ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
МІСЬКОГО ГОСПОДАРСТВА ІМЕНІ О. М. БЕКЕТОВА

Пояснювальна записка
до кваліфікаційної роботи бакалавра

на тему: Проектування застосунку для управління бібліотечними архівними даними

Виконав: студент 4 курсу, групи КН 2022-1
спеціальності 122 Комп'ютерні науки
(шифр і назва спеціальності)

Кирило КАМЕНКОВ
(прізвище та ініціали)

Керівник: Олександр КОСТЕНКО
(прізвище та ініціали)

Рецензент: Марія ВОЄВОДІНА
(прізвище та ініціали)

м. Харків – 2026 рік

Харківський національний університет міського господарства імені О. М. Бекетова
(повне найменування закладу вищої освіти)
Навчально-науковий Інститут енергетичної, інформаційної та транспортної інфраструктури
Кафедра комп'ютерних наук та інформаційних технологій
Рівень вищої освіти перший (бакалаврський)
Спеціальність 122 Комп'ютерні науки
(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри КНтаІТ

Марина НОВОЖИЛОВА
«_23___» __червня___2026 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ
Каменкова Кирила Ігоровича
(прізвище, ім'я, по батькові)

1. Тема роботи Проектування застосунку для управління бібліотечними архівними даними

керівник роботи канд. фіз.-мат. наук, доц. Костенко О. Б.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом закладу вищої освіти від «22» травня 2026 р. № 440-03

2. Термін подання здобувачем роботи 15 червня 2026 р.

3. Вихідні дані до роботи Рекомендації щодо реалізації застосунку для управління бібліотечними архівними даними

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

− дослідження загальних принципів керування бібліотечними та архівними даними;
− опис та порівняння наявних аналогів;
− розробка архітектури застосунку, структури бази даних;
− розробка прототипа застосунку для управління бібліотечними архівними даними, розробка критично важливих модулів застосунку;
− розглянути питання, пов'язані із організацією та забезпеченням охорони праці як важливої складової безпеки працівників.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Презентація – 18 аркушів

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Розділ I	Костенко О.Б., доцент каф КНтаІТ	11.05.2026	23.05.2026
Розділ II	Костенко О.Б., доцент каф КНтаІТ	24.05.2026	02.06.2026
Розділ III	Костенко О.Б., доцент каф КНтаІТ	03.06.2026	10.06.2026
Розділ IV	Малишева В.В., доцент каф. ОПтаБЖ	11.06.2026	14.06.2026

7. Дата видачі завдання 11.05. 2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Вибір теми кваліфікаційної роботи	11.05.2026	
2	Затвердження тем, наукових керівників, завдань та календарного плану підготовки кваліфікаційної роботи	15.05.2026	
3	Написання I розділу	23.05.2026	
4	Написання II розділу	02.06.2026	
5	Написання III розділу	10.06.2026	
6	Написання IV розділу	14.06.2026	
7	Подання кваліфікаційної роботи керівнику	15.06.2026	
8	Робота по усуненню зауважень керівника, уточнення і доповнення практичного матеріалу, оформлення додатків до роботи	16.06.2026	
9	Подання доопрацьованого варіанту роботи керівнику	16.06.2026	
10	Захист матеріалів кваліфікаційної роботи на засіданні кафедри	18.06.2026	
11	Офіційний захист матеріалів кваліфікаційної роботи на засіданні Державної екзаменаційної комісії	25.06.2026	

Студент _____ (підпис) Кирило КАМЕНКОВ (прізвище та ініціали)

Керівник роботи _____ (підпис) Олександр КОСТЕНКО (прізвище та ініціали)"

АНОТАЦІЯ

Структура та обсяг роботи. Пояснювальна записка кваліфікаційної роботи бакалавра студента групи КН 2022-1 спеціальності 122 Комп'ютерні науки Каменкова Кирила Ігоровича за темою «Проектування застосунку для управління бібліотечними архівними даними» складається з 4 розділів, містить 11 рисунків, 4 таблиці, 24 джерел.

Кваліфікаційну роботу бакалавра присвячено розробці прототипа застосунку для управління бібліотечними архівними даними, що спириятиме збереженню та покращить доступ до бібліотечних архівних даних.

У першому розділі описано предметне середовище, розглянуто концептуальні питання автоматизації бібліотек, стан в Україні, були наведені наявні аналоги та проведено їх порівняння. Також сформований об'єкт та предмет дослідження.

У аналітичному розділі описане предметне середовище з точки зору функціональних вимог, побудовано UML-діаграми, описано архітектуру застосунку, спроєктовано базу даних, розглянуто алгоритми ключових процесів.

У розділі програмного та технічного забезпечення описано вибір засобів програмної реалізації, реалізовано основні модулі системи: автентифікації і розмежування прав, повнотекстового пошуку та фільтрації, завантаження та збереження цифрових копій, надано рекомендації щодо розробки інтерфейса користувача та тестування системи.

У розділі охорони праці визначені вимоги до організації робочого оточення та були запропоновані рекомендації щодо покращення умов праці та зниження рівня професійних ризиків.

Ключові слова: БІБЛІОТЕЧНА ІНФОРМАЦІЙНА СИСТЕМА, ПРОЄКТУВАННЯ ЗАСТОСУНКІВ, UML-ДІАГРАМИ, АРХІВНІ ДАНІ, БАЗА ДАНИХ, АВТОМАТИЗАЦІЯ ПРОЦЕСІВ.

SUMMARY

Structure and scope of work. The explanatory note of the bachelor's qualification work of the student of the group CN 2022-1 specialty 122 Computer Science Kamenkov Kyrylo Ihorovych on the topic " Designing an application for managing library archival data " consists of 4 sections, contains 11 figures, 4 tables, 24 sources.

The bachelor's thesis is devoted to the developing a prototype application for managing library archival data that will support the preservation and improve access to library archival data.

In the first section, the subject environment is described, conceptual issues of library automation are considered, the state in Ukraine is considered, existing analogues are presented and their comparison is carried out. The object and subject of the study are also formed.

The analytical section, the subject environment is described from the point of view of functional requirements, UML diagrams are constructed, the application architecture is described, the database is designed, and the algorithms of key processes are considered.

In the software and hardware section, the choice of software implementation tools is described, the main modules of the system are implemented: authentication and rights delimitation, full-text search and filtering, downloading and saving digital copies, recommendations are provided for user interface development and system testing.

The section of labor protection defines the requirements for the organization of the working environment, taking into account harmful and dangerous production factors.

Keywords: LIBRARY INFORMATION SYSTEM, APPLICATION DESIGN, UML DIAGRAMS, ARCHIVE DATA, DATABASE, PROCESS AUTOMATION.

ЗМІСТ

ВСТУП

Історія бібліотек налічує тисячі років, і роль бібліотек у людському житті весь цей час змінювалась. Століттями це слово використовувалося виключно для позначення фізичних приміщень зібрання книг.

Змінювались носії інформації: від глиняних табличок в Вавілоні та папірусів у стародавньому Єгипті, до паперу і всієї різноманітності носіїв інформації у сучасному світі. Згадаємо, що Олександрійська бібліотека виступала головним центром знань античності.

У Середньовіччі та у новий час центрами збереження рукописів ставали монастирі. Саме у цей час з’явилось друкарство і збільшилась кількість грамотних людей – споживачів послуг бібліотек. Згодом з'явилися університетські та національні книгозбірні (наприклад, Британська бібліотека). В Україні першу бібліотеку заснував Ярослав Мудрий у 1037 р.

З ростом фондів бібліотек постало питання переходу від хаотичного зберігання до єдиних правил каталогізації та обміну даними, який розпочався у XIX столітті та завершився створенням уніфікованих міжнародних протоколів. У 1841 р. у Британському музеї було розроблено перший сучасний стандарт описової каталогізації, який уніфікував пошук книг для великих зібрань . У 1876 р. у США вперше була розроблена систематизація та розміщення книг за галузями знань на полицях.

В епоху цифрової революції, що припала на 1960-ті роки, почався процес автоматизації, розроблено формат MARC (Machine-Readable Cataloging), який став глобальним стандартом для машиночитаної каталогізації бібліотечних фондів.

З 2013 р. відбувається перехід провідних бібліотек світу на стандарт RDA (Resource Description and Access), розроблений для цифрового середовища, зручного зв'язку баз даних та мереж Web 2.0.

Пройшовши еволюцію від царських архівів до сучасного стану бібліотеки перетворилися на осередки інформації. Сучасна бібліотека – це не просто сховище книг, а багатофункціональний відкритий простір та інтерактивний хаб. Вона поєднує функції коворкінгу, освітнього центру та зони для відпочинку, надаючи вільний доступ до технологій, літератури та спілкування; вона поєднує інформацію на традиційних паперових носіях, на електронних цифрових носіях та стрімінгових та хмарних базах даних, мультимедійні матеріали.

Отже, основні відмінності та переваги сучасної бібліотеки:

- Мультимедійність. Замість паперових архівів – електронні бази даних, доступ до світових цифрових бібліотек, аудіокниг та лекторіїв.
- Відкритий простір. Це комфортна локація для роботи, ділових зустрічей, презентацій та майстер-класів.
- Діджиталізація та технології. Надання вільного доступу до швидкісного інтернету, техніки (ноутбуки, 3D-принтери, VR-окуляри) та допомога у розвитку цифрової грамотності.
- Розвиток громад. Локація для обговорень, суспільної самоорганізації, культурних заходів та соціалізації різних вікових груп.
- Інклюзивність та безбар'єрність. Створення рівних умов доступу, включаючи адаптовані простори для людей з інвалідністю, дітей та молоді.

Завдяки такій трансформації Українська бібліотечна асоціація розглядає бібліотеку як потужний осередок розвитку людського капіталу[1,2].

Проектування застосунку для управління бібліотечними архівними даними потребує створення надійної архітектури, що забезпечить безпечне зберігання, швидкий пошук та довготривале збереження оцифрованих фондів, рідкісних видань чи рукописів.

РОЗДІЛ 1 ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1 Опис предметного середовища

Автоматизація бібліотек – це впровадження сучасних бібліотечних інформаційних систем (АБІС) для оптимізації роботи з фондами та обслуговування читачів. Автоматизація бібліотечних процесів виникла на пустому місці. В її основі лежать бібліотечні технології, розроблені і впроваджені ще до появи комп'ютерної техніки.

Це і збільшення точок доступу до документу (ключові слова, предметні рубрики, коди систематизації), і розмежування діяльності відділів за призначенням або типом документів зберігання, і стандартизація бібліографічних описів та навіть карткових каталогів і шаф. Слід зазначити, що деякі з бібліотечних технологій пошуку широко застосовані в сучасних пошукових системах.

З появою АБІС бібліотека перетворилась на сучасний інформаційний хаб: паперові процеси змінили цифрові, спростився пошук книг і відкрився доступ до електронних баз даних.

Сформулюємо основні завдання автоматизації.

По-перше, електронний каталог. Відмова від паперових карток на користь електронних записів заради швидкого пошуку за автором, назвою, ISBN, кодами систематизації, ключовими словами, тощо. Електронний каталог поєднує в собі два паперові: алфавітний і систематичний; він має більше точок доступу та можливостей швидкого пошуку.

По-друге, обслуговування користувачів: Електронні читацькі квитки, автоматизована видача та повернення книг за рахунок штрих-кодування фонду та використання спеціалізованого обладнання.

По - третє, обробка фондів. Спрощення обліку нових надходжень, списання та інвентаризації (за рахунок роботи з електронним каталогом, штрихкодування).

По-четверте, звітність. Автоматичне формування статистики відвідуваності та книговидачі (електронний каталог, агрегування електронних читацьких квитків).

Переваги впровадження АБІС для читачів та бібліотекарів (рис.1.1):

- Для відвідувачів: доступ до каталогів з будь-якого пристрою 24/7, онлайн-бронювання книг.
- Для працівників: економія часу, відсутність рутинної паперової роботи, миттєвий обмін даними.

Майбутнє та інновації у подальшому розвитку АБІС:

- Хмарні технології: Доступ до системи через інтернет без прив'язки до локального комп'ютера.
- Електронні бібліотеки: Інтеграція мультимедійних ресурсів та світових наукових баз.
- RFID-технології: Використання міток для швидкого сканування книг та захисту від крадіжок.

Рисунок 1.1 – Надбання АБІС

Отже, автоматизація – це не просто комп'ютеризація, а перетворення бібліотек на сучасні інформаційні центри, комфортні для користувачів та ефективні для роботи[1,2].

1.2 Основні функціональні модулі застосунку та їх взаємозв’язок

Застосунок для управління бібліотечними архівними даними є окремою частиною АБІС. Його призначення – працювати з окремою специфічною часткою бібліотечного фонду, що складає його дуже цінну та унікальну частку.

Термін «архівні дані» у контексті бібліотечних застосунків використовують через особливий статус, цінність та специфіку зберігання цієї інформації.

Основні причини, чому ці дані називають архівними:

1. Довготривале та постійне збереження

Мета: На відміну від звичайних книг, які з часом списують через зношення, архівні дані підлягають вічному або наддовгому зберіганню.

Захист: Їх не можна видаляти чи перезаписувати, оскільки вони мають історичну, наукову або культурну цінність.

2. Специфіка першоджерел

Унікальність: Часто це оцифровані копії рідкісних видань, рукописів, стародруків, листів або карт, які існують в одному екземплярі.

Особливий статус: Фізичні оригінали зберігаються в спеціальних сховищах (архівах) з обмеженим доступом, а застосунок керує їхніми цифровими копіями.

3. Невживаність у щоденному побуті

Рідкісний доступ: Ці дані не потрібні користувачам щодня (як художня література чи свіжі підручники).

4. Дослідницький характер

До них звертаються переважно науковці, історики або аналітики для спеціальних досліджень.

1. Основні функціональні модулі

- Каталогізація та інвентаризація: Введення метаданих (автор, рік, ISBN, УДК, ключові слова) та прив'язка до фізичного розташування (фонд, стелаж, полиця).
- Модуль оцифрування: Завантаження медіафайлів (PDF, TIFF, JPEG) та їх автоматична конвертація/оптимізація для перегляду.
- Пошукова система: Повнотекстовий пошук за текстом документів (OCR), фільтрація за категоріями та тегами.
- Керування доступом: Розмежування ролей: Гість/Студент (читання), Архіваріус (редагування), Адміністратор (керування доступом).
- Система резервного копіювання: Забезпечення захисту від втрати історичних даних. [1, 2]

2. Архітектура застосунку

Для ефективної роботи з великими архівами використовується клієнт-серверна архітектура:

- Бекенд (Backend): Мови програмування Python (Django або FastAPI) або Java (Spring Boot) завдяки потужним бібліотекам для обробки даних та інтеграції з алгоритмами розпізнавання тексту (OCR).
- Фронтенд (Frontend): React.js або Vue.js для створення зручного інтерактивного інтерфейсу користувача.
- База даних: Гібридний підхід. Реляційна БД (наприклад, PostgreSQL) для зберігання метаданих користувачів та каталогів. Нереляційна БД (наприклад, MongoDB) для зберігання гнучких метаданих різних типів документів (рукописи, карти, газети).

1.3 Стандарти і безпека в галузі

Щоб застосунок був сумісним із загальноприйнятими світовими бібліотечними системами, необхідно передбачити підтримку наступних протоколів та стандартів та забезпечити збереження даних та фонду.

Стандарти

- MARC21: Міжнародний формат для обміну бібліографічними записами.
- Dublin Core: Стандарт метаданих для опису цифрових ресурсів.
- OAI-PMH: Протокол, що дозволяє іншим системам (наприклад, національним архівам) збирати метадані з вашого застосунку.

Безпека та фізичне збереження

- Шифрування: Захист даних під час передачі (SSL/TLS) та збереження конфіденційних даних у базі.
- Автентифікація: Впровадження захищеної авторизації через електронний підпис (Дія.Підпис), або інтеграція з корпоративними каталогами (Active Directory).
- Аудит дій: Журналювання (логірування) усіх дій користувачів (хто переглядав, редагував або видаляв архівні дані)[3,4].

1.4 Огляд розповсюджених АБІС в Україні

В Україні на 2020 р. налічується 16 тис. бібліотек, які можуть стати цифровими хабами. Тому потрібно більшість з них підключити до якісного та швидкісного інтернету, що є необхідною умовою цифровізації [3].

До будь-якої технології підключені лише 4 662 бібліотек з 16 038 – це 29%. Виняток – тільки Київ, де 90% бібліотек мають інтернет. Доступ до Wi-Fi мережі є у 3 733 бібліотек.

До оптики підключені лише 954 бібліотеки із загальної кількості. Тобто 92% бібліотек не мають якісного інтернету, і це означає, що в таких закладах є суттєві обмеження у використанні контенту (табл. 1.1)

Таблиця 1.1 – Наявність будь-якого інтернету в бібліотеці

Області України	Інтернет відсутній	Інтернет наявний	Разом
Вінницька	590	332	922
Волинська	417	142	559
Дніпропетровська	381	269	650
Донецька	270	155	425
Житомирська	646	147	793
Закарпатська	356	120	476
Запорізька	320	158	478
Івано-Франківська	541	185	726
Київ	14	129	143
Київська	720	145	865
Кіровоградська	385	185	570
Луганська	164	135	299
Львівська	1025	259	1284
Миколаївська	258	224	482
Одеська	581	195	776
Полтавська	448	334	782
Рівненська	400	168	568
Сумська	381	164	545
Тернопільська	676	148	824
Харківська	555	259	814
Херсонська	300	147	447
Хмельницька	549	254	803
Черкаська	513	202	715
Чернівецька	280	114	394
Чернігівська	606	92	698
Разом	**11376**	**4662**	**16038**

Рисунок 1.2 – Наявність інтернету в бібліотеках України загалом і в Харківській області (порівняння)

Розповсюджені в Україні АБІС:

- Koha: Найвідоміша у світі безкоштовна АБІС із відкритим вихідним кодом, що ідеально підходить для бібліотек різного типу.
- MAPK-SQL: Популярна система для створення електронних каталогів та повнотекстових баз.
- ІРБІС: Традиційна система для комплексної автоматизації бібліотечних процесів.

Відомо, які саме АБІС впроваджені в бібліотеках України. Це: ALEPH 500, Koha, ІрбІс, УФД (Український фондовий дім), UniLib, MAPK-SQL, Absotheque UNICODE (рис. 1.3).

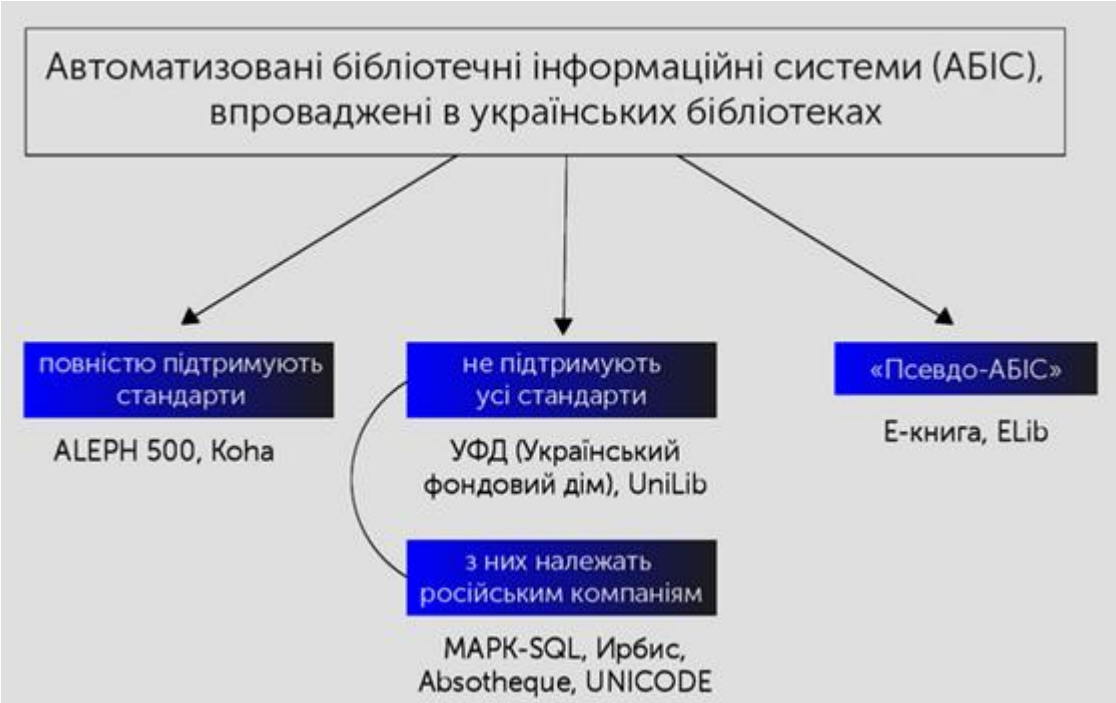


Рисунок 1.3 – АБІС, що впроваджено в Україні

Основна вимога до АБІС – підтримка сучасних міжнародних стандартів. Саме завдяки стандартам можна повноцінно поєднати свої бази даних з іншими бібліотечними базами даних, обмінюватись бібліографічними та авторитетними записами, брати участь у майбутніх національних (система централізованої каталогізації та зведений каталог) та наявних міжнародних (WorldCat, VIAF) проєктах.

Серед названих ПЗ повністю такі стандарти підтримують лише ALEPH 500 та Koha. Деякі із цих систем розроблені та належать компаніям країни-агресора(МАРК-SQL, ІРБІС, Absotheque UNICODE), тому відбувається поступова відмова від них і через загрозу кібербезпеці та санкцій, українські академічні й публічні книгозбірні нині масово переходять з російського програмного забезпечення на відкриту систему Koha[2,3].

Звичайно, у світі є й інші АБІС, які відповідають необхідним стандартам, що наведені у Довіднику бібліотек світу, в якому представлено і 113 бібліотек з України.

В Україні найбільш поширеними автоматизованими бібліотечними інформаційними системами (АБІС) є українська УФД/Бібліотека, російський ІРБІС (від використання якого наразі активно відмовляються через безпекові обмеження) та безкоштовна міжнародна система Koha з відкритим кодом.

УФД/Бібліотека: Найпопулярніша комплексна система, розроблена в Києві. Повністю підтримує штрихкодування, RFID-технології. Впроваджено у НБ Державного торговельно-економічного університету, НБ Запорізького національного університету та численних університетських й публічних закладах.

АБІС Koha: Перша у світі вільна бібліотечна система з відкритим кодом. З 2005-2006 років активно адаптується та підтримується спільнотою в Україні. Головна перевага – повна незалежність, відсутність ліцензійних зборів та зручний онлайн-доступ. Впроваджено у НТБ Тернопільського національного технічного університету, НБ Харківського національного університету міського господарства ім. О. М. Бекетова, НТБ КПІ ім. Ігоря Сікорського та Чернівецькій обласній універсальній бібліотеці.

ALEPH (ExLibris): Потужна ізраїльська система для великих національних та наукових книгозвірень, має високу вартість. Впроваджено в Львівській національній науковій бібліотеці України імені В. Стефаника.

Як правило, ці системи використовуються університетськими й універсальними науковими бібліотеками.

В Харкові найпоширенішими АБІС є:

Уфд:

- НТБ Національного аерокосмічного університету ім. М. Є. Жуковського («ХАІ»)
- Наукова бібліотека Харківського національного університету радіоелектроніки («ХНУРЕ»)
- Бібліотека Харківської державної академії культури («ХДАК»)
- Наукова бібліотека Державного біотехнологічного університету

Koha**:**

- Наукова бібліотека Харківського національного університету міського господарства імені О. М. Бекетова.

ІРБІС:

- Харківська державна наукова бібліотека ім. В. Г. Короленка – головна обласна наукова бібліотека, яка будує свій зведений електронний каталог на базі платформи АБІС «ІРБІС-64».
- Харківська обласна універсальна наукова бібліотека – використовує систему «ІРБІС» для формування свого електронного каталогу.
- Наукова бібліотека Національного юридичного університету імені Ярослава Мудрого – застосовує «ІРБІС» як базову основу інформаційного середовища та для забезпечення доступу до електронного каталогу університету.
- Наукова бібліотека Національного фармацевтичного університету – використовує АБІС ІРБІС та штрихкодування для автоматизації книговидачі та ведення електронних каталогів.
- Наукова бібліотека Української інженерно-педагогічної академії (УІПА) – впроваджує технології «ІРБІС» для автоматизації бібліотечних процесів.
- Наукова бібліотека Харківської державної академії культури (ХДАК) – застосовує можливості системи в роботі зі студентами.
- Інші спеціалізовані установи – наприклад, бібліотека Інституту проблем кріобіології і кріомедицини НАН України[3].

1.5 Визначення об’єкту та предмету дослідження

Актуальність розробки застосунку для управління архівними бібліотечними даними є нагальною потребою. Цифровізація бібіліотек в наш час вже досить поширена, але всі продукти автоматизації скоріше

пристосовані до стандартних бібліотечних документів. Тому створення спеціалізованого застосунку для архівних (нестандартних) документів є актуальним.

Такий застосунок може працювати у поєднанні з іншими АБІС, і в співдружності з музеями, архівами, і в проєкті «Національна електронна бібліотека України» за підтримки ЮНЕСКО як спільної платформи бібліотек, музеїв та архівів для збереження культурної спадщини. Сьогодні колекції рідкісних і цінних видань, рукописів та краєзнавчі фонди бібліотек України через війну та окупацію зазнають безпрецедентних втрат. Понад 130 бібліотек повністю зруйновані, сотні потребують ремонту, а загалом книгозбірні втратили понад 1,5 млн примірників.

В умовах активних бойових дій найцінніші колекції (інкунабули, палеотипи, стародруки та рукописи) переміщують у безпечніші регіони або облаштовують для них захищені сховища; також діють програми зі створення цифрових копій. Це дозволяє зберегти доступ до унікальних документів, навіть якщо фізичний оригінал постраждає від обстрілів. [4]

Мета розробки: створення спеціалізованого програмного забезпечення для цифровізації архівних фондів бібліотеки, що забезпечить надійне зберігання метаданих, розмежування прав доступу користувачів та спрощення процедури пошуку документів, що сприятиме збереженню культурної спадщини.

Призначення розробки: організація єдиного інформаційного простору для обліку, збереження та оперативного пошуку бібліотечних архівних даних, а також автоматизація щоденних рутинних операцій персоналу бібліотеки та забезпечення швидкого санкціонованого доступу користувачів до електронних інформаційних ресурсів.

Об’єктом дослідження є процеси автоматизації роботи з інформаційними ресурсами (зберігання, пошук, обробка та архівація).

Предмет дослідження: сукупність процесів, методів та алгоритмів, що забезпечують формування, структурування, збереження, пошук, захист та взаємодію з інформаційними ресурсами (метаданими, цифровими копіями, каталогами).

Цілі розробки: Запропонувати технічні рішення щодо розробки застосунку автоматизації роботи з об’єктами культурної спадщини в галузі бібліотек.

Сформулюємо задачі дослідження:

1. Провести глибокий аналіз предметної області.
2. Розробити оптимальну архітектуру застосунку.
3. Вивчити потреби користувачів (бібліотекарів, архіваріусів, читачів)
4. Створити прототип інтуїтивного, швидкого та функціонального цифрового продукту

Висновки до розділу

В розділі розглянуто концептуальні питання дослідження автоматизації бібліотечних процесів.

В розділі було описано предметне середовище, byd prjведений аналіз цифровізації бібліотек та впровадження АБІС в Україні.

Був сформований об’єкт дослідження – процеси автоматизації роботи з інформаційними ресурсами.

Також визначений предмет дослідження – сукупність процесів, методів та алгоритмів, що забезпечують формування, структурування, збереження, пошук, захист та взаємодію з інформаційними ресурсами.

РОЗДІЛ 2 АНАЛІТИЧНИЙ РОЗДІЛ

2.1 Аналіз предметної області

Інженерний підхід до будь-якої задачі передбачає перетворення вимог, що висуває предметна область, на технічні схеми та архітектурні моделі.

В першому розділі було досліджено предметну область, тепер перейдемо до функціональних вимог до застосунку[5] (табл. 2.1).

Таблиця 2.1 – Функціональні вимоги як результат аналізу предметної області

Результат аналізу предметної області	Функціональна вимога застосунку
Читачі витрачають багато часу на пошук паперових карток у старих каталогах.	Система повинна реалізовувати повнотекстовий пошук за автором, назвою, роком видання та ключовими словами.
Старі архівні документи мають обмежений фізичний доступ через ризик пошкодження.	Застосунок повинен підтримувати завантаження, зберігання та відображення цифрових копій (PDF, JPEG) документів
Кожен архівний фонд має свій унікальний шифр зберігання (номер фонду, опису, справи)..	База даних та інтерфейс системи повинні містити обов'язкові поля для введення індексів архівного шифрування
Бібліотека має щомісяця звітувати про кількість виданих та повернутих книг.	Застосунок повинен мати модуль генерації звітів із можливістю експорту даних у формат Excel/PDF за обраний період.

2.1.1 Функціональні вимоги (Use Case)

Функціональні вимоги – це опис того, що саме повинна робити система. Вони визначають функції, поведінку, сервіси та завдання, які застосунок зобов'язаний виконувати у відповідь на дії користувача або інші зовнішні події.[5,6] Для технічної чіткості їх прийнято групувати за ролями

користувачів (акторами системи). У системі виділено три основні дійові особи (Actors).

- Користувач (Читач/Дослідник, Reader). Зовнішній користувач, який шукає інформацію, переглядає відкриті цифрові копії та замовляє фізичні документи. Він реєструється та авторизується в системі, здійснює повнотекстовий та атрибутивний пошук (за автором, шифром, фондом), формує запити на видачу книги або перегляд архівного документа.
- Бібліотекар / Архіваріус (Archivist). Внутрішній користувач (співробітник), відповідальний за ведення фондів, оцифрування документів та видачу матеріалів. Він внносить нові надходження (книг, періодики, рукописів), цифровізує та прикріплює скановані копії (PDF, JPEG) до архівних карток, змінює статуси документів («в архіві», «видано», «на реставрації»).
- Адміністратор (Admin). Технічний спеціаліст, який керує доступами, обліковими записами та архівацією самої системи. Він керує правами доступу (ролями), проводить аудит дій користувачів (лог завантажень рідкісних документів), проводить резервне копіювання бази даних.

Деталізуємо критично важливі процеси Use Case за допомогою текстового сценарію.

Опишемо дії акторів при роботі з системою за допомогою діаграми прецедентів (рис. 2.1).

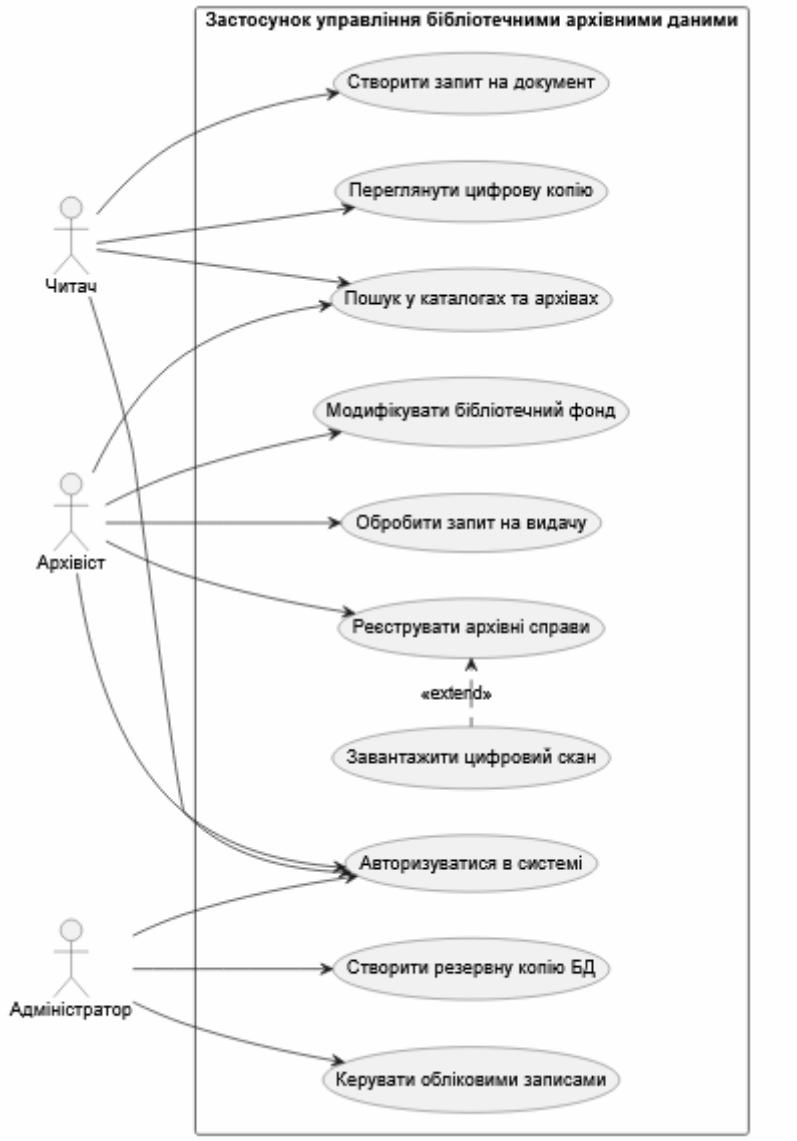


Рисунок 2.1 – Use Case (прецеденти використання)

2.1.2 Специфікація прецеденту «Створити запит на документ»

- Актор: Читач.
- Мета: Надіслати заявку на отримання доступу до книги чи архівного документу.
- Передумови: Користувач успішно пройшов авторизацію в системі.

Основний сценарій (Main Flow):

- Читач виконує пошук у системі та знаходить потрібну книгу чи архівний документ.
- Читач натискає кнопку «Створити запит».
- Система перевіряє статус документа (чи не виданий він уже іншому читачу).
- Система генерує електронну форму запиту, де автоматично підтягуються дані користувача та інвентарний номер документа.
- Читач підтверджує надсилання.
- Система реєструє новий запит у таблиці Document_Requests зі статусом "новий" та надсилає сповіщення архіваріусу.

Альтернативний сценарій:

- Якщо документ має статус "видано" або "на реставрації", система блокує кнопку створення запиту та виводить повідомлення про неможливість замовлення із зазначенням дати очікуваного звільнення матеріалу.

2.1.3 Специфікація прецеденту «Обробити запит на видачу»

- Актор: Бібліотекар / Архівіст.
- Мета: Затвердити або відхилити запит користувача на отримання документа.
- Передумови: У системі з'явився новий запит від читача.

Основний сценарій (Main Flow):

- Архівіст відкриває список нових заявок в робочій панелі застосунку.
- Вибирає конкретну заявку для розгляду.
- Архівіст перевіряє фізичну наявність документа на полиці або рівень доступу користувача до архіву.
- Архівіст натискає кнопку «Затвердити видачу».

- Система змінює статус запиту на "видано", фіксує поточну дату як request_date та автоматично змінює статус самого документа в каталозі на "недоступно".

Виключна ситуація (Скасування):

- Якщо документ фізично пошкоджено або він відсутній, архівіст натискає «Відхилити запит», вказує причину у текстовому полі, а система надсилає сповіщення в особистий кабінет читача, залишаючи статус книги як "доступно".

2.3.3 Діаграма послідовності (Sequence Diagram) для процесу авторизації

У процесі беруть участь чотири компоненти системи (згідно з трирівневою архітектурою)[7]:

- User (Користувач) – актор, який намагається увійти в систему.
- Client / View (Фронтенд інтерфейс) – форма вводу (наприклад, сторінка на React).
- AuthService (Бекенд контролер) – серверна логіка, що обробляє запити (наприклад, Node.js або Python API).
- Database (База даних) – сховище даних (PostgreSQL), де лежать хедші.

Процес автентифікації користувачів у системі управління бібліотечними та архівними даними реалізовано за допомогою захищеного протоколу із використанням токенів доступу (JWT)[8]. Хронологічний порядок взаємодії компонентів системи під час авторизації складається з таких кроків:

1. Ініціалізація: Користувач вводить свій логін (username) та пароль у форму інтерфейсу LoginView.

2. Первинна перевірка: На стороні клієнта (Frontend) виконується базова валідація даних (перевірка на наявність порожніх полів та довжину рядка).

3. Передача запиту: Клієнтський застосунок надсилає асинхронний POST-запит до серверного компонента AuthService. Пароль передається виключно через захищене з'єднання HTTPS.

4. Запит до БД: AuthService звертається до бази даних PostgreSQL із SQL-запитом для пошуку користувача за вказаним логіном. База даних повертає рядок із даними, що містить ідентифікатор ролі (role_id) та захешований пароль (password_hash).

5. Верифікація: Серверний модуль за допомогою криптографічних бібліотек здійснює хешування введеного користувачем пароля та порівнює результат із хешем із БД.

6. Розгалуження (Умовний оператор Alt):

- Сценарій А (Успіх): Якщо хеші збігаються, сервер генерує унікальний сесійний JWT-токен, куди зашивається інформація про права доступу користувача. Клієнт отримує статус 200 OK, зберігає токен у LocalStorage та перенаправляє користувача у відповідний інтерфейс (читача, архівіста або адміна).

- Сценарій Б (Помилка): Якщо логін не знайдено або паролі не збігалися, сервер повертає код помилки 401 Unauthorized. Інтерфейс виводить безпечне інформаційне повідомлення ("Невірний логін або пароль"), не розкриваючи, яке саме поле було введено невірно (це стандартна вимога безпеки OWASP – Open Worldwide Application Security Project – для захисту від підбору паролів).

Діаграма послідовності (Sequence Diagram) для процесу авторизації представлена у ДОДАТКУ Б.

2.2 Архітектура системи (System Architecture)

Для сучасного застосунку управління даними найкраще обрати трирівневу архітектуру (Three-Tier Architecture) на базі клієнт-серверної моделі (рис.2.2). Це забезпечить масштабованість та незалежність інтерфейсу від бази даних та стабільність застосунку[8].

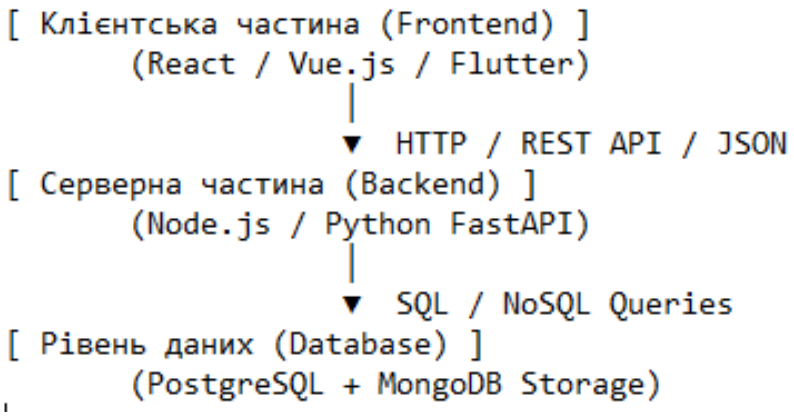


Рисунок 2.2 – Схема трирівневої архітектури застосунку

- Presentation Tier (Фронтенд): Веб-інтерфейс, який відправляє асинхронні запити.
- Logic Tier (Бекенд): REST API сервер, який обробляє бізнес-логіку (перевірка прав, валідація інвентарних номерів).
- Data Tier (База даних): Зберігає структуровані каталоги та посилання на файлові архіви.

2.2.1 Компоненти трирівневої архітектури інформаційної системи

Опишемо структуру системи через її основні рівні (рис.2.3):

1. Рівень представлення (Presentation Tier / Frontend):
   - Призначення: Взаємодія з користувачем, відображення каталогів, форм запитів та електронних документів.

- Реалізація: Односторінковий вебзастосунок (SPA) на базі фреймворку React.js або Vue.js. Він є повністю «легким» – не виконує обчислень, а лише надсилає асинхронні запити HTTP/HTTPS до сервера.

2. Рівень бізнес-логіки (Application Tier / Backend API):

- Призначення: Обробка запитів від фронтенду, перевірка прав доступу за рольовою моделлю, валідація інвентарних номерів книг та формування відповідей у форматі JSON.
- Реалізація: Серверний застосунок на Node.js (Express) або Python (FastAPI). Цей рівень є ізольованим від прямого доступу користувачів.

3. Рівень управління даними (Data Tier / Databases):

- Призначення: Надійне та безпечне збереження інформації.
- Реалізація:
  - Реляційна СКБД (PostgreSQL / MySQL) – для збереження структурованих даних (користувачі, книги, логіка видачі та запитів).
  - Файлове сховище (Object Storage / AWS S3 або MinIO) – для зберігання «важких» цифрових копій архівних документів (PDF, TIFF, JPEG). Бекенд зберігає в текстовому полі БД лише унікальний URL-шлях до файлу.

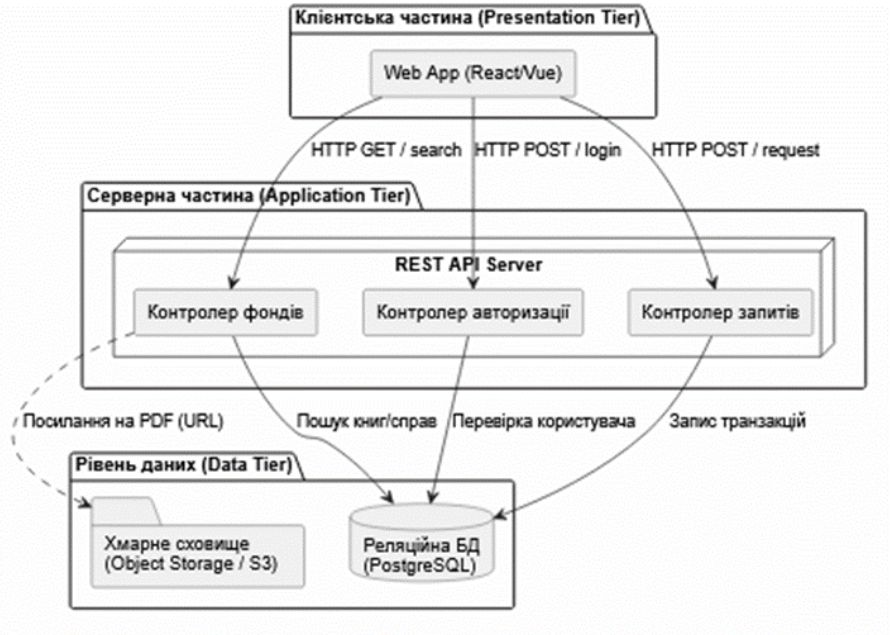


Рисунок 2.3 – Компоненти трирівневої архітектури інформаційної системи

2.2.2 Опис взаємодії компонентів

Запропонована архітектурна модель забезпечує високу масштабованість, гнучкість розробки та підвищену безпеку системи. Взаємодія між рівнями відбувається за таким алгоритмом:

1. Запит користувача: Користувач через веббраузер відкриває інтерфейс застосунку. При спробі знайти архівну справу, клієнтська частина (Frontend) генерує асинхронний HTTP-запит до відповідної кінцевої точки (Endpoint) бекенду, наприклад, GET /api/archive?search=1920.

2. Обробка сервером: Серверний модуль REST API приймає запит. Спеціальний проміжний шар програмного забезпечення (Middleware) перевіряє JWT-токен користувача, щоб підтвердити його рівень доступу. Якщо запит надіслано на закритий архівний фонд, а роль користувача – "Читач", сервер миттєво повертає помилку, не навантажуючи базу даних.

3. Робота з даними: Якщо перевірку пройдено, FundCtrl формує оптимізований SQL-запит до СКБД PostgreSQL. База даних обробляє запит і повертає метадані справи, включаючи унікальне посилання storage_link.

4. Формування відповіді: Бекенд конвертує отримані дані у формат JSON і відправляє назад на фронтенд. Браузер користувача рендерить картку документа, а за потреби перегляду скану – робить прямий захищений запит до файлового сховища Storage за отриманим одноразовим посиланням.

Такий підхід дозволяє уникнути перевантаження сервера бізнес-логики великими файлами сканів та забезпечує швидкість відгуку системи (Response Time) в межах кількох мілісекунд. Реляційні бази даних погано пристосовані для зберігання великих бінарних файлів (BLOB). Зберігання сканів у хмарному Object Storage економить дисковий простір сервера БД, спрощує створення резервних копій (backups) та підвищує загальну швидкість пошуку по текстових полях каталогу[9].

2.3 Проектування бази даних

Проєктування бази даних включає детальну структуру сутностей, опис типів даних, зв'язків та обмежень цілісності. Оскільки застосунок працює і з бібліотечними (структурованими), і з архівними (часто слабкоструктурованими) даними, логічно проектувати реляційну модель.

Проектування реляційної бази даних інформаційної системи грунтується на принципах нормалізації для забезпечення цілісності даних та мінімізації їх надлишковості. Спроектована модель даних приведена до третьої нормальної форми (3NF), оскільки всі неключові атрибути таблиць повною мірою залежать від первинних ключів і між ними відсутні транзитивні залежності.

Концептуальна модель системи складається з 5 основних сутностей: Roles, Users, Library_Funds, Archive_Documents та Document_Requests.

2.3.1. Специфікація сутностей та атрибутів

Кожна сутність (таблиця) має унікальний набір полів, що відповідає логіці управління бібліотекою та архівом:

Сутність Roles (Ролі) використовується як довідник для реалізації рольової моделі доступу. Первинним ключем є id. Поле role_name має тип VARCHAR, що обмежує назву ролі до 50 символів, забезпечуючи оптимальне використання пам'яті.

Сутність Users (Користувачі) зберігає облікові дані. Для захисту інформації паролі користувачів не зберігаються у відкритому вигляді – замість цього передбачено поле password_hash (VARCHAR(255)). Зв'язок із таблицею ролей реалізовано через зовнішній ключ role_id.

Сутність Library_Funds (Бібліотечний фонд) містить метадані друкованих видань. Ключовим атрибутом є унікальний інвентарний номер (inventory_number), для якого встановлено обмеження UNIQUE Key для

запобігання дублювання фізичних книг.

Сутність Archive_Documents (Архівні документи) розроблена для обліку документів, що зберігаються в спецфондах, мають цифрові копії. Атрибут storage_link (VARCHAR(512)) зберігає текстовий шлях (URL) до файлу у хмарному сховищі, що дозволяє не перевантажувати реляційну БД важкими бінарними даними. В кожній бібліотеці архівний фонд може набувати окремих властивостей: наприклад, до архівних фондів іноді відносять стародруки, архівні документи, які зберігаються окремо, мають обмежений доступ, спеціальні умови зберігання (температура, вологість, розмір шаф), тощо. Такі унікальні речі підлягають оцифруванню в першу чергу, тому що вони є об’єктами культурної спадщини.

Сутність Document_Requests (Запити та видача): є центральною таблицею транзакцій (або таблицею фактів). Вона фіксує, який саме користувач (user_id) надіслав запит на конкретну книгу (library_fund_id) або архівний документ (archive_doc_id). Поля зв'язків із фондами мають властивість Nullable, оскільки один запит може стосуватися або лише книги, або лише архівного документа.

2.3.2 Обґрунтування зв'язків між сутностями

Зв'язки між таблицями спроектовані за допомогою класичної нотації т.з. «воронячі лапки» (Crow's Foot) та мають наступну логіку:

Roles – Users (зв'язок 1:M, "один до багатьох"). Один системний рівень доступу (наприклад, "Архіваріус") може бути призначений багатьом користувачам, але кожен окремий користувач в один момент часу володіє лише однією роллю.

Users – Document_Requests (зв'язок 1:M, "один до багатьох"). Один читач або дослідник має право створювати необмежену кількість запитів на видачу документів упродовж усього часу роботи із застосунком.

Library_Funds / Archive_Documents – Document_Requests (зв'язок 1:M,

"один до багатьох"). Фізична книга або унікальний архівний документ може фігурувати в історії запитів системи безліч разів, що дозволяє повністю відстежувати життєвий цикл документа та верифікувати його поточний статус за допомогою поля status.

Опишемо структуру основних таблиць (сутностей) та зв'язків між ними та представимо її на ER-діаграмі (рис.2.4). Позначимо:

<<PK>> – Primary Key (Первинний ключ)

<<FK>> – Foreign Key (Зовнішній ключ)

<<UK>> – Unique Key (Унікальне поле/інвентарний номер)

[AI] – Auto Increment (Автоматичне збільшення ID)

Таблиця Users (Користувачі): id (PK), username, password_hash, email, role_id (FK).

Таблиця Roles (Ролі): id (PK), role_name (Admin, Archivist, Reader).

Таблиця Library_Funds (Книжковий фонд): id (PK), title, author, isbn, pub_year, status, inventory_number.

Таблиця Archive_Documents (Архівні справи): id (PK), fund_number (Номер фонду), inventory_number, case_title (Назва справи), chronological_frames (Датування), storage_link (Шлях до сканованого PDF).

Таблиця Document_Requests (Логи видачі/запитів): id (PK), user_id (FK), document_id (FK, зв'язок з архівом або книгою), request_date, return_date, status (Схвалено, Очікує, Повернено).

Тип зв'язків: Один користувач (Users) може мати багато запитів (Document_Requests) – зв'язок 1:M. Одна книга чи документ може фігурувати в багатьох запитах у часі – зв'язок 1:M.

В ДОДАТКУ А наведено код SQL-скрипта DDl (CREATE TABLE), який створює таблиці БД.

Рисунок 2.4 – Графічна модель ERD (*entity-relationship diagram*) бази даних

2.4. Алгоритм пошуку та безпеки

Важливою частиною проектування є опис логіки ключових процесів. У кваліфікаційній роботі це оформлено у вигляді діаграм діяльності (Activity Diagrams) або текстових алгоритмів:

Алгоритм доступу до архіву:

1. Користувач надсилає запит на документ.
2. Система перевіряє рівень доступу (чи є документ публічним, чи потребує спецдозволу архіваріуса).
3. Якщо документ секретний/рідкісний – система генерує форму запиту для архіваріуса.
4. Після схвалення архіваріусом користувач отримує тимчасове посилання (Token-based URL) на перегляд файлу без можливості прямого скачування (захист авторських прав).

Діаграма діяльності (Activity Diagram) відображає динаміку системи – покроковий алгоритм виконання певного бізнес-процесу з урахуванням розгалужень (умовних операторів).

Для системи управління бібліотечними та архівними даними ключовим процесом є алгоритм обробки запиту на видачу документа з перевіркою рівня доступу.

Наведемо текстовий опис цього алгоритму.

Алгоритм діяльності системи під час обробки запиту користувача на отримання або перегляд документа складається з таких кроків:

1. Ініціалізація процесу: Авторизований користувач (Читач) надсилає запит на отримання доступу до конкретної одиниці зберігання (книги або архівної справи).

2. Перевірка типу документа: Система аналізує, до якого саме фонду належить документ.

Якщо це книга з бібліотечного фонду: Система перевіряє її статус у таблиці Library_Funds. Якщо статус «доступно», книга автоматично бронюється, а запит надходить до бібліотекаря для видачі. Якщо статус «видано», користувач отримує відмову з пропозицією стати в чергу.

Якщо це архівний документ: Система переходить до аналізу політики безпеки та обмежень.

3. Перевірка рівня доступу (Авторизація транзакції): Система аналізує поле access_level архівного документа.

Якщо рівень "відкритий", система миттєво схвалює запит, генерує тимчасове захищене посилання (Token-based URL) до хмарного сховища і відкриває Читачу доступ до перегляду цифрового скану. Процес завершується.

Якщо рівень "службовий" або "закритий", автоматичне схвалення неможливе. Запит перенаправляється в робочу панель Архівіста.

4. Обробка запиту Архівістом (Ручна верифікація): Архівіст розглядає запит, перевіряючи мету дослідження та права Читача.

Якщо він приймає рішення "Затвердити", статус запиту змінюється на «схвалено», Читач отримує доступ до документа (цифрового або фізичного в читальному залі).

Якщо його рішення –"Відхилити", Архівіст вказує причину відмови, система реєструє її у логах, надсилає сповіщення Читачу і завершує процес.

Рисунок 2.5 –Графічна модель діаграми діяльності (Activity Diagram)

Висновки до розділу

У розділі зроблено аналіз предметного середовища з точки зору функціональних вимог. Використані ключові інструменти візуального моделювання в UML, що дозволяють описати логіку та структуру майбутньої програми.

Побудована діаграма прецедентів USE CASE, що визначає функції, поведінку, сервіси та завдання застосунку. Наведено діаграму послідовності Sequence Diagram для процесу авторизації.

Описано трирівневу архітектуру застосунку.

Спроєктовано базу даних у вигляді графічної моделі ERD.

Наведені алгоритми логіки ключових процесів та безпеки (Activity Diagram – діаграма діяльності).

РОЗДІЛ 3 ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

Розділ кваліфікаційної роботи присвячено практичному етапу створення інформаційної системи – безпосередній програмній реалізації застосунку для управління бібліотечними та архівними даними. На основі аналізу предметної області та сформованих у попередніх розділах функціональних вимог, у цьому розділі здійснюється перехід від теоретичних моделей та архітектурних схем до створення діючого програмного продукту.

Ефективність функціонування сучасних бібліотечно-архівних систем критично залежить від швидкості обробки пошукових запитів, надійності збереження великих масивів даних (зокрема, цифрових копій документів) та гнучкості розмежування прав доступу користувачів. З огляду на це, першочерговим завданням є обґрунтований вибір сучасного інструментарію розробки, який забезпечить масштабованість, безпеку та високу продуктивність застосунку.

Ключові завдання розділу:

- обґрунтування та формування технологічного стеку для серверної (Backend) та клієнтської (Frontend) частин системи, а також вибір системи управління базами даних (СУБД);
- деталізація архітектури програмного комплексу та опис структури модулів проєкту;
- програмна реалізація базових алгоритмів: автентифікації користувачів, повнотекстового пошуку в архівних фондах, обліку видачі літератури та збереження медіафайлів;
- рекомендації щодо розробки графічного інтерфейсу користувача та проведення тестування розробленого програмного забезпечення для підтвердження його відповідності технічному завданню.

Реалізація зазначених завдань дозволить створити модулі цілісної, захищеної та зручної у використанні інформаційної системи, спроможної

автоматизувати рутинні процеси обліку та значно прискорити доступ користувачів до архівних ресурсів.

3.1. Обґрунтування вибору технологічного стеку

Розглянемо вибір інструментів реалізації застосунку, які є оптимальними для управління архівами (наприклад, через швидкість розробки, роботу з великими обсягами даних або безпеку)[10,11] – таблиця 3.1.

Таблиця 3.1 – Запропонований варіант вибору інструментів для реалізації застосунку

Модуль застосунку	Інструмент
Серверна частина (Backend)	Варіант 1 (Python / Django або FastApi): Висока швидкість розробки, наявність готових ORM для роботи з базами даних, потужні бібліотеки для обробки документів та PDF (PyPDF2, Pillow). Варіант 2 (Node.js / Express): Асинхронність, висока продуктивність при великій кількості одночасних запитів від користувачів.
Клієнтська частина (Frontend)	React.js / Vue.js: Компонентний підхід дозволяє створити динамічний інтерфейс каталогу з миттєвою фільтрацією та пошуком без перезавантаження сторінок.
Система управління базами даних (СУБД)	PostgreSQL: Потужна реляційна СУБД, що підтримує складні зв'язки (автор-книга-екземпляр) та має вбудовані інструменти повнотекстового пошуку (Full-Text Search), що критично для архівів.
Допоміжні інструменти	Docker (для контейнеризації), Git (для контролю версій).

Для програмної реалізації вебзастосунку управління бібліотечними та архівними даними було проведено аналіз сучасних технологій та інструментів розробки. Головними критеріями вибору були: висока продуктивність під час обробки пошукових запитів, надійність збереження великих масивів інформації, безпека даних та швидкість розробки.

3.1.1. Серверна частина (Backend)

Для реалізації серверної логіки, бізнес-процесів та API застосунку було запропоновано обрати два рішення: мову програмування Python та фреймворк FastAPI або Django REST Framework.[12,13] Порівняно з іншими популярними платформами (наприклад, Node.js або PHP), цей вибір обґрунтовано такими чинниками:

- Висока продуктивність: FastAPI є одним із найшвидших вебфреймворків завдяки підтримці асинхронності (async/await) та використанню серверів Uvicorn і Starlette. Це критично для систем, що обробляють велику кількість одночасних запитів до архіву.
- Автоматична документація: Фреймворк автоматично генерує інтерактивну документацію API (Swagger UI та ReDoc), що значно спрощує інтеграцію з клієнтською частиною.
- Робота з даними та файлами: Python має потужну екосистему бібліотек для обробки цифрових документів, парсингу та конвертації форматів (наприклад, PDF, DOCX, зображень архівних сторінок), що необхідно для наповнення медіатеки.

Нижче наведено порівняльну таблицю між FastAPI та Django REST Framework (DRF), адаптовану під специфіку застосунку для управління бібліотечними та архівними даними (табл. 3.2).

Таблиця 3.2. Порівняльний аналіз фреймворків FastAPI та Django REST Framework (DRF)

Критерій порівняння	FastAPI	Django REST Framework (DRF)	Вплив на бібліотечно-архівний застосунок
Архітектурний підхід	Мікрофреймворк. Надає лише базовий інструментарій, інше підключається модульно.	Монолітний («все включено»). Має вбудовані інструменти для всього циклу розробки.	DRF має готову адмін-панель, яка ідеально підходить для робочого місця бібліотекаря без написання додаткового коду.
Продуктивність та швидкість	Екстремально висока. Базується на асинхронності (ASGI, Starlette) та є одним із найшвидших фреймворків.	Середня. Класична синхронна архітектура (WSGI), яка поступається у швидкості при високих навантаженнях.	FastAPI краще підійде, якщо в архіві планується масове одночасне завантаження, сканування або завантаження важких PDF-файлів.
Валідація та типізація даних	Вбудована та автоматична завдяки бібліотеці Pydantic та сучасній типізації Python.	Реалізована через механізм Серіалізаторів (Serializers), що вимагає написання більшої кількості коду.	FastAPI автоматично перевіряє правильність заповнення карток книг (наприклад, формат року, наявність інвентарного номера).
Документування API	Автоматичне. Інтерактивна	Вимагає встановлення та	FastAPI спрощує інтеграцію з

	документація (Swagger UI, ReDoc) доступна одразу «з коробки».	налаштування сторонніх бібліотек (наприклад, drf-yasg або spectacular).	вашим фронтендом на React.js, оскільки розробник бачить усі ендпоінти автоматично.
Робота з базою даних (ORM)	Не має власної ORM. Зазвичай інтегрується з SQLAlchemy або Tortoise ORM.	Має потужну вбудовану Django ORM з підтримкою складних зв'язків та міграцій.	DRF виграє у швидкості побудови складних реляційних зв'язків (наприклад, Книга → Автор → Жанр → Історія видачі).
Адмін-панель	Відсутня. Потребує розробки з нуля на фронтенді або підключення складних сторонніх бібліотек.	Потужна вбудована адмін-панель, яка готова до роботи одразу після створення моделей даних.	DRF дозволяє з коробки отримати готовий інтерфейс для архіваріуса для внесення нових документів та керування користувачами.

Монолітна структура Django REST Framework та наявність готової адміністративної панелі дозволяють суттєво скоротити час на розробку кабінету бібліотекаря, а потужна вбудована ORM забезпечує надійне керування складними зв'язками між архівними фондами.

Вибір зумовлений потребою в асинхронній архітектурі для швидкої обробки великої кількості одночасних запитів користувачів до цифрового каталогу, високою швидкістю передачі медіафайлів (скан-копій документів)

та автоматичною валідацією даних за допомогою Pydantic – аргументи на користь FastAPI.

Після проведення порівняльного аналізу, для реалізації системи управління бібліотечними архівними даними було обрано Django REST Framework.

3.1.2. Клієнтська частина (Frontend)

Для розробки користувацького інтерфейсу обрано бібліотеку React.js. Вибір React.js замість класичного підходу (рендеринг сторінок на сервері) або використання інших фреймворків (Angular, Vue.js) зумовлений такими перевагами:

- Компонентно-орієнтована архітектура: Дозволяє створити універсальні та перевикористовувані елементи інтерфейсу (картки книг, форми пошуку, таблиці користувачів), що полегшує підтримку та розширення коду.
- Технологія Virtual DOM: Забезпечує миттєве оновлення інтерфейсу без перезавантаження всієї сторінки. Це важливо під час динамічного очищення фільтрів або пагінації у великих каталогах.
- Широка екосистема: Наявність готових бібліотек компонентів (наприклад, Tailwind CSS або Material-UI) дозволяє швидко створити адаптивний та сучасний дизайн для архіваріусів та читачів.

3.1.3. Система управління базами даних (СУБД)

Як основне сховище даних обрано реляційну СУБД PostgreSQL. Оскільки бібліотечні та архівні дані мають чітку структуру та складні внутрішні зв'язки (наприклад: один автор написав багато книг; один примірник може бути виданий лише одному читачу одночасно), використання NoSQL-рішень (наприклад, MongoDB) є недоцільним. Переваги PostgreSQL для цього проєкту:

- Підтримка ACID (атомарність, узгодженість, ізольованість, довговічність): Гарантує 100% надійність транзакцій, що виключає помилки під час фіксації видачі або повернення архівних документів.
- Вбудований повнотекстовий пошук (Full-Text Search): Дозволяє реалізувати швидкий та релевантний пошук за назвами, авторами чи ключовими словами без підключення важких сторонніх систем типу Elasticsearch.
- Робота з JSON-даними: Дозволяє гнучко зберігати специфічні метадані архівних документів (які можуть відрізнятися залежно від типу фонду), поєднуючи переваги реляційних та нереляційних баз.

3.1.4. Допоміжні інструменти розробки

Сучасний процес розробки програмного забезпечення вимагає використання додаткових інструментів автоматизації, контролю та контейнеризації. У межах цього проєкту було інтегровано систему контролю версій Git та платформу контейнеризації Docker.

- Git: Система контролю версій для відстеження змін у коді та спільної розробки.

Використання Git є обов’язковим стандартом для керування вихідним кодом застосунку. У цьому проєкті інструмент вирішує такі ключові завдання:

Фіксація та відстеження змін: Кожен етап розробки (створення моделей БД, написання JWT-авторизації, розробка інтерфейсу React) фіксується за допомогою окремих комітів (commits). Це дозволяє бачити історію розвитку проєкту та за потреби повернутися до стабільної версії коду.

Розгалуження (Branching): Створення нових функцій (features) відбувалося в ізольованих гілках (наприклад, feature/auth, feature/search). Це унеможливило появу помилок у вже працюючому коді головної гілки (main) під час експериментів із новими алгоритмами.

Інтеграція з хмарними репозиторіями: Код проєкту зберігається на віддаленому сервері (GitHub/GitLab), що гарантує захист від втрати даних у разі виходу з ладу локального комп’ютера розробника.

- Docker: Технологія контейнеризації, яка використовується для ізоляції застосунку та швидкого розгортання бази даних і сервера в будь-якому середовищі без конфліктів залежностей.

Docker використовується для ізоляції компонентів застосунку (Backend, Frontend, СУБД PostgreSQL) у незалежні віртуальні контейнери, які ділять між собою ядро однієї операційної системи[15].

Для системи управління архівами Docker є критично важливим через такі чинники:

- Вирішення проблеми «Працює лише на моєму комп’ютері»: Docker гарантує, що застосунок, розроблений на локальній машині (наприклад, під керуванням Windows або macOS), запуститься та працюватиме абсолютно ідентично на демонстраційному сервері чи комп’ютері членів екзаменаційної комісії. Контейнер уже містить у собі правильні версії Python, Node.js, PostgreSQL та всі необхідні залежності.
- Швидке розгортання СУБД: Замість тривалого ручного встановлення та налаштування сервера PostgreSQL на комп'ютер, Docker дозволяє розгорнути готову та сконфігуровану базу даних однією командою.
- Оркестрація за допомогою Docker Compose: Оскільки застосунок складається з трьох окремих частин (сервер, клієнт, база даних), у проєкті створено конфігураційний файл docker-compose.yml. Він дозволяє запустити весь програмний комплекс (зв'язавши фронтенд із бекендом, а бекенд із БД) натисканням однієї кнопки.

Приклад файлу конфігурації надано у ДОДАТКУ В.

3.2. Архітектура та структура програмного комплексу

Для забезпечення гнучкості, масштабованості та простоти підтримки застосунку управління бібліотечними архівними даними було обрано розділену архітектуру (Decoupled Architecture) на основі архітектурного стилю REST API. Програмний комплекс чітко розділений на два незалежні компоненти[16]:

1. Backend (Серверна частина): Реалізований на базі Django REST Framework. Він відповідає за логіку, безпеку, взаємодію з базою даних PostgreSQL та надання RESTful-ендпоінтів.
2. Frontend (Клієнтська частина): Може бути реалізований на базі React.js. Він відповідає за візуалізацію даних, взаємодію з користувачем та надсилання асинхронних запитів до сервера.

3.2.1. Структура каталогів проєкту

Проєкт має модульну структуру, що є стандартною практикою для розробки на Django та React (рис.3.1).

```
library_archive_project/
│
├── backend/                          # Серверна частина (Django)
│   ├── manage.py                     # Скрипт керування проєктом Django
│   ├── core/                         # Головна конфігураційна директорія проєкту
│   │   ├── __init__.py
│   │   ├── settings.py               # Основні налаштування (БД, JWT, медіа)
│   │   ├── urls.py                   # Головний маршрутизатор API
│   │   └── wsgi.py / asgi.py
│   │
│   ├── archive_app/                  # Модуль управління архівними фондами
│   │   ├── models.py                 # Опис структури таблиць (Книги, Документи, Автори)
│   │   ├── serializers.py            # Перетворення моделей Django в JSON і навпаки
│   │   ├── views.py                  # Логіка обробки REST-запитів (API ViewSets)
│   │   ├── urls.py                   # Логіка маршрутів для цього модуля
│   │   └── admin.py                  # Налаштування вбудованої адмін-панелі
│   │
│   └── users_app/                    # Модуль користувачів та автентифікації
│       ├── models.py                 # Кастомна модель користувача (Читач, Бібліотекар)
│       ├── serializers.py            # Серіалізатори для профілів та реєстрації
│       └── views.py                  # Логіка авторизації та видачі JWT-токенів
│
└── frontend/                         # Клієнтська частина (React.js)
    ├── package.json                  # Конфігурація проєкту та залежностей
    ├── public/                       # Статичні файли (іконки, index.html)
    └── src/                          # Вихідний код SPA-застосунку
        ├── assets/                   # Стилі (CSS), логотипи, зображення
        ├── components/               # Перевикористовувані компоненти (Картки, Кнопки)
        ├── context/                  # Збереження стану (наприклад, AuthContext для JWT)
        ├── services/                 # Модулі для HTTP-запитів (Axios сервіси для API)
        ├── pages/                    # Сторінки (Каталог, Кабінет, Авторизація)
        ├── App.js                    # Головний компонент та налаштування маршрутів (React Router)
        └── index.js                  # Точка входу в React-застосунок
```

Рисунок 3.1 – Дерево каталогів програмного комплексу

3.2.2. Схема взаємодії компонентів системи

Взаємодія між клієнтом (React) та сервером (DRF) відбувається асинхронно через HTTP-протокол із використанням форматів обміну даними JSON (JavaScript Object Notation)[17]. Типовий життєвий цикл запиту в системі виглядає так:

1. Ініціація дії: Користувач у браузері відкриває сторінку каталогу та вводить у пошуковий рядок назву книги, наприклад, "Історія України".

2. Надсилання запиту: Перехоплюючи введення, клієнтський сервіс services/api.js (за допомогою бібліотеки axios) надсилає асинхронний GET запит до бекенду на адресу: https://library.local.

3. Маршрутизація та автентифікація: URL-маршрутизатор Django перенаправляє запит на відповідний BookViewSet. Класи безпеки (Permissions)

перевіряють JWT-токен у заголовку запиту, визначаючи рівень доступу користувача.

4. Взаємодія з БД (ORM): Контролер через Django ORM робить оптимізований SQL-запит до СУБД PostgreSQL, здійснюючи фільтрацію та пошук у таблиці книг.

5. Серіалізація: Знайдені об'єкти з бази даних передаються у BookSerializer, який трансформує складні об'єкти мови Python у плоский формат JSON.

6. Формування відповіді: Сервер повертає HTTP-відповідь зі статусом 200 OK та JSON-масивом знайдених книг.

7. Рендеринг інтерфейсу: React отримує дані, оновлює внутрішній стан компонента (state), і технологія Virtual DOM миттєво перемальовує картки книг на екрані користувача без перезавантаження сторінки.

3.3. Реалізація основних модулів та алгоритмів системи

В цій частині розділу наводяться фрагменти коду ключових модулів застосунку для управління бібліотечними архівними даними.

3.3.1. Модуль автентифікації та розмежування прав доступу

Модуль автентифікації та розмежування прав доступу реалізує безпеку, система має відрізняти звичайного читача від архіваріуса.

Для захисту персональних даних користувачів, забезпечення конфіденційності архівних фондів та розмежування прав доступу в інформаційній системі реалізовано безсесійний (stateless) механізм автентифікації за допомогою JWT (JSON Web Tokens).

Використання JWT-токенів замість класичних сесій у базі даних дозволяє знизити навантаження на СУБД PostgreSQL, оскільки вся необхідна

інформація про користувача та його роль зашифрована всередині самого токена.

Алгоритм роботи механізму автентифікації в системі.

1. Користувач вводить логін і пароль у формі авторизації на фронтенді (React.js).

2. Запит POST /api/v1/token/ відправляється на сервер.

3. Django REST Framework перевіряє правильність облікових даних. Якщо вони коректні, генерується пара токенів:

- Access Token: Короткотривалий токен (час життя – 30 хвилин) для автентифікації кожного HTTP-запиту.
- Refresh Token: Довготривалий токен (час життя – 7 днів) для безпечного оновлення Access-токена без повторного введення пароля.

4. React-застосунок зберігає отримані токени в пам'яті (State) або захищених файлах кукі (HttpOnly Cookies) і прикріплює Access Token до заголовка Authorization: Bearer <token> для всіх наступних запитів.

Для реалізації цього функціоналу на бекенді використано пакет djangorestframework-simplejwt. Програмний код кастомного серіалізатора, який додає інформацію про роль користувача (читач, бібліотекар, архіваріус) безпосередньо в тіло JWT-токена наведено у ДОДАТКУ Г.1.

Для обмеження доступу до певних дій (наприклад, додавання нової книги до архіву або редагування картки документа) створено клас розмежування прав доступу (Permissions). Фрагмент реалізації контролера (View) із захистом ендпоінтів наведено у ДОДАТКУ Г.1.

На стороні клієнтського застосунку (React.js) створено інтерцептор (Interceptor) для бібліотеки Axios. Він автоматично підставляє JWT-токен у заголовки кожного запиту. Якщо сервер повертає помилку 401 Unauthorized (термін дії Access-токена закінчився), перехоплювач самостійно відправляє запит на оновлення токена за допомогою Refresh-токена, забезпечуючи безперервну роботу користувача із системою.

3.3.2. Алгоритм повнотекстового пошуку та фільтрації документів

Швидкість та точність пошуку інформації є головними критеріями ефективності системи управління бібліотечно-архівними даними. Стандартний пошук за допомогою оператора LIKE в SQL-запитах (contains в Django ORM) є неефективним для великих масивів текстових даних, оскільки він виконує повне сканування таблиці (Full Table Scan) та не враховує морфологію мови (відмінки, закінчення слова)[18].

Для вирішення цієї проблеми в інформаційній системі реалізовано повнотекстовий пошук (Full-Text Search, FTS) на рівні СУБД PostgreSQL засобами вбудованого модуля django.contrib.postgres.

Принцип роботи алгоритму повнотекстового пошуку:

1. Токенізація та нормалізація: Текстові поля (назва документа, анотація, ключові слова) розбиваються на окремі слова та приводяться до єдиного регістра.
2. Стемінг (Stemming): Видаляються префікси та закінчення для пошуку за конем слова (наприклад, слова "архівний", "архіву", "архіви" зводяться до однієї лексеми).
3. Векторизація (SearchVector): Створюється пошуковий вектор текстового вмісту картки документа.
4. Ранжування (SearchRank): Результати пошуку сортуються за релевантністю. Збіг у назві документа має вищу вагу (пріоритет), ніж збіг у тексті довгого опису чи анотації.

Поряд із текстовим пошуком реалізовано модуль багатофакторної фільтрації за категоріями: роком видання, автором та унікальним архівним шифром зберігання. Програмний код реалізації цього алгоритму в контролері Django REST Framework наведено у ДОДАТКУ Г.2.

Під час виконання пошукового запиту Django ORM транслює цей код у високоефективний SQL-запит до бази даних PostgreSQL. Нижче наведено приклад згенерованого SQL-запиту до СУБД PostgreSQL для книги з п. 3.2.2:

```
SELECT id, title, publication_year,
    ts_rank_cd(
        to_tsvector('ukrainian', coalesce(title, '')) ||
        to_tsvector('ukrainian', coalesce(description, '')),
        to_tsquery('ukrainian', 'історія')
    ) AS rank
FROM archive_documents
WHERE to_tsvector('ukrainian', coalesce(title, '')) @@ to_tsquery('ukrainian', 'історія')
ORDER BY rank DESC;
```

На фронтенді (React.js) для користувача цей алгоритм реалізовано у вигляді єдиного пошукового рядка з технологією Debouncing (затримка відправки запиту на 300 мс під час введення тексту). Це дозволяє не навантажувати сервер зайвими запитами на кожну введену літеру, забезпечуючи плавність роботи інтерфейсу.

3.3.3. Модуль завантаження та збереження цифрових копій

Архіви вимагають збереження скан-копій (PDF, JPEG). Опишемо, як реалізовано завантаження файлів, їх валідацію за розміром/форматом та збереження на сервері (або у хмарі).

У системах управління архівами критично важливою функцією є збереження та відображення електронних (оцифрованих) копій першоджерел: рідкісних книг, історичних рукописів, наказів чи періодичних видань. Модуль завантаження медіафайлів відповідає за приймання цифрових копій від архіваріуса, їх сувору валідацію за типом і розміром, автоматичну генерацію унікальних імен для запобігання конфліктам та безпечне збереження на сервері.

Основні технічні та безпекові вимоги до модуля:

- Обмеження за форматами: Дозволяється завантаження лише фіксованих форматів документів (переважно PDF для текстових фоліантів) та графічних зображень високої якості (JPEG, PNG) для посторінкових скан-копій.
- Контроль розміру файлів: Обмеження максимального обсягу файлу на рівні сервера (наприклад, до 50 МБ), щоб уникнути переповнення дискового простору та перевантаження каналу зв'язку.
- Ізоляція файлової структури: Файли мають структуруватися за категоріями та роками розгортання архіву. Напряму звертатися до файлів в обхід системи автентифікації заборонено для забезпечення конфіденційності.

Для реалізації цього функціоналу в Django REST Framework було розроблено спеціальну функцію динамічної генерації шляхів збереження та валідатор файлів. Програмна реалізація логіки обробки та валідації файлів надана у ДОДАТКУ Г.3.

Для того, щоб клієнтська частина на React.js могла завантажувати файли, стандартний формат JSON не підходить, оскільки він оперує лише текстом. Обмін даними відбувається за допомогою об'єкта FormData, який дозволяє передавати бінарні дані (файли) через HTTP multipart/form-data запити.

Нижче представлено фрагмент налаштування контролера (View), який забезпечує коректну обробку мультимедійних запитів: підключає парсери для обробки завантаження файлів з форми, проводить аудит дій, тощо.

Фрагмент контролера поданий у ДОДАТКУ Г.3.

На клієнтській стороні (React.js) цей модуль представлений у кабінеті архіваріуса у вигляді інтерактивного поля «Drag-and-Drop» (перетягування файлу мишкою). Перед безпосереднім відправленням файлу на сервер клієнтський скрипт також виконує первинну перевірку розміру файлу, щоб моментально попередити користувача та зекономити трафік у разі помилки.

3.4. Інтерфейс користувача та тестування працездатності

3.4.1. Схема та інформаційна архітектура графічного інтерфейсу (UI/UX)

Проєктування інтерфейсу користувача виконано за принципом SPA (Single Page Application), що забезпечує миттєву зміну контенту без перезавантаження сторінок браузера. В основі UI/UX лежить чітка ієрархія та дворівнева навігація, яка адаптується під роль авторизованого користувача. Застосунок має модульну структуру екранів, логічний взаємозв'язок між якими відображено на схемі нижче: (рис. 3.2).

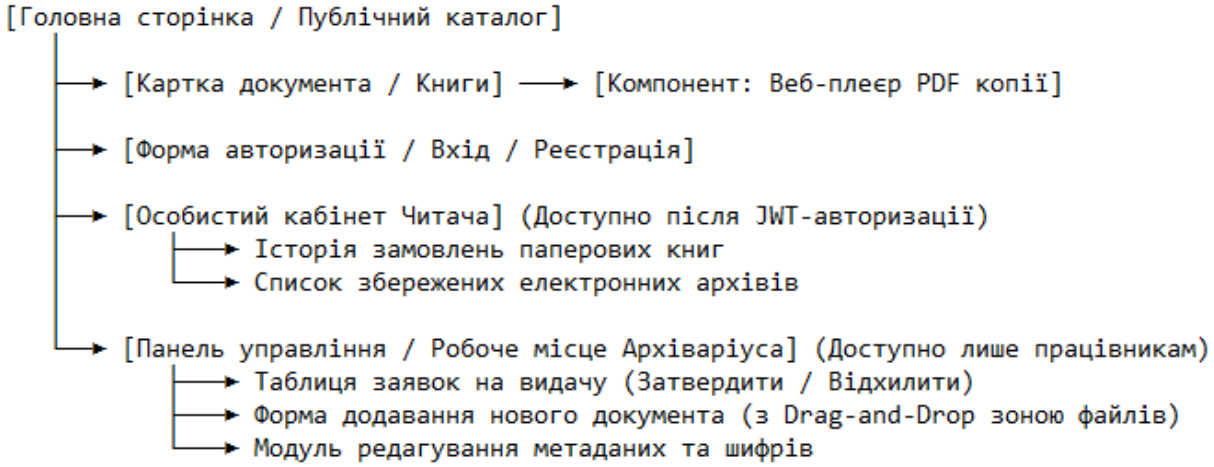


Рисунок 3.2 – Структурна схема сторінок застосунку

Для забезпечення консистентності дизайну всі сторінки системи використовують єдину сітку розташування елементів (Grid Layout). Компонент Navigation Bar (Верхня панель) присутній на всіх сторінках. Містить логотип бібліотеки, посилання на каталог, індикатор статусу авторизації. Якщо користувач є працівником архіву, з'являється кнопка швидкого переходу до «Панелі управління».

Екран перегляду архівного документа містить детальну інформаційну картка (повний опис, історія створення, ключові слова, інвентарний номер) та

інтегрований фрейм (Iframe) для безпечного посторінкового перегляду PDF-файлу без можливості несанкціонованого скачування в обхід ліцензії, або кнопка «Надіслати запит на читання», якщо документ має обмежений доступ.

3.4.2. Тестування працездатності системи

Для підтвердження надійності, безпеки та відповідності розробленого продукту технічному завданню було проведено тестування, яке відноситься до першого рівня – модульного тестування (Unit Testing).

Модульні тести призначені для перевірки ізольованої логіки окремих функцій бекенду без залучення реальної бази даних. Тестування реалізовано за допомогою стандартного фреймворку django.test та бібліотеки pytest.

У межах проєкту було протестовано алгоритм валідації завантажених файлів (перевірка реакції системи на заборонені формати файлів та файли розміром понад 50 МБ) (код наведено у ДОДАТКУ Г.4).

Висновки до розділу

У третьому розділі кваліфікаційної роботи було порівняно різні сучасні технології та обрано програмне забезпечення для реалізації застосунку:

принцип розділеної архітектури (Decoupled Architecture) з використанням REST API.

для серверної частини - фреймворк Django REST Framework (Python)

сховище даних СУБД PostgreSQL.

Програмно реалізовано ключові модулі системи, серед яких:

- Модуль автентифікації на основі JWT-токенів (Access/Refresh), розмежування прав доступу на базі рольової моделі («Читач», «Бібліотекар», «Архіваріус»), що надійно захищає конфіденційні архівні фонди від несанкціонованого редагування чи видалення;

- Реалізовано алгоритм повнотекстового пошуку засобами вбудованих модулів PostgreSQL, що враховує морфологію української мови та ранжує результати за релевантністю;
- Модуль динамічної валідації та безпечного завантаження цифрових копій першоджерел (у форматі PDF та графічних зображень) розміром до 50 МБ.
- Проведено обмежене тестування системи, яке містило модульні (Unit) тести для перевірки ізольованих функцій-валідаторів.

РОЗДІЛ 4 ОХОРОНА ПРАЦІ

4.1 Організаційно-правові основи забезпечення безпеки праці

Система охорони праці в Україні базується на комплексі законодавчих, організаційних та профілактичних заходів, спрямованих на забезпечення безпечних і здорових умов праці для працівників різних сфер діяльності. У сучасних умовах цифровізації бібліотечної та архівної справи питання безпеки праці набувають особливого значення, оскільки працівники дедалі більше взаємодіють із комп’ютерною технікою, електронними базами даних, цифровими архівами та автоматизованими інформаційними системами.

Правову основу системи охорони праці в Україні формує Конституція України, яка гарантує кожному працівникові право на належні, безпечні та здорові умови праці. Конституційні положення визначають пріоритет життя та здоров’я людини і покладають на державу обов’язок забезпечувати захист працівників у процесі трудової діяльності [19, 20].

Ключовим нормативно-правовим актом у сфері безпеки праці є Закон України «Про охорону праці». Цей закон встановлює основні принципи державної політики у сфері охорони праці, визначає права та обов’язки роботодавців і працівників, а також регулює питання організації безпечних умов праці на підприємствах та в установах.

Відповідно до законодавства роботодавець повинен:

- створювати безпечні умови праці;
- забезпечувати належний технічний стан обладнання;
- організовувати навчання та інструктажі;
- контролювати стан виробничого середовища;
- забезпечувати дотримання санітарних норм;
- проводити оцінку професійних ризиків.

Важливе значення для регулювання трудових відносин має Кодекс законів про працю України, який визначає вимоги щодо режиму праці та відпочинку, тривалості робочого часу, захисту прав працівників та порядку організації трудового процесу.

Для бібліотечної та архівної сфери особливу роль відіграють державні санітарні норми та правила, які регулюють умови роботи з комп'ютерною технікою та організацію офісних робочих місць.

Окреме значення для бібліотечних установ мають нормативні документи у сфері пожежної безпеки. Архівні та бібліотечні фонди містять велику кількість паперових носіїв інформації, тому до таких приміщень висуваються підвищені вимоги щодо протипожежного захисту, електробезпеки та організації евакуації.

У бібліотечній та архівній діяльності також враховуються вимоги щодо забезпечення належного санітарного стану приміщень. Робота з паперовими документами та архівними матеріалами може супроводжуватися накопиченням пилу, що впливає на якість повітря та самопочуття працівників. Саме тому нормативні документи передбачають регулярне прибирання приміщень, контроль вологості та підтримання належного мікроклімату.

Сучасна система охорони праці в Україні значною мірою формується під впливом міжнародних стандартів та європейських підходів до забезпечення безпеки працівників. Важливу роль у цьому процесі відіграє Міжнародна організація праці (МОП), яка розробляє міжнародні конвенції та рекомендації щодо захисту працівників, організації безпечних умов праці та профілактики професійних ризиків [19, 20].

Україна є учасником багатьох міжнародних ініціатив у сфері охорони праці та поступово адаптує національне законодавство до європейських стандартів. У процесі євроінтеграції в українське законодавство впроваджується ризик-орієнтований підхід, який передбачає не лише

реагування на нещасні випадки, а й систематичне попередження небезпечних ситуацій [21, 22].

Європейські стандарти охорони праці приділяють значну увагу безпеці офісної та інформаційної роботи. Такі підходи є особливо актуальними для фахівців з управління бібліотечними архівними даними, діяльність яких пов’язана з тривалою роботою за комп’ютером, високим інформаційним навантаженням та необхідністю постійної концентрації уваги.

Сучасні міжнародні практики також сприяють розвитку систем управління охороною праці на підприємствах та в установах. Основна увага приділяється профілактиці професійних ризиків, удосконаленню організації робочих місць та створенню безпечного й комфортного робочого середовища.

4.2 Характеристика об’єкта та виявлення потенційних небезпек

Фахівець з електронного архівування бібліотечних даних виконує роботи, пов’язані з веденням, систематизацією, обробкою та збереженням цифрових архівних матеріалів бібліотеки. Основним завданням такого працівника є організація електронного обліку архівних документів, підтримання актуальності цифрових баз даних та забезпечення зручного доступу до архівної інформації.

У сучасних бібліотечних установах значна частина архівних матеріалів переводиться у цифровий формат. Саме тому діяльність фахівця поєднує роботу з паперовими архівами, сканувальним обладнанням, електронними каталогами та спеціалізованими інформаційними системами. Працівник виконує оцифрування документів, перевіряє правильність внесення даних, формує електронні архіви, працює з базами даних та контролює цілісність інформації.

Робота має переважно інтелектуальний та аналітичний характер і потребує високої уважності, точності та здатності тривалий час концентрувати

увагу на однотипних операціях. Особливої уваги потребує робота з великим обсягом інформації та архівними документами, де навіть незначні помилки можуть призводити до втрати або спотворення даних.

Робоче місце фахівця з електронного архівування бібліотечних даних зазвичай організовується у службових приміщеннях бібліотеки або архівного відділу та належить до автоматизованих офісних робочих місць.

До складу робочого місця входять:

- персональний комп'ютер;
- монітор;
- клавіатура та комп'ютерна миша;
- сканер;
- принтер;
- засоби зберігання інформації;
- мережеве обладнання;
- архівні полиці та паперові носії інформації;
- програмне забезпечення для управління електронними архівами.

Працівник більшу частину робочого часу проводить за комп'ютером у сидячому положенні. Значна частина роботи пов'язана з переглядом електронних документів, введенням інформації, перевіркою архівних записів та роботою зі сканованими матеріалами.

Робочий день зазвичай має стандартний офісний режим із регламентованими перервами. Однак через великий обсяг інформації та необхідність тривалої концентрації уваги працівник може зазнавати значного психофізіологічного навантаження навіть за відсутності фізично важкої роботи.

Для забезпечення комфортних умов праці важливе значення мають:

- правильне освітлення;
- ергономічне розташування обладнання;
- підтримання належного мікроклімату;

- достатня вентиляція приміщення;

- організація режиму праці та відпочинку.

Аналіз небезпечних та шкідливих факторів дає можливість згрупувати їх за наступними групами [21].

1. Психофізіологічні фактори.

Одним із основних факторів ризику є значне інформаційне навантаження. Працівник постійно працює з великим обсягом цифрової інформації, архівних записів та електронних документів. Робота потребує високої точності та уважності, оскільки помилки під час внесення або обробки даних можуть впливати на цілісність електронного архіву.

Додатковим фактором є монотонність праці. Багато операцій мають повторюваний характер, що може призводити до зниження концентрації уваги, перевтоми та психоемоційного виснаження.

Під час тривалої роботи з інформацією також може виникати психоемоційне перенапруження, особливо у разі великого обсягу роботи або необхідності виконання термінових завдань.

2. Ергономічні фактори.

Через постійну роботу за комп’ютером працівник тривалий час перебуває у сидячому положенні. Це створює статичне навантаження на м’язи спини, шиї та плечового поясу. За неправильної організації робочого місця можуть виникати болі у спині, порушення постави та м’язова втома.

Суттєвим фактором є навантаження на органи зору. Працівник тривалий час переглядає текстову інформацію, електронні каталоги та скановані документи, що може викликати втому очей, подразнення слизової оболонки та головний біль.

3. Фізичні фактори.

У бібліотечних та архівних приміщеннях можуть виникати несприятливі параметри мікроклімату. Через велику кількість паперових носіїв інформації

у повітрі може накопичуватися пил, який негативно впливає на органи дихання та загальне самопочуття працівників.

Недостатня вентиляція, сухе повітря або підвищена температура також можуть погіршувати умови праці та сприяти швидкій втомі.

Джерелом додаткового шуму можуть бути системи вентиляції, сканувальне обладнання та офісна техніка.

4. Електричні та технічні фактори.

Під час роботи використовується комп'ютерна та офісна техніка, тому існує ризик ураження електричним струмом у разі несправності обладнання або порушення правил експлуатації.

Також можливе виникнення короткого замикання або перегріву техніки, що створює пожежну небезпеку, особливо в приміщеннях із великою кількістю паперових архівних матеріалів.

4.3 Дослідження ризику реалізації потенційних небезпек на об'єкті проектування та розробка заходів щодо їх попередження

У сучасній системі охорони праці оцінка ризиків розглядається як один із головних механізмів попередження небезпечних ситуацій та збереження здоров'я працівників. Її основне призначення полягає не лише у виявленні небезпек, а й у розумінні того, наскільки серйозними можуть бути наслідки впливу різних факторів виробничого середовища на працівника.

Оцінка ризиків дозволяє підприємству не лише виявляти проблемні аспекти організації праці, а й визначати пріоритетність заходів щодо їх усунення [23, 24]. Це дає можливість більш ефективно організовувати робочі процеси, раціонально використовувати ресурси та підтримувати стабільну роботу установи.

Важливою особливістю сучасного підходу до оцінки ризиків є орієнтація на профілактику. Якщо раніше система охорони праці переважно реагувала на

вже виниклі аварії або нещасні випадки, то сьогодні основна увага приділяється попередженню потенційних небезпек ще до виникнення негативних наслідків.

Якісно проведена оцінка ризиків має декілька характерних ознак [23, 24]. Насамперед вона повинна бути системною та охоплювати всі етапи трудового процесу. Аналіз має враховувати не лише очевидні небезпеки, а й фактори, які поступово впливають на здоров'я працівника, наприклад психоемоційне навантаження, втому або несприятливі ергономічні умови.

Важливою ознакою якісної оцінки є її об'єктивність. Аналіз ризиків повинен базуватися на реальних умовах праці, технічному стані обладнання, особливостях робочого середовища та фактичному характері виконуваних робіт. Формальний підхід до оцінювання не дозволяє своєчасно виявляти дійсні проблеми та знижує ефективність системи охорони праці.

Ефективна оцінка ризиків також повинна бути практично орієнтованою. Її результатом мають ставати конкретні рекомендації щодо покращення умов праці, усунення небезпечних факторів або зниження рівня їх впливу на працівників.

Ще однією важливою ознакою є регулярність проведення оцінювання. Умови праці, технічне оснащення та організація робочих процесів можуть змінюватися, тому оцінка ризиків повинна періодично оновлюватися та враховувати нові фактори.

Для оцінювання професійних ризиків застосовуються різні методи [23, 24]. Одним із найбільш поширених є метод матриці ризиків. Він дозволяє визначати рівень ризику шляхом поєднання ймовірності виникнення небезпечної ситуації та тяжкості можливих наслідків. Такий метод є зручним, наочним та підходить для більшості сфер діяльності, зокрема для офісної та інформаційної роботи.

Досить ефективним методом також вважається метод «дерево відмов». Його перевага полягає у можливості графічно відобразити взаємозв'язки між

різними причинами виникнення небезпечної ситуації. Це дозволяє детально аналізувати складні процеси та визначати ключові фактори ризику (рис. 4.1).

Для аналізу умов праці часто використовуються контрольні списки та експертні методи оцінювання. Вони дозволяють систематизувати інформацію про небезпечні фактори та швидко визначати проблемні ділянки організації праці.

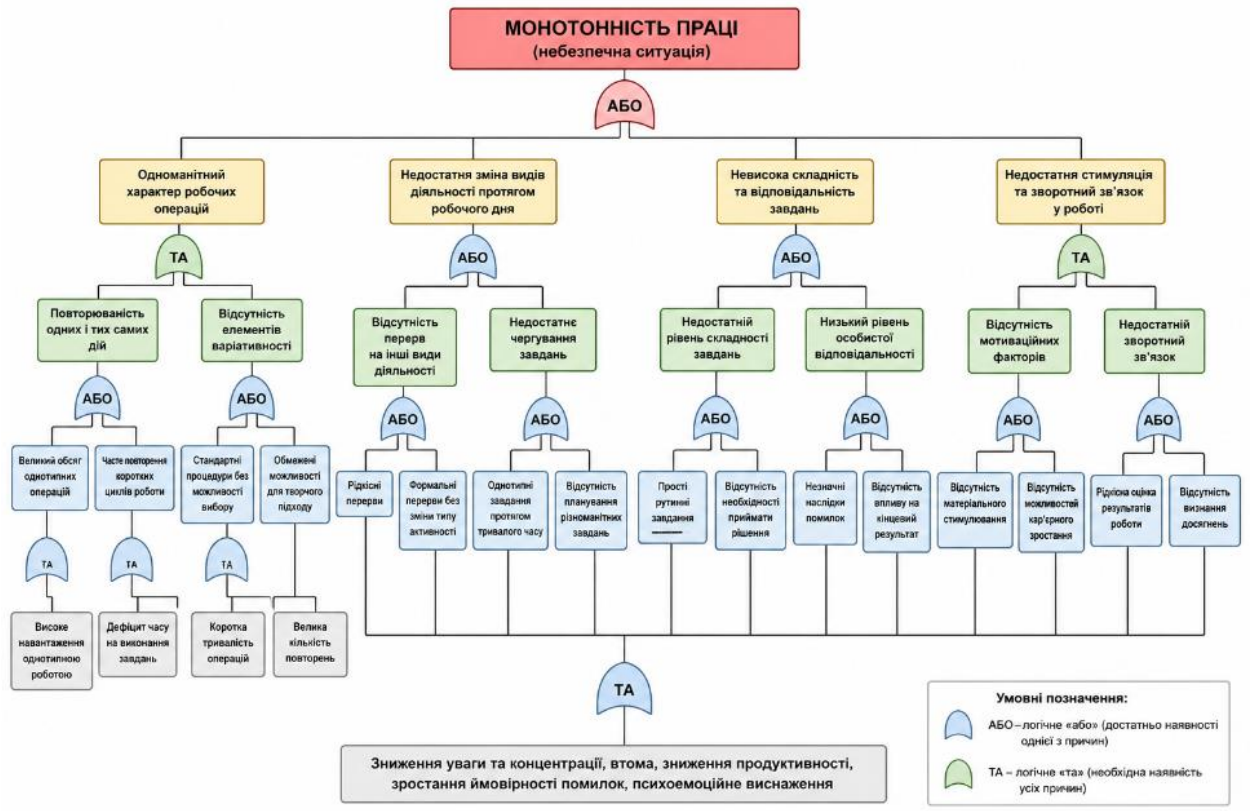


Рисунок 4.1 – Дерево відмов для ситуації, пов’язаної із монотонністю праці

Для зменшення негативного впливу монотонності праці необхідно правильно організовувати робочий процес та забезпечувати чергування різних видів діяльності. Постійне виконання однотипних операцій протягом тривалого часу призводить до швидкої втоми, зниження концентрації уваги та погіршення працездатності, тому важливо створювати умови для періодичної зміни характеру роботи.

Одним із найбільш ефективних заходів є чергування робочих завдань. Працівнику доцільно поєднувати роботу з електронними архівами, перевірку документів, сканування матеріалів та інші види діяльності, що дозволяє зменшити одноманітність роботи.

Важливу роль відіграє організація регулярних перерв протягом робочого дня. Короткочасний відпочинок дозволяє знизити психоемоційне напруження, відновити концентрацію уваги та попередити перевтому.

Для покращення умов праці доцільно підтримувати достатній рівень рухової активності. Навіть нетривала зміна положення тіла або легка розминка допомагають зменшити втому та покращити самопочуття працівника.

Позитивний вплив має також раціональний розподіл робочого навантаження. Необхідно уникати тривалого виконання однотипних операцій без зміни діяльності або відпочинку.

Для зниження психологічної втоми важливо підтримувати комфортну робочу атмосферу та сприятливий психологічний клімат у колективі. Спілкування між працівниками та можливість короткочасної зміни діяльності допомагають зменшити вплив монотонної праці.

Доцільним є також використання сучасних інформаційних систем та засобів автоматизації, які дозволяють скоротити кількість рутинних операцій та спростити обробку інформації.

Висновки до розділу

У розділі було розглянуто основні аспекти організації охорони праці для фахівця з електронного архівування бібліотечних даних. Було охарактеризовано законодавчі та організаційні основи забезпечення безпеки праці, проаналізовано особливості національного та міжнародного підходів до формування безпечного робочого середовища, а також визначено основні

нормативні вимоги, що регулюють охорону праці у бібліотечній та інформаційній сфері.

У процесі дослідження було проведено аналіз умов праці працівника, особливостей організації його робочого місця та специфіки професійної діяльності. Встановлено, що робота фахівця з електронного архівування бібліотечних даних супроводжується впливом психофізіологічних, ергономічних, фізичних та технічних факторів.

Особливу увагу було приділено оцінці професійних ризиків як одному з основних елементів сучасної системи охорони праці. Розглянуто значення оцінювання ризиків для попередження небезпечних ситуацій, підвищення рівня безпеки працівників та вдосконалення організації праці.

Для однієї з характерних небезпек було побудовано дерево небезпек із використанням логічних операторів «ТА» та «АБО», що дозволило наочно відобразити взаємозв’язки між причинами виникнення небезпечної ситуації та можливими наслідками її впливу на працівника.

На основі проведеного аналізу були запропоновані рекомендації щодо покращення умов праці та зниження рівня професійних ризиків.

ВИСНОВКИ

У кваліфікаційній роботі бакалавра виконано теоретичне обґрунтування, проєктування та програмну реалізацію комплексної інформаційної системи (застосунку) для управління бібліотечними та архівними даними. Отримані результати дозволяють зробити такі загальні висновки:

1. Проведено ґрунтовний аналіз предметної області та сучасного стану цифровізації архівних і бібліотечних фондів. Дослідження виявило потребу в розробці гнучких, легких та економічно доступних вебзастосунків, адаптованих під специфіку вітчизняних установ.

2. Сформульовано перелік вимог та спроєктовано архітектуру майбутнього застосунку. За допомогою уніфікованої мови моделювання UML побудовано діаграми прецедентів (Use Case), діяльності та послідовностей. Розроблено концептуальну та логічну моделі бази даних за стандартами реляційного моделювання.

3. Обґрунтовано сучасний стек технологій, побудований на принципі розділеної архітектури (Decoupled Architecture) з використанням REST API. Серверна частина (Backend) реалізована на базі фреймворку Django REST Framework (Python). Як сховище даних використано СУБД PostgreSQL.

4. Програмно реалізовано ключові модулі системи, серед яких:

- Модуль безпеки та автентифікації що забезпечує надійний захист персональних даних та гнучке обмеження прав доступу до архівних документів.
- Алгоритм повнотекстового пошуку.
- Модуль збереження цифрових копій першоджерел (у форматах PDF, JPG, PNG).

5. Проведено тестування системи, яке містило модульні (Unit) тести для перевірки ізольованих функцій-валідаторів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Автоматизована бібліотечна інформаційна система (АБІС) // Українська бібліотечна енциклопедія. Національна бібліотека України імені Ярослава Мудрого. – URL: nlu.org.ua

2. https://ula.org.ua/

3. Костенко Л. Й. Автоматизовані бібліотечно-информаційні системи: сучасні тенденції розвитку. / Л. Й. Костенко // Науково-технічна інформація. – 2018. – №2. – С. 14–22.

4. Дем'янюк Л. Ініціативи міжнародних організацій на підтримку бібліотечно-інформаційного комплексу України / Л. Дем'янюк // Бібліотечний вісник. - 2023. - № 2. - С. 102-113. - Режим доступу: http://nbuv.gov.ua/UJRN/bv_2023_2_12

5. Маркова В. О. Проектування архітектури та інтерфейсів електронних бібліотек і цифрових архівів / В. О. Маркова // Суспільство. Документ. Комунікація. – 2023. – Вип. 19. – С. 112–128.

6. Ігнатюк М. Упровадження АІБС у практику роботи сучасних бібліотек: досвід та інновації / М. Ігнатюк // Бібліотечний вісник. – 2021. – № 6. – С. 18–26.

7. An Integrated Library System on the CERN Document Server / Library Technology Reports. – CERN, 2010. – 45 p. – URL: http://librarytechnology.org/docs/20845.pdf.

8. Next-Generation Library Information Systems: Evaluating Native Multi-Model Database Technology / ResearchGate. – 2026. – Vol. 14, No. 2. – pp. 89–104

9. Каменков К. І., Воєводіна М. Ю. КЛЮЧОВІ ОСНОВИ МАЙБУТНЬОЇ ПЕРСПЕКТИВНОЇ ТЕХНОЛОГІЇ 6G / Інформаційні технології: теорія і практика: Матеріали III (IX) Міжнародної інтернет-

конференції здобувачів вищої освіти і молодих учених (Харків-Запоріжжя-Дніпро, 25–27 березня 2026 р.) / М-во освіти і науки України, ХНУМГ ім. О. М. Бекетова [Електронний ресурс] Електрон. дані. – Дніпро : Свідлер А.Л., 2026. с. 665-666

10. jango REST Framework Documentation (Офіційна документація фреймворку). – URL: django-rest-framework.org

11. PostgreSQL Full-Text Search Optimization Guide // PostgreSQL Global Development Group. – URL: postgresql.org

12. Melchiorre P. Full-text search in Django with PostgreSQL / P. Melchiorre // Python Software Foundation. – 2020. – Slides on EuroPython. – URL: https://www.paulox.net/2017/12/22/full-text-search-in-django-with-postgresql/.

13. Чань В. Разработка веб-приложений в Django REST Framework и React / В. Чань. – К. : Діалектика, 2022. – 320 с.

14. Персіваль Г. Тестування веб-застосунків за допомогою Python, Django та Selenium / Г. Персіваль. – Львів : Видавництво Бакалавр, 2021. – 412 с.

15. Бюел С. Docker на практиці: контейнеризація веб-застосунків / С. Бюел. – Х. : Фабула, 2022. – 288 с.

16. Фаулер М. Архітектура корпоративних програмних додатків : пер. з англ. / М. Фаулер. – К. : Вільямс, 2019. – 544 с.

17. JSON Web Token (JWT) Authentication for Stateless Web Applications / Internet Engineering Task Force (IETF), RFC 7519. – URL: ietf.org

18. Step-by-Step Filtering and Full-Text Search in Python Web Applications / Django Ninja & Frameworks Guide. – 2025. – URL: https://django-ninja-aio.com/latest/tutorial/filtering/.

19. Addressing Data Governance and Risks in Electronic Archive Systems / Solix Information Technologies. – 2026. – URL: https://www.solix.com/uk/products/answers/addressing-data-governance-and-fragmented-retention-risks/.

20. До питання державної політики України у сфері охорони праці з огляду на інтеграцію до Європейського Союзу. – Режим доступу: https://veche.kiev.ua/journal/3615/.

21. Nahorna A.M., Ogorodnyk A.N. Implementation Of International Standards Ratified By Ukraine To National Legislation On Reducing The Risk Of Occupational Diseases And Occupational Injuries. Ukrainian Journal of Occupational Health, 2022, 18 (2), 83-95.

22. Гігієнічна класифікація праці за показниками шкідливості та небезпечності факторів виробничого середовища, важкості та напруженості трудового процесу. – Режим доступу: https://zakon.rada.gov.ua/laws/show/z0472-14#Text.

23. Risk assessments: what they are, why they're important and how to complete them. – Режим доступу: https://www.britsafe.org/training-and-learning/informational-resources/risk-assessments-what-they-are-why-theyre-important-and-how-to-complete-them.

24. Основні етапи та методи оцінки ризиків. – Режим доступу: https://expert-ochorona-praci.com/%D0%BC%D0%B5%D1%82%D0%BE%D0%B4%D0%B8-%D0%BE%D1%86%D1%96%D0%BD%D0%BA%D0%B8-%D1%80%D0%B8%D0%B7%D0%B8%D0%BA%D1%96%D0%B2/.

ДОДАТКИ

ДОДАТОК А

SQL-скрипт DDl (CREATE TABLE), який створює таблиці БД (код)

```
-- 1. Створення таблиці ролей (Довідник для рольової моделі RBAC)
CREATE TABLE Roles (
    id SERIAL PRIMARY KEY,
    role_name VARCHAR(50) NOT NULL UNIQUE
);

-- 2. Створення таблиці користувачів (Працівники та читачі)
CREATE TABLE Users (
    id SERIAL PRIMARY KEY,
    username VARCHAR(50) NOT NULL UNIQUE,
    password_hash VARCHAR(255) NOT NULL,
    email VARCHAR(100) NOT NULL UNIQUE,
    role_id INT NOT NULL,
    CONSTRAINT fk_user_role FOREIGN KEY (role_id)
        REFERENCES Roles(id)
        ON DELETE RESTRICT ON UPDATE CASCADE
);

-- 3. Створення таблиці бібліотечного фонду (Книги, періодика)
CREATE TABLE Library_Funds (
    id SERIAL PRIMARY KEY,
    title VARCHAR(255) NOT NULL,
    author VARCHAR(255) NOT NULL,
    isbn VARCHAR(13) NULL,
    pub_year INT NOT NULL,
    inventory_number VARCHAR(50) NOT NULL UNIQUE,
```

```
    status VARCHAR(50) NOT NULL DEFAULT 'доступно'
);

-- 4. Створення таблиці архівних документів (Унікальні документи та скани)
CREATE TABLE Archive_Documents (
    id SERIAL PRIMARY KEY,
    fund_number VARCHAR(50) NOT NULL,
    inventory_number VARCHAR(50) NOT NULL UNIQUE,
    case_title TEXT NOT NULL,
    chronological_frames VARCHAR(100) NOT NULL,
    storage_link VARCHAR(512) NOT NULL,
    access_level VARCHAR(50) NOT NULL DEFAULT 'відкритий'
);

-- 5. Створення центральної таблиці запитів та видачі документів (Логи транзакцій)
CREATE TABLE Document_Requests (
    id SERIAL PRIMARY KEY,
    user_id INT NOT NULL,
    library_fund_id INT NULL,
    archive_doc_id INT NULL,
    request_date DATE NOT NULL DEFAULT CURRENT_DATE,
    return_date DATE NULL,
    status VARCHAR(50) NOT NULL DEFAULT 'новий',

    -- Зовнішні ключі для зв'язків з іншими таблицями
    CONSTRAINT fk_request_user FOREIGN KEY (user_id)
        REFERENCES Users(id) ON DELETE CASCADE,
```

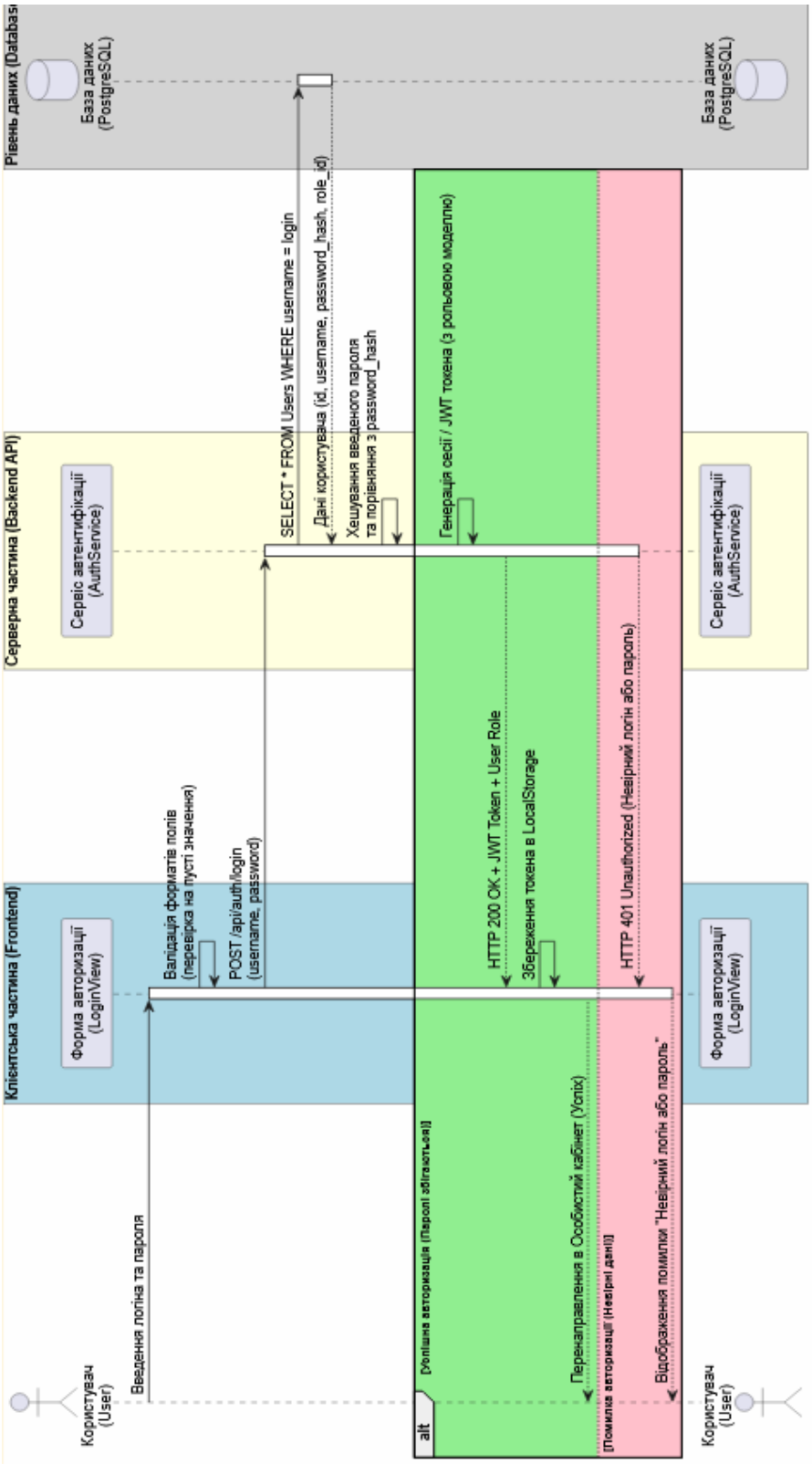
```
    CONSTRAINT fk_request_library FOREIGN KEY (library_fund_id)
        REFERENCES Library_Funds(id) ON DELETE SET NULL,

    CONSTRAINT fk_request_archive FOREIGN KEY (archive_doc_id)
        REFERENCES Archive_Documents(id) ON DELETE SET NULL,

    -- Бізнес-правило: запит повинен містити АБО книгу, АБО архівний документ, але не бути порожнім
    CONSTRAINT chk_document_presence CHECK (
        (library_fund_id IS NOT NULL AND archive_doc_id IS NULL) OR
        (library_fund_id IS NULL AND archive_doc_id IS NOT NULL)
    )
);
```

ДОДАТОК Б

Діаграма послідовності (Sequence Diagram) для процесу авторизації



ДОДАТОК В

Конфігураційний файл docker-compose.yml для запуску всього комплексу

```
version: '3.8'
services:
 # 1. Контейнер бази даних PostgreSQL
 db:
  image: postgres:15
  environment:
   POSTGRES_DB: library_archive
   POSTGRES_USER: admin
   POSTGRES_PASSWORD: secret_password
  ports:
   - "5432:5432"

 # 2. Контейнер серверної частини Django REST Framework
 backend:
  build: ./backend
  command: python manage.py runserver 0.0.0.0:8000
  volumes:
   - ./backend:/app
  ports:
   - "8000:8000"
  depends_on:
   - db  # Запускати лише після старту бази даних

 # 3. Контейнер клієнтської частини React.js
 frontend:
  build: ./frontend
  ports:
```

```
    - "3000:3000"
  volumes:
    - ./frontend:/app
  depends_on:
    - backend
```

ДОДАТОК Г

Програмна реалізація ключових модулів (код)

Г.1 Модуль автентифікації та розмежування прав доступу

Реалізація кастомного JWT-токена у файлі users_app/serializers.py:

```
from rest_framework_simplejwt.serializers import
TokenObtainPairSerializer

class CustomTokenObtainPairSerializer(TokenObtainPairSerializer):
    @classmethod
    def get_token(cls, user):
        # Отримуємо стандартний токен
        token = super().get_token(user)

        # Додаємо кастомні поля користувача у корисне
навантаження (payload) токена
        token['username'] = user.username
        token['email'] = user.email
        token['is_staff'] = user.is_staff          # Працівник бібліотеки
        token['is_archivist'] = user.is_archivist  # Спеціальна роль
архіваріуса

        return token
```

Фрагмент реалізації контролера (View) із захистом ендпоінтів та розмежуванням прав у файлі archive_app/views.py:

```
from rest_framework import viewsets
```

```
from rest_framework.permissions import IsAuthenticated,
BasePermission
from .models import ArchiveDocument
from .serializers import ArchiveDocumentSerializer


class IsArchivistOrReadOnly(BasePermission):
    """
    Кастомний дозвіл: читати архів можуть усі автентифіковані
користувачі,
    але редагувати чи видаляти – лише архіваріуси.
    """
    def has_permission(self, request, view):
        # Дозволити безпечні HTTP-методи (GET, HEAD, OPTIONS)
для всіх авторизованих
        if request.method in ['GET', 'HEAD', 'OPTIONS']:
            return request.user and request.user.is_authenticated

        # Для методів POST, PUT, DELETE перевіряємо прапорець
архіваріуса
        return request.user and request.user.is_authenticated and
getattr(request.user, 'is_archivist', False)


class ArchiveDocumentViewSet(viewsets.ModelViewSet):
    """
    Контролер для керування архівними документами.
    """
    queryset = ArchiveDocument.objects.all()
    serializer_class = ArchiveDocumentSerializer
```

```
    # Підключення захисту: автентифікація по JWT + перевірка кастомної ролі
    permission_classes = [IsArchivistOrReadOnly]
```

Г.2 Алгоритм повнотекстового пошуку та фільтрації документів

Програмний код модуля пошуку та фільтрації у файлі archive_app/views.py:

```
    from rest_framework import viewsets, filters
    from django.contrib.postgres.search import SearchVector, SearchQuery, SearchRank
    from django_filters.rest_framework import DjangoFilterBackend
    from .models import ArchiveDocument
    from .serializers import ArchiveDocumentSerializer

    class ArchiveSearchViewSet(viewsets.ModelViewSet):
        """
        Контролер для оптимізованого повнотекстового пошуку та
        багатофакторної фільтрації архівних документів.
        """
        queryset = ArchiveDocument.objects.all()
        serializer_class = ArchiveDocumentSerializer

        # Підключаємо стандартну фільтрацію та кастомний пошуковий бекенд
        filter_backends = [DjangoFilterBackend, filters.OrderingFilter]
```

```
    # Визначаємо поля для точної фільтрації (Фільтрація за категоріями)
    filterset_fields = {
        'publication_year': ['gte', 'lte', 'exact'], # Фільтр за роком (діапазон або точний)
        'author__id': ['exact'],                     # Фільтр за конкретним автором
        'document_type': ['exact'],                  # Тип документа (книга, рукопис, стаття)
    }
    ordering_fields = ['publication_year', 'created_at'] # Сортування результатів

    def get_queryset(self):
        queryset = super().get_queryset()
        query_param = self.request.query_params.get('q', None)

        if query_param:
            # 1. Формуємо пошуковий запит користувача
            query = SearchQuery(query_param, config='ukrainian') # Використовуємо українську морфологію

            # 2. Створюємо вектор пошуку із заданням ваг (A - найвищий пріоритет, B - нижчий)
            vector = SearchVector('title', weight='A', config='ukrainian') + \
                     SearchVector('description', weight='B', config='ukrainian') + \
                     SearchVector('keywords', weight='B', config='ukrainian')
```

```
            # 3. Фільтруємо за наявністю збігів та вираховуємо ранг
    релевантності
            queryset = queryset.annotate(
                rank=SearchRank(vector, query)
            ).filter(rank__gte=0.1).order_by('-rank') # Сортуємо: спочатку
    найбільш релевантні

            return queryset
```

Г3 Модуль завантаження та збереження цифрових копій

Програмна реалізація логіки обробки та валідації файлів у archive_app/models.pypythonimport os:

```
from django.db import models
from django.core.exceptions import ValidationError

def get_upload_path(instance, filename):
    """
    Генерує динамічний унікальний шлях до файлу.
    Приклад: uploads/pdf/2026/unique_name.pdf
    """
    ext = filename.split('.')[-1].lower()
    # Формуємо ім'я файлу на основі унікального архівного номера
(шифру)
    filename = f"{instance.archive_code}_{instance.id}.{ext}"
    # Групуємо файли за типом та роком додавання в базу
    return os.path.join('uploads', ext, str(instance.publication_year), filename)
```

```
def validate_file_extension_and_size(value):
    """
    Валідатор: перевіряє розширення файлу та його розмір.
    """
    ext = os.path.splitext(value.name)[1].lower()
    valid_extensions = ['.pdf', '.jpg', '.jpeg', '.png']

    # 1. Перевірка формату файлу
    if not ext in valid_extensions:
        raise ValidationError('Непідтримуваний формат файлу. Дозволено лише PDF, JPG, PNG.')

    # 2. Перевірка розміру файлу (максимум 50 МБ = 52,428,800 байт)
    limit = 50 * 1024 * 1024
    if value.size > limit:
        raise ValidationError('Файл занадто великий. Максимальний розмір — 50 МБ.')

class ArchiveDocument(models.Model):
    # Основні атрибути документа
    title = models.CharField(max_length=255, verbose_name="Назва")
    archive_code = models.CharField(max_length=50, unique=True, verbose_name="Архівний шифр")
    publication_year = models.IntegerField(verbose_name="Рік видання")

    # Поле для збереження цифрової копії з підключеним валідатором та динамічним шляхом
    digital_copy = models.FileField(
        upload_path=get_upload_path,
```

```
            validators=[validate_file_extension_and_size],
            null=True,
            blank=True,
            verbose_name="Цифрова копія"
        )
```

Фрагмент контролера для обробки Multipart-запитів у archive_app/views.py:

```
pythonfrom rest_framework import viewsets, parser_classes
from rest_framework.parsers import MultiPartParser, FormParser
from .models import ArchiveDocument
from .serializers import ArchiveDocumentSerializer

class DocumentUploadViewSet(viewsets.ModelViewSet):
    queryset = ArchiveDocument.objects.all()
    serializer_class = ArchiveDocumentSerializer

    # Підключаємо парсери для обробки завантаження файлів з форми
    parser_classes = (MultiPartParser, FormParser)
```

Г4 Модульний тест валідації файлів у файлі archive_app/tests.py

```
from django.test import TestCase
from django.core.exceptions import ValidationError
from django.core.files.uploadedfile import SimpleUploadedFile
from .models import validate_file_extension_and_size

class ArchiveValidatorTestCase(TestCase):
```

```
    def test_invalid_file_extension_raises_error(self):
        """Перевірка: система має заблокувати файл невідомого формату (.exe)"""
        incorrect_file = SimpleUploadedFile("virus.exe", b"file_content")
        with self.assertRaises(ValidationError) as context:
            validate_file_extension_and_size(incorrect_file)
        self.assertIn('Непідтримуваний формат файлу', str(context.exception))

    def test_large_file_size_raises_error(self):
        """Перевірка: система має заблокувати файл розміром більше 50 МБ"""
        # Створюємо фейковий великий файл
        large_file = SimpleUploadedFile("big_book.pdf", b"0" * (51 * 1024 * 1024))
        with self.assertRaises(ValidationError) as context:
            validate_file_extension_and_size(large_file)
        self.assertIn('Файл занадто великий', str(context.exception))
```