НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Навчально-науковий інститут атомної та теплової енергетики

Кафедра інженерії програмного забезпечення в енергетиці

ДО ЗАХИСТУ ДОПУЩЕНО
В.о. завідувача кафедри
_____Олександр КОВАЛЬ
«_____»_____2023р.

Дипломна робота

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Інженерія програмного забезпечення інтелектуальних кібер-фізичних систем і веб-технологій»
спеціальності 121 Інженерія програмного забезпечення
на тему: «Автоматизація бібліотечних процесів»

Виконала:
студентка IV курсу, групи ТІ-91
Пироговська Уляна Володимирівна _____
(підпис)

Керівник:
ст. викл. Сарибога Г.В._____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент:
к.т.н., ст. викл. Наливайчук М.В___ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студентка_____
(підпис)

Київ – 2023

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Навчально-науковий інститут атомної та теплової енергетики
Кафедра інженерії програмного забезпечення в енергетиці
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 Інженерія програмного забезпечення
Освітньо-професійна програма «Інженерія програмного забезпечення інтелектуальних кібер-фізичних систем і веб-технологій»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Олександр КОВАЛЬ (підпис)
«____»_____202_р.

ЗАВДАННЯ
на дипломну роботу студенту

_____Пироговській Уляні Володимирівні_____
(прізвище, ім'я, по батькові)

1. Тема роботи
«Автоматизація бібілотечних процесів»_____
керівник роботи_____ст. викл. Сарибога Г.В._____
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від "29" травня 2023 року №2039-с
2. Строк подання студентом роботи 12.06.2023
3. Вихідні дані до роботи:__мова програмування PHP, фреймворк Laravel, мова програмування Javascript_____
4. Зміст (дипломної роботи) пояснювальної записки (перелік завдань, які потрібно розробити створення платформи адміністрування бібліотечними ресурсами, забезпечення зручного користувацького інтерфейсу та ефективного пошуку по ресурсам репозиторію _____
5. Перелік ілюстративного матеріалу: Аналоги систем репозиторіїв, приклади використання MUI компонентів , користувацький інтерфейс phpMyAdmin, схеми архітектури системи та моделі бази даних, приклади використання користувацького інтерфейсу.

6. Дата видачі завдання «30» вересня 2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Строки виконання етапів роботи	Примітка
1	Отримання завдання	30.09.2022	виконано
2	Дослідження предметної області	02.10 - 10.11.2023	виконано
3	Постановка вимог до проєктування системи	10.11 - 07.04.2023	виконано
4	Дослідження існуючих рішень	07.04 - 17.04.2023	виконано
5	Розробка програмного продукту	17.04 - 11.05.2023	виконано
6	Тестування	12.05 - 14.05.2023	виконано
7	Захист програмного продукту	18.05.2023	виконано
8	Оформлення дипломної роботи	20.05 – 30.05.2023	виконано
9	Передзахист	5.06.2023 – 11.06.2023	виконано
10	Захист	19.06.2023 – 25.06.2023	виконано

Студент _____ Уляна Пироговська
(підпис) (ім’я, прізвище)

Керівник роботи _____ Ганна Сарибога
(підпис) (ім’я, прізвище)

РЕФЕРАТ

Структура та обсяг дипломної роботи. Робота містить 53 сторінки, 12 рисунків, 1 додаток та 10 посилань.

Метою роботи було створити єдину систему автоматизації бібліотечних процесів з простим та зрозумілим інтерфейсом для задоволення потреб персоналу бібліотеки та користувачів платформи. Процес розробки системи включав аналіз вимог, спілкування з різними групами користувачів та визначення їхніх потреб та очікувань.

Для досягнення поставленої мети виконано такі завдання:

- проаналізовано аналогічні програмні системи репозиторіїв;
- описана архітектура системи для ефективного зберігання ресурсів бібліотеки;
- розроблено зручний користувацький інтерфейс.

У рамках проекту має бути розроблено систему, що дозволяє бібліотекарям легко вести каталог книг, включаючи інформацію про авторів, жанри, видання та наявність книг. Крім того, система має надавати можливість обліковувати користувачів, зберігати їх особисті дані та історію позичань.

Практичне значення одержаних результатів полягає в розробці ефективної системи автоматизації бібліотечних процесів з можливістю забезпечення контролю над ресурсами бібліотеки, оновленнями сайту, забезпечення зручного користувацького інтерфейсу.

Ключові слова: автоматизація, бібліотечні процеси, ефективне управління, доступ до ресурсів, безпека даних.

ABSTRACT

Structure and scope of the thesis. The work contains 53 pages, 12 figures, 1 appendix and 10 references.

The purpose of the work is to create a unified library process automation system with a simple and intuitive interface to meet the needs of library staff and platform users. The system development process included requirements analysis, communication with different user groups, and identification of their needs and expectations.

To achieve this goal, the following tasks were performed:

- analysed similar software repository systems;
- describe the architecture of the system for efficient storage of library resources;
- develop a user-friendly interface.

The project is to develop a system that allows librarians to easily maintain a catalogue of books, including information about authors, genres, editions, and availability. In addition, the system should be able to register users, store their personal data and borrowing history.

The practical significance of the obtained results lies in the development of an effective system for automating library processes with the ability to control library resources, website updates, and provide a convenient user interface.

Keywords: automation, library processes, effective management, access to resources, data security.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

JSON (JavaScript Object Notation) – текстовий формат обміну даних.

ORM (Object-Relational Mapping) – технологія, яка встановлює зв'язок між об'єктно-орієнтованою моделлю програми і реляційною базою даних.

MUI (Material-UI) – бібліотека компонентів для розробки інтерфейсу користувача на основі дизайну Material Design.

API (Application Programming Interface) – це набір правил для взаємодії між програмними компонентами.

СУБД (Система управління базами даних) – це програмне забезпечення для створення, керування та оптимізації баз даних.

ВСТУП

Проект автоматизації бібліотечних процесів мав на меті створити комплексну систему, що спрощує та оптимізує роботу бібліотеки. Основними завданнями були розробка зручного інтерфейсу для бібліотекарів та користувачів, ефективне управління книгами, авторами, користувачами та позичаннями, а також забезпечення швидкого та зручного доступу до бібліотечних ресурсів.

У рамках проекту було розроблено систему, що дозволяє бібліотекарям легко вести каталог книг, включаючи інформацію про авторів, жанри, видання та наявність книг. Крім того, система надає можливість обліковувати користувачів, зберігати їх особисті дані та історію позичань. Це спрощує процес реєстрації користувачів, ведення обліку їх запозичень та повернень книг.

Одним з ключових аспектів проекту було забезпечення зручного пошуку та доступу до бібліотечних ресурсів. Система автоматизації бібліотечних процесів має потужний пошуковий механізм, що дозволяє користувачам швидко знаходити необхідні книги, журнали, наукові статті та інші джерела. Крім того, вона забезпечує доступ до електронних ресурсів, які можуть бути використані онлайн або завантажені для офлайн використання.

Одним з викликів у розробці системи автоматизації бібліотечних процесів була інтеграція з іншими системами, що використовуються в бібліотеці. Наприклад, система успішно інтегрується з електронним каталогом, що дозволяє користувачам переглядати наявність книг у режимі реального часу. Також, вона підтримує інтеграцію з електронними ресурсами, базами даних та системами керування бібліотекою, що робить процес роботи з різноманітними ресурсами ще більш зручним та ефективним.

Питання безпеки та конфіденційності даних є однією з найважливіших аспектів у системі автоматизації бібліотечних процесів. Всі дані про користувачів, книги та інші бібліотечні ресурси зберігаються в захищеній базі даних з обмеженим доступом. Застосовуються сучасні методи шифрування та автентифікації, щоб забезпечити безпеку та конфіденційність інформації.

У підсумку, система автоматизації бібліотечних процесів, розроблена у рамках даного проекту, значно полегшує роботу бібліотекарів та покращує обслуговування користувачів. Вона забезпечує ефективне управління ресурсами, швидкий доступ до бібліотечних матеріалів та забезпечує безпеку даних. Результатом реалізації проекту є покращення функціонування бібліотеки та задоволення потреб користувачів у доступі до знань та інформації.

1 ЗАДАЧА СТВОРЕННЯ СИСТЕМИ АВТОМАТИЗАЦІЇ БІБЛІОТЕЧНИХ ПРОЦЕСІВ

Метою роботи було розробити єдину систему автоматизації бібліотечних процесів з простим та зрозумілим інтерфейсом.

Для досягнення поставленої мети потрібно виконати задачі описані нижче.

1. Проаналізувати всі існуючі сайти та платформи керуваннями ресурсами бібліотеки, виявити недоліки та переваги систем, та сформулювати вимоги для створення нової системи.
2. Розробити архітектуру нової системи, яка б задовільнила усі потреби користувачів, була б доступною та зрозумілою у користуванні, як для персоналу бібліотеки, так і для користувачів платформи.
3. Провести аналіз наявних інструментів, бібліотек та платформ для створення системи.
4. Розробити веб додаток з інтеграцією бази даних для збереження даних у відповідності з розробленою архітектурою.

Веб додаток повинен мати функціонал описаний далі.

1. Створення облікового запису для керування наповнення сайту.
2. Можливість додавати, редагувати та видаляти дані про книги та їх додаткову інформацію.
3. Можливість інтегрованого пошуку ресурсів бібліотеки по різним параметрам(назва, автор, рік видання, категорія).
4. Можливість перегляду оновлень сайту, забезпечення створення додаткових статичних сторінок.

1.1 Аналіз загальних вимог до систем автоматизації бібліотечних процесів

Під час процесу аналізу загальних вимог до систем автоматизації бібліотечних процесів було проведено детальне вивчення потреб і очікувань різних груп

користувачів. Наприклад, під час спілкування з бібліотекарями виявлено, що їм необхідна зручна система каталогізації, яка дозволить швидко знаходити інформацію про книги, вести облік позичень та повернень, а також здійснювати резервування та контроль термінів позичання. Для цього було вирішено використати потужні можливості Eloquent ORM у поєднанні з базою даних, що дозволить ефективно зберігати та керувати даними про ресурси бібліотеки.

Крім того, під час спілкування з користувачами було виявлено потребу в зручному інтерфейсі, який дозволяє швидко знайти та переглянути інформацію про книги, а також здійснювати замовлення та взаємодіяти з бібліотекою через онлайн-канали. Для реалізації цих функцій було вирішено використати фреймворк React[3] та MUI Components, які дозволять розробити зручний та інтуїтивно зрозумілий інтерфейс для користувачів.

У процесі аналізу також було виявлено необхідність в інтеграції з іншими системами, наприклад, електронним каталогом або системою керування бібліотекою. Це дозволить забезпечити єдину точку доступу до бібліотечних ресурсів, уникнути дублювання даних та покращити ефективність роботи бібліотеки. Для цього використовується механізм API та взаємодія з іншими системами за допомогою бібліотеки Axios.

Загалом, аналіз показав, що розробка системи автоматизації бібліотечних процесів має великий потенціал для поліпшення роботи бібліотеки та задоволення потреб користувачів. Проте, було також виявлено певні виклики і недоліки, такі як складність інтеграції з існуючими системами, необхідність забезпечення безпеки даних та високої доступності системи. Для вирішення цих проблем необхідно враховувати найкращі практики розробки та використання відповідних технологій та інструментів.

Висновки до розділу 1

Проведений аналіз загальних вимог до систем автоматизації бібліотечних процесів надав важливі висновки щодо потреб і очікувань користувачів та

бібліотечного персоналу. Цей аналіз дозволив зрозуміти, які функціональні можливості повинна мати нова система для задоволення потреб користувачів.

У процесі аналізу були ідентифіковані ключові вимоги, такі як можливість каталогізації та управління ресурсами, інтегрований пошук, створення облікових записів та взаємодія з користувачами через зручний інтерфейс. З метою забезпечення цих вимог було прийнято рішення про використання Eloquent ORM для ефективного зберігання та керування даними, а також фреймворку React та MUI Components для розробки зручного та інтуїтивно зрозумілого інтерфейсу.

Крім того, аналіз показав необхідність інтеграції з існуючими системами, такими як електронний каталог або система керування бібліотекою. Це дозволить забезпечити єдину точку доступу до ресурсів, уникнути дублювання даних та підвищити ефективність роботи бібліотеки. Для забезпечення такої інтеграції було використано механізм API та бібліотеку Axios.

У результаті проведеного аналізу було досягнуто розуміння вимог та потреб користувачів, що дозволить розробити систему, яка задовольнятиме їхні очікування. Однак, виявлено певні виклики, такі як складність інтеграції з існуючими системами та необхідність забезпечення безпеки даних. Для успішної реалізації проекту необхідно урахувати ці виклики та використовувати найкращі практики розробки та відповідні технології.

Отже, висновок з цього розділу полягає в тому, що аналіз загальних вимог до системи автоматизації бібліотечних процесів виявив ключові функціональні вимоги та потреби користувачів. Це надало важливу інформацію для подальшої розробки та виконання мети проекту.

2 ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ

Бібліотека є важливим інститутом культури, який забезпечує доступ до різноманітних друкованих та електронних ресурсів для освіти, науки та розвитку особистості. Вона є центром збереження, організації та поширення знань, літератури, наукових досліджень, мистецтва та культурного надбання.

У предметній області бібліотеки важливо враховувати різні аспекти, такі як книжковий фонд, автори, жанри, користувачі, позичання та повернення книг, каталогізація, архівування та зберігання документів. Бібліотекарі відіграють ключову роль у підтримці та розвитку цієї області, забезпечуючи організацію та доступ до бібліотечних ресурсів.

Основні завдання бібліотеки включають формування колекцій книг, проведення каталогізації та класифікації документів, позичання та повернення книг, облік користувачів, а також забезпечення зручного доступу до бібліотечних ресурсів. Важливо також забезпечити безпеку даних та конфіденційність користувачів.

Зважаючи на розмаїття та складність бібліотечних процесів, розробка системи автоматизації бібліотеки є необхідною для поліпшення ефективності та якості надання бібліотечних послуг, спрощення роботи бібліотекарів та зручності користувачів.

Перевагою створення єдиної системи автоматизації бібліотечних процесів є можливість оптимізувати роботу з книговим фондом, автоматично веде облік книг та інших ресурсів бібліотеки, допомагає з легкістю знаходити та каталогізувати матеріали. Це робить процес організації та збереження документів більш ефективним та швидким. Система автоматизації надає користувачам зручний інтерфейс для пошуку, аналізу та отримання необхідної інформації. Користувачі можуть швидко знаходити потрібні книги, статті, журнали та інші потрібні їм ресурси.

2.1 Огляд існуючих систем та технологій

У сучасному світі існує безліч систем та технологій, спрямованих на управління бібліотечними ресурсами. Важливим завданням для бібліотек є вибір найбільш підходящих інструментів, які задовольняють їхні потреби та сприяють оптимізації робочих процесів. У цій частині ми розглянемо кілька існуючих систем та технологій для управління бібліотечними ресурсами, зокрема ELA.KPI, яка є однією з них. Проаналізуємо їх переваги та недоліки з метою зрозуміти, які інструменти можуть бути найбільш вигідними та ефективними для нашої бібліотеки. Далі будемо розглядати основні характеристики та можливості цих систем, що дозволить зробити обґрунтований вибір та визначити найкращий напрямок для подальшого розвитку нашої бібліотеки.

2.1.1 Система управління архівами даних ELA.KPI

ELA.KPI - це система управління бібліотекою, розроблена КПІ ім. Ігоря Сікорського. Вона має кілька переваг, які роблять її привабливою для бібліотек та користувачів.

Однією з ключових переваг ELA.KPI є ефективне керування бібліотечними ресурсами. Система надає бібліотекам зручний інтерфейс(рис. 2.1) для керування різноманітними ресурсами, такими як книги, наукові видання, журнали та інші матеріали. Бібліотекарі можуть вести облік колекцій, контролювати процеси видачі та повернення ресурсів, а також забезпечувати оновлення та організацію каталогування ресурсів.

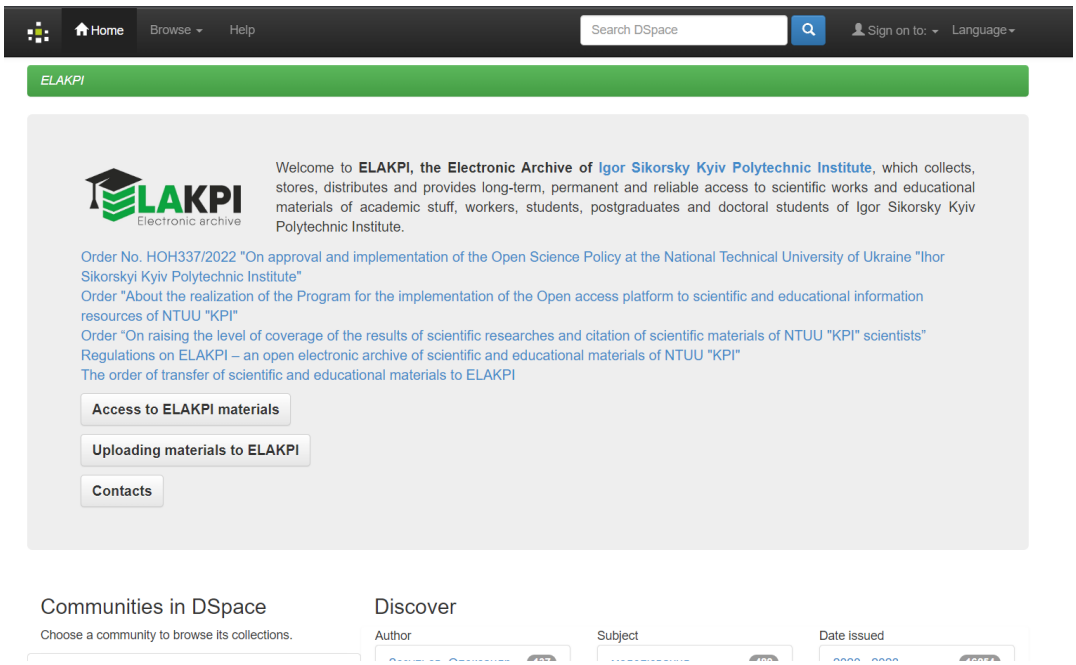

Рисунок 2.1 – Інтерфейс репозиторію ELA.KPI

Ще однією перевагою ELA.KPI є зручний пошук та доступ до бібліотечних ресурсів для користувачів. Система надає зручний інтерфейс для пошуку ресурсів за різними критеріями, такими як назва, автор, тематика тощо. Користувачі можуть швидко знайти потрібні матеріали та отримати доступ до них, що сприяє зручності та ефективності використання бібліотеки.

Проте, як і в усіх системах, ELA.KPI має свої недоліки. Наприклад, вона може вимагати певного часу та ресурсів для впровадження та налаштування. Крім того, іноді користувачам може знадобитися певний час для оволодіння інтерфейсом та функціями системи. Також, система може мати певні обмеження або недоліки у випадку роботи з великим обсягом даних чи вимогами до швидкості та масштабованості.

Враховуючи всі переваги та недоліки, ELA.KPI є потужним інструментом для управління бібліотеками, який може покращити ефективність та доступність бібліотечних послуг для користувачів. Використання ELA.KPI дозволяє бібліотекам зосередитися на якісному наданні ресурсів та покращенні користувацького досвіду.

2.1.2 Репозиторій Digital Commons

Digital Commons - це платформа, розроблена компанією Bepress, яка надає університетам та коледжам можливість створити власний репозиторій для зберігання та поширення наукових матеріалів. Вона має ряд переваг, які роблять її привабливим вибором для університетських спільнот.

Однією з переваг Digital Commons є його широкий функціонал. Платформа надає зручні інструменти для завантаження, каталогізації та організації різноманітних документів, включаючи наукові статті, тези, звіти, дисертації та інші академічні матеріали. Крім того, Digital Commons має можливості для інтеграції з іншими системами та базами даних, що дозволяє покращити доступність та видимість наукових робіт.

Ще одною перевагою Digital Commons є його користувацький інтерфейс(рис. 2.2). Він розроблений з урахуванням потреб користувачів, забезпечуючи зручну навігацію, пошукові можливості та інтуїтивно зрозумілий процес завантаження та організації документів. Це дозволяє користувачам легко знайти необхідну інформацію та взаємодіяти з науковими матеріалами.

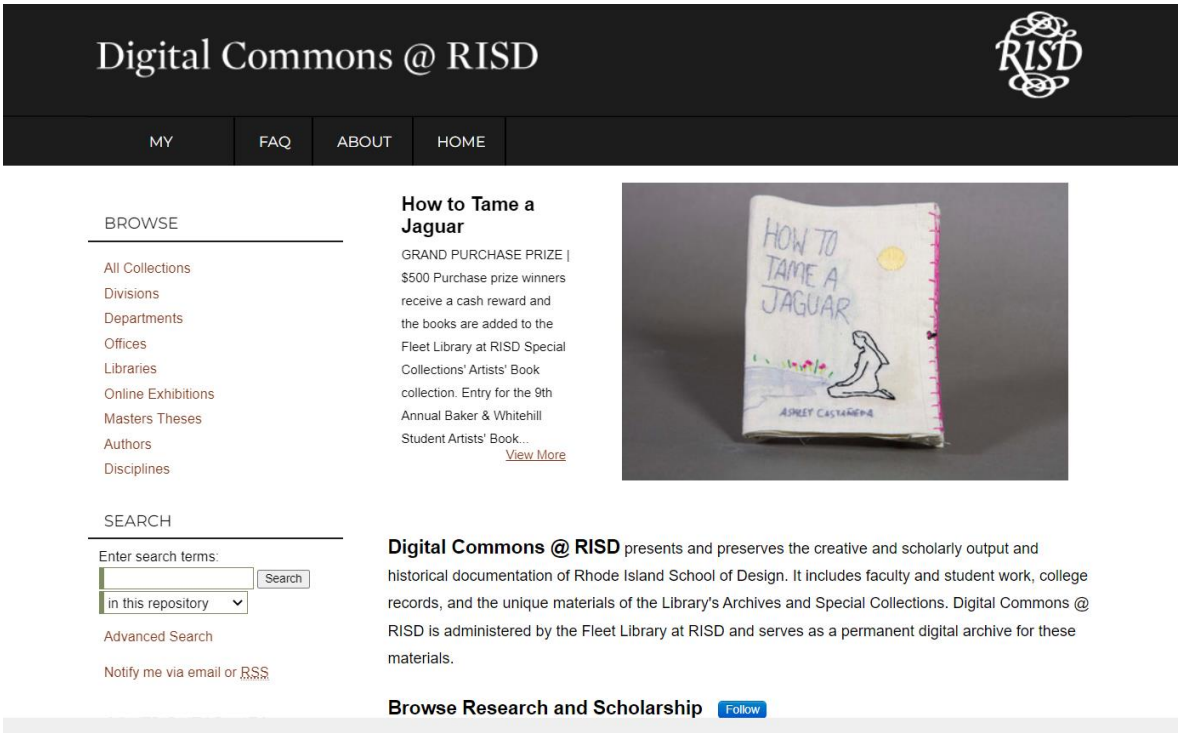

Рисунок 2.2 – Користувацьки інтерфейс платформи Digital Commons

Незважаючи на переваги, Digital Commons також має певні недоліки, які варто врахувати. Один з них - це комерційна природа платформи. Це означає, що використання Digital Commons може бути пов'язане зі значними витратами для університетів та коледжів, особливо для невеликих закладів з обмеженим бюджетом. Крім того, обмежена гнучкість платформи може ускладнювати налаштування під конкретні потреби та вимоги кожного університету.

У результатах аналізу Digital Commons було виявлено, що вона є потужним інструментом для зберігання та поширення наукових матеріалів університетськими спільнотами. Її функціонал та користувацький інтерфейс сприяють зручності використання та доступу до наукових робіт. Однак, необхідно враховувати комерційну природу платформи та обмежену гнучкість, які можуть впливати на її вартість та адаптацію до конкретних потреб університету.

Висновки до розділу 2

В результаті аналізу існуючих систем та технологій для управління бібліотечними ресурсами, включаючи ELA.KPI, можна зробити кілька важливих висновків. По-перше, відзначається широкий спектр функціональних можливостей, які пропонують ці системи. Вони дозволяють ефективно керувати даними про книжки, авторів та інші ресурси, а також забезпечують зручний інтерфейс для користувачів.

Далі, багато з існуючих систем пропонують гнучкість та налаштовуваність, що дозволяє враховувати потреби конкретної бібліотеки. Це може включати можливість додавання власних полів, налаштування прав доступу та інші параметри, що сприяють індивідуалізації системи.

Також важливо відзначити, що деякі системи пропонують інтеграцію з іншими платформами, такими як електронні каталоги, бази даних авторів та інші зовнішні джерела інформації. Це дозволяє збільшити обсяг доступної інформації та забезпечити користувачів бібліотеки більш повною та розширеною базою даних.

Незважаючи на переваги, варто враховувати певні недоліки, які можуть бути пов'язані з існуючими системами. Деякі з них можуть мати складний інтерфейс або вимагати додаткового навчання для користувачів. Крім того, вартість впровадження та підтримки таких систем може бути значною, особливо для менших бібліотек з обмеженим бюджетом.

Загалом, аналіз існуючих систем та технологій показує, що вибір відповідних інструментів для управління бібліотечними ресурсами є критичним завданням. Потрібно уважно вивчити можливості та характеристики кожної системи, порівняти їх з потребами та обмеженнями власної бібліотеки, а також враховувати фінансові та технічні ресурси, щоб забезпечити успішне впровадження та подальший розвиток системи управління бібліотекою.

3 ЗАСОБИ РОЗРОБКИ СИСТЕМИ

Аналіз предметної області та наявних рішень дав змогу виявити основні потреби користувачів та виокремити найбільш актуальні та бібліотеки для створення системи автоматизації бібліотечних процесів. Під час вибору інструментів для розробки нашої системи були враховані різноманітні критерії, що дозволили уникнути недоліків наявних рішень та забезпечити переваги в порівнянні з ними.

3.1 Мова програмування Javascript

JavaScript є високорівневою, інтерпретованою мовою програмування, яка широко використовується для розробки веб-додатків. Вона надає можливість створювати динамічні інтерактивні елементи на веб-сторінках, забезпечує взаємодію з користувачем та маніпуляцію вмістом сторінки.

JavaScript є однією з трьох основних технологій веб-розробки разом з HTML і CSS. Вона використовується для створення різноманітних функцій, включаючи валідацію форм, анімацію, маніпулювання DOM-елементами, взаємодію з веб-сервером за допомогою AJAX, роботу з кукісами та багато іншого. JavaScript також дозволяє створювати повноцінні веб-додатки з використанням фреймворків, таких як React, Angular та Vue.js.

Одним з ключових аспектів JavaScript є його можливість працювати з подіями. За допомогою обробників подій можна реагувати на взаємодію користувача з елементами сторінки, такими як кліки, наведення курсора, натискання клавіш та інші. Це дозволяє створювати динамічні інтерфейси, які реагують на дії користувача миттєво.

JavaScript підтримує об'єктно-орієнтований підхід до програмування, де дані та функції пов'язані в об'єкти. Це сприяє створенню модульних та розширюваних додатків. Мова також має широку підтримку стандартів, що дозволяє розробникам використовувати нові функції та можливості мови, що постійно розширюються.

Однією з найбільших переваг JavaScript є його широке поширення та активна спільнота розробників. Існує велика кількість ресурсів, документації, форумів та бібліотек, що спрощують розробку та вирішення проблем. Крім веб-розробки, JavaScript також використовується для розробки мобільних додатків, настільних програм та навіть для створення серверної частини за допомогою платформи Node.js.

Загалом, JavaScript є потужним інструментом розробки, який дозволяє створювати інтерактивні, динамічні та сучасні веб-додатки. Вона надає розробникам гнучкість, широкі можливості та надійну базу для творчості та інновацій у сфері веб-розробки.

JavaScript є мовою з відкритим вихідним кодом, що дозволяє розробникам вносити власні зміни та вдосконалення до мови. Це сприяє постійному розвитку мови та відкриває можливості для співпраці та спільної роботи всієї громади розробників над покращенням мови та стандартів[1].

Одним з основних переваг JavaScript є його платформонезалежність. Це означає, що код JavaScript може виконуватись на різних платформах, таких як веб-переглядачі, мобільні пристрої, сервери та навіть вбудовані системи. Це робить JavaScript універсальним інструментом розробки, що забезпечує сумісність та мобільність додатків.

JavaScript також має велику кількість сторонніх бібліотек та фреймворків, що спрощують розробку та розширення функціональності додатків. Наприклад, jQuery, React, Vue.js, Angular і Node.js є популярними інструментами, які дозволяють розробникам швидко будувати складні, масштабовані та ефективні додатки.

JavaScript також використовується для розробки ігор, аудіо/відео плеєрів, анімації та взаємодії з графікою. За допомогою HTML5 Canvas і WebGL, розробники можуть створювати захоплюючі візуальні ефекти та ігри безпосередньо у веб-браузері, що розширює можливості JavaScript щодо створення різноманітних додатків[2].

У сучасному цифровому світі JavaScript виявляється незамінним інструментом для веб-розробників. Він надає потужні можливості для створення інтерактивних та динамічних веб-додатків, що забезпечують багатофункціональність, швидкість та

зручний інтерфейс для користувачів. З його допомогою розробники можуть створювати інноваційні та ефективні рішення, що сприяють прогресу в сфері веб-розробки та покращують веб-середовище в цілому.

3.2 React Js

React.js є потужним інструментом для розробки веб-додатків, який здобув велику популярність серед розробників. Його головна перевага полягає в тому, що він дозволяє будувати інтерактивні та ефективні користувацькі інтерфейси[3].

Одним з ключових аспектів React.js є використання компонентного підходу. Замість написання монолітного коду, React.js дозволяє розбити його на невеликі компоненти, які можуть бути повторно використані і організовані в ієрархію. Це спрощує розробку та підтримку додатку, а також дозволяє зосередитися на окремих частинах функціональності.

Ще одна перевага React.js полягає у використанні віртуального DOM (Document Object Model). Він забезпечує швидку перерендеринг та ефективне керування змінами у користувацькому інтерфейсі. React.js автоматично визначає, які компоненти потребують оновлення та виконує мінімальну кількість операцій для оновлення лише потрібних елементів.

Більш того, React.js має велику спільноту розробників, що забезпечує наявність багато інструментів, бібліотек та розширень. Зокрема, інтеграція з Redux, React Router та іншими популярними рішеннями дозволяє побудувати масштабовані та добре організовані додатки.

Узагальнено, React.js є потужним інструментом для розробки веб-додатків, який пропонує компонентний підхід, швидку перерендеринг та багато інструментів для полегшення розробки. Це робить його популярним вибором для створення сучасних, ефективних та інтерактивних веб-додатків.

Нижче описані головні причини обрати React.js.

1. Ефективність та швидкодія – React.js використовує віртуальний DOM, що дозволяє оптимізувати процес оновлення інтерфейсу. Він визначає лише

необхідні зміни та застосовує їх до реального DOM, що робить додаток швидким та реактивним.

2. Компонентна архітектура – React.js базується на компонентах, які можна перевикористовувати, складати та комбінувати між собою. Це дозволяє побудувати модульну структуру додатку, полегшує його розширення та підтримку.

3. Декларативний підхід – Завдяки декларативному синтаксису, React.js дозволяє описувати бажаний стан інтерфейсу, а не розробляти кожну окрему деталь. Це спрощує розробку та збереження коду, полегшує його читабельність та розуміння.

4. Багата екосистема – React.js має широку спільноту розробників, що призводить до наявності багато корисних бібліотек, компонентів та інструментів. Ви можете легко знайти готові рішення для багатьох задач, що прискорює розробку та покращує якість проекту.

5. Інтеграція з іншими технологіями – React.js можна успішно поєднувати з різними інструментами та бібліотеками, такими як Redux для керування станом, React Router для маршрутизації або Axios для здійснення запитів до сервера. Це робить його універсальним і гнучким використанням у різних проектах.

3.2 MUI react components

MUI (Material-UI) – це потужна бібліотека компонентів, розроблена на основі Material Design, яка дозволяє створювати красивий та сучасний інтерфейс веб-додатків. Нижче описані основні причини обрати MUI як інструмент розробки[7].

1. Готові компоненти – MUI надає широкий набір готових компонентів, таких як кнопки, поля введення, таблиці, модальні вікна та багато інших. Це дозволяє розробникам швидко та легко будувати інтерфейс, перевикористовуючи готові елементи.

2. Стилізація та темизація – MUI надає розширені можливості для стилізації компонентів. Ви можете налаштувати вигляд компонентів, змінюючи кольори, шрифти, тіні та багато інших параметрів. Крім того, ви можете легко створювати власні теми для забезпечення єдиної стилістики в додатку.

3. Реактивність та адаптивність – MUI підтримує принципи реактивного дизайну, що дозволяє інтерфейсу змінюватися динамічно в залежності від розміру екрану та пристрою. Це забезпечує зручну та оптимізовану роботу на різних пристроях, включаючи комп'ютери, планшети та мобільні телефони.

4. Документація та спільнота – MUI має докладну документацію, яка надає вичерпну інформацію про використання компонентів та функціональності бібліотеки. Крім того, у MUI є активна спільнота розробників, яка пропонує підтримку та надає відповіді на запитання та проблеми.

5. Інтеграція з React та іншими бібліотеками – MUI побудована на основі React, що дозволяє легко інтегрувати її з існуючими React-проєктами. Крім того, MUI також підтримує інтеграцію з іншими бібліотеками та інструментами, такими як Redux або TypeScript.

Завдяки MUI ви можете швидко розробляти красиві та функціональні інтерфейси для своїх веб-додатків, спрощуючи процес розробки та покращуючи візуальний досвід користувачів, приклад використання зображений на рисунку 3.1.

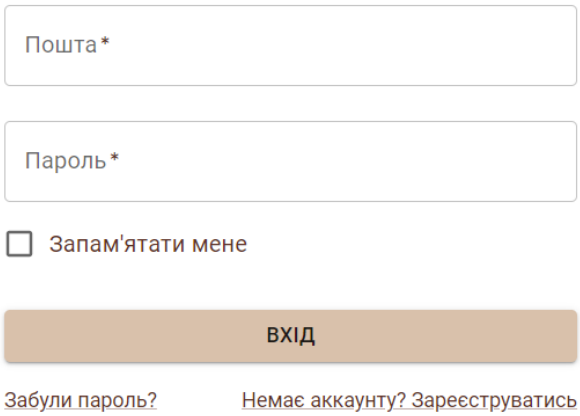

Рисунок 3.1 – Приклад використання MUI компонентів

3.3 Laravel PHP

Laravel – це потужний фреймворк для розробки бекенду веб-додатків, написаний на мові програмування PHP[6]. Нижче описані основні причини обрати Laravel як інструмент розробки бекенду.

1. Елегантна синтаксис та простота використання – Laravel має чистий і читабельний синтаксис, що робить його дуже простим у використанні. Він надає широкий спектр готових функцій та інструментів, які спрощують розробку і забезпечують швидке впровадження функціональності.

2. MVC архітектура – Laravel побудований на основі патерну проектування MVC (Model-View-Controller), що дозволяє розподілити логіку додатку на логічно зв'язані компоненти. Це полегшує розробку, підтримку та масштабування коду.

3. Багата функціональність – Laravel надає велику кількість готових компонентів та функцій, таких як аутентифікація, маршрутизація, робота з базою даних та багато інших. Це дозволяє розробникам ефективно використовувати готові рішення та прискорює розробку додатків.

4. Екосистема та спільнота – Laravel має активну спільноту розробників, яка надає підтримку, документацію та відповіді на запитання. Крім того, в екосистемі Laravel доступні різноманітні сторонні пакети, які допомагають розширити функціональність фреймворку.

5. Безпека – Laravel має вбудовану систему для захисту від потенційних загроз безпеки, таких як SQL-ін'єкції, XSS-атаки та інші. Це дозволяє розробникам зосередитися на функціональності додатку, маючи впевненість у його безпеці.

Також, Laravel надає зручний інтерфейс для відправлення HTTP-запитів та керування базою даних. Він має вбудований механізм для виконання запитів до бази даних, що дозволяє розробникам зручно працювати з даними. Laravel підтримує різні типи баз даних, такі як MySQL, PostgreSQL, SQLite та багато інших.

Для відправлення HTTP-запитів Laravel використовує фасад Guzzle, який забезпечує простий та зрозумілий інтерфейс для взаємодії з зовнішніми API. Це дозволяє легко виконувати запити до сторонніх сервісів та обмінюватися даними з ними.

Крім того, Laravel має вбудовану систему міграцій, яка дозволяє легко керувати структурою бази даних та виконувати міграції між різними версіями додатку. Це забезпечує зручний спосіб оновлення схеми бази даних без необхідності вручного втручання.

Загалом, використання Laravel як інструменту для відправлення запитів та керування базою даних робить розробку бекенду ще більш зручною та ефективною.

Laravel надає потужні засоби для реалізації аутентифікації (authentication) в веб-додатках. Вона спрощує процес налаштування та управління аутентифікацією користувачів і забезпечує безпеку та захист даних.

Основні можливості аутентифікації в Laravel описані нижче.

1. Система реєстрації: Laravel надає готові компоненти та маршрути для створення нових облікових записів користувачів. Це включає можливість валідації та збереження даних, підтвердження електронної пошти та автоматичне створення хешованого пароля.

2. Входу користувача: Laravel дозволяє швидко налаштувати форму входу користувача з перевіркою введених даних та автентифікацією. Вона також підтримує різні методи аутентифікації, такі як аутентифікація за допомогою електронної пошти або ім'я користувача, аутентифікація через сторонні служби соціальних мереж тощо.

3. Захист маршрутів: Laravel надає простий спосіб захистити певні маршрути від доступу неавтентифікованих користувачів. За допомогою middleware, ви можете легко вказати, які маршрути або контролери повинні бути доступні лише автентифікованим користувачам.

4. Робота з ролями та дозволами: Laravel дозволяє робити розподіл доступу до різних ресурсів або функціональності в залежності від ролей користувачів. Ви можете визначити ролі, призначати їх користувачам та перевіряти права доступу для кожної ролі.

5. Сесії та пам'ять аутентифікації – Laravel надає можливість зберігати стан аутентифікації користувача через сесії або кукі. Це дозволяє зберігати інформацію про автентифікацію між запитами та забезпечує зручний механізм виходу з системи.

3.4 Axios

Axios – це бібліотека JavaScript, яка надає зручний інтерфейс для виконання асинхронних HTTP-запитів на стороні клієнта. Вона широко використовується в розробці веб-додатків і забезпечує простий та потужний спосіб взаємодії з сервером.

Нижче описані основні переваги Axios.

1. Простий у використанні – Axios надає простий та зрозумілий API для відправлення різних видів HTTP-запитів, таких як GET, POST, PUT, DELETE тощо. Це дозволяє розробникам швидко налаштовувати та виконувати запити без зайвих зусиль.

2. Підтримка промісів – Axios повністю підтримує проміси, що робить його інтеграцію з сучасними функціональними підходами, такими як async/await, дуже простою. Це дозволяє легко обробляти відповіді сервера та управляти асинхронними операціями.

3. Міжплатформовість – Axios підтримує виконання HTTP-запитів на різних платформах, включаючи браузери та Node.js. Це дозволяє використовувати одну і ту ж бібліотеку для здійснення запитів незалежно від платформи, на якій працює додаток.

4. Перехоплення помилок – Axios надає зручні механізми для перехоплення та обробки помилок, які виникають під час виконання запитів. Це дозволяє розробникам легко впоратися з помилками та прийняти відповідні дії.

5. Інтерсептори – Axios дозволяє використовувати інтерсептори для перехоплення та маніпуляції запитами та відповідями перед їх відправкою або обробкою. Це дає можливість розробникам реалізувати різноманітні функціональність, таку як автоматична авторизація, обробка помилок, додавання заголовків тощо.

В цілому, Axios є потужним та зручним інструментом для виконання HTTP-запитів на стороні клієнта. Він надає простий API, підтримку промісів, міжплатформовість та інші корисні функції, що роблять його популярним вибором для розробки веб-додатків.

3.5 MySQL

MySQL є однією з найпопулярніших відкритих реляційних систем управління базами даних (СУБД), яка використовує мову SQL (Structured Query Language) для зберігання та керування даними[4]. Нижче описані численні переваги, які роблять її частим вибором для розробників та організацій.

1. Надійність та стабільність – MySQL є дуже стабільною СУБД, яка виконується без збоїв та має довгий термін підтримки. Вона підтримує транзакції з ACID-властивостями (атомарність, консистентність, ізольованість, стійкість до відмов), що гарантує цілісність та надійність даних.

2. Простота використання – MySQL має зрозумілий та легкий у використанні інтерфейс, який дозволяє розробникам швидко створювати, змінювати та запитувати дані. Вона підтримує стандартну мову SQL, що робить роботу з базою даних зручною та простою.

3. Гнучкість – MySQL надає широкий набір можливостей та налаштувань, що дозволяють розробникам виконувати різноманітні завдання та оптимізувати продуктивність бази даних. Вона підтримує індексацію, зовнішні ключі, перегляди, хранилища процедур та багато іншого.

4. Масштабованість – MySQL може працювати з великими обсягами даних та високими навантаженнями. Вона підтримує розподілені системи, реплікацію, шардування та кластеризацію, що дозволяє збільшувати продуктивність та доступність бази даних.

5. Велика спільнота та підтримка – MySQL має велику та активну спільноту розробників, що надає різноманітні додаткові ресурси, документацію, форуми підтримки та оновлення безпеки.

Перегляд, аналіз та користування базою даних виконувалось безпосередньо через phpMyAdmin, зручне програмне забезпечення СУБД, приклад зображено на рисунку 3.2.

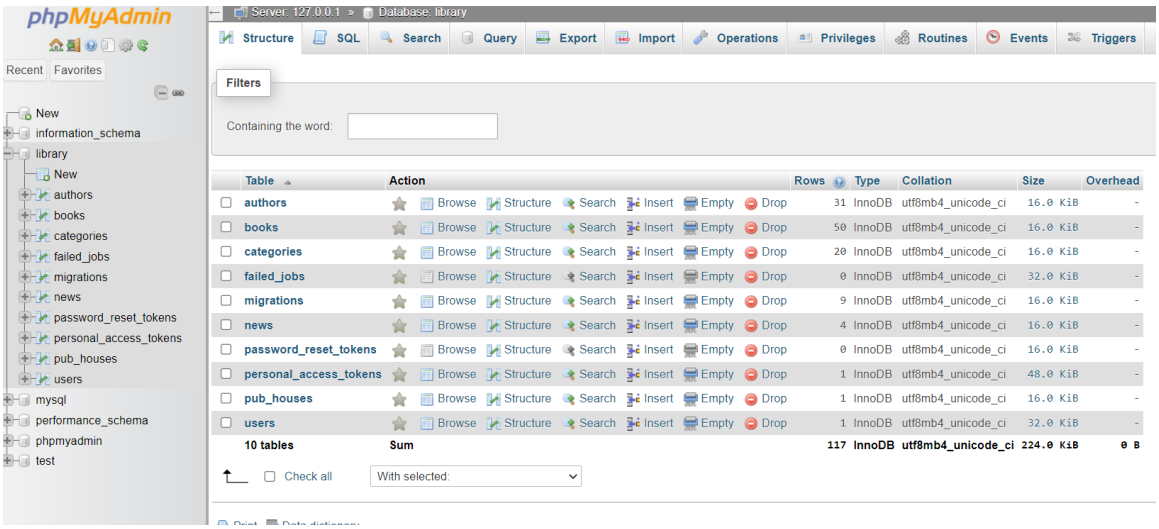

Рисунок 3.2 – Вигляд бази даних MySQL у phpMyAdmin

За допомогою phpMyAdmin можна відобразити структуру бази даних[9], переглянути дані у таблицях, редагувати їх, виконувати складні запити для аналізу і фільтрації даних. Також можна здійснювати адміністративні завдання, такі як створення нових баз даних, користувачів, надання прав доступу та інше.

3.6 Git

Git – це розподілена система керування версіями, яка дозволяє розробникам ефективно працювати з кодом та відстежувати його зміни. Нижче описані основні принципи, на яких ґрунтується Git.

1. Розподілена архітектура – Кожен розробник має повну копію репозиторію, що дозволяє локально працювати над проектом. Це дозволяє ефективно галузити код, вносити зміни та спільно працювати з іншими розробниками.

2. Гілки (branches) – Git дозволяє створювати гілки, що представляють собою незалежні лінії розвитку проекту. Це дозволяє розробникам працювати паралельно над різними функціональностями чи виправленнями помилок, не впливаючи один на одного.

3. Злиття (merging) – Після завершення роботи над функціональністю чи виправленням помилок, гілку можна об'єднати з основною гілкою проекту. Це

називається злиттям. Git автоматично виявляє конфлікти та допомагає розробникам їх вирішити.

4. Історія комітів – Git зберігає повну історію змін у репозиторії, включаючи коментарі до комітів, автора, дату та час. Це дозволяє розробникам легко відстежувати зміни, переглядати старі версії файлів та відновлювати код до попередніх станів.

5. Віддалений доступ – Git дозволяє взаємодіяти з віддаленими репозиторіями, що дозволяє розробникам спільно працювати над проектом. Це забезпечує можливість завантажувати та зберігати зміни на віддаленому сервері, обмінюватися кодом з іншими розробниками та управляти версіями проекту.

Git є одним з найпоширеніших інструментів для керування версіями в сфері розробки програмного забезпечення. Його гнучкість, швидкість та можливості спільної роботи роблять його незамінним інструментом для командної розробки та збереження історії змін в проектах.

3.7 Середовище розробки Visual Studio Code

Visual Studio Code (VS Code) - це високопопулярне та функціональне інтегроване середовище розробки (IDE), розроблене компанією Microsoft. Воно отримало широку популярність серед розробників для програмування на різних мовах, включаючи JavaScript, Python, C++, HTML, CSS та багато інших.

Однією з ключових переваг Visual Studio Code є його простота використання та можливість налаштування. Воно має чистий і зрозумілий інтерфейс, що дозволяє розробникам швидко розпочати роботу та легко орієнтуватись у своїх проектах. VS Code також підтримує велику кількість розширень, які дозволяють налаштувати середовище розробки під свої потреби та розширити його функціональність.

У Visual Studio Code вбудована підтримка систем контролю версій, таких як Git, що дозволяє розробникам комфортно працювати зі своїми проектами та відстежувати зміни. Також воно надає потужні інструменти для відлагодження коду,

автоматичного завершення коду, перевірки синтаксису та багато іншого, що полегшує процес розробки та підвищує продуктивність розробника.

VS Code має вбудовану підтримку для розробки веб-додатків, включаючи популярні фреймворки, такі як React, Angular та Vue.js. Воно надає можливість редагування HTML, CSS та JavaScript коду зі зручною навігацією по файлам та швидким переглядом результатів.

Ще однією важливою особливістю Visual Studio Code є його платформна незалежність. Воно доступне для операційних систем Windows, macOS та Linux, що дозволяє розробникам використовувати його на будь-якій платформі за своїм вибором.

Узагальнюючи, Visual Studio Code є потужним та зручним інструментом розробки, який надає розробникам широкі можливості для створення та підтримки різноманітних проектів. З його допомогою розробники можуть працювати ефективно, швидко та комфортно, що робить його одним з найпопулярніших виборів серед розробників по всьому світу.

Застосування плагінів у розробці є важливою складовою для підвищення продуктивності та зручності роботи розробників. Вони розширюють можливості інтегрованого середовища розробки, надаючи додаткові функціональність та інструменти, спеціально розроблені для певних мов програмування або фреймворків.

Було встановлено декілька плагінів для розробки. Один з них - "React Snippets" – надає набір корисних скорочень для розробки на React-фреймворку. Це спрощує процес написання коду, забезпечуючи швидкий доступ до шаблонів коду та автозаповнення для реактивних компонентів, роутінгу, стилів та інших елементів, пов'язаних з React. Використання цього плагіна допомагає прискорити розробку React-проектів та покращити якість коду.

Другий плагін, який було встановлено, називається "PHP". Цей плагін призначений для підтримки розробки на мові програмування PHP. Він надає розширені можливості, такі як автозаповнення коду, підказки про синтаксис, перевірка помилок та інші інструменти, що полегшують роботу з PHP-проектами у Visual Studio Code.

Крім того, був встановлений плагін "Composer Integration", який допомагає взаємодіяти з пакетним менеджером Composer у ваших проектах на PHP. Він надає зручний інтерфейс для управління залежностями та установки необхідних пакетів з використанням Composer безпосередньо в середовищі розробки.

Завдяки встановленим плагінам, було отримано розширені можливості розробки, забезпечивши швидкість, зручність та покращення продуктивності при роботі з проектами на React та PHP.

Висновки до розділу 3

У даному розділі було розглянуто основні інструменти розробки нашого програмного застосунку, в деталях описано їх переваги та можливі недоліки з урахуванням альтернатив. Було чітко проаналізовано можливі проблеми що можуть виникнути у ході розробки нашого додатку та винесено відповідні висновки щодо основних інструментів розробки.

У нашому веб-застосунку ми використовуємо ряд потужних інструментів розробки, які допомагають нам ефективно керувати процесом розробки та забезпечити високу якість нашого продукту. JavaScript є основною мовою програмування, яку ми використовуємо для створення динамічної та інтерактивної функціональності на стороні клієнта. За допомогою JavaScript ми забезпечуємо зручний та ефективний інтерфейс для наших користувачів.

Одним з ключових інструментів, що використовуємо, є React - потужна бібліотека, яка дозволяє нам розробляти компонентну структуру нашого веб-застосунку. Завдяки React ми можемо розбити складну логіку на невеликі, самодостатні компоненти, які легко керувати, перевикористовувати та тестувати. Крім того, ми використовуємо MUI Components (Material-UI), готові стилізовані компоненти, які допомагають нам швидко та зручно оформляти наш інтерфейс.

У нашому веб-застосунку також активно використовується фреймворк Laravel для розробки потужного та ефективного бекенду. Laravel надає нам широкий набір готових функцій та інструментів для роботи з базами даних, маршрутизації,

аутентифікації та інших важливих аспектів розробки. Це дозволяє нам створювати стабільну та безпечну інфраструктуру для нашого веб-застосунку.

Для забезпечення зручного взаємодії між фронтендом та бекендом ми використовуємо бібліотеку Axios, яка дозволяє нам легко виконувати HTTP-запити та обробляти їх відповіді. Завдяки Axios ми можемо здійснювати асинхронні запити до сервера, обмінюватися даними та ефективно взаємодіяти з нашим бекендом.

Останнім, але не менш важливим, інструментом є Git - система керування версіями. Git дозволяє нам ефективно працювати з кодом, стежити за змінами, створювати гілки для розвитку функціональності та злиття змін без втрати даних. Використовуючи Git, ми можемо спільно працювати над розробкою, відстежувати зміни та забезпечити стабільність нашого кодової бази.

Крім цього, ми використовуємо Visual Studio Code як наш головний редактор коду. Він забезпечує нам зручне та потужне середовище розробки з багатьма корисними функціями, такими як підсвічування синтаксису, автодоповнення, відладка та багато іншого. Visual Studio Code допомагає нам бути продуктивними та ефективними під час розробки нашого веб-застосунку.

Таким чином, забезпечено повну функціональність та ефективність нашого додатку з урахуванням усіх можливих ризиків у процесі розробки, тестування та впровадження системи до роботи на сервері.

4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

Вивчення предметної області, вибір необхідних інструментів розробки та аналіз поставлених завдань є ключовим етапом у плануванні архітектури майбутньої системи. Ці кроки надають нам важливе розуміння того, як система повинна функціонувати та які компоненти і технології варто використовувати для досягнення поставлених цілей. За допомогою цього аналізу ми здатні зрозуміти вимоги користувачів, виявити проблеми, що виникають у наявних системах та знайти оптимальні рішення для розробки майбутньої системи.

4.1 Архітектура системи

Систему розгортання можна описати таким чином: працівник взаємодіє з системою за допомогою програми управління веб додатком. Ця програма виконує запити і отримує відповіді через web-сервер. Web-сервер надає користувацький інтерфейс і використовує інтерфейс бази даних для доступу до даних, зміни, видалення та модифікації (рис. 4.1).

Рисунок 4.1 – Діаграма розгортання системи

Система має двох акторів: адміністратора системи та гостьового користувача. При логіні адміністратор може змінювати наповнення сайту, відвідувати відповідні сторінки з відображенням аналітики даних сайту. Гостьовий користувач Це особа або суб'єкт, який використовує систему для виконання певних завдань або отримання інформації. Користувачі можуть мати різні ролі і повноваження в системі. Вони взаємодіють з інтерфейсом системи, вводять дані, переглядають інформацію, здійснюють пошук, редагують свої налаштування та взаємодіють з функціональністю, доступною для них. Користувачі можуть бути зареєстровані в системі або анонімними. Рисунок 4.2. представляє діаграму прецендентів що описує весь функціонал доступний користувачам системи.

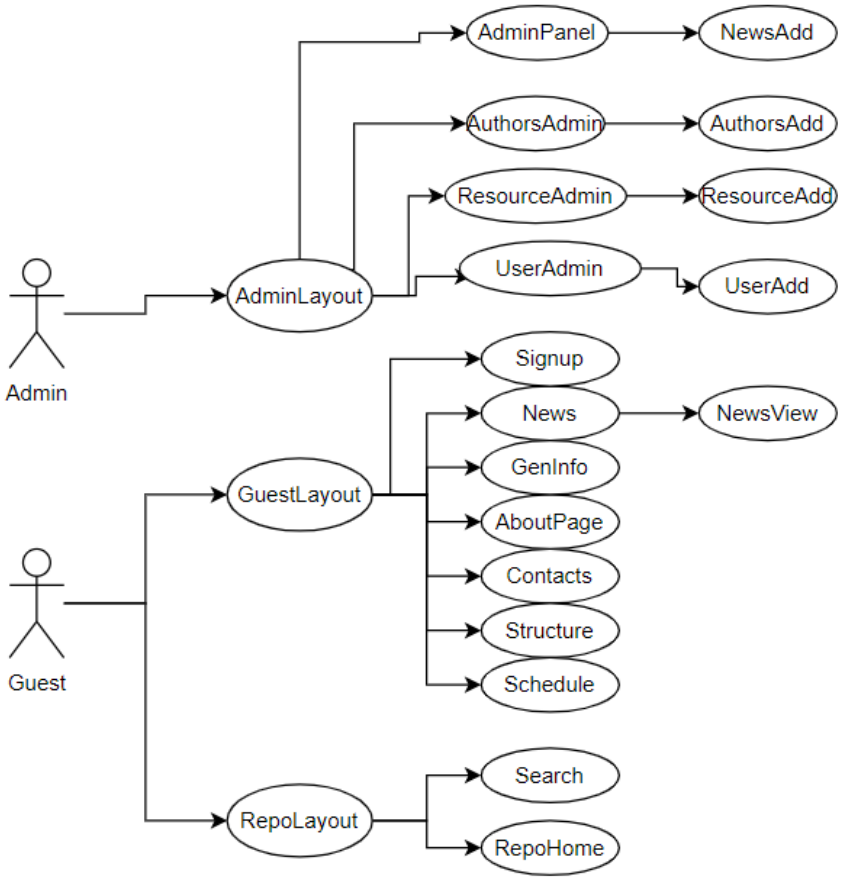

Рисунок 4.2 – Діаграма прецедентів додатку

Представлена архітектура системи задовольняє вимоги технічного завдання та забезпечує користувача повним функціоналом, потрібним для керування ресурсами середньосатистичною бібліотекою.

4.1.1 Single Page Application, або «додаток однієї сторінки»

З використанням React та Laravel у розробці нашого застосунку в методології SPA (Single Page Application) ми можемо отримати значні переваги.

Однією з ключових переваг використання React є його компонентний підхід до розробки інтерфейсу користувача. React дозволяє створювати компоненти, які мають свою внутрішню структуру, стан і логіку. Це спрощує розробку та підтримку коду, оскільки компоненти можуть бути повторно використані, забезпечуючи модульність і зручну організацію функціональності. Крім того, React забезпечує швидкий рендеринг компонентів завдяки використанню віртуального DOM (Document Object Model), що сприяє високій продуктивності застосунку.

Що стосується Laravel, цей фреймворк для розробки на мові PHP надає широкі можливості для побудови потужних серверних додатків. Laravel пропонує зручний маршрутизатор, міграції бази даних, систему шаблонів та інші інструменти, які допомагають швидко створити потрібний функціонал для серверної частини застосунку. За допомогою Laravel можна легко створити API для взаємодії з клієнтською частиною React[10].

Разом React та Laravel утворюють потужну комбінацію для розробки SPA. React забезпечує багатофункціональний та зручний інтерфейс користувача, тоді як Laravel надає потужний серверний функціонал. Застосовуючи методологію SPA, ми можемо зменшити час завантаження сторінок, покращити продуктивність та створити більш інтерактивний користувацький досвід. Крім того, використання React та Laravel спрощує розробку, організацію коду та підтримку застосунку.

Загалом, використання React та Laravel у методології SPA дозволяє нам створити потужний та ефективний веб-додаток, який пропонує зручний інтерфейс користувача та високу продуктивність на стороні клієнта та сервера.

З використанням методології SPA (Single Page Application) у розробці нашого застосунку, ми можемо визначити ряд значних переваг, які наведено нижче.

1. Покращений користувацький досвід: SPA забезпечує більш зручний та плавний користувацький досвід, оскільки немає необхідності в повторному завантаженні сторінок при навігації. Весь вміст відображається динамічно, без

помітних перерв або затримок. Користувачі отримують швидку та безперервну взаємодію з додатком, що поліпшує загальний враження та сприяє збільшенню задоволення від використання.

2. Менше навантаження на сервер: Оскільки в методології SPA сторінки завантажуються один раз при початковому завантаженні, навантаження на сервер значно зменшується. Після першого завантаження, додаток використовує AJAX-запити для отримання необхідних даних, що дозволяє економити пропускну здатність та забезпечує більш швидку відповідь від сервера.

3. Зменшення часу завантаження: Завантаження сторінок в методології SPA відбувається лише один раз, тому час завантаження загального додатку значно зменшується. Це особливо важливо в сучасних умовах, коли користувачі очікують швидкого доступу до інформації та зручного використання додатків навіть при обмежених мережевих умовах.

4. Більша мобільність: SPA ідеально підходить для мобільних пристроїв, оскільки вони мають обмежені розміри екрану та ресурси. Застосунок SPA може бути оптимізований для мобільних пристроїв, забезпечуючи відзивчивий та легкий інтерфейс, а також ефективне використання мережевих ресурсів.

5. Простота розробки та підтримки: Розробка на базі SPA дозволяє розподілити завдання між клієнтською та серверною частинами. Фреймворки, такі як React та Laravel, надають розширені можливості для швидкої та ефективної розробки SPA. Крім того, підтримка та оновлення додатку стають простішими, оскільки вносити зміни можна безпосередньо на клієнтській стороні без необхідності повного перезавантаження сторінок.

6. Більша можливість інтеракції та персоналізації: SPA дозволяє реалізувати більше можливостей для інтеракції з користувачем, таких як асинхронні оновлення, динамічні ефекти та анімація. Це сприяє покращенню залучення користувачів та створенню більш персоналізованого досвіду використання.

7. Узагалі, використання методології SPA разом з React та Laravel в нашому застосунку дозволяє нам отримати значні переваги, такі як покращений користувацький досвід, ефективну роботу з сервером, зменшення часу завантаження,

більшу мобільність, простоту розробки та підтримки, а також більші можливості для інтеракції та персоналізації. Ці переваги допомагають створити потужний та конкурентоспроможний веб-додаток, який задовольняє потреби наших користувачів.

4.2 Структура бази даних

У ході розробки програмного застосунку було змодельовано базу даних для керування даними про наявні ресурси бібілотеки, а саме: книжки, авторів, категорії книжок, видавництво, новини, статичні сторінки веб-сайту, користувачі та їх дані (рис. 4.3).

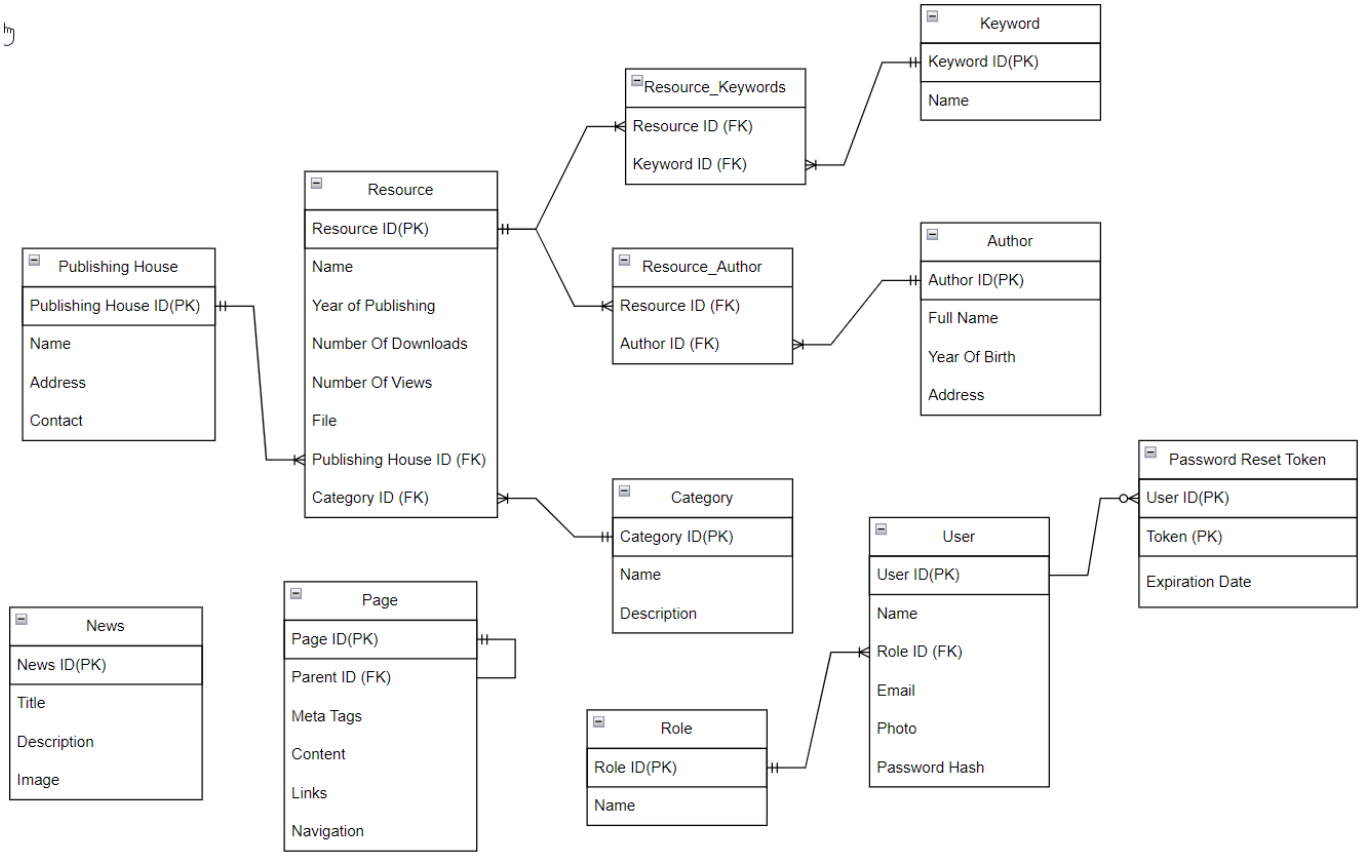

Рисунок 4.3 – Логічна модель бази даних

Оскільки в ході розробки платформи було використано інструмент PHP фреймворку Laravel, Eloquent, усі таблиці було створено за допомогою моделей зі встановленими обмеженнями та деталями для міграції бази даних.

Таблиця ресурсів містить в собі відповідні поля: id, name, year, isbn, num_of_pages, num_of_views, num_of_down , pub_house_id, category_id, більш детально типи змінних, що зберігаються в таблиці ресурсів описано в таблиці 4.1.

Таке використання PHP фреймворку Laravel та Eloquent надає численні переваги у розробці платформи. Laravel забезпечує потужні інструменти для швидкої і ефективної розробки веб-додатків, включаючи систему маршрутизації, обробку запитів та управління сесіями. Eloquent, з свого боку, спрощує взаємодію з базою даних, надаючи зручну ORM (Object-Relational Mapping) для виконання запитів та роботи з моделями даних.

Таблиця 4.1 – Дані таблиці Ресурсів

Назва поля	**Тип**
id	varchar(255)
name	varchar(255)
year	int(11)
file	varchar(255)
num_of_views	int(11)
num_of_down	int(11)
pub_house_id	bigint(20) unsigned
category_id	bigint(20) unsigned

Таблиця ресурсів (Resources) містить в собі наступні поля:

1) id: унікальний ідентифікатор ресурсу;
2) name: назва ресурсу;
3) year: рік випуску ресурсу;
4) file: рядок що містить посилання на ресурс, збережений на сервері репозиторію;
5) num_of_views: кількість переглядів ресурсу;
6) num_of_down: кількість завантажень ресурсу;
7) pub_house_id: ідентифікатор видавництва, пов'язаного з ресурсом;

8) category_id: ідентифікатор категорії, до якої належить ресурс.

Ці змінні дозволяють зберігати інформацію про різноманітні ресурси, такі як книги, статті, документи тощо. Вони допомагають ідентифікувати та описувати кожен ресурс у базі даних, надають можливість відстежувати його властивості, такі як рік випуску, кількість сторінок, популярність та категорію, а також пов'язують ресурс з відповідним видавництвом та категорією.

Створення моделей Eloquent на прикладі таблиці "resources" дозволяє зручно працювати з даними цієї таблиці у зручному об'єктно-орієнтованому стилі. Кожна модель відображає один рядок у таблиці та надає зручні методи для отримання та зміни даних.

Наприклад, для таблиці "resources" можна створити модель "Resource". У цій моделі можна визначити поля, які відповідають полям таблиці, такі як "id", "name", "year",тощо. За допомогою методів Eloquent, таких як find(), all(), where(), можна зручно взаємодіяти з даними цієї таблиці.

Наприклад, для отримання всіх ресурсів з таблиці можна викликати метод all() на моделі "Resource". Якщо потрібно знайти конкретний ресурс за ідентифікатором, можна викликати метод find($id), де $id - ідентифікатор ресурсу.

Таким чином, використання моделей Eloquent дозволяє спростити взаємодію з даними таблиці "resources" і надає зручний спосіб роботи з даними в об'єктно-орієнтованому середовищі.

4.2.1 Застосування зв’язку багато до багатьох

Зв'язок "багато до багатьох" між книгами (ресурсами) та ключовими словами є потужним інструментом для організації та управління даними у нашому додатку. Використання Eloquent в Laravel дозволяє нам легко встановлювати цей зв'язок без необхідності писати складний SQL-код.

Цей зв'язок дозволяє нам прив'язати кілька ключових слів до кожної книги (ресурсу) і навпаки, кілька книг може мати одне й те саме ключове слово. Такий підхід дозволяє нам створювати багатоаспектні та динамічні зв'язки між різними об'єктами.

Зв'язок "багато до багатьох" дає нам багато переваг. Він дозволяє нам ефективно використовувати та організовувати наші дані, забезпечувати гнучкість та масштабованість. Наприклад, ми можемо швидко отримати всі ключові слова, пов'язані з певною книгою, або отримати всі книги, що мають певне ключове слово. Це дозволяє нам легко виконувати пошук, фільтрацію та сортування даних в нашому додатку.

Також використання зв'язку "багато до багатьох" дозволяє нам зберігати наші дані більш організовано та структуровано. Ми можемо легко додавати та видаляти зв'язки між книгами та ключовими словами, без необхідності модифікувати самі таблиці бази даних. Це спрощує розробку та підтримку нашого додатку.

Крім того, зв'язок "багато до багатьох" дозволяє нам забезпечити більш гнучку та масштабовану архітектуру додатку. Ми можемо легко розширювати нашу систему, додавати нові книги та ключові слова, і вони автоматично будуть пов'язані між собою. Це дає нам можливість легко адаптуватись до змін у вимогах та розширювати функціональність нашого додатку без значних зусиль.

Отже, використання зв'язку "багато до багатьох" з допомогою Eloquent в Laravel дозволяє нам створювати потужні та гнучкі зв'язки між книгами (ресурсами) та ключовими словами. Це допомагає нам зберігати, організовувати та легко отримувати доступ до наших даних, робить наш додаток більш гнучким, масштабованим та підтримуваним.

Для створення зв'язку "багато до багатьох" (many-to-many) між книгами (ресурсами) та ключовими словами використовується Eloquent, популярна ORM (Object-Relational Mapping) бібліотека в Laravel. Цей зв'язок дозволяє нам прив'язати декілька ключових слів до кожної книги та зворотно, прив'язати кілька книг до кожного ключового слова.

Для реалізації зв'язку "багато до багатьох" у Laravel використовуються складові описані нижче.

1. Моделі. Створюється модель для книги (ресурсу) та модель для ключового слова. Наприклад, були створені моделі Book та Keyword.

2. Міграції. Створюються міграції для створення таблиць у базі даних. Наприклад, для таблиці книг була створена міграція books, а для таблиці ключових слів - keywords.

3. Таблиця проміжного зв'язку. Створюється третя таблиця у базі даних для зберігання зв'язку між книгами та ключовими словами. Наприклад, була створена міграція book_keyword, яка міститиме зовнішні ключі до таблиць books та keywords.

4. Відношення Eloquent. У моделях Book та Keyword визначаються відношення для зв'язку "багато до багатьох". Для цього можна ми використовуємо метод belongsToMany().

Також, використовуючи той самий підхід було розроблено зв’язок багато до багатьох для забезпечення можливості мати багато авторів для одного ресурсу, задля зручного пошуку та зберігання інформації. Використання зв'язку "багато до багатьох" дозволяє нам прив'язати кількох авторів до кожного ресурсу і навпаки, кожен автор може мати багато ресурсів, забезпечуючи гнучкість та розширюваність нашої системи.

Для реалізації зв'язку "багато до багатьох" між авторами та ресурсами у Laravel, ми можемо використати ту саму концепцію, яку описували раніше для зв'язку "багато до багатьох" між книгами та ключовими словами. Процес створення зв'язку між авторами та ресурсами включає кроки описані нижче.

1. Створюється модель для автора (наприклад, Author) та модель для ресурсу (наприклад, Resource).

2. Створюються міграції для створення таблиць авторів та ресурсів у базі даних.

3. Створюється третя таблиця у базі даних, яка буде містити зв'язок між авторами та ресурсами. У цій таблиці будуть зберігатись зовнішні ключі до таблиць авторів та ресурсів.

4. У моделі Author визначається метод belongsToMany(), який вказує на зв'язок "багато до багатьох" з моделлю Resource. Аналогічно, у моделі Resource визначається метод belongsToMany(), який вказує на зв'язок з моделлю Author.

5. Після визначення відношень ми можемо легко використовувати їх для отримання пов'язаних даних. Наприклад, ми можемо отримати всіх авторів, пов'язаних з певним ресурсом, або всі ресурси, що мають певного автора.

Використання зв'язку "багато до багатьох" дозволяє нам ефективно керувати відношеннями між авторами та ресурсами. Він дозволяє нам легко додавати, видаляти та оновлювати зв'язки між ними. Крім того, цей підхід робить нашу систему гнучкою, оскільки дозволяє мати будь-яку кількість авторів для одного ресурсу та навпаки.

4.2.2 Організація даних користувача системи

Система користувачів, автоматично створена в Laravel, є потужним інструментом для управління аутентифікацією і авторизацією в веб-додатку. Laravel надає вбудовану функціональність для швидкого створення системи користувачів без необхідності писати власний код.

У системі користувачів Laravel зазвичай використовується модель User, яка представляє користувача в базі даних. При створенні нового Laravel проекту за допомогою команди php artisan new project, вже налаштовується таблиця users в базі даних, яка містить необхідні поля для збереження даних користувачів, такі як ім'я, електронна пошта та пароль.

Laravel також надає вбудовану автентифікацію, яка дозволяє користувачам реєструватися, увійти до своїх облікових записів та виконувати різні дії, залежно від їх ролей та дозволів. Аутентифікацію можна налаштувати за допомогою команди php artisan make:auth, яка автоматично створює необхідні маршрути, контролери та представлення для реєстрації, входу, виходу та керування користувачами.

Додатково, Laravel надає різні методи аутентифікації, такі як аутентифікація по електронній пошті, аутентифікація по токену та аутентифікація через соціальні мережі. Це дає можливість реалізувати різноманітні сценарії аутентифікації, в залежності від потреб проекту.

Крім аутентифікації, Laravel надає інструменти для авторизації користувачів, що дозволяють контролювати доступ до різних ресурсів та дій в додатку. Laravel

використовує концепцію ролей та дозволів для управління авторизацією. За допомогою вбудованих механізмів Laravel, можна легко визначити права доступу для користувачів на основі їх ролей та дозволів.

Система користувачів, автоматично створена в Laravel, дозволяє швидко налагодити безпечну та гнучку систему аутентифікації та авторизації для вашого веб-додатку. Вона забезпечує базовий функціонал для роботи з користувачами і може бути легко розширена та налаштована залежно від потреб проекту. Поля таблиці користувача зображені на таблиці 4.2.

Таблиця 4.2 – Дані таблиці користувача

Назва поля	**Тип**
id	varchar(255)
name	varchar(255)
email	varchar(255)
photo	varchar(255)
password_hash	int(11)
role_id	bigint(20) unsigned

У таблиці користувача, створеній в Laravel, є декілька полів, які забезпечують збереження різних атрибутів користувачів:

1) id – це унікальний ідентифікатор користувача. Кожен користувач має унікальне значення id, яке використовується для ідентифікації конкретного запису користувача в базі даних;

2) name – це поле зберігає ім'я користувача. Воно представляє особисті дані користувача і може використовуватись для відображення імені користувача в додатку;

3) email – це поле зберігає електронну пошту користувача. Воно використовується для ідентифікації користувача при аутентифікації і може бути використано для сповіщень та комунікації з користувачем;

4) photo – це поле зберігає посилання на фотографію користувача. Воно може використовуватись для відображення профільного зображення користувача в додатку;

5) password_hash – це поле зберігає хешовану версію паролю користувача. Замість зберігання паролю відкритим текстом, Laravel використовує хешування для безпечного зберігання паролів користувачів. Хешовані паролі не можуть бути відновлені в оригінальний текст, тому це забезпечує безпеку паролів користувачів;

6) role_id – це поле зберігає ідентифікатор ролі користувача. Воно використовується для встановлення рівня доступу та дозволів користувача. Через зв'язок з іншою таблицею, наприклад, з таблицею ролей, можна встановити різні рівні доступу для користувачів.

Ці поля дозволяють зберігати основні дані користувачів і використовуються для реалізації функціоналу аутентифікації, авторизації та керування користувачами в додатку, розробленому з використанням Laravel.

Додатково до таблиці користувачів, в системі автоматично створеній Laravel, може бути і таблиця генерації токенів, яка забезпечує безпечний механізм аутентифікації на основі токенів. Ось опис полів, які можуть бути присутніми в цій таблиці:

1) user_id (primary key) – це поле представляє зв'язок з таблицею користувачів. Воно містить ідентифікатор користувача, з яким пов'язаний токен. Це поле виступає як зовнішній ключ, що вказує на користувача, якому належить токен;

2) token_id (primary key) – це унікальний ідентифікатор токену. Кожен токен має свій унікальний ідентифікатор, який дозволяє однозначно ідентифікувати його в системі;

3) expiration_date – це поле вказує на дату і час, коли токен стане недійсним або припинить свою дію. Воно використовується для контролю терміну дії токену і забезпечення безпеки. Після закінчення терміну дії токен не може бути використаний для аутентифікації.

В контексті таблиці токенів, використання подвійного ключа може бути корисним для забезпечення унікальності та ефективної організації даних.

У нашому випадку, де ми маємо таблицю "Токени", яка зберігає інформацію про токени аутентифікації, подвійний ключ може складатися з двох полів: user_id та token_id.

user_id використовується для встановлення зв'язку з таблицею "Користувачі". Воно містить ідентифікатор користувача, якому належить токен. Це поле дозволяє нам легко відстежувати, який користувач належить до кожного токена.

token_id – унікальний ідентифікатор токена, який дозволяє однозначно ідентифікувати кожен токен в системі. Використання подвійного ключа з цим полем допомагає забезпечити унікальність комбінації user_id та token_id, що гарантує, що кожен токен має свій унікальний ідентифікатор в контексті користувача.

Додатково до цих полів, можуть бути присутні й інші поля в таблиці, такі як expiration_date, що вказує на термін дії токена, або інші додаткові дані, які можуть бути корисні для функціональності системи токенів.

Використання подвійного ключа в таблиці токенів дозволяє нам ефективно зберігати і керувати інформацією про токени аутентифікації. Він забезпечує унікальність комбінації user_id та token_id, що допомагає точно ідентифікувати кожен токен і його власника.

4.3 Трансформування даних у формат JSON

Задля можливості використовувати дані з бази даних у наших React-компонентах, маємо забезпечити надійну трансформацію даних у формат JSON.

Застосування ресурсів (resources) у Laravel дозволяє легко трансформувати дані моделей у формат JSON. Ресурси визначаються для кожної моделі та вказують, які поля повинні бути включені у вихідному JSON-відповідь.

Наприклад, для моделі "Book" було створено відповідний ресурс, "BookResource". У цьому ресурсі вказано, які поля потрібно включити у вихідний JSON-об'єкт, а також визначити будь-які додаткові перетворення або маніпуляції з даними.

Для використання ресурсів у контролері можна викликати метод BookResource::collection($books), де $books - колекція моделей "Book". Цей метод поверне колекцію ресурсів у вигляді JSON.

Крім того, Laravel також надає можливість використовувати ресурси для одного конкретного екземпляра моделі. Наприклад, можна викликати метод new BookResource($book), де $book - конкретний екземпляр моделі "Book", щоб отримати його представлення у вигляді JSON.

Застосування ресурсів у Laravel дозволяє зручно контролювати трансформацію даних у формат JSON та надає гнучкість для додавання додаткової логіки або маніпуляцій з даними перед їх відправкою у відповідь.

Відповідно після успішного отримання даних за допомогою хешування AxiosClient у наших компонентах, отримані дані з таблиці бази даних матимуть вигляд js масиву.

4.4 Розробка адміністративної частини сайту

Адміністративна частина нашого веб-застосунку дозволяє працівникам бібліотеки та технічним адміністраторам ефективно керувати даними, які відображаються на сторінках веб-сайту. Вона надає можливість додавати нову інформацію про книжки (ресурси), авторів, а також переглядати аналітику переглядів та завантажень документів.

Одним з основних завдань адміністративної частини є забезпечення безпеки та контролю доступу. Щоб досягти цього, ми впровадили надійну систему авторизації на бекенді нашого веб-застосунку. Це означає, що лише авторизовані користувачі, такі як працівники бібліотеки та адміністратори, мають доступ до адміністративної частини.

Структура системи була побудована на основі типів сторінок та рівнів доступу до них. Використовуючи компонент Outlet бібліотеки React, ми змогли створити шаблони для кожної частини веб-застосунку. Це дозволило нам легко розподіляти та перевикористовувати компоненти, уникнувши копіювання коду для кожної окремої

сторінки. Замість цього, ми могли використовувати загальні компоненти і динамічно змінювати їх вміст залежно від поточного контексту та доступу користувача.

Такий підхід до структури та управління компонентами дозволив нам зекономити час і зусилля на розробці, підтримці та оновленні адміністративної частини. Крім того, це спростило розширення функціональності, оскільки нам не потрібно було вносити зміни в кожну окрему сторінку, а лише відповідні компоненти.

При кожному завантаженні будь-якої сторінки нашого застосунку відповідний компонент Router у головному рендері перенаправляє нас до певних шаблонів сторінок в залежності від рівня доступу, забезпеченого нашому користувачу. Дані про користувача ми актуалізуємо за допомогою генерації та збереження токенів за допомогою userContext компоненту, що забезпечує збереження «поточного стану» застосунку та контроль змін, що могли відбутися[5, 8].

Таким чином, при зміні стану нашого контексту сторінка оновлюватиметься иа компонент Router перенаправлятиме нас до різних видів шаблонів в залежності від дії яку було впроваджено.

Прикладом такої дії може бути вихід з системи. Функція кнопки виходу забезпечує очищення поточного токена користувача, його об’єкту в цілому, відповідно після цієї зміни стану користувач буде перенаправлений до гостьової сторінки адмін панелі(форми реєстрації).

Також варто зазначити, що система додавання, редагування та видалення також працює за допомогою використання стану поточно сторінки. До прикладу, якщо ми хочемо додати новий ресурс до нашої бібліотеки, ми повинні отримати пусту форму з відповідними полями до заповнення користувачем. Задля використання одного і того ж самого компоненту для редагування будь якого елементу нашої системи ми використовуємо стан сторінки, який перевіряє запит який ми використовуємо для доступу до сторінки та чи було передано по цьому запиту реєстраційний номер потрібного нам елемента для редагування (id). Якщо параметр було передано до форми, сторінка заповнює поля відповідними значеннями, отриманими з бази даних, популюючи об’єкт заданий на початку ініціалізації сторінки. Таким чином наш додаток не перенасичений додатковими сторінками на кожен випадок потрібного нам

функціоналу, а ефективно перевикористовує компонент форми для додавання елементів до бази даних для редагування вмісту кожного об’єкту по нашому запиту.

Висновки до розділу 4

Результатом нашої розробки є веб-застосунок, який має добре організовану архітектуру системи. Ми використовуємо популярний фреймворк Laravel, який дозволяє нам розбити логіку застосунку на окремі компоненти та модулі, що полегшує розробку, тестування та розширення.

У нашій системі ми використовуємо базу даних, яка була розроблена за допомогою Eloquent ORM (Object-Relational Mapping). Це дозволяє нам працювати з базою даних, використовуючи об'єктно-орієнтований підхід, замість написання складних SQL-запитів. Eloquent надає зручні методи для створення, читання, оновлення та видалення даних, спрощуючи взаємодію з базою даних.

Особливу увагу ми приділили розробці адміністративної панелі з надійною авторизацією. Це забезпечує, що доступ до адміністративної частини застосунку мають лише працівники бібліотеки та технічні адміністратори. Завдяки розумній системі авторизації, ми забезпечуємо конфіденційність та безпеку наших даних.

У підсумку, реалізація нашого веб-застосунку заслуговує високої оцінки. Була ретельно опрацьовано архітектуру системи, розроблено базу даних за допомогою Eloquent, а також створили надійну адміністративну панель з міцною системою авторизації. Ці елементи спільно працюють для забезпечення ефективного управління бібліотечними ресурсами, забезпечуючи нам зручний інтерфейс, аналітику та контроль над даними.

5 РОБОТА КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

5.1 Системні вимоги

Для доступу до реактивного застосунку, розробленого з використанням React, необхідно, щоб браузер користувача задовольняв певні системні вимоги, але зазвичай сучасні версії популярних браузерів задовольняють ці системні вимоги. Рекомендується користуватись оновленими версіями браузерів для найкращої сумісності та продуктивності реактивних застосунків.

5.2 Взаємодія з веб застосунком

На головній сторінці веб застосунку можна ознайомитись з новинами бібліотеки та інформаційними розділами, доступним усім користувачам(рис. 5.1.)

Рисунок 5.1 – Головна сторінка веб застосунку

Також, за допомогою посилань розташованих на головній сторінці веб застосунку, користувач може безпосередньо скористатися пошуком ресурсів з репозиторію. Приклад результатів пошуку зображений на рисунку 5.2.

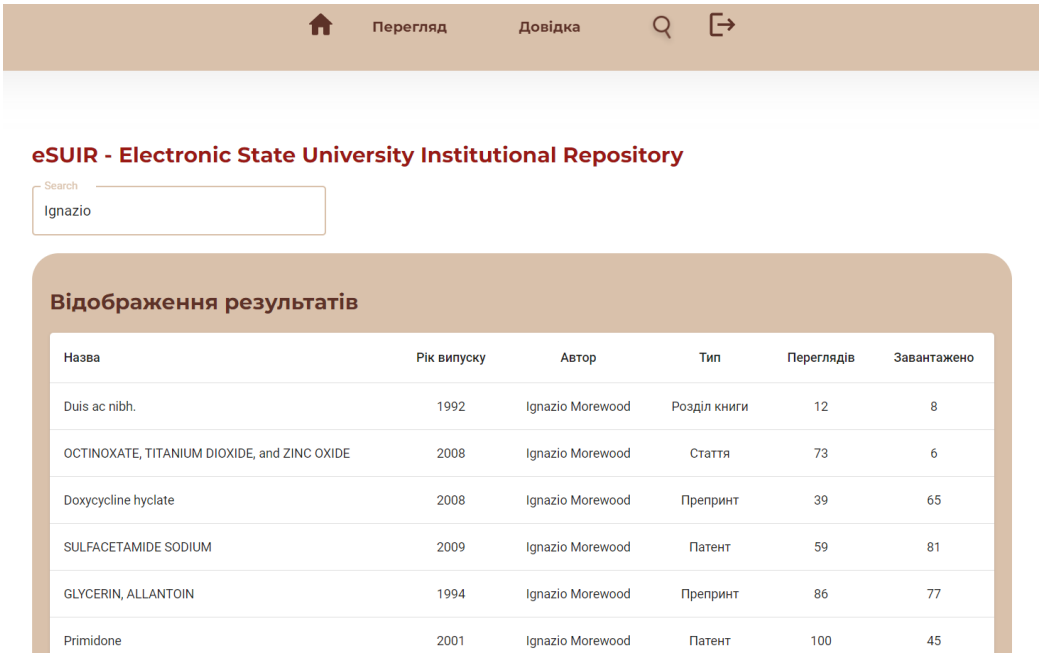

Рисунок 5.2 – Результати пошуку

Якщо користувач хоче отримати доступ до адмін панелі веб застосунку, він має пройти аутентифікацію через відповідну форму логіну, яка з’являється при намаганні доступу до маршруту ‘/admin’ неавторизованим користувачем(рис. 5.3.)

Рисунок 5.3 – Форма логіну до адмін панелі веб застосунку

При успішній авторизації користувача, нас перенаправить на сторінку управлінням даними про книжки, авторів, видання, користувачів та категорії ресурсів бібліотеки, як зображено на рисунку 5.4.

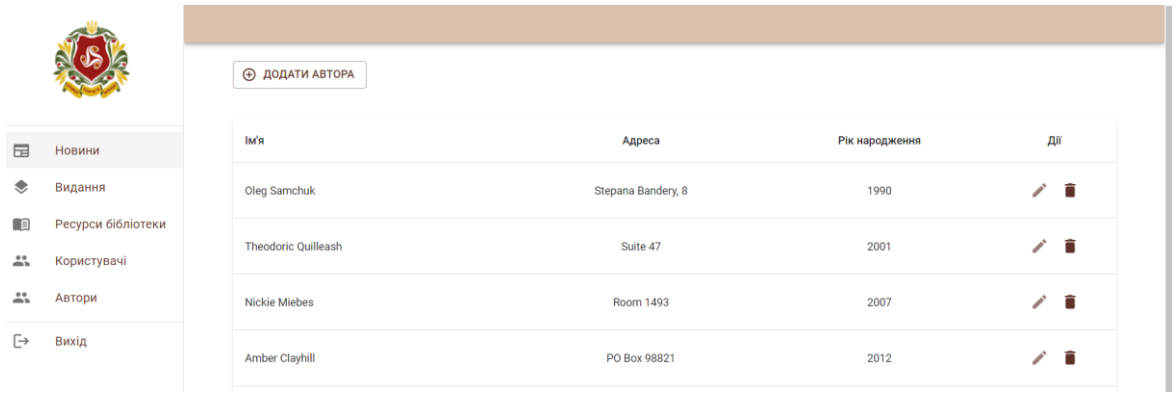

Рисунок 5.4 – Сторінка управлінням даними веб застосунку

Також, розглянемо форми до заповнення алмінітратором веб-платформи. Прикладом однієї з таких форм є форма додавання новин на наш розділ новин веб-сайту (рис 5.5).

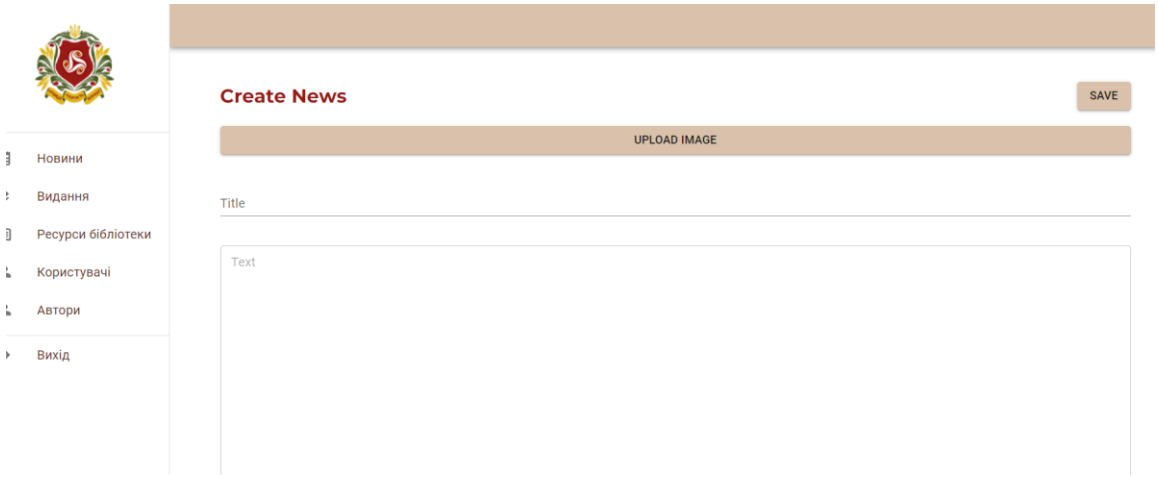

Рисунок 5.5 – Форма додавання новин на розділ новин

Таким чином, дизайн повністю відповідає інтерфейсу програми та забезпечує користувача усіма можливими функціями керування ресурсами бібліотеки.

Після використання форми, заповнення усі відповідних полей, новини з'являтимуться на екранні управлінням контенту розділу новин та, в свою чергу, безпосередньо у розділі новин для відвідувачів веб-сайту.

Висновки до розділу 5

В ході дослідження та аналізу розділу про роботу користувача з нашою системою, системними вимогами та використанням імплементованого користувацького інтерфейсу, було винесено кілька важливих висновків.

По-перше, розроблений користувацький інтерфейс системи є ефективним та інтуїтивно зрозумілим. Він надає зручну навігацію та логіку взаємодії з різними функціями системи. Користувачі можуть швидко звикнути до інтерфейсу та легко знаходити необхідні опції та функціонал.

По-друге, системні вимоги до нашої системи досить зрозумілі та доступні для більшості користувачів. Вони включають необхідні характеристики апаратного та програмного забезпечення, які забезпечують надійну та ефективну роботу системи. Зазначені системні вимоги дозволяють користувачам використовувати систему на різних платформах та пристроях з високою продуктивністю та безпекою.

По-третє, робота користувача з нашою системою є зручною та ефективною. Імплементований користувацький інтерфейс дозволяє користувачам швидко та легко взаємодіяти з різними функціями системи. Наприклад, користувачі можуть додавати, редагувати та видаляти дані про ресурси бібліотеки, використовувати інтегрований пошук для знаходження необхідних матеріалів та контролювати свої позички.

Загалом, аналіз розділу про роботу користувача з нашою системою, системними вимогами та використанням імплементованого користувацького інтерфейсу підтверджує, що розроблена система відповідає потребам користувачів та вимогам університетської бібліотеки. Користувачі зможуть легко орієнтуватися в системі, використовувати її функціонал та задовольняти свої інформаційні потреби. Дотримання системних вимог забезпечить стабільну та ефективну роботу системи на різних платформах та пристроях. Розроблений інтерфейс сприятиме зручності та задоволенню користувачів при використанні системи.

ВИСНОВКИ

У процесі роботи було проаналізовано всі існуючі сайти та платформи керуваннями ресурсами бібліотеки, виявити недоліки та переваги систем, та сформульовано вимоги для створення нової системи.

Програмний продукт розроблений протягом цієї роботи має вирішити поставлені задачі, виправити недоліки інших платформ, максимально задовільнити користувацькі потреби.

Було проведено огляд засобів розробки програмного продукту та обґрунтування вибору цих засобів. Поєднання обраних технологій забезпечує швидку та синхронізовану роботи системи, доступність та зрозумілість.

Результатом цієї веб додаток з інтеграцією бази даних для збереження даних у відповідності з розробленою архітектурою.

Впровадження такої системи також сприятиме автоматизації бібліотечних процесів, таких як оновлення книжкового фонду, замовлення нових книг та моніторинг популярності та використання різних видань. Це допоможе забезпечити оптимальне використання ресурсів та підвищить якість обслуговування користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Vickler A. Javascript: javascript front end programming. Independently Published, 2021 – 426 с.
2. Сучасний підручник з JavaScript : веб-сайт. URL: https://uk.javascript.info/ (дата звернення: 12.05.2023)
3. React.js Documentation: веб-сайт. URL: https://react.dev/ (дата звернення: 12.05.2023)
4. MySQL Documentation: веб-сайт. URL: https://dev.mysql.com/doc/ (дата звернення: 12.05.2023)
5. Build and Deploy React + Laravel project in 10 hours: веб-сайт. URL: https://youtu.be/bHRe5XNP5l8 (дата звернення: 13.05.2023)
6. Laravel Documentation: веб-сайт. URL: https://laravel.com/docs/ (дата звернення: 13.05.2023)
7. MUI Core Docs: веб-сайт. URL: https://mui.com/material-ui/getting-started/overview/ (дата звернення: 13.05.2023)
8. React + Laravel Full-stack Application | Build and Deploy: веб-сайт. URL: https://youtu.be/qJq9ZMB2Was (дата звернення: 13.05.2023)
9. PhpMyAdmin Documentation: веб-сайт. URL: https://www.phpmyadmin.net/docs/ (дата звернення: 13.05.2023)
10. Laravel API Tutorial: веб-сайт. URL: https://magecomp.com/blog/laravel-8-create-application-programing-interface/ (дата звернення: 13.05.2023)

ДОДАТОК А

Автоматизація бібліотечних процесів

Програмний код

Аркушів 7

2023

```
Route::middleware('auth:sanctum')->group(function () {
    Route::post('/logout', [AuthController::class, 'logout']);
    Route::apiResource('users', UserController::class);
});

Route::apiResource('news', NewsController::class);
Route::post('/signup', [AuthController::class, 'signup']);
Route::post('/login', [AuthController::class, 'login']);
Route::apiResource('category', CategoryController::class);
Route::apiResource('authors', AuthorController::class);
Route::apiResource('resources', BookController::class);
Route::apiResource('keywords', KeyWordController::class);
Route::apiResource('pubhouse', PubHouseController::class);
Route::get('/news/get-by-slug/{news:slug}', [NewsController::class, 'getBySlug']);
import { Box, Button, Container, TextField, Typography } from '@mui/material'
import React, { useEffect, useState } from 'react'
import axiosClient from '../../axios';
import { useNavigate, useParams } from 'react-router-dom';
import { userStateContext } from '../../contexts/ContextProvider';

function NewsAdd() {
    const [news, setNews] = useState({
        title: "",
        slug: "",
        body: "",
        image: null,
        image_url: null,
    });
    const { id } = useParams();
    const { setLoad } = userStateContext();
    const navigate = useNavigate();

    useEffect(() => {
        if (id) {
            setLoad(true);
            axiosClient.get(`/news/${id}`).then(({ data }) => {
                setNews(data.data);
                setLoad(false);
            });
        }
    }, []);

    const onImageChoose = (ev) => {
        const file = ev.target.files[0];

        const reader = new FileReader();
        reader.onload = () => {
            setNews({
                ...news,
                image: file,
                image_url: reader.result,
            });
```

```
            ev.target.value = "";
        };
        reader.readAsDataURL(file);
    };

    function handleSubmit(ev) {
        ev.preventDefault();

        const payload = { ...news };
        if (payload.image) {
            payload.image = payload.image_url;
        }
        delete payload.image_url;
        let res = null;
        if (id) {
            res = axiosClient.put(`/news/${id}`, payload);
        } else {
            res = axiosClient.post("/news", payload);
        }
        res.then((res) => {
            console.log(res);
            navigate("/admin/news");
        })
            .catch((err) => {
                if (err && err.response) {
                    setError(err.response.data.message);
                }
                console.log(err, err.response);
            });
    };

    return (
        <Container>
            <Box component='form' onSubmit={handleSubmit} display={'flex'} flexDirection={'column'}
justifyContent={'space-between'}>
                <Box display={'flex'} flexDirection={'row'} justifyContent={'space-between'} alignItems={'center'}>
                    <h2>Create News</h2>
                    <Button
                        color="primary" type="submit"
                        variant="contained">
                        Save
                    </Button>
                </Box>
                {news.image_url && (
                    <img
                        src={news.image_url}
                        alt=""
                        width={500}
                        style={{ margin: "auto", padding: "30px" }}
                    />
                )}
                <Button
                    variant="contained"
                    component="label"
                    sx={{ marginBottom: "30px" }}
                >
                    Upload Image
                    <input
                        type="file"
                        hidden
                        onChange={onImageChoose}
                    />
                </Button>
```

```
            <TextField id="outlined-basic" label="Title" variant="standard" sx={{ marginBottom: "30px" }}
              name="title"
              value={news.title}
              onChange={(ev) =>
                setNews({ ...news, title: ev.target.value })
              }
            />
            <TextField
              placeholder="Text"
              multiline
              minRows={20}
              maxRows={Infinity}
              sx={{ height: "500px" }}
              value={news.body}
              onChange={(ev) =>
                setNews({ ...news, body: ev.target.value })
              }
            />
          </Box>
        </Container>
    )
}

export default NewsAdd

    function CustomPaginationActionsTable() {
      const { setLoad } = userStateContext();
      const [page, setPage] = useState(0);
      const [rowsPerPage, setRowsPerPage] = useState(10);
      const [autors, setResources] = useState([]);

      const getAuthors = () => {
        axiosClient.get('/resources').then(({ data }) => {
          setResources(data.data);
          setLoad(false);
        })
      }

      const deleteResources = (id) => {
        axiosClient.delete(`/resources/${id}`);
        const auxN = autors.filter((n) => {
          return n.id != id;
        });
        setResources(auxN);
        setPage(0);
      }

      useEffect(() => {
        getAuthors();
      }, []);

      // Avoid a layout jump when reaching the last page with empty rows.
      const emptyRows =
        page > 0 ? Math.max(0, (1 + page) * rowsPerPage - autors.length) : 0;

      const handleChangePage = (event, newPage) => {
        setPage(newPage);
      };

      const handleChangeRowsPerPage = (event) => {
        setRowsPerPage(parseInt(event.target.value, 10));
        setPage(0);
      };
```

```
return (
  <TableContainer component={Paper}>
    <Table sx={{ minWidth: 500 }} aria-label="custom pagination table">
      <TableHead>
        <TableCell component="th" scope="row">
          Назва
        </TableCell>
        <TableCell align="center" component="th" scope="row">
          Рік видання
        </TableCell>
        <TableCell align="center" component="th" scope="row">
          ISBN
        </TableCell>
        <TableCell align="center" component="th" scope="row">
          Кількість сторінок
        </TableCell>
        <TableCell align="center" component="th" scope="row">
          Кількість переглядів
        </TableCell>
        <TableCell align="center" component="th" scope="row">
          Кількість завантажень
        </TableCell>
        <TableCell align="center" component="th" scope="row">
          Дії
        </TableCell>
      </TableHead>
      <TableBody>
        {(rowsPerPage > 0
          ? autors.slice(page * rowsPerPage, page * rowsPerPage + rowsPerPage)
          : autors
        ).map((row) => (
          <TableRow key={row.id}>
            <TableCell width={400} component="th" scope="row">
              {row.name}
            </TableCell>
            <TableCell align="center">
              {row.year}
            </TableCell>
            <TableCell align="center">
              {row.isbn}
            </TableCell>
            <TableCell align="center">
              {row.num_of_pages}
            </TableCell>
            <TableCell align="center">
              {row.num_of_views}
            </TableCell>
            <TableCell align="center">
              {row.num_of_down}
            </TableCell>
            <TableCell style={{ width: 160 }} align="center">
              <IconButton color='secondary' component={Link} to={`/admin/autors/${row.id}`}>
                <EditRoundedIcon></EditRoundedIcon>
              </IconButton>
              <IconButton color='darker' onClick={() => deleteResources(row.id)}>
                <DeleteRoundedIcon></DeleteRoundedIcon>
              </IconButton>
            </TableCell>
          </TableRow>
        ))}

        {emptyRows > 0 && (
```

```
                <TableRow style={{ height: 53 * emptyRows }}>
                  <TableCell colSpan={6} />
                </TableRow>
              )}
            </TableBody>
            <TableFooter>
              <TableRow>
                <TablePagination
                  rowsPerPageOptions={[10, 15, 25, { label: 'All', value: -1 }]}
                  colSpan={3}
                  count={autors.length}
                  rowsPerPage={rowsPerPage}
                  page={page}
                  SelectProps={{
                    inputProps: {
                      'aria-label': 'rows per page',
                    },
                    native: true,
                  }}
                  onPageChange={handleChangePage}
                  onRowsPerPageChange={handleChangeRowsPerPage}
                  ActionsComponent={TablePaginationActions}
                />
              </TableRow>
            </TableFooter>
          </Table>
        </TableContainer>
    );
}

function ResourceAdmin() {
    return (
        <Container>
            <Box paddingBottom={5}>
                <NavLink to={'/admin/resource/add'}>
                    <Button startIcon={<ControlPointRoundedIcon></ControlPointRoundedIcon>} color='darker'
variant="outlined" >Додати книжку</Button>
                </NavLink>
            </Box>
            <CustomPaginationActionsTable></CustomPaginationActionsTable>
        </Container>
    )
}

function CustomPaginationActionsTable() {
    const { setLoad } = userStateContext();
    const [page, setPage] = useState(0);
    const [rowsPerPage, setRowsPerPage] = useState(10);
    const [news, setNews] = useState([]);

    const getNews = () => {
        axiosClient.get('/news').then(({ data }) => {
            setNews(data.data);
            setLoad(false);
        })
    }

    const deleteNews = (id) => {
        axiosClient.delete(`/news/${id}`);
        const auxN = news.filter((n) => {
            return n.id != id;
        });
        setNews(auxN);
```

```
    setPage(0);
  }

  useEffect(() => {
    getNews();
  }, []);

  // Avoid a layout jump when reaching the last page with empty rows.
  const emptyRows =
    page > 0 ? Math.max(0, (1 + page) * rowsPerPage - news.length) : 0;

  const handleChangePage = (event, newPage) => {
    setPage(newPage);
  };

  const handleChangeRowsPerPage = (event) => {
    setRowsPerPage(parseInt(event.target.value, 10));
    setPage(0);
  };

  return (
    <TableContainer component={Paper}>
      <Table sx={{ minWidth: 500 }} aria-label="custom pagination table">
        <TableHead>
          <TableCell component="th" scope="row">
            Назва
          </TableCell>
          <TableCell align="center" component="th" scope="row">
            Опис
          </TableCell>
          <TableCell align="center" component="th" scope="row">
            Дата створення
          </TableCell>
          <TableCell align="center" component="th" scope="row">
            Дата редагування
          </TableCell>
          <TableCell align="center" component="th" scope="row">
            Дії
          </TableCell>
        </TableHead>
        <TableBody>
          {(rowsPerPage > 0
            ? news.slice(page * rowsPerPage, page * rowsPerPage + rowsPerPage)
            : news
          ).map((row) => (
            <TableRow key={row.id}>
              <TableCell component="th" scope="row">
                {row.title}
              </TableCell>
              <TableCell style={{ width: 190 }} align="center">
                {row.body.slice(0, 20) + '...'}
              </TableCell>
              <TableCell style={{ width: 190 }} align="center">
                {row.created_at}
              </TableCell>
              <TableCell style={{ width: 190 }} align="center">
                {row.updated_at}
              </TableCell>
              <TableCell style={{ width: 160 }} align="center">
                <IconButton color='secondary' component={Link} to={`/admin/news/${row.id}`}>
                  <EditRoundedIcon></EditRoundedIcon>
                </IconButton>
                <IconButton color='darker' onClick={() => deleteNews(row.id)}>
```

```
                        <DeleteRoundedIcon></DeleteRoundedIcon>
                      </IconButton>
                    </TableCell>
                  </TableRow>
                ))}

                {emptyRows > 0 && (
                  <TableRow style={{ height: 53 * emptyRows }}>
                    <TableCell colSpan={6} />
                  </TableRow>
                )}
              </TableBody>
              <TableFooter>
                <TableRow>
                  <TablePagination
                    rowsPerPageOptions={[10, 15, 25, { label: 'All', value: -1 }]}
                    colSpan={3}
                    count={news.length}
                    rowsPerPage={rowsPerPage}
                    page={page}
                    SelectProps={{
                      inputProps: {
                        'aria-label': 'rows per page',
                      },
                      native: true,
                    }}
                    onPageChange={handleChangePage}
                    onRowsPerPageChange={handleChangeRowsPerPage}
                    ActionsComponent={TablePaginationActions}
                  />
                </TableRow>
              </TableFooter>
            </Table>
          </TableContainer>
        );
}

function NewsAdmin() {

  return (
    <Container>
      <Box paddingBottom={5}>
        <NavLink to={'/admin/news/add'}>
          <Button startIcon={<ControlPointRoundedIcon></ControlPointRoundedIcon>} color='darker'
variant="outlined" >Додати новину</Button>
        </NavLink>
      </Box>
      <CustomPaginationActionsTable></CustomPaginationActionsTable>
    </Container>
  )
}

export default NewsAdmin
```