МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

ПЕТРИШИН ВІТАЛІЙ СЕРГІЙОВИЧ

Допускається до захисту:
в.о. завідувача кафедри
інформаційних технологій
канд. техн. наук, доцент
_____ О. В. Зелінська
«____»_____ 20__ р.

РОЗРОБКА КЛІЄНТСЬКОЇ ЧАСТИНИ ВЕБДОДАТКА ЕЛЕКТРОННОЇ БІБЛІОТЕКИ

Спеціальність 122 Комп’ютерні науки

Кваліфікаційна (бакалаврська) робота

Керівник:
Н. Р. Веселовська, професор
кафедри інформаційних технологій
д.т.н., професор
_____

Оцінка: _____ / _____ / _____
(бали/за шкалою ЄКТС/за національною шкалою)

Голова ЕК: _____

Вінниця – 2025

ЗМІСТ

ВСТУП 3
РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАННЯ 6
1.1 Поняття електронної бібліотеки 6
1.2 Аналіз існуючих рішень 7
1.3 Опис цільового користувача 11
1.4 Функціональні вимоги та сценарії використання 12
РОЗДІЛ 2. ПРОЄКТУВАННЯ КЛІЄНТСЬКОЇ ЧАСТИНИ ВЕБДОДАТКУ 16
2.1 Вибір технологій для реалізації 16
2.2 Структура вебдодатку 18
2.3 Опис компонентів інтерфейсу 21
2.4 Розробка макетів 23
РОЗДІЛ 3. РЕАЛІЗАЦІЯ КЛІЄНТСЬКОЇ ЧАСТИНИ ВЕБДОДАТКУ 29
3.1 Інтеграція з базою даних через Docker 29
3.2 Структура клієнтської частини вебдодатку 31
3.3 Основний функціонал вебінтерфейсу 33
3.4 Система автентифікації та управління профілем 41
ВИСНОВКИ 47
СПИСОК ВИКОРИСТАНИХ ПОСИЛАНЬ 49
ДОДАТКИ 52

ВСТУП

Актуальність теми

У сучасному світі цифровізація проникає в усі сфери життя, і галузь освіти не є винятком. Одним із важливих напрямів цифрової трансформації є створення електронних бібліотек, які надають користувачам зручний та швидкий доступ до навчальних матеріалів, наукової літератури, художніх творів та інших інформаційних ресурсів. Зростання кількості навчальних закладів, дистанційного навчання та попиту на електронні джерела зумовлює потребу у зручних, адаптивних та ефективних вебдодатках, що дозволяють користувачам взаємодіяти з бібліотечними ресурсами в режимі онлайн.

Клієнтська частина вебдодатку відіграє ключову роль у забезпеченні зручності користування системою. Від її якості, інтуїтивності та функціональності залежить, наскільки легко користувач зможе знаходити потрібні книги, реєструватися, авторизуватись, переглядати описи, зберігати обране тощо. Тому розробка сучасного, адаптивного та інтерактивного інтерфейсу є актуальним завданням, що відповідає вимогам сьогодення.

Особливого значення це набуває у контексті розвитку відкритих освітніх платформ, де важлива доступність знань для широкого кола користувачів незалежно від їхнього місця перебування чи типу пристрою. Таким чином, тема дипломної роботи є актуальною як з технічної, так і з освітньо-соціальної точки зору.

Мета та завдання дослідження

Метою дипломної роботи є розробка клієнтської частини вебдодатку електронної бібліотеки, яка забезпечує зручну взаємодію користувача з інформаційною системою, дозволяє ефективно здійснювати пошук, перегляд та управління бібліотечними ресурсами, а також адаптується до різних типів пристроїв.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- Провести аналіз предметної області та вивчити існуючі рішення електронних бібліотек

- Визначити функціональні та нефункціональні вимоги до клієнтської частини вебдодатку.
- Обрати оптимальні технології для реалізації інтерфейсу користувача.
- Розробити структуру та макети вебінтерфейсу електронної бібліотеки.
- Реалізувати основні функціональні компоненти клієнтської частини: реєстрація, авторизація, пошук, перегляд інформації про книги, управління обраним.

Об’єкт, предмет та методи дослідження

Об’єктом дослідження є процес розробки вебзастосунків, зокрема клієнтської частини, яка забезпечує взаємодію користувача з інформаційною системою.

Предметом дослідження є сукупність технологій, методів та інструментів, які використовуються для проєктування, реалізації та тестування клієнтської частини вебдодатку електронної бібліотеки, а також особливості побудови зручного, функціонального та адаптивного інтерфейсу користувача.

Методи дослідження, використані у роботі

- Аналіз наукових джерел та аналогів — для вивчення сучасного стану розробки клієнтських частин електронних бібліотек, оцінки існуючих підходів до реалізації вебінтерфейсів, їхньої функціональності та зручності користування.
- Синтез — для об’єднання отриманих знань і підходів у єдину концепцію структури вебінтерфейсу бібліотеки.
- Індукція та дедукція — для формування висновків щодо вибору найефективніших рішень на основі вивчених прикладів та окремих спостережень.
- Системний підхід — для побудови логічної та функціональної структури клієнтської частини, враховуючи взаємодію всіх її компонентів із серверною частиною.

- Моделювання — для формування сценаріїв взаємодії користувача з інтерфейсом і побудови макетів сторінок із застосуванням Figma.
- Емпіричні методи (спостереження, експеримент) — для перевірки працездатності інтерфейсу в реальних умовах, оцінки зручності користування.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1 Поняття електронної бібліотеки

Електронна бібліотека — це інформаційна система нового покоління, що забезпечує накопичення, збереження, систематизацію, обробку, пошук та надання доступу до електронних інформаційних ресурсів за допомогою комп'ютерних технологій. Вона є логічним етапом еволюції традиційних бібліотек у напрямку цифровізації та глобальної доступності інформації. Сьогодні електронні бібліотеки відіграють ключову роль у науково-освітньому процесі, дозволяючи мільйонам користувачів у будь-якій точці світу отримувати доступ до знань швидко, зручно та ефективно.

Зміст електронної бібліотеки зазвичай включає:

- електронні копії друкованих видань;
- електронні книги, створені безпосередньо у цифровому форматі;
- наукові статті, дисертації, звіти, аналітичні матеріали;
- аудіо- та відеоматеріали;
- мультимедійні інтерактивні ресурси.

Однією з найважливіших переваг електронних бібліотек є можливість організовувати доступ до великого масиву даних у зручний та структурований спосіб. Це досягається завдяки використанню метаданих, класифікаційних схем, систем каталогізації та індексації. За допомогою пошукових систем користувачі можуть швидко знаходити необхідну інформацію, фільтрувати її за темою, автором, роком видання та іншими параметрами.

Електронні бібліотеки реалізуються за допомогою сучасних вебтехнологій, де важливу роль відіграє клієнтська частина вебдодатку. Саме клієнтська частина є тією складовою, з якою безпосередньо взаємодіє користувач. Вона повинна забезпечувати зручний і зрозумілий інтерфейс, коректне відображення вмісту на різних пристроях (комп'ютерах, планшетах, смартфонах), адаптивний дизайн,

високий рівень юзабіліті, а також інтерактивність — наприклад, можливість додавати книги до обраного, залишати рецензії, формувати добірки тощо.

Крім функціональних можливостей, важливою складовою електронної бібліотеки є безпека даних — як особистих, так і авторських. Тому при розробці таких систем часто застосовуються механізми аутентифікації, авторизації, шифрування та обмеження прав доступу. Це особливо актуально для академічних або комерційних бібліотек, де доступ до окремих документів може бути платним або ліцензованим.

Слід також відзначити, що електронні бібліотеки бувають:

- публічні — доступні для широкого кола користувачів (наприклад, Національна бібліотека України імені В. І. Вернадського);
- академічні — для студентів і викладачів навчальних закладів (наприклад, eLibrary, ScienceDirect, SpringerLink);
- спеціалізовані — що обслуговують певну галузь знань (медичні, технічні, правові тощо);
- корпоративні — внутрішні бібліотеки підприємств або організацій.

1.2 Аналіз існуючих рішень

На сьогодні існує велика кількість електронних бібліотек, які відрізняються функціональністю, дизайном, архітектурою, специфікою вмісту та масштабом. У цьому пункті буде проведено аналіз декількох популярних рішень, які можуть слугувати орієнтиром для розробки власного вебдодатку. Основну увагу буде зосереджено на клієнтській частині: інтерфейсі, юзабіліті, функціоналі та зручності для користувача.

1. Національна бібліотека України імені В. І. Вернадського (http://www.nbuv.gov.ua)

Це одна з найбільших національних електронних бібліотек, яка забезпечує доступ до наукових та освітніх ресурсів. Інтерфейс сайту є функціональним, але

досить застарілим з точки зору сучасних UI/UX стандартів. Основні недоліки: відсутність адаптивного дизайну, складна навігація, перевантаженість інформацією на головній сторінці. Водночас, функціонал пошуку та сортування документів є досить потужним, що підходить для наукової роботи.

2. Google Books (https://books.google.com)

Google Books — одна з наймасштабніших платформ, яка містить мільйони оцифрованих книг різними мовами. Клієнтська частина реалізована на сучасному рівні: мінімалістичний дизайн, адаптивність, зручний пошук, інтеграція з акаунтом Google, можливість попереднього перегляду частини книги. Недоліком є обмежений доступ до повних текстів — більшість матеріалів доступні лише частково.

3. Project Gutenberg (https://www.gutenberg.org)

Проєкт Гутенберг — це некомерційна електронна бібліотека, яка містить понад 60 000 книг, переважно класичної літератури, що перебуває у відкритому доступі. Користувацький інтерфейс є дуже простим, але при цьому зручним. Основна увага приділена функціональності: швидкий пошук, завантаження книг у різних форматах (EPUB, Kindle, TXT, HTML). Сайт ідеально підходить для прикладу легкої та доступної реалізації клієнтської частини.

4. Open Library (https://openlibrary.org)

Open Library — проєкт Internet Archive, який має на меті створити «вебсторінку для кожної книги». Інтерфейс бібліотеки простий, сучасний і зрозумілий навіть для новачків. Присутній особистий кабінет, списки прочитаного, можливість «брати книги у користування» (за аналогією з реальними бібліотеками). Клієнтська частина реалізована з урахуванням адаптивності, юзабіліті та доступності.

Таблиця 1.1 Порівняння інтерфейсів електронних бібліотек

Платформа	Адаптивність	Зручність пошуку	Сучасний інтерфейс	Відкритий доступ	Особливості
НБУВ	Ні	Так	Ні	Частково	Потужний пошук, але застарілий дизайн
Google Books	Так	Так	Так	Обмежено	Пошук за фрагментами тексту
Project Gutenberg	Так		Ні	Так	Відкриті формати книг
Open Library	Так	Так	Так	Частково	Власний акаунт, видача книг

На основі аналізу можна зробити висновок, що сучасна клієнтська частина електронної бібліотеки повинна бути:

- адаптивною до мобільних і настільних пристроїв;
- зручною та інтуїтивною для користувача;
- мінімалістичною за дизайном, але з широким функціоналом;
- здатною до розширення (інтеграції з API, соціальними мережами тощо).

Сильні та слабкі сторони існуючих рішень

На основі аналізу сучасних електронних бібліотек (Національної бібліотеки України ім. В. І. Вернадського, Google Books, Project Gutenberg, Open Library тощо), можна виокремити ключові переваги та недоліки цих рішень з

точки зору клієнтської частини, що має велике значення для подальшої розробки вебдодатку.

Сильні сторони

1. *Зручний доступ до великої кількості ресурсів*
Більшість сучасних електронних бібліотек надають користувачам можливість переглядати або завантажувати тисячі цифрових документів з різних джерел. Це робить такі платформи корисними як для пересічних читачів, так і для науковців.

2. *Розширений пошук та фільтрація*
У Google Books, Open Library та інших платформах реалізовано зручний повнотекстовий пошук, сортування результатів, фільтри за категоріями, авторами, датами публікацій, що значно полегшує навігацію.

3. *Адаптивний дизайн*
Деякі рішення (зокрема, Google Books, Open Library) мають повноцінну адаптивність, що дозволяє користуватися бібліотекою на смартфонах, планшетах та інших пристроях без втрати функціональності.

4. *Підтримка декількох форматів документів*
Сайти на кшталт Project Gutenberg дозволяють завантажувати книги в різних форматах — PDF, EPUB, TXT, HTML — залежно від уподобань користувача.

5. *Можливості персоналізації*
У деяких бібліотеках реалізовано функціонал особистого кабінету, історії читання, створення добірок, що покращує користувацький досвід.

Слабкі сторони

1. *Застарілий інтерфейс і слабкий UX*
Багато електронних бібліотек, особливо академічного спрямування (як-от НБУВ, eLibrary), мають застарілий візуальний стиль і складну навігацію. Це робить їх малозручними для сучасних користувачів, особливо молодої аудиторії.

2. *Відсутність адаптивності на всіх пристроях*
Деякі платформи не підтримують повноцінну адаптацію до мобільних екранів, через що знижуються доступність і юзабіліті при використанні смартфонів або планшетів.

3. *Низька інтерактивність інтерфейсу*
Бракує сучасних функцій взаємодії з користувачем — інтерактивних підказок, drag-and-drop, миттєвих реакцій інтерфейсу на дії користувача (без перезавантаження сторінки).

4. *Складність пошуку та перевантаження інформацією*
У деяких бібліотеках головна сторінка перевантажена елементами (текст, посилання, банери), що відволікає користувача і ускладнює пошук основної інформації.

5. *Відсутність сучасних технологій фронтенду*
Багато рішень побудовані без використання сучасних фреймворків (React, Vue, Angular), що обмежує швидкість роботи, зручність оновлення інтерфейсу, гнучкість у масштабуванні.

1.3 Опис цільового користувача

Цільовими користувачами електронної бібліотеки є:

- Студенти вищих навчальних закладів, які шукають навчальні матеріали, підручники, наукові статті та інші ресурси для підготовки до занять або написання курсових/дипломних робіт.
- Викладачі, які використовують платформу для пошуку та розповсюдження навчального контенту, формування рекомендованої літератури, а також для перевірки наявності потрібних джерел.
- Науковці та дослідники, яким необхідний швидкий доступ до академічної літератури для проведення досліджень.
- Загальна аудиторія — користувачі, які цікавляться самоосвітою, читанням книг або пошуком інформації.

Основні очікування користувачів від електронної бібліотеки:

- Простий та інтуїтивно зрозумілий інтерфейс.
- Швидкий пошук потрібних матеріалів.
- Можливість фільтрації та сортування результатів.
- Збереження історії перегляду та обраних книг.
- Доступ до повного тексту книг або фрагментів.

1.4 Функціональні вимоги та сценарії використання

Функціональні вимоги визначають перелік основних можливостей, які повинна реалізовувати клієнтська частина вебдодатку. У випадку електронної бібліотеки ці вимоги спрямовані на забезпечення комфортного доступу користувачів до інформаційних ресурсів, керування власним профілем та взаємодії з контентом. Нижче наведено перелік ключових функцій, які мають бути реалізовані.

Каталог книг

- Виведення списку усіх книг, доступних у бібліотеці.
- Пошук за такими параметрами: назва, автор.
- Сортування за жанром

Сторінка книги

- Відображення детальної інформації про книгу: назва, автор, жанр, рік видання, короткий опис, обкладинка.
- Кнопка «Читати онлайн» — відкриття книги у вбудованому рідері.
- Кнопка «Завантажити» — перелік доступних форматів для завантаження (PDF, TXT, FB2, DOCX тощо).
- Додавання книги в обране.

Сторінка для читання книги

- Відкриття повного тексту книги в окремому вікні або розділі.
- Можливість змінювати розмір шрифту.

Сценарії використання

Для ефективного тестування та демонстрації роботи клієнтської частини вебдодатку були сформульовані типові сценарії використання системи. Вони відображають найпоширеніші дії, які користувачі виконують під час взаємодії з електронною бібліотекою.

Реєстрація нового користувача

Умова: користувач не має облікового запису.

Дії:

- Перехід на сторінку реєстрації.
- Заповнення полів: ім’я, email, пароль, підтвердження паролю.
- Натискання кнопки «Зареєструватися».

Результат: після успішної реєстрації користувач перенаправляється на сторінку входу.

Вхід до системи

Умова: користувач має обліковий запис.

Дії:

- Перехід на сторінку входу.
- Введення email і пароля.
- Натискання кнопки «Увійти».

Результат: користувача перенаправляє до каталогу книг, у хедері з’являється його ім’я.

Перегляд каталогу книг

Умова: користувач знаходиться на головній сторінці.

Дії:

- Перегляд карток книг.

- Використання пошуку за назвою або автором.

Результат: відображаються лише релевантні книги відповідно до введених параметрів.

Перегляд інформації про книгу

Умова: користувач вибирає певну книгу з каталогу.

Дії:

- Натискання на кнопку «Переглянути» або назву книги.

Результат: відкривається детальна сторінка з описом, автором, жанром, рейтингом і кнопками «Читати онлайн» / «Завантажити».

Читання книги онлайн

Умова: користувач знаходиться на сторінці книги.

Дії:

- Натискання кнопки «Читати онлайн».
- Регулювання розміру шрифту при читанні.

Результат: відкривається текст книги у зручному форматі з можливістю змінювати розмір шрифту.

Додавання книги до обраного

Умова: користувач авторизований.

Дії:

- Натискання на іконку зірочки на картці книги або на сторінці книги.

Результат: книга додається до списку обраного користувача.

Перегляд обраного

Умова: користувач переходить на сторінку «Обране».

Дії:

- Перегляд книг, які були збережені.
- Можливість видалення книги зі списку.

Результат: відображається лише перелік улюблених книг; зміни відбуваються без перезавантаження сторінки.

Редагування профілю

Умова: користувач авторизований.

- Дії:
- Перехід на сторінку профілю.
- Натискання кнопки «Редагувати профіль».
- Зміна імені або email, збереження змін.

Результат: оновлені дані зберігаються у системі та localStorage.

Вихід із системи

Умова: користувач перебуває у своєму профілі.

Дії:

- Натискання кнопки «Вийти з профілю».

Результат: дані користувача видаляються з localStorage, інтерфейс оновлюється (замість імені знову відображається кнопка «Увійти»).

РОЗДІЛ 2. ПРОЄКТУВАННЯ КЛІЄНТСЬКОЇ ЧАСТИНИ ВЕБДОДАТКУ

2.1 Вибір технологій для реалізації

При розробці клієнтської частини вебдодатку електронної бібліотеки особлива увага була приділена вибору сучасних технологій, які дозволяють створити ефективний, надійний, швидкий і зручний для користувачів додаток. Обрані технології оптимально поєднують простоту використання, швидкість розробки, доступність та можливість масштабування.

Технології фронтенду (клієнтська частина)

HTML (HyperText Markup Language)

Для створення структури вебсторінок використано HTML, який визначає базову розмітку документів. Саме за допомогою HTML визначено каркас і основні компоненти сторінок, такі як шапка сайту, навігаційні елементи, форми реєстрації та входу, картки книг, профіль користувача та інші елементи інтерфейсу. Використано семантичні теги, такі як <header>, <main>, <footer> для покращення доступності та SEO-оптимізації.

CSS (Cascading Style Sheets)

CSS застосовано для стилізації елементів і створення візуально привабливого та адаптивного дизайну. Було реалізовано адаптивну верстку за допомогою Flexbox і CSS Grid, що дозволило забезпечити комфортний перегляд вебдодатку як на десктопних пристроях, так і на мобільних. Використано медіазапити (@media), що роблять інтерфейс зручним для користувачів із різними пристроями та розмірами екрану.

JavaScript (JS)

JavaScript використано для забезпечення інтерактивності та динамічності вебдодатку. За допомогою JavaScript були реалізовані наступні функції:

- Завантаження даних про книги та авторів із JSON-файлів через асинхронні fetch-запити.
- Динамічне формування контенту сторінок (каталог книг, детальна інформація про книгу, сторінка читання книги).

- Реалізація пошуку за назвою книг та авторами, а також фільтрація за жанрами.
- Обробка форм реєстрації, входу та редагування профілю користувачів.
- Робота з localStorage для збереження даних про авторизацію користувачів та обрані книги.
- Можливість змінювати розмір шрифту на сторінці читання книг.

Для покращення користувацького досвіду також застосовано інтерактивні елементи, наприклад, автоматичне оновлення списку книг при зміні параметрів пошуку та фільтрації без перезавантаження сторінки.

Технології бекенду (серверна частина)

Docker

1. Для розгортання та управління серверною частиною додатку обрано Docker — технологію контейнеризації, яка дозволяє створити повністю ізольоване середовище для роботи додатку. Docker забезпечує:
2. Просте налаштування і швидке розгортання вебдодатку, незалежно від середовища виконання (операційної системи, версій бібліотек тощо).
3. Легкість масштабування і управління версіями програмного забезпечення.
4. Зручність налаштування середовища для розробки та тестування різних компонентів додатку.

PostgreSQL

1. Для зберігання та управління даними вибрано PostgreSQL, що є однією з найбільш потужних і стабільних систем управління реляційними базами даних. Переваги PostgreSQL:
2. Висока продуктивність, стабільність і надійність роботи з великими обсягами даних.
3. Розширені можливості для роботи з типами даних, складними запитами, індексами.

4. Простота інтеграції з різними мовами програмування і бібліотеками, що дозволяє гнучко масштабувати базу даних в процесі розробки і підтримки додатку.

Інструменти та допоміжні засоби

1. JSON (JavaScript Object Notation) — формат даних для передачі інформації між сервером і клієнтом, використовується для опису та зберігання даних про книги, авторів, профілі користувачів.

2. Figma — для створення прототипів і макетів сторінок вебдодатку, які допомогли чітко уявити кінцевий вигляд проєкту перед його реалізацією.

3. Visual Studio Code — як інтегроване середовище розробки (IDE), яке забезпечує зручну та ефективну розробку завдяки багатому набору плагінів і розширень.

Переваги вибраних технологій

1. Простота та зрозумілість у використанні для швидкого початку розробки.

2. Висока продуктивність та швидкість роботи вебдодатку завдяки використанню асинхронних запитів і локальному зберіганню даних.

3. Надійність збереження та обробки даних завдяки сучасній та стабільній базі даних PostgreSQL.

4. Гнучкість і масштабованість завдяки використанню контейнеризації Docker.

5. Простота підтримки та розширення функціоналу, включаючи додавання нових компонентів або модулів без суттєвого переписування наявного коду.

Таким чином, комплексне застосування вищезгаданих технологій дозволило створити функціональний, надійний і простий в обслуговуванні вебдодаток, що відповідає сучасним стандартам розробки та забезпечує високий рівень задоволеності користувачів.

2.2 Структура вебдодатку

Для забезпечення чіткої організації та зручності підтримки проєкту було розроблено наступну структуру вебдодатку, яка представлена двома основними частинами: клієнтською (frontend) та серверною (backend).

Структура серверної частини (backend):

Серверна частина вебдодатку відповідає за зберігання, управління та обробку даних. Вона має таку структуру:

- data/ - папка для зберігання JSON-файлів, які використовуються для початкового імпорту даних до бази даних.
- auth.js – файл, що забезпечує логіку автентифікації користувачів на стороні сервера.
- db.js – налаштування підключення до бази даних PostgreSQL та взаємодія з нею.
- import-data.js – скрипт для імпортування початкових даних із JSON-файлів у базу даних.
- init.sql – SQL-скрипт для ініціалізації схеми бази даних.
- Dockerfile – конфігурація контейнера для запуску бекенду.
- package.json – файл, який описує проєкт, керує залежностями та скриптами запуску.
- server.js – головний файл сервера, який обробляє HTTP-запити, реалізує API для взаємодії з клієнтом.

Структура клієнтської частини (frontend):

Клієнтська частина відповідає за візуалізацію інформації та взаємодію з користувачем. Її структура містить такі основні компоненти:

- assets/ - містить статичні ресурси вебдодатку: зображення книг, іконки тощо.
- css/ - папка для зберігання файлів стилів (CSS), які забезпечують оформлення та адаптивність сторінок.
- js/ - містить JavaScript-файли, що реалізують логіку роботи клієнтської частини, зокрема динамічне завантаження даних, взаємодію з формами, керування інтерфейсом та локальним зберіганням.

Основні HTML-файли сторінок клієнтської частини включають:

- index.html — головна сторінка, що містить каталог книг.
- book.html — детальна сторінка конкретної книги з описом та функціями читання й завантаження.
- author.html — сторінка окремого автора з його біографією та списком книг.
- authors.html — сторінка, яка містить список усіх авторів бібліотеки.
- favorites.html — сторінка, що містить книги, додані користувачем до обраних.
- genre.html — сторінка з книгами певного жанру.
- genres.html — сторінка зі списком доступних жанрів.
- login.html — сторінка входу в особистий кабінет користувача.
- register.html — сторінка реєстрації нових користувачів.
- profile.html — сторінка профілю користувача, що дозволяє переглядати та редагувати персональні дані.
- read.html — сторінка для читання книги онлайн з можливістю змінювати розмір шрифту.

Фронтенд-додаток розгортається в окремому Docker-контейнері, що дозволяє легко керувати середовищем розробки, тестування та публікації.

Переваги обраної структури:

1. Модульність і ясність: чітке розділення на фронтенд та бекенд дозволяє легко орієнтуватись у проєкті, зручно керувати змінами та підтримувати код.
2. Гнучкість і масштабованість: додавання нових сторінок, функцій чи сервісів не потребує суттєвого втручання в загальну структуру.
3. Простота підтримки: зрозуміла організація файлів полегшує подальший розвиток та підтримку вебдодатку, а також дозволяє швидко локалізувати й вирішувати можливі проблеми.

Таким чином, структура вебдодатку забезпечує оптимальну ефективність, легкість у масштабуванні й підтримці, що особливо важливо для динамічного розвитку проєкту.

2.3 Опис компонентів інтерфейсу

Інтерфейс клієнтської частини вебдодатку електронної бібліотеки було спроєктовано відповідно до принципів зручності, простоти та доступності. Всі сторінки виконані в єдиному стилі, мають чітку візуальну ієрархію та зберігають спільні елементи навігації, що забезпечує послідовність і передбачуваність взаємодії для користувача.

Кожна сторінка має фіксовану верхню панель навігації (header), яка включає логотип сайту, а також основні пункти меню: «Каталог», «Обране», «Автори», «Жанри» та область для авторизації користувача, де відображається його ім’я або кнопка «Увійти». Така структура дозволяє користувачеві швидко переміщатися між основними розділами вебсайту незалежно від того, на якій сторінці він перебуває.

Головна сторінка — це каталог усіх книг, реалізований у вигляді сітки карток. Над списком книг розміщено поле пошуку, що дозволяє здійснювати фільтрацію за назвою. Кожна картка книги містить обкладинку, назву, посилання на автора та жанр, кнопку «Перейти» та іконку у вигляді зірочки для додавання книги до обраного. Користувач може взаємодіяти з цими елементами без потреби переходити на інші сторінки, що значно покращує зручність використання каталогу. Дизайн адаптовано до різних розмірів екранів — відображення книжок змінюється залежно від ширини вікна браузера, завдяки використанню CSS Grid.

При переході на сторінку конкретної книги користувач бачить велику обкладинку, назву, автора, жанр, рейтинг, а також кнопки для читання онлайн та завантаження книги у різних форматах (наприклад, TXT або FB2). Нижче розміщується розгорнутий опис книги. Такий формат дозволяє отримати повну інформацію про книгу без перевантаження інтерфейсу.

Сторінка «Жанри» містить заголовок та список усіх доступних жанрів, кожен з яких є активним посиланням. Після вибору жанру користувач потрапляє

на сторінку з відфільтрованими книгами лише цього жанру, оформленими у вигляді карток. Структура карток ідентична тій, що в головному каталозі.

Сторінка «Обрані книги» також відображає книги у вигляді сітки. Вона динамічно наповнюється відповідно до вибору користувача. Якщо в обраному немає книг — виводиться інформативне повідомлення про відсутність доданих позицій.

Усі автори представлені на окремій сторінці у вигляді вертикальних карток, які містять фотографію автора, його ім’я та перелік жанрів. Для зручності реалізоване поле пошуку автора за ім’ям. При натисканні на картку відкривається сторінка з детальною інформацією про автора, де відображено біографічну довідку та перелік книг, написаних ним.

Сторінка профілю користувача дає змогу змінити особисті дані: ім’я, електронну адресу, пароль. Також відображається дата реєстрації. Нижче на сторінці виводиться список обраних книг цього користувача, які дублюють структуру карток з каталогу. Крім того, доступна кнопка виходу з облікового запису.

Сторінка читання книги забезпечує зручний перегляд тексту у форматі .txt без завантаження файлу. Користувач може змінювати розмір шрифту завдяки інтерактивному випадаючому списку, що робить читання комфортним з різних пристроїв. Є також кнопка повернення на сторінку книги.

Форма реєстрації та входу реалізована у вигляді окремих сторінок із простим вертикальним розміщенням полів. Вона містить текстові поля для введення імені, електронної адреси та пароля, а також кнопку підтвердження. Передбачена базова валідація введених даних і повідомлення про помилки (наприклад, якщо паролі не співпадають або email вже зареєстрований).

Таким чином, інтерфейс вебдодатку побудований на принципах логічності, мінімалізму та функціональності. Усі сторінки мають уніфіковану структуру, що полегшує орієнтацію користувача, а елементи інтерфейсу є зрозумілими і взаємопов’язаними. Такий підхід дозволяє забезпечити зручний доступ до

інформації, спрощує навігацію та покращує загальне враження від використання додатку.

2.4 Розробка макетів

На етапі проєктування клієнтської частини вебдодатку було створено низку макетів сторінок, які дозволили візуально змоделювати структуру, логіку розміщення елементів інтерфейсу та загальний вигляд користувацького інтерфейсу. Макети виконували роль попередніх шаблонів, що дали змогу сформувати єдину стилістичну концепцію майбутнього вебзастосунку, протестувати зручність інтерфейсу та передбачити потенційні UX-проблеми ще до написання коду.

Сторінка входу

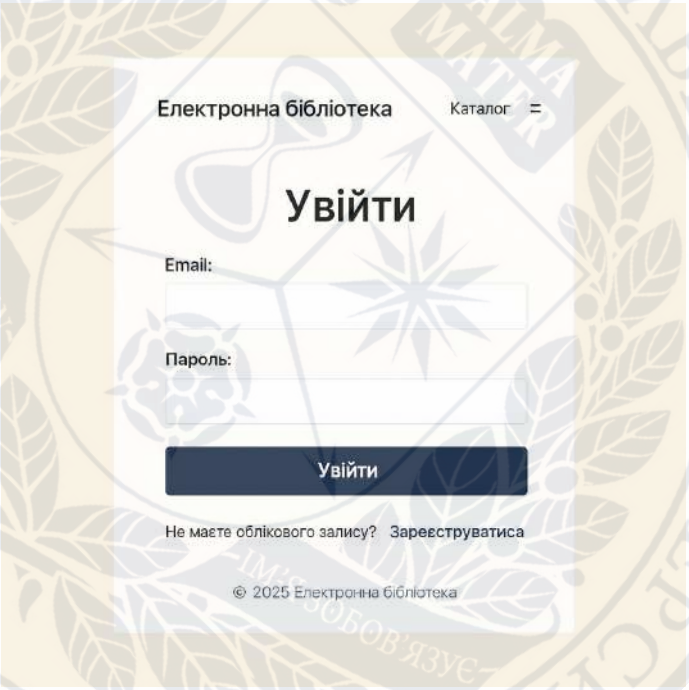


Рисунок 3.2 – Макет сторінки входу до електронної бібліотеки

Макет сторінки входу відображає мінімалістичний підхід до форми авторизації користувача. В центрі сторінки розташована форма, яка містить два поля вводу — електронну пошту та пароль, кнопку підтвердження «Увійти», а також посилання для переходу до форми реєстрації. Верхня частина сторінки містить логотип сайту та спрощене меню навігації з посиланням на каталог. Структура макета забезпечує інтуїтивно зрозумілу взаємодію користувача з додатком при вході в систему.

Головна сторінка (каталог книг)

Рисунок 3.3 – Макет головної сторінки каталогу

Макет головної сторінки слугує вітальним екраном та має на меті ознайомлення користувача з функціональністю електронної бібліотеки. У верхній частині макета розміщено привітальний заголовок, короткий опис можливостей ресурсу та поле пошуку з кнопкою «Шукати». Нижче структуровано два блоки: «Популярні книги» та «Новинки». Кожна книга представлена у вигляді кольорової картки, що містить умовну назву, автора та жанр. Такий макет дозволяє реалізувати динамічний вміст на основі запитів або оцінок користувачів і сприяє швидкому ознайомленню з найцікавішими матеріалами бібліотеки.

Сторінка окремої книги

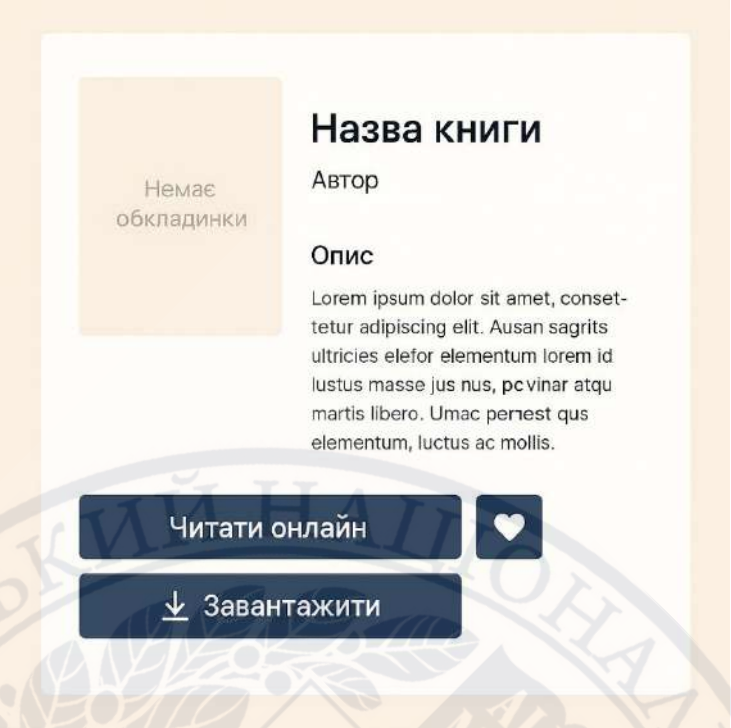


Рисунок 3.4 – Макет сторінки детального перегляду книги

Макет сторінки книги демонструє структуру перегляду інформації про конкретну книгу. Зліва передбачено місце для обкладинки (з плейсхолдером «Немає обкладинки» у разі її відсутності), справа — назва книги, автор, короткий опис, а також інтерактивні елементи: кнопка «Читати онлайн», кнопка завантаження файлу і піктограма додавання до обраного. Така компоновка дозволяє користувачеві швидко ознайомитись із основною інформацією про книгу та обрати зручний спосіб її перегляду чи завантаження.

Використання макетів на етапі проєктування значно спростило подальшу реалізацію клієнтської частини, оскільки дозволило заздалегідь узгодити елементи навігації, шаблони виводу контенту, адаптивну поведінку інтерфейсу та загальну візуальну ідентичність системи. Усі макети були побудовані з урахуванням адаптивного дизайну, що забезпечує зручне користування вебдодатком як на стаціонарних, так і на мобільних пристроях.

Навігація між сторінками

Навігація є одним із ключових елементів клієнтської частини вебдодатку, оскільки вона визначає зручність взаємодії користувача з інтерфейсом, забезпечує логічну послідовність переходів та дозволяє швидко знаходити

потрібну інформацію. В рамках розробки електронної бібліотеки навігація між сторінками реалізована як за допомогою постійної верхньої панелі навігації, так і за допомогою динамічних внутрішніх посилань, що змінюються в залежності від контексту.

Головним навігаційним елементом є універсальна шапка сайту (header), яка присутня на всіх основних сторінках вебдодатку. Вона містить логотип «Електронна бібліотека» (який за потреби може використовуватись як посилання на головну сторінку), а також горизонтальне меню з пунктами: Каталог, Обране, Автори, Жанри. Крім цього, праворуч у шапці розташовано кнопку або ім'я поточного користувача. У разі, якщо користувач не авторизований, ця область відображає посилання на сторінку входу. Якщо користувач увійшов у систему, замість кнопки «Увійти» з'являється його ім'я як активне посилання на профіль.

Каталог книг є центральною сторінкою вебдодатку. Саме з неї починається взаємодія користувача з вмістом бібліотеки. Кожна книга представлена у вигляді картки з обкладинкою, назвою, автором, жанром та кнопкою «Перейти». Натиснувши на цю кнопку, користувач переходить на сторінку книги, де може дізнатися більше про зміст, автора, жанр, переглянути рейтинг, а також — при бажанні — прочитати книгу онлайн або завантажити її у доступному форматі. Посилання на автора та жанр, що розміщені в описі книги, також є активними: вони дають змогу перейти на сторінку певного автора або жанру, відкриваючи пов'язану інформацію та список відповідних книг.

Окрему логіку має сторінка жанрів, яка відображає усі доступні жанри у вигляді вертикального списку. Кожен елемент списку — це гіперпосилання, яке веде на сторінку, де фільтруються книги лише за обраним жанром. Таким чином, реалізовано інтуїтивний механізм тематичної навігації. Сторінка обраного дозволяє користувачеві швидко перейти до збережених у localStorage книг, які були позначені як обрані за допомогою іконки-зірочки. Якщо таких книг немає, користувач отримає відповідне повідомлення.

Усі автори, наявні у базі даних, виводяться на окремій сторінці у вигляді сітки карток, кожна з яких містить фотографію, ім'я та жанрову належність. Вгорі

розташоване поле пошуку, що дозволяє здійснювати фільтрацію за іменем. Натиснувши на ім’я автора, користувач переходить до його персональної сторінки, де окрім розширеного опису також відображаються всі книги, написані цим автором, з можливістю перейти на сторінку кожної книги.

Сторінка профілю користувача виконує функцію персонального кабінету. Звідси користувач може змінити ім’я, email, пароль, а також переглянути дату реєстрації. У нижній частині сторінки повторно відображаються книги, які були додані в обране — що створює ефект індивідуальної бібліотеки. Крім того, реалізована можливість виходу з профілю — після натискання на відповідну кнопку локальне збереження даних скасовується, і користувач автоматично повертається до публічного режиму (без авторизації), а в шапці знову відображається кнопка «Увійти».

Сторінки входу та реєстрації є базовими для автентифікації. Вони взаємопов’язані через посилання у нижній частині форми, яке дозволяє змінити тип взаємодії: з реєстрації перейти до входу і навпаки. У разі успішної реєстрації або авторизації користувача автоматично перенаправляє на головну сторінку каталогу.

Сторінка читання книги онлайн відкривається за посиланням зі сторінки книги. Вона реалізує перегляд .txt-файлу, що виводиться у форматованому блоці. Є можливість змінювати розмір шрифту за допомогою випадаючого списку, що робить читання комфортним на різних пристроях. У верхній частині сторінки розміщено кнопку повернення, яка веде назад до сторінки книги.

Навігація між усіма сторінками реалізується без перезавантаження всієї сторінки, оскільки логіка побудована на використанні посилань з параметрами (?id=, ?name=), які обробляються скриптами JavaScript. Це забезпечує динамічне оновлення вмісту, а також дозволяє використовувати дані з локального сховища (localStorage) для збереження стану сесії, обраних книг і даних профілю.

Загалом, навігаційна структура вебдодатку логічна, симетрична та зручна для користувача. Вона забезпечує швидкий доступ до всіх функціональних розділів бібліотеки, підтримує глибоку зв’язність між сутностями (книга → автор

→ жанр → книга) та забезпечує плавний користувацький досвід незалежно від порядку переходів. Такий підхід відповідає сучасним вимогам до UX-дизайну та робить вебдодаток привабливим і доступним для широкої аудиторії.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ КЛІЄНТСЬКОЇ ЧАСТИНИ ВЕБДОДАТКУ

3.1 Інтеграція з базою даних через Docker

Для зберігання та обробки даних у вебдодатку використовується реляційна база даних PostgreSQL, розгорнута за допомогою Docker. Такий підхід забезпечує зручність налаштування, масштабованість та ізольоване середовище для бекенд-сервісів.

Розгортання бази даних у Docker

Базу даних було створено з використанням docker-compose.yml, який запускає окремий контейнер PostgreSQL. Цей контейнер автоматично виконує SQL-інструкції з файлу init.sql, що визначає структуру таблиць.

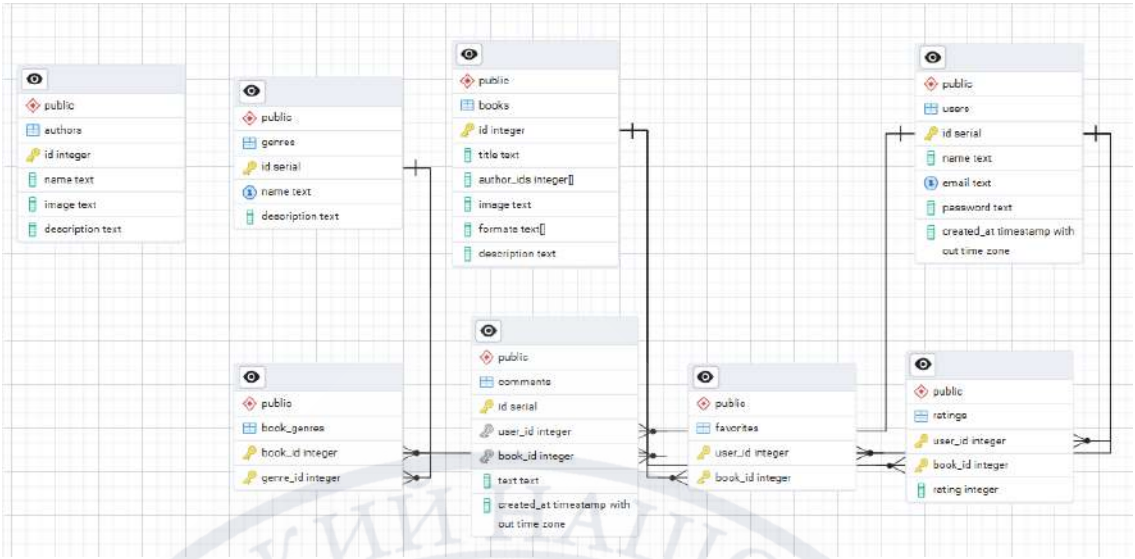
```
services:
  db:
    image: postgres:14
    environment:
      POSTGRES_DB: library
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: password
    ports:
      - "5432:5432"
    volumes:
      - ./backend/init.sql:/docker-entrypoint-initdb.d/init.sql
```

Структура бази даних

Базу даних було спроєктовано з урахуванням основних сутностей електронної бібліотеки. Вона складається з таких таблиць:

- users – зберігає інформацію про користувачів;
- books – книги, з полем author_ids, яке дозволяє прив’язку до кількох авторів;
- authors – довідник авторів;
- genres – довідник жанрів;
- book_genres – зв’язна таблиця many-to-many між книгами й жанрами;
- favorites – таблиця для обраних книг;
- ratings – рейтинги користувачів;
- comments – коментарі до книг.

Детальну схему зв'язків між таблицями подано на діаграмі нижче (рис. 1)



SQL-структура (витяг з init.sql)

```
CREATE TABLE IF NOT EXISTS books (
  id INTEGER PRIMARY KEY,
  title TEXT NOT NULL,
  author_ids INTEGER[] NOT NULL,
  image TEXT,
  formats TEXT[],
  description TEXT
);

CREATE TABLE IF NOT EXISTS users (
  id SERIAL PRIMARY KEY,
  name TEXT NOT NULL,
  email TEXT UNIQUE NOT NULL,
  password TEXT NOT NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Також реалізовано зв’язки з ON DELETE CASCADE, що дозволяє автоматично видаляти пов’язані записи (наприклад, обране чи коментарі при видаленні книги або користувача).

Таблиця 4.1 Таблиця маршрутів API та їх призначення.

Метод	Шлях	Призначення
GET	/api/books	Отримати список книг з жанрами
GET	/api/authors	Отримати всіх авторів
POST	/api/register	Зареєструвати нового користувача
POST	/api/login	Авторизація

PUT	/api/users/:id	Оновлення профілю користувача
POST	/api/ratings	Додати або оновити рейтинг книги
GET	/api/ratings/:bookId	Середній рейтинг книги
GET	/api/favorites/:userId	Отримати обране користувача

Фронтенд-запити до API:

```
fetch("http://localhost:3000/api/books");
fetch("http://localhost:3000/api/authors");
```

Сервер автоматично об’єднує книги з жанрами, отримуючи дані з таблиць books, book_genres та genres:

```
const booksRes = await pool.query("SELECT * FROM books");
const genresRes = await pool.query(`
  SELECT bg.book_id, g.name
  FROM book_genres bg
  JOIN genres g ON g.id = bg.genre_id
`);
```

Цей підхід забезпечує повноцінну інтеграцію клієнтської частини з базою даних, дозволяючи динамічно керувати контентом та користувачами у вебдодатку.

3.2 Структура клієнтської частини вебдодатку

Клієнтська частина вебдодатку реалізована за допомогою стандартного стеку технологій: HTML, CSS, JavaScript. Вся структура організована у вигляді звичних для фронтенд-проєктів папок: окремо для сторінок, стилів, скриптів, даних і ресурсів. Такий підхід забезпечує зручність розробки, масштабування та читабельність коду.

Загальна структура файлової системи

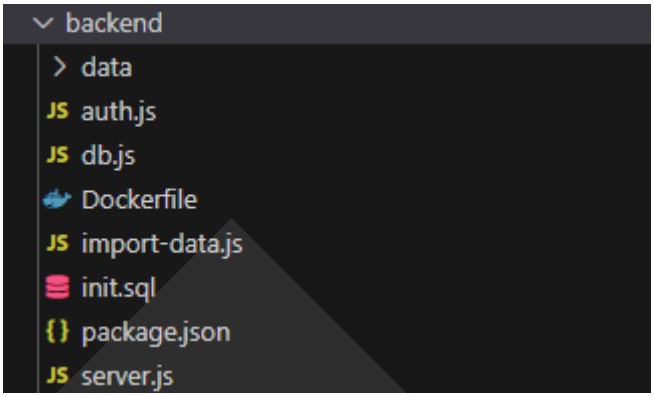


Рисунок 4.1 Структура серверної частини вебдодатку

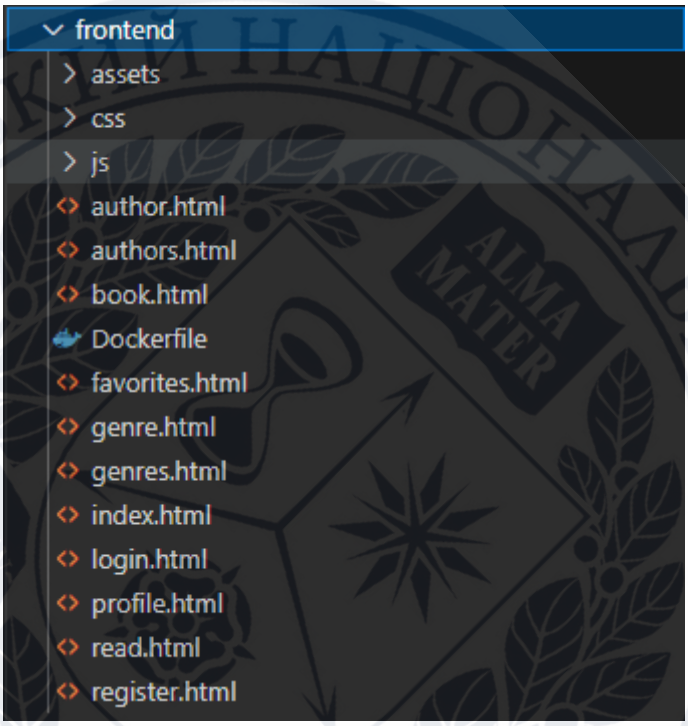


Рисунок 4.2 Структура клієнтської частини вебдодатку

Призначення сторінок

- catalog.html — головна сторінка з каталогом усіх книг. Реалізовано пошук за назвою, автором та фільтрацію за жанром.
- book.html — показує детальну інформацію про обрану книгу: назву, авторів, жанри, опис, рейтинг, коментарі.
- read.html — дозволяє читати книгу онлайн з вибором розміру шрифту.
- favorites.html — сторінка зі списком збережених користувачем обраних книг.
- profile.html — перегляд особистої інформації користувача.

- edit-profile.html — редагування імені, email-адреси та інших даних.
- login.html / register.html — форми для входу та реєстрації користувачів.

Зв’язок сторінок між собою

Кожна сторінка має навігаційне меню у хедері, що дозволяє переміщатися між основними розділами: каталог, обране, профіль. Додатково, за допомогою параметрів URL (book.html?id=1), реалізовано перехід між окремими книгами та сторінками читання.

3.3 Основний функціонал вебінтерфейсу

Каталог книг

Однією з ключових сторінок клієнтської частини вебдодатку є сторінка каталогу книг, яка виконує функцію головного вікна для перегляду наявних публікацій. Вона реалізована на основі HTML-файлу index.html і JavaScript-файлу catalog.js. Користувач може ознайомитися з повним списком книг, здійснювати пошук за назвою, переходити до перегляду детальної інформації про книгу, а також додавати або видаляти книги з обраного.

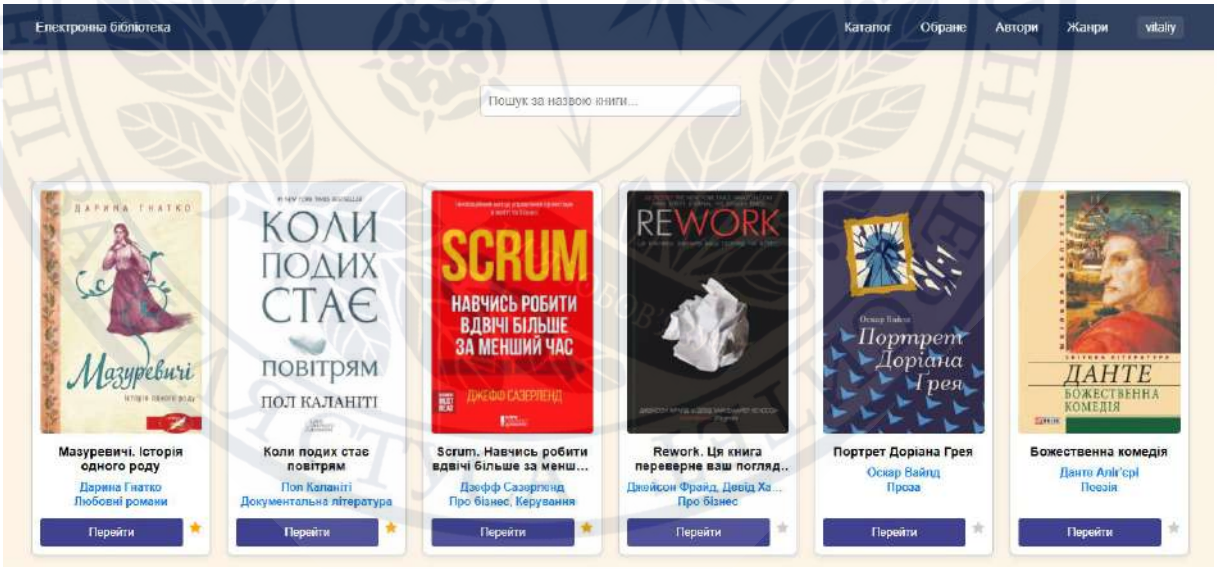


Рисунок 4.3 Інтерфейс сторінки каталогу книг

Сторінка має класичну структуру: у верхній частині знаходиться хедер із назвою вебдодатку та навігаційним меню (посилання на каталог, обране, жанри, автори та профіль користувача або кнопку входу). Нижче розміщено пошукову

стрічку, за допомогою якої користувач може в режимі реального часу фільтрувати список книг за назвою.

У центрі сторінки динамічно генерується сітка книжкових карток, які містять:

- зображення обкладинки книги;
- назву книги (обмежену по висоті в 2 рядки);
- список авторів (з активними посиланнями на сторінки авторів);
- жанри книги (посилання на сторінки жанрів);
- кнопку «Перейти», що веде на сторінку перегляду книги;
- кнопку у вигляді зірочки, яка дозволяє додати або видалити книгу з обраного.

Цей функціонал забезпечується за рахунок JavaScript-файлу catalog.js, у якому реалізована вся логіка обробки даних, фільтрації та взаємодії з сервером.

Пояснення дій користувача:

При відкритті сторінки відбувається автоматичне завантаження списку книг та авторів з API:

```
const [booksRes, authorsRes] = await Promise.all([
  fetch("http://localhost:3000/api/books"),
  fetch("http://localhost:3000/api/authors")
]);
```

Користувач може вводити запит у поле пошуку — функція фільтрує масив книг у реальному часі:

```
document.getElementById("search").addEventListener("input", (e) => {
  const searchTerm = e.target.value.toLowerCase().trim();
  const filtered = books.filter(book =>
    book.title.toLowerCase().includes(searchTerm)
  );
  displayBooks(filtered);
});
```

При натисканні на кнопку зірочки виконується POST-запит на сервер для додавання або видалення книги з обраного:

```
await fetch("http://localhost:3000/api/favorites", {
  method: "POST",
  headers: { "Content-Type": "application/json" },
  body: JSON.stringify({
    user_id: user.id,
    book_id: bookId
```

```
  })
});
```

Назви авторів та жанрів у картці є посиланнями, які ведуть на відповідні сторінки:

```
<a href="author.html?id=3">Автор</a>
<a href="genre.html?name=Фентезі">Фентезі</a>
```

Таким чином, сторінка каталогу реалізує одразу кілька важливих функцій вебдодатку: перегляд контенту, динамічне фільтрування, навігацію до деталей книги, взаємодію з обраним та інтеграцію з API. Її структура проста та інтуїтивно зрозуміла, що підвищує зручність користування ресурсом.

Сторінка книги

Сторінка перегляду книги реалізована у файлі book.html і є однією з найважливіших у вебдодатку. Вона дозволяє користувачу отримати детальну інформацію про конкретну книгу: її назву, авторів, жанр, рейтинг, опис, можливість читати онлайн, завантажити у різних форматах, залишити коментар, оцінити книгу та переглянути рекомендовані книги.

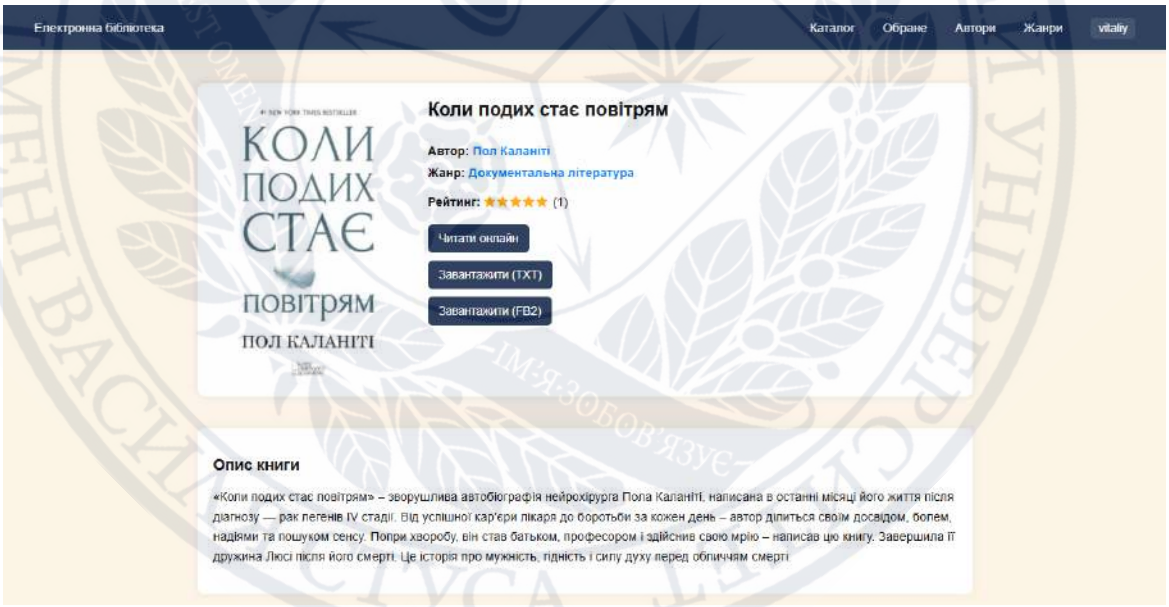


Рисунок 4.4 Інтерфейс сторінки перегляду книги

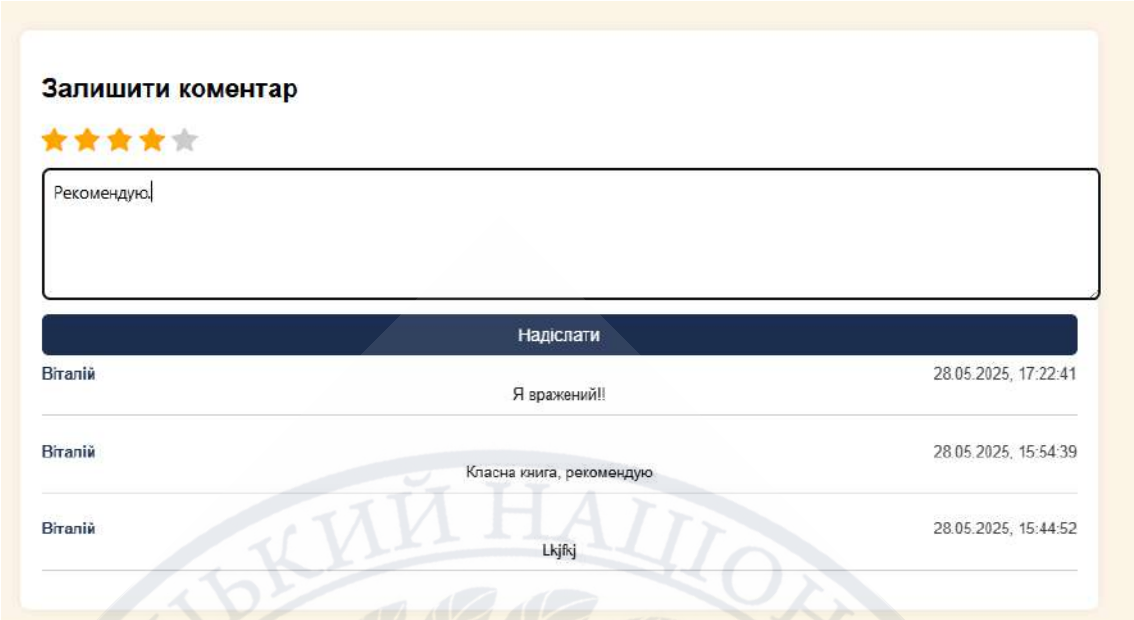


Рисунок 4.5 Форма коментаря

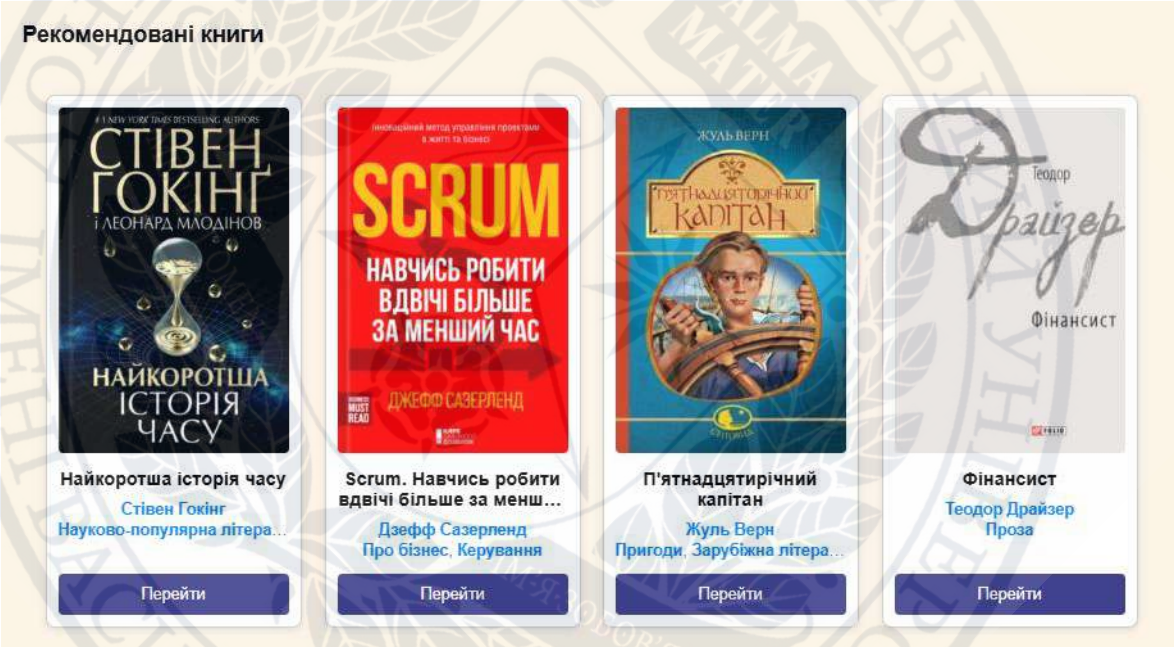


Рисунок 4.6 Рекомендовані книги

Пояснення інтерфейсу та дій користувача:

Динамічне завантаження даних:

Після відкриття сторінки параметр id зчитується з адресного рядка, і виконується запит до API:

```
const bookId = params.get("id");
const booksRes = await fetch("http://localhost:3000/api/books");
const book = books.find(b => String(b.id) === String(bookId));
```

Відображення інформації про книгу:

- Назва, зображення обкладинки
- Автори (з активними посиланнями на author.html)
- Жанри (посилання на genre.html)
- Кнопка «Читати онлайн», що веде на сторінку read.html?id=...
- Кнопки завантаження книги в різних форматах (TXT, FB2)
- Рейтинг книги — середня оцінка зірочками + кількість голосів

Система коментарів та рейтингу:

Авторизований користувач може:

- оцінити книгу за шкалою від 1 до 5 зірок;
- написати текстовий коментар;
- переглянути коментарі інших користувачів (ім’я, дата, текст).

Відправка рейтингу:

```
await fetch("http://localhost:3000/api/ratings", {
  method: "POST",
  headers: { "Content-Type": "application/json" },
  body: JSON.stringify({ user_id, book_id, rating })
});
```

Відправка коментаря:

```
await fetch("http://localhost:3000/api/comments", {
  method: "POST",
  headers: { "Content-Type": "application/json" },
  body: JSON.stringify({ user_id, book_id, text })
});
```

Блок опису книги:

Якщо книга містить розширений опис — він виводиться в окремому блоці нижче основної інформації.

Рекомендовані книги:

У нижній частині сторінки виводяться 4 випадково вибрані книги (крім поточної), що дозволяє користувачу відкривати нові видання без повернення до каталогу.

Функціональні можливості сторінки:

- Завантаження даних через API
- Відображення інформації в адаптивному макеті
- Можливість залишати зворотний зв'язок (рейтинг і коментар)

- Перехід до пов'язаних сторінок (автор, жанр, читання)
- Робота з даними користувача через localStorage (авторизація)

Сторінка для читання

Однією з ключових функцій електронної бібліотеки є можливість читати книгу безпосередньо у веббраузері. Цей функціонал реалізовано на окремій сторінці read.html, яка дозволяє користувачу зручно переглядати текст книги з можливістю регулювання розміру шрифту. Вся логіка завантаження контенту та управління відображенням реалізована у JavaScript-файлі read.js.

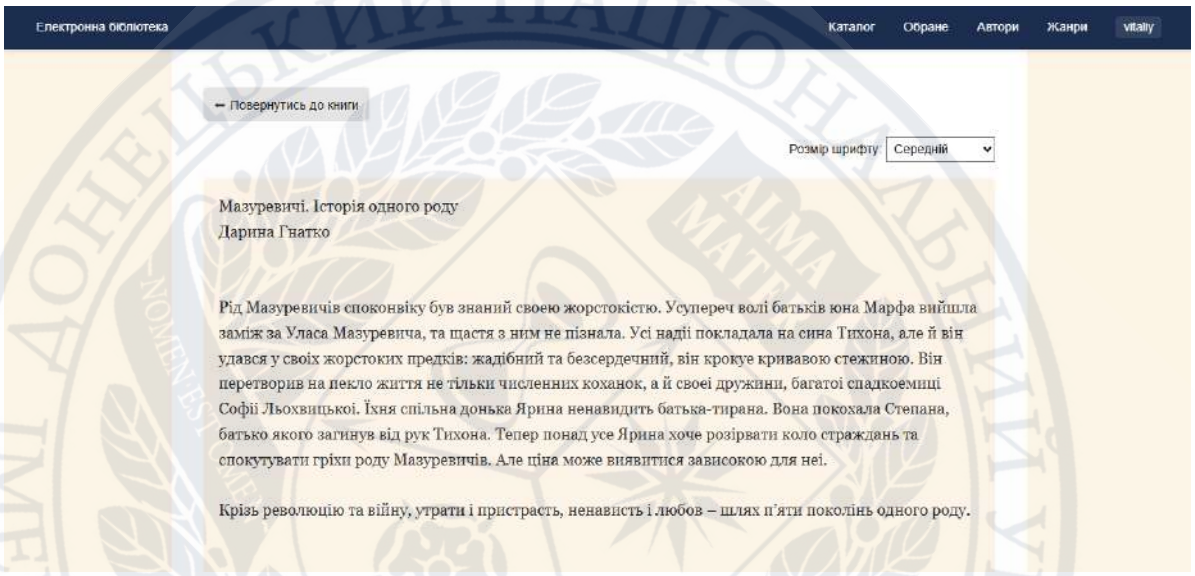


Рисунок 4.7 Інтерфейс сторінки для онлайн читання

Пояснення інтерфейсу та дій користувача:

Завантаження тексту книги:

При переході на сторінку read.html?id=..., параметр id зчитується з URL, після чого виконується запит до API для отримання даних про книгу:

```
const booksRes = await fetch("http://localhost:3000/api/books");
const books = await booksRes.json();
const book = books.find(b => String(b.id) === String(bookId));
```

Далі виконується пошук .txt формату серед доступних форматів книги. Якщо відповідний файл знайдено — він підвантажується через HTTP-запит:

```
const response = await fetch(txtFile);
const text = await response.text();
bookTextDiv.textContent = text;
```

Вивід тексту:

Весь вміст книги виводиться в елемент . Стилізація забезпечує зручність для тривалого читання (відступи, міжрядковість, ширина рядка тощо).

Керування шрифтом:

Користувач може змінити розмір тексту через випадаючий список:

```
<select id="font-size">
  <option value="16px">Маленький</option>
  <option value="20px" selected>Середній</option>
  <option value="24px">Великий</option>
  <option value="28px">Дуже великий</option>
</select>
```

Зміна застосовується динамічно:

```
fontSizeSelect.addEventListener("change", () => {
  bookTextDiv.style.fontSize = fontSizeSelect.value;
});
```

Повернення до книги

У верхній частині сторінки динамічно вставляється посилання, що дозволяє повернутися на сторінку перегляду книги:

```
backLinkDiv.innerHTML = `<a href="book.html?id=${bookId}" class="back-button"> Повернутись до книги</a>`;
```

Особливості реалізації:

- Додаткова перевірка на наявність .txt формату дозволяє уникнути помилок при завантаженні;
- Весь текст книги відображається у звичному для читача форматі;
- Регулювання шрифту забезпечує доступність для людей з різними зоровими потребами;
- Використання асинхронного завантаження робить інтерфейс швидким та зручним.

Обране

Сторінка обраного надає користувачеві можливість швидко переглянути ті книги, які він позначив як улюблені. Цей розділ є важливою частиною функціоналу персоналізації у вебдодатку. Користувач може не лише переглядати свої обрані книги, а й прибирати їх зі списку одним натисканням. Всі дані завантажуються динамічно з бази даних через API.

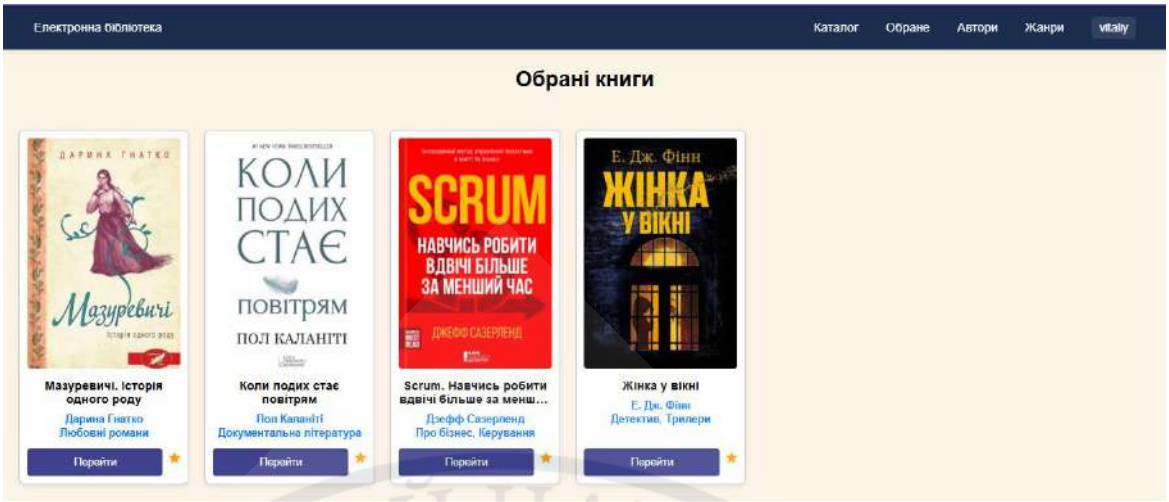


Рисунок 4.8 Інтерфейс сторінки обраного

Пояснення інтерфейсу та дій користувача:

Автоматичне вивантаження обраних книг

При завантаженні сторінки відбувається перевірка авторизації:

```
const user = getCurrentUser();
if (!user) {
  favoritesList.innerHTML = "<p>Щоб переглянути обране, увійдіть у систему.</p>";
  return;
}
```

Якщо користувач авторизований, система виконує запити на отримання книг, авторів та списку обраного:

```
const [booksRes, authorsRes, favRes] = await Promise.all([
  fetch("http://localhost:3000/api/books"),
  fetch("http://localhost:3000/api/authors"),
  fetch(`http://localhost:3000/api/favorites/${user.id}`)
]);
```

Відображення книжкових карток

Усі обрані книги відображаються у вигляді карток, схожих на ті, що використовуються в каталозі. Кожна картка містить:

- зображення обкладинки;
- назву книги (до 2 рядків);
- посилання на автора(ів) та жанри;
- кнопку «Перейти» на сторінку книги;
- кнопку у вигляді зірки для вилучення з обраного.

Вилучення з обраного в один клік

При натисканні на зірочку надсилається POST-запит на той самий API /api/favorites, що автоматично видаляє книгу:

```
const res = await fetch("http://localhost:3000/api/favorites", {
  method: "POST",
  headers: { "Content-Type": "application/json" },
  body: JSON.stringify({
    user_id: user.id,
    book_id: bookId
  })
});
```

У разі успішного запиту — книжкова картка зникає зі сторінки. Якщо не залишилося жодної книги, виводиться повідомлення:

```
favoritesList.innerHTML = "<p>У вас немає обраних книг.</p>";
```

Весь вміст оновлюється без перезавантаження сторінки, що робить взаємодію швидкою та зручною. Дані в обраному залишаються актуальними при кожному новому вході.

3.4 Система автентифікації та управління профілем

Реєстрація та авторизація користувача

- Можливість створення нового облікового запису із зазначенням імені, електронної пошти та пароля.
- Вхід до системи для зареєстрованих користувачів.
- Вихід із системи.

Особистий кабінет користувача

- Перегляд особистої інформації (ПІБ, дата реєстрації, email тощо).
- Редагування профілю (зміна пароля, оновлення контактних даних).
- Перегляд історії переглянутих або завантажених книг.
- Список обраних книг (улюблені).

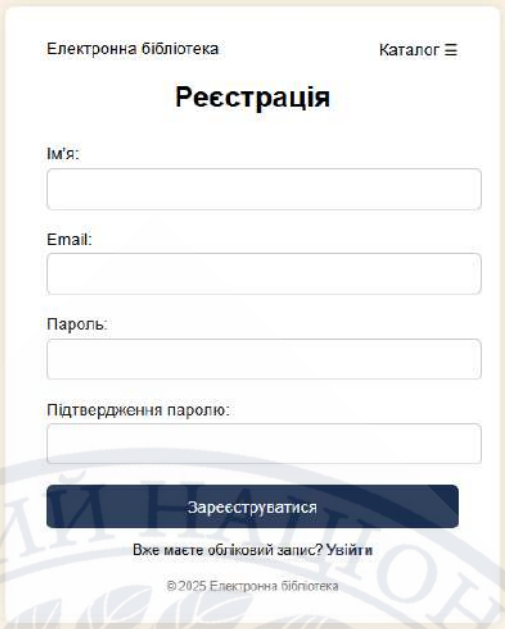


Рисунок 4.9 Форма реєстрації користувача

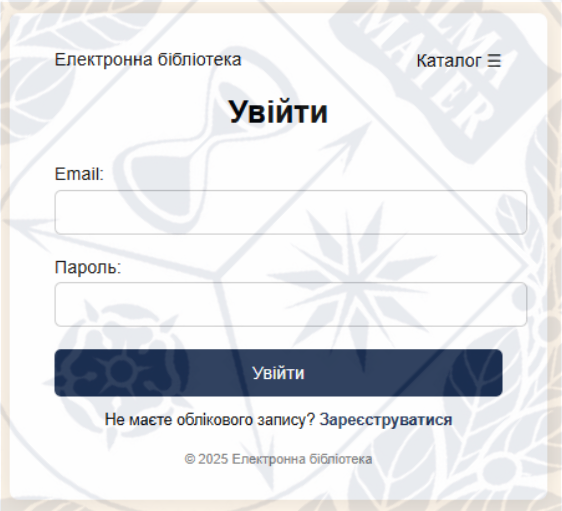


Рисунок 4.10 Форма входу користувача

Пояснення інтерфейсу та дій користувача:

Реєстрація нового користувача

На сторінці register.html реалізована форма з наступними полями:

- Ім’я користувача
- Email
- Пароль
- Підтвердження паролю

Перед відправленням форми перевіряється збіг паролів:

```
if (password !== confirm) {
  alert("Паролі не співпадають.");
  return;
```

```
}
```

Далі викликається функція registerUser() з файлу auth.js, яка надсилає POST-запит до API:

```
await fetch("http://localhost:3000/api/register", {
  method: "POST",
  headers: { "Content-Type": "application/json" },
  body: JSON.stringify({ name, email, password })
});
```

На сервері перевіряється, чи email вже використовується, після чого пароль хешується з використанням bcrypt, і дані записуються в базу:

```
const hashedPassword = await bcrypt.hash(password, 10);
await pool.query(
  "INSERT INTO users (name, email, password) VALUES ($1, $2, $3)",
  [name, email, hashedPassword]
);
```

У разі успішної реєстрації користувач перенаправляється на сторінку входу.

Вхід зареєстрованого користувача

На сторінці login.html реалізовано форму з двома полями — email і пароль. Після натискання кнопки «Увійти» викликається функція loginUser():

```
const res = await fetch("http://localhost:3000/api/login", {
  method: "POST",
  headers: { "Content-Type": "application/json" },
  body: JSON.stringify({ email, password })
});
```

На сервері перевіряється:

1. Чи існує користувач з таким email.
2. Чи співпадає пароль збережений у базі (перевірка bcrypt.compare()).

У разі успіху повертаються основні дані користувача:

```
{ "id": 5, "name": "Vitalii", "email": "vitalii@example.com" }
```

Ці дані зберігаються в localStorage:

```
localStorage.setItem("currentUser", JSON.stringify(user));
```

Після входу користувач автоматично перенаправляється на сторінку каталогу.

Фрагмент коду: auth.js

```
function getCurrentUser() {
  return JSON.parse(localStorage.getItem("currentUser"));
}

function logoutUser() {
  localStorage.removeItem("currentUser");
  window.location.href = "catalog.html";
```

```
}
```

Захист від помилок:

- Повідомлення про зайнятий email або неправильний пароль виводяться через alert().
- Усі запити обгорнуті в try/catch, що дозволяє обробляти ситуації з відсутністю сервера.

Профіль користувача та редагування особистих даних

Функціонал особистого профілю дозволяє авторизованому користувачеві переглядати та змінювати свої особисті дані (ім’я, email), переглядати дату реєстрації, вийти з облікового запису, а також бачити список усіх доданих до обраного книг. Така сторінка є центром взаємодії користувача з персоналізованими можливостями електронної бібліотеки.

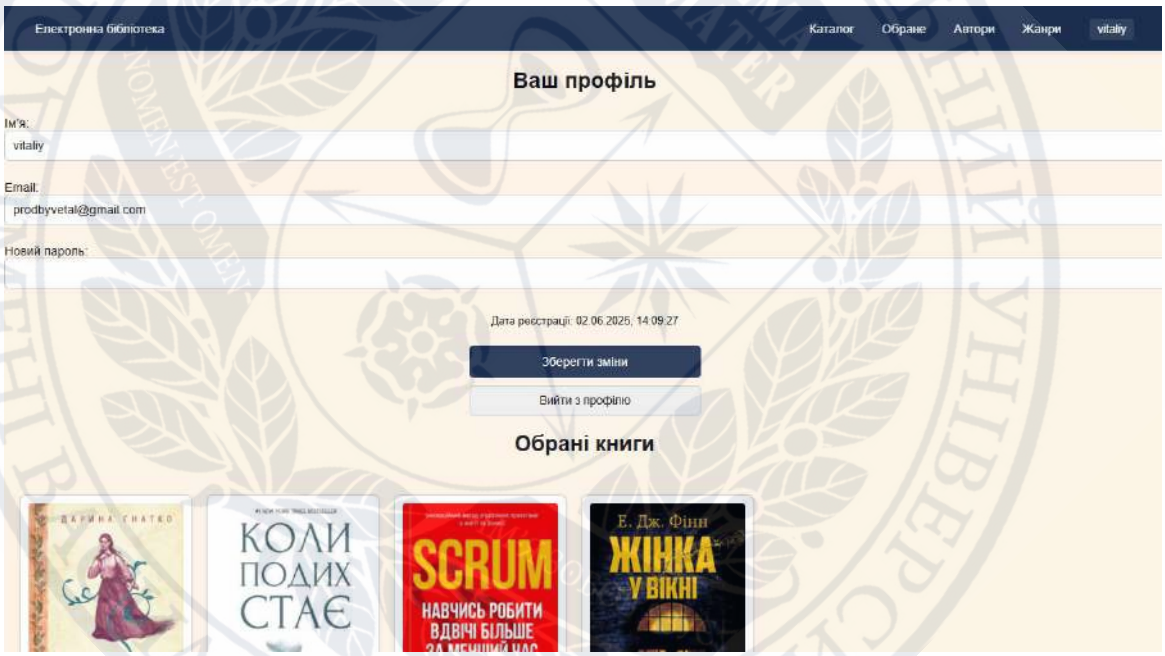


Рисунок 4.11 Інтерфейс сторінки профіля

Пояснення інтерфейсу та дій користувача

Завантаження профільної інформації

При відкритті сторінки автоматично виконується перевірка, чи користувач авторизований. Якщо ні — його перенаправляють на сторінку входу:

```
const user = getCurrentUser();
if (!user) return (window.location.href = "login.html");
```

Ім’я та email заповнюються у відповідні поля форми:

```
document.getElementById("name").value = user.name;
document.getElementById("email").value = user.email;
```

Дата реєстрації завантажується з API:

```
const userData = await fetch(`http://localhost:3000/api/users/${user.id}`).then(res => res.json());
document.getElementById("registration-date").textContent = new Date(userData.created_at).toLocaleString();
```

Редагування особистих даних

Користувач може змінити своє ім’я або email. При натисканні кнопки "Зберегти зміни" виконується PUT-запит до серверу:

```
const res = await fetch(`http://localhost:3000/api/users/${user.id}`, {
  method: "PUT",
  headers: { "Content-Type": "application/json" },
  body: JSON.stringify({ name, email })
});
```

Успішно оновлені дані зберігаються в localStorage:

```
localStorage.setItem("currentUser", JSON.stringify(updated));
```

Вихід з профілю

При натисканні кнопки «Вийти з профілю» викликається функція logoutUser() з auth.js, яка очищає сесію:

```
logoutUser(); // localStorage.clear + redirect
```

Перегляд обраних книг

У нижній частині сторінки відображається сітка всіх книг, доданих у обране. Для цього з API завантажуються:

- всі книги (/api/books)
- усі автори (/api/authors)
- список обраного користувача (/api/favorites/:id)

Після цього відображаються картки книг, аналогічні до каталогу:

```
<div class="book-card">
  <img src="..." />
  <h3>Назва</h3>
  <p>Автори</p>
  <p>Жанри</p>
  <a href="book.html?id=...">Перейти</a>
  <span class="fa fa-star checked"></span>
</div>
```

Користувач може вилучити книгу з обраного, натиснувши на зірочку — картка миттєво зникає.

Отже, було реалізовано клієнтську частину вебдодатку електронної бібліотеки відповідно до попередньо створених макетів у Figma. Реалізовано всі основні функції, зокрема:

- перегляд каталогу книг
- пошук книги
- перегляд основної інформації про окрему книгу (опис, жанр, автор, рейтинг)
- можливість залишити коментар до книги і поставити оцінку
- додавання книг до списку обраного
- онлайн-читання з можливістю зміни розміру шрифту
- реєстрацію, вхід і редагування профілю користувача

Клієнтська частина реалізована засобами HTML, CSS і JavaScript. Взаємодія з даними про книги, авторів та користувачів організована через API-запити.

Для запуску вебдодатку в середовищі розробника використано Docker-контейнери, що забезпечують стабільну роботу сервісів та можливість швидкого розгортання проєкту.

ВИСНОВКИ

У результаті виконання дипломної роботи було досягнуто поставлену мету – розроблено функціональну, адаптивну та зручну клієнтську частину вебдодатку електронної бібліотеки, яка дозволяє користувачам ефективно взаємодіяти з інформаційними ресурсами.

У процесі роботи було здійснено комплексний аналіз предметної області, розглянуто сучасні рішення в галузі електронних бібліотек, визначено їхні переваги та недоліки. На основі цього аналізу були сформульовані функціональні вимоги до інтерфейсу та визначено технічні засоби реалізації.

Клієнтська частина була створена з використанням HTML, CSS і JavaScript, що забезпечило високу швидкість роботи, адаптивність до різних пристроїв і зручний інтерфейс для користувачів. Інтеграція з серверною частиною та базою даних PostgreSQL була реалізована через API, що надало змогу динамічно працювати з книгами, авторами, обраним, рейтингами та профілем користувача. Для розгортання середовища використано Docker, що спростило процес тестування та запуску.

Серед основного функціоналу реалізовано:

- каталог книг з пошуком і фільтрацією;
- реєстрацію та вхід користувача;
- особистий кабінет із редагуванням профілю;
- перегляд детальної інформації про книги;
- онлайн читання з регулюванням розміру шрифту;
- додавання книг до обраного;
- виставлення рейтингу та коментарі до книг.

Оформлення інтерфейсу виконано з урахуванням принципів зручності, логіки навігації та єдиного візуального стилю. Завдяки використанню локального сховища (localStorage) реалізовано збереження сесії та персоналізованих даних без потреби повторної авторизації.

Таким чином, розроблений вебдодаток повністю відповідає поставленим завданням і може бути використаний як основа для впровадження електронної бібліотеки у навчальних закладах або для подальшого розвитку в комерційних та соціальних проєктах.

СПИСОК ВИКОРИСТАНИХ ПОСИЛАНЬ

1. Національна бібліотека України імені В. І. Вернадського. [Електронний ресурс]. URL: http://www.nbuv.gov.ua (дата звернення 22.05.2025)
2. Google Books. [Електронний ресурс]. URL: https://books.google.com (дата звернення 22.05.2025)
3. Project Gutenberg. [Електронний ресурс]. URL: https://www.gutenberg.org (дата звернення 22.05.2025)
4. Open Library. [Електронний ресурс]. URL: https://openlibrary.org (дата звернення 22.05.2025)
5. MDN Web Docs. HTML: HyperText Markup Language. [Електронний ресурс]. URL: https://developer.mozilla.org/en-US/docs/Web/HTML (дата звернення 20.05.2025)
6. MDN Web Docs. CSS: Cascading Style Sheets. [Електронний ресурс]. URL: https://developer.mozilla.org/en-US/docs/Web/CSS (дата звернення 21.05.2025)
7. MDN Web Docs. JavaScript. [Електронний ресурс]. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript (дата звернення 21.05.2025)
8. PostgreSQL documentation. [Електронний ресурс]. – Режим доступу: https://www.postgresql.org/docs/ (дата звернення 27.05.2025)
9. Docker documentation. [Електронний ресурс]. – Режим доступу: https://docs.docker.com/ (дата звернення 27.05.2025)
10. Figma. Офіційний сайт для створення макетів інтерфейсів. [Електронний ресурс]. URL : https://www.figma.com/ (дата звернення 20.05.2025)
11. Visual Studio Code. Редактор коду. [Електронний ресурс]. URL: https://code.visualstudio.com/ (дата звернення 18.05.2025)
12. JSON: JavaScript Object Notation. [Електронний ресурс]. URL: https://www.json.org/json-en.html (дата звернення 20.05.2025)

13. W3Schools Online Web Tutorials. [Електронний ресурс]. URL: https://www.w3schools.com/ (дата звернення 20.05.2025)
14. Freecodecamp. Learn to code for free. [Електронний ресурс]. URL: https://www.freecodecamp.org/ (дата звернення 21.05.2025)
15. Mozilla Developer Network. Web APIs. [Електронний ресурс]. URL: https://developer.mozilla.org/en-US/docs/Web/API (дата звернення 20.05.2025)
16. DigitalOcean Tutorials. [Електронний ресурс]. URL: https://www.digitalocean.com/community/tutorials (дата звернення 20.05.2025)
17. DevDocs API Documentation Browser. [Електронний ресурс]. URL: https://devdocs.io/ (дата звернення 19.05.2025)
18. Stack Overflow. [Електронний ресурс]. URL: https://stackoverflow.com/ (дата звернення 26.05.2025)
19. CSS-Tricks. Поради, приклади та довідники з веброзробки. [Електронний ресурс]. URL: https://css-tricks.com/ (дата звернення 26.05.2025)
20. Чумак Є. Електронні бібліотеки як вагомий складник національного інформаційного простору України. Вісник Книжкової палати, 2023, №5. [Електронний ресурс]. URL: https://libkor.com.ua/storage/php/periodic_theme_files/elektronni_biblioteki.pdf (дата звернення 15.05.2025)
21. Бібліотека в системі наукової електронної комунікації. [Електронний ресурс]. URL: https://www.nbuv.gov.ua/sites/default/files/msd/0710kop.pdf (дата звернення 15.05.2025)
22. World Digital Library. [Електронний ресурс]. URL: https://www.wdl.org/ (дата звернення 22.05.2025)
23. Building a Frontend for a REST API using JavaScript. [Електронний ресурс]. URL: https://medium.com/@devrmichael/rest-apis-for-

frontend-developers-a-simple-guide-83074731e600 (дата звернення 22.05.2025)

24. pgAdmin. Офіційний сайт інструменту адміністрування PostgreSQL. [Електронний ресурс]. URL: https://www.pgadmin.org/ (дата звернення 27.05.2025)
25. PostgreSQL Tutorial. [Електронний ресурс]. URL: https://www.w3schools.com/postgresql/postgresql_pgadmin4.php (дата звернення 27.05.2025)

ДОДАТКИ

ДОДАТОК А

HTML файли

index.html

```
<!DOCTYPE html>
<html lang="uk">
<head>
  <meta charset="UTF-8" />
  <title>Каталог книг — Електронна бібліотека</title>
  <link rel="stylesheet" href="css/style.css" />
  <script src="js/catalog.js" defer></script>
  <script src="js/auth.js" defer></script>
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/4.7.0/css/font-awesome.min.css">
</head>
<body>
  <header class="main-header">
    <div class="logo-and-toggle">
      <div class="logo">Електронна бібліотека</div>
      <div class="menu-toggle" id="menu-toggle">&#9776;</div>
    </div>
    <nav>
      <a href="index.html">Каталог</a>
      <a href="favorites.html">Обране</a>
      <a href="authors.html">Автори</a>
      <a href="genres.html">Жанри</a>
      <span id="user-area"></span>
    </nav>
  </header>
  <div class="search-wrapper" style="text-align: center; margin: 40px auto;">
    <input type="text" id="search" placeholder="Пошук за назвою книги..."
style="width: 300px;font-size: 16px;" />
  </div>
  <main class="book-grid" id="book-list">
    <!-- Картки книг генеруються скриптом -->
  </main>
  <div id="pagination" class="pagination"></div>
  <footer class="footer">
    © 2025 Електронна бібліотека
  </footer>
</body>
</html>
```

book.html

```
<!DOCTYPE html>
<html lang="uk">
<head>
  <meta charset="UTF-8" />
  <title>Книга — Електронна бібліотека</title>
  <link rel="stylesheet" href="css/style.css" />
  <script src="js/auth.js" defer></script>
  <script src="js/book.js" defer></script>
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/4.7.0/css/font-awesome.min.css">
</head>
<body>
  <header class="main-header">
    <div class="logo">Електронна бібліотека</div>
    <nav>
      <a href="index.html">Каталог</a>
```

```
      <a href="favorites.html">Обране</a>
      <a href="authors.html">Автори</a>
      <a href="genres.html">Жанри</a>
      <span id="user-area"></span>
    </nav>
  </header>

  <main class="book-page">
    <div id="book-container">
    </div>
  </main>

  <footer class="footer">
    © 2025 Електронна бібліотека
  </footer>
</body>
</html>
```

read.html

```
<!DOCTYPE html>
<html lang="uk">
<head>
  <meta charset="UTF-8" />
  <title>Читати книгу</title>
  <link rel="stylesheet" href="css/style.css" />
  <script src="js/auth.js" defer></script>
  <script src="js/read.js" defer></script>
</head>
<body>
  <header class="main-header">
    <div class="logo">Електронна бібліотека</div>
    <nav>
      <a href="index.html">Каталог</a>
      <a href="favorites.html">Обране</a>
      <a href="authors.html">Автори</a>
      <a href="genres.html">Жанри</a>
      <span id="user-area"></span>
    </nav>
  </header>

  <main class="reader">
    <div class="back-to-book" id="back-link">
    </div>
    <div class="reader-controls">
      <label>Розмір шрифту:
        <select id="font-size">
          <option value="16px">Маленький</option>
          <option value="20px" selected>Середній</option>
          <option value="24px">Великий</option>
          <option value="28px">Дуже великий</option>
        </select>
      </label>
    </div>
    <div id="book-text" class="book-text">
    </div>
  </main>
  <footer class="footer">
    © 2025 Електронна бібліотека
  </footer>
</body>
</html>
```

Favorites.html

```
<!DOCTYPE html>
```

```
<html lang="uk">
<head>
  <meta charset="UTF-8">
  <title>Обрані книги</title>
  <link rel="stylesheet" href="css/style.css">
  <script src="js/favorites.js" defer></script>
  <script src="js/auth.js" defer></script>
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/4.7.0/css/font-awesome.min.css">
</head>
<body>
  <header class="main-header">
    <nav>
      <a href="index.html">Каталог</a>
      <a href="favorites.html">Обране</a>
      <a href="authors.html">Автори</a>
      <a href="genres.html">Жанри</a>
      <span id="user-area"></span>
    </nav>
  </header>

  <main>
    <h1>Обрані книги</h1>
    <div class="book-grid" id="favorites-list">
    </div>
  </main>

  <footer class="footer">
    © 2025 Електронна бібліотека
  </footer>
</body>
</html>
```

Profile.html

```
<!DOCTYPE html>
<html lang="uk">
<head>
  <meta charset="UTF-8">
  <title>Профіль</title>
  <link rel="stylesheet" href="css/style.css">
  <script src="js/auth.js" defer></script>
  <script src="js/profile.js" defer></script>
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/4.7.0/css/font-awesome.min.css">
</head>
<body>
  <header class="main-header">
    <div class="logo">Електронна бібліотека</div>
    <nav>
      <a href="index.html">Каталог</a>
      <a href="favorites.html">Обране</a>
      <a href="authors.html">Автори</a>
      <a href="genres.html">Жанри</a>
      <span id="user-area"></span>
    </nav>
  </header>
  <main class="profile-page">
    <section class="profile-info">
      <h1>Ваш профіль</h1>
      <form id="profile-form">
        <label>Ім’я: <input type="text" id="name" required></label>
        <label>Email: <input type="email" id="email" required></label>
        <label>Новий пароль: <input type="password" id="password"></label>
        <p>Дата реєстрації: <span id="registration-date"></span></p>
```

```
      </form>
      <div style="display: flex; flex-direction: column; align-items: center; ">
        <button type="submit" class="primary-btn">Зберегти зміни</button>
        <button id="logout-btn" class="secondary-btn">Вийти з профілю</button>
      </div>
    </section>
    <section class="favorites-section">
      <h1>Обрані книги</h1>
      <div id="favorites-list" class="book-grid"></div>
    </section>
  </main>
</body>
</html>
```

Login.html

```
<!DOCTYPE html>
<html lang="uk">
<head>
  <meta charset="UTF-8" />
  <title>Увійти — Електронна бібліотека</title>
  <link rel="stylesheet" href="css/style.css" />
  <script src="js/auth.js" defer></script>
  <script defer>
    document.addEventListener("DOMContentLoaded", () => {
      document.getElementById("login-form").addEventListener("submit", async (e)
=> {
        e.preventDefault();
        const email = document.getElementById("email").value.trim();
        const password = document.getElementById("password").value;

        const success = await loginUser(email, password);
        if (success) {
          window.location.href = "catalog.html";
        }
      });
    });
  </script>
</head>
<body>
  <div class="container">
    <header>
      <div>Електронна бібліотека</div>
      <a href="index.html">Каталог ☰</a>
    </header>
    <h1>Увійти</h1>
    <form id="login-form">
      <label for="email">Email:</label>
      <input type="email" id="email" required />
      <label for="password">Пароль:</label>
      <input type="password" id="password" required />
      <button type="submit">Увійти</button>
    </form>
    <p>Не маєте облікового запису? <a
href="register.html">Зареєструватися</a></p>
    <footer>© 2025 Електронна бібліотека</footer>
  </div>
</body>
</html>
```

register.html

```
<!DOCTYPE html>
<html lang="uk">
<head>
  <meta charset="UTF-8" />
```

```
  <title>Реєстрація — Електронна бібліотека</title>
  <link rel="stylesheet" href="css/style.css" />
  <script src="js/auth.js" defer></script>
  <script defer>
    document.addEventListener("DOMContentLoaded", () => {
      document.getElementById("register-form").addEventListener("submit", async
(e) => {
        e.preventDefault();

        const name = document.getElementById("name").value.trim();
        const email = document.getElementById("email").value.trim();
        const password = document.getElementById("password").value;
        const confirm = document.getElementById("confirm").value;

        if (password !== confirm) {
          alert("Паролі не співпадають.");
          return;
        }

        const success = await registerUser(name, email, password);
        if (success) {
          window.location.href = "login.html";
        }
      });
    });
  </script>
</head>
<body>
  <div class="container">
    <header>
      <div>Електронна бібліотека</div>
      <a href="index.html">Каталог ☰</a>
    </header>

    <h1>Реєстрація</h1>
    <form id="register-form">
      <label for="name">Ім’я:</label>
      <input type="text" id="name" required />

      <label for="email">Email:</label>
      <input type="email" id="email" required />

      <label for="password">Пароль:</label>
      <input type="password" id="password" required />

      <label for="confirm">Підтвердження паролю:</label>
      <input type="password" id="confirm" required />

      <button type="submit">Зареєструватися</button>
    </form>

    <p>Вже маєте обліковий запис? <a href="login.html">Увійти</a></p>

    <footer>© 2025 Електронна бібліотека</footer>
  </div>
</body>
</html>
```

Author.html

```
<!DOCTYPE html>
<html lang="uk">
<head>
  <meta charset="UTF-8" />
  <title>Автор — Електронна бібліотека</title>
  <link rel="stylesheet" href="css/style.css" />
  <script src="js/auth.js" defer></script>
  <script src="js/author.js" defer></script>
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/4.7.0/css/font-awesome.min.css">
</head>
<body>
  <header class="main-header">
    <div class="logo">Електронна бібліотека</div>
    <nav>
      <a href="index.html">Каталог</a>
      <a href="favorites.html">Обране</a>
      <a href="authors.html">Автори</a>
      <a href="genres.html">Жанри</a>
      <span id="user-area"></span>
    </nav>
  </header>

    <main class="author-page">
    <div class="author-card">
        <img id="author-image" src="" alt="Фото автора" class="author-img" />
        <div class="author-info">
        <h1 id="author-name"></h1>
        <p><strong>Жанри:</strong> <span id="author-genres"></span></p>
        <p id="author-description"></p>
        </div>
    </div>

    <section class="author-books">
        <h2>Книги автора</h2>
        <div id="book-list"></div>
    </section>
    </main>
  </main>
</body>
</html>
```

Genre.html

```
<!DOCTYPE html>
<html lang="uk">
<head>
  <meta charset="UTF-8">
  <title>Книги за жанром</title>
  <link rel="stylesheet" href="css/style.css">
  <script src="js/auth.js" defer></script>
  <script src="js/genre.js" defer></script>
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/4.7.0/css/font-awesome.min.css">
</head>
<body>
  <header class="main-header">
    <div class="logo">Електронна бібліотека</div>
    <nav>
      <a href="index.html">Каталог</a>
      <a href="favorites.html">Обране</a>
      <a href="authors.html">Автори</a>
      <a href="genres.html">Жанри</a>
      <span id="user-area"></span>
```

```
    </nav>
  </header>

  <main>
    <h1 id="genre-title">Жанр</h1>
    <div id="book-list" class="book-grid"></div>
  </main>
</body>
</html>
```

ДОДАТОК Б
JavaScript-файли

auth.js

```
function setCurrentUser(user) {
  localStorage.setItem("currentUser", JSON.stringify(user));
}

function getCurrentUser() {
  return JSON.parse(localStorage.getItem("currentUser"));
}

function logoutUser() {
  localStorage.removeItem("currentUser");
  window.location.href = "catalog.html";
}

async function registerUser(name, email, password) {
  try {
    const res = await fetch("http://localhost:3000/api/register", {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({ name, email, password })
    });

    if (!res.ok) {
      const data = await res.json();
      alert(data.message || "Помилка реєстрації");
      return false;
    }

    alert("Реєстрація успішна! Тепер можете увійти.");
    return true;
  } catch (err) {
    console.error("Помилка реєстрації:", err);
    alert("Сервер недоступний");
    return false;
  }
}

async function loginUser(email, password) {
  try {
    const res = await fetch("http://localhost:3000/api/login", {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({ email, password })
    });

    if (!res.ok) {
      const data = await res.json();
      alert(data.message || "Невірні дані для входу");
      return false;
    }

    const user = await res.json();
    setCurrentUser(user);
    return true;
  } catch (err) {
    console.error("Помилка входу:", err);
    alert("Сервер недоступний");
    return false;
  }
}
```

```
async function updateProfile(userId, name, email) {
  try {
    const res = await fetch(`http://localhost:3000/api/users/${userId}`, {
      method: "PUT",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({ name, email })
    });

    if (!res.ok) {
      const data = await res.json();
      alert(data.message || "Помилка оновлення профілю");
      return false;
    }

    const updatedUser = await res.json();
    setCurrentUser(updatedUser);
    alert("Профіль оновлено успішно!");
    return true;
  } catch (err) {
    console.error("Помилка оновлення профілю:", err);
    alert("Сервер недоступний");
    return false;
  }
}
```

author.js

```
document.addEventListener("DOMContentLoaded", async () => {
  const userArea = document.getElementById("user-area");
  const authorNameEl = document.getElementById("author-name");
  const authorDescEl = document.getElementById("author-description");
  const authorGenresEl = document.getElementById("author-genres");
  const authorImageEl = document.getElementById("author-image");
  const bookList = document.getElementById("book-list");

  const user = getCurrentUser();
  userArea.innerHTML = user
    ? `<a href="profile.html">${user.name}</a>`
    : `<a href="login.html">Увійти</a>`;

  const params = new URLSearchParams(location.search);
  const authorId = Number(params.get("id"));

  const [booksRes, authorsRes] = await Promise.all([
    fetch("http://localhost:3000/api/books"),
    fetch("http://localhost:3000/api/authors")
  ]);

  const books = await booksRes.json();
  const authors = await authorsRes.json();

  const author = authors.find(a => a.id === authorId);
  if (!author) {
    authorNameEl.textContent = "Автор не знайдений";
    return;
  }

  authorNameEl.textContent = author.name;
  authorDescEl.textContent = author.description || "Опис відсутній.";
  authorImageEl.src = author.image || "assets/images/author-placeholder.jpg";
  const authorBooks = books.filter(book =>
    Array.isArray(book.author_ids)
      ? book.author_ids.includes(authorId)
      : book.author_ids === authorId
```

```
  );

  const genresSet = new Set();
  authorBooks.forEach(book => {
    const genres = Array.isArray(book.genre) ? book.genre : [book.genre];
    genres.forEach(g => genresSet.add(g));
  });

  authorGenresEl.innerHTML = [...genresSet]
    .map(g => `<a href="genre.html?name=${encodeURIComponent(g)}">${g}</a>`)
    .join(", ");
  let favoriteIds = [];
  if (user) {
    const favRes = await
fetch(`http://localhost:3000/api/favorites/${user.id}`);
    const favData = await favRes.json();
    favoriteIds = favData.map(b => b.id);
  }

  displayBooks(authorBooks, authors, favoriteIds, user);
});
function displayBooks(books, authors, favoriteIds, user) {
  const bookList = document.getElementById("book-list");
  bookList.innerHTML = "";
  books.forEach(book => {
    const authorIds = Array.isArray(book.author_ids) ? book.author_ids :
[book.author_ids];
    const matchedAuthors = authors.filter(a => authorIds.includes(a.id));
    const isFav = favoriteIds.includes(book.id);
    const card = document.createElement("div");
    card.className = "book-card";
    card.innerHTML = `
      <div class="book-info">
        <a href="book.html?id=${book.id}">
          <img src="${book.image}" alt="${book.title}" />
        </a>
        <h3 class="clamp-2">${book.title}</h3>
        <p class="genre clamp-1">
          ${Array.isArray(book.genre)
            ? book.genre.map(g => `<a
href="genre.html?name=${encodeURIComponent(g)}">${g}</a>`).join(", ")
            : `<a
href="genre.html?name=${encodeURIComponent(book.genre)}">${book.genre}</a>`}
        </p>
      </div>
      <div class="actions">
        <a href="book.html?id=${book.id}" class="download-btn">Перейти</a>
        <span class="fav-btn fa fa-star${isFav ? " checked" : ""}" data-
id="${book.id}" title="Обране"></span>
      </div>
    `;
    bookList.appendChild(card);
  });
  document.querySelectorAll(".fav-btn").forEach(btn => {
    btn.addEventListener("click", async () => {
      const bookId = +btn.dataset.id;

      await fetch("http://localhost:3000/api/favorites", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({
          user_id: user.id,
          book_id: bookId
        })
      });
```

```
      btn.classList.toggle("checked");
    });
  });
}
```

book.js

```
document.addEventListener("DOMContentLoaded", async () => {
  const params = new URLSearchParams(window.location.search);
  const bookId = params.get("id");

  const userArea = document.getElementById("user-area");
  const user = getCurrentUser();
  userArea.innerHTML = user
    ? `<a href="profile.html">${user.name}</a>`
    : `<a href="login.html">Увійти</a>`;

  const [booksRes, authorsRes] = await Promise.all([
    fetch("http://localhost:3000/api/books"),
    fetch("http://localhost:3000/api/authors")
  ]);

  const books = await booksRes.json();
  const authors = await authorsRes.json();

  const book = books.find(b => String(b.id) === String(bookId));
  const container = document.getElementById("book-container");

  if (!book) {
    container.innerHTML = "<p>Книгу не знайдено.</p>";
    return;
  }

  const authorIds = Array.isArray(book.author_ids) ? book.author_ids :
[book.author_ids];
  const matchedAuthors = authors.filter(a => authorIds.includes(a.id));

  const formats = book.formats || [];
  const formatLinks = formats.map(format => {
    const ext = format.split('.').pop().toUpperCase();
    return `<a href="${format}" download class="format-link">Завантажити
(${ext})</a>`;
  }).join(" ");
  const ratingRes = await fetch(`http://localhost:3000/api/ratings/${bookId}`);
  const ratingData = await ratingRes.json();
  container.innerHTML = `
    <div class="book-detail book-mockup">
      <img src="${book.image}" alt="${book.title}" />
      <div class="book-detail-info">
        <h1>${book.title}</h1>
        <p class="author clamp-1">
          <strong>Автор:</strong>
          ${matchedAuthors.map(a => `<a
href="author.html?id=${a.id}">${a.name}</a>`).join(", ")}
        </p>
        <p class="genre clamp-1">
          <strong>Жанр:</strong>
          ${Array.isArray(book.genre)
            ? book.genre.map(g => `<a
href="genre.html?name=${encodeURIComponent(g)}">${g}</a>`).join(", ")
            : `<a
href="genre.html?name=${encodeURIComponent(book.genre)}">${book.genre}</a>`}
        </p>
        <div class="book-rating">
          <strong>Рейтинг:</strong>
```

```
          <span class="stars-display">${getStarIcons(ratingData.average)}</span>
          <span>(${ratingData.votes})</span>
        </div>
        <div class="actions">
          <a href="read.html?id=${book.id}" class="read-btn">Читати онлайн</a>
          ${formatLinks}
        </div>
      </div>
    </div>
  `;

  if (book.description) {

    const descBlock = document.createElement("div");
    descBlock.className = "book-description";
    descBlock.innerHTML = `
      <h2>Опис книги</h2>
      <p>${book.description}</p>
    `;
    container.appendChild(descBlock);
  }

  const commentSection = document.createElement("section");
  commentSection.className = "comments-section";
  commentSection.innerHTML = `
    <h2>Залишити коментар</h2>
    <form id="comment-form">
      <div class="rating-stars" id="comment-stars">
        <span class="fa fa-star" data-value="1"></span>
        <span class="fa fa-star" data-value="2"></span>
        <span class="fa fa-star" data-value="3"></span>
        <span class="fa fa-star" data-value="4"></span>
        <span class="fa fa-star" data-value="5"></span>
      </div>
      <textarea id="comment-text" placeholder="Ваш коментар..."
required></textarea>
      <button type="submit">Надіслати</button>
    </form>
    <div id="comments-list"></div>
  `;

  let selectedRating = 0;
  container.appendChild(commentSection);

  document.querySelectorAll("#comment-stars .fa-star").forEach(star => {
    star.addEventListener("click", () => {
      selectedRating = +star.dataset.value;
      document.querySelectorAll("#comment-stars .fa-star").forEach(s => {
        s.classList.toggle("checked", +s.dataset.value <= selectedRating);
      });
    });
  });


  async function loadComments() {
    const res = await fetch(`http://localhost:3000/api/comments/${bookId}`);
    const comments = await res.json();
    const list = document.getElementById("comments-list");
    list.innerHTML = comments.map(c => `
      <div class="comment">
        <div class="comment-header">
          <a href="profile.html?id=${c.user_id}">${c.user_name}</a>
          <span>${new Date(c.created_at).toLocaleString()}</span>
        </div>
        <p>${c.text}</p>
```

```
      </div>
    `).join("");

  }

  document.getElementById("comment-form")?.addEventListener("submit", async (e)
=> {
    e.preventDefault();
    const text = document.getElementById("comment-text").value.trim();
    if (!text || !user) return;

    if (selectedRating > 0) {
      await fetch("http://localhost:3000/api/ratings", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({
          user_id: user.id,
          book_id: bookId,
          rating: selectedRating
        })
      });
    }

    await fetch("http://localhost:3000/api/comments", {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({
        user_id: user.id,
        book_id: bookId,
        text
      })
    });
    document.getElementById("comment-text").value = "";
    loadComments();
  });

  function getStarIcons(rating) {
    const fullStars = Math.floor(rating);
    const hasHalfStar = rating % 1 >= 0.25 && rating % 1 <= 0.75;
    const stars = [];

    for (let i = 0; i < fullStars; i++) {
      stars.push('<span class="fa fa-star checked"></span>');
    }

    if (hasHalfStar) {
      stars.push('<span class="fa fa-star-half-o checked"></span>');
    }

    const emptyStars = 5 - stars.length;
    for (let i = 0; i < emptyStars; i++) {
      stars.push('<span class="fa fa-star"></span>');
    }

    return stars.join("");
  }

  loadComments();

  const recommendedBlock = document.createElement("section");
  recommendedBlock.className = "recommended-books";
  recommendedBlock.innerHTML = `<h2>Рекомендовані книги</h2>`;

  const recommendedGrid = document.createElement("div");
  recommendedGrid.className = "book-grid";
```

```
  const otherBooks = books.filter(b => b.id !== book.id);
  const shuffled = otherBooks.sort(() => 0.5 - Math.random());
  const recommended = shuffled.slice(0, 4);

  recommended.forEach(b => {
    const authorIds = Array.isArray(b.author_ids) ? b.author_ids :
[b.author_ids];
    const matchedAuthors = authors.filter(a => authorIds.includes(a.id));

    const card = document.createElement("div");
    card.className = "book-card";
    card.innerHTML = `
      <div class="book-info">
        <a href="book.html?id=${b.id}">
          <img src="${b.image}" alt="${b.title}" />
        </a>
        <h3 class="clamp-2">${b.title}</h3>
        <p class="author clamp-1">
          ${matchedAuthors.map(a => `<a
href="author.html?id=${a.id}">${a.name}</a>`).join(", ")}
        </p>
        <p class="genre clamp-1">
          ${Array.isArray(b.genre)
            ? b.genre.map(g => `<a
href="genre.html?name=${encodeURIComponent(g)}">${g}</a>`).join(", ")
            : `<a
href="genre.html?name=${encodeURIComponent(b.genre)}">${b.genre}</a>`}
        </p>
      </div>
      <div class="actions">
        <a href="book.html?id=${b.id}" class="download-btn">Перейти</a>
      </div>
    `;
    recommendedGrid.appendChild(card);
  });

  recommendedBlock.appendChild(recommendedGrid);
  container.appendChild(recommendedBlock);

});
```

catalog.js

```
document.addEventListener("DOMContentLoaded", async () => {
  const bookList = document.getElementById("book-list");
  const userArea = document.getElementById("user-area");


  let books = [];
  let authors = [];
  let favoriteIds = [];

  async function loadFavorites() {
    if (!user) return;
    const res = await fetch(`http://localhost:3000/api/favorites/${user.id}`);
    const data = await res.json();
    favoriteIds = data.map(b => b.id); // id книг
  }

  const user = getCurrentUser();
  userArea.innerHTML = user
    ? `<a href="profile.html">${user.name}</a>`
    : `<a href="login.html">Увійти</a>`;
```

```
  async function loadData() {
    const [booksRes, authorsRes] = await Promise.all([
      fetch("http://localhost:3000/api/books"),
      fetch("http://localhost:3000/api/authors")
    ]);
    books = await booksRes.json();
    console.log("Завантажені книги:", books);
    authors = await authorsRes.json();
    await loadFavorites();
    displayBooks(books);
  }

  function displayBooks(list) {
    bookList.innerHTML = "";

    list.forEach(book => {
      const authorIds = Array.isArray(book.author_ids) ? book.author_ids :
[book.author_ids];
      const matchedAuthors = authors.filter(a => authorIds.includes(a.id));
      const authorNames = matchedAuthors.map(a => a.name).join(", ");
      const isFav = isFavorite(book.id);

      const card = document.createElement("div");
      card.className = "book-card";
      card.innerHTML = `
        <div class="book-info">
          <a href="book.html?id=${book.id}">
            <img src="${book.image}" alt="${book.title}" />
          </a>
          <h3 class="clamp-2">${book.title}</h3>
          <p class="author clamp-1">
            ${matchedAuthors.map(a => `<a
href="author.html?id=${a.id}">${a.name}</a>`).join(", ")}
          </p>
          <p class="genre clamp-1">
            ${Array.isArray(book.genre)
              ? book.genre.map(g => `<a
href="genre.html?name=${encodeURIComponent(g)}">${g}</a>`).join(", ")
              : `<a
href="genre.html?name=${encodeURIComponent(book.genre)}">${book.genre}</a>`}
          </p>

        </div>
        <div class="actions">
          <a href="book.html?id=${book.id}" class="download-btn">Перейти</a>
          <span class="fav-btn fa fa-star${isFav ? " checked" : ""}" data-
id="${book.id}" title="Обране"></span>
        </div>
      `;
      bookList.appendChild(card);
    });

    document.querySelectorAll(".fav-btn").forEach(btn => {
      btn.addEventListener("click", async () => {
        const bookId = +btn.dataset.id;

        await fetch("http://localhost:3000/api/favorites", {
          method: "POST",
          headers: { "Content-Type": "application/json" },
          body: JSON.stringify({
            user_id: user.id,
            book_id: bookId
          })
        });
```

```
        btn.classList.toggle("checked");
      });
    });
  }

  function isFavorite(bookId) {
    return favoriteIds.includes(bookId);
  }


  loadData().then(() => {
    const params = new URLSearchParams(window.location.search);
    const searchTerm = params.get("search")?.toLowerCase().trim();

    if (searchTerm) {
      const filtered = books.filter(book =>
        book.title.toLowerCase().includes(searchTerm)
      );
      displayBooks(filtered);
    } else {
      displayBooks(books);
    }
    document.getElementById("search").addEventListener("input", (e) => {
    const searchTerm = e.target.value.toLowerCase().trim();
    const filtered = books.filter(book =>
      book.title.toLowerCase().includes(searchTerm)
    );
    displayBooks(filtered);
  });
  });
});
```

favorites.js

```
document.addEventListener("DOMContentLoaded", async () => {
  const favoritesList = document.getElementById("favorites-list");
  const userArea = document.getElementById("user-area");
  const user = getCurrentUser();

  userArea.innerHTML = user
    ? `<a href="profile.html">${user.name}</a>`
    : `<a href="login.html">Увійти</a>`;

  if (!user) {
    favoritesList.innerHTML = "<p>Щоб переглянути обране, увійдіть у
систему.</p>";
    return;
  }

  const [booksRes, authorsRes, favRes] = await Promise.all([
    fetch("http://localhost:3000/api/books"),
    fetch("http://localhost:3000/api/authors"),
    fetch(`http://localhost:3000/api/favorites/${user.id}`)
  ]);

  const books = await booksRes.json();
  const authors = await authorsRes.json();
  const favorites = await favRes.json();

  const favBookIds = favorites.map(f => f.id);

  const favBooks = books.filter(book => favBookIds.includes(book.id));

  if (favBooks.length === 0) {
    favoritesList.innerHTML = "<p>У вас немає обраних книг.</p>";
```

```
    return;
  }

  displayBooks(favBooks, authors, user);
});

function displayBooks(books, authors, user) {
  const favoritesList = document.getElementById("favorites-list");
  favoritesList.innerHTML = "";

  books.forEach(book => {
    const authorIds = Array.isArray(book.author_ids) ? book.author_ids :
[book.author_ids];
    const matchedAuthors = authors.filter(a => authorIds.includes(a.id));

    const card = document.createElement("div");
    card.className = "book-card";
    card.dataset.bookId = book.id;

    card.innerHTML = `
      <div class="book-info">
        <a href="book.html?id=${book.id}">
          <img src="${book.image}" alt="${book.title}" />
        </a>
        <h3 class="clamp-2">${book.title}</h3>
        <p class="author clamp-1">
          ${matchedAuthors.map(a => `<a
href="author.html?id=${a.id}">${a.name}</a>`).join(", ")}
        </p>
        <p class="genre clamp-1">
          ${Array.isArray(book.genre)
            ? book.genre.map(g => `<a
href="genre.html?name=${encodeURIComponent(g)}">${g}</a>`).join(", ")
            : `<a
href="genre.html?name=${encodeURIComponent(book.genre)}">${book.genre}</a>`}
        </p>
      </div>
      <div class="actions">
        <a href="book.html?id=${book.id}" class="download-btn">Перейти</a>
        <span class="fav-btn fa fa-star checked" data-id="${book.id}"
title="Прибрати з обраного"></span>
      </div>
    `;
    favoritesList.appendChild(card);
  });

  document.querySelectorAll(".fav-btn").forEach(btn => {
    btn.addEventListener("click", async () => {
      const bookId = +btn.dataset.id;

      const res = await fetch("http://localhost:3000/api/favorites", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({
          user_id: user.id,
          book_id: bookId
        })
      });

      if (res.ok) {
        const card = btn.closest(".book-card");
        card.remove();

        if (document.querySelectorAll(".book-card").length === 0) {
          favoritesList.innerHTML = "<p>У вас немає обраних книг.</p>";
```

```
        }
      } else {
        alert("Не вдалося оновити обране.");
      }
    });
  });
}
```

genre.js

```
document.addEventListener("DOMContentLoaded", async () => {
  const user = getCurrentUser();
  const userArea = document.getElementById("user-area");
  userArea.innerHTML = user
    ? `<a href="profile.html">${user.name}</a>`
    : `<a href="login.html">Увійти</a>`;
  const params = new URLSearchParams(window.location.search);
  const genre = params.get("name"); // <-- genreName було помилкою
  document.getElementById("genre-title").textContent = `Жанр: ${genre}`;
  const bookList = document.getElementById("book-list");
  const [booksRes, authorsRes] = await Promise.all([
    fetch("http://localhost:3000/api/books"),
    fetch("http://localhost:3000/api/authors")
  ]);
  const books = await booksRes.json();
  const authors = await authorsRes.json();
  let favoriteIds = [];
  if (user) {
    const favRes = await
fetch(`http://localhost:3000/api/favorites/${user.id}`);
    const favData = await favRes.json();
    favoriteIds = favData.map(b => b.id);
  }
  const filteredBooks = books.filter(book => {
    const genres = Array.isArray(book.genre) ? book.genre : [book.genre];
    return genres.some(g => g.toLowerCase() === genre.toLowerCase());
  });

  displayBooks(filteredBooks, authors, favoriteIds, user);
});

function displayBooks(list, authors, favoriteIds, user) {
  const bookList = document.getElementById("book-list");
  bookList.innerHTML = "";

  list.forEach(book => {
    const authorIds = Array.isArray(book.author_ids) ? book.author_ids :
[book.author_ids];
    const matchedAuthors = authors.filter(a => authorIds.includes(a.id));
    const isFav = favoriteIds.includes(book.id);

    const card = document.createElement("div");
    card.className = "book-card";
    card.innerHTML = `
      <div class="book-info">
        <a href="book.html?id=${book.id}">
          <img src="${book.image}" alt="${book.title}" />
        </a>
        <h3 class="clamp-2">${book.title}</h3>
        <p class="author clamp-1">
          ${matchedAuthors.map(a => `<a
href="author.html?id=${a.id}">${a.name}</a>`).join(", ")}
        </p>
        <p class="genre clamp-1">
          ${Array.isArray(book.genre)
            ? book.genre.map(g => `<a
href="genre.html?name=${encodeURIComponent(g)}">${g}</a>`).join(", ")
            : `<a
href="genre.html?name=${encodeURIComponent(book.genre)}">${book.genre}</a>`}
        </p>
      </div>
      <div class="actions">
        <a href="book.html?id=${book.id}" class="download-btn">Перейти</a>
```

```
        <span class="fav-btn fa fa-star${isFav ? " checked" : ""}" data-
id="${book.id}" title="Обране"></span>
      </div>
    `;
    bookList.appendChild(card);
  });

  document.querySelectorAll(".fav-btn").forEach(btn => {
    btn.addEventListener("click", async () => {
      const bookId = +btn.dataset.id;

      if (!user) {
        alert("Увійдіть, щоб додавати в обране.");
        return;
      }

      await fetch("http://localhost:3000/api/favorites", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({
          user_id: user.id,
          book_id: bookId
        })
      });

      btn.classList.toggle("checked");
    });
  });
}
```

profile.js

```
document.addEventListener("DOMContentLoaded", async () => {
  const user = getCurrentUser();
  if (!user) return (window.location.href = "login.html");

  document.getElementById("user-area").innerHTML = `<a
href="profile.html">${user.name}</a>`;
  document.getElementById("name").value = user.name;
  document.getElementById("email").value = user.email;

  const userData = await
fetch(`http://localhost:3000/api/users/${user.id}`).then(res => res.json());
  document.getElementById("registration-date").textContent = new
Date(userData.created_at).toLocaleString();

  document.getElementById("profile-form").addEventListener("submit", async (e)
=> {
    e.preventDefault();
    const name = document.getElementById("name").value.trim();
    const email = document.getElementById("email").value.trim();

    const res = await fetch(`http://localhost:3000/api/users/${user.id}`, {
      method: "PUT",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({ name, email })
    });

    if (res.ok) {
      const updated = await res.json();
      localStorage.setItem("currentUser", JSON.stringify(updated));
      alert("Профіль оновлено");
      location.reload();
    } else {
      const data = await res.json();
```

```
      alert(data.message || "Помилка при оновленні профілю");
    }
  });
  document.getElementById("logout-btn").addEventListener("click", () => {
    logoutUser(); // правильна функція з auth.js
  });

  const [books, authors, favorites] = await Promise.all([
    fetch("http://localhost:3000/api/books").then(r => r.json()),
    fetch("http://localhost:3000/api/authors").then(r => r.json()),
    fetch(`http://localhost:3000/api/favorites/${user.id}`).then(r => r.json())
  ]);

  const favIds = favorites.map(f => f.book_id || f.id);
  const favoriteBooks = books.filter(b => favIds.includes(b.id));

  displayBooks(favoriteBooks, authors, user);
});

function displayBooks(books, authors, user) {
  const favoritesList = document.getElementById("favorites-list");
  favoritesList.innerHTML = "";

  books.forEach(book => {
    const authorIds = Array.isArray(book.author_ids) ? book.author_ids :
[book.author_ids];
    const matchedAuthors = authors.filter(a => authorIds.includes(a.id));

    const card = document.createElement("div");
    card.className = "book-card";
    card.dataset.bookId = book.id;

    card.innerHTML = `
      <div class="book-info">
        <a href="book.html?id=${book.id}">
          <img src="${book.image}" alt="${book.title}" />
        </a>
        <h3 class="clamp-2">${book.title}</h3>
        <p class="author clamp-1">
          ${matchedAuthors.map(a => `<a
href="author.html?id=${a.id}">${a.name}</a>`).join(", ")}
        </p>
        <p class="genre clamp-1">
          ${Array.isArray(book.genre)
            ? book.genre.map(g => `<a
href="genre.html?name=${encodeURIComponent(g)}">${g}</a>`).join(", ")
            : `<a
href="genre.html?name=${encodeURIComponent(book.genre)}">${book.genre}</a>`}
        </p>
      </div>
      <div class="actions">
        <a href="book.html?id=${book.id}" class="download-btn">Перейти</a>
        <span class="fav-btn fa fa-star checked" data-id="${book.id}"
title="Прибрати з обраного"></span>
      </div>
    `;
    favoritesList.appendChild(card);
  });

  document.querySelectorAll(".fav-btn").forEach(btn => {
    btn.addEventListener("click", async () => {
      const bookId = +btn.dataset.id;

      await fetch("http://localhost:3000/api/favorites", {
        method: "POST",
```

```
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({ user_id: user.id, book_id: bookId })
      });

      btn.closest(".book-card")?.remove();
    });
  });
}
```

read.js

```
document.addEventListener("DOMContentLoaded", async () => {
  const params = new URLSearchParams(window.location.search);
  const bookId = params.get("id");
    const backLinkDiv = document.getElementById("back-link");
    backLinkDiv.innerHTML = `<a href="book.html?id=${bookId}" class="back-
button">⬅ Повернутись до книги</a>`;


  const userArea = document.getElementById("user-area");
  const user = getCurrentUser();
  userArea.innerHTML = user
    ? `<a href="profile.html">${user.name}</a>`
    : `<a href="login.html">Увійти</a>`;

  const bookTextDiv = document.getElementById("book-text");

  const booksRes = await fetch("http://localhost:3000/api/books");
  const books = await booksRes.json();

  const book = books.find(b => String(b.id) === String(bookId));
  if (!book) {
    bookTextDiv.textContent = "Книгу не знайдено.";
    return;
  }

  const txtFile = book.formats?.find(f => f.toLowerCase().endsWith(".txt"));

  if (!txtFile) {
    bookTextDiv.textContent = "Файл для читання (.txt) не знайдено.";
    return;
  }

  try {
    const response = await fetch(txtFile);
    const text = await response.text();
    bookTextDiv.textContent = text;
  } catch (err) {
    bookTextDiv.textContent = "Не вдалося завантажити текст.";
  }

  const fontSizeSelect = document.getElementById("font-size");
  fontSizeSelect.addEventListener("change", () => {
    bookTextDiv.style.fontSize = fontSizeSelect.value;
  });
});
```

ДОДАТОК В

JSON-файли

Author.json (приклад)

```
...
{
    "id": 4,
    "name": "Джейсон Фрайд",
    "image": "assets/authors/Джейсон Фрайд.png",
    "description": "Джейсон Фрайд — американський підприємець, автор і співзасновник компанії Basecamp, яка розробляє інструменти для організації роботи команд. Відомий своїм нестандартним підходом до бізнесу, мінімалізмом у менеджменті та критикою традиційної корпоративної культури. Спів-автор бестселерів «Rework», «Remote» та «It Doesn’t Have to Be Crazy at Work», у яких ділиться практичними порадами для ефективної та здорової роботи без стресу та перевантаження. Його ідеї надихають підприємців усього світу переосмислювати підхід до праці та продуктивності."
  },
...
```

books.json (приклад)

```
...
{
    "id": 4,
    "title": "Rework. Ця книга переверне ваш погляд на бізнес",
    "authorId": [
      4,
      5
    ],
    "genre": [
      "Про бізнес"
    ],
    "image": "assets/images/rework.jpg",
    "formats": [
      "assets/books/rework.txt",
      "assets/books/rework.fb2"
    ],
    "description": "«Rework. Ця книга переверне ваш погляд на бізнес» – нестандартний погляд на підприємництво від Джейсона Фрайда та Девіда Хайнемайєра Хенссона. Автори доводять, що багато традиційних бізнес-підходів застаріли, і пропонують нові, практичні ідеї для тих, хто хоче працювати ефективніше. Книга наповнена короткими порадами, реальними прикладами та інсайтами, які руйнують стереотипи й надихають діяти. Ідеально підходить для підприємців, менеджерів і всіх, хто прагне змін у своїй справі."
  },
...
```

ДОДАТОК Д
Docker / Backend

Init.sql

```
CREATE TABLE IF NOT EXISTS users (
  id SERIAL PRIMARY KEY,
  name TEXT NOT NULL,
  email TEXT UNIQUE NOT NULL,
  password TEXT NOT NULL
);

CREATE TABLE IF NOT EXISTS authors (
  id INTEGER PRIMARY KEY,
  name TEXT NOT NULL,
  image TEXT,
  description TEXT
);

CREATE TABLE IF NOT EXISTS books (
  id INTEGER PRIMARY KEY,
  title TEXT NOT NULL,
  author_ids INTEGER[] NOT NULL,
  image TEXT,
  formats TEXT[],
  description TEXT
);

CREATE TABLE IF NOT EXISTS genres (
  id SERIAL PRIMARY KEY,
  name TEXT UNIQUE NOT NULL,
  description TEXT
);

CREATE TABLE IF NOT EXISTS book_genres (
  book_id INTEGER REFERENCES books(id) ON DELETE CASCADE,
  genre_id INTEGER REFERENCES genres(id) ON DELETE CASCADE,
  PRIMARY KEY (book_id, genre_id)
);

CREATE TABLE IF NOT EXISTS favorites (
  user_id INTEGER REFERENCES users(id) ON DELETE CASCADE,
  book_id INTEGER REFERENCES books(id) ON DELETE CASCADE,
  PRIMARY KEY (user_id, book_id)
);

CREATE TABLE IF NOT EXISTS comments (
  id SERIAL PRIMARY KEY,
  user_id INTEGER REFERENCES users(id) ON DELETE CASCADE,
  book_id INTEGER REFERENCES books(id) ON DELETE CASCADE,
  text TEXT NOT NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE IF NOT EXISTS ratings (
  user_id INTEGER REFERENCES users(id) ON DELETE CASCADE,
  book_id INTEGER REFERENCES books(id) ON DELETE CASCADE,
  rating INTEGER CHECK (rating BETWEEN 1 AND 5),
  PRIMARY KEY (user_id, book_id)
);
```

Server.js

```
const express = require("express");
const cors = require("cors");
const pool = require("./db");
const app = express();
app.use(cors());
app.use(express.json());

app.get("/api/books", async (req, res) => {
  try {
    const booksRes = await pool.query("SELECT * FROM books");
    const genresRes = await pool.query(`
      SELECT bg.book_id, g.name
      FROM book_genres bg
      JOIN genres g ON g.id = bg.genre_id
    `);

    const books = booksRes.rows;
    const genreMap = {};
    genresRes.rows.forEach(row => {
      if (!genreMap[row.book_id]) genreMap[row.book_id] = [];
      genreMap[row.book_id].push(row.name);
    });

    const booksWithGenres = books.map(book => ({
      ...book,
      genre: genreMap[book.id] || []
    }));

    res.json(booksWithGenres);

  } catch (err) {
    console.error("Помилка при отриманні книг:", err);
    res.status(500).json({ error: "Не вдалося отримати книги" });
  }
});


app.get("/api/authors", async (req, res) => {
  try {
    const result = await pool.query("SELECT * FROM authors");
    res.json(result.rows);
  } catch (err) {
    console.error(err);
    res.status(500).json({ error: "Помилка при отриманні авторів" });
  }
});

app.post("/api/ratings", async (req, res) => {
  const { user_id, book_id, rating } = req.body;
  try {
    await pool.query(`
      INSERT INTO ratings (user_id, book_id, rating)
      VALUES ($1, $2, $3)
      ON CONFLICT (user_id, book_id) DO UPDATE SET rating = $3
    `, [user_id, book_id, rating]);
    res.sendStatus(200);
  } catch (err) {
    console.error("Помилка при збереженні рейтингу:", err);
    res.status(500).json({ error: "Помилка при збереженні рейтингу" });
  }
});

app.get("/api/ratings/:bookId/:userId", async (req, res) => {
```

```
  const { bookId, userId } = req.params;
  try {
    const avgRes = await pool.query(
      "SELECT ROUND(AVG(rating), 1) AS average FROM ratings WHERE book_id = $1",
      [bookId]
    );
    const userRes = await pool.query(
      "SELECT rating FROM ratings WHERE book_id = $1 AND user_id = $2",
      [bookId, userId]
    );

    res.json({
      average: avgRes.rows[0].average || 0,
      userRating: userRes.rows[0]?.rating || 0
    });
  } catch (err) {
    console.error("Помилка при отриманні рейтингу:", err);
    res.status(500).json({ error: "Помилка при отриманні рейтингу" });
  }
});

// Реєстрація користувача
app.post("/api/register", async (req, res) => {
  const { name, email, password } = req.body;
  try {
    const exists = await pool.query("SELECT * FROM users WHERE email = $1",
[email]);
    if (exists.rows.length > 0) {
      return res.status(400).json({ error: "Користувач з таким email вже існує."
});
    }

    const result = await pool.query(
      "INSERT INTO users (name, email, password) VALUES ($1, $2, $3) RETURNING
id, name, email",
      [name, email, password]
    );

    res.status(201).json(result.rows[0]);
  } catch (err) {
    console.error("Помилка при реєстрації:", err);
    res.status(500).json({ error: "Помилка сервера" });
  }
});

// Вхід користувача
app.post("/api/login", async (req, res) => {
  const { email, password } = req.body;
  try {
    const result = await pool.query(
      "SELECT id, name, email FROM users WHERE email = $1 AND password = $2",
      [email, password]
    );

    if (result.rows.length === 0) {
      return res.status(401).json({ error: "Невірний email або пароль" });
    }

    res.json(result.rows[0]);
  } catch (err) {
    console.error("Помилка при вході:", err);
    res.status(500).json({ error: "Помилка сервера" });
  }
});
```

```
// Оновлення профілю
app.put("/api/users/:id", async (req, res) => {
  const { id } = req.params;
  const { name, email } = req.body;
  try {
    const exists = await pool.query(
      "SELECT * FROM users WHERE email = $1 AND id != $2",
      [email, id]
    );
    if (exists.rows.length > 0) {
      return res.status(400).json({ error: "Цей email вже зайнятий іншим
користувачем." });
    }

    const result = await pool.query(
      "UPDATE users SET name = $1, email = $2 WHERE id = $3 RETURNING id, name,
email",
      [name, email, id]
    );

    res.json(result.rows[0]);
  } catch (err) {
    console.error("Помилка при оновленні профілю:", err);
    res.status(500).json({ error: "Помилка сервера" });
  }
});

// Додати/забрати з обраного
app.post("/api/favorites", async (req, res) => {
  const { user_id, book_id } = req.body;
  try {
    const check = await pool.query("SELECT * FROM favorites WHERE user_id=$1 AND
book_id=$2", [user_id, book_id]);
    if (check.rows.length > 0) {
      await pool.query("DELETE FROM favorites WHERE user_id=$1 AND book_id=$2",
[user_id, book_id]);
    } else {
      await pool.query("INSERT INTO favorites (user_id, book_id) VALUES ($1,
$2)", [user_id, book_id]);
    }
    res.sendStatus(200);
  } catch (err) {
    console.error("Помилка з обраним:", err);
    res.status(500).json({ error: "Помилка при оновленні обраного" });
  }
});

// Отримати всі обрані книги користувача
app.get("/api/favorites/:userId", async (req, res) => {
  const { userId } = req.params;
  try {
    const result = await pool.query(`
      SELECT b.* FROM favorites f
      JOIN books b ON b.id = f.book_id
      WHERE f.user_id = $1
    `, [userId]);

    res.json(result.rows);
  } catch (err) {
    console.error("Помилка при завантаженні обраного:", err);
    res.status(500).json({ error: "Помилка при завантаженні обраного" });
  }
});

app.post("/api/comments", async (req, res) => {
```

```
  const { user_id, book_id, text } = req.body;
  try {
    await pool.query(`
      INSERT INTO comments (user_id, book_id, text)
      VALUES ($1, $2, $3)
    `, [user_id, book_id, text]);

    res.sendStatus(201);
  } catch (err) {
    console.error("Помилка при додаванні коментаря:", err);
    res.status(500).json({ error: "Не вдалося додати коментар" });
  }
});

app.get("/api/comments/:bookId", async (req, res) => {
  const { bookId } = req.params;
  try {
    const result = await pool.query(`
      SELECT c.text, c.created_at, c.user_id, u.name AS user_name
      FROM comments c
      JOIN users u ON u.id = c.user_id
      WHERE c.book_id = $1
      ORDER BY c.created_at DESC
    `, [bookId]);

    res.json(result.rows);
  } catch (err) {
    console.error("Помилка при завантаженні коментарів:", err);
    res.status(500).json({ error: "Не вдалося отримати коментарі" });
  }
});

app.get("/api/ratings/:bookId", async (req, res) => {
  const { bookId } = req.params;
  try {
    const avgRes = await pool.query(
      "SELECT ROUND(AVG(rating)::numeric, 1) AS average, COUNT(*) as votes FROM
ratings WHERE book_id = $1",
      [bookId]
    );

    res.json({
      average: avgRes.rows[0].average || 0,
      votes: +avgRes.rows[0].votes || 0
    });
  } catch (err) {
    console.error("Помилка при отриманні рейтингу:", err);
    res.status(500).json({ error: "Помилка при отриманні рейтингу" });
  }
});

app.get("/api/users/:id", async (req, res) => {
  const { id } = req.params;
  try {
    const result = await pool.query("SELECT id, name, email, created_at FROM
users WHERE id = $1", [id]);
    if (result.rows.length === 0) return res.status(404).json({ error:
"Користувача не знайдено" });
    res.json(result.rows[0]);
  } catch (err) {
    console.error(err);
    res.status(500).json({ error: "Помилка сервера" });
  }
});
```

```
app.get("/api/genres", async (req, res) => {
  try {
    const result = await pool.query("SELECT * FROM genres ORDER BY name");
    res.json(result.rows);
  } catch (err) {
    console.error("Помилка при отриманні жанрів:", err);
    res.status(500).json({ error: "Не вдалося завантажити жанри" });
  }
});
```

Docker-compose.yml

```
version: "3.8"
services:
  backend:
    build: ./backend
    ports:
      - "3000:3000"
    environment:
      DATABASE_URL: postgres://user:password@db:5432/librarydb
    depends_on:
      - db

  frontend:
    build: ./frontend
    ports:
      - "8080:80"
    volumes:
      - ./frontend:/usr/share/nginx/html


  db:
    image: postgres:15
    restart: always
    environment:
      POSTGRES_USER: user
      POSTGRES_PASSWORD: password
      POSTGRES_DB: librarydb
    ports:
      - "5432:5432"
    volumes:
      - db_data:/var/lib/postgresql/data
      - ./backend/init.sql:/docker-entrypoint-initdb.d/init.sql

  pgadmin:
    image: dpage/pgadmin4
    ports:
      - "5050:80"
    environment:
      PGADMIN_DEFAULT_EMAIL: admin@admin.com
      PGADMIN_DEFAULT_PASSWORD: admin
    depends_on:
      - db

volumes:
  db_data:
```