НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І
ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ
Завідувач кафедри
комп'ютерних наук

_____/Голуб Б.Л., доц., к.т.н. /
підпис ПІБ, вчене звання і ступінь

«_2_»____червня_____2025 р

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА
на тему:

«Програмне забезпечення системи автоматизованого супроводження читання книг»

Спеціальність F2 «Інженерія програмного забезпечення»

ОП Інженерія програмного забезпечення

Гарант освітньої програми
_____к.т.н., доцент_____/ Вайганг Г.О./
Науковий ступень та вчене звання підпис ПІБ

Керівник бакалаврської кваліфікаційної роботи :_____/Бушма О.В./
підпис (ПІБ)
Виконав:_____/Ільєнко С.О./
підпис (ПІБ студента)

КИЇВ-2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ЗАТВЕРДЖУЮ
Завідувач кафедри
Комп'ютерних наук

_____ / Голуб Б.Л., доцент, к.т.н /
підпис
" 16 " грудня 2024 р.

З А В Д А Н Н Я

на виконання бакалаврської кваліфікаційної роботи студенту
Ільєнку Сергію Олександровичу

Спеціальність F2 «Інженерія програмного забезпечення»

ОПП Інженерія програмного забезпечення

1. Тема роботи: Програмне забезпечення системи автоматизованого супроводження читання книг Затверджена наказом ректора НУБіП України № 2249 "С" від 16.12.2024

2. Термін подання завершеної роботи на кафедру 2025 . 05 . 25
рік, місяць, число

3. Вихідні дані до роботи: опис програмного забезпечення

4. Перелік питань що розглядаються:

1. Супровід у читанні книги як об'єкт дослідження.
2. Інформаційне забезпечення системи.
3. Проєктування та розробка програмного забезпечення.
4. Впровадження та експлуатація системи.

Керівник бакалаврської кваліфікаційної роботи_____ / Бушма О.В. /
підпис ініціали та прізвище

Завдання прийняв до виконання _____ / Ільєнко С.О. /
підпис ініціали та прізвище

Дата отримання завдання 2024 . 12 . 16
рік, місяць, число

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД – база даних.

АС – автоматизована система.

UML – unified modeling language.

СУБД – система управління бази даних.

CSV – comma-separated values.

ІЗ – інформаційне забезпечення.

SQL – structured query language.

ОС – операційна система.

НФ – нормальна форма.

REST API – representational state transfer application programming interface.

IDE – integrated development environment.

ISBN – International Standard Book Number.

ВСТУП

Актуальність. Читання є невід’ємною часткою людини для різних сферах діяльності як навчання, дозвілля, саморозвиток, яке завжди супроводжувалося індивідуальною організацією цього процесу. З розвитком спосіб читання змінився: від звичайних паперових книг до більш екологічних і зручних варіантів як електронні книги, аудіокниги. Люди як і інформаційні технології змінювалися, тож у зв’язку з цим постала потреба у програмних рішеннях, котрі можуть допомогти читачам не тільки читати книги, а й ефективно організовувати супроводжене читання та ділитися своїми враженнями про прочитане.

У час широкого впровадження інформаційних технологій у різних сферах діяльності постала потреба в оптимізації процесів, пов’язаних із читанням, а саме супроводу, що включає в себе управління нотатками, написання рецензій, що впливають на вибір книг для читання, відстеження читацької активності: час і кількість прочитаних сторінок. Звичний підхід супроводу до взаємодії з книгами включав вжиток різноманітних фізичних матеріалів – зошитів для запису і зберігання інформації у вигляді нотаток, закладок для орієнтування в прочитаному, канцелярського приладдя для ведення обліку сторінок й аналізу прочитаного на основі чого власноруч формувалася статистика. Однак, такий підхід є не лише незручним, а й вимагає додаткових зусиль і фінансових витрат на фізичні матеріали, що насамперед сприяють надмірному споживанню ресурсів та негативному впливу на екологію через вирубку дерев для виробництва паперу, закладок, канцелярського приладдя тощо. Також окрім впливу на навколишнє середовище, цей підхід обмежує читачеві можливість швидкого застосування та аналізу великого об’єму інформації, адже відсутність цифрових інструментів перешкоджає організації інформації з прочитаного, а також ставить під сумнів їхню коректність.

Мета розробки. Основна мета розробки програмного забезпечення системи автоматизованого супроводження читання книг полягає у створенні системи, яка оптимізує читачам процес супроводу і забезпечує не тільки комфортне

здобування досвіду в читанні різноманітних книжок, а й вільний доступ у використанні цифрових інструментів для цих процесів.

Методи та технології. Під час розробки системи автоматизованого супроводу читанні книг використовувалися сучасні веб-технології. Для реалізації клієнтської сторони використано мову JavaScript і бібліотеку React, що дозволить створювати динамічні, чуйні інтерфейси для користувачів, чиї вимоги з роками стають вимогливішими. Для стилізації використано CSS із сучасними підходами до створення стилів, що забезпечать не тільки гарний дизайн, а й адаптивність до більшості розмірів дисплеїв. Для реалізації серверної сторони вибір впав на найпопулярніший дует з Node.js та Express, оскільки разом вони забезпечать обробку запитів від користувача, логіку серверну, взаємодію з базою даних і зовнішніми сервісами. Стосовно бази даних обрано реляційну СУБД SQL Server, яка забезпечує реляційну структуру БД. Для проведення тестування використовується програмний застосунок Postman.

Апробація програмного продукту. Опубліковано тези "Організований супровід у читанні книжок в умовах розвитку інформаційних технологій" на Міжнародна науково-практична конференція студентів, аспірантів "Актуальні питання розвитку науки та техніки в умовах глобалізації" (м. Боярка, 14.05.25).

Структура записки. Дипломного проєкту складається з таких частин: вступ, 4 розділи та висновки. Перший розділ зосереджений на аналізі предметної області, який описує проблематику та представляє моделювання її за допомогою UML, і формулюванні постановки завдань. Другий розділ описує логічну модель даних та обґрунтовує вибір СУБД для подальшої БД. Третій розділ вже представляє розробку програмного забезпечення, що включає вибору інструментарію та середовища розробки. Четвертий, останній розділ, присвячений тестуванню розробленої системи, її експлуатації та опису вимог апаратних, програмних вимог.

1 СУПРОВІД У ЧИТАННІ КНИГИ ЯК ОБ'ЄКТ ДОСЛІДЖЕННЯ

1.1 Опис предметної області

Один із найважливіших і найперших етапів дослідження предметної області є його аналіз. Такий процес протоптує стежину у підготовці до розробки системи. Як результат, це допомагає глибше зрозуміти досліджувану тему та виявити проблематику, важливі частини її та елементи для майбутнього масштабування. Правильне аналізування предметної області дає повну картину про уявлення предметної області для подальшої розробки.

Дана предметна область зосереджена на читачах, котрі взаємодіють із книгами під час супроводженого читання. Насамперед, процес супроводу в читанні розпочинається з того, що читач знаходить собі цікаву книжку, яку розміщує на полиці на власному стелажі. Далі, якщо читач має бажання вести супровід під час читання певної книги, він купляє необхідне канцелярське приладдя та зошит, без яких організований супровід у читанні книжок не провести. Здебільшого, такий процес допомагає людям покращити свій читацький досвід та зробити унікальним.

Супровід є важливим процесом під час читання книжок для читачів. Він охоплює багато дій, одна з яких створення нотаток до важливих моментів у книзі, мета яких зорієнтувати читача у прочитаному матеріалі. У подібних записах зберігаються не тільки моменти з книги, а й враження, думки, висновки, коментарі, цитати тощо. Важливість нотаток у тому, щоб краще усвідомлювати прочитане. Однак, такий процес має свою проблему, яка полягає в тому, що їхня кількість поступово зростає відносно цікавості людини до книг, а зберігати стає все важче, через що легко втратити чи сплутати з іншими нотатками.

Окрім нотаток, супровід охоплює дії як відстеження часу за допомогою таймера та підрахунку кількості прочитаних сторінок. Як результат зібрана інформація застосовується у формуванні статистики, що включає різноманітні

показники: середня швидкість читання, кількість нотаток та днів за яку книгу прочитали тощо. Важливість такої інформації для читача є обов'язковою, адже знати з якої сторінки почали і закінчили читати, скільки часу витрачено для цього допомагає їм ретельно розподіляти час, набувати читацький досвід, а також аналізувати свою активність внаслідок самостійного обрахунку статистики.

Насамкінець, після прочитання книги, читачу важливо поділитися своїми враженнями та емоціями про книгу, сформувавши рецензію. Рецензія – це читацька оцінка з детально написаним відгуком, мета якої вплинути на вибір книжки читачем, який витрачатиме свій час та сили на неї. Зазвичай рецензії містять загальне враження про книгу, однак деякі рецензенти воліють викласти набагато більше інформації про неї, наприклад, інформацію про характер персонажів, сюжет, всесвіт, цікаві факти, особливості написання автора, а також і недоліки твору. Сама по собі рецензія є особистим вкладом у свій читацький досвід, тому її оприлюднення на загал залишається за бажанням читача. Публікація зазвичай відбувається по різному, але гарний прикладом є літературні клуби, які цінують внесок своїх учасників і виносять на обговорення або публікують у журналі їхні рецензії, однак охопити велику кількість людей дуже важко.

1.2 Огляд інформаційних рішень

На ринку вже існують рішення, які схожі із запропонованою системою автоматизованого супроводження читання книг. Ознайомлення з ними дозволить сильніше поглибитися у дослідженні предметної області завдяки огляду можливостей, плюсів, недоліків аналогічних рішень, які слід уникнути, а також сформувати уявлення про очікування авдиторії. Нижче розглянуто декілька готових рішень, зокрема Goodreads, Basmo і Hardcover, що вже використовуються в предметній області.

1. Basmo.

Це мобільний застосунок для відстеження читання книг, який завжди під рукою для швидкої організації читання. Він реалізує для користувача

можливості встановлення цілі, відстеження прогресу в читанні, записувати свої враження від читання та організувати книжкові полиці в колекції. На рис. 1 представлено скріншот інтерфейсу застосунку Basmo.

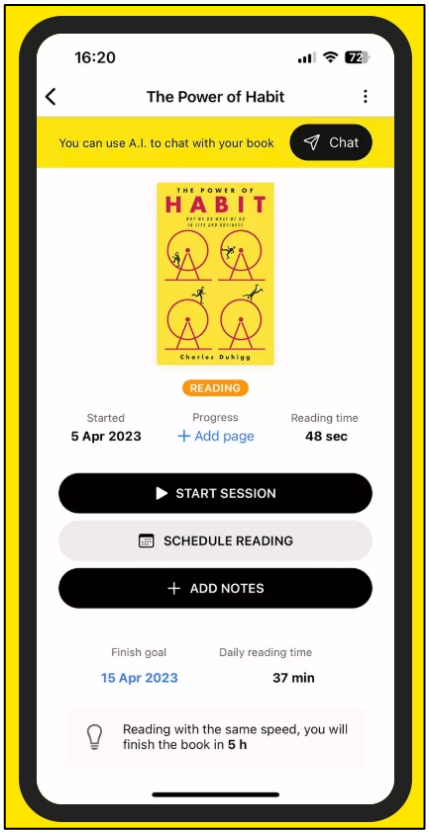


Рис. 1 Інтерфейс застосунку Basmo

Недоліки Basmo.

- Відсутня можливість рецензування книжок і перегляд рецензій іншими користувачами застосунку.
- Статистика формується загальна, через що дізнатися корисні дані за певною книжкою неможливо.
- Обмежена функціональність через внутрішню підписку, де для безкоштовної версії передбачено лишень 2 сеанси читання на день, недоступні сканування тексту та інші ключові функції без підписки.
- Обмежена доступність, існує лише як застосунок.

2. Goodreads.

Один із найвідоміших і найстаріших сайтів для читачів та рекомендацій книг. Його можна вважати осередком для любителів книг, у якому читачі діляться своїми враженнями про книги, відстежують книги, можуть підписуватися на інших зареєстрованих читачів та стежити за їхніми вподобаннями та написаними рецензіями. Сайт володіє великою БД книг, а також користується підтримкою з боку компанії Amazon. Goodreads є одним із перших, хто представив уявлення про те, якими можуть бути книжкові онлайн-платформи. На рис. 2 представлено скріншот інтерфейсу сайту Goodreads.

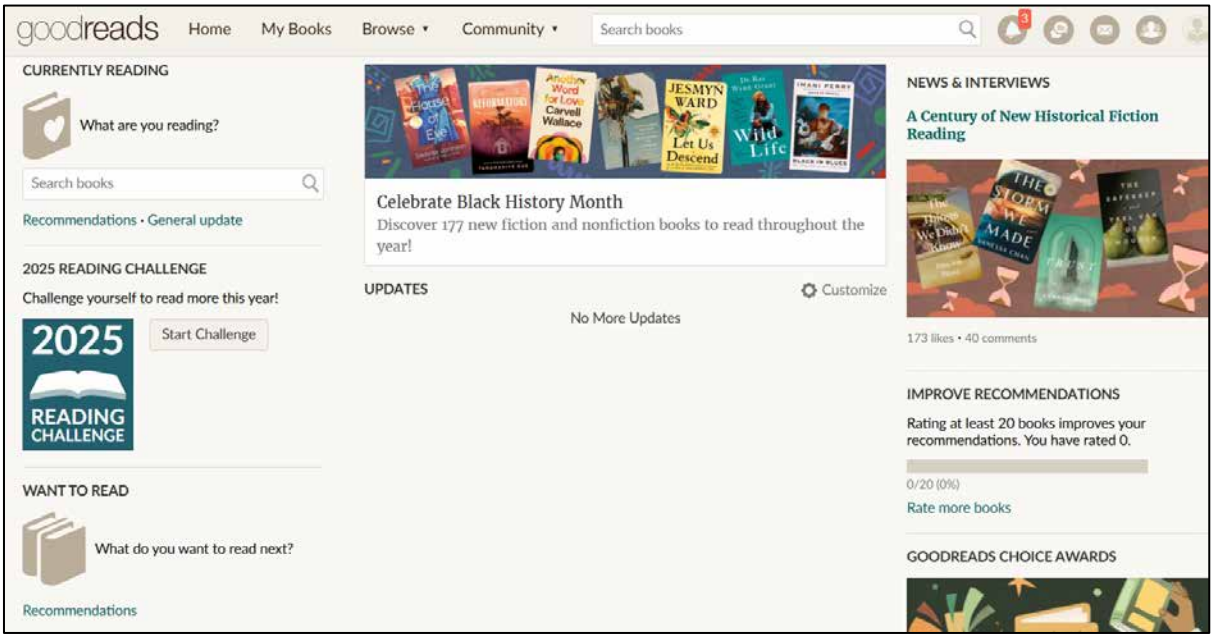


Рис. 2 Інтерфейс сайту Goodreads

Недоліки Goodreads.

- Застарілий дизайн інтерфейсу.
- Нема функціоналу відстеження швидкості, часу читання або кількості прочитаних сторінок.
- Функція формування статистики як загальна, так і до книги відсутня, через що доводиться застосовувати сторонні сайти, які надають таку можливість читачам через сформований файл формату CSV.

3. Hardcover.

Це новий сайт, що має власний мобільний застосунок, який позиціонує себе як сучасна альтернатива Goodreads для книголюбів. Сайт дає можливість вести читацьку активність та соціальним комунікацію між читачами, що дозволяє ділитися враженнями та рецензіями на книжки. На рис. 3 представлено скріншот інтерфейсу сайту Hardcover.

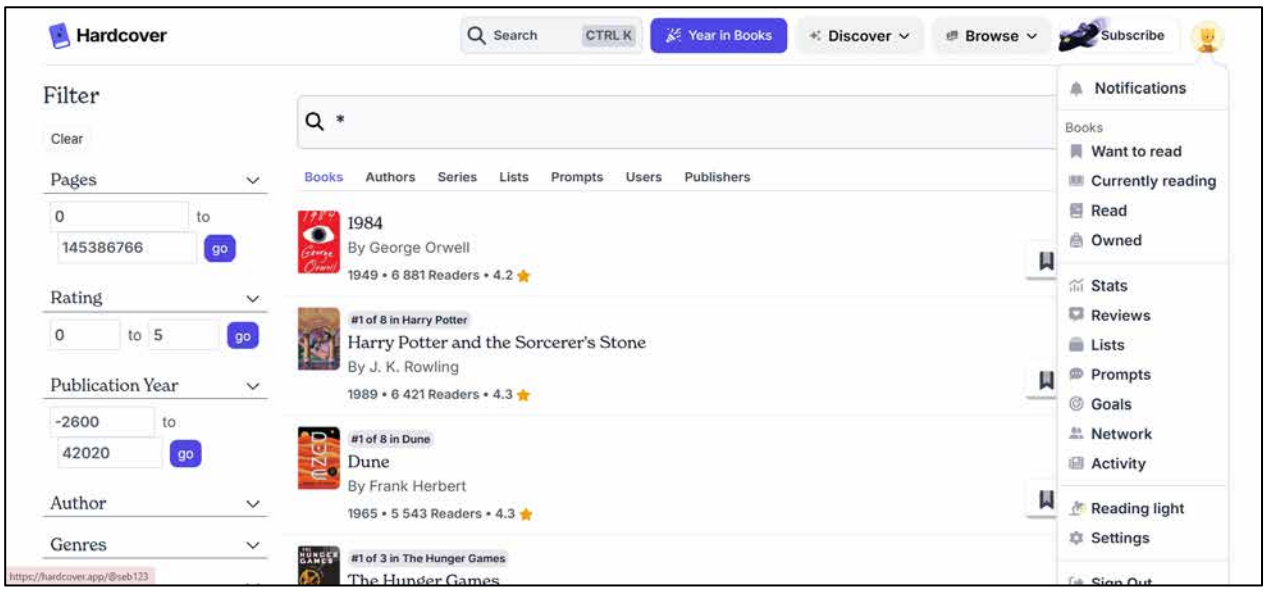


Рис. 3 Інтерфейс сайту Hardcover

Недоліки Hardcover.

- Невелика за розміром БД книг, через що не завжди можна знайти потрібну книгу, а також пошук видає неточні результати.
- Недружелюбний інтерфейс.
- Відсутні функції відстеження часу читання.
- Оновлення сайту відбувається не часто, через що деякі функції залишаються недопрацьованими, як наприклад створення статистики. Недолік пов’язаний з тим, що над Hardcover працює команда лише з трьох осіб, які фінансують проєкт самостійно.

1.3 Постановка завдання

Найголовніша ціль розроблення запропонованої системи – надати читачу зручний та інтуїтивно простий у використанні інструмент, який допоможе організувати супроводжене читання книги.

Система повинна взяти на себе завдання забезпечити спеціальний функціонал для своїх користувачів.

Виділено основних користувачів системи.

1) читач – центральна особа на якій ґрунтується розроблювальна система, що має ряд своїх операцій для організованого супроводу читання;

2) рецензент – особа нічим не відрізняється від читача, окрім потребою у написанні відгуків та оцінки до книги, ціль якої спрямована на формування спільноти й обмін думками;

3) модератор – найголовніша дійова особа, яка відповідальна за дотримання порядку в системі: розміщення рецензії, а також добросовісної поведінки своїх користувачів.

У реалізації системи також важливо визначити вимоги. Їх можна поділити за типом: функціональні та нефункціональні. Функціональні формулюють як система повинна поводитися, тобто визначає, що має робити система, щоб задовольнити потреби або очікування користувача. Нефункціональні задають атрибути якості системи, тобто наскільки добре має працювати система. Розуміння обох типів вимог гарантує успішну реалізацію програмного забезпечення.

Функціональні вимоги для читача і рецензента.

- Реєстрація та вхід: передбачається створення облікового запису, де у разі успішного входу система надає відповідний функціонал залежно від ролі користувача. Вимога дає гарантію, що всі дані зареєстрованого користувача будуть збережені, а персоналізований досвід користування системою буде доступний лише після входу в обліковий запис. Для реєстрації достатньо ввести

пошту, ім’я та пароль, що відслідковується валідацією, а можливість входу забезпечується введенням правильного email та пароля від облікового запису.

- Пошук книжок: вимога є обов’язковою для пошуку книг за одним із параметрів: назва, ISBN, автор. Пошук книжок відбуватиметься за допомогою Google Books API.
- Створення каталогів: створення та організація каталогів для зберігання книг, що дозволяє користувачеві легко структурувати книжки за категоріями.
- Збереження книг: збереження знайденої книги у певний каталог для подальших взаємодій у супроводженому читанні.
- Відслідковування часу читання та кількості прочитаних сторінок: пропонується аби користувач мав можливість записувати власний витрачений час на читання та кількість прочитаних сторінок. Також це знадобиться у випадку, коли читач повністю прочитає книгу і йому стане доступна можливість сформувати статистику читання користувача за книжкою.
- Створення нотаток: створення особистих нотаток, які завжди можна видалити чи відредагувати.
- Написання та публікація рецензії: можливість написання рецензії через текстовий відгук та виставлення оцінки до книги. У результаті вона буде оприлюднена в системі.
- Формування статистики: створення статистики на основі даних активності користувача: часу читання в середньому, кількості нотаток, скільки часу знадобилося для повного прочитання.

Функціональні вимоги для модератора.

1) видалення читача із системи у випадку порушень правил поведінки;
2) загальний перегляд списку користувачів у системі;
3) зміна ролі читача системи на модератора;
4) видалення рецензій із системи у разі порушень правил цензури на публікацію.

Нефункціональні вимоги.

- Зручність використання: інтерфейс читачу має бути зрозумілим і простим у використанні, а також адаптивним під будь-який розмір екрана пристрою.

- Підтримка стабільності роботи: означає, що система повинна бути повністю доступною та функціональною незалежно від типу пристрою, з якого користувач здійснює вхід у неї. Це більше стосується не адаптивного дизайну інтерфейсу, а технічної стабільності роботи. Наприклад: правильне завантаження сторінок та її елементів, відсутність збоїв, збереження даних навіть у випадку тимчасового відключення інтернету чи випадкового закриття сторінки.

Запропонована система має широкий вибір сфер діяльності, у яких може бути корисною. Наприклад, вона може бути використана в наступних сферах:

1) під час навчання в університеті, коледжі чи школі, які працюють з художньою літературою і не тільки;

2) домашнє користування звичайного читача, який воліє дотримуватись більшої організованості під час читання, а також поліпшити рівень свого досвіду читання;

3) книжкових клубах, у яких ведуться колективні обговорення прочитаної літератури.

1.4 Моделювання предметної області

Створення моделі предметної області – це ще один важливий етап для майбутньої розробки системи. Він дає змогу побудувати уявлення як між собою об’єкти взаємодіють, спілкуються та як крок за кроком діють в межах області. Виконання цього етапу обумовлено метою простішого представлення взаємодії в предметній область, щоб будь-яка інша людина могла без детального пояснення зрозуміти як усе функціонує та хто бере участь у ній.

Щоб представити модель у рамках теми диплому, застосовуються різні групи діаграм, котрі побудовані через використання UML, зокрема діаграми

прецедентів, послідовності і активності, що є різновидом об’єктно-орієнтованого моделювання, яке забезпечує саме UML. Усі вони дадуть цілісне уявлення про учасників, структуру та поведінку майбутньої системи супроводу читання книжок.

UML (Уніфікована мова моделювання) – інструмент графічного відображення структури, логіки та взаємозв’язків елементів програмної системи. Більшість розробників ПЗ розуміють її та можуть застосовувати при моделюванні предметної області або самої системи [1]. У цього інструмента єдина та чітка ціль – показати, які дії виконує описаний актор, які об’єкти задіяні в ній, яка взаємодія між ними в області.

Діаграми діляться за типами.

- Структурні – це тип діаграми зосереджується саме на відображенні статичної будови системи. Кожна діаграма цього типу показують з яких елементів складається цілісна система, і які між ними взаємозв’язки.
- Поведінкові – увага в цьому типі діаграми зосереджена на тому, щоб відобразити те, що відбувається всередині системи, тобто як поводяться об’єкти у відповідь на певні події чи дії користувачів [1]. Акцент саме на обміні повідомленнями між об’єктами і що вони у відповідь зроблять.

1.4.1 Діаграма прецедентів. Діаграма прецедентів – це тип поведінкової діаграми UML, яку переважно застосовують для аналізу різних систем та предметної області. Діаграма дозволяє представити виділені ролі предметної області та показати як вони взаємодіють у предметній області.

Основні елементи для побудови.

- Актор – це виділена роль, що виконується сутностями, які взаємодіють у межах обраної області.
- Прецедент – це варіант використання певної дії одним із акторів.
- Зв’язок – спеціальне відношення між актором і прецедентом. Однак не всі прецеденти повинні бути пов’язані з акторами, але це залежить і від типу зв’язку: асоціація, розширення та включення.

Асоціація позначається звичайною лінією від актора до прецедента. Варто зазначити, що будь-який актор має бути пов’язаний хоча б з одним прецедентом. Розширення (Extend) позначається пунктирною лінією із стрілкою, що використовується як необов’язковий до виконання прецедент, але який може існувати окремо. Включення позначається також як і розширення, однак він показує як поведінка одного прецеденту включається в базовий прецедент, і він обов’язковий до виконання і окремо не може існувати.

Перед побудовою діаграми проаналізовано опис предметної області для визначення акторів і прецедентів, які працюють в ній.

Актори.

1. Читач – головний актор у предметній області, оскільки основні дії зосереджені за ним.

2. Рецензент – такий же актора як і Читач, але він виконує власні дії , які пов’язані із рецензуванням книг.

3. Працівник сховища – актор, який забезпечує доступ до великого сховища книг як для Читача, так і Рецензента;

4. Модератор – актор, суть якого пов’язана із взаємодією з рецензіями і людьми.

Прецеденти акторів.

1. Читач

- Знайти книгу – пошук книжки за назвою, автором чи ISBN номером.
- Переглянути інформацію – розширювальний (Extend) прецедент, у якому переглядаємо інформацію про книжку, але цей прецедент є опціональним, проте він може існувати окремо, тому з’єднаний із актором асоціацією.
- Ознайомитися із рецензіями – розширювальний (Extend) прецедент, де можна переглянути усі рецензії про знайдену книжку, але цей прецедент є опціональним, проте може існувати окремо, тому з’єднаний із актором асоціацією.
- Взяти книжку – взяти книжку для подальшої взаємодії із нею.

- Покласти в каталог – розширювальний (Extend) прецедент, де користувач може зберегти книжку до певного каталогу, яку до цього взяв, але цей прецедент опціональний, проте він може існувати окремо, тому з’єднаний із актором асоціацією.
- Створити каталог – створення власного каталогу для організації книжок.
- Сформувати нотатку – створення спеціального запису стосовно книжки.
- Відстежити час читання – можливість записувати час читання книги, що включає також і кількості прочитаних сторінок.
- Проаналізувати статистику – самоаналіз сформованої статистики по певній книзі.
- Сформувати статистику – формування статистичних даних, які пов’язані з читанням книжки: час читання, кількість сторінок за годину тощо.

2. Рецензент

- Публікувати рецензію – розмістити ретельно сформовану рецензію, яку в подальшому інші читачі можуть ознайомлюватися.
- Сформувати рецензію – написати сформований відгук на книжку, а також поставити оцінку.
- Також актор з’єднаний зв’язком узагальненням до актора Читач, через що він має такі ж прецеденти, що й він.

3. Працівник сховища

- Дати книжку за запитом – дати книжку, яку шукають читачі за запитом.
- Надати інформацію про книжку – дати повну інформацію читачу про бажану книжку.

4. Модератор

- Перевірити рецензії – перевірка надісланих рецензій на зміст, який спойлерить книжку або висловлює лайку.
- Переглянути зміст – перегляд повної інформації опублікованої рецензії.

- Видалити рецензію – прецедент, у якому відбувається видалення розглянутих рецензій, які виявилися некоректними.
- Переглянути опубліковані рецензії – переглядання всіх опублікованих рецензій.
- Переглянути список читачів – переглядання сформованого списку читачів.
- Заблокувати читача – прецедент, у якому відбувається блокування читача, який наприклад залишив некоректну рецензію.
- Переглянути дані про читача – можливість переглянути дані про читача.
- Сформувати список читачів – можливість створити список читачів.

Побудована діаграма прецедентів представлена в Додатку А.

1.4.2 Діаграма послідовності. Діаграма послідовності – це тип поведінкової діаграми UML, яка представляє як об’єкти її взаємодіють одне з одним повідомленнями за впорядкованим часом. Діаграма також показує послідовність виконання різних дій між об’єктами відповідно вже побудованій діаграмі прецедентів [2, 4].

Переваги використання.

- Показує взаємодію між об’єктами послідовно і по черзі відносно часу.
- Дає візуальне представлення, що полегшує розуміння логіки виконання дій об’єктів.

Основні елементи діаграми послідовності.

- Актори чи об’єкти, які взаємодіють між собою.
- Життєва лінія, яка відображає існування об’єкта протягом часу.
- Повідомлення або Виклик позначають виклик методу або обмін даними між акторами чи об’єктами. Вони поділяються на 2 типи: синхронні та асинхронні.
- Блоки активності означають періоди, коли об'єкт виконує певну дію на життєвій лінії.

- Блоки-обгортки (alt, loop, opt тощо) означають рамки для моделювання умовних, необов’язкових та повторюваних дій між об’єктами та їхіми викликами.

Побудована діаграма послідовності до предметної області диплому використовує ті ж актори, що й в діаграмі прецедентів, щоб показати послідовність дій між ними.

Побудована діаграма послідовності представлена в Додатку Б.

1.4.3 Діаграма активності. Діаграма активності – це тип поведінкової діаграми UML, яка показує у вигляді блок-схеми логіку виконання дій різних акторів. Вона вважається більш функціональним підходом для представлення предметної області. "Головна мета діаграми послідовності – показати порядок виконання, або послідовність дій. Водночас діаграма діяльності потрібна для опису роботи всієї системи, вона показує перехід від однієї дії до іншої" [3].

Основні її елементи.

- Початок і кінець – початок показує стартову точку процесу, а кінець завершення усіх потоків процесу.
- Дія – конкретна діяльність, яку виконує користувач в предметній області.
- Рішення – місця, де процес залежить від умови поставленої
- Вилка і злиття – вилка розподіляє потік на кілька паралельних потоків без прийняття рішення, а злиття об’єднує ці розподілені потоки.
- Доріжки – спосіб показати, які дії виконуються одним актором.

Для побудови діаграми активності до предметної області диплому використовуються також актори з побудованої діаграми прецедентів.

Побудована діаграма активності представлена в Додатку В.

2 ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

2.1 Загальний опис ER-діаграми

Розробка будь-якої ПЗ завжди супроводжується роботою з даними, і тому уникнути використання БД у сучасних розробках систем неможливо, оскільки вона забезпечує важливі процеси для інформаційного забезпечення як збереження, обробка і надання інформації користувачу. БД бувають різних типів, містять різний розмір, будь-яку кількість таблиць і зв'язків. Використання правильної спроєктованої БД дозволить в подальшому уникнути помилок і гармидеру серед великої кількості накопиченої інформації відповідно сформованих вимог користувача. Щоб зрозуміти, які дані необхідні та на які частини можна розділити в плані сутностей, щоб розуміти взаємодії між ними використовують моделювання даних, що поділяється на логіне та фізичне.

Логічне моделювання слугує етапом під час якого визначається структура даних, їхні властивості та взаємозв’язки. Воно ґрунтується на результатах аналізу предметної області та має за мету створення абстрактної моделі, яка чітко відображає сутності, атрибути їхні та логіку перетікання цих даних у межах розроблюваної системи. Якраз для побудови такої моделі використовують ER-діаграму, що забезпечить чітку та логічну структуру БД, яка легко адаптуватиметься до змін і задовольнятиме вимоги користувача. Різновид такої діаграма дозволить наочно представити логічну структуру бази даних завдяки 3 важливим концепціям: сутність, атрибут та зв'язок [5].

Визначення концепцій ER-діаграми.

- Сутність – це екземпляр реального чи абстрактного об'єкта предметної області, що володіє особистими властивостями.
- Атрибут – це властивість визначеної сутності. Вони поділяються на ключові та неключові, і їх може бути декілька.
- Зв’язок – це іменований асоціативний зв’язок між сутностями. Іменований зв'язок повинен супроводжуватися дієслівною фразою.

Основна мета використання ER-діаграми – це продемонструвати, які на вигляд взаємозв’язки між визначеними сутностями та їх атрибути в предметній області, що в подальшому слугуватиме підґрунтям для створення БД, тобто коли ще не створені таблиці, але ми чітко розуміємо, які дані мають зберігатися, де вони знаходяться та як вони пов’язані.

Етапи проєктування логічної моделі даних за ER-діаграмою.

- Визначення сутності предметної області
- Аналіз та визначення первинного ключа у сутності.
- Будування зв’язків між сутностями (ідентифікуючи чи неідентифікуючий, один-до-одного, один-до-багатьох, багато-до-багатьох).
- Визначення усіх атрибутів для сутностей, котрі будуть характеризувати його складову.
- Проведення нормалізації, що допоможе уникнути зв’язків "багато-до-багатьох" і забезпечити 3 нормальну форму.

Це проміжний етап між аналізом предметної області, що вже зроблено, та фізичним проєктуванням бази даних, яке почнеться після отримання логічної моделі даних розроблювальної системи.

2.2 Побудова ER-діаграми

Для побудови ER-діаграми використано програму CA Erwin Data Modeler. Це програма, що надає корисні функції розробнику по створенню моделей бази даних з подальшою трансформацією фізичної моделі в БД [6].

ERwin має два типи подання моделі.

- Логічний рівень: показує абстрактний погляд даних, що подаються у вигляді сутності з реального світу, що має свої характеристики і зв’язки. Найголовніше те, що на ньому нема прив’язки до типів даних конкретної СУБД, оскільки логічна модель зосереджена на структурі даних, а не на їх реалізації.
- Фізичний рівень: фізичний рівень моделювання необхідний для вже остаточного створення БД. Тобто на цьому рівні атрибути сутностей мають типи даних, що відповідають обраній СУБД.

Побудовано ER-діаграму на логічному рівні в програмі Erwin на рис. 4.

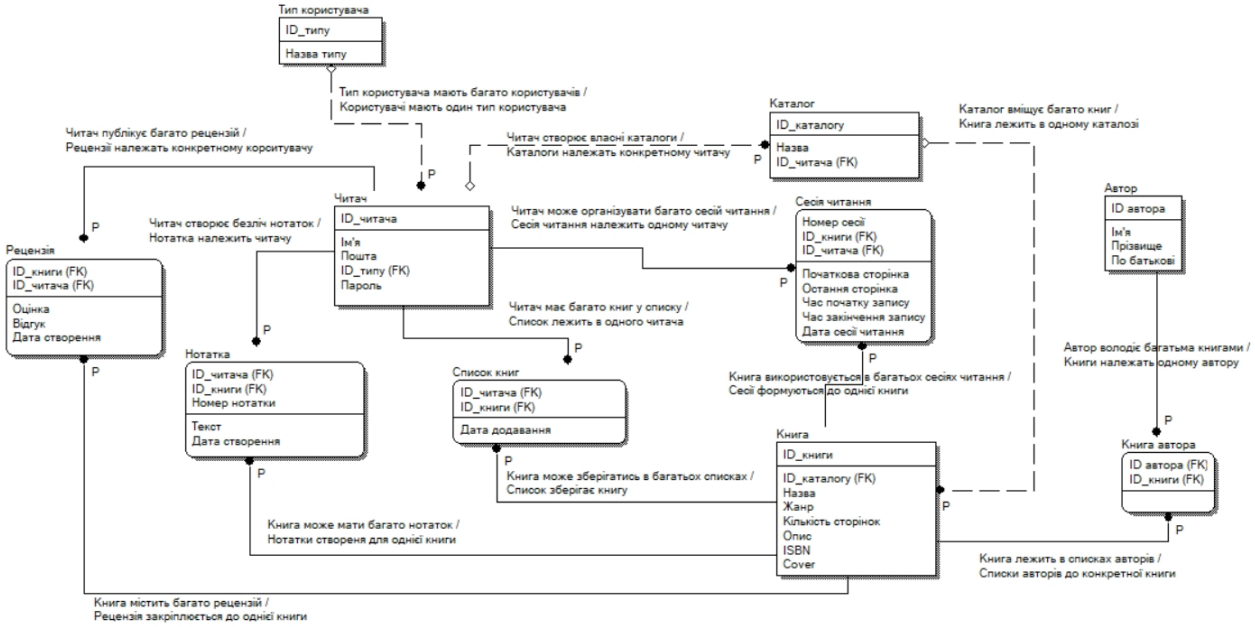


Рис. 4 Логічний рівень ER-діаграми

Опис побудованої ER-діаграми.

Сутність "Список книг"

- Атрибути сутності

1. ID_книги (ключовий атрибут).

2. ID_читача (ключовий атрибут).

3. Дата додавання.

- Зв’язок

1. Зв’язок від "Книги" до "Список книг" – ідентифікуючий (Книга лежить в списках книг, а список може зберігати книгу).

Сутність "Книга автора"

- Атрибути:

1. ID_книги (ключовий атрибут).

2. ID_автора (ключовий атрибут).

- Зв’язок:

1. Зв’язок від "Автор" до "Книга автора" – ідентифікуючий (автор володіє багатьма книгами, а книги належать одному автору).

2. Зв’язок від "Книги" до "Книга автора" – ідентифікуючий (Книга лежить в списках книг авторів, а книги авторів мають книгу).

Сутність "Автор"

- Атрибути:

1. ID_автора (ключовий атрибут).
2. Ім’я.
3. Прізвище.
4. По батькові.

- Зв’язок:

1. Зв’язок від "Автор" до "Книга автора" – ідентифікуючий (автор володіє багатьма книгами, а книги належать одному автору).

Сутність "Тип користувача"

- Атрибути:

1. ID_типу (ключовий атрибут).
2. Назва типу.

- Зв’язок:

1. Зв’язок від "Тип користувача" до "Читач" – неідентифікуючий (тип користувача мають багато користувачів, а користувач мають один тип користувача).

Сутність "Книга"

- Атрибути:

1. ID_книги (ключовий атрибут).
2. ID_каталогу (FK).
3. Назва.
4. Жанр.
5. Кількість сторінок.
6. Опис.
7. ISBN.
8. Cover

- Зв’язок:

1. Зв’язок від "Книга" до "Нотатка" – ідентифікуючий (одна книга може містити багато нотаток, а нотатки формуються до конкретної книги).

2. Зв’язок від "Книга" до "Сесія читання" – ідентифікуючий (одна книга використовується у багатьох сесіях читання, а сесія прив’язується до конкретної книги).

3. Зв’язок від "Книга" до "Рецензія" – ідентифікуючий (одна книга містить багато рецензій, а рецензія публікується за конкретною книгою).

4. Зв’язок від "Книга" до "Книга автора" – ідентифікуючий (книга лежить в списках авторів, а списки авторів до конкретної книги).

Сутність "Книга автора".

- Атрибути:

1. ID_автора (ключовий атрибут).

2. ID_книги (ключовий атрибут).

- Зв’язок:

1. Зв’язок від "Книга" до " Книга автора" – ідентифікуючий (книга лежить в списках авторів, а список авторів до конкретної книги).

2. Зв’язок від "Автор" до " Книга автора" – ідентифікуючий (автор володіє багатьма книгами, а книга лежить).

Сутність "Читач"

- Атрибути:

1. ID_читача (ключовий атрибут).

2. Ім’я.

3. ID_типу.

4. Пошта.

- Зв’язок:

1. Зв’язок від "Читач" до "Рецензія" – неідентифікуючий (читач публікує багато рецензій, а рецензії належать конкретному читачу).

2. Зв’язок від "Читач" до "Сесія читання" – неідентифікуючий (читач організовує багато сесій читання, кожна сесія належить одному читачу).

3. Зв’язок від "Читач" до "Нотатка" – ідентифікуючий (читач створює багато нотаток, а нотатка належить читачу).

4. Зв’язок "Читач" до "Список книг" – ідентифікуючий (читач має багато книг у списку, а список лежить в одного читача).

5. Зв’язок "Читач" до "Каталог"– неідентифікуючий (читач створює власні каталоги, а каталоги належать конкретному читачу).

Сутність "Сесія читання"

- Атрибути сутності:

1. ID_читача (FK).
2. ID_книги (FK).
3. Номер сесії.
4. Початкова сторінка.
5. Остання сторінка.
6. Час початку запису.
7. Час закінчення запису.
8. Дата сесії читання.

- Зв’язок:

1. Зв’язок від "Читач" до "Сесія читання" – ідентифікуючий (читач організовує багато сесій читання, а кожна сесія належить одному читачу).

2. Зв’язок від "Книга" до "Сесія читання" – ідентифікуючий (одна книга використовується в багатьох сесіях читання, а сесія читання пов’язана за конкретною книгою).

Сутність "Нотатка"

- Атрибути:

1. ID_читача (FK).
2. ID_книги (FK).
3. Номер нотатки (FK).

4. Текст.

5. Дата створення.

- Зв’язок:

1. Зв’язок від "Читач" до "Нотатка" – ідентифікуючий (читач створює багато нотаток, а нотатки належать конкретному читачу).

2. Зв’язок від "Книга" до "Нотатка" – ідентифікуючий (книга містить багато нотаток, а нотатки стосуються конкретної книги).

Сутність "Каталог"

- Атрибути:

1. ID_каталогу (ключовий атрибут).

2. Назва.

3. ID_читача (FK)

- Зв’язок:

1. Зв’язок від "Каталог" до "Книга"– неідентифікуючий (каталог містить багато книг, а книга зберігається в одному каталозі).

Сутність "Рецензія"

- Атрибути:

1. ID_читача (FK).

2. ID_книги (FK).

3. Оцінка.

4. Відгук.

5. Дата створення.

- Зв’язок:

1. Зв’язок від "Читач" до "Рецензія" – ідентифікуючий (читач публікує багато рецензій, а рецензія належить одному читачу).

2. Зв’язок від "Книга" до "Рецензія" – ідентифікуючий (книга містить багато рецензій, а рецензія закріплюється за конкретною книгою).

Наступною логічною дією після побудови діаграми є нормалізація, що являє із себе процесом над побудованою діаграмою, де приводиться структура до

меншої надмірності, поліпшує цілісність даних і забезпечує ефективність виконання SQL запитів в подальщій реалізації. Процес нормалізації проходить поетапно, у вигляді нормальних форм [12]. Кожна наступна форма ґрунтується на попередній і їх існує загалом шість, але достатньо мати перші 3 форми:

1) перша нормальна форма – атрибути мають містити атомарні значення, тобто неділимі.

2) друга нормальна форма – відповідає першій нормальній формі і неключові атрибути залежать від ключового.

3) третя нормальна форма – відповідає другій нормальній формі і має бути відсутня транзитивна залежність, де неключовий атрибут залежить від іншого також неключового атрибута.

Проведено перевірку сутностей ER-діаграми на дотримання 3 нормальної форми.

Сутність "Читач".

- Відповідає 1НФ, оскільки всі атрибути містять лише атомарні значення.
- Відповідає 2НФ, оскільки немає часткових залежностей (атрибути залежать тільки від ID_читача).
- Відповідає 3НФ, оскільки немає транзитивних залежностей. Загалом усі атрибути залежать лише від первинного ключа.

Сутність "Книга".

- Відповідає 1НФ, оскільки кожен атрибут містить лише одиничне значення, тобто вони атомарні.
- Відповідає 2НФ, оскільки всі атрибути залежать від ID_книги, а не від його частини.
- Відповідає 3НФ, оскільки не містить транзитивних залежностей. Загалом усі атрибути напряму залежать від первинного ключа.

Сутність "Книга автора".

- Відповідає 1НФ, оскільки кожен атрибут містить лише одиничне значення, тобто вони атомарні.

- Відповідає 2НФ, оскільки всі атрибути залежать від ключового атрибуту.
- Відповідає 3НФ, оскільки не містить транзитивних залежностей. Загалом усі атрибути напряму залежать від первинного ключа.

Сутність "Тип користувача".

- Відповідає 1НФ, оскільки кожен атрибут містить лише одиничне значення, тобто вони атомарні.
- Відповідає 2НФ, оскільки всі атрибути залежать від ключового атрибуту.
- Відповідає 3НФ, оскільки не містить транзитивних залежностей. Загалом усі атрибути напряму залежать від первинного ключа.

Сутність "Автор".

- Відповідає 1НФ, бо всі атрибути є атомарними.
- Відповідає 2НФ, бо всі атрибути залежать від складеного первинного ключа (ID_книги, ID_читача, Номер сесії).
- Відповідає 3НФ, оскільки всі атрибути безпосередньо залежать від первинного ключа, транзитивних залежностей немає.

Сутність "Каталог".

- Відповідає 1НФ, тому що атрибути сутності містять тільки атомарні значення.
- Відповідає 2НФ, тому що всі атрибути залежать від унікального ключового атрибуту (ID_каталогу).
- Відповідає 3НФ, тому що немає транзитивних залежностей.

Сутність "Список книг".

- Відповідає 1НФ, тому що кожен атрибут атомарний.
- Відповідає 2НФ, тому що неключовий атрибут залежить від складеного первинного ключа (ID_читача, ID_книги).
- Відповідає 3НФ, тому що немає транзитивних залежностей, тобто всі атрибути залежать тільки від первинного ключа.

Сутність "Нотатка".

- Відповідає 1НФ, оскільки всі атрибути містять лише атомарні значення.
- Відповідає 2НФ, оскільки всі атрибути залежать від складеного первинного ключа (ID_читача, ID_книги, Номер нотатки).
- Відповідає 3НФ, оскільки немає транзитивних залежностей – всі атрибути напряму залежать від первинного ключа.

Сутність "Рецензія".

- Відповідає 1НФ, тому що всі атрибути атомарні.
- Відповідає 2НФ, оскільки всі атрибути залежать від складеного первинного ключа (ID_читача, ID_книги).
- Відповідає 3НФ, оскільки всі атрибути напряму залежать від первинного ключа, транзитивних залежностей немає.

Сутність "Сесія читання".

- Відповідає 1НФ, тому що всі атрибути атомарні.
- Відповідає 2НФ, оскільки всі атрибути залежать від складеного первинного ключа (ID_читача, ID_книги).
- Відповідає 3НФ, оскільки всі атрибути напряму залежать від первинного ключа, транзитивних залежностей немає.

Як результат, побудована діаграма охоплює всі ключові сутності предметної області, їхні характеристики та взаємозв’язки. Реляційний тип у даній діаграмі виражається через сутності, де кожен атрибут, як ключовий чи не ключовий, представляє інформацію про сутність. Маючи таку модель, стала доступна можливість побудови цієї моделі на фізичному рівні.

2.3 Обґрунтування вибору СУБД

СУБД – це засіб для повної взаємодії з базами даних: від самого створення, до повного керування її частин за допомогою SQL запитів. Її роль полягає в забезпеченні доступу до інформації для створення, управління, зберігання та безпеки. Вибір ідеальної для системи СУБД відіграє важливу роль у забезпеченні правильного функціонування інформаційної системи, бо саме вона відповідає за

різні дії в БД. Вибір також залежить від конкретних потреб і вимог користувачів до СУБД.

СУБД можна поділити на декілька типів: реляційні, об'єктно-орієнтовані та навіть NoSQL [9].

Пояснення типів СУБД.

- Реляційні: вони побудовані на основі характерних для реляційної моделі таблиць, де самі дані показані у вигляді стовпців і рядків, що наразі є основою для більшості сучасних систем.
- NoSQL: такий тип використовуються для розподілених сховищ даних. Призначенням NoSQL виступає зберігання великих обсягів гнучко структурованих або неструктурованих даних. Такі бази ефективні для високонавантажених і розподілених систем.
- Об’єктно-орієнтовані: такий тип з очевидної назви використовує парадигму об’єктно-реляційного програмування. Тобто тип подібний до описаного вгорі реляційного типу, однак одночасно використовує і об'єктно-орієнтовну, яка налічує такі елементи як класи, об’єкти, спадкування. Це забезпечить глибоку взаємодію із системами, побудованими на об’єктно-орієнтованих мовах.

Враховуючи вимоги до системи автоматизованого супроводження читання книг і побудованої ER-діаграми до неї, можна виділити наступні вимоги до вибору СУБД, завдяки яким підбереться підхожий тип з яких працюватиме людина:

1. Підтримка реляційної моделі – організація даних у вигляді таблиць з атрибутами з чітко визначеними зв’язками.

2. Забезпечення пошуку, модифікації та збереження даних.

3. Мати непогану екосистему, тобто сумісність з технологіями, середовищем виконання, а також іншими технологіями, що використовуються (Node.js, .NET, Visual Studio).

4. Резервне копіювання.

5. Висока затребуваність на сучасному ринку.

6. Підтримка безпеки.

7. Цілісність даних.

Підібрано СУБД відповідно поставленим вимогам – Microsoft SQL Server, яка є сучасна реляційна СУБД, яка підтримує як локальне, так і хмарне зберігання інформації. Вона забезпечує масштабованість та безпеку, що робить її популярним вибором для великих підприємств та фінансових організацій. SQL Server в основному побудований навколо таблиць і рядків, яка з’єднує пов’язані елементи даних у різних таблицях один з одним, уникаючи необхідності надмірного зберігання даних у кількох місцях у базі даних, тобто це характерно для реляційної моделі. Реляційна модель тут забезпечує цілісність та інші обмеження цілісності для підтримки точності даних.

Особливості СУБД Microsoft SQL Server [7, 10].

- Створення клієнт-серверних архітектур: чудово підходить для інформаційних систем, котрі працюють за архітектурою "клієнт-сервер", де сервер виконує всі основні операції обробки запитів і зберігання даних.
- Підтримка цілісності даних: завдяки суворому дотриманню ACID (атомарність, консистентність, ізольованість, довговічність), SQL Server забезпечує надійне управління даних у БД.
- Підтримка паралельної обробки: дозволяє здійснювати паралельну обробку запитів для підвищення продуктивності.
- Безпека: високий рівень безпеки завдяки механізмам шифрування даних та групи доступності, що постійно ввімкнені, а також завдяки автентифікацією користувачів, контролю доступу.
- Інтеграція з іншими технологіями Microsoft: чудова підтримка екосистеми Microsoft, яка містить різні, наприклад: Visual Studio, Power BI, технологію .NET. SQL Server найкраще працює в середовищі Windows, що водночас може виступати як обмеження для людей, які якраз використовують інші ОС.

- Можливість масштабування: завдяки підтримці розподілених обчислень і хмарних рішень, SQL Server може обробляти великі обсяги даних і забезпечувати високу продуктивність у системах. Загалом, СУБД може адаптуватися до мінливих потреб розробника.

- Технологія займає у топі 3 СУБД, що частіше використовують, обговорюють і які затребувані на ринку за рейтингом DB-Engines.

Недоліки СУБД Microsoft SQL Server.

- Складність міграції: перенесення даних на інші СУБД може бути складним процесом через специфічні формати збереження даних та залежність від мови T-SQL, що є родзинкою SQL Server. Це ускладнює зміну платформи у випадку потреби, тож цю СУБД потрібно обирати обережно.

- Високі вимоги до ресурсів: використовуються великі обчислювальні ресурси, особливо у роботі з великими обсягами даних. Для ефективної функціональності потрібне серверне обладнання, що насамперед підвищує витрати на інфраструктуру.

- Застарілість і складність інтерфейсу: нема інтуїтивного розуміння що, де і як використовується.

- Висока вартість ліцензування: у порівнянні з іншими СУБД, проте для малих проєктів вистачить можливостей, що надає SQL Server.

SQL Server є оптимальним вибором для проєктування БД з реляційною моделлю для ПЗ системи автоматизованого супроводження читання книг. Вона забезпечить надійність і цілісність даних, безпеку і підтримку розширених можливостей для обробки та аналізу даних, що робить її чудовим рішенням для експлуатації як у великих організацій та проєктів, так і малих, що працюють як із великими обсягами даних, так і малими, що отримають стабільність й гнучкість.

2.4 Створення БД

Створення БД розпочалось після побудови ER-діаграми в програмному продукті ERwin Data Modeler як на логічному, так і фізичному рівнів для якої

спершу обрано СУБД, і на основі цього вибору визначено типи даних для кожного атрибуту, первинні і зовнішні ключі. Фізичний рівень ER-діаграми представлено в Додатку Д.

Одразу як БД створено, написано SQL запити на створення таблиць із визначеними обмеженнями цілісності, включаючи значення цих полів як NOT NULL, UNIQUE, CHECK, FOREIGN KEY, а також тригера. Для БД системи супроводження читання книг важливо мати такі обмеження, які задають правила стовпцям таблиць, щоб уникнути неточностей даних і помилкових зв’язків між таблицями в подальшій розробці, а також тригера, що виконуватиме перевірку з даними до того, як вони будуть записані в БД [11].

Пояснення використаних обмежень в побудові БД.

- FOREIGN KEY – обмеження для запобігання руйнувань зв’язків між таблицями. Саме по собі це значення поля в таблиці, яке посилається на ключовий атрибут іншої таблиці з якою має зв’язок.
- UNIQUE – обмеження, що задає унікальність стовпцю, тобто збережений запис буде унікальним завдяки такому стовпцю і його не можна буде використати в наступних записах.
- CHECK – обмеження, що визначає діапазон значень, якими не можна заповнювати у стовпці. У ньому можна також використовувати умовні оператори як у звичайній мові програмування. Наприклад при створенні таблиці ReadingSession в БД це обмеження дуже знадобилося, щоб не можна було створювати сесію читання, де кінцева сторінка менша вказану початкову.
- NOT NULL – обмеження, яке робить так, що при створенні запису в таблиці стовпець обов’язково має містити значення, інакше виникне помилка.

Особливу увагу акцентовано зовнішнім зв’язкам під час створення таблиць, де в особливих випадках використано корисне обмеження ON DELETE CASCADE. "Використовується для автоматичного видалення рядків з дочірньої таблиці, коли рядки з батьківської таблиці видаляються" [13]. Обмеження корисне в розроблювальній БД, бо дозволяє автоматично видаляти залежні

записи, наприклад: сесії читання, нотатки, рецензії користувача після його видалення. Воно зменшує ймовірність появи так званих "висячих" записів та забезпечує логічну послідовність збережених даних, і можна вважати, що він частково замінює використання тригера для видалення вищезгаданих записів.

У СУБД є 2 способи створення таблиць що продемонстровано на рис. 5 і 6.

1. Запитом SQL

```
CREATE TABLE TypeUser (
    ID_Type char(10) PRIMARY KEY,
    TypeName NVARCHAR(50) NOT NULL
);

CREATE TABLE Reader (
    ID_Reader CHAR(10) PRIMARY KEY,
    Nickname NVARCHAR(100) NOT NULL,
    Email NVARCHAR(255) UNIQUE NOT NULL,
    UserPassword NVARCHAR(255) NOT NULL DEFAULT '',
    ID_Type char(10) NOT NULL,
    FOREIGN KEY (ID_Type) REFERENCES TypeUser(ID_Type)
);
```

Рис. 5 Приклад створення таблиць через SQL запит

2. Конструктором

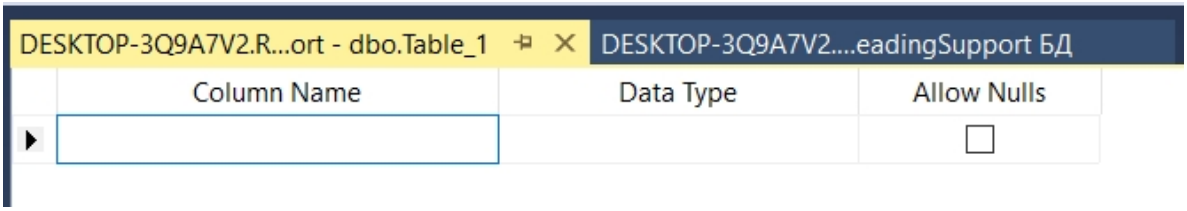


Рис. 6 Приклад створення таблиць через конструктор

Створення таблиць і тригера зроблено завдяки SQL запитам, оскільки такий спосіб дає впевненість у правильності їх створенні. Запити на створення усіх таблиць і тригерів представлено в Додатку Е.

Також до вищесказаного створено уявлення в БД. Уявленнями можна вважати спеціальними віртуальними таблицями, котрі створенні на основі вже тих, що створенні в БД. "Вони пов'язані між собою, тобто, якщо зміни відбуваються в таблиці уявлення, це відображається в оригінальній таблиці, і якщо зміни внесено в оригінальну таблицю, то вони відображаються в таблиці уявлення" [14, 15].

Цілі використання уявлень.

- Спростити доступ до використання даних: уявлення дозволять зберігати складні SQL запити як окремі об'єкти, до яких користувач може звернутись по інформацію.
- Забезпечити простоту роботи з об’єднанням і агрегацією: уявлення дозволяють користувачам об'єднувати дані з окремих створених таблиць БД, у яких буде відбуватися підрахунок, вираховування середнього значення і т. д. Як результат, усе відобразиться у вигляді однієї таблички.
- Покращити читабельність і підтримку коду: передбачається те, щоб не повторювати складні запити в різних частинах програми або аналітичних звітах, натомість мати все в одному місці – уявленні.
- Убезпечити дані: можна обмежити доступ лише до необхідних полів або рядків таблиць, тобто не обов’язково давати повний перегляд таблиці.

Опис використаних уявлень в системі.

- View_ReaderNote, яке відображає всі нотатки читача заразом з інформацією про автора та книгу, до якої створена нотатка. Використання такого уявлення спростить доступ до нотаток для подальшого виводу в інтерфейсі СУБД або інтерфейсі читача.
- View_ReaderProgressHisBook для представлення прогресу читання конкретної книги. У ньому якраз обчислюєть прогрес читання книги: кількість прочитаних сторінок і відсоток завершення читання. Для своєї роботи уявлення використовує таблиці Book, Reader і ReadingSession, яке об’єднує дані і показує своєрідну аналітику по певній книзі.

Запити на створення уявлень представлено в Додатку Е.

У результаті створено БД для системи супроводу читання книжок в Microsoft SQL Server. Представлено схему БД, що являє собою угрупування об'єктів бази даних, а також список усіх створених таблиць (об’єктів), що складають схему, яку зображено на рис. 7 і 8.

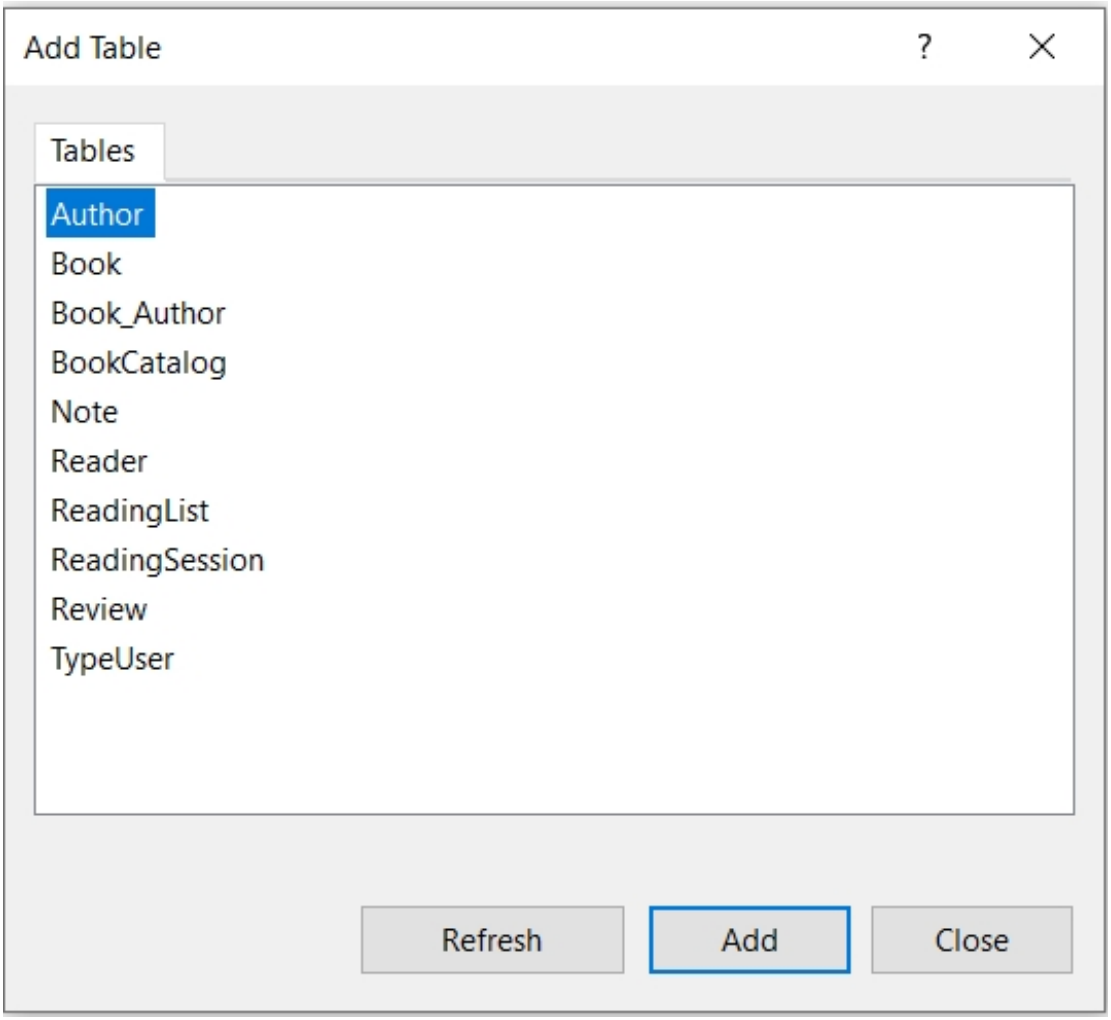


Рис. 7 Список таблиць схеми

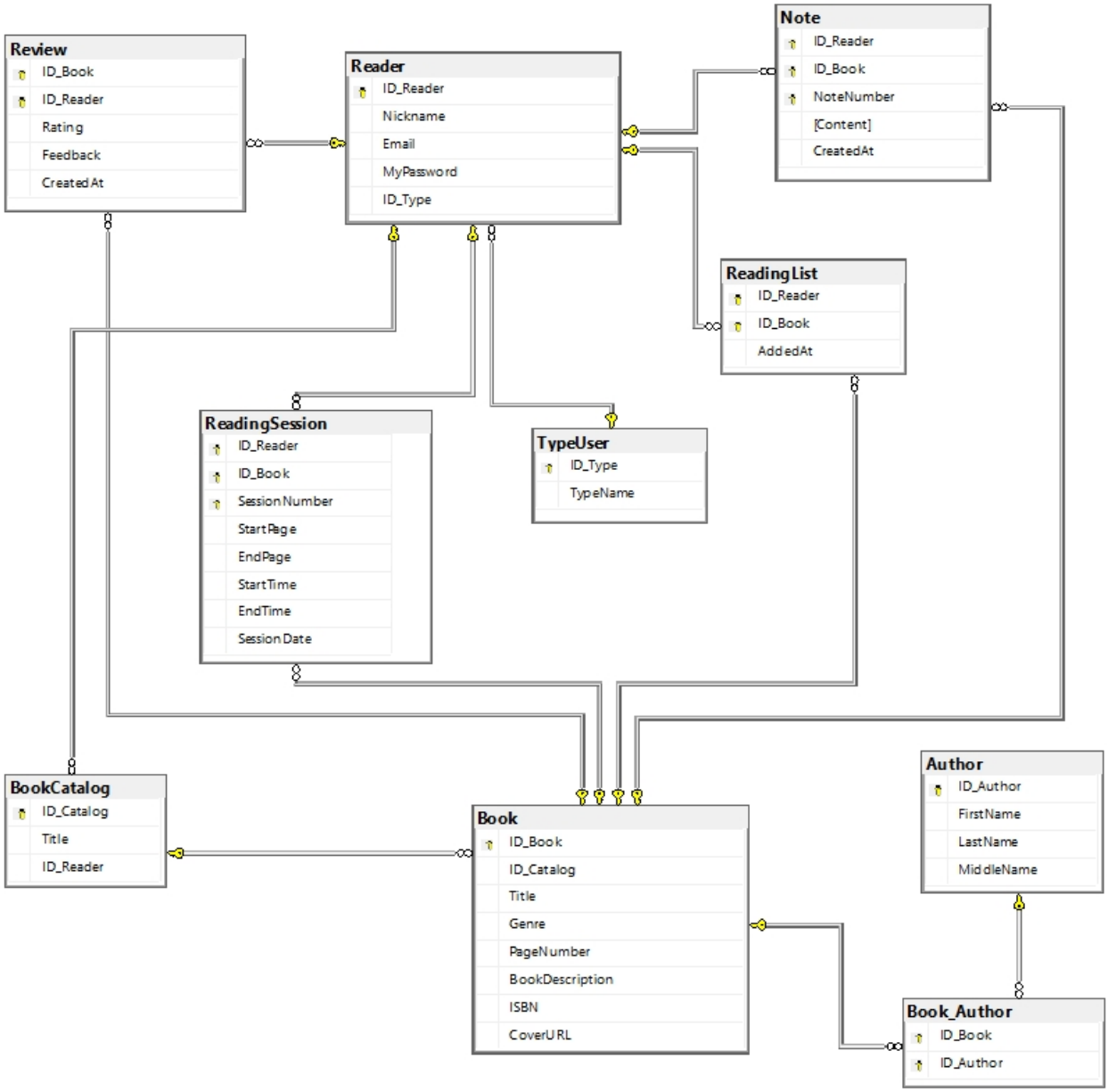


Рис. 8 Побудована схема БД

3 ПРОЄКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Вибір інструментарію та середовища розробки

Проєктування є новим етапом у розробці, який наближує до повноцінної реалізації задуманої ПЗ системи. Мета етапу полягає в описі процесу безпосереднього проєктування, що спершу включає вибір інструментарію для розробки системи, що буде складатися з таких частин: фронтенд і бекенд. Далі описують побудови структурних діаграм за допомогою UML, що покажуть елементи, їх взаємозв'язки з чого взагалі складається система.

Проаналізувавши предметну область і з'ясувавши вимоги до розроблювальної системи, виділено інструментарій, що допоможе в реалізації частин системи, що складатимуть єдине ціле – фронтенд і бекенд.

Фронтенд частина.

Невід'ємна частина, що відповідає за побудову інтерфейсу веб-сайту, акцент якого зосереджений на взаємодію з користувачем. "Інтерфейс є ключовою ланкою у взаємодії користувача з вебсайтом, виступаючи своєрідним посередником між складним програмним кодом та людиною" [16].

Стандартні складові фронтенд частини [16]:

- HTML – для побудови структури веб-сторінки.
- CSS – для стилізації сторінки.
- JavaScript – для створення інтерактивності, що робить сторінку динамічною.

Для реалізації сучасних рішень в побудові користувацьких інтерфейсів використовуються різні інструменти, одним з яких є найпопулярніша бібліотека React. Головною перевагою цієї бібліотеки є компонентний принцип, суть якого у розбитті інтерфейсу на смислові частини, які потім можна перевикористовувати для побудови вже складніших інтерфейсів.

Однак, щоб побачити побудований інтерфейс за допомогою React, де вірогідно використовується JSX-синтаксис у створенні компонентів, який браузер не розуміє, імпорти з окремих файлів, маршрутизацію, стилізацію компонентів, необхідно використати спеціальний інструмент, котрий об’єднає усе в єдине ціле, що браузер зрозуміє. Цим інструменм є Vite збірник, метою якого є забезпечення швидшого та спрощеного процесу розробки сучасних веб-проєктів, зокрема розроблювальної системи на React бібліотеці [17]. Використання цього інструменту дало можливість не витрачати дарма час на налаштування середовища, а приділити його на побудову інтерфейсу і написанню логіки роботи системи.

Що стосується комунікації із серверною частиною (бекенд), для цього задіяно архітектурний стиль REST API, для якого передача даних проходить через HTTP-запити з методами: GET, POST, PUT, DELETE. "REST API слугує мостом між цими двома частинами. Він дає змогу frontend надсилати запити до backend і отримувати дані звідти" [18].

Наведено можливі застосування методів REST API у розробці системи.

1. GET: метод для отримання даних із сервера. Наприклад для отримання переліку читачів на сайті, списку рецензій.

2. POST: метод для створення нового ресурсу і надсилання його на сервер. Наприклад, коли додається новий користувач у систему, нова нотатка до книги чи сама книга.

3. PUT: метод для повного оновлення ресурсу на сервері. Наприклад для зміна ролі користувача з читача на модератора.

4. DELETE: метод для видалення ресурсу на сервері. Наприклад для видалення користувача із системи, нотатки для книги або написаної рецензії, що порушує правила цензури.

Щоб спростити роботу з HTTP розглянуто одну з також популярних бібліотек – Axios, яка використовується для створення HTTP-запитів, дозволяючи користувачу надсилати з клієнтської частини (фронтенд) дані й

отримувати у відповідь результат від сервера у форматі JSON [19]. Axios легко вбудовується у React застосунок, забезпечуючи асинхронність роботи і без зайвого ускладнення написання запитів.

Для реалізації фронтенд частини розроблювальної системи використано React із застосуванням CSS стилізації, Axios з REST API архітектурою для зв’язку з бекенд частиною, а також інструмент для створення збірки – Vite.

Бекенд частина.

Розробка системи з лише реалізованою фронтенд частиною не дозволяє виконувати повноцінну взаємодію з БД, а також спеціальну бізнес-логіку згідно задуманих функціональних вимог. Через це інтерфейс системи має статичну оболонку, що показує наперед готову інформацію з якою ніяк не можна взаємодіяти. Саме цією частиною якраз займається бекенд, що відповідає за взаємодію користувача з внутрішніми даними, які потім відображає фронтенд [20]. Тож з цієї причини додано серверну частину (бекенд), котра забезпечить обробку запитів від клієнта, зберігання, зміну та передачу даних між клієнтом і БД. Окрім цього, бекенд забезпечить виконання бізнес-логіки для реалізації функціональних вимог, які користувачу надасть інтерфейс [20].

Взаємодія обох частин відбувається по колу.

1) користувач відправляє інформацію на сервер;
2) програма на сервері опрацьовує інформацію;
3) інформація повертається користувачу в зрозумілій для формі.

Щоб реалізувати цю частину обрано Node.js і фреймворк Express – частовживаний серед бекенд-розробників дует, котрий дозволяє розробляти гнучкі серверні частини систем.

Node.js надає зручні інструменти та бібліотеки для створення серверної частини, а також забезпечить просту взаємодію з фронтенд частиною на основі JavaScript. Завдяки цієї мови на обох частинах (фронтенд і бекенд) розробка стає послідовною, цілісною і зрозумілою, особливо для тих, хто вже мав досвід із цією мовою [21].

Стосовно Express – це фреймворк для Node.js, що надає прості й ефективні функції для обробки HTTP-запитів від користувача, маршрутизації, а також для з’єднання з БД, щоб виконувати різні дії із нею в залежності поставлених вимог. Він заснований на стандартних API Node.js, що робить його простим у вивченні та використанні, оскільки дозволяє будувати маршрути, обробники запитів, підключати middleware обробники та взаємодіяти з будь-якою БД [22].

Для реалізації бекенд частини розроблювальної системи використано середовище Node.js із фреймворком Express.

Окрім вибору інструментарію для фронтенд і бекенд частин, варто потурбуватися і про контроль процесу розробки, який забезпечить інструмент GIT, котрий виступає розподіленою системою для контролю версій, що дозволить фіксувати зміни в розробці, відстежувати всю історію змін і за бажанням повертатися до попередніх версій, що дасть гнучкість у розробці. З цією метою для розробки системи автоматизованого супроводження читання книг було обрано цей інструмент, завдяки якому створено репозиторій у якому зберігатимуться усі зміни під час розробки.

Отже, після вибору перерахованих інструментів, подальша розробка не обійдеться без IDE, що являє із себе середовище, у якому розроблюється програма. IDE пропонує необхідні інструменти для написання коду та його тестування, що буде виконуватися в єдиному середовищі, яке поєднає редактор коду, компілятор, інтерпретатор, відладчик. "Інтегровані середовища розробки створені для того, щоб максимізувати продуктивність програміста, надавши йому пов'язані інструменти розробки зі схожими інтерфейсами як одну програму, в якій відбуватиметься весь процес розробки й яка надає необхідні функції для модифікації, компілювання, розгортання та налагодження програмного забезпечення" [23]. Одним із представників IDE можна взяти Visual Studio Code. Це один із дуже відомих середовищ розробки, який навіть на тепер не втрачає популярності, а радше навпаки. Можна відзначити його сильною перевагою саме швидкість, стабільність та дуже інтуїтивно зрозумілий

інтерфейс, який не перевантажує користувача, але при цьому він надає багато можливостей через інтегровані розширення.

У розробці системи обрано Visual Studio Code як основне середовище для написання як клієнтської, так і серверної частини, забезпечуючи при цьому зручний і продуктивний робочий процес завдяки вбудованому GIT, а також технології IntelliSense, що дає ряд корисних можливостей як підсвічування коду, надання опису до функції чи іншого елемента у певній мові програмування, пропонування різних варіантів завершення написання коду тощо.

Для проведення тестування основним інструментом виступить Postman, який допоможе з роботою API в системі.

3.2 Побудова діаграм для проєктування ПЗ

Проєктування є новим етапом, який наближує до повноцінної реалізації задуманої ПЗ системи. Важливість етапу зосереджена у формуванні структури за допомогою діаграм класів із зв’язком асоціація, пакетів, кооперацій та компонентів, що представлять основні елементи майбутньої системи.

Перед початком проєктування, що включає побудову діаграм, спочатку необхідно визначитись із абстракціями сутностей проаналізованої предметної області, які мають визначити для чого вони використовуються в ній.

Визначено абстракції предметної області і їхні властивості та обов’язки, які описані в таблицях 1-8.

Таблиця 1

Абстракція "Книга"

Властивості	**Обов’язки**
Назва; Автор; Жанр; ISBN номер; Опис;	Зберігати в каталогах; Відображати інформацію про себе; Представляти рецензії читачів про себе;

Таблиця 2

Абстракція "Читач"

Властивості	**Обов’язки**
Ім’я; Пошта; Статистика читання; Кількість прочитаних книг; Кількість написаних рецензій;	Читати книжки; Взаємодіяти з книжкою; Переглядати прогрес читання та формувати статистику;

Таблиця 3

Абстракція "Рецензент"

Властивості	**Обов’язки**
Ім’я; Кількість написаних рецензій;	Ознайомлюватись з рецензіями інших людей; Публікувати власні рецензії;

Таблиця 4

Абстракція "Модератор"

Властивості	**Обов’язки**
Ім’я; Пошта; Дата призначення на роль;	Стежити за дотриманням правил публікації рецензій; Приймати рішення щодо видалення контенту або блокування читачів;

Таблиця 5

Абстракція "Каталог"

Властивості	**Обов’язки**
Назва; Список книг; Дата створення;	Структурувати книжки за довільним критерієм (жанром, статусом читання, приналежність до автора, за алфавітом);

Таблиця 6

Абстракція "Рецензія"

Властивості	**Обов’язки**
Текст; Оцінка; Ім’я автора; Дата створення; Назва книги;	Ділитися думками про книгу; Підбивати підсумок прочитаного; Впливати на рейтинг книги; Впливати на вибір книги;

Таблиця 7

Абстракція "Нотатка"

Властивості	**Обов’язки**
Текст; Дата створення; Назва книжки;	Зберігати думки та враження до книг; Зорієнтовувати читача в прочитаному;

Таблиця 8

Абстракція "Статистика"

Властивості	**Обов’язки**
Загальний витрачений час на читання; Середній час читання; Середня швидкість читання сторінок; Кількість прочитаних сторінок; Кількість створених нотаток; Дата початку читання; Дата закінчення читання;	Обчислює загальний прогрес читання; Відображати інформацію активності читача за книгою;

Пройшовши цей крок, можна переходити безпосередньо до побудови діаграми класів із зв’язком асоціація.

Діаграма класів.

На етапі аналізу використовуються діаграми класів, щоб підкреслити загальні ролі та властивості сутностей, які забезпечать необхідну поведінку майбутньої системи, а от вже на етапі проєктування використовується для відображення структур класів, які складають архітектуру системи. Діаграма більше робить нахил на показі предметної області, однак робить це з прив’язкою до системи.

Побудована діаграма класів представлена в Додатку Ж. На ній представлено класи зі своїми атрибутами, які мають асоціативний зв’язком, що представляє структурні відносини між об'єктами (+1, +1..*, +0..1):

1. Книга: клас, що відповідає за книгу читача для супроводженого читання.

2. Читач: клас, що представляє людину, що бажає організовувати супровід під час читання.

3. Нотатка: клас, що відповідає за нотатки читача до книги.

4. Статистика: клас, що відповідає за показ активності читання за книгою.

5. Каталог: клас, що відповідає за каталог книг, що зберігає книги.

6. Рецензія: клас, що відповідає за рецензії читача за книгою.

7. Модератор: клас, що відповідає за дотримання правил публікації рецензій і контролю читачів.

Діаграма класів вважається тим моментом, коли завершується остаточно етап аналізу предметної області, і починається етап проєктування системи.

Діаграма пакетів.

Діаграма пакетів для етапу проєктування потрібна, щоб представити у загальному вигляді структуру системи, поділяючи її основні елементи на пакети. Пакет є основним будівельним блоком у цій діаграмі, що об’єднує в собі пов’язані між собою класи для представлення функціоналу. Зв’язки між цими пакетами показують, як ці частини взаємодіють між собою, і які з них залежать одна від одної. На рис. 9 представлено діаграму пакетів системи.

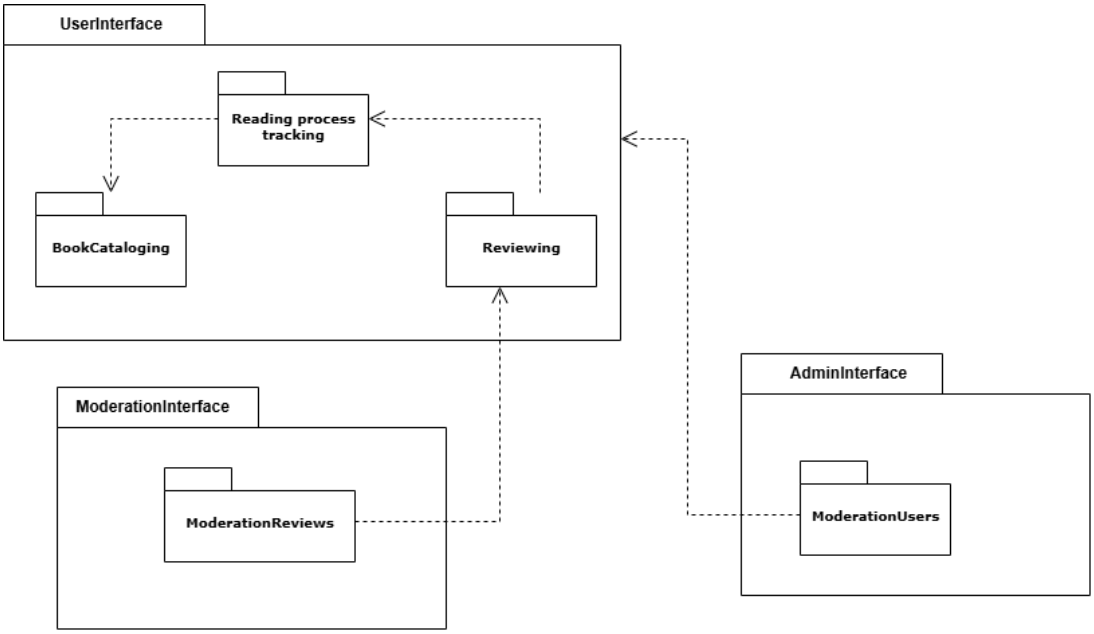


Рис. 9 Діаграма пакетів

Опис пакетів і залежностей діаграми пакетів.

1. UserInterface: представляє графічний інтерфейс для взаємодії з підпакетами. Забезпечує введення параметрів, перегляд інформації, редагування. Пакет містить підпакети BookCataloging, Reading process tracking і Reviewing, бо вони забезпечують функціональність, доступну користувачу через інтерфейс.

2. BookCataloging: підпакет, що дає можливість стурктурувати книжки у каталоги. Включає у собі створення каталогів, а також управління каталогами книг.

3. Reading process tracking: підпакет відповідає організацію супроводу читання книг, зокрема пошуком книг, зберігання, відстеженням часу і кількості сторінок для сесії читання, створенням нотаток, формуванню статистики. Він залежить від BookCataloging у якому зберігаються книги для яких і організовано супровід у читанні.

4. Reviewing: підпакет, що надає можливість написання рецензії до книжки після її прочитання і публікації. Він має залежність від Reading process tracking, оскільки рецензія пишеться до книги для якої проводилося організований супровід у читанні.

5. AdminInterface: містить графічний інтерфейс для взаємодії з пакетами, котрі стосуються адміністратора. Забезпечує перегляд інформації про читача і подальший доступ його в системі. Пакет містить підпакет ModerationUsers, бо він забезпечує функціональність, доступну адміністратору через інтерфейс.

6. ModerationUsers: підпакет, що надає можливість модератору контроль над читачами, де у випадку порушення, встановлених правил системою, відбувається блокування читача. Підпакет має залежність з пакетом UserInterface, оскільки модерація користувачів безпосередньо пов’язана з їхньою взаємодією із системою. Якщо адміністратор вирішує заблокувати користувача, то його дія вплине на використання інтерфейсу з функціоналом.

7. ModerationInterface: містить графічний інтерфейс притаманний адміністратору (модератору) для взаємодії з підпакетами. Пакет містить підпакет ModerationReviews, який відповідає за перевірку опублікованих рецензій і видалення у разі порушення встановлених правил системи.

8. ModerationReviews: підпакет, що перевіряє опубліковані рецензії і застосовує відповідні дії у разі порушення правил. Він має залежність із Reviewing, оскільки модератори працюють із рецензіями.

Як результат побудовано діаграму пакетів, де кількість пакетів складається з кількості простих кооперацій, проте цією кількістю не обмежується вони.

Діаграма простих кооперацій.

Для побудови простих кооперацій необхідно розбити побудовану діаграму класів на частини за смислом, тому насамперед ідентифіковано механізми, які можна представити у простих кооперціях, а потім класи, що братимуть участь в даній кооперації для кожного механізму. На рис. 10-12 представлені прості кооперації, а декілька прикладів коду реалізації представлено в Додатку Л.

У діаграмі використовують наступні механізми.

- Рецензування: оцінювання і написання відгуку на книгу. Використовуються класи: Читач, Книга, Рецензія, Модератор.

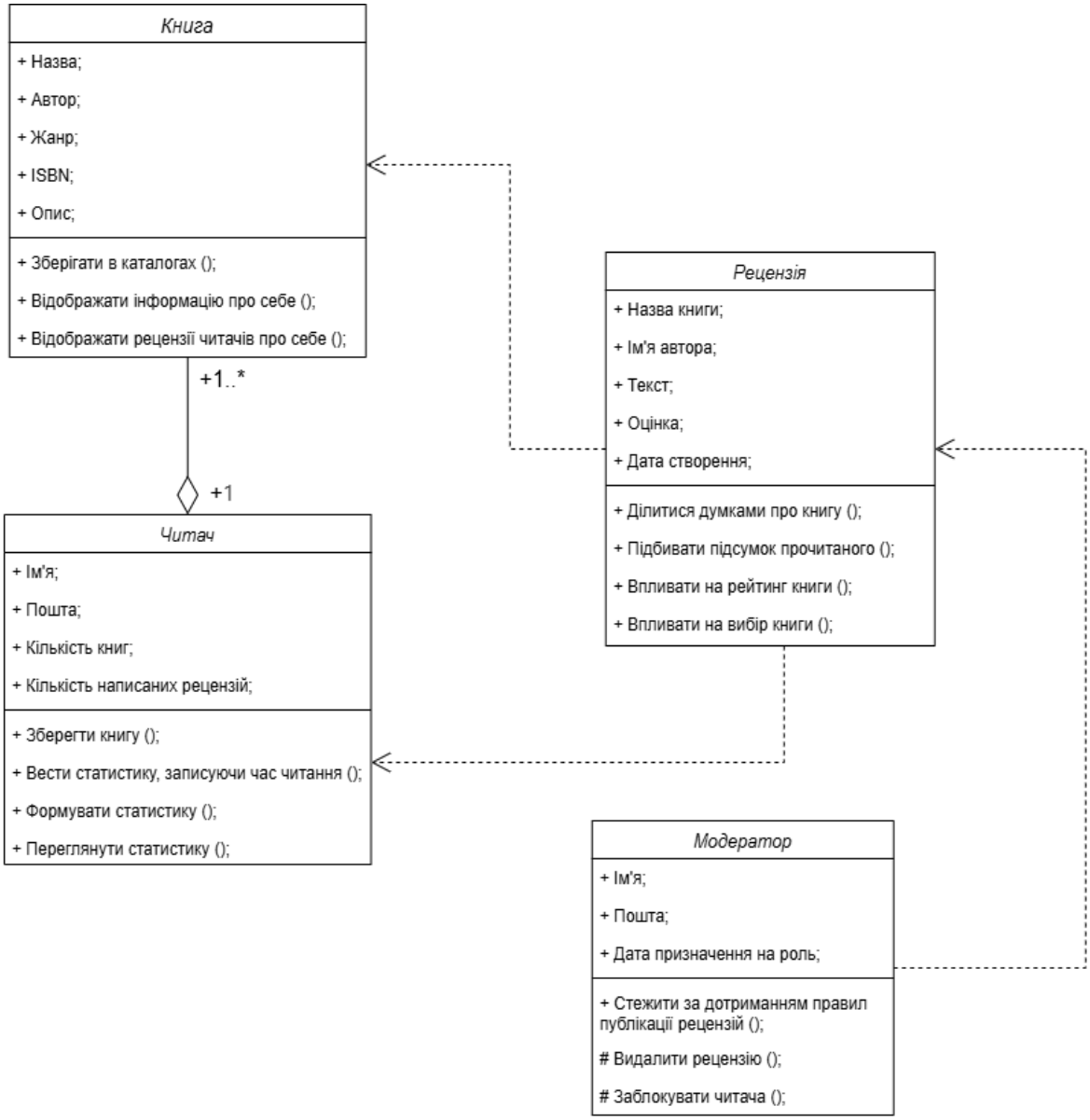


Рис. 10 Проста кооперація "Рецензування"

- Структуризація: організація чіткої структури, зберігаючи книжки у каталогах. Використовуються класи: Каталог, Книга, Читач, Працівник сховища.

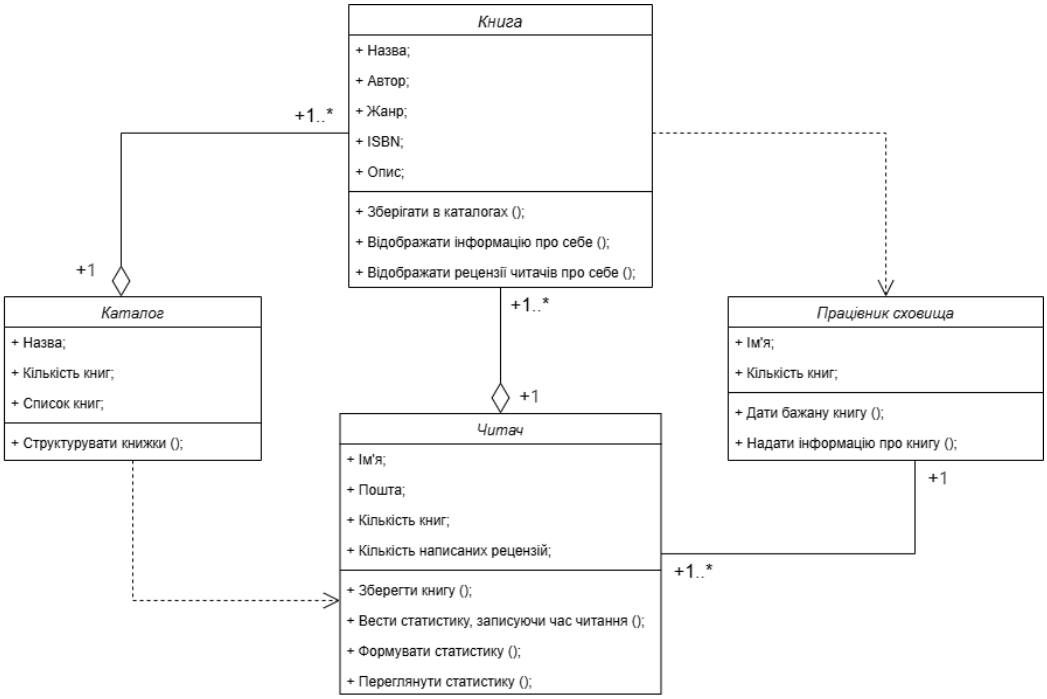


Рис. 11 Проста кооперація "Структуризація"

- Модерація: стеження за дотриманням правил публікації рецензій і приймання відповідних рішень щодо видалення рецензій або читачів. Використовуються класи: Модератор, Читач, Книга, Рецензія.

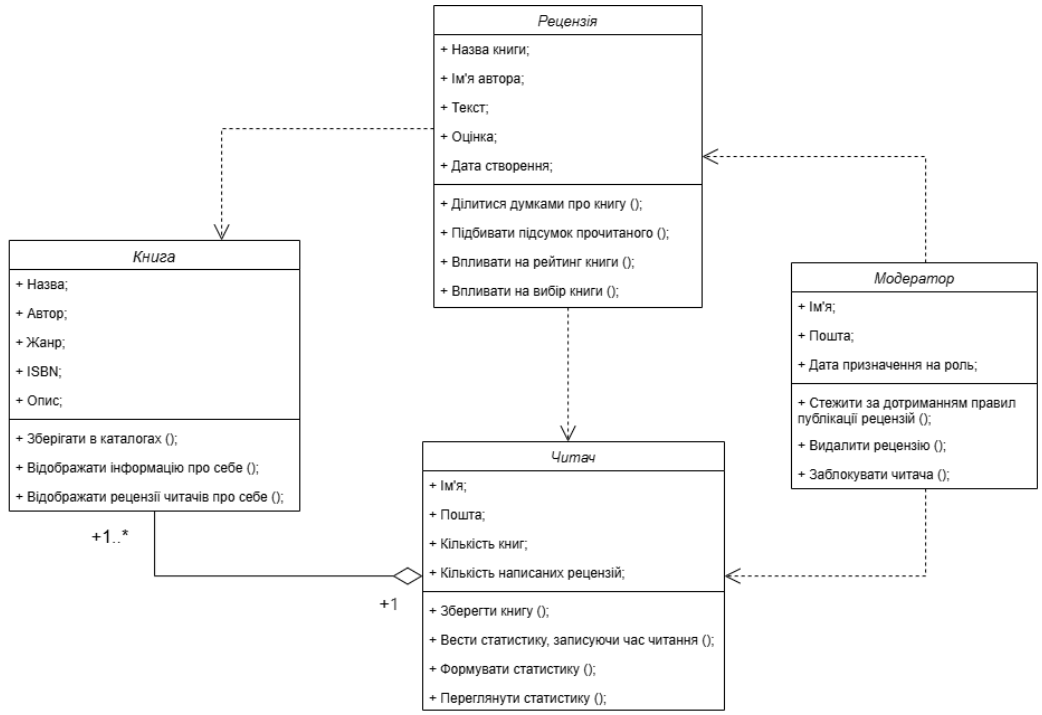


Рис. 12 Проста кооперація "Модерація"

Побудова ґрунтується на вже побудованій діаграмі класів із вже різними зв’язками (узагальнення, залежність, агрегування, композиція), котрі показують, як класи взаємодіють один з одним для моделювання механізмів, тобто поведінки частин модельованої системи, що з'являється в результаті взаємодії цих класів.

Діаграма компонентів.

Діаграма компонентів базується на тому, щоб відобразити частини системи як компонент, що мають зв’язки з іншими компонентами, що реалізують певну функціональну можливість системи. "Кожен компонент відповідає за одну чітку мету в рамках усієї системи та взаємодіє з іншими важливими елементами лише за умови необхідності" [24]. На рис. 13 представлено діаграму компонентів розроблювальної системи.

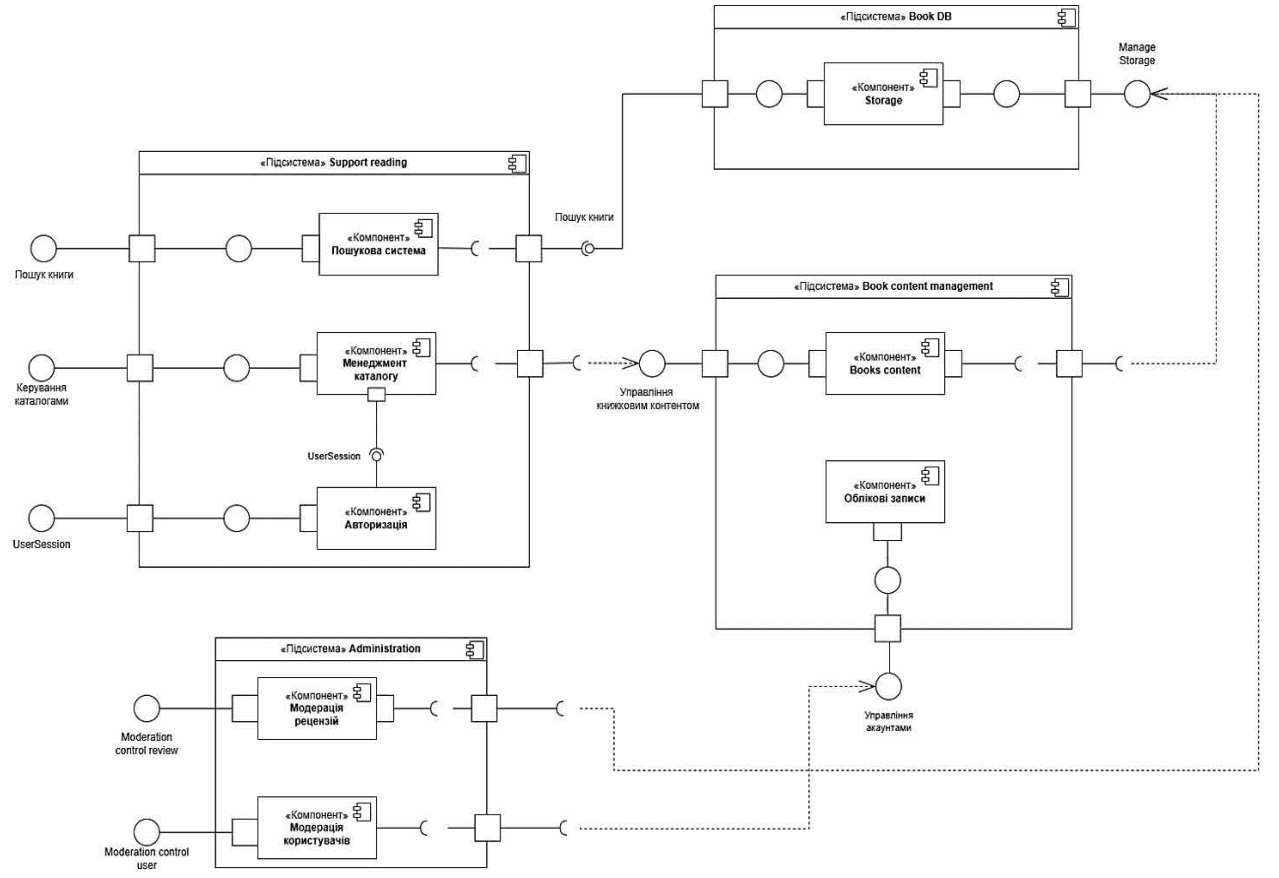


Рис 13 Діаграма компонентів

Складові діаграми компонентів системи.

- Підсистема Support reading містить 3 компоненти, які пов'язані із взаємодією користувача із системою - пошукову систему, менеджмент каталогу і авторизацію:

1. Компонент Пошукова системи дозволяє здійснювати пошук книг і використовує необхідний інтерфейс Пошук книги у Storage, який надається компонентом Storage з підсистеми BooksStorage і інтерфейс Доступ до книг у Google Books API.

2. Компонент Менеджмент каталогу використовує інтерфейс Управління книжковим контентом, наданий компонентом Books content, а також використовує інтерфейс UserSession компонента Authentication.

3. Компонент Авторизація надає інтерфейс UserSession, що дозволяє читачам створювати облікові записи, входити в систему або виходити з системи. Компонент надає інтерфейс UserSession, який дозволяє користувачам створювати сесії в системі.

- Підсистема Books Storage містить компонент Storage, який забезпечує 2 інтерфейси – Пошук книг у Storage і Manage Storage, які використовуються іншими підсистемами. Цей компонент надає інтерфейс Manage Storage для управління збереженням книжкового контенту (рецензії, нотатки тощо.) для підсистеми Book content managment і використовує інтерфейс Пошук книг у Storage.

- Підсистема Administration призначена для адміністративного управління та включає 2 компоненти: Модерація рецензій і Модерація користувачів.

1. Компонент Модерація рецензій надає інтерфейс Moderation control для здійснення модерації, а також використовує інтерфейс Manage Storage, що стосується адміністрування системи.

2. Компонент Модерація користувачів надає інтерфейс Moderation control user для модерування користувачів у системі, він використовує інтерфейс Управління акаунтами задля адміністративних завдань.

- Підсистема Book content managment призначена для управління контентом книг і користувачами та включає 3 компоненти: Books content і Облікові записи.

1. Компонент Books content відповідає за управління книжковим контентом у системі, дозволяючи можливість користувачам, переглядати та керувати контентом книжковим у системі. Компонент надає інтерфейс Управління книжковим контентом, і компонент залежить від інтерфейсу Manage Storage, що надається компонентом Storage з підсистеми BooksStorage для управління, збереженням книжкового контенту у каталог для подальших взаємодій.

2. Компонент Акаунти відповідає за управління обліковими записами користувачів у системі. Він надає інтерфейс Управління акаунтами, який забезпечує можливість створення, редагування, видалення та перегляду акаунтів користувачів.

4 ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЯ СИСТЕМИ

4.1 Тестування системи

Фінальним етапом у розробці ПЗ можна назвати тестування. Його задум полягає в тому, щоб здійснювати перевірки працездатності і правильної роботи задуманої системи згідно поставленим вимогам. Тестування дозволить виявити помилки ще до самого впровадження системи в експлуатацію, що є важливим моментом для забезпечення її надійності.

З розвитком технологій з'являлись дедалі більше видів тестування, основними з яких можна виділити:

1) функціональне тестування, яке зосереджене на розроблених функцій, згідно поставлених вимог до розробки системи;

2) нефункціональне тестування, яке на відміну від функціонального зосереджується не на розроблених функціях, а на зручності використання, стабільності роботи в різних браузерах, безпеці;

3) структурне тестування, яке зосереджене на структурі коду і логіки роботи всередині системи: покриття рішень, умов, операторів, шляхів, рядків коду [25];

4) тестування змін, який проводиться після внесених змін у коді, щоб перевірити чи не зламається функціонал вже справної системи;

5) тестування API, суть якого зосереджена на перевірці правильності обробки запитів від користувача, валідності відповідей, а також роботи зі статусами помилок.

Знаючи архітектуру розроблювальної системи, що складається з частин фронтенду і бекенду, який має поставлений зв'язок із БД, було проведено тестування інтерфейсу, а також обробки API запитів, що надсилаються до серверу.

Для тестування API запитів використано програмний продукт Postman, результат в якому наведено на рис. 14-17.

Приклади тестування в Postman для системи.

1. Підключення до БД: тестування полягає в тому, щоб перевірити налагоджений зв’язок із БД.

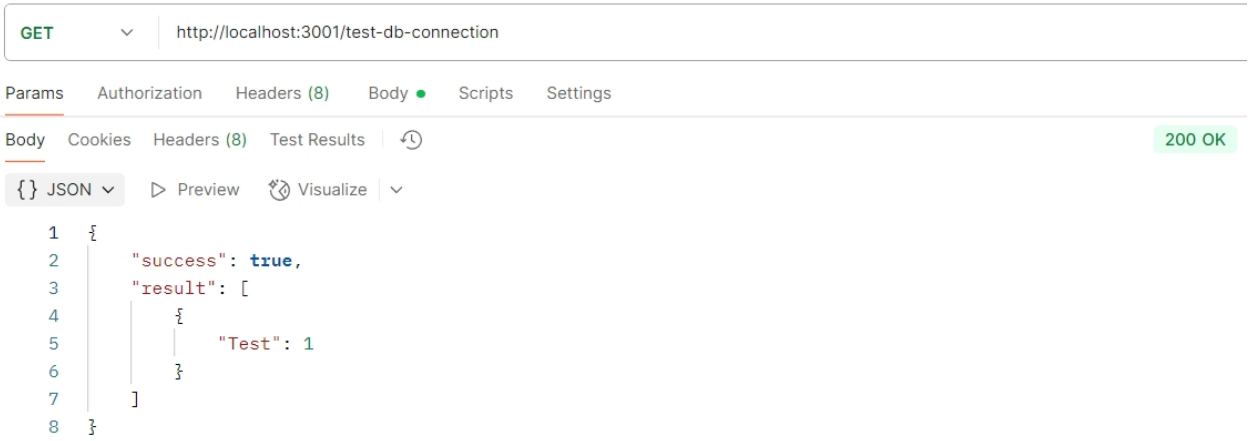


Рис 14 Результат тестування отримання даних про Каталоги

2. Отримання даних про всі каталоги: тестування полягає в отриманні всіх даних з таблиці BookCatalog БД. Для цього у файлі index.js на стороні сервера створено ендпоінт GET для шляху /catalog, який витягуватиме дані з таблиці БД і надаватиме їх у вигляді JSON завдяки Express.

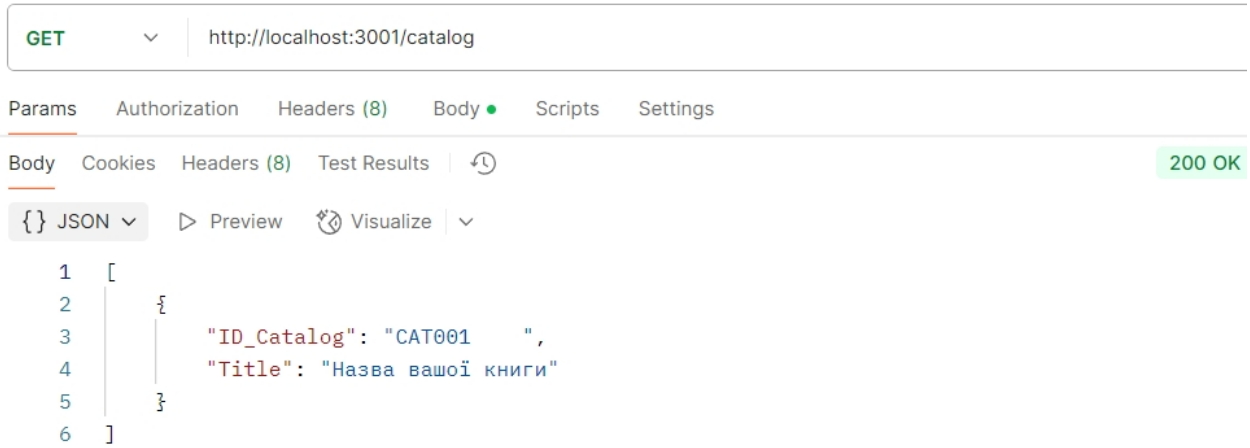


Рис 15 Результат тестування отримання даних про Каталоги

3. Зміна ролі користувача: у файлі userRoutes.js на стороні сервера створено ендпоінт PUT для шляху /api/readers/:id/role, який приймає ID користувача як параметр та нову роль у тілі запиту. Тут реалізована логіка перевірки користувача на існування та оновлення його ролі у БД.

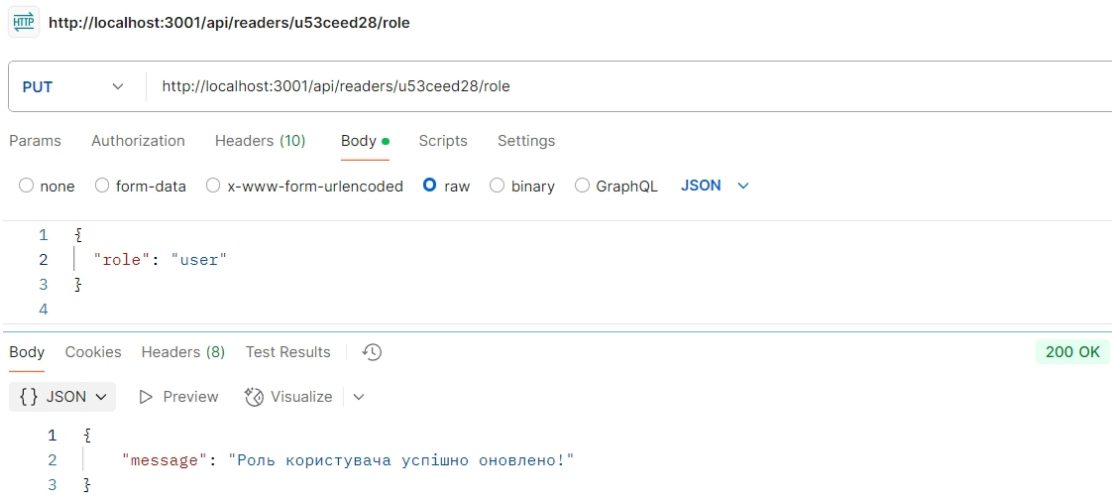


Рис 16 Результат тестування успішної зміни ролі

4. Авторизація: у файлі index.js на стороні сервера створено ендпоінт POST для шляху /login, який приймає email та пароль у тілі запиту. Як результат, створюється JWT токен після входу в систему вже зареєстрованого користувача.

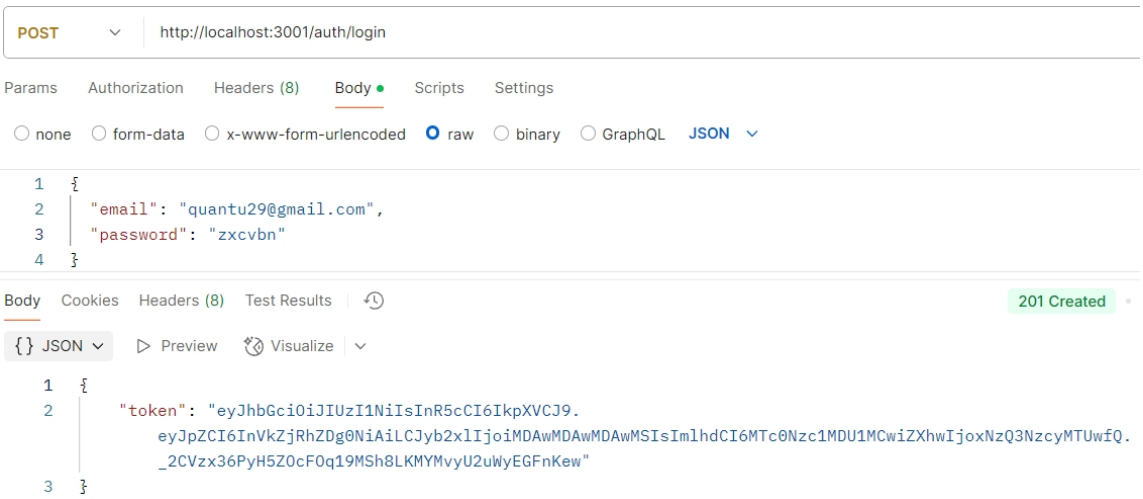


Рис 17 Результат тестування успішного входу в систему

4.2 Діаграма розгортання

Діаграма розгортання – це тип діаграми для показу конфігурації вузлів обробки часу виконання та компонентів, що на них працюють [26]. Тобто загалом, це схема системи для користувача, що дозволяє зобразити фізичну структуру системи.

Елементи діаграми розгортання.

1. Вузол – це фізично існуючий елемент в системі, наприклад: смартфон, ноутбук, вебсервер, сервер БД, а також вузлом може бути середовище виконання. Вони здатні виконувати програмні компоненти (артефакти).

2. Артефакт – це програмний компонент, що виконується у вузлі. Наприклад, файл з певним кодом.

3. Зв’язки – це лінії, що позначають як взаємодіють вузли чи елементи. На них зазвичай передається певна інформація: SQL запити, API запити, дані від користувача тощо.

Побудова діаграми розгортання для розроблювальної системи представлено на рис. 18.

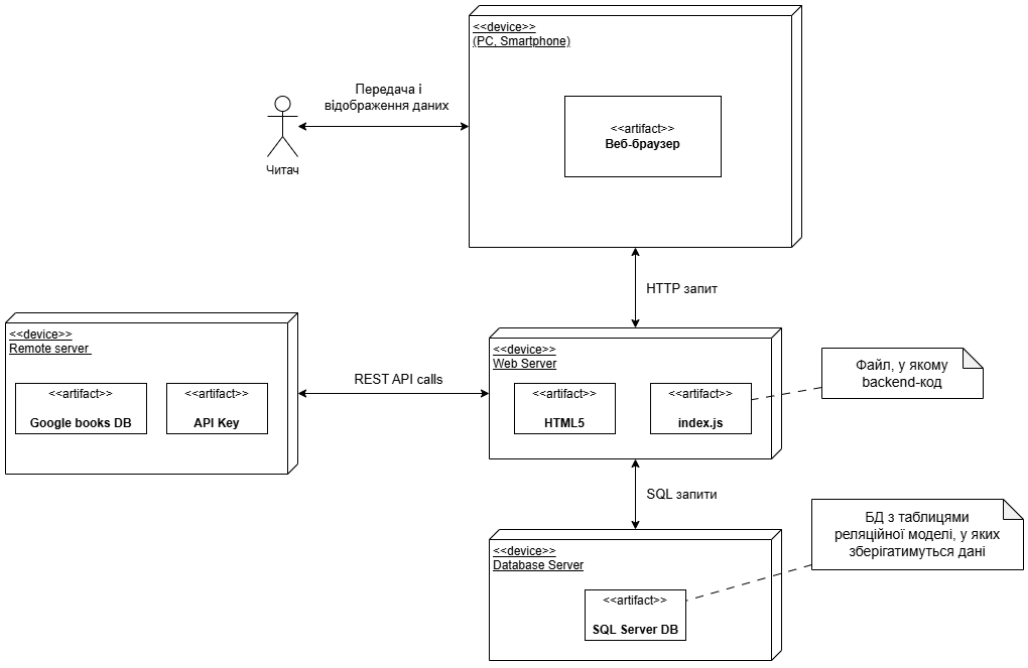


Рис. 18 Діаграма розгортання

Серверна частина використовується для обробки запитів від користувача завдяки браузеру, взаємодії з БД і з іншими серверами, обробки вводу користувача та структурування веб-застосунків. Клієнтом в системі виступає читач, який користується веб-браузером для взаємодії із системою через інтерфейс.

Опис побудованої діаграми розгортання.

- Вузли з артефактами:

1. PC, Smartphone: вузол представляє апаратний пристрій комп'ютера чи смартфона, які підтримують середовище виконання системи. Він містить артефакт Веб-браузер з яким читач взаємодіє.

2. Web Server: сервер на якому розміщується бекенд-частина. Він відповідає за обробку запитів від клієнтів (браузерів), взаємодію з базою даних та зовнішнім сервером (Google Books API), що надаватиме свої сервіси. Вузол містить артефакт Server.js, що відповідатиме за бекенд-частину системи, де відбуватиметься бізнес-логіка, для якої може знадобитися доступ до бази даних і Google Books API. Також є артефакт HTML5 у якому є інтерфейс сторінки.

3. Database Server: вузол, у якому міститься реляційна база даних. Він відповідає за зберігання, обробку та видачу даних. У ньому міститься артефакт SQL Server DB, який є базою даних у SQL Server.

4. Remote server: вузол, який є зовнішнім сервером для пошуку книг і інформації на них від Google Books. У вузлі міститься артефакт API Key для доступу до Google Books API і База даних Google Books, де зберігаються метадані про книги від самого Google.

- Зв'язки:

1. HTTP/HTTPS: зв'язок клієнта з веб-сервером через браузер.

2. SQL запити: зв'язок, де WebServer підключається до БД SQL Server для збереження/зчитування даних, і для цього він використовуватиме SQL запити.

3. API calls: зв'язок, де WebServer робить запити до Google Books API для пошуку книг.

Як підсумок, побудована діаграма розгортання представило загальну фізичну структуру системи автоматизованого супроводження читання книг, яке дає розуміння звичайному користувачу як усе працює всередині.

4.3 Вимоги апаратного і програмного забезпечення

"Вимоги до апаратного та програмного забезпечення стосуються мінімальних технічних характеристик, необхідних для належного функціонування системи, програми чи служби. Ці вимоги забезпечують сумісність, продуктивність та безпеку під час використання програмного забезпечення або роботи системи" [27].

Визначено апаратні і програмні вимоги для користувачів системи, які представлені в таблицях 9-10.

Таблиця 9

Апаратні вимоги з точки зору користувача

Компонент	**Мінімальні вимоги**
Процесор	Intel Core i3 або аналогічний
Оперативна пам’ять	Від 4 ГБ
Відеоадаптер	Будь-який
Вільне місце на диску	Від 500 МБ (для кешу браузера)
Мережеве з’єднання	Стабільне підключення до інтернету
Екран	Будь-яка роздільна здатність

Таблиця 10

Програмні вимоги з точки зору користувача

Компонент	**Опис**
Браузер	Сучасний браузер
ОС	Будь-яка, але краще Windows
Мережеве з’єднання	Стабільне підключення до інтернету

Визначено апаратні і програмні вимоги для розробника системи, які представлені в таблицях 11-12.

Таблиця 11

Апаратні вимоги з точки зору розробника

Компонент	**Мінімальні вимоги**
Процесор	Будь-який процесор 4-ядерний з 2.4 ГГЦ або вище
Оперативна пам’ять	Від 8 ГБ і більше
Відеоадаптер	Будь-який, що підтримує сучасні браузери
SSD	250 ГБ
Мережеве з’єднання	Постійне підключення до інтернету
Екран	Будь-яка роздільна здатність

Таблиця 12

Програмні вимоги з точки зору розробника

Програмне забезпечення	**Опис**
Браузер	Сучасний браузер (Google Chrome, Firefox, Opera)
ОС	Windows 10 або 11
Мережеве з’єднання	Стабільне підключення до інтернету
Node.js	Версія v18.x або новіша версія
SQL Server	SQL Server 2019 або новіша версія

Поставлені програмні вимоги до власника системи мають дати не лише запуск і доступність бекенд частини системи автоматизованого супроводження читання книг, а й майбутню підтримку, спостереження і оновлення, щоб гарантувати те, що робота всієї системи буде плавною і без недоліків.

4.4 Інсталяційний пакет

Після кропіткої, багатоетапної розробки та тестування, важливим і правильним кроком є розгортання самої системи із врахуванням наявності серверної і клієнтської частин. Цей крок дає можливість серверного розгортання та включає необхідні компоненти для запуску як клієнтської, так і серверної частин, які наявні якраз у розроблювальній системі. Продумування процесу розгортання формувалося з урахуванням використаних технологій під час реалізації системи, зокрема React з Vite та Node.js з Express. Важливо те, щоб серверна частина була запущена раніше, аніж клієнтська, бо саме вона забезпечує обробку запитів, з'єднання з БД та підтягування необхідних даних.

Склад інсталяційного пакету.

- Папка client/ – клієнтська частина, реалізована з використанням React;
- Папка server/ – серверна частина, реалізована на Node.js з використанням Express.
- Файл README.md у кожній папці – текстовий файл з інструкцією встановлення та запуску.
- Файли конфігурації (.env) для налаштування змінних середовища.
- Файл package.json із залежностями у кожній папці.
- SQL скрипт із запитами для створення структури БД.
- Папка dist, який містить збірку вихідного коду клієнтської частини сайту для розгортування.

У підсумку склад інсталяційного пакету для обох частин включає не лише програмний код, а й усі необхідні конфігураційні файли, що дають змогу швидко розгорнути систему та забезпечити її стабільну роботу.

4.5 Опис роботи системи

Робота із ПЗ системи автоматизованого супроводження читання книг бере початок зі сторінки на якій за допомогою CSS розміщено одночасно реєстрація і вхід, що представлено на рис. 19 і 20.

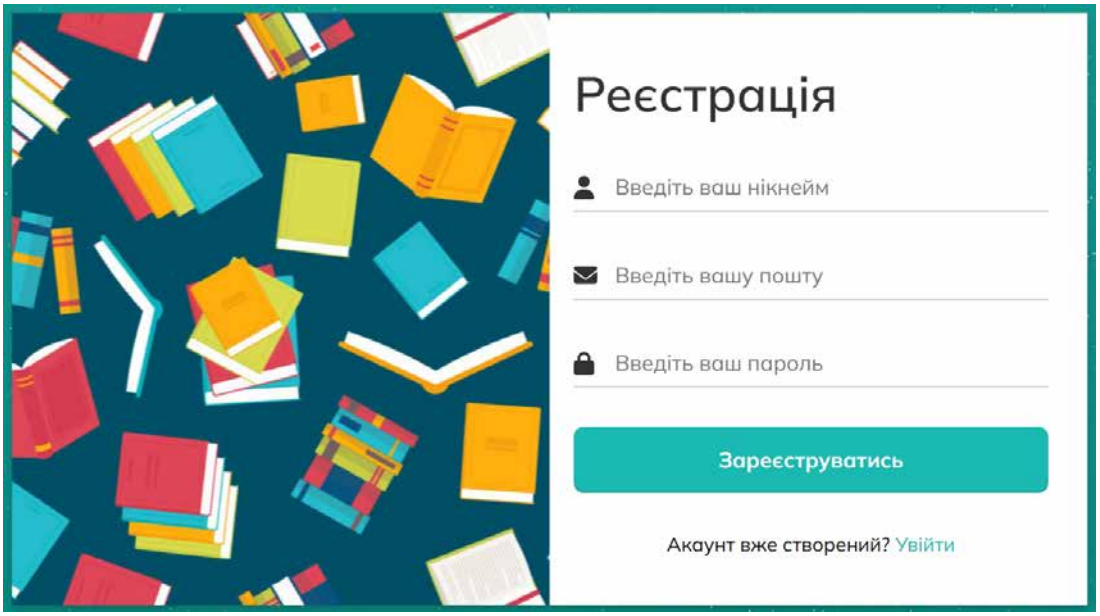


Рис. 19 Сторінка з формою Реєстрації

Кожен новий зареєстрований користувач набуває роль "user" за замовчуванням. Після реєстрації можна почати використовувати можливості створеної системи, але перед цим потрібно увійти. В залежності від типу користувача спрацює захищена маршрутизація, котра перекине на сторінки, які відповідають чинній ролі в системі.

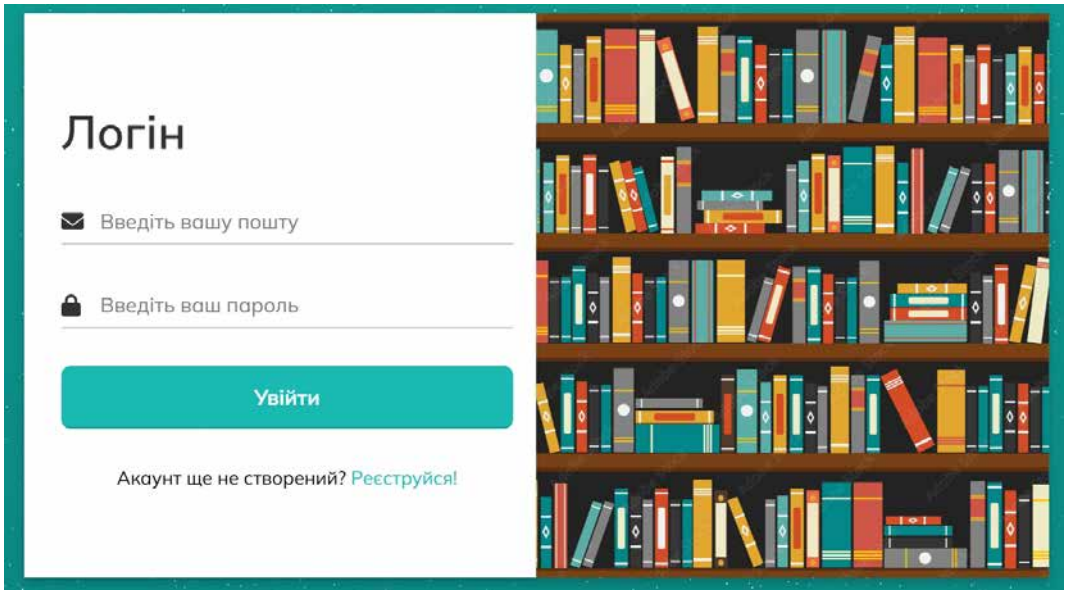


Рис. 20 Сторінка з формою Входу

Користувач з роллю "admin" перейде до свої відповідних сторінок, де він виконуватиме дії у ролі адміністратора. На рис. 21 представлено сторінку адміністратора із списком читачів та діями: видалити або змінити роль.

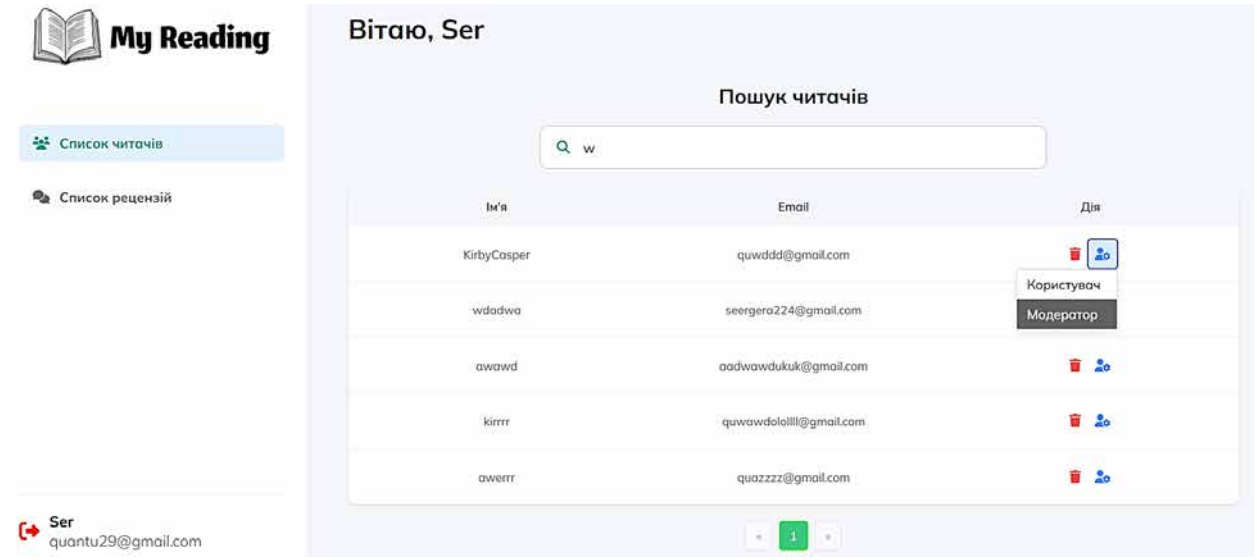


Рис. 21 Сторінка із списком читачів

Перехід на сторінку із списком усіх опублікованих рецензій відбувається при натисканні в бічному меню на кнопку "Список рецензій". На рис. 22 представлена сторінка із усіма рецензіями, які у разі порушення можна видалити.

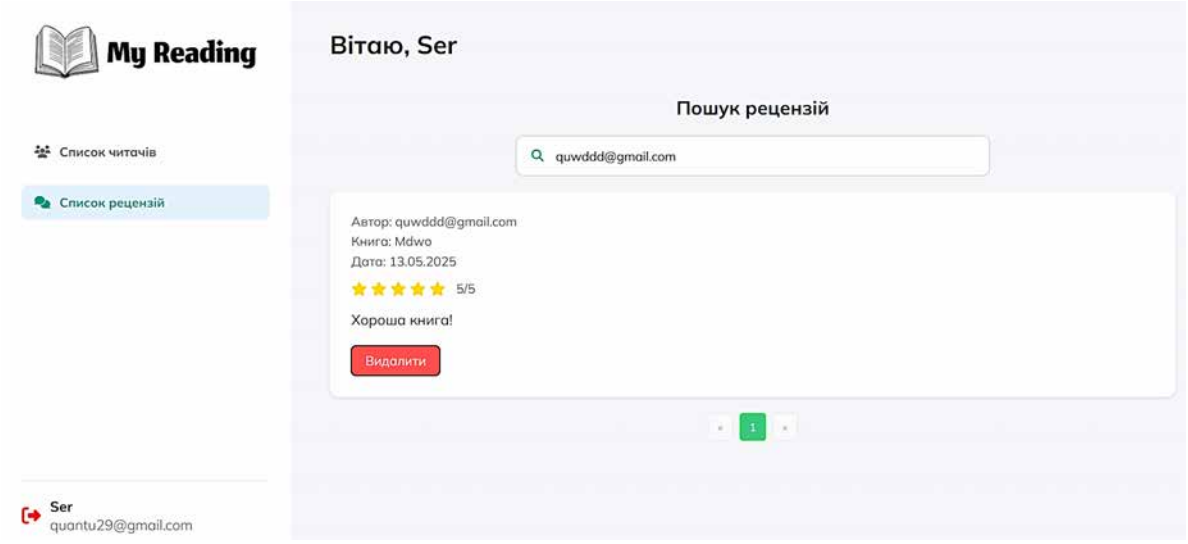


Рис. 22 Сторінка із списком рецензій

Звичайний користувач із роллю "user" перейде до своїх сторінок, одна з яких – домашня сторінка, яка зображена на рис. 23. На ній показуються усі збережені книги, а також створенні каталоги, що також містять книги.

Рис. 23 Сторінка читача з його книгами і каталогами

Перехід на сторінку пошуку книг здійсниться після натискання в бічному меню на кнопку – "Пошук книг". На рис. 24 представлено сторінку пошуку книги за певними критеріями: назва, автор, ISBN.

Рис. 24 Сторінка пошуку книг

Далі з’являється можливість дізнатися детальну інформацію за знайденою книгою, і для цього потрібно натиснути на картку книги. На рис. 25 представлено сторінку з детальною інформацією за знайденою книгою, а також усі опубліковані рецензії для неї.

Рис. 25 Сторінка з інформацією про книгу

Книжку можна зберегти в каталог чи загальний список, щоб потім виконати супроводжене читання. Щоб це зробити треба на домашній сорінці у картці книги натиснути "Відстежити читання", де як результат відбудеться перехід на сторінку супроводженого читання, що зображено на рис. 26.

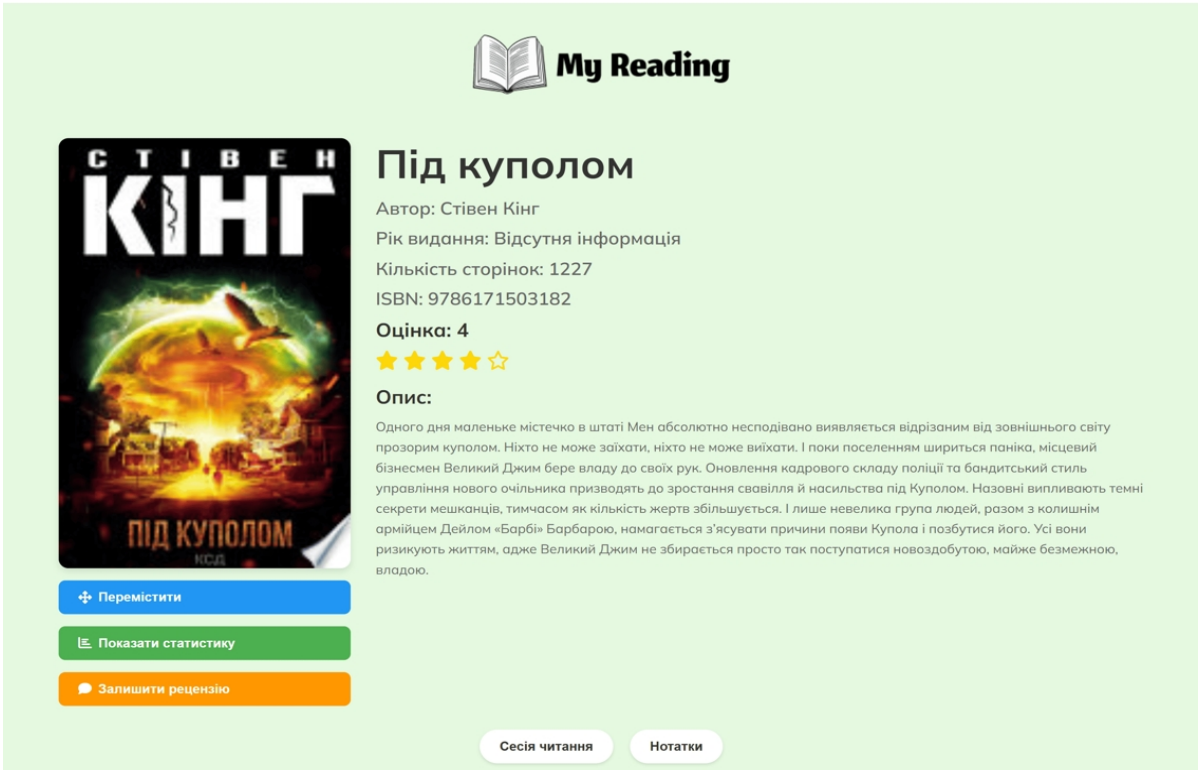


Рис. 26 Сторінка для супроводженого читання

Щоб почати супроводжене читання треба натиснути на кнопку "Сесія читання", де з'явиться знизу відповідний інтерфейс для створення сесій. На рис. 27 показано інтерфейс створення сесій читання.

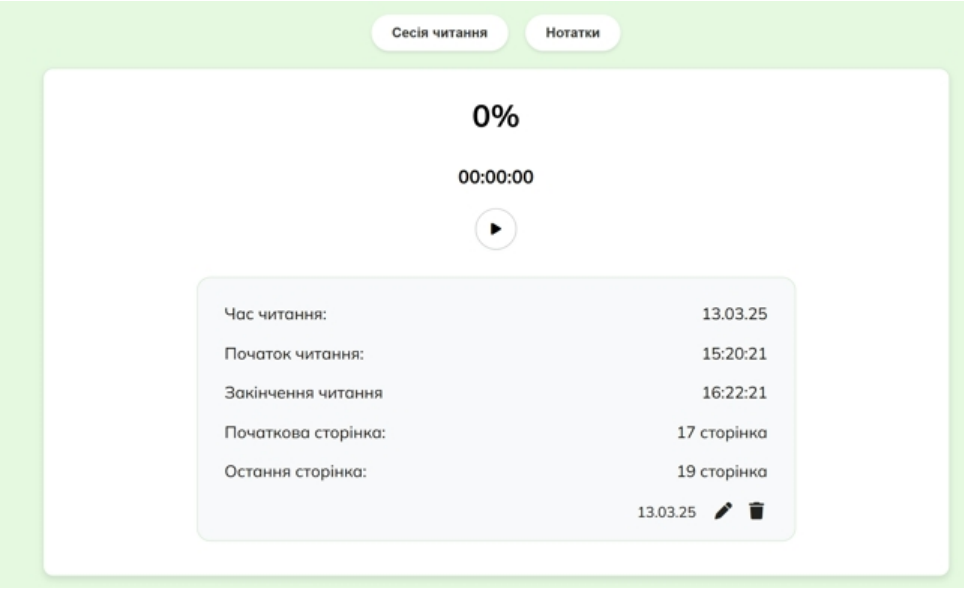


Рис. 27 Інтерфейс створення сесії читання

Щоб залишити нотатку, треба натиснути на кнопку "Нотатки" для появи відповідного інтерфейсу для здійснення такої операції. На рис. 28 показано інтерфейс створення нотаток.

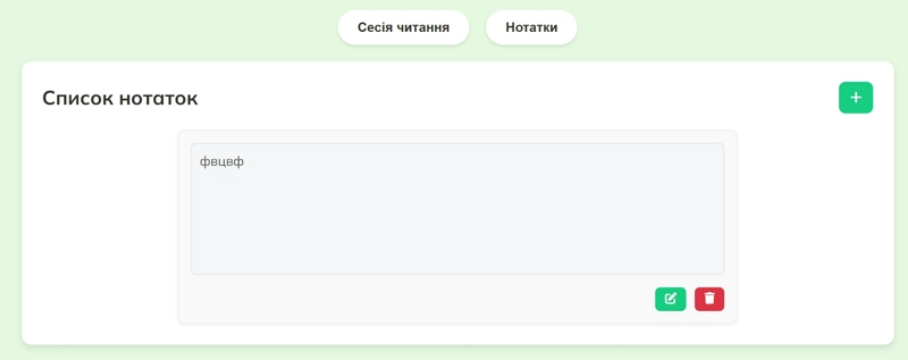


Рис. 28 Інтерфейс створення нотаток

Після прочитання книги надається можливість залишити рецензію або побачити автоматично сформовану статистику до прочитаної книги через модульні вікна. На рис. 28 і 29 представлено модульні вікна статистики і написання рецензії.

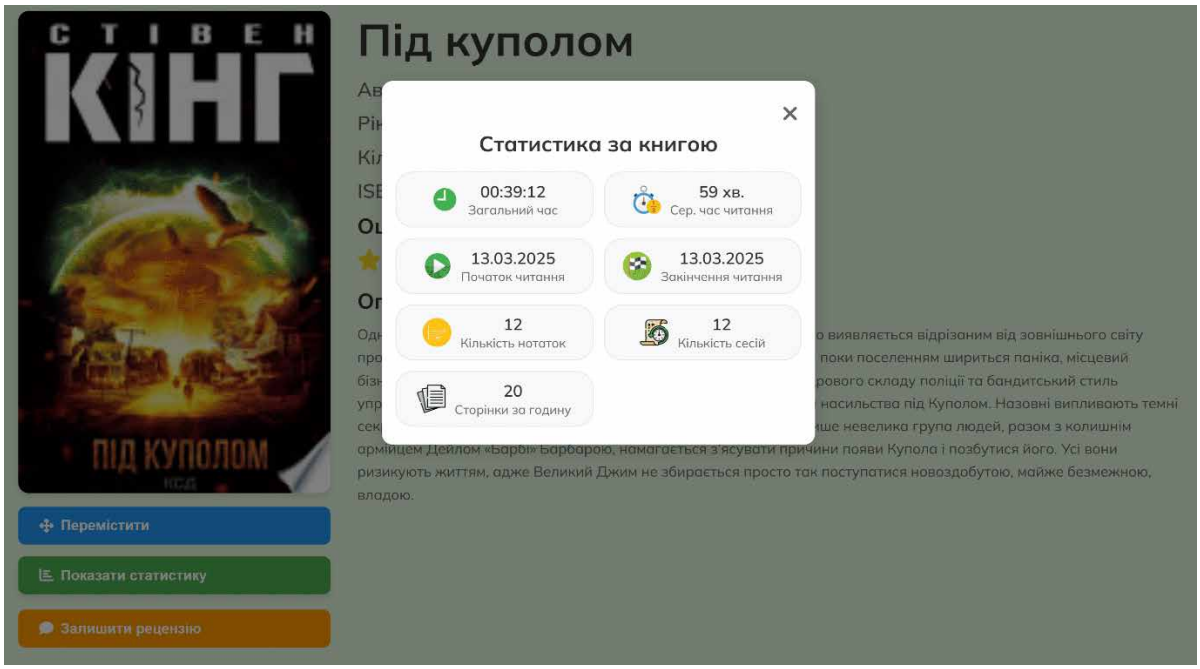


Рис. 29 Модульне вікно показу статистики

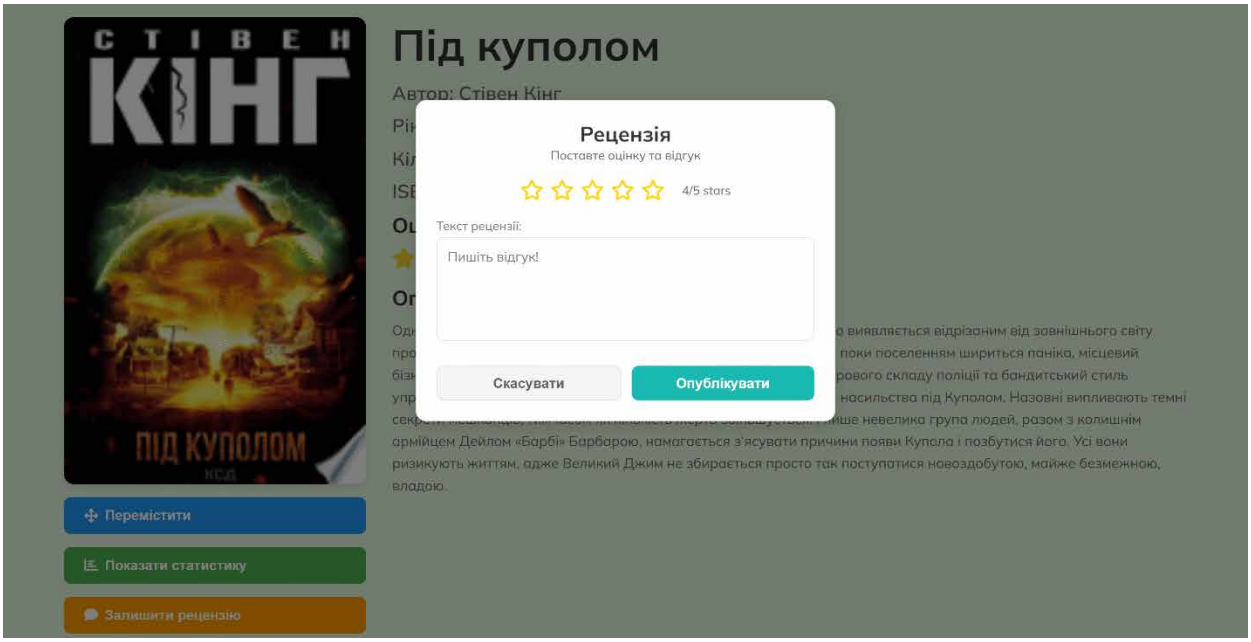


Рис. 30 Модульне вікно створення рецензії

ВИСНОВКИ

У межах постановки завдання бакалаврської кваліфікаційної роботи було успішно спроєктовано та реалізовано програмне забезпечення системи автоматизованого супроводження читання книг, що надає можливість читачам вести організований супровід під час читання книжок. Розробка відбувалася поетапно: аналіз предметної області, проєктування бази даних, проєктування та розробка програмного забезпечення, а також впровадження реалізованого програмного забезпечення.

Насамперед проведено етап детального аналізу предметної області, у якій виявлено основні процеси, котрі можна автоматизувати: пошук книг, зберігання знайдених книг, вираховування статистики до прочитаної книги, рецензування, а також інші важливі процеси для зручного супроводу під час читання книг. В основу аналізу також лягло дослідження існуючих рішень, моделювання предметної області за допомогою UML діаграм, що дали чітке уявлення про роботу предметної області. Після аналізу написано постановку завдання для розробки майбутнього ПЗ системи.

Приділено особлива увага розробці логічної моделі даних у вигляді ER-діаграми, щоб продемонструвати, які на вигляд взаємозв’язки між визначеними сутностями та їх атрибути для вибору згодом системи управління бази даних. Вибрано SQL Server, у якій сформовано SQL запити створення БД для зберігання даних, а також об’єкти в ній – тригери та уявлення.

Перед розробкою фронтенд і бекенд частин обрано середовище виконання Visual Studio Code, який забезпечив зручну та гнучку розробку завдяки також вбудованій системі контролю версій Git. Для самої розробки системи ретельно обрано сучасні технології: React, Node.js з веб-фреймворком Express, що забезпечили створення гарного, адаптивного та інтерактивного інтерфейсу для читачів.

Фінальний етап реалізації системи супроводжувався тестуванням усіх поставлених функцій перед самим розгортанням для реального користування.

Проведене тестування підтвердило працездатність системи, зокрема у Postman за допомогою якого було перевірено виконання HTTP запитів, а також поведінку системи у випадку неправильно реалізованих функцій з чітко визначеної постановки завдання.

Таким чином, мета у створенні функціонального та ефективного рішення для автоматизованого супроводження читання книг була успішно досягнута, через що розроблене рішення готове до використання. У підсумку розроблена система має свої перспективи для майбутнього розвитку, зокрема у сферах видавничої діяльності, особливо в навчальному процесі.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Типи діаграм UML [Електронний ресурс] Спосіб доступу – URL: https://www.lucidchart.com/blog/types-of-UML-diagrams (дата звернення: 07.05.2025).

2. Дідковська М. В. Тестування програмного забезпечення. Лекція 6 [Електронний ресурс] Спосіб доступу – URL: https://surl.lu/sgqicn (дата звернення: 08.05.2025).

3. Як будувати UML-діаграми. Розбираємо три найпопулярніші варіанти [Електронний ресурс] Спосіб доступу – URL: https://dou.ua/forums/topic/40575/ (дата звернення: 09.05.2025).

4. Зосим М. Діаграма послідовності (Sequence Diagrams) [Електронний ресурс] Спосіб доступу – URL: https://www.maxzosim.com/sequence-diagrams/ (дата звернення: 09.05.2025).

5. ER-діаграми: навчальний посібник [Електронний ресурс] Спосіб доступу – URL: https://www.guru99.com/uk/er-diagram-tutorial-dbms.html (дата звернення: 10.05.2025).

6. Erwin Data Modeler: особливості використання [Електронний ресурс] Спосіб доступу – URL: https://surl.li/fxbwsu (дата звернення: 10.05.2025).

7. Рейтинг систем керування базами даних [Електронний ресурс] Спосіб доступу – URL: https://db-engines.com/en/ranking (дата звернення: 10.05.2025).

8. Що таке MySQL [Електронний ресурс] Спосіб доступу – URL: https://www.oracle.com/mysql/what-is-mysql/ (дата звернення: 10.05.2025).

9. Управління базами даних: приклади [Електронний ресурс] Спосіб доступу – URL: https://surl.li/usevuh (дата звернення: 10.05.2025).

10. Порівняння MySQL та SQL Server [Електронний ресурс] Спосіб доступу – URL: https://www.astera.com/knowledge-center/mysql-sql-server/ (дата звернення: 10.05.2025).

11. SQL constraints: ключові обмеження [Електронний ресурс] Спосіб доступу – URL: https://codedamn.com/news/databases/sql-constraints-primary-key-foreign-key-not-null-unique-check (дата звернення: 10.05.2025).

12. Нормалізація баз даних [Електронний ресурс] Спосіб доступу – URL: https://greenhouse.cv.ua/?p=697 (дата звернення: 10.05.2025).

13. MySQL ON DELETE CASCADE [Електронний ресурс] Спосіб доступу – URL: https://www.geeksforgeeks.org/mysql-on-delete-cascade-constraint/ (дата звернення: 10.05.2025).

14. Уявлення в SQL [Електронний ресурс] Спосіб доступу – URL: https://javascript.org.ua/ostannya-stattya-po-sql-uyavlennya-v-sql/ (дата звернення: 11.05.2025)

15. CREATE VIEW у Transact-SQL [Електронний ресурс] Спосіб доступу – URL: https://surli.cc/yhwbyc (дата звернення: 11.05.2025).

16. Що таке front-end розробка [Електронний ресурс] Спосіб доступу – URL: https://wezom.com.ua/ua/blog/chto-takoe-front-end-razrabotka (дата звернення: 13.05.2025).

17. Vite: офіційна документація [Електронний ресурс] Спосіб доступу – URL: https://vite.dev/guide/ (дата звернення: 13.05.2025).

18. Що таке REST API [Електронний ресурс] Спосіб доступу – URL: https://foxminded.ua/shcho-take-rest-api/ (дата звернення: 13.05.2025).

19. Що таке Axios [Електронний ресурс] Спосіб доступу – URL: https://www.geeksforgeeks.org/what-is-axios/ (дата звернення: 14.05.2025).

20. Чим відрізняється фронтенд та бекенд [Електронний ресурс] Спосіб доступу – URL: https://frontend.lviv.ua/chym-vidriznyayetsya-frontend-ta-backend-rozrobka-shho-obraty (дата звернення: 14.05.2025).

21. Все про Node.js [Електронний ресурс] Спосіб доступу – URL: https://wezom.com.ua/ua/blog/vse-chto-nuzhno-znat-o-nodejs (дата звернення: 14.05.2025).

22. Express.js: огляд [Електронний ресурс] Спосіб доступу – URL: https://foxminded.ua/express-js/ (дата звернення: 14.05.2025).

23. Інтегроване середовище розробки [Електронний ресурс] Спосіб доступу – URL: https://w3schoolsua.github.io/hyperskill/ide.html#gsc.tab=0 (дата звернення: 17.05.2025).

24. Що таке компонентна діаграма [Електронний ресурс] Спосіб доступу – URL: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-component-diagram/ (дата звернення: 17.05.2025).

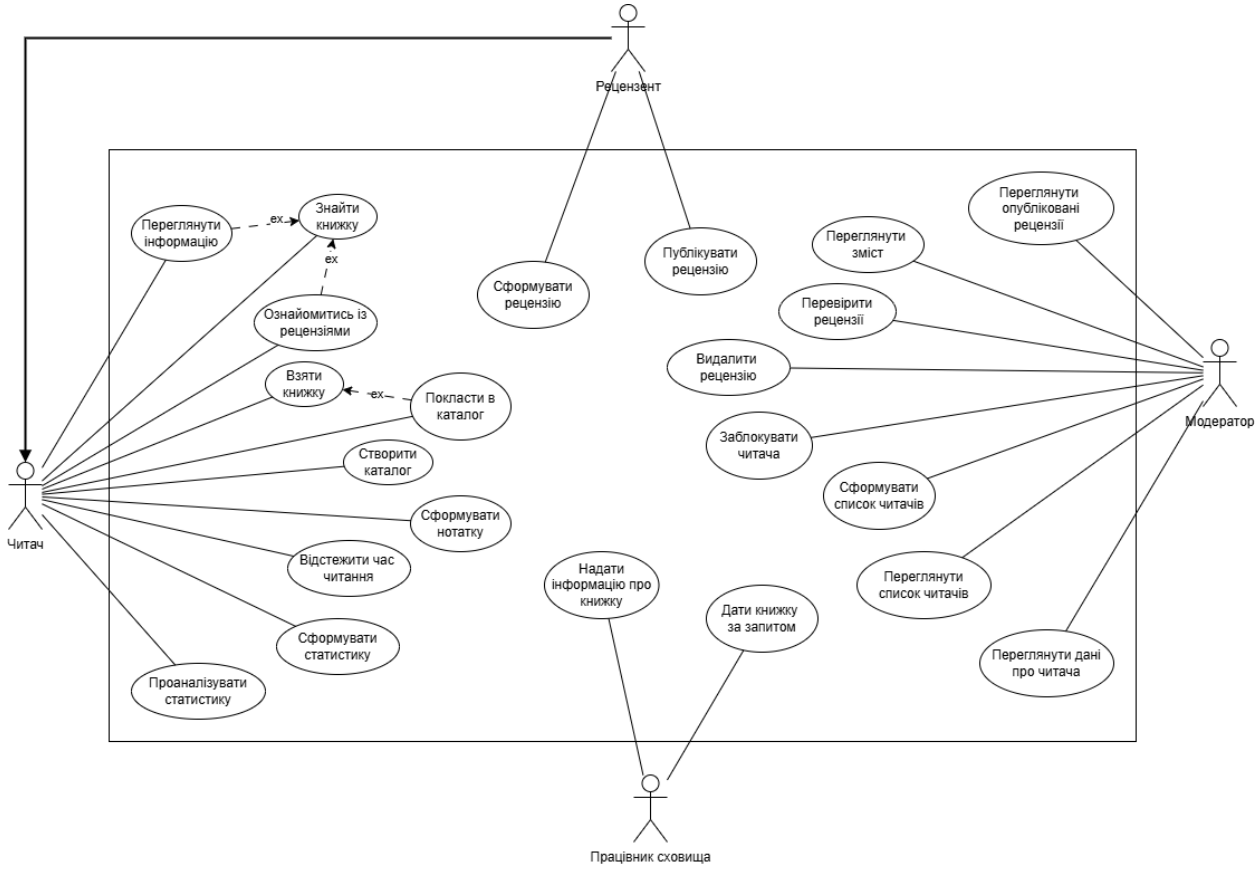
25. Огляд типів тестування [Електронний ресурс] Спосіб доступу – URL: https://training.qatestlab.com/blog/technical-articles/review-the-types-of-testing/ (дата звернення: 20.05.2025).

26. Що таке діаграма розгортання [Електронний ресурс] Спосіб доступу – URL: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-deployment-diagram/ (дата звернення: 20.05.2025).

27. Вимоги до апаратного та програмного забезпечення [Електронний ресурс] Спосіб доступу – URL: https://surl.li/rpgngn (дата звернення: 20.05.2025).

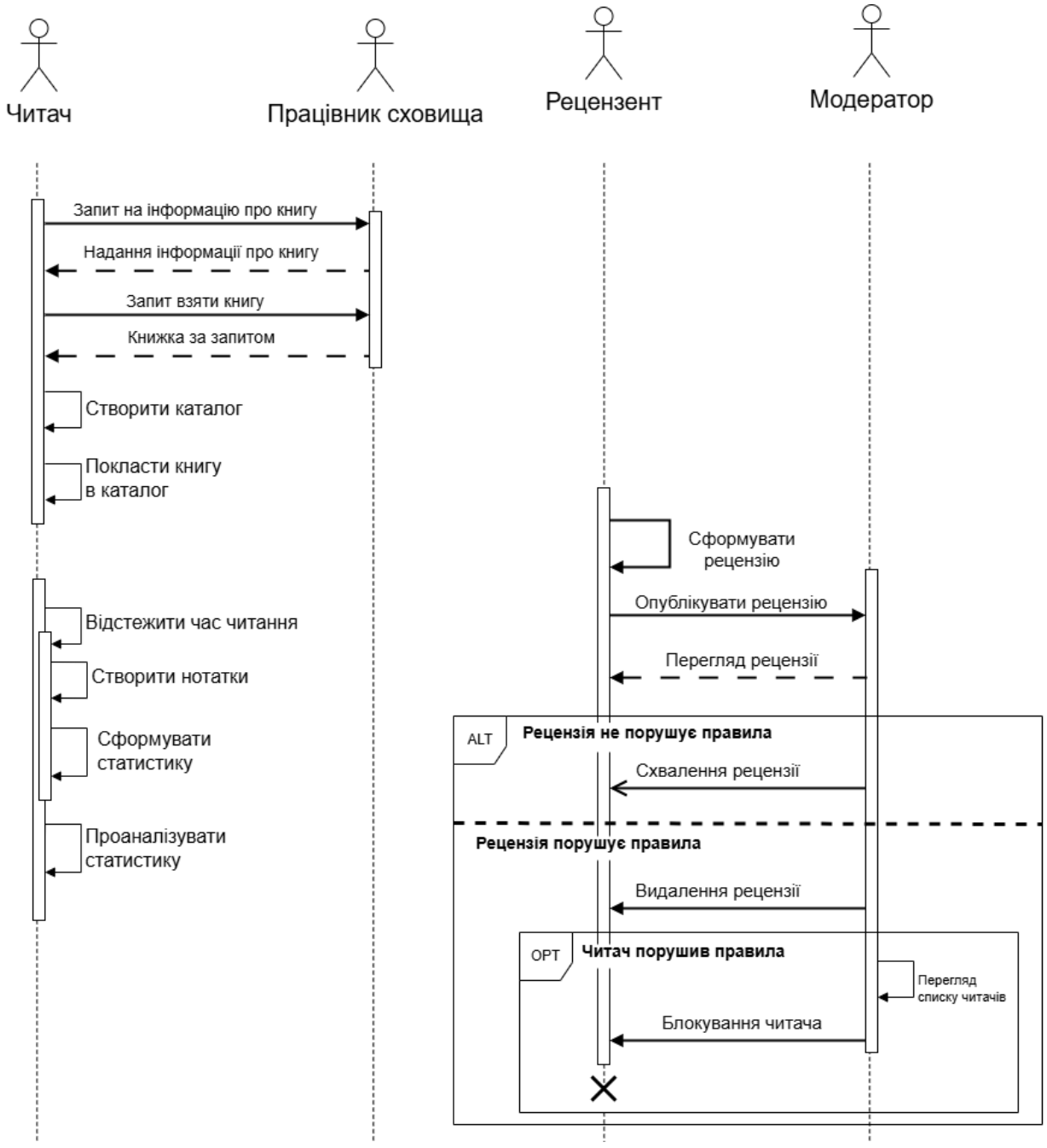
ДОДАТОК А

Діаграма прецедентів



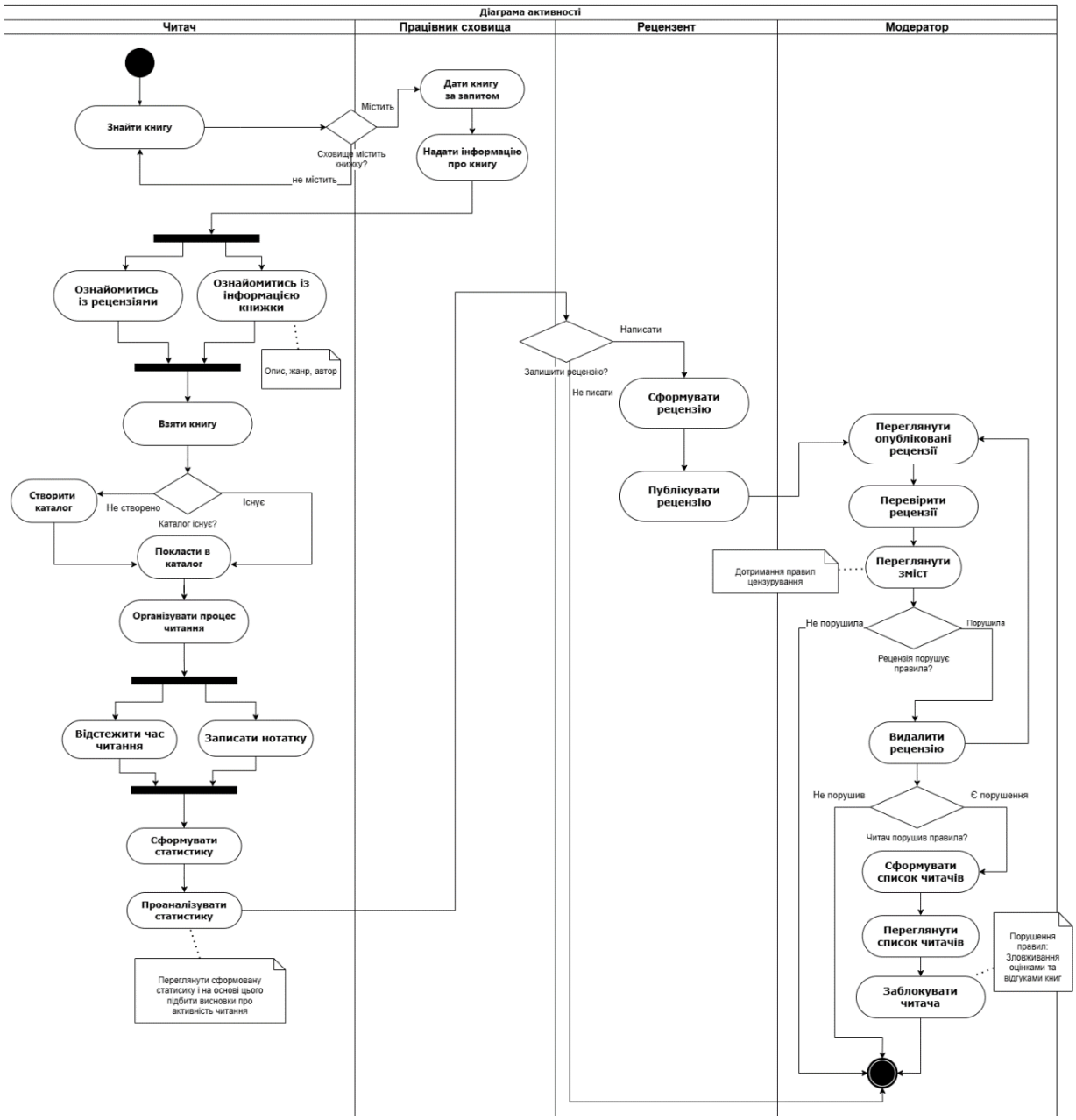
ДОДАТОК Б

Діаграма послідовності



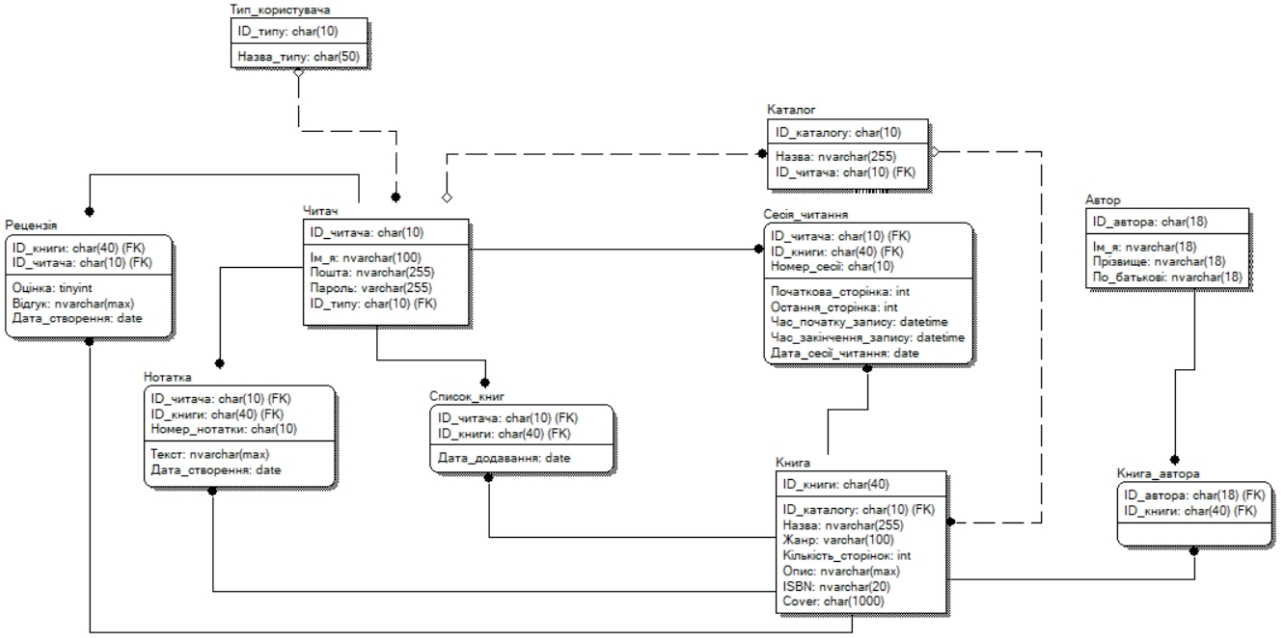
ДОДАТОК В

Діаграма активності



ДОДАТОК Д

ER-діаграма фізичного рівня



ДОДАТОК Е

SQL запити створення таблиць, тригерів і уявлення в БД

```
DROP TABLE IF EXISTS ReadingList;
DROP TABLE IF EXISTS ReadingSession;
DROP TABLE IF EXISTS Note;
DROP TABLE IF EXISTS Review;
DROP TABLE IF EXISTS Book_Author;
DROP TABLE IF EXISTS Book;
DROP TABLE IF EXISTS BookCatalog;
DROP TABLE IF EXISTS Author;
DROP TABLE IF EXISTS Reader;
DROP TABLE IF EXISTS TypeUser;
CREATE TABLE TypeUser (
   ID_Type char(10) PRIMARY KEY,
   TypeName NVARCHAR(50) NOT NULL
);
CREATE TABLE Reader (
   ID_Reader CHAR(10) PRIMARY KEY,
   Nickname NVARCHAR(100) NOT NULL,
   Email NVARCHAR(255) UNIQUE NOT NULL,
   MyPassword NVARCHAR(255) NOT NULL DEFAULT '',
   ID_Type char(10) NOT NULL,
   FOREIGN KEY (ID_Type) REFERENCES TypeUser(ID_Type)
);
CREATE TABLE BookCatalog (
   ID_Catalog CHAR(10) PRIMARY KEY,
   Title NVARCHAR(255) NOT NULL,
   ID_Reader CHAR(10) NOT NULL,
```

```
    FOREIGN KEY (ID_Reader) REFERENCES Reader(ID_Reader) ON DELETE
CASCADE
);
CREATE TABLE Book (
    ID_Book CHAR(40) PRIMARY KEY,
    ID_Catalog CHAR(10) NULL,
    Title NVARCHAR(255) NOT NULL,
    Genre NVARCHAR(100) NOT NULL,
    PageNumber INT CHECK (PageNumber > 0),
    BookDescription NVARCHAR(MAX),
    ISBN NVARCHAR(20) UNIQUE NULL,
    CoverURL NVARCHAR(1000) NULL,
    FOREIGN KEY (ID_Catalog) REFERENCES BookCatalog(ID_Catalog) ON
DELETE SET NULL
);
CREATE TABLE Author (
    ID_Author CHAR(18) PRIMARY KEY,
    FirstName NVARCHAR(18) NOT NULL,
    LastName NVARCHAR(18) NOT NULL,
    MiddleName NVARCHAR(18)
);
CREATE TABLE Book_Author (
    ID_Book CHAR(40) NOT NULL,
    ID_Author CHAR(18) NOT NULL,
    PRIMARY KEY (ID_Book, ID_Author),
    FOREIGN KEY (ID_Book) REFERENCES Book(ID_Book) ON DELETE
CASCADE,
    FOREIGN KEY (ID_Author) REFERENCES Author(ID_Author) ON DELETE
CASCADE
```

```
);
CREATE TABLE Review (
   ID_Book CHAR(40) NOT NULL,
   ID_Reader CHAR(10) NOT NULL,
   Rating TINYINT CHECK (Rating BETWEEN 1 AND 5) NOT NULL,
   Feedback NVARCHAR(MAX),
   CreatedAt DATE DEFAULT GETDATE(),
   PRIMARY KEY (ID_Book, ID_Reader),
   FOREIGN KEY (ID_Book) REFERENCES Book(ID_Book) ON DELETE
CASCADE,
   FOREIGN KEY (ID_Reader) REFERENCES Reader(ID_Reader) ON DELETE
CASCADE
);
CREATE TABLE Note (
   ID_Reader char(10) NOT NULL,
   ID_Book CHAR(40) NOT NULL,
   NoteNumber char(10) NOT NULL,
   Content NVARCHAR(MAX) NOT NULL,
   CreatedAt DATE DEFAULT GETDATE(),
   PRIMARY KEY (ID_Reader, ID_Book, NoteNumber),
   FOREIGN KEY (ID_Reader) REFERENCES Reader(ID_Reader) ON DELETE
CASCADE,
   FOREIGN KEY (ID_Book) REFERENCES Book(ID_Book) ON DELETE
CASCADE
);
CREATE TABLE ReadingSession (
   ID_Reader CHAR(10) NOT NULL,
   ID_Book CHAR(40) NOT NULL,
   SessionNumber CHAR(10) NOT NULL,
```

```
    StartPage INT NOT NULL CHECK (StartPage > 0),
    EndPage INT NOT NULL CHECK (EndPage > 0),
    StartTime TIME NOT NULL,
    EndTime TIME NOT NULL,
    SessionDate DATE NOT NULL,
    CONSTRAINT CHK_PageOrder CHECK (EndPage >= StartPage),
    CONSTRAINT CHK_TimeOrder CHECK (EndTime > StartTime),
    PRIMARY KEY (ID_Reader, ID_Book, SessionNumber),
    FOREIGN KEY (ID_Reader) REFERENCES Reader(ID_Reader) ON DELETE
CASCADE,
    FOREIGN KEY (ID_Book) REFERENCES Book(ID_Book) ON DELETE
CASCADE
);
CREATE TABLE ReadingList (
    ID_Reader char(10) NOT NULL,
    ID_Book CHAR(40) NOT NULL,
    AddedAt DATE DEFAULT GETDATE(),
    PRIMARY KEY (ID_Reader, ID_Book),
    FOREIGN KEY (ID_Reader) REFERENCES Reader(ID_Reader) ON DELETE
CASCADE,
    FOREIGN KEY (ID_Book) REFERENCES Book(ID_Book) ON DELETE
CASCADE
);
CREATE TRIGGER trg_ReadingSession_LimitPagesPerBook
ON ReadingSession
INSTEAD OF INSERT
AS
BEGIN
    IF EXISTS (
```

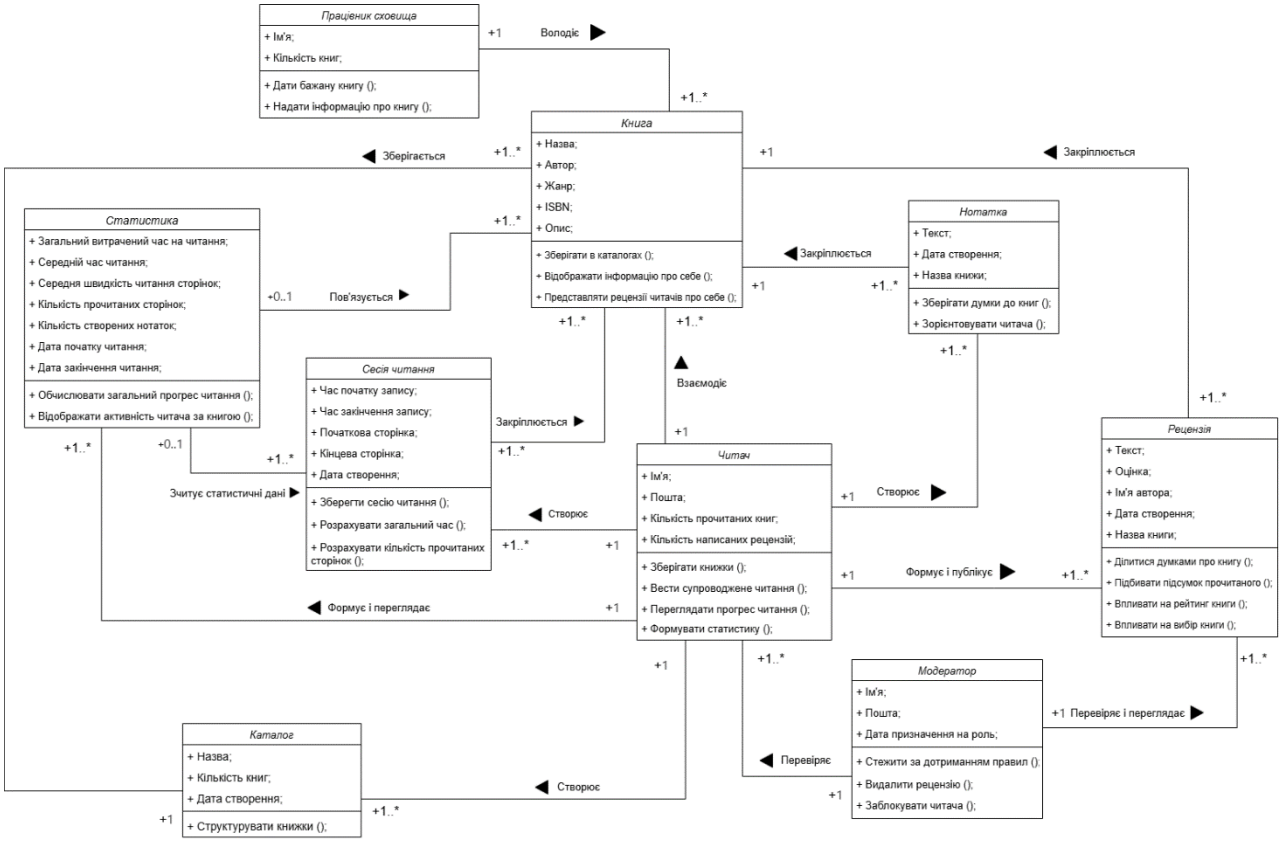
```
        SELECT 1
        FROM inserted i
        JOIN Book b ON i.ID_Book = b.ID_Book
        WHERE i.EndPage > b.PageNumber
    )
    BEGIN
        RAISERROR('Кінцева сторінка перевищує кількість сторінок книги!', 16,
1);
        ROLLBACK TRANSACTION;
        RETURN;
    END
    IF EXISTS (
        SELECT 1
        FROM inserted i
        JOIN Book b ON i.ID_Book = b.ID_Book
        WHERE i.StartPage > b.PageNumber
    )
    BEGIN
        RAISERROR('Початкова сторінка сесії перевищує кількість сторінок
книги!', 16, 1);
        ROLLBACK TRANSACTION;
        RETURN;
    END
    INSERT INTO ReadingSession (
        ID_Reader, ID_Book, SessionNumber, StartPage, EndPage,
        StartTime, EndTime, SessionDate
    )
    SELECT
        ID_Reader, ID_Book, SessionNumber, StartPage, EndPage,
```

```
        StartTime, EndTime, SessionDate
    FROM inserted;
END;

CREATE TRIGGER trg_DeleteUnusedAuthors
ON Book
AFTER DELETE
AS
BEGIN
    DELETE FROM Author
    WHERE ID_Author NOT IN (
        SELECT DISTINCT ID_Author FROM Book_Author
    );
END;

CREATE VIEW View_ReaderNote AS
SELECT
    R.ID_Reader,
    R.Nickname,
    B.ID_Book,
    B.Title AS BookTitle,
    N.NoteNumber,
    N.Content,
    N.CreatedAt
FROM Note N
JOIN Reader R ON R.ID_Reader = N.ID_Reader
JOIN Book B ON B.ID_Book = N.ID_Book;

CREATE VIEW View_ReaderProgressHisBook AS
```

```
SELECT
    R.ID_Reader,
    R.Nickname,
    B.ID_Book,
    B.Title AS BookTitle,
    B.PageNumber,
    SUM(RS.EndPage - RS.StartPage + 1) AS PagesRead,
    CAST(
        CAST(SUM(RS.EndPage - RS.StartPage + 1) AS FLOAT) /
NULLIF(B.PageNumber, 0) * 100
        AS DECIMAL(5,2)
    ) AS ReadPercentage
FROM Reader R
JOIN ReadingSession RS ON R.ID_Reader = RS.ID_Reader
JOIN Book B ON B.ID_Book = RS.ID_Book
     GROUP BY R.ID_Reader, R.Nickname, B.ID_Book, B.Title, B.PageNumber;
```

ДОДАТОК Ж

Діаграма класів із асоціацією



ДОДАТОК Л

Приклади коду

```
// Зміна ролі користувача
const handleUpdateRole = async (userId, newRole) => {
    try {
      const token = localStorage.getItem('token');
      const response = await
axios.put(`http://localhost:3001/api/readers/${userId}/role`,
        { role: newRole },
        {
          headers: {
            Authorization: `Bearer ${token}`,
          }
        }
      );

      if (response.status === 200) {
        window.alert(`Роль користувача успішно оновлено на ${newRole}`);
        fetchReaders();
      }
    } catch (error) {
      window.alert('Помилка оновлення ролі користувача');
      console.error('Помилка оновлення ролі користувача:', error);
    }
  };

<button
  className={`role-option ${
    reader.ID_Type === "0000000001" ? "current-role" : ""
```

```
  }`}
  onClick={() => {
    onUpdateRole(reader.ID_Reader, "admin");
    setActiveDropdown(null);
  }}
></button>;
exports.changeRole = async (req, res) => {
  try {
    const { id } = req.params;
    const { role } = req.body;
    const myDB = await poolPromise;
    const roleTypeMap = {
      'user': '0000000002',
      'admin': '0000000001'
    };
    const ID_Type = roleTypeMap[role];
    if (!ID_Type) {
      return res.status(400).json({ message: 'Невірна роль!' });
    }
    const result = await myDB
      .request()
      .input('idReader', sql.VarChar(10), id)
      .input('idType', sql.VarChar(10), ID_Type)
      .query('UPDATE Reader SET ID_Type = @idType WHERE ID_Reader =
@idReader');
    if (result.rowsAffected[0] === 0) {
      return res.status(404).json({ message: 'Користувача не знайдено!' });
    }
    return res.status(200).json({ message: 'Роль користувача успішно оновлено!' });
```

```
  } catch (error) {
    console.error(error);
    res.status(500).json({ message: 'Помилка оновлення ролі користувача!' });
  }
};

//Створення каталогу
const handleDeleteCatalog = async (catalogId, e) => {
    e.stopPropagation();
    if (window.confirm('Ви впевнені, що хочете видалити цей каталог?')) {
      try {
        await axios.delete(
          `http://localhost:3001/api-catalog/deleteCatalog/${catalogId}`,
          {
            headers: {
              Authorization: `Bearer ${localStorage.getItem("token")}`,
            },
          }
        );
        setCatalogs(prev => prev.filter(cat => cat.ID_Catalog !== catalogId));
      } catch (error) {
        console.error("Помилка видалення каталогу:", error);
      }
    }
  };
exports.createCatalog = async (req, res) => {
  try {
    const { Title } = req.body;
    const ID_Reader = req.user.id;
```

```
    if (!Title || Title.trim() === '') {
      return res.status(400).json({ message: 'Назва каталогу є обов’язковою' });
    }
    const newCatalogId = `c${crypto.randomBytes(4).toString('hex')}`;
    const pool = await poolPromise;
    await pool.request()
      .input('ID_Catalog', sql.Char(10), newCatalogId)
      .input('Title', sql.NVarChar(255), Title)
      .input('ID_Reader', sql.Char(10), ID_Reader)
      .query(`
        INSERT INTO BookCatalog (ID_Catalog, Title, ID_Reader)
        VALUES (@ID_Catalog, @Title, @ID_Reader)
      `);
    res.status(201).json({
      ID_Catalog: newCatalogId,
      Title,
      ID_Reader
    });
  } catch (error) {
    console.error('Помилка створення каталогу:', error);
    res.status(500).json({ message: 'Помилка створення каталогу' });
  }
};

//Видалення рецензії
const handleDeleteReview = async (bookId, readerId) => {
    const confirmDelete = window.confirm("Ти впевнений, що хочеш видалити цю рецензію?");
```

```
    if (!confirmDelete) return;
    try {
      const token = localStorage.getItem('token');
      const response = await
axios.delete(`http://localhost:3001/api/delreviews/${bookId}/${readerId}`, {
        headers: {
          Authorization: `Bearer ${token}`,
        }
      });
      if (response.status === 200) {
        console.log('Рецензію успішно видалено');
        fetchReviews();
      }
    } catch (error) {
      window.alert('Помилка видалення рецензії');
      console.error('Помилка видалення рецензії:', error);
    }
  };
const { poolPromise, sql } = require('../config/database');
exports.getReviews = async (req, res) => {
  try {
    const myDB = await poolPromise;
    const result = await myDB
      .request()
      .query(`
        SELECT r.ID_Book, r.ID_Reader, r.Rating, r.Feedback, r.CreatedAt,
          b.Title as book_title,
          rd.Email as author
        FROM Review r
```

```
        JOIN Book b ON r.ID_Book = b.ID_Book
        JOIN Reader rd ON r.ID_Reader = rd.ID_Reader
      `);
    return res.status(200).json(result.recordset);
  } catch (error) {
    console.error(error);
    res.status(500).json({ message: 'Помилка отримання списку рецензій!' });
  }
};
exports.deleteReview = async (req, res) => {
  try {
    const { bookId, readerId } = req.params;
    const myDB = await poolPromise;
    const result = await myDB
      .request()
      .input('bookId', sql.VarChar(10), bookId)
      .input('readerId', sql.VarChar(10), readerId)
      .query('DELETE FROM Review WHERE ID_Book = @bookId AND ID_Reader
= @readerId');
    if (result.rowsAffected[0] === 0) {
      return res.status(404).json({ message: 'Рецензію не знайдено!' });
    }
    return res.status(200).json({ message: 'Рецензію успішно видалено!' });
  } catch (error) {
    console.error(error);
    res.status(500).json({ message: 'Помилка видалення рецензії!' });
  }
};
```