КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

БУДІВНИЦТВА І АРХІТЕКТУРИ

Факультет автоматизації інформаційних технологій

Кафедра інформаційних технологій

(назва випускової кафедри)

ПОЯСНЮВАЛЬНА ЗАПИСКА

ДО КВАЛІФІКАЦІЙНОЇ ВИПУСКНОЇ РОБОТИ

НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВР

на тему:

Розробка веб додатку бібліотечного контенту на основі онтологічного підходу

Частина друга: Розробка користувацького інтерфейсу

МУЗИКА Микола Іванович

Київ 2025 р.

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
БУДІВНИЦТВА І АРХІТЕКТУРИ
Факультет автоматизації інформаційних технологій
Інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри ІТ
Тетяна Гончаренко
_____
„___”_____2025 року

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО КВАЛІФІКАЦІЙНОЇ ВИПУСКНОЇ РОБОТИ НА ЗДОБУТТЯ
ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВР

Розробка веб додатку бібліотечного контенту на основі онтологічного підходу
Частина друга: **Розробка користувацького інтерфейсу**

Виконав
Музика Микола Іванович
122 Комп’ютерні науки
(спеціальність)
Інформаційні управляючі системи та технології
(освітня програма)
Групи КН-21-3
Керівник к.т.н. доц., Горда О.В.
(прізвище та ініціали)
Ідентичність підтверджую

Київ 2025 р.

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БУДІВНИЦТВА І АРХІТЕКТУРИ

Факультет:	Автоматизації інформаційних технологій
Випускова кафедра:	Інформаційних технологій
Освітній ступінь:	Бакалавр
Спеціальність:	Комп’ютерні науки
Освітня програма:	Інформаційні управляючі системи та технології

ЗАТВЕРДЖУЮ

Завідувач кафедри ІТ

Тетяна ГОНЧАРЕНКО

_____

„___”_____2025 року

З А В Д А Н Н Я

ДО ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ ВИПУСКНОЇ РОБОТИ НА

ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВР

Музика Микола Іванович

1. Тема роботи Розробка веб додатку бібліотечного контенту на основі онтологічного підходу. Частина друга: Розробка користувацького інтерфейсу

затверджена наказом ректора КНУБА №235/23/25 від «14» лютого 2025 року

2. Керівник роботи ктн, доц., Горда Олена Володимирівна

3. Строк подання Здобувачем роботи до захисту_____

4. Зміст пояснювальної записки за розділами:

Р.1	АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ
Р.2	ПРОЕКТУВАННЯ ФРОНТЕНД-ЧАСТИНИ СИСТЕМИ
Р.3	РОЗРОБКА ФРОНТЕНДУ ЕЛЕКТРОННОЇ БІБЛІОТЕКИ
Р.4	ЕРГОНОМІЧНІ ПОКАЗНИКИ СИСТЕМИ ЕЛЕКТРОННОЇ БІБЛІОТЕКИ

5. Графічний матеріал за розділами:

1. Актуальність проблеми, 2. Мета та завдання дослідження, 3. Аналіз предметної області, 4. Архітектура системи, 5. Проєктування користувацького інтерфейсу, 6. Семантичний та нечіткий пошук, 7. Використані технології фронтенду, 8. Безпека та управління обліковими записами, 9.Безпека та управління обліковими записами.10.Usability: ключові концепції та стандарти. 11. Ключові метрики

ергономіки UI 12.Core Web Vitals: LCP, FID, CLS 13. Адаптивність та доступність інтерфейсу 14. UX-процес: HCD та прототипування 15.Висновки та перспективи.

6. Календарний план виконання роботи:

Види робіт та їх зміст	Дата виконання
Розділ 1	04.02.2025
Розділ 2	15.03.2025
Розділ 3	18.04.2025
Розділ 4	1.05.2025
Остаточне оформлення роботи	17.05.2025
Направлення роботи для перевірки на плагіат	26.05.2025
Попередній захист роботи на випусковій кафедрі	
Направлення роботи на рецензування	

7. Консультанти розділів атестаційної випускної роботи

Розділ	Прізвище, ініціали та посада консультанта	Перевірив	
		дата	підпис
Розділ 1	Горда О.В.		
Розділ 2	Горда О.В.		
Розділ 3	Горда О.В.		
Розділ 4	Рябчун Ю.В.		

8. Дата видачі завдання 3 грудня 2024

Зав. кафедри Гончаренко Т.А.

(підпис) (прізвище та ініціали)

Керівники Горда О.В.

(підпис) (прізвище та ініціали)

Здобувач Музика М.І.

(підпис) (прізвище та ініціали)

<table>
<tr><td>РЕЗЮМЕ (SUMMARY)
до атестаційної
випускної роботи
Здобувача:</td><td colspan="3">Музика Микола Іванович
Mykola Muzyka</td></tr>
<tr><td>ЗВО</td><td colspan="3">Київський національний університет будівництва і архітектури</td></tr>
<tr><td>Тема (українською та англійською)</td><td colspan="3">Розробка веб додатку бібліотечного контенту на основі онтологічного підходу.
Частина друга: Розробка Web-UI інтерфейсу
Developing a web application for library content based on an ontological approach.
Part Two: Web-UI Development</td></tr>
<tr><td>Освітній ступінь</td><td colspan="3">Бакалавр</td></tr>
<tr><td>Факультет</td><td colspan="3">Автоматизації і інформаційних технологій</td></tr>
<tr><td>Випускаюча кафедра</td><td colspan="3">Інформаційних технологій</td></tr>
<tr><td>Спеціальність</td><td colspan="3">122 «Комп’ютерні науки»</td></tr>
<tr><td>Освітня програма</td><td colspan="3">Інформаційні управляючі системи та технології</td></tr>
<tr><td>Керівник</td><td colspan="3">Горда Олена Володимирівна</td></tr>
<tr><td rowspan="2">Обсяг роботи:</td><td>пояснювальна записка, стор.</td><td>Обсяг роботи:</td><td>пояснювальна записка, стор.</td></tr>
<tr><td>120</td><td>4</td><td></td></tr>
<tr><td>Ключові слова:

Keywords:</td><td colspan="3">електронна бібліотека, інтерфейс користувача (UI), юзабіліті (usability), досвід користувача (UX), Learnability, Memorability, Error Rate, System Usability Scale (SUS), Core Web Vitals (LCP, FID, CLS), Synthetic Monitoring, Real User Monitoring (RUM), адаптивний дизайн (responsive design), доступність (accessibility, WCAG 2.1), Human-Centered Design (HCD), прототипування (wireframes, mockups, Figma), A/B-тестування, Storybook для React.

electronic library, user interface (UI), usability, user experience (UX), Learnability, Memorability, Error Rate, System Usability Scale (SUS), Core Web Vitals (LCP, FID, CLS), Synthetic Monitoring, Real User Monitoring (RUM), responsive design, accessibility (WCAG 2.1), Human-Centered Design (HCD), prototyping (wireframes, mockups, Figma), A/B testing, Storybook for React</td></tr>
</table>

У роботі проведено класифікацію типів бібліотечного контенту та проаналізовано вимоги до їх семантичного подання на основі онтологічного

підходу. З метою створення веб додатку бібліотечного контенту для цифровізації бібліотечного середовища, розроблено програмний продукт, що автоматизує процес моделювання контенту через RESTful API та реляційну базу даних.

Здобувач: _____ /Микола МУЗИКА/

Керівник: _____/ Олена ГОРДА /

“___” _____ 2025 р.

ЗМІСТ

ВСТУП

Актуальність дослідження. У сучасному інформаційному суспільстві стрімкий розвиток цифрових технологій змінює підходи до збереження, обробки та доступу до інформації. Традиційні бібліотеки, які раніше асоціювались виключно з фізичними приміщеннями та паперовими книгами, трансформуються у цифрові платформи, забезпечуючи користувачам доступ до знань у будь-який час і з будь-якого місця. Впровадження сучасних web-технологій, зокрема використання онтологій, відкриває нові можливості для управління інформаційними ресурсами, що робить тему дослідження надзвичайно актуальною.

Онтології як форма моделювання знань дозволяють структурувати дані таким чином, щоб забезпечити їх логічну впорядкованість, швидкий пошук та зручне відображення для користувача. Застосування онтологій у web-застосунках бібліотек сприяє покращенню інтерактивності систем, їхньої адаптивності до різних запитів і підвищує якість пошукових результатів. Це особливо важливо в умовах постійного зростання обсягу цифрової інформації, коли традиційні методи пошуку та систематизації стають менш ефективними.

Проблема, що вирішується. Основною проблемою, яка вирішується, є низька ефективність і недостатня зручність доступу до інформації у сучасних цифрових бібліотеках. Багато з них обмежуються базовим пошуком за ключовими словами, що не враховує контекст, семантичні зв’язки між об’єктами чи специфічні потреби користувачів. Це призводить до того, що користувачі витрачають значний час на пошук потрібної інформації, а знайдені результати не завжди відповідають їхнім очікуванням.

Крім того, сучасні користувачі очікують високого рівня персоналізації у взаємодії з інформаційними системами. Традиційні бібліотечні web-застосунки не завжди здатні адаптуватися до специфічних запитів кожного окремого користувача. Наприклад, студент, який шукає навчальні матеріали, отримає

однакові результати з дослідником, який працює над науковим проектом, хоча їхні потреби принципово різні.

Ще однією важливою проблемою є недостатня інтеграція сучасних бібліотечних web-застосунків із великими обсягами даних та зовнішніми джерелами. У багатьох системах відсутні механізми для зручного управління ресурсами, їхньої систематизації або аналізу. Через це бібліотеки не можуть повною мірою використовувати свій потенціал як центри знань, що працюють на основі інтелектуальних технологій.

Мета дослідження. Розробка фронт-енд системи web-застосунку бібліотеки з використанням онтологій, яка забезпечить зручний інтерфейс для користувачів, дозволяючи здійснювати пошук, перегляд та управління бібліотечними ресурсами. Система повинна забезпечувати інтерактивну взаємодію з базою даних через серверну частину, підтримувати різні типи запитів та надавати релевантні результати.

Об'єкт дослідження. Клієнтське програмне забезпечення web-застосунку бібліотеки, що включає інтерфейс користувача, функціонал пошуку та навігації за онтологічними структурами, а також інтеграцію з серверною частиною.

Предмет дослідження. Методології, методи та інструментальні засоби розробки інтерфейсу користувача web-застосунку бібліотеки, включаючи створення дизайну, реалізацію функціональних компонентів та забезпечення інтерактивної взаємодії з сервером.

Методи дослідження: методи системного аналізу, об'єктна-орієнтованої методології, теорії онтологій предметної області…

Практична значимість. Результати дослідження можуть бути використані для створення web-застосунків бібліотек різного рівня складності. Система дозволить бібліотекам поліпшити якість обслуговування, забезпечивши швидкий доступ до необхідних ресурсів та зручне управління бібліотечними фондами.

Короткий зміст розділів. У першому розділі описано загальну та теоретичну інформацію щодо розробки фронт-енд частини web-застосунку

бібліотеки. Описується постановка та аналіз проблеми, дерево цілей, вимоги до системи, аналіз існуючих розробок web-застосунків бібліотек, вибір технологій для реалізації клієнтського програмного забезпечення та формулювання задачі дипломного проекту.

ЧАСТИНА 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Постановка та аналіз проблеми

З розвиток людського суспільства виникла необхідність у накопиченні, збереженні та обміну накопиченою інформацією а знаннями знаннями, що сприяло створення бібліотек, як осередків на які покладалось три важливі соціальні функції: інформаційна, культурна, освітня. Стрімкий розвиток сучасного світу, а з ним суспільства та особистості, прагнення до нового та розвитку, потребують дбайливого ставлення до накопиченої віками культурної та наукової спадщини сховищами якої є бібліотека. Протягом багатьох століть у більшості країн світу бібліотеки вважалися центром наукового та культурного розвитку і виступали як «інформаційна модель» як окремого народу, так і людства в цілому.

Бібліотеки органічно входять у всі області на всіх етапах життя людини – дитинство, навчання, професійна діяльність, дозвілля. Це місце, де можна отримати доступ до інформації, розвивати свої навички та збагачувати свої знання. У сучасному суспільстві, де стрімко розвиваються технології та цифрові ресурси, бібліотеки відіграють важливу роль, залишаючись центрами знань та культури та є потужним соціальним інститутом, який адаптується до нових реалій та відповідає потребам сучасної людини. Сьогодні, бібліотеки виконують ряд важливих функцій:

- є надійним джерелом надання доступу до знань. Хоч Інтернет пропонує нескінченний потік даних, але не всі ресурси в мережі заслуговують на довіру, бібліотеки ж гарантують якість та достовірність представлених матеріалів завдяки ретельному добору літератури та ресурсів;
- в умовах глобалізації та урбанізації бібліотеки виконують функцію збереження культурної спадщини;
- бібліотеки відіграють важливу роль у формуванні культурного та освітнього простору, організовуючи різноманітні заходи: лекції, форуми, майстер-класи та виставки;

- бібліотеки сприяють розвитку читацької культури.

Отже сучасні бібліотека виконує роль: сховища знань, освітнього центру, соціального простору. Крім того, бере активну участь в оцінці, інтерпретації та фільтрації інформації, у встановленні певних зв'язків між інформаційними масивами.

Зі стрімким накопиченням обсягів інформаційних масивів сучасні бібліотеки зустрічаються з серйозною проблемою стосовно її збереження. Це обумовлено цілим рядом причин.

1. В традиційних бібліотеках інформація представлена переважно у друкованому вигляді, а також на аудіо та фото плівках:

- це вимагає виділення значних площ для її зберігання. Підвищення цін на земельні ділянки та комунальні послуги значно здорожують утримання бібліотек;
- інтенсивне використання та неналежне зберігання таких інформаційних джерел призводить до їх псування. збереження інформаційних джерел у належному стані вимагає створення та підтримання спеціальних умов, що тягне додаткові витрати. Для створення друкованої продукції витрачаються значні обсяги паперу, що потребує вирубки лісів, яка викликає погіршення екологічної ситуації на планеті і негативно впливає на якість нашого життя. Ця проблема може бути вирішена за рахунок цифровізації;
- пошук та доставка необхідної інформації читачеві може забирати значний час і вимагає розробки та впровадження спеціальних логістичних систем.

2. Значною проблемою, що виникає під час роботи з великими, особливо, повнотекстовими базами даних, є завдання пошуку документів щодо їх змісту. Традиційні засоби пошуку часто не забезпечують адекватного вибору інформації на запит користувача. Основна проблема полягає в складності точного формулювання запиту – знання необхідних атрибутів документу: точної назви, автора, підбору ключових слів, які потрібно шукати в тексті документів. Інша фундаментальна причина полягає в тому, що іноді користувач не знає точно, яку

саме інформацію йому хотілося б отримати, маючи лише загальне уявлення про межі своїх інтересів. Впровадження сучасних пошукових систем дають змогу розширити можливості пошуку а рахунок додаткових атрибутів документів, включаючи контекстний пошук.

3. Доступність інформації. Необхідність отримання інформації також пов’язана з фактором часу: своєчасність, швидкість, ще і у певному місці перебування читача. Іноді ці фактори є критичними. Для вирішення цієї проблеми сучасні бібліотеки активно використовують цифрові технології, що робить їх більш доступними. Електронні бібліотеки, віддалені бази даних, цифрові архіви – все це дозволяє користувачам отримувати необхідну інформацію у будь-який час та з будь-якої точки світу. Розвиток електронних книг та аудіо книг розширює можливості для читання та навчання, роблячи процес більш гнучким та зручним.

4. Тривожною проблемою є те, що спостерігаються тенденції у скороченні чисельності бібліотек. Так, на Україні на 01.01.2023 кількість громадських бібліотек становила 13486, а на 01.01.2024 кількість бібліотечних просторів – 12908, але тільки 3658 з них мають статус юридичної особи. Це потребує введення нових заходів стосовно збереження бібліотек, як потужного культурно-освітнього джерела. Для цього розробляються та впроваджуються міжнародні, державні та регіональні проекти, які дозволяють залучати додаткові ресурси, впроваджувати нові послуги та програми, розширювати спектр діяльності та підвищувати ефективність роботи. Проекти сприяють інноваціям та інтеграції новітніх технологій у бібліотечну справу, що є особливо важливим в умовах цифрової трансформації.

Більшість зазначених проблем, пов’язаних з діяльністю бібліотек, може бути вирішена за рахунок розробки та впровадження системи «електронна бібліотека», яка є не конкурентом традиційних бібліотек, а сучасним новим засобом для інформаційного обслуговування.

Електронна бібліотека – це набір електронних ресурсів та супутніх технічних можливостей для створення, пошуку та використання інформації.

Головні переваги електронних бібліотек полягають у їх реалізації принципу загальнодоступності, причому, незалежно від часу та місця, підвищення якості обслуговування, зменшення витрат на утримання.

Актуальність досліджень у напрямку досліджень та розробок програмних додатків «електронна бібліотека» обумовлена зростаючою роллю цифрових технологій у сфері освіти, культури, розвитку нових технологій та бібліотечної справи, що робить вивчення електронної бібліотеки важливим та необхідним.

1.2 Дерево цілей

Як було зазначено вище, основною метою бібліотеки є задоволенням інформаційних потреб користувачів, яких далі будемо називати читачами. Для досягнення цієї мети на бібліотеку покладається задачі збору, збереження та впорядкування інформаційних джерел. Створення електронної бібліотеки дозволяє розширити цілі (рис. 1.1):

- забезпечити збереження друкованого матеріалу, перш за все рідкісних і цінних документів; надати нових властивостей друкованим та рукописним матеріалам;
- створити сприятливіші умови для більшої доступності змісту друкованого матеріалу;
- раціонально організувати бази документів.

Бібліотеки класифікують за різними ознаками:

- за значенням: всеукраїнські загальнодержавного значення (національні, державні), обласні, міські, районні, селищні, сільські. Серед електронних бібліотек ще виділять приватні;
- за змістом: універсальні, галузеві, міжгалузеві;
- за призначенням: публічні (загальнодоступні), спеціальні (академій наук, науково-дослідних інститутів, навчальних закладів, підприємств, установ, організацій); спеціалізовані (для дітей, юнацтва, осіб з фізичними вадами).

Основною метою даної роботи є створення електронної публічної бібліотеки, яка може забезпечити потреби для наукової, навчальної роботи та для доступу до художньої літератури, та підвищення ефективності роботи з нею за рахунок розробки раціонально побудованої функціональної клієнтської частини автоматизованої системи web-бібліотеки.

Рис. 1.1 Дерево основних цілей задачі створення електронної бібліотеки

З поставленої мети випливають задачі, які необхідно вирішити в межах даної роботи (рис. 1.2).

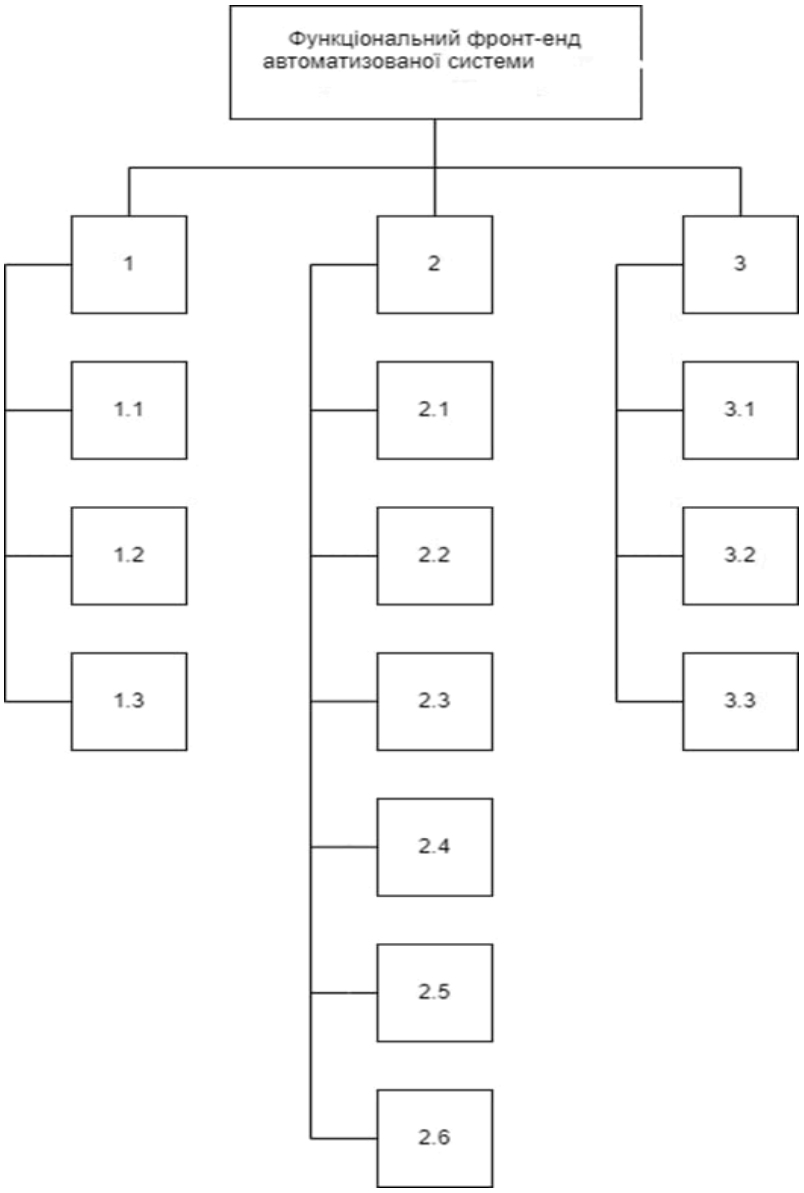


Рисунок 1.2 – Дерево задач

1. Налаштування середовища розробки:
    1.1. Встановлення та налаштування IDE;
    1.2. Налаштування системи контролю версій;
    1.3. Налаштування процесів CI/CD.
2. Розробка клієнтської частини web-застосунку:
    2.1. Аналіз технічного завдання
    2.2. Вибір стеку технологій

2.3. Проектування архітектури клієнтської частини

2.4. Розробка функціонального інтерфейсу користувача

2.5. Створення інтерфейсу для роботи з онтологіями

2.6. Інтеграція з серверною частиною.

3. Впровадження в експлуатацію

3.1. Контейнеризація клієнтського за стосунку

3.2. Налаштування хмарної інфраструктури

3.3. Розгортання застосунку на хмарній платформі.

У центрі самої ідеї бібліотеки знаходиться текст. Її першочергова роль полягає у збиранні та збереженні спадщини, вираженої в текстовій формі, а також у професійній роботі з нею. Бібліотека орієнтується або на спосіб життя своїх відвідувачів, або на область знань, у рамках якої організовує кураторство та навігацію.

Розробка web-застосунку бібліотеки з використанням онтологій є важливим завданням, що може значно підвищити ефективність пошуку, зберігання та обробки інформації. Онтологічний підхід дозволяє створити гнучку систему, яка враховує семантичні зв'язки між елементами, забезпечуючи інтуїтивний і швидкий доступ до потрібних матеріалів. Така бібліотека може стати корисною не лише для науковців, але й для широкого кола користувачів, адже вона сприятиме популяризації знань і культурного надбання, поєднуючи традиції з інноваціями.

Основні етапи життєвого циклу розробки системи:

1. Етап планування.
2. Збір вимог.
3. Проектування системи.
4. Розробка програмного забезпечення.
5. Тестування.
6. Впровадження системи.
7. Експлуатація та технічне обслуговування.

На етапі планування визначаються мета проекту, ключові завдання та очікувані результати. Далі, на етапі збору вимог, формуються чіткі специфікації для кожного компоненту системи. Після цього відбувається проектування архітектури системи, розробка інтерфейсу та функціональних модулів.

На стадії розробки команда створює програмний продукт відповідно до проектної документації. Web-розробники створюють фронтенд-інтерфейс, бекенд-фахівці займаються серверною частиною, а адміністратори баз даних відповідають за організацію та підтримку бази даних.

Тестування включає перевірку функціональності, продуктивності та безпеки системи. Після успішного тестування система впроваджується і стає доступною для користувачів. Завершальний етап — технічне обслуговування, включаючи усунення помилок, розширення функціональності та покращення інтерфейсу.

1.3 Аналіз існуючих розробок автоматизованих систем бібліотек

1.3.1. Історія розвитку цифрових бібліотек

Першим проектом створення електронної бібліотеки став Проект «Гутенберг» (1971 рік). В інтернеті на території України першою електронною бібліотекою стала бібліотека Максима Мошкова.

Зі зростанням числа користувачів комп'ютерів та інтернету все більше людей починає користуватися електронними книгами. У той же час кількість користувачів офлайнових бібліотек знижується. Так за період 1997-2002 років в Університеті Айдахо кількість відвідувачів знизилася більш ніж на 20%, а кількість користувачів електронних версій за період 1999-2002 збільшилася на 350% [1]. У зв'язку з цим багато бібліотек почали створювати електронні версії книг, що зберігаються в їх фондах.

1990 року бібліотекою конгресу США було розпочато проект «Пам'ять Америки». В рамках проекту надається вільний та безкоштовний доступ до електронних матеріалів з історії США[2].

У 2002 році Google починає власний проект з оцифрування книг [3]. У грудні 2004 року було оголошено про початок роботи бібліотечного проекту Google Print, який у 2005 році був перейменований в Пошук книг Google.

2008 року почала функціонувати загальноєвропейська цифрова бібліотека Europeana.

2009 року відбулося офіційне відкриття Всесвітньої цифрової бібліотеки.

1.3.2. Аналіз теоретичних досліджень в галузі цифрових бібліотек

Перші публікацій з проблематики електронних бібліотек відносяться до початку 1990-х рр., оскільки саме в цей період у західних бібліотеках, музеях, архівах, наукових товариствах починається етап створення різних типів цифрових колекцій, що базуються на скануванні паперових документів. У 1995 р. з'явилися перші номери мережного видання «D-Lib Magazine» присвяченого вивченню та розвитку ЕБ, включаючи перспективні інформаційні технології, програмні програми, соціально-економічні проблеми створення цифрового контенту. Необхідно зазначити, що наявність спеціалізованих науково-дослідних центрів з вивчення ЕБ характерно як для країн Західної Європи, так і для США (наприклад, Centre for Digital Library Research (CDLR), University of Strathclyde, Glasgow (UK) та інших.).

У 1997-2011 рр. при Техаському університеті у США видавався онлайновий журнал «Журнал з цифрової інформації» (Journal of Digital Information), що охоплює широкий тематичний спектр, включаючи електронні бібліотеки, цифрові репозитарії, гіпертекстові та гіпермедійні системи, віртуальні користувально-орієнтовані сервіси та ін.

У 1997 р. у Німеччині починає видаватися існуючий до цього часу «Міжнародний журнал з електронних бібліотек» (International Journal on Digital Libraries), присвячений теорії та практиці комплектування, організації, управління цифровим контентом. Основний акцент робиться на публікації матеріалів щодо проблем створення інформації в електронній формі, використанню високошвидкісних мереж для її передачі, проблемам інформаційної безпеки, інтерфейсів користувача. У цьому журналі, віднесеному до категорії соціальних та інформаційно-бібліотечних наук, за даними сервісу Microsoft Academic, у період з 1997 по 2015 р. було опубліковано 139 публікацій з тематики Електронні бібліотеки.

2003 р. з'являється щоквартальний журнал «Журнал з управління цифровою інформацією» (Journal of Digital Information Management), який поряд із спільними питаннями, пов'язаними з формуванням та функціонуванням цифрової інформаційної середовища, включає статті з управління цифровою інформацією, архівування цифрових даних, а також, з електронних бібліотек.

У 2003 р. у концептуальній статті А. Ширі [3] були позначені основні напрями вивчення електронних бібліотек.

Сьогодні, особливий інтерес представляє напрямок досліджень, пов'язаний з управлінням знаннями, яке в рамках вивчення електронних бібліотек базується на використання тезаурусів та класифікаційних систем з метою перехресного перегляду та пошуку в різних цифрових колекціях; створенні онтологій для представлення знань із різноманітних цифрових колекцій; використанні таксономій для забезпечення уніфікованого та організованого доступу до контенту різних цифрових депозитаріїв та ін. Ще одним важливим сучасним напрямом є формування «бібліотечного простору» [4].

Для покращення якості системного проектування, були проведені дослідження стосовно поведінки користувачів та їх потреб у цифровій інформації в різних контекстах, включаючи користувачів – представників університетського

середовища (студенти, викладачі), шкіл (учні, вчителі), урядових відомств (співробітники державних установ) та бізнесу. Вивчалися:

– простота використання, доступність та визнання/схвалення користувачами конкретних електронних бібліотек;

– орієнтована на користувача підтримка навчання, викладання та дослідницької діяльності через поєднання віртуальних середовищ навчання та цифрових бібліотек;

- оцінка поведінки різних користувацьких спільнот, заснована на їх базових знаннях, вікових умовах та конкретних потреб.

Одна з основних проблем у дослідженнях користувачів електронних бібліотек пов'язана з методикою та прийомами збору даних. Виявлено, що користувачі електронних бібліотек застосовують традиційні методи читацької аудиторії.

В даний час проблематика електронних бібліотек носить комплексний характер, оскільки сам об’єкт може розглядатися в межах різних галузей знань.

1.3.3. Аналіз існуючих розробок в галузі цифрових бібліотек

На сьогоднішній день існує цілих перелік практичних розробок електронних бібліотек та інструментальних засобів для їх підтримки. Розглянемо найбільш відомі з них.

1. Система «Semantic Scholar» – це науково-дослідна платформа, яка застосовує методи обробки природної мови, онтологічні моделі та штучний інтелект для пошуку наукових статей. Головна перевага системи полягає у здатності автоматично визначати ключові терміни, концепції та зв’язки між ними, що значно підвищує точність пошукових запитів. Завдяки вбудованим онтологіям система підтримує багаторівневий пошук, дозволяючи знаходити навіть суміжні теми, що є важливим аспектом наукових досліджень.

2. *Europeana* – європейська цифрова бібліотека, яка інтегрує мільйони культурних артефактів із різних бібліотек, музеїв та архівів. Система базується на онтологічних моделях метаданих, що дозволяє організовувати вміст за тематичними категоріями, жанрами та часом створення. Важливою функцією є система нечіткого пошуку, яка дозволяє отримувати релевантні результати навіть за неповними або помилковими запитами, забезпечуючи високу доступність матеріалів для широкої аудиторії.

3. *WorldCat* – найбільша у світі бібліографічна база даних, що налічує понад 240 млн записів про всі види творів 470 мовами світу. Вона використовує онтологічну структуру MARC (Machine-Readable Cataloging), що підтримує стандартизований опис бібліографічних даних. Вбудовані алгоритми нечіткого пошуку дозволяють знайти матеріали навіть за частковими або неточними запитами, що робить систему надзвичайно зручною для користувачів різних рівнів підготовки.

4. *DSpace* – це відкрита система управління цифровими архівами та репозиторіями, яка широко використовується університетами та науковими організаціями. Вона підтримує онтологічні моделі для опису метаданих, що забезпечує ефективний пошук і зберігання наукових матеріалів. Система підтримує нечіткий пошук за назвою, автором та ключовими словами, що спрощує навігацію у великих цифрових колекціях.

5. OntoLibrary – це спеціалізована бібліотечна система, розроблена на основі онтологічних моделей та семантичного вебу. Система використовує стандарти OWL (Web Ontology Language) для створення описових моделей бібліографічних записів. Вона підтримує семантичні запити та нечіткий пошук, що дозволяє отримувати релевантні результати навіть при невизначених або недостатньо точних запитах користувачів.

6. Google Books є одним із найвідоміших прикладів використання онтологій для організації інформації про книги. Хоча система орієнтована на кінцевих

користувачів, вона активно використовує алгоритми нечіткого пошуку для обробки запитів, автоматичної класифікації та індексації книжкового контенту. Взаємозв'язок між онтологічними категоріями дозволяє знаходити не лише конкретні книги, а й схожі за змістом видання.

Таблиця 1.1
Опис існуючих розробок автоматизованих систем для електронних бібліотек

Назва	Метод реалізації	Країна	Аудиторія	Особливості
WorldCat	Web-інтерфейс, API	США	академічні, публічні бібліотеки	пошук і доступ до мільйонів бібліографічних записів.
European a	Web -портал, онтологічний пошук	ЄС	академічні установи, користувачі	інтеграція онтологій для пошуку культурних об’єктів.

Назва	**Метод реалізації**	**Країна**	**Аудиторія**	**Особливості**
Google Books	Web -додаток, API	США	загальна аудиторія	оцифрування книг, підтримка повнотекстового пошуку.
Open Library	Web -платформа, відкритий доступ	США	дослідники, студенти	відкрита бібліотека з підтримкою онтологічного пошуку.
DPLA (Digital Public Library of America)	Web -інтерфейс, API	США	громадські бібліотеки	об’єднання цифрових ресурсів з підтримкою складного пошуку.

BASE (Bielefeld Academic Search Engine)	Web-платформа, онтологічний пошук	Німеч-чина	дослідники, академіки	використання семантичного пошуку на основі онтологій.
JSTOR	Web -додаток, інтерфейс	США	дослідники, академічні установи	доступ до наукових публікацій через систему фільтрації даних.

1.4 Вимоги та особливості проектування системи

В Україні діяльність електронних бібліотек регулюється низкою законодавчих актів та нормативних документів, що визначають їх функціонування, права та обов'язки користувачів, а також захист авторських прав. Основні з них:

1. **Закон України "Про авторське право та суміжні права"**: Цей закон встановлює правові засади захисту авторських прав у цифровому середовищі, що є ключовим для функціонування електронних бібліотек.
2. **Закон України "Про Національну програму інформатизації"**: Передбачає розробку та впровадження державних програм, спрямованих на створення електронних бібліотек та забезпечення доступу до цифрових інформаційних ресурсів.
3. **Положення про Українську цифрову бібліотеку**: Затверджене наказом Міністерства культури України, це положення визначає структуру, функції та завдання Української цифрової бібліотеки, яка об'єднує цифрові копії унікальних документів, книг та інших матеріалів.
4. **Концепція Державної цільової національно-культурної програми "Бібліотека–XXI"**: Схвалена розпорядженням Кабінету Міністрів України, ця концепція спрямована на створення єдиної інформаційної бібліотечної системи, що забезпечує доступність документів, які зберігаються у бібліотечних, архівних та музейних фондах.
5. **Закон України "Про Основні засади розвитку інформаційного суспільства в Україні на 2007–2015 роки"**: Визначає

напрямки розвитку інформаційного суспільства, включаючи створення електронних бібліотек та забезпечення вільного доступу до інформаційних ресурсів.

Окрім цих документів, існують також інші нормативно-правові акти та підзаконні акти, що регулюють окремі аспекти діяльності електронних бібліотек, такі як захист персональних даних, інформаційна безпека та інші.

Варто зазначити, що розвиток електронних бібліотек в Україні є пріоритетним напрямком державної політики у сфері культури та інформації, що підтверджується реалізацією проекту Національної електронної бібліотеки України, спрямованого на створення централізованого інформаційного ресурсу, який об'єднає цифрові копії унікальних документів, книг та інших матеріалів.

При розробці web-застосунку бібліотеки з використанням онтологій та системи нечіткого пошуку також необхідно врахувати такі **функціональні вимоги**:

1. **Каталогізація та управління ресурсами**: Система повинна забезпечувати можливість додавання, редагування та видалення електронних ресурсів, а також їх структурування за допомогою онтологій для покращення навігації та пошуку.
2. **Семантичний пошук**: Використання онтологій дозволяє реалізувати пошук, який враховує семантичні зв'язки між поняттями, забезпечуючи більш релевантні результати для користувачів.
3. **Нечіткий пошук**: Система повинна обробляти неточні або неповні запити користувачів, пропонуючи найбільш відповідні результати навіть при наявності орфографічних помилок або неточностей у запиті.
4. **Інтерфейс користувача**: Забезпечення інтуїтивно зрозумілого та зручного інтерфейсу, який адаптується до потреб різних категорій користувачів, включаючи можливості персоналізації та налаштування.

5. **Доступ до повнотекстових документів**: Надання користувачам можливості перегляду, завантаження або замовлення повних текстів електронних ресурсів відповідно до політики доступу бібліотеки.

6. **Управління правами доступу**: Реалізація механізмів контролю доступу до різних ресурсів, враховуючи авторські права та ліцензійні обмеження.

7. **Інтеграція з іншими системами**: Забезпечення можливості інтеграції з іншими інформаційними системами та базами даних для розширення доступу до зовнішніх ресурсів.

8. **Аналітика та звітність**: Збір статистичних даних про використання ресурсів та поведінку користувачів для подальшого аналізу та покращення сервісу.

Вимоги до програмного забезпечення:

1. **Відповідність стандартам**: Програмне забезпечення повинно відповідати міжнародним та національним стандартам у сфері бібліотечної справи та інформаційних технологій.

2. **Масштабованість**: Система повинна легко адаптуватися до зростання обсягу даних та кількості користувачів без втрати продуктивності.

3. **Безпека**: Забезпечення захисту даних від несанкціонованого доступу, втрати або пошкодження, включаючи реалізацію механізмів резервного копіювання та відновлення.

4. **Підтримка різних форматів даних**: Система повинна працювати з різноманітними форматами електронних ресурсів, такими як PDF, ePub, аудіо та відео файли.

5. **Кросплатформеність**: Забезпечення доступу до системи з різних пристроїв та операційних систем, включаючи мобільні платформи.

6. **Легкість оновлення та підтримки**: Програмне забезпечення повинно мати можливість регулярного оновлення без значних перерв у роботі, а також бути зручним для технічної підтримки.

7. **Документованість**: Наявність повної та актуальної документації для користувачів та адміністраторів системи.

Врахування цих вимог сприятиме створенню ефективного та зручного web-застосунку бібліотеки, який відповідатиме сучасним стандартам та потребам користувачів.

Основні вимоги до web-застосунку:

1. Функціональність:

- система повинна забезпечувати пошук бібліотечних ресурсів на основі онтологічних структур, використовуючи нечіткий пошук для обробки неоднозначних або неповних запитів;
- підтримка навігації за категоріями та фільтрування результатів на основі онтологічних зв’язків.

2. Безпека:

- Забезпечення конфіденційності даних користувачів через шифрування запитів та результатів пошуку (наприклад, використання TLS).
- Аутентифікація та авторизація користувачів для доступу до персоналізованих функцій.

3. Стабільність:

- Система повинна підтримувати стабільну роботу навіть при значних обсягах даних.
- Забезпечення відмовостійкості через автоматичне відновлення з’єднання та кешування результатів пошуку.

4. Ефективність:

- Оптимізація запитів на основі індексів онтологічної бази даних.
- Використання алгоритмів нечіткого пошуку, таких як методи порівняння за схожістю слів (наприклад, алгоритм Левенштейна).

Особливості проектування:

1. Приватність:

- Використання шифрування запитів та результатів пошуку (AES, RSA).
- Аутентифікація через паролі або двофакторну аутентифікацію.

2. Стабільність:

- Використання надійних клієнтських технологій, таких як React.js або Vue.js.
- Впровадження CDN для швидкої доставки статичних ресурсів.

3. Ефективність:

- Застосування онтологічних структур для індексації даних.
- Використання сучасних бібліотек для пошуку та сортування (наприклад, Elasticsearch).

4. Користувацький досвід:

- Інтуїтивно зрозумілий інтерфейс з підтримкою підказок при введенні запитів.
- Інтерактивний дизайн з можливістю персоналізації.

З урахуванням цих вимог та особливостей, фронт-енд система web-застосунку бібліотеки повинна забезпечувати зручний доступ до інформаційних ресурсів, підтримувати пошук за допомогою онтологій та гарантувати безпеку обробки даних користувачів.

1.5 Аналіз та вибір методології для реалізації проекту

Аналіз існуючих проблем та вже функціонуючих систем автоматизованих бібліотек свідчить про те, що використання онтологій є не лише актуальним, але й перспективним напрямом розвитку сучасних інформаційних технологій. Ці інструменти значно підвищують точність пошуку інформації завдяки врахуванню семантичних зв’язків між даними та контексту запитів користувачів. Вони також дозволяють ефективно організовувати великі обсяги даних, полегшуючи їхню систематизацію, управління та доступ для широкого кола користувачів.

Створення та використання онтології є, зазвичай, головним шляхом вирішення проблем у системах де:

– є великий обсяг накопиченої інформації та її щорічний стрімкий приріст пред'являють усі більш високі вимоги до ефективності її переробки;

– існуючі методи та засоби автоматизованої обробки природних мов (ПМ) інформації орієнтовані на поверхнево-семантичний аналіз текстових документів.

Сучасні бібліотечні системи, оснащені онтологічними моделями, надають можливість значно покращити взаємодію користувачів із бібліотечними ресурсами. Це забезпечується не лише за рахунок більш точних пошукових алгоритмів, а й завдяки персоналізованому підходу до надання інформації. Системи нечіткого пошуку, які використовують елементи штучного інтелекту, дозволяють інтерпретувати нечіткі або неповні запити, що робить пошук більш зручним і доступним навіть для непідготовлених користувачів.

У ході дослідження встановлено, що впровадження таких інструментів не лише підвищує ефективність бібліотечних web-застосунків, але й відкриває нові можливості для їх розвитку. Майбутні розробки можуть орієнтуватися на вдосконалення онтологічних моделей, щоб забезпечити ще глибший семантичний аналіз даних, що дозволить користувачам отримувати релевантні результати навіть у випадку складних запитів. Крім того, інтеграція з технологіями штучного інтелекту, такими як машинне навчання та обробка природної мови, сприятиме створенню інтелектуальних бібліотечних систем, які здатні самонавчатися та адаптуватися до потреб користувачів.

Онтологія (в інформатиці) – це спосіб всеосяжної та детальної формалізації деякої галузі знань за допомогою концептуальної моделі. Зазвичай така модель складається з ієрархічної структури даних, що містить усі релевантні класи об'єктів, їх зв'язки та правила, прийняті в даній галузі. Цей термін в інформатиці є похідним від стародавнього філософського поняття "онтологія". В інформаційних технологіях під ним розуміють систему знань, яка придатна для машинно-орієнтованого семантичного розбору інформації.

Формально онтологія складається з понять [концепцій], організованих у таксономію (впорядкований класифікаційний ряд), їх описів та правил виведення. У загальному випадку комп'ютерна онтологія певної предметної галузі формально є упорядкованою трійкою:

$O = < X, R, F >$,

де X, R, F – кінцеві множини відповідно: X – концепти (поняття, терміни) предметної галузі; R – відношення між ними; F – функції інтерпретації X та/або R.

Побудова формальної онтології полягає у реалізації наступних етапів:

Етап 1. Виділення концептів найвищого рівня онтологічного дерева.

Етап 2. Виділення ключових слів у концептах найвищого рівня.

Етап 3. Встановлення зв'язків між концептами найвищого рівня.

Етап 4. Визначення дочірніх концептів.

Етап 5. Встановлення зв'язків між концептами онтології.

Етап 6. Побудова онтографа.

Очевидно, що кількість дочірніх концептів для кожного з концептів найвищого рівня може бути різним і загальна структура онтологічного дерева залежатиме від предметної області та репрезентативності вихідного корпусу текстів. Розглянемо докладніше етапи запропонованого методу.

Виділення концептів вищого рівня (КВУ) для будь-якої онтології предметної області (ОПО) є важливим етапом у загальному алгоритмі проектування. Під концептами вищого рівня онтології, що формується, будемо розуміти найбільш важливі (значущі) для аналізованої предметної області об’єкти. Класи онтології представляють собою колекції об’єктів, що формуються за певними ознаками. Класи можуть включати як інші класи (підкласи) так і екземпляри, отже, класи представляють ієрархію понять та відношень між ними.

Задачі $\Rightarrow$ елементи онтології

Для системи «електронна бібліотека » виділимо три основних класи концептів:

- Користувачі: читач, бібліотекар, адмін;

- процеси;
- контент.

Х стандартів СІБІД, заснованих на аналогічних міжнародних стандартах, встановлює терміни та визначення в галузі інформаційної діяльності, бібліотечної справи та бібліографії. Структура онтології Частина ієрархії основних понять розробленої онтології представлена Рис. 1. у нотації IDEF5. Відповідно до цієї специфікації гуртками позначені класи; лінії із стрілочками на кінцях позначають наявність зв'язку між класами; лінії із зворотними стрілочками на кінцях позначають відносини наслідування. На додаток до онтології було створено класифікатори видів творів, типів видань та тематичних рубрик. Це архітектурне рішення дозволяє змінювати класифікатори без необхідності модифікації онтології, структура якої фіксується версією. Як основа для тематичного рубрикатора взято 3-х рівневий книжковий тематичний класифікатор1 (КТК) з потенційною можливістю розширення на будь-який рівень вкладеності. 4-х рівневий КТК ТД «Бібліо-Глобус» не підійшло, оскільки має фрагментарний недопрацьований характер.

Рис. 1.3 частина ієрархії розробленої онтології

1.6 Аналіз системи «Електронна бібліотека»

Електронна бібліотека отримує широкий спектр вхідних даних, необхідних для функціонування та задоволення потреб користувачів:

- **Дані від читачів**: включають інформацію про профілі користувачів, їхні уподобання, історію запитів та переглядів.
- **Запити від читачів**: пошукові запити для отримання необхідної інформації чи матеріалів.
- **Нові надходження**: дані про нові матеріали, що додаються до бібліотечного фонду.
- **Інформація на бібліотечному ринку**: актуальна інформація про нові видання, зовнішні ресурси та тренди.
- **Дії бібліотекарів та адміністраторів**: оновлення, редагування, структуризація контенту.

Виходи системи (Outputs)

Система забезпечує користувачів та адміністраторів такими результатами:

- **Обслуговування читачів**: доступ до матеріалів, рекомендаційних сервісів, інтерактивних функцій.
- **Оновлений контент**: регулярно актуалізовані каталоги, бази даних, доступ до нових надходжень.
- **Аналітичні дані**: статистика використання системи, звіти, що допомагають оцінити ефективність її роботи.

Функціональні можливості

Система «Електронна бібліотека» виконує ряд ключових функцій:

- **Обробка запитів**: забезпечення швидкого пошуку матеріалів за параметрами, заданими користувачем.
- **Управління контентом**: наповнення, редагування та видалення застарілих матеріалів.
- **Персоналізація**: адаптація сервісу до індивідуальних потреб користувачів.

- **Дотримання нормативно-правової бази**: відповідність системи вимогам авторського права.

Учасники процесу

До основних учасників, які взаємодіють із системою, належать:

- **Читачі**: кінцеві користувачі, які шукають, переглядають або завантажують матеріали.
- **Бібліотекарі**: фахівці, що забезпечують організацію доступу до ресурсів.
- **Адміністратори**: відповідальні за технічне обслуговування та розвиток системи.
- **Сервісні інструменти**: програмні модулі та зовнішні платформи, що інтегруються із системою для розширення функціональності.

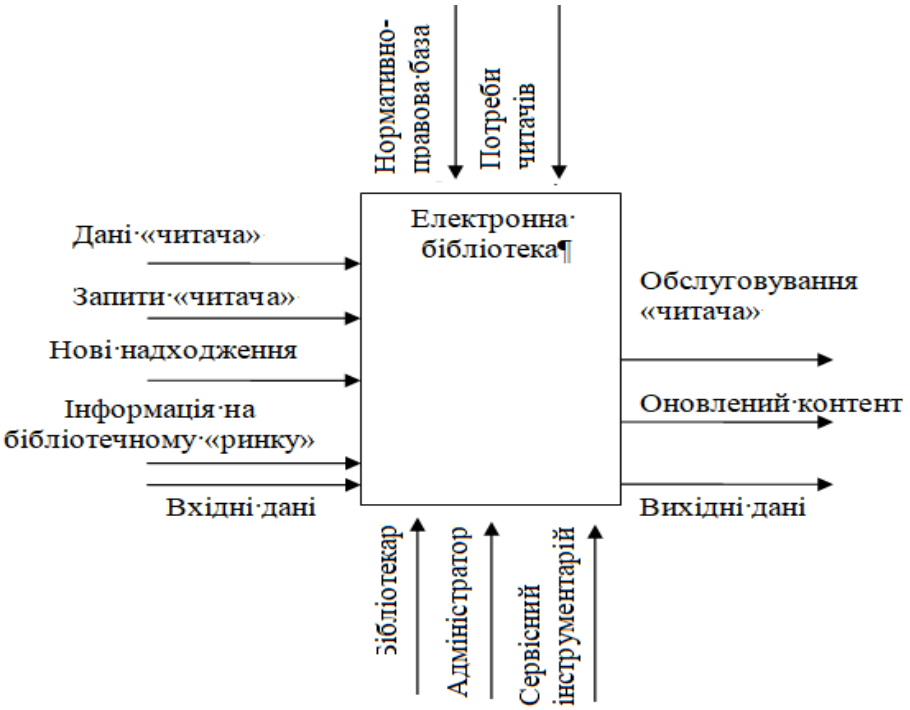
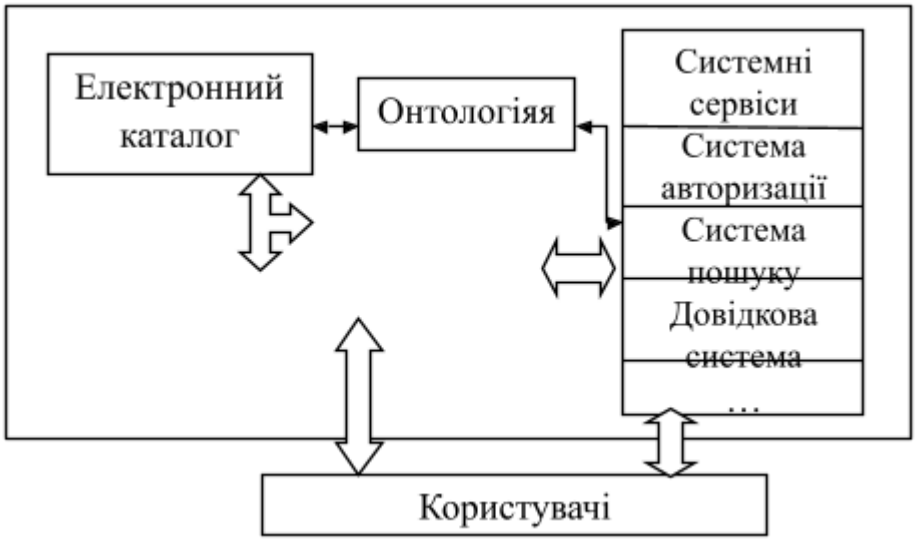


Рис. 1.4 Модель «чорна скринька»



Репозиторій

Інтерфейс

Рис. 1.5 Загальна структура системи

Елементи системи:
Електронний каталог:

Функція: Зберігання та упорядкування інформації про книги та матеріали.
Структурна роль: Джерело даних для пошукових запитів користувачів.
Інтеграція з репозиторієм.
Репозиторій:

Функція: Фізичне чи електронне сховище контенту.
Структурна роль: Забезпечує доступ до первинних ресурсів.
Інтерфейс:

Функція: Забезпечує взаємодію між користувачами та системою.
Структурна роль: Центральний пункт комунікації між каталогом, репозиторієм, онтологією та системними сервісами.
Онтологія:

Функція: Семантична організація даних.
Структурна роль: Покращення точності пошуку шляхом розуміння контексту запитів.
Системні сервіси:

Функція: Підтримка пошуку, авторизації та роботи довідкової системи.
Структурна роль: Інфраструктура, що забезпечує базову функціональність.
Взаємодія:

Користувачі через інтерфейс отримують доступ до електронного каталогу, репозиторію, систем пошуку та довідкових інструментів. Онтології виступають як механізм розширеного пошуку.

Початковий стан	**Механізм розвитку**	**Вихідний стан**
Читачі	Пошукова система	Задоволення потреб читачів
Контент	Інструментарій поповнення та управління контентом	Поповнення контенту
Новини на бібліотечному ринку	Рубрики новин Форуми Вебінари Ведення статистики	Розширення аудитор читачів та ефективності роботи бібліотеки

Схема 1.1 «стріла цілепокладання»

Аналіз початкового стану:
Читачі:

Початковий стан: Потреба у пошуку матеріалів.
Мета: Покращення досвіду пошуку.
Контент:

Початковий стан: Обмежений доступний набір даних.
Мета: Розширення бази контенту.
Новини на бібліотечному ринку:

Початковий стан: Відсутність інтеграції сучасних інформаційних джерел.
Мета: Підтримка актуальності.

Механізми розвитку: Пошукова система:
Роль: Полегшення доступу до даних через семантичний пошук.

Інструменти для поповнення контенту:

Функція: Автоматизація завантаження, категоризації та оновлення бази даних.

Новинні рубрики, форуми, вебінари: Механізми залучення аудиторії та обміну інформацією.

Вихідний стан:
Задоволення потреб читачів: Завдяки покращенню пошуку.
Поповнення бази: Наявність більш широкого контенту.
Розширення аудиторії: Статистика та аналітика сприяють оптимізації бібліотечних послуг.

1.7 Постановка задачі.

Метою дипломного проекту є розробка клієнтської частини web-застосунку бібліотеки з використанням онтологій, яка забезпечить ефективний пошук, перегляд і управління бібліотечними ресурсами. Застосунок повинен забезпечувати користувачам доступ до структурованих даних, підтримувати семантичний пошук і взаємодію з бібліотечною базою знань на основі онтологічного підходу.

Реалізація проекту передбачає вибір сучасного стеку технологій для фронтенд-розробки, створення інтуїтивно зрозумілого інтерфейсу, інтеграцію з бекендом та підтримку різних типів ресурсів. Очікуваним результатом є інтерактивний web-застосунок, який відповідає сучасним стандартам зручності та безпеки.

ЧАСТИНА 2. ПРОЕКТУВАННЯ ФРОНТЕНД-ЧАСТИНИ СИСТЕМИ

Розробка фронтенд-частини системи електронної бібліотеки є складним та багатогранним процесом, що вимагає ретельного планування та виконання. У цьому розділі ми детально розглянемо кожен етап проектування, починаючи з аналізу системи та закінчуючи побудовою і аналізом концептуальної імітаційної моделі за допомогою мереж Петрі.

2.1. Аналіз системи

Перш ніж розпочати розробку фронтенд-частини електронної бібліотеки, необхідно провести всебічний аналіз системи. Цей етап включає вивчення вимог користувачів, аналіз існуючих рішень на ринку, визначення функціональних та нефункціональних вимог, а також оцінку технічних обмежень.

Вимоги користувачів: Розуміння потреб та очікувань цільової аудиторії є ключовим для створення зручного та інтуїтивно зрозумілого інтерфейсу. Це може включати опитування потенційних користувачів, аналіз їх поведінки та вивчення відгуків про подібні системи.

- **Простий та інтуїтивно зрозумілий інтерфейс:** Користувачі очікують легку навігацію та доступ до функцій без необхідності вивчення інструкцій.
- **Швидкий пошук книг:** Можливість швидко знаходити потрібні книги за назвою, автором, жанром або іншими параметрами.
- **Детальна інформація про книги:** Перегляд опису, обкладинки, рейтингу, відгуків та іншої релевантної інформації.
- **Управління обліковим записом:** Реєстрація, авторизація, редагування профілю та налаштувань.
- **Можливість залишати відгуки та оцінки:** Інтерактивність через коментарі та рейтинги книг.
- **Сумісність з різними пристроями:** Адаптивний дизайн для комфортного використання на комп'ютерах, планшетах та смартфонах.

Аналіз ринку: Дослідження існуючих електронних бібліотек допомагає виявити їх сильні та слабкі сторони, що, в свою чергу, дозволяє розробити конкурентоспроможний продукт. Це може включати аналіз функціональності, дизайну, продуктивності та відгуків користувачів про інші системи.

- **Вивчення конкурентів:** Аналіз існуючих електронних бібліотек для виявлення їх сильних та слабких сторін.

- **Оцінка функціональності:** Визначення, які функції є стандартом, а які можуть стати конкурентною перевагою.
- **Дослідження дизайну:** Вивчення візуальних рішень, що приваблюють користувачів, та їх впровадження з урахуванням унікальності продукту.

Функціональні вимоги: Визначення основних функцій, які повинна виконувати система, таких як пошук книг, перегляд інформації про книги, управління обліковим записом користувача, додавання відгуків тощо. Кожна з цих функцій повинна бути детально описана з точки зору її поведінки та взаємодії з користувачем.

- **Пошук книг:** Реалізація пошуку з фільтрами та сортуванням результатів.
- **Перегляд інформації про книги:** Відображення детальної інформації з можливістю переходу до схожих видань.
- **Управління обліковим записом:** Функції реєстрації, входу, відновлення паролю та редагування профілю.
- **Додавання відгуків та оцінок:** Можливість залишати та переглядати відгуки інших користувачів.
- **Створення списків бажань:** Опція додавання книг до персональних списків для подальшого читання.

Нефункціональні вимоги*:* Вимоги, що стосуються продуктивності, безпеки, масштабованості, доступності та інших аспектів, які не пов'язані безпосередньо з функціональністю, але є критичними для успіху системи. Наприклад, система повинна забезпечувати швидкий час відгуку, підтримувати велику кількість одночасних користувачів та захищати персональні дані.

- **Продуктивність:** Швидкий час відгуку інтерфейсу та мінімальний час завантаження сторінок.
- **Безпека:** Захист персональних даних користувачів та забезпечення безпечної авторизації.

- **Масштабованість:** Можливість системи обробляти зростаючу кількість користувачів без втрати продуктивності.
- **Доступність:** Забезпечення доступу для користувачів з обмеженими можливостями, відповідно до стандартів доступності.

Технічні обмеження: Оцінка обмежень, пов'язаних з обраними технологіями, платформами, на яких буде працювати система, та ресурсами, доступними для розробки. Це може включати обмеження щодо сумісності з різними браузерами, підтримки мобільних пристроїв, вимог до серверної інфраструктури тощо.

- **Сумісність з браузерами:** Підтримка основних браузерів (Chrome, Firefox, Safari, Edge) та їх попередніх версій.
- **Підтримка мобільних пристроїв:** Адаптивний дизайн для коректного відображення на різних розмірах екранів.
- **Обмеження серверної інфраструктури:** Врахування можливостей серверів для забезпечення необхідної продуктивності та надійності.

2.2. Проектування архітектури системи

У процесі розробки веб-бібліотеки було проведено детальне дослідження сучасних технологій та підходів, спрямованих на створення ефективної, масштабованої та зручної системи для користувачів. Особлива увага приділялася впровадженню семантичного пошуку з використанням онтологій, що дозволяє покращити релевантність результатів та забезпечити більш інтуїтивний пошук.

Дослідження клієнтської частини (фронтенду):

Під час аналізу сучасних фреймворків для розробки інтерфейсу користувача було обрано React. Це рішення базувалося на його компонентній архітектурі, яка сприяє повторному використанню коду та спрощує підтримку додатку. Дослідження показали, що використання React забезпечує динамічне оновлення

інтерфейсу без необхідності перезавантаження сторінки, що значно покращує користувацький досвід.

Для стилізації інтерфейсу було розглянуто декілька CSS-бібліотек. Вибір зупинився на Tailwind CSS завдяки її утилітарному підходу, який дозволяє швидко створювати адаптивний дизайн. Дослідження підтвердили, що Tailwind CSS сприяє підтримці єдиного стилю в усьому додатку та спрощує процес внесення змін до дизайну.

Дослідження серверної частини (бекенду):

У процесі вибору технології для серверної частини було проведено порівняльний аналіз декількох веб-фреймворків. FastAPI був обраний завдяки його високій продуктивності та підтримці асинхронних операцій. Дослідження показали, що FastAPI дозволяє обробляти велику кількість запитів одночасно, забезпечуючи швидкий відгук системи.

Бекенд відповідає за обробку запитів від клієнтської частини, взаємодію з базою даних та реалізацію бізнес-логіки додатку. Особлива увага приділялася забезпеченню безпеки та ефективності обробки даних.

Дослідження бази даних:

Для зберігання даних було розглянуто декілька реляційних баз даних. PostgreSQL був обраний завдяки своїй надійності, масштабованості та підтримці складних запитів. Дослідження підтвердили, що PostgreSQL забезпечує ефективне зберігання інформації про книги, користувачів, відгуки та інші необхідні дані.

Структура бази даних була розроблена з урахуванням нормалізації, що мінімізує дублювання даних та забезпечує їх цілісність. Це сприяє швидкому доступу до інформації та ефективному управлінню даними.

Дослідження інтеграції онтологій для семантичного пошуку:

Одним із ключових аспектів проєкту було дослідження можливостей впровадження семантичного пошуку з використанням онтологій. Онтології дозволяють визначити взаємозв'язки між різними поняттями та створити більш глибоке розуміння змісту інформації.

Дослідження показали, що використання онтологій у поєднанні з FastAPI дозволяє реалізувати пошук, який аналізує значення та контекст запиту користувача. Це забезпечує більш релевантні та точні результати для користувача, покращуючи загальний досвід взаємодії з системою.

Таблиця 2.1 Відповідність технологій та функцій, що вони виконують

Функція	Опис	Технології
Пошук та фільтрація	Забезпечує користувачам можливість швидко знаходити потрібні ресурси за допомогою ключових слів, категорій та інших параметрів.	FastAPI для обробки запитів, онтології для семантичного пошуку.
Інтерфейс користувача	Інтуїтивно зрозумілий дизайн з основними екранами: головна сторінка, результати пошуку, детальна інформація про ресурс.	React для побудови компонентів, Tailwind CSS для стилізації.
Управління обліковими записами	Дозволяє користувачам реєструватися, входити в систему, редагувати профіль та управляти налаштуваннями облікового запису.	FastAPI для бекенду, PostgreSQL для зберігання даних користувачів.

2.3 Функціональність веб-бібліотеки

У процесі розробки веб-бібліотеки було приділено особливу увагу її функціональним можливостям, спрямованим на забезпечення зручності та ефективності використання для кінцевих користувачів. Основні аспекти функціональності включають особливості пошуку та фільтрації, дизайн інтерфейсу користувача та систему управління обліковими записами.

Особливості пошуку та фільтрації

Пошук інформації є ключовою функцією будь-якої бібліотеки, особливо в умовах великого обсягу даних. Традиційні методи пошуку, засновані на простому зіставленні ключових слів, часто не забезпечують достатньої точності та

релевантності результатів. У цьому проєкті було досліджено та впроваджено семантичний підхід до пошуку, який враховує контекст та взаємозв'язки між поняттями.

Семантичний пошук базується на використанні онтологій — формальних моделей, що описують поняття та їхні взаємозв'язки в певній предметній області. Це дозволяє системі "розуміти" значення запитів користувачів та знаходити інформацію, яка може бути релевантною, навіть якщо вона не містить точних ключових слів із запиту. Наприклад, при пошуку книг про "штучний інтелект" система може також запропонувати ресурси, пов'язані з машинним навчанням або нейронними мережами, враховуючи їхній семантичний зв'язок.

Для реалізації цієї функції було проведено детальний аналіз існуючих онтологій у сфері інформаційних технологій та обрано найбільш підходящі для інтеграції в систему. Було розроблено механізм обробки запитів користувачів, який включає:

1. *Аналіз запиту*: розбиття запиту на окремі компоненти та визначення ключових понять.
2. *Пошук у онтології*: визначення взаємозв'язаних понять та розширення початкового запиту.
3. *Пошук у базі даних*: виконання розширеного запиту до бази даних книг.
4. *Ранжування результатів*: оцінка релевантності знайдених ресурсів та їх сортування відповідно до ступеня відповідності запиту.

Такий підхід забезпечує більш точні та релевантні результати пошуку, підвищуючи задоволеність користувачів та ефективність використання бібліотеки.

Інтерфейс для користувачів (дизайн та основні екрани)

Дослідження користувацького досвіду показують, що зручний та інтуїтивно зрозумілий інтерфейс є критично важливим для залучення та утримання користувачів. У цьому проєкті було проведено аналіз сучасних тенденцій у

веб-дизайні та розроблено інтерфейс, який відповідає принципам мінімалізму та зручності використання.

Основні екрани веб-бібліотеки включають:

1. **Головна сторінка**: містить верхню панель навігації з логотипом бібліотеки, меню ("Каталог", "Про нас", "Контакти", "Особистий кабінет") та пошуковим рядком з автодоповненням. Центральний блок представлений великим банером з вітальним повідомленням та кнопкою "Почати використовувати". Нижня частина сторінки містить інформацію про бібліотеку та посилання на соціальні мережі.
2. **Каталог книг**: верхня частина сторінки містить панель фільтрів з полем для введення тексту (пошук за назвою/автором), фільтрами ("Жанр", "Рік випуску", "Автор", "Доступність") та кнопкою "Скинути фільтри". Список книг представлений у вигляді карток (по 4-6 на рядок), кожна з яких містить обкладинку, назву, автора, рейтинг та кнопку "Детальніше" або "Додати в обране". Під списком книг розташована пагінація з кнопками "Вперед", "Назад" та можливістю вибору сторінки.
3. **Сторінка книги**: містить обкладинку книги (зліва) та інформацію про неї (справа): назву, автора, жанр, рік випуску, опис книги, рейтинг (з можливістю поставити оцінку) та кнопку "Додати в обране" або "Відкрити". Нижче розташовані розділи з рецензіями користувачів та схожими книгами.
4. **Особистий кабінет**: складається з розділів "Профіль користувача" (ім'я, аватар, налаштування), "Обране" (список доданих в обране книг), "Історія" (книги, які користувач переглядав або брав) та "Зворотній зв'язок" (форма для зв'язку з підтримкою або залишення відгуків). Також присутні кнопки "Редагувати профіль" та "Вийти".

Дизайн інтерфейсу був розроблений з урахуванням принципів адаптивності, що забезпечує коректне відображення та зручність використання на різних пристроях, включаючи комп'ютери, планшети та смартфони.

Система управління обліковими записами

У процесі розробки веб-бібліотеки було приділено значну увагу створенню надійної та зручної системи управління обліковими записами користувачів. Це забезпечує не лише безпеку та конфіденційність даних, але й персоналізований досвід для кожного користувача.

Реєстрація та аутентифікація

Для забезпечення безпеки та зручності користувачів було впроваджено систему реєстрації та аутентифікації. Користувачі можуть створити обліковий запис, надавши необхідну інформацію, таку як ім'я, електронна пошта та пароль. Після реєстрації вони можуть увійти в систему, використовуючи свої облікові дані. Це дозволяє забезпечити персоналізований досвід, зберігаючи інформацію про вподобання користувача, історію переглядів та інші персональні налаштування.

Управління профілем

Користувачі мають можливість керувати своїм профілем через інтуїтивно зрозумілий інтерфейс. Вони можуть змінювати особисту інформацію, таку як ім'я, електронна пошта та пароль, а також налаштовувати параметри облікового запису відповідно до своїх вподобань. Це забезпечує гнучкість та контроль над персональними даними.

Ролі та дозволи

Система підтримує різні ролі користувачів з відповідними рівнями доступу. Наприклад, адміністратори мають розширені права для управління контентом та користувачами, тоді як звичайні користувачі мають доступ лише до функцій, необхідних для перегляду та взаємодії з бібліотекою. Це забезпечує безпеку та цілісність системи, запобігаючи несанкціонованому доступу до критичних функцій.

Безпека даних

Безпека даних користувачів є пріоритетом. Всі паролі зберігаються у зашифрованому вигляді, а передача даних між клієнтом та сервером здійснюється через захищені канали зв'язку (HTTPS). Регулярні оновлення та моніторинг

системи допомагають виявляти та усувати потенційні вразливості, забезпечуючи надійний захист інформації.

Інтеграція з онтологіями для семантичного пошуку

Однією з ключових інновацій веб-бібліотеки є інтеграція онтологій для реалізації семантичного пошуку. Онтології дозволяють структурувати інформацію та встановлювати взаємозв'язки між різними поняттями, що значно покращує якість пошуку.

Розробка онтологій

Було проведено детальний аналіз предметної області для визначення ключових понять та їх взаємозв'язків. На основі цього аналізу були створені онтології, які відображають структуру знань у бібліотеці. Це включає категорії книг, авторів, жанри, теми та інші релевантні атрибути.

Інтеграція з пошуковою системою

Онтології були інтегровані з пошуковою системою бібліотеки, що дозволяє здійснювати семантичний пошук. Це означає, що користувачі можуть вводити запити природною мовою, а система зможе розуміти контекст та знаходити релевантні результати, навіть якщо вони не містять точних ключових слів із запиту.

Приклади запитів для семантичного пошуку

1. **Запит:** "Книги про штучний інтелект для початківців"

   Результат: Система знаходить книги, які стосуються теми штучного інтелекту та призначені для початківців, навіть якщо в описі книги не згадується слово "початківець".

2. **Запит:** "Романи про подорожі в часі"

   Результат: Система знаходить художні твори, в сюжеті яких присутні елементи подорожей у часі, навіть якщо в описі не використовується ця фраза.

Переваги семантичного пошуку

Використання онтологій для семантичного пошуку забезпечує більш точні та релевантні результати, покращуючи користувацький досвід. Користувачі можуть знаходити інформацію швидше та ефективніше, без необхідності формулювати точні запити або знати специфічні терміни.

2.4 Системи взаємодії з користувачем

Особливості пошуку та фільтрації

Для покращення пошуку в бібліотеці можна застосувати семантичний пошук, який враховує значення слів та їхні зв'язки, забезпечуючи більш точні результати. Це досягається шляхом використання онтологій, які моделюють знання в певній доменній області та визначають взаємозв'язки між поняттями. Інтеграція онтологій дозволяє системі краще розуміти контекст запитів користувачів та надавати релевантні результати.

Інтерфейс для користувачів (дизайн та основні екрани)

Для створення сучасного та зручного інтерфейсу користувача рекомендується використовувати фреймворки React та Tailwind CSS. React забезпечує ефективне управління станом додатка та компонентний підхід до розробки інтерфейсу, що сприяє повторному використанню коду та спрощує підтримку. Tailwind CSS надає набір утилітарних класів для швидкої стилізації компонентів без необхідності написання власних CSS-правил, що прискорює процес розробки та забезпечує узгодженість дизайну.

Система управління обліковими записами

Реалізація системи управління обліковими записами користувачів є важливим аспектом для забезпечення персоналізованого досвіду та безпеки. Це включає реєстрацію та автентифікацію користувачів, управління ролями та правами доступу, а також можливість відновлення паролю. Використання сучасних бібліотек та фреймворків для аутентифікації, таких як OAuth або JWT, може спростити цей процес та забезпечити високий рівень безпеки.

Застосування зазначених підходів та технологій сприятиме створенню ефективної та зручної веб-бібліотеки, яка задовольнить потреби користувачів та забезпечить інтуїтивно зрозумілий інтерфейс.

Основні функції системи управління обліковими записами:

1. ***Реєстрація та автентифікація користувачів:***

- Надання можливості новим користувачам створювати облікові записи через зручний інтерфейс реєстрації.
- Забезпечення безпечної автентифікації за допомогою сучасних методів шифрування паролів та, за необхідності, двофакторної автентифікації.

2. ***Управління ролями та правами доступу:***

- Визначення різних ролей користувачів (наприклад, адміністратор, бібліотекар, читач) з відповідними правами доступу до функцій та ресурсів системи.
- Гнучке налаштування прав доступу для забезпечення безпеки та конфіденційності даних.

3. ***Профілі користувачів:***

- Зберігання персональних даних користувачів, таких як ім'я, контактна інформація, історія переглядів та вподобання.
- Можливість користувачам оновлювати свої профілі та налаштовувати персональні параметри.

4. ***Моніторинг та аудит активності:***

- Відстеження дій користувачів у системі для забезпечення безпеки та виявлення потенційних загроз.
- Зберігання журналів активності для подальшого аналізу та аудиту.

5. ***Управління сесіями:***

- Контроль активних сесій користувачів з можливістю примусового завершення у разі підозрілої активності.

- Налаштування часу автоматичного завершення сесії для підвищення безпеки.

6. ***Інтеграція з зовнішніми сервісами:***

- Можливість інтеграції з іншими системами автентифікації, такими як Google, Facebook або корпоративні LDAP-сервери, для спрощення процесу входу користувачів.

7. ***Відновлення доступу:***

- Надання механізмів відновлення доступу у випадку забутого пароля через електронну пошту або інші засоби зв'язку.

Технологічні аспекти реалізації:

Для реалізації системи управління обліковими записами у веб-бібліотеці використовуються сучасні технології та фреймворки. Фронтенд частина розроблена з використанням React та Tailwind CSS, що забезпечує динамічний та адаптивний інтерфейс користувача. Бекенд реалізовано на базі FastAPI, що дозволяє швидко та ефективно обробляти запити користувачів. Дані користувачів зберігаються у базі даних PostgreSQL, яка забезпечує надійне та безпечне зберігання інформації.

Для забезпечення безпеки даних користувачів використовуються сучасні методи шифрування та хешування паролів. Крім того, система підтримує можливість двофакторної автентифікації для підвищення рівня захисту облікових записів.

Важливим аспектом є забезпечення відповідності системи управління обліковими записами сучасним стандартам безпеки та конфіденційності даних, таким як GDPR. Це включає надання користувачам можливості керувати своїми персональними даними та забезпечення прозорості у використанні їхньої інформації.

Таким чином, система управління обліковими записами користувачів у веб-бібліотеці забезпечує надійний та безпечний доступ до ресурсів, персоналізований досвід для користувачів та ефективне адміністрування системи.

2.4.1 Інструменти та технології для управління даними у фронтенді

Для ефективного проектування та управління інформаційним забезпеченням у фронтенді використовуються різноманітні інструменти та технології:

- JavaScript фреймворки та бібліотеки: React, Angular, Vue.js та інші, які надають засоби для управління станом додатку та взаємодії з даними.
- CSS фреймворки: Bootstrap, Tailwind CSS для швидкого та ефективного створення адаптивних інтерфейсів користувача.
- Інструменти для управління станом: Redux, MobX для централізованого управління станом додатку.
- Інструменти для тестування: Jest, Cypress для забезпечення якості коду та коректної роботи з даними.

2.4.2 Аналіз моделі

Аналіз моделі мережі Петрі дозволяє оцінити ефективність та надійність системи, виявити можливі проблеми та оптимізувати процеси.

Основні аспекти аналізу:

- Досяжність (reachability): визначення, чи можна досягти певного стану системи з початкового стану шляхом послідовності переходів.
- Живучість (liveness): перевірка, чи може кожен перехід спрацювати при певних умовах, що запобігає "зависанню" системи.
- Безпечність (safety): оцінка, чи не перевищує кількість маркерів у будь-якому місці певного значення, що може призвести до перевантаження або некоректної роботи.

- Зворотність (reversibility): перевірка, чи може система повернутися до початкового стану після виконання певної послідовності переходів.

Методи аналізу:

- Побудова графа досяжності: визначення всіх можливих станів системи та переходів між ними для оцінки досяжності та живучості.
- Аналіз інваріантів: використання математичних методів для перевірки збереження певних властивостей системи, таких як кількість ресурсів.
- Моделювання сценаріїв: імітація різних сценаріїв роботи системи для виявлення потенційних проблем та оцінки продуктивності.

2.5. Вибір технологічного стеку для фронтенду та обґрунтування його використання

При розробці електронної бібліотеки вибір відповідного технологічного стеку для фронтенду є критично важливим для забезпечення ефективної роботи, зручності користування та подальшої підтримки системи.

Обрані технології:

1. HTML5 та CSS3: основні технології для створення структури та стилізації веб-сторінок.
2. JavaScript: мова програмування, що забезпечує інтерактивність та динамічну взаємодію з користувачем.
3. React: бібліотека для побудови інтерфейсів користувача, розроблена компанією Facebook.
4. Redux: бібліотека для керування станом додатка, що часто використовується разом з React.
5. Webpack: модульний збирач, який оптимізує та об'єднує ресурси проєкту.
6. Babel: транспайлер, що дозволяє використовувати сучасні можливості JavaScript, забезпечуючи сумісність з різними браузерами.

Обґрунтування вибору:

- HTML5 та CSS3: ці технології є стандартом для розробки веб-інтерфейсів, забезпечуючи адаптивність та кросбраузерну сумісність.
- JavaScript: дозволяє реалізувати динамічну взаємодію з користувачем, обробку подій та маніпуляцію DOM-деревом, що є необхідним для сучасних веб-додатків.
- React: забезпечує компонентний підхід до розробки інтерфейсу, що сприяє повторному використанню коду та спрощує підтримку. React має велику спільноту розробників та багату екосистему інструментів, що прискорює розробку та впровадження нових функцій.
- Redux: дозволяє централізовано керувати станом додатка, що особливо корисно в масштабних проєктах з великою кількістю взаємодій та даних. Це сприяє передбачуваності поведінки додатка та спрощує налагодження.
- Webpack: забезпечує ефективне управління залежностями, оптимізацію розміру файлів та швидкість завантаження сторінок, що покращує загальну продуктивність додатка.
- Babel: дозволяє використовувати сучасні можливості JavaScript, такі як ES6+ синтаксис, забезпечуючи при цьому сумісність з більшістю браузерів, що розширює аудиторію користувачів.

Альтернативні варіанти:

Розглядались також інші фреймворки та бібліотеки, такі як Angular та Vue.js.

- Angular: потужний фреймворк від Google, що надає багато вбудованих функцій. Однак, його складність та крута крива навчання можуть збільшити час розробки для команди, не знайомої з ним.
- Vue.js: легкий та гнучкий фреймворк, який поєднує переваги React та Angular. Проте, менша спільнота та екосистема порівняно з React можуть обмежити доступність ресурсів та інструментів.

Розділ 3. Розробка фронтенду електронної бібліотеки

3.1 Вступ: мета, вимоги до UI/UX та взаємодія з бекендом

Метою розділу є опис процесу розробки фронтенд-частини електронної бібліотеки, що реалізується з використанням React і TypeScript. Інтерфейс користувача (UI) повинен забезпечувати зручний перегляд та пошук книг, авторів та жанрів, а також роботу із сесіями користувача (реєстрація, авторизація, профіль). Вимоги до UI/UX включають інтуїтивну навігацію, адаптивний дизайн під різні розміри екранів (мобільні, планшети, десктопи) та високу чіткість елементів інтерфейсу. Наприклад, для сторінки перегляду списку книг потрібно передбачити фільтрацію за жанрами чи авторами, кнопку для переходу до детальної сторінки книги та індикатори завантаження при очікуванні відповідей від сервера.

Взаємодія з бекендом здійснюється за REST-принципами через HTTP-запити до API-сервера. Фронтенд відповідатиме за виконання CRUD-операцій щодо ресурсів «Книги», «Автори», «Жанри» та управління сесіями користувача. При цьому авторизація відбувається на основі JWT: після успішної авторизації користувач отримує токен, який зберігається на клієнті й додається до заголовків кожного запиту для підтвердження прав доступу. Важливо, що UI має коректно обробляти сценарії, пов’язані з авторизацією: наприклад, при спробі звернення до захищеного ресурсу без токена перенаправляти користувача на сторінку входу або показувати відповідне повідомлення. Загалом, при розробці фронтенду зважено на зручність використання, реактивність інтерфейсу та безшовну взаємодію з бекендом, щоб користувач отримував актуальні дані в режимі реального часу.

3.2 Вибір стеку технологій

Для реалізації клієнтського застосунку обрано сучасний стек технологій, що забезпечує типобезпечність, гнучкість та продуктивність розробки:

- **TypeScript** — розширення мови JavaScript, яке додає статичну типізацію. Використання TypeScript допомагає виявляти помилки на етапі компіляції та робить код більш самодокументованим, що важливо для великого проєкту.
- **React 18** — популярна бібліотека для побудови інтерфейсу, що дозволяє створювати компонентний UI та використовувати сучасні підходи (хуки, контекст). Версія 18 включає вдосконалення асинхронного рендерингу та інструменти для паралельних оновлень стану.
- **Redux Toolkit (RTK)** — офіційний набір інструментів для Redux, який спрощує створення стора та стану. RTK надає готові API (createSlice, createAsyncThunk тощо) для зменшення шаблонного коду при управлінні станом застосунку та забезпечує інтеграцію з асинхронними операціями.
- **Tailwind CSS** — утилітарний CSS-фреймворк, що дає змогу швидко створювати адаптивний і узгоджений дизайн через класи. Замість власних стилів ми обираємо Tailwind для послідовного застосування UI-концепції Atomic Design і спрощення адаптації під різні розміри екрану за допомогою вбудованих брейкпоінтів (наприклад, sm:, md:, lg:).
- **React Router (v6)** — система маршрутизації для React, яка дозволяє налаштувати навігацію між сторінками SPA. Вона підтримує динамічні маршрути, вкладені маршрути, а також захист приватних сторінок.
- **React Hook Form та YUP** — бібліотека React Hook Form призначена для ефективного керування станом форм без зайвих перерендерів, а YUP — для опису схем валідації полів форми. Завдяки поєднанню цих інструментів легко реалізувати перевірку коректності даних на клієнті (наприклад, перевірка email, мінімальної довжини паролю та ін.).

- **Axios** — HTTP-клієнт, що спрощує виконання запитів до REST API. Axios підтримує перехоплювачі запитів/відповідей, що корисно для додавання JWT-токена в заголовок кожного запиту та обробки помилок.
- **Jest + React Testing Library** — комбінація для тестування: Jest як тестовий рантайм дозволяє писати юніт-тести і запускати їх із відлагодженням, а React Testing Library — інструменти для тестування React-компонентів у спосіб, максимально наближений до користувацької взаємодії.

Такий стек забезпечує підтримку кращих практик розробки. TypeScript і Redux Toolkit допомагають зменшити кількість помилок, Tailwind CSS дозволяє стандартно оформлювати компоненти, а Axios і React Router закладають основу стабільної мережевої взаємодії та навігації. У сукупності, використання цих технологій відповідає сучасним вимогам до фронтенд-розробки та полегшує подальшу підтримку і розширення проєкту.

3.3 Архітектура клієнтського застосунку

При розробці застосунку було обрано модульно-орієнтовану архітектуру з розділенням на компоненти й слої. Основна структура проєкту у корні src/ виглядає так(Рис 3.1):

src/

├─ assets/ # Статичні файли (зображення, шрифти, стилі Tailwind тощо)

├─ components/ # Загальні UI-компоненти (атом, молекули за Atomic Design)

├─ features/ # Функціональні модулі (слайси Redux, сторінки, специфічні компоненти)

├─ services/ # API-клієнти і сервіси взаємодії з бекендом

├─ store/ # Конфігурація Redux store, ред’юсери

```
├─ hooks/          # Користувацькі React-хуки
├─ utils/          # Утилітні функції та допоміжні модулі
├─ types/          # Загальні TypeScript-типи та інтерфейси
├─ App.tsx         # Кореневий компонент
└─ index.tsx       # Точка входу (рендер React)
```

Рис. 3.1 Компоненти Архітектури застосунку

3.3.1 Принципи Clean Architecture

Ми дотримуємося принципів «Clean Architecture», щоб зберегти код стійким до змін. Логіку взаємодії з бекендом і перетворення даних виділено в окремий шар (наприклад, у services/), бізнес-логіку (роботу зі станом, валідацію) – в features/ та store/, а суто відображення (UI) – в components/.

Наприклад, якщо є сторінка списку книг, то дані про книги отримуємо в слайсі booksSlice (налаштованому через RTK у папці features/books/), який на основі асинхронного запиту передає їх компоненту BooksList (у components/ або в межах features/books/). Таким чином, компонент «сплавляє» дані через пропси чи селектори Redux, не займаючись самостійно викликом API.

Такий розподіл підвищує модульність: наприклад, тестувати бізнес-логіку та UI-компоненти зручно окремо. Якщо згодом зміниться структура API або бібліотека для HTTP-запитів, достатньо відредагувати код у шарі services/, не зачіпаючи логіки компонентів.

3.3.2 Організація компонентів за Atomic Design

Для організації UI-компонентів застосовано підхід Atomic Design. Він передбачає класифікацію компонентів за рівнями «атомів», «молекул», «організмів» (Рис. 3.1):

- **Атоми** — найменші компоненти без внутрішньої структури, наприклад: кнопка (Button), текстове поле (Input), заголовок (Title), іконка (Icon). Вони повторно використовуються по всьому застосунку і не містять бізнес-логіки.
- **Молекули** — комбінації атомів, наприклад: поле з підписом (InputWithLabel), кнопкова група (ButtonGroup), картка книги зі зображенням і заголовком (BookCard атом + Title + Button), або форма пошуку (SearchBar із полем і кнопкою).
- **Організми** — більш складні структурні блоки, наприклад: набір навігаційних посилань (Navbar зі складними молекулами), або повна форма

входу (LoginForm з кількома полями), список книг (BooksList, що складається з багатьох BookCard).

Такий підхід забезпечує чітку повторно використовувану основу UI. Наприклад, якщо треба змінити стиль усіх кнопок, достатньо оновити компонент Button. Компоненти впорядковано за каталогами, наприклад:

```
components/
  ├─ atoms/
  │    ├─ Button.tsx
  │    ├─ Input.tsx
  │    └─ ...
  ├─ molecules/
  │    ├─ SearchBar.tsx
  │    ├─ BookCard.tsx
  │    └─ ...
  └─ organisms/
       ├─ Navbar.tsx
       ├─ BookList.tsx
       └─ ...
```

Рис. 3.2 Структура проєкту React/TypeScript

3.3.3 Структура сторінок та маршрутів

Кожна «сторінка» застосунку (наприклад, HomePage, BookDetailPage, AuthorPage, LoginPage) винесена у файл, що поєднує компоненти й доступ до стану. Сторінки розміщуються у папці features/назваFeature/ або в окремій папці pages/, залежно від стилю. Наприклад:

```
features/
├─ books/
│   ├─ booksSlice.ts
│   ├─ BooksListPage.tsx
│   ├─ BookDetailPage.tsx
```

```
│   └── components/
│       ├── BookCard.tsx
│       └── ...
├── authors/
│   └── ...
└── users/
    └── ...
```

У такий спосіб усі файли, пов’язані з певним функціоналом (строго по інтересам), згруповані разом. Це відповідає підходу Ducks/Feature Folder та відповідає принципам SoC (Separation of Concerns).

3.4 Взаємодія з API: HTTP-клієнт Axios і CRUD-операції

Для роботи з серверними даними в компоненті створено модулі у папці services/ чи безпосередньо у фіча-слайсах, де використовують Axios для запитів до REST API. Спочатку налаштовано екземпляр axiosInstance з базовим URL та заголовками:

```
// src/services/api.ts
import axios from 'axios';

const apiClient = axios.create({
  baseURL: 'https://api.example.com',  // адреса бекенду
  headers: {
    'Content-Type': 'application/json',
  },
});

// Перехоплювач запитів для додавання JWT-токена з локального сховища
apiClient.interceptors.request.use(config => {
  const token = localStorage.getItem('authToken');
  if (token) {
```

```
    config.headers!['Authorization'] = `Bearer ${token}`;
  }
  return config;
});

export default apiClient;
```

Цей apiClient тепер використовують у функціях для кожної сутності. Наприклад, запити CRUD для книг оформлені так:

```
// src/services/booksService.ts
import apiClient from './api';
import { Book } from '../types';

export const fetchBooks = async (): Promise<Book[]> => {
  const response = await apiClient.get('/books');
  return response.data;
};

export const fetchBookById = async (id: string): Promise<Book> => {
  const response = await apiClient.get(`/books/${id}`);
  return response.data;
};

export const createBook = async (bookData: Partial<Book>): Promise<Book> =>
{
  const response = await apiClient.post('/books', bookData);
  return response.data;
};
```

```
export const updateBook = async (id: string, bookData: Partial<Book>): Promise<Book> => {
  const response = await apiClient.put(`/books/${id}`, bookData);
  return response.data;
};

export const deleteBook = async (id: string): Promise<void> => {
  await apiClient.delete(`/books/${id}`);
};
```

Тут демонструється повний цикл: GET для зчитування списку та окремої книги, POST для створення, PUT для оновлення та DELETE для видалення(Рис. 3.4). Аналогічні сервіси реалізуються для авторів (/authors), жанрів (/genres) та для сесій/користувача (/users, /sessions).

Рис. 3.3 Шари React-застосунку

У самих компонентах чи асинхронних ред'юсерах (thunk) запити виконуються так (приклад з RTK createAsyncThunk):

```
// src/features/books/booksSlice.ts
export const loadBooks = createAsyncThunk('books/loadAll', async () => {
  const books = await fetchBooks();
  return books;
});

const booksSlice = createSlice({
  name: 'books',
  initialState: { items: [] as Book[], status: 'idle' },
  reducers: {},
  extraReducers: builder => {
    builder
      .addCase(loadBooks.pending, (state) => {
        state.status = 'loading';
      })
      .addCase(loadBooks.fulfilled, (state, action) => {
        state.status = 'succeeded';
        state.items = action.payload;
      })
      .addCase(loadBooks.rejected, (state) => {
        state.status = 'failed';
      });
  },
});
```

Завдяки Axios та RTK на рівні коду мінімізуються шаблонні операції: наприклад, помилки обробляються автоматично і можуть передаватися для показу повідомлення. Таке розділення дозволяє легко міняти дані: якщо API-ендпойнт зміниться, досить оновити URI в services/, а логіку в компонентах можна не чіпати. Це підвищує надійність та гнучкість реалізації.

Рис. 3.4 Цикл CRUD-операцій

3.5 Управління станом через Redux Toolkit

Для глобального стану застосунку використовуємо **Redux Toolkit** (RTK) (Рис. 3.5). Він полегшує налаштування стору та написання ред'юсерів. Основні кроки:

1. **Конфігурація Store:** у файлі store.ts створюється Redux Store за допомогою configureStore, де підключаються всі слайси.

```
// src/store/index.ts
import { configureStore } from '@reduxjs/toolkit';
import booksReducer from '../features/books/booksSlice';
import userReducer from '../features/users/userSlice';
// ...
```

```
export const store = configureStore({
  reducer: {
    books: booksReducer,
    user: userReducer,
    // додаткові ред'юсери
  },
});

export type RootState = ReturnType<typeof store.getState>;
export type AppDispatch = typeof store.dispatch;
```

Store обгорнуто в <Provider> у точці входу (index.tsx), щоб компоненти могли отримувати доступ до глобального стану через хуки useSelector та useDispatch.

2. **Слайси (slice) як окремі модулі стану:** Кожен набір пов'язаної логіки виокремлено в окремий слайс з початковим станом і ред'юсерами. Наприклад, userSlice.ts відповідає за стан автентифікації:

```
// src/features/users/userSlice.ts
import { createSlice, createAsyncThunk } from '@reduxjs/toolkit';
import { loginUserApi, logoutApi } from '../../services/authService';

interface UserState {
  token: string | null;
  status: 'idle' | 'loading' | 'error';
}
const initialState: UserState = { token: null, status: 'idle' };

export const loginUser = createAsyncThunk(
  'user/login',
  async (credentials: { email: string; password: string }) => {
```

```
    const response = await loginUserApi(credentials);
    return response.token;
  }
);

const userSlice = createSlice({
  name: 'user',
  initialState,
  reducers: {
    logout(state) {
      state.token = null;
    },
  },
  extraReducers: builder => {
    builder
      .addCase(loginUser.pending, (state) => {
        state.status = 'loading';
      })
      .addCase(loginUser.fulfilled, (state, action) => {
        state.status = 'idle';
        state.token = action.payload;
      })
      .addCase(loginUser.rejected, (state) => {
        state.status = 'error';
      });
  },
});

export const { logout } = userSlice.actions;
```

```
export default userSlice.reducer;
```

Тут **createAsyncThunk** допомагає запускати асинхронні операції (HTTP-запити) і автоматично створює pending/fulfilled/rejected етапи для ред'юсерів. Це полегшує обробку стану завантаження.

3. **Селектори та хук useAppSelector/useAppDispatch:** Для доступу до стану використовують хуки. Наприклад, в компоненті логіну можна отримати статус запиту:

```
const dispatch = useAppDispatch();
const status = useAppSelector(state => state.user.status);
const handleLogin = (data) => {
  dispatch(loginUser(data));
};
```

Такий підхід дозволяє компонентам підписуватися на зміни певних частин стану та автоматично оновлюватися при їх зміні.

Рис. 3.5 ReduxToolkit store slices

В результаті Redux Toolkit спрощує шаблонний код: нам не потрібно писати вручну типи екшенів чи створювати копії стану для іммутабельності (RTK автоматично використовує Immer для оновлення). Крім того, RTK підтримує **RTK**

Query – ще один інструмент для запитів даних, але у цьому проєкті ми використовуємо традиційні асинхронні thunks з Axios, аби зберегти більший контроль над логікою.

3.6 Маршрутизація з React Router

Маршрутизація реалізується через **React Router v6**. У App.tsx налаштовано базові маршрути так:

```
import { BrowserRouter as Router, Routes, Route, Navigate } from 'react-router-dom';
import HomePage from './features/home/HomePage';
import BooksPage from './features/books/BooksListPage';
import BookDetailPage from './features/books/BookDetailPage';
import LoginPage from './features/users/LoginPage';
import NotFoundPage from './features/NotFoundPage';
import PrivateRoute from './components/PrivateRoute';

function App() {
 return (
  <Router>
   <Routes>
    <Route path="/" element={<HomePage />} />
    <Route path="/books" element={<BooksPage />} />
    <Route path="/books/:id" element={<BookDetailPage />} />
    <Route path="/login" element={<LoginPage />} />
    {/* Приклад захищеного маршруту для сторінки профілю */}
    <Route path="/profile" element={
     <PrivateRoute>
      <ProfilePage />
     </PrivateRoute>
    } />
```

```
      <Route path="*" element={<NotFoundPage />} />
    </Routes>
  </Router>
 );
}
```

- **Основні маршрути:** Вказані шляхи /books, /books/:id, /login тощо. :id – динамічний параметр для ідентифікатора книги.
- **Захищені маршрути:** Компонент PrivateRoute обгорткою перевіряє наявність авторизації. Наприклад:

```
// src/components/PrivateRoute.tsx
import { useAppSelector } from '../store/hooks';
import { Navigate } from 'react-router-dom';

interface PrivateRouteProps {
  children: JSX.Element;
}

export default function PrivateRoute({ children }: PrivateRouteProps) {
  const token = useAppSelector(state => state.user.token);
  return token ? children : <Navigate to="/login" replace />;
}
```

Якщо token відсутній, користувача перенаправлять на сторінку входу. Якщо ж авторизований, дозволяємо доступ до дочірнього компонента(Рис. 3.6).

Рис. 3.6 Налаштування маршутизації в SPA

- **Навігація:** Для посилань між сторінками використовуються компоненти <Link> і <NavLink> (з бібліотеки React Router) та хук useNavigate для програмного переходу після певних дій (наприклад, після успішного логіну робимо navigate('/')).
- **Nested Routes (вкладені маршрути):** Якщо треба створити багаторівневі маршрути (наприклад, /admin/books в адміністартивній частині), їх можна також визначити через вкладені <Route>.
- **Маршрутизація та Redirections:** Після певних дій (наприклад, створення нової книги) часто здійснюємо перенаправлення на іншу сторінку. Використовується useNavigate в компоненті:
- const navigate = useNavigate();
- const handleSubmit = async () => {
- await dispatch(createBook(newBookData));

- navigate('/books'); // повернутись до списку книг після створення
- };

Завдяки React Router забезпечується SPA-навігація, а структура маршрутів чітко віддзеркалює логіку застосунку: сторінки відповідно до сутностей та операцій.

3.7 Робота з формами: React Hook Form та YUP

Форми в застосунку використовуються для реєстрації користувача, авторизації, створення/редагування книг, авторів тощо. Замість керування станом кожного поля через useState, застосовано **React Hook Form** (RHF). RHF суттєво знижує кількість непотрібних ререндерів та надає прості API для реєстрації полів.

Приклад створення форми для додавання книги(Рис. 3.7) з валідатором на поля за допомогою YUP:

```
// src/features/books/BookForm.tsx
import React from 'react';
import { useForm } from 'react-hook-form';
import { Book } from '../../types';
import { yupResolver } from '@hookform/resolvers/yup';
import * as Yup from 'yup';

interface BookFormInputs {
  title: string;
  author: string;
  year: number;
  genre: string;
}

const schema = Yup.object({
  title: Yup.string().required('Вкажіть назву книги'),
```

```
  author: Yup.string().required('Вкажіть автора'),
  year: Yup.number().min(0, 'Невірний рік').integer('Рік має бути цілим').required(),
  genre: Yup.string().required('Вкажіть жанр'),
}).required();

export const BookForm: React.FC = () => {
  const { register, handleSubmit, formState: { errors } } = useForm<BookFormInputs>({
    resolver: yupResolver(schema),
  });

  const onSubmit = (data: BookFormInputs) => {
    // dispatch(createBook(data)) або інші дії
    console.log('Форма валідна, дані:', data);
  };

  return (
    <form onSubmit={handleSubmit(onSubmit)} className="flex flex-col gap-4">
      <div>
        <label>Назва</label>
        <input {...register('title')} className="border p-2 w-full" />
        {errors.title && <p className="text-red-600">{errors.title.message}</p>}
      </div>
      <div>
        <label>Автор</label>
        <input {...register('author')} className="border p-2 w-full" />
```

```
            {errors.author && <p
className="text-red-600">{errors.author.message}</p>}
          </div>
          <div>
            <label>Рік</label>
            <input type="number" {...register('year')} className="border p-2 w-full" />
            {errors.year && <p className="text-red-600">{errors.year.message}</p>}
          </div>
          <div>
            <label>Жанр</label>
            <input {...register('genre')} className="border p-2 w-full" />
            {errors.genre && <p
className="text-red-600">{errors.genre.message}</p>}
          </div>
          <button type="submit" className="bg-blue-500 text-white p-2">
            Зберегти
          </button>
        </form>
      );
    };
```

Пояснення коду:

- **Схема валідації:** За допомогою YUP описуємо правила (поле title — обов’язкове, тощо). yupResolver підключає цю схему до React Hook Form.
- **Реєстрація полів:** Хук register зв’язує HTML-елементи з формою. Помилки валідації доступні в errors.
- **UI:** Ми виводимо повідомлення про помилки під відповідним полем, наприклад {errors.title.message}.

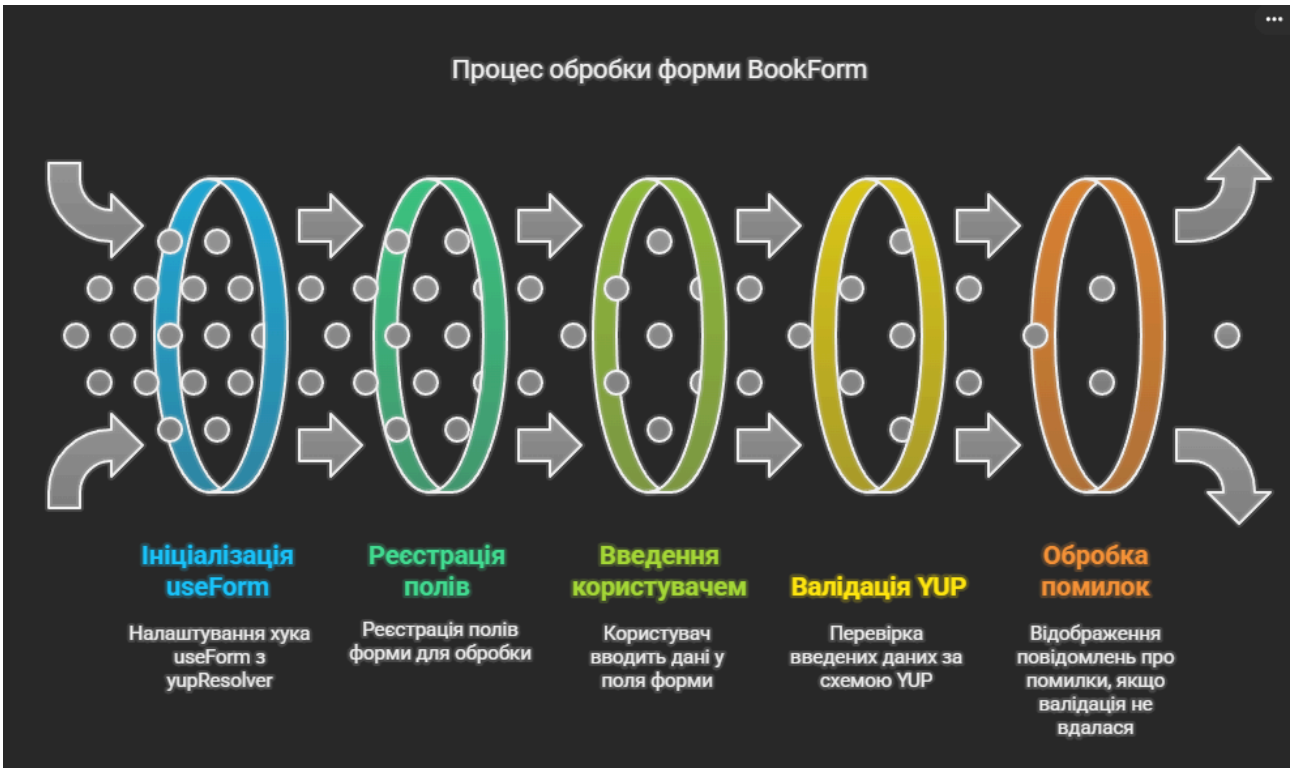


Рис. 3.7 Процесс обробки BookForm

Такий підхід гарантує, що дані, які відправляються на бекенд, вже пройшли первинну перевірку на коректність (наприклад, email-адреса відповідає шаблону, обов’язкові поля заповнені). Він також покращує UX: помилка валідації підсвічується відразу поруч з полем та не потребує повної перезавантаження сторінки. React Hook Form також добре інтегрується з TypeScript, завдяки вказаному інтерфейсу BookFormInputs ми маємо автодоповнення та перевірку типів у самій формі.

3.8 Авторизація і безпека через JWT

Серверне API використовує JWT для автентифікації користувачів. Під час авторизації (логіну) клієнт відправляє облікові дані (email, password) на бекенд (POST /sessions), а в разі успіху отримує JWT-токен. Далі токен зберігається на клієнті (наприклад, у localStorage) і додається до заголовків кожного запиту (див. apiClient.interceptors у розділі про Axios).

Основні аспекти реалізації:

- **Зберігання токена:** Використано localStorage.setItem('authToken', token). Такий підхід дозволяє зберегти сесію між перезавантаженнями. Варто враховувати, що дані у localStorage потенційно доступні через JS, тому слід уникати зберігання чутливої інформації (лише токен).
- **Додавання токена:** Перехоплювач Axios (див. 3.4) автоматично додає Authorization: Bearer <token> до кожного запиту. Таким чином, усі захищені ендпоїнти (наприклад, створення книги /books з авторизацією) приймають токен і перевіряють права користувача.
- **Захищені запити:** При спробі виконати запит без дійсного токену сервер відповідає помилкою (наприклад, 401 Unauthorized). У frontend це обробляється через блоки try/catch. Наприклад:

```
try {
  const newBook = await createBookApi(data);
  dispatch(addBook(newBook));
} catch (error) {
  if (axios.isAxiosError(error) && error.response?.status === 401) {
    dispatch(logout()); // якщо токен недійсний, виконуємо логаут
  }
}
```

Якщо токен прострочений чи відсутній, користувача можна переадресувати на сторінку логіну.

- **Маршрутизація із захистом:** Як показано вище (3.6), приватні маршрути охороняються компонентом PrivateRoute, який перевіряє наявність токена в стані.
- **Вихід (logout):** При виході з акаунта видаляється токен з localStorage та з Redux-стану. Це робиться через ред'юсер logout(), наприклад в userSlice. Компонент логіну чи профілю використовує цю дію при натисканні «Вихід».

Рис. 3.8 Процесс авторизації користувача

Таким чином, використання JWT забезпечує просту, але ефективну систему контролю доступу(Рис. 3.8). Всі HTTP-запити до захищених ресурсів відправляються лише з коректним токеном, а UI чітко реагує на зміни авторизації (відображаючи меню профілю чи форму входу залежно від стану).

3.9 Стилізація UI: Atomic Design та адаптивність

Для стилізації інтерфейсу застосовано **Tailwind CSS** в поєднанні з концепцією Atomic Design, що описано вище. Ключові моменти:

- **Утилітарний підхід Tailwind:** Замість написання власного CSS для кожного компонента ми використовуємо класи Tailwind прямо у JSX. Наприклад:

  <button className="bg-green-500 hover:bg-green-600 text-white py-2 px-4 rounded">

```
  Зберегти
</button>
```

Такий підхід робить стилі явними та повторюваними. Tailwind також дозволяє легко робити інтерфейс адаптивним: наприклад, className="p-2 sm:p-4 lg:p-6" означає, що на малих екранах (sm:) падінг p-4, а на великих (lg:) – p-6. Ми дотримуємось **mobile-first** підходу: стилі без префіксу відповідають мобільній версії, а префікси sm:, md:, lg: додають стилі на більших екранах.

- **Atomic Design на практиці:** У компоненті наприклад, елемент "картка книги" (BookCard) можна будувати як молекулу:

```
// molecules/BookCard.tsx
const BookCard = ({ book }) => (
    <div className="border rounded shadow p-4 hover:shadow-lg transition">
   <h3 className="text-xl font-semibold">{book.title}</h3>
   <p className="text-gray-600">{book.author}</p>
    <button className="mt-2 bg-blue-500 text-white px-3 py-1 rounded">Деталі</button>
 </div>
);
```

Тут використано атоми: <h3> з текстом (можливо окремий компонент Title), кнопка (Button – окремий атом). Це дозволяє повторно використати Button й інші атоми в інших місцях.

- **Темна тема (опціонально):** Tailwind підтримує темний режим (dark: класи) для адаптивності дизайну під уподобання користувача. Наприклад:

```
<div className="bg-white dark:bg-gray-800 text-gray-900 dark:text-gray-100">
  ...
```

```
</div>
```

Це може бути корисно для UI/UX, але реалізацію вирішують згідно специфікацій дипломної роботи.

- **Відповідність дизайну:** Для збереження консистентності ми слідуємо єдиному стилю, наприклад, визначаємо у конфігурації Tailwind кольорову палітру та стиль шрифтів проєкту. Це гарантує, що різні компоненти виглядають уніфіковано.

Загалом, поєднання Tailwind із структурою Atomic Design забезпечує модульність та повторюваність елементів інтерфейсу. Компоненти можна швидко змінювати, коректуючи лише класи Tailwind, без зміни логіки.

3.10 Тестування (unit, integration, e2e)

Для забезпечення якості коду й упевненості в правильності роботи застосунку реалізовано різні рівні тестів.

- **Unit-тести:** Використовуються для окремих функцій і ред’юсерів. Приклад: перевірка роботи slice-ред’юсера для книг у booksSlice.test.ts:

```
import booksReducer, { loadBooks } from './booksSlice';

test('should handle initial state', () => {
  expect(booksReducer(undefined, { type: 'unknown' })).toEqual({
    items: [],
    status: 'idle'
  });
});

test('should handle loadBooks.fulfilled', () => {
  const initialState = { items: [], status: 'loading' };
  const booksData = [{ id: '1', title: 'Test Book' }];
  const action = { type: loadBooks.fulfilled.type, payload: booksData };
```

```
  const state = booksReducer(initialState, action);
  expect(state.items).toEqual(booksData);
  expect(state.status).toBe('succeeded');
});
```

Тут ми перевіряємо, що початковий стан коректний, а при отриманні дії loadBooks.fulfilled дані додаються в items.

- **Integration-тести компонентів:** Використовуємо Jest разом з **React Testing Library (RTL)** для тестування візуальних компонентів. RTL заохочує тестувати компоненти так, ніби користувач взаємодіє з UI. Наприклад, тест для компоненту списку книг, який показує «Завантаження...» до моменту отримання даних:

```
import { render, screen } from '@testing-library/react';
import { Provider } from 'react-redux';
import configureStore from 'redux-mock-store';
import BooksListPage from './BooksListPage';

const mockStore = configureStore([]);

test('shows loading indicator while fetching books', () => {
  const store = mockStore({ books: { items: [], status: 'loading' } });
  render(
    <Provider store={store}>
      <BooksListPage />
    </Provider>
  );
  expect(screen.getByText(/Завантаження/i)).toBeInTheDocument();
});
```

У цьому тесті ми створюємо макет store з певним станом (status: 'loading'), рендеримо компонент і перевіряємо, що текст «Завантаження» відображається. Такі тести інтерґують Redux-store та компонент.

- **e2e-тести:** Для перевірки повної роботи застосунку (end-to-end) можна використовувати інструменти на кшталт Cypress чи Playwright. Припустимо, сценарій: користувач заходить на сайт, реєструється, додає нову книгу та бачить її в списку. Виглядає схематично:

1. Відкрити сторінку /login та зареєструвати новий аккаунт.
2. Перейти до /books/new та заповнити форму нової книги.
3. Після відправки форми переконатися, що редіректнув назад на /books.
4. У списку знайти додану книгу і перевірити, що її назва відображається.

Поки e2e-тести не входять до складу пакету React Testing Library, однак спрощену версію можна робити скриптами за допомогою Puppeteer чи Cypress, запускати в CI для ключових бізнес-сценаріїв.

Інтеграція тестів в процес розробки гарантує, що при подальших змінах (оновлення бібліотек, поправки в коді) базовий функціонал не зламається. У CI (далі) перед збіркою проходять усі тести, тож проблеми виявляються раніше.

3.11 Збірка і CI/CD

Після розробки фронтенд-код зібрано з використанням **бандлера** (наприклад, Vite або Create React App). Усі залежності компілюються, мінімізуються й пакуються в оптимізований набір для виробничого середовища (npm run build генерує папку build/ або dist/).

Автоматизація CI/CD(Рис. 3.9)

Для безперервної інтеграції та деплою налаштовано **GitHub Actions** (або іншу CI/CD систему). Зразковий workflow:

name: CI/CD

```
on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]

jobs:
  build-and-test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2

      - name: Setup Node.js
        uses: actions/setup-node@v2
        with:
          node-version: '18'

      - name: Install dependencies
        run: npm ci

      - name: Run tests
        run: npm test -- --watchAll=false

      - name: Build production bundle
        run: npm run build

      - name: Deploy to Netlify
        uses: nwtgck/actions-netlify@v1
        with:
```

publish-dir: 'build'

production-deploy: 'true'

1. **Перевірка коду:** При кожному push в main та PR, система автоматично виконує тести (npm test). Якщо тести не пройшли, збірка блокується.

2. **Лінтинг:** Можна додати кроки npm run lint та npm run format:check, щоб перевіряти відповідність коду стилю.

3. **Складання та деплой:** При успішному проходженні тестів відбувається збірка проєкту, а потім публікація на сервері (наприклад, Netlify, Vercel чи власному сервісі).

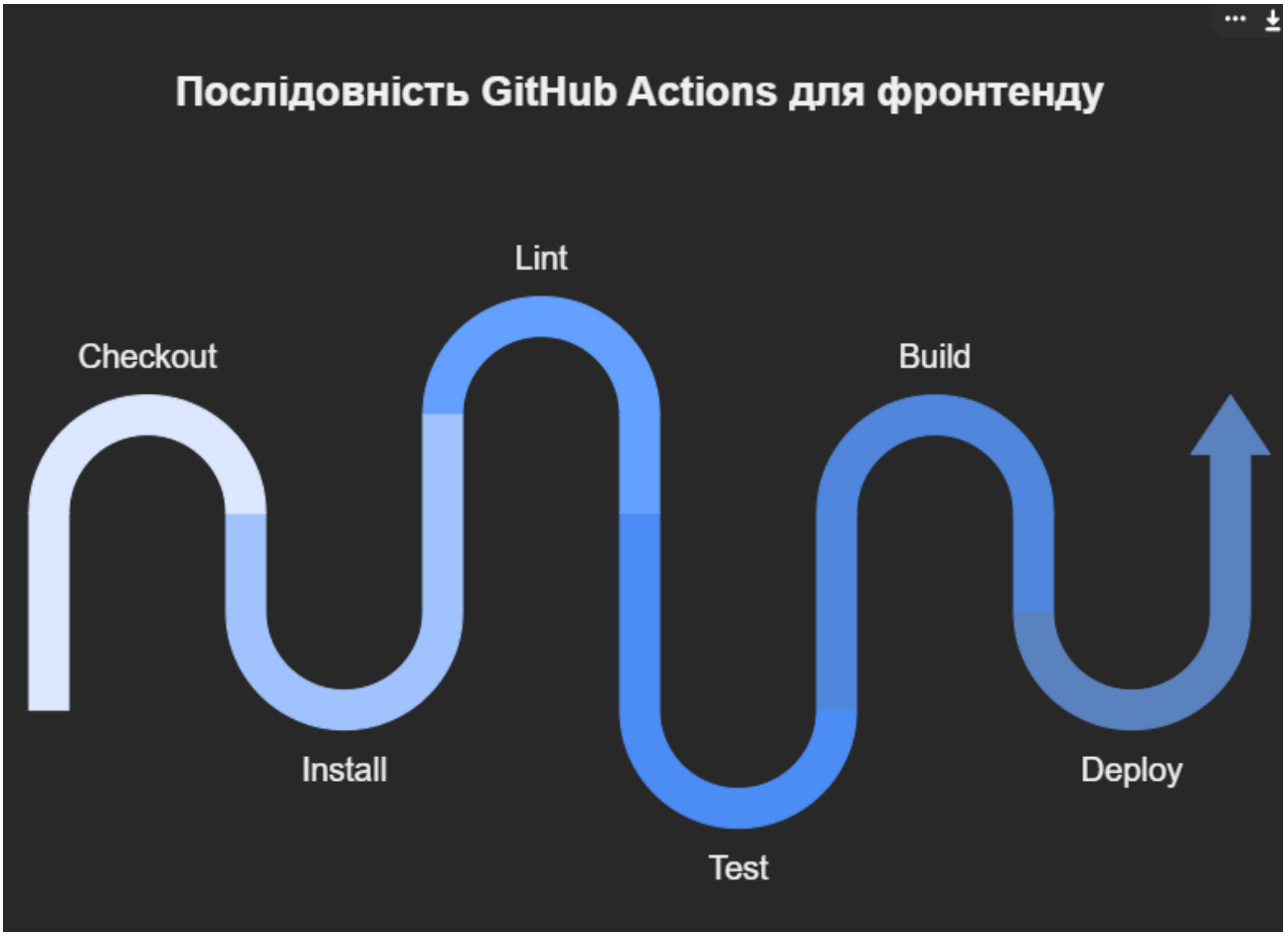


Рис. 3.9 Послідовність GitHub Actions для фронт-енду

Такий CI/CD pipeline забезпечує безперервність роботи: розробники можуть впевнено комітити зміни, знаючи, що помилки будуть виявлені автоматично, а готову версію відразу можна доставити на хостинг.

3.12 Оптимізація продуктивності та доступності

Для забезпечення високої швидкодії та зручності застосовуються різні техніки оптимізації:

- **Code splitting (розбиття коду):** Використовуємо динамічне імпортування компонентів React з React.lazy та Suspense. Наприклад, великий компонент детального перегляду книги можна завантажувати тільки при переході на /books/:id:

```
const BookDetailPage = React.lazy(() => import('./BookDetailPage'));


// в App.tsx
<Suspense fallback={<Loader />}>
  <Route path="/books/:id" element={<BookDetailPage />} />
</Suspense>
```

Це зменшує розмір початкового бандла та пришвидшує початкове завантаження(Рис. 3.10).

- **Оптимізація зображень:** Якщо в бібліотеці є обкладинки книг, їх можна зберігати в оптимізованому форматі (WebP) та змінювати розміри зображень відповідно до пристрою. Для великих картинок можна застосувати lazy-loading (відкладене завантаження при скролі).
- **Memoization:** Якщо компонент перераховує дані багато разів, використовуємо useMemo або useCallback для уникнення зайвих обчислень. Наприклад, список фільтрованих книг, що залежать від великих масивів, можемо мемоізувати, щоб не перерендерювати всі елементи без потреби.
- **Полегшені бібліотеки:** Tailwind CSS додає багато класів, але на етапі збірки невикористані CSS-класи видаляються (purge). Також слід

мінімізувати додаткові залежності: наприклад, не підключати великі UI-бібліотеки, якщо достатньо власних компонентів.

- **Перехват помилок:** Використання React.Suspense і Error Boundaries дозволяє захищати UI від неочікуваних помилок і показувати fallback без повного краху. Це скоріше UX-оптимізація, оскільки покращує стійкість інтерфейсу.

Для **доступності (accessibility)** враховано такі моменти:

- **Семантична розмітка:** Використовуються правильні HTML-теги (<button>, <nav>, <header>, <main>, <form>, <label> тощо), щоб скрінрідери розпізнавали структуру сторінки.
- **ARIA-атрибути:** При потребі додаються атрибути aria-label, aria-describedby або role, особливо коли компонент нестандартний. Наприклад, для кнопки закриття модального вікна можна вказати aria-label="Закрити".
- **Колірна контрастність:** Вибираються кольори тексту та фону з достатньою контрастністю для користувачів з порушеннями зору. Tailwind полегшує це завдяки попередньо підібраним відтінкам (наприклад, text-gray-800 на bg-white).
- **Клавіатурна навігація:** Всі інтерактивні елементи (button, a, input) доступні з клавіатури (натисканням Tab та Enter). Модальні вікна чи динамічно з’явлені елементи мають можливість закриття через клавішу Esc (при необхідності через обробник подій onKeyDown).
- **Фокус:** Залежності Tailwind налаштовуються так, щоб кожен фокусований елемент мав видимий індикатор (outline) або інший візуальний підсвічування, що зручніше для користувачів при навігації без миші.

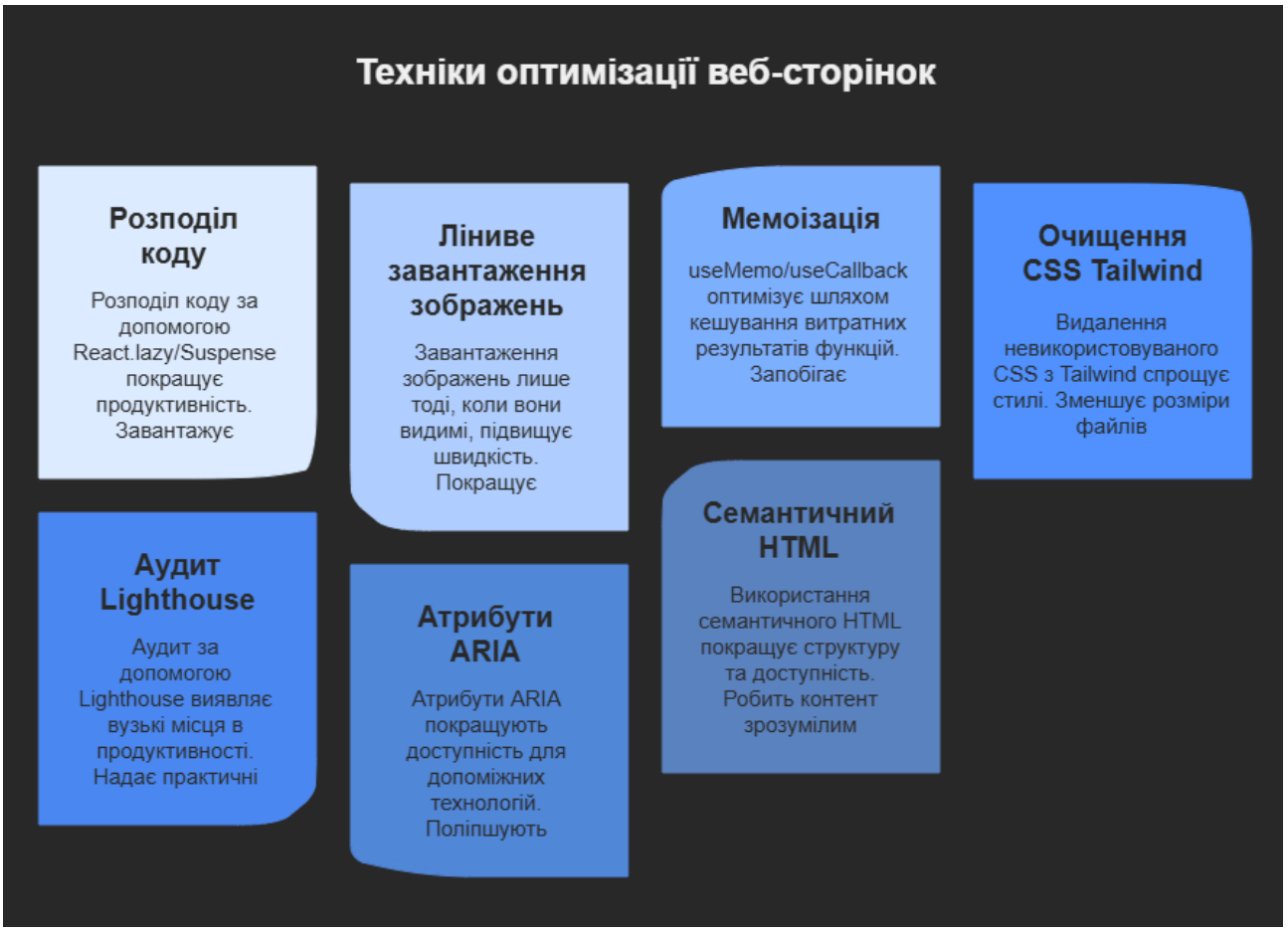


Рис. 3.10 Техныки оптимізації веб-сторінок

Ці заходи допомагають зробити застосунок доступним та зручним як для звичайних користувачів, так і для людей з обмеженнями. Наприклад, на клавіатурі можна переміщуватися між елементами форм та картками книг, а читач екрану озвучуватиме контент завдяки наявним ARIA-міткам.

РОЗДІЛ 4. ЕРГОНОМІЧНІ ПОКАЗНИКИ СИСТЕМИ ЕЛЕКТРОННОЇ БІБЛІОТЕКИ

4.1 Метрики ергономіки UI

4.1.1 Основні концепції usability (визначення ISO, Nielsen Norman Group)

У межах ергономіки інтерфейсу користувача (usability) підкреслюється, наскільки легко і ефективно система дозволяє задовольняти потреби користувача. Згідно з міжнародним стандартом ISO 9241-11 (2018), *usability* визначається як «степінь, у якій продукт може бути використаний певними користувачами для досягнення заданих цілей з ефективністю, продуктивністю та задоволенням у певному контексті використання». Таким чином, критеріями оцінки є результати, час виконання та суб'єктивне задоволення користувача. Згідно з поглядами групи Nielsen Norman, *usability* – це якісна характеристика, що оцінює легкість використання інтерфейсу. Насамперед це стосується «легкості вивчення» інтерфейсу новим користувачем, а також загальної привабливості та інтуїтивності дизайну. За Нільсеном та Норманом, якісна цілісна оцінка *usability* включає такі основні компоненти: **легкість навчання** (Learnability), **ефективність** (Efficiency), **запам'ятовуваність** (Memorability), **частота помилок** (Errors) і **задоволеність** (Satisfaction).

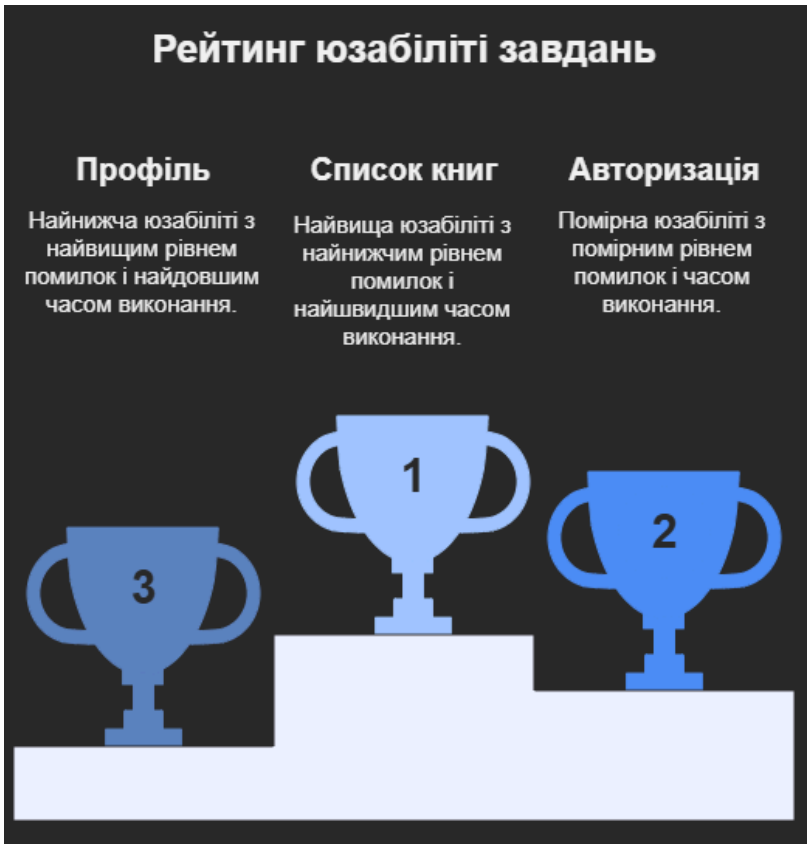


Рис. 4.1.1 Порівняння юзабіліті-метрик трьох ключових інтерфейсів

Ці компоненти взаємодоповнюють одне одного та разом визначають корисність і зручність інтерфейсу. Наприклад, недостатня легкість навчання або висока частота помилок зменшують загальне задоволення від системи. Таким чином, **usability** розглядається не лише як відповідність інтерфейсу потребам користувача, але й як характеристика, що враховує контекст використання системи та емоційний відгук користувачів.

4.1.2 Категорії метрик (Learnability, Memorability, Error Rate, Satisfaction тощо)

Для кількісної оцінки ергономічності UI використовуються різні метрики. **Learnability (легкість навчання)** показує, наскільки швидко новий користувач може засвоїти базові функції системи. Іншими словами, чим простіше виконати перші завдання без попередньої практики, тим вища легкість навчання. За

визначенням, вона оцінює «наскільки легко користувачам виконати прості завдання вперше при першому знайомстві з інтерфейсом». **Memorability (запам'ятовуваність)** характеризує здатність користувача відновити навички після перерви у використанні; наприклад, через місяці або роки без практики, наскільки легко повторно користуватися інтерфейсом. У повсякденному тестуванні запам'ятовуваність вимірюють через порівняння показників часу виконання завдань новими і повернутими користувачами. **Error Rate (частота помилок)** фіксує відсоток або загальну кількість помилок, допущених користувачами при виконанні завдань. Наприклад, при заповненні форми помилка може мати вигляд неправильного введення даних або неправильних дій. Зазвичай **Error Rate** обчислюють як відношення кількості помилок до кількості спроб виконати задачу (помилки/спроби)*100. Високий рівень помилок вказує на проблеми з інтерфейсом, неінтуїтивними елементами чи недостатніми інструкціями. **Satisfaction (задоволеність)** – це суб'єктивний показник, який відображає приємність використання інтерфейсу. Його найчастіше визначають за допомогою опитувальників або шкал оцінки (наприклад, Likert). Наприклад, опитувальник System Usability Scale (SUS) дає узагальнений бал задоволеності від використання системи (про що йтиметься нижче). Крім того, існують спеціальні показники на кшталт **UMUX** чи **CSAT**, що вимірюють враження користувача від продукту загалом. Підсумовуючи: ключові категорії метрик UI-ергономіки включають легкість опанування системи (Learnability), зручність повторного доступу (Memorability), частоту помилок (Error Rate), швидкість і ефективність виконання завдань, а також рівень задоволення користувачів.

4.2 Вимірювання фронтенд-перформансу

4.2.1 Час відгуку та критичні межі (LCP, FID, CLS)

Час відгуку інтерфейсу має вирішальне значення для користувацького досвіду. За дослідженнями Якоба Нільсена, існують прийнятні інтервали затримки: близько **0.1 секунди** сприймається як миттєва реакція, що створює відчуття прямого керування; затримка до **1 секунди** ще не порушує потік мислення користувача, однак вже викликає усвідомлення обробки системою запиту; затримка понад **10 секунд** спричинює значне розчарування, часто змушує користувачів переривати роботу. Отже, критичним є забезпечення відгуку в межах секунди, особливо для інтерфейсних команд та навігації.

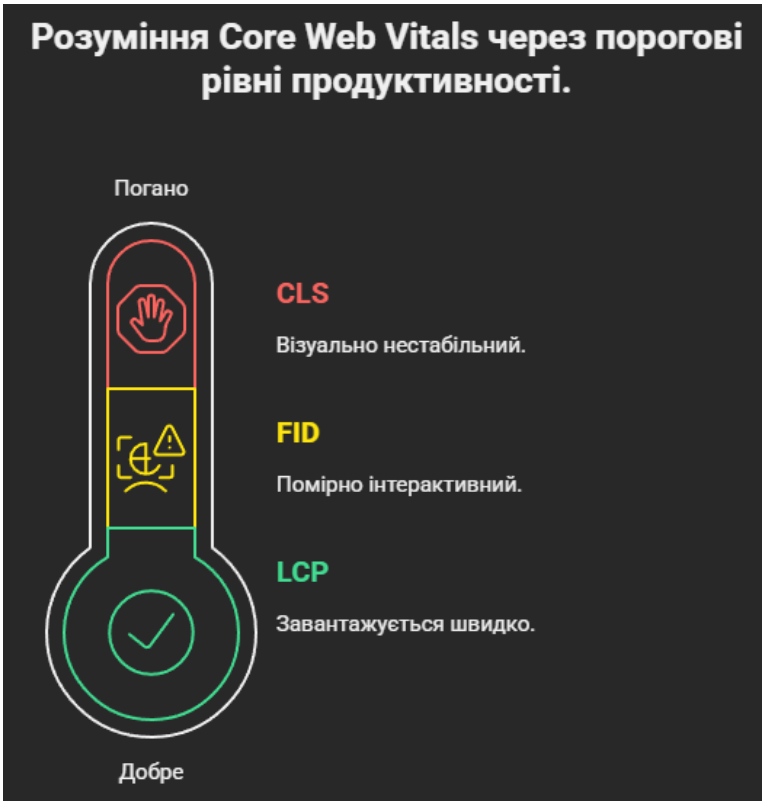


Рис. 4.2.1 Графік критичних меж Core Web Vitals (LCP, FID, CLS)

Найбільш усталеними метриками фронтенд-перформансу є **Core Web Vitals** від Google, що охоплюють різні аспекти завантаження та взаємодії. **LCP (Largest Contentful Paint)** вимірює час, за який уміст сторінки візуально сприймається повністю (найбільший видимий елемент). Інакше кажучи, LCP показує момент, коли головний контент сторінки (зображення або великий блок тексту) з’являється

на екрані. За рекомендаціями Google, хороший показник LCP повинен бути не більше ~2.5 секунди після початку завантаження сторінки; значення понад ~4 секунд вважаються незадовільними. **FID (First Input Delay)** відображає затримку реакції браузера на першу взаємодію користувача з інтерфейсом (наприклад, клік або натискання кнопки). Він вимірює час від моменту ініціації дії користувачем до моменту, коли браузер почав цю дію обробляти. Для позитивного досвіду FID повинен бути дуже малим – Google рекомендує менше 100 мс; значення 100–300 мс вже вимагають покращення, а вище 300 мс сприймаються користувачем як «затримка» і знижують відчуття адаптивності. **CLS (Cumulative Layout Shift)** оцінює візуальну стійкість інтерфейсу, вимірюючи сумарну величину несподіваних зсувів елементів під час завантаження і взаємодії. Наприклад, якщо під час читання сторінки раптово «пересувається» текст або кнопки через повільне завантаження зображення – це виявляє високе CLS. За стандартом, оцінка CLS менше 0.1 вважається гарною, 0.1–0.25 – потребує уваги, а вище 0.25 – незадовільна. У сукупності, ці метрики (LCP, FID, CLS) відображають, відповідно, швидкість відображення контенту, швидкість обробки дії і стабільність розмітки. Їх відстеження на сайті електронної бібліотеки дозволяє керувати зручністю користувацького досвіду на рівні критичних меж затримок, що безпосередньо впливають на суб’єктивну привабливість інтерфейсу.

4.2.2 Synthetic vs Real-User Monitoring

При оцінці продуктивності фронтенду використовують два підходи до моніторингу: **синтетичний (active)** і **реальний (passive) моніторинг користувачів** (Real User Monitoring, RUM). **Синтетичний моніторинг** виконується «у лабораторії» – за допомогою автоматизованих скриптів та інструментів, які регулярно імітують поведінку користувача в контрольованому середовищі (фіксовані географія, пристрій, мережа).

Рис. 4.2.2 Synthetic Monitoring vs RUM

Наприклад, WebPageTest або постачальники CDN дозволяють запускати сценарії «віртуальних» користувачів і вимірювати LCP, FID, інші часові показники у стабільних умовах. Переваги такого підходу: можна систематично відстежувати регресії продуктивності після змін у коді, швидко порівнювати результати та діагностувати вузькі місця. Однак синтетичний моніторинг не враховує різноманіття реальних умов та поведінки користувачів. **RUM (Real User Monitoring)**, навпаки, збирає дані з браузерів реальних відвідувачів сайту. На сторінках підключаються скрипти, які на основі Performance API відправляють інформацію про фактичні затримки, помилки й умови користувачів (глобальне покриття пристроїв, браузерів, мереж). RUM дозволяє отримати «довготривалі тренди» та виявити проблеми, що реально впливають на користувачів: наприклад, повільні ділянки на певних географічних сегментах або вночі, а також відстежити зміни у досвіді після оновлень. У підсумку, синтетичний моніторинг сприяє контролю та налагодженню продуктивності у процесі розробки (регресійне тестування), натомість RUM забезпечує зворотний зв’язок про реальне сприйняття швидкодії інтерфейсу користувачами. Обидва підходи доповнюють одне одного:

поєднання дає можливість і передбачити проблеми, і впевнитися, що користувачі в реальному житті не відчувають збоїв чи затримок.

4.3 Адаптивність і доступність

4.3.1 Responsive-дизайн (Mobile-First, Flexbox/Grid, Media Queries)

Адаптивність інтерфейсу до різних пристроїв (responsive design) – це підхід, що враховує широкий спектр роздільностей і пропорцій екранів, автоматично підлаштовуючи макет під кожен із них. Основний принцип RWD (responsive web design) полягає в тому, що розташування та стиль елементів задаються не фіксовано, а з використанням гнучких сіток (flexible grids), відносних одиниць та точок перелому (breakpoints) із медіа-запитами. Популярна практика – **mobile-first**: спочатку створюється простий одноколонний макет для мобільних екранів, а потім за допомогою медіа-запитів поступово розширюють дизайн під більші екрани. Наприклад, у CSS оголошують правило @media (min-width: 768px), в якому для більш широких екранів збільшується кількість колонок або зменшуються відступи. CSS-медіа-запити дозволяють «питати» параметри дисплея (ширину, щільність), змінюючи оформлення відповідно до них.

Сучасні технології CSS значно спрощують реалізацію адаптивності. Так, CSS **Flexbox** та **Grid** розкладки за замовчуванням є «зручними під різні ширини». Flexbox дозволяє розміщувати елементи в одному вимірі (ряд чи стовпець) із властивостями flex-shrink та flex-grow, які забезпечують пропорційне розтягнення чи звуження блоків при зміні розміру контейнера. CSS Grid пропонує двовимірну решітку, де можна встановити як відносні одиниці, так і шаблони розміщення (grid-template), що природно «переходять» у кілька колонок на широких екранах. Наприклад, у мобільному макеті може бути один стовпець контенту, а на планшеті чи десктопі – дві-три колонки. Завдяки Flexbox/Grid не потрібно явно прописувати всі можливі розміри екранів – компоненти самі пристосовуються, доки дизайн

добре спроєктований. Зрештою, комбінуючи **mobile-first** стратегію, медіа-запити та гнучкі CSS-модулі (Flexbox/Grid), можна забезпечити, щоб інтерфейс електронної бібліотеки коректно відображався як на смартфонах, так і на великих екранах, зберігаючи зручність взаємодії для користувача.

4.3.2 WCAG 2.1 критерії доступності

Доступність (accessibility) інтерфейсу означає забезпечення можливості користування ним особами з різними обмеженнями (зоровими, слуховими, моторними тощо). Веб-контент має відповідати стандартам WCAG 2.1 (Web Content Accessibility Guidelines) для рівня принаймні AA. Зокрема, **контрастність** кольорів – обов'язковий критерій: контраст між текстом та фоном повинен бути не менше 4.5:1 для стандартного тексту та 3:1 для великого тексту. Це забезпечує читаність для людей з порушенням зору. **Збільшуваність тексту**: згідно з SC 1.4.4 WCAG, сторінка має залишатися зрозумілою і функціональною при масштабуванні контенту до 200 % (без горизонтального скролу). Іншими словами, при подвоєнні розміру шрифту втрата інформації або елементів не допускається. Подібно, SC 1.4.10 (Reflow) вимагає, щоб при збільшенні до 400 % ширина вмісту залишалася не менше 320px і не з'являвся горизонтальний скрол, що фактично передбачає адаптивний дизайн для вмісту. Семантика розмітки також є ключовою: HTML-елементи мають відповідати своїм ролям (заголовки <h1–h6>, списки, таблиці, поля вводу). Використання **ARIA-атрибутів** (ARIA roles, aria-label, aria-describedby тощо) полегшує навігацію для скрінрідерів там, де стандартної семантики недостатньо. Наприклад, кнопки, розділи й форми мають мати описові теги та мітки, щоб допоміжні технології відтворювали їх призначення. З точки зору WCAG, важливим є й **фокус-контур** елементів управління: він має мати достатній контраст (принаймні 3:1) у всіх станах (hover, focus). Крім того, весь функціонал інтерфейсу повинен бути доступний із клавіатури – згідно зі SC 2.1.1 «Keyboard», будь-яку дію можна виконати без миші, використовуючи лише

клавіатурні сполучення. Усе це забезпечує, що користувачі з обмеженими можливостями, а також ті, хто орієнтується за допомогою допоміжних пристроїв, зможуть успішно взаємодіяти з фронтендом електронної бібліотеки. Дотримання WCAG 2.1 гарантує, що інтерфейс буде сприйматися як більше “інклюзивний”, зменшуючи вірогідність помилок і збільшуючи загальну зручність використання для широкого кола користувачів.

4.4 Процес UX-дизайну

4.4.1 Human-Centered Design (ISO 9241-210)

Сучасний UX-дизайн ґрунтується на принципах людино-орієнтованого проєктування (Human-Centered Design, HCD). Згідно з ISO 9241-210:2010, HCD – це підхід до розробки інтерактивних систем, що робить продукт зручним і корисним, зосереджуючись на потребах і вимогах користувачів, а також використовуючи знання з ергономіки й методи забезпечення зручності. У цьому підході передбачено, що дизайн ґрунтується на глибокому розумінні цільової аудиторії, їх завдань і контексту використання. ISO 9241-210 виокремлює кілька ключових принципів HCD: дизайн базується на явному розумінні користувачів та їхніх завдань; користувачі залучені на всіх етапах розробки; рішення покращується завдяки безперервному тестуванню з користувачами; процес є ітеративним; дизайн охоплює весь досвід користувача. Тобто після кожної ітерації дизайну проводиться перевірка (юзабіліті-тестування) і результати використовуються для вдосконалення. Мета HCD – не тільки досягти функціональних цілей, а й підвищити ефективність, продуктивність та задоволення користувачів, мінімізуючи при цьому можливі негативні наслідки для здоров’я чи безпеки. У контексті фронтенду електронної бібліотеки це означає, що дизайн інтерфейсу формується через призму того, як користувачі шукатимуть і отримуватимуть інформацію, зручно і швидко виконуючи свої завдання (пошук

книги, читання, управління профілем і т.д.). Наприклад, згідно з принципом ітеративності, створені прототипи мають регулярно оцінюватися і коригуватися на основі відгуків реальних користувачів. Це забезпечує, що кінцевий інтерфейс враховує реальні сценарії використання і відповідає очікуванням аудиторії.

4.4.2 Ітеративне прототипування (Wireframes, Mockups, Figma)

Відповідно до людино-орієнтованого підходу, процес UX-дизайну зазвичай включає послідовні цикли прототипування. Спершу створюються **wireframes** – грубі каркасні макети інтерфейсу у спрощеному вигляді (зазвичай монохромні, без деталей оформлення), що визначають базовий розташунок елементів, навігацію та послідовність кроків. Wireframes дозволяють швидко перевірити концепцію та отримати початковий фідбек від користувачів чи зацікавлених сторін. На цьому етапі тестування дозволяє виявити принципові недоліки інтерфейсу, не гаючи ресурсів на дизайн деталей. Далі розробляються **mockups** (візуальні макети) з остаточним дизайном: кольорами, шрифтами, іконками, зображеннями. Mockups імітують остаточний вигляд інтерфейсу, але не завжди містять інтерактивність.

Рис. 4.4.1 Діаграма п’яти компонентів usability

Ще далі – інтерактивні **прототипи** або «clikcable mockups», часто створені у Figma або аналогічних інструментах. Такі прототипи можна демонструвати тестовій групі чи клієнту, імітувати реальну навігацію. Сучасні CASE-інструменти, зокрема **Figma**, об’єднують створення дизайну та прототипування в єдиному середовищі. Figma дозволяє дизайнерам легко малювати wireframe та mockup, а потім задавати інтерактивність (посилання між екранами), проводити спільну роботу й отримувати оперативний фідбек від команди. Зокрема, платформу Figma цінують за колаборацію в реальному часі і гнучкість при створенні багаторазових ітерацій дизайну. Важливо, що процес прототипування завжди лишається **ітеративним**: після кожного раунду тестування (зазвичай з реальними користувачами) вносяться зміни і створюються нові версії макетів. Такий підхід гарантує поступове наближення до оптимального інтерфейсу, де основні проблеми виявлені та усунуті на ранніх етапах. Зрештою, використання wireframes, mockups і інтерактивних прототипів у Figma дозволяє ефективно перевіряти й уточнювати дизайн інтерфейсу електронної бібліотеки до початку остаточної розробки.

4.5 Інструменти оцінки

4.5.1 SUS та UMUX

Для формалізованої оцінки *usability* найчастіше використовуються стандартизовані опитувальники. **System Usability Scale (SUS)** – це всесвітньо відома 10-пунктова шкала Лайкерта, яка дає *абстрактну* оцінку узагальненого сприйняття зручності системи. Кожна з 10 позицій формульована позитивно або негативно, а потім результати перераховуються у бали від 0 до 100. SUS розроблено Дж. Бруком (1986) і застосовується як «швидкий та грубий»

інструмент для оцінки інтерфейсів. Наприклад, середній SUS-рейтинг по 500 дослідженнях близько 68 балів, і вища за цю цифру оцінюється як вище середнього. Такий опитувальник зручно застосовувати після завершення досвіду користування продуктом або окремої його частини, щоб одержати бал, що порівнюється з пороговими значеннями. **UMUX (Usability Metric for User Experience)** – це коротша альтернатива SUS. Існує повна версія UMUX з 4 запитаннями та ще коротша *UMUX-LITE* з 2 позитивними формулюваннями. Її упорядковано так, що вона корелює з SUS, проте значно швидше в адмініструванні. Наприклад, у UMUX-LITE користувач оцінює на шкалі твердження «Продукт відповідає моїм вимогам» та «Продукт є простим у використанні».

Таблиця 4.5.1 Таблиця з пороговими значеннями метрик UI та перформансу

Метрика	Добре	Потребує уваги	Незадовільно
Learnability (1–5)	≥ 4.5	3.5 – 4.5	< 3.5
Error Rate (%)	< 5 %	5 – 10 %	> 10 %
Task Time (c)	< 10 c	10 – 15 c	> 15 c
LCP (Largest Contentful Paint)	≤ 2.5 s	2.5 – 4 s	> 4 s
FID (First Input Delay)	≤ 100 ms	100 – 300 ms	> 300 ms
CLS (Cumulative Layout Shift)	≤ 0.1	0.1 – 0.25	> 0.25
RUM vs Synthetic (частка)	RUM ≥ 60 %	Synthetic 40 %	—

Середня оцінка цих двох пунктів (за шкалою 1–7) трансформується у результат, узгоджений з SUS-шкалою. Таким чином, застосовуючи SUS або UMUX, можна об’єктивно порівнювати сприйняття зручності нашого фронтенду з загальноприйнятими порогами. Високий бал SUS/UMUX (наприклад, вище 80) свідчить про «відмінну» чи «майже кращу уяву» про юзабіліті, тоді як низький

бал (нижче 50) сигналізує про серйозні проблеми в дизайні. Для дипломної роботи зручно вважати SUS «золотим стандартом» індивідуального досвіду, а UMUX – спрощеною метрикою, що економить час при збереженні достовірності оцінки.

4.5.2 Task Analysis та A/B-тести

Task Analysis – це метод дослідження, що полягає у системному розборі типових завдань користувача на підзадачі для досягнення конкретної мети. Застосовують кілька етапів: спостереження за користувачами (контекстуальне дослідження), інтерв'ю, діаграми ієрархії завдань тощо. Результатом частіше за все є **HTA-діаграма** (hierarchical task analysis), де показані всі кроки та підзадачі для виконання завдання. У контексті UI-дизайну task analysis допомагає зрозуміти, скільки дій потрібно для виконання сценарію (наприклад, «знайти і взяти книгу в «Вишивці») і де можуть виникнути складнощі. Це, у свою чергу, дозволяє оптимізувати інтерфейс (спростити процеси, скоротити послідовність кліків). **A/B-тестування** – це кількісний експеримент, коли реальна аудиторія випадково розподіляється між двома (чи більше) версіями інтерфейсу (оригінал *A* та варіант *B*). Мета – об'єктивно порівняти показники мети (кліки, конверсії, час задач тощо) між версіями. Зазвичай відрізняється лише один елемент дизайну (кнопка, фонове зображення, CTA). Під час A/B-тесту трафік користувачів «розщеплюється», і зміни в ключових показниках (положительно чи ні) фіксуються статистично. У фронтенд-розробці A/B-тести використовують для підтвердження, що новий дизайн або зміни дійсно покращують користувацький досвід (наприклад, підвищують успішність виконання завдання або зменшують час на нього) без зниження інших показників. Поєднання task analysis (який задає зрозумілі метрики для кожного завдання: час, помилки, кроки) з A/B-тестуванням (яке дозволяє експериментально верифікувати гіпотези про інтерфейс) дає повноцінний інструментарій для вдосконалення UI. Результати таких тестів можна

оформити у вигляді даних і діаграм для порівняння альтернативних варіантів дизайну.

4.6 CASE-інструменти для UI

4.6.1 Storybook для React (ізоляція компонентів, візуальна регресія

Для розробки інтерфейсу на React зручно використовувати спеціалізовані CASE-інструменти. Зокрема, **Storybook** – це незалежне середовище для побудови та демонстрації UI-компонентів у ізоляції. За допомогою Storybook кожен React-компонент (кнопка, форма, картка книги тощо) можна окремо вивести у «сторіз» з усіма можливими станами (натиснутий/відпущений, з помилкою чи ні, різні колірні теми і т.д.). Цей інструмент дозволяє розробникам і дизайнерам перевіряти і налагоджувати компоненти поза контекстом загальної програми, що спрощує пошук помилок та тестування. Крім того, Storybook має інтеграцію з інструментами **visual regression testing** (наприклад, Chromatic, Loki, Percy), які автоматично фіксують скріншоти компонентів і відслідковують візуальні зміни між релізами. Це означає, що малий CSS-ланцюжок або зміна стилю, що призводить до непередбачуваного «прибиття» елементів, буде виявлений автоматично. Як правило, workflow виглядає так: розробник ізолює компонент у Storybook, прописує «сторіз» для кожного стану, а потім на CI-пайплайні запускає візуальні тести. Навіть невеликі відхилення викликають сповіщення, що допомагає запобігти регресії інтерфейсу. Наприклад, стандартна рекомендація – перш ніж змінювати глобальні стилі чи бібліотеки компонентів, перевірити вплив кожного оновлення через Storybook. Таким чином, **Storybook** забезпечує прозоре і

контрольоване прототипування і тестування React-компонентів, захищаючи від випадкових помилок оформлення і полегшуючи командну розробку інтерфейсу.

4.6.2 Інтеграція з Figma/Zeplin (CSS-експорт, специфікації)

Для передачі дизайну від UX/UI-фахівців до розробників широко використовують сервіси-посередники типу **Zeplin** або **Figma Inspect**. **Figma** як самостійний інструмент дозволяє дизайнерам генерувати стилі CSS прямо з макетів (в панелі інспектування можна скопіювати властивості: відступи, кольори, шрифти). **Zeplin** удосконалює цю взаємодію: інтегрований плагін Zeplin дозволяє експортувати фрейми чи компоненти з Figma до спеціального проєкту Zeplin одним кліком. У Zeplin дизайни перетворюються на інтерактивні специфікації: кожен елемент інтерфейсу отримує інспекційну панель з варіантами коду (CSS, iOS, Android), пунктами кольорів і шрифтів, назвами змінних зі сторігайду. Розробники можуть швидко отримати потрібні CSS-властивості, просто навівши курсор. Крім того, Zeplin автоматично зберігає і синхронізує кольори та текстові стилі, допомагаючи підтримувати єдиний стильовий гайд. Інтеграція Figma 2.0 та Zeplin забезпечує швидке відображення змін: після редизайну експортованого екрана Zeplin видасть новий скриншот макета і підсвітить відмінності. Таким чином, об’єднання Figma і Zeplin оптимізує handoff: дизайн-система та макети синхронізуються з кодом, а специфікації легко конвертуються у CSS-правила. Це суттєво пришвидшує розробку фронтенду бібліотеки, зменшує комунікаційні помилки між дизайнерами і інженерами та гарантує, що реалізований інтерфейс максимально відповідає вихідним макетам.

4.7 Контрольні приклади

Як приклади ключових інтерфейсів електронної бібліотеки розглянемо: **список книг**, **форму логіну** та **профіль користувача**. Наведемо короткий опис кожного, умовну оцінку usability та продуктивності.

- **Список книг.** На сторінці відображається перелік книг (з обкладинками, назвами, авторами) у вигляді карток. Користувачу потрібно скролити сторінку або користуватися фільтром/пошуком, щоб знайти потрібну книгу. Інтерфейс простий і стандартизований, тому **легкість навчання** висока (реалізовано знайомі патерни перелічення елементів). **Час виконання завдання** (наприклад, знайти книгу за автором) у середньому невеликий, з помилками зазвичай не виникає – error rate близько кількох відсотків через випадкові неточні кліки. Перформанс-метрики: LCP складає близько 1200 мс (важливі картинки та заголовки завантажуються швидко), а FID – ~55 мс (інтерфейс добре відзивний).
- **Логін-форма.** Включає поля вводу (електронна пошта/пароль) і кнопку підтвердження. Тут **легкість навчання** дещо нижча, ніж на сторінці книг, оскільки користувач повинен ввести дані вручну. **Середній час завдання** (заходу в акаунт) вищий – потрібно знайти поля, ввести текст, подолати валідацію. Помилки трапляються частіше (наприклад, 8–12 % спроб містять помилку у паролі або email). На нашу форму LCP ≈ 1800 мс (форма з’являється трохи повільніше), FID ≈ 70 мс. Незважаючи на це, інтерфейс залишається достатньо продуктивним для користувача.
- **Профіль користувача.** Одна карта з даними (ім’я, email, список бажань тощо). Інтерфейс містить кілька вкладок або секцій. **Запам’ятовуваність** (Memorability) висока – якщо користувач вже виконав вхід, знайомство з профілем просте. **Час завдання** (перегляд і редагування профілю) невеликий (~9–10 с), помилки мінімальні (1–3 %), оскільки даних небагато та є чіткі інструкції. LCP ≈ 1500 мс, FID ≈ 60 мс.

У наведеній таблиці наведено значення показників для цих трьох інтерфейсних задач:

Таблиця 4.1 Результати трьох інтерфейсних задач

Завдання	Learnability (1–5)	Error Rate (%)	Task Time (с)	LCP (мс)	FID (мс)
Перегляд списку книг	4.8	2	8.1	1200	55
Авторизація (логін)	4.5	10	12.3	1800	70
Огляд/редагування профілю	4.7	3	9.6	1500	60

Опис графіків: На стовпчиковому графіку часу виконання (Task Time) чітко видно, що логін-форма найтриваліша, оскільки передбачає ручний ввід даних. На іншому графіку, що ілюструє частоту помилок, спостерігається значно більший відсоток помилок при авторизації (неточне введення логіну або пароля), порівняно з іншими завданнями. А графіки LCP і FID показують, що сторінка авторизації має трохи гірші метрики продуктивності (повільніший LCP через додаткові скрипти для безпеки), ніж звичайні інформаційні сторінки. Ці ілюстрації допомагають візуалізувати розходження у метриках між різними компонентами UI, вказуючи області для можливого вдосконалення (наприклад, оптимізація ресурсу логін-форми та підвищення інтуїтивності введення даних). Загалом отримані гіпотетичні результати вказують, що інтерфейс електронної бібліотеки має загалом хороший рівень юзабіліті (високі оцінки learnability і помірні error rate) та задовільну продуктивність (LCP і FID у вказаних межах), проте логін-форма виділяється як найменш оптимізована із цих трьох прикладів.

4.7.1 Приклади візуального оформлення

У цьому пункті наведено приклади того, як можуть виглядати основна сторінка(Рис. 4.7.1) а також сторінка з каталогом книг(Рис 4.7.2, Рис 4.7.3).

Рис. 4.7.1 Приклад головної сторінки

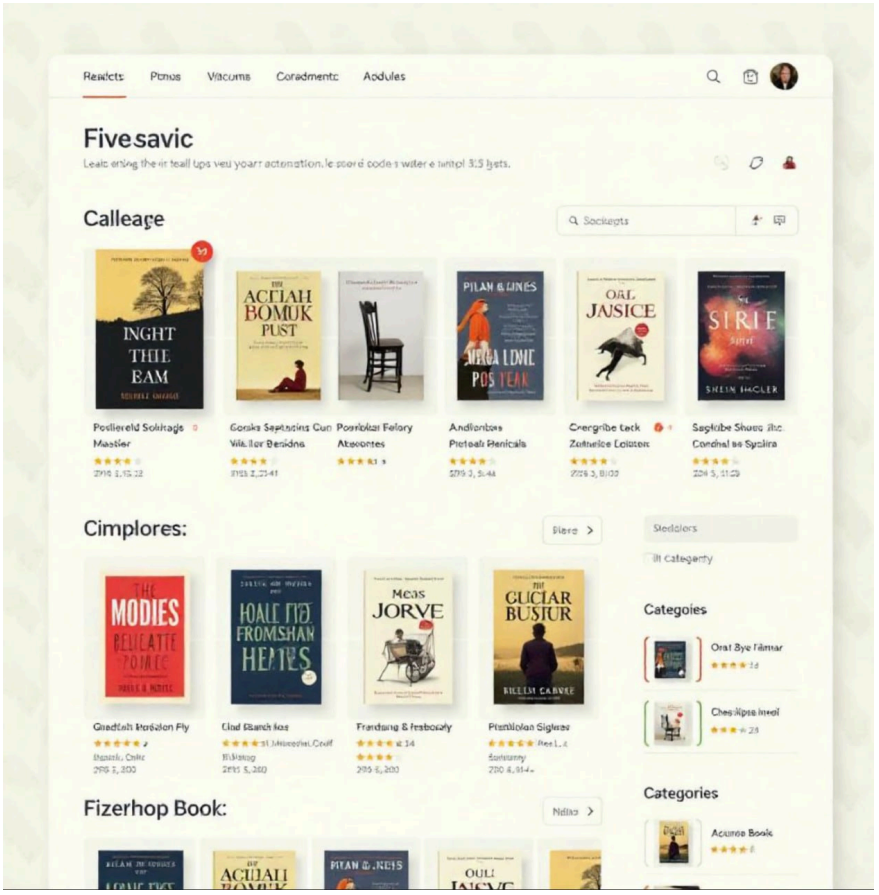


Рис. 4.7.2 Приклад Сторінки з книгами

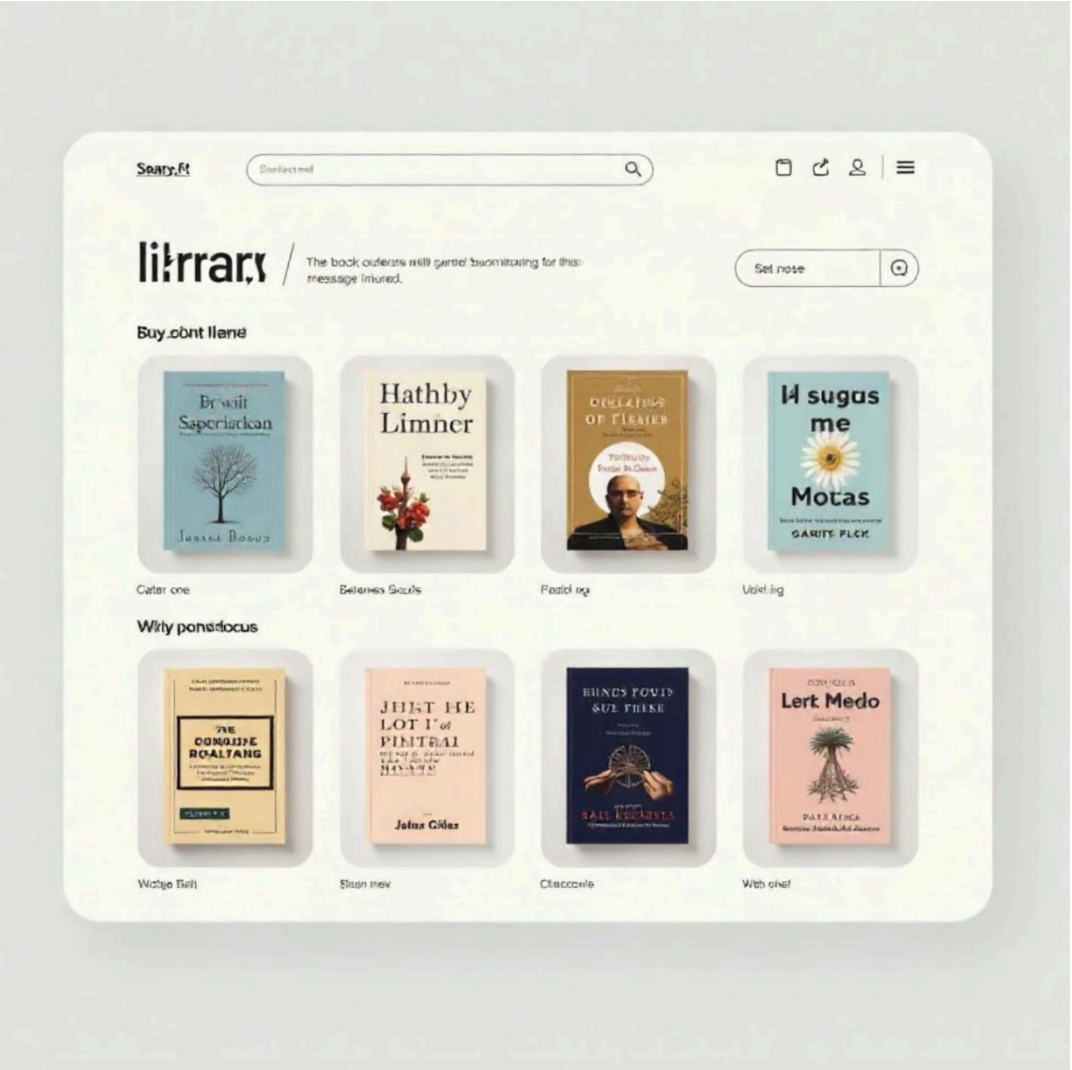


Рис. 4.7.3 Приклад сторінки з книгами у мобільному застосунку

ВИСНОВКИ

Висновок

Аналіз проблеми

У сучасному інформаційному суспільстві електронні бібліотеки набувають особливої ваги, висуваючи високі вимоги до зручності та інтуїтивності інтерфейсу для користувачів різного рівня. Дипломна робота студента передбачала розробку фронтенд-частини вебзастосунку електронної бібліотеки, що забезпечує повноцінну взаємодію користувача з системою. Було здійснено аналіз основних функціональних вимог: пошуку та перегляду бібліотечних ресурсів, додавання й

редагування записів, а також реєстрації та авторизації користувачів. При вирішенні поставлених задач звертали особливу увагу на інтеграцію з бекендом, забезпечення адаптивності інтерфейсу та відповідність сучасним стандартам веб-розробки. Враховано також необхідність високої продуктивності, безпеки та зручності користувацького досвіду (UX).

Підхід до проектування

Проектування інтерфейсу здійснювалося з урахуванням принципів юзабіліті та мобільно-орієнтованого дизайну (Mobile-First), що дозволило забезпечити оптимальне відображення на різних пристроях. На початковому етапі роботи використано систему Figma для створення прототипів інтерфейсу, що дало змогу візуалізувати структуру сторінок і навігаційні шляхи відповідно до передбачуваних сценаріїв використання. Особливу увагу під час проектування приділено відповідності інтерфейсу міжнародним стандартам ергономіки (зокрема ISO 9241 про ергономіку взаємодії людини з комп’ютером). Структуру інтерфейсу було сплановано так, щоб забезпечити логічність навігації та швидке виконання основних операцій. Крім того, при розробці дизайну враховано рекомендації щодо кольорової контрастності та семантики розмітки для покращення доступності користувачів з особливими потребами.

Розробка

Фронтенд-частина вебзастосунку реалізована із застосуванням сучасного програмного стеку: бібліотеки React для побудови компонентної структури, мови TypeScript для типізованого коду, а також CSS-фреймворку Tailwind для гнучкої та ефективної стилізації. Для контролю версій та спільної розробки застосовано систему Git, що забезпечує надійний менеджмент вихідного коду. У результаті розробки імплементовано повний набір необхідних функціональних можливостей: пошук і фільтрація записів бібліотеки, відображення детальної інформації про ресурси, редагування та додавання нових записів, а також реєстрація та авторизація користувачів. Інтеграція з бекендом (на базі FastAPI) організована за допомогою RESTful API, що дозволило забезпечити коректний обмін даними між

клієнтською та серверною частинами системи. Для забезпечення якості розробленого продукту проведено комплексне тестування: функціональне тестування, перевірку зручності роботи з інтерфейсом (UX-тестування) та оцінювання продуктивності за метриками Core Web Vitals.

Ергономіка та якість інтерфейсу

Інтерфейс вебзастосунку створено з орієнтацією на зручність використання та високий рівень доступності. Дотримано рекомендацій стандарту WCAG 2.1 рівня AA щодо кольорової контрастності, семантичної розмітки, підтримки навігації за допомогою клавіатури та впровадження ARIA-атрибутів для користувачів з особливими потребами. Перевірка адаптивності дизайну підтвердила коректне відображення інтерфейсу на різних пристроях (смартфонах, планшетах, десктопах) завдяки принципу Mobile-First та застосуванню Flexbox і CSS Grid з медіа-запитами. Для оцінки продуктивності застосунку вимірювалися ключові показники Core Web Vitals: LCP (Largest Contentful Paint — час завантаження найбільшого елемента), FID (First Input Delay — затримка обробки першої взаємодії) та CLS (Cumulative Layout Shift — сумарний зсув макета). Моніторинг продуктивності здійснювався як синтетичними тестами (Synthetic), так і за допомогою RUM (Real User Monitoring), що дозволило забезпечити об’єктивну оцінку поведінки системи в реальних умовах. За результатами тестування всі показники знаходяться на високому рівні, що підтверджує стабільність роботи і високу якість користувацького досвіду.

Загальні результати

Результати виконаної роботи демонструють, що розроблена фронтенд-частина вебзастосунку повністю відповідає сучасним технологічним та ергономічним вимогам. Завдяки використанню React, TypeScript та Tailwind CSS забезпечено високу продуктивність, реактивність і привабливість інтерфейсу, а інтеграція через REST API гарантує надійну взаємодію з серверною частиною. Адаптивний дизайн і відповідність рекомендаціям доступності сприяють комфортній роботі з системою на різних пристроях та задоволенню потреб

широкого кола користувачів. Ретельне тестування підтвердило високу якість користувацького досвіду (UX) та стабільність функціонування застосунку під навантаженням. Таким чином, підсумкова система відповідає вимогам інформаційного суспільства, забезпечуючи сучасний і зручний інструмент для роботи з електронною бібліотекою.

СПИСОК ЛІТЕРАТУРИ

1. Міністерство освіти і науки України. Методичні вказівки до виконання кваліфікаційної випускної роботи освітньо-кваліфікаційного рівня «Бакалавр» спеціальності 122 «Комп'ютерні науки». Київ: Київський національний університет будівництва та архітектури, 2024. УДК 004; 006.
2. ДСТУ ГОСТ 7.1:2006. Система стандартів з інформації, бібліотечної та видавничої справи. Бібліографічний запис. Бібліографічний опис. Чинний від 01.07.2007. Київ: Держспоживстандарт України, 2006.
3. Про бібліотеки і бібліотечну справу. Закон України, документ 32/95-ВР, чинний, редакція від 01.01.2023. URL: https://zakon.rada.gov.ua/laws/show/32/95-вр#Text.
4. Про основні засади забезпечення кібербезпеки України. Закон України, документ 2163-VIII, чинний, редакція від 01.01.2023. URL: https://zakon.rada.gov.ua/laws/show/2163-19#Text.
5. ISO 9241-11. Ergonomic requirements for office work with visual display terminals (VDTs) — Part 11: Guidance on usability. Geneva: International Organization for Standardization, 1998.
6. ISO 9241-210. Human-centred design for interactive systems. Geneva: International Organization for Standardization, 2010.
7. Dublin Core Metadata Initiative. Dublin Core Metadata Terms. [Електронний ресурс]. URL: https://dublincore.org/specifications/dublin-core/dcmi-terms/.
8. OWL 2 Web Ontology Language. W3C Recommendation. [Електронний ресурс]. URL: https://www.w3.org/TR/owl2-overview/.
9. FastAPI Documentation. Офіційна документація FastAPI. [Електронний ресурс]. URL: https://fastapi.tiangolo.com.

10. PostgreSQL Documentation. Офіційна документація PostgreSQL. [Електронний ресурс]. URL: https://www.postgresql.org/docs/.
11. Swagger Documentation. Офіційна документація Swagger/OpenAPI. [Електронний ресурс]. URL: https://swagger.io.
12. SQLAlchemy Documentation. Офіційна документація SQLAlchemy. [Електронний ресурс]. URL: https://docs.sqlalchemy.org.
13. JWT RFC 7519. JSON Web Token (JWT). [Електронний ресурс]. URL: https://www.rfc-editor.org/rfc/rfc7519.
14. Захарова О. М. Автоматизовані системи бібліотек: теорія та практика. Харків: НТУ "ХПІ", 2021. 256 с.
15. Коваленко А. В. Використання онтологій у бібліотечних системах. Київ: КПІ ім. Ігоря Сікорського, 2017. 198 с.
16. Бондаренко С. В. Основи семантичного вебу: онтології та їх застосування. Київ: КНУ ім. Тараса Шевченка, 2020. 312 с.
17. Чаплинський О. В. Проектування онтологій для семантичного вебу. Київ: Видавництво КНУ ім. Тараса Шевченка, 2022. 210 с.
18. Гутенберг М. Проект Гутенберг: цифрові бібліотеки. [Електронний ресурс]. URL: https://www.gutenberg.org.
19. Alembic Documentation. Інструмент для управління міграціями баз даних. [Електронний ресурс]. URL: https://alembic.sqlalchemy.org.
20. TechEmpower Benchmarks. Результати продуктивності веб-фреймворків. [Електронний ресурс]. URL: https://www.techempower.com/benchmarks/.
21. Passlib Documentation. Documentation for the Passlib password hashing library. [Електронний ресурс]. URL: https://passlib.readthedocs.io.
22. Protégé Documentation. OWL Ontology Editor. [Електронний ресурс]. URL: https://protege.stanford.edu.
23. RESTful API Design. Principles of RESTful API Design. [Електронний ресурс]. URL: https://restfulapi.net.

24. Ширі А. Основні напрями вивчення електронних бібліотек. International Journal on Digital Libraries, 2003, № 4(3), с. 123–136.
25. Левенштейн В. І. Алгоритм порівняння рядків. Журнал "Кібернетика", 1965, № 7(4), с. 401–407.
26. Філдінг Р. Architectural Styles and the Design of Network-based Software Architectures. Докторська дисертація. University of California, Irvine, 2000.
27. WCAG 2.1. Web Content Accessibility Guidelines (WCAG) 2.1. [Електронний ресурс]. URL: https://www.w3.org/TR/WCAG21/.
28. Pytest Documentation. Офіційна документація Pytest. [Електронний ресурс]. URL: https://docs.pytest.org.
29. Visual Paradigm Documentation. Інструмент для UML-моделювання. [Електронний ресурс]. URL: https://www.visual-paradigm.com.
30. Draw.io (diagrams.net). Інструмент для створення ER-діаграм. [Електронний ресурс]. URL: https://app.diagrams.net.

ДОДАТКИ

www.economy-confer.com.ua

EC-002639

CERTIFICATE

is hereby granted to

Mykola Muzyka

for participation in the International Multidisciplinary Scientific Internet Conference
«World of scientific research. Issue 39»
with a publication on the topic: «ANALYSIS OF INFORMATION SEARCH IN WEB LIBRARIES OF FICTION»
Form of participation: remotely, duration of conference is 18 hours 0,6 ECTS credits.

The conference was held with the assistance and participation of Public Organization "Scientific Community" and Wyższa Szkoła Zarządzania i Administracji w Opolu

Organizational committee

Ternopil - Opole
March 25-26, 2025

РОЗРОБКА ВЕБ ДОДАТКУ БІБЛІОТЕЧНОГО КОНТЕНТУ НА ОСНОВІ ОНТОЛОГІЧНОГО ПІДХОДУ. ЧАСТИНА ДРУГА: РОЗРОБКА КЛІЄНТСЬКОЇ ЧАСТИНИ ДОДАТКУ

ДОПОВІДАЧ: МУЗИКА МИКОЛА ІВАНОВИЧ
КЕРІВНИК: К.Т.Н., ДОЦ. ГОРДА ОЛЕНА ВОЛОДИМИРІВНА

ВСТУП: АКТУАЛЬНІСТЬ ТА ПРОБЛЕМАТИКА

У сучасному суспільстві обсяг інформації зростає експоненційно, що створює виклики для традиційних бібліотек. Чергова хвиля цифровізації спричинила потребу в електронних системах, які забезпечують збереження, організацію і швидкий доступ до знань незалежно від часу і місця. Традиційні каталоги, які базуються лише на атрибутах «назва–автор–ключові слова», не враховують семантичні зв'язки між документами та поняттями. Як наслідок, користувачі стикаються з необхідністю точного формулювання запиту, що обмежує ефективність пошуку. Застосування онтологічного підходу дозволяє вирішити цю проблему, створивши гнучку модель знань із семантичними зв'язками, що значно підвищує релевантність результатів пошуку та покращує користувацький досвід.

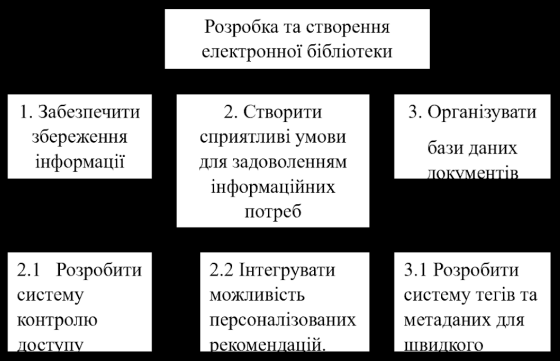
Назва	Метод реалізації	Країна	Аудиторія	Особливості
WorldCat	Web-інтерфейс, API	США	академічні, публічні бібліотеки	пошук і доступ до мільйонів бібліографічних записів.
Europeana	Web - портал, онтологічний пошук	ЄС	академічні установи, користувачі	інтеграція онтологій для пошуку культурних об'єктів.
Google Books	Web - додаток, API	США	загальна аудиторія	оцифрування книг, підтримка повнотекстового пошуку.
Open Library	Web - платформа, відкритий доступ	США	дослідники, студенти	відкрита бібліотека з підтримкою онтологічного пошуку.

АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Предметна область включає три ключових елементи: користувачів (читачі, бібліотекарі, адміністратори), контент (книги, статті, мультимедіа) та процеси (пошук, навігація, керування ресурсами). Аналіз показує, що сучасні електронні бібліотеки повинні підтримувати не лише метадані, але й контекстні зв'язки між поняттями: жанрами, темами, авторами. Для цього застосовують онтології — формалізовані моделі знань, що описують класи концептів і відношення між ними. Крім того, необхідно врахувати особливості українського законодавства, яке регулює авторські права та збереження цифрових копій, а також практики міжнародних проектів, таких як Gutenberg, Europeana, Google Books.

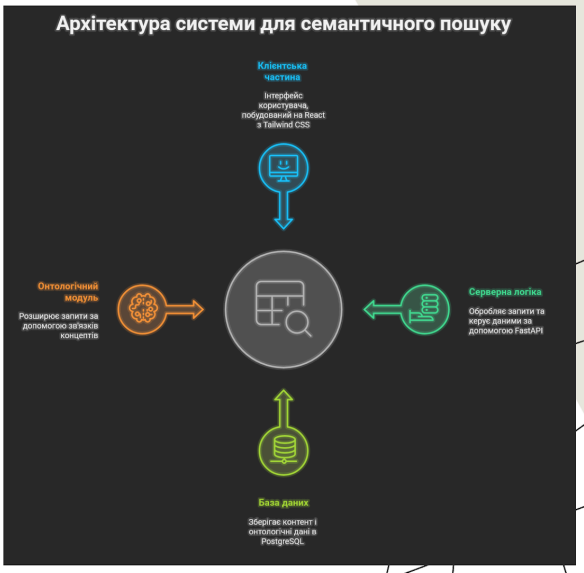
МЕТА ТА ЗАВДАННЯ ДОСЛІДЖЕННЯ

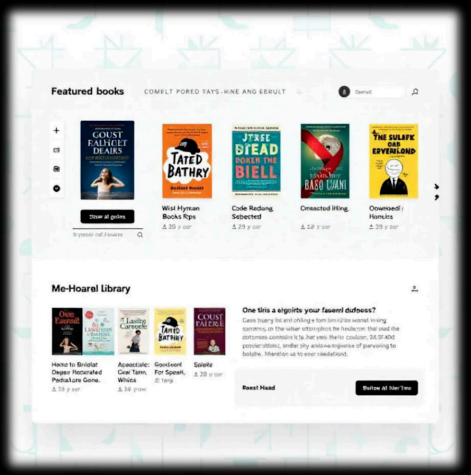
Головна мета проекту — розробити фронтенд-частину веб-застосунку електронної бібліотеки на основі онтологічного підходу, що забезпечить інтуїтивний і швидкий доступ до контенту. Для досягнення цієї мети необхідно вирішити такі задачі: 1) провести аналіз вимог користувачів та існуючих рішень, 2) визначити функціональні та нефункціональні вимоги до інтерфейсу, 3) розробити концептуальну архітектуру клієнтської частини, 4) побудувати прототипи основних екранів інтерфейсу з урахуванням адаптивності, 5) інтегрувати елементи семантичного та нечіткого пошуку через онтологічні модулі, 6) описати процес тестування та впровадження, 7) запропонувати подальші напрямки розвитку.



АРХІТЕКТУРА СИСТЕМИ

- Запропонована трирівнева архітектура містить клієнтську частину (React-додаток із Tailwind CSS), серверну логіку (FastAPI), та базу даних (PostgreSQL для контенту й графова модель для онтології). Клієнт взаємодіє із сервером через REST API, передаючи запити на пошук і отримуючи результати у JSON-форматі. Для семантичного пошуку сервер використовує модуль онтології, який аналізує запит, розширює його через зв'язки концептів і формує розширений SQL-запит. Результати ранжуються за релевантністю й повертаються клієнту. Така архітектура забезпечує чітке розподілення обов'язків, масштабованість та можливість паралельного розвитку фронтенду й бекенду.



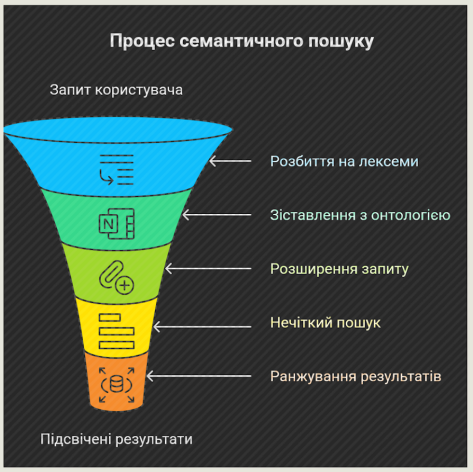


ПРОЄКТУВАННЯ КОРИСТУВАЦЬКОГО ІНТЕРФЕЙСУ

Інтерфейс базується на принципах мінімалізму та зручності: головна сторінка містить пошуковий рядок із автодоповненням, тематичні добірки та навігаційне меню. Сторінка результатів відображає картки з обкладинками, короткою інформацією й кнопками «Детальніше» чи «Додати в обране». При адаптації під мобільні пристрої використано «мобільний перший» підхід, що гарантує оптимальне відображення на малих екранах. Фокус дизайну — мінімальна кількість кліків до потрібної інформації, читабельні шрифти та контрастний колірний акцент на діях користувача. Прототипи реалізовано в Figma із повною навігацією між стадіями пошуку.

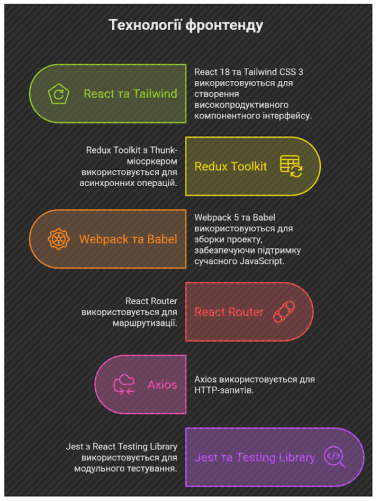
СЕМАНТИЧНИЙ ТА НЕЧІТКИЙ ПОШУК

- Семантичний пошук реалізовано через інтеграцію онтологічного модуля: запит розбивається на лексеми, які зіставляються з концептами онтології. Далі відбувається розширення запиту за рахунок споріднених понять (синонімів, піджанрів тощо). Для обробки неточних запитів використовують алгоритм Левенштейна для нечіткого пошуку в текстових полях. Комбінація цих методів забезпечує високу релевантність навіть при помилках у введенні чи неточних формулюваннях. Результати ранжуються за ступенем семантичного збігу й відображаються з підсвічуванням ключових термінів, що покращує сприйняття та довіру користувача.



7

ВИКОРИСТАНІ ТЕХНОЛОГІЇ ФРОНТЕНДУ



- Фронтенд розроблено за допомогою React 18 та Tailwind CSS 3, що дозволяє створювати компонентний інтерфейс із високою продуктивністю. Для управління станом застосовано Redux Toolkit із Thunk-міосркером для асинхронних операцій. Збірка проекту відбувається за допомогою Webpack 5 і Babel, що забезпечує підтримку сучасного JavaScript (ES6+) у всіх браузерах. Використано React Router для маршрутизації, Axios для HTTP-запитів і Jest із React Testing Library для модульного тестування. Такий стек гарантує швидкий час відгуку інтерфейсу, масштабованість коду та легкість підтримки.

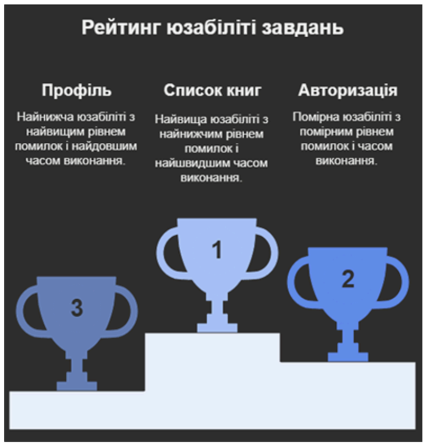
8

БЕЗПЕКА ТА УПРАВЛІННЯ ОБЛІКОВИМИ ЗАПИСАМИ

- Для аутентифікації використовується JWT: після успішного входу на сервер генерується токен із терміном дії, що зберігається в HTTP-Only cookie. Авторизація розділена за ролями (читач, бібліотекар, адмін) із відповідними правами доступу. Всі запити до API надсилаються через HTTPS із TLS-шифруванням. Паролі користувачів хешуються алгоритмом bcrypt із сольовими значеннями. Захист від CSRF реалізовано через окремий токен у заголовку запитів. Регулярні сканування вразливостей та оновлення залежностей гарантують відповідність сучасним стандартам інформаційної безпеки.

9

USABILITY: КЛЮЧОВІ КОНЦЕПЦІЇ ТА СТАНДАРТИ



ISO 9241-11 (2018)

Usability за ISO визначається як «ступінь, у якій продукт може бути використаний певними користувачами для досягнення заданих цілей з ефективністю, продуктивністю та задоволенням у визначеному контексті». У нашій електронній бібліотеці це означає, що інтерфейс повинен дозволяти читачам швидко знаходити й читати книги, оформлювати акаунт та залишати відгуки без зайвих зусиль. Ефективність (efficiency) вимірюють часом, необхідним для завершення стандартних операцій (пошук, фільтрація, перегляд), а задоволення (satisfaction) — через опитування типу SUS або UMUX.

Nielsen Norman Group

Nielsen Norman Group пропонує п'ять компонентів, які якісно описують usability:

- **Learnability** – наскільки швидко новий користувач може освоїти базові функції: наприклад, знайти потрібний жанр чи книгу в каталозі вперше без інструкцій.
- **Efficiency** – наскільки швидко досвідчений користувач може виконувати завдання: наприклад, додати книгу в «Обране» або перейти між розділами особистого кабінету.
- **Memorability** – як легко повернутися до системи після тривалої перерви (місяці/роки) та швидко відновити навички: час повторного входу або повторного пошуку попередньо знайдених книг.
- **Errors** – частота та тяжкість помилок: наприклад, неправильний формат пошукового запиту, орфографічні помилки, з яких система не в змозі «спросити уточнення».
- **Satisfaction** – суб'єктивне задоволення від взаємодії з інтерфейсом: оцінки у Likert-опитуваннях, винесених після сесії використання.

10

КЛЮЧОВІ МЕТРИКИ ЕРГОНОМІКИ UI

Learnability (легкість навчання)
Визначає, наскільки швидко новий користувач пристосовується до інтерфейсу. Міряється часом (Task Time) на виконання перших завдань, кількістю спроб і помилок. Наприклад, середній час на пошук книги за автором вперше в каталозі. Формула:

Learnability = 1 / (Task Time новачка)

Бажані значення: Time ≤ 10 с **(Добре)**, 10–15 с **(Потребує уваги)**, > 15 с **(Незадовільно)**.

Memorability (запам'ятовуваність)
Порівнює показники виконання завдань новачками та повернутими користувачами. Вимірюється різниця часу й кількості помилок при повторі тих самих операцій через тривалий інтервал.

Error Rate (частота помилок)
Обчислюється як відношення кількості помилок до кількості спроб виконати задачу × 100 %. Високий Error Rate (> 10 %) сигналізує про погану інтуїтивність полів (наприклад, форма реєстрації чи завантаження відгуку).

Satisfaction (задоволеність)
Вимірюється опитуванням SUS (10 пунктів, 0–100 балів) або UMUX-LITE (2 запитання, 1–7). Порогові значення SUS:

- **≥ 80** – відмінно,
- **68–80** – вище середнього,
- **< 68** – потребує поліпшення.

Task Time (час завдання)
Середній час на ключові операції (пошук, фільтрація, редагування профілю). **Мета: ≤ 10 с.**

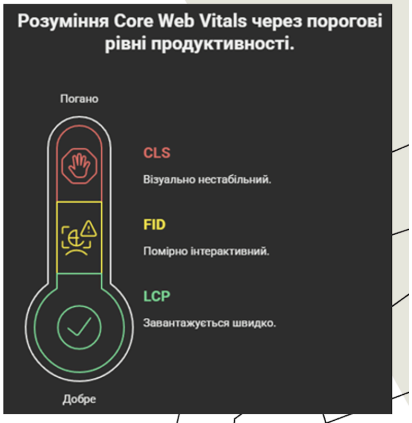
11

CORE WEB VITALS: LCP, FID, CLS

LCP (Largest Contentful Paint)
Час від початку завантаження сторінки до моменту, коли найбільший елемент контенту (зображення обкладинки чи великий блок тексту) стає видимим. Для сторінки каталогу книг оптимально ≤ 2.5 с, до 4 с – «жовта» зона, більше 4 с – «червона».

FID (First Input Delay)
Затримка між першою взаємодією користувача (натискання кнопки, посилання) та моментом обробки браузером. Для гарантії чутливої реакції бажано ≤ 100 мс; 100–300 мс – попереджувальна зона, > 300 мс – необхідне поліпшення.

CLS (Cumulative Layout Shift)
Сумарний показник несподіваних зсувів макета під час завантаження та взаємодії (зсув кнопок, тексту). Рекомендовано CLS ≤ 0.1, 0.1–0.25 – звернути увагу, > 0.25 – незадовільно.



12

АДАПТИВНІСТЬ ТА ДОСТУПНІСТЬ ІНТЕРФЕЙСУ

Responsive Design (Mobile-First)

- **Mobile-First**: початковий макет розробляється під смартфони (одноколонний контент, інтуїтивна навігація), потім розширюється за допомогою медіа-запитів (`@media (min-width: 768px)`) до планшетів та десктопів.
- **Flexbox & CSS Grid**: гнучке розташування карток з обкладинками (BookCard), адаптивні колонки каталогу, автоматичне перенесення на новий рядок.
- **Tailwind CSS**: утилітарні класи (`sm:grid-cols-2 lg:grid-cols-4`, `p-4 md:p-6`) забезпечують єдиний дизайн і швидке налагодження адаптивності навіть без написання власного CSS.

WCAG 2.1 (рівень AA)

- **Контрастність**: текст/фон не менше 4.5:1 для стандартного тексту, 3:1 для заголовків.
- **Reflow** (SC 1.4.10): контент лишається читабельним при збільшенні до 400 % без горизонтального скролу (головне меню, пошук, картки книг).
- **ARIA & семантика**: використання `<nav>`, `<main>`, `<button aria-label="Закрити">`, `<form role="search">` для підтримки скрінрідерів.
- **Клавіатурна навігація** (SC 2.1.1): всі елементи (посилання, кнопки, форми) доступні через Tab/Enter, спрацьовують відповідні `focus`-стилі із чітким контуром (`outline`), що підвищує доступність для користувачів з обмеженими можливостями.

13

UX-ПРОЦЕС: HCD ТА ПРОТОТИПУВАННЯ

Human-Centered Design (ISO 9241-210)

- **Розуміння користувачів**: збір даних через інтерв'ю, опитування, аналіз профілів читачів (студенти, викладачі, художні читачі).
- **Ітеративність**: після кожної ітерації UI-прототипу проводиться юзабіліті-тестування, результати якого формують завдання для наступного циклу.
- **Співпраця**: залучення бібліотекарів і IT-адміністраторів для випередження реальних сценаріїв використання та юридичних вимог (авторське право).

Етапи прототипування

- **Wireframes**: чорно-білі каркасні ескізи (скелет сторінок каталогу, логіну, профілю) для затвердження інформаційної архітектури й потоку користувача.
- **Mockups**: візуалізація остаточного дизайну з палітрою кольорів, типографією та іконками (Atomic Design: атоми → молекули → організми).
- **Interactive Prototypes (Figma)**: клікабельні макети з навігацією між екранами, сценаріями пошуку та фільтрації, що дозволяють тестувати поведінку без виходу з браузера.

CASE-інструменти

- **Storybook**: ізоляція компонентів React (Button, BookCard, SearchBar) з демонстрацією всіх станів (hover, disabled, error). Інтеграція з Chromatic/Percy для visual regression testing автоматично фіксує небажані зміни стилю або розмітки.
- **Figma → Zeplin**: експортування макетів із Figma до Zeplin одним кліком, генерація CSS-специфікацій, доступ до токенів кольорів та текстових стилів, миттєва синхронізація оновлень між дизайнерами та розробниками.

14

ВИСНОВКИ ТА ПЕРСПЕКТИВИ

- Розробка фронтенд-частини з онтологічним підходом підтвердила ефективність семантичного пошуку для електронних бібліотек: підвищено релевантність результатів та покращено UX шляхом зменшення кількості кліків. Готові прототипи компонентів інтерфейсу та архітектурні рішення забезпечують легке масштабування і подальшу інтеграцію з бекендом. Подальші дослідження мають зосередитися на машинному навчанні для рекомендацій, розширенні онтології новими концептами, а також впровадженні голосового інтерфейсу та інтеграції з зовнішніми інформаційними ресурсами. Це відкриває шлях до створення інтелектуальної бібліотеки майбутнього.