунку, застосовується тема з використанням шрифту, що підтримує кирилицю, та задається головний екран – WelcomeScreen, який завантажується першим. Файл організовано таким чином, щоб забезпечити чітку структуру та зручність розширення в майбутньому. Програмний код наведено в лістингу 3.28.

Лістинг 3.28 – Точка входу в проєкт

```
import 'package:flutter/material.dart';
import 'welcome_screen.dart';
void main() {
 runApp(MyApp());
}
class MyApp extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
   return MaterialApp(
     title: 'Перевірка БД',
     theme: ThemeData(
       fontFamily: 'Roboto',
     ),
     home: WelcomeScreen(),
   );
 }
}
```

Файл auth_service.dart реалізує окремий модуль для обробки автентифікації користувачів. Клас AuthService містить метод login, який відповідає за відправлення POST-запиту на сервер із введеними користувачем електронною поштою та паролем. У разі успішної автентифікації сервер повертає userId, який використовується для подальшої взаємодії з системою. Також передбачена обробка типових помилок, таких як неправильні облікові дані або відсутність мережевого з'єднання, що підвищує надійність застосунку. Програмний код наведено в додатку І.

Для збереження стилістичної єдності в інтерфейсі користувача створюється окремий файл app_styles.dart, у якому зосереджено всі візуальні налаштування – від стилів тексту до оформлення полів введення та кнопок. Винесення стилів у самостійний модуль дозволяє зручно централізовано змінювати зовнішній вигляд застосунку та дотримуватись принципів повторного використання коду. Таким чином, усі візуальні елементи застосунку виглядають узгоджено, незалежно від того, на якому екрані вони використовуються. В додатку К наведено програмний код даного файлу.

В окремому файлі, який знаходиться в додатку Л, реалізовано функціонал введення або імпортування першого розділу твору. У цьому компоненті користувач може обрати файл зі свого пристрою – наприклад, у форматі .txt, .doc, .docx чи .rtf, після чого його вміст автоматично завантажується у текстове поле. Якщо користувач не хоче працювати з файлами, він може ввести текст вручну. Після цього є можливість як зберегти чернетку, так і одразу опублікувати розділ. Усі дії супроводжуються короткими сповіщеннями, які інформують користувача про успішність або помилки під час виконання. Ретельна обробка помилок – зокрема, у випадку скасування вибору файлу або помилки при зчитуванні – покращує загальний досвід користувача.

У межах розробки клієнтської частини мобільного застосунку функціонує сторінка створення твору. Її функціонал представлений у вигляді окремого стану керованого віджета, який забезпечує зручний інтерфейс для введення основної інформації про літературний твір. Користувач може ввести назву, опис, обрати жанр із динамічного списку категорій, вказати статус твору, а також додати теги для кращої класифікації твору. Дані для списків жанрів, тегів та статусів динамічно завантажуються з серверної частини застосунку за допомогою HTTP-запитів. При завантаженні даних передбачена обробка можливих помилок із виведенням сповіщення про збій. Для тегів реалізовано окремий підкомпонент з підтримкою пошуку за назвою, вибору й зняття вибраного з використанням FilterChip, а також інтегрованими підказками через Tooltip з описами тегів. Після заповнення всіх необхідних полів та валідації форми користувач може перейти до наступного етапу – створення розділів твору. Код, що реалізує вищезазначений інтерфейс, наведено в додатку М.

Головний екран застосунку реалізовується як StatefulWidget – клас HomeScreen з внутрішнім станом _HomeScreenState. У стані зберігаються два списки рекомендацій: персональні та популярні твори, які завантажуються асинхронно з сервера за допомогою HTTP GET-запитів. Дані отримуються у форматі JSON, парсяться у моделі Recommendation, що зображено в додатку Н.

Для завантаження та відображення даних використовується FutureBuilder, який показує індикатор завантаження, обробляє помилки, а також коректно відображає списки у вигляді карток із деталями творів. У верхній панелі є кнопки переходу до екранів сповіщень та історії. Механізм авторизації реалізовується у вигляді окремого екрана, що відповідає за обробку введення облікових даних. Конструкція працює як динамічний віджет, що дозволяє обробляти введення електронної пошти та пароля через форму (рисунок 3.2).

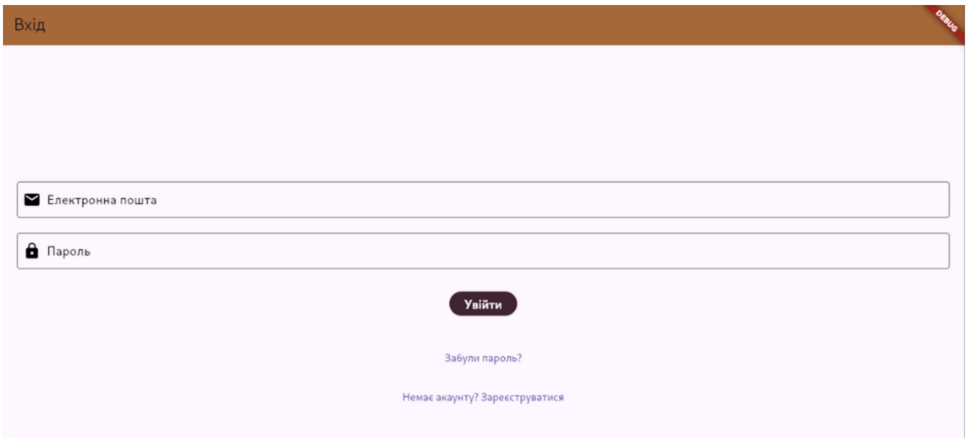

Рисунок 3.2 – Сторінка входу

Валідація форми забезпечує перевірку на наявність даних у відповідних полях. У разі успішного введення інформації здійснюється виклик методу login із класу AuthService, після чого відбувається перенаправлення на головний екран, якщо авторизація пройшла успішно. У випадку помилки користувачу виводиться відповідне повідомлення. Додатково реалізовано перехід на екран реєстрації, а також механізм відновлення пароля за допомогою модального вікна. Візуальні елементи інтерфейсу стилізовано за допомогою зовнішніх стилів, раніше визначених у окремому файлі.

Код реалізації цього екрану наведено в Додатку П.

Реєстрація користувача на платформі функціонує через окремий інтерфейс, який розташований у класі RegistrationScreen. Він відображає форму, де користувач вводить основні дані: ім’я користувача, ім’я, прізвище, електронну пошту, пароль та дату народження (рисунок 3.3). Для цього використовується стандартна форма Flutter з валідацією, що дає змогу перевірити коректність введених даних ще до надсилання на сервер.

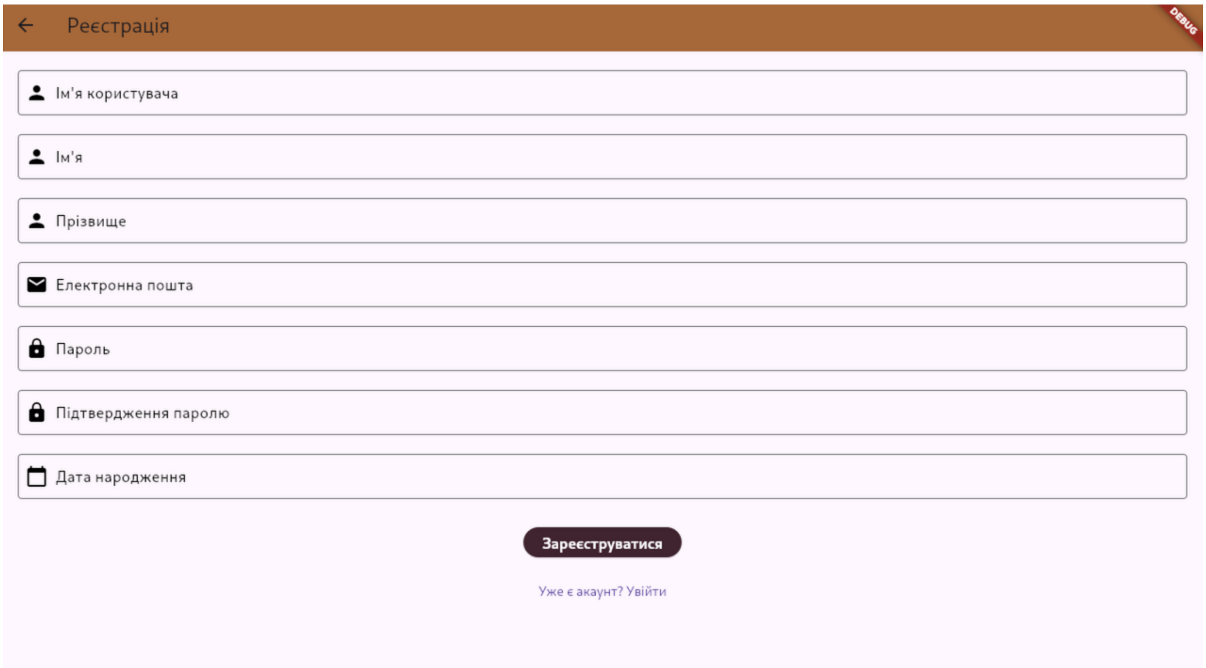

Рисунок 3.3 – Форма реєстрації

Форма включає окремі поля з відповідними іконками та підказками. Для вибору дати народження реалізоване вікно вибору дати, яке відкривається при натисканні на відповідне поле. Перевірка віку також реалізується через функцію валідації, що допомагає відсіяти користувачів, яким ще не виповнилося 13 років. Крім того, під час введення пароля використовується окрема перевірка складності пароля, а поле підтвердження перевіряє, чи обидва значення співпадають.

Після заповнення форми та натискання кнопки «Зареєструватися» відбувається надсилання POST-запиту на відповідну адресу серверу. У випадку успішної відповіді користувач отримує повідомлення про успішну реєстрацію й автоматично повертається до попереднього екрана. Якщо ж сталася помилка на стороні сервера або виникли проблеми з мережею, ці випадки обробляються й користувачеві показується відповідне повідомлення.

Усю логіка, пов’язана із надсиланням запиту та обробкою відповіді, забезпечується в методі _register. Для HTTP-запитів використовується бібліотека http, а дані передаються у форматі JSON. Крім того, при натисканні кнопки повернення на попередній екран передбачена можливість швидкого повернення для тих, хто вже має акаунт. Описаний інтерфейс реєстрації користувача розміщено в окремому файлі, а відповідний код представлено в додатку Р.

Також створюється вітальний екран, що відкривається під час першого запуску застосунку. Він реалізує карусель із декількох слайдів, кожен із яких містить ілюстрацію, заголовок і короткий опис функціоналу (рисунок 3.4).

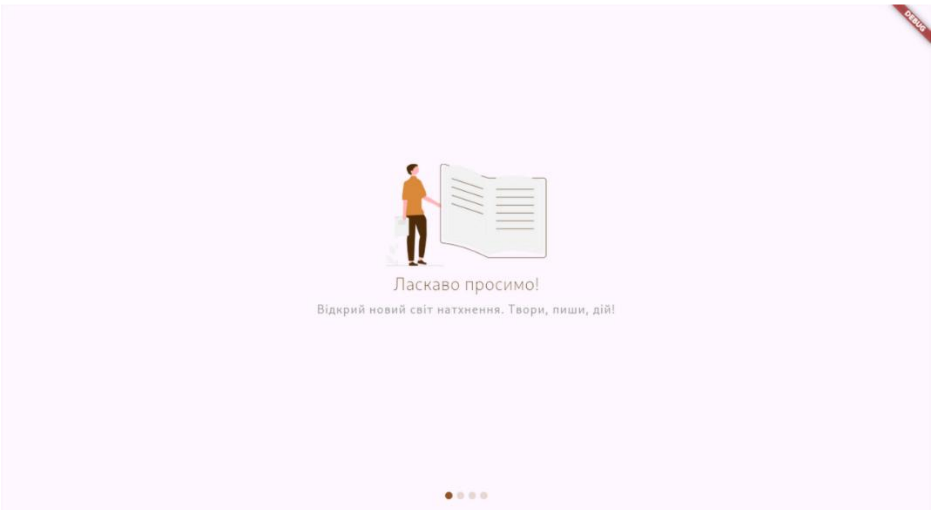

Рисунок 3.4 – Екран привітання

На останньому слайді з’являється кнопка «Увійти», яка переводить користувача на сторінку авторизації. Такий екран знайомить нових користувачів із можливостями додатку та створює позитивне перше враження. Програмний код екрану привітання наведено в додатку С.

3.3 Тестування програмного засобу

Функціональне тестування проведено з метою перевірки відповідності реалізованого програмного забезпечення встановленим функціональним вимогам.

Проведене тестування підтверджує, що компонент пошуку працює як належить. Після введення назви, імені автора або певного тегу, система коректно відображала релевантні результати. У разі відсутності збігів виводилося повідомлення про те, що нічого не знайдено (рисунок 3.5).

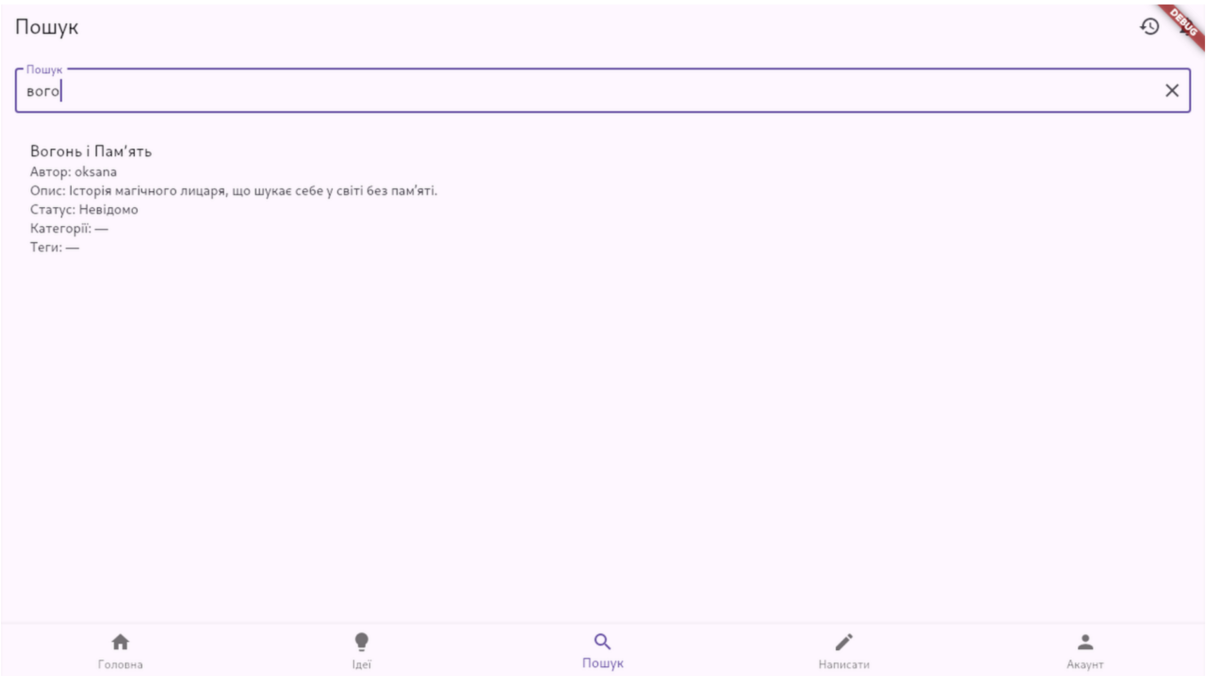

Рисунок 3.5 – Сторінка пошуку

Фільтрація творів за жанрами, віковими рейтингами, популярністю та новизною працює стабільно та ефективно , забезпечуючи користувачеві зручний інструмент для пошуку контенту. Кожен фільтр реалізовано таким чином, щоб забезпечити гнучкість у виборі параметрів. Користувач має змогу обирати одну або кілька категорій одночасно, що дозволяє точно підбирати твори відповідно до власних уподобань, настрій або цільової аудиторії.

Вибір будь-якого параметра миттєво оновлює список результатів без потреби додаткового підтвердження чи перезавантаження сторінки, що значно покращує користувацький досвід (рисунок 3.6). Такий підхід сприяє швидкому знаходженню релевантного контенту та мотивує користувачів до подальшого ознайомлення з новими творами на платформі.

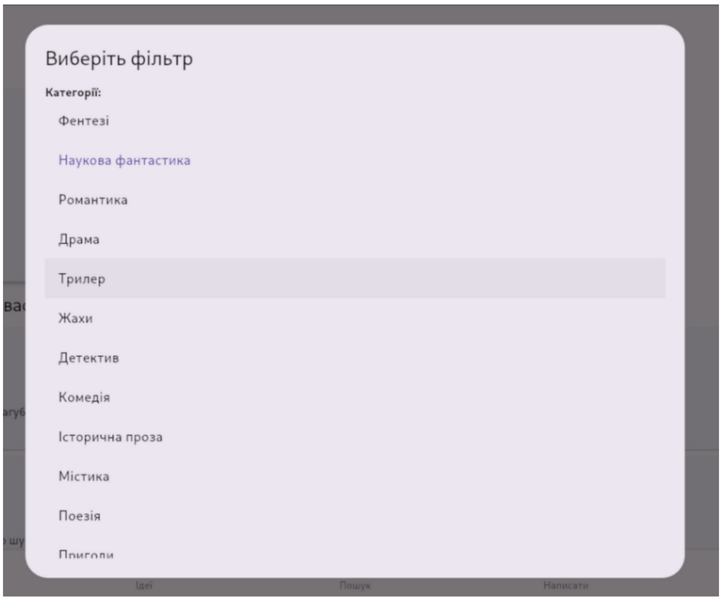

Рисунок 3.6 – Тестування фільтрації

Механізм збереження чернеток реалізовано коректно та надійно, що забезпечує зручність і гнучкість для користувачів під час роботи над власними творами. Передбачена можливість зберігати незавершені тексти у вигляді чернеток дозволяє користувачам працювати над вмістом поступово, без необхідності негайної публікації. У будь-який момент можна зупинити написання, зберегти проміжний результат і повернутися до нього пізніше, що особливо корисно в умовах обмеженого часу чи натхнення. Такий підхід значно знижує ризик втрати даних і сприяє більш комфортному творчому процесу.

Функція персоналізованих рекомендацій забезпечує формування добірки творів, які потенційно зацікавлять конкретного користувача. Система аналізує поведінку на платформі – зокрема прочитані твори, їх жанри та тематики – й на основі цих даних пропонує нові тексти, що можуть відповідати вподобанням користувача. Після ознайомлення з кількома творами головна сторінка оновлюється, демонструючи персоналізовані рекомендації, які сприяють глибшому залученню до платформи та відкриттю нового контенту (рисунок 3.7).

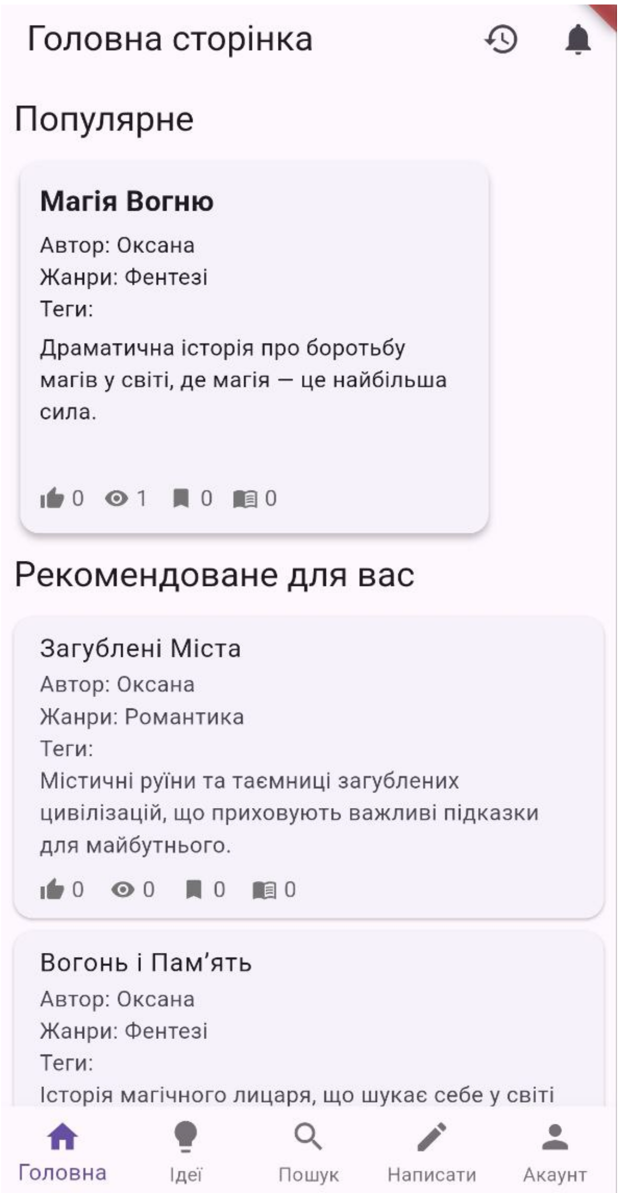

Рисунок 3.7 – Головна сторінка з рекомендаціями

Пояснення до жанрів і тропів реалізовано за допомогою підказок, які з'являються при наведенні курсора. Це полегшує розуміння незнайомих термінів, що особливо корисно для нових користувачів (рисунок 3.8).

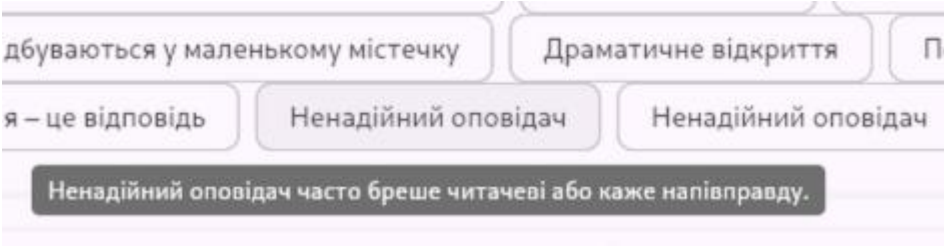

Рисунок 3.8 – Підказки при наведенні курсора для пояснення тегів

Історія читання є важливою складовою платформи, яка дозволяє зберігати перелік творів, які користувач нещодавно переглядав. Для користувачів, які ще не переглядали жодного твору відображається порожній список, інформуючи про відсутність історії переглядів (рисунок 3.9). Такий підхід допомагає уникнути плутанини і створює зрозуміле середовище для подальшої взаємодії з платформою.

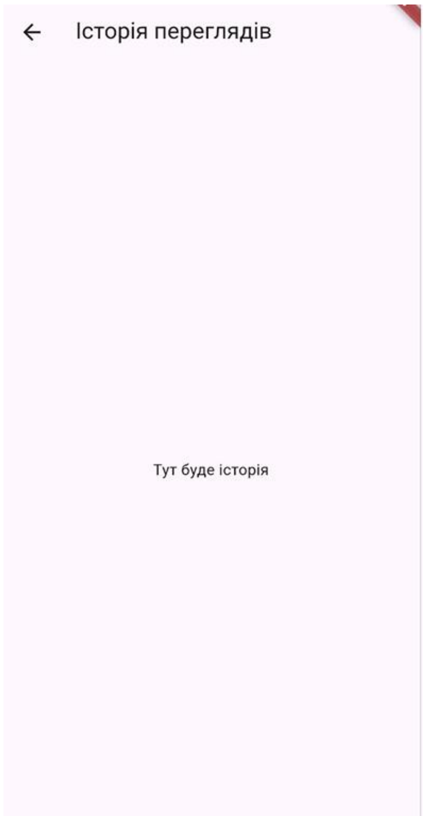

Рисунок 3.9 – Сторінка історії користувача за відсутності переглянутих творів

Функція додавання творів до розділів «Улюблене» та «Прочитати пізніше» працює стабільно та відповідно до очікувань користувачів. Користувач має можливість швидко додати будь-який твір до відповідної категорії без зайвих кроків, що значно покращує зручність користування платформою. Обрані твори зберігаються у персональному кабінеті користувача, що гарантує їх доступність у будь-який час з будь-якого пристрою після авторизації.

Відповідні вкладки «Улюблене» та «Прочитати пізніше» відображають повний список доданих творів, що дозволяє легко організувати власну бібліотеку для швидкого доступу до вподобаного чи запланованого для прочитання контенту. Крім того, реалізовано можливість видалення творів із цих закладок. Ця операція виконується швидко та без помилок, миттєво оновлюючи список і забезпечуючи користувачеві контроль над власною колекцією творів. Завдяки цьому механізму користувачі можуть ефективно керувати своїми уподобаннями, підтримуючи порядок у власному кабінеті.

Для тих користувачів, які вже мають історію читання, система автоматично зберігає інформацію про останні переглянуті твори та позицію в кожному з них та відображає на відповідній сторінці застосунку. Детальна ілюстрація цього функціоналу наведена на рисунку 3.10, де показано, як система фіксує та відображає історію читання.

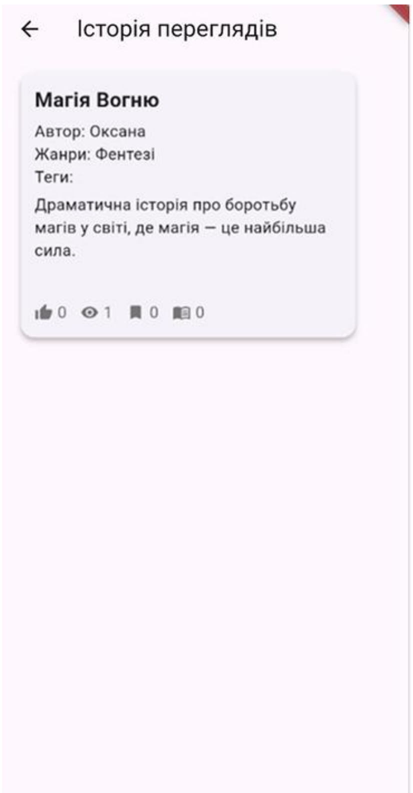

Рисунок 3.10 – Сторінка історії користувача з нещодавно відкритими творами

Під час тестування системи створення та редагування творів було перевірено збереження введеного тексту до бази даних. Усі дані, які вводить автор під час написання твору, зберігаються коректно після натискання кнопки підтвердження. Таким чином забезпечується збереження змісту навіть для великих обсягів тексту.

Також перевірено валідацію обов’язкових полів, зокрема поля з текстом твору. Якщо користувач намагається зберегти розділ без введення жодного символу, система виводить повідомлення про помилку, нагадуючи про необхідність заповнення текстового поля. Це допомагає уникнути створення порожніх або некоректних записів (рисунок 3.11).

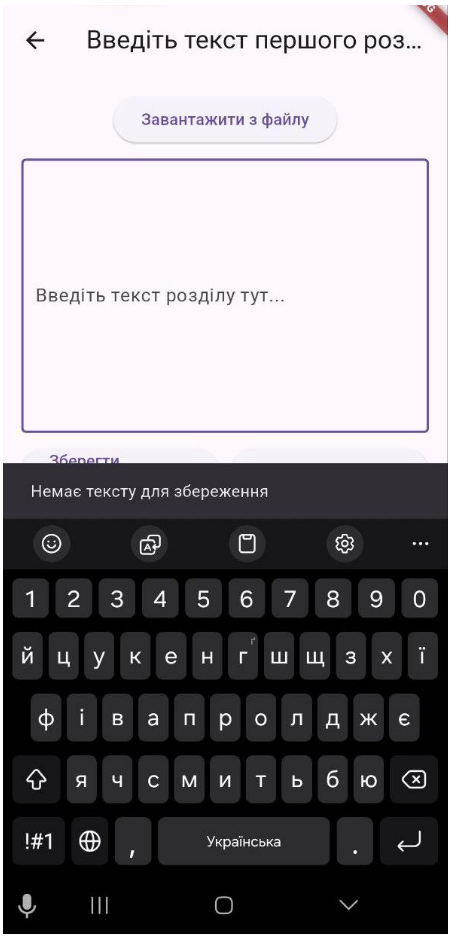

Рисунок 3.11 – Спроба збереження пустого розділу

Було проведено тестування базової функціональності системи реєстрації та входу на платформу. Під час реєстрації користувач може створити новий обліковий запис, заповнивши необхідні поля. Введені дані коректно записуються у базу даних, що підтверджується наявністю нового запису у відповідній таблиці.

Якщо всі дані були коректно введені в форму реєстрації, то знизу екрану висвітлюється повідомлення про успішну реєстрацію та здійснюється автоматичний перехід на сторінку входу.

Наступним етапом було перевірено правильність обробки некоректних даних, введених у форму реєстрації. Зокрема, при введенні пароля, який не відповідає встановленим вимогам безпеки, система відображає відповідне повідомлення про помилку, інформуючи користувача про необхідність виправлення. Аналогічним чином працює перевірка унікальності електронної пошти: якщо користувач вводить адресу, яка вже зареєстрована в системі, з'являється повідомлення про те, що така пошта вже використовується, і реєстрація не відбувається (рисунок 3.12).

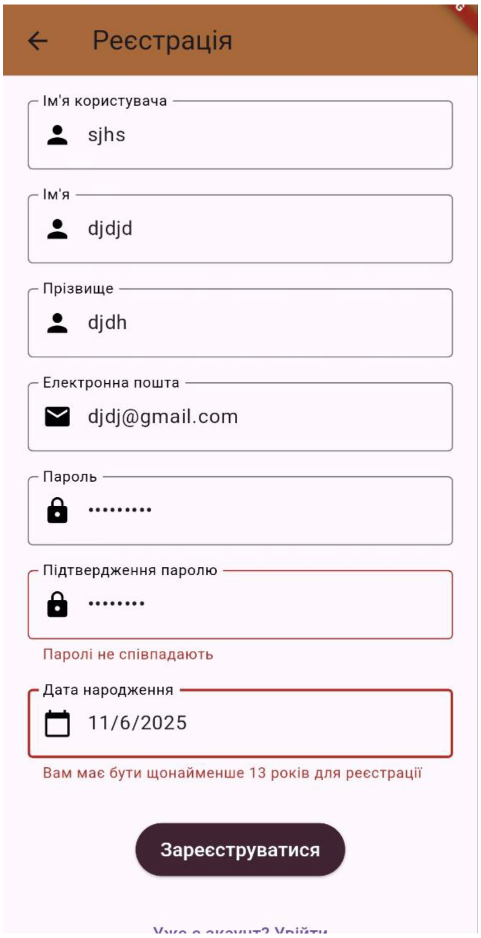

Рисунок 3.12 – Форма реєстрації при введені некоректних даних

Також було протестовано процес входу в систему, якщо користувач вводить неправильний логін або пароль, система коректно ідентифікує помилку і виводить повідомлення про невірні облікові дані. Це дозволяє уникнути несанкціонованого доступу та покращує безпеку платформи. На рисунку 3.13 продемонстровано приклад такого повідомлення, яке з’являється при спробі входу з некоректними даними.

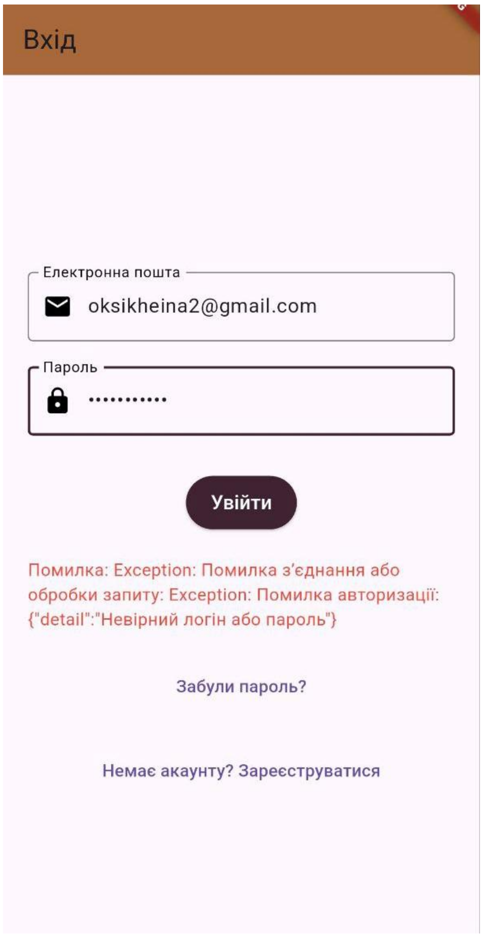

Рисунок 3.13 – Вхід в акаунт при введенні неправильного паролю

Окремо тестувалося завантаження файлів із розділами творів. Перевірка показала, що користувач може безперешкодно завантажити текстові файли у встановленому форматі, після чого система автоматично зчитує вміст і додає його як окремі глави. Завантажені файли зберігаються у відповідній директорії, а дані оновлюються в базі даних.

Повторне відкриття твору підтвердило збереження розділів у правильному порядку та зчитування вмісту без пошкоджень. Також було перевірено обмеження щодо розміру файлу та підтримуваних форматів – система коректно повідомляє про помилки у разі невідповідностей.

Інтерфейс платформи було протестовано в різних середовищах, щоб переконатися в його стабільній роботі та адаптивності. Платформа успішно функціонувала як десктопна програма на операційній системі Linux, як мобільний застосунок на пристроях з Android, а також як вебсайт, відкритий у браузері Google Chrome.

У всіх випадках інтерфейс автоматично підлаштовувався під розміри екрана, елементи відображались коректно, зберігаючи свою функціональність та зручність взаємодії для користувача. Зокрема, було перевірено роботу форм реєстрації та входу, навігації, сторінок перегляду творів та інших основних компонентів.

На рисунку 3.14 представлено приклад роботи платформи у вебверсії через браузер Chrome.

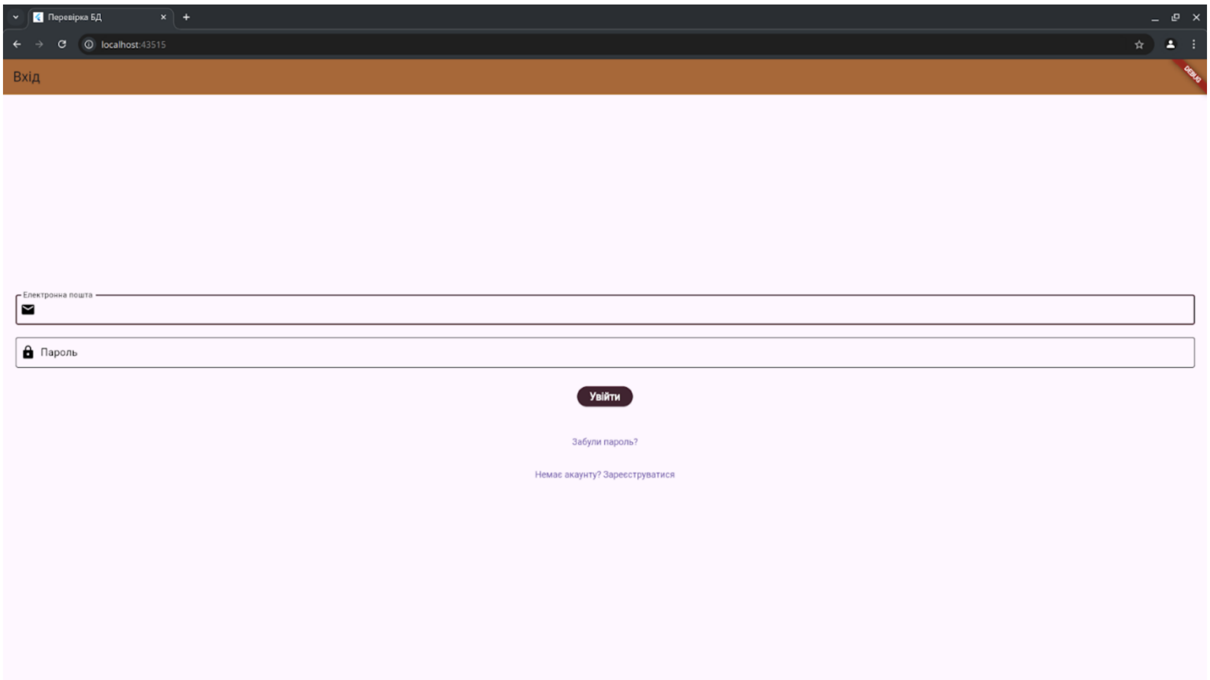

Рисунок 3.14 – Інтерфейс платформи у вебверсії через браузер Chrome

Навігація між екранами інтуїтивно зрозуміла. Переходи між розділами – головна сторінка, каталог творів, профіль користувача, читання твору, створення – відбуваються плавно без збоїв або затримок .

У результаті тестування було встановлено, що всі основні функції програмного засобу працюють коректно. Критичних помилок не виявлено. Застосунок стабільний і відповідає функціональним вимогам, заявленим на етапі проєктування.

Під час тестування взаємодії з базою даних було перевірено основні операції, зокрема збереження, оновлення та отримання даних. При створенні нового облікового запису дані користувача коректно записувалися до відповідної таблиці. Аналогічно, при додаванні нових творів, розділів або коментарів усі введені дані успішно передавалися на сервер і зберігалися в базі даних без помилок. Оновлення записів також відбувалося відповідно до очікувань. Наприклад, при редагуванні інформації про твір чи зміні профілю користувача внесені зміни відповідно зберігались у базі та одразу відображалися в інтерфейсі.

Операції з отриманням інформації – як-от завантаження списку творів, коментарів або історії читання – виконувались стабільно та без затримок. Запити до бази даних повертали актуальну інформацію, що підтверджує правильність реалізації механізмів читання та обробки даних.

Таким чином, усі ключові процеси взаємодії з базою даних функціонують згідно з технічними вимогами та забезпечують стабільну роботу платформи.

Під час тестування безпеки особливу увагу було приділено захисту персональних даних користувачів та стабільності роботи платформи в умовах потенційного несанкціонованого доступу. Зокрема, перевірено механізми автентифікації та обробки облікових даних. Одним із ключових аспектів стало збереження паролів у захищеному вигляді: перед збереженням у базі даних вони проходять процес хешування з використанням надійного криптографічного алгоритму. Завдяки цьому навіть у разі витоку бази даних зловмисник не зможе отримати паролі у відкритому вигляді (рисунок 3.15).

name	last_name	username	email	password	phone_numbe
jfu	igif	stus44	oksikheina2@gmail.com	$2b$12$6gNC7UKhnt67luELph/3E.u.Xt1	[NULL]
gb	gfv	user	user@gmail.com	$2b$12$JKtYDQ8k1tqSGEoVEG71kupFW	[NULL]

Рисунок 3.15 – Захищене збереження паролів

Також протестовано систему валідації вхідних даних, що дозволяє уникнути базових атак, таких як SQL-ін’єкції та XSS. Серверна частина фільтрує запити та не допускає введення шкідливого коду.

Також було протестовано роботу таблиці sessions, яка використовується для зберігання інформації про активні сесії користувачів. Вона містить такі поля, як ідентифікатор сесії, пов’язаний користувач, час створення та завершення сесії, IP-адреса і дані про клієнтський пристрій. Це дає змогу фіксувати факти входу та при потребі реалізувати контроль за активністю облікових записів.

Окрім цього, перевірено, що поля з обліковими даними не допускають некоректного або потенційно небезпечного введення. Система реагує на спроби некоректного входу або реєстрації з уже існуючою поштою відповідними повідомленнями. Таким чином, реалізовані заходи забезпечують базову безпеку системи, зокрема захист облікових записів користувачів та можливість фіксації сесій взаємодії з платформою.

Розроблена система демонструє грамотний підхід до архітектурного планування та технологічного вибору. Використання операційної системи Arch Linux як основи для розробки забезпечило стабільність та гнучкість, а комбінація Flutter для клієнтської частини з FastAPI для серверної створила потужну основу для кросплатформного застосунку. Вибір MariaDB як системи управління базами даних виявився обґрунтованим рішенням для надійного зберігання інформації.

Особливою перевагою розробленої системи є гібридна рекомендаційна система, що поєднує контентну та колаборативну фільтрацію. Використання алгоритмів машинного навчання дозволяє надавати персоналізовані рекомендації на основі аналізу поведінки користувачів та схожості контенту.

В цілому, створена платформа «ToRead» є технічно вдосконаленим і надійним продуктом, що успішно поєднує сучасні веб-технології з алгоритмами машинного навчання для створення персоналізованого досвіду читання та творчості. Система готова до практичного використання та має потенціал для подальшого розширення функціональності.

4 ТЕХНІКО-ЕКОНОМІЧНЕ ОБҐРУНТУВАННЯ

4.1 Аналіз ринку

Аналіз ринку продукту, який представляє платформу для публікації літературної творчості, показує, що цей виріб поєднує в собі як нові підходи до автоматизації процесів, так і модифікації існуючих рішень на ринку. Він дозволяє значно полегшити та пришвидшити процес модерації контенту завдяки використанню сучасних технологій, таких як автоматичні алгоритми для перевірки текстів та адміністрування.

Платформа не є абсолютно новим продуктом, але її концепція включає вдосконалення та інтеграцію нових функцій у вже існуючі рішення для публікації творчих робіт. Подібні продукти вже присутні на ринку, але їхні можливості щодо автоматизації процесів можуть бути обмеженими, а також відсутність чіткої інтеграції з літературними та творчими аспектами.

Потенційними замовниками цієї платформи є як окремі автори, так і літературні організації, видавці, редактори та платформи для публікації творів. Це може бути корисно як для досвідчених авторів, так і для аматорів, які хочуть публікувати свої роботи безпосередньо на платформі.

Очікується, що попит на цей продукт буде зростати в міру розвитку культурних та літературних спільнот, які все більше орієнтуються на цифрові платформи. Інтерес з боку молодих авторів та невеликих видавництв буде особливо високим, оскільки вони шукають ефективні способи публікації та монетизації творчих робіт.

Організація сервісного обслуговування повинна включати також обслуговування після випуску проєкту. Перш за все, це підтримка користувачів на всіх етапах використання платформи – від налаштування до вирішення технічних питань. Крім того, система оснащена онлайн-допомогою, інструкціями та FAQ для ефективної підтримки користувачів.

Обсяг продажу продукту може бути значним, оскільки цей продукт орієнтований на великий ринок літературних платформ, включаючи індивідуальних авторів, літературних агентів, редакторів та видавців. Потенційно, за допомогою правильної маркетингової стратегії, продукт може знайти популярність серед різних категорій користувачів.

Конкурентами на ринку є вже існуючі платформи для публікації творів, такі як Wattpad, Medium, або інші сайти для публікації. Однак ці платформи часто не мають таких глибоких можливостей автоматизації і модерації контенту, що може стати перевагою для нової платформи. Дизайн та функціональні можливості конкурентів зазвичай орієнтовані на простоту використання, але можуть не задовольняти вимоги професійних авторів чи видавців.

4.2 Економічне обґрунтування розробки

Для визначення економічної доцільності створення платформи для публікації літературної творчості необхідно провести розрахунок витрат, пов'язаних з її розробкою. Розрахунок охоплює основні категорії витрат, які виникають у процесі реалізації проєкту, включаючи оплату праці, внески до фондів, витрати на електроенергію, програмне забезпечення, технічні засоби та інші супутні витрати.

Загальна сума витрат на розробку позначається як $С_{розр}$ і визначається за формулою:

$$С_{розр} = С_{зп} + С_{вн} + С_{ел} + С_{пр} + С_{інш}$$

де:

– $С_{зп}$ – витрати на заробітну плату.

– $С_{вн}$ – витрати на внески до фондів (ЄСВ).

– $С_{ел}$ – витрати на електроенергію

– $С_{пр}$ – витрати на придбання або оренду програмного забезпечення та технічних засобів.

– $С_{інш}$ – інші супутні витрати (канцелярські товари, транспорт тощо).

Одним із ключових елементів є заробітна плата розробника, яка обчислюється за формулою:

$$С_{зп} = Т_{розр} * Р_{год}$$

де:

– $T_{розр}$ – загальна тривалість розробки в годинах.

– $P_{год}$ – погодинна оплата праці.

У таблиці 4.1 наведено погодинні ставки, обсяг виконаних робіт та загальну кількість годин для кожного фахівця, залученого до розробки проєкту.

Таблиця 4.1 – Розрахунок зарплати працівників

Роль	$P_{год}$ (грн)	$T_{розр}$ (год)	$C_{зп}$ (грн)
Проєктний менеджер	140	320	44 800
UI/UX дизайнер	120	80	9 600
Full-stack розробник	160	480	76 800
QA-інженер	130	160	20 800
SMM-спеціаліст	100	160	16 000

Загальні витрати на заробітну плату під час розробки:

$$C_{зп} = 44\,800 + 9\,600 + 76\,800 + 20\,800 + 16\,000 = 168\,000 \text{ грн}$$

Відповідно до чинного законодавства, роботодавець зобов’язаний сплатити єдиний соціальний внесок (ЄСВ), який становить 22% від суми заробітної плати:

$$C_{вн} = C_{зп} * 0{,}22 = 168\,000 * 0{,}22 = 36\,960 \text{ грн}$$

Таким чином, загальні витрати на оплату праці, включаючи ЄСВ, складають:

$$C_{праця} = C_{зп} + C_{вн} = 168\,000 + 36\,960 = 204\,960 \text{ грн}$$

Окремо було проведено розрахунок витрат на електроенергію. Для цього враховувалась середня потужність одного робочого місця, яка становить 0,044 кВт (виходячи з потужності ноутбука – 20 Вт та монітора – 24 Вт). Вартість 1 кВт·год електроенергії у 2025 році становить 6 грн. SMM-спеціаліст використовував для роботи лише мобільний телефон, що має середню потужність 0,01 кВт. Розрахунок проводився для кожного учасника команди відповідно до кількості відпрацьованих годин, що зображено в таблиці 4.2.

Таблиця 4.2 – Розрахунок витрат на електроенергію

Посада	Кількість годин	Потужність, кВт	Тариф, грн/кВт·год	Витрати, грн
Проєктний менеджер	320	0,044	6	84,48
UI/UX дизайнер	80	0,044	6	21,12
Full-stack розробник	480	0,044	6	126,72
QA-інженер	160	0,044	6	42,24
SMM-спеціаліст	160	0,010	6	9,60

Загальні витрати на електроенергію за весь період розробки склали:

$$С_{ел} = 84,48 + 21,12 + 126,72 + 42,24 + 9,6 = 284,16 \text{ грн}$$

Для хостингу платформи планується в подальшому оренда сервера на платформі AWS. Вартість оренди серверу залежить від обраної конфігурації. У цьому випадку було вибрано тип інстансу t3a.small, який має 2 ГБ оперативної пам’яті та 2 vCPU.

Вартість оренди такого серверу на AWS складає 0,0208 USD на годину. Для розрахунку місячної вартості оренди множимо на кількість годин у місяці (приблизно 730 годин).

$$С_{міс} = 0,0208 \text{ USD} * 730 \text{ годин} = 15,18 \text{ USD на місяць.}$$

При поточному курсі 1 USD = 41,81 грн, вартість оренди серверу на місяць складе:

$$С_{міс} = 15,18 * 41,81 = 634,6758 \text{ грн на місяць}$$

Так як під час розробки платформи сервер потрібен впродовж 3 місяців, то витрати на оренду обладнання склали 1 904 грн.

Крім основних витрат, також враховано супутні витрати, пов’язані із забезпеченням комфортної та стабільної роботи команди. До таких витрат належать послуги психолога для підтримки ментального здоров'я розробника, кава для працівників та оплата інтернет-зв'язку. Деталізований розрахунок наведено в таблиці 4.3.

Таблиця 4.3 – Розрахунок додаткових витрат

Найменування	Кількість	Ціна за одиницю, грн	Загальна сума, грн
Послуги психолога	1	2 000	2 000
Кава для працівників	5	100	500
Інтернет (тариф Columbus, 3 міс.)	3	170	510

Витрати на заробітну плату з урахуванням ЄСВ становили 204 960 грн. Додатково було нараховано 36 960 грн єдиного соціального внеску (22% від заробітної плати). Витрати на електроенергію під час розробки склали 284,16 грн. Оренда програмного забезпечення становить 1 904 грн. Інші витрати, зокрема послуги психолога, кава для команди та оплата інтернету, становили загалом 3 010 грн.

$$C_{розр} = 190\,400 + 36\,960 + 284{,}16 + 1\,904 + 3010 = 232\,558{,}16$$

Таким чином, загальні витрати на розробку платформи становлять 232 558,16 грн.

Окрім витрат на розробку, доцільно також врахувати щомісячні витрати на утримання платформи після її запуску.

До постійних витрат належать оплата праці фахівців, які забезпечують стабільну роботу платформи та її присутність у медіапросторі. Зокрема, модератор працює на умовах часткової зайнятості – 4 години на день, 20 робочих днів на місяць. За погодинної оплати 80 грн, його щомісячна заробітна плата становить 6 400 грн.

SMM-спеціаліст продовжує роботу і після завершення розробки. Його місячна заробітна плата становить 16 000 грн.

Вартість оренди серверу складає 634,6758 грн на місяць.

Отже, сукупні витрати на утримання платформи в місяць складають приблизно 23 034 грн.

Враховуючи соціальну орієнтацію проєкту, який не має на меті безпосередню монетизацію чи отримання прибутку, економічний ефект можна оцінювати через низку культурних і ресурсних показників, що сприяють розвитку української літератури та підвищенню доступності для авторів і читачів.

Впровадження платформи має важливий культурний ефект, оскільки сприяє популяризації української літератури. Це може позитивно вплинути на розвиток національної культури, створюючи додаткові можливості для публікації літературних творів, що стимулює інтерес до українського слова на глобальному рівні. Культурна доступність творів через платформу допоможе українським авторам знайти свою аудиторію, що має довгостроковий вплив на розвиток культурного сектору в Україні.

Одним з ключових ефектів є покращення умов для авторів, особливо для нових або маловідомих. Платформа надає їм можливість публікувати свої твори без обмежень, пов'язаних із традиційними видавництвами. Вартість доступу для авторів та читачів є низькою або безкоштовною, що дозволяє зменшити бар'єри для публікацій і читацької аудиторії. Це дозволяє молодим письменникам реалізувати свій творчий потенціал і розвивати літературну кар'єру, тим самим стимулюючи подальший розвиток української літератури.

На платформі автоматизовані такі процеси, як модерація контенту, рейтингування та інші адміністративні функції. Це дозволяє знизити потребу в ручній праці, зменшити час, що витрачається на виконання цих завдань, а також зекономити значні кошти.

$$E = (T_{ручн} - T_{авт}) * P_{год} * N$$

де:

$T_{ручн}$ – час, витрачений вручну на виконання завдання.

$T_{авт}$ – час після автоматизації.

$P_{год}$ – середня вартість години праці.

N – кількість повторень завдання за певний період.

Для даної платформи підставлено наступні значення:

$T_{ручн}$ – на кожну перевірку контенту витрачалося в середньому 30 хвилин вручну.

$T_{авт}$ – після автоматизації цей процес займає лише 5 хвилин.

$P_{год}$ –вартість години праці модератора становить 80 грн.

N – за місяць виконуються близько 200 таких завдань (публікація та перевірка контенту).

Отже, підставляючи ці значення в формулу, ми отримуємо:

$$E = (0{,}5 - 0.08) * 80 * 200 = 6\ 720 \text{ грн}$$

Завдяки автоматизації процесу публікації та перевірки контенту, економія часу складає 84 години на місяць, що дозволяє заощадити 6 720 грн на місяць на заробітній платі модератора. Що у рік складає 80 640 грн.

Автоматизація процесів на платформі дозволяє зменшити кількість працівників, які займаються рутинними завданнями, такими як публікація контенту та модерація. Це призводить до економії коштів на зарплатах.

Для розрахунку економічного ефекту від скорочення кількості працівників використовуємо наступну формулу:

$$E = Z * K * 12$$

де:

Z – середня зарплата одного працівника (грн/міс).

K – кількість працівників, яких можна вивільнити.

12 – кількість місяців у році (для річного ефекту).

Середня зарплата одного модератора на повну ставку становить 12 800 грн, а автоматизація дозволяє вивільнити трьох працівників. Тоді:

$$E = 12\ 800 * 2 * 12 = 307\ 200 \text{ грн/рік}$$

Отже, економічний ефект від скорочення кількості працівників становить 307 200 грн на рік. Це дозволяє зменшити витрати на персонал та використовувати заощаджені кошти для подальшого розвитку платформи.

Для розрахунку окупності проєкту важливо враховувати загальний економічний ефект, який дозволяє визначити, коли витрати на розробку та впровадження будуть компенсовані завдяки зекономленим коштам або додатковому прибутку.

$$Е_{загальний} = 80\ 640 \text{ грн} + 307\ 200 \text{ грн} = 387\ 840 \text{ грн}$$

У цьому випадку загальний економічний ефект складається з двох основних компонентів: економії часу працівників, а також зекономлених коштів за рахунок скорочення кількості працівників.

Для того, щоб розрахувати термін окупності проєкту, скористаємося формулою:

$$Т_{ок} = С_{розр} / Е$$

де:

$Т_{ок}$ – термін окупності проєкту (у роках або місяцях).

$С_{розр}$ – загальні витрати на розробку проєкту.

Е – економічний ефект (прибуток або зекономлені кошти за рік).

Підставимо необхідні значення в формулу:

$$Т_{ок} = 232\ 558,16 / 387\ 840 = 0,5996239686468647$$

Щоб перевести термін окупності в місяці, множимо отриманий результат на 12:

$$Т_{ок} = 0,5996 * 12 \approx 7,2 \text{ місяці}$$

Термін окупності проєкту складає 0,6 року або 7,2 місяці. Це означає, що витрати на розробку проєкту будуть повністю компенсовані за цей період, після чого проєкт почне приносити чистий економічний ефект.

Для розрахунку коефіцієнта економічної ефективності було використано формулу $К_{еф} = Е / С_{розр}$, де Е=387840 – економічний ефект, а $С_{розр}$ = 232558,16 – загальні витрати на розробку проєкту. За результатами розрахунку коефіцієнт економічної ефективності склав 1,67. Оскільки коефіцієнт більший за 1, можна стверджувати, що проєкт є вигідним, оскільки економічний ефект значно перевищує витрати на його розробку.

4.3 Обґрунтування необхідності розробки

Необхідність розробки платформи для публікації літературної творчості зумовлена численними викликами, з якими стикаються сучасні автори та читачі. Для авторів ця платформа є необхідним інструментом для публікації власних творів, який забезпечує простоту, доступність і сучасний підхід до розміщення текстів, дозволяючи митцям уникати посередників, таких як видавництва, що часто накладають свої обмеження. Крім того, платформа надає авторам можливість безпосередньо взаємодіяти з читачами, отримувати зворотний зв'язок, що стимулює розвиток їх творчості.

Для читачів важливим аспектом є можливість доступу до якісного контенту, зокрема української літератури, що набуває все більшої популярності як серед національної аудиторії, так і серед іноземних користувачів, які прагнуть ознайомитись з українською культурною спадщиною. Платформа дає можливість швидко знаходити цікаві твори за жанром, темою чи настроєм, а також відкриває нові горизонти для самовираження як авторів, так і читачів.

Запровадження системи на платформі дозволяє суттєво знизити витрати часу на модерацію, обробку контенту, ведення статистики та аналіз взаємодій завдяки автоматизації багатьох процесів. Це не лише знижує навантаження на персонал, але й дає змогу ефективніше використовувати людські ресурси, зменшуючи витрати на ручну працю та підвищуючи продуктивність команди.

Економічний ефект від впровадження цієї системи буде полягати в значній економії часу, що в свою чергу дозволить зменшити кількість помилок у процесі роботи та підвищити якість обробки даних. Зростання кількості користувачів платформи створить умови для майбутньої монетизації через рекламу, підписки чи партнерські програми, що дозволить зробити проект фінансово стабільним і самодостатнім.

Крім того, створення цієї платформи несе благодійну мету, оскільки вона сприяє розвитку української культури, допомагаючи зберігати та поширювати українську літературну спадщину в умовах глобалізації та культурної інтеграції. Платформа стає важливим інструментом для популяризації української мови та літератури, особливо серед молодого покоління, яке активно шукає нові способи вираження та відчуває потребу в підтримці національної ідентичності.

Це також допоможе вивести українських авторів на міжнародний рівень, сприяючи їх творчому росту та міжнародному визнанню. Платформа надає можливість не лише знайти нових читачів у різних країнах, а й підтримати активне культурне обговорення між українськими авторами та міжнародними колами.

Таким чином, розробка цієї платформи є важливим кроком для розвитку української літературної традиції та підтримки культурного обміну, що матиме позитивний вплив на розвиток культури в Україні та за її межами.

У цьому розділі було проведено техніко-економічне обґрунтування створення платформи для публікації літературної творчості. Аналіз ринку засвідчив актуальність і перспективність розробки такого програмного продукту. Платформа має потенціал зайняти свою нішу серед аналогічних сервісів завдяки поєднанню автоматизованої модерації, орієнтації на потреби авторів та використанню сучасних технологій.

Було визначено основні витрати на розробку проєкту, серед яких найбільшу частку займає заробітна плата команди розробників.

Загалом техніко-економічне обґрунтування підтверджує доцільність та можливість реалізації даного проєкту з урахуванням поточних ринкових умов і витрат. Очікується, що завдяки правильно вибраній маркетинговій стратегії, платформа зможе привернути увагу широкої аудиторії користувачів, включаючи авторів, видавців та творчі спільноти.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було успішно досягнуто поставленої мети створення сучасного застосунку для публікації, обміну та популяризації літературних творів. Усі завдання роботи виконано повністю: проведено аналіз предметної області та існуючих рішень, спроектовано архітектуру системи, реалізовано основні функціональні модулі, здійснено комплексне тестування та оптимізацію.

Розроблена платформа забезпечує авторам зручні інструменти для публікації творів різних жанрів, а читачам надає можливості для пошуку та оцінювання літературних робіт. Система включає модулі реєстрації та автентифікації користувачів, публікації та редагування творів, коментування та рейтингування, а також адміністративну панель управління.

Практична цінність результатів полягає у створенні готового до впровадження програмного продукту з високим ступенем функціональності. Платформа може використовуватися літературними спільнотами, видавництвами та освітніми установами для організації публікації творів та проведення конкурсів.

Подальший розвиток платформи передбачає реалізацію розширених функціональних можливостей. Планується впровадження функції обміном творами у соціальних мережах для збільшення аудиторії читачів, системи сповіщень про активність для своєчасного інформування користувачів про коментарі та оцінки. Для авторів буде додано покращену статистику творів з детальною аналітикою переглядів та взаємодії читачів, онлайн-редактор тексту з можливостями форматування та автозбереження, інструменти колаборації для спільної роботи над творами, систему досягнень для мотивації активності.

Розроблена платформа представляє собою сучасне технічне рішення, що успішно вирішує завдання впровадження цифрових технологій в літературній творчості та має значний потенціал для подальшого розвитку.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Українці стали більше читати, проте друковані книжки купують рідше: опитування. *Happy Monday* : вебсайт. URL: https://happymonday.ua/ukrayintsi-staly-bilshe-chytaty-opytuvannya (дата звернення: 07.02.2025).

2. Українська правда. Життя. Українці читають, але є нюанси. Аналіз результатів нового книжкового дослідження. *Українська правда. Життя* : вебсайт. URL: https://life.pravda.com.ua/culture/skilki-i-chogo-chitayut-ukrajinci-analiz-doslidzhennya-uik-305345/ (дата звернення: 12.02.2025).

3. Wattpad - Where stories live. *Wattpad - Where stories live* : вебсайт. URL: https://www.wattpad.com/ (дата звернення: 09.03.2025).

4. Головна сторінка | Гоголівська академія. *Гоголівська академія* : вебсайт. URL: http://www.gak.com.ua/ (дата звернення: 13.03.2025).

5. Аркуш. *Arkush.net* : вебсайт. URL: https://arkush.net/ (дата звернення: 13.03.2025).

6. Кросплатформова розробка додатків: що це таке, переваги кросплатформової розробки mobile app. *IT-компанія повного циклу розробки програмних продуктів WEZOM - Київ, Україна* : вебсайт. URL: https://wezom.com.ua/ua/blog/krossplatformennaya-razrabotka-prilozhenij (дата звернення: 18.04.2025).

7. React Native Learn once, write anywhere. *React Native Learn once, write anywhere* : вебсайт. URL: https://reactnative.dev/ (дата звернення: 19.04.2025).

8. Node.js – Run JavaScript Everywhere. *Node.js – Run JavaScript Everywhere* : вебсайт. URL: https://nodejs.org/en (дата звернення: 19.04.2025).

9. PostgreSQL. *PostgreSQL* : вебсайт. URL: https://www.postgresql.org/ (дата звернення: 23.04.2025).

10. Перші кроки в NLP: розглядаємо Python-бібліотеку scikit-learn в реальному завданні. *DOU* : вебсайт. URL: https://dou.ua/lenta/articles/first-steps-in-nlp-scikit-learn/ (дата звернення: 30.04.2025).

ДОДАТКИ

Додаток А

Сутності бази даних

Сутність	Опис
users	Зберігає дані користувачів
works	Містить інформацію про літературні твори
chapters	Представляє окремі розділи твору
categories	Описує жанрову або тематичну приналежність творів.
Tags	Ключові слова для класифікації творів, дозволяють реалізувати гнучкий пошук.
Work_tags	Проміжна таблиця для зв'язку творів із тегами.
Work_statuses	Містить статуси творів
user_interactions	Фіксує взаємодію користувачів з творами
comments	Містить коментарі користувачів до розділів творів.
Ratings	Зберігає оцінки, поставлені користувачами творам.
Subscriptions	Реалізує механізм підписки на твори для відстеження оновлень.
Sessions	Відповідає за зберігання інформації про сесії входу користувача
comment_reports	Зберігає скарги на коментарі, подані користувачами
ideas	Містить ідеї творів, запропоновані користувачами

Додаток Б

SQL-скрипт реалізації бази даних

```
CREATE TABLE work_statuses (
     id INT PRIMARY KEY,
     name VARCHAR(20) NOT NULL
);
CREATE TABLE users (
     id UUID PRIMARY KEY,
     name VARCHAR(50) NOT NULL,
     last_name VARCHAR(50) NOT NULL,
     username VARCHAR(50) UNIQUE NOT NULL,
     email VARCHAR(100) UNIQUE NOT NULL,
     password VARCHAR(255) NOT NULL,
     phone_number VARCHAR(15),
     avatar_path VARCHAR(150),
     birth INT,
     bio TEXT
);
CREATE TABLE categories (
     id INT PRIMARY KEY,
     name VARCHAR(100) NOT NULL,
     description TEXT
);
CREATE TABLE works (
     id UUID PRIMARY KEY,
     title VARCHAR(100) NOT NULL,
     author UUID,  -- Змінено тип на UUID
     description TEXT,
     cover_path VARCHAR(150),
     file_path VARCHAR(150),
     created_at TIMESTAMP NOT NULL,
     updated_at TIMESTAMP NOT NULL,
     category_id INT,
     age_limit INT,
     status_id INT,
     FOREIGN KEY (status_id) REFERENCES work_statuses(id),
     FOREIGN KEY (category_id) REFERENCES categories(id),
     FOREIGN KEY (author) REFERENCES users(id)  -- Тепер
типи сумісні
);
CREATE TABLE subscriptions (
     id UUID PRIMARY KEY,
     work_id UUID,
     user_id UUID,
     created_at TIMESTAMP NOT NULL,
     FOREIGN KEY (work_id) REFERENCES works(id),
     FOREIGN KEY (user_id) REFERENCES users(id)
);
CREATE TABLE chapters (
     id UUID PRIMARY KEY,
```

```
        work_id UUID,
        num INT NOT NULL,
        title VARCHAR(255) NOT NULL,
        file_path VARCHAR(255),
        FOREIGN KEY (work_id) REFERENCES works(id)
    );
    CREATE TABLE user_interactions (
        id INT PRIMARY KEY,
        work_id UUID,
        user_id UUID,
        is_saved BOOLEAN NOT NULL DEFAULT FALSE,
        is_liked BOOLEAN NOT NULL DEFAULT FALSE,
        is_viewed BOOLEAN NOT NULL DEFAULT FALSE,
        is_read BOOLEAN NOT NULL DEFAULT FALSE,
        FOREIGN KEY (work_id) REFERENCES works(id),
        FOREIGN KEY (user_id) REFERENCES users(id)
    );
    CREATE TABLE comments (
        id INT PRIMARY KEY,
        user_id UUID,
        chapter_id UUID,
        text TEXT NOT NULL,
        created_at TIMESTAMP NOT NULL,
        FOREIGN KEY (user_id) REFERENCES users(id),
        FOREIGN KEY (chapter_id) REFERENCES chapters(id)
    );
    CREATE TABLE ratings (
        id INT PRIMARY KEY,
        work_id UUID,
        user_id UUID,
        rating INT NOT NULL,
        FOREIGN KEY (work_id) REFERENCES works(id),
        FOREIGN KEY (user_id) REFERENCES users(id)
    );
    CREATE TABLE tags (
        id INT PRIMARY KEY,
        name VARCHAR(50) NOT NULL
    );
    CREATE TABLE work_tags (
        work_id UUID,
        tag_id INT,
        FOREIGN KEY (work_id) REFERENCES works(id),
        FOREIGN KEY (tag_id) REFERENCES tags(id),
        PRIMARY KEY (work_id, tag_id)
    );
    CREATE TABLE sessions (
        id UUID PRIMARY KEY,
        user_id UUID,
        created_at TIMESTAMP NOT NULL,
        expires_at TIMESTAMP NOT NULL,
        ip_address VARCHAR(45),
        user_agent TEXT,
        FOREIGN KEY (user_id) REFERENCES users(id));
```

```
CREATE TABLE comment_reports (
     id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
     comment_id UUID REFERENCES comments(id) ON DELETE
CASCADE,
     user_id UUID REFERENCES users(id) ON DELETE CASCADE,
     reason TEXT NOT NULL,
     created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
CREATE TABLE ideas (
     id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
     user_id UUID NOT NULL,
     title VARCHAR(100) NOT NULL,
     description TEXT NOT NULL,
     created_at TIMESTAMP NOT NULL DEFAULT
CURRENT_TIMESTAMP,
     linked_work_id UUID,

     FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE
CASCADE,
     FOREIGN KEY (linked_work_id) REFERENCES works(id) ON
DELETE SET NULL
);
```

Додаток В

Лістинг програмного коду ORM-моделі

```
from sqlalchemy import Column, String, Integer, Text, UUID,
Boolean, TIMESTAMP, ForeignKey
from sqlalchemy.orm import relationship
from sqlalchemy.dialects.postgresql import UUID as PGUUID
from sqlalchemy.ext.declarative import declarative_base
import uuid
Base = declarative_base()
class WorkStatus(Base):
     __tablename__ = 'work_statuses'

     id = Column(Integer, primary_key=True)
     name = Column(String(20), nullable=False)

class User(Base):
     __tablename__ = 'users'
     id = Column(PGUUID(as_uuid=True), primary_key=True)
     name = Column(String(50), nullable=False)
     last_name = Column(String(50), nullable=False)
     username = Column(String(50), unique=True,
nullable=False)
     email = Column(String(100), unique=True,
nullable=False)
     password = Column(String(255), nullable=False)
     phone_number = Column(String(15))
     avatar_path = Column(String(150))
     birth = Column(Integer)
     bio = Column(Text)

class Category(Base):
     __tablename__ = 'categories'

     id = Column(Integer, primary_key=True)
     name = Column(String(100), nullable=False)
     description = Column(Text)
class Work(Base):
     __tablename__ = 'works'
     id = Column(UUID(as_uuid=True), primary_key=True,
default=uuid.uuid4)
     title = Column(String(100), nullable=False)
     author = Column(UUID(as_uuid=True),
ForeignKey('users.id'), nullable=False)
     description = Column(Text)
     cover_path = Column(String(150))
     file_path = Column(String(150))

     created_at = Column(TIMESTAMP, nullable=False)
     updated_at = Column(TIMESTAMP, nullable=False)
```

```
     category_id = Column(Integer,
ForeignKey('categories.id'))
     age_limit = Column(Integer)
     status_id = Column(Integer,
ForeignKey('work_statuses.id'))

     ratings = relationship("Rating", back_populates="work")
     author_user = relationship("User", backref="works")
     category = relationship("Category", backref="works")
     status = relationship("WorkStatus", backref="works")
     tags = relationship("Tag", secondary="work_tags",
backref="works", overlaps="work,tag")

class Subscription(Base):
     __tablename__ = 'subscriptions'

     id = Column(PGUUID(as_uuid=True), primary_key=True)
     work_id = Column(PGUUID(as_uuid=True),
ForeignKey('works.id'), nullable=False)
     user_id = Column(PGUUID(as_uuid=True),
ForeignKey('users.id'), nullable=False)
     created_at = Column(TIMESTAMP, nullable=False)

     work = relationship('Work')
     user = relationship('User')

class Chapter(Base):
     __tablename__ = 'chapters'

     id = Column(PGUUID(as_uuid=True), primary_key=True)
     work_id = Column(PGUUID(as_uuid=True),
ForeignKey('works.id'), nullable=False)
     num = Column(Integer, nullable=False)
     title = Column(String(255), nullable=False)
     file_path = Column(String(255))
     work = relationship('Work')

class UserInteraction(Base):
     __tablename__ = 'user_interactions'
     id = Column(Integer, primary_key=True)
     work_id = Column(PGUUID(as_uuid=True),
ForeignKey('works.id'), nullable=False)
     user_id = Column(PGUUID(as_uuid=True),
ForeignKey('users.id'), nullable=False)
     is_saved = Column(Boolean, default=False,
nullable=False)
     is_liked = Column(Boolean, default=False,
nullable=False)
     is_viewed = Column(Boolean, default=False,
nullable=False)
     is_read = Column(Boolean, default=False,
nullable=False)
```

```
    work = relationship('Work')
    user = relationship('User')

class Comment(Base):
    __tablename__ = 'comments'

    id = Column(Integer, primary_key=True)
    user_id = Column(PGUUID(as_uuid=True),
ForeignKey('users.id'), nullable=False)
    chapter_id = Column(PGUUID(as_uuid=True),
ForeignKey('chapters.id'), nullable=False)
    text = Column(Text, nullable=False)
    created_at = Column(TIMESTAMP, nullable=False)

    user = relationship('User')
    chapter = relationship('Chapter')
class Rating(Base):
    __tablename__ = 'ratings'

    id = Column(Integer, primary_key=True)
    work_id = Column(UUID(as_uuid=True),
ForeignKey("works.id"))
    user_id = Column(UUID(as_uuid=True),
ForeignKey("users.id"))
    rating = Column(Integer, nullable=False)

    work = relationship("Work", back_populates="ratings")
class Tag(Base):
    __tablename__ = 'tags'
    id = Column(Integer, primary_key=True)
    name = Column(String(50), nullable=False)
    description = Column(String)

class WorkTag(Base):
    __tablename__ = 'work_tags'

    work_id = Column(PGUUID(as_uuid=True),
ForeignKey('works.id'), primary_key=True)
    tag_id = Column(Integer, ForeignKey('tags.id'),
primary_key=True)
    work = relationship('Work', overlaps="tags,works")
    tag = relationship('Tag', overlaps="tags,works")

class Session(Base):
    __tablename__ = 'sessions'
    id = Column(PGUUID(as_uuid=True), primary_key=True)
    user_id = Column(PGUUID(as_uuid=True),
ForeignKey('users.id'), nullable=False)
    created_at = Column(TIMESTAMP, nullable=False)
    expires_at = Column(TIMESTAMP, nullable=False)
    ip_address = Column(String(45))
    user_agent = Column(Text)
    user = relationship('User')
```

Додаток Г

Лістинг програмного коду системи авторизації

```
from fastapi import APIRouter, Depends, HTTPException,
status
from sqlalchemy.orm import Session
import logging
import uuid
from schemas import LoginRequest, RegisterRequest
from schemas import RegisterRequest
from database import get_db
from models import User
from passlib.context import CryptContext

pwd_context = CryptContext(schemes=["bcrypt"],
deprecated="auto")
def hash_password(password: str) -> str:
     return pwd_context.hash(password)
def verify_password(plain_password: str, hashed_password:
str) -> bool:
     return pwd_context.verify(plain_password,
hashed_password)

router = APIRouter()
from datetime import datetime
@router.post('/register')
async def register(request: RegisterRequest, db: Session =
Depends(get_db)):
     logging.info(f"Реєстрація користувача:
{request.email}")
     if db.query(User).filter(User.email ==
request.email).first():
     logging.warning(f"Спроба реєстрації з існуючим email:
{request.email}")
     raise
HTTPException(status_code=status.HTTP_400_BAD_REQUEST,
detail="Email вже використовується")
     if db.query(User).filter(User.username ==
request.username).first():
     logging.warning(f"Спроба реєстрації з існуючим
username: {request.username}")
     raise
HTTPException(status_code=status.HTTP_400_BAD_REQUEST,
detail="Ім'я користувача вже використовується")
     birth_year = None
     if request.birth:
     try:
          dt =
datetime.strptime(request.birth.split("T")[0], "%Y-%m-%d")
          birth_year = dt.year
     except ValueError:
```

```
            raise HTTPException(status_code=400,
detail="Невірний формат дати народження.")

      new_user = User(
      id=uuid.uuid4(),
      name=request.name,
      last_name=request.last_name,
      username=request.username,
      email=request.email,
      password=hash_password(request.password),
      phone_number=request.phone_number,
      avatar_path=request.avatar_path,
      birth=birth_year,
      bio=request.bio
      )

      db.add(new_user)
      db.commit()
      db.refresh(new_user)
      return {
      "success": True,
      "message": "Успішна реєстрація",
      "userId": str(new_user.id)
      }
@router.post('/login')
async def login(request: LoginRequest, db: Session =
Depends(get_db)):
      logging.info(f"Login attempt: {request.email}")
      user = db.query(User).filter(User.email ==
request.email).first()

      if user is None or not
verify_password(request.password, user.password):
      logging.warning("Невірний логін або пароль")
      raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Невірний логін або пароль",
      )

      logging.info(f"Користувач {request.email} увійшов
успішно")
      return {
      "success": True,
      "message": "Успішний вхід",
      "userId": str(user.id)
      }
```

Додаток Ґ

Лістинг програмного коду отримання популярних творів

```
def get_popular_works(db: Session = Depends(get_db)):
     interaction_counts = (
     db.query(
          UserInteraction.work_id,
          func.count().filter(UserInteraction.is_viewed ==
True).label("views"),
          func.count().filter(UserInteraction.is_liked ==
True).label("likes"),
          func.count().filter(UserInteraction.is_read ==
True).label("reads"),
          func.count().filter(UserInteraction.is_saved ==
True).label("saves"),
     )
     .group_by(UserInteraction.work_id)
     .subquery()
     )
     works_with_data = (
     db.query(Work,
               interaction_counts.c.views,
               interaction_counts.c.likes,
               interaction_counts.c.reads,
               interaction_counts.c.saves)
     .join(interaction_counts, Work.id ==
interaction_counts.c.work_id)
     .options(
          joinedload(Work.author_user),
          joinedload(Work.category),
          joinedload(Work.tags)
     )
     .order_by(desc(interaction_counts.c.views))
     .all()
     )
     result = []
     for work, views, likes, reads, saves in works_with_data:
     result.append({
          "id": str(work.id),
          "title": work.title,
          "description": work.description,
          "author": work.author_user.name if work.author_user
else "Невідомий автор",
          "genres": [work.category.name] if work.category else
[],
          "tags": [tag.name for tag in work.tags],
          "views": views,
          "likes": likes,
          "reads": reads,
          "saves": saves
     })
```

Додаток Д

Лістинг програмного коду додавання коментарів

```
def add_comment(
     comment_data: CommentCreate,
     db: Session = Depends(get_db),
     current_user: User = Depends(get_current_user)
):
     """Додавання коментарів до творів або розділів"""
     if not comment_data.work_id and not
comment_data.chapter_id:
     raise HTTPException(status_code=400, detail="Потрібно
вказати або work_id, або chapter_id")

     if comment_data.work_id:
     if not
db.query(Work).filter_by(id=comment_data.work_id).first():
          raise HTTPException(status_code=404, detail="Твір не
знайдено")
     if comment_data.chapter_id:
     if not
db.query(Chapter).filter_by(id=comment_data.chapter_id).first
():
          raise HTTPException(status_code=404, detail="Розділ
не знайдено")

     comment = Comment(
     user_id=current_user.id,
     work_id=comment_data.work_id,
     chapter_id=comment_data.chapter_id,
     content=comment_data.content,
     )
     db.add(comment)
     db.commit()
     db.refresh(comment)
     return comment
```

Додаток Е

Програмний код функції гнучкого оновлення профілю

```
def update_user_profile(db: Session, user_id: uuid.UUID,
name: Optional[str] = None,
                         last_name: Optional[str] = None,
username: Optional[str] = None,
                         email: Optional[str] = None,
phone_number: Optional[str] = None,
                         avatar_path: Optional[str] = None,
birth: Optional[int] = None,
                         bio: Optional[str] = None):

     user = db.query(User).filter(User.id == user_id).first()
     if not user:
     return None

     if name is not None:
     user.name = name
     if last_name is not None:
     user.last_name = last_name
     if username is not None:
     user.username = username
     if email is not None:
     user.email = email
     if phone_number is not None:
     user.phone_number = phone_number
     if avatar_path is not None:
     user.avatar_path = avatar_path
     if birth is not None:
     user.birth = birth
     if bio is not None:
     user.bio = bio

     db.commit()
     return user
```

Додаток Є

Універсальна функція пошуку з багатокритеріальною фільтрацією

```
def search_works(db: Session,
               title: Optional[str] = None,
               author_name: Optional[str] = None,
               tag_names: Optional[List[str]] = None,
               genre_id: Optional[int] = None,
               status_id: Optional[int] = None,
               order_by_popularity: bool = False) ->
List[Work]:

     query = db.query(Work)
     if title:
     query = query.filter(Work.title.ilike(f"%{title}%"))
     if author_name:
     query = query.join(User, Work.author == User.id).filter(
          (User.name.ilike(f"%{author_name}%")) |
(User.username.ilike(f"%{author_name}%"))
     )
     if genre_id:
     query = query.filter(Work.category_id == genre_id)
     if status_id:
     query = query.filter(Work.status_id == status_id)
     if tag_names:
     query = query.join(WorkTag).join(Tag)
     for tag_name in tag_names:
          query = query.filter(Tag.name.ilike(tag_name))
     if order_by_popularity:
     avg_rating = func.avg(Rating.rating)

     query =
query.outerjoin(Rating).group_by(Work.id).order_by(desc(avg_r
ating))
     return query.all()
```

Додаток Ж

Програмний код реалізації системи рекомендацій

```
from typing import List
import pandas as pd
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.feature_extraction.text import TfidfVectorizer
from sqlalchemy.orm import Session
from sqlalchemy import func
from fastapi import APIRouter, Depends, HTTPException
from database import get_db
from models import Rating, Work
from sqlalchemy.orm import joinedload
router = APIRouter()
def create_user_item_matrix(db: Session):
     ratings = db.query(Rating).all()
     data = [(str(r.user_id), str(r.work_id), r.rating) for
r in ratings]
     df = pd.DataFrame(data, columns=['user_id', 'work_id',
'rating'])
     if df.empty:
     return pd.DataFrame()
     matrix = df.pivot(index='user_id', columns='work_id',
values='rating').fillna(0)
     return matrix
def calculate_content_similarity(works: List[Work]):
     descriptions = [w.description or '' for w in works]
     vectorizer = TfidfVectorizer(stop_words='english')
     tfidf_matrix = vectorizer.fit_transform(descriptions)
     return cosine_similarity(tfidf_matrix)
def calculate_user_similarity(user_item_matrix:
pd.DataFrame):
     return cosine_similarity(user_item_matrix)
def get_content_based(user_id: str, works: List[Work],
content_sim):
     user_rated_works = [w for w in works if str(user_id) in
[str(r.user_id) for r in w.ratings]]
     if not user_rated_works:
     return []
     rated_indices = [works.index(w) for w in
user_rated_works if w in works]
     similarity_scores = np.mean([content_sim[i] for i in
rated_indices], axis=0)
     top_indices = np.argsort(similarity_scores)[::-1][:6]

return [works[i] for i in top_indices if works[i] not in
user_rated_works]
def get_collaborative_based(user_id: str, matrix:
pd.DataFrame, similarity):
```

```
    if user_id not in matrix.index:
    return []
    user_idx = matrix.index.get_loc(user_id)
    sim_scores = list(enumerate(similarity[user_idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1],
reverse=True)
    top_users_idx = [i for i, _ in sim_scores[1:6]]
    top_users = matrix.iloc[top_users_idx]
    mean_scores =
top_users.mean().sort_values(ascending=False)
    recommended_work_ids = mean_scores.index[:5].tolist()
    return recommended_work_ids
def hybrid_recommendation(user_id: str, db: Session):
    works = db.query(Work).all()
    if not works:
    return []
    matrix = create_user_item_matrix(db)
    content_sim = calculate_content_similarity(works)
    collab_sim = calculate_user_similarity(matrix) if not
matrix.empty else None
    content_recs = get_content_based(user_id, works,
content_sim)
    collab_recs_ids = get_collaborative_based(user_id,
matrix, collab_sim) if collab_sim is not None else []
    collab_recs = [w for w in works if str(w.id) in
collab_recs_ids]
    if not collab_recs:
    print(f"Користувач {user_id} не має колаборативних
рекомендацій, вибір за популярністю.")
    popular_works = db.query(Work,
func.count(Rating.id).label('rating_count'),
func.avg(Rating.rating).label('avg_rating')) \
        .join(Rating, Rating.work_id == Work.id) \
        .group_by(Work.id) \
        .order_by(func.count(Rating.id).desc(),
func.avg(Rating.rating).desc()) \
        .limit(5) \
        .all()
    return [work[0] for work in popular_works]
    combined = list({w.id: w for w in content_recs +
collab_recs}.values())[:5]
    return combined
@router.get("/recommendations/{user_id}")
async def get_recommendations(user_id: str, db: Session =
Depends(get_db)):
    try:
    recommendations = hybrid_recommendation(user_id, db)

    recommendations_data = []
    for work in recommendations:
        work_with_details = db.query(Work).options(
            joinedload(Work.category),
            joinedload(Work.author_user),
```

```
                joinedload(Work.tags)
            ).filter(Work.id == work.id).first()
            if not work_with_details:
                continue
            work_data = {
                "id": str(work_with_details.id),
                "title": work_with_details.title,
                "description": work_with_details.description,
                "author": work_with_details.author_user.name
if work_with_details.author_user else "Невідомий автор",
                "genres": [work_with_details.category.name]
if work_with_details.category else [],
                "tags": [tag.name for tag in
work_with_details.tags],
            }
            recommendations_data.append(work_data)
        return recommendations_data
        except Exception as e:
        raise HTTPException(status_code=500, detail=f"Не
вдалося отримати рекомендації: {str(e)}")
```

					КР.КН 25.588.07.000 ПЗ	Арк.
						105
Змн.	Арк.	№ докум.	Підпис	Дата		

Додаток І

Виконання HTTP-запиту для входу користувача

```
import 'package:http/http.dart' as http;
import 'dart:convert';
import 'dart:io';
class AuthService {
 Future<String?> login(String email, String password) async
{
   try {
     final response = await http.post(
       Uri.parse('http://192.168.170.157:8000/login'),
       headers: {'Content-Type': 'application/json'},
       body: json.encode({
         'email': email,
         'password': password,
       }),
     );
     if (response.statusCode == 200) {
       final data = json.decode(response.body);
       if (data['success'] == true && data['userId'] !=
null) {
         return data['userId'].toString();
else {
         throw Exception('Сервер не повернув userId');
       }
     } else if (response.statusCode == 401) {
       final data = json.decode(response.body);
       throw Exception(data['detail'] ?? 'Невірні логін або
пароль');
     } else {
       throw Exception('Сервер повернув помилку:
${response.statusCode}');
     }
   } on SocketException {
     throw Exception(
         'Помилка з’єднання: сервер недоступний або
відсутнє інтернет-з’єднання.');
   } catch (e) {
     throw Exception('Помилка з’єднання або обробки запиту:
$e');
   }
 }
}
```

Додаток К

Оформлення стилів інтерфейсу в Flutter

```
import 'package:flutter/material.dart';
import 'colors.dart';

class AppStyles {
 static InputDecoration inputDecoration = InputDecoration(
   labelStyle: TextStyle(color: Colors.black),
   prefixIconColor: Colors.black,
   border: OutlineInputBorder(
     borderSide: BorderSide(
       color: AppColors.borderColor,
       width: 2.0,
     ),
   ),
   focusedBorder: OutlineInputBorder(
     borderSide: BorderSide(
       color: AppColors.secondaryColor,
       width: 2.0,
     ),
   ),
 );

 static TextStyle titleTextStyle = TextStyle(
   fontSize: 24.0,
   fontWeight: FontWeight.bold,
   color: Colors.black,
 );

 static TextStyle bodyTextStyle = TextStyle(
   fontSize: 16.0,
   color: Colors.black54,
 );
 static ButtonStyle elevatedButtonStyle =
ElevatedButton.styleFrom(
   backgroundColor: AppColors.secondaryColor,
   foregroundColor: Colors.white,
   padding: EdgeInsets.symmetric(horizontal: 20.0, vertical:
12.0),
   textStyle: TextStyle(fontSize: 16.0, fontWeight:
FontWeight.w600),
 );
}
```

Додаток Л

Реалізація введення та завантаження тексту розділу твору

```
import 'package:flutter/material.dart';
import 'package:file_picker/file_picker.dart';
import 'dart:io';

class ChapterInputPage extends StatefulWidget {
 const ChapterInputPage({super.key});
 @override
 ChapterInputPageState createState() =>
ChapterInputPageState();
}
class ChapterInputPageState extends State<ChapterInputPage> {
 final TextEditingController _chapterController =
TextEditingController();
 void _uploadFile() async {
   FilePickerResult? result = await
FilePicker.platform.pickFiles(
     type: FileType.custom,
     allowedExtensions: ['txt', 'doc', 'docx', 'rtf'],);
   if (result != null && result.files.single.path != null) {
     String filePath = result.files.single.path!;
     try {
       final fileContent = await
File(filePath).readAsString();
       setState(() {
         _chapterController.text = fileContent;
       });
     } catch (e) {
       ScaffoldMessenger.of(context).showSnackBar(
         SnackBar(content: Text('Помилка читання файлу:
$e')),);}
   } else {
     ScaffoldMessenger.of(context).showSnackBar(
       SnackBar(content: Text('Вибір файлу скасовано')),
     );}}
 void _publish() {
   final text = _chapterController.text.trim();
   if (text.isEmpty) {
     ScaffoldMessenger.of(context).showSnackBar(
       SnackBar(content: Text('Текст першого розділу
порожній')),
     );
     return; }
   ScaffoldMessenger.of(context).showSnackBar(
     SnackBar(content: Text('Твір опубліковано!')),
   );}
 void _saveDraft() {
   final text = _chapterController.text.trim();
```

```
    if (text.isEmpty) {
         ScaffoldMessenger.of(context).showSnackBar(
           SnackBar(content: Text('Немає тексту для
    збереження')),
         );
         return;
       }
       ScaffoldMessenger.of(context).showSnackBar(
         SnackBar(content: Text('Чернетка збережена')),
       );}
     @override
     Widget build(BuildContext context) {
       return Scaffold(
         appBar: AppBar(
           title: Text('Введіть текст першого розділу'),
         ),
         body: Padding(
           padding: const EdgeInsets.all(16.0),
           child: Column(
             children: [
               ElevatedButton(
                 onPressed: _uploadFile,
                 child: Text('Завантажити з файлу'),
               ),
               SizedBox(height: 10),
               Expanded(
                 child: TextField(
                   controller: _chapterController,
                   maxLines: null,
                   expands: true,
                   decoration: InputDecoration(
                     border: OutlineInputBorder(),
                     hintText: 'Введіть текст розділу тут...',
                   ),
                 ),),
               SizedBox(height: 10),
               Row(
                 children: [
                   Expanded(
                     child: ElevatedButton(
                       onPressed: _saveDraft,
                       child: Text('Зберегти чернетку'),
                     ),
                   ),
                   SizedBox(width: 10),
                   Expanded(
                     child: ElevatedButton(
                       onPressed: _publish,
                       child: Text('Опублікувати')),
                   ),],
               )],
           ),
         ));}}
```

Додаток М

Лістинг програмного коду створення твору

```
import 'package:flutter/material.dart';
import 'package:http/http.dart' as http;
import 'dart:convert';
import 'chapter_input_page.dart';
class CreateWorkScreen extends StatefulWidget {
 @override
 _CreateWorkScreenState createState() =>
_CreateWorkScreenState();
}
class _CreateWorkScreenState extends
State<CreateWorkScreen> {
 final _formKey = GlobalKey<FormState>();
 final _titleController = TextEditingController();
 final _descriptionController = TextEditingController();
 String? selectedCategory;
 String? selectedStatus;
 final Set<String> selectedTagIds = {};
 List<Map<String, dynamic>> categories = [];
 List<Map<String, dynamic>> tags = [];
 List<Map<String, dynamic>> statuses = [];
 bool isLoading = true;
@override
 void initState() {
   super.initState();
   loadFormData();
 }
Future<void> loadFormData() async {
   try {
     final responses = await Future.wait([
http.get(Uri.parse('http://192.168.170.157:8000/categories'
)),
http.get(Uri.parse('http://192.168.170.157:8000/tags')),

http.get(Uri.parse('http://192.168.170.157:8000/work-
statuses')),
     ]);
     if (responses.every((res) => res.statusCode == 200)) {
       setState(() {
         categories = List<Map<String, dynamic>>.from(
json.decode(utf8.decode(responses[0].bodyBytes)));
         tags = List<Map<String, dynamic>>.from(
json.decode(utf8.decode(responses[1].bodyBytes)));
         statuses = List<Map<String, dynamic>>.from(
json.decode(utf8.decode(responses[2].bodyBytes)));
         isLoading = false;
       });
     } else {
       throw Exception("Не вдалося завантажити дані");
```

```
      }

} catch (e) {
     print("Помилка: $e");
     ScaffoldMessenger.of(context).showSnackBar(
       SnackBar(content: Text("Помилка завантаження
даних")),
     );
   }
 }
 @override
 Widget build(BuildContext context) {
   return Scaffold(
     appBar: AppBar(title: Text("Створити твір")),
     body: isLoading
         ? Center(child: CircularProgressIndicator())
         : Padding(
             padding: const EdgeInsets.all(16.0),
             child: Form(
               key: _formKey,
               child: ListView(
                 children: [
                   TextFormField(
                     controller: _titleController,
                     decoration: InputDecoration(labelText:
"Назва твору"),
                     validator: (value) => (value == null
|| value.isEmpty)
                         ? "Введіть назву"
                         : null,
                   ),
                   TextFormField(
                     controller: _descriptionController,
                     decoration: InputDecoration(labelText:
"Опис"),
                     maxLines: 3,
                   ),
                   SizedBox(height: 16),
                   DropdownButtonFormField<String>(
                     decoration: InputDecoration(labelText:
"Жанр"),
                     value: selectedCategory,
                     items: categories
                         .map((cat) =>
DropdownMenuItem<String>(
                              value: cat['id'].toString(),
                              child: Text(cat['name']),
                            ))
                         .toList(),
                     onChanged: (value) =>
                         setState(() => selectedCategory =
value),
                     validator: (value) =>
```

					КР.КН 25.588.07.000 ПЗ	Арк.
						111
Змн.	Арк.	№ докум.	Підпис	Дата		

```
                         value == null ? 'Оберіть жанр' :
null,
                   ),
                   DropdownButtonFormField<String>(
                     decoration: InputDecoration(labelText:
"Статус"),
                     value: selectedStatus,
                     items: statuses
                         .where((status) =>
                             status['id'].toString() !=
'4')
                         .map((status) =>
DropdownMenuItem<String>(
                               value:
status['id'].toString(),
                               child: Text(status['name']),
                             ))
                         .toList(),
                     onChanged: (value) =>
                         setState(() => selectedStatus =
value),
                     validator: (value) =>
                         value == null ? 'Оберіть статус' :
null,
                   ),
                   SizedBox(height: 16),
                   TagSelector(tags: tags, selectedTags:
selectedTagIds),
                   SizedBox(height: 20),
                   ElevatedButton(
                     onPressed: () {
                       if
(_formKey.currentState!.validate()) {
                         Navigator.push(
                           context,
                           MaterialPageRoute(
                               builder: (context) =>
ChapterInputPage()),
                         );
                       }
                     },
                     child: Text("Продовжити"),
                   ),
                 ],
               ),
             ),
           ),
    );
  }
}
class TagSelector extends StatefulWidget {
 final List<Map<String, dynamic>> tags;
 final Set<String> selectedTags;
```

```
 const TagSelector({
   super.key,
   required this.tags,
   required this.selectedTags,
 });
@override
 State<TagSelector> createState() => _TagSelectorState();
}
class _TagSelectorState extends State<TagSelector> {
 final TextEditingController _searchController =
TextEditingController();
 String _searchText = "";
@override
 void initState() {
   super.initState();
   _searchController.addListener(() {
     setState(() {
       _searchText =
_searchController.text.trim().toLowerCase();
     });
   });}
@override
 Widget build(BuildContext context) {
   final selectedTagList = widget.tags.where(
     (tag) =>
widget.selectedTags.contains(tag['id'].toString()),
   );
   final filteredTagList = _searchText.isEmpty
       ? []
       : widget.tags.where((tag) {
           final tagId = tag['id'].toString();
           final tagName =
tag['name'].toString().toLowerCase();
           return tagName.contains(_searchText) &&
               !widget.selectedTags.contains(tagId);
         }).toList();
return Column(
     crossAxisAlignment: CrossAxisAlignment.start,
     children: [
       TextField(
         controller: _searchController,
         decoration: InputDecoration(
           labelText: "Пошук тегів",
           prefixIcon: Icon(Icons.search),
           border: OutlineInputBorder(),
         ),),
       const SizedBox(height: 12),
       if (widget.selectedTags.isNotEmpty) ...[
         Text("Вибрані теги:"),
         const SizedBox(height: 6),
         Wrap(
```

```
            spacing: 8.0,
            children: selectedTagList.map((tag) {
              final tagId = tag['id'].toString();
              final tagName = tag['name'];
              final tagDescription = tag['description'] ??
'Немає опису';
return Tooltip(
                message: tagDescription,
                waitDuration: Duration(milliseconds: 400),
                child: FilterChip(
                  label: Text(tagName),
                  selected: true,
                  onSelected: (isSelected) {
                    setState(() {
                      widget.selectedTags.remove(tagId);
                    });
                  },
                ),
              );
            }).toList(),
          ),
          const SizedBox(height: 16),
        ],
        if (_searchText.isNotEmpty) ...[
          Text("Результати пошуку:"),
          const SizedBox(height: 6),
          Wrap(
            spacing: 8.0,
            children: filteredTagList.map((tag) {
              final tagId = tag['id'].toString();
              final tagName = tag['name'];
              final tagDescription = tag['description'] ??
'Немає опису';
              return Tooltip(
                message: tagDescription,
                waitDuration: Duration(milliseconds: 400),
                child: FilterChip(
                  label: Text(tagName),
                  selected:
widget.selectedTags.contains(tagId),
                  onSelected: (isSelected) {
                    setState(() {
                      if (isSelected) {
                        widget.selectedTags.add(tagId);
                      } else {
                        widget.selectedTags.remove(tagId);
                      }
                    });},
                ),
              );
            }).toList(),
          ),]],
    ); }}
```

					КР.КН 25.588.07.000 ПЗ	Арк.
						114
Змн.	Арк.	№ докум.	Підпис	Дата		

Додаток Н

хом скрін

```
import 'dart:convert';
import 'package:flutter/material.dart';
import 'package:http/http.dart' as http;
import 'package:toread/models/recommendation.dart';
import 'create_work_screen.dart';
class NotificationsScreen extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
   return Scaffold(
     appBar: AppBar(title: Text("Сповіщення")),
     body: Center(child: Text("Тут будуть сповіщення")),);
 }}
class HistoryScreen extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
   return Scaffold(
     appBar: AppBar(title: Text("Історія")),
     body: Center(child: Text("Тут буде історія")),);
 }}
class HomeScreen extends StatefulWidget {
 final String userId;
 HomeScreen({required this.userId});
 @override
 _HomeScreenState createState() => _HomeScreenState();
}
class _HomeScreenState extends State<HomeScreen> {
 late Future<List<Recommendation>> recommendations;
 late Future<List<Recommendation>> popularWorks;
 int _selectedIndex = 0;
 @override
 void initState() {
   super.initState();
   recommendations = fetchRecommendations(widget.userId);
   popularWorks = fetchPopularWorks();
 }
 Future<List<Recommendation>> fetchRecommendations(String
userId) async {
   final response = await http.get(
     Uri.parse('http://192.168.170.157:8000/recommendations/$
userId'),
   );
   if (response.statusCode == 200) {
     List<dynamic> data =
json.decode(utf8.decode(response.bodyBytes));
     return data.map((item) =>
Recommendation.fromJson(item)).toList();
   } else {
     throw Exception('Не вдалося завантажити рекомендації');
   }
```

```
 }
 Future<List<Recommendation>> fetchPopularWorks() async {
   final response = await http.get(
     Uri.parse('http://192.168.170.157:8000/popular'),
   );
   if (response.statusCode == 200) {
     List<dynamic> data =
json.decode(utf8.decode(response.bodyBytes));
     return data.map((item) =>
Recommendation.fromJson(item)).toList();
   } else {
     throw Exception('Не вдалося завантажити популярні
твори');
   }}
 void _onItemTapped(int index) {
   setState(() {
     _selectedIndex = index;});}
 Widget _buildCard(Recommendation recommendation) {
   return Card(
     margin: EdgeInsets.symmetric(horizontal: 8.0, vertical:
4.0),
     child: ListTile(
       title: Text(recommendation.title),
       subtitle: Column(
         crossAxisAlignment: CrossAxisAlignment.start,
         children: [
           Text('Автор: ${recommendation.author}'),
           Text('Жанри: ${recommendation.genres.join(',
')}'),
           Text('Теги: ${recommendation.tags.join(', ')}'),
           Text(recommendation.description),
         ],),
     ),);
 }
 Widget _buildBody() {
   switch (_selectedIndex) {
     case 0:
       return SingleChildScrollView(
         child: Column(
           crossAxisAlignment: CrossAxisAlignment.start,
           children: [
             Padding(
               padding: const EdgeInsets.all(8.0),
               child: Text('Популярне',
                   style:
Theme.of(context).textTheme.titleLarge),
             ),
             SizedBox(
               height: 240,
               child: FutureBuilder<List<Recommendation>>(
                 future: popularWorks,
                 builder: (context, snapshot) {
```

```
                   if (snapshot.connectionState ==
ConnectionState.waiting) {
                     return Center(child:
CircularProgressIndicator());
                   } else if (snapshot.hasError) {
                     return Center(child: Text('Помилка:
${snapshot.error}'));
                   } else if (!snapshot.hasData ||
snapshot.data!.isEmpty) {
                     return Center(child: Text('Немає
популярних творів'));
                   } else {
                     return ListView.builder(
                       scrollDirection: Axis.horizontal,
                       itemCount: snapshot.data!.length,
                       itemBuilder: (context, index) {
                         final recommendation =
snapshot.data![index];
                         return Container(
                           width: 300,
                           margin:
EdgeInsets.symmetric(horizontal: 8.0),
                           child: Card(
                             elevation: 3,
                             child: Padding(
                               padding: const
EdgeInsets.all(12.0),
                               child: Column(
                                 crossAxisAlignment:
CrossAxisAlignment.start,
                                 children: [
                                   Text(recommendation.title,
                                       style: TextStyle(
                                           fontSize: 18,
                                           fontWeight:
FontWeight.bold)),
                                   SizedBox(height: 4),
                                   Text('Автор:
${recommendation.author}'),
                                   Text(
                                       'Жанри:
${recommendation.genres.join(', ')}'),
                                   Text(
                                       'Теги:
${recommendation.tags.join(', ')}'),
                                   SizedBox(height: 4),
                                   Expanded(
                                       child:
Text(recommendation.description,
                                           overflow:
TextOverflow.ellipsis,
                                           maxLines: 4)),
                                 ],
```

```
                              ),),),
                        );},);}},),),
             Padding(
               padding: const EdgeInsets.all(8.0),
               child: Text('Рекомендоване для вас',
                   style:
Theme.of(context).textTheme.titleLarge),
             ),
             FutureBuilder<List<Recommendation>>(
               future: recommendations,
               builder: (context, snapshot) {
                 if (snapshot.connectionState ==
ConnectionState.waiting) {
                   return Center(child:
CircularProgressIndicator());
                 } else if (snapshot.hasError) {
                   return Center(child: Text('Помилка:
${snapshot.error}'));
                 } else if (!snapshot.hasData ||
snapshot.data!.isEmpty) {
                   return Center(child: Text('Немає доступних
рекомендацій'));
                 } else {
                   return ListView.builder(
                     shrinkWrap: true,
                     physics: NeverScrollableScrollPhysics(),
                     itemCount: snapshot.data!.length,
                     itemBuilder: (context, index) {
                       final recommendation =
snapshot.data![index];
                       return _buildCard(recommendation);
                     },
                   );}},),],
         ),);
     case 1:
       return Center(child: Text("Ідеї"));
     case 2:
       return Center(child: Text("Пошук"));
     case 3:
       return CreateWorkScreen();
     case 4:
       return Center(child: Text("Акаунт"));
     default:
       return Center(child: Text("Невідома сторінка"));}}
 @override
 Widget build(BuildContext context) {
   return Scaffold(
     appBar: AppBar(
       title: Text(_getTitle()),
       actions: [
         IconButton(
           icon: Icon(Icons.history),
           onPressed: () {
```

```
              Navigator.push(
                context,
                MaterialPageRoute(builder: (context) =>
HistoryScreen()),);
            },),
          IconButton(
            icon: Icon(Icons.notifications),
            onPressed: () {
              Navigator.push(
                context,
                MaterialPageRoute(builder: (context) =>
NotificationsScreen()), );
            },),
        ],),
      body: _buildBody(),
      bottomNavigationBar: BottomNavigationBar(
        currentIndex: _selectedIndex,
        onTap: _onItemTapped,
        type: BottomNavigationBarType.fixed,
        items: const [
          BottomNavigationBarItem(
            icon: Icon(Icons.home),
            label: 'Головна', ),
          BottomNavigationBarItem(
            icon: Icon(Icons.lightbulb),
            label: 'Ідеї',),
          BottomNavigationBarItem(
            icon: Icon(Icons.search),
            label: 'Пошук',),
          BottomNavigationBarItem(
            icon: Icon(Icons.edit),
            label: 'Написати',),
          BottomNavigationBarItem(
            icon: Icon(Icons.person),
            label: 'Акаунт',
          ),],
      ), );}
  String _getTitle() {
    switch (_selectedIndex) {
      case 0:
        return "Головна сторінка";
      case 1:
        return "Ідеї";
      case 2:
        return "Пошук";
      case 3:
        return "Написати";
      case 4:
        return "Акаунт";
      default:
        return "";
    }
  }}
```

Додаток П

Програмний код сторінки входу

```
import 'package:flutter/material.dart';
import 'home_screen.dart';
import 'app_styles.dart';
import 'colors.dart';
import 'RegistrationScreen.dart';
import 'auth_service.dart';
class LoginScreen extends StatefulWidget {
 @override
 _LoginScreenState createState() => _LoginScreenState();
}
class _LoginScreenState extends State<LoginScreen> {
 final _formKey = GlobalKey<FormState>();
 final _emailController = TextEditingController();
 final _passwordController = TextEditingController();
 bool isLoading = false;
 String? error;

 void _login() async {
   if (_formKey.currentState!.validate()) {
     setState(() {
       isLoading = true;
       error = null;
     });
     try {
       final userId = await AuthService().login(
         _emailController.text,
         _passwordController.text,
       );

       if (userId != null) {
         Navigator.pushReplacement(
           context,
           MaterialPageRoute(
             builder: (context) => HomeScreen(userId:
userId),
           ),);
       } else {
         setState(() {
           error = 'Невірний логін або пароль';
           isLoading = false;
         });
       }
     } catch (e) {
       setState(() {
         error = 'Помилка: $e';
         isLoading = false;
       });
```

```
        }
      }
    }
    @override
    Widget build(BuildContext context) {
      return Scaffold(
        appBar: AppBar(
          title: Text('Вхід'),
          backgroundColor: AppColors.anotherbrown,
        ),
        body: Padding(
          padding: const EdgeInsets.all(20.0),
          child: Form(
            key: _formKey,
            child: Column(
              mainAxisAlignment: MainAxisAlignment.center,
              children: <Widget>[
                TextFormField(
                  controller: _emailController,
                  decoration:
  AppStyles.inputDecoration.copyWith(
                    labelText: 'Електронна пошта',
                    prefixIcon: Icon(Icons.email),
                  ),
                  keyboardType: TextInputType.emailAddress,
                  validator: (value) {
                    if (value == null || value.isEmpty) {
                      return 'Будь ласка, введіть електронну
  пошту'; }
                    return null;
                  },),
                SizedBox(height: 20.0),
                TextFormField(
                  controller: _passwordController,
                  obscureText: true,
                  decoration:
  AppStyles.inputDecoration.copyWith(
                    labelText: 'Пароль',
                    prefixIcon: Icon(Icons.lock),),
                  validator: (value) {
                    if (value == null || value.isEmpty) {
                      return 'Будь ласка, введіть пароль';}
                    return null;
                  },),
                SizedBox(height: 30.0),
                isLoading
                    ? CircularProgressIndicator()
                    : ElevatedButton(
                        onPressed: _login,
                        style: AppStyles.elevatedButtonStyle,
                        child: Text('Увійти'),),
                SizedBox(height: 20.0),
                error != null
```

```
                 ? Text(error!, style: TextStyle(color:
Colors.red))
                 : SizedBox(),
             SizedBox(height: 20.0),
             TextButton(
               onPressed: () {
                 showDialog(
                   context: context,
                   builder: (context) {
                     return AlertDialog(
                       title: Text('Забули пароль?'),
                       content: Text(
                           'Введіть вашу електронну пошту для
відновлення паролю.'),
                       actions: <Widget>[
                         TextButton(
                           onPressed: () {
                             Navigator.of(context).pop();
                           },
                           child: Text('Закрити'),
                         ),
                         TextButton(
                           onPressed: () {
                             ScaffoldMessenger.of(context).sh
owSnackBar(
                               SnackBar(
                                   content: Text(
                                       'Інструкції надіслано
на вашу електронну пошту')),);
                             Navigator.of(context).pop();
                           },
                           child: Text('Відновити'),
                         ),],
                     );},
                 );},
               child: Text('Забули пароль?'),),
             SizedBox(height: 20.0),
             TextButton(
               onPressed: () {
                 Navigator.push(
                   context,
                   MaterialPageRoute(
                       builder: (context) =>
RegistrationScreen()),
                 );},
               child: Text('Немає акаунту? Зареєструватися'),
             ),],
         ),),
     ), );
 }}
```

Додаток Р

Лістинг програмного коду екрана реєстрації користувача

```
import 'package:flutter/material.dart';
import 'app_styles.dart';
import 'colors.dart';
import 'register_form.dart';
import 'package:http/http.dart' as http;
import 'dart:convert';
class RegistrationScreen extends StatefulWidget {
 @override
 _RegistrationScreenState createState() =>
_RegistrationScreenState();
}
class _RegistrationScreenState extends
State<RegistrationScreen> {
 final _formKey = GlobalKey<FormState>();
 final _emailController = TextEditingController();
 final _passwordController = TextEditingController();
 final _confirmPasswordController = TextEditingController();
 final _usernameController = TextEditingController();
 final _firstNameController = TextEditingController();
 final _lastNameController = TextEditingController();
 final _dateController = TextEditingController();
 DateTime? _selectedDate;
 Future<void> _selectDate(BuildContext context) async {
   final DateTime? picked = await showDatePicker(
     context: context,
     initialDate: DateTime.now(),
     firstDate: DateTime(1940),
     lastDate: DateTime.now(),
   );
   if (picked != null && picked != _selectedDate) {
     setState(() {
       _selectedDate = picked;
       _dateController.text =
           '${picked.year}-
${picked.month.toString().padLeft(2, '0')}-
${picked.day.toString().padLeft(2, '0')}';
     });
   }}
 void _register() async {
   if (_formKey.currentState!.validate()) {
     final Map<String, dynamic> data = {
       "username": _usernameController.text,
       "name": _firstNameController.text,
       "last_name": _lastNameController.text,
       "email": _emailController.text,
       "password": _passwordController.text,
       "birth": _dateController.text,
     };
```

```
      try {
        final response = await http.post(
          Uri.parse(
              'http://192.168.170.157:8000/register'),
          headers: {
            'Content-Type': 'application/json',
          },
          body: jsonEncode(data),
        );
        if (response.statusCode == 200 || response.statusCode
== 201) {
          ScaffoldMessenger.of(context).showSnackBar(
            SnackBar(content: Text('Реєстрація успішна!')),
          );
          Navigator.pop(context);
        } else {
          final Map<String, dynamic> responseData =
              jsonDecode(utf8.decode(response.bodyBytes));
          final errorMessage = responseData['detail'] ??
'Помилка реєстрації';
          ScaffoldMessenger.of(context).showSnackBar(
            SnackBar(content: Text(errorMessage)),
          );}
      } catch (e) {
        ScaffoldMessenger.of(context).showSnackBar(
          SnackBar(content: Text('Помилка підключення: $e')),
        );}
    }}
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Реєстрація'),
        backgroundColor: AppColors.anotherbrown,
      ),
      body: SingleChildScrollView(
        child: Padding(
          padding: const EdgeInsets.all(20.0),
          child: Form(
            key: _formKey,
            child: Column(
              mainAxisAlignment: MainAxisAlignment.start,
              children: <Widget>[
                TextFormField(
                  controller: _usernameController,
                  decoration:
AppStyles.inputDecoration.copyWith(
                    labelText: 'Ім\'я користувача',
                    prefixIcon: Icon(Icons.person),
                  ),
                  validator: (value) {
                    if (value == null || value.isEmpty) {
```

```
                        return 'Будь ласка, введіть ім\'я
користувача';
                      }
                      return null;
                    },),
                  SizedBox(height: 20.0),
                  TextFormField(
                    controller: _firstNameController,
                    decoration:
AppStyles.inputDecoration.copyWith(
                      labelText: 'Ім\'я',
                      prefixIcon: Icon(Icons.person),
                    ),
                    validator: (value) {
                      if (value == null || value.isEmpty) {
                        return 'Будь ласка, введіть ім\'я';
                      }
                      return null;
                    },),
                  SizedBox(height: 20.0),
                  TextFormField(
                    controller: _lastNameController,
                    decoration:
AppStyles.inputDecoration.copyWith(
                      labelText: 'Прізвище',
                      prefixIcon: Icon(Icons.person),),
                    validator: (value) {
                      if (value == null || value.isEmpty) {
                        return 'Будь ласка, введіть прізвище';}
                      return null;
                    },),
                  SizedBox(height: 20.0),
                  TextFormField(
                    controller: _emailController,
                    decoration:
AppStyles.inputDecoration.copyWith(
                      labelText: 'Електронна пошта',
                      prefixIcon: Icon(Icons.email),),
                    keyboardType: TextInputType.emailAddress,
                    validator: (value) {
                      if (value == null || value.isEmpty) {
                        return 'Будь ласка, введіть електронну
пошту';}
                      return null;
                    },
                  ),
                  SizedBox(height: 20.0),
                  TextFormField(
                    controller: _passwordController,
                    obscureText: true,
                    decoration:
AppStyles.inputDecoration.copyWith(
                      labelText: 'Пароль',
```

```
                  prefixIcon: Icon(Icons.lock),
                ),
                validator: (value) {
                  String? error =
FormValidation.validatePassword(value!);
                  return error;
                },),
              SizedBox(height: 20.0),
              TextFormField(
                controller: _confirmPasswordController,
                obscureText: true,
                decoration:
AppStyles.inputDecoration.copyWith(
                  labelText: 'Підтвердження паролю',
                  prefixIcon: Icon(Icons.lock),
                ),
                validator: (value) {
                  if (value == null || value.isEmpty) {
                    return 'Будь ласка, підтвердіть пароль';
                  }
                  if (value != _passwordController.text) {
                    return 'Паролі не співпадають'; }
                  return null;},),
              SizedBox(height: 20.0),
              TextFormField(
                controller: _dateController,
                decoration:
AppStyles.inputDecoration.copyWith(
                  labelText: 'Дата народження',
                  prefixIcon: Icon(Icons.calendar_today),
                  hintText: 'Виберіть дату',),
                readOnly: true,
                onTap: () => _selectDate(context),
                validator: (value) {
                  String? error =
FormValidation.validateAge(_selectedDate);
                  return error;
                },),
              SizedBox(height: 30.0),
              ElevatedButton(
                onPressed: _register,
                style: AppStyles.elevatedButtonStyle,
                child: Text('Зареєструватися'),),
              SizedBox(height: 20.0),
              TextButton(
                onPressed: () {
                  Navigator.pop(context);},
                child: Text('Уже є акаунт? Увійти'),),
            ],),
        ),),
    ),);
 }}
```

Додаток С

Лістинг програмного коду вітального екрану

```
import 'package:flutter/material.dart';
import 'package:toread/items.dart';
import 'package:flutter/services.dart';
import 'package:flutter/gestures.dart';
import 'login_screen.dart';
import 'package:auto_size_text/auto_size_text.dart';

class WelcomeScreen extends StatefulWidget {
 @override
 _WelcomeScreenState createState() =>
_WelcomeScreenState();
}

class _WelcomeScreenState extends State<WelcomeScreen> {
 List<Widget> slides = items
     .map((item) => Container(
           padding: EdgeInsets.symmetric(horizontal: 18.0),
           child: Column(
             children: <Widget>[
               Flexible(
                 flex: 1,
                 fit: FlexFit.tight,
                 child: Image.asset(
                   item['image'],
                   fit: BoxFit.fitWidth,
                   width: 220.0,
                   alignment: Alignment.bottomCenter,
                 ),
               ),
               Flexible(
                 flex: 1,
                 fit: FlexFit.tight,
                 child: Container(
                   padding:
EdgeInsets.symmetric(horizontal: 30.0),
                   child: Column(
                     children: <Widget>[
                       AutoSizeText(
                         item['header'],
                         style: TextStyle(
                             fontWeight: FontWeight.w300,
                             color: Color(0XFF633A14),
                             height: 2.0),
                         maxLines: 1,
                         minFontSize: 24,
                       ),
                       AutoSizeText(
```

```
                              item['description'],
                              style: TextStyle(
                                  color: Colors.grey,
                                  letterSpacing: 1.2,
                                  fontSize: 16.0,
                                  height: 1.3),
                              textAlign: TextAlign.center,
                              maxLines: 3,
                              minFontSize: 16,
                            ),
                          ],
                        ),
                      ),
                    )
                  ],
                ),
              ))
          .toList();

  List<Widget> indicator() => List<Widget>.generate(
        slides.length,
        (index) => Container(
          margin: EdgeInsets.symmetric(horizontal: 3.0),
          height: 10.0,
          width: 10.0,
          decoration: BoxDecoration(
            color: currentPage.round() == index
                ? Color(0XFF955627)
                : Color(0XFF955627).withOpacity(0.2),
            borderRadius: BorderRadius.circular(10.0),
          ),
        ),
      );

  double currentPage = 0.0;
  final _pageViewController = PageController();

  @override
  void initState() {
    super.initState();
    _pageViewController.addListener(() {
      setState(() {
        currentPage = _pageViewController.page ?? 0.0;
      });
    });
  }
  void _onKey(RawKeyEvent event) {
    if (event is RawKeyDownEvent) {
      if (event.logicalKey == LogicalKeyboardKey.arrowRight
||
          event.logicalKey == LogicalKeyboardKey.enter ||
          event.logicalKey == LogicalKeyboardKey.space) {
        if (currentPage < slides.length - 1) {
```

```
          _pageViewController.nextPage(
            duration: Duration(milliseconds: 300),
            curve: Curves.ease,
          );
        }
      } else if (event.logicalKey ==
LogicalKeyboardKey.arrowLeft) {
        if (currentPage > 0) {
          _pageViewController.previousPage(
            duration: Duration(milliseconds: 300),
            curve: Curves.ease,
          );
        }
      }
    }
  }
  void _onScroll(PointerSignalEvent event) {
    if (event is PointerScrollEvent) {
      if (event.scrollDelta.dy > 0) {
        if (currentPage < slides.length - 1) {
          _pageViewController.nextPage(
            duration: Duration(milliseconds: 300),
            curve: Curves.ease,
          );
        }
      } else if (event.scrollDelta.dy < 0) {
        if (currentPage > 0) {
          _pageViewController.previousPage(
            duration: Duration(milliseconds: 300),
            curve: Curves.ease,
          );
        }}
    }
  }
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: RawKeyboardListener(
        focusNode: FocusNode(),
        onKey: _onKey,
        child: Listener(
          onPointerSignal: _onScroll,
          child: Stack(
            children: <Widget>[
              PageView.builder(
                controller: _pageViewController,
                itemCount: slides.length,
                itemBuilder: (BuildContext context, int
index) {
                  return slides[index];
                },
              ),
              Align(
```

```
                alignment: Alignment.bottomCenter,
                child: Container(
                  margin: EdgeInsets.only(top: 70.0),
                  padding: EdgeInsets.symmetric(vertical:
40.0),
                  child: Row(
                    mainAxisAlignment:
MainAxisAlignment.center,
                    children: indicator(),
                  ),),
              ),
              if (currentPage.round() ==
                  slides.length - 1)
                Align(
                  alignment: Alignment.bottomCenter,
                  child: Padding(
                    padding: EdgeInsets.only(bottom: 30.0),
                    child: ElevatedButton(
                      onPressed: () {
                        Navigator.pushReplacement(
                          context,
                          MaterialPageRoute(
                            builder: (context) =>
LoginScreen(),
                          ),
                        );},
                      child: AutoSizeText(
                        'Увійти',
                        style: TextStyle(fontSize: 18),
                        maxLines: 1,
                        minFontSize: 12,
                      ),
                      style: ElevatedButton.styleFrom(
                        backgroundColor: Color(0XFF955627),
                        foregroundColor: Colors.white,
                        padding: EdgeInsets.symmetric(
                            horizontal: 50.0, vertical:
15.0),),
                    ),),
                ),],
            ),),
        ),);
  }}
```