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

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

# КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

**Магістр**
(назва освітнього ступеня)

на тему: Методи та засоби побудови програмно-апаратної платформи інформаційного забезпечення процесу обміну книгами

Виконав: студент(ка) 6 курсу, групи СІм-61
спеціальності 123 «Комп'ютерна інженерія»
(шифр і назва спеціальності)

(підпис) Вінтонів С. О. (прізвище та ініціали)

Керівник (підпис) Тиш Є. В. (прізвище та ініціали)

Нормоконтроль (підпис) Луцик Н.С. (прізвище та ініціали)

Завідувач кафедри (підпис) Осухівська Г.М. (прізвище та ініціали)

Рецензент (підпис) Гладьо Ю. Б. (прізвище та ініціали)

Тернопіль
2024

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

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

Кафедра комп'ютерних систем та мереж
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ *Осухівська Г.М.*
(підпис) (прізвище та ініціали)
«11» листопада 2024 р.

# З А В Д А Н Н Я
## НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня *магістр*
(назва освітнього ступеня)

за спеціальністю *123 «Комп'ютерна інженерія»*
(шифр і назва спеціальності)

студенту *Вінтоніву Святославу Олеговичу*
(прізвище, ім'я, по батькові)

1. Тема роботи *Методи та засоби побудови програмно-апаратної платформи інформаційного забезпечення процесу обміну книгами*

Керівник роботи *Тиш Євгенія Володимирівна, канд. техн. наук*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від « *29* » *листопада 2024* року № 4/7-1144
2. Термін подання студентом завершеної роботи *16.12.2024*
3. Вихідні дані до роботи *Функціональні вимоги до платформи*

4. Зміст роботи (перелік питань, які потрібно розробити)
*Вступ*
*1 Аналіз предметної області та методів розробки api*
*2 Проєктування програмно-апаратної платформи інформаційного забезпечення процесу обміну книгами*
*3 Ралізація платформи інформаційного забезпечення процесу обміну книгами з використанням обраних методів*
*4 Охорона праці та безпека в надзвичайних ситуаціях*
*Висновки*
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
*1. Актуальність і мета дослідження.*
*2. Завдання, об'єкт і предмет, наукова новизна і практична цінність дослідження.*
*3. Аналіз і вибір методів розробки API та формування вимог до системи*
*4. Проєктування архітектури бази дани та реалізація серверних класів*
*5. Побудова GraphQL API та його оптимізація з використанням DataLoader*
*6. Оптимізація кількості запитів до БД з використанням DataLoader*
*7. Реалізація графічного інтерфейсу та його тестування*
*8. Висновки*

6. Консультанти розділів роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| *Охорона праці* | *Осухівська Г.М., зав.каф. КС* | *05.12.2024* | |
| *Безпека в надзвичайних ситуаціях* | *Стручок В. С., каф. ОХ* | *09.12.2024* | |

7. Дата видачі завдання *11.11.2024*

## КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів роботи | Термін виконання етапів роботи | Примітка |
|---|---|---|---|
| 1. | *Аналіз існуючих методів та засобів розв'язання науково-технічних проблем, що відповідають тематиці кваліфікаційної роботи.* | *20.11.2024р. – 30.11.2024р..* | *Викон.* |
| 2. | *Аналіз предметної області та методів розробки API* | *30.11.2024р. – 03.12.2024р.* | *Викон.* |
| 3. | *Приєктування програмно-апаратної платфоми інформаційного забезпечення процесу обміну книгами* | *03.12.2024р. – 06.12.2024р.* | *Викон.* |
| 4. | *Реалізація платформи інформаційного забезпечення процесу обміну книгами з використанням обраних методів* | *06.12.2024р. – 10.12.2024р.* | *Викон.* |
| 5. | *Охорона праці та безпека в надзвичайних ситуаціях* | *11.12.2024р.* | *Викон.* |
| 6. | *Оформлення пояснювальної записки і графічного матеріалу* | *12.12.2024р. – 16.12.2024р.* | *Викон.* |
| 7. | *Попередній захист кваліфікаційної роботи магістра* | *16.12.2024* | *Викон.* |
| 8. | *Захист кваліфікаційної роботи магістра* | *23.12.2024* | *Викон.* |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

Студент _____ _____
(підпис) (прізвище та ініціали)

Керівник роботи _____ _____
(підпис) (прізвище та ініціали)

# АНОТАЦІЯ

Вінтонів С. О. Методи та засоби побудови програмно-апаратної платформи інформаційного забезпечення процесу обміну книгами: робота на здобуття кваліфікаційного ступеня магістра: спец. 123 — комп'ютерна інженерія / наук.кер. Є. В. Тиш. Тернопіль: Тернопільський національний технічний університет імені Івана Пулюя, 2024. — 93 с.

Ключові слова: react, graphql, dataloader, книга, користувач, оптимізація, api, запит, мутація, схема, графічний інтерфейс.

Кваліфікаційна робота присвячена дослідженню побудови програмно апаратної платформи інформаційного забезпечення процесу обміну книгами. Робота включає аналіз предметної області, визначення ключових вимог та вибір відповідних технологій для побудови масштабованої та ефективної системи. За допомогою Node.js було реалізовано GraphQL API для забезпечення гнучкого пошуку даних та оптимізації продуктивності запитів. Для підвищення ефективності було застосовано DataLoader для пакетної обробки та кешування запитів до бази даних, а для гнучкого зберігання даних було обрано MongoDB.

Графічний інтерфейс системи був розроблений з використанням React і Bootstrap, що робить його інтуєтивно зрозумілим. Платформа забезпечує безпечну автентифікацію користувачів, ефективне управління книгами та безперешкодну взаємодію між користувачами та адміністраторами. Результати цього дослідження демонструють практичне застосування сучасних інструментів і методів для побудови масштабованих систем, що відповідають функціональним і продуктивним вимогам платформи для обміну книгами.

# ANNOTATION

Vintoniv S. O. Methods and tools for developing a software and hardware platform to support the book exchange process. Master’s Graduation Thesis: speciality 123 — Computer engineering / supervisor Ie.V. Tysh. Ternopil: Ternopil Ivan Puluj National Technical University, 2024. — 93 p.

Keywords: react, graphql, dataloader, book, user, optimization, api, query, mutation, schema, graphical interface.

The qualification work is devoted to the study of developing a software and hardware platform to support the book exchange process. The work includes analyzing the subject area, identifying key requirements, and selecting appropriate technologies to build a scalable and efficient system. The GraphQL API was implemented using Node.js to provide flexible data search and optimize query performance. To increase efficiency, DataLoader was used to batch process and cache database queries, and MongoDB was chosen for flexible data storage.

The system's graphical interface was developed using React and Bootstrap, which makes it intuitive. The platform provides secure user authentication, efficient ledger management, and seamless interaction between users and administrators. The results of this research demonstrate the practical application of modern tools and methods for building scalable systems that meet the functional and performance requirements of a book exchange platform.

ЗМІСТ

# ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

БД – база даних

СУБД – система управління базами даних

CSS – Cascading Style Sheets

ВВ – Варіанти використання

ПЗ – Програмне забезпечення

CRM – Customer Relationship Management

HTML – Hyper Text Markup Language

JS – Java Script

NPM – Node Package Manager

DOM – Document Object Model

SPA – Single Page Application

API – Application Programming Interface

КПО – Коефіцієнт природньої освітленості

# ВСТУП

**Актуальність роботи.** розробка платформи обміну книгами полягає в її здатності впоратися зі зростанням як кількості користувачів так і обсягів даних. Очікується, що платформи для обміну контентом продовжуватимуть набирати популярність, за прогнозами до 2025 року кількість користувачів цих сервісів збільшиться на 20%, що створить нові вимоги до системної архітектури. Таке зростання означає, що кожна платформа повинна впроваджувати технології які забезпечують стабільну роботу в умовах високого трафіку.

Сучасні цифрові платформи для обміну контентом, особливо книгами, повинні надавати пріоритет надійності та масштабованості, щоб забезпечити безперебійну роботу для численних користувачів. Зважаючи на популярність таких платформ, як Wattpad та Goodreads потреба в ефективних API, що забезпечують узгоджене зберігання та обмін даними є очевидною. Ця кваліфікаційна робота спрямована на створення масштабованого API для платформи обміну книгами з використанням сучасних технологій, включаючи Node.js, Apollo Server та MongoDB.

Дослідження підкреслюють критичну роль сучасних підходів для досягнення стабільності та масштабованості системи. Наприклад, у статті "Scalability in Cloud Computing" підкреслюється необхідність горизонтального та вертикального масштабування для оптимізації продуктивності великих систем таких, як платформи для обміну контентом [1]. В іншому дослідженні «Архітектурні підходи до розробки масштабованих веб-додатків», підкреслюється важливість використання ефективних API і розподілених систем для підтримки стабільної продуктивності під великим навантаженням [2].

**Метою кваліфікаційної роботи** є дослідження сучасних підходів до розробки платформи інформаційного забезпечення процесу обміну книгами з акцентом на створення API здатного підтримувати платформу обміну книгами. Воно має забезпечувати стабільну роботу системи навіть при значному зростанні кількості користувачів та обсягу даних. Розроблене API повинно бути стійким до

високих навантажень, забезпечувати швидкий та надійний доступ до контенту, відповідати сучасним технологічним стандартам та бути зручним для інтеграції з клієнтськими додатками. Також важливим є високий рівень безпеки даних і можливість швидкого масштабування платформи відповідно до зростання попиту.

**Завдання кваліфікаційної роботи:**

- провести аналіз літератури пов’язаної з розробкою програмно апаратних платформ;
- дослідити предметну область та сучасні методи розробки API для подальшого їх провадження, описати методи та інструменти для оптимізації запитів до бази даних, щоб підвищити продуктивність та зменшити навантаження на систему;
- сформулювати та реалізувати основні функціональні вимоги до платформи;
- провести проєктування архітектури платформи та бази даних;
- реалізувати серверну частину платформи, провести стрес тестування та оптимізувати її за необхідності;
- розробити графічний інтерфейс для тестування всієї платформи.

*Об'єкт дослідження:* є процес проектування та створення API для платформи, яка надає користувачам можливість обміну книгами.

*Предмет дослідження:* є методи та технології, що застосовуються для розробки API на основі JavaScript, зокрема з використанням Node.js, Apollo Server та MongoDB.

**Методи дослідження:** У процесі розробки API для платформи обміну книгами було використано методи системного аналізу, що дозволили оцінити вимоги до масштабованості та продуктивності системи. Для визначення оптимальних технологій та архітектурних рішень застосовано методи порівняльного аналізу існуючих платформ та технологій таких, як REST та GraphQL, а також їх впливу на ефективність обробки запитів. Для підвищення швидкодії було досліджено застосування DataLoader для обробки запитів до бази

даних, щоб зменшити кількість запитів та підвищити ефективність API у великих обсягах даних.

**Наукова новизна отриманих результатів:**

- Проведено аналіз методів та технологій, що використовуються для побудови масштабованих API, що дозволило виділити найбільш ефективні підходи для розробки платформи обміну книгами з високою продуктивністю.
- Запропоновано інноваційне використання DataLoader для оптимізації запитів при зверненні до бази даних, що мінімізує кількість зайвих звернень і значно знижує навантаження на сервер, особливо при великій кількості одночасних запитів.
- Розроблено оптимізовану архітектуру платформи, яка забезпечує швидкий та безперебійний доступ до контенту для всіх користувачів, зберігаючи стабільність навіть при збільшенні навантаження.

**Практичне значення одержаних результатів.** Розроблена платформа може бути використана для реалізації інформаційної системи, що дозволяє користувачам легко завантажувати, зберігати, ділитися та оцінювати контент. Завдяки масштабованості API, платформа здатна обробляти великий обсяг запитів і підходить для впровадження у великих комерційних або освітніх проєктах, де важлива швидкодія та гнучкість у роботі з даними. Використання ефективного механізму кешування запитів забезпечує швидкий доступ до часто запитуваних даних, що підвищує продуктивність і комфорт користувачів.

**Публікації.** Результати дослідження апробовано на XI науково-технічній конференції Тернопільського національного технічного університету імені Івана Пулюя «Інформаційні моделі, системи та технології», XII міжнародній науково-технічній конференція молодих учених та студентів «Актуальні задачі сучасних технологій», у вигляді тез конференцій [3, 4].

**Структура роботи.** Робота складається з пояснювальної записки та графічної частини. Пояснювальна записка складається з вступу, 4 розділів, висновків, списку використаної літератури та додатків. Обсяг роботи: пояснювальна записка – 93 арк. формату А4, графічна частина – 7 аркушів формату А1.

# РОЗДІЛ 1
# АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА МЕТОДІВ РОЗРОБКИ API

## 1.1. Аналіз предметної області

У сучасну епоху цифрових технологій платформи для обміну та споживання контенту кардинально трансформували способи взаємодії людей із літературою та оповідями. Ці інноваційні платформи відкривають перед читачами та авторами нові горизонти, надаючи можливість спілкуватися, досліджувати широкий спектр творчих робіт і навіть співпрацювати над їх створенням, незалежно від географічного розташування. Перехід до онлайн-спільнот книголюбів став глобальною тенденцією, яка відображає значний попит на цифрові платформи такого типу. Ця тенденція пояснюється зростанням аудиторії цифрових читачів, широкою доступністю електронних видань і популяризацією контенту, що створюється безпосередньо користувачами. Таким чином розвиток платформ для обміну книжками відображає не лише зміну читацьких звичок, але й переосмислення способів створення, поширення та обговорення літературних творів у цифрову еру.

Суть платформи для обміну книжками полягає у сприянні обміну літературними творами, що дозволяє авторам ділитися своїми історіями, а читачам – знайомитися з новими жанрами та ідеями. Ці платформи задовольняють різні потреби, від демонстрації творчості письменників-аматорів до надання простору для професійних авторів, щоб вони могли продавати свої книги. Крім того, такі функції, як рецензії, рейтинги та персоналізовані рекомендації, підвищують залученість користувачів, створюючи надійну екосистему де контент постійно відкривається та обговорюється.

Зростання ринку цифрового обміну книгами підкреслює зростаючий попит на платформи, які сприяють безперешкодній взаємодії між читачами та авторами. Це зростання зумовлене поєднанням кількох факторів, зокрема поширенням доступних пристроїв для електронного читання, переходом до онлайн-спільнот для

літературної взаємодії та розширенням можливостей для самвидаву. Такі платформи, як Wattpad, Goodreads та Inkitt, скористалися цією тенденцією, пропонуючи динамічні екосистеми, де користувачі можуть читати, писати та обмінюватися контентом, створюючи середовище співпраці, яке приносить користь як творцям, так і споживачам.

Світовий ринок електронних книг наочно демонструє динамічний розвиток цієї сфери. За прогнозами галузевих аналітиків до 2025 року глобальний дохід від продажу електронних книг перевищить 15 мільярдів доларів, що підкреслює стрімке зростання популярності цифрового читання [5]. Мільйони користувачів щороку долучаються до цього процесу, активно сприяючи його розвитку. Такий прогноз відображає не лише фінансовий потенціал ринку, але й культурні та технологічні зміни, які впливають на те, як сучасні читачі споживають літературний контент. Зростання попиту на електронні книги свідчить про поступове зміщення фокусу від друкованих видань до зручніших і доступніших цифрових форматів, що відповідають потребам сучасного суспільства.

Важливим чинником зростання ринку електронних книг є популяризація незалежного публікування, яке надає авторам можливість напряму взаємодіяти зі своєю аудиторією минаючи традиційних видавців. Аналітичні дані свідчать, що частка незалежного публікування у продажах цифрових книжок невпинно зростає, що підкреслює ключову роль платформ, які підтримують цей формат. Онлайн-сервіси для обміну книжками не лише спрощують поширення контенту, але й сприяють відкриттю нових літературних голосів, розширюючи доступ до різноманітного культурного досвіду та роблячи ці платформи ще більш актуальними й привабливими для користувачів.

Існуючі платформи для обміну книжками часто не повністю відповідають потребам користувачів, зокрема через обмежені можливості для безпечного обміну матеріалами. Проблеми конфіденційності, відсутність ефективної інтеграції через API та недостатній рівень персоналізації вказують на суттєві прогалини, які можуть бути усунуті завдяки впровадженню інноваційних рішень. Створення спеціалізованого API для платформи обміну книжками здатне вирішити ці

виклики, забезпечуючи зручну інтеграцію з іншими програмами, гнучкий доступ до даних та масштабовані інструменти, що враховують потреби зростаючої аудиторії. Такий API міг би стати ключовим елементом, що підвищує функціональність платформи та її конкурентоспроможність.

Розробка API для платформи обміну книжками стає дедалі актуальнішою в умовах зростаючого попиту на інтегровані цифрові екосистеми. Це дозволяє розробникам створювати додатки, які не лише доповнюють функціонал платформи, але й розширюють її можливості. Наприклад, API може забезпечити такі функції, як сповіщення в реальному часі про публікацію нових історій, деталізовану аналітику користувацької активності, інтеграцію з популярними соціальними мережами та інші інструменти, що покращують взаємодію з користувачами. Такий підхід базується на сучасних принципах розробки програмного забезпечення, зосереджуючись на масштабованості, зручності у використанні та забезпеченні високого рівня безпеки, що робить платформу більш привабливою як для користувачів, так і для розробників.

Статистичні дані, що відображають зростання ринку онлайн-обміну книжками, підкреслюють його величезний потенціал і зростаюче значення цього сегмента в ширшій цифровій економіці. Щоб краще зрозуміти цю тенденцію, важливо проаналізувати ключові показники, такі як кількість активних користувачів на основних платформах для обміну та продажу електронних книг, а також ринкову вартість цієї індустрії. Ці статистичні дані дають уявлення про траєкторію розвитку ринку та можливості, які він відкриває як для розробників, так і для видавців та споживачів.

Таблиця 1.1 нижче ілюструє зростання ринку електронних книг та платформ для обміну ними з 2020 по 2024 рік, та приблизний прогноз зростання ринку на період з 2024 по 2028 рік. Дані було сформовано на основі аналізу двох джерел статистиці ринку електронних книг [6] та прогнозі ринку електронних книг [7].

*Таблиця 1.1*

**Зростання ринку цифрових книг**

| Рік | Активні користувачі на основних платформах | Ринкова вартість |
|---|---|---|
| 2020 | 286 млн. | 15 553 млн. доларів |
| 2021 | 326 млн. | 17 476 млн. доларів |
| 2022 | 358 млн. | 19 205 млн. доларів |
| 2023 | 381 млн. | 20 876 млн. доларів |
| 2024 | 408 млн. | 22 448 млн. доларів |
| 2025 | 435 млн. | 23 592 млн. доларів |
| 2026 | 457 млн. | 24 748 млн. доларів |
| 2027 | 479 млн. | 26 060 млн. доларів |
| 2028 | 504 млн. | 27 387 млн. доларів |

Ці дані відображають основні тенденції, зокрема активне впровадження на початку 2020-х років, спричинене зростанням доступу до інтернету та розвитком технологій для цифрового читання. До середини 2020-х років темпи зростання стають ще більш виразними: платформи розширюють свій асортимент і виходять на нові ринки. Водночас наприкінці 2020-х років ринок починає демонструвати ознаки поступової стабілізації, досягаючи насичення в економічно розвинених регіонах. Таке розуміння ринкових змін підкреслює важливість створення масштабованих і гнучких API, які дозволять ефективно задовольняти потреби зростаючої аудиторії.

Розробка API для платформи обміну книгами є не лише актуальною, але й необхідною для задоволення потреб зростаючої цифрової аудиторії. Усуваючи існуючі недоліки та забезпечуючи безперешкодну інтеграцію, цей проект покликаний створити надійну та функціональну основу, яка дозволить користувачам не тільки ділитися новими книгами, але й взаємодіяти між собою у зручний і безпечний спосіб. Це сприятиме формуванню активної спільноти книголюбів, яка підтримуватиме культурний обмін та стимулюватиме творчість.

## 1.2. Аналіз методів розробки API

Створення сучасних API передбачає вирішення багатьох складних завдань, таких як забезпечення ефективної обробки даних, гарантування безпеки, забезпечення масштабованості та розробка з урахуванням потреб користувачів. Зі збільшенням обсягів платформ для обміну контентом, API стикаються з необхідністю підтримувати зростаючий трафік запитів, водночас забезпечуючи стабільну продуктивність і надійність. Це вимагає застосування передових рішень, які можуть адаптуватися до змін у навантаженні та архітектурі платформи. У цьому розділі аналізуються сучасні підходи, що використовуються для вирішення цих завдань, зосереджуючись на практичних прикладах їх реалізації та перевагах для різних категорій додатків. Додатково розглядаються аспекти оптимізації роботи API у контексті зростаючого попиту на інтеграцію з іншими системами [8].

Ефективна обробка даних є основою для розробки API, особливо для платформ, що генерують мільйони запитів та взаємодій щодня. Одним із ключових методів підвищення продуктивності є використання асинхронної обробки запитів, яка дозволяє API працювати з декількома запитами одночасно, уникаючи затримок через блокуючі операції. Наприклад, цей підхід широко використовується для обробки складних запитів до бази даних або інтеграції зі сторонніми API, що вимагають значних обчислювальних ресурсів. Node.js, завдяки своїй подієво-орієнтованій архітектурі, яка не блокується, вважається одним із найкращих рішень для створення високопродуктивних API. Інструменти, такі як Express, додатково полегшують роботу з асинхронними запитами, спрощуючи створення складних серверних застосунків [9]. Важливо також враховувати використання балансування навантаження для підтримки стабільної роботи API у випадках різких піків трафіку.

Ще одним критично важливим компонентом для підвищення продуктивності API є кешування даних. Цей підхід передбачає тимчасове зберігання часто використовуваної інформації для швидкого доступу, що зменшує навантаження на сервери та значно скорочує час відгуку на запити. Зокрема, технології кешування в

оперативній пам’яті, такі як Redis та Memcached, дозволяють обробляти великі обсяги даних із мінімальними затримками. У випадку з GraphQL кешування може здійснюватися на рівні запитів, що дозволяє уникати надмірних обчислень і знижувати навантаження на мережу. Крім того, сучасні платформи часто комбінують кешування з іншими методами оптимізації, такими як попередня обробка даних або використання CDN (мереж доставки контенту), що покращує користувацький досвід і забезпечує стабільну роботу API навіть при високому навантаженні. Усі ці підходи є ключовими для забезпечення масштабованості та ефективності сучасних API [10].

Безпека API є ключовим аспектом у забезпеченні надійності сучасних цифрових систем, адже API служать точками входу до критично важливих даних і сервісів. Для захисту від таких загроз, як SQL-ін'єкції, атаки типу DDoS та несанкціонований доступ, впроваджуються багаторівневі стратегії безпеки. Механізми автентифікації, такі як OAuth і JSON Web Tokens (JWT), забезпечують, що тільки автентифіковані користувачі можуть взаємодіяти з API. У свою чергу, системи авторизації надають ще більшу гранулярність доступу, дозволяючи обмежувати доступ до ресурсів залежно від рівня дозволів користувачів. Додатково застосовуються такі методи, як перевірка вхідних даних, що запобігає ін'єкційним атакам, і обмеження швидкості запитів, яке захищає сервери від перевантаження [11]. Наприклад, проміжне програмне забезпечення у Node.js значно полегшує імплементацію цих захисних заходів. При використанні GraphQL важливою є підтримка строгих схем в Apollo Server, які вбудовано забезпечують валідацію запитів, мінімізуючи ризик експлуатації помилок у структурі даних.

Масштабованість є ще однією фундаментальною вимогою до API, особливо для платформ із високими обсягами трафіку та різкими піками навантаження. Горизонтальне масштабування, що передбачає розподіл запитів між кількома серверами або хмарними інстансами, є стандартним підходом до підтримки продуктивності під час зростання трафіку. Інструменти, як-от NGINX або AWS Elastic Load Balancer, дозволяють рівномірно розподіляти навантаження, запобігаючи перевантаженню окремих серверів [12]. Використання GraphQL надає

ще одну значну перевагу — можливість вибірки тільки необхідних даних через один запит, що істотно зменшує обсяг передачі даних і оптимізує взаємодію між клієнтом і сервером. У порівнянні з REST API, які часто вимагають декількох викликів для отримання необхідної інформації, GraphQL дозволяє зменшити навантаження на сервер і підвищити ефективність роботи платформи. Apollo Server додатково спрощує управління цими процесами, інтегруючи кешування та оптимізацію запитів безпосередньо у свою архітектуру.

Модерні фреймворки та платформи надають розробникам інструменти для створення API, які відповідають найвищим стандартам продуктивності, гнучкості та зручності. Наприклад, Node.js у поєднанні з Express забезпечує ефективну основу для розробки легких серверів, які можуть обробляти тисячі одночасних запитів. У контексті GraphQL, Apollo Server стає важливим компонентом, який пропонує інтуїтивне управління схемами, розширені можливості запитів та підтримку функцій реального часу таких, як підписки. Це особливо цінно для платформ, які потребують динамічного оновлення контенту, як-от обмін книгами. Крім того, Apollo Server забезпечує автоматизовану валідацію схем і запитів, зменшуючи кількість помилок і роблячи API більш надійними та безпечними.

Сукупність цих методів і технологій демонструє важливість сучасних підходів до розробки API в умовах швидкозмінного цифрового середовища. Завдяки інтеграції таких інструментів, як GraphQL та Apollo Server, розробники можуть не тільки підвищити продуктивність і масштабованість, але й забезпечити безперебійну роботу своїх платформ навіть у складних умовах. Цей комплексний підхід дозволяє задовольнити потреби сучасних користувачів і забезпечити довготривалу стабільність та ефективність системи.

## 1.3. Аналіз існуючих платформ

Серед платформ, які забезпечують подібний функціонал до тієї, яка буде розроблятися є Wattpad [13] – це одна з найпопулярніших онлайн-платформ для публікації та читання книг, яка набула величезної популярності серед як

початківців, так і досвідчених авторів. Заснована у 2006 році, платформа об'єднала мільйони користувачів по всьому світу та стала важливим місцем для створення та споживання контенту. Wattpad пропонує авторам можливість публікувати власні роботи, а читачам — відкривати нові історії та взаємодіяти з ними у реальному часі (рис. 1.1).

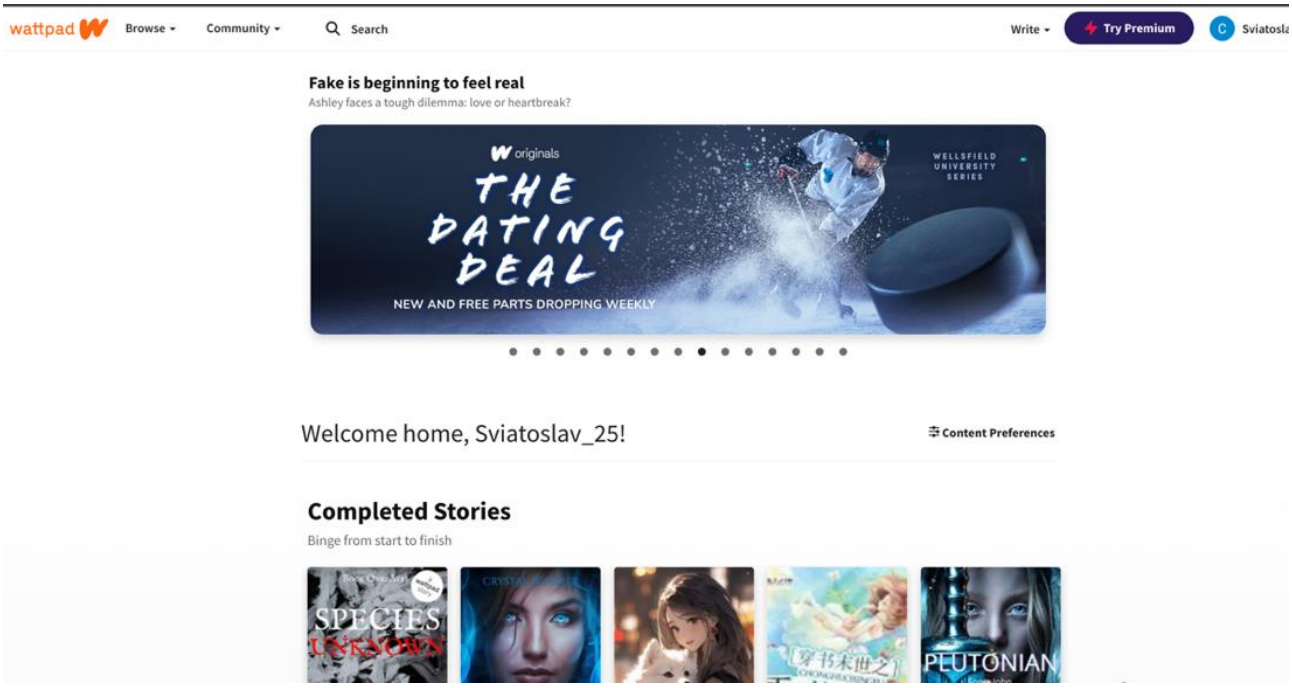


Рис. 1.1. Платформа Wattpad

Основна сила Wattpad полягає в її активній спільноті користувачів, інтерактивних функціях та інтуїтивному інтерфейсі, що забезпечує зручність використання. Однак, як і будь-яка інша платформа, вона має свої недоліки, які можуть вплинути на користувацький досвід і розвиток платформи в майбутньому. Нижче наведено аналіз сильних та слабких сторін Wattpad, які допоможуть глибше зрозуміти конкурентні переваги та проблеми цього рішення.

Сильні сторони:

- Wattpad має глобальну аудиторію з мільйонами користувачів, що створює великий трафік і активну спільноту для обміну історіями.

- Платформа дозволяє користувачам коментувати історії в реальному часі, підкреслювати улюблені фрагменти, що сприяє тісній взаємодії між авторами та читачами.
- Wattpad має добре розроблений мобільний додаток, який забезпечує зручний доступ до платформи з будь-якого пристрою.
- Платформа використовує потужні алгоритми для персоналізації контенту та надання рекомендацій, що допомагає користувачам швидше знаходити цікаві історії.
- Wattpad надає авторам можливість заробляти через рекламу, публікацію книг або підтримку фанатів, що робить платформу привабливою для професійних авторів.

Слабкі сторони:

- Через велику кількість контенту новим або менш відомим авторам складно виділитися, а читачам важко знайти якісний матеріал серед великого обсягу низькоякісного контенту.
- Wattpad має базові функції приватності, що може бути проблемою для користувачів, які хочуть більш гнучко керувати своїм контентом або обмежити його доступність.
- Платформа не завжди ефективно перевіряє вміст на відповідність правилам або на оригінальність, що призводить до випадків плагіату та порушень авторських прав.
- Редактор тексту на Wattpad є досить базовим і не надає багато можливостей для структурованого редагування, що може бути незручним для більш досвідчених авторів.
- Wattpad не пропонує розширених можливостей через API, що обмежує інтеграції з іншими платформами та сервісами.

Після аналізу платформи Wattpad, яка орієнтована на взаємодію авторів і читачів у реальному часі та публікацію нових книг, важливо звернути увагу на іншу популярну платформу для обміну книгами та контентом — Goodreads [14]. Хоча ці платформи мають різні підходи та аудиторію, обидві грають важливу роль у світі

літературних платформ. Далі буде проведено аналіз сильних і слабких сторін Goodreads з точки зору її функціоналу, спільноти та можливостей для користувачів (рис 1.2).

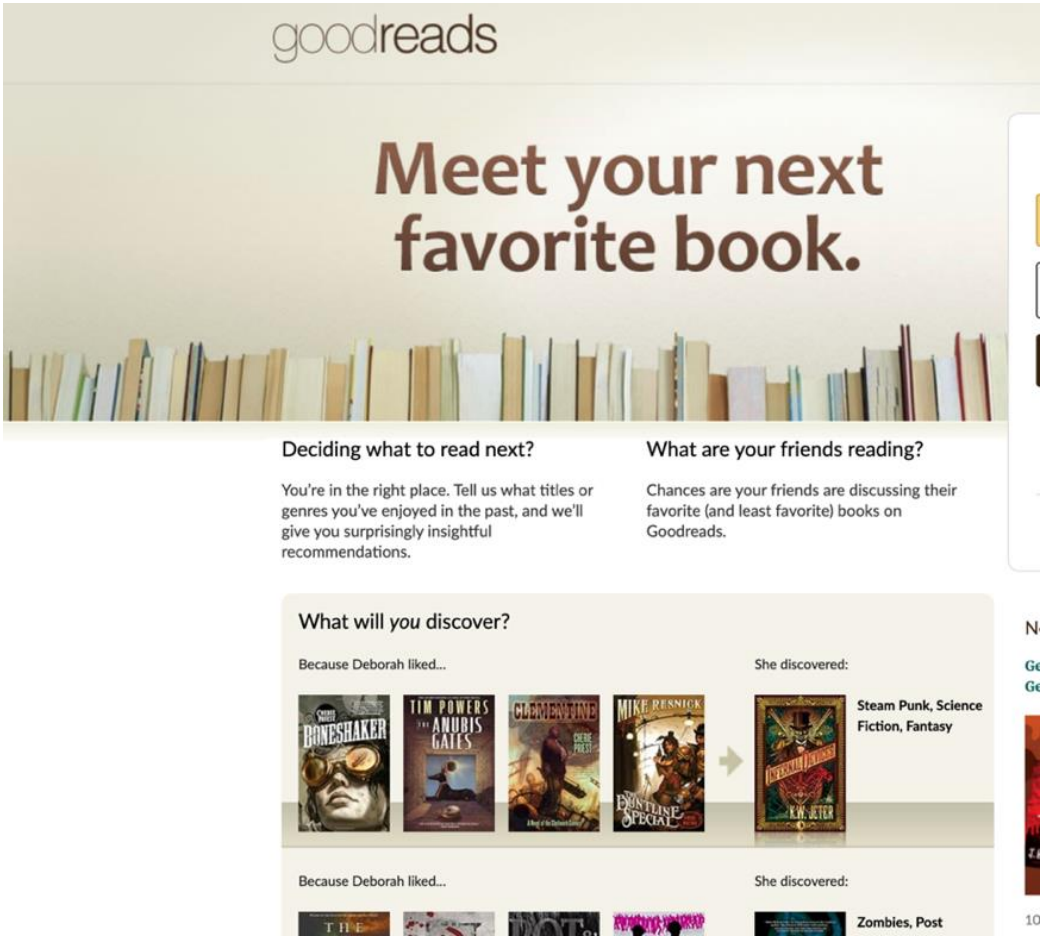


Рис. 1.2. Платформа Goodreads

Сильні сторони:

- Goodreads є однією з найбільших онлайн-спільнот для книголюбів з доступом до мільйонів рецензій, оцінок та рекомендацій. Величезна кількість користувачів допомагає створити жваву дискусійну атмосферу щодо книг і авторів.
- Як платформа, що належить Amazon, Goodreads має інтеграцію з Kindle, що дозволяє легко переглядати й купувати книги, підвищуючи зручність для користувачів.

- Goodreads пропонує потужну систему рекомендацій, яка базується на рецензіях та оцінках користувачів. Це сприяє персоналізованому пошуку нових книг, які можуть бути цікавими користувачам.
- Користувачі можуть створювати власні списки бажаних книг, приєднуватися до літературних груп і обговорень, що робить платформу інтерактивною.
- Платформа пропонує статистику читання, дозволяючи користувачам слідкувати за своїм прогресом, що приваблює любителів систематизованого підходу до читання.

Слабкі сторони:

- Інтерфейс Goodreads виглядає дещо застарілим і менш інтерактивним порівняно з більш сучасними платформами, такими, як Wattpad. Це може знизити привабливість платформи для нових користувачів.
- На відміну від Wattpad, Goodreads не фокусується на взаємодії між авторами і читачами в реальному часі, що обмежує залучення аудиторії до авторського процесу.
- Хоча платформа пропонує рекомендації, їхня точність не завжди висока через обмежені можливості аналізу індивідуальних вподобань користувачів.
- Goodreads більше орієнтована на рецензії та рекомендації щодо книг, а не на публікацію оригінальних творів, що обмежує можливості для нових авторів.
- Платформа відстає у впровадженні нових технологічних рішень, що знижує її гнучкість і конкурентоспроможність на ринку.

Після аналізу таких платформ, як Wattpad і Goodreads, варто звернути увагу на ще одного важливого гравця у сфері літературних платформ — Inkitt [15] (рис. 1.3).

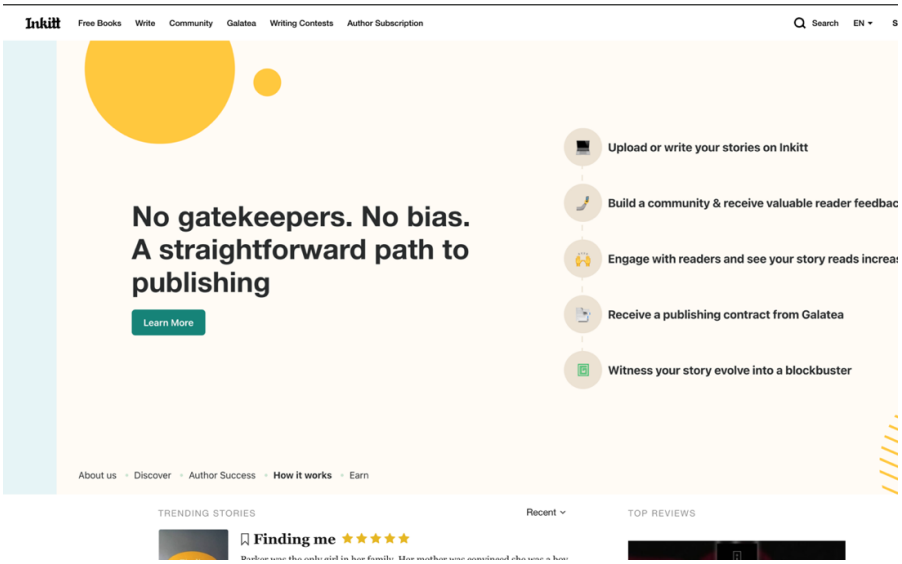


Рис. 1.3. Платформа Inkitt

Inkitt позиціонує себе як платформа для нових авторів, які прагнуть опублікувати свої твори та отримати зворотний зв'язок від спільноти. Водночас Inkitt відрізняється своєю унікальною пропозицією для авторів, оскільки платформа використовує алгоритми для виявлення потенційних бестселерів. Необхідно проаналізувати сильні та слабкі сторони цієї платформи, щоб зрозуміти її місце на ринку.

Сильні сторони:

- Однією з головних переваг Inkitt є використання алгоритмів для аналізу популярності книг та визначення творів, які мають потенціал стати бестселерами. Це дає авторам можливість опублікувати свої твори на професійному рівні.
- Inkitt фокусується на підтримці нових письменників, пропонуючи їм простір для зростання та розвитку. Автори можуть отримувати відгуки від читачів і мати шанс на публікацію.
- Після того, як твір на Inkitt показує високі показники за допомогою алгоритмів, платформа пропонує авторам укладати угоди з видавцями, що створює можливість виходу на професійний ринок.
- Inkitt пропонує зручний мобільний додаток, що дозволяє читати книги на ходу, забезпечуючи зручний доступ до контенту. Це допомагає розширити аудиторію та збільшити залученість читачів.

- Автори можуть публікувати твори в різних жанрах і форматах, що надає платформі багатофункціональності та приваблює різні категорії користувачів.

Слабкі сторони:

- Порівняно з іншими платформами, Inkitt має меншу кількість доступних книг і авторів, що може знизити її привабливість для широкої аудиторії.
- Хоча Inkitt сприяє взаємодії між авторами і читачами, рівень активності спільноти, особливо у вигляді рецензій та обговорень, значно нижчий у порівнянні з Wattpad або Goodreads.
- Використання алгоритмів для визначення популярних творів може стати викликом для авторів, оскільки процес відбору не завжди прозорий і зрозумілий.
- На відміну від Goodreads, яка інтегрована з Amazon, Inkitt не має таких великих партнерських угод, що обмежує її маркетингові можливості.
- Платформа переважно спрямована на новачків, тому досвідчені автори можуть не знайти для себе значних переваг чи можливостей для розвитку.

Аналіз трьох популярних платформ для обміну книгами та історіями — Wattpad, Goodreads і Inkitt – дозволяє виділити ключові вимоги, які варто врахувати при розробці платформи.

Вимоги до платформи:

- Як показав аналіз Wattpad, користувачі активно шукають можливість взаємодії між авторами та читачами. Тому необхідно запровадити функціонал, який дозволить читачам залишати коментарі, рецензії та отримувати швидку відповідь від авторів.
- Подібно до Goodreads, платформа має бути масштабованою, щоб підтримувати великий обсяг користувачів і контенту. Це означає необхідність гнучкої архітектури API, здатної обробляти великі обсяги запитів і даних.
- Одним із сильних аспектів Goodreads є інтеграція з Amazon, що значно збільшує її маркетингові можливості. Тому необхідно реалізувати інтеграцію з популярними сервісами.

- Як було продемонстровано на прикладі Inkitt, можливість використовувати алгоритми для аналізу популярності творів є важливою перевагою.
- Щоб конкурувати з платформами на зразок Wattpad і Goodreads, розроблювана платформа повинна мати інтуїтивно зрозумілий інтерфейс, який полегшує доступ до контенту як авторам, так і читачам. Приділяючи увагу зручності користувача, можна залучити більшу аудиторію.
- Як і Inkitt, платформа має підтримувати нових авторів, надаючи їм можливість публікувати свої твори без складних формальностей і створюючи умови для їхнього зростання та розвитку.

## 1.4. Висновки до розділу

У цьому розділі було сформовано базу для розробки API для платформи обміну книгами через аналіз предметної області, методів створення API та існуючих платформ. Дослідження показало зростаючий інтерес до онлайн-платформ, що сприяють взаємодії між авторами та читачами, з акцентом на масштабованості, залученні користувачів і доступності. Розгляд сучасних підходів до розробки API продемонстрував переваги GraphQL завдяки його ефективності, гнучкості та здатності забезпечувати високий рівень масштабованості.

Аналіз платформ Wattpad, Goodreads та Inkitt дозволив визначити ключові аспекти, на які слід орієнтуватися: створення інструментів для взаємодії, інтеграція з зовнішніми сервісами та підтримка нових авторів. Цей огляд допоміг визначити основні вимоги, зокрема інтерактивні функції, обробку великих обсягів даних і створення зручного для користувача інтерфейсу.

На основі цих результатів було сформовано цілісне бачення створення надійної, масштабованої та конкурентоспроможної платформи для книгообміну, яка адаптується до потреб сучасних користувачів.

# РОЗДІЛ 2
# ПРОЄКТУВАННЯ ПРОГРАМНО-АПАРАТНОЇ ПЛАТФОРМИ ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ ПРОЦЕСУ ОБМІНУ КНИГАМИ

## 2.1. Теоретичні аспекти розробки API для програмно-апаратної платформи інформаційного забезпечення процесу обміну книгами

Розробка надійного та ефективного API для програмно-апаратної платформи інформаційного забезпечення процесу обміну книгами є важливим завданням, оскільки саме від цього залежить якість взаємодії користувачів із платформою. На ранніх етапах було вирішено застосувати монолітну архітектуру в поєднанні з GraphQL та Node.js, що дає змогу спростити процес розробки, оптимізувати роботу із запитами та створити передумови для подальшого вдосконалення. Монолітний підхід полягає в об’єднанні всіх компонентів — автентифікації користувачів, управління книгами, систем оцінювання та доставки контенту — в рамках єдиного застосунку. Такий спосіб організації значно полегшує процес впровадження нових можливостей, спрощує тестування і прискорює процес розгортання нових версій. Крім того, централізоване керування даними дає змогу мінімізувати затримки, оскільки всі операції, від пошуку книжок до взаємодії з користувачами, виконуються в єдиному середовищі, що знижує складність налаштування інтеграційних зв’язків.

Обрані технології допомагають ефективно реагувати на зростаючі обсяги даних та користувацьких запитів. GraphQL, як ключовий компонент архітектури, дозволяє клієнтам запитувати тільки необхідну їм інформацію, уникаючи надмірної чи недостатньої вибірки [16]. Це робить роботу з даними більш гнучкою і оптимізованою: користувачі можуть отримати потрібні метадані про книжки, інформацію про авторів чи відомості про рейтинги, не перевантажуючи систему зайвими викликами. Node.js, зі своєю неблокуючою моделлю вводу-виводу, забезпечує одночасну обробку численних запитів, зокрема завантаження файлів, формування рекомендацій і динамічне оновлення інформації. Така взаємодія

GraphQL і Node.js не тільки покращує загальну продуктивність, а й спрощує підтримку та розвиток API, оскільки змінювати чи розширювати його функціонал можна без істотних ускладнень [17].

Проте навіть монолітна архітектура потребує продуманого підходу до зростання системних вимог і обробки великої кількості запитів. Під час розробки було враховано можливість ефективного управління запитами, впроваджено інструменти оптимізації, такі як механізми пакетної обробки чи кешування. Застосування DataLoader, наприклад, дає змогу мінімізувати дубльовані звернення до бази даних та оптимізувати складні операції з пов’язаними даними. Поряд із цим, завдяки використанню контейнеризації та інструментів оркестрації, на кшталт Docker та Kubernetes, у майбутньому можна буде розподіляти навантаження між декількома екземплярами системи, покращуючи стійкість до пікових навантажень. Оптимізація бази даних, належна індексація, ретельний вибір схеми зберігання та продумана робота з індексами забезпечать ефективне керування даними книжок, транзакцій і користувацької активності.

Таким чином, поєднання монолітної архітектури з GraphQL і Node.js створює міцну основу для розробки надійної та зручної платформи для книгообміну. Такий підхід дає змогу швидко реагувати на зміни потреб користувачів і підвищувати ефективність системи без перешкод для подальшого розвитку й інтеграції з новими зовнішніми сервісами, а також забезпечує більш природну та прогнозовану еволюцію всієї платформи.

## 2.2. Вибір методів для розробки API

На основі аналізу методів розробки API було обрано конкретні підходи та технології для реалізації API для платформи обміну книжками. У процесі прийняття рішення основну увагу було приділено таким вимогами, як масштабованість, продуктивність, безпека та зручність обслуговування. Для розробки були обрані методи та технології які описані нижче.

Асинхронна обробка запитів гарантує, що API може обробляти декілька клієнтських запитів одночасно, не блокуючись при цьому довготривалими операціями. З цієї причини Node.js з його подієво-керованою, неблокуючою моделлю вводу/виводу був обраний в якості основного середовища виконання. Це дозволяє API ефективно обробляти такі завдання, як завантаження книг, пошук контенту та сповіщення, що є критично важливими для платформи, яка обробляє динамічну взаємодію з користувачами. Цей метод покращує продуктивність і швидкість реакції в умовах високого навантаження.

Для зменшення навантаження на базу даних та оптимізації продуктивності запитів буде реалізовано кешування на рівні запитів GraphQL. Завдяки кешуванню результатів часто виконуваних запитів, наступні запити можуть обслуговуватися швидше без повторної вибірки даних з бази даних. Такий підхід значно підвищує ефективність системи, особливо для таких операцій, як пошук популярних книг, профілів авторів або інших даних, до яких часто звертаються.

Забезпечення безпечного доступу до платформи має важливе значення, особливо при роботі з конфіденційними даними користувачів і платним контентом. Для реалізації автентифікації та авторизації будуть використовуватися веб-маркери JSON (JWT). JWT гарантує, що тільки авторизовані користувачі можуть взаємодіяти з API, запобігаючи несанкціонованому доступу. Токени видаватимуться під час входу користувача та перевірятимуться при наступних викликах API, забезпечуючи безпечний та легкий механізм управління сеансами користувачів [18].

Apollo Server буде використовуватися як реалізація GraphQL, пропонуючи підтримку для визначення схем і валідації запитів. Суворі схеми будуть застосовуватися, щоб забезпечити узгодженість і безпеку структури запитів і відповідей API. Завдяки перевірці всіх вхідних запитів на відповідність попередньо визначеним схемам, ризик використання структури даних або неочікуваних помилок зводиться до мінімуму. Цей метод також підвищує надійність API та забезпечує чіткий зворотній зв’язок з клієнтами, зменшуючи потенційні проблеми інтеграції.

Вибрані методи в сукупності вирішують основні проблеми, виявлені під час аналізу методів розробки API. Node.js забезпечує здатність API ефективно обробляти великі обсяги одночасних запитів. Кешування на рівні GraphQL зменшує непотрібні обчислення і прискорює час відгуку для часто використовуваних даних. JWT забезпечує надійний механізм автентифікації, гарантуючи, що API залишається безпечним, одночасно підтримуючи масштабоване управління сеансами. Нарешті, суворе дотримання схеми Apollo Server гарантує стабільність, узгодженість і безпеку API [19].

Разом ці методи створюють міцний фундамент для побудови масштабованого, продуктивного і безпечного API, який відповідає функціональним вимогам платформи. Завдяки використанню сучасних інструментів і технологій, API забезпечить безперебійну роботу як для користувачів, так і для розробників, будучи готовим до майбутнього зростання.

## 2.3. Формування основних вимог до платформи

Створення сучасної платформи для обміну книжками потребує чітко визначених функціональних вимог, які враховують інтереси як авторів, так і читачів [20]. Ці вимоги визначають базові принципи платформи, спрямовані на забезпечення інтуїтивного інтерфейсу, ефективного управління контентом та здатності до масштабування відповідно до зростаючих потреб спільноти. Особливу увагу приділено аспектам доступності, інтерактивності та гнучкості, щоб створити платформу, яка стимулює взаємодію та сприяє розвитку творчого середовища.

Ключові функції платформи охоплюють основні потреби користувачів, забезпечуючи безперешкодну взаємодію між ними та створюючи умови для розширення спільноти. Нижче детально описано вимоги, які лягли в основу розробки, із врахуванням сучасних тенденцій та потреб цифрової аудиторії.

Вимоги:

- Платформа повинна включати надійні механізми реєстрації та авторизації, які забезпечать безпечний і персоналізований доступ для кожного користувача.
- Адміністраторам має надаватися можливість управляти ролями користувачів, що дозволить ефективно контролювати доступ до функціоналу платформи.
- Система деактивації облікових записів користувачів повинна надавати адміністраторам інструменти для управління активністю на платформі та підтримання політик спільноти.
- Інструменти для аналізу статистики повинні надавати адміністраторам доступ до детальної інформації про активність користувачів, популярність контенту та загальну продуктивність системи.
- Користувачам надаватиметься можливість редагувати свої профілі, що сприятиме персоналізації та підвищенню залученості до роботи з платформою.
- Автори повинні мати змогу створювати, редагувати та видаляти книги, додаючи важливі метадані, такі як назва, обкладинка, жанр, налаштування конфіденційності, кількість сторінок, файл книги, вартість (для платних книг), співавтори та опис.
- Система повинна надавати можливість налаштування конфіденційності, які дозволять користувачам обирати, чи будуть їхні книги публічними або приватними.
- Функція пошуку повинна надавати користувачам можливість швидко знаходити книги, використовуючи різні критерії.
- Користувачі мають мати змогу оцінювати книги та сортувати їх за рейтингом або популярністю, що полегшить пошук якісного контенту.
- Читачам має надаватися можливість підписуватися на улюблених авторів, щоб отримувати сповіщення про нові публікації.
- Система має надавати можливість додавання книг до списку обраного, який дозволить користувачам зберігати улюблені книги для швидкого доступу до них у майбутньому.

- Платформа має підтримувати купівлю платних книг, створюючи авторам можливість монетизувати свою творчість і залучати нову аудиторію.

Ці вимоги мають створити міцний фундамент для платформи, пропонуючи поєднання основних і додаткових функцій. Разом вони спрямовані на створення привабливого, масштабованого і орієнтованого на користувача середовища.

## 2.4. Визначення варіантів використання

Визначення варіантів використання є важливим кроком у проектуванні та розробці будь-якої програмної системи. Він забезпечує структурований спосіб фіксації взаємодії між користувачами і системою, гарантуючи, що функціональність відповідає вимогам і цілям, визначеним на попередніх етапах проекту. У контексті цієї програмно-апаратної платформи інформаційного забезпечення процесу обміну книгами, варіанти використання допомагають чітко окреслити ролі учасників системи та їхню взаємодію з функціями платформи. Ці приклади використання формують основу для впровадження та тестування системи, щоб переконатися, що вона ефективно відповідає потребам користувачів.

В розроблюваній платформі буде три основні ролі: гість, користувач і адміністратор. Кожна з цих ролей має унікальні обов'язки та доступ до певних функцій платформи. Гість має обмежені привілеї, в першу чергу обмежені реєстрацією та входом в систему. Після реєстрації користувач отримує доступ до основних функцій, включаючи створення, редагування та взаємодію з книгами, оцінювання контенту, а також взаємодію з іншими користувачами за допомогою таких функцій, як підписка та обране. Роль адміністратора охоплює розширені можливості, такі як управління ролями користувачів, блокування облікових записів та аналіз статистики платформи, щоб підтримувати нагляд за системою та забезпечувати її безперебійну роботу.

Діаграму варіантів використання для програмно-апаратної платформи інформаційного забезпечення процесу обміну книгами наведено на рисунку 2.1

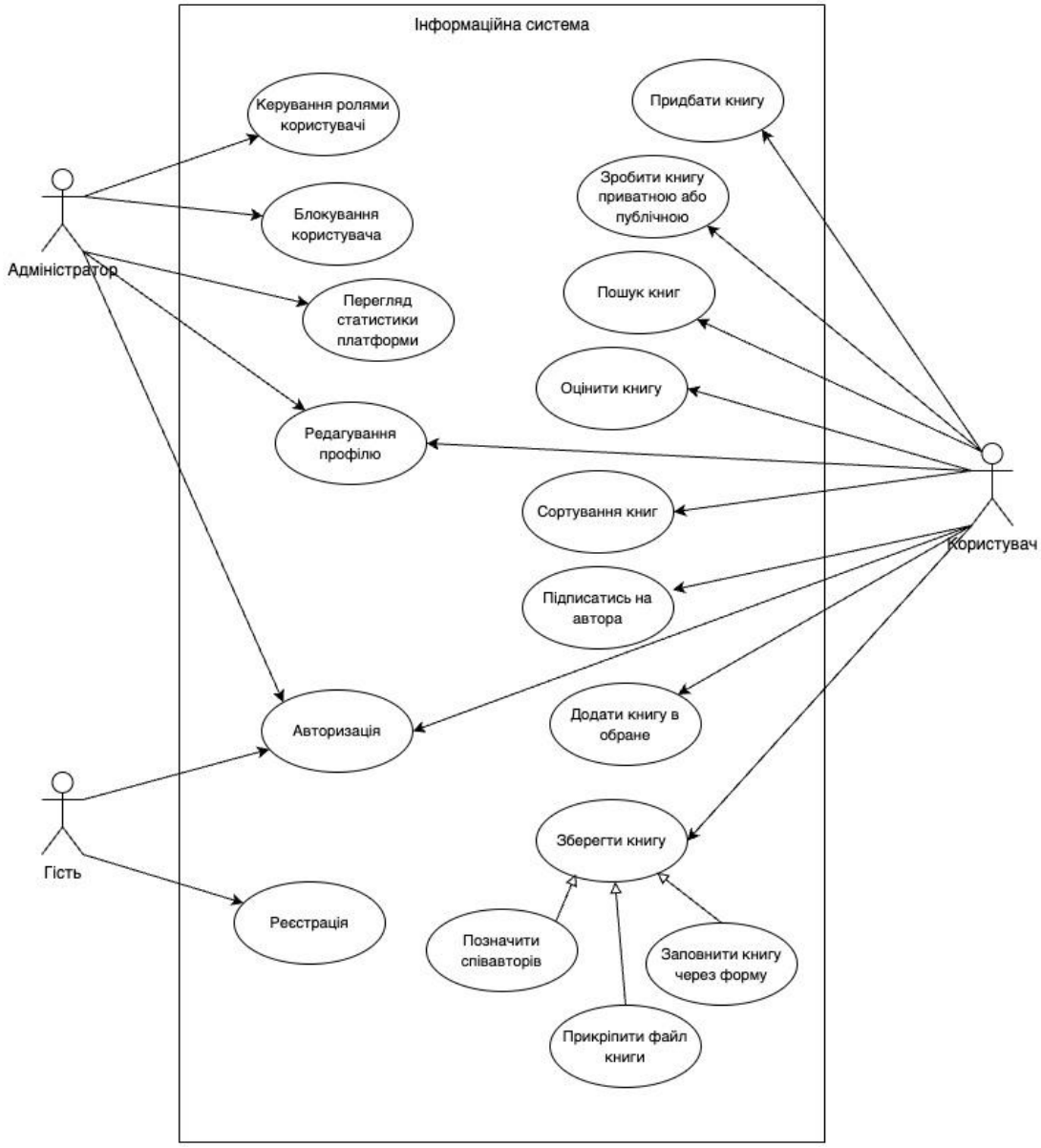


Рис. 2.1. Діаграма варіантів використання

Детальний опис основних варіантів використання для програмно-апаратної платформи інформаційного забезпечення процесу обміну книгами приведено нижче:

- «Реєстрація та авторизація в системі». Цей варіант використання необхідний для того, щоб гість міг створити обліковий запис і авторизуватися на платформі. Під час реєстрації гість заповнює форму з обов'язковими полями, такими як адреса електронної пошти та пароль. Якщо всі дані введені правильно і підтверджені системою, обліковий запис створюється і гість отримує повідомлення з підтвердженням. У разі відсутності або неправильності даних (наприклад,

слабкий пароль або дублікат електронної пошти), система повідомляє користувача про необхідність внести виправлення. Після реєстрації гість може увійти в систему, вказавши свою електронну пошту та пароль. Якщо облікові дані дійсні, користувач буде авторизований і перенаправлений на основний інтерфейс.

- «Зберігання та керування книгами». Цей варіант використання дозволяє користувачеві зберігати, редагувати і видаляти книги на платформі. При додаванні нової книги користувач заповнює форму, що містить такі поля, як: назва, жанр, опис, кількість сторінок, співавтори, та налаштування конфіденційності (приватний/публічний). Після заповнення форми завантажує файлу книги (наприклад, PDF) та зображення обкладинки. Після заповнення всіх даних система перевіряє введені дані і зберігає книгу в базі даних. Якщо якісь обов'язкові поля відсутні, користувач отримує сповіщення про необхідність заповнити форму. Пізніше користувач може відредагувати дані про книгу або повністю видалити її. У всіх випадках система оновлює запис про книгу і підтверджує операцію.

- «Пошук книг». Цей варіант використання дозволяє Користувачеві шукати книги, що зберігаються в системі. Користувач вводить критерії пошуку, такі як назва книги, жанр або ім'я автора в рядок пошуку. Система обробляє запит і видає релевантні результати на основі введених критеріїв. Якщо знайдено відповідні книги вони відображаються у вигляді списку із зазначенням таких важливих деталей, як назва, автор і рейтинг. Якщо результатів не знайдено, система сповіщає користувача про те, що книг які відповідають запиту не знайдено.

- «Оцінювання та сортування книг». Цей варіант використання дозволяє користувачеві оцінювати книги і сортувати їх за рейтингом або популярністю. Коли користувач обирає книгу, він може поставити їй оцінку а саме значення від 1 до 5, щоб відобразити свою думку. Система перевіряє введені дані і оновлює середній рейтинг книги. Користувачі також мають доступ до опцій сортування для відображення книг.

- «Підписка на авторів». Цей варіант використання дозволяє користувачеві підписатися на автора, щоб отримувати повідомлення про нові публікації. Користувач переходить до профілю автора і вибирає опцію

«Підписатися». Система реєструє підписку і оновлює профіль користувача. Коли підписаний автор публікує нову книгу, система генерує повідомлення для всіх підписників. Користувачі отримують сповіщення в особистому кабінеті.

- «Керування ролями користувачів». Цей варіант використання дозволяє адміністратору керувати ролями користувачів. Адміністратор отримує доступ до інтерфейсу «Керування ролями», де він бачить список усіх зареєстрованих користувачів. Адміністратор вибирає конкретного користувача і оновлює його роль (наприклад, призначає роль «Адміністратор» звичайному користувачеві). Після внесення змін система підтверджує операцію, зберігає оновлену роль і повідомляє про це відповідного користувача.

## 2.5. Вибір технологій для розробки API

Вибір відповідних технологій відіграє ключову роль у забезпеченні ефективності, масштабованості та підтримки сучасної програмно-апаратної платформи. Для платформи обміну книгами обрані технології мають підтримувати основні функціональні можливості, такі як безпечне управління даними, зручні інтерфейси користувача та безперебійна взаємодія між компонентами системи. Дальше буде описано основні технологія які будуть вибрані для реалізації серверної частини застосунку. Також буде реалізовано базовий інтерфейс користувача для мануального тестування та графічного представлення можливостей системи. В якості графічного інтерфейсу буде розроблено веб-застосунок.

Для зберігання даних буде використано MongoDB, це документно-орієнтована NoSQL база даних, була обрана як основна база даних для платформи [21]. Її гнучкість і масштабованість роблять її ідеальною для управління різноманітними й неструктурованими типами даних, що зазвичай асоціюються з платформами обміну книгами. MongoDB зберігає дані у форматі BSON (Binary JSON), що добре узгоджується з ієрархічною природою записів про книги, зокрема полями, такими як заголовки, жанри, автори, оцінки та деталі співавторів.

Основні переваги MongoDB включають:

- MongoDB підтримує горизонтальне масштабування за допомогою шардінгу, забезпечуючи ефективну продуктивність у міру зростання обсягу книг, облікових записів користувачів і рейтингів.
- Її схема без фіксованої структури дозволяє легко додавати нові поля без порушення існуючих структур даних, що особливо важливо при розвитку платформи.
- Запити до MongoDB є швидкими й ефективними, що дозволяє реалізувати функції пошуку книг, сортування та фільтрації за параметрами користувача.

Крім того, фреймворк агрегації MongoDB дозволяє виконувати складні запити, наприклад обчислення середнього рейтингу книг або виявлення популярних авторів, з мінімальною затримкою.

Для забезпечення безпечної автентифікації користувачів і захисту конфіденційних даних використовується bcrypt для хешування паролів. Bcrypt — це широко вживана бібліотека хешування паролів, що включає соль, завдяки чому вона стійка до атак методом перебору й використання таблиць відповідностей.

Основні особливості bcrypt:

- завдяки можливості налаштовувати обчислювальну складність (work factor), bcrypt адаптується до сучасного обладнання, забезпечуючи високий рівень безпеки;
- соль гарантує, що навіть однакові паролі матимуть унікальні хеші, що підвищує рівень захисту.
- bcrypt легко інтегрується з Node.js, забезпечуючи безпечну обробку облікових даних користувачів під час реєстрації та входу

Використання bcrypt забезпечує надійний захист паролів користувачів навіть у разі несанкціонованого доступу до бази даних.

Для мануального тестування та представлення графічного інтерфейсу користувача буде використано описані нижче технології.

Веб-застосунок платформи буде розроблено за допомогою React, популярної JavaScript-бібліотеки для створення динамічних і зручних інтерфейсів користувача. React був обраний через його можливість створювати багаторазові компоненти, що спрощує розробку й тестування фронтенду.

Переваги використання React:

- react дозволяє створювати модульні, багаторазові компоненти для функцій, таких як картки книг, панелі пошуку та профілі користувачів, що полегшує підтримку й масштабування коду.
- Віртуальний DOM React оптимізує продуктивність рендерингу, забезпечуючи ефективне оновлення інтерфейсу (наприклад, відображення книг або сповіщень).
- Велика екосистема React, включно з React Developer Tools і бібліотеками, такими як React Router спрощує процес розробки.

React слугуватиме основним фреймворком фронтенду для взаємодії з API, дозволяючи тестувати функціональність бекенду.

Для ефективної взаємодії з GraphQL API у React-застосунок буде інтегровано бібліотеку @apollo/client. Apollo Client спрощує процес надсилання запитів і мутацій до сервера та управління локальним станом та кешем застосунку.

Переваги Apollo Client:

- Інтуїтивний спосіб виконання GraphQL-запитів і мутацій для динамічного отримання та маніпуляції даними.
- Вбудований механізм кешування зменшує кількість зайвих запитів, покращуючи загальну продуктивність.
- Містить хороші інструменти для обробки помилок і налагодження, спрощуючи розробку та тестування.

Використання Apollo Client у поєднанні з React забезпечить ефективну взаємодію фронтенду з бекендом, зокрема реальні оновлення, як-от завантаження нових книг, оцінок і сповіщень.

Щоб забезпечити хороший, інтуїтивно зрозумілий та адаптивний інтерфейс користувача, для стилізації фронтенду буде використано Bootstrap. Bootstrap надає

готові компоненти (модальні вікна, кнопки, форми) та систему сіток, що дозволяє швидко розробляти інтерфейс із мінімальними зусиллями.

Інтеграція MongoDB, Node.js, bcrypt, React і Apollo Client забезпечує безперебійний процес розробки для компонентів як севера, так і клієнта. Node.js слугуватиме середовищем виконання для бекенду, обробляючи GraphQL-запити, автентифікацію користувачів (захищену bcrypt) та взаємодіючи з базою даних MongoDB для зберігання й отримання даних. На клієнті React використовуватиме Apollo Client для взаємодії з API, забезпечуючи зручний та інтуїтивний інтерфейс користувача. Bootstrap удосконалить дизайн платформи, забезпечуючи доступність на різних пристроях.

## 2.6. Проєктування архітектури бази даних

Створення продуманої схеми бази даних є одним із ключових чинників забезпечення ефективної роботи будь-якої складної інформаційної системи. Для платформи з обміну книжками це особливо актуально, оскільки правильне структурування даних гарантує не лише швидкий пошук та обробку інформації, а й логічне відображення реальних взаємодій між користувачами, книжками та іншими елементами системи. Збалансований дизайн бази даних допомагає підтримувати узгодженість, цілісність і передбачуваність структури, одночасно спрощуючи подальший розвиток та адаптацію.

У рамках платформи для книгообміну схему бази даних необхідно формувати так, щоб вона чітко відображала взаємозв’язки між користувачами які можуть підписуватися на авторів, оцінювати видання та взаємодіяти з різними книжковими ресурсами. Ретельне моделювання цих взаємодій підвищує надійність даних і покращує загальну продуктивність, оскільки кожна сутність та її стосунки з іншими стають прозорими та логічно обґрунтованими. Оптимальна схема не лише запобігає надмірності чи аномаліям, а й дозволяє легко формувати складні запити, необхідні для повноцінного функціонування платформи.

Нижче буде детально розглянуто основні сутності системи, їхні поля та взаємозв’язки, які забезпечують єдиний узгоджений простір даних. Такий підхід гарантує, що база даних залишатиметься гнучкою та стабільною, готовою до розширення функціональності й зростання навантаження. Завдяки чітко визначеним сутностям та їх логічним зв’язкам цей розділ створює міцне підґрунтя для ефективної й надійної системи управління даними.

Нижче буде детально розглянуто сутність користувача, оскільки це є одна з базових сутностей системи. Дана сутність має зберігати наступну інформацію: імейл користувача, захешований пароль, роль користувача, інформацію про вибрані книги користувача на його підписників.

Схему сутності користувача можна побачити на рисунку 2.2.

| **User** | |
|---|---|
| _id | ObjectId |
| hashPassword | String |
| email | String |
| favoriteBookIds | Array<ObjectId> |
| subscriberIds | Array<ObjectId> |
| createdAt | Date |
| updatedAt | Date |
| role | String |

Рис. 2.2. Схема сутності користувача

Сутність користувача буде містити наступні поля:

- _id – ідентифікатор користувача;
- hashPassword – захешовний пароль;
- email – електронна пошта користувача;
- favoriteBookIds – масив унікальних ідентифікаторів книг, які користувач додав в список обраних;

- subscriberIds – масив ідентифікаторів користувачів які підписані на даного користувача;
- role – поле яке вказує на роль користувача, воно може мати значення або «administrator», або «user»;
- createdAt – поле яке вказує на дату, коли даний запис було створено;
- updatedAt – поле яке вказує на дату, коли даний запис було останній раз оновлено.

Ще однією важливою сутністю в системі є книга. Ця сутність призначена для зберігання даних про книги, вона повинна зберігати наступну інформацію: назву книги, посилання на фотографію обкладинки книги, жанр, інформацію про співавторів якщо вони є, кількість сторінок, вартість якщо книга є платною, опис та посилання на файл з вмістом книги, інформацію про те, чи книга є приватною, та посилання на автора книги.

Схему сутності книги можна побачити на рисунку 2.3

| **Book** | |
|---|---|
| _id | ObjectId |
| name | String |
| img | String |
| genre | String |
| otherAuthorNames | Array<String> |
| pageQuantity | Int32 |
| isPaid | Bool |
| price | Int32 |
| description | String |
| bookURL | String |
| isPrivate | Bool |
| createdAt | Date |
| updatedAt | Date |
| authorId | ObjectId |

Рис. 2.3. Схема сутності книги

Сутність книги буде містити наступні поля:

- _id – ідентифікатор книги;
- name – назва книги;
- img – посилання на фото обкладинки книги;
- genre – жанр книги;
- otherAuthorNames – масив з іменами співавторів, якщо вони є;
- pageQuantity – кількість сторінок;
- isPaid – логічне значення яке вказує чи книг є платною;
- price – вартість якщо книга платна;
- description – опис книги;
- bookURL – посилання на вміст книги;
- isPrivate – логічне значення яке вказує чи книга є приватною;
- authorId – унікальний ідентифікатор автора книги;
- createdAt – поле яке вказує на дату, коли книгу було створено;
- updatedAt – поле яке вказує на дату, коли книгу було останній раз оновлено.

Також важливою сутністю є профіль, в нього зберігаються дані про користувача. Ця сутність повинна зберігати наступну інформацію: псевдонім користувача, фотографію профіля, прізвище та ім’я користувача якщо він захоче їх вказувати, електрону пошту для листування, номер телефону, та інформацію яку користувач хоче надати про себе.

Схему сутності профіля наведено на рисунку 2.4.

| Profile | |
|---|---|
| _id | ObjectId |
| userId | ObjectId |
| nickname | String |
| profilePhoto | String |
| fullName | String |
| email | String |
| phoneNumber | String |
| aboutMyself | String |
| createdAt | Date |
| updatedAt | Date |

Рис. 2.4. Схема сутності профілю користувача

Сутність профіля буде містити наступні поля:

- _id – ідентифікатор профіля;
- userId – ідентифікатор користувача до якого відноситься профіль;
- nickname – псевдонім користувача;
- fullName – прізвище та ім’я користувача;
- email – адреса електронної пошти користувача яка буде видима і призначена для листування;
- profilePhoto – посилання на фотографію профіля користувача;
- phoneNumber – номер телефону користувача;
- aboutMyself – інформація яку користувач може написати про себе;
- createdAt – поле яке вказує на дату, коли профіль створено;
- updatedAt – поле яке вказує на дату, коли профіль було останній раз оновлено.

У системі буде також присутня сутність яка буде відповідати за рейтинг книг. Дана сутність повинна зберегти наступну інформацію: посилання на користувача

який оцінив книгу, також посилання на книгу яку було оцінено, також дату коли дану оцінку було зроблено.

Схема сутності рейтингу зображена на рисунку 2.5.

| RatingsForBook | |
|---|---|
| _id | ObjectId |
| userId | ObjectId |
| bookId | ObjectId |
| rating | Int32 |
| createdAt | Date |

Рис. 2.5. Схема сутності оцінки книги

Сутність оцінки книги буде містити наступні поля:

- _id – ідентифікатор рейтингу;
- userId – ідентифікатор користувача який оцінив книгу;
- bookId – ідентифікатор книги яку було оцінено;
- rating – оцінка книги;
- createdAt – дата коли оцінку було зроблено.

Останньою з основних сутностей в систему буде сутність повідомлення про реліз книги. Ця сутність повинна зберігати наступну інформацію: посилання на користувача якому слід відіслати повідомлення, книга яку випустили, також інформацію про те, чи повідомлення було прочитане, та дану створенні та оновлення повідомлення.

Схема сутності повідомлення про випуск книги зображено на рисунку 2.6.

| NoticesAboutBookReleased | |
|---|---|
| _id | ObjectId |
| userId | ObjectId |
| bookId | ObjectId |
| isRead | Bool |
| createdAt | Date |
| updatedAt | Date |

Рис. 2.6. Схема сутності повідомлення про випуск книги

Сутність повідомлення про випуск книги буде мати наступні поля:

- _id – ідентифікатор повідомлення;
- userId – ідентифікатор користувача якого слід повідомити;
- bookId – ідентифікатор книги яку було випущено;
- isRead – логічне значення яке вказує на те, чи повідомлення було прочитано;
- createdAt – поле яке вказує на дату, коли повідомлення було створено;
- updatedAt – поле яке вказує на дату, коли повідомлення було останній раз оновлено.

Після того як було спроєктовано основні сутності системи було розроблено схему бази даних (рис. 2.7).

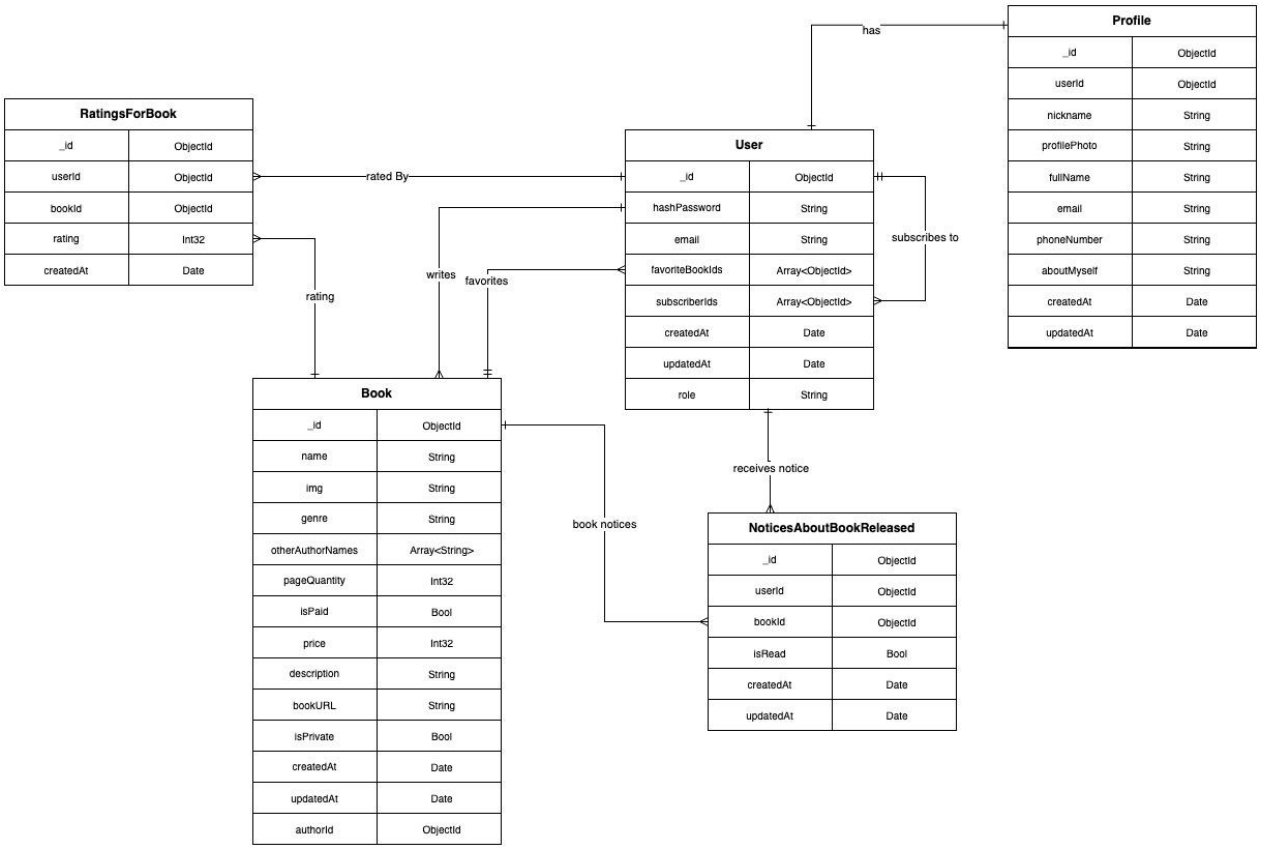


Рис 2.7. Схема бази даних

На схемі бази даних зображено основні сутності системи та зв’язки між ними. Відношення між сутностями в системі:

- сутність користувача має зв’язок один до багатьох з сутністю книги, оскільки користувач може випустити декілька книг.
- Сутність користувача яка представляє автора має зв’язок багато до багатьох з такою ж сутністю користувача яка представляє підписника, оскільки один автор може мати багатьох підписників, так само користувач може бути підписаний на багатьох авторів.
- Сутність користувача також має відношення багато до багатьох з сутністю обраних книг, оскільки один користувач може додати багато книги в обране, так самок одна книга може бути в списку обраного в багатьох користувачів.
- Сутність користувача має відношення одне до одного з сутністю профіля, оскільки один користувач має один профіль.

- Сутність користувача має відношення один до багатьох з сутністю оцінки книги, оскільки один користувач може поставити оцінку багатьом книгам.
- Сутність користувача має відношення один до багатьох з сутністю повідомлення про випуск книги, оскільки один користувач може бути підписаний на багатьох авторів і отримувати багато повідомлень.
- Сутність книги пов’язана відношенням один до багатьох з сутністю оцінки, оскільки одна книга може мати велику кількість оцінок від різних користувачів.
- Сутність книги пов’язана відношенням один до багатьох з сутністю повідомлення про випуск книги оскільки на автора який випустив книгу може бути підписано багато користувачів.

## 2.7. Висновки до розділу

У цьому розділі наведено основні кроки, що забезпечили створення надійного та безпечного рішення для платформи. Теоретичні аспекти розробки API допомогли обрати відповідні підходи, зокрема асинхронну обробку запитів, валідацію схем і кешування, завдяки яким платформа здатна ефективно працювати з великими обсягами даних.

Формування базових вимог і чітке визначення варіантів використання дозволять забезпечити реалізацію ключових функцій, зокрема зберігання книг, управління користувачами та сповіщеннями. Застосування таких технологій як: MongoDB, bcrypt, а також інструментів для клієнтської частини, таких як React та Apollo Client, разом з продуманою архітектурою бази даних, створює гнучке та продуктивне середовище для подальшого розвитку платформи. Цей інтегрований підхід дає змогу не лише оптимізувати обробку даних і підвищити рівень безпеки, а й забезпечити якісний користувацький досвід. Завдяки цьому платформа залишається готовою до масштабування, інтеграцій з новими сервісами та задоволення зростаючих потреб аудиторії.

# РОЗДІЛ 3
# РАЛІЗАЦІЯ ПЛАТФОРМИ ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ ПРОЦЕСУ ОБМІНУ КНИГАМИ З ВИКОРИСТАННЯМ ОБРАНИХ МЕТОДІВ

## 3.1. Реалізація основних класів серверної частини

Після того як було проведено аналіз методів для розробки API та спроєктовано архітектуру бази даних а також теоретично досліджено сферу розробки платформ для забезпечення інформаційного процесу обміну книгами. Слід розпочати реалізацію основник класів серверної частини для забезпечення роботи бізнес логіки платформи.

Одним з базових класів серверної частини буде AuthService, він призначений для генерації та валідації токенів доступу для подальшої реалізації системи авторизації в системі.

Основними методами цього класу є метод для генерації токену доступу в якому будуть закодовані дані користувача для того, щоб розпізнавати його в системі. Також важливим є метод для генерації рефреш-токену, він призначений для того, щоб коли термін дії токена доступу закінчився використати рефреш-токену і по ньому згенерувати новий токен доступу не змушуючи користувача авторизуватись знову. В класі також важливим є метод для перевірки токена доступу і отримання даних користувача на його основі. Ці методи наведені на рисунку 3.1.

```
generateAccessToken(user) {
  const payload = { _id: user._id, email: user.email };
  return jwt.sign(payload, config.jwtAccessTokenSecret, { expiresIn: '50m' });
}

generateRefreshToken(user) {
  const payload = { _id: user._id, email: user.email };
  const refreshToken = jwt.sign(payload, config.jwtRefreshTokenSecret, { expiresIn: '100d' });
  this.refreshTokens.push(refreshToken);
  return refreshToken;
}

verifyAccessToken(accessToken) {
  return jwt.verify(accessToken, config.jwtAccessTokenSecret);
}
```

Рис. 3.1. Основні методи класу AuthService

Метод generateAccessToken приймає об’єкт користувача і на основі його даних генерує токен доступу. Також об’єкт користувача приймає метод generateRefreshToken але він на основі його даних генерує рефреш-токен, а метод verifyAccessToken перевіряє токен доступу і декодує з нього дані користувача.

Для того, щоб авторизація була повністю реалізована слід написати функцію, яка буде з токена доступу декодувати користувача і вставляти його в контекст, щоб в подальшому його можна було використати у резолверах, щоб знати який саме користувач зробив цей запит, реалізацію цієї функції наведено на рисунку 3.2.

```
const getUserContext = (req) => {
  const ctx = {
    jwtUser: null,
    getUser: () => null,
    userId: null,
  };
  const authHeader = req.headers.authorization;
  const token = authHeader;
  if (!token) {
    return ctx;
  }
  try {
    const jwtUser = AuthService.verifyAccessToken(token);
    return {
      ...ctx,
      jwtUser,
      getUser: () => {
        return UserService.findById(jwtUser._id);
      },
      userId: jwtUser._id,
    };
  } catch (e) {
    return ctx;
  }
};
```

Рис. 3.2. Функція для отримання даних користувача на основі токена доступу

Черговим базовим класом серверної частини є UserService, цей клас відповідає за роботу з сутністю користувача, а також використовується для системи авторизації. Наприклад метод createAccount використовується для створення при реєстрації користувача (рис 3.3).

```
async createAccount({ email, password }) {
  let user = await this.findByEmail(email);
  if (user) {
    throw new Error('user with this email already registered');
  }
  if(!this.verifyUser(user)){
    throw new Error('User data does not meet the requirements')
  }
  const hashPassword = await bcrypt.hash(password, 10);
  user = { hashPassword, email, createAt: new Date() };
  const result = await this.getCollection().insertOne(user);
  await ProfileService.createProfile({ userId: result.insertedId, email });
}
```

Рис. 3.3. Метод для створення акаунта користувача

Метод для створення акаунту користувача перевіряє чи користувача з такою електронною поштою немає в систему, також перевіряється правильність даних, якщо всі перевірки пройдено, то дані користувача зберігаються в БД, також створюється профіль для нього.

В класі UserService також є метод для авторизації в системі під назвою loginWithPassword (рис 3.4).

```
async loginWithPassword({ email, password }) {
  const user = await this.findByEmail(email, true);
  if (!user) {
    throw new Error('User not found');
  }
  const isPasswordCorrect = await bcrypt.compare(password, user.hashPassword);
  if (!isPasswordCorrect) {
    throw new Error('incorrect password');
  }
  const accessToken = AuthService.generateAccessToken(user);
  const refreshToken = AuthService.generateRefreshToken(user);
  const userWithoutPrivateFields = this.#omitPrivateFields(user);
  return { accessToken, refreshToken, user: userWithoutPrivateFields };
}
```

Рис. 3.4. Метод для авторизації користувача в системі

В даному методі перевіряється, чи користувач існує в БД, також перевіряється пароль, якщо все правильно то генеруються токени та повертаються разом з об’єктом користувача. Даний клас має ще декілька методів для роботи з сутністю користувача, такі як повернення списку користувачі з використанням пагінації та інші. Більш детально з іншими методами даного класу можна ознайомитися в додатках до роботи.

Важливим класом в системі є також BookService він призначений для роботи з сутністю книги, цей клас містить велику кількість методів, одним з основних є createBook він призначений для створення книги (рис. 3.5).

```
createBook = async (book, authorId) => {
  if (!this.verifyBook(book)) throw new Error('Book data does not meet the requirements');

  const bookData = this.prepareData(book, authorId);
  const result = await BooksCollection.insertOne(bookData);
  await SubscriptionsService.bookReleased({ authorId, bookId: result.insertedId });
  return { ...book, _id: result.insertedId };
};
```

Рис. 3.5. Метод для створення книги

Метод який зображений на рисунку приймає дані про книгу та ідентифікатор автора, після чого перевіряє чи дані книги відповідають вимогам, тоді викликається метод для підготування даних, він повертає об’єкт книги і з доданими до них мета даними після чого книга зберігається в БД і викликається метод bookReleased для того, щоб повідомити про випуск книги.

В класі BookService також є важливим метод searchBooks який призначений для пошуку книг (рис. 3.6).

```
searchBooks = async (lineForSearch, userId) => {
  const books = await BooksCollection
    .find({
      $or: [
        { name: { $regex: lineForSearch, $options: 'i' } },
        { description: { $regex: lineForSearch, $options: 'i' } },
      ],
      isPrivate: false,
    })
    .sort({ createdAt: -1 })
    .toArray();
  return RatingService.userCanAddRatingForBooks({ userId, bookList: books });
};
```

Рис. 3.6. Метод для пошуку книг

Наведений на рисунку метод використовується для пошуку всіх книжок опис або назва яких відповідає заданому рядку. Детальніше усіма з методами цього класу можна ознайомитися в додатках.

Слід також розглянути клас RatingService, цей клас використовується в системі для збереження та розрахування рейтингу, а також для сортування по ньому. Метод addRatingForBooks призначений для збереження рейтингу для книги в БД (рис. 3.7).

```
addRatingForBooks = async ({ bookId, userId, rating }) => {
  return RantingsForBooksCollection.insertOne({
    bookId: new ObjectID(bookId),
    userId: new ObjectID(userId),
    rating,
    createAt: new Date(),
  });
};
```

Рис. 3.7. Метод для збереження рейтингу для книги

Даний метод приймає ідентифікатор книги і користувача, а також рейтинг який їх поставив користувач і зберігає ці дані в БД.

Одним із головних методів в цьому класі є calculateRatingForBook він призначений для вираховування середнього рейтингу для книги на основі даних з БД (рис. 3.8).

```
calculateRatingForBook = async (bookId) => {
  const result = await RantingsForBooksCollection
    .aggregate([{ $group: { _id: '$bookId', avg: { $avg: '$rating' } } }, { $match: { _id: bookId } }])
    .toArray();
  const [rating] = result;
  if (!rating) return 0;
  return rating.avg;
};
```

Рис. 3.8. Метод для розрахунку середнього рейтингу книги

Метод який продемонстровано на рисунку приймає ідентифікатор книги для якої слід розрахувати рейтинг та знаходить всі записи в колекції рейтингу і за допомогою агрегації та оператора $avg, який призначений для розрахунку середнього значення знаходить значення рейтингу для даної книги, якщо в БД немає жодного запису для книги, тобто ще ніхто не ставив їй оцінку то даний метод поверне 0.

Важливим серверним класом є також NoticesAboutReleasedService він призначений для того, щоб підписники автора могли отримувати повідомлення про випуск книги. Нижче розглянемо два важливі методи цього класу, першим є метод bookReleased, він викликається коли автор випускає книгу та призначений для створення повідомлень для всіх підписників і збереження цих повідомлень в БД (рис. 3.9).

```
bookReleased = async ({ authorId, userIdList, bookId }) => {
  const dataForInsert = userIdList.map((item) => {
    return {
      userId: new ObjectId(item.userId),
      authorId: new ObjectId(authorId),
      bookId: new ObjectId(bookId),
      isRead: false,
      createdAt: new Date(),
    };
  });
  return NoticesAboutBookReleasedCollection.insertMany(dataForInsert);
};
```

Рис. 3.9. Метод для створення повідомлення про випуск книги

Даний метод приймає ідентифікатор автора, також книги і його підписників, після чого для кожного з підписників створюється запис (повідомлення) в БД.

Важливим також є метод getNoticeForReleasedBook, він призначений для того, щоб повернути всі повідомлення для користувача (рис. 3.10).

```
getNoticeForReleasedBook = async (userId) => {
  return NoticesAboutBookReleasedCollection
  .find({ userId: new ObjectId(userId) })
  .toArray();
};
```

Рис. 3.10. Метод для отримання всіх повідомлень користувача

Метод який зображено на рисунку вище приймає ідентифікатор користувача та повертає всі повідомлення про випуск книг для нього.

Детальніше з серверними класами такими як: SubscriptionsService, FavoriteService та іншими, а також з їхніми методами можна ознайомитися в додатках до роботи.

## 3.2. Побудова GraphQL API

Створення добре організованого GraphQL API є важливим етапом у розробці платформи для обміну книжками. GraphQL надає гнучкий і точний спосіб пошуку даних, уникаючи зайвих запитів і покращуючи ефективність використання ресурсів сервера. Правильне визначення схеми GraphQL гарантує чіткі запити, точну валідацію та зрозумілі зв'язки між сутностями (книгами, користувачами, сповіщеннями), що сприяє узгодженості та гнучкості системи. Завдяки цьому платформу можна легко масштабувати та підтримувати реальну взаємодію користувачів і полегшувати майбутній розвиток.

Базовою буде схема для сутності користувача, оскільки навколо неї зв'язана вся бізнес логіка системи. Вигляд схеми користувача зображено на рисунку 3.11.

```
export const typeDefs = gql`
  extend type Query {
    users(searchString: String): [User!]!
    userBooks(userId: ID!): [Book]!
    userById(userId: ID!): User!
  }

  type User {
    _id: ID!
    email: String!
    profile: Profile!
    books: [Book!]!
    subscribers: [User!]!
    favoriteBooks: [Book!]!
  }
`;
```

Рис. 3.11. GraphQL схема користувача

На рисунку зображено схему для користувача, в ній є три запити: users - повертає масив всіх користувачів в системі, userBooks – повертає книги по ідентифікатору автора, userById – повертає користувача по його ідентифікатору. Сам тип User має особливі поля такі як:

- profile, який призначений для зберігання даних профілю користувача;
- books в цьому масиві зберігаються всі книги для яких даний користувач є автором;
- subscribers це поле представляє масив підписників користувача;

- favoriteBooks є масивом обраних книг користувача.

Для даної схеми слід створити резолвери, тобто функції які будуть викликатися під час виконання запиту, або якщо в схемі яка приходить з клієнту застосовується поле і його слід обробити. Опис резолверів для схеми користувача наведено на рисунку 3.12.

```
export const resolvers = {
  Query: {
    users: async (_, { searchString }, context) => {
      isAuthorizedUser(context);
      return UserService.users({ searchString });
    },
    userBooks: async (_, { userId }, context) => {
      isAuthorizedUser(context);
      return BookService.getBooksByUserId(userId);
    },
    userById: async (_, { userId }, context) => {
      isAuthorizedUser(context);
      return UserService.findById(userId);
    },
  },
  User: {
    profile: async ({ _id }) => {
      return ProfileService.getProfileByUserId(_id);
    },
    books: async ({ _id }) => {
      return BookService.getBooksByUserId(_id);
    },
    subscribers: async ({ subscriberIds }) => {
      return UserService.getUsersByIds(subscriberIds);
    },
    favoriteBooks: async ({ favoriteBookIds }) => {
      return BookService.getBooksByIds(favoriteBookIds);
    },
  },
};
```

Рис. 3.12. Резолвери для схеми користувача

В резолверах які наведені на рисунку виконуються запити, а також для полів які є об’єктами або масивами на основі ідентифікаторів повертаються необхідні дані.

Важливою є також GraphQL схема, яка використовується для реєстрації та авторизації користувачів в системі (рис. 3.13).

```
export const typeDefs = gql`
  extend type Query {
    me: User
  }

  type AuthResponse {
    accessToken: String!
    refreshToken: String
  }

  extend type Mutation {
    login(email: String!, password: String!): AuthResponse!
    logout(token: String!): Boolean
    registration(email: String!, password: String!): AuthResponse!
    token(refreshToken: String!): AuthResponse!
  }
`;
```

Рис. 3.13. Схема для реєстрації та авторизації

В схемі, яка зображена на рисунку є запит для отримання даних авторизованого користувача, а також мутації для авторизації та реєстрації, також для виходу з системи і оновлення токену.

Резолвери які слугують для обробки запитів зображено на рисунку 3.14.

```
export const resolvers = {
  Query: {
    me: async (root, params, context) => {
      const user = await context.getUser();
      return user;
    },
  },

  Mutation: {
    login: async (root, { email, password }) => {
      const { accessToken, refreshToken } = await UserService.loginWithPassword({ email, password });
      return { accessToken, refreshToken };
    },
    logout: (root, { token }) => {
      UserService.logout(token);
      return true;
    },
    registration: async (root, { email, password }) => {
      await UserService.createAccount({ email, password });
      const { accessToken, refreshToken } = await UserService.loginWithPassword({ email, password });
      return { accessToken, refreshToken };
    },
    token: async (root, { token }) => {
      const accessToken = UserService.loginWithRefreshToken(token);
      return { accessToken };
    },
  },
};
```

Рис. 3.14. Резолвери для реєстрації та авторизації в системі

В зображених обробниках приймаються запити та викликаються методи для класу UserService, щоб створити акаунт користувача при реєстрації, або токени, якщо викликається запит авторизації.

Останньою продемонстрованою буде схема для книг, вона призначена для обробки запитів пов’язаних з книгами (рис. 3.15).

```
export const typeDefs = gql`
  type Book {
    _id: ID!
    name: String!
    img: String!
    authorId: ID!
    author: User!
    genre: String!
    otherAuthors: [String]!
    pagesQuantity: Int!
    isPaid: Boolean!
    price: Float
    rating: Float!
    description: String!
    userCanAddRating: Boolean!
    bookURL: String!
    createAt: Float!
    updateAt: Float!
    isPrivate: Boolean!
    isFavorite: Boolean
  }

  input BookCreateInput {
    name: String!
    img: String!
    genre: String!
    otherAuthors: [String]!
    pagesQuantity: Int!
    isPaid: Boolean!
    price: Float
    description: String!
    bookURL: String!
    isPrivate: Boolean!
  }
  input BookUpdateInput {
    name: String
    img: String
    genre: String
    otherAuthors: [String]
    pagesQuantity: Int
    isPaid: Boolean
    price: Float
    description: String
    bookURL: String
    isPrivate: Boolean
  }

  extend type Query {
    books(searchString: String!): [Book]!
    book(id: ID!): Book!
  }

  extend type Mutation {
    createBook(input: BookCreateInput!): Book!
    addRatingForBook(bookId: ID!, rating: Int!): Book!
    deleteBook(bookId: ID!): Boolean!
    updateBook(bookId: ID!, input: BookUpdateInput!): Book!
    changePrivacyOfBook(bookId: ID!, isPrivate: Boolean!): Book!
    addBookToFavorites(bookId: ID!): Boolean!
    removeBookFromFavorites(bookId: ID!): Boolean!
  }
`;
```

Рис. 3.15. Схема для книг

В зображені схемі описано всі необхідні поля для сутності книги, також два об’єкта: BookCreateInput та BookUpdateInput, перший використовується для створення книги, другий для її оновлення. Також в схемі описано запити для отримання однієї книги та всіх книг, існують і ще мутація призначення який модифікація даних пов’язаних з книгою. Резолвери для даної схеми можна знайти в додатках, оскільки вони занадто великі для представлення в основній частині.

Детальний опис схем таких як: профілю, підписок та інші, а також їхні резолвери наведено в додатках до роботи.

## 3.3. Тестування та виправлення помилок в роботі API

В цьому розділі буде проведено тестування основних функції серверного API. Для початку потрібно протестувати функцію реєстрації в системі (рис. 3.16).

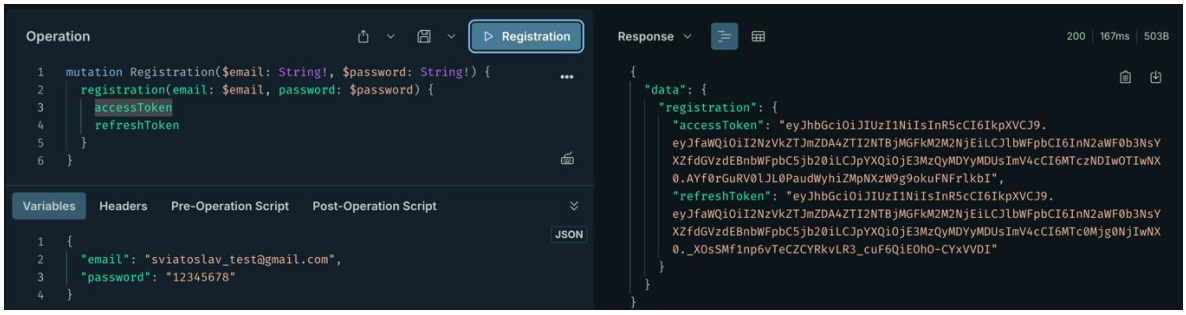


Рис. 3.16. Функція реєстрації в системі

Як можна спостерігати на рисунку, після реєстрації повертаються два токени, а отже акаунт для користувача успішно створено.

Важливим також є запит для авторизації, його тестування можна спостерігати на рисунку 3.17.

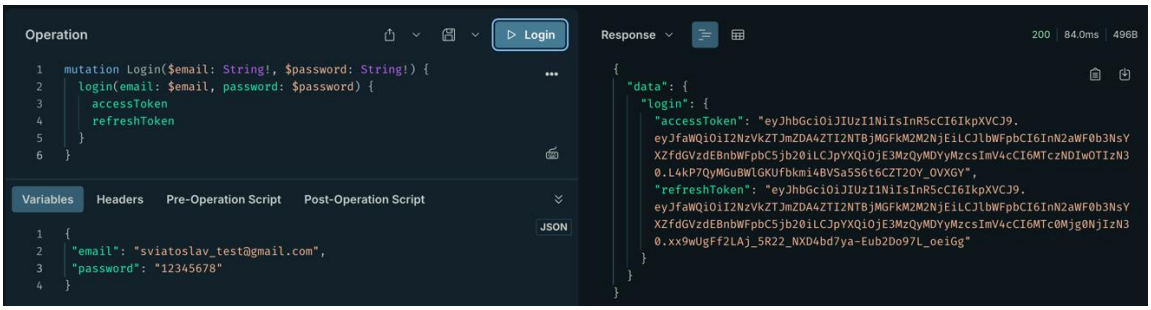


Рис. 3.17. Запит авторизації в системі

Після тестування запитів для входу в систему токени, які були повернуті можна використати для виконання інших запитів уже в ролі авторизованого користувача.

Вважливою мутацією, яку слід протестувати буде створення книги, оскільки вона є базовою для програмно-апаратної платформи інформаційного забезпечення процесу обміну книгами. Виконання мутації для створення книги зображено на рисунку 3.18.

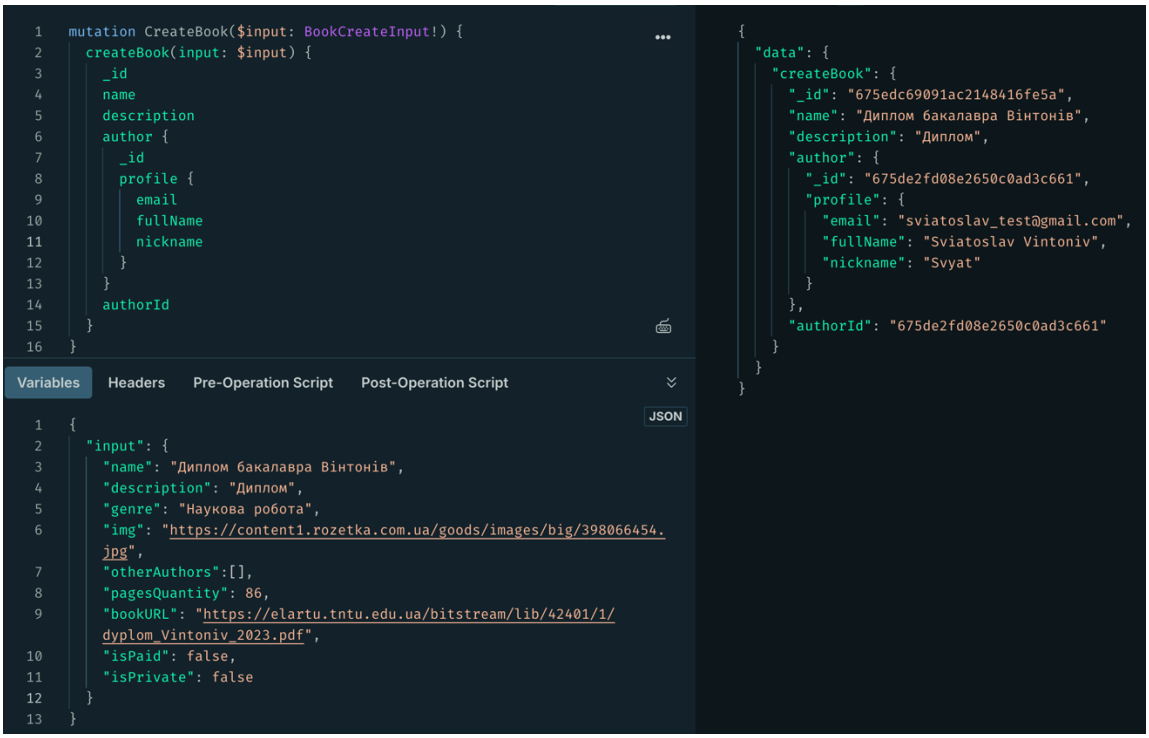


Рис. 3.18. Запит для створення книги

Як можна спостерігати на рисунку, запит виконався коректно і повернув дані створеної книги.

Одним з основних є запит для повернення всіх користувачі, оскільки ця сутність є базовою і на ній зав’язані майже всі зв’язки в системі. Виконання запиті для отримання користувачів з системи можна спостерігати на рисунку 3.19.

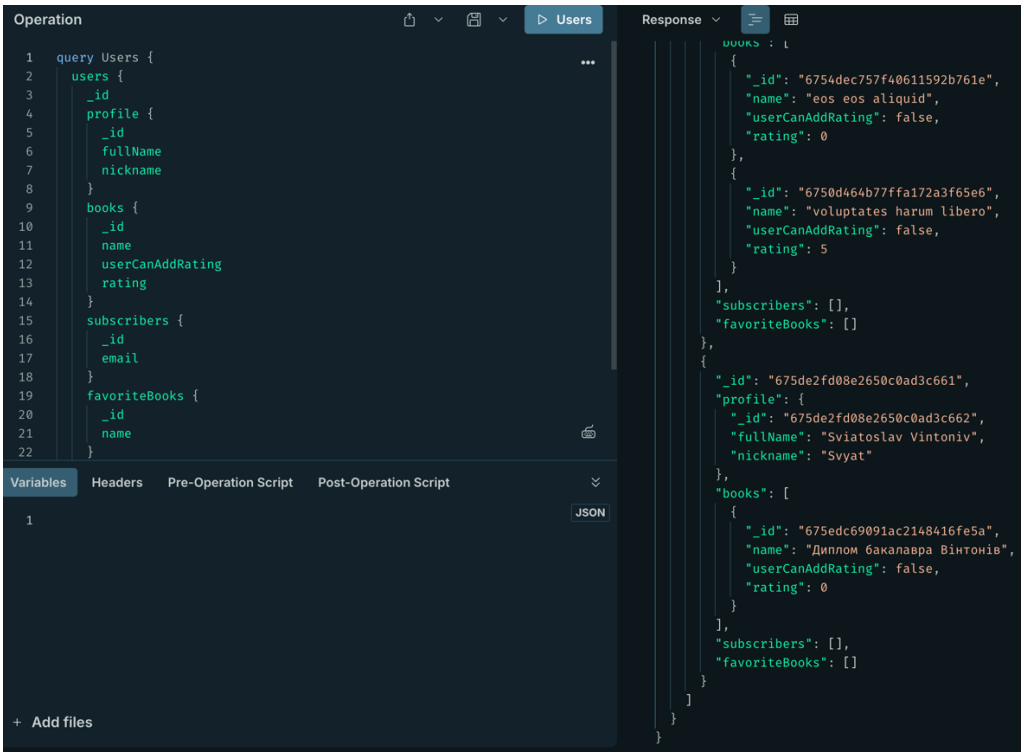


Рис. 3.19. Запит для отримання користувачів

Як можна спостерігати на рисунку вище, запит для отримання користувачів виконався успішно і повернув дані користувачів та дані всіх пов’язаних і з ними сутностей.

Інші запити та мутації також було протестовано, в ході тестування були виявлені деякі проблеми, які одразу було виправлено.

## 3.4. Використання DataLoader для оптимізації запитів та створення обгортки над запитами до БД

Як показали тести сервера, він не погано справляється з навантаженнями, але по мірі росту користувацької бази слід переконати, що сервер працюватиме без перебоїв. Оскільки велика кількість запитів до бази даних виконується в

револьверах, то по мірі росту даних БД може не справитись з кількістю запитів, тому слід продумати про оптимізацію цього місця, виконати це можна за допомогою DataLoadres, які призначенні для того, щоб мінімізувати кількості запитів до БД.

За допомогою скрипта було створено тисячу користувачів та заповнено їх випадковими даними, також для кожного користувача було створено по 10 книг, це необхідно для проведення стрес тестування API. Звісно, в таких випадках слід використовувати пагінацію для отримання користувачів порціями, але для того, щоб максимально навантажити сервер та базу даних в запитах було проігноровано розбиття на сторінки.

Спочатку слід виконати запит без використання завантажувача даних, щоб перевірити час його виконання (рис. 3.20).


Рис. 3.20. Виконання запиту без використання завантажувача даних

Як можна побачити на рисунку, час виконання запиту для отримання даних всіх користувачів склав 11.22 секунди, і це враховуючи той факт, що база даних і сервер запущено на одному девайсі, якщо б вони були на різних серверах і розміщені далеко один від одного, то час виконання був би набагато більшим.

Для оптимізації запитів було з використанням завантажувачів даних використано бібліотеку «dataloader». Завдяки цій бібліотеці реалізовано завантажувач даних для профілю по ідентифікаторі користувача (рис. 3.21).

```
const ProfileLoader = new DataLoader(
  async (userIds) => {
    const profiles = await ProfileCollection.find({ userId: { $in: makeObjectIds(userIds) } }).toArray();
    const profilesByUserId = keyBy(profiles, 'userId');
    return userIds.map((userId) => profilesByUserId[userId]);
  },
  { cacheKeyFn: (key) => key.toString() }
);

const server = new ApolloServer({
  typeDefs,
  resolvers,
  context: ({ req }) => {
    return { ...getUserContext(req), profileLoader: ProfileLoader };
  },
});
```

Рис. 3.21. Завантажувач даних для сутності профілю по ідентифікаторі користувача

На рисунку можна спостерігати створення завантажувача даних для сутності профілю він по ідентифікаторах користувача отримає їх профілі, що повинно зменшити кількість запитів до БД з тисячі до одного. Після того, як завантажувач даних створено він додається в контекст для подальшого його використання.

Використання завантажувача даних для сутності користувача наведено на рисунку 3.22.

```
profile: async ({ _id }, _, {profileLoader}) => {
  return profileLoader.load(_id);
},
```

Рис. 3.22. Використання завантажувача даних для сутності профілю

Після впровадження завантажувача даних для профілю можна спробувати знову зробити запит для отримання користувачі (рис. 3.23).

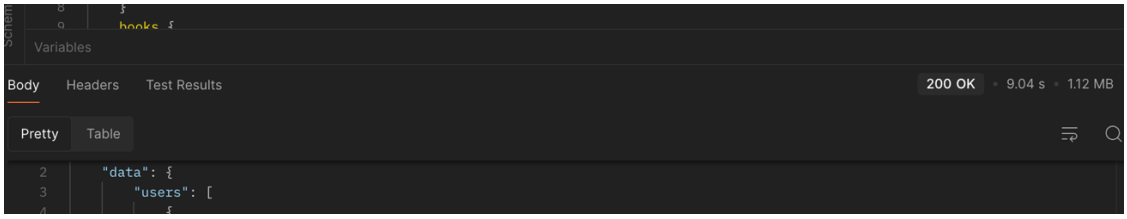


Рис. 3.23. Результат виконання запиту для отримання користувачів

Завдяки впровадженню завантажувача даних для профілю вдалося зменшити час завантаження до 9.04 секунди, для того, що ще покращити результат слід реалізувати dataLoader і для інших колекцій, проте це зробити не дуже просто, оскільки для того, щоб отримати рейтинг книги, в завантажував даних слід передати агрегацію а не ідентифікатор, тому слід використати інший підхід до реалізації завантажувачів даних.

Мною було прийнято рішення не зберігати завантажувачів в контекст, а зробити обгортку над запитами які виконуються до бази даних та застосувати dataLoader там. Також слід реалізувати завантажувач даних, який буде приймати не ідентифікатор а Mongo-query і використовувати його в агрегації, щоб зробити один запит замість багатьох. Обгортка над запитами до БД також полегшить використання завантажувачів даних оскільки не потрібно буде постійно діставати їх з контексту.

Реалізація функції обгортки над колекцією наведено на рисунку 3.24.

```
export const loaderByQuery = (Collection) => {
  const loader = new DataLoader(
    async (args) => {
      loader.clearAll();
      const initialMatch = { $match: { $or: [] } };
      const facetQuery = args.reduce((acc, { query, sort }, i) => {
        initialMatch.$match.$or.push(query);
        const pipeline = [{ $match: query || { _id: null } }];
        const sortToPipeline = sort && Object.keys(sort).length ? sort : {};
        if (!sortToPipeline._id) sortToPipeline._id = 1;
        pipeline.push({ $sort: sortToPipeline });
        acc[i] = pipeline;
        return acc;
      }, {});
      const arr = [initialMatch, { $facet: facetQuery }];
      const [resultObject] = await Collection.aggregate(arr).toArray();
      return args.map((_q, i) => resultObject[i] || []);
    },
    { cacheKeyFn: (key) => JSON.stringify(key) }
  );
  return async (data) => {
    return loader.load({ ...data });
  };
};
```

Рис 3.24. Функція для використання завантажувача даних на рівні запитів до бази даних

На рисунку зображено функцію, яка приймає колекцію та реалізує завантажувач даних на основі Mongo-запитів, а саме перетворює запити в одну агрегацію і повертає функцію зворотного виклику. Таким чином замість декількох запитів до бази даних буде виконано одну агрегацію, що має прискорити процес отримання даних. Для прикладу для кожної книги буде виконуватись запит для отримання її рейтингу, у випадку з запитом який повертає 1000 користувачів у кожного з яких є 10 книг, це буде 10000 запитів для отримання рейтингу книг, цей завантажувач даних перетворить їх на одну агрегацію, що має зменшити час виконання.

Було також реалізовано обгортку над колекціями яка використовує завантажувач даних для сутності по її ідентифікатору. Ця функція працює схоже до представленого вище завантажувача сутності профілю, її реалізацію наведено в додатках до роботи.

У випадку якщо є можливість використати завантажувач по ідентифікатору, то краще надавати перевагу йому перед завантажувачем по Mongo-запиту, оскільки перший використовує простий запит, а другий агрегацію яка працює набагато повільніше.

Після реалізації обгортків над колекціями які надають функції завантажувачів даних їй було використано у всіх місцях де це можливо для оптимізації кількості запитів до БД. Та для тестування знову використано запит для отримання всіх користувачів (рис. 3.25).

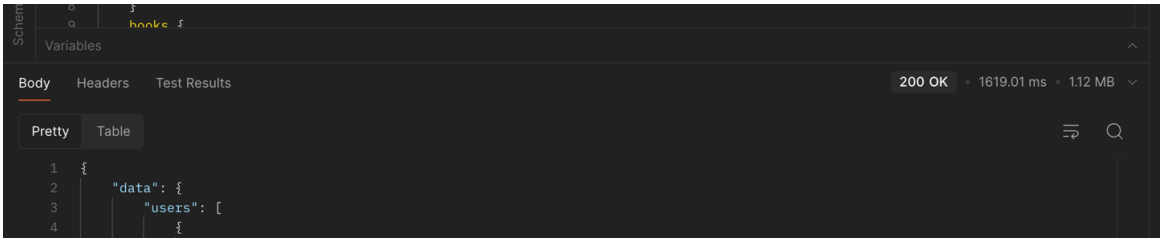


Рис. 3.25. Оптимізований запит для отримання всіх користувачів

Як можна побачити на рисунку завдяки використанню завантажувачів даних на рівні запитів до БД час виконання запиту вдалося скоротити з 11.22 секунд до 1.619 секунди, що є доволі хорошим результатом.

## 3.5. Реалізація графічного інтерфейсу програмної системи

Графічний інтерфейс буде представлено веб-застосунком, з використання бібліотеки React, для стилізації буде використано Bootstrap, а для виконання запитів на сервер використовуватиметься Apollo Client.

Одним з основних є компонент для відображення списку книг (рис. 3.26).

```
<>
  <Container className="mt-3">
    <Search
      initialValue={lineForSearch}
      isSearching={isSearchingBooks}
      search={searchBooks}
      deleteSearch={deleteSearch}
      isFound={isBooksFound}
      isResettingSearch={isResettingSearch}
    />
    <Link style={{ float: 'right', marginRight: '20px' }} to={paths.addBook}>
      <Button>Add book</Button>
    </Link>
    {error && !isBooksFound ? <Alert variant="danger">{error.message}</Alert> : null}
  </Container>
  {isLoading && !books ? <>Loading...</> : null}
  {isBooksFound ? (
    <ItemsFound
      changeRating={onAddRating}
      isUpdateRating={isUpdateRating}
      lineForSearch={lineForSearch}
      setIsSearching={setIsSearchingBooks}
      nameItems={BOOKS}
    />
  ) : (
    books?.map((book) => {
      return (
        <Col key={book._id} lg="3" md="4" sm="6" xs="6" className="mt-4">
          {idOfSelectedBook === book._id ? (
            <BookCard addRating={onAddRating} book={book} isUpdate={isUpdateRating} />
          ) : (
            <BookCard addRating={onAddRating} book={book} />
          )}
        </Col>
      );
    })
  )}
</>
```

Рис 3.26. Компонент для відображення списку книг

На рисунку представлено компонент для відображення списку книг, а також рядка пошуку і кнопки додавання нової книги.

Слід також продемонструвати хук за допомогою якого виконується запит на сервер для отримання книг (рис. 3.27).

```
const booksQuery = gql`
  query Books($searchString: String) {
    books(searchString: $searchString) {
      _id
      name
      description
      rating
      img
      userCanAddRating
    }
  }
`;

const useBooks = (searchString = '') => {
  const { loading, data, error, ...rest } = useQuery(booksQuery, {
    variables: { searchString },
  });
  return [getFirstResult(data) || [], { loading, error, ...rest }];
};
```

Рис. 3.27. Запит для отримання книг

Даний запит приймає стрічку для пошуку книг, якщо вона порожня то запит повертає всі книги.

Веб-застосунок складається з великої кількості компонентів та запитів, також стилів для цих компонентів, більш детально ознайомитися з кодом клієнтської частини платформи можна в додатках до роботи.

## 3.6. Тестування графічного інтерфейсу розробленої системи

Після завершення розробки платформи інформаційного забезпечення процесу обміну книгами сліт провести тестування готової системи та її інтерфейсу.

Підчас входу в систему користувач бачить вікно авторизації, якщо в нього ще немає акаунта то йому слід провести реєстрацію. Вікно авторизації зображено на рисунку 3.28.


Рис. 3.28. Вікно авторизації в системі

Після авторизації в системі користувач отримує доступ до сторінки зі списком всіх книг (рис.3.29).

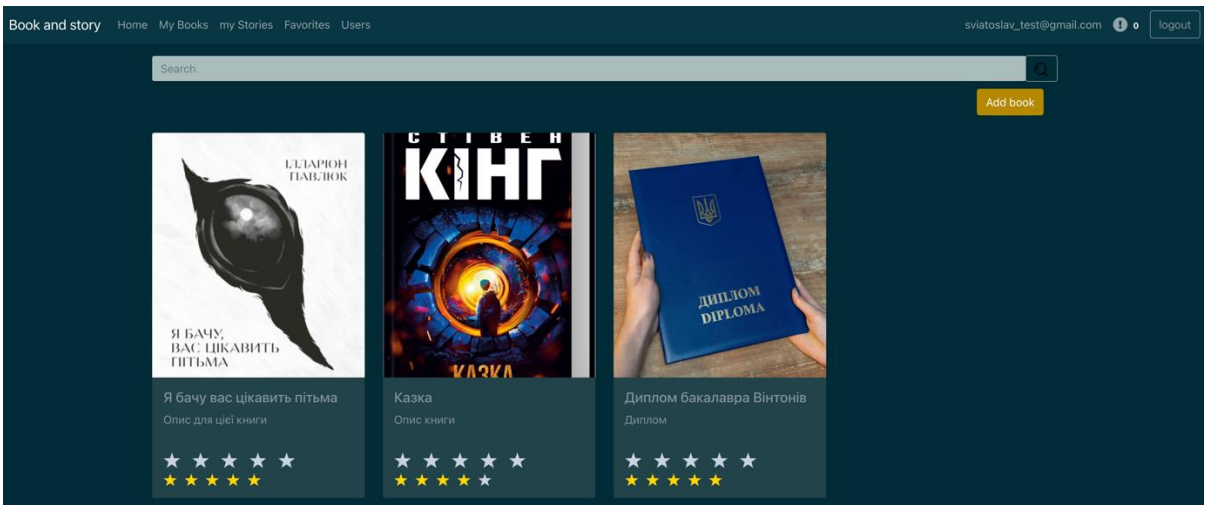


Рис. 3.29. Сторінка зі список всіх книг

Користувач має можливість додати власну книгу натиснувши на кнопку «Add book» (рис. 3. 30).

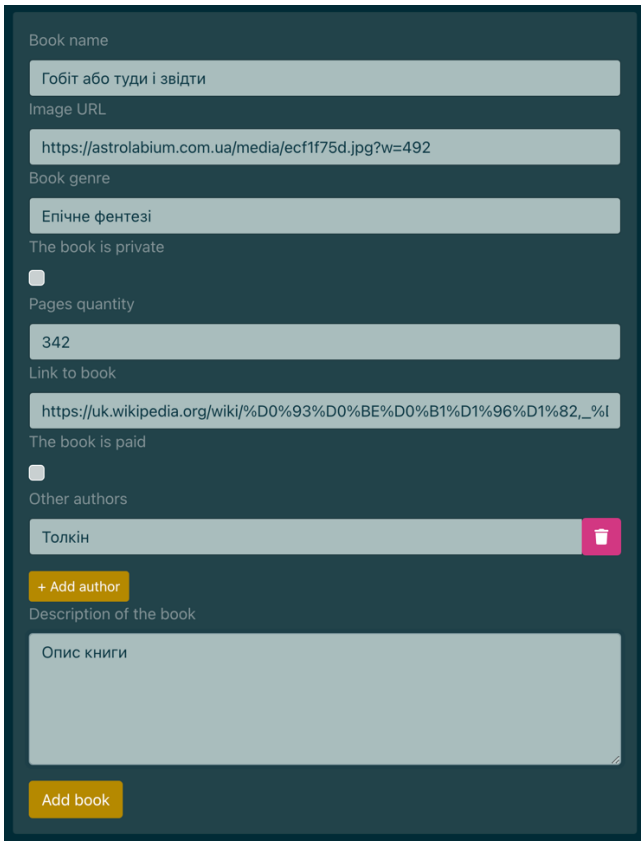


Рис. 3.30. Форма додавання книги

Після того, як форму заповнено слід натиснути кнопку для додавання книги і її буде додано до списку (рис. 3.31).

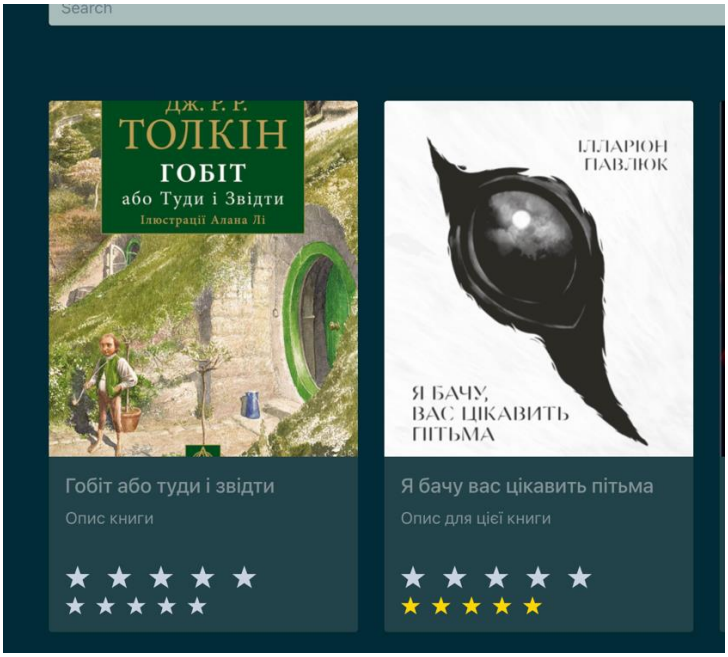


Рис. 3.31. Книга додана до списку

Кожен користувач може один раз оцінити кожну книгу, що впливатиме на її рейтинг (рис. 3.32).

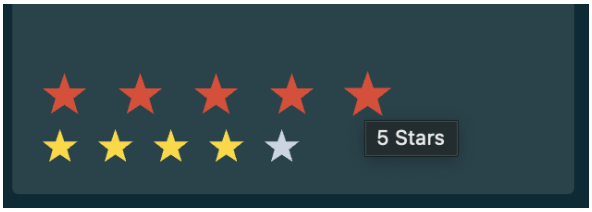


Рис. 3.32. Процес оцінки книги

Верхній рядок використовується для оцінки книги, нижній є середнім рейтингом книги. Після оцінки користувач більше не бачитиме верхній рядок оцінок (рис.3.33).


Рис. 3.33. Середній рейтинг книги

Також в системі є функції які включають перегляд своїх книг, редагування та можливість позначити їх як приватні, щоб приховати від інших користувачів. Серед можливостей є додавання книг в список обраних. В системі реалізовано функцію підписок яка дозволяє отримувати оповіщення про випуск книги авторів на яких підписався користувач. Всі користувачі можуть редагувати інформацію про себе. Для всіх вище перерахованих функцій було проведено тестування, щоб впевнитись в правильності їх роботи.

## 3.7. Висновки до розділу

У даному розділі описано реалізацію ключових компонентів, необхідних для правильної роботи платформи. Внутрішня система була побудована з

використанням GraphQL, з сильним акцентом на проєктуванні схем та ефективності запитів. Передові методи оптимізації запитів, такі як інтеграція DataLoader та спеціальної обгортки запитів, забезпечили зменшення навантаження на базу даних та покращили продуктивність системи. Реалізований сервер пройшов ретельне тестування та налагодження, щоб гарантувати стабільність та надійність.

Графічний інтерфейс системи розроблений з використанням сучасних інструментів, таких як React та Bootstrap, надає користувачам інтуїтивно зрозумілий та адаптивний інтерфейс для взаємодії з книгами, авторами та сповіщеннями. Всебічне тестування графічного інтерфейсу підтвердило його зручність та функціональність. Поєднавши ефективну оптимізацію сервера зі зручним графічним інтерфейсом, дозволило платформі успішно досягати своїх цілей щодо масштабованості, продуктивності та простоти використання, забезпечуючи надійне рішення для процесу книгообміну.

# РОЗДІЛ 4
# ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

## 4.1. Охорона праці

У кваліфікаційній роботі проводилось дослідження методів та засобів побудови програмно-апаратної платформи інформаційного забезпечення процесу обміну книгами, також було спроєктовано та реалізовано серверне API. Виконання даної роботи проводилося з використанням ЕОМ тому при роботі з даною технікою є необхідним дотримання правил охорони праці та техніки безпеки.

При роботі з ЕОМ потрібно дотримуватись державних стандартів України. Робочі місця працівників, які обладнані комп'ютерами пристроями, повинні відповідати вимогам ДСТУ 8604:2015 «Дизайн і ергономіка. Робоче місце для виконання робіт у положенні сидячи. Загальні ергономічні вимоги» [22].

Основні положення ДСТУ 8604:2015:

- Конструкція робочого місця та взаємне розташування всіх його елементів (сидіння, органів керування, засобів відображення інформації тощо) повинні відповідати антропометричним, фізіологічним і психологічним вимогам, а також характеру виконуваної роботи.
- Конструкцією робочого місця повинно бути забезпечено виконання трудових операцій у межах зони досяжності моторного поля.
- Висоти сидіння та підставки для ніг (у разі нерегульованої висоти робочої поверхні). У цьому разі висоту робочої поверхні встановлюють за номограмою для працюючих зростом 1800 мм. Оптимальна робоча поза для менших за зростом працюючих досягається збільшенням висоти робочого сидіння й підставки для ніг на величину, що дорівнює різниці між висотою робочої поверхні для працюючого зростом 1800 мм і висотою робочої поверхні, оптимальної для зросту даного працюючого.
- Підставка для ніг повинна бути регульованою по висоті. Її ширина повинна бути не менше ніж 300 мм, довжина - не менше ніж 400 мм. Поверхня

підставки повинна бути рифленою. По передньому краю доцільно передбачати бортик висотою 10 мм.

Також на робочих місцях працівників які взаємодіють з екранними пристроями слід забезпечити дотримання вимог, які визначені в НПАОП 0.00-7.15-18 «Вимог щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями» [23]. Згідно з цим наказом освітлення робочого місця працівника з екранними пристроями має створювати відповідний контраст між екраном і навколишнім середовищем (з урахуванням виду роботи) та відповідати вимогам ДСанПІН 3.3.2.007-98 [24].

Основні положення ДСанПІН 3.3.2.007-98:

- Площа на одне робоче місце має становити не менше ніж 6,0 м$^2$, а об'єм не менше ніж 20,0 м$^3$.
- Приміщення для роботи повинні мати природне та штучне освітлення.
- Природне освітлення має здійснюватись через світлові прорізи, орієнтовані переважно на північ чи північний схід і зебезпечувати коефіцієнт природною освітленості (КПО) не нижче ніж 1,5%.
- Для внутрішнього оздоблення приміщень слід використовувати дифузно-відбивні матеріали з коефіцієнтами відбиття для стелі 0,7-0,8, для стін 0,5-0,6.
- Яскравість світильників загального освітлення в зоні кутів випромінювання від 50 до 90 градусів з вертикаллю в повздовжній та поперечній площинах має становити не більше ніж 200 кд/м$^2$, захисний кут світильників - не менше ніж 40 градусів
- Показник засліплення у разі використання джерел загального штучного освітлення у виробничих приміщеннях має не перевищувати 20.
- Необхідно обмежувати нерівномірність розподілу яскравості в полі зору працюючих з екранами. При цьому співвідношення яскравості робочих поверхонь має бути не більшим ніж 3:1, а співвідношення яскравості робочих поверхонь та поверхонь стін, обладнання тощо - 5:1.

Таким чином згідно з вимогами щодо охорони праці для працівників які працюють з ЕОМ приміщення, яке використовувалося для виконання кваліфікаційної роботи магістра було забезпечене природнім і штучним освітленням. Також для роботи з ЕОМ було обрано місце, щоб в поле зору не потрапляли вікна або освітлювальні прилади. Шкідливим є також розміщення вікон за спинами працівників, оскільки на моніторі з'являється відбиття світла. Для уникнення проблем зі засліпленням природнім світлом приміщення було обладнано з використанням жалюзі з можливістю регулювання світлових потоків, щоб захистити робоче місце від попадання прямих сонячних променів. Адже вікна приміщення орієнтовані на південний схід.

Приміщення також було забезпечене штучним освітленням у вигляді комбінованих систем освітлення з використанням люмінесцентних ламп які розташовуються над робочими поверхнями у рівномірному порядку, щоб запобігти освітленню екрану ЕОМ прямими світловими потоками.

На всі робочих місцях для працівників які взаємодіють з ЕОМ забезпечена рівномірна освітленість за допомогою відбитого або розсіяного світла. Світлових відблисків від будь-яких частин робочої поверхні або ЕОМ у напрямку очей працівника немає.

Таким чином кваліфікаційна робота магістра виконувалась з урахуванням ДСТУ 8604:2015 та відповідала вимогам ДСанПІН 3.3.2.007-98, що дозволило працювати в умовах які відповідають правилам охорони праці та техніки безпеки. Робоче місце було обладнано всіма необхідними для комфортної роботи з ЕОМ засобами. Приміщення було забезпечене рівномірним та правильним освітленням, робочі місця розміщені таким чином, щоб природнє освітлення додавало природності робочому середовищу та зменшувало втомленість очей. Штучне освітлення було розподілене рівномірно і мало достатній рівень яскравості для зручного використання комп'ютера та читання документів. Такий підхід сприяє не лише ефективній роботі, але й збереженню здоров'я та підвищенню загального рівня задоволення від роботи з ЕОМ.

## 4.2. Проведення аварійно-відновлювальних робіт на комп’ютерних та електричних мережах

У випадку якщо виникає надзвичайної ситуації, незалежно від її природи для оперативної ліквідації наслідків та захисту населення діє єдина загальнодержавна система цивільного захисту. Ця система охоплює комплекс управлінських органів, сил та засобів, що належать центральним і місцевим органам виконавчої влади, органам місцевого самоврядування, а також підприємствам, установам та організаціям. Завдяки їх скоординованій діяльності реалізується державна політика у сфері цивільного захисту, що включає попередження надзвичайних ситуацій, реагування на них, ліквідацію їхніх наслідків та надання допомоги постраждалим, у мирний час та в особливий період [25].

Для ефективного надання допомоги та швидкого подолання наслідків катастроф надзвичайно важливо налагодити безперебійну комунікацію між усіма відповідальними ланками. Зв’язок виступає одним з основних чинників у координації дій управлінських структур під час надзвичайних ситуацій. На кожному рівні єдиної державної системи діють як координаційні, так і постійно діючі органи управління, які займаються профілактикою надзвичайних ситуацій, захистом населення та територій а також мають у своєму розпорядженні резерви матеріальних і фінансових ресурсів, відповідне обладнання, системи зв’язку та інформаційного забезпечення.

З метою вчасного реагування та інформування у міських та позаміських пунктах на базі автоматизованих систем централізованого оповіщення, загальнодержавних комунікаційних мереж, радіомовлення й спеціальних приладів формується система оповіщення та інформаційного забезпечення. Вона являє собою складний організаційно-технічний комплекс, призначений для передачі сигналів та розпоряджень органам державної влади, керівництву підприємств, установ і організацій, силам цивільного захисту а також населенню.

Автоматизована система оповіщення – сукупність організаційно і технічно поєднаних програмних і технічних засобів, телекомунікаційних мереж, телемереж

та інших засобів оброблення та передачі (відображення) інформації, що призначені для своєчасного доведення сигналів та інформації з питань ЦЗ до органів виконавчої влади, органів місцевого самоврядування, органів управління і сил ЦЗ, підприємств, установ, організацій та населення [26]. Система оповіщення та інформаційного забезпечення використовується централізовано. Архітектура технології провідного доступу зображена на 4.1.

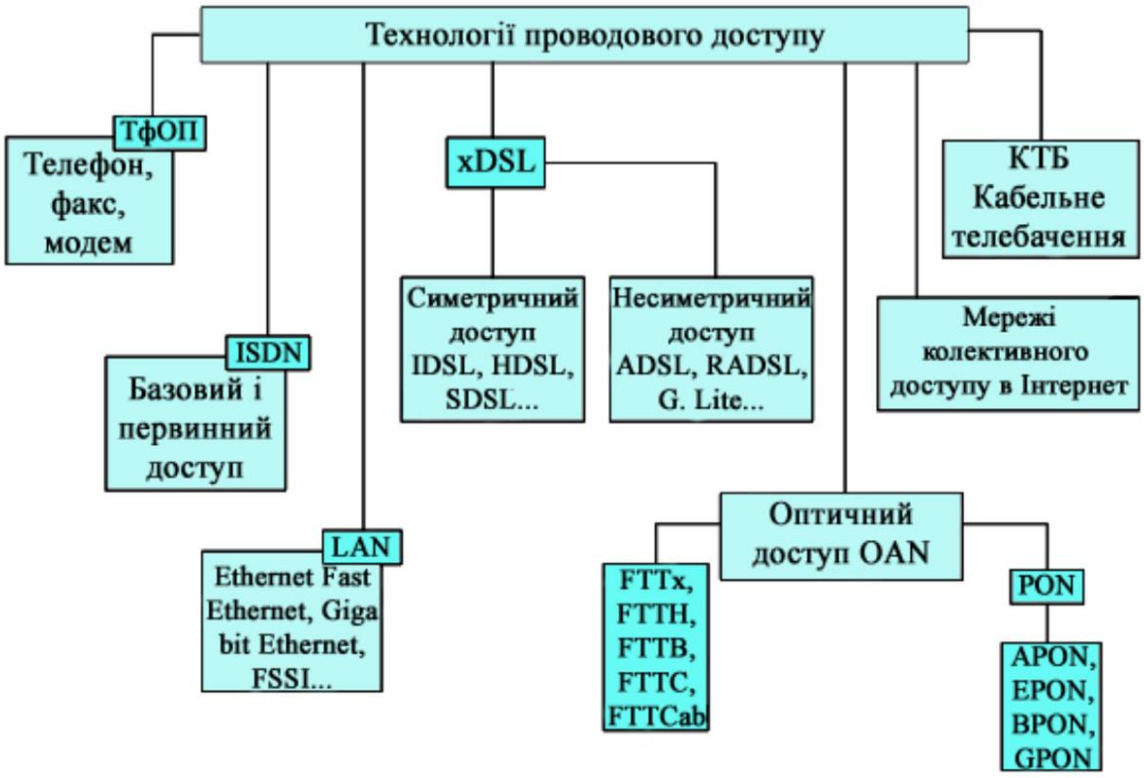


Рис. 4.1. Передача інформації з використанням провідних технологій

Оскільки комп’ютерні мережі застосовуються для обміну інформацією, забезпечення їх безперебійної роботи та швидке усунення можливих несправностей є одним із головних пріоритетів. Нормативну базу, яка регулює термінове виконання рятувальних та інших невідкладних заходів у зоні надзвичайних ситуацій, складають такі документи: Кодекс цивільного захисту України; постанова Кабінету Міністрів України від 14 червня 2002 р. №843, якою затверджено Загальне положення про спеціальну Урядову комісію з ліквідації надзвичайних ситуацій техногенного та природного характеру, а також загальне

положення про аналогічні комісії для регіонального, місцевого та об'єктового рівнів; постанова Кабінету Міністрів України від 19 серпня 2002 р. №1201, якою затверджено Положення про штаб з ліквідації надзвичайних ситуацій техногенного та природного характеру.

Термінові відновлювальні роботи з ремонту пошкоджених ліній зв'язку, комп'ютерних та телефонних, проводяться разом із заходами щодо відновлення електропостачання, підсилення конструкцій та розблокування шляхів для створення можливості пересування, якщо вони перекриті. Лінії телефонного зв'язку можуть використовуватися як передавальне середовище для мережевих сигналів завдяки технології ADSL

Для виконання аварійно-відновлювальних робіт залучаються сили цивільного захисту, зокрема підрозділи Оперативно-рятувальної служби, воєнізовані аварійно-рятувальні загони, спеціалізовані та невоєнізовані формування. Безпосередньо на місці подій організовуються групи працівників та службовців, які розподіляються за своїми завданнями та напрямками діяльності. Для відновлення інформаційних мереж залучаються переважно загальнопрофільні групи цивільного захисту, до складу яких входять:

- зведені рятувальні загони (команди, групи);
- зведені загони (команди) з механізації робіт;
- рятувальні загони (команди, групи);
- формування загальної розвідки (розвідувальні команди, групи, ланки).

Роботи, пов'язані з відновленням комп'ютерних та електричних мереж, належать до невідкладних завдань, адже саме завдяки їм забезпечується можливість оперативного реагування на надзвичайну ситуацію, доступ до критично важливої інформації та задоволення нагальних потреб населення.

# ВИСНОВКИ

Основні наукові та практичні результати полягають в наступному.

1. Проаналізовано літературу, яка пов’язана з розробкою програмно-апаратних платформ, цей аналіз показав, що важливою є масштабованість та оптимізація систем, щоб забезпечити її відповідність сучасним вимогам.

2. Досліджено предметну область та сучасні методи розробки API, в результаті вибрано сучасні рішення та технології такі, як GraphQL для побудови ефективного API, Node.js для асинхронної обробки запитів. Крім того, були описані методи та інструменти такі, як DataLoader для пакетної обробки та кешування, щоб оптимізувати запити до БД.

3. Проведено аналіз існуючих рішень, та їхніх сильних і слабких сторін, що дозволило сформулювати функціональні вимоги до системи, та в подальшому реалізувати їх.

4. Спроєктовано архітектору платформи та бази даних, що забезпечило гнучкість та ефективну взаємодію між компонентами системи.

5. Було реалізовано серверну частину з використанням Node.js та GraphQL, також зроблено оцінку продуктивності системи під високими навантаженнями. Для оптимізації системи було використано DataLoader на рівні запитів до бази даних, що дозволило значно покращити продуктивність системи.

6. Було розроблено графічний інтерфейс з використанням React, це дозволило зробити його інтуїтивно зрозуміли та забезпечити безперебійний зв'язок з API, що надає користувачам можливість зручної взаємодії з платформою інформаційного забезпечення процесу обміну книгами.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Стаття про масштабованість в хмарних обчисленнях. URL: https://www.nops.io/blog/cloud-scalability/ (дата звернення: 30.11.2024 р.)

2. Стаття про архітектурні підходи до розробки масштабованих веб-застосунків. URL: https://csecurity.kubg.edu.ua/index.php/journal/article/view/613 (дата звернення: 30.11.2024 р.)

3. Вінтонів С. О., Тиш Є. В. Проєктування та розробка API для платформи обміну книгами. Матеріали XI науково-технічній конференції Тернопільського національного технічного університету імені Івана Пулюя «Інформаційні моделі, системи та технології». Тернопіль: 2024. 543с.

4. Вінтонів С. О., Тиш Є. В. Оптимізація кількості запитів до бази даних для GraphQL-API з використанням DataLoader. Матеріали XII міжнародній науково-технічній конференція молодих учених та студентів «Актуальні задачі сучасних технологій». Тернопіль: 2024.

5. Статистика цифрових продуктів за 2024 рік. URL: https://whop.com/blog/digital-product-statistics/ (дата звернення: 01.12.2024 р.)

6. Дохід від продажів електронних книг Trade у США з 2017 по 2023 рік. URL: https://www.statista.com/statistics/278235/e-book-sales-revenue-in-the-us/ (дата звернення: 01.12.2024 р.)

7. Прогноз росту ринку електронних книг. URL: https://www.futuremarketinsights.com/reports/global-ebook-market\ (дата звернення: 01.12.2024 р.)

8. API Design: Principles for Building Effective APIs. URL: https://medium.com/@teja.ravi474/api-design-principles-for-building-effective-apis-5d391b6844a3 (дата звернення: 03.12.2024 р.)

9. Kharchenko A., Bodnarchuk I., Yatcysyn V. The Method for Comparative Evaluation of Software Architecture with Accounting of Trade-offs. American Journal of Information Systems. 2014. Vol. 2, No. 1. P. 20-25.

10. API and Database Performance Optimization Strategies. URL: https://dzone.com/articles/api-and-database-performance-optimization-strategi (дата звернення: 04.12.2024 р.)

11. Безпека API і як її реалізувати. URL: https://medium.com/@siddiquimohammad0807/api-security-introduction-216d46968c8b (дата звернення: 04.12.2024 р.)

12. Важливість застосування AWS Elastic Load Balancing. URL: https://blurify.com/blog/aws-elastic-load-balancing-why-is-it-important/ (дата звернення: 04.12.2024 р.)

13. Wattpad. URL: https://www.wattpad.com/home (дата звернення: 04.12.2024 р.)

14. Goodreads. URL: https://www.goodreads.com/ (дата звернення: 04.12.2024 р.)

15. Inkitt. URL: https://www.inkitt.com/ (дата звернення: 04.12.2024 р.)

16. Samer Buna. GraphQL in Action. Manning, 2021. P. 384.

17. Anthony Nandaa. Beginning API Development with Node.js. Packt Publishing, 2018. P. 100.

18. Corey J. Ball. Hacking Apis. Breaking Web Application Programming Interfaces. 2022. P. 368.

19. Prabath Siriwardena. Advanced API Security: Securing APIs with OAuth 2.0, OpenID Connect, JWS, and JWE. 2014. P. 260.

20. Харченко О., Яцишин В. Розробка та керування вимогами до програмного забезпечення на основі моделі якості. Вісник ТДТУ. Тернопіль, 2009. Т. 14. №1. С. 201-207.

21. Yatsyshyn V., Pastukh O., Palamar A., Zharovskyy R. Technology of relational database management systems performance evaluation during computer systems design. Scientific Journal of TNTU, Ternopil, Ukraine, 2023. Vol. 109, No 1. P. 54–65.

22. ДСТУ 8604:2015. URL: https://online.budstandart.com/ua/catalog/doc-page.html?id_doc=71028 (дата звернення: 11.12.2024 р.)

23. Наказ Мінсоцполітики від 14.02.2018, № 207. URL: https://zakon.rada.gov.ua/laws/show/z0508-18#Text (дата звернення: 11.12.2024 р.)

24. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин. ДСанПІН 3.3.2.007-98. URL: https://zakon.rada.gov.ua/rada/show/v0007282-98#Text (дата звернення: 11.12.2024 р.)

25. Кодекс цивільного захисту України. URL: https://zakon.rada.gov.ua/laws/main/5403-17 (дата звернення: 11.12.2024 р.)

26. Стручок В.С. Техноекологія та цивільна безпека. Частина «Цивільна безпека». Навчальний посібник. Тернопіль: ТНТУ. 2022. 150 с.

27. Стручок В.С. Безпека в надзвичайних ситуаціях. Методичний посібник для здобувачів освітнього ступеня «магістр» всіх спеціальностей денної та заочної (дистанційної) форм навчання. Тернопіль: ТНТУ. 2022. 155 с.

28. Луцик Н.С., Луцків А.М., Осухівська Г.М., Тиш Є.В. Програма та методичні рекомендації з проходження практики за тематикою кваліфікаційної роботи для студентів спеціальності 123 «Комп'ютерна інженерія» другого (магістерського) рівня вищої освіти усіх форм навчання. Тернопіль: ТНТУ. 2024. 45 с.

29. Луцик Н.С., Луцків А.М., Осухівська Г.М., Тиш Є.В. Методичні рекомендації до виконання кваліфікаційної роботи магістра для студентів спеціальності 123 «Комп'ютерна інженерія» другого (магістерського) рівня вищої освіти усіх форм навчання. Тернопіль. 2024. 44 с.

30. Варавін А.В., Лещишин Ю.З., Чайковський А.В. Методичні вказівки до виконання курсового проєкту з дисципліни «Дослідження і проєктування комп'ютерних систем та мереж» для здобувачів другого (магістерського) рівня вищої освіти спеціальності 123 «Комп'ютерна інженерія» усіх форм навчання. Тернопіль: ТНТУ, 2024. 32 с.

Додаток А

Тези конференцій

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**
**Тернопільський національний технічний університет імені Івана Пулюя (Україна)**
**Університет імені П'єра і Марії Кюрі (Франція)**
**Маріборський університет (Словенія)**
**Технічний університет у Кошице (Словаччина )**
**Вільнюський технічний університет ім. Гедимінаса (Литва)**
**Наукове товариство ім. Т.Шевченка**

# АКТУАЛЬНІ ЗАДАЧІ СУЧАСНИХ ТЕХНОЛОГІЙ

## Збірник
тез доповідей

**XIII Міжнародної науково-практичної конференції молодих учених та студентів**
11-12 грудня 2024 року

**УКРАЇНА**
**ТЕРНОПІЛЬ – 2024**

*Матеріали XIII Міжнародної науково-практичної конференції молодих учених та студентів*
*«АКТУАЛЬНІ ЗАДАЧІ СУЧАСНИХ ТЕХНОЛОГІЙ» – Тернопіль, 11-12 грудня 2024 року*

**УДК 004.738.5**
**С. Вінтонів; Є. Тиш, канд. техн. наук**
(Тернопільський національний технічний університет імені І. Пулюя, Україна)

## ПРОЄКТУВАННЯ ТА РОЗРОБКА API ДЛЯ ПЛАТФОРМИ ОБМІНУ КНИГАМИ

**S. Vintoniv; Ie. Tysh, Ph.D.**
Ternopil Ivan Puluj National Technical University, Ukraine

## DESIGNING AND DEVELOPING AN API FOR A BOOK EXCHANGE PLATFORM

Сучасні цифрові платформи для обміну книжками значною мірою покладаються на надійні API для забезпечення безперешкодної взаємодії між користувачами та сервісами платформи. При розробці API пріоритетами мають бути масштабованість, продуктивність і безпека, особливо для платформ, які обробляють динамічну і складну взаємодію з користувачами. У цій роботі досліджується розробка API для платформи обміну книжками з акцентом на інноваційні методи і технології, що забезпечують ефективність, надійність і безпеку платформи.

Однією з головних проблем розробки API є ефективна обробка великих обсягів даних. Сучасні рішення цих проблем включають асинхронну обробку запитів, кешування даних та оптимізовані запити до бази даних. Асинхронна природа Node.js у поєднанні з такими фреймворками, як Express та Apollo Server, забезпечує швидку та одночасну обробку декількох запитів, що є критично важливим для забезпечення адаптивного користувацького досвіду.

Важливим підходом до підвищення продуктивності API є кешування. Використовуючи такі інструменти, як Redis або сховища даних в пам'яті, до яких часто звертаються, можуть бути швидко отримані без навантаження БД. GraphQL також підтримує механізми кешування на рівні запитів, які зменшують навантаження.

Безпека залишається критично важливим питанням при розробці API. Впровадження найкращих практик, таких як автентифікація на основі токенів, обмеження швидкості та валідація вхідних даних, захищає систему від таких загроз, як DDoS-атаки або несанкціонований доступ до даних.

Результати цього дослідження демонструють, що поєднання асинхронних технологій, ефективних механізмів кешування, протоколів безпеки та масштабованих архітектур забезпечує надійну основу для створення сучасного API.

**Література**

1. API Design: Principles for Building Effective APIs. URL: https://medium.com/@teja.ravi474/api-design-principles-for-building-effective-apis-5d391b6844a3

2. API and Database Performance Optimization Strategies. URL: https://dzone.com/articles/api-and-database-performance-optimization-strategi

3. API Security — Introduction. URL: https://medium.com/@siddiquimohammad0807/api-security-introduction-216d46968c8b

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**
**ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ**
**імені ІВАНА ПУЛЮЯ**
**НАУКОВЕ ТОВАРИСТВО ім. ШЕВЧЕНКА**

**М А Т Е Р І А Л И**

**XII НАУКОВО-ТЕХНІЧНОЇ КОНФЕРЕНЦІЇ**

# «ІНФОРМАЦІЙНІ МОДЕЛІ, СИСТЕМИ ТА ТЕХНОЛОГІЇ»

**18-19 грудня 2024 року**

**ТЕРНОПІЛЬ**
**2024**

**УДК 004.738.5**

**С. Вінтонів; Є. Тиш, канд. техн. наук**
(Тернопільський національний технічний університет імені І. Пулюя, Україна)

## ОПТИМІЗАЦІЯ КІЛЬКОСТІ ЗАПИТІВ ДО БАЗИ ДАНИХ ДЛЯ GRAPHQL-API З ВИКОРИСТАННЯМ DATALOADER

**S. Vintoniv; Ie. Tysh, Ph.D.**
Ternopil Ivan Puluj National Technical University, Ukraine

## OPTIMIZING THE NUMBER OF DATABASE QUERIES FOR GRAPHQL-API USING DATALOADER

Оптимізація запитів до бази даних є важливим аспектом сучасної розробки інформаційних платформ особливо для систем, що використовують GraphQL. На відміну від REST API, GraphQL дозволяє клієнтам запитувати точні підмножини даних, що часто призводить до множинних запитів до бази даних («проблема N+1»). Ця проблема може призвести до зниження продуктивності та неефективного використання ресурсів. Щоб вирішити цю проблему, використовується DataLoader - утиліта пакетної обробки та кешування для оптимізації виконання запитів для GraphQL API. DataLoader дозволяє об'єднувати запити до бази даних в один запит, мінімізуючи надлишкові операції і скорочуючи час відгуку.

На додаток до пакетної обробки запитів, розширені функціональні можливості, такі як реалізація користувацьких обгорток над базою даних, можуть ще більше підвищити продуктивність. Ці обгортки попередньо обробляють запити, ефективно використовують індекси бази даних і структурують конвеєри для агрегації. Поєднуючи пакетну обробку запитів з оптимізованими обгортками запитів, можна значно зменшити навантаження на базу даних і підвищити загальну продуктивність системи.

Цей підхід відповідає сучасним практикам GraphQL, де виконання запитів і оптимізація продуктивності є ключовими для обробки великих наборів даних і високого рівня паралелізму. Інтеграція цих методів з базами даних на основі документів, такими як MongoDB, забезпечує масштабований та ефективний пошук даних.

Вивчення таких інструментів і методів дає цінну інформацію для розробки GraphQL API, підкреслюючи необхідність добре структурованих стратегій пакетної обробки, кешування та оптимізації запитів.

### Література

1. Optimizing GraphQL Queries with DataLoader. URL: https://moldstud.com/articles/p-optimizing-graphql-queries-with-dataloader
2. Declarative Data Fetching with GraphQL. URL: https://css-tricks.com/declarative-data-fetching-graphql/
3. How to solve the GraphQL N+1 problem. URL: https://hygraph.com/blog/graphql-n-1-problem

# Додаток Б

## Лістинг коду серверної частини платформи

Лістинг коду 1 – Програмний код класу AuthService

```
import jwt from 'jsonwebtoken';
import config from '../constants/config';

class AuthService {
  refreshTokens = [];

  generateAccessToken(user) {
    const payload = { _id: user._id, email: user.email };
    return jwt.sign(payload, config.jwtAccessTokenSecret, { expiresIn:
'50m' });
  }

  generateRefreshToken(user) {
    const payload = { _id: user._id, email: user.email };
    const refreshToken = jwt.sign(payload,
config.jwtRefreshTokenSecret, { expiresIn: '100d' });
    this.refreshTokens.push(refreshToken);
    return refreshToken;
  }

  invalidateRefreshToken(refreshToken) {
    this.refreshTokens = this.refreshTokens.filter((token) => token !==
refreshToken);
  }

  issueNewAccessToken(refreshToken) {
    const decodeUser = this.verifyRefreshToken(refreshToken);
    return this.generateAccessToken(decodeUser);
  }

  verifyRefreshToken(refreshToken) {
    if (!this.refreshTokens.includes(refreshToken)) {
      throw new Error('Refresh token is not valid');
    }
    return jwt.verify(refreshToken, config.jwtRefreshTokenSecret);
  }

  verifyAccessToken(accessToken) {
    return jwt.verify(accessToken, config.jwtAccessTokenSecret);
  }
}

export default new AuthService();
```

Лістинг коду 2 – Програмний код класу UserService

```
import bcrypt from 'bcrypt';
```

```
import { omit } from 'lodash';
import { ObjectId, ObjectID } from 'mongodb';
import AuthService from './AuthService';
import MongoClientProvider from './MongoClientProvider';
import ProfileService from './ProfileService';

class UserService {
  collectionName = 'users';

  getCollection = () => {
    return MongoClientProvider.db.collection(this.collectionName);
  };

  #privateFields = ['hashPassword'];

  #omitPrivateFields(user) {
    return omit(user, this.#privateFields);
  }

  async findByEmail(email, shouldIncludePrivateFields) {
    const user = await this.getCollection().findOne({ email });
    if (!user) {
      return null;
    }
    if (shouldIncludePrivateFields) {
      return user;
    }
    return this.#omitPrivateFields(user);
  }

  async findById(_id, shouldIncludePrivateFields) {
    const user = await this.getCollection().findOne({ _id: new
ObjectId(_id) });
    if (!user) {
      return null;
    }
    if (shouldIncludePrivateFields) {
      return user;
    }
    return this.#omitPrivateFields(user);
  }

  async createAccount({ email, password }) {
    let user = await this.findByEmail(email);
    if (user) {
      throw new Error('user with this email already registered');
    }
    const hashPassword = await bcrypt.hash(password, 10);
    user = { hashPassword, email, createAt: new Date() };
    const result = await this.getCollection().insertOne(user);
    await ProfileService.createProfile({ userId: result.insertedId,
email });
  }

  async loginWithPassword({ email, password }) {
    const user = await this.findByEmail(email, true);
    if (!user) {
```

```
            throw new Error('User not found');
          }
          const isPasswordCorrect = await bcrypt.compare(password,
user.hashPassword);
          if (!isPasswordCorrect) {
            throw new Error('incorrect password');
          }
          const accessToken = AuthService.generateAccessToken(user);
          const refreshToken = AuthService.generateRefreshToken(user);
          const userWithoutPrivateFields = this.#omitPrivateFields(user);
          return { accessToken, refreshToken, user: userWithoutPrivateFields
};
        }

        loginWithRefreshToken(refreshToken) {
          return AuthService.issueNewAccessToken(refreshToken);
        }

        logout(refreshToken) {
          AuthService.invalidateRefreshToken(refreshToken);
        }
      }

      export default new UserService();
```

Лістинг коду 3 – Програмний код класу BookService

```
      import { ObjectID, ObjectId } from 'mongodb';
      import FavoriteService from './FavoriteService';
      import MongoClientProvider from './MongoClientProvider';
      import NoticesAboutReleasedService from
'./NoticesAboutReleasedService';
      import RatingService from './RatingService';
      import SubscriptionsService from './SubscriptionsService';

      class BookService {
        collectionName = 'books';

        getCollection() {
          return MongoClientProvider.db.collection(this.collectionName);
        }

        getBooks = async (userId) => {
          const bookList = await this.getCollection().find({ isPrivate: false
}).sort({ createdAt: -1 }).toArray();
          const bookListWithRating = await
RatingService.calculateRatingForBookList(bookList);
          return RatingService.userCanAddRatingForBooks({ userId, bookList:
bookListWithRating });
        };

        createBook = async (book, authorId) => {
          const bookData = { ...book, createdAt: new Date(), updateAt: new
Date(), authorId: new ObjectID(authorId) };
          const result = await this.getCollection().insertOne(bookData);
          await SubscriptionsService.bookReleased({ authorId, bookId:
result.insertedId });
```

```
          return { ...book, _id: result.insertedId };
        };

        getBookById = async (_id, userId) => {
          const book = await this.getCollection().findOne({ _id:
ObjectId(_id) });
          if (!book) {
            throw new Error('Book not found');
          }
          if (book.isPrivate && !book.authorId.equals(new ObjectId(userId)))
{
            throw new Error('Book is Private');
          }
          book.isFavorite = await
FavoriteService.isFavoriteBookForCurrentUser({ bookId: _id, userId });
          return RatingService.calculateRatingForBook(book);
        };

        searchBooks = async (lineForSearch, userId) => {
          const books = await this.getCollection()
            .find({
              $or: [
                { name: { $regex: lineForSearch, $options: 'i' } },
                { description: { $regex: lineForSearch, $options: 'i' } },
              ],
              isPrivate: false,
            })
            .sort({ createdAt: -1 })
            .toArray();
          const bookListWithRating = await
RatingService.calculateRatingForBookList(books);
          return RatingService.userCanAddRatingForBooks({ userId, bookList:
bookListWithRating });
        };

        getMyBooks = async ({ userId: authorId }) => {
          const books = await this.getCollection()
            .find({ authorId: new ObjectID(authorId) })
            .sort({ createdAt: -1 })
            .toArray();
          return books;
        };

        deleteBook = async (bookId, { userId: authorId }) => {
          await RatingService.getCollectionForBook().remove({ bookId: new
ObjectID(bookId) });
          await FavoriteService.getCollectionForBooks().remove({ bookId: new
ObjectID(bookId) });
          await NoticesAboutReleasedService.removeBook(bookId);
          const result = await this.getCollection().removeOne({
            _id: new ObjectID(bookId),
            authorId: new ObjectID(authorId),
          });
          if (result.result.n === 0) {
            throw new Error('book has not been deleted');
          }
        };
```

```
    updateBook = async (_id, data, { userId: authorId }) => {
      return   this.getCollection().updateOne({   _id:   ObjectID(_id),
authorId: new ObjectID(authorId) }, { $set: data });
    };

    canViewBook = (book, userId) => {
      const authorObjectId = new ObjectID(book.authorId);
      const userObjectId = new ObjectID(userId);
      if (authorObjectId.equals(userObjectId)) {
        return true;
      }
      return false;
    };

    changePrivacyOfBook = ({ bookId, authorId, isPrivate }) => {
      return this.getCollection().updateOne(
        { _id: new ObjectID(bookId), authorId: new ObjectID(authorId) },
        { $set: { isPrivate } }
      );
    };

    getFavoritesBooks = async (userId) => {
      const result = await FavoriteService.getFavoritesBooks(userId);
      if (result.length === 0) {
        return [];
      }
      const booksId = result.map((item) => {
        return { _id: item.bookId };
      });
      const favoritesBooks = await this.getCollection()
        .find({ $or: booksId, isPrivate: false })
        .sort({ createdAt: -1 })
        .toArray();
      return RatingService.calculateRatingForBookList(favoritesBooks);
    };

    getBooksByUserId = async (userId) => {
      const userBooks = await this.getCollection()
        .find({ authorId: new ObjectId(userId), isPrivate: false })
        .sort({ createdAt: -1 })
        .toArray();
      return RatingService.calculateRatingForBookList(userBooks);
    };
  }

  export default new BookService();
```

Лістинг коду 4 – Програмний код класу GraphQL схеми книг

```
import { gql } from 'apollo-server-express';
import BookService from '../../services/BookService';
import RatingService from '../../services/RatingService';
import isAuthorizedUser from '../isAuthorizedUser';
import UserService from '../../services/UserService';
import FavoriteService from '../../services/FavoriteService';
```

```
export const typeDefs = gql`
  type Book {
    _id: ID!
    name: String!
    img: String!
    authorId: ID!
    author: User!
    genre: String!
    otherAuthors: [String]!
    pagesQuantity: Int!
    isPaid: Boolean!
    price: Float
    rating: Float!
    description: String!
    userCanAddRating: Boolean!
    bookURL: String!
    createAt: Float!
    updateAt: Float!
    isPrivate: Boolean!
  }

  input BookCreateInput {
    name: String!
    img: String!
    genre: String!
    otherAuthors: [String]!
    pagesQuantity: Int!
    isPaid: Boolean!
    price: Float
    description: String!
    bookURL: String!
    isPrivate: Boolean!
  }
  input BookUpdateInput {
    name: String
    img: String
    genre: String
    otherAuthors: [String]
    pagesQuantity: Int
    isPaid: Boolean
    price: Float
    description: String
    bookURL: String
    isPrivate: Boolean
  }

  extend type Query {
    books: [Book]!
    book(id: ID!): Book!
    booksSearch(searchString: String!): [Book]!
    myBooks: [Book]!
    favoritesBooks: [Book]!
  }

  extend type Mutation {
    createBook(input: BookCreateInput!): Book!
    addRatingForBook(bookId: ID!, rating: Int!): Book!
```

```
        deleteBook(bookId: ID!): Boolean!
        updateBook(bookId: ID!, input: BookUpdateInput!): Book!
        changePrivacyOfBook(bookId: ID!, isPrivate: Boolean!): Book!
        addBookToFavorites(bookId: ID!): Boolean!
        removeBookFromFavorites(bookId: ID!): Boolean!
      }
    `;

    const safeFindBook = async (bookId, userId) => {
      const book = await BookService.getBookById(bookId, userId);
      if (!book) {
        throw new Error('Book not found');
      }
      const canViewBook = BookService.canViewBook(book, userId);
      if (!canViewBook) {
        throw new Error('Access denied');
      }
      return book;
    };

    export const resolvers = {
      Query: {
        books: async (root, params, context) => {
          isAuthorizedUser(context);
          const { userId } = context;
          const books = await BookService.getBooks(userId);
          return books;
        },
        book: async (root, { id }, context) => {
          isAuthorizedUser(context);
          const { userId } = context;
          const book = await BookService.getBookById(id, userId);
          return book;
        },
        booksSearch: async (root, { searchString }, context) => {
          isAuthorizedUser(context);
          const { userId } = context;
          const booksFound = await BookService.searchBooks(searchString,
userId);
          return booksFound;
        },
        myBooks: async (root, params, context) => {
          isAuthorizedUser(context);
          const { userId } = context;
          const books = await BookService.getMyBooks({ userId });
          return books;
        },
        favoritesBooks: async (root, params, context) => {
          isAuthorizedUser(context);
          const { userId } = context;
          const favoritesBooks = await
BookService.getFavoritesBooks(userId);
          return favoritesBooks;
        },
      },

      Mutation: {
```

```
        createBook: async (root, { input }, context) => {
          isAuthorizedUser(context);
          const { userId } = context;
          const book = await BookService.createBook(input, userId);
          return book;
        },
        addRatingForBook: async (root, { bookId, rating }, context) => {
          isAuthorizedUser(context);
          const { userId } = context;
          await RatingService.addRatingForBooks({
            bookId,
            userId,
            rating,
          });
          const book = await BookService.getBookById(bookId, userId);
          return book;
        },
        deleteBook: async (root, { bookId }, context) => {
          isAuthorizedUser(context);
          const { userId } = context;
          await safeFindBook(bookId, userId);
          await BookService.deleteBook(bookId, { userId });
          return true;
        },
        updateBook: async (root, { bookId, input }, context) => {
          isAuthorizedUser(context);
          const { userId } = context;
          await BookService.updateBook(bookId, input, { userId });
          const book = await BookService.getBookById(bookId, userId);
          return book;
        },
        changePrivacyOfBook: async (root, { bookId, isPrivate }, context)
=> {
          isAuthorizedUser(context);
          const { userId } = context;
          await safeFindBook(bookId, userId);
          await   BookService.changePrivacyOfBook({   bookId,   isPrivate,
authorId: userId });
          const book = await BookService.getBookById(bookId, userId);
          return book;
        },
        addBookToFavorites: async (root, { bookId }, context) => {
          isAuthorizedUser(context);
          const { userId } = context;
          await FavoriteService.addBookToFavorites({ bookId, userId });
          return true;
        },
        removeBookFromFavorites: async (root, { bookId }, context) => {
          isAuthorizedUser(context);
          const { userId } = context;
          await  FavoriteService.removeBookFromFavorite({  bookId,  userId
});
          return true;
        },
      },
      Book: {
        author: async (root) => {
```

```
            const user = await UserService.findById(root.authorId);
            return user;
          },
        },
      };
```

Лістинг коду 5 – Програмний код DataLoader по Mongo запиту

```
export const loaderByQuery = (Collection) => {
  const loader = new DataLoader(
    async (args) => {
      loader.clearAll();
      const initialMatch = { $match: { $or: [] } };
      const facetQuery = args.reduce((acc, { query, sort }, i) => {
        initialMatch.$match.$or.push(query);
        const pipeline = [{ $match: query || { _id: null } }];
        const sortToPipeline = sort && Object.keys(sort).length ? sort : {};
        if (!sortToPipeline._id) sortToPipeline._id = 1;
        pipeline.push({ $sort: sortToPipeline });
        acc[i] = pipeline;
        return acc;
      }, {});
      const arr = [initialMatch, { $facet: facetQuery }];
      const [resultObject] = await Collection.aggregate(arr).toArray();
      return args.map((_q, i) => resultObject[i] || []);
    },
    { cacheKeyFn: (key) => JSON.stringify(key) }
  );
  return async (data) => {
    return loader.load({ ...data });
  };
};
```

Лістинг коду 6 – Програмний код DataLoader по ідентифікатору

```
const loaderById = (Collection) => {
  const loader = new DataLoader(
    async (data) => {
      loader.clearAll();
      const obj = {};
      const users = await Collection.find({
        _id: { $in: data.map(€ => e._id) },
      }).fetchAsync();
      users.forEach((u) => {
        obj[u._id] = u;
      });
      return data.map(({ _id }) => obj[_id] || null);
    },
    { cacheKeyFn: (key) => JSON.stringify(key) }
  );
  const load = async (_id) => {
    if (!_id) return undefined;
    return loader.load({ _id });
  };
  return load;
};
```