ТОВ «ТЕХНІЧНИЙ УНІВЕРСИТЕТ «МЕТІНВЕСТ ПОЛІТЕХНІКА»
Факультет автоматизації виробництва та цифрових технологій
Кафедра цифрових технологій та проєктно-аналітичних рішень

«Допущено до захисту»
Гарант ОПП
Олександр КОСТІКОВ

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавра

за підсумками виконання
освітньо-професійної програми
«Комп'ютерні науки»
за спеціальністю 122 Комп`ютерні науки

на тему «Автоматизоване робоче місце бібліотекара технічної бібліотеки підприємства»

Керівник роботи Олександр КОСТІКОВ

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

Здобувач Ілля ВОЛКОВ

Підсумкова оцінка за атестацію			

Голова ЕК Олена ПАВЛЕНКО

ЗАПОРІЖЖЯ 2025

ТОВ «ТЕХНІЧНИЙ УНІВЕРСИТЕТ «МЕТІНВЕСТ ПОЛІТЕХНІКА»

Факультет	автоматизації виробництва та цифрових технологій
Кафедра	цифрових технологій та проектно-аналітичних рішень
Ступінь вищої освіти	бакалавр
Спеціальність	122 Комп'ютерні науки
ОПП	Комп'ютерні науки

ЗАТВЕРДЖУЮ
Гарант ОПП

Олександр КОСТІКОВ

«05» червня 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА

Волкову Іллі Олеговичу
(прізвище, ім'я, по батькові здобувача)

1. Тема роботи *«Автоматизоване робоче місце бібліотекара технічної бібліотеки підприємства»*
керівник роботи *Костіков Олександр Анатолійович, доцент, канд. фіз.-мат. наук,*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом Університету від *31.03.2025 р. №81/31.03.2025*
2. Термін подання роботи *01.07.2025 р.*
3. Вихідні дані до роботи *Навчальна література, державні стандарти та методичні вказівки з дисциплін «Системний аналіз», «Інженерія програмного забезпечення», «Бази даних», «Управління IT-проєктами», документація з розробки ПЗ на React і Express, наукові публікації з тематики автоматизації бібліотечних процесів, дослідження в галузі побудови інформаційних систем, результати власного проєктування, тестування, експериментальних розрахунків та економічного обґрунтування впровадження автоматизованого робочого місця бібліотекара.*
4. Зміст пояснювальної записки (перелік питань) *Реферат. Зміст. Вступ. Розділ 1. Аналіз стану автоматизації бізнес-процесів в роботі бібліотеки. Розділ 2. Моделювання процесів автоматизації в роботі бібліотеки підприємства. Розділ 3. Проєктування та реалізація програми автоматизації робочого місця бібліотекара технічної бібліотеки підприємства. Розділ 4. Економічна доцільність розробки програми автоматизації робочого місця бібліотекара технічної бібліотеки підприємства Висновки. Перелік використаних джерел. Додатки.*
5. Перелік графічного (демонстраційного) матеріалу (з точним зазначенням обов'язкових креслень): *Актуальність, мета, об'єкт, предмет та завдання дослідження; логічна модель системи управління бібліотекою; SADT-діаграми; ER-діаграма структури бази даних; діаграми UML (прецедентів, класів, послідовностей); інтерфейси форм користувача; приклади реалізованих звітів і зв'язків таблиць у СУБД; результати розрахунків витрат та терміну окупності системи; висновки до роботи; (за наявності) – копії публікацій або розробленої документації.*

6. Консультанти по роботі, із зазначенням розділів роботи, що стосуються їх.

Розділ	Прізвище, ініціали та посада консультанта
1	Костіков О.А.., доц. каф. ЦТПАР
2	Костіков О.А.., доц. каф. ЦТПАР
3	Костіков О.А.., доц. каф. ЦТПАР
4	Костіков О.А.., доц. каф. ЦТПАР

7. Дата видачі завдання *05.07.2025*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи
1	Розділ 1. Аналіз стану автоматизації бізнес-процесів в роботі бібліотеки	05.06.2025 - 10.06.2025
2	Розділ 2. Моделювання процесів автоматизації в роботі бібліотеки підприємства.	11.06.2025 - 15.06.2025
3	Розділ 3. Проєктування та реалізація програми автоматизації робочого місця бібліотекара технічної бібліотеки підприємства.	16.06.2025 – 22.06.2025
5	Розділ 4. Економічна доцільність розробки програми автоматизації робочого місця бібліотекара технічної бібліотеки підприємства	23.06.2025 - 25.06.2025
6	Висновки, перелік посилань, вступ, зміст, реферат	26.06.2025 – 27.06.2025
7	Остаточне оформлення та подання завершеної роботи.	28.06.2025 – 30.06.2025
8	Підготовка презентаційного матеріалу, рецензування завершеної роботи. Захист	01.07.2025 – 04.07.2025

Здобувач (Ілля ВОЛКОВ)

Керівник роботи (Олександр КОСТІКОВ)

РЕФЕРАТ

Волков І.О. Автоматизоване робоче місце бібліотекара технічної бібліотеки підприємства.

Кваліфікаційна робота на здобуття освітнього ступеня бакалавра комп'ютерних наук за спеціальністю 122 «Комп'ютерні науки». – ТОВ «ТЕХНІЧНИЙ УНІВЕРСИТЕТ «МЕТІНВЕСТ ПОЛІТЕХНІКА» МОН України, м. Запоріжжя, 2025.

Метою роботи є створення програми автоматизації робочого місця бібліотекара технічної бібліотеки підприємства з можливістю ведення каталогу книг, ведення обліку користувачів із збереженням їх особистих даних та історії позичань.

Об'єкт дослідження - бібліотечні процеси, з яких складається робота бібліотек.

Предмет дослідження – процес автоматизації бібліотечних процесів за допомогою спеціалізованого програмного забезпечення.

Методологія – структурний аналіз, методика SADT, моделювання інформаційних потоків, об'єктно-орієнтоване програмування, проєктування баз даних, економічна оцінка розробки.

У роботі проведено аналіз потреб бібліотекара, побудовано логічну модель системи, створено програму автоматизації робочого місця бібліотекара підприємства. Реалізовано інтерфейс користувача, систему авторизації, ведення каталогу книг, облік користувачів, бронювання та позичання книг. Визначено техніко-економічні показники та розраховано термін окупності системи.

Розроблений програмний продукт дозволяє зменшити трудовитрати персоналу, ефективно здійснювати обслуговування користувачів та вести облік бібліотечного фонду . Система придатна до масштабування та подальшого розширення функціоналу. Результати можуть бути використані іншими підприємствами.

КЛЮЧОВІ СЛОВА: БІБЛІОТЕЧНІ ПРОЦЕСИ, АВТОМАТИЗАЦІЯ, АВТОМАТИЗОВАНЕ РОБОЧЕ МІСЦЕ, БАЗА ДАНИХ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, MYSQL, VISUAL STUDIO, ІНФОРМАЦІЙНІ ПОТОКИ, ЕКОНОМІЧНА ЕФЕКТИВНІСТЬ

ABSTRACT

Volkov I.O. Automated librarian workplace of the enterprise technical library

Qualification work for obtaining the degree of Bachelor of Computer Science in the specialty 122 Computer Science. - LLC «TECHNICAL UNIVERSITY «METINVEST POLYTECHNIC» MES of Ukraine, Kryvyi Rih, 2025.

The aim of the work is to create a workplace automation program for a librarian of an enterprise technical library with the ability to maintain a book catalog, keep records of users with the preservation of their personal data and borrowing history.

Object of research - library processes that make up the work of libraries.

The subject of the study is the process of automation of library processes using specialized software.

Methodology – structured analysis, SADT methodology, modeling of information flows, object-oriented programming, database design, economic evaluation of the project.

The work analyzes the needs of the librarian, builds a logical model of the system, and creates a program for automating the workplace of the enterprise librarian. The user interface, authorization system, book catalog, user accounting, booking and borrowing of books are implemented. Technical and economic indicators were determined, and the payback period of the system was calculated.

The developed software product allows us to reduce staff labor costs, efficiently provide customer service and keep records of the library collection. The system is suitable for scaling and further expansion of functionality. The results can be used by other enterprises.

KEYWORDS: LIBRARY PROCESSES, AUTOMATION, AUTOMATED WORKSTATION, DATABASE, SOFTWARE, MYSQL, VISUAL STUDIO, INFORMATION FLOWS, ECONOMIC EFFECTIVENESS

ЗМІСТ

ВСТУП

Проект з розробки автоматизованого робочого місця бібліотекара був спрямований на створення інтегрованої системи, яка спрощує управління бібліотекою. Основні завдання включали розробку інтуїтивного інтерфейсу для бібліотекарів і користувачів, автоматизацію бібліотечних процесів, що включає ефективне адміністрування книг, авторів і позичань, а також забезпечення швидкого доступу до бібліотечних ресурсів та інші функції бібліотекара.

У межах проекту була створена система, що спрощує управління бібліотечним фондом. Вона дозволяє бібліотекарам ефективно каталогізувати книги, зазначаючи їх авторів, жанри, видання та доступність. Окрім цього, система забезпечує реєстрацію користувачів, зберігання їхніх персональних даних і історії позичань, що значно полегшує контроль за видачею та поверненням літератури.

Одним із центральних елементів проекту стало забезпечення ефективного пошуку та доступу до бібліотечних матеріалів. Автоматизована система включає потужний пошуковий механізм, що дозволяє користувачам швидко знаходити книги, журнали, наукові статті та інші джерела. Окрім цього, вона забезпечує доступ до електронних ресурсів, які можна переглядати онлайн або завантажувати для офлайн користування.

Одним із ключових викликів при розробці системи автоматизації бібліотечних процесів стала її інтеграція з іншими бібліотечними системами. Зокрема, вона успішно синхронізується з електронним каталогом, що дає змогу користувачам переглядати наявність книг у режимі реального часу. Крім того, система підтримує інтеграцію з електронними ресурсами, базами даних та інструментами управління

бібліотекою, що підвищує зручність та ефективність роботи з інформаційними матеріалами.

Захист даних та їх конфіденційність є критично важливими аспектами автоматизованої бібліотечної системи. Вся інформація про користувачів, книги та інші бібліотечні ресурси зберігається у безпечній базі даних із обмеженим доступом. Для забезпечення високого рівня безпеки застосовуються передові методи шифрування та автентифікації, що гарантують надійність збереження і доступу до даних.

Таким чином, впроваджена програма автоматизації робочого місця бібліотекара (в подальшому АРМ) значно спрощує роботу бібліотекарів і покращує обслуговування користувачів. Вона оптимізує управління ресурсами, забезпечує швидкий доступ до бібліотечних матеріалів та гарантує захист даних. Завдяки цьому бібліотека працює ефективніше, а користувачі отримують зручний та безпечний доступ до знань і інформації.

РОЗДІЛ 1. АНАЛІЗ СТАНУ АВТОМАТИЗАЦІЇ БІЗНЕС-ПРОЦЕСІВ В РОБОТІ БІБЛІОТЕКИ

1.1 Аналіз предметної області бібліотеки

Автоматизація бібліотечних процесів є важливим напрямом розвитку сучасних бібліотек, оскільки вона сприяє підвищенню ефективності управління ресурсами, покращенню доступу до інформації та оптимізації роботи персоналу.

Актуальність автоматизації обробки даних технічних бібліотек визначається необхідністю підвищення ефективності бібліотечних процесів:

- Оптимізація управління фондами – автоматизовані системи дозволяють швидко каталогізувати, зберігати та знаходити необхідні матеріали.

- Підвищення доступності – цифрові бібліотеки та електронні каталоги забезпечують доступ до ресурсів з будь-якого місця та пристрою.

- Ефективність обслуговування користувачів – автоматизовані бібліотечні інформаційні системи (АБІС) спрощують процес видачі та повернення книг, а також дозволяють персоналізувати рекомендації.

- Зменшення витрат – автоматизація скорочує потребу в ручній обробці даних, що знижує витрати на адміністрування.

- Інтеграція з іншими інформаційними системами – бібліотеки можуть взаємодіяти з корпоративними базами даних, науковими репозитаріями та освітніми платформами.

Мета роботи полягала у створенні інтегрованої системи автоматизації бібліотечних процесів із зручним та інтуїтивно зрозумілим інтерфейсом.

Для досягнення цієї мети потрібно виконати наступні задачі:

1. Оцінити наявні веб-сайти та платформи для керування бібліотечними ресурсами, визначити їх сильні та слабкі сторони, а також сформулювати ключові вимоги до розробки нової системи.

2. Спроєктувати архітектуру нової системи, яка відповідатиме всім вимогам користувачів, забезпечуватиме зручність та доступність у використанні як для бібліотечного персоналу, так і для відвідувачів платформи.

3. Виконати оцінку доступних інструментів, бібліотек і платформ для розробки системи, визначивши їх можливості та відповідність поставленим вимогам.

4. Створити веб-додаток із вбудованою базою даних для збереження інформації відповідно до розробленої архітектури.

Веб-додаток повинен мати наступний функціонал

1. Реєстрація облікового запису для адміністрування та управління контентом сайту.

2. Функціонал для додавання, редагування та видалення інформації про книги та їхні додаткові характеристики.

3. Функція інтегрованого пошуку бібліотечних ресурсів за різними критеріями, такими як назва, автор, рік видання та категорія.

Об’єктом автоматизації є бібліотечні процеси, з яких складається робота бібліотеки.

Бібліотека є важливим культурним центром, що забезпечує доступ до друкованих та цифрових ресурсів, сприяючи освіті, науковим дослідженням та особистісному розвитку. Вона відіграє ключову роль у збереженні, організації та поширенні знань, літератури, мистецтва та наукових напрацювань.

У сфері бібліотечної діяльності важливо враховувати такі аспекти, як книжковий фонд, автори, жанри, користувачі, процеси позичання та повернення книг, каталогізацію, архівування та збереження документів. Бібліотекарі відіграють центральну роль у підтримці та розвитку бібліотеки, забезпечуючи ефективне управління ресурсами та доступ до інформації.

Основні функції бібліотеки охоплюють формування книжкових колекцій, каталогізацію та класифікацію документів, управління процесами позичання та повернення книг, а також ведення обліку користувачів. Важливим аспектом є забезпечення безпеки даних і конфіденційності інформації.

З огляду на різноманітність та складність бібліотечних процесів, впровадження системи автоматизації є ключовим фактором для підвищення ефективності роботи, покращення якості обслуговування та створення більш зручного середовища для користувачів.

Перевага створення єдиної системи автоматизації бібліотечних процесів полягає в оптимізації роботи з книжковим фондом: вона автоматично веде облік книг та інших ресурсів, спрощує їх пошук і каталогізацію. Це значно підвищує ефективність організації та збереження документів. Крім того, система забезпечує користувачам зручний інтерфейс для пошуку, аналізу та доступу до необхідної інформації, дозволяючи швидко знаходити книги, статті, журнали та інші матеріали.

1.2 Огляд аналогів програмного забезпечення для автоматизації роботи сучасних бібліотек

У сучасному світі існує широкий спектр систем та технологій для управління бібліотечними ресурсами. Одним із ключових завдань бібліотек є вибір оптимальних інструментів, які відповідають їхнім потребам та сприяють удосконаленню робочих процесів. У цьому розділі ми детально розглянемо кілька таких систем, зокрема DSpace, визначимо їхні переваги та недоліки, щоб оцінити їхню ефективність для нашої бібліотеки. Аналіз характеристик та функціональних можливостей допоможе ухвалити обґрунтоване рішення та обрати найбільш перспективний напрямок розвитку бібліотеки.

DSpace – це пакет відкритого програмного забезпечення, розроблений для управління цифровими активами та використання як основа для колективних архівів [1] . Він також слугує платформою для цифрового зберігання даних.

З моменту свого першого випуску у 2002 році, створеного в рамках співпраці HP та MIT, DSpace знайшов застосування у понад 1400 установах по всьому світу. Його використовують університети, коледжі, культурні організації та дослідницькі центри. Система підтримує різноманітні типи даних, зокрема книги, дисертації, 3D-сканування, фотографії, відео, наукові набори даних та інший цифровий контент.

Розповсюджується за ліцензією BSD, що дає можливість користувачам адаптувати та розширювати функціональність відповідно до своїх потреб.

Перша версія DSpace була представлена в листопаді 2002 року завдяки спільним зусиллям розробників із Массачусетського

технологічного інституту (MIT) та дослідницької лабораторії HP у Кембриджі, штат Массачусетс.

17 липня 2007 року компанія HP та MIT оголосили про заснування Фонду DSpace – некомерційної організації, покликаної координувати розвиток платформи та надавати підтримку спільноті користувачів DSpace.

12 травня 2009 року Fedora Commons та Фонд DSpace об'єднали свої зусилля, створивши спільну некомерційну організацію під назвою DuraSpace. Її основна місія – впровадження інновацій та розвиток технологій відкритого коду і хмарних рішень для бібліотек, університетів, наукових центрів та культурних установ. DuraSpace координує діяльність та надає підтримку спільнотам DSpace і Fedora.

DSpace побудований на основі Java та JSP, використовуючи Java Servlet Framework. Він працює з реляційними базами даних і підтримує PostgreSQL та Oracle. На даний момент платформа пропонує два основні веб-інтерфейси:

- JSPUI – класичний інтерфейс, що базується на JSP та Java Servlet API.

- XMLUI – новий інтерфейс, заснований на Apache Cocoon, з використанням XML та XSLT.

Записи DSpace доступні через веб-інтерфейс, а також підтримується протокол OAI-PMH (версія 2.0) та можливість експорту метаданих у форматі METS.

На базі DSpace створено IRTUMIP - Інституційний репозитарій ТОВ "ТЕХНІЧНИЙ УНІВЕРСИТЕТ «МЕТІНВЕСТ ПОЛІТЕХНІКА» [2], який є платформою для збирання, систематизації та збереження інтелектуальних напрацювань університетської спільноти. Він сприяє поширенню цих матеріалів у цифровому форматі через Інтернет-технології, забезпечуючи інтеграцію у світове науково-освітнє середовище(Рис.1.1).

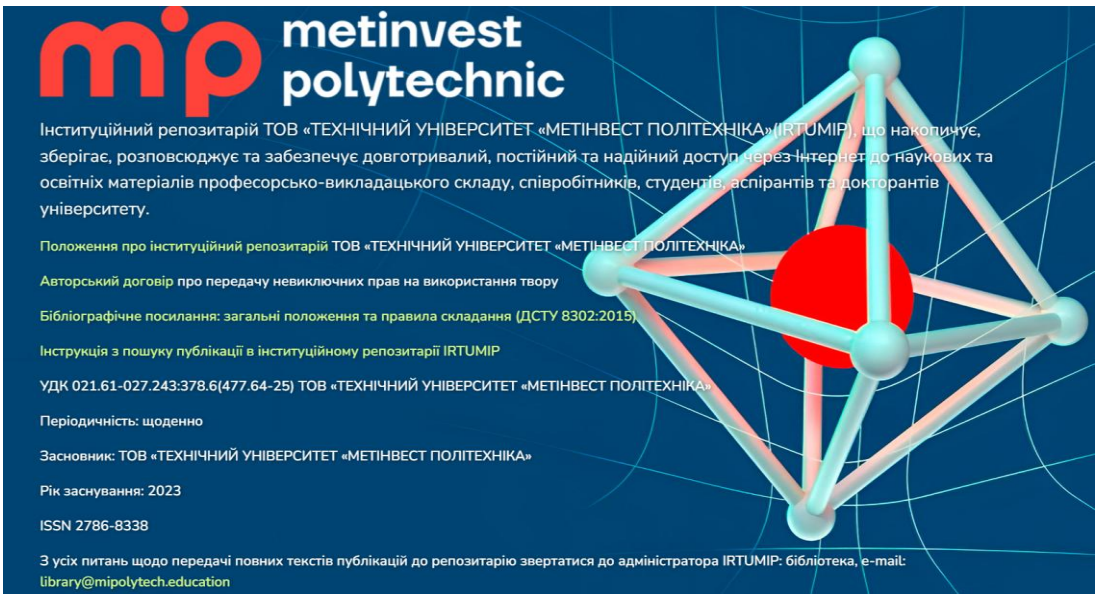

Рисунок 1.1 – Інтерфейс репозиторію ТОВ "ТЕХНІЧНИЙ УНІВЕРСИТЕТ «МЕТІНВЕСТ ПОЛІТЕХНІКА»

Kortext – це провідна платформа для персоналізованого навчання, що надає доступ до електронних книг і цифрового освітнього контенту від авторитетних видавців . Вона включає понад 13 000 безкоштовних книг та спеціалізовані видання, загальна кількість яких перевищує 2 мільйони екземплярів від 4 800 видавництв по всьому світу, доступних за додатковою підпискою.

Платформа також надає можливість розміщення контенту, на який університет має авторські права. Вбудована функція перекладу сприяє не лише читанню англійською мовою, але й удосконаленню навичок її розуміння.

Посилання на навчальні матеріали Kortext інтегровані в освітні курси та використовуються в середовищі Moodle, що розширює можливості контентного забезпечення навчальних програм.

Бібліотека ТОВ "ТЕХНІЧНИЙ УНІВЕРСИТЕТ «МЕТІНВЕСТ ПОЛІТЕХНІКА» забезпечує доступ до платформи Kortex [3] (рис.2.1)

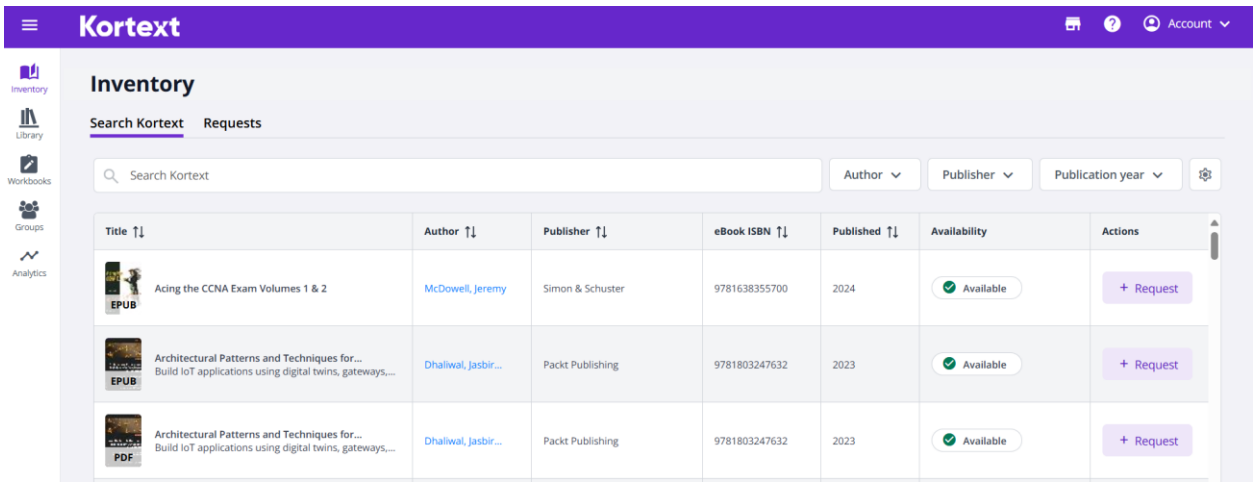

Рисунок 1.2 – Платформа Kortex.

Digital Commons – це платформа, створена компанією Bepress, яка дозволяє університетам і коледжам організовувати власні репозиторії для збереження та поширення наукових матеріалів. Завдяки своєму широкому функціоналу вона є привабливим рішенням для академічних спільнот [4].

Однією з ключових переваг Digital Commons є зручні інструменти для завантаження, каталогізації та структурованого зберігання документів, таких як наукові статті, тези, звіти, дисертації та інші освітні матеріали. Крім того, платформа підтримує інтеграцію з різними системами та базами даних, що сприяє підвищенню доступності та видимості наукових робіт.

Ще однією перевагою Digital Commons є його зручний користувацький інтерфейс (рис. 1.3), розроблений з урахуванням потреб користувачів. Він забезпечує інтуїтивну навігацію, розширені пошукові можливості та простий процес завантаження й організації документів, що значно спрощує доступ до наукових матеріалів і взаємодію з ними.

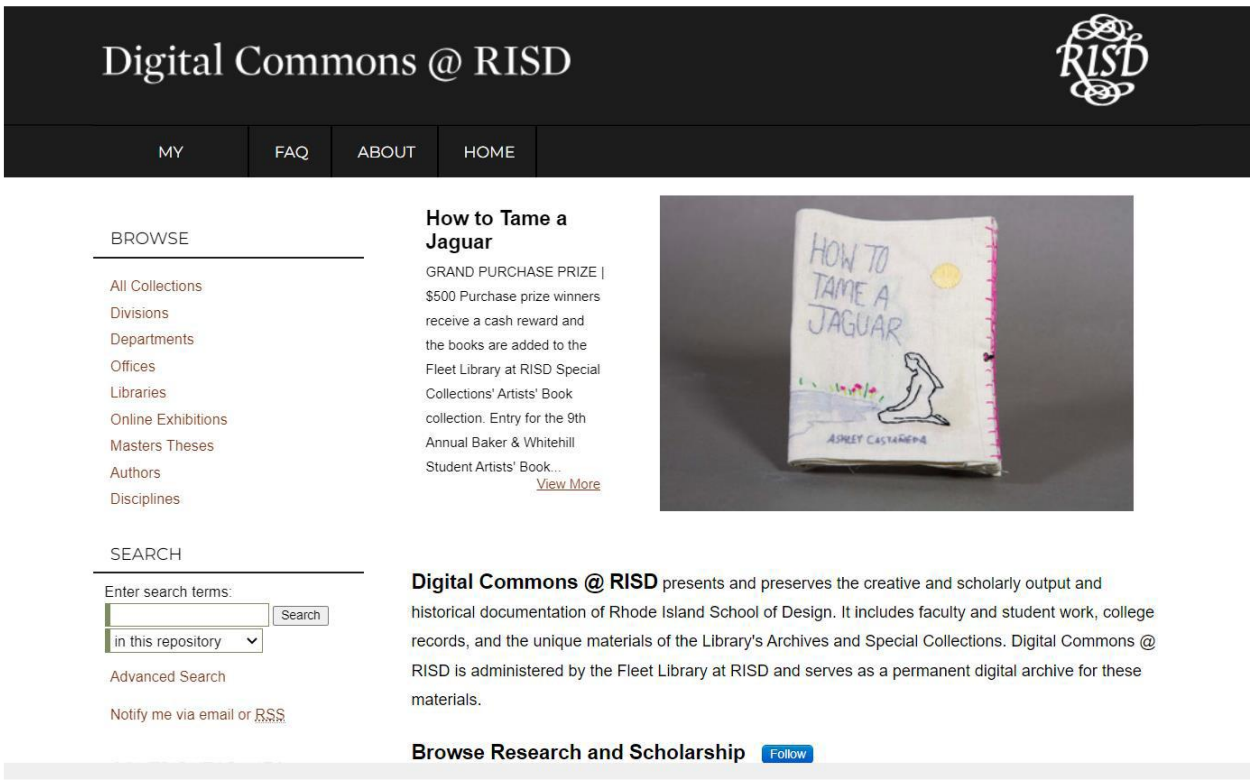

Рисунок 1.3 – Користувацьки інтерфейс платформи Digital Commons

Попри численні переваги, платформа Digital Commons має й певні недоліки, які слід враховувати. Одним із них є її комерційний характер, що може спричиняти значні фінансові витрати для університетів і коледжів, особливо для невеликих навчальних закладів із обмеженим бюджетом. Крім того, обмежена гнучкість налаштувань може створювати труднощі в адаптації платформи під специфічні потреби кожної установи.

Аналіз показав, що Digital Commons є ефективним інструментом для збереження та поширення наукових матеріалів в академічному середовищі. Її функціонал і зручний інтерфейс полегшують доступ до наукових робіт. Водночас важливо враховувати фінансову модель платформи та її можливості кастомізації, які можуть впливати на її застосування в конкретних умовах.

Система автоматизації бібліотек ІРБІС, розроблена асоціацією ЕБНІТ, широко використовується в українських бібліотеках, зокрема Національною бібліотекою України імені В. І. Вернадського [5], Національною науковою медичною бібліотекою та іншими закладами. Вона забезпечує роботу в локальних мережах без обмежень кількості користувачів, легко інтегрується в корпоративні бібліотечні системи, підтримує створення електронного каталогу та словників для швидкого пошуку. Крім того, ІРБІС має інструменти для ведення авторитетних файлів, алфавітного покажчика і тезауруса, дозволяє друкувати документацію, використовувати штрих-коди, переглядати тексти й графіку через Інтернет, змінювати мовний інтерфейс, а також адаптувати систему під потреби окремої бібліотеки. Користувачі можуть самостійно вносити зміни для зручності роботи.

Система автоматизації бібліотек, розроблена ТОВ «Український фондовий дім» у Києві, є комплексним рішенням для бібліотечних установ. Вона виконує всі ключові виробничі процеси та широко використовується в Україні, зокрема в університетських, публічних і відомчих бібліотеках, де вже встановлено понад 200 інсталяцій. Система підтримує як український, так і російський інтерфейси.

Основні можливості:

- ефективний пошук, сортування та перегляд інформації електронного каталогу з можливістю друку;

- каталогізація документів, включаючи підтримку класифікаторів (ББК, УДК, Дьюі), авторитетних записів та складання бібліографічних довідок;

- управління бібліотечними фондами, що охоплює роботу з книгопостачальниками, аналіз потреб у літературі та замовлення на її придбання;

- обслуговування читачів із веденням їхньої інформації, реєстрацією повернення документів, а також підтримкою технологій штрих-кодування та RFID.

UniLib–клієнт-серверна система, розроблена в Харкові, що базується на безкоштовній кросплатформеній СУБД Firebird 2.5. Вона не потребує витрат на підтримку бази даних, працює стабільно без спеціального втручання, а також може функціонувати на застарілому обладнанні без втрати продуктивності. Firebird не вимагає спеціального апаратного забезпечення чи навчених технічних фахівців, має високу стійкість до збоїв і гарантує безпеку збереження даних.

Система забезпечує підтримку Unicode, дозволяючи працювати з будь-якими символами, та надає можливість створення необмеженої кількості робочих місць. Дані зберігаються централізовано на сервері, що усуває потребу в дублюванні чи реплікації, а встановлена програма може використовуватися без додаткових витрат часу. UniLib підтримує всі стандартні бібліотечні процеси, враховуючи специфіку різних типів бібліотек в Україні. Вона є масштабованою, що дозволяє ефективно обслуговувати як невеликі бібліотеки, так і великі установи з розгалуженою мережею філій.

Koha – перша вільна автоматизована бібліотечна інформаційна система (АБІС) з відкритими вихідними кодами, розроблена у 1999 році компанією Katipo Communications для бібліотечної спілки Гороугенуа в Новій Зеландії [6]. Перша інсталяція відбулася в січні 2000 року. Назва «Koha» у мові маорі означає «подарунок» або «дар».

Система надає зручний та інтуїтивний інтерфейс для бібліотекарів і читачів, підтримує гнучке налаштування пошуку та модуль каталогізації з інтегрованим клієнтом Z39.50/SRU. Вона охоплює повний цикл надходжень, включаючи бюджетні витрати, цінові дані та співпрацю з постачальниками, а також спрощену систему надходжень для невеликих бібліотек. Koha дозволяє працювати з необмеженою

кількістю підрозділів, категорій читачів і документів, валют та інших параметрів. Додатково передбачено функціонал для управління періодичними виданнями та формування списків прочитаного.

Основними перевагами системи є її відкритий код, що сприяє міжнародному співробітництву та обміну даними між понад 15 000 бібліотек світу. Спільнота користувачів має можливість спільно інвестувати в розвиток нового функціоналу, виконувати самостійні оновлення та налаштування, а також інтегрувати систему з іншими бібліотечними сервісами, включаючи RFID та системи доступу. Відкритий доступ до програмного коду дозволяє оперативно виправляти помилки через участь користувачів.

Недоліком системи є її недостатня адаптованість до окремих аспектів українського законодавства та наявність незначних технічних проблем. Хоча базові процеси перевірені багатьма користувачами, специфічні запити бібліотек можуть потребувати додаткових налаштувань.

Система ALEPH, розроблена в Єврейському університеті Єрусалима, є інноваційним рішенням для бібліотек, що успішно функціонує у 500 установах по всьому світу [7]. Її підтримкою та розвитком займається компанія Aleph Yissum, а українські користувачі отримують супровід від ExLibris у Києві.

Система включає низку інтегрованих модулів: OPAC (онлайновий публічний доступ до каталогу), каталогізацію, комплектування, циркуляцію, адміністрування, облік періодики, міжбібліотечний абонемент та інші. Вона базується на відкритих системах, забезпечуючи гнучкість та можливість адаптації.

Особливістю ALEPH є підтримка понад 20 мов, використання MARC-сумісних форматів, ISO-стандартів, інформаційно-пошукових мов, авторитетних файлів та штрихових кодів. Вона забезпечує роботу в глобальних інформаційних мережах через World-Wide Web,

дозволяючи користувачам отримати доступ через веб- або Windows-інтерфейс.

Система створена для оптимізації бібліотечних процесів, зменшення часу на облік та опрацювання фондів, а також підвищення комфортності обслуговування читачів. Вона функціонує у провідних бібліотеках України, зокрема у ЛННБУ ім. В. Стефаника, бібліотеці НаУКМА, КПІ ім. І. Сікорського та Київській бібліотеці ім. Лесі Українки.

РОЗДІЛ 2. МОДЕЛЮВАННЯ ПРОЦЕСІВ АВТОМАТИЗАЦІЇ РОБОТИ ТЕХНІЧНОЇ БІБЛІОТЕКИ ПІДПРИЄМСТВА

2.1 Вибір засобів розробки системи

Аналіз предметної області та існуючих рішень дозволив визначити ключові потреби користувачів та виокремити найактуальніші бібліотеки для розробки АРМ. У процесі вибору інструментів для створення цієї системи були враховані різні критерії, що допомогло уникнути недоліків попередніх рішень та забезпечити конкурентні переваги. Розглянемо програмні засоби, що використовувалися при розробці АРМ.

Javascript. Народження JavaScript датується серединою 1990-х років [8,9]. Спочатку він був розроблений для роботи в конкретному веб-браузері, який називався Netscape Navigator. Але, оскільки його популярність швидко зростала, його адаптували і до інших веб-браузерів, за що він отримав прізвисько "мова Інтернету" (Miller, 2018). JavaScript - це, по суті, об'єктно-орієнтована мова програмування, зазвичай вона найбільш відома завдяки тому, що використовується як мова сценаріїв у веб-розробці на стороні клієнта. Отже, в цій ситуації вона запускається в браузері і використовується для взаємодії з графічним інтерфейсом користувача. Це робиться для того, щоб досягти більш динамічних та інтерактивних веб-сайтів, динамічно адаптуючи відображення сторінки, контролюючи те, що з'являється на сторінці за певних умов. Отже, таке використання JavaScript більш відоме як JavaScript на стороні клієнта/

Відомо, що JavaScript схожа на інші популярні об'єктно-орієнтовані мови програмування, такі як C++ та Java. Таким чином, JavaScript має

багато можливостей, які очікуються від об'єктно-орієнтованої мови програмування, проте в більш легкому сенсі. Але, в принципі, багато концепцій підтримуються JavaScript. Різні типи даних, такі як цілі числа, рядки, булеві, а також масиви, логічні операції, такі як порівняння, умовне виконання, такі як оператори if та цикли. Таким чином, можливості, які надає JavaScript для веб-розробки, дозволяють створювати ширші веб-додатки, особливо для кращої взаємодії з користувачем. Можна знайти багато рішень для сучасних проблем веб-розробки: обробка взаємодії користувача з HTML-формами та елементами форм, обробка розподілу даних, навіть забезпечення зв'язку з серверами для надсилання та отримання даних.

З іншого боку, незважаючи на те, що між JavaScript та вищезгаданими мовами програмування високого рівня, такими як C++ та Java, існує певна структурна схожість, JavaScript має певні обмеження у порівнянні з ними. Перш за все, JavaScript має полегшений синтаксис, також відомий як "Java-lite", не такий суворий до правил синтаксису, як вищезгадані мови високого рівня. Крім того, наприклад, на відміну від цих мов програмування високого рівня, JavaScript не має можливостей взаємодії з операційною системою клієнтського пристрою, таких як доступ до клієнтських засобів вводу/виводу, запуск програм і читання або запис файлів.

Оскільки JavaScript стала найпоширенішою мовою програмування для веб-розробки за останні роки, фреймворки, написані на JavaScript, з'являються так само часто. Потреба у фреймворках виникає, коли веб-додатки стають складнішими та важчими в підтримці. Визначення фреймворку можна сформулювати як групу написаних утиліт, функцій або бібліотек на мові програмування для створення і розміщення коду для багаторазового використання для вирішення конкретних завдань розробки. Наведемо кілька прикладів, адже одними з найпопулярніших фреймворків інтерфейсу користувача для JavaScript є React, Angular та

Vue. Такі фреймворки дають розробникам можливість швидко і легко створювати і підтримувати сучасний, складний, адаптивний інтерфейс користувача для веб-додатків.

Хоча JavaScript здебільшого відома як клієнтська мова програмування, яка запускається у браузері, розробники шукали способи запуску JavaScript в інших середовищах. Однією з найочевидніших причин для цього є те, що розробники могли б також розробляти серверні додатки, використовуючи JavaScript. Таким чином, повноцінний веб-додаток можна було б розробити за допомогою JavaScript, не розробляючи серверний додаток іншою мовою, такою як PHP, Perl або Python. Таким чином, існували способи створення середовищ або рушіїв, які дозволяли запускати JavaScript в інших середовищах розробки. Сьогодні одним з найбільш поширених середовищ для виконання JavaScript є NodeJS, яке широко використовується у багатьох серверних додатках.

React. Створення інтерфейсу користувача для веб-додатків за допомогою фронтенд-фреймворку JavaScript є дуже поширеним явищем у наш час. ReactJS [10] (більш відомий як React) є одним з найпоширеніших фреймворків. React був розроблений і підтримується компанією Facebook, спочатку для вирішення проблеми з інтерфейсом користувача, з якою нещодавно зіткнулася їхня команда розробників, а потім був випущений як бібліотека у 2013 році [11]. Основна мета React - легко та швидко створювати інтерактивний та реактивний інтерфейс користувача. Це досягається шляхом створення фрагментів елементів інтерфейсу, які можна використовувати повторно, так званих "компонентів".

React керує елементом, який називається "Віртуальний DOM", копією HTML DOM. Кожного разу, коли в React-компоненті змінюється стан, піддерева вузлів рендерингуються відповідно до цього віртуального DOM, замість того, щоб перерендерити HTML DOM.

Різниця між віртуальним DOM та HTML DOM порівнюється, і в HTML DOM оновлюється лише та секція або секції, які містять зміни. Це призводить до меншої кількості втручань в HTML DOM, що робить його значно легшим, ніж деякі інші подібні фреймворки для фронтенду.

React-компоненти можуть бути написані звичайним нативним синтаксисом JavaScript. Однак існує спеціальний синтаксис, який можна використовувати для створення React-компонентів, який називається JavaScript XML (JSX). Цей синтаксис схожий на синтаксис будь-якої мови розмітки, наприклад, XML або HTML. Таким чином, оскільки це дозволяє створювати компоненти за синтаксисом, подібним до HTML, це полегшує як написання, так і читання в цілому. Хоча React-компоненти можна писати і без використання синтаксису JSX, він не такий популярний і здебільшого не рекомендується. Тому що, окрім того, що структуру розмітки легше читати та візуалізувати, кількість рядків коду, що пишеться, також зменшується. Крім того, React має можливість конвертувати синтаксис JSX у власний синтаксис JavaScript. Це перетворення здійснюється за допомогою JSX-трансформатора, який перетворює JSX в JavaScript у браузері. Таким чином, перетворення відбувається не під час виконання, а перед розгортанням додатку, за допомогою JSX-трансформатора. Прикладом JSX-трансформатора є Babel.

Раніше згадувалося, що багаторазові фрагменти елементів інтерфейсу користувача, які можна створювати за допомогою React і які називаються "компонентами", елементи, які називаються "компонентами", мають безліч власних функцій і властивостей, починаючи від управління своїми даними, станами відображення і навіть можуть бути використані для налагодження. Дані, що містяться в компоненті, можуть бути як внутрішніми, так і зовнішніми. Це означає, що дані можуть бути ініціалізовані та доступні всередині компонента, що називається станом, але також можуть бути доступні ззовні компонента,

що називається реквізитом. Крім того, компонент має обов'язковий метод "render", який здебільшого повертає єдиний макет JSX, який є відображенням компонента.

NodeJS. Сьогодні такі мови, як HTML5, JavaScript та фреймворки, які використовуються для розробки веб-додатків, дозволяють розробляти складні додатки. Крім того, за допомогою адаптивного підходу до дизайну можна розробляти додатки, які схожі на мобільні нативні додатки. Таким чином, надання та отримання великого потоку даних у стабільному вигляді клієнтським додатком є дуже важливим. При такій складності поточних стандартних веб-додатків, концепція інтеграції розробки на стороні клієнта і сервера, тобто розробка серверних додатків за допомогою JavaScript, є дуже бажаною.

NodeJS був вперше представлений у 2009 році як платформа для запуску JavaScript на серверах. Відтоді вона стала надійним вибором для розробників для створення серверних додатків для складних веб-додатків. NodeJS як фреймворк забезпечує середовище для створення масштабованих і швидких бекенд-додатків, а також хорошу продуктивність, працюючи на движку JavaScript V8 від Google [12].

JavaScript Option Notation, також відома як JSON, - це нотація, яка використовується для надання даних кінцевій точці. JSON є підмножиною JavaScript і є дуже корисним у наданні користувацькому застосунку даних, які інтерпретуються за допомогою JSON. Таким чином, на стороні клієнта отримані дані можуть бути легко проаналізовані та інтегровані. NodeJS працює як однопотоковий додаток. Це означає, що кожен запит обробляється в одному потоці, замість того, щоб створювати потік для кожної операції, як в інших подібних бекенд-середовищах. Таким чином, NodeJS забезпечує неблокуючу модель вводу/виводу з асинхронним програмуванням за допомогою асинхронних інструментів JavaScript, таких як Promises та async/await. Коли сервер створюється за допомогою NodeJS, він

прослуховує порт для прийому вхідних запитів, коли запит отримано, він пізніше розміщується і обробляється в циклі обробки подій NodeJS. Таким чином, за допомогою NodeJS створюється кероване подіями асинхронне середовище на стороні сервера.

MySQL. MySQL - це реляційна система баз даних з відкритим вихідним кодом, яка є відносно швидкою, стабільною і легкою в освоєнні, тому дуже популярною. Вона має широкий спектр функцій, які притаманні якісній, стандартній системі баз даних. Як випливає з назви, MySQL підтримує мову структурованих запитів (SQL), яка є стандартною мовою запитів до бази даних [13].

Як і багато інших, MySQL насправді є системою клієнт/сервер, тобто транзакція між базою даних і користувачем здійснюється між клієнтською програмою і сервером, на якому зберігаються дані. Таким чином, транзакції з базою даних, такі як запити до даних, збереження або редагування даних, надсилаються клієнтом і обробляються на сервері MySQL. Крім того, MySQL легко інтегрується з NodeJS.

Express. Express - це фреймворк, який використовується в серверних додатках NodeJS і забезпечує більш просте використання основної функціональності NodeJS [14]. При цьому однією з головних цілей фреймворку є мінімалізм, гнучкість, швидкість, а також простота встановлення та налаштування для бекенд-розробки.

Одна з особливостей Express називається "проміжне програмне забезпечення" і використовується для виконання запитів і відповідей до/від сервера. Проміжне програмне забезпечення можна отримати у вигляді сервісів, які технічно є функціями. Ці функції проміжного програмного забезпечення відповідають за управління як HTTP-запитами, так і відповідями. Таким чином, за допомогою проміжного програмного забезпечення запити розбиваються на менші частини, відповідно, обробляючи одну секцію.

Ще одна особливість Express, яка називається "маршрутизація", дозволяє обробляти різні запити різними обробниками запитів. Зокрема, створюються певні маршрути, за якими слідує функція проміжного програмного забезпечення. Коли один з цих маршрутів відвідується, виконується відповідна функція. Як простий приклад, ми можемо створити такий маршрут "/hello", з функцією, яка повертає значення "Hello World", при HTTP.GET запиті до цього маршруту отримується значення "Hello World". Отже, ці маршрути можна використовувати як кінцеві точки, з яких клієнтська програма надсилає запити до сервера.

2.2 Побудова логічної моделі системи управління бібліотекою

У цьому розділі представлено логічну структуру бібліотечних процесів підприємства та функціональну модель системи на етапі до її впровадження. Основною метою є формалізація предметної області з метою подальшого проєктування та реалізації програмного комплексу.

Бібліотечні процеси охоплюють повний життєвий цикл роботи з інформаційними ресурсами та обслуговуванням користувачів. Вони включають в себе формування бібліотечного фонду(аналіз інформаційних потреб користувачів, вибір, закупівля, передплата джерел (друкованих та електронних), реєстрація та облік надходжень), обробка та організація інформаційних ресурсів(бібліографічний опис документів, індексація та систематизація за класифікаційними системами (УДК, ББК), створення електронних та карткових каталогів), надання доступу та обслуговування користувачів (видача ресурсів у користування (читальні зали, абонемент, електронний доступ),

консультації, довідково-бібліографічне обслуговування, реєстрація та підтримка читачів), облік, збереження та списання фонду (проведення інвентаризації, забезпечення умов для збереження документів, вибракування та списання застарілих або зношених матеріалів), довідково-інформаційна та просвітницька діяльність(проведення виставок, оглядів, культурно-просвітницьких заходів, підготовка бібліографічних покажчиків, тематичних добірок)

Логічна модель дозволяє ідентифікувати основні інформаційні потоки, встановити вхідні та вихідні документи, а також визначити виконавців на кожному етапі процесу. Для формалізації функціональних блоків застосовано SADT-діаграми.

Методологія SADT (Structured Analysis and Design Technique) передбачає ієрархічну декомпозицію об’єкта дослідження на підпроцеси, кожен з яких характеризується чітко визначеними входами, виходами, механізмами реалізації та елементами управління.

SADT зарекомендувала себе як ефективний інструмент структурного аналізу та проєктування систем у різних галузях, зокрема:

- розробка програмного забезпечення для телекомунікацій;
- системне обслуговування та технічна діагностика;
- стратегічне і довгострокове планування;
- автоматизоване проєктування та виробництво;
- конфігурація обчислювальних систем;
- підготовка та навчання персоналу;
- розробка вбудованих систем для оборонної сфери;
- управління фінансами та матеріально-технічне забезпечення .

SADT вирізняється серед інших методів завдяки таким ключовим перевагам:

- забезпечує відображення таких критичних характеристик системи, як управління, зворотний зв’язок та взаємодія виконавців;

- підтримує командну роботу через розвинуті процедурні механізми;

- ефективна на ранніх етапах проєктування;

- легко інтегрується з іншими структурними підходами, виконуючи функцію зв’язувальної ланки між різнорівневими описами систем.

На рисунку 2.1 представлено SADT-діаграму нульового рівня процесу А-0 «Автоматизоване робоче місце бібліотекара технічної бібліотеки підприємства». Її детальний опис наведено в таблиці 2.1.

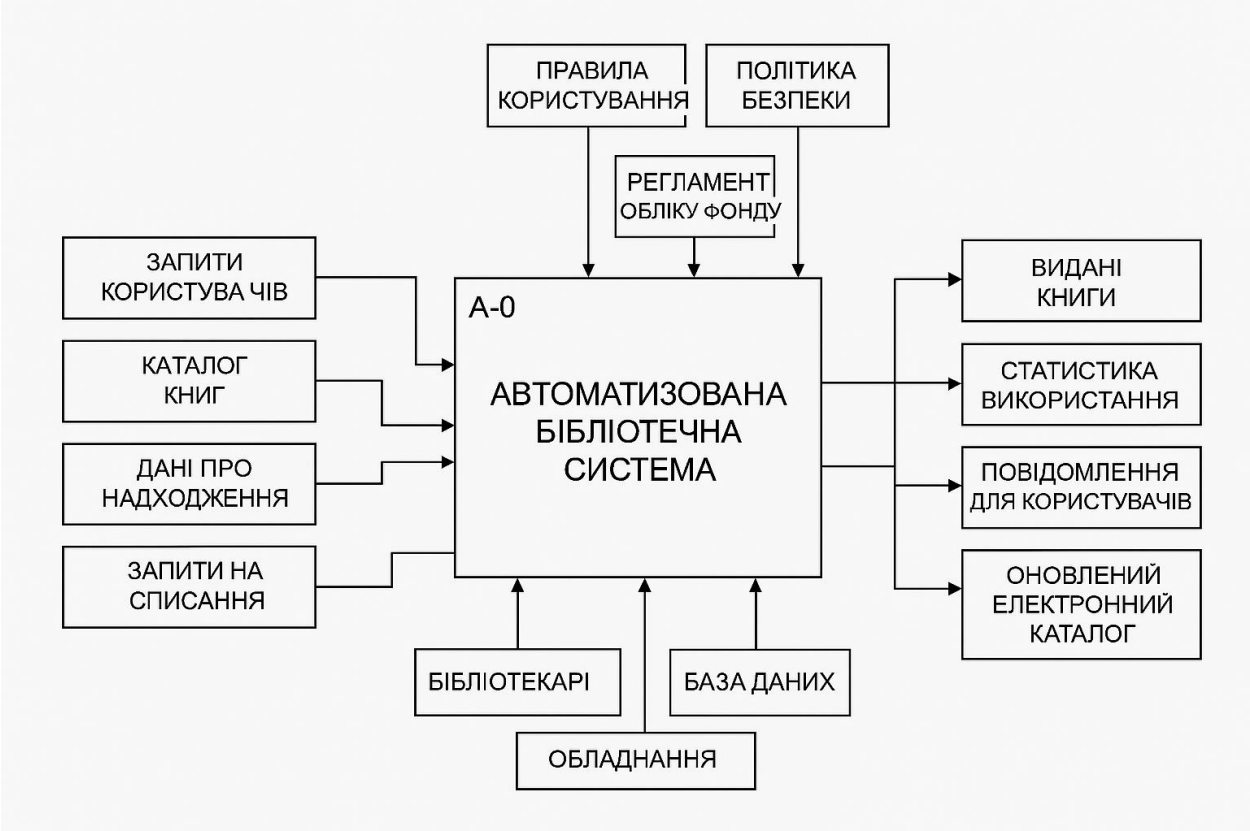

Рисунок 2.1 - SADT-діаграма 0-рівня процесу А-0 «Автоматизоване робоче місце бібліотекара технічної бібліотеки підприємства»

Таблиця 2.1 - Опис SADT-діаграми 0 рівня процесу А-0 «Автоматизоване робоче місце бібліотекара технічної бібліотеки підприємства»

№	Вхідні дані	Управління	Механізми	Вихідні дані
A0	Запити користувачів Каталог книг Дані про надходження Запити на списання	Правила користування Політика безпеки Регламент обліку фонду	Бібліотекарі Інформаційна система База даних Обладнання (сканери, термінали тощо)	Видані книги Статистика використання Повідомлення для користувачів Оновлений електронний каталог

Зробимо декомпозицію процесу А0 на підпроцеси(табл.2.2)

Позначення	Назва процесу	Опис функції
A1	Реєстрація користувачів	Додавання нових читачів, перевірка даних, створення облікових записів
A2	Пошук і бронювання літератури	Надання інтерфейсу для пошуку, фільтрації та резервування книг
A3	Видача та повернення книг	Оформлення видачі, контроль термінів, облік повернення
A4	Облік фонду та інвентаризація	Ведення електронного каталогу, списання зношених примірників, перевірка складу
A5	Генерація звітів і аналітики	Формування статистичних звітів щодо відвідуваності, популярності літератури

В таблиці 2.2 наведено вхідні дані, управління, механізми та вихідні дані для цих підпроцесів.

Таблиця 2.2 - Опис SADT-діаграми 1 рівня підпроцесу А-1 «Автоматизоване робоче місце бібліотекара технічної бібліотеки підприємства»

Блок	Назва функції	Вхідні дані	Керування	Механізми	Вихідні дані
A1	Реєстрація користувачів	Документи користувача	Правила бібліотеки, політика доступу	АБІС, бібліотекар, база даних користувачів	Зареєстрований користувач
A2	Пошук і бронювання літератури	Пошукові запити	Правила бібліотеки	АБІС, каталог книг	Результати пошуку, заброньовані книги
A3	Видача та повернення книг	Запити на видачу/повернення	Правила бібліотеки, інструкції з обслуговування	АБІС, бібліотекар	Видані/повернуті книги
A4	Облік фонду та інвентаризація	Надходження/списання книг	Правила бібліотеки, регламент обліку фонду	АБІС, бібліотекар	Оновлений фонд, інвентаризаційні звіти
A5	Генерація звітів і аналітики	Дані про діяльність	Правила бібліотеки, алгоритми формування звітів	АБІС, аналітичні модулі	Аналітичні та статистичні звіти

РОЗДІЛ 3. ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ ПРОГРАМИ АВТОМАТИЗАЦІЇ РОБОЧОГО МІСЦЯ БІБЛІОТЕКАРА ТЕХНІЧНОЇ БІБЛІОТЕКИ ПІДПРИЄМСТВА

Система управління бібліотекою, яку також називають LMS (що означає "система управління бібліотекою"), - це адаптивний веб-додаток, розроблений для управління та перегляду бібліотеки України в освітньому середовищі, як для користувачів бібліотеки, так і для її співробітників. Такий додаток покращить якість освіти в напруженому повсякденному житті студентів чи викладачів, і не менш важливо, щоб працівники бібліотек виконували свою роботу легко та ефективно

Гості, які не є зареєстрованими користувачами, можуть шукати ресурс (книгу, статтю, журнал, газету тощо), переглядати деталі ресурсу, шукати автора, переглядати ресурси за автором і шукати ресурси на полиці або на поверсі. Крім того, користувачі бібліотеки можуть увійти до свого облікового запису LMS, щоб зробити запит на позику ресурсу, взяти ресурс, зарезервувати ресурс, якщо він наразі позичений іншим користувачем, а також відстежувати будь-які процеси позики або резервування, які відбуваються в даний момент у них.

З іншого боку, додаток надає більше функціональних можливостей для працівників бібліотеки. Окрім пошуку та перегляду ресурсів, типів ресурсів, авторів, поверхів та полиць, працівники бібліотеки також мають права адміністратора, щоб керувати цими елементами, додаючи, редагуючи та видаляючи ресурси, типи ресурсів, авторів, поверхи та полиці. Співробітники бібліотеки також можуть перевіряти і схвалювати/відхиляти запити на видачу, а також переглядати будь-який процес видачі або резервування. На рис. 3.1 показана діаграма діаграма варіантів використання розроблювальної системи.

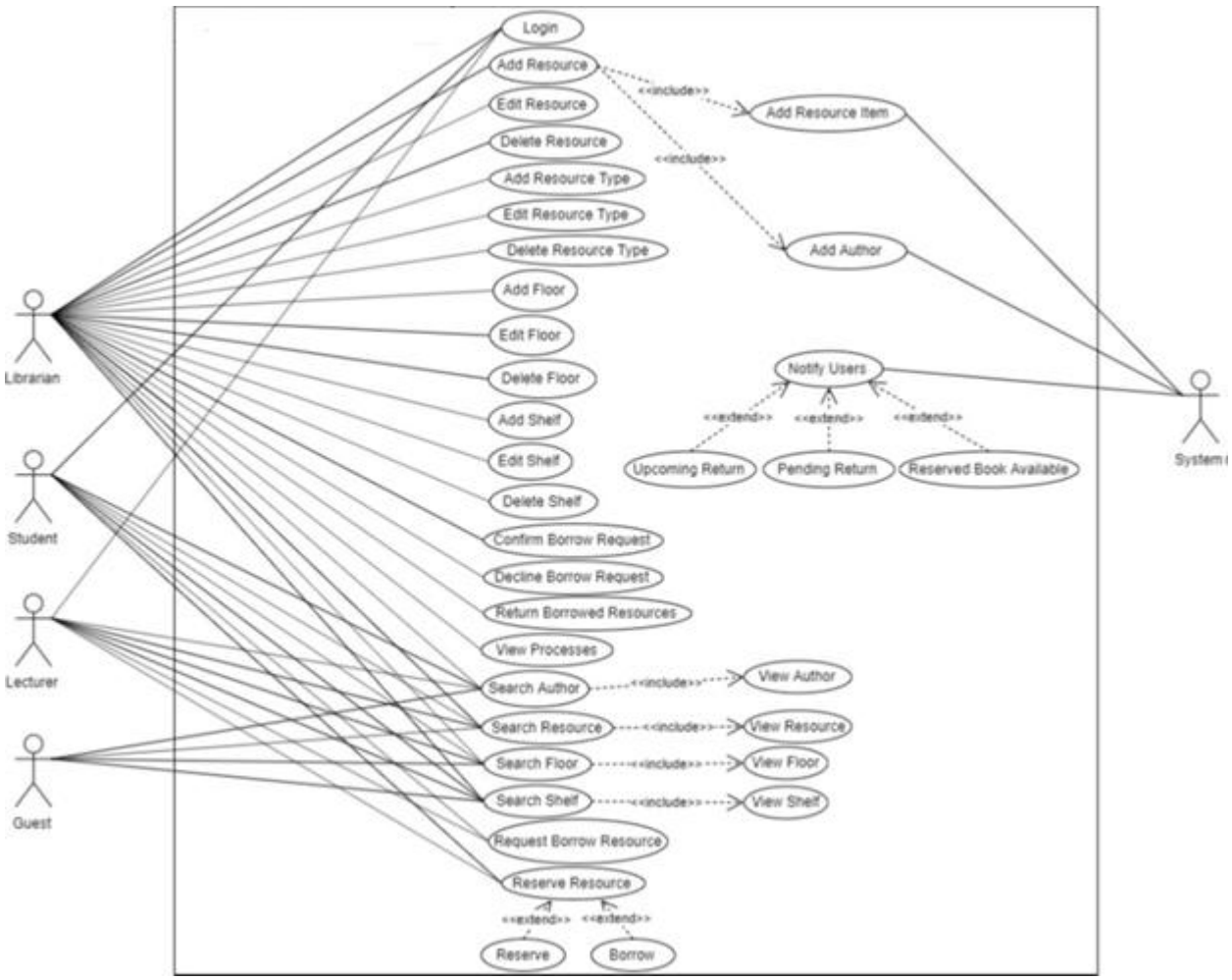

Рисунок 3.1 - Діаграма варіантів використання системи управління бібліотекою.

Сьогодні динамічні веб-додатки, засновані на моделі клієнт-сервер, які працюють з відображенням та управлінням інформацією, складаються з двох частин, які можна розглядати як два різних додатки. Ці додатки називаються фронтенд або клієнтська частина і бекенд або серверна частина. Фронтенд-додатки запускаються в інтернет-браузері користувача і в основному відповідають за відображення графічного інтерфейсу. Але крім того, вони також обробляють запити до серверного додатку для виконання будь-якого процесу або отримання інформації та відображають інформацію, отриману від серверного додатку. І навпаки, бекенд-додаток має зв'язок з базою даних, через яку

надсилаються запити, а потім обробляється і надається інформація, яку отримує фронтенд-додаток.

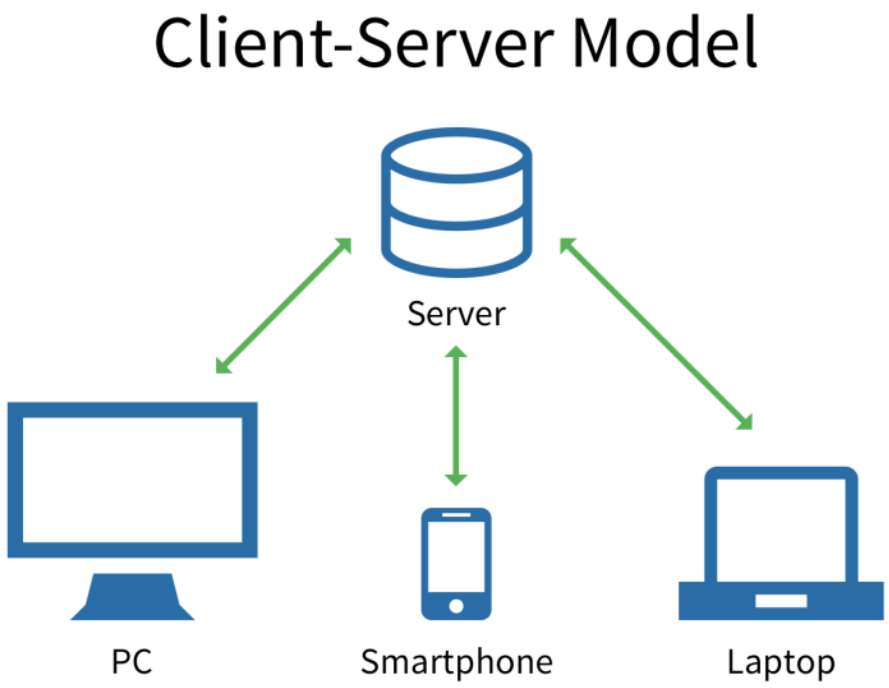

Рисунок 3.2 - Візуалізація клієнт-серверної моделі [15]

Розроблена LMS була побудована з використанням багатьох різних технологій та підходів. Фронтенд-додаток було створено з використанням таких технологій, як HTML, CSS, JavaScript та фреймворк React, із застосуванням адаптивного підходу до дизайну. Тоді як внутрішній додаток був побудований з використанням JavaScript в середовищі NodeJS та фреймворку ExpressJS разом з реляційною системою управління базами даних MySQL.

Загалом, АРМ було побудовано з урахуванням фреймворкового підходу, як для фронтенду, так і для бекенду додатків. У цьому дослідженні під фреймворковим підходом мається на увазі певний спосіб структурування проекту, який має на меті швидшу, простішу та надійнішу розробку. Для досягнення цієї мети обидва додатки структуровані таким чином, що багато процесів і функцій

автоматизовані та використовуються повторно. Деталі цієї структури та підходу будуть розглянуті далі для обох додатків, як згадувалося раніше, у відповідних розділах.

3.1 Бекенд застосунок

У проекті LMS бекенд-додаток складається з технологій і методологій, необхідних для обробки запитів, а потім створення та надання відповіді клієнту. Код, який управляє цим процесом, працює на сервері, який включає логіку, що обробляє запит і відповідно надає відповідь. Оскільки в LMS використовується MySQL, база даних розташована на кластері, а бекенд-додаток має підключення до цієї бази даних MySQL, в якій зберігаються дані LMS.

У бекенд-додатку міститься вся логіка, яка визначає, як відповідати на запит. За допомогою спеціальних функцій ExpressJs створюються маршрути, куди клієнт може надсилати запити. Ці функції, які виконуються на бекенді, називаються проміжним програмним забезпеченням. Як правило, проміжне програмне забезпечення - це додаток, який запускається щоразу, коли сервер отримує запит або повертає відповідь. За допомогою цих функцій проміжного програмного забезпечення можна обробляти багато процесів на серверній стороні. Наведемо кілька прикладів: можна модифікувати об'єкти запиту, робити запити до бази даних або обробляти інформацію, яка буде надіслана у відповідь.

Внутрішній додаток LMS працює на сервері, створеному за допомогою фреймворку ExpressJS на NodeJS. На сервері виконується базова маршрутизація додатку, яка також виконується за допомогою

ExpressJS. База даних LMS створена за допомогою MySQL. Потім встановлюється з'єднання з цією базою даних з бекенд-додатку.

3.1.1 Таблиці бази даних

База даних LMS містить наступні таблиці.

Resource: Ця таблиця представляє елементи в бібліотеці. Ці ресурси можуть бути різних типів, наприклад, книги, журнали, газети, журнали тощо. Таблиця Resource має поля 'id', 'isbn', 'name', 'language', 'publisher', 'publish_date', 'page', 'summary', 'subject', 'cover_image', 'back_cover_image', 'author' та 'shelf'. Поля 'author' та 'shelf' є зовнішніми ключами, що вказують на ідентифікатори автора та полиці.

Resource item: Ця таблиця представляє ресурсні одиниці в бібліотеці. Наприклад, книга може мати дві копії в бібліотеці та цифрову версію. Таблиця Resource item: має поля "id", "type" і "resource". Поля "type" і "resource"вказують на тип ресурсу та ідентифікатор ресурсу.

Resource type: Ця таблиця представляє типи ресурсів, як вже згадувалося в таблиці ресурсів, такі як книга, журнал, газета, журнал і т.д. Ця таблиця має поля 'id', 'name' та 'hard_copy', причому поле 'hard_copy' вказує на те, чи є ресурс друкованою або цифровою копією.

Shelf: Ця таблиця представляє полиці в бібліотеці. Вона має поля 'id', 'name' та 'floor', поле 'floor' вказує на ідентифікатор полиці.

Floor: Ця таблиця показує поверхи в бібліотеці. Вона має поля 'id', 'name' і 'number'.

Author: Ця таблиця представляє авторів, які можуть бути збережені в LMS. Вона має поля 'id', ‘full_name’ та date_of_birth'.

Process: Ця таблиця представляє процеси, які можна виконати в LMS. Це може бути позика або резервування книги. Вона має поля 'id',

'date', 'start_date', 'end_date', 'status' і 'type'. Поле "date" - це дата створення процесу, "start_date" - дата початку позичання або резервування, а "end_date" - дата закінчення позичання або резервування. Поле "status" вказує на статус процесу, наприклад, "Затверджено", "Відхилено" або "Повернуто", а поле "type" вказує на ідентифікатор типу процесу.

Process type: Ця таблиця представляє типи процесів в LMS, як нещодавно згадувалося, типи процесів - резерв та позика. Вона має поля "id" та "name".

User process: Ця таблиця представляє процеси, які виконує користувач, і які відбуваються в даний момент. Вона має поля 'id', 'user', 'process' і 'resource', які вказують на відповідні ідентифікатори.

Notification: Ця таблиця відображає сповіщення, які показуються користувачам про прострочені платежі, майбутні платежі або нещодавні позики. Вона має поля 'content', 'title', 'type', 'user' і 'process'. Поле "зміст" містить зміст сповіщення, поле "заголовок" містить заголовок сповіщення, поле "тип" містить тип сповіщення, наприклад, "помилка", "попередження" або "успіх", і використовується для відображення відповідної піктограми сповіщення. Поле "користувач" вказує на ідентифікатор користувача, а поле "процес" вказує на ідентифікатор процесу.

User: Ця таблиця представляє користувачів, які використовують LMS. Вона містить поля 'id', 'name', 'surname', 'username', 'password', 'user_id', 'date_of_birth', 'department' і 'type'. Поле "type" вказує на ідентифікатор ролі користувача.

User role: Ця таблиця представляє ролі користувачів у LMS. Вона має поля "id" та "name".

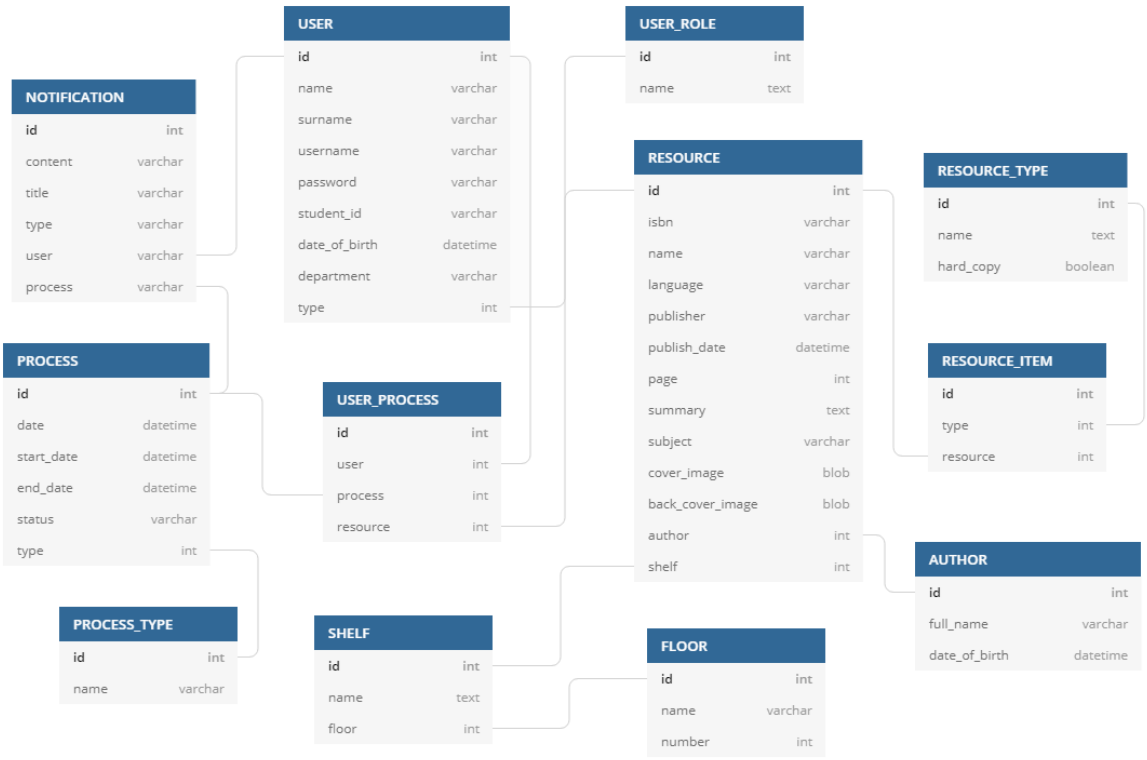

Рисунок 3.3 – Схема бази даних системи управління бібліотекою

3.1.2 RESTful маршрути

Маршрути, створені за допомогою Express, можна відвідати за допомогою визначеного URI. Наприклад, вони структуровані точно так само, як URL. Таким чином, клієнт надсилає запити і отримує відповіді, вказуючи на ці маршрути.

У проекті LMS використання цих маршрутів структуроване, подібно до моделей у програмній архітектурі Model-View-Controller, яка широко відома як MVC. Якщо коротко пояснити, то "модель" в архітектурі MVC відповідає структурі даних, яка є динамічною та

незалежною від інших частин програми, таких як графічний інтерфейс користувача.

Таким чином, кожна таблиця в базі даних має створений для неї маршрут, який включає інформацію, необхідну для обробки операцій створення, читання, оновлення, видалення бази (також відомі як CRUD-операції), а також інші допоміжні операції, такі як отримання даних з таблиць подання, пошук відповідей на запити та автоматичне створення таблиць бази даних.

3.1.3 Базовий маршрут

Маршрути LMS успадковуються від класу, який називається "Базовий маршрут". Базовий маршрут визначає структуру даних для класів-спадкоємців, які називаються "Змодельовані маршрути". Крім того, у базовому маршруті визначено методи, згадані раніше, операції CRUD та інші допоміжні операції. Ці базові операції в базовому маршруті є загальними і використовуються всіма модельованими маршрутами. Визначена структура даних використовується як у методах базових операцій, так і в зазначених методах у змодельованих маршрутах.

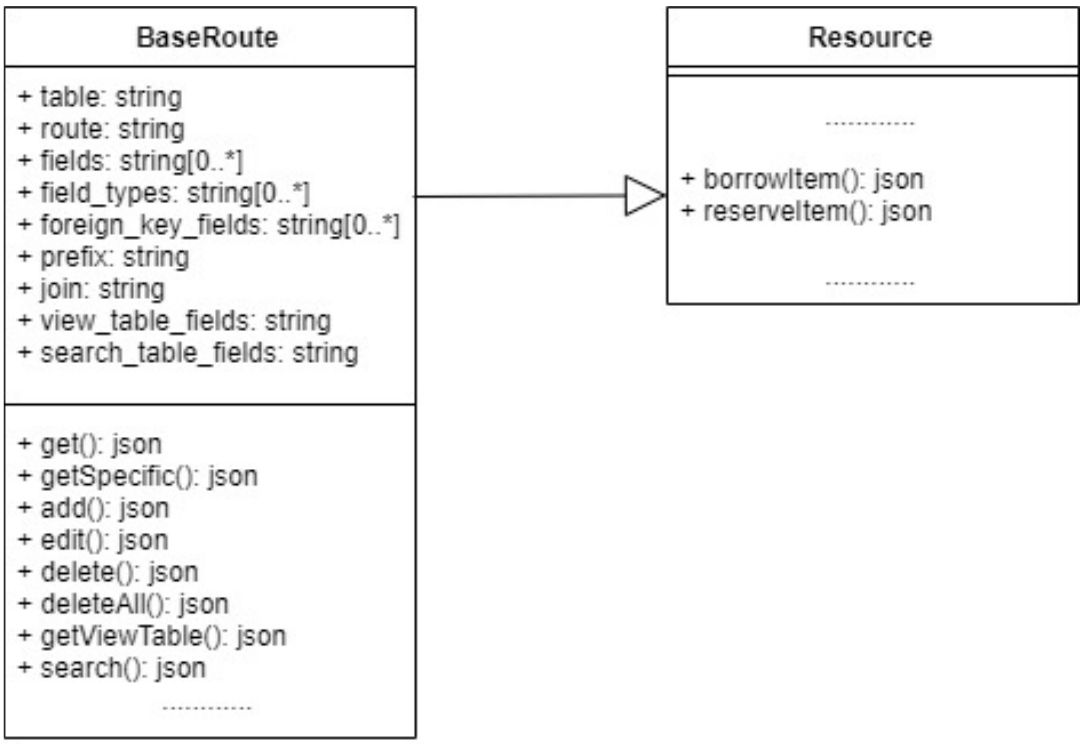

Рисунок 3.4 Приклад співвідношення базового маршруту та змодельованого маршруту

Структура даних Базового маршруту містить такі дані: таблиця, маршрут, поля, типи полів, поля зовнішніх ключів, префікс, з'єднання, поля для перегляду таблиці та поля для пошуку в таблиці.

Table: Поле "table" - це назва таблиці маршруту в базі даних. Це поле використовується операціями CRUD та допоміжними операціями.

Route: Поле "route" - це ім'я, яке присвоюється маршруту і використовується в URI маршруту, як у прикладі http://server/route-name.

Fields: Поле " fields " - це масив рядків, які містять назви полів таблиці бази даних маршруту. Ці поля використовуються операціями CRUD та допоміжними операціями.

Field types: Поле "field types" - це масив рядків, які містять типи полів таблиці бази даних маршруту. Ці поля використовуються при створенні таблиці бази даних.

Foreign key fields: Поле " foreign key fields " - це масив рядків, що містить назви полів зовнішнього ключа, які є в таблиці бази даних маршруту. Ці поля використовуються під час пошуку елементів, коли вони додаються до секції SELECT SQL-запиту на вибірку. Таким чином, зовнішні ключі враховуються під час пошуку елементів, створюючи більш повні результати пошуку.

Prefix: Як частина структури, поля таблиці бази даних починаються з префікса. Таким чином, поле "prefix" означає префікс полів таблиці бази даних маршруту. Це поле використовується при отриманні конкретного значення, де ідентифікатор поля отримується як 'prefix_ID', подібно до 'USER_ID', як приклад.

Join: Поле 'join' - це оператор приєднання, який можна передати до SQL-запиту. Це поле використовується допоміжними операціями, такими як пошук і отримання таблиць представлення.

View table fields: Поле " view table fields " - це рядок, який містить назви полів, що будуть повернуті у відповіді таблиці перегляду. За допомогою цього поля можна налаштувати, які поля буде показано у таблиці подання.

Search table fields: Поле "Search table fields" - це рядок, що містить назви полів, які будуть використані у відповіді на пошук. За допомогою цього поля можна налаштувати, які поля будуть повернуті у відповіді на пошук.

3.1.3 Змодельовані маршрути

Змодельовані маршрути, які успадковують Базовий маршрут, включають структуру даних, методи роботи CRUD і допоміжні методи, які успадковані від Базового маршруту. Таким чином, модельний

маршрут може бути створений для представлення будь-якої таблиці в базі даних. Потім можна виконати будь-яку операцію CRUD або допоміжну операцію. Крім базових операцій, будь-який необхідний специфічний метод записується у власному класі змодельованого маршруту. Таким чином, дотримуючись цієї загальної структури, фаза розробки може бути завершена швидше, ефективніше і краще організована.

3.1.4 Прикладний програмний інтерфейс (API)

"Прикладний програмний інтерфейс " (API) визначається як група методів і процедур, що забезпечує передачу інформації клієнтському додатку від серверного додатку. Таким чином, в LMS всі методи, які визначені як в базовому, так і в модельованому маршрутах, представляють собою API в цілому. Кожен метод представлений у вигляді певної структури, до якої можна отримати доступ з клієнтського додатку і надсилати запити або отримувати відповіді. Ці специфічні адресні структури вказуються у вигляді http://server/route-name/method. Так, в якості прикладу можна навести маршрут http://server/BaseRoute/USER/getAll, який поверне у відповідь всі дані з таблиці 'USER'. Будь-який інший специфічний метод, який відрізняється від базових операцій, може бути досягнутий, наприклад, http://server/USER/getAllUserDepartments, який знаходиться з адреси модельованого маршруту, а не базового маршруту. Таким чином, колекція цих адрес представляє API LMS.

3.1.5 Аутеніфікація

В LMS є процеси з обмеженим доступом, для виконання яких необхідно пройти автентифікацію. Наприклад, позичити книгу, для чого користувач повинен увійти в систему. Або перегляд усіх заявок на видачу/резервування, для якого користувач має увійти в систему як користувач бібліотекар. Отже, вхід користувача в систему і перевірка автентичності ролі користувача обробляються у внутрішньому застосунку. Для виконання цих операцій автентифікації створюються і використовуються відповідно проміжні програми. Для будь-якої автентифікованої операції, подібної до наведеного вище прикладу, використовуються ці проміжні програмні засоби автентифікації. Ці проміжні програми автентифікації визначаються наступним чином:

Вхід в систему: Перевіряється, чи існує користувач у базі даних, потім перевіряється, чи збігається пароль. Якщо користувач існує і пароль правильний, для нього створюється токен, який повертається клієнту. Цей токен генерується з ідентифікатора користувача та спеціально визначеного секрету, що означає, що кожен токен є унікальним для кожного входу.

Підтвердити токен: Будь-яка операція, що вимагає від користувача входу в систему, вимагає надання токену, який має бути згенерований при вході користувача. Цей токен має бути підтверджений, щоб перейти до операції з обмеженим входом. Якщо маркер не надано або не підтверджено, операція не виконується, а клієнту надсилається відповідне повідомлення.

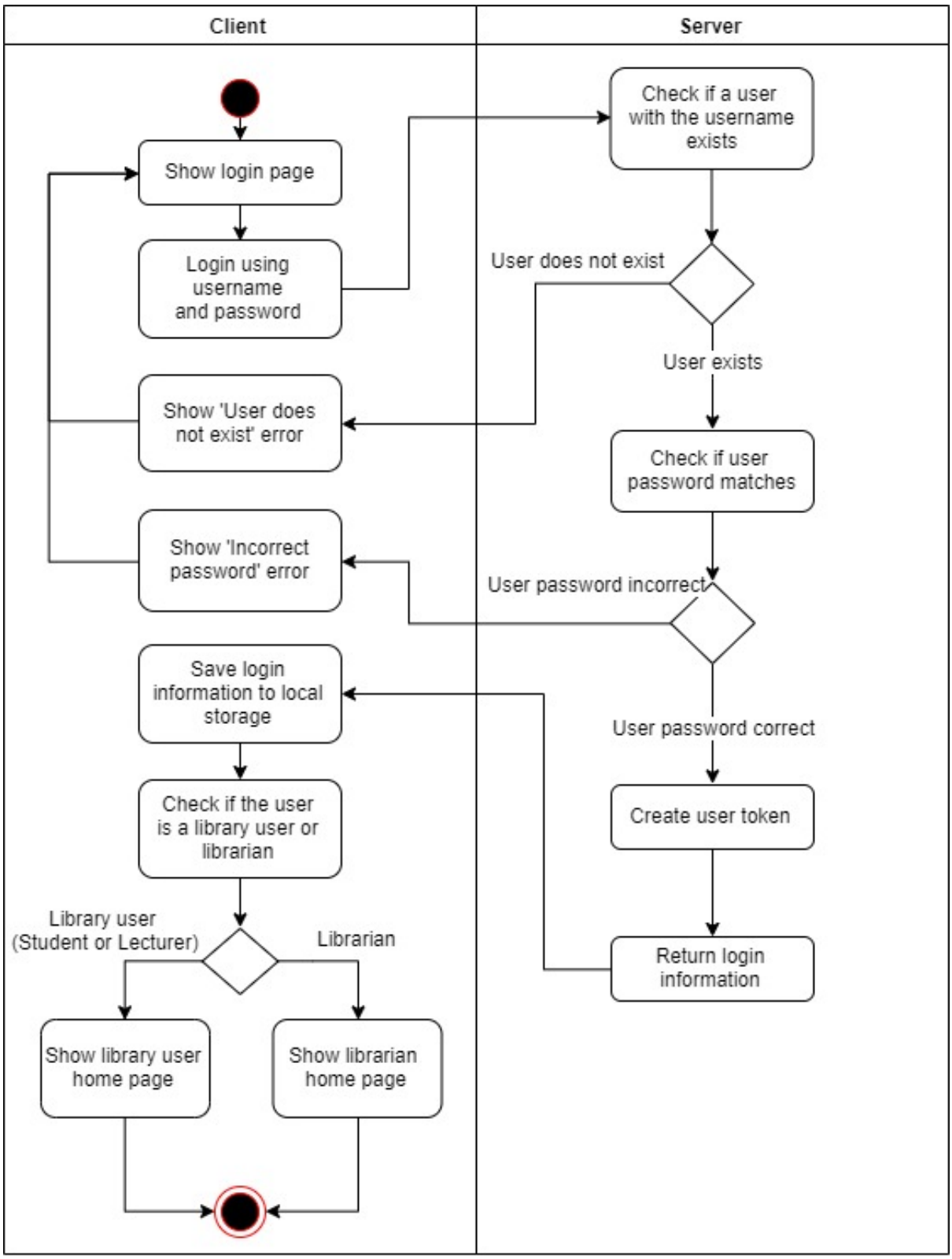

Рисунок 3.4 – Діаграма діяльності для входу в систему

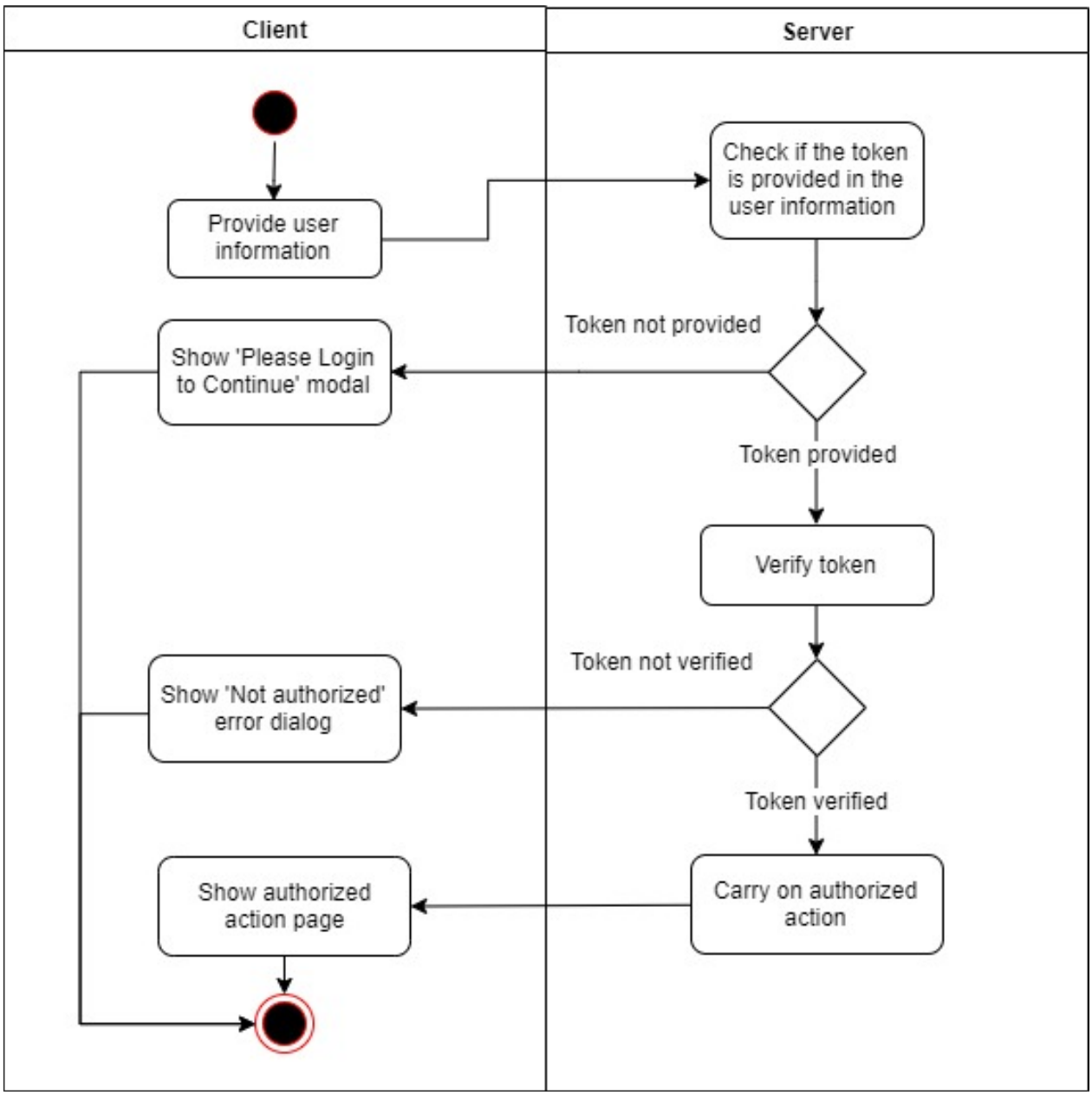

Рисунок 3.5 – Діаграма діяльності перевірки токену

Є студентом або викладачем: Роль користувача перевіряється в базі даних, якщо роль користувача не є студентом або викладачем, повертається відповідна відповідь і операція не виконується. Але, якщо роль користувача є студентом або викладачем, операція виконується.

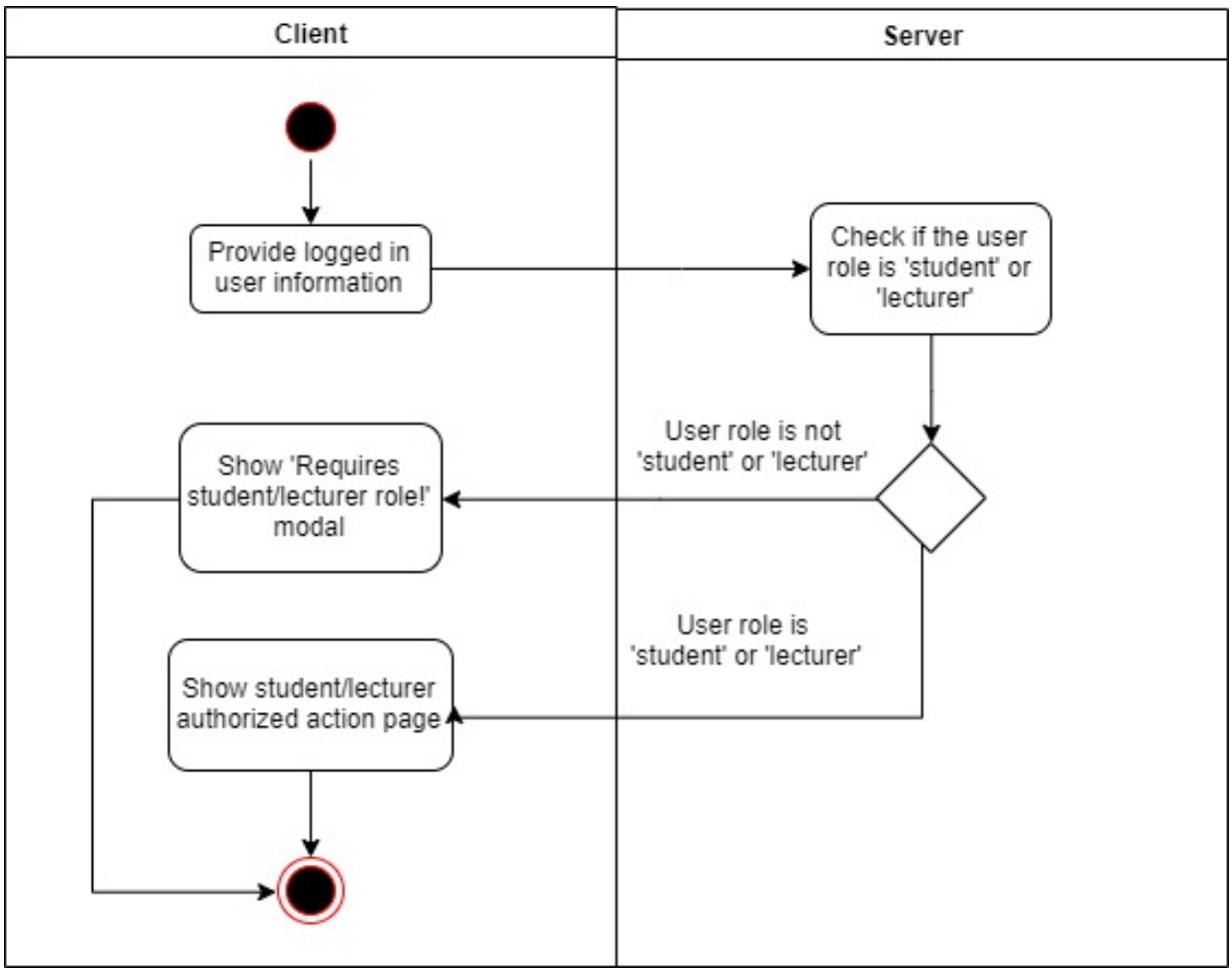

Рисунок 3.6 – Діаграма діяльності перевірки ролей користувачів бібліотеки

Чи є бібліотекарем: Так само, як і при автентифікації студента/викладача, роль користувача перевіряється в базі даних. Якщо роль користувача не є бібліотекарем, повертається відповідна відповідь, і операція не виконується. Але, якщо роль користувача - бібліотекар, операція продовжується.

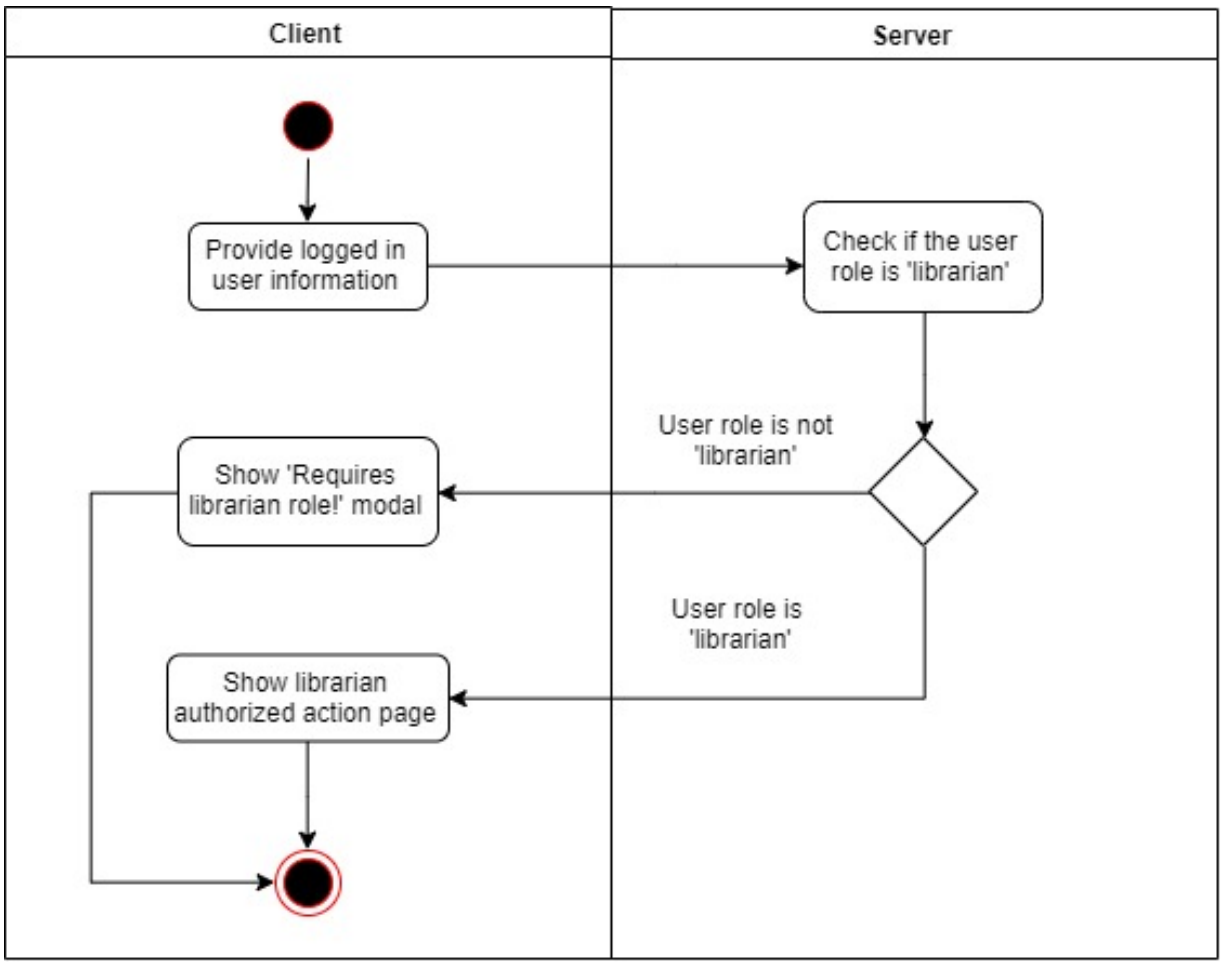

Рисунок 3.7 – Діаграма діяльності перевірки ролі бібліотекаря

В підсумку, при виконанні автентифікованих операцій, ці дії супроводжуються використанням відповідних проміжних програм, залежно від того, яка автентифікація потрібна. Наприклад, при отриманні запиту на запозичення ці дії виконуються в наступному порядку: "перевірити токен, перевірити, чи є користувач студентом або викладачем, виконати операцію запозичення".

3.1.6 Безпека

Для будь-якого веб-додатку безпека є важливим аспектом. Якщо додаток не захищений, він може бути вразливим і відкритим для атак. Розділ "Автентифікація", про який ми нещодавно згадували, забезпечує базовий рівень безпеки, оскільки користувачі повинні бути автентифіковані щоразу, коли вони виконують операції, які є конфіденційними і повинні бути автентифіковані.

Крім того, зловмисники найчастіше атакують бекенд-додатки, особливо HTTP-заголовки. Причиною цього є потенційна вразливість HTTP-заголовків, оскільки через них може відбуватися витік конфіденційної інформації. Ця конфіденційна інформація може бути використана зловмисником для атак на веб-додаток, таких як міжсайтовий скриптинг (також відомий як XSS) та інших атак, які називаються "перехопленням кліків" або "винюхуванням". Щоб захистити додаток від таких атак, у бекенд-додатку LMS використовується модуль під назвою "helmet" модуля NodeJS, який відповідає за безпеку HTTP-заголовків. Helmet забезпечує багато заходів безпеки, ось кілька найважливіших з них:

- X-Frame-Options: Використовується для запобігання атаці "перехоплення кліків".
- X-Content-Type-Options: Використовуються для запобігання атакам типу "sniffing".
- X-XSS-захист: Використовується для додаткового захисту від XSS-атак.

Ще одним загальним заходом безпеки в LMS є шифрування пароля користувача. Шифрування здійснюється шляхом хешування пароля bcrypt за допомогою модуля NodeJS 'bcrypt'. Bcrypt - це алгоритм хешування паролів, заснований на шифрі Blowfish . У бекенд-додатку

цей модуль використовується для хешування пароля перед збереженням його в базі даних. Таким чином, пароль користувача не зберігається в базі даних у відкритому вигляді. При вході в систему модуль bcrypt знову використовується для порівняння та перевірки відповідності паролю користувача.

Крім того, MySQL має функції резервного копіювання та відновлення. Однією з них є експорт бази даних до файлу SQL. Таким чином, можна створити резервну копію бази даних LMS, експортувавши її у файл, який можна легко відновити, імпортувавши експортований файл. Процес експорту також можна автоматизувати за допомогою різних інструментів. Також для автоматизації імпорту резервної копії можна використовувати власну команду MySQL "mysqldump", а також утиліту Linux "cron", яка є утилітою планування завдань за часом, для імпорту резервної копії файлу з певними інтервалами, наприклад, кожні 24 години .

3.2 Фронтенд-додаток

У проекті LMS фронтенд-додаток складається з технологій і методологій, необхідних для побудови графічного інтерфейсу, який призначений для відображення інформації, що надається з бекенд-додатку, а також для подання форм для додавання і редагування. Код, який обробляє ці процеси, виконується на стороні клієнта, яким є інтернет-браузер користувача. Загалом, React використовується для маршрутизації клієнтської частини та рендерингу сторінок, написаних за допомогою JSX, а запити до API обробляються за допомогою інтерфейсу JavaScript fetch. Таким чином, усі сторінки, якими користувач пересувається та взаємодіє з LMS, побудовані на React .

Подібно до бекенд-додатку, фронтенд-додаток також структурований таким чином, щоб використовувати моделі. Структура, що використовує моделі, призначена для обробки запитів на виконання базових CRUD-операцій до API, а також для автоматичного створення форм і перегляду таблиць.

3.2.1 Моделі представлення

Моделі інтерфейсу, які називаються "Моделі представлення", успадковуються від "Базової моделі представлення". Для того, щоб використовувати структуру моделі, моделі створюються для кожного модельованого маршруту у внутрішній програмі. Таким чином, URI, створений у внутрішньому додатку, може бути запрошений з API. У базовій моделі представлення визначена певна структура даних. За допомогою цієї структури даних можна відповідно створювати запити до API.

У базовій моделі представлення створюються базові методи, які можуть бути використані для генерації запитів до CRUD-операцій. Ці запити генеруються з використанням специфічних полів даних, визначених для кожної моделі представлення. Таким чином, з будь-якою створеною моделлю представлення можна швидко і легко виконати CRUD-операції. Окрім базової моделі представлення та базових операцій, у будь-якій моделі представлення можуть бути визначені більш специфічні операції.

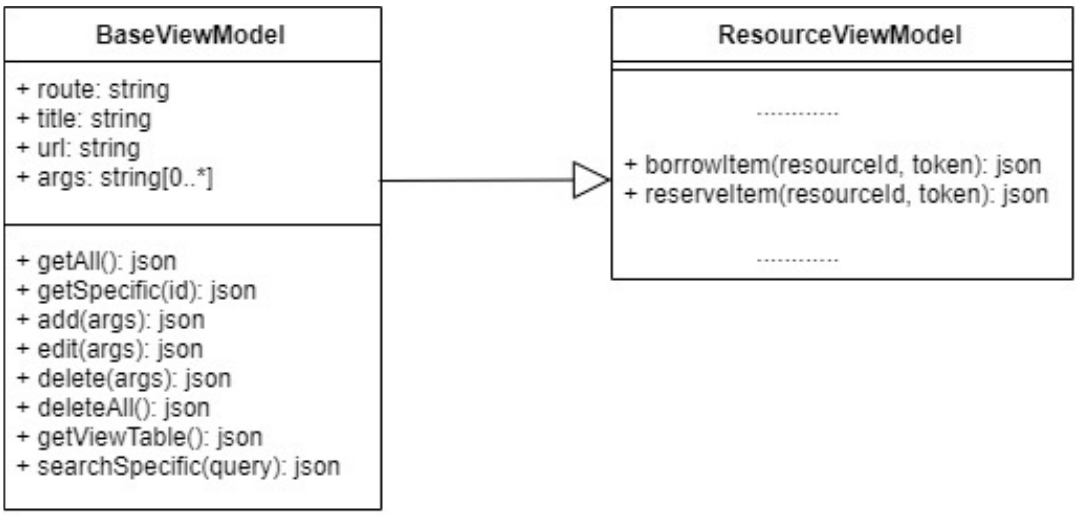

Рисунок 3.8 - Приклад базової моделі представлення та зв’язку моделі представлення

На сторінці, де потрібно виконати дію, пов'язану з моделлю представлення, наприклад, переглянути таблицю ресурсів у бібліотеці, слід визначити відповідну модель представлення, щоб виконати потрібну дію. Як у нашому прикладі, на сторінці, подібній до "Переглянути ресурси", слід визначити модель представлення ресурсів, де з цієї моделі представлення викликається метод "отримати все", і всі ресурси отримуються з бекенд-додатку. Отримана як JSON-об'єкт, ця інформація встановлюється в стан React і застосовується до графічного інтерфейсу. Подібно до цієї ідеї, будь-які базові операції можуть бути використані для моделей представлення, якими є форми, пошук і таблиці представлення. Окрім базових операцій, будь-яка інша специфічна операція може бути використана після її визначення у відповідній моделі представлення.

Базова модель представлення містить поля даних 'route', 'title', 'url' та 'arguments'. Ці поля можна пояснити наступним чином:

- Route: Це поле є рядком, який представляє маршрут, визначений у внутрішньому застосунку. Це поле використовується при створенні URL-адреси, яка буде запитуватися.

- Title: Це поле являє собою рядок, який буде використовуватися для заголовків форм, результатів пошуку та таблиць перегляду
- URL: Це поле являє собою рядок, який представляє URL-адресу і створюється шляхом об'єднання імені сервера з маршрутом.
- Arguments: Це поле є масивом об'єктів JavaScript, який представляє кожне поле змодельованого маршруту в моделі подання. Ці "аргументи" використовуються при автоматичному створенні форм і таблиць подання. Кожен об'єкт JavaScript містить назву, мітку і значення поля, а також інші допоміжні поля. Ці поля не є обов'язковими і використовуються за потреби, наприклад, у таких операціях, як відсилання та перегляд таблиць.

3.2.2 Форми

Форми є фундаментальною функціональністю багатьох веб-додатків. Так само, як і в LMS, форми є базовим функціоналом у фронтенд-додатку. Таким чином, всі операції створення, перевірки та надсилання форми виконуються у фронтенд-додатку.

Форми генеруються шляхом перевірки кожного аргументу, визначеного у моделі представлення. Для кожного аргументу перевіряються відповідні допоміжні поля, щоб згенерувати та перевірити форму. "Приховане" поле перевіряється, щоб визначити, чи включено аргумент у форму як поле, "обов'язкове" поле перевіряється, щоб визначити, чи дозволено вводити поле порожнім. Крім того, використовується логічне поле з назвою "число", щоб встановити для поля тільки числові значення, і поле з назвою "цифра", щоб встановити максимальну цифру. Коли форма надсилається, викликається базова

операція "додати" для відповідної моделі представлення з відповідними полями.

3.2.3 Пошук

Подібно до пошукової системи, користувач може здійснювати пошук у бібліотеці за допомогою функції пошуку в LMS. Цей пошук зазвичай здійснюється за ресурсом, але також можна шукати авторів, полиці та поверхи. Таким чином, користувач може переглянути інформацію про будь-який ресурс, автора, полицю та поверх за допомогою пошуку. Від бекенд-додатку отримується конкретна відповідь, що містить інформацію про кожен результат пошуку. Таким чином, подання будуються конкретно відповідно до цього запиту. Функціонал пошуку є базовою операцією у фронтенд-додатку LMS, а це означає, що його можна використовувати для будь-якої моделі подання.

3.2.4 Таблиці представлення

Таблиці представлення - це ефективний спосіб відображення інформації для користувача. Це ще одна базова функціональність інтерфейсу LMS, яка реалізується за допомогою базової структури та таблиць Material-UI. Таким чином, таблиці представлення можуть бути створені для будь-якого елемента в LMS, наприклад, ресурсів, поверхів, полиць тощо.

Запитуючи дані таблиці представлення з API для моделі подання, дані отримуються відповідно до полів таблиці подання Modelled Route з бекенд-додатку. На основі отриманої інформації таблиця

представлення генерується автоматично за допомогою табличного компонента Material-UI. Ці таблиці представлення також мають такі функції, як сортування стовпців, збільшення пагінації, збільшення кількості рядків для відображення, а також пошук і фільтрація будь-якого рядка за допомогою пошуку в реальному часі

3.2.5 Аутентифікація

Як згадувалося раніше, в АРМ є завдання, які вимагають автентифікації. Хоча деякі завдання, такі як пошук ресурсу, перевірка інформації про ресурс тощо, можна виконати як гість, інші завдання, такі як запозичення або резервування ресурсу, вимагають, щоб користувач був студентом або викладачем.

Аутентифікація у зовнішньому додатку обробляється, як описано вище; коли користувач успішно входить в систему, внутрішній додаток відповідає інформацією про користувача, яка містить ідентифікатор користувача, ім'я, прізвище, студентський квиток, ім'я користувача, дату народження, відділ, роль і токен доступу.

Інформація, отримана від внутрішньої програми, зберігається локально. Коли виконується дія, що вимагає автентифікації, токен доступу також додається до запиту як параметр, що підлягає перевірці у внутрішньому додатку. Наприклад, інформація про користувача перевіряється перед тим, як взяти книгу, і якщо користувач не увійшов до системи, йому/їй пропонується увійти до свого облікового запису і спробувати взяти книгу ще раз.

Будь-яка сторінка, для доступу до якої потрібна автентифікація, також перевіряється системою, чи користувач увійшов з локального сховища, якщо ні, то користувач буде перенаправлений на іншу сторінку, наприклад, на сторінку входу в систему. Наприклад, всі

сторінки користувача бібліотекаря можна побачити, тільки якщо користувач увійшов до системи як бібліотекар, в іншому випадку він буде перенаправлений на сторінку входу в систему. З іншого боку, оскільки студенти і викладачі можуть переглядати деякі сторінки як гості, якщо гість намагається переглянути сторінку, яка вимагає автентифікації, наприклад, сторінку своїх процесів, він буде перенаправлений на попередню сторінку, на якій він перебував.

3.2.6 Тестування

В інженерії програмного забезпечення тестування проводиться для того, щоб зібрати або надати інформацію про якість софтверної системи. Існує кілька способів і методів тестування програмного забезпечення. Ці методи, як правило, спрямовані як на пошук помилок або помилок в системі, так і на вимірювання зручності використання системи. Коротко кажучи, це означає запуск частини програмного забезпечення, щоб переконатися, що воно відповідає певним факторам, таким як відповідність вимогам, правильна реакція на вхідні дані, продуктивність і належна зручність використання.

Деякі з найпоширеніших методів тестування програмного забезпечення називаються тестуванням "чорної скриньки" та тестуванням "білої скриньки". Тестування "чорного ящика" проводиться без доступу до вихідного коду, тобто тестувальник знає, що повинно робити програмне забезпечення, але не має жодних технічних знань про нього. З іншого боку, тестування білого ящика проводиться тестувальниками, які знають внутрішню роботу програмного забезпечення, таким чином проводячи тести на рівні модулів. Як комбінація цих двох методів, існує ще один метод, який називається тестуванням сірої скриньки (gray-box testing). При тестуванні сірим

ящиком тест проводить тестувальник, який володіє знаннями про внутрішню роботу програмного забезпечення, визначаючи тестові кейси на основі цих знань. Це дозволяє тестувальнику виконувати тестові кейси, подібні до тестування чорного ящика, але більш ретельно перевіряти якості програмного забезпечення.

АРМ розроблявся ітеративно, де на кожній ітерації ставилося завдання додати нову функцію в систему. Оскільки розробка проєкту здійснювалась одноосібно, не було можливості провести тестування "чорного ящика". Таким чином, проводилось тестування сірого ящика в кінці кожної ітерації, перш ніж переходити до нової, щоб переконатися, що функції ітерації відповідають вимогам, працюють за призначенням, і немає явних багів або помилок.

Під час тестування за методом сірої скриньки стратегія полягала у визначенні вхідних даних, очікуваних результатів, сценаріїв користувача та розробці комплексних тестових кейсів щодо цих факторів. Потім кожен тестовий кейс був протестований мною, або пройшовши, якщо не було проблем, або не пройшовши, якщо в результаті були виявлені проблеми. Наприкінці, будь-яка проблема, помилка або баг, виявлена в результаті невдалого тестування, була виправлена. Мета цієї стратегії полягала в тому, щоб переконатися у відсутності помилок перед тим, як переходити до нової ітерації розробки, як зазначалося раніше.

3.3 Середовище розробки

У веб-розробці зазвичай використовуються інструменти з гнучкими вимогами. Часто це залежить від того, що розробник вважає за краще використовувати. Для написання JavaScript-коду при розробці веб-додатків потрібен лише текстовий редактор, оскільки він не

потребує компіляції. З цієї причини можна використовувати навіть легкий, простий текстовий редактор, однак сьогодні існує багато інтегрованих середовищ розробки та редакторів, таких як VSCode, Sublime Text або WebStorm, кожен з яких має різні можливості, такі як комбінації клавіш, виявлення помилок, форматування коду і так далі. Але, як уже згадувалося, це повністю залежить від того, що розробник вважає за краще використовувати в якості редактора коду.

Окрім редактора, для полегшення та підвищення ефективності веб-розробки можна використовувати й інші інструменти. Одним з найважливіших таких інструментів є інструмент контролю версій. Інструмент контролю версій - це інструмент, який використовується для управління змінами в розробці, наприклад, Git.

Інструменти, такі як візуальний конструктор баз даних або клієнт API - це інші інструменти, які можна використовувати під час веб-розробки. Інструмент візуального проектування баз даних - це інструмент, який легко використовувати для створення, зміни, запитів до баз даних та оновлення даних за допомогою графічного інтерфейсу. Тоді як клієнт API - це інструмент, який використовується для тестування викликів API, шляхом надсилання викликів, отримання та представлення відповіді. Ці інструменти роблять надсилання складних або повторюваних тестових викликів API простішими та ефективнішими, оскільки запити можна зберігати або редагувати.

При розробці LMS використовувався Visual Studio Code IDE, Git з GitHub для контролю версій, інструмент проектування баз даних MySQL Workbench для MySQL, Postman для тестування бекенд API, на операційній системі Windows 10.

3.4 Робота з Web-застосунком

Головна сторінка вебзастосунку містить інформаційні розділи, відкриті для перегляду всіма користувачами (рис. 3.9)

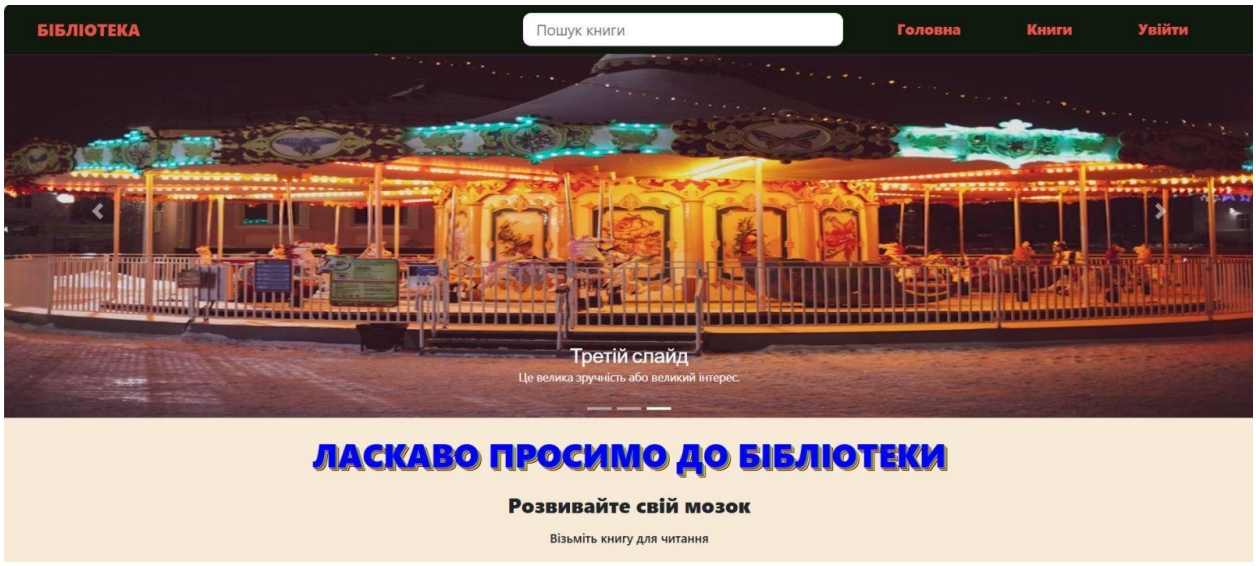

Рисунок 3.9 – Головна сторінка застосунку.

Користувач може подивитися список книг, натиснувши на пункт меню «Книги»(рис. 3.10)

Рисунок 3.10 – Список книг.

Для входу в систему потрібно пройти авторизацію (рис. 3.11)

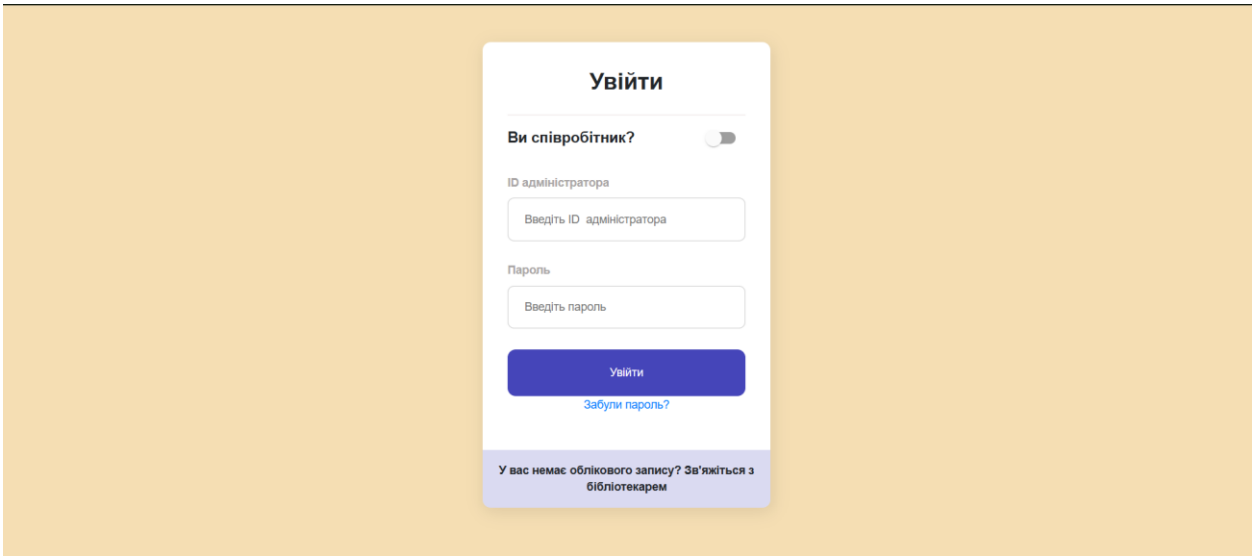

Рисунок 3.11 – Форма авторизації

Якщо зареєструватися як співробітник, то можна подивитися список книг, які знаходяться у читачів (рис. 3.12), подивитися статистику (рис.3.13) та виконувати інші адміністративні функції.

Рисунок 3.12 – Список книг, які знаходяться у читачів

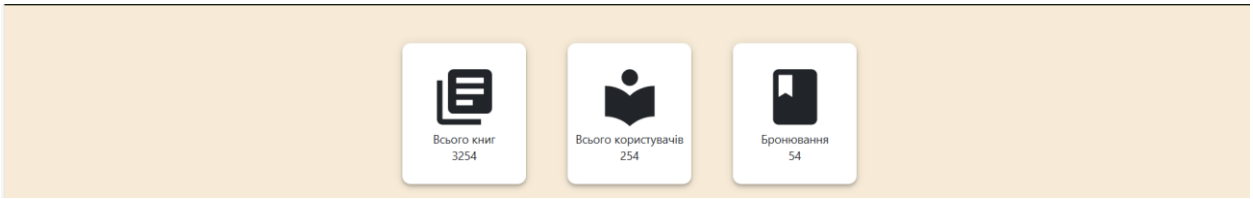

Рисунок 3.13 – Статистичні дані по бібліотеці

РОЗДІЛ 4. ЕКОНОМІЧНА ДОЦІЛЬНІСТЬ РОЗРОБКИ ПРОГРАМНОГО ПРОДУКТУ

4.1 Вартісний аналіз розробки програмного продукту

4.1.1 Розрахунок повної собівартості програмного продукту

Фактична або повна собівартість програмного продукту (ПП) визначається в процесі проведення калькуляції собівартості та є сумою виробничої собівартості, адміністративних витрат та витрат на збут.

Виробнича собівартість розраховується за допомогою визначення поточних витрат на його розробку (або функціонально–необхідних витрат на створення ПП – $C_{Свир.}$) і визначається за формулою:

$$\mathrm{C_{вир}} = \mathrm{C_{ЗПр}} + \mathrm{C_{ЕСВ}} + \mathrm{C_{М.ч.}} + \mathrm{C_{З-в.Н.}} + \mathrm{C_{М}}, \qquad (4.1)$$

де $\mathrm{C_{ЗПр}}$ – заробітна плата розробників ПП, грн;

$\mathrm{C_{ЕСВ}}$ – відрахування на соціальне страхування, грн;

$\mathrm{C_{М.ч.}}$ – вартість машинного часу, необхідного для розробки та налаштування ПП, грн;

$\mathrm{C_{З-в.Н.}}$ – загальновиробничі (накладні) витрати (витрати на оплату праці управлінського персоналу, оплату службових відряджень, консультаційно–інформаційні витрати, ремонт і технічне обслуговування інших основних фондів, окрім ПК, оренда приміщення тощо);

$\mathrm{C_{М}}$ – вартість матеріалів, комплектуючих, грн.

Отже, розрахуємо повну собівартості програмного продукту для з’ясування раціональності розробки.

4.1.2 Калькуляція собівартості програмного продукту

До заробітної плати розробників ПП ($С_{ЗПр}$) належать витрати на виплату основної та додаткової зарплати виконавців, обчислені згідно із системою оплати праці, прийнятими в організації, включаючи будь–які види матеріальних та грошових доплат. Визначається за формулою:

$$С_{ЗПр.} = С_{ЗПосн.} + С_{ЗПдод.} \quad (4.2)$$

Основна заробітна плата розробників ПП:

$$С_{ЗПосн.} = С_{ЗПден.} \times Т_{заг.}, \quad (4.4)$$

де $С_{ЗПден.}$– денна зарплата програміста, грн;

$Т_{заг.}$ – загальна трудомісткість розробки ПП (комп'ютерної системи), людино–дні.

Денну заробітну плату визначають, виходячи з місячних окладів:

$$С_{ЗПден.} = \frac{С_{Ок.} \times 12}{Ф_{р.ч.}}, \quad (4.4)$$

де $С_{Ок.}$ – місячний оклад розробника ПП, грн;

$Ф_{р.ч.}$ – річний фонд робочого часу, днів; (трудовим законодавством встановлено на 2024 р.: $Ф_{р.ч.} = 262$).

По Україні в середньому місячний оклад розробника з меншим ніж рік досвідом складає 500$. На травень 2025 року це приблизно 21000 грн.

Розрахуємо денну заробітну плату:

$$\text{С}_{\text{ЗПден.}} = \frac{21000 * 12}{262} = \frac{252000}{262} = 961{,}8 \text{ грн}$$

Отримали розмір денної заробітної плати у розмірі $961{,}8$ грн.

Загальну трудомісткість можна визначити за формулою:

$$\text{Т}_{\text{заг.}} = N_{\text{час}} \times k_{\text{скл.}} \times k_{\text{м.}} \times k_{\text{станд.}} \times k_{\text{станд.ПП}}, \qquad (4.5)$$

де $\text{Т}_{\text{заг.}}$– загальна трудомісткість, людино–дні;

$N_{\text{час}}$ – норма часу, людино–дні (у середньому: 61 – 83 людино–дні);

$k_{\text{скл.}}$ – поправковий коефіцієнт складності контролю вхідної та вихідної інформації (складає 1,08);

$k_{\text{м.}}$ – поправковий коефіцієнт використання мови певного рівня складності (мова високого рівня дорівнює 1);

$k_{\text{станд.}}$ – коефіцієнт використання стандартних програм (приблизно складає 0,7);

$k_{\text{станд.ПП}}$ – коефіцієнт розробки стандартного ПП (1,2 – 1,6).

Розробляючи ПП, використовуються стандартні модулі і/або пакети прикладних програм, чи типові програми, тому норму часу коригують за допомогою коефіцієнта $k_{\text{станд.}} = 0.6 - 0.8$.

Норму часу візьмемо 70 людино-днів, коефіцієнт розробки стандартного ПП – 1.2 та проведемо розрахунки:

$$\text{Т}_{\text{заг.}} = 70 * 1{,}08 * 1 * 0{,}7 * 1{,}2 = 63{,}5 \text{ л/д}$$

Після отримання загальної трудомісткості розробки та денної заробітної плати можемо розрахувати основну заробітну плату за формулою 4.4:

$$С_{ЗПосн.} = 961{,}8 * 63{,}5 = 61074 \text{ грн}$$

Додаткова заробітна плата (премії, одноразові заохочення тощо) розраховується згідно з нормативом, який установлює підприємство і який складає 10 – 40 % від основної зарплати. Витрати на додаткову заробітну плату визначаються за формулою:

$$С_{ЗПдод.} = k_{ЗПдод.} \times С_{ЗПосн.}, \quad (4.6)$$

де $k_{ЗПдод.}$ – нормативний коефіцієнт додаткової заробітної плати ($k_{ЗПдод.} = 0.1 \ldots 0.4$);

$С_{ЗПосн.}$– витрати на основну заробітну плату, грн.

Розрахуємо додаткову заробітну плату при нормативному коефіцієнті 20%:

$$С_{ЗПдод.} = 0{,}2 * 61074 = 12214{,}8 \text{ грн}$$

До витрат на сплату єдиного соціального внеску (СЄСВ) належать витрати, що здійснюються у порядку та розмірах, передбачених чинним законодавством України (тобто це нарахування від суми основної та додаткової зарплати, які беруться з підприємства; згідно з нормативами, діючими на 01.01.2012 р., і які складають 36,76%, на 2025 рік цей внесок становить 22%).

Витрати на сплату єдиного соціального внеску визначаються за формулою 4.7.

$$\text{С}_{\text{ЄСВ}} = k_{\text{ЄСВ}} \times \left(\text{С}_{\text{ЗПосн.}} + \text{С}_{\text{ЗПдод.}}\right), \quad (4.7)$$

де $k_{\text{ЄСВ}}$ – коефіцієнт витрат на сплату ЄСВ ($k_{\text{ЄСВ}} = 0.22$);

$$\text{С}_{\text{ЄСВ}} = 0{,}22 * (61074 + 12214{,}8) = 16123{,}5 \text{ грн}$$

Вартість машинного часу, необхідного для розробки та налаштування ПП ($C_{М.ч.}$) визначається:

$$\text{С}_{\text{М.ч.}} = \text{С}_{\text{м.–г.}} \times t_{\text{м.}}, \quad (4.8)$$

де $t_{\text{м.}}$– тривалість машинного часу (сума часу машинних і машинно–ручних операцій), необхідного для розробки ПП, грн.

$$t_{\text{м.}} = \text{Р}_{\text{комп.}} \times 0{,}15 + \text{Т}_{\text{заг.}} \times \text{С}_{\text{електроен}} \quad (4.9)$$

$$t_{\text{м.}} = 19000 * 0{,}15 + 63{,}5 * 4{,}32 = 2850 + 274{,}3 = 3124{,}3 \text{ грн}$$

Собівартість однієї машино–години роботи ПК:

$$\text{С}_{\text{м.–г.}} = \frac{\text{С}_{\text{р.екс.}}}{\text{Т}_{\text{еф.}}}, \quad (4.10)$$

де $\text{С}_{\text{р.екс.}}$– річні експлуатаційні поточні витрати на обслуговування ПК (грн), які охоплюють:

– основну та додаткову ЗП спеціаліста (інженера–електронника), який обслуговує машину з урахуванням його зайнятості на обслуговування ПК, грн;

– витрати на сплату ЄСВ ($\text{С}_{\text{ЄСВ}}$ = 36,76 % від фонду оплати праці, 22% на 2025 рік), грн;

– амортизаційні відрахування (Ам), які розраховуються із залишкової вартості ПК і норми амортизаційних відрахувань (згідно з нормами складає 15 % від балансової вартості ПК, а=0.15), грн;

– витрати на ремонт та профілактику ПК ($С_{рем.}$), грн;

– витрати на оплату електроенергії, які визначаються як добуток тарифу за 1 кВт–г електроенергії на обсяг потужності, що споживається ПК і на ефективний годинний фонд часу роботи ПК за рік (тариф за 1 кВт–г електроенергії в березні 2025 р.: 4.32 грн), грн.

Основна ЗП інженера–електронника:

$$С_{ЗПосн.ел} = (С_{ок.} * n_i)/Т_{заг.,} \quad (4.11)$$

де $C_{Ок.}$ – місячний оклад інженера–електронника, грн;

n_i – кількість одиниць ПК, що використовуються для розробки ПП.

$$С_{ЗПосн.ел} = \frac{13500}{63,5} = 212,6$$

$$С_{ЗПдод.ел} = 0,2 * 212,6 = 42,5$$

$$С_{ЄСВ\ ел.} = 0,22 * (212,6 + 42,5) = 56,1$$

$$А_М = 19000 * 0,15 = 2850\ грн$$

Витрати на ремонт та профілактику:

$$С_{рем.} = (k_{рем.} \times P_{ком.}) \times n_i\ , \quad (4.12)$$

де $k_{рем.}$– коефіцієнт поточного ремонту та обслуговування ПК (залежить від $a_{рем.}$);

$a_{\text{рем.}}$– середньостатистичний норматив витрат на поточний ремонт і обслуговування ПК ($a_{\text{рем.}} = 4\%$);

$k_{\text{рем.}} = 0.04$;

$P_{\text{ком.}}$ – ціна ПК, грн.

Тоді:

$$\text{С}_{\text{рем.}} = (0{,}04 * 19000) * 1 = 760 \text{ грн}$$

Ефективний годинний фонд часу роботи ПК за рік $T_{\text{еф.}}$ визначається, виходячи з календарного річного фонду часу, зменшеного з урахуванням вихідних, святкових днів і добового режиму роботи ПК з урахуванням поточного ремонту, годин:

$$T_{\text{еф.}} = \text{Т}_{\text{рік.}} \times (t_{\text{ном.}} - t_{\text{рем.}}), \qquad (4.13)$$

де $\text{Т}_{\text{рік.}}$– кількість робочих днів за рік,

$t_{\text{ном.}}$– номінальна кількість годин цілодобової роботи обладнання, ($t_{\text{ном.}} = 8$ год.);

$t_{\text{рем.}}$– число годин на добу для поточного ремонту та обслуговування ПК:

$$t_{\text{рем.}} = 0{,}15 * \ t_{\text{ном.}} = 1.2 \text{ год.}$$

$$T_{\text{еф.}} = 262 \times (8 - 1.2) = 1781{,}6$$

$$\text{С}_{\text{електр.}} = 4{,}32 * 0{,}5 * 1781{,}6 = 3833{,}6 \text{ грн.}$$

$$\text{С}_{\text{м.–г.}} = \frac{212{,}6 + 42{,}5 + 56{,}1 + 2850 + 760 + 3833{,}6}{1781{,}6} = 4{,}4 \text{ грн}$$

$$\text{С}_{\text{М.ч.}} = \text{С}_{\text{м.}-\text{г.}} \times t_{\text{м.}} = 4{,}4 * 3124{,}3 = 13746{,}9 \text{ грн}$$

Експлуатаційні витрати поділяють на виробничі й адміністративні витрати.

Загальновиробничі (накладні) витрати ($C_{З-в.Н.}$): диски, картриджі, папір для роздрукування тощо.

Ураховуючи, що собівартість ПП визначається в умовах обмеженої інформації щодо технології розробки, до загальновиробничих витрат можуть належати також витрати на:

– освоєння нової розробки;

– відшкодування зносу спеціальних інструментів і пристроїв цільового призначення;

– утримання та експлуатацію устаткування.

При такому комплексному складі загальновиробничих витрат їх норматив досягає 50 – 150 % від основної З/П. Загальновиробничі (накладні) витрати визначаються за формулою:

$$\text{С}_{\text{з}-\text{в.Н.}} = k_{\text{з}-\text{в.}} \times \text{С}_{\text{ЗПосн.}}, \qquad (4.14)$$

де $k_{\text{з}-\text{в.}}$ – коефіцієнт загальновиробничих (накладних) витрат ($k_{\text{з}-\text{в.}} = 0.5 \div 1.5$). Обрано $k_{\text{з}-\text{в.}} = 0{,}5$:

$$\text{С}_{\text{з}-\text{в.Н.}} = 0{,}5 * 61074 = 30537 \text{ грн}$$

Вартість витратних матеріалів, комплектуючих ($C_{М.}$) рекомендовано взяти в розмірі 4 % від фонду заробітної плати ($k_{М.} = 0{,}04$).

$$\text{С}_{\text{М.}} = 0{,}04 * 61074 = 2443 \text{ грн}$$

Окрім вказаних поточних витрат на розробку ПП, собівартість розробки та реалізації ПП передбачає визначення:

– адміністративних витрат (організаційні витрати, витрати на службові відрядження, страхування, амортизацію, опалення, освітлення, водопостачання, охорону; винагорода за професійні послуги: юридичні, аудиторські; витрати на зв'язок; витрати за послуги банку);

– витрат на збут (на рекламу та дослідження ринку: маркетинґ; витрати на гарантійний ремонт і гарантійне сервісне обслуговування; комісійні витрати; витрати, пов'язані з безпосереднім постачанням: страхування, амортизація, охорона);

– повної (фактичної) собівартості (у грошовому виразі індивідуальні витрати певного розробника ПП у даних умовах).

Повна собівартість формується в процесі щоденного оперативно–технічного та бухгалтерського обліку витрат на розробку ПП, виконання робіт і забезпечення всіма матеріально–технічними, трудовими ресурсами.

Адміністративні витрати. Ці витрати належать до собівартості ПП пропорційно основній З/П і складають 100 – 200 % від основної З/П. Адміністративні витрати визначаються за формулою:

$$\mathrm{C}_{\text{адмін.}} = k_{\text{адмін.}} \times \mathrm{C}_{\text{ЗПосн.}}, \tag{4.15}$$

де $k_{\text{адмін.}}$ – коефіцієнт адміністративних витрат ($k_{\text{адмін.}} = 1 \div 2$). Обрано $k_{\text{адмін.}} = 1$:

$$\mathrm{C}_{\text{адмін.}} = 1 * 61074 = 61074 \text{ грн}$$

Витрати на збут. Витрати за цією статтею складають 2,5 – 5 % від виробничої собівартості. Витрати на збут визначаються за формулою:

$$\text{С}_{\text{збут.}} = k_{\text{збут.}} \times \text{С}_{\text{вир.}}, \quad (4.16)$$

де $k_{\text{збут.}}$ – коефіцієнт витрат на збут ($k_{\text{збут.}} = 0.025 \div 0{,}05$).

$\text{С}_{\text{вир.}}$– виробнича собівартість. Згідно формулі 5.5 вона дорівнює:

$$\text{С}_{\text{вир.}} = 61074 + 12214{,}8 + 16123{,}5 + 13746{,}9 + 30537 + 2443 = 136139{,}2 \text{ грн}$$

Тоді витрати на збут:

$$\text{С}_{\text{збут.}} = 0{,}025 * 136139{,}2 = 3403{,}5 \text{ грн}$$

Повна (фактична) собівартість:

$$\text{С}_{\text{С}_\text{П}} = \text{С}_{\text{вир.}} + \text{С}_{\text{збут.}} + \text{С}_{\text{адмін.}} = 136139{,}2 + 3403{,}5 + 61074 = 200616{,}7 \text{ грн}$$

Результати виконаних розрахунків занесено до таблиці 4.1.

Таблиця 4.1 – Калькуляція собівартості ПП

№ статті	Статті витрат	Сума, грн	Питома вага, %
1	Основна заробітна плата	61074	30,44
2	Додаткова заробітна плата	12214,8	6,09
3	Витрати на сплату єдиного соціального внеску	16123,5	8,04
4	Вартість машинного часу, необхідного для розробки та налаштування ПП	13746,9	6,85
5	Загальновиробничі (накладні) витрати	30537	15,22
6	Вартість витратних матеріалів, комплектуючих	2443	1,22
7	Виробнича собівартість ($C_{C_{вир.}}$)	136139,2	67,86
8	Адміністративні витрати	61074	30,44
9	Витрати на збут	3403,5	1,70
10	Повна (фактична) собівартість ($\text{С}_{\text{С}_\text{П}}$)	200616,7	100,00

4.2 Визначення ціни програмного продукту на основі вартості його розробки

Ціна ПП ($P_{\text{дог.}}$) формується на основі економічно обґрунтованої собівартості його розробки, норми рентабельності, прибутку (певного відсотку торговельної надбавки) та податку на додану вартість (ПДВ):

$$P_{\text{дог.}} = \left(\text{С}_{\text{С}_\text{П}} + \text{Р}_{\text{рентаб.Н}} + m\right) + \text{ПДВ}, \qquad (4.17)$$

або

$$P_{\text{дог.}} = \left(\text{С}_{\text{С}_\text{П}} + \left(1 + \frac{\text{Р}_{\text{рентаб.Н}} + m}{100}\right)\right) \times \text{ПДВ} \qquad (4.18)$$

де $\text{С}_{\text{С}_\text{П}}$ – собівартість або поточні витрати на розробку ПП (системи, мережі), грн;

$\text{Р}_{\text{рентаб.Н}}$ – нормативний рівень рентабельності, (дорівнює 15 %, коефіцієнт складає 0,15), грн;

m – торговельна надбавка ($m = 5-12\%$), грн.

$$P_{\text{дог.}} = 200616{,}7 * 1{,}2 = 240740 \text{ грн}$$

4.3 Оцінка конкурентоспроможності програмного продукту

У процесі дослідження виділимо основні технічні та економічні параметри базового та нового варіантів ПП. Також наводяться додаткові функції нового ПП.

Таблиця 4.2 – Характеристика основних техніко–економічних параметрів базового та нового варіантів ПП

№ П\П	Назва параметру	Одиниця виміру	Варіант		Характерис-тика параметра нового варіанта відносно базового (↑, ↓ чи =)
			базовий (аналог)	новий	
1.	Вартість ПП	грн	28735 (10)	239461 (9)	↑
2.	Простота та зручність інтерфейсу	бали	3	5	↑
3.	Кількість функцій ПП	шт.	4	8	↑
4.	Вага ПП	мбайт	15 (6)	25 (10)	↓
5.	Час виконання запиту	мкс	16 (10)	14 (9)	↑
6.	Можливість нарощування функціональних характеристик	бали	4	3	↓
7.	Час відновлення системи після збою	сек	40 (10)	30 (8)	↑
8.	Кількість людей, необхідних для обслуговування	чол.	1	1	=

Виходячи з отриманих результатів, розробка є кращою відносно базового варіанта за п’ятьма наступними параметрами:

1 – вартість ПП;

2 – простота та зручність інтерфейсу;

3 – кількість функцій ПП;

5 – час виконання запиту;

7 – час відновлення системи після збою.

За одним параметром новий і базовий варіанти ПП є ідентичними (8 – кількість людей, необхідних для обслуговування), а за параметром (4 – вага ПП та 6 – можливість нарощування функціональних характеристик) новий варіант продукту поступається базовому.

4.4 Моделювання конкурентоспроможності розробленого програмного продукту та його аналогу

Моделювання оцінки конкурентоспроможності нового виробу базується на побудові номограми для порівняння параметрів конкурентоспроможності. Відповідно, кожній із восьми осей з використанням певного масштабу виміру відмічаються точки, що відповідають певним значенням параметрів, наведені в таблиці 4.2. Лінія, що проходить через них, утворює багатокутник, таким чином отримуємо номограму для порівняння параметрів конкурентоспроможності базового та нового варіантів ПП (рис. 4.1).

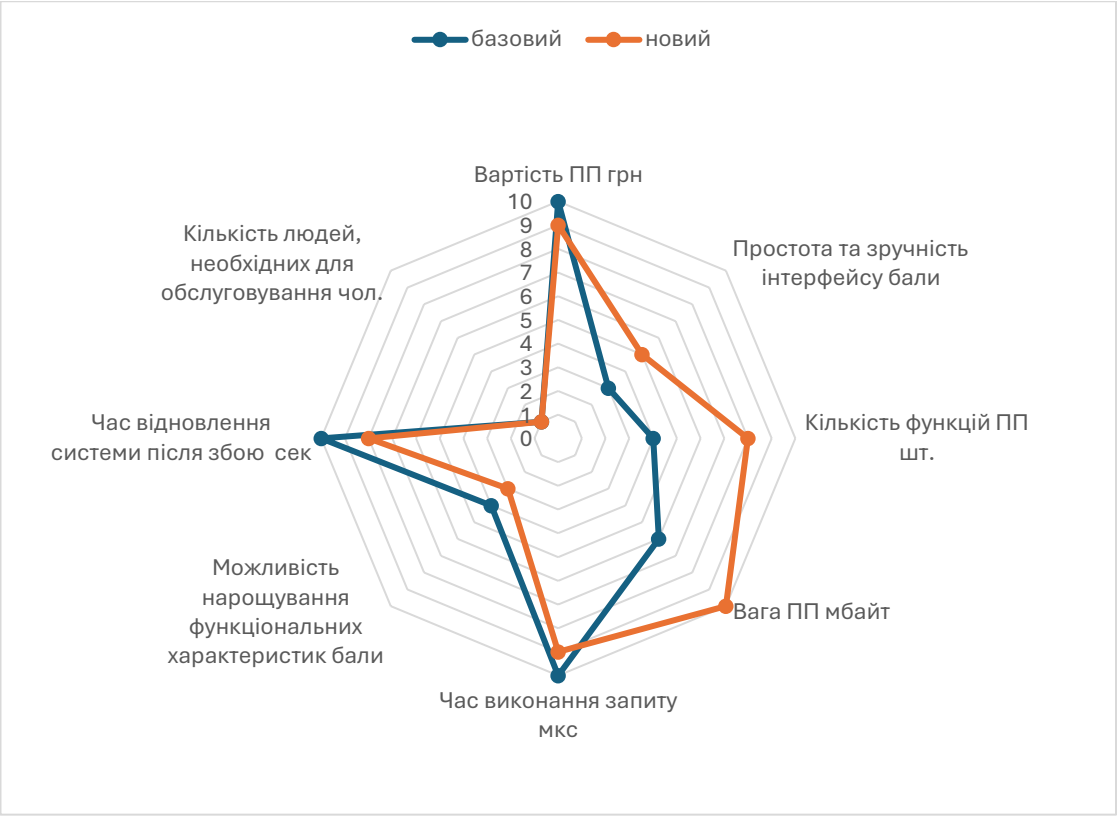

Рисунок 4.1 – Номограма для порівняння параметрів конкурентоспроможності базового та нового варіантів програмних продуктів

ВИСНОВКИ

Сьогодні переваги сучасного, адаптивного веб-додатку очевидні. Оскільки додаток, доступ до якого можна отримати через веб-браузер, можна використовувати як на комп'ютері, так і на будь-якому іншому мобільному пристрої з повною сумісністю, це забезпечує більшу гнучкість, ніж традиційний неадаптивний веб-додаток або нативний мобільний додаток. У поєднанні з простим, легким у використанні, орієнтованим на користувача дизайном, бібліотечний додаток в освітньому середовищі можна значно вдосконалити.

Метою проекту є розробка додатку, який полегшить користування бібліотекою навчального закладу, з більш орієнтованим на користувача та адаптивним дизайном, як для студентів, так і для викладачів та співробітників бібліотеки. Загалом, користувачі бібліотеки можуть в основному шукати ресурси в бібліотеці, переглядати інформацію про ресурси та легко здійснювати запити на отримання або резервування ресурсів, а співробітники бібліотеки можуть відповідати на запити на отримання або резервування та перевіряти поточні процеси видачі та резервування.

АРМ має на меті забезпечити користувачам бібліотеки універсальне використання як у навчанні, так і в повсякденному житті. Таким чином, вона має бути розроблена з використанням сучасних підходів, методологій та технологій. На додаток до цього, LMS також забезпечує рішення для виснажливого процесу пошуку, позичання або резервування ресурсу, відстеження позичання або резервування за допомогою сучасного адаптивного веб-додатку, пропонуючи в цілому простіший спосіб використання та управління бібліотекою ЄВС. АРМ приносить багато переваг як для студентів, так і для викладачів та бібліотекарів. Користувачі бібліотеки можуть переглядати та

використовувати всю систему будь-де, будь-коли та з будь-якого пристрою, таким чином, роблячи ці процеси повністю онлайн. Крім того, користувачі бібліотеки отримують сповіщення про будь-який зарезервований або взятий напрокат ресурс, коли у нього закінчується термін отримання або повернення, і можуть легко відслідковувати процеси резервування або взяття напрокат в режимі он-лайн. З іншого боку, бібліотекарі можуть керувати відстеженням бібліотечного фонду повністю онлайн, здійснювати операції з видачі також повністю онлайн без необхідності паперової роботи, а також мати більш безпечне середовище, оскільки тільки студенти та викладачі можуть резервувати або позичати ресурси.

Як веб-додаток, АРМ містить два додатки і працює як комбінація цих двох додатків. Внутрішній додаток, який отримує запити від клієнта і надає відповідь користувачеві, а також зовнішній додаток, де користувач взаємодіє з системою за допомогою графічного інтерфейсу. Орієнтуючись на сучасні методології та технології, бекенд-додаток був побудований з використанням NodeJS, ExpressJS та MySQL, тоді як фронтенд-додаток був побудований з використанням HTML, CSS, JavaScript та React.

Таким чином, результатом цієї роботи є веб додаток з інтеграцією бази даних для збереження даних у відповідності з розробленою архітектурою.

ПЕРЕЛІК ПОСИЛАНЬ

1. DSpace Home - DSpace. DSpace. URL: https://dspace.org/ (date of access: 26.06.2025).
2. IRTUMIP :: Головна. IRTUMIP :: Головна. URL: https://dspace.mipolytech.education/home (дата звернення: 02.07.2025).
3. Kortext Login. Kortext Login. URL: https://app.kortext.com/identity/signin (date of access: 26.06.2025).
4. Digital Commons @ Rhode Island School of Design. Digital Commons @ Rhode Island School of Design. URL: https://digitalcommons.risd.edu/ (date of access: 26.06.2025).
5. Каталоги - НБУВ Національна бібліотека України імені В. І. Вернадського. LIBNAS | LIBRARY PORTAL OF NATIONAL ACADEMY OF SCIENCES OF UKRAINE. URL: https://irbis-nbuv.gov.ua/aref/ (дата звернення: 26.06.2025).
6. Official Website of Koha Library Software. Official Website of Koha Library Software. URL: https://koha-community.org/ (date of access: 26.06.2025).
7. Library Technology Guides: ALEPH 500 Profile. Library Technology Guides: Documents, Databases, News, and Commentary. URL: https://librarytechnology.org/product/aleph/ (date of access: 26.06.2025).
8. Vickler A. Javascript: javascript front end programming. Independently Published, 2021. 426 с.
9. Сучасний підручник з JavaScript : веб-сайт. URL: https://uk.javascript.info/ (дата звернення: 26.06.2025)
10. React.js Documentation: веб-сайт. URL: https://react.dev/ (дата звернення: 26.06.2025)
11. Sammie Smith. Full Stack Web Development Guide: Everything Node JS, Express, APIs, EJS, React JS, Database Fundamentals, SQL Databases. Independently Published, 2023. 744 с.

12. Node. js Web Development. .: Server-Side Web Development Made Easy with Node 14 Using Practical Examples. de Gruyter GmbH, Walter, 2020.

13. MySQL Documentation: веб-сайт. URL: https://dev.mysql.com/doc/ (дата звернення: 26.06.2025)

14. React, Node.js, Express and MySQL CRUD app. *Corbado - Simplify passkeys for large-scale applications*. URL: https://www.corbado.com/blog/react-express-crud-app-mysql (date of access: 26.06.2025).

15. Christensson P. Client-Server Model. TechTerms.com - The Computer Dictionary. URL: https://techterms.com/definition/client-server_model (date of access: 26.06.2025).

ДОДАТОК А. ВІДОМОСТІ РОБОТИ

Таблиця А.1 – Відомості роботи

Формат	№ п/п	Назва документу	Найменування об’єкту або вибору	Кількість сторінок
А4	1	Пояснювальна записка	КЦТПАР.122-22-1п.00.00.00.ПЗ	98
Графічна частина				
А4	2	SADT-діаграма 0-рівня процесу А-0 «Автоматизоване робоче місце бібліотекара технічної бібліотеки підприємства»	КЦТПАР.122-22-1п.01.00.00.ПЛ	1
А4	3	Діаграма варіантів використання системи управління бібліотекою	КЦТПАР.122-22-1п.02.00.00.ПЛ	1
А4	4	Схема бази даних системи управління бібліотекою	КЦТПАР.122-22-1п.03.00.00.ПЛ	1
А4	5	Діаграма діяльності для входу в систему	КЦТПАР.122-22-1п.04.00.00.ПЛ	1
А4	6	Діаграма діяльності перевірки токену	КЦТПАР.122-22-1п.05.00.00.ПЛ	1
А4	7	Діаграма діяльності перевірки ролей користувачів бібліотеки	КЦТПАР.122-22-1п.06.00.00.ПЛ	1
А4	8	Діаграма діяльності перевірки ролі бібліотекаря	КЦТПАР.122-22-1п.07.00.00.ПЛ	1
А4	9	Екрані форми	КЦТПАР.122-22-1п.08.00.00.ПЛ	2

ДОДАТОК Б. ВИХІДНИЙ КОД

Код головного компонента React App.js

```
import Home from './Pages/Home';
import Signin from './Pages/Signin'
import { BrowserRouter as Router, Switch, Redirect, Route } from
"react-router-dom";
import MemberDashboard from
'./Pages/Dashboard/MemberDashboard/MemberDashboard.js';
import Allbooks from './Pages/Allbooks';
import Header from './Components/Header';
import AdminDashboard from
'./Pages/Dashboard/AdminDashboard/AdminDashboard.js';
import { useContext } from "react"
import { AuthContext } from "./Context/AuthContext.js"

function App() {

  const { user } = useContext(AuthContext)

  return (
    <Router>
      <Header />
      <div className="App">
        <Switch>
          <Route exact path='/'>
            <Home />
          </Route>
          <Route exact path='/signin'>
            {user ? (user.isAdmin ? <Redirect
to='/dashboard@admin' />:<Redirect to='/dashboard@member' />) :
<Signin />}
          </Route>
```

```
          <Route exact path='/dashboard@member'>
            {user ? (user.isAdmin === false ? <MemberDashboard
/> : <Redirect to='/' />) : <Redirect to='/' />}
          </Route>
          <Route exact path='/dashboard@admin'>
            {user ? (user.isAdmin === true ? <AdminDashboard />
: <Redirect to='/' />) : <Redirect to='/' />}
          </Route>
          <Route exact path='/books'>
            <Allbooks />
          </Route>
        </Switch>
      </div>
    </Router>
  );
}

export default App;
```

Код компонента Home

```
import React from 'react'

import About from '../Components/About'
import Footer from '../Components/Footer'
import ImageSlider from '../Components/ImageSlider'
import News from '../Components/News'
import PhotoGallery from '../Components/PhotoGallery'
import PopularBooks from '../Components/PopularBooks'
import RecentAddedBooks from '../Components/RecentAddedBooks'
import ReservedBooks from '../Components/ReservedBooks'
import Stats from '../Components/Stats'
import WelcomeBox from '../Components/WelcomeBox'
function Home() {
    return (
```

```
        <div id='home'>
            <ImageSlider/>
            <WelcomeBox/>
            <About/>
            <Stats/>
            <RecentAddedBooks/>
            <PopularBooks/>
            <ReservedBooks/>
            <News/>
            <PhotoGallery/>
            <Footer/>
        </div>
    )
}

export default Home
```

Код компонента SignIn

```
import React, { useContext, useState } from 'react'
import './Signin.css'
import axios from 'axios'
import { AuthContext } from '../Context/AuthContext.js'
import Switch from '@material-ui/core/Switch';

function Signin() {
    const [isStudent, setIsStudent] = useState(true)
    const [admissionId, setAdmissionId] = useState()
    const [employeeId,setEmployeeId] = useState()
    const [password, setPassword] = useState()
    const [error, setError] = useState("")
    const { dispatch } = useContext(AuthContext)

    const API_URL = process.env.REACT_APP_API_URL
```

```
    const loginCall = async (userCredential, dispatch) => {
        dispatch({ type: "LOGIN_START" });
        try {
            const res = await
axios.post(API_URL+"api/auth/signin", userCredential);
            dispatch({ type: "LOGIN_SUCCESS", payload: res.data
});
        }
        catch (err) {
            dispatch({ type: "LOGIN_FAILURE", payload: err })
            setError("Wrong Password Or Username")
        }
    }

    const handleForm = (e) => {
        e.preventDefault()
        isStudent
        ? loginCall({ admissionId, password }, dispatch)
        : loginCall({ employeeId,password }, dispatch)
    }

    return (
        <div className='signin-container'>
            <div className="signin-card">
                <form onSubmit={handleForm}>
                    <h2 className="signin-title"> Log in</h2>
                    <p className="line"></p>
                    <div className="persontype-question">
                        <p>Are you a Staff member ?</p>
                        <Switch
                            onChange={() =>
setIsStudent(!isStudent)}
                            color="primary"
                        />
                    </div>
```

```
                    <div className="error-
message"><p>{error}</p></div>
                    <div className="signin-fields">
                        <label
htmlFor={isStudent?"admissionId":"employeeId"}>
<b>{isStudent?"Admission ID":"Employee ID"}</b></label>
                        <input className='signin-textbox'
type="text" placeholder={isStudent?"Enter Admission ID":"Enter
Employee ID"} name={isStudent?"admissionId":"employeeId"}
required onChange={(e) => {
isStudent?setAdmissionId(e.target.value):setEmployeeId(e.target.
value) }}/>
                        <label
htmlFor="password"><b>Password</b></label>
                        <input className='signin-textbox'
type="password" minLength='6' placeholder="Enter Password"
name="psw" required onChange={(e) => {
setPassword(e.target.value) }} />
                        </div>
                    <button className="signin-button">Log
In</button>
                    <a className="forget-pass"
href="#home">Forgot password?</a>
                </form>
                <div className='signup-option'>
                    <p className="signup-question">Don't have an
account? Contact Librarian</p>
                </div>
            </div>
        </div>
    )
}
export default Signin
```