ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка web-застосунку для відстеження прогресу читання книг»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Софія ПОНОМАРЕНКО
(підпис)

Виконав: здобувачка вищої освіти групи ПД-43

Софія ПОНОМАРЕНКО

Керівник: Віталій ЗАЛИВА
доктор філософії (PhD)

Рецензент: _____

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Пономаренко Софії Михайлівні

1. Тема кваліфікаційної роботи: «Розробка web-застосунку для відстеження прогресу читання книг»
керівник кваліфікаційної роботи доктор філософії (PhD) Віталій ЗАЛИВА,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» лютого 2025 р. № 56.

2. Строк подання кваліфікаційної роботи «02» червня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: офіційна документація HTML5, CSS3, JavaScript, Bun.js, Chart.js; технічна документація OAuth 2.0 та Google Authorization API; приклади реалізації адаптивного інтерфейсу та збереження конфіденційних змінних у .env; інструменти Figma, Postman, SQLite та VS Code.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Аналіз предметної галузі та існуючих способів відстеження прогресу читання книг.
2. Проектування та розробка в застосунку для відстеження прогресу читання книг.

3. Програмна реалізація та тестування веб застосунку для відстеження прогресу читання книг.
4. Тестування веб застосунку для відстеження прогресу читання книг.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Вимоги до програмного забезпечення.
3. Програмні засоби реалізації.
4. Діаграма варіантів використання.
5. Діаграма класів MVC
6. Структура бази даних
7. Діаграма діяльності таймера
8. Екранні форми.
9. Демонстраційне відео роботи застосунку
10. Апробація результатів дослідження

6. Дата видачі завдання «25» лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	25.02.-04.03.2025	
2	Аналіз та дослідження існуючих аналогів	05.03-13.03.2025	
3	Огляд існуючих методів та аналогів web-застосунку для відстеження прогресу читання книг	14.03-19.03.2025	
4	Проектування web-застосунку для відстеження прогресу читання книг	20.03-26.03.2025	
5	Програмна реалізація застосунку	27.03-23.04.2025	
6	Тестування застосунку	24.03-28.03.2025	
7	Оформлення роботи: вступ, висновки, реферат	29.04-11.05.2025	
8	Розробка демонстраційних матеріалів	12.05-18.05.2025	
9	Попередній захист роботи	19.05-30.05.2025	

Здобувачка вищої освіти _____ Софія ПОНОМАРЕНКО
(підпис)

Керівник
кваліфікаційної роботи _____ Віталій ЗАЛИВА
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 48 стор., 4 табл., 17 рис., 15 джерел.

Мета роботи – спрощення відстеження прогресу читання книг за допомогою web-застосунку з впровадженими засобами інтерактивного моніторингу.

Об'єкт дослідження – відстеження прогресу читання книг.

Предмет дослідження – засоби розробки web-застосунку для відстеження прогресу читання книг.

Короткий зміст роботи: В роботі розроблено web-застосунок для відстеження прогресу читання книг. Проаналізовано предметну область, способи фіксації читацького прогресу, функціональні та нефункціональні вимоги до системи. Розроблено структуру бази даних, архітектуру застосунку за шаблоном MVC та діаграми, що моделюють логіку роботи та є програмно основні функції: реєстрація та вхід користувача, додавання, редагування та видалення книг, запуск таймера читання, перегляд статистики у вигляді графіків, зміна теми інтерфейсу з урахуванням доступності для людей із вадами зору. В роботі використано HTML, CSS і JavaScript для клієнтської частини, Bun.js для серверної логіки, SQLite як систему управління базами даних, Chart.js для побудови графіків, OAuth 2.0 для авторизації через Google, а також Postman для тестування API.

Сферою використання застосунку є впорядкування особистих читацьких записів та організація процесу читання.

КЛЮЧОВІ СЛОВА: WEB-ЗАСТОСУНОК, ЧИТАННЯ, ПРОГРЕС ЧИТАННЯ, ВІДСТЕЖЕННЯ ПРОГРЕСУ, BUN.JS, OAUTH 2.0, CHART.JS, АВТОРИЗАЦІЯ

ЗМІСТ

ВСТУП

Актуальність теми та обґрунтування її вибору: Нині, у сучасному інформаційному середовищі, велика кількість людей прагне розвиватися через читання, виникає потреба у зручному способі контролю за власним читацьким прогресом.

Запам’ятовувати деталі прочитаного, кількість сторінок, дату завершення читання книги або навіть назви - з часом стає складним завданням. Водночас багато користувачів хочуть не просто читати, а й аналізувати своє читання, свою активність, писати нотатки, оцінювати книги, робити списки книг які вони читали або планують прочитати.

Завдяки розвитку веб-технологій стало можливим створення web-застосунків, що автоматизують ці процеси, забезпечують зберігання даних, візуалізацію статистики та доступ до особистого кабінету з будь-якого пристрою. Це особливо актуально для студентів, учнів, вчителів та просто активних читачів. Розробка такого застосунку відповідає сучасним запитам користувачів і має високу практичну значущість.

Об’єктом дослідження відстеження прогресу читання книг.

Предметом дослідження є засоби розробки web-застосунку для відстеження прогресу читання книг.

Метою роботи є спрощення відстеження прогресу читання книг за допомогою web-застосунку з впровадженими засобами інтерактивного моніторингу.

Методи дослідження включають аналіз літературних та інтернет-джерел, порівняльний аналіз аналогів, моделювання архітектури програмного забезпечення, прототипування користувацького інтерфейсу, реалізація та тестування.

Для досягнення поставленої мети в рамках бакалаврської роботи виконано наступні задачі:

1. Провести огляд та аналіз існуючих способів відстеження прогресу читання книг: запам’ятовування: ручне фіксування, цифрове фіксування .
2. Сформувати функціональні та нефункціональні вимоги до web-застосунку для відстеження прогресу читання книг, з урахуванням потреб користувачів та технічних обмежень.
3. Спроектувати архітектурну модель, базу даних і користувацький інтерфейс web-застосунку для відстеження прогресу читання книг.
4. Реалізувати web-застосунок для відстеження прогресу читання книг з базою даних, серверною логікою та користувацьким інтерфейсом на основі сучасних web-технологій.
5. Здійснити тестування функціональності роботи web-застосунку для відстеження прогресу читання книг: ручне тестування, тестування бази даних, тестування адаптивності.

Практична значущість результатів полягає створенні зручного цифрового інструменту, який дозволяє користувачам систематизувати інформацію про прочитані книги, вести нотатки, переглядати динаміку читання, встановлювати цілі та контролювати свою активність. Розроблений web-застосунок може бути використаний у повсякденному житті студентами, викладачами, бібліотекарями та всіма охочими організувати свій читацький досвід у зручній цифровій формі.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ІСНУЮЧИХ СПОСОБІВ ВІДСТРЕЖЕННЯ ПРОГРЕСУ ЧИТАННЯ КНИГ

1.1 Аналіз предметної галузі

Читання - це одна з основних форм засвоєння інформації, яка має глибоке історичне, культурне та психологічне значення. Воно є не лише інструментом навчання, а й засобом особистого розвитку, пізнання світу та самовираження.

Історично, читання виконувало ключову роль у збереженні та передачі знань. З появою письма почався процес фіксації інформації, що дало змогу людству передавати досвід від покоління до покоління, розвивати науку, філософію, літературу, право. У минулому здатність до читання було ознакою того, що людина була освіченою та належала до еліти, а нині, з розвитком освіти, читання стало доступним майже кожному.

Доведено, що регулярне читання покращує пам’ять, аналітичні здібності, знижує рівень стресу, сприяє розвитку уяви та креативності [1]. Також, для багатьох, читання виконує і терапевтичну функцію - слугує відпочинком, способом втекти від повсякдення, пережити власні емоції через героїв.

В сучасному світі цифрових технологій процес читання літератури зазнає трансформацій, які стосуються не лише формату книг, але й того, як користувач взаємодіє з інформацією, яку він читає [2]. У відповідь на потреби активних читачів розвивається новий напрям у сфері інформаційних технологій - цифрове відстеження читацької активності.

Читачі сьогодення є не лише споживачами літературного контенту, а й активними учасниками процесу планування та саморозвитку [3]. Багато з них ставлять перед собою цілі, пов’язані з кількістю прочитаних книг, часовими межами, жанровими пріоритетами або вивченням певної тематики. У зв’язку з цим зростає потреба у створенні інтерактивних інструментів, які дозволяють

створювати власні читацькі історії, візуалізувати статистику та відстежувати власний прогрес.

Проєкт охоплює сферу інформаційних технологій у культурно-освітньому контексті, зокрема ті аспекти, що пов’язані з підтримкою та стимулюванням читацької активності. Програмні продукти, розроблені в цій галузі, мають на меті покращити досвід користувачів під час взаємодії з літературою.

1.2 Аналіз методів відстеження прогресу читання книг

Існує декілька основних методів, за допомогою яких можна відстежувати своє читання. Найпоширенішими з них є запам’ятовування, ручне фіксування даних (наприклад, у блокнотах або таблицях), а також використання електронних засобів і спеціалізованих web-застосунків, які автоматизують та спрощують цей процес. Залежно від цілей користувача, обсягу прочитаного та звичок, кожен із цих підходів має свої переваги та недоліки.

1.2.1 Запам’ятовування як метод фіксації прогресу читання

Запам’ятовування - це самий простий, інтуїтивно закладений [4], але водночас найменш надійний спосіб збереження інформації про прочитані книги. У цьому випадку людина покладається лише на власну пам’ять, намагаючись утримати в голові, які книги вже були прочитані, на якому етапі читання він зупинився, про що говорилося в останньому розділі тощо.

Такий підхід часто використовується людьми, які не читають часто або не мають потреби в системному веденні читацької історії. На перший погляд запам’ятовування здається зручним та простим, оскільки не вимагає додаткових зусиль, інструментів чи часу. Однак на практиці він має значну низку недоліків, особливо якщо людина читає велику кількість книг або робить тривалі перерви між читанням.

По-перше, людська пам’ять є обмеженою. Навіть самі організовані люди з часом починають втрачати спогади - важко згадати не лише точне місце, де була

пауза в читанні, але й зміст, важливі цитати, враження від книги. Це може призвести до зниження ефективності повторного читання та навіть зменшити бажання продовжувати читати.

По-друге, запам’ятовування не дозволяє вести об’єктивний облік власних читацьких досягнень. Наприклад, людина не завжди може точно згадати, скільки книг вона прочитала за рік, чи зможе порівняти кількість сторінок, прочитаних у різні часові періоди життя. Для багатьох сучасних користувачів, які прагнуть систематизувати свою діяльність, відсутність статистики ускладнює визначення та формування читацьких цілей.

По-третє, емоційна пам’ять також може підвести. Забуваються деталі сюжету, описи та імена персонажів, те що людина відчувала під час процесу читання, а отже - знижується загальне враження від книги. Емоційні спогади по-різному згадуються залежно від віку, цілей пошуку спогаду та настрою [5]. Повторне читання твору без контексту або незрозуміле продовження можуть зруйнувати цілісність сприйняття літератури.

1.2.2 Ручне фіксування прогресу читання

Ручне фіксування читацького прогресу - це традиційний спосіб, який передбачає внесення інформації про прочитані книги вручну, без використання спеціалізованих інструментів. Він досить добре працює, як метод для запам’ятовування даних [6], охоплює різноманітні форми запису, але основною формою є запис простих нотаток у зошитах та блокнотах.

Цей підхід довгий час залишався основним методом обліку читацької активності. Навіть у наш час, незважаючи на технологічний розвиток, багато людей продовжують використовувати ручне фіксування як візуально зручний, особистий і контрольований спосіб збереження даних про прочитане.

Цей метод має свої переваги, наприклад гнучкість формату - читачі можуть самостійно вибирати, що саме фіксувати - назви книг, авторів, дати початку та завершення, жанри, короткі враження, улюблені цитати або кількість сторінок. Для кращого сприймання читачі можуть навіть робити невеликі малюнки чи примітки,

щоб запис був більш візуалізований та міг передавати більше інформації. Структура записів у ручному форматі повністю контролюється користувачем, а самі дані записів є необмеженими.

Для деяких людей, ручна фіксація має тактильний ефект. Для багатьох читачів сам процес написання в блокноті чи ведення читацького щоденника має естетичну та емоційну цінність. Він сприяє концентрації, зануренню в процес та створює відчуття особистого архіву.

Також ручне фіксування не потребує доступу до інтернету, наявності смартфону або ПК. Це особливо важливо для людей, які свідомо хочуть обмежити цифрове навантаження або не мають вільного доступу до техніки.

Тим не менш, ручне ведення записів має певні обмеження. По-перше, існують обмеження щодо обробки та аналізу даних. Зробити це вручну досить складно та трудомістко, якщо користувач хоче знати скільки книг він прочитав за рік, скільки сторінок улюбленого жанру він опрацював або якщо він хоче скласти графік свого прогресу. Автоматизація відсутня, а отже - потребує додаткових зусиль.

По-друге, є висока ймовірність того, що дані будуть втрачені або пошкоджені. Фізичні носії, такі як паперові зошити, наприклад, можуть загубитися, намокнути або зіпсуватися.

По-третє, низький рівень інтерактивності. Ручні записи не надають користувачеві зворотного зв’язку, нагадувань, візуалізації прогресу у вигляді діаграм тощо. саме ці функції мотивують користувачів читати, особливо якщо користувач ставить перед собою цілі.

Ще одним важливим аспектом є те, що ручне фіксування не масштабоване. Чим більше книг читає людина, тим складніше підтримувати порядок у своїх записах. Зі зростанням кількості даних, пошук необхідної інформації, систематизація та оновлення своїх записів стають дедалі складнішими.

Попри це, ручне фіксування залишається популярним серед багатьох людей, особливо шанувальників “повільного читання”, педагогів, літературознавців, а

також тих, хто цінує фізичну взаємодію з текстом. Для багатьох - це своєрідна ритуальна складова читання, яка робить процес більш усвідомленим і глибоким.

Таким чином, ручне фіксування можна вважати перехідною ланкою між запам’ятовуванням і використанням сучасних електронних засобів. Хоча воно не забезпечує автоматизацію та цифрову інтеграцію, воно дозволяє структурувати дані які людина записує без жодних обмежень. У контексті цифрового світу воно все частіше поступається місцем спеціалізованим web-застосункам, які об’єднують переваги ручного контролю та технологічної зручності.

1.2.3 Цифрове фіксування прогресу читання

Швидкий розвиток цифрового читання популяризує його серед людей, і це є доводить його ефективність [7]. Цифрове фіксування прогресу читання стало логічним продовженням еволюції читацької культури в умовах стрімкого розвитку інформаційних технологій. На відміну від запам’ятовування та ручного фіксування, цифрові засоби забезпечують автоматизацію, візуалізацію та інтерактивність, перетворюючи процес читання на більш контрольований, гнучкий і мотивувальний.

Сюди входять різні типи цифрових інструментів:

- web-застосунки, які працюють у браузері та дозволяють користувачу з будь-якого пристрою мати доступ до свого читацького кабінету;
- мобільні застосунки, орієнтовані на смартфони і планшети;
- електронні таблиці або журнали з розширеною логікою;
- вбудовані функції в електронні книги;
- онлайн-сервіси для створення читацьких списків та викликів.

Ці засоби зазвичай поєднують в собі зберігання даних, статистику, планування, нотатки та візуальні індикатори прогресу, а також мають свої переваги.

Більшість web-застосунків автоматично підраховують кількість прочитаних сторінок, книг, годин, середню швидкість читання. Завдяки цьому користувач може відстежувати динаміку за день, тиждень, місяць або навіть рік. Також можливе

створення візуалізацій у вигляді діаграм, гістограм, таймлайнів, які допомагають краще зрозуміти власні читацькі звички.

Завдяки хмарним технологіям, цифрові засоби дозволяють синхронізувати дані на різних пристроях - комп’ютері, планшеті, смартфоні. Це забезпечує безперервність збереження досвіду, незалежно від того, де саме користувач читає або вносить зміни.

Багато застосунків дозволяють встановлювати цілі щодо читання (наприклад, 20 книг на рік), виклики (наприклад, “30 днів читання підряд”) або ж щоденні нагадування. Це створює ігровий ефект, який підвищує мотивацію та підтримує дисципліну. Читання перестає бути нудним чи випадковим, і набуває форми особистого проєкту.

Цифрові трекери дозволяють створювати детальні картки книги - з обкладинкою, жанром, оцінкою, нотатками, цитатами. Це набагато простіше, ніж вести записи вручну, і дозволяє створити повну електронну бібліотеку користувача, яку можна легко сортувати, фільтрувати або переглядати відповідно до різних критеріїв.

На відміну від ручних методів, цифрові засоби зазвичай мають автоматичне резервне копіювання або дозволяють експортувати дані у формат CSV, JSON, PDF. Це зменшує ризики втрати важливої інформації.

Деякі веб-застосунки, мають соціальну складову, що означає, що ви можете ділитися своїми враженнями з друзями, бачити рецензії інших людей, обговорювати книги та отримувати рекомендації. Це додає відчуття соціуму та спільності, які посилюють мотивацію до читання.

Разом з цим, цифровий підхід має і свої недоліки, наприклад потреба в інтернеті або підключенні до сервера (не всі сервіси працюють онлайн); проблеми з конфіденційністю, якщо сервіс вимагає авторизацію або використовує особисті дані; перенасичення функціоналом, коли користувач губиться в складному інтерфейсі; технічна залежність, бо при збоях або видаленні сервісу дані можуть бути втрачені.

Загалом цифрові інструменти значно перевершують традиційні методи, особливо в контексті довготривалого та системного читання, роботи з великим обсягом книжок або бажання зберігати повну читацьку історію. Сучасні web-застосунки об’єднують у собі найкращі риси ручного контролю та цифрової аналітики, надаючи користувачам зручний інструмент для організації особистої бібліотеки, планування, фіксації та мотивації до читання.

1.3 Огляд та аналів застосунків-аналогів для відстеження прогресу читання книг

На сьогодні існує низка готових цифрових рішень, що дозволяють користувачам відстежувати читацький прогрес, створювати віртуальні книжкові полиці та зберігати інформацію про прочитані твори. До таких аналогових web-застосунків належать Italic Type, LibraryThing та Libib. Кожен із них має власний функціонал, орієнтований на різні аудиторії - від особистих читацьких журналів до ведення повноцінного бібліотечного каталогу.

1.3.1 Веб-застосунок Italic Type

Italic Type [8] - це веб-застосунок для читачів, які прагнуть глибшого, усвідомленого та спокійного досвіду читання. Він створений командою з трьох осіб які прагнули створити простір для читачів, вільний від реклами та соціального шуму. Платформа підкреслює значення глибокого занурення в текст, рефлексії та спільноти однодумців. Орієнтований застосунок на якість, а не на кількість, і створений для тих, хто цінує рефлексію, приватність та мінімалізм.

Його основними функціями є:

- Ведення читацького журналу - кожна книга має власну “Book Board” - дошку, простір для нотаток, цитат, ідей та вражень. Вона дозволяє зберігати особисті роздуми в одному місці.

- Імпорт даних - застосунок має можливість легко імпортувати читацьку історію з Goodreads або The StoryGraph, включаючи рейтинги, дати та списки для читання.
- Додаток зберігає приватність - всі дані є приватні за замовчуванням. Italic Type не продає та не передає особисту інформацію третім сторонам.
- Мобільний доступ - додаток доступний як веб-застосунок, так і додаток для iOS.

Italic Type має переваги, такі як:

- Мінімалістичний дизайн, в якому інтерфейс не містить зайвих елементів, що сприяє концентрації на читання.
- Застосунок безкоштовний, з можливістю підтримати розробників через добровільні внески.

Italic Type, ідеально підходить для читачів, які шукають спокійний, глибокий та особистий досвід читання, без відволікаючих факторів та соціального тиску.

1.3.2 Веб-застосунок LibraryThing

LibraryThing [9] - потужний веб-додаток для каталогізації книг, який поєднує функції бібліотечного менеджера з елементами соціальної мережі. Створений у 2005 році сервіс дозволяє користувачам створювати власні каталоги книг, обмінюватися рекомендаціями та взаємодіяти з іншими читачами.

Серед основних функцій можна визначити такі:

- Користувачі можуть каталогізувати книги, додавати їх до своєї бібліотеки, використовуючи ISBN, назву або автора. Сервіс підтримує імпорт даних з понад 2 000 бібліотек, включаючи Бібліотеку Конгресу США, Британську бібліотеку та інші.
- Теги та класифікація. Застосунок дає можливість додавати теги до книг для зручної організації та пошуку.

- Соціальні функції: Користувачі можуть обговорювати книги на форумах, переглядати бібліотеки інших учасників і отримувати рекомендації на основі спільних інтересів.
- Також застосунок має TinyCat - спеціально розроблений для малих бібліотек інтерфейс, який дозволяє організовувати та керувати колекціями до 20 000 предметів.

Перевагами застосунку можна назвати:

- Глибоку каталогізацію, підтримку детального опису книг, включаючи різні видання та переклади.
- Можливість імпорту даних з численних бібліотек світу завдяки інтеграції з бібліоетками. З
- Активну спільноту, де форуми та обговорення сприяють обміну думками та рекомендаціями книг.

LibraryThing доступний як веб-застосунок, а також має мобільні додатки для iOS та Android. Сервіс безкоштовний для користувачів, що додають до 200 книг; для більших колекцій передбачена платна підписка. LibraryThing є ідеальним інструментом для читачів, які бажають систематизувати свою бібліотеку, отримувати рекомендації та взаємодіяти з іншими книголюбами.

1.3.3 Веб-застосунок Libib

Libib [10] - це багатофункціональний веб-застосунок для каталогізації та управління особистими або організаційними бібліотеками. Він дозволяє користувачам створювати та керувати своєю медіа-колекцією, забезпечуючи зручний інтерфейс для організації та доступу до медіа-ресурсів.

Основними його функціями є:

- Каталогізація медіа. Libib підтримує створення до 100 колекцій різних типів медіа, таких як книги, фільми, музика, відеоігри та ігри на мобільному телефоні.

- Хмарна синхронізація - щоб гарантувати актуальність даних, зміни, внесені через веб-інтерфейс або мобільний додаток, автоматично синхронізуються між пристроями.
- Користувачі можуть імпортувати колекції з CSV-файлів та експортувати свої дані для резервного копіювання або аналізу.
- В додатку також є публікація колекцій. Створення публічних сторінок для колекцій, дозволяє ділитися ними з іншими користувачами або громадськістю.

Перевагами можна назвати:

- Простоту використання, завдяки інтерфейсу який можна легко зрозуміти, що допомагає швидкому додаванню та організації елементів.
- Гнучкість. Підтримуються різні види медіа, а також є можливість створювати змішані колекції.
- Безпеку даних, бо Libib не використовує реклами та не передає особисту інформацію третім сторонам.
- Масштабованість, яка підходить як для особистого використання, так і для великих організацій з розширеними потребами.

Libib можна використовувати і безплатно, але він також має декілька варіантів платних версій:

- Libib Pro: Призначений для шкіл, бібліотек та організацій. Включає функції управління позиками, читачами, багатокористувацький доступ, генерацію звітів та інтерактивний онлайн-каталог.
- Libib Ultimate: Орієнтований на великі установи з кількома філіями. Пропонує централізоване управління кількома обліковими записами, інтеграцію з важливими бібліотеками та пріоритетну підтримку.

Libib є ефективним інструментом для організації та управління медіа-колекціями, що поєднує простоту використання з потужними функціями для різних категорій користувачів.

1.3.4 Порівняльний аналіз застосунків-аналогів

У таблиці 1.1 показано проведений аналіз чотирьох веб-застосунків за ключовими функціональними критеріями, які мають значення для сучасного читача. Це дозволяє виявити як сильні, так і слабкі сторони кожного з рішень.

Italic Type використовується переважно для запису вражень, отриманих від прочитаного, і створення особистого читацького простору. Незважаючи на те, що він має адаптивний інтерфейс і зручний для особистих нотаток, він не підтримує таймер, не надає статистику у вигляді графіків і не дозволяє авторизувати себе через Google. Приватність, мінімалізм і якість взаємодії з текстом є головними пріоритетами.

LibraryThing - один з найдавніших сервісів для каталогізації книг. Платформа орієнтована на створення повноцінної електронної бібліотеки, підтримує великий набір метаданих, адаптована до різних пристроїв а також має велику аудиторію. Однак вона не надає таймер для читання, не має авторизації через Google і не спеціалізується на інтерактивній візуалізації читацької статистики.

Libib - багатофункціональний застосунок, що дозволяє керувати не лише книгами, а й іншими типами медіа. Має адаптивний інтерфейс, дозволяє зберігати дані про книги та налаштовувати обліковий запис. Проте, як і LibraryThing, не підтримує графічну статистику, таймер і авторизацію через Google, що обмежує його інтерактивність як засобу для щоденного контролю прогресу читання.

BookProgress - унікальний web-застосунок, який поєднує гнучкість введення даних, графічну статистику (сторінки за місяць, жанри, статуси читання), адаптивний дизайн, таймер для відстеження часу читання. На відміну від інших щастосунків BookProgress підтримує авторизацію в систему за допомогою Google. Він також дозволяє налаштовувати обліковий запис користувача, змінювати тему оформлення, показує візуальні статистичні дані на основі тих що вводив користувач працює як персоналізований читацький щоденник. Завдяки цьому BookProgress охоплює повний цикл взаємодії користувача з книгами - від додавання до аналізу результатів.

Таблиця 1.1

Порівняння існуючих програм-аналогів

	Italic Type	LibraryThing	Libib	BookProgress
Збереження інформації введеної користувачем про книги до бібліотеки	-	+	+	+
Графічне відображення статистики читання	-	+	+	+
Адаптивність web-застосунку до різних пристроїв	+	+	+	+
Таймер для читання	-	-	-	+
Налаштування облікового запису	+	+	+	+
Авторизація за допомогою Google	-	-	-	+

2 ПРОЄКТУВАННЯ ТА РОЗРОБКА WEB-ЗАСТОСУНКУ ДЛЯ ВІДСТЕЖЕННЯ ПРОГРЕСУ ЧИТАННЯ КНИГ

2.1 Визначення вимог до web-застосунку

На етапі розробки будь-якого програмного забезпечення визначення вимог є одним із найважливіших кроків. Саме правильно сформульовані вимоги дозволяють розробнику зрозуміти, яким має бути кінцевий продукт, які функції він повинен виконувати, як взаємодіяти з користувачем, які обмеження існують і що вважається успішним результатом [11]. Без чіткого переліку вимог є ризик створити систему, яка не відповідатиме очікуванням користувача або не покриватиме реальні потреби.

Вимоги до програмного забезпечення слугують своєрідною “інструкцією”, що дозволяє планувати архітектуру, розробку, тестування та подальший розвиток проєкту. Вони можуть бути сформовані на основі аналізу предметної галузі, дослідження аналогів, очікувань цільової аудиторії та можливостей реалізації. Загалом вимоги поділяють на функціональні та нефункціональні.

У таблиці 2.1 описані вимоги до web-застосунку для відстеження прогресу читання книг.

Функціональні вимоги описують конкретні дії, які має виконувати система. Ці функції забезпечують основну взаємодію користувача з застосунком та відповідають його ключовим очікуванням. Функціональними вимогами web-застосунку для відстеження прогресу читання книг “BookProgress” є:

1. Реєстрація та авторизація користувачів.
2. Збереження інформації введеної користувачем про книги до бібліотеки.
3. Можливість пошуку книг за назвою або автором.
4. Оцінювання книг.
5. Графічне відображення статистики читання.
6. Таймер для читання.
7. Налаштування облікового запису.

Нефункціональні вимоги, у свою чергу, визначають якість, обмеження або зовнішні характеристики системи, які не пов’язані безпосередньо з її функціональністю. Такі вимоги роблять застосунок не лише корисним, а й зручним, безпечним, доступним та сучасним. Нефункціональними вимогами web-застосунку для відстеження прогресу читання книг “BookProgress” є:

1. Адаптивність web-застосунку до різних пристроїв.
2. Захист даних від SQL-ін’єкцій.
3. Можливість зміни теми інтерфейсу (темна/світла).
4. Підтримка монохромної теми інтерфейсу для людей з вадами зору.
5. Авторизація за допомогою Google.

Чітке формулювання як функціональних, так і нефункціональних вимог є обов’язковою умовою для створення ефективного, якісного та відповідного до очікувань користувача програмного продукту. У BookProgress саме вимоги стали основою, на яку спиралась вся логіка реалізації системи. читання книг.

2.2 Огляд використаних програмних засобів реалізації проєкту

Для розробки сучасних web-застосунків важливо поєднувати інструменти, що забезпечують як візуальну частину інтерфейсу, так і логіку взаємодії, обробку даних та безпеку. У процесі створення BookProgress було використано набір технологій, які охоплюють як фронтенд, так і бекенд розробку.

Завдяки гармонійному поєднанню цих засобів вдалося реалізувати повноцінний функціональний web-застосунок, який підтримує авторизацію, зберігання даних, аналіз прогресу, адаптивність і сучасний дизайн.

2.2.1 Інструмент для дизайну інтерфейсів Figma

Figma - це хмарний інструмент для дизайну інтерфейсів, створення прототипів та спільної роботи над макетами. Вона працює прямо в браузері (або через десктопний застосунок), не потребує встановлення важкого програмного забезпечення та дозволяє одночасно працювати кільком користувачам у реальному

часі. Figma особливо популярна серед веб-дизайнерів і frontend-розробників завдяки простому, інтуїтивному інтерфейсу, а також можливості створювати масштабовані, адаптивні макети, які точно відображають майбутній вигляд веб-застосунку. У ній можна створювати сторінки, компоненти, інтерактивні переходи, а також додавати опис елементів, що значно спрощує подальшу реалізацію дизайну у коді.

У проєкті BookProgress Figma була використана на етапі планування зовнішнього вигляду застосунку - перш ніж розпочати верстку HTML і CSS. Прототип сторінки web-застосунку зображений на рисунку 2.1. В цьому інструменті було створено макети основних сторінок: додавання книги, перегляду книги, списку книг та сторінки зі статистикою. Це дозволило наочно побачити, як виглядатиме структура сторінки, де саме будуть розташовані кнопки, поля, заголовки, графіки та обкладинка книги. Завдяки візуальному макету можна було легко протестувати кілька варіантів дизайну, не змінюючи жодного рядка коду, і обрати найбільш зручне та естетичне рішення для користувача.

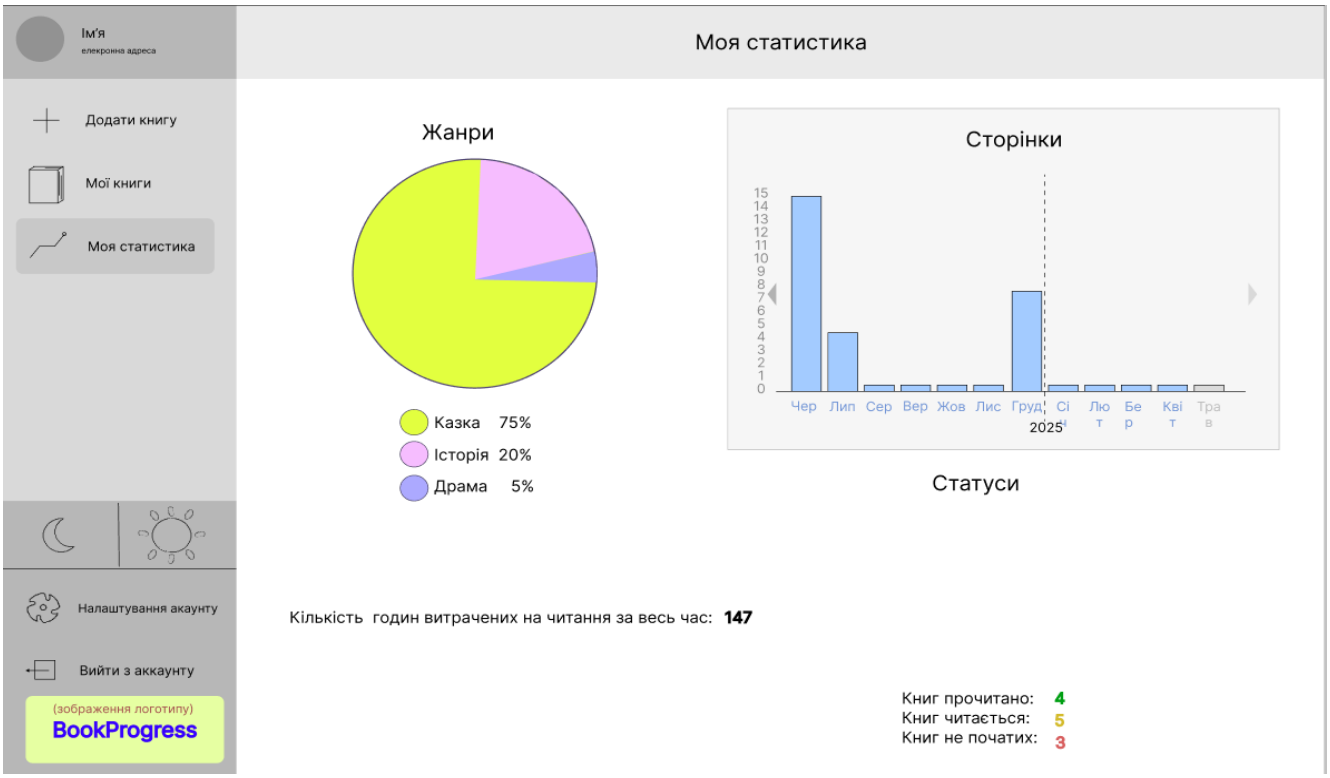


Рис. 2.1 Прототип сторінки статистики

Крім того, Figma допомогла забезпечити єдність стилю в усьому застосунку - однакові відступи, розміри шрифтів, кольори, кнопки та картки були спроєктовані в макеті, а потім точно відтворені у верстці. У проєкті також використовувалась можливість перегляду Figma-макетів на різних розмірах екранів, що дозволило заздалегідь передбачити адаптивність і зручність інтерфейсу на мобільних пристроях. Це суттєво скоротило час на виправлення під час верстки і зробило загальний дизайн більш цілісним.

Таким чином, Figma у BookProgress виконала роль візуального конструктора інтерфейсу, який забезпечив чітке уявлення про кінцевий результат, дозволив спланувати зручну структуру кожної сторінки та передав дизайн у реалізацію максимально точно і швидко. Вона була важливою ланкою між ідеєю та втіленням, допомагаючи створити естетичний, продуманий і функціональний веб-застосунок.

2.2.2 Мова розмітки HTML

HTML (HyperText Markup Language) - це мова розмітки, яка використовується для створення структури вебсторінок. Вона не є мовою програмування, адже не виконує логічних операцій або обчислень, але саме HTML відповідає за те, які елементи з’являються на сторінці та в якому порядку. Саме завдяки HTML браузер “розуміє”, де мають бути розташовані заголовки, абзаци, зображення, таблиці, кнопки, форми введення та інші складові інтерфейсу. Його можна уявити як “скелет” веб-застосунку, на який згодом “накладається” стилізація (CSS) та поведінка (JavaScript).

У проєкті BookProgress HTML відіграє ключову роль у створенні кожної сторінки застосунку. Наприклад, на сторінці додавання книги за допомогою HTML визначаються всі поля для введення інформації - назва, автор, кількість сторінок, опис, жанр, рейтинг, статус читання. HTML також використовується для створення таблиць або блоків, у яких відображаються дані про книги у розділі “Мої книги”, або для виведення детальної інформації на сторінці перегляду книги. Кожна форма, кнопка, напис чи зображення у проєкті має свою HTML-структуру.

Крім того, HTML дозволяє задавати атрибути, які є важливими для функціонування логіки взаємодії. Наприклад, кнопки типу “submit”, поля з унікальними “id”, які використовуються в JavaScript для зчитування або запису значень, а також контейнерні елементи (“div”, “section”, “main”), які об’єднують частини інтерфейсу. У поєднанні з адаптивними технологіями, HTML також слугує основою для того, щоб сторінка відображалась коректно як на комп’ютері, так і на телефоні. Завдяки чіткій HTML-структурі, застосунок BookProgress є логічно організованим, зручним для користувача та придатним для подальшого розширення.

Таким чином, HTML у цьому проєкті - це фундамент, без якого було б неможливо реалізувати жодну функціональну або візуальну частину застосунку. Він визначає не лише зовнішній вигляд кожної сторінки, а й те, як користувач взаємодіє з системою на базовому рівні.

2.2.3 Мова стилів CSS

CSS (Cascading Style Sheets) - це мова стилів, яка використовується для оформлення зовнішнього вигляду вебсторінок. Якщо HTML відповідає за структуру, то CSS визначає, як саме ця структура виглядатиме: кольори, шрифти, відстані між елементами, вирівнювання, розміри блоків, анімації, теми та адаптивність до різних екранів. CSS розділяє вміст від дизайну, що дозволяє змінювати вигляд інтерфейсу без потреби змінювати HTML-код. Він також дозволяє підтримувати єдиний стиль для всього сайту, повторно використовуючи стилі для різних сторінок або компонентів.

У проєкті BookProgress CSS відповідає за створення приємного, сучасного та зручного інтерфейсу. Завдяки CSS були стилізовані всі ключові сторінки - додавання, перегляд і редагування книг, розділ статистики, сторінки входу та реєстрації. Наприклад, саме CSS визначає кольори кнопок, тінь карток книг, рамки полів вводу, а також колірну схему світлої та темної теми, яку можна перемикати. За допомогою медіа-запитів CSS адаптує сторінку до розміру екрана, завдяки чому застосунок зручно виглядає і на комп’ютері, і на смартфоні.

Окрім базових стилів, у застосунку використовуються й деякі анімації та ефекти - наприклад, м’які переходи при наведенні, зміна кольору активного елемента, плавне відображення діаграм. CSS також дозволяє створити привабливу структуру макету: розміщення обкладинки ліворуч, інформації праворуч, сітки елементів у списках книг. Усі ці речі роблять застосунок візуально привабливим і легким у використанні, що покращує загальне враження користувача.

Загалом, CSS у проєкті BookProgress - це не лише спосіб “розфарбувати” сайт, а повноцінний інструмент, який формує зовнішню ідентичність застосунку, робить його впізнаваним, стильним та зручним у користуванні. Саме завдяки CSS BookProgress не виглядає як простий набір елементів, а сприймається як цілісний, завершений, професійно оформлений інтерфейс.

2.2.4 Мова програмування JS

JavaScript (JS) - це мова програмування, яка використовується для додавання інтерактивності та динамічної поведінки до вебсторінок. На відміну від HTML і CSS, які відповідають лише за структуру та вигляд, саме JavaScript дозволяє зробити сайт “живим” - реагувати на дії користувача, змінювати вміст без перезавантаження сторінки, взаємодіяти з сервером, обробляти дані у реальному часі. JavaScript виконується безпосередньо в браузері користувача, що робить його надзвичайно важливим для створення зручного і швидкого інтерфейсу.

У проєкті BookProgress JavaScript відіграє ключову роль у забезпеченні логіки поведінки веб-застосунку. З його допомогою реалізовано завантаження та відображення списку книг на сторінці “Мої книги”, підвантаження детальної інформації про кожну книгу на окремій сторінці перегляду, а також передача даних із форм (наприклад, при додаванні чи редагуванні книги) до серверної частини. JavaScript також відповідає за обробку подій: натискання на кнопки, перемикання теми, активацію таймера читання, валідацію введених даних та взаємодію з графіками.

Ще одна важлива роль JavaScript у цьому застосунку - це створення графіків за допомогою бібліотеки Chart.js. Саме за допомогою JS здійснюється підрахунок

даних (кількість прочитаних сторінок за місяць, розподіл жанрів, статуси книг) та побудова на їх основі візуалізацій. Крім того, JavaScript забезпечує спілкування з сервером через API: відправку запитів, отримання відповідей, обробку помилок або повідомлень для користувача. Це робить BookProgress інтерактивним і зручним - користувач не лише переглядає статичну інформацію, а й активно з нею взаємодіє.

Таким чином, JavaScript у цьому проєкті є не просто додатковим інструментом, а серцем динамічного функціоналу. Саме він забезпечує ту гнучкість, швидкість та зручність, які очікує сучасний користувач від якісного веб-застосунку.

2.2.5 Бібліотека Chart.js

Chart.js - це популярна JavaScript-бібліотека для створення інтерактивних графіків і діаграм на веб-сторінках. Вона дозволяє легко візуалізувати числові дані у зрозумілому графічному вигляді: лінійні графіки, гістограми, кругові діаграми, радарні діаграми тощо. Бібліотека використовує HTML-елемент <canvas> для побудови графіки, а також підтримує анімації, підказки, легенди та динамічне оновлення даних. Chart.js цінується за простоту використання, сучасний вигляд графіків і зручність інтеграції з JavaScript-кодом, що робить її дуже популярною серед розробників веб-застосунків.

У проєкті BookProgress бібліотека Chart.js відіграє важливу роль у візуалізації читацької статистики, роблячи її наочною та доступною для користувача. Вона використовується на сторінці статистики для створення декількох типів графіків, що відображають особистий прогрес читання. Наприклад, з її допомогою реалізована кругова діаграма, яка показує розподіл прочитаних книг за жанрами, а також стовпчастий графік, що відображає кількість прочитаних сторінок по місяцях. Ці графіки створюються на основі даних, отриманих із бази даних, і динамічно оновлюються під час завантаження сторінки.

Завдяки Chart.js користувач бачить не лише сухі цифри, а й яскраву візуалізацію, яка допомагає краще оцінити свій прогрес, виявити читацькі звички або зміни в активності протягом року. Це мотивує продовжувати читати, досягати

нових цілей та відстежувати результати. Графіки інтегровані в інтерфейс органічно, відповідають темі оформлення застосунку та адаптивно виглядають як на великих, так і на мобільних екранах. Таким чином, Chart.js є невід'ємною частиною функціоналу BookProgress, що підсилює цінність і зручність використання застосунку для кожного користувача.

2.2.6 Протокол OAuth 2.0

OAuth 2.0 - це відкритий протокол авторизації, який дозволяє веб-застосункам безпечно отримувати доступ до обмежених ресурсів користувача на сторонніх сервісах без потреби передавати або зберігати його логін і пароль [12]. Найчастіше він використовується для входу через популярні платформи, такі як Google, Facebook, GitHub тощо. Ідея полягає в тому, що користувач підтверджує свою особу не безпосередньо в застосунку, а через вже наявний обліковий запис у великому, довіреному сервісі. Після авторизації цей сервіс видає токен доступу, який застосунок використовує для ідентифікації користувача та взаємодії з його даними - лише в межах дозволеного.

У проєкті BookProgress протокол OAuth 2.0 реалізовано через інтеграцію з Google, що дає змогу користувачам швидко і безпечно входити у свій акаунт за допомогою кнопки "Увійти через Google". Це знімає потребу у створенні складної системи реєстрації з підтвердженням пошти, вигадуванням паролів і їх зберіганням. Усе, що потрібно користувачу - це натиснути кнопку входу, дозволити доступ до основних даних профілю (зазвичай це ім'я, електронна пошта та аватар), після чого застосунок автоматично реєструє нового користувача або входить під вже існуючим.

У бекенд-частині BookProgress запит до Google обробляється через сервер, який отримує унікальний токен після підтвердження користувачем. Далі цей токен перевіряється, і якщо він дійсний, користувач отримує доступ до свого персонального кабінету, де може додавати книги, бачити статистику та змінювати налаштування. При цьому конфіденційні дані (наприклад, токени або Google-клієнт-секрет) зберігаються у файлі .env, що гарантує безпеку.

Використання OAuth 2.0 у проєкті не лише полегшує життя користувачу, а й підвищує довіру до застосунку. Завдяки цьому механізму в BookProgress забезпечено зручний, швидкий і безпечний вхід без паролів, що відповідає сучасним вимогам UX та захисту персональних даних..

2.2.7 Середовище виконання Bun.js

Bun.js - це сучасне середовище виконання JavaScript і TypeScript, яке поєднує в собі можливості запуску серверного коду, управління пакетами та інструменти для зборки проєктів. Його головна особливість - висока швидкість і продуктивність, адже Bun розроблений на мові Zig і має вбудований JavaScript-движок. На відміну від класичного Node.js, Bun є більш оптимізованим та “легким”, що робить його привабливим для створення швидких і простих веб-сервісів. В свою чергу Bun – це інструмент “все в одному”. З ним розробник отримує практично все необхідне для роботи: відслідковування змін коду, HTTP та вебсокет сервер, транспіляцію, збірку проєкту, роботу з пакетами, тестування. Окрім підтримки стандартного для Node.js NPM надається і власний менеджер пакетів [13].

У проєкті BookProgress середовище Bun.js використовується як основа для запуску серверної частини веб-застосунку. Саме через нього обробляються всі запити користувачів: додавання книги, редагування, перегляд статистики, авторизація через Google, отримання або збереження даних у базі. Сервер, створений за допомогою Bun, взаємодіє з базою даних SQLite, відправляє й приймає JSON-запити, керує сесіями користувачів і виконує авторизацію через протокол OAuth 2.0.

Bun дозволив організувати бекенд-логіку швидко та просто: без зайвих налаштувань, із високою продуктивністю та зручною структурою коду. У порівнянні з Node.js, у якому часто потрібно підключати додаткові інструменти для кожного етапу (як-от, тестування, серверна обробка), Bun надає багато з цих можливостей.

Це зменшило складність розгортання проєкту, прискорило час відповіді сервера і зробило систему більш стабільною.

Загалом, у контексті BookProgress Bun.js відіграє роль серверного ядра, яке забезпечує всю функціональність взаємодії з клієнтською частиною. Завдяки йому веб-застосунок працює швидко, обробляє дії користувача в реальному часі і підтримує повний цикл взаємодії з даними - від запиту до бази до повернення результату у браузер..

2.2.8 Конфігурація .env

.env - це спеціальний конфігураційний файл, який використовується для зберігання змінних середовища у проектах, пов’язаних із розробкою програмного забезпечення. Його головна мета - винести конфіденційні або специфічні для середовища значення поза межі основного коду. Це можуть бути ключі API, паролі, ідентифікатори, порти, URL-адреси серверів або токени доступу. Завдяки використанню .env ці значення зберігаються в окремому, захищеному файлі, який зазвичай не потрапляє в публічні репозиторії - це дозволяє підвищити безпеку проєкту і спростити його конфігурацію.

У проєкті BookProgress файл .env використовується для зберігання важливої службової інформації, яка необхідна для роботи серверної частини, але не повинна бути відкритою у вихідному коді. Наприклад, тут зберігаються клієнтські ідентифікатори та секрети, які потрібні для авторизації через Google OAuth 2.0. Також у цьому файлі можуть бути вказані налаштування для підключення до бази даних, шлях до внутрішніх ресурсів або змінні для налаштування серверного середовища, такі як номер порту або режим запуску.

Під час запуску сервера на базі Bun ці змінні завантажуються з файлу .env у процес за допомогою бібліотеки dotenv, що дозволяє звертатися до них у коді через process.env. Такий підхід дає змогу легко змінювати налаштування проєкту без необхідності редагувати сам код. Наприклад, при зміні Google-клієнта достатньо оновити значення в .env, і вся система продовжить працювати коректно.

У підсумку, .env у BookProgress виконує роль централізованого сховища конфіденційних параметрів, яке не лише підвищує безпеку, а й забезпечує гнучкість при налаштуванні проєкту. Це дозволяє безпечно розгортати застосунок локально з мінімальними змінами в логіці самої програми.

2.2.9 Система керування базами даних SQLite

SQLite - це легка, вбудована реляційна система керування базами даних, яка не потребує окремого серверного процесу чи складної установки. Вона зберігає всі дані у звичайному локальному файлі на диску, що робить її зручною для невеликих веб-застосунків, мобільних застосунків або проєктів, де важлива простота розгортання. Незважаючи на свою легкість, SQLite підтримує більшість стандартних SQL-операцій: створення таблиць, фільтрацію, сортування, зв’язки між таблицями, індексацію тощо. Ця система чудово підходить для застосунків із помірним навантаженням, де потрібне збереження даних, але не потрібна розподілена інфраструктура.

У проєкті BookProgress база даних SQLite використовується для збереження всієї основної інформації: книг, які додає користувач, облікових записів, статусів читання, нотаток, жанрів, дат початку і завершення читання, а також результатів, які потім відображаються на графіках статистики. Під час взаємодії з інтерфейсом застосунку користувач може додавати нову книгу, змінювати її параметри, видаляти або переглядати деталі - і всі ці операції відбуваються шляхом зчитування або оновлення записів у базі SQLite. Завдяки швидкому доступу до файлу бази даних, зміни відображаються миттєво, без затримок.

Крім того, з боку сервера (на базі Bun.js) реалізована логіка запитів до цієї бази - що дозволяє безпечно виконувати всі операції через API. Наприклад, при відкритті сторінки “Мої книги” застосунок надсилає запит до сервера, який у свою чергу звертається до SQLite, отримує потрібні записи й повертає їх у вигляді даних для відображення. Так само працює система статистики - сервер витягує з бази кількість сторінок, дати прочитання та статуси, які потім візуалізуються на графіках.

Таким чином, у BookProgress SQLite є центральним сховищем усіх користувацьких даних, яке забезпечує збереження, обробку й швидкий доступ до інформації. Його простота, ефективність і невибагливість до ресурсів ідеально пасують для невеликого персонального веб-застосунку, орієнтованого на одного користувача або обмежену кількість облікових записів.

2.2.10 Інструмент для тестування Postman

Postman - це популярний інструмент для тестування API (інтерфейсів прикладного програмування), який дозволяє розробникам надсилати HTTP-запити до сервера, переглядати відповіді, перевіряти правильність роботи маршрутизаторів, авторизації, передачі даних тощо. Його основне призначення - дати змогу перевірити, як працює бекенд-застосунок, не використовуючи інтерфейс сайту.

Це особливо корисно під час розробки або налагодження серверної логіки, коли важливо протестувати окрему функцію - наприклад, чи правильно сервер зберігає книгу в базу, чи повертає відповідь про помилку, чи вивантажує статистику.

У проєкті BookProgress Postman активно використовувався для тестування взаємодії з базою даних SQLite через сервер Bun. За допомогою цього інструменту надсилались HTTP-запити типу POST, GET, PUT і DELETE до відповідних маршрутів API. Наприклад, перед тим як реалізувати інтерфейс для додавання книги, перевірялось, чи коректно працює запит POST /books, чи дані з нього зберігаються в базу, і чи сервер повертає очікувану відповідь. Аналогічно, за допомогою Postman тестувалось видалення книг, оновлення даних про книгу, авторизація через токени, а також отримання користувацької статистики.

Цей підхід дозволяв перевіряти серверну логіку незалежно від браузера чи клієнтської частини, що значно пришвидшувало процес розробки і відлагодження. Крім того, Postman дозволяє зберігати колекції запитів, що зручно для повторного використання під час подальшого тестування або демонстрації. У випадку з BookProgress це було особливо корисно, коли потрібно було перевірити, чи

правильно працює оновлення даних у базі при зміні статусу читання або чи відображаються точні значення на графіках.

Отже, Postman у цьому проєкті відіграв роль надійного інструменту для контролю і перевірки функціонування всіх API-запитів, які взаємодіють з базою даних. Він допоміг переконатися, що всі операції з даними - від створення до виведення - працюють коректно, без помилок і відповідно до логіки застосунку..

2.2.11 Середовище розробки VS Code

Visual Studio Code (VS Code) - це сучасне, безкоштовне середовище розробки (IDE), створене компанією Microsoft. Воно призначене для написання, редагування й організації програмного коду, а також для зручної роботи з проєктами, файлами, терміналом і системами контролю версій. VS Code підтримує велику кількість мов програмування, має розширення для автодоповнення, підсвітки синтаксису, перевірки помилок у реальному часі, дебагу, роботи з Git, а також дозволяє запускати скрипти й сервери безпосередньо з вікна редактора.

Ця гнучкість робить його ідеальним інструментом як для новачків, так і для професійних розробників.

У проєкті BookProgress саме VS Code використовувалось як основне середовище розробки. У ньому створювались та редагувались усі файли HTML, CSS і JavaScript, а також серверні скрипти на Bun.js і конфігураційні файли, зокрема .env.

Важливу роль відіграли розширення для підсвітки синтаксису JavaScript і TypeScript, автодоповнення HTML-розмітки, а також лінтери, які підказували про помилки або невідповідність коду стилістичним стандартам. Завдяки вбудованому терміналу можна було одразу запускати сервер на Bun, тестувати взаємодію з базою SQLite, встановлювати бібліотеки, оновлювати залежності або запускати локальний хостинг сайту.

Крім цього, у VS Code зручно працювати з файловою структурою проєкту: усі компоненти BookProgress - сторінки (наприклад, addbook.html, mybooks.html, viewbook.html), стилі, скрипти, серверні файли та база даних - зібрані в єдиній

робочій папці, доступ до якої здійснюється з бокової панелі. Також редактор дає змогу швидко знаходити потрібний фрагмент коду, шукати змінні чи функції, перемикатися між файлами й перевіряти зміни в реальному часі.

Загалом, VS Code у проєкті BookProgress забезпечував комфортне, швидке й організоване середовище для розробки, яке дозволило зосередитись на реалізації логіки та функціоналу, а не на боротьбі з технічними обмеженнями. Його зручність і потужність значно пришвидшили процес створення повноцінного веб-застосунку - від ідеї до завершеної системи.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ WEB-ЗАСТОСУНКУ ДЛЯ ВІДСТЕЖЕННЯ ПРОГРЕСУ ЧИТАННЯ КНИГ

3.1 Архітектурне та структурне моделювання системи

Для розробки якісного програмного забезпечення недостатньо просто реалізувати окремі функції, важливо продумати логіку роботи системи, структуру зберігання даних, взаємодію між її частинами та сценарії виконання ключових дій. Такі моделі дозволяють краще зрозуміти побудову проєкту, узгодити його логіку з функціональними вимогами, а також забезпечують основу для подальшої реалізації, тестування та супроводу.

Діаграма варіантів використання (прецедентів) у проєкті BookProgress, зображена на рисунку 3.1, використовується для візуального представлення функціональності системи з точки зору кінцевого користувача. Вона показує, які дії може виконувати користувач, у яких ситуаціях відбувається взаємодія із системою, а також як пов’язані між собою різні процеси. Така діаграма є важливою частиною аналізу вимог і проєктування, оскільки дозволяє зрозуміти зовнішню поведінку системи ще до початку розробки або написання коду.

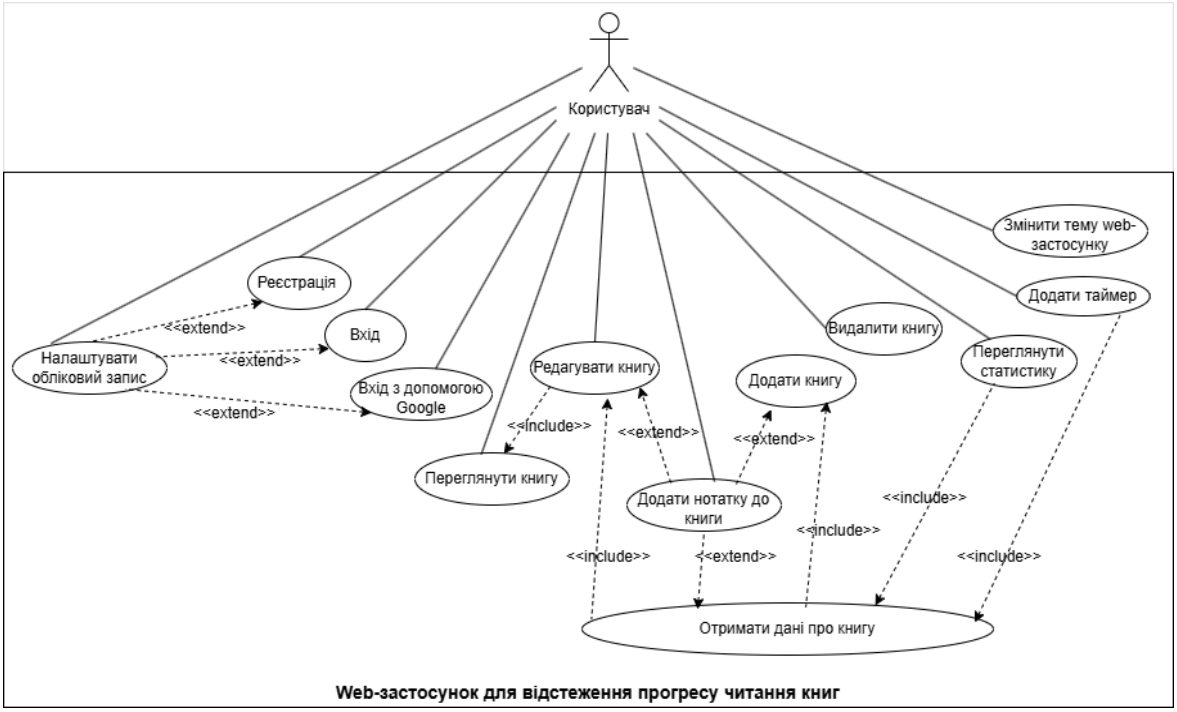


Рис. 3.1 Діаграма прецедентів

Діаграма класів в шаблоні MVC, зображена на рисунку 3.2 - архітектурна схема, яка демонструє поділ проєкту BookProgress на три взаємопов'язані компоненти: модель (дані та взаємодія з базою), представлення (інтерфейс, з яким працює користувач) та контролер (обробка логіки та запитів). Така діаграма дозволяє зрозуміти, які частини відповідають за що, як між собою взаємодіють контролери, моделі й відображення. Вона важлива для забезпечення чистої архітектури, спрощення масштабування проєкту та полегшення командної розробки. Такий підхід дозволяє чітко структурувати код та відокремити логіку від візуального представлення.

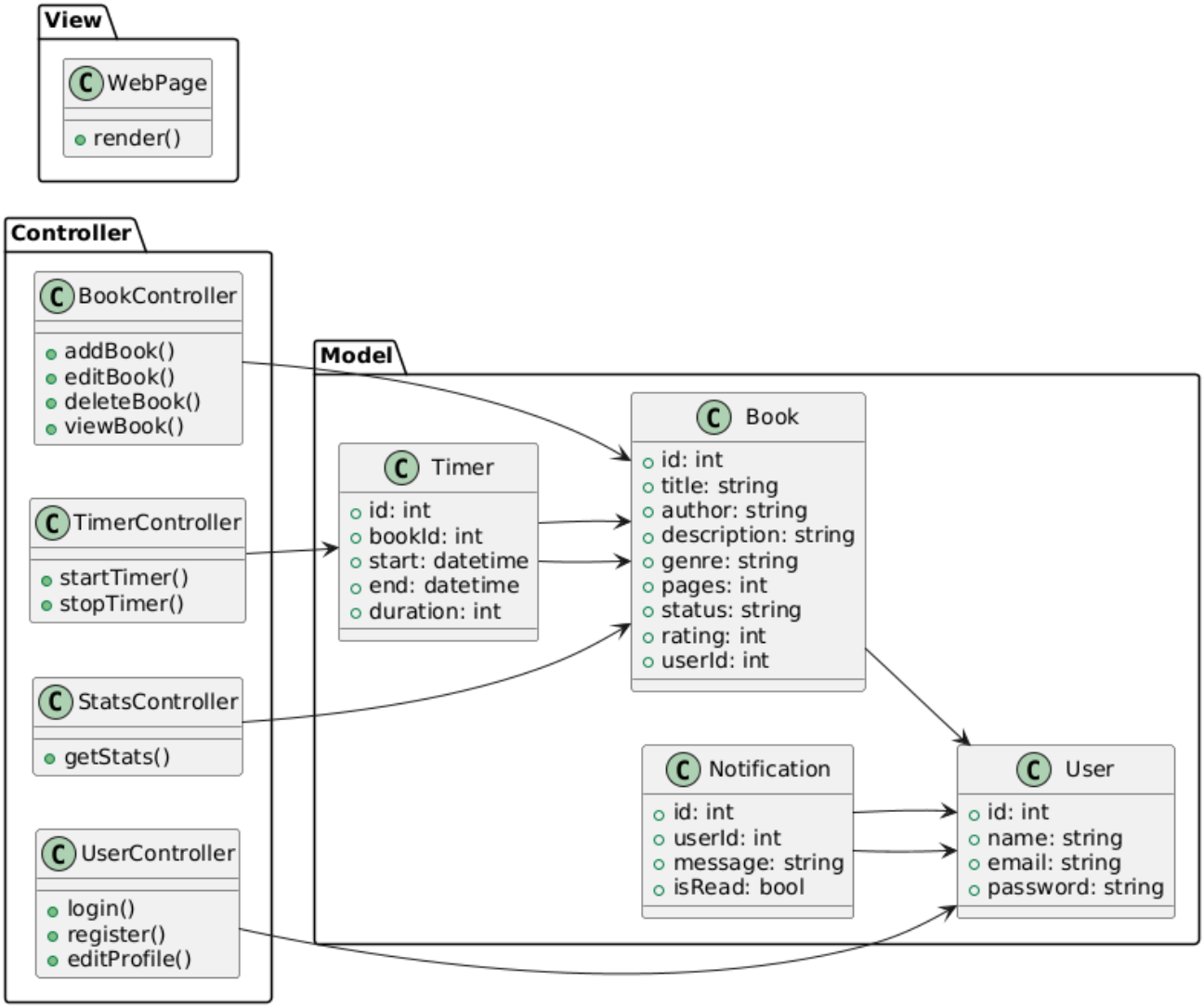


Рис 3.2 Діаграма класів у шаблоні MVC

Структура бази даних, зображена на рисунку 3.3 - це логічна схема, яка показує таблиці, поля та зв'язки між ними. Вона відображає таблиці бази даних SQLite, їхні поля та взаємозв'язки. Використовується для планування зберігання даних, забезпечення узгодженості інформації та оптимізації запитів. Така структура допомагає зрозуміти, як організовані сутності (наприклад, користувачі, товари, замовлення), які поля вони містять і як пов'язані між собою, що критично для побудови надійної системи збереження даних.

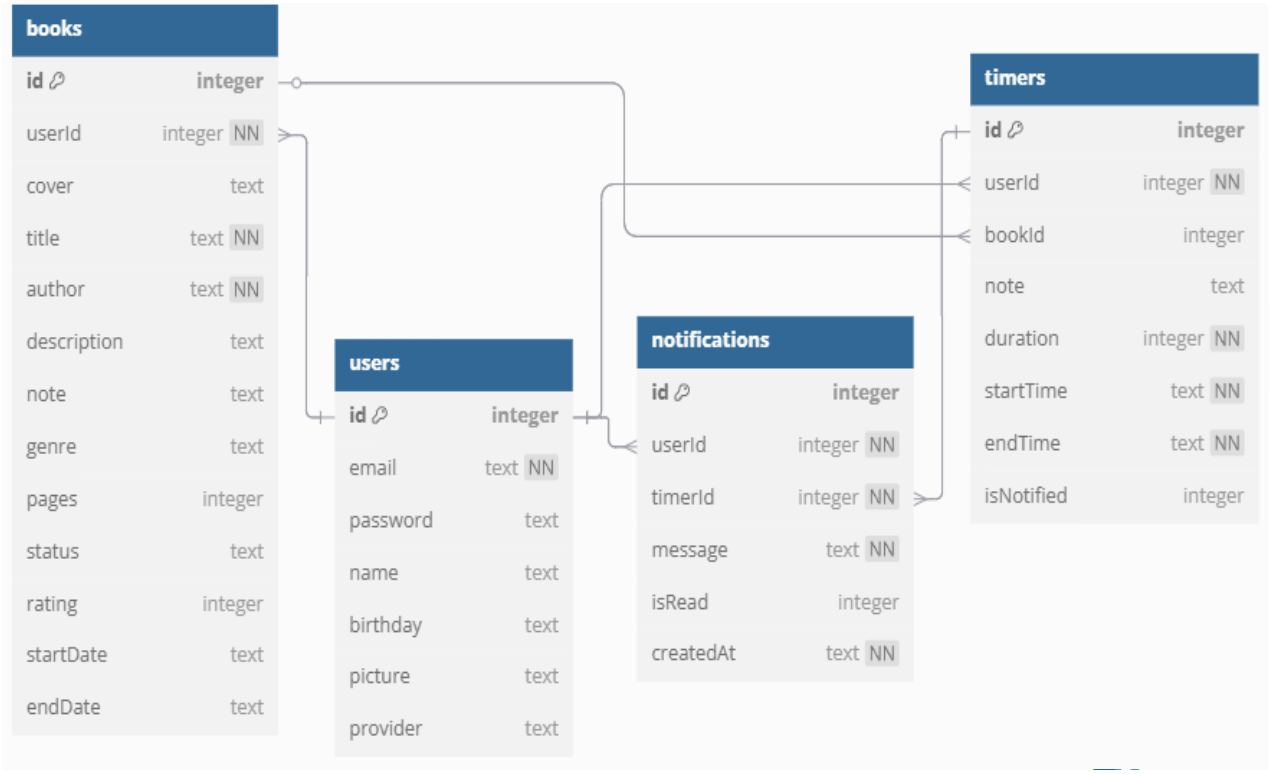


Рис. 3.3 Структура бази даних

Діаграма діяльності, зображена на рисунку 3.4 - моделює процес виконання певної дії або послідовність кроків, які відбуваються у системі чи під час взаємодії користувача з нею. Вона дозволяє візуалізувати логіку сценарію - від початку до завершення, з розгалуженнями, перевірками умов, циклами. Описана діаграма діяльності показує взаємодію користувача з таймером та його поведінку.

Ці моделі є важливими інструментами аналізу та проєктування програмного забезпечення, що забезпечують структурований підхід до розробки, зменшують кількість помилок і полегшують розуміння системи як окремим розробником, так і цілою командою.

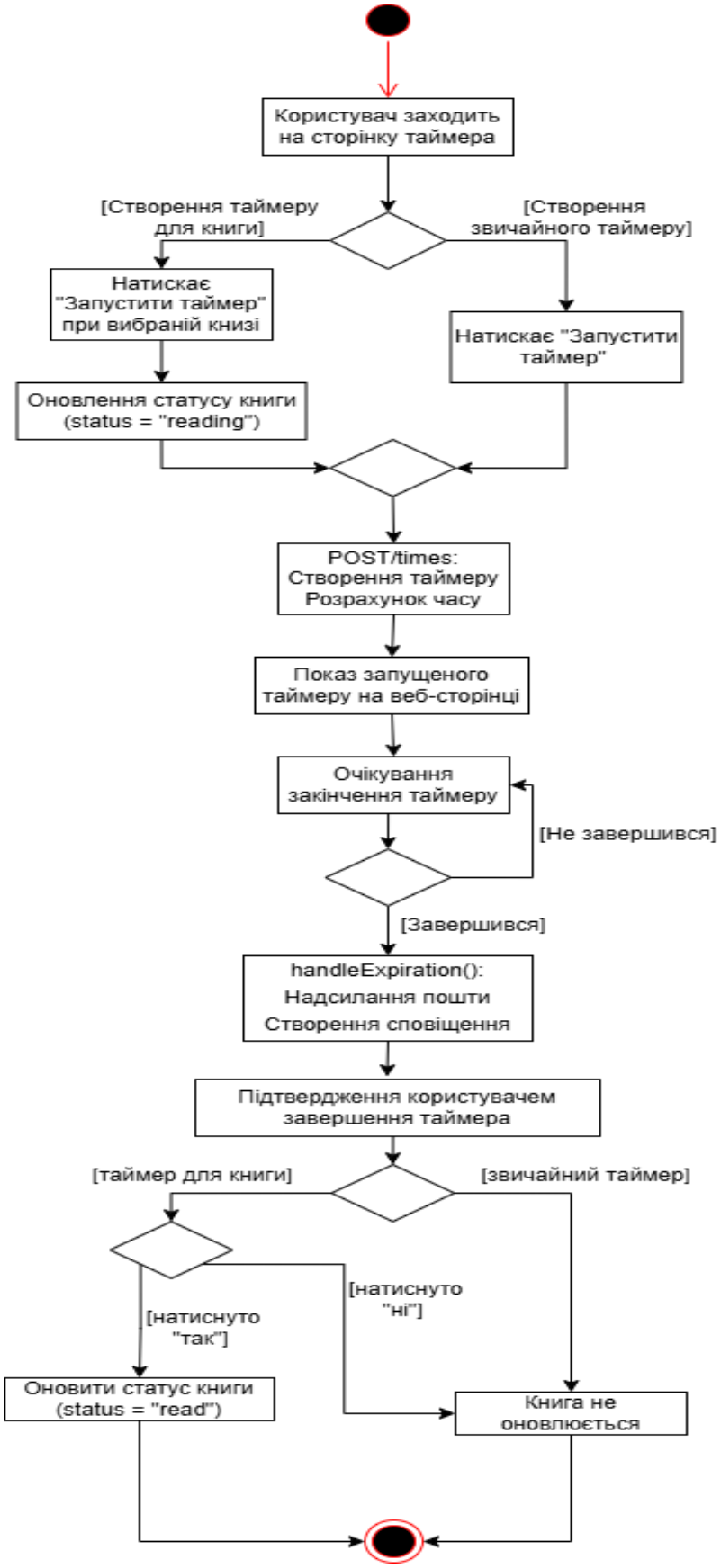


Рис 3.4 Діаграма діяльності таймера

3.2 Опис функціонування веб-застосунку

3.2.1 Головна сторінка

Після запуску web-застосунку BookProgress, користувач потрапляє на головну сторінку, де відображається одна з двох форм: реєстрація або вхід у систему. Сторінка містить поля для введення електронної пошти, пароля, імені та дати народження (під час реєстрації), а також кнопку для входу через Google. Авторизація через Google реалізована за допомогою OAuth 2.0 і дозволяє швидко отримати доступ без створення локального пароля.

Під формою входу розміщено надпис “Не можете увійти?”, його натискання надішле на електронну адресу посилання на сторінку відновлення пароля. Усі дані, введені користувачем, зберігаються у базі даних SQLite. Після успішної авторизації користувач потрапляє у сам web-застосунок на сторінку “Мої книги”, де може керувати своїми книгами, запускати таймер читання та переглядати статистику. Головну сторінку входу зображено на рисунку 3.5.

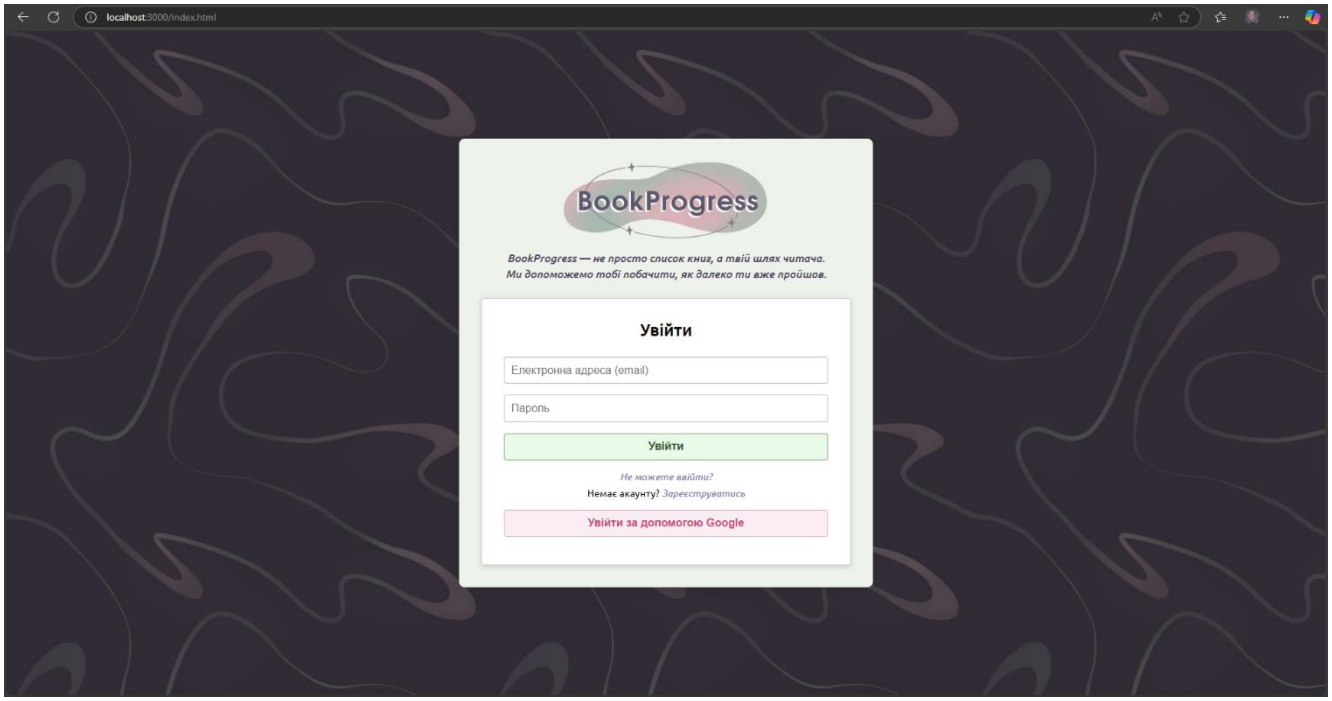


Рис. 3.5 Сторінка входу та реєстрації

3.2.2 Сторінка “Додати книгу”

Сторінка “Додати книгу”, зображена на рисунку 3.6, дає змогу користувачеві внести книгу до своєї бібліотеки. У центрі розміщено форму з полями для назви, автора, опису, нотатки, жанру, кількості сторінок, статусу читання та дат початку й завершення. Ліворуч можна завантажити обкладинку книги та поставити оцінку.

Після заповнення полів і натискання кнопки “Зберегти книгу” інформація надсилається на сервер і зберігається в базі даних. Сторінка дозволяє користувачу формувати свою читацьку бібліотеку та зберігати всі важливі деталі про кожну прочитану або заплановану книгу.

В лівій частині сторінки, як і на всіх подальших сторінках, знаходиться блок який показує ім’я, електронну адресу та аватар користувача, під ним знаходиться навігаційна панель із кнопками: “Додати книгу”, “Мої книги”, “Моя статистика”, “Таймер”, а також “Налаштування акаунту”. Доступність веб-сайтів є важливою для забезпечення доступу та ефективного використання інтернету всіма людьми, незалежно від їхніх фізичних чи когнітивних здібностей [14], тому сайт має блок перемикання теми інтерфейсу - світлу, темну або з високою контрастністю для зручності людей з вадами зору. У нижній частині бокового меню розташовано логотип застосунку.

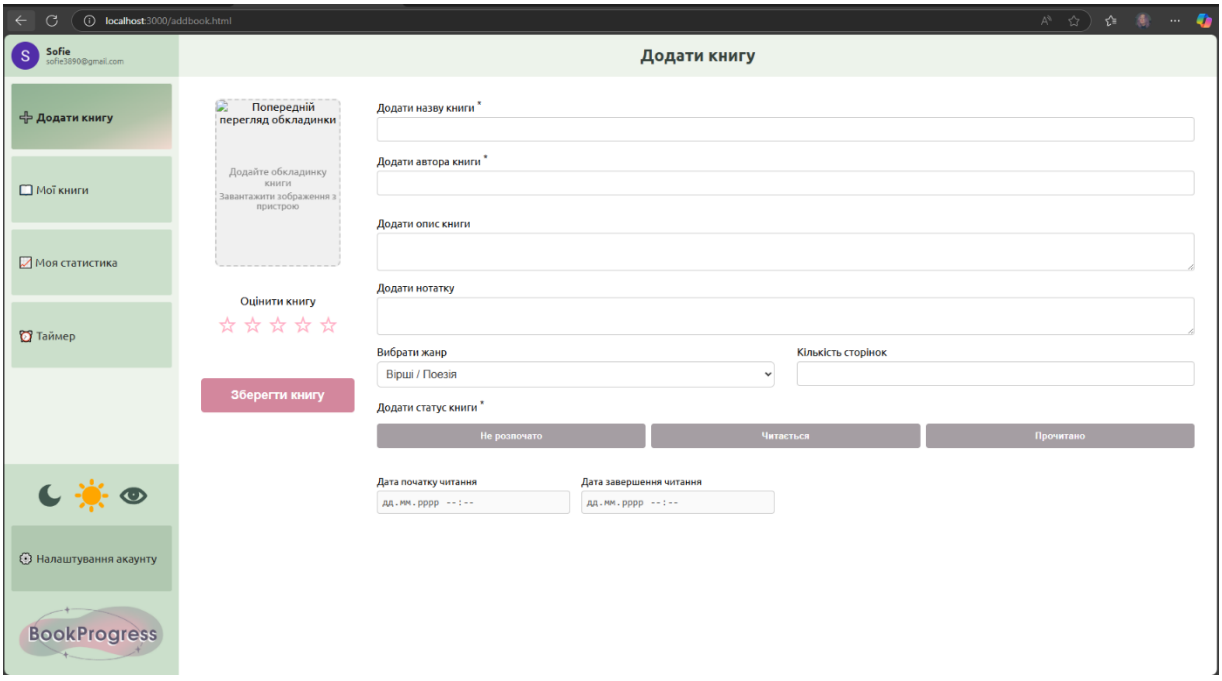


Рис 3.6 Сторінка “Додати книгу”

3.2.3 Сторінка “Мої книги”

Після входу до системи користувач потрапляє на сторінку “Мої книги”, яка є основним робочим простором у веб-застосунку BookProgress. На цій сторінці відображається список усіх книг, доданих користувачем до своєї особистої бібліотеки. У верхній частині сторінки розміщено заголовок “Мої книги” та поле пошуку, яке дозволяє швидко знаходити книги за назвою або автором.

У центральній частині відображається перелік книг - для кожної показано обкладинку, назву, автора, рейтинг, статус (наприклад, “Прочитано”) та коротку візуальну інформацію. Книги згруповані за алфавітом українського та латинського алфавітів, що спрощує навігацію при великій кількості записів. При натисканні на картку книги відкривається сторінка з детальнішою інформацією про неї.

Сторінку “Мої книги” зображено на рисунку 3.7.

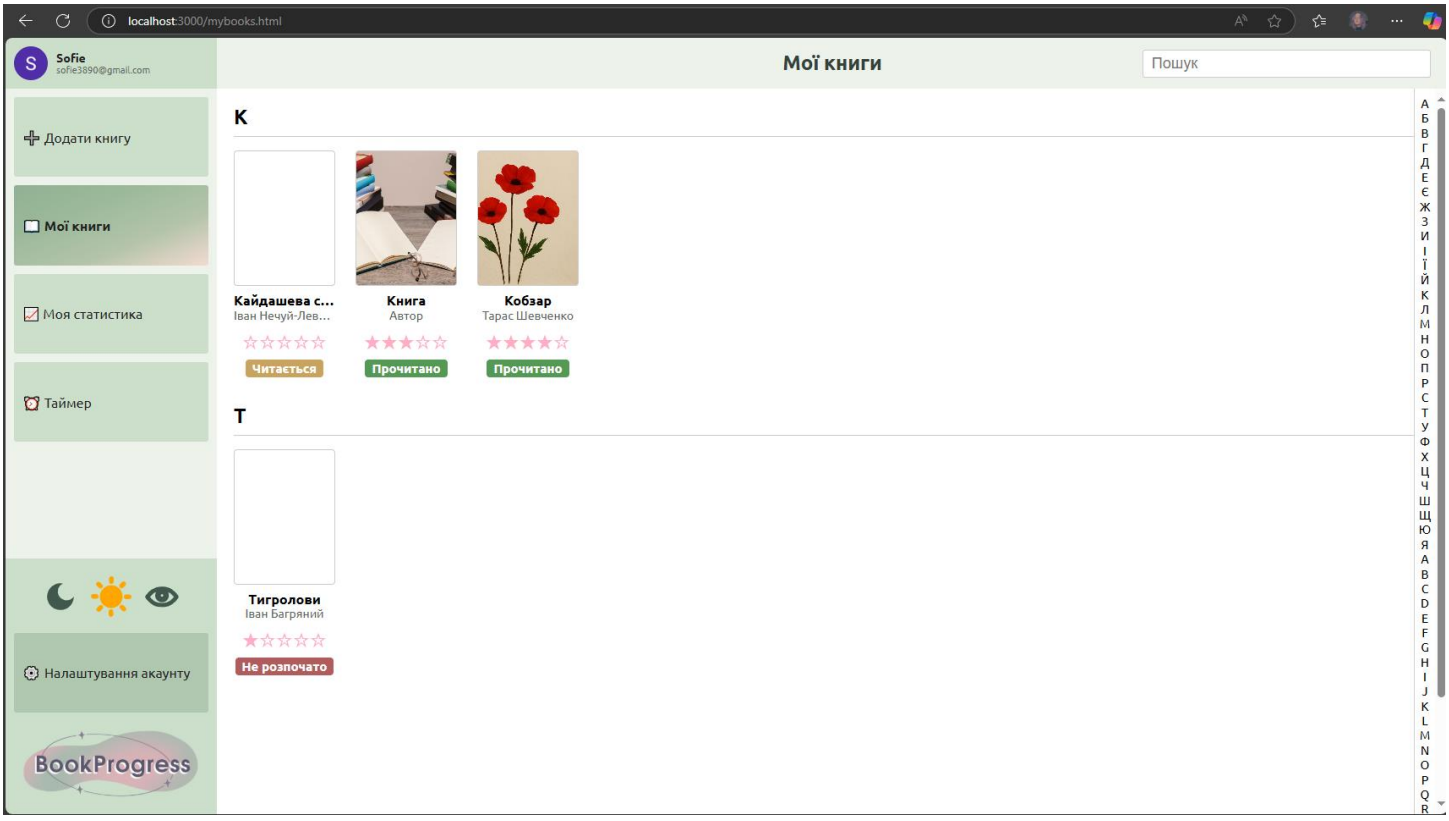


Рис 3.7 Сторінка “Мої книги”

3.2.4 Сторінка “Моя статистика”

Сторінка “Моя статистика” у веб-застосунку BookProgress надає користувачеві візуалізовану інформацію про його читацьку активність. У верхній частині розміщено два графіки: кругова діаграма показує розподіл книг за жанрами, а стовпчаста (або альтернативна) - кількість прочитаних сторінок за місяцями. Якщо сторінки не вказані під час додавання книг, графік відповідно порожній, про що відображається повідомлення.

У нижній частині сторінки наведено ключові числові показники: кількість книг у кожному зі статусів (“Прочитано”, “Читається”, “Не розпочато”), а також загальний час, витрачений на читання - у годинах і хвилинах. Ця статистика оновлюється автоматично на основі введених користувачем даних і зафіксованого таймера читання, що дозволяє слідкувати за своїм прогресом у зручній, наочній формі.

Сторінку “Моя статистика” зображено на рисунку 3.8.

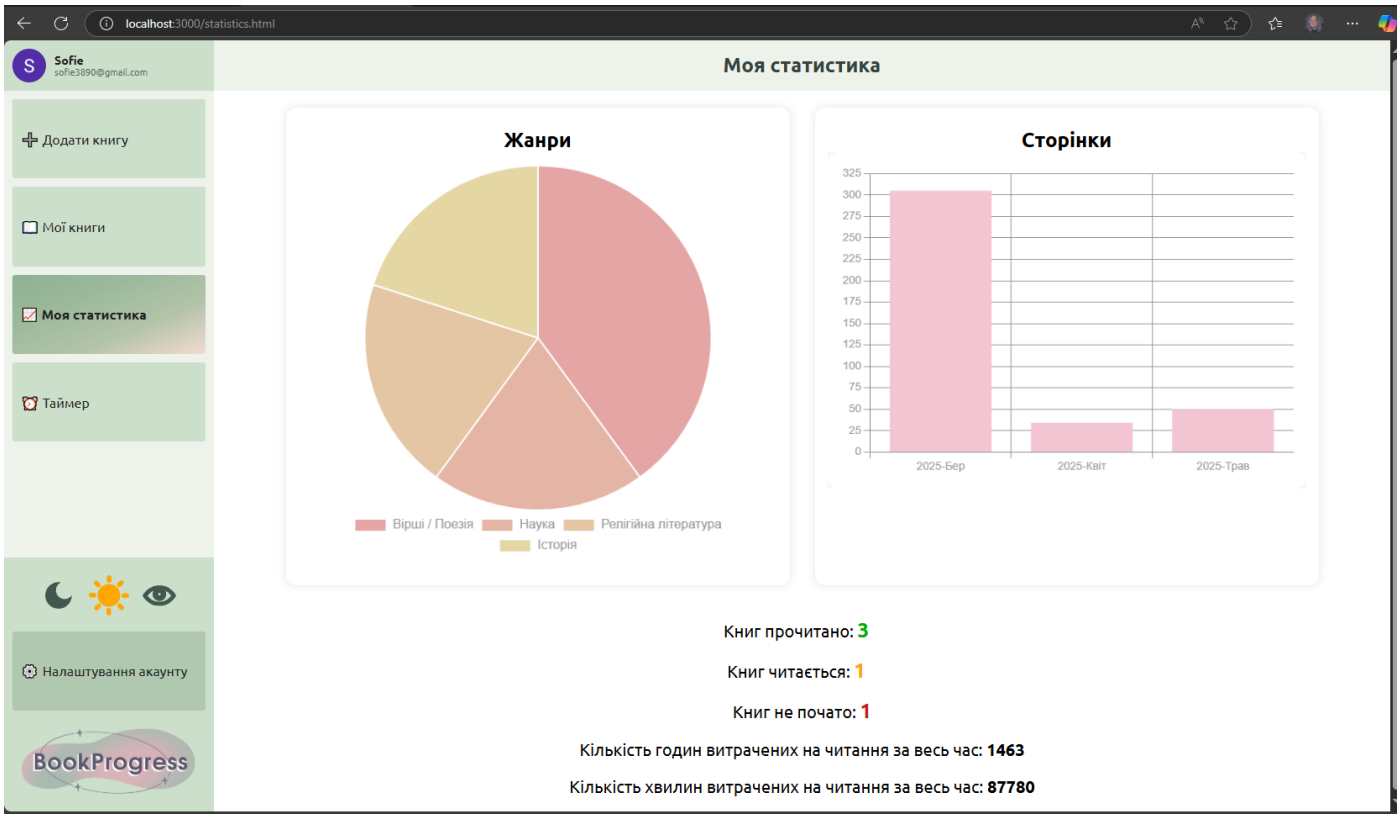


Рис 3.8. Сторінка “Моя статистика”

3.2.5 Сторінка “Таймер”

Сторінка “Таймер” дозволяє користувачеві фіксувати час, витрачений на читання. Тут можна вибрати книгу (опціонально), ввести нотатку та вказати тривалість таймера у хвилинах. Після натискання кнопки “Запустити таймер” він з’являється у списку активних і починає відлік у реальному часі.

Активні таймери відображаються нижче зі вказаною нотаткою, часом, що минув, і кнопкою для видалення. Є можливість очистити всі таймери однією кнопкою. Ця функція допомагає відстежувати реальний час, проведений за читанням, і враховується у загальній статистиці.

По закінченню часу на екрані з’являється форма з повідомленням про це, а також на електронну адресу приходить повідомлення про те що у таймера вийшов час.

Сторінку “Таймер” зображено на рисунку 3.9.

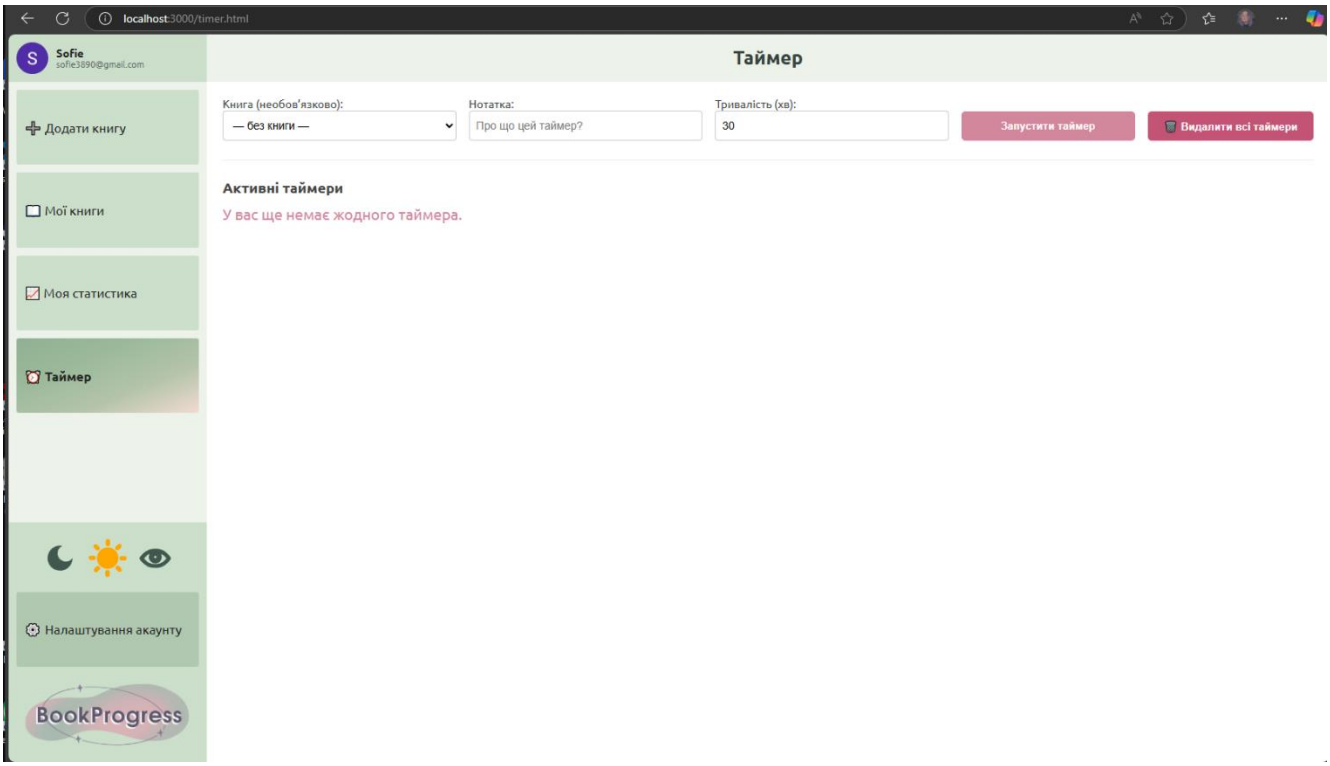


Рис 3.9. Сторінка “Таймер”

3.2.6 Сторінка “Налаштування акаунту”

Сторінка “Налаштування акаунту” дозволяє користувачеві переглянути та змінити особисті дані. У верхній частині розміщено великий аватар із кнопкою для його заміни. Нижче відображається ім’я, електронна пошта (незмінна) та дата народження, яку можна відредагувати.

Користувач може зберегти зміни особистих даних або натиснути “Змінити пароль”, щоб оновити доступ. У нижній частині сторінки також доступні дії для керування обліковим записом: видалити всі книги, вийти з акаунту або повністю видалити акаунт. Ця сторінка забезпечує повний контроль над персональними налаштуваннями користувача.

Сторінку “Налаштування акаунту” зображено на рисунку 3.10.

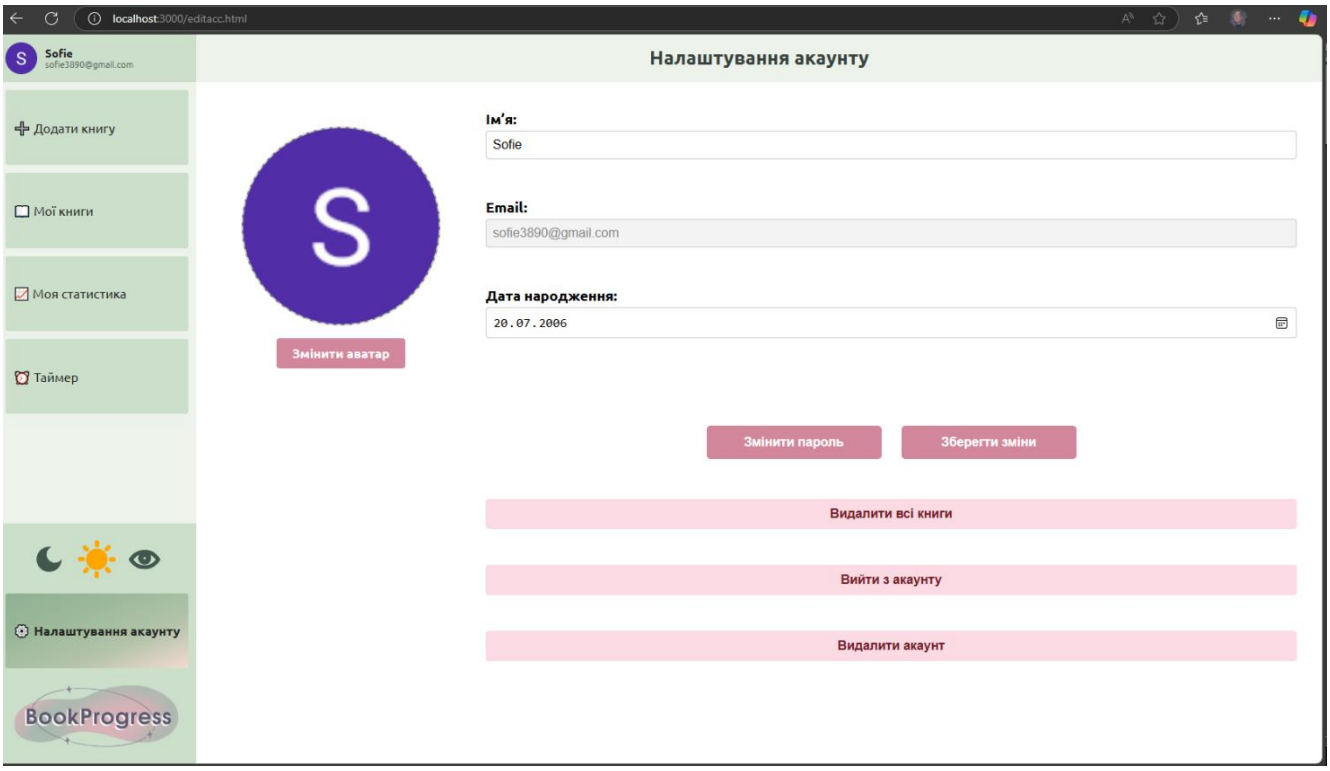


Рис 3.10 Сторінка “Налаштування акаунту”

4 ТЕСТУВАННЯ WEB-ЗАСТОСУНКУ ДЛЯ ВІДСТЕЖЕННЯ ПРОГРЕСУ ЧИТАННЯ КНИГ

4.1 Ручне тестування

У межах web-застосунку для відстеження прогресу читання “BookProgress” книг вручну було проведено мануальне(ручне) тестування. Воно передбачає покрокову перевірку всіх елементів сайту без використання автоматизованих засобів. У процесі тестування ретельно перевірялась робота основних функцій, таких як кнопки, поля вводу, форми, переходи між сторінками, логіка збереження даних, відображення повідомлень та навігація в межах інтерфейсу.

Були створені три тест-кейси для проведення ручного тестування:

1. Посилання для скидання паролю.
2. Відкриття сторінки перегляду книги.
3. Сповіщення про видалення книг.

4.1.1 Тест-кейс 1 : Посилання для скидання паролю

Таблиця 4.1 показує перевірку Тест-кейсу 1: Чи надсилається лист для скидання пароля після натискання на посилання “Не можете увійти?” на сторінці входу.

Таблиця 4.1

Тест-кейс 1

ID	1
Summary	Під посиланням “Не можете ввійти” має з’являтись сповіщення про надіслане на пошту посилання, користувачеві має приходити на пошту повідомлення з посиланням на сторінку скидання паролю http://localhost:3000/reset-password.html

Продовження таблиці 4.1

Тест-кейс 1

ID	1
Preconditions	Відкрита головна сторінка http://localhost:3000/index.html , Введена електронна адреса
Priority	Середній
Severity	Середній
Steps to reproduce	Натиснути на посилання “ Не можете ввійти ”
Actual result	З’являється надпис, посилання приходить на пошту
Expected results	З’являється надпис, посилання приходить на пошту

4.1.2 Тест-кейс 2 : Відкриття сторінки перегляду книги

Таблиця 4.2 показує перевірку Тест-кейсу 2: Чи відкривається сторінка перегляду книги після натискання на книгу в особистій бібліотеці.

Таблиця 4.2

Тест-кейс 2

ID	2
Summary	При натисканні на книгу на сторінці http://localhost:3000/mybooks.html, відкривається сторінка перегляду книги.
Preconditions	Користувач авторизувався на сайті Відкрита сторінка http://localhost:3000/mybooks.html,
Priority	Високий
Severity	Середній
Steps to reproduce	Натиснути книгу
Actual result	Відкривається сторінка перегляду книги.
Expected results	Відкривається сторінка перегляду книги.

4.1.3 Тест-кейс 3: Сповіщення про видалення книг

Таблиця 4.3 показує перевірку Тест-кейсу 3: Чи з'являється вікно підтвердження перед видаленням усіх книг при натисканні на відповідну кнопку у налаштуваннях акаунту.

Таблиця 4.3

Тест-кейс 3

ID	3
Summary	При натисканні на кнопку "Видалити всі книги" на сторінці http://localhost:3000/editacc.html з'являється alert-сповіщення з проханням підтвердити чи скасувати видалення книг.
Preconditions	Користувач авторизувався на сайті Відкрита сторінка http://localhost:3000/editacc.html,
Priority	Середній
Severity	Високий
Steps to reproduce	Натиснути на кнопку "Видалити всі книги"
Actual result	З'являється alert-сповіщення з проханням підтвердити чи скасувати видалення книг.
Expected results	З'являється alert-сповіщення з проханням підтвердити чи скасувати видалення книг.

4.2 Тестування бази даних

SQL-ін'єкція (SQL injection) - це тип атаки, при якому зловмисник вводить шкідливий SQL-код у поля введення (наприклад, email чи пароль) з метою змінити логіку запиту до бази даних. Через це можна отримати доступ до конфіденційної інформації, змінити або видалити записи, обійти авторизацію, а в деяких випадках - навіть повністю знищити базу. Такі атаки є однією з найпоширеніших загроз для веб-застосунків [15], особливо якщо введені дані безпосередньо вставляються в

SQL-запит без належної перевірки чи екранування. Саме тому база даних була написана з урахуванням цієї вразливості.

Тестування безпеки web-застосунку для відстеження прогресу читання книг проводиться за допомогою інструменту Postman, який дозволяє вручну надсилати HTTP-запити до серверу та змінювати дані у запиті. Для перевірки були обрані найбільш вразливі точки - поля введення електронної пошти та пароля при вході, реєстрації, а також параметри при додаванні книги.

Тестування наведено у рисунках 4.1-4.3.

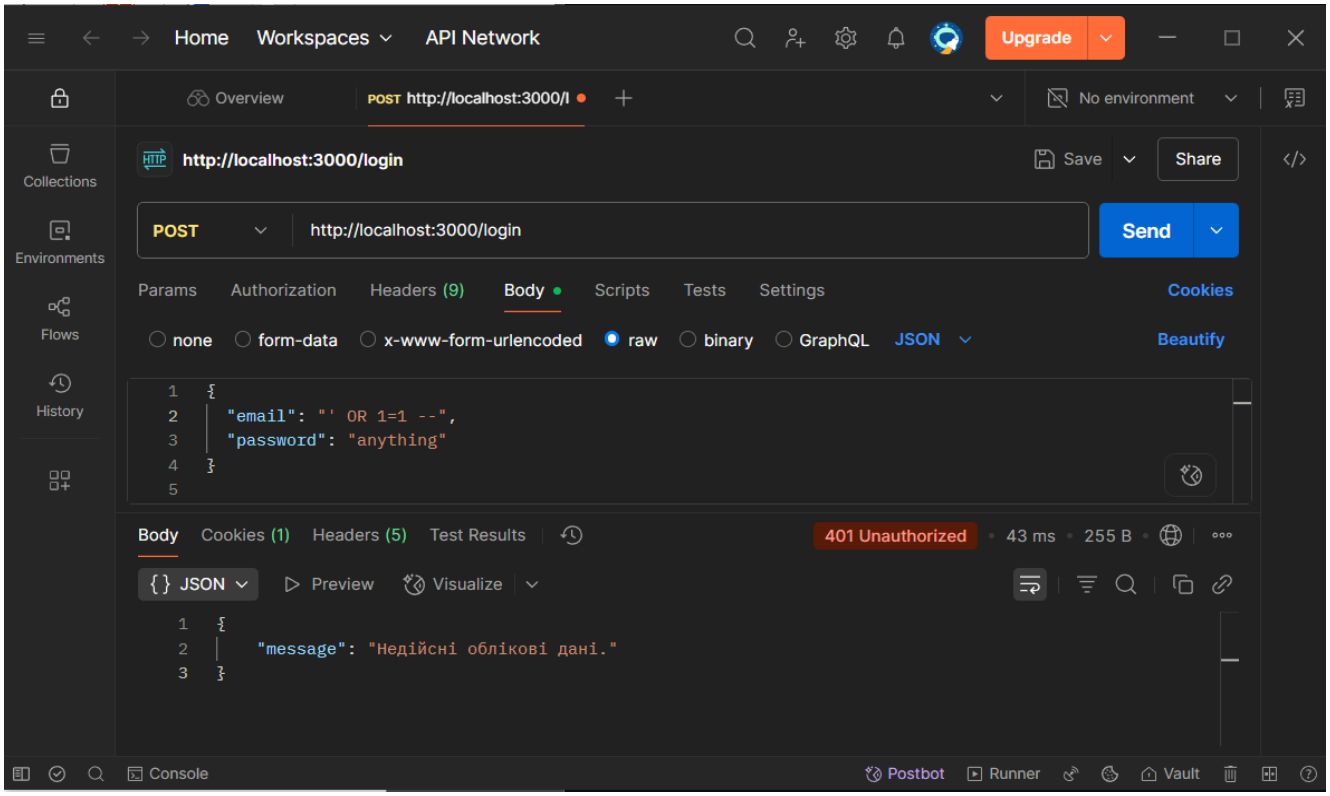


Рис. 4.1 Тестування вразливості бази даних до SQL-ін’єкцій

Ці запити надсилались у полі email або password у тілі POST-запиту на /login.

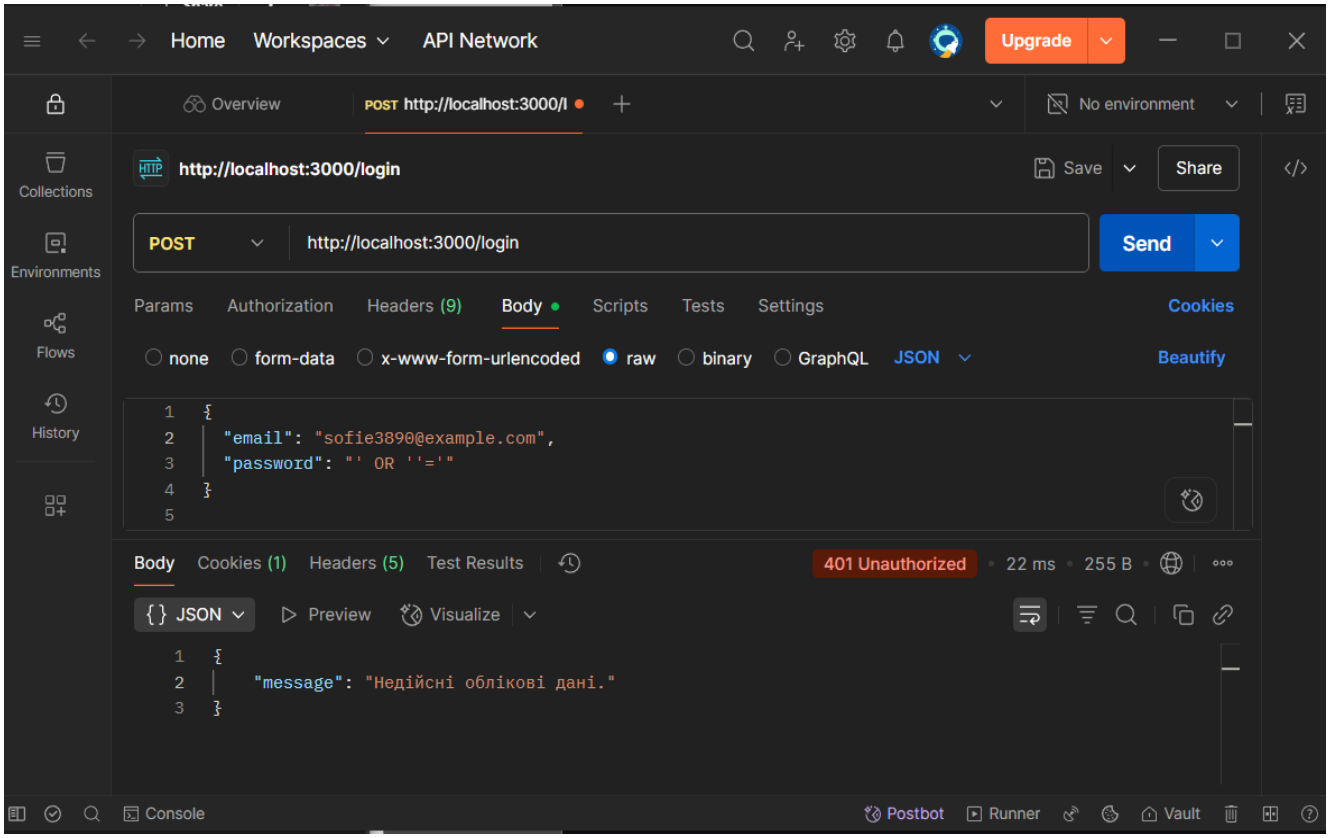


Рис. 4.2 Тестування вразливості бази даних до SQL-ін’єкції через пароль

Метою було перевірити, чи можна обійти обліковий запис або чи система реагує на некоректно сформовані SQL-запити.

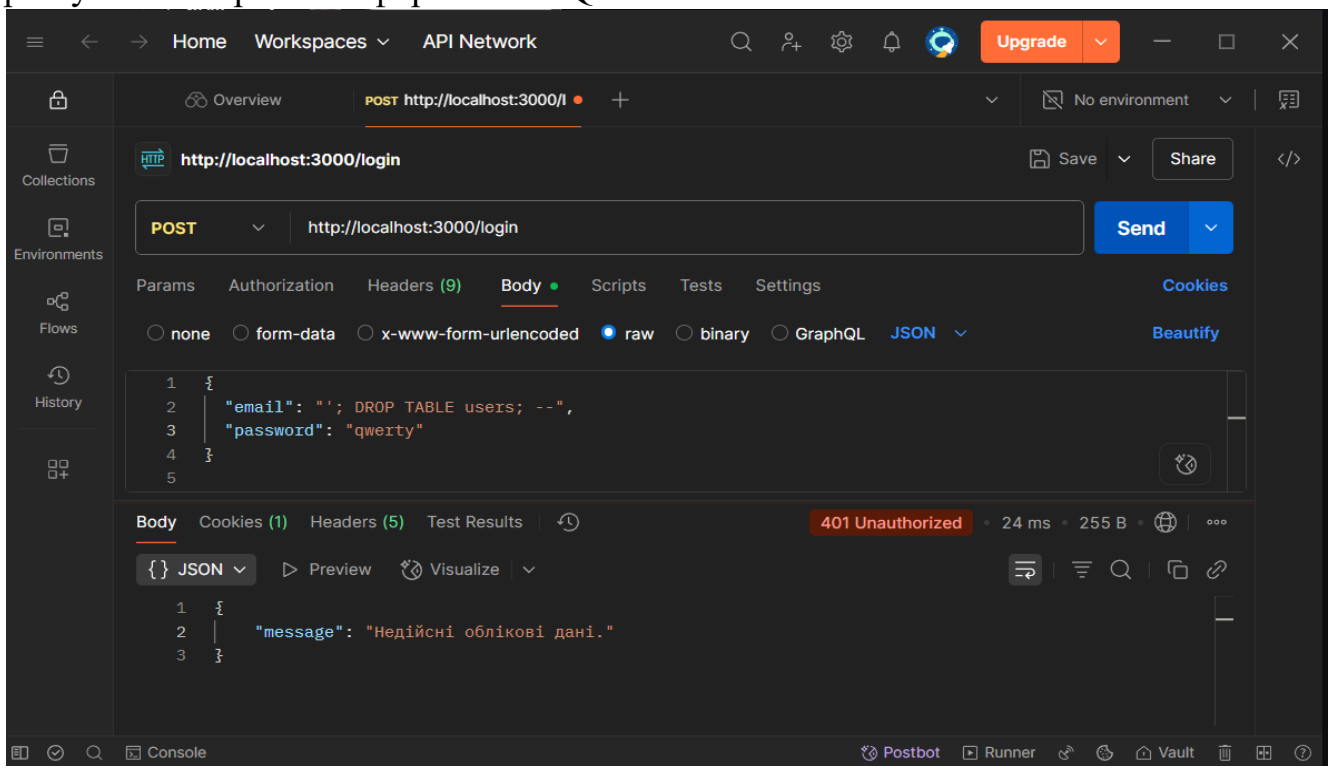


Рис. 4.3 Тестування вразливості бази даних до SQL-ін’єкцій

В усіх випадках сервер повертає повідомлення про помилку входу, не допускаючи виконання запиту.

У результаті тестування підтверджено, що застосунок не вразливий до SQL-ін’єкцій, оскільки всі введені дані проходять екранування або передаються через параметризовані запити, що є надійним методом захисту бази даних.

Таке тестування є критично важливим для захисту персональних даних користувачів і цілісності всієї системи. Його регулярне проведення дозволяє уникнути потенційних атак і підтримувати високий рівень безпеки веб-застосунку.

4.3 Тестування адаптивності

Тестування адаптивності web-застосунку для відстеження прогресу читання книг проводилось з метою перевірки коректності відображення вмісту сторінок на різних типах екранів. Для цього були використані вбудовані інструменти розробника браузера Google Chrome (Device Mode), які дозволяють емулювати популярні розміри екранів та пристрої.

Перевірка охоплює наступні типи пристроїв:

- Десктоп (ширина 1024 пікселів і більше), зображений на рисунку 4.4.;
- Планшет (ширина ~768 пікселів), зображений на рисунку 4.5.;
- Смартфон (ширина 320-480 пікселів), зображений на рисунку 4.6.;

На кожному з типів пристроїв перевіряються:

- Коректність відображення основного вмісту (текстів, зображень, таблиць, кнопок);
- Доступність та читабельність елементів інтерфейсу;
- Робота навігації (меню, посилання, кнопки);
- Поведінка сторінок при зміні орієнтації (портретна / ландшафтна);
- Відсутність горизонтального скролінгу, якщо він не передбачений дизайном.

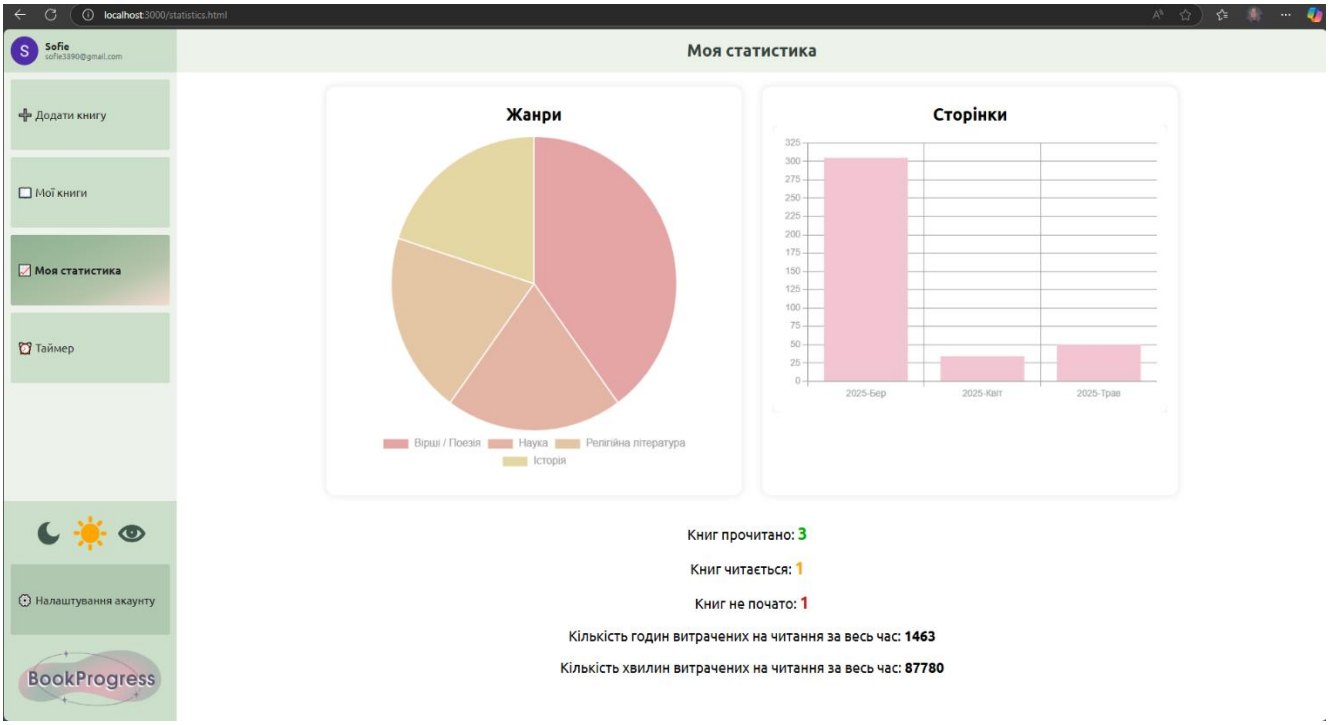


Рис. 4.4. Сторінка статистики в режимі екрану десктоп

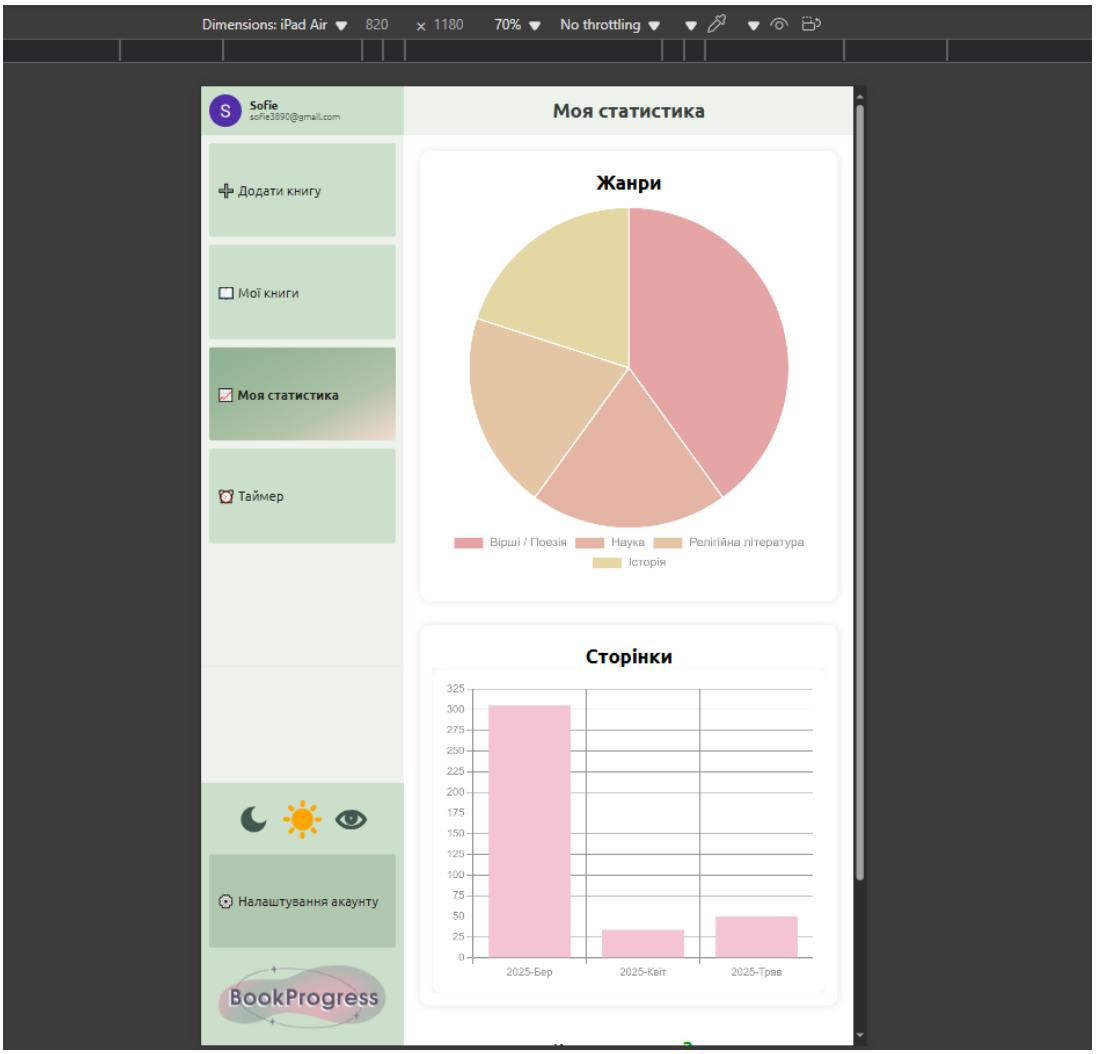


Рис. 4.5. Тестування сторінки в режимі екрану планшету, емульовано iPad Air

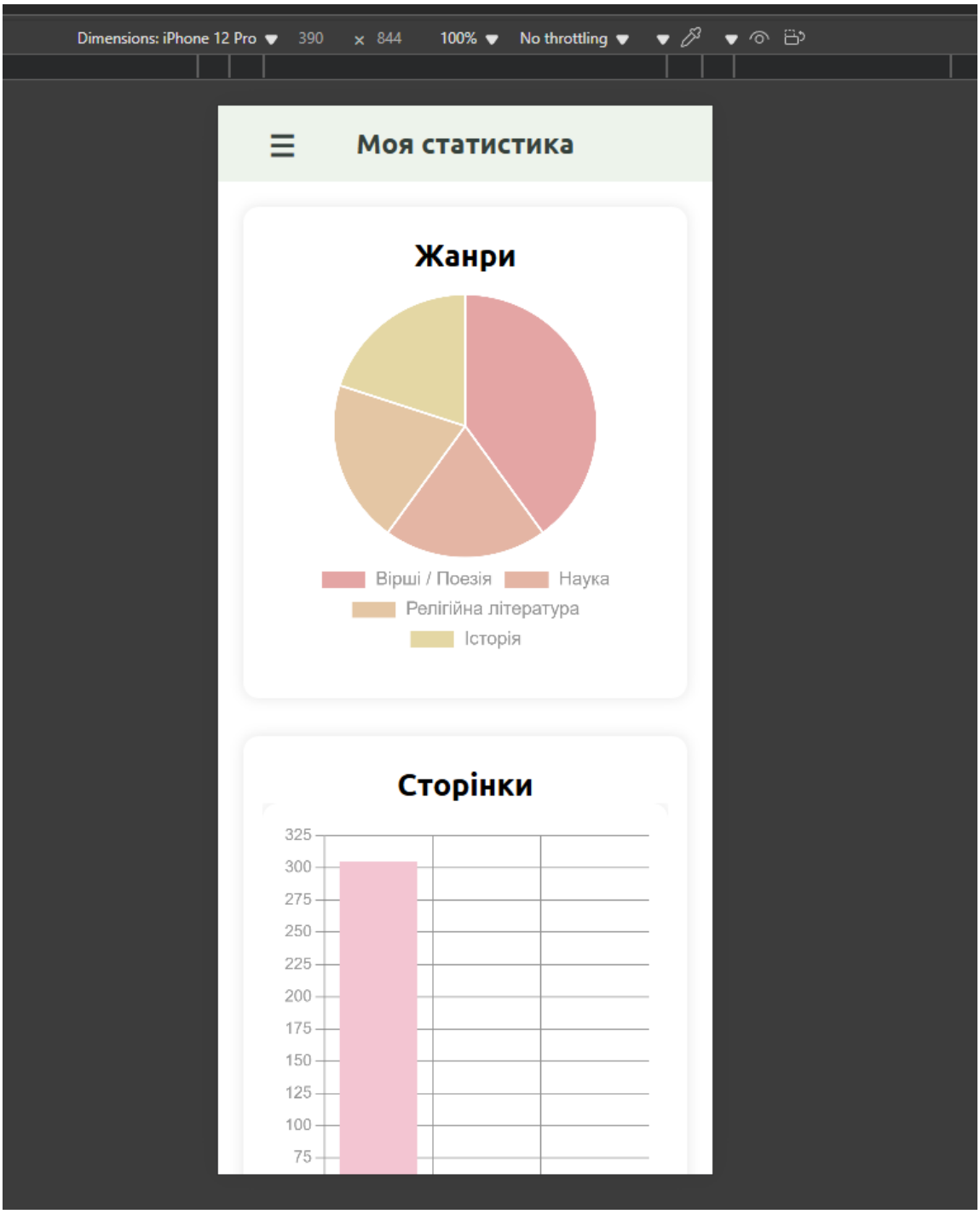


Рис. 4.6. Тестування сторінки в режимі екрану телефону, емульовано iPhone 12 Pro

У результаті тестування встановлено, що всі сторінки web-застосунку адаптуються до ширини екрана. Елементи інтерфейсу автоматично перебудовуються, не порушуючи структури та забезпечуючи зручний доступ до функціоналу з мобільних пристроїв.

ВИСНОВКИ

У процесі виконання бакалаврської кваліфікаційної роботи було проведено аналіз предметної галузі, яка охоплює цифрові інструменти для запису прогресу читання , і було вивчено сучасні методи впровадження таких систем. Було досліджено методи запам’ятовування, ручного та цифрового відстеження прогресу читання, а також розглянуто можливості їх автоматизації засобами web-технологій.

На основі аналізу аналогів сформульовано функціональні та нефункціональні вимоги до застосунку. Було розроблено архітектуру web-застосунку за шаблоном MVC, а також створено діаграму варіантів використання, діаграму діяльності, структуру бази даних і схему взаємодії компонентів системи.

Таким чином, реалізовано web-застосунок для відстеження прогресу читання книг, що дозволяє користувачам додавати книги, залишати нотатки, вказувати статуси читання, переглядати графічну статистику й керувати власним акаунтом. Застосунок включає систему авторизації через Google (OAuth 2.0), таймер для фіксації часу читання, світлу/темну/контрастну тему, а також адаптований інтерфейс для різних пристроїв.

Було проведено тестування функціональності веб-застосунку, що включає ручне тестування, перевірку захисту від SQL-ін’єкцій, адаптивність інтерфейсу, а також виявлення та усунення помилок, щоб забезпечити стабільну роботу продукту.

Отже, виконано всі завдання та задачі, визначені в межах бакалаврської роботи. Розроблений web-застосунок для відстеження прогресу читання книг є функціональним, безпечним та придатним для реального використання широким колом користувачів, які бажають вести електронний облік прочитаних книг та мотивувати себе до систематичного читання.

ПЕРЕЛІК ПОСИЛАНЬ

1. Chetail F. Reading books: The positive impact of print exposure on written word recognition. *Cognition*. 2024. Vol. 251. P. 105905. URL: https://doi.org/10.1016/j.cognition.2024.105905 (date of access: 27.05.2025).

2. Saputra W., Cahyani I., Sastromiharjo A. Trends and Innovation in Digital Reading Assessment: A Decade of Bibliometric Insights. *Journal of Languages and Language Teaching*. 2025. Vol. 13, no. 1. P. 46. URL: https://doi.org/10.33394/jollt.v13i1.12182 (date of access: 27.05.2025).

3. Spjeldnæs K., Karlsen F. How digital devices transform literary reading: The impact of e-books, audiobooks and online life on reading habits. *New Media & Society*. 2022. P. 146144482211261. URL: https://doi.org/10.1177/14614448221126168 (date of access: 27.05.2025).

4. Chan C. K. Can reading increase cognitive reserve?. *International Psychogeriatrics*. 2021. Vol. 33, no. 1. P. 15–17. URL: https://doi.org/10.1017/s1041610220001246 (date of access: 27.05.2025).

5. Faul L., Kensinger E. A. Emotional memory. *Reference Module in Neuroscience and Biobehavioral Psychology*. 2024. URL: https://doi.org/10.1016/b978-0-443-15754-7.00011-0 (date of access: 28.05.2025).

6. Jansen R. S., Lakens D., IJsselsteijn W. A. An integrative review of the cognitive costs and benefits of note-taking. *Educational Research Review*. 2017. Vol. 22. P. 223–233. URL: https://doi.org/10.1016/j.edurev.2017.10.001 (date of access: 28.05.2025).

7. Liu J., Wang Y., Chang L. What is effective digital reading? A systematic review of the effectiveness factors of digital reading. *The Electronic Library*. 2024. URL: https://doi.org/10.1108/el-01-2024-0002 (date of access: 28.05.2025).

8. Italic Type [Електронний ресурс]. – Режим доступу: https://www.italictype.com/

9. LibraryThing [Електронний ресурс]. – Режим доступу:

https://www.librarything.com/

10. Libib [Електронний ресурс]. – Режим доступу: https://www.libib.com/

11. Alfonso Aguilar J., Garrigós I., Mazón J.-N. Requirements Engineering in the Development Process of Web Systems: A Systematic Literature Review. *Acta Polytechnica Hungarica*. 2016. Vol. 13, no. 3. P. 61–80. URL: https://doi.org/10.12700/aph.13.3.2016.3.4 (date of access: 28.05.2025).

12. OAuth 2.0 / S. S. M. Rahman et al. *Advances in Wireless Technologies and Telecommunication*. 2020. P. 92–139. URL: https://doi.org/10.4018/978-1-7998-3355-0.ch005 (date of access: 28.05.2025).

13. Ostretsova T., Pereiaslavska S., Ostretsov D. COMPARATIVE ANALYSIS OF THE PERFORMANCE OF NODE AND BUN PLATFORMS IN EDUCATIONAL WEB APPLICATIONS. *OPEN EDUCATIONAL E-ENVIRONMENT OF MODERN UNIVERSITY*. 2025. No. 18. P. 119–131. URL: https://doi.org/10.28925/2414-0325.2025.1810 (date of access: 28.05.2025).

14. Makati T., Tigwell G. W., Shinohara K. The Promise and Pitfalls of Web Accessibility Overlays for Blind and Low Vision Users. *ASSETS '24: The 26th International ACM SIGACCESS Conference on Computers and Accessibility*, St. John's NL Canada. New York, NY, USA, 2024. P. 1–12. URL: https://doi.org/10.1145/3663548.3675650 (date of access: 28.05.2025).

15. SQL Injection Prevention in Web Application: A Review / J. H. B. Johny et al. *Communications in Computer and Information Science*. Singapore, 2021. P. 568–585. URL: https://doi.org/10.1007/978-981-16-8059-5_35 (date of access: 28.05.2025).

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка web-застосунку для відстеження прогресу читання книг

Виконала студентка 4 курсу
Групи ПД-43
Пономаренко Софія Михайлівна
Керівник роботи
доктор філософії (PhD), старший викладач кафедри ІПЗ Залива Віталій Вікторович

Київ – 2025

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: спрощення відстеження прогресу читання книг за допомогою web-застосунку з впровадженими засобами інтерактивного моніторингу.

Об'єкт дослідження: відстеження прогресу читання книг.

Предмет дослідження: засоби розробки web-застосунку для відстеження прогресу читання книг.

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Провести огляд та аналіз існуючих способів відстеження прогресу читання книг.
2. Сформувати функціональні та нефункціональні вимоги до web-застосунку для відстеження прогресу читання книг.
3. Спроєктувати архітектурну модель, базу даних і користувацький інтерфейс web-застосунку для відстеження прогресу читання книг.
4. Реалізувати web-застосунок для відстеження прогресу читання книг з базою даних, серверною логікою та користувацьким інтерфейсом.
5. Здійснити тестування функціональності роботи web-застосунку застосунку для відстеження прогресу читання книг.

3

АНАЛІЗ АНАЛОГІВ

Показник	Italic Type	LibraryThing	Libib	BookProgress
Збереження інформації введеної користувачем про книги до бібліотеки	-	+	+	+
Графічне відображення статистики читання	-	+	+	+
Адаптивність web-застосунку до різних пристроїв	+	+	+	+
Таймер для читання	-	-	-	+
Налаштування облікового запису	+	+	+	+
Авторизація за допомогою Google	-	-	-	+

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги:

1. Реєстрація та авторизація користувачів.
2. Збереження інформації введеної користувачем про книги до бібліотеки.
3. Можливість пошуку книг за назвою або автором.
4. Оцінювання книг.
5. Графічне відображення статистики читання.
6. Таймер для читання.
7. Налаштування облікового запису.

Нефункціональні вимоги:

1. Адаптивність web-застосунку до різних пристроїв.
2. Захист даних від SQL-ін'єкцій.
3. Можливість зміни теми інтерфейсу (темна/світла).
4. Підтримка монохромної теми інтерфейсу для людей з вадами зору.
5. Авторизація за допомогою Google.

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

JS

CSS

HTML

Тестування

Oauth 2.0


Bun.js
Backend,
запуск сервера

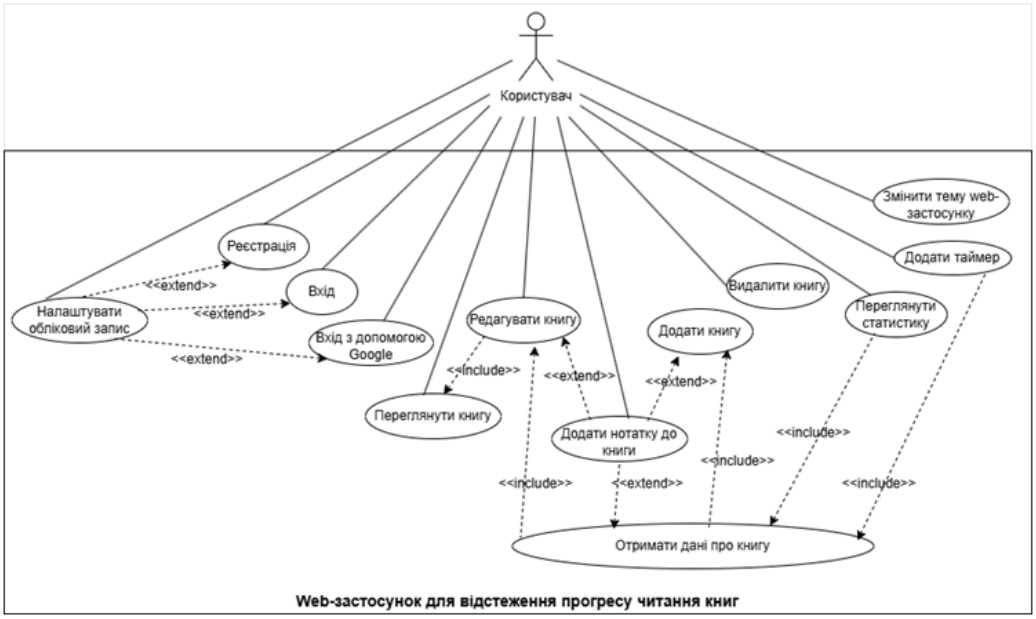
Зереження
конфіденційних
даних


VS Code


Figma

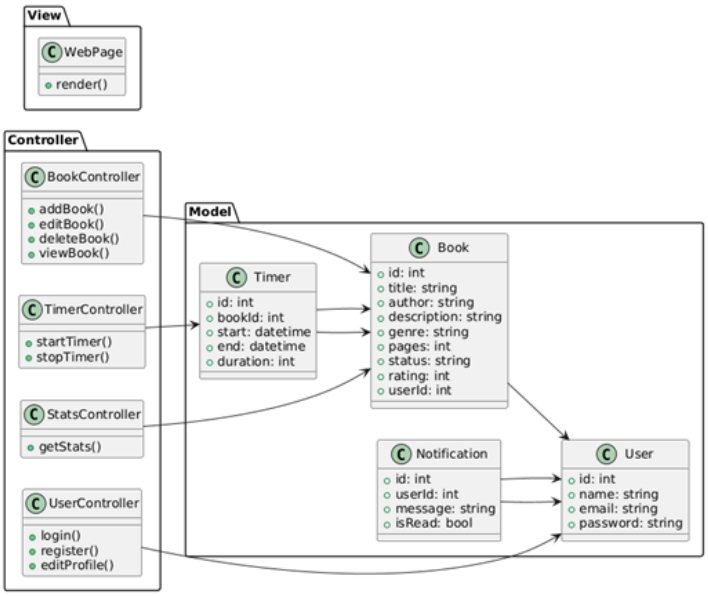
6

ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ



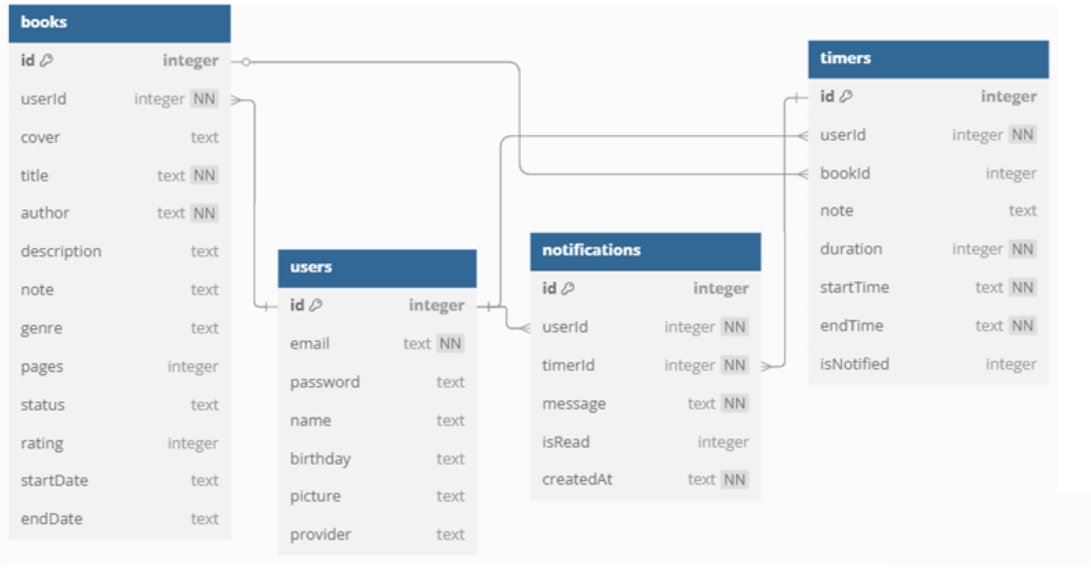
7

ДІАГРАМА КЛАСІВ MVC



8

СТРУКТУРА БАЗИ ДАНИХ

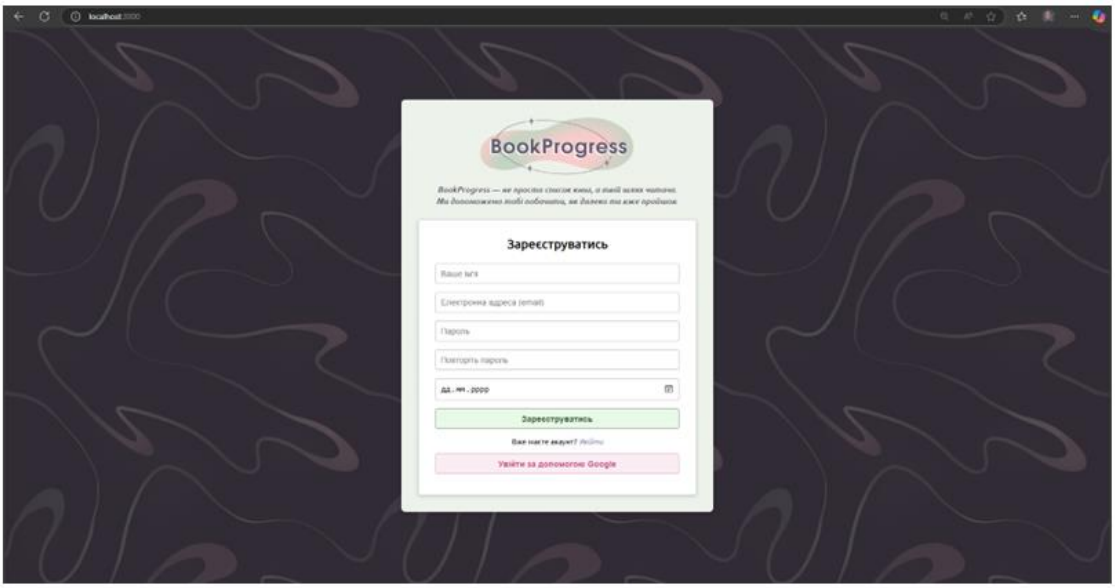


9

ДІАГРАМА ДІЯЛЬНОСТІ ТАЙМЕРА

10

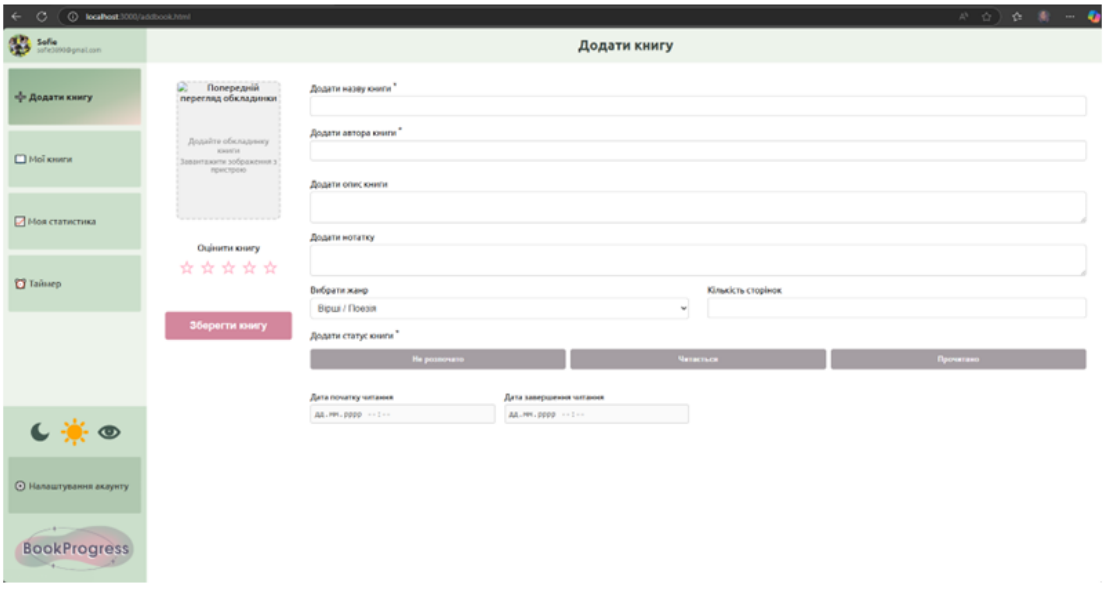
ЕКРАННІ ФОРМИ



Сторінка входу та реєстрації

11

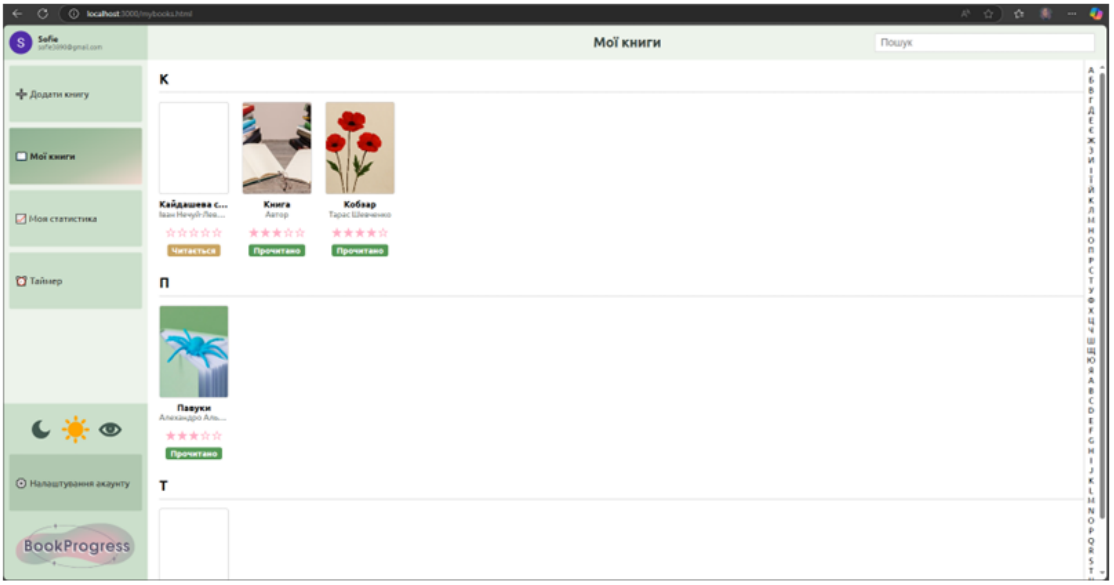
ЕКРАННІ ФОРМИ



Сторінка додавання книг

12

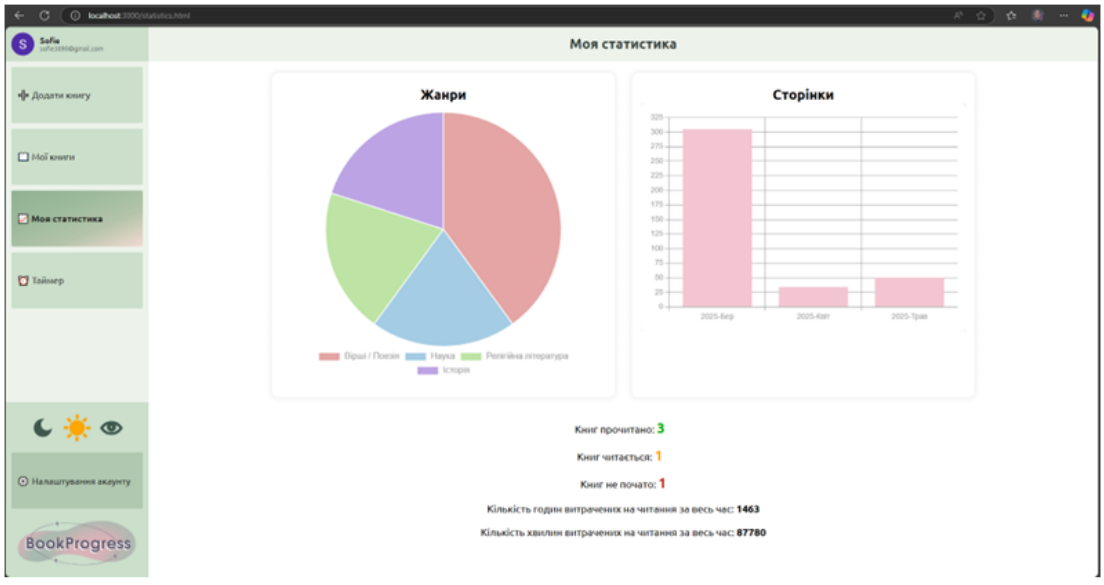
ЕКРАННІ ФОРМИ



Сторінка з усіма книгами

13

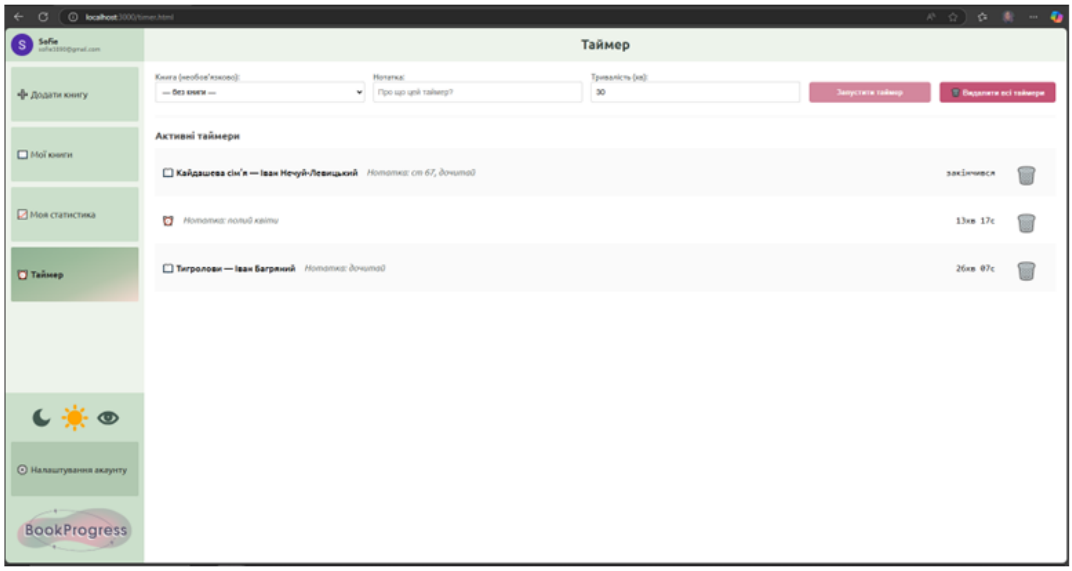
ЕКРАННІ ФОРМИ



Сторінка статистики

14

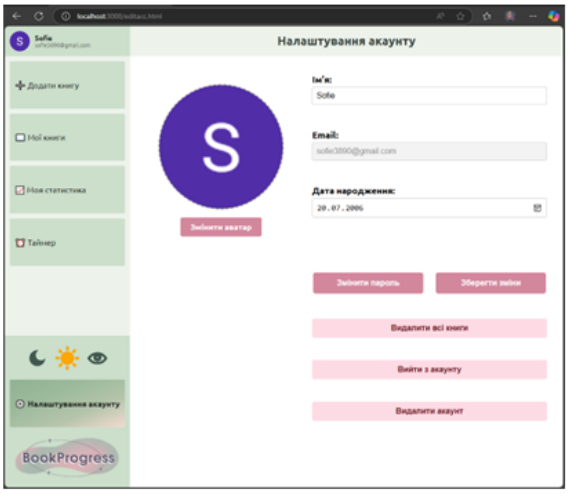
ЕКРАННІ ФОРМИ



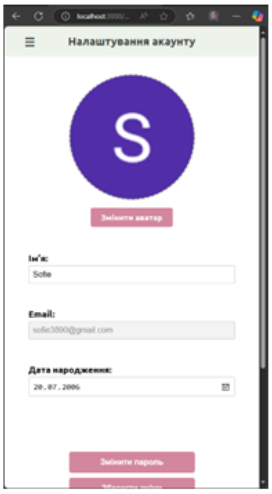
Сторінка додавання таймерів

15

ЕКРАННІ ФОРМИ



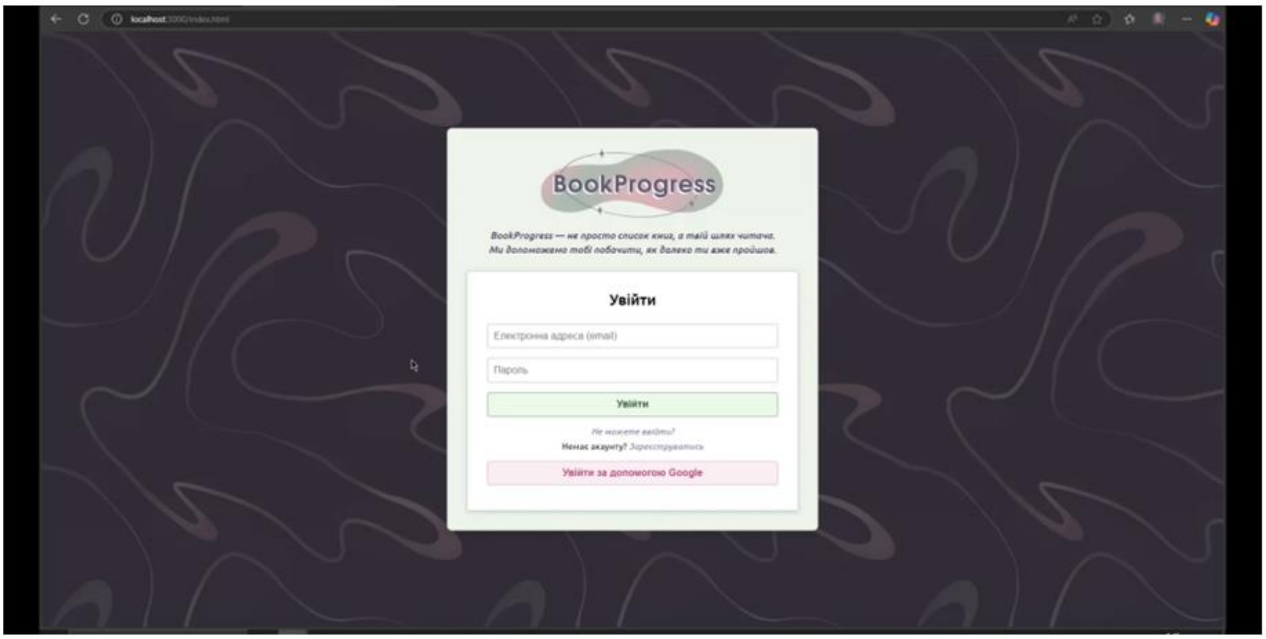
Сторінка для перегляду та редагування акаунту у веб-режимі



Сторінка для перегляду та редагування акаунту у режимі екрану мобільного телефону.

16

ЕКРАННІ ФОРМИ



17

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Пономаренко С.М., Залива В.В. Важливість розробки веб-застосунку для відстеження прогресу читання книг. Матеріали Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». Збірник тез. 24 квітня 2025 року, ДУІКТ, м. Київ, с 120-121.

2. Пономаренко С.М. Використання параметризованих запитів для захисту баз даних SQL-ін'єкцій. Матеріали XII Всеукраїнської науково-практичної конференції молодих учених «Інформаційні технології – 2025», 15.05.25, КСУ імені Бориса Грінченка, Київ, Україна, с 314-315.

18

ВИСНОВКИ

1. Проведено огляд та аналіз існуючих способів відстеження прогресу читання книг: запам’ятовування, ручне фіксування, цифрове фіксування .
2. Сформовано функціональні та нефункціональні вимоги до web-застосунку для відстеження прогресу читання книг, з урахуванням потреб користувачів та технічних обмежень.
3. Спроектовано архітектурну модель, базу даних і користувацький інтерфейс web-застосунку для відстеження прогресу читання книг.
4. Реалізовано web-застосунок для відстеження прогресу читання книг з базою даних, серверною логікою та користувацьким інтерфейсом на основі сучасних web-технологій.
5. Здійснено тестування функціональності роботи web-застосунку для відстеження прогресу читання книг: ручне тестування, тестування бази даних, тестування адаптивності.

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

```
import express from "express";
import { OAuth2Client } from "google-auth-library";
import { Database } from "bun:sqlite";
import bcrypt from "bcryptjs";
import cookieParser from "cookie-parser";
import dotenv from "dotenv";
dotenv.config();
import { unlink } from "fs/promises";
import nodemailer from "nodemailer";
import crypto from "crypto";
const app = express();
const db = new Database("users.db");
app.use(express.static("public"));
app.use(express.json({ limit: '5mb' }));
app.use(cookieParser());
function checkNameServer(name) {
  if (typeof name !== 'string' || !name.trim()) {
    return 'Ім’я має бути непустим рядком.';
  }
  if (name.length > 80) {
    return 'Ім’я має бути не довше 80 символів.';
  }
  if (!/[A-Za-z\u0400-\u04FF]/.test(name)) {
    return 'Ім’я має містити хоча б одну літеру.';
  }
  if (/^[^A-Za-z\u0400-\u04FF]+$/.test(name)) {
    return 'Ім’я не може складатися лише з цифр або лише зі
знаків.';
  }
  return null;
}

function checkPasswordServer(pw, oldPw = null) {
  if (typeof pw !== 'string' || pw.length < 8)
    return 'Пароль має бути щонайменше 8 символів.';
  if (!/[a-za-яёїієґ]/u.test(pw))
    return 'Пароль має містити малу літеру.';
  if (!/[A-ZA-ЯЁЇІЄҐ]/u.test(pw))
    return 'Пароль має містити велику літеру.';
  if (!/\d/.test(pw))
    return 'Пароль має містити цифру.';
  if (!/[\W_]/.test(pw))
    return 'Пароль має містити спеціальний символ.';
  if (oldPw !== null && pw === oldPw)
    return 'Новий пароль не може співпадати зі старим.';
  return null;
}
export const transporter = nodemailer.createTransport({
  host: "smtp.gmail.com",
  port: 587,
  secure: false,
  auth: {
    user: process.env.SMTP_USER,
    pass: process.env.SMTP_PASS
  },
  tls: { rejectUnauthorized: false }
});

transporter.verify()
  .then(() => console.log("✓ SMTP ready"))
  .catch(err
const googleClient = new
OAuth2Client(GOOGLE_CLIENT_ID);
app.get("/auth/google", (req, res) => {
  const url = googleClient.generateAuthUrl({
    access_type: "offline",
    scope: ["email", "profile"],
    redirect_uri: GOOGLE_REDIRECT_URI
  });
  res.redirect(url);
});

// Google OAuth логін
app.post("/auth/google/login", async (req, res) => {
  const { code } = req.body;
  if (!code) return res.status(401).json({ message:
"Неавторизовано" });

  try {
    const tokenRes = await
fetch("https://oauth2.googleapis.com/token", {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({
        code,
        client_id: GOOGLE_CLIENT_ID,
        client_secret: GOOGLE_CLIENT_SECRET,
        redirect_uri: GOOGLE_REDIRECT_URI,
        grant_type: "authorization_code",
      }),
    }).then(r => r.json());

    if (!tokenRes.id_token) {
      console.error("No id_token:", tokenRes);
      return res.status(500).json({ message: "Не отримали ID
токен" });
    }

    const ticket = await googleClient.verifyIdToken({
      idToken: tokenRes.id_token,
      audience: GOOGLE_CLIENT_ID,
    });
    const { email, name, picture } = ticket.getPayload();

    const existingUser = db
      .query(`SELECT * FROM users WHERE email = ?`)
      .get(email);

    if (!existingUser) {
      const randomPass =
crypto.randomBytes(16).toString("hex");
      const hashed = await bcrypt.hash(randomPass, 10);

      db.query(`
```

```
      INSERT INTO users (email, password, name, picture,
provider)
       VALUES (?, ?, ?, ?, 'google')
     `).run(email, hashed, name, picture);

   } else if (!existingUser.password) {
     const randomPass =
crypto.randomBytes(16).toString("hex");
     const hashed = await bcrypt.hash(randomPass, 10);

     db.query(`
       UPDATE users
       SET password = ?
       WHERE email = ?
     `).run(hashed, email);
   }

   res.cookie("token", email, generateCookieOptions());
   res.json({ message: "Вхід завдяки Google успішний" });

  } catch (err) {
   console.error("OAuth Error:", err);
   res.status(500).json({ message: "Помилка Google OAuth"
});
  }
});
app.post("/register", async (req, res) => {
  const { email, password, name, birthday } = req.body;

  if (!email || !password || !name) {
    return res.status(400).json({ message: "Відсутні обов'язкові
поля." });
  }
  let err = checkNameServer(name);
  if (err) return res.status(400).json({ message: err });
  err = checkPasswordServer(password);
  if (err) return res.status(400).json({ message: err });

  const exists = db.query(`SELECT 1 FROM users WHERE
email = ?`).get(email);
  if (exists) {
    return res.status(409).json({ message: "Електронна пошта
вже зареєстрована." });
  }

  const hashed = await bcrypt.hash(password, 10);
  try {
    db.query(`
     INSERT INTO users (email, password, name, birthday,
provider, emailVerified)
     VALUES (?, ?, ?, ?, ?, 0)
   `).run(email, hashed, name, birthday ? birthday : null,
'local');

  } catch (dbErr) {
    console.error("DB error on register:", dbErr);
    return res.status(500).json({ message: "Помилка бази
даних." });
  }

  const token = crypto.randomBytes(32).toString("hex");
  verificationTokens.set(token, email);
  setTimeout(() => verificationTokens.delete(token), 24 * 60 *
60 * 1000);

  const verifyUrl = `http://localhost:3000/verify-
email?token=${token}`;
  try {
    const info = await transporter.sendMail({
      from: `"BookProgress" <${process.env.SMTP_USER}>`,
      to: email,
      subject: "Підтвердження email",
      text: `Щоб підтвердити пошту, перейдіть за
посиланням:\n\n${verifyUrl}`
    });
    console.log("Confirmation email sent:", info.messageId);
  } catch (mailErr) {
    console.error("Error sending confirmation email:", mailErr);
    return res
      .status(500)
      .json({ message: "Не вдалося відправити листа
підтвердження." });
  }

  res.json({ message: "Реєстрація успішна! Лист
підтвердження надіслано." });
});

app.get("/verify-email", (req, res) => {
  const { token } = req.query;
  const email = verificationTokens.get(token);
  if (!email) {
    return res
      .status(400)
      .send("<h1>Невірний або прострочений токен</h1>");
  }
  db.query(`UPDATE users SET emailVerified = 1 WHERE
email = ?`).run(email);
  verificationTokens.delete(token);
  res.send("<h1>Email підтверджено! Тепер можете
увійти.</h1>");
});

app.post("/login", async (req, res) => {
  console.log("BODY:", req.body);
  const { email, password } = req.body;

  const user = db.query(`SELECT * FROM users WHERE
email = ?`).get(email);

  if (!user) {
    return res.status(401).json({ message: "Недійсні облікові
дані." });
  }

  const validPassword = await bcrypt.compare(password,
user.password);
  if (!validPassword) {
    return res.status(401).json({ message: "Недійсні облікові
дані." });
  }

  res.cookie("token", email, generateCookieOptions());
  res.json({ message: "Вхід успішний!" });
```

```
});
app.post("/logout", (req, res) => {
  res.clearCookie("token");
  res.json({ message: "Ви вийшли з системи" });
});
app.post("/books", isAuthenticated, (req, res) => {
  const {
    cover, title, author,
    description, note,
    genre, pages,
    status, rating,
    startDate, endDate
  } = req.body;

  if (!title || !author || !status) {
    return res.status(400).json({ message: "Необхідні поля
(назва, автор, статус) не заповнені." });
  }

  const exists = db.query(`
    SELECT id FROM books
    WHERE userId = ? AND LOWER(title) = LOWER(?) AND
LOWER(author) = LOWER(?)
  `).get(req.user.id, title.trim(), author.trim());

  if (exists) {
    return res.status(400).json({ message: "Така книга вже
існує." });
  }

  const genreStr = Array.isArray(genre) ?
JSON.stringify(genre) : (genre || null);

  const result = db.query(`
    INSERT INTO books
      (userId, cover, title, author, description, note, genre, pages,
status, rating, startDate, endDate)
    VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
  `).run(
    req.user.id,
    cover || null,
    title, author,
    description || null,
    note || null,
    genreStr,
    pages || null,
    status,
    rating || 0,
    startDate || null,
    endDate || null
  );

  res.json({ message: "Книгу додано", bookId:
result.lastInsertRowid });
});

app.get("/books", isAuthenticated, (req, res) => {
  const rows = db.query(`
    SELECT * FROM books WHERE userId = ?
  `).all(req.user.id);

  const books = rows.map(b => ({
    ...b,
    genre: b.genre ? JSON.parse(b.genre) : []
  }));
  res.json(books);

});
app.get("/books/:id", isAuthenticated, (req, res) => {
  const id = Number(req.params.id);
  const row = db.query(`SELECT * FROM books WHERE id =
? AND userId = ?`)
    .get(id, req.user.id);
  if (!row) return res.status(404).json({ message: "Не
знайдено" });
  row.genre = row.genre ? JSON.parse(row.genre) : [];
  res.json(row);
});
app.put("/books/:id", isAuthenticated, async (req, res) => {
  const id = Number(req.params.id);
  const { cover, title, author, description, note, genre, pages,
status, rating, startDate, endDate } = req.body;
  const genreStr = Array.isArray(genre) ?
JSON.stringify(genre) : (genre ? JSON.stringify([genre]) :
null);

  const exists = db.query(`
    SELECT id FROM books
    WHERE userId = ? AND LOWER(title) = LOWER(?) AND
LOWER(author) = LOWER(?)
  `).get(req.user.id, title.trim(), author.trim());

  if (exists && exists.id !== id) {
    return res.status(400).json({ message: "Інша книга з такою
ж назвою й автором вже існує." });
  }

  await db.query(`
    UPDATE books
    SET cover=?, title=?, author=?, description=?, note=?,
      genre=?, pages=?, status=?, rating=?, startDate=?,
endDate=?
    WHERE id=? AND userId=?
  `).run(
    cover, title, author, description, note,
    genreStr, pages, status, rating, startDate, endDate,
    id, req.user.id
  );

  res.json({ message: "Оновлено" });
});
app.delete('/books/:id', isAuthenticated, (req, res) => {
  const bookId = Number(req.params.id);
  const userId = req.user.id;

  try {
    const result = db.run(
      `DELETE FROM books WHERE id = ? AND userId = ?`,
      [bookId, userId]
    );

    if (result.changes === 0) {
      return res.status(404).json({ message: "Книга не
знайдена" });
```

```
    }

    res.sendStatus(204);
  } catch (err) {
    console.error('DB DELETE error:', err);
    res.status(500).json({ message: "Не вдалося видалити
книгу" });
  }
});
app.get('/check-book', isAuthenticated, (req, res) => {
  res.setHeader("Cache-Control", "no-store");

  const { title, author, id } = req.query;
  const userId = req.user.id;

  if (!title || !author) {
    return res.status(400).json({ exists: false });
  }

  const trimmedTitle = title.trim().toLowerCase();
  const trimmedAuthor = author.trim().toLowerCase();

  const rows = db.prepare(`
    SELECT id FROM books
    WHERE userId = ? AND LOWER(title) = ? AND
LOWER(author) = ?
  `).all(userId, trimmedTitle, trimmedAuthor);

  const isDuplicate = rows.length > 0 && (!id || rows.some(row
=> row.id.toString() !== id.toString()));

  res.json({ exists: isDuplicate });
});
app.post("/timers", isAuthenticated, (req, res) => {
  const { bookId, note, duration } = req.body;
  const startTime = new Date().toISOString();
  const endTime = new Date(Date.now() + duration *
60000).toISOString();
  const result = db.query(`
    INSERT INTO timers (userId, bookId, note, duration,
startTime, endTime)
    VALUES (?, ?, ?, ?, ?, ?)
  `).run(req.user.id, bookId || null, note || null, duration,
startTime, endTime);
  scheduleTimer(result.lastInsertRowid);
  res.json({ message: "Таймер запущено", timerId:
result.lastInsertRowid });
});

app.get("/timers", isAuthenticated, (req, res) => {
  const rows = db.query("SELECT * FROM timers WHERE
userId = ?").all(req.user.id);
  res.json(rows);
});

app.post("/timers/:id/read", isAuthenticated, (req, res) => {
  const timerId = Number(req.params.id);
  const { confirm } = req.body;
  db.query("UPDATE timers SET isNotified = 1 WHERE id =
?").run(timerId);
  const timer = db.query("SELECT * FROM timers WHERE id
= ?").get(timerId);
  if (!timer) return res.status(404).json({ message: "Таймер не
знайдено" });

  if (timer.bookId && confirm) {
    db.query(`
      UPDATE books
      SET status = 'read',
          endDate = ?
      WHERE id = ?
    `).run(timer.endTime, timer.bookId);
  }

  res.json({ message: "Дякую за підтвердження!", confirmed:
!!confirm });
});
app.delete("/timers/:id", isAuthenticated, (req, res) => {
  const timerId = Number(req.params.id);
  const exists = db.query(
    "SELECT COUNT(*) as cnt FROM timers WHERE id = ?
AND userId = ?"
  ).get(timerId, req.user.id).cnt;
  if (!exists) {
    return res.status(404).json({ message: "Таймер не
знайдено" });
  }
  db.query("DELETE FROM timers WHERE id =
?").run(timerId);
  res.json({ success: true });
});
app.delete("/timers", isAuthenticated, (req, res) => {
  db.query("DELETE FROM timers WHERE userId =
?").run(req.user.id);
  res.json({ success: true });
});
app.delete("/timers", isAuthenticated, (req, res) => {
  db.query("DELETE FROM timers WHERE userId =
?").run(req.user.id);
  res.json({ success: true });
});
function scheduleTimer(timerId) {
  const timer = db.query("SELECT * FROM timers WHERE id
= ? AND isNotified = 0").get(timerId);
  if (!timer) return;
  const ms = new Date(timer.endTime) - new Date();
  if (ms <= 0) return handleExpiration(timer);
  setTimeout(() => handleExpiration(timer), ms);
}

async function handleExpiration(timer) {
  db.query("UPDATE timers SET isNotified = 1 WHERE id =
?").run(timer.id);
  const user = db.query("SELECT email FROM users WHERE
id = ?").get(timer.userId);
  let subject, text;
  if (timer.bookId) {
    const book = db.query(
      "SELECT title, author FROM books WHERE id = ?"
    ).get(timer.bookId);
```

```
    const notePart = timer.note ? `\nНотатка: ${timer.note}` :
"";

    subject = `Ваш таймер для книги "${book.title}" сплив.`;
    text = [
      "Вітаємо!",
      "Ваш таймер для книги сплив.",
      "",
      "Ваша книга:",
      `  "${book.title}" - ${book.author || "(невідомий)"}` +
notePart,
      "",
      "Чи прочитали ви цю книгу за встановлений час?",
      "Підтвердіть прочитання: http://localhost:3000/timer.html"
    ].join("\n");
  } else {
    subject = `Ваш таймер сплив.`;
    text = [
      "Вітаємо!",
      "Ваш таймер сплив.",
      "",
      `Ваш таймер:`,
      `Нотатка: "${timer.note || ""}"`,
      "",
      "Не забувайте за свій таймер!",
      "Ваші таймери: http://localhost:3000/timer.html"
    ].join("\n");
  }

  await transporter.sendMail({
    from: `"BookProgress" <${process.env.SMTP_USER}>`,
    to: user.email,
    subject,
    text
  });

  const book = timer.bookId
    ? db.query("SELECT title, author FROM books WHERE id
= ?")
      .get(timer.bookId)
    : null;

  const notePart = timer.note ? ` Нотатка: ${timer.note}` : "";

  const inAppMessage = timer.bookId
    ? `Таймер для книги "${book.title}" — ${book.author ||
"(невідомий)"}, сплив.` +
    `${notePart}` +
    ` Ви прочитали книгу?`
    : `Ваш нотатковий таймер сплив.${notePart}`;

  db.query(`
    INSERT INTO notifications (userId, timerId, message,
createdAt)
    VALUES (?, ?, ?, ?)
  `).run(
    timer.userId,
    timer.id,
    inAppMessage,
    new Date().toISOString()
  );

  if (timer.bookId) {
    const bookState = db.query(
      "SELECT status FROM books WHERE id = ?"
    ).get(timer.bookId);
    if (bookState.status === "notStarted") {
      db.query(
        "UPDATE books SET status = 'reading', startDate = ?
WHERE id = ?"
      ).run(timer.startTime, timer.bookId);
    }
  }
}
app.get("/notifications", isAuthenticated, (req, res) => {
  const notes = db.query(`
    SELECT n.*, t.bookId
    FROM notifications n
    LEFT JOIN timers t ON t.id = n.timerId
    WHERE n.userId = ? AND n.isRead = 0
  `).all(req.user.id);

  res.json(notes);
});

app.post("/notifications/:id/dismiss", isAuthenticated, (req, res)
=> {
  db.query("UPDATE notifications SET isRead = 1 WHERE id
= ?").run(Number(req.params.id));
  res.json({ message: "Сповіщення приховано" });
});
app.post("/forgot-password", async (req, res) => {
  const { email } = req.body;
  const user = db.query(`SELECT * FROM users WHERE
email = ?`).get(email);

  if (!user) return res.json({ message: "Ми надіслали
посилання для створення нового паролю вам на електронну
адресу." });

  const token = crypto.randomBytes(32).toString("hex");
  resetTokens.set(token, email);
  setTimeout(() => resetTokens.delete(token), 1000 * 60 * 15);

  const resetUrl = `http://localhost:3000/reset-
password.html?token=${token}`;
  try {
    const info = await transporter.sendMail({
      from: `"BookProgress" <${process.env.SMTP_USER}>`,
      to: email,
      subject: "Скидання паролю",
      text: `Щоб створити новий пароль, перейдіть за
посиланням:\n\n${resetUrl}`
    });
    console.log("→ Reset password email sent:",
info.messageId);
  } catch (err) {
    console.error(" Error sending reset-password email:", err);
  }

  res.json({ message: "Ми надіслали посилання для
створення нового паролю вам на електронну адресу." });
});
```

```
app.post("/reset-password", async (req, res) => {
  const { token, newPassword } = req.body;
  const email = resetTokens.get(token);
  if (!email) {
    return res.status(400).json({ message: "Недійсний або
прострочений токен." });
  }

  const err = checkPasswordServer(newPassword);
  if (err) {
    return res.status(400).json({ message: err });
  }

  const user = db.query(`SELECT * FROM users WHERE
email = ?`).get(email);
  if (!user) {
    return res.status(400).json({ message: "Користувача не
знайдено." });
  }

  if (user.password) {
    const same = await bcrypt.compare(newPassword,
user.password);
    if (same) {
      return res.status(400).json({ message: "Новий пароль не
може бути використаним раніше" });
    }
  }

  const hashed = await bcrypt.hash(newPassword, 10);
  db.query(`UPDATE users SET password = ? WHERE email
= ?`).run(hashed, email);
  resetTokens.delete(token);

  res.json({ message: "Пароль змінено." });
});
app.get("/stats", isAuthenticated, (req, res) => {
  const books = db.query(`SELECT * FROM books WHERE
userId = ?`).all(req.user.id);

  const stats = {
    read: 0,
    reading: 0,
    notStarted: 0,
    totalHours: 0,
    months: {},
    genres: {}
  };

  const monthNames = [
    "Січень", "Лютий", "Березень", "Квітень", "Травень",
"Червень",
    "Липень", "Серпень", "Вересень", "Жовтень",
"Листопад", "Грудень"
  ];

  for (const book of books) {
    const pages = book.pages || 0;

    if (book.status === "read") stats.read++;
    else if (book.status === "reading") stats.reading++;
    else stats.notStarted++;

    if (
      book.status === "read" &&
      book.startDate &&
      book.endDate &&
      new Date(book.endDate) > new Date(book.startDate)
    ) {
      const start = new Date(book.startDate);
      const end = new Date(book.endDate);
      const hours = (end - start) / (1000 * 60 * 60);

      stats.totalHours += hours;
    }

    if (book.startDate && book.endDate) {
      const start = new Date(book.startDate);
      const end = new Date(book.endDate);
      const totalMs = end - start;

      if (totalMs > 0) {
        const hourMs = 1000 * 60 * 60;
        const totalHours = Math.ceil(totalMs / hourMs);

        const hourlyDistribution = {};

        for (let t = +start; t < +end; t += hourMs) {
          const hour = new Date(t);
          const year = hour.getFullYear();
          const month = monthNames[hour.getMonth()].slice(0,
3);
          const key = `${year}-${month}`;

          if (!hourlyDistribution[key]) hourlyDistribution[key] = 0;
          hourlyDistribution[key] += 1;
        }

        for (const key in hourlyDistribution) {
          const portion = hourlyDistribution[key] / totalHours;
          if (!stats.months[key]) stats.months[key] = 0;
          stats.months[key] += pages * portion;
        }
      }
    }

    if (book.genre) {
      let genreValue = book.genre;

      try {
        const parsed = JSON.parse(book.genre);
        if (Array.isArray(parsed)) genreValue = parsed[0];
      } catch (e) { }

      if (genreValue && typeof genreValue === "string" &&
genreValue.trim() !== "") {
        if (!stats.genres[genreValue]) stats.genres[genreValue] =
0;
        stats.genres[genreValue]++;
      }
    }
  }

  res.json(stats);
```

```
});

app.post("/delete", isAuthenticated, async (req, res) => {
  try {
    const user = req.user;

    if (user.picture && user.picture.startsWith("/uploads/")) {
      try {
        await unlink(`public${user.picture}`);
      } catch (err) {
        console.warn("Не вдалося видалити зображення
профілю при видаленні акаунта:", err.message);
      }
    }
    db.query(`DELETE FROM users WHERE email =
?`).run(user.email);
    res.clearCookie("token");
    res.json({ message: "Акаунт видалено" });
  } catch (error) {
    console.error(error);
    res.status(500).json({ message: "Помилка при видаленні
акаунта." });
  }
});
const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {
  console.log(`Server listening on http://localhost:${PORT}`);
});
```