Військовий інститут телекомунікацій та інформатизації імені Героїв Крут

Кафедра

“Комп’ютерних інформаційних технологій”

Курсовий проект

З дисципліни: “Технологія розробки програмного забезпечення”

на тему: “Бібліотека”

Виконали:

Курсант 2002 навчальної групи,

штаб-сержант Олександр НЕСТЕРЕНКО

Керівник:

Старший викладач кафедри № 22, к.т.н

підполковник Сергій РОМАНЕНКО

Київ 2024

# АНОТАЦІЯ

курсового проекту на тему: “Бібліотека“.

Курсова робота містить: 66 сторінок, 19 рисунків, 35 джерел.

В роботі наведено аналіз існуючих підходів до створення інформаційних систем для управління бібліотекою. Спроектовано архітектуру програмного додатку, розроблено *UML*-діаграми для опису процесів взаємодії користувача з системою та варіанти реалізації функціональних модулів.

Розроблено клієнт-серверний веб-застосунок для управління бібліотечним фондом, обліку книг, ведення інформації про читачів та надання доступу до пошуку і стану книг.

*ANNOTATION*

*of the course project on the topic: "Library"*

*Term paper contains: 66 pages, 19 figures, 35 sources.*

*The paper analyses existing approaches to creating information systems for library management. The architecture of the software application is designed, UML diagrams are developed to describe the processes of user interaction with the system and options for implementing functional modules.*

*A client-server web application has been developed for managing the library collection, accounting for books, maintaining information about readers and providing access to search and status of books.*

ЗМІСТ

[АНОТАЦІЯ 2](#_Toc179302633)

[ЗМІСТ 4](#_Toc179302634)

[ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ 5](#_Toc179302635)

[ВСТУП 6](#_Toc179302636)

[РОЗДІЛ 1 АНАЛІЗ БІБЛІОТЕКИ 8](#_Toc179302637)

[1.1 Аналіз бібліотки 8](#_Toc179302638)

[1.2 Аналіз існуючих підходів до вирішення завдання 9](#_Toc179302639)

[Висновок до розділу 1 10](#_Toc179302640)

[РОЗДІЛ 2 ПРОЕКТУВАННЯ І ОСОБЛИВОСТІ РОЗРОБКИ ВЕБ-ДОДАТКУ 11](#_Toc179302641)

[2.1 Вимоги до програми 11](#_Toc179302642)

[2.2 Системні вимоги програми 11](#_Toc179302643)

[2.3 Діаграми та схеми 12](#_Toc179302644)

[2.4 Особливості програмної реалізації 14](#_Toc179302645)

[Висновок до розділу 2 23](#_Toc179302646)

[РОЗДІЛ 3 ОСОБЛИВОСТІ ВИКОРИСТАННЯ ВЕБ-ДОДАТКУ 24](#_Toc179302647)

[3.1 Порядок використання програмного додатку 24](#_Toc179302648)

[Висновок до розділу 3 49](#_Toc179302649)

[ВИСНОВКИ 50](#_Toc179302650)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 51](#_Toc179302651)

[ДОДАТОК А 53](#_Toc179302652)

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

*UI – User Interface*

*UML – Unified Modeling Language*

*ID - Indentifier*

БД – База даних

*API - Application Programming Interface*

*CRUD – Create Read Update Delete*

# ВСТУП

У сучасних умовах функціонування бібліотек важливою складовою їхньої успішної діяльності є ефективне управління бібліотечним фондом та інформаційними потоками. Зокрема, бібліотека відіграє ключову роль у забезпеченні доступу до знань, що зумовлює необхідність впровадження сучасних технологічних рішень для оптимізації її діяльності.

Однією з найважливіших задач, що стоять перед бібліотекою, є забезпечення якісного обліку літератури та управління інформацією про читачів. Цей аспект є надзвичайно важливим для забезпечення належного функціонування бібліотеки, а також для оптимізації використання ресурсів. Впровадження системи обліку дозволить не лише підвищити точність та оперативність обліку, але й зменшити ризики втрат та нераціонального використання книжкового фонду.

Крім того, в умовах зростаючого обсягу інформації та потреби у швидкому пошуку необхідної літератури виникає потреба в розробці ефективної системи пошуку та резервування книжок. Така система забезпечить своєчасний доступ читачів до необхідних матеріалів, а також сприятиме покращенню якості обслуговування користувачів.

Метою даної курсової роботи є формування умінь з розробки та створення сучасних веб-застосунків для обраної предметної області на основі отриманих теоретичних знань. Основні завдання включають проектування архітектури системи, розробку серверної частини та клієнтського інтерфейсу, а також забезпечення інтеграції з існуючими базами даних та системами безпеки.

В роботі наведено аналіз існуючих підходів до створення інформаційних систем для управління бібліотекою. Спроектовано архітектуру програмного додатку, розроблено *UML*-діаграми для опису процесів взаємодії користувача з системою та варіанти реалізації функціональних модулів.

Розроблено клієнт-серверний веб-застосунок для управління бібліотечним фондом, обліку книг, ведення інформації про читачів та надання доступу до пошуку і резервування літератури.

РОЗДІЛ 1 АНАЛІЗ БІБЛІОТЕКИ

* 1. Аналіз бібліотеки

Бібліотека є важливою складовою сучасного освітнього та культурного середовища, яка забезпечує доступ до інформаційних ресурсів і сприяє збереженню знань. Вона виконує роль центру зберігання, систематизації та надання інформаційних матеріалів для широкого кола користувачів, включаючи студентів, науковців, викладачів та інших зацікавлених осіб.

У сучасних умовах швидкого розвитку інформаційних технологій і збільшення обсягу цифрових даних виникає необхідність у автоматизації бібліотечних процесів. Це стосується як збереження, обробки та аналізу даних про наявні ресурси, так і представлення інформації через сучасні веб-технології. Автоматизація допомагає підвищити ефективність управління бібліотекою, знизити витрати часу на пошук і обробку інформації та покращити якість обслуговування користувачів.

Основні функції бібліотеки включають:

─ Надання доступу до літературних ресурсів: Бібліотека надає доступ до книг, періодичних видань, електронних ресурсів та інших матеріалів, необхідних для навчання, досліджень і саморозвитку користувачів.

─ Управління бібліотечним фондом: Включає облік і систематизацію книжкового фонду, моніторинг стану літератури, а також організацію її оновлення відповідно до потреб користувачів.

─ Обслуговування читачів: Забезпечення своєчасного і зручного доступу до необхідних матеріалів, консультації з пошуку літератури, а також організація заходів, спрямованих на популяризацію читання.

─ Інформаційна підтримка: Надає допомогу користувачам у пошуку та використанні інформаційних ресурсів, забезпечує навчання навичкам інформаційної грамотності.

─ Організація культурно-освітніх заходів: Проведення лекцій, семінарів, презентацій книг та інших заходів, які сприяють розвитку культурного і освітнього рівня громади.

З огляду на специфіку діяльності бібліотеки, а також вимоги до курсової роботи, основні потреби в автоматизації можна виділити наступним чином:

─ Система обліку літератури: Оскільки бібліотека має значний обсяг книжкового фонду, необхідно мати ефективний інструмент для ведення обліку. Це допоможе забезпечити точний контроль за наявністю, станом та переміщенням книг.

─ Модуль пошуку і резервування літератури: Для забезпечення зручного доступу користувачів до книжкового фонду, необхідно впровадити ефективний механізм пошуку книг за різними критеріями, а також можливість резервування літератури.

─ Модуль управління інформацією про читачів: Включає ведення обліку зареєстрованих читачів, контроль за виданими книгами та автоматизацію процесу повернення літератури.

─ Інформаційний модуль: Для оперативного інформування користувачів про нові надходження, події та інші важливі новини бібліотеки необхідно впровадити зручний спосіб публікації інформації.

Таким чином, автоматизація основних процесів бібліотеки дозволить покращити якість обслуговування користувачів, підвищити ефективність управління ресурсами та забезпечити зручний доступ до інформації для всіх зацікавлених осіб.

* 1. Аналіз існуючих підходів до вирішення завдання

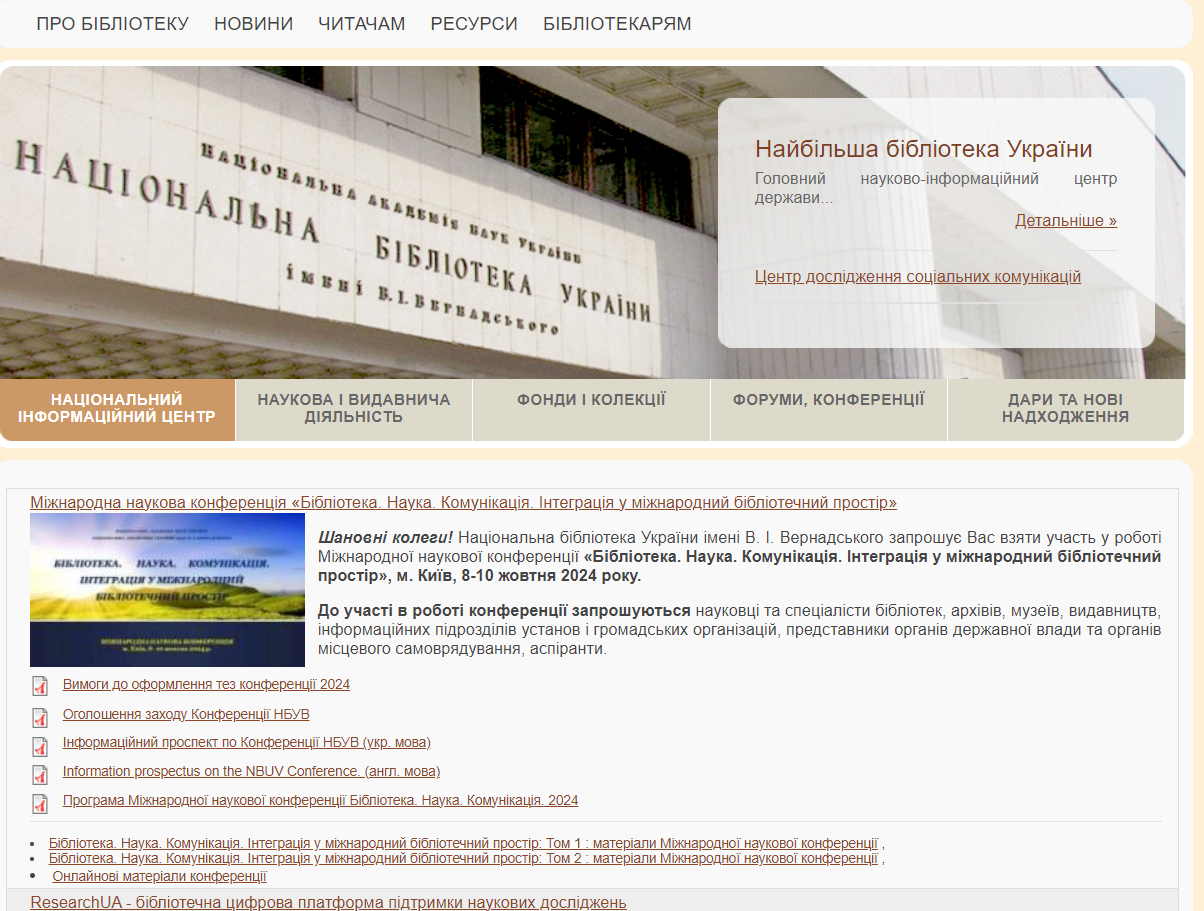


Рисунок 1.2.1 – Інтерфейс веб-сайту “Національна бібліотека України імені В. І. Вернадського”

Сайт Національної бібліотеки України імені В. І. Вернадського є яскравим прикладом впровадження сучасних технологій для забезпечення доступу до бібліотечних ресурсів. На цьому сайті реалізовано декілька важливих функцій, які відповідають потребам користувачів у сучасних умовах:

─ Електронний каталог: Сайт містить електронний каталог, який забезпечує зручний пошук книжок, періодичних видань та інших ресурсів за різними критеріями (автор, назва, ключові слова). Такий підхід дозволяє користувачам швидко знаходити необхідну літературу.

─ Інтеграція з науковими базами даних: Система забезпечує інтеграцію з іншими науковими базами даних, що дозволяє користувачам отримувати доступ до широкого кола наукових статей та публікацій. Це є важливим аспектом для дослідників та студентів, які займаються науковими роботами.

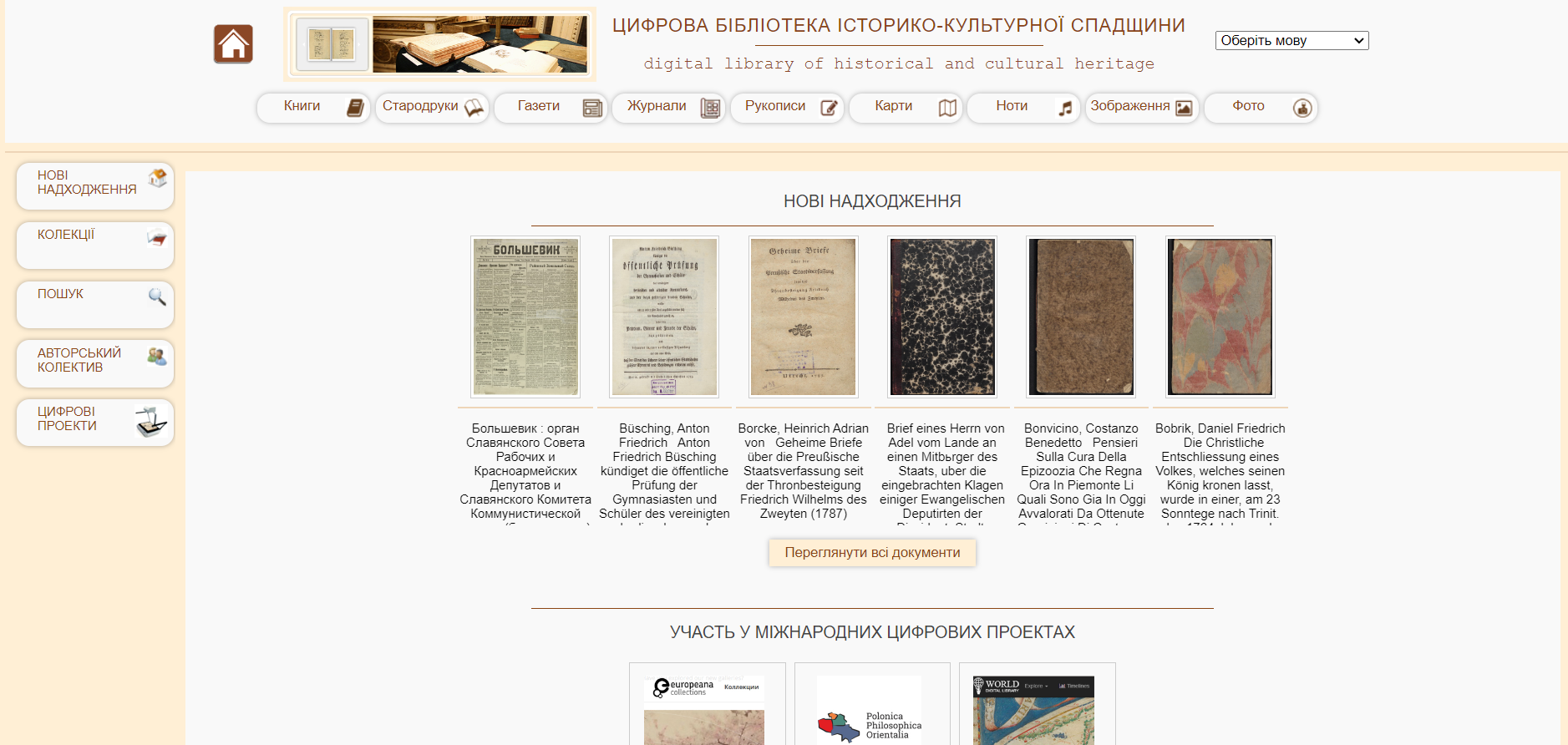


Рисунок 1.2.2 – Інтерфейс каталогу “Національна бібліотека України імені В. І. Вернадського”

─ Система резервування: На сайті реалізовано можливість резервування літератури, що дозволяє користувачам замовити книгу заздалегідь та отримати її у зручний час. Це сприяє підвищенню якості обслуговування та оптимізації роботи бібліотеки.

─ Інформаційний портал: Сайт також виконує функцію інформаційного порталу, на якому публікуються новини про нові надходження, заходи, наукові конференції та інші важливі події, що відбуваються у бібліотеці. Це забезпечує актуальність інформації та підвищує інтерес користувачів до діяльності бібліотеки.



Рисунок 1.2.3 – Інтерфейс порталу “Національна бібліотека України імені В. І. Вернадського”

─ Персональні кабінети: Система надає можливість користувачам створювати персональні кабінети, де вони можуть керувати своїми замовленнями, переглядати історію користування послугами та отримувати рекомендації на основі їхніх інтересів.

З огляду на функціонал сайту Національної бібліотеки України, можна зробити висновок, що використання електронних каталогів, систем резервування, інтеграції з науковими базами даних та персоналізованого обслуговування є ефективними підходами для вирішення задач управління бібліотекою. Проте застарілий та перегружений дизайн відштовхує користувача. Якщо вирішити проблему з дизайном, то ці підходи дозволяють підвищити зручність використання бібліотечних ресурсів, оптимізувати процеси обліку та забезпечити оперативне інформування користувачів про діяльність бібліотеки.

Таким чином, автоматизація основних процесів бібліотеки дозволить покращити якість обслуговування користувачів, підвищити ефективність управління ресурсами та забезпечити зручний доступ до інформації для всіх зацікавлених осіб.

Висновок до розділу 1

Після аналізу, стало зрозуміло, що повинно бути розроблено програмне забезпечення для бібліотеки. Адже в сучасному світі, більша частина інформації перебуває в інтернеті. Що вимагає створення діджиталізованого рішення для популяризації та більш широкого охоплення відвідувачами бібліотеки.

# РОЗДІЛ 2 ПРОЕКТУВАННЯ І ОСОБЛИВОСТІ РОЗРОБКИ ВЕБ-ДОДАТКУ

2.1 Вимоги до програми

Інформаційний портал повинен містити:

* Головну сторінку
  + Шапку сайту(з логотипом)
  + Меню
  + Слайдер
  + Вивід 10 головних новин порталу
  + Підвал сайту(Футер)

Меню складається:

* Про сайт
* Галерея картинок
* Новини
* Контакти(з виведенням карти *Google* на сайт)

Дизайн повинен бути адаптивним (мобільна та десктопна версії)

2.2 Системні вимоги програми

Для належної роботи програмного додатку на стороні клієнту, системі користувачанеобхідно мати доступ до інтернету, а також наявність веб-браузеру. Для старту програми на стороні серверу необхідно мати доступ до глобальної мережі інтернет, встановлений *Git* та *Docker* для розгортанню проекту в який включене розгортання бази даних в (далі - БД) *MySQL* з відповідними налаштуваннями в коді проекту для доступу до неї.

Зображення, що містить схема, ескіз, текст, План

Автоматично згенерований опис2.3 Діаграми та схеми

Рисунок 2.3.1 – Блок-схема роботи програми

На цій блок-схемі (рисунок 2.3.1) відображено сторінки на які користувач буде переходити при використанні сайту. При цьому перед цим він повинен авторизуватися на сайті.

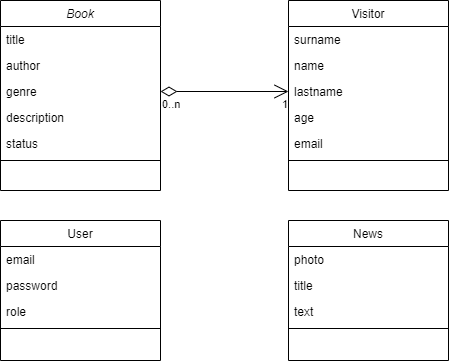


Рисунок 2.3.2 – *Unified Modeling Language* (далі - *UML*) діаграма моделей БД застосунку

Діаграма класів (рисунок 2.3.1) показує необхідність реалізації:

─ файли сутностей БД (*News*.*js , User*.*js*, *Book*.*js, Visitor*.*js*)

─ файли, що відповідають за ініціалізацію схеми БД та її постійне оновлення;

─ файли-роутери для визначення кінцевих точок *API* на які будуть надсилати запити;

─ файли-контроллери для маніпуляцій з даними БД: *newsController* - відповідає за маніпуляції з новинами; *bookController* - відповідає за маніпуляції з інформацією про книги бібліотеки; *visitorController* - відповідає за роботу інформацією про відвідувачів бібліотеки; *userController* – відповідає за роботу з користувачами сайту; *photoController* - відповідає за роботу з фотографіями, що відображаються на сайті в тому числі за роботу галереї; *authController* – реєструє або авторизовує користувачів сайту);

─ файл-ініціалізатор основних залежностей, функціоналу та налаштувань сайту (*app*.*js*);

─ файл для запуску сервера (*server*.*js*).

2.4 Особливості програмної реалізації

Розглянемо декілька варіантів архітекури веб-застосунку:

* Моде́ль–вигляд–контро́лер (*MVC*) — архітектурний шаблон, який використовується під час проєктування та розробки програмного забезпечення.

Мета шаблону — гнучкий дизайн програмного забезпечення, який повинен полегшувати подальші зміни чи розширення програм, а також надавати можливість повторного використання окремих компонентів програми. Крім того використання цього шаблону у великих системах сприяє впорядкованості їхньої структури і робить їх більш зрозумілими за рахунок зменшення складності. —

Цей шаблон передбачає поділ системи на три взаємопов'язані частини: модель даних, вигляд (інтерфейс користувача) та модуль керування. Застосовується для відокремлення даних (моделі) від інтерфейсу користувача (вигляду) так, щоб зміни інтерфейсу користувача мінімально впливали на роботу з даними, а зміни в моделі даних могли здійснюватися без змін інтерфейсу користувача.

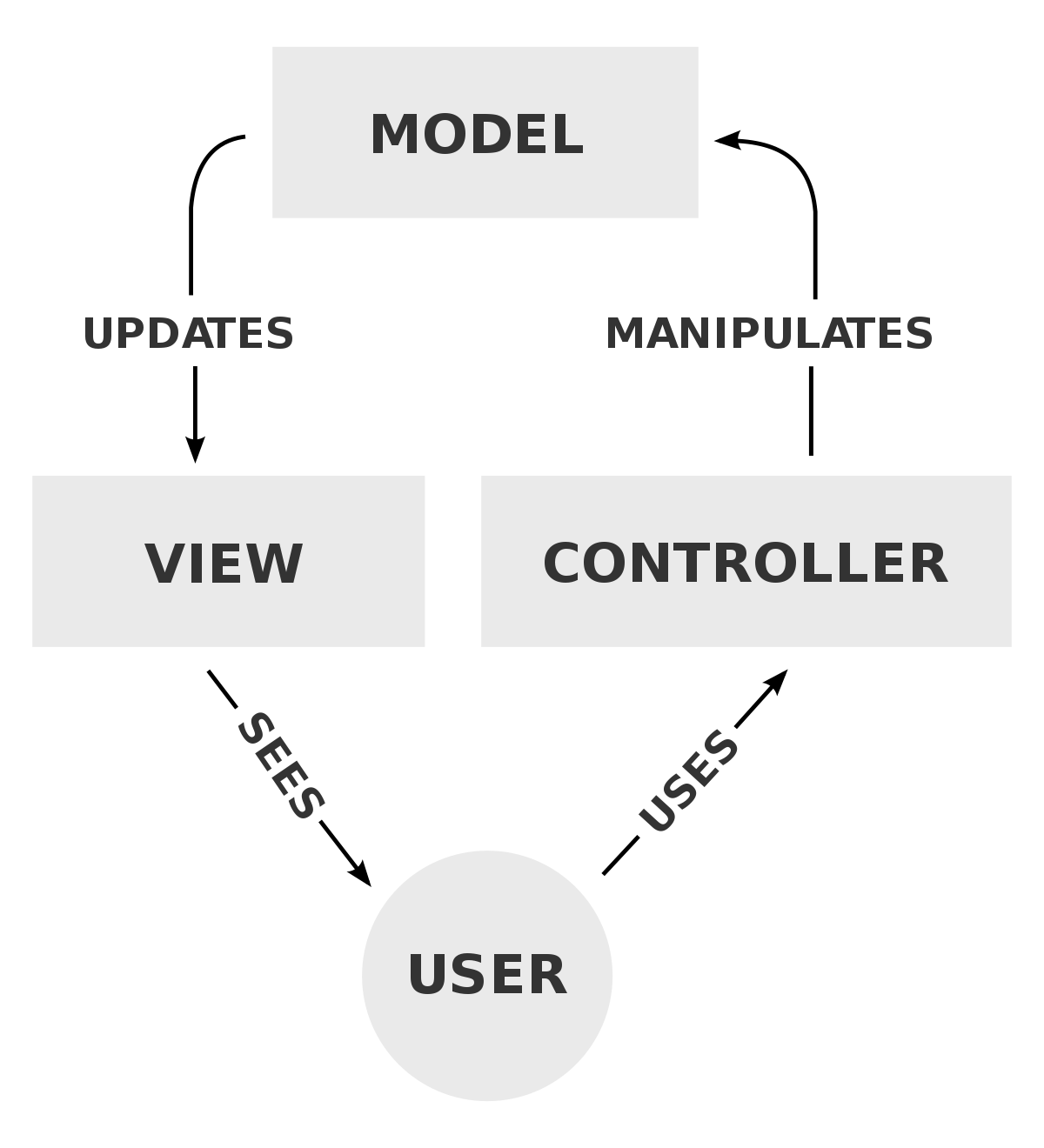


Рисунок 2.4.1 – шаблон *Model View Controll*

*RESTful API* — це інтерфейс, що використовується двома комп’ютерними системами для безпечного обміну інформацією через Інтернет. Більшість бізнес-прикладень повинні взаємодіяти з іншими внутрішніми та сторонніми додатками для виконання різних завдань. Наприклад, щоб створити щомісячні платіжні дані, ваша внутрішня бухгалтерська система повинна обмінюватися з банківською системою вашого клієнта, щоб автоматизувати виставлення рахунків і взаємодіяти з внутрішнім додатком за підсумком робочого часу.

*RESTful API* підтримує такий обмін інформацією, оскільки вони є безпечними, надійними та ефективними стандартами програмної взаємодії.

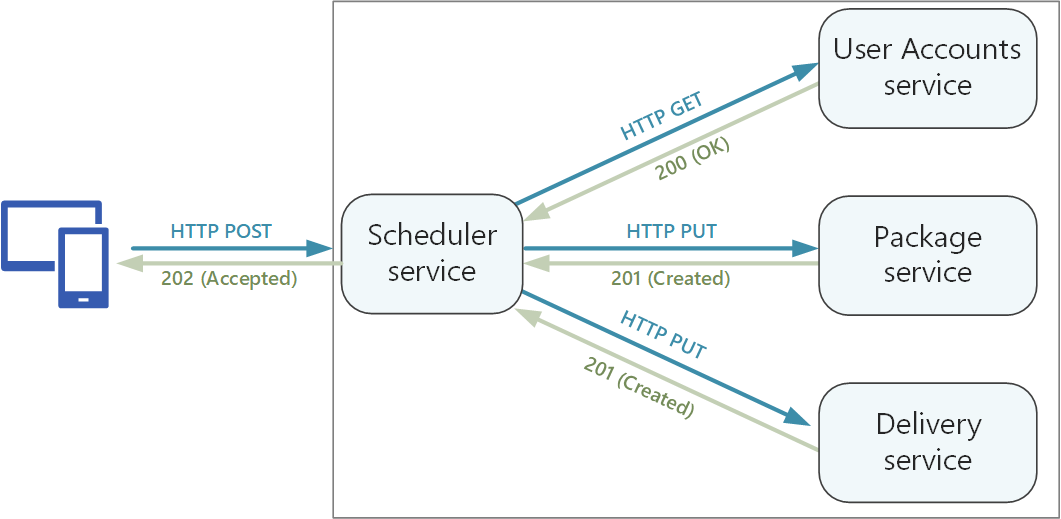


Рисунок 2.4.2 – шаблон *Rest* *API*

Для обробки та опрацювання даних, отриманих завдяки *RESTfull services* — використовується надані фреймворком *REST* контролери які перетворюють *json* строку у *data transfer object*, в заздалегідь створеному класі.

Ще один варіант такого застосунку, це клієнт-серверна архітектура.

В основі клієнт-серверної архітектури лежать два компоненти: клієнт і сервер.

Клієнт – комп'ютер на стороні користувача, який відправляє запит до сервера для надання інформації або виконання певних дій.

Сервер – більш потужний комп'ютер або обладнання, призначене для вирішення певних завдань з виконання програмних кодів, виконання сервісних функцій за запитом клієнтів, надання користувачам доступу до певних ресурсів, зберігання інформації і баз даних.

Модель такої системи полягає в тому, що клієнт відправляє запит на сервер, де він обробляється, і готовий результат відправляється клієнтові. Сервер може обслуговувати кілька клієнтів одночасно. Якщо одночасно приходить більше одного запиту, то вони встановлюються в чергу і виконуються сервером послідовно. Іноді запити можуть мати пріоритети. Запити з більш високими пріоритетами повинні виконуватися раніше.



Рисунок 2.4.3 – схема клієнт-серверного застосунку

Функції, які реалізуються на сервері:

* зберігання, доступ, захист і резервне копіювання даних;
* обробка клієнтського запиту;
* відправлення результату (відповіді) клієнту.

Функції, які реалізуються на стороні клієнта:

* надання користувальницького інтерфейсу;
* формулювання запиту до сервера і його відправка;
* отримання результатів запиту і відправка додаткових команд (запитів на додавання, оновлення або видалення даних).

Клієнт-серверний підхід наразі є найбільш сучасним та прогресивним, а також найбільш оптимальний по ресурсам. Тому для застосунку буде обрано клієнт-серверний шаблон архітектури. Відповідно до вимог курсової роботи, сервер буде реалізований на *Node.js*, а клієнт – на *React.js*.

*Node.js* — це однопоточне кросплатформове середовище виконання з відкритим вихідним кодом і бібліотека, яка використовується для запуску веб-додатків, написаних на *JavaScript*, поза браузером клієнта. *Node.js* — це програмне середовище, яке дозволяє запускати програми, написані мовою *JavaScript*, поза браузером. Історично програми, написані на *JavaScript*, на відміну від інших мов програмування, можна було запустити лише у браузерах, які мали спеціальний вбудований движок виконання коду цієї мови. Поза браузером *JavaScript*, можна сказати, не працював. При розробці *Node.js* за основу було взято двигун виконання *JavaScript* під назвою *V8*, який був створений компанією *Google* і використовувався в браузері *Google* *Chrome*. Оскільки після створення *Node.js* *JavaScript* код можна запустити фактично в будь-якому середовищі, за допомогою цієї бібліотеки можна написати не лише фронтенд, а й серверну частину веб-програми. Простіше кажучи, це означає, що цілі сайти тепер можуть працювати з використанням єдиного «стека», що робить розробку та обслуговування набагато швидшими та легшими, дозволяючи зосередитися на досягненні бізнес-цілей проєкту. *Node.js* має відкритий код, тому працювати з ним можна абсолютно безкоштовно. Його і сьогодні продовжує розвивати та покращувати глобальна спільнота розробників. Важливо розуміти, що *Node.js* насправді не фреймворк і бібліотека, як у випадку з традиційним програмним забезпеченням, а середовище виконання. Він є легким, гнучким і простим у розгортанні, а всі його функції допоможуть оптимізувати та прискорити вашу програму.

*Node.js* написано мовою *JavaScript*. Сьогодні це найпопулярніша мова програмування у світі. Більшість програмістів вже добре знайомі з *JavaScript*, її роботою та іншими базовими й просунутими концепціями. Це робить *Node.js* простим для розуміння та вивчення. Крім того, *JavaScript* також використовується в стеку технологій для розробки клієнтського інтерфейсу, а додаючи використання *Node.js* розробники можуть створювати повноцінні робочі веб-проєкти, знаючи тільки *JavaScript*.

Важливою особливістю *Node.js* є асинхронний характер. Термін асинхронний означає, що сервер, створений з використанням *Node.js*, не повинен чекати, поки дані повернуться, при виконанні різних внутрішніх запитів. Водночас він також має неблокуючий введення-виведення. Це означає, що кілька різних процесів можуть виконуватись паралельно, не блокуючи один одного. Обидві ці властивості роблять *Node.js* вкрай швидкою і забезпечують кращий інтерфейс користувача.

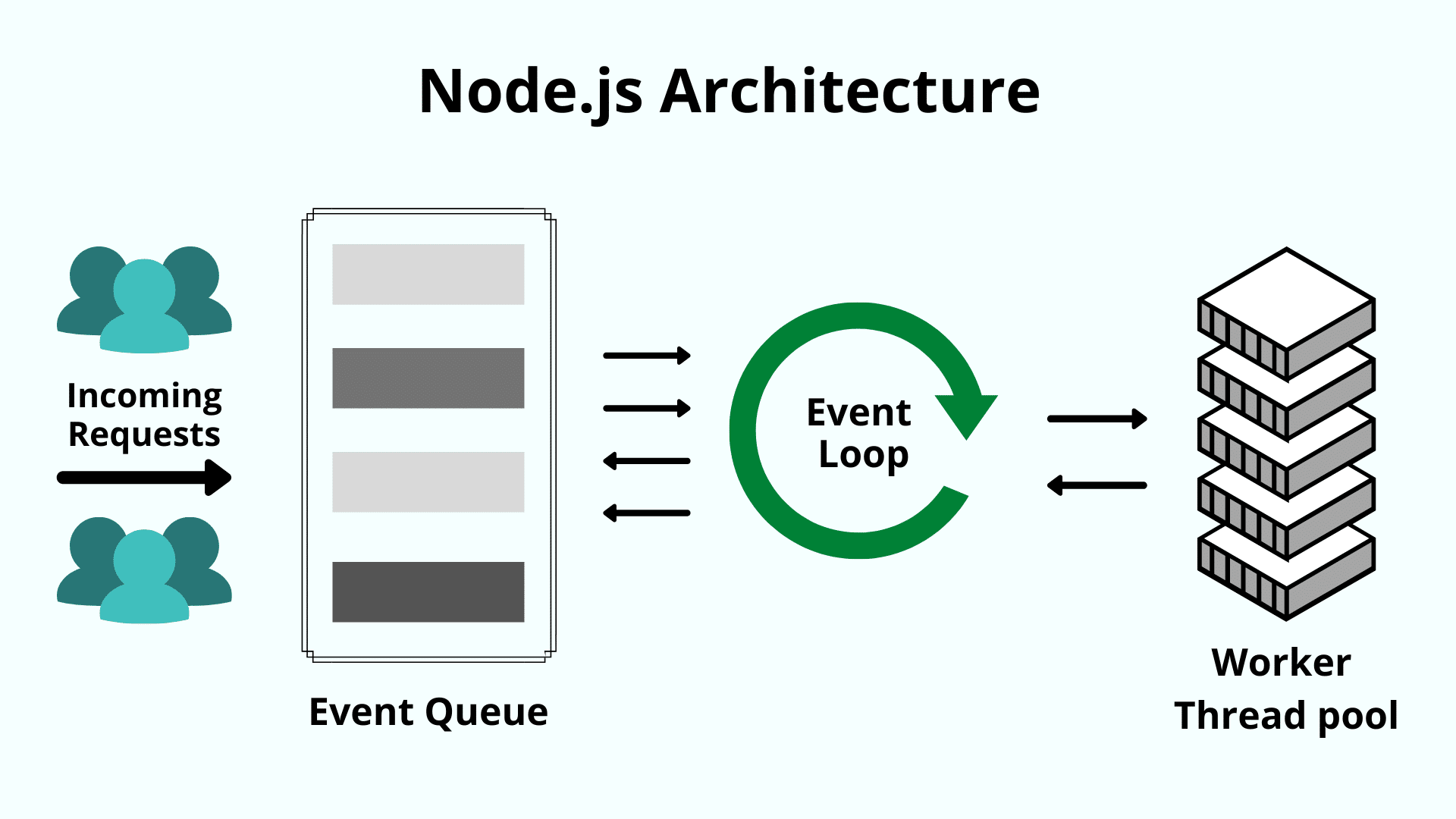


Рисунок 2.4.4 – Архітектура застосунку на *Node.js*

Для клієнтської сторони застосунку буде використано *React.js*.

*React.js* — це відкритий *JavaScript*-фреймворк, а точніше, бібліотекою *JavaScript*, яка використовується для розробки інтерфейсів користувача. Він був створений компанією *Facebook* і швидко набув популярності серед розробників з усього світу. *React.js* дозволяє ефективно створювати застосунки з високою продуктивністю і масштабованістю. Одним з ключових концепцій у *React.js* є компоненти. Вони представляють собою незалежні блоки коду, які відповідають за рендеринг певної частини користувацького інтерфейсу.

*React.js* використовується для розробки користувацького інтерфейсу веб-додатків. Основною метою *React.js* є створення інтерактивних, динамічних та зручних інтерфейсів для користувачів. *React.js* дозволяє створювати багатофункціональні та інтерактивні застосунки зі швидким рендерингом і переходом між сторінками. Також він має такі переваги:

* використання віртуального *DOM* та ефективного алгоритму оновлення дозволяє робити мінімальні зміни в реальному *DOM*, що покращує продуктивність застосунків;
* *React.js* базується на компонентній архітектурі, що дозволяє розбивати інтерфейс на незалежні компоненти. Це спрощує розробку, тестування та підтримку коду, оскільки компоненти можуть бути повторно використані та легко змінюються;
* *React.js* пропагує односторонній потік даних, що сприяє простоті та передбачуваності управління станом додатків, полегшує відлагодження та тестування застосунків;
* спільнота розробників, що використовує *React.js* велика та активна. Є безліч ресурсів, бібліотек та інструментів для розробки. Також існує багато сторонніх бібліотек, які підтримують *React.js* і допомагають розширити його функціональність;

Оскільки веб-сайт є більш універсальним і доступним варіантом для користувачів, він є більш підходящим варіантом для нашого проекту. Веб-сайт може бути доступний з будь-якого пристрою з підключенням до Інтернету і не вимагає встановлення додатків або його залежностей на користувачеві комп'ютер. Веб-сайт також дозволяє забезпечувати безпеку даних користувачів, використовуючи захисні механізми на стороні сервера та на стороні клієнта свої програми через утиліти командного рядка.

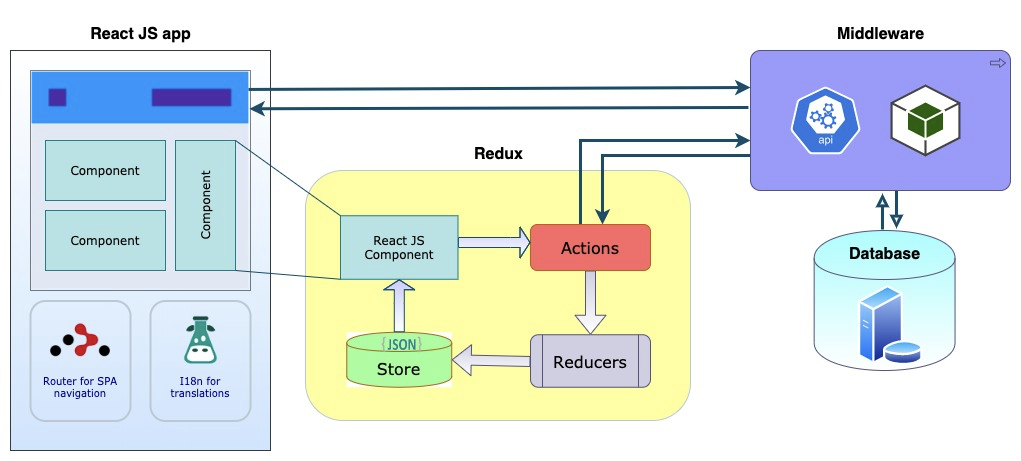


Рисунок 2.4.5 – Архітектура застосунку з використанням *React.js*

Також для збереження даних потрібно використати бази даних. Один із варіантів це реляційна БД – *MySQL*.

База даних являє собою структуровану сукупність даних. Ці дані можуть бути будь-якими - від простого списку майбутніх покупок до переліку експонатів картинної галереї або величезної кількості інформації в корпоративній мережі. Для запису, вибірки і обробки даних, що зберігаються в комп'ютерній базі даних, необхідна система управління базою даних, якою є, наприклад, ПЗ *MySQL*.

Оскільки комп'ютери чудово справляються з обробкою великих обсягів даних, управління базами даних відіграє центральну роль в обчисленнях. Реалізовано таке управління може бути по-різному - як у вигляді окремих утиліт, так і у вигляді коду, що входить до складу інших додатків.

*MySQL* - це система керування базами даних. У реляційній базі даних дані зберігаються не всі скопом, а в окремих таблицях, завдяки чому досягається виграш в швидкості і гнучкості. Таблиці зв'язуються між собою за допомогою відносин, завдяки чому забезпечується можливість об'єднувати при виконанні запиту дані з декількох таблиць. *SQL* як частина системи *MySQL* можна охарактеризувати як мову структурованих запитів плюс найбільш поширений стандартний мова, яка використовується для доступу до баз даних.

*MySQL* - це ПЗ з відкритим кодом. Застосовувати його і модифікувати може будь-хто. Таке ПЗ можна одержувати по *Internet* і використовувати безкоштовно. При цьому кожен користувач може вивчити вихідний код і змінити його відповідно до своїх потреб.

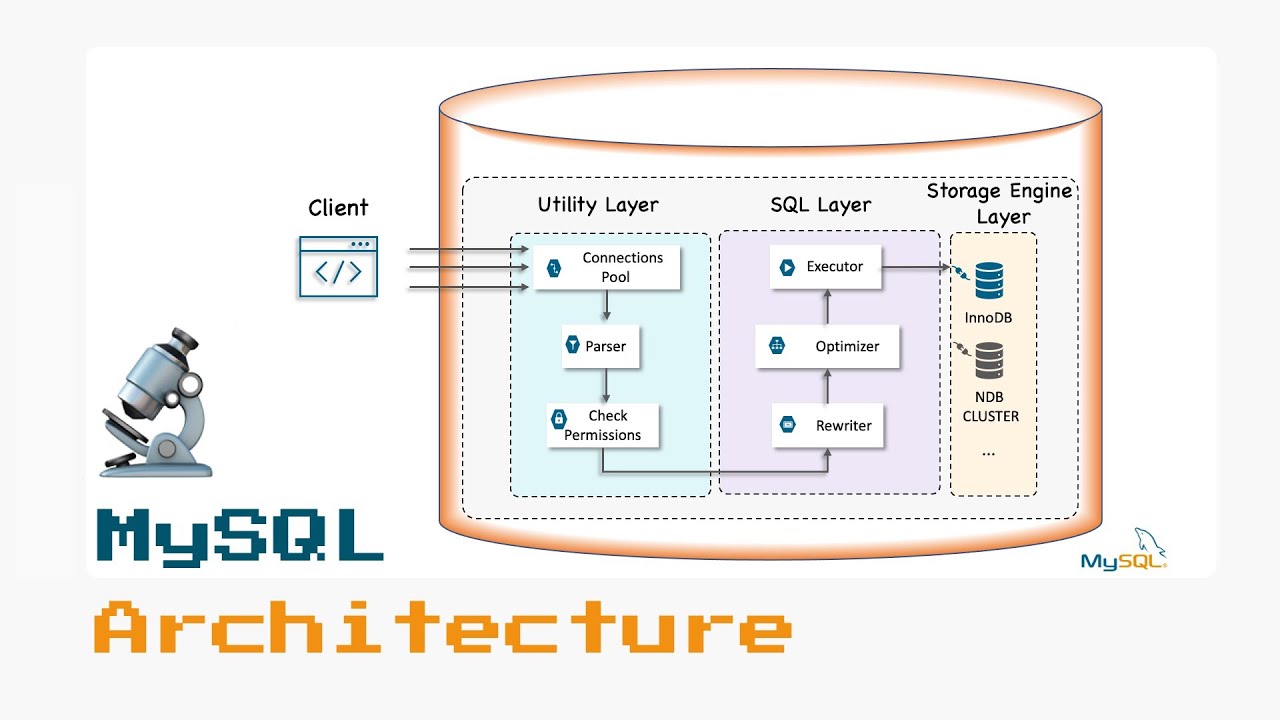


Рисунок 2.4.6 – Архітектура *MySQL*

Чому веб-програмісти віддають перевагу СУБД *MySQL*? Вона є дуже швидкою, надійною і легкою у використанні. Якщо вам потрібні саме ці якості, спробуйте попрацювати з цією БД. *MySQL* володіє також рядом зручних можливостей, розроблених в тісному контакті з користувачами. Спочатку сервер *MySQL* розроблявся для управління великими базами даних з метою забезпечити більш високу швидкість роботи в порівнянні з існуючими на той момент аналогами. І ось уже протягом кількох років даний сервер успішно використовується в умовах промислової експлуатації з високими вимогами. Незважаючи на те що *MySQL* постійно вдосконалюється, він уже сьогодні забезпечує широкий спектр корисних функцій. Завдяки своїй доступності, швидкості і безпеки *MySQL* дуже добре підходить для доступу до баз даних по *Internet*.

Альтернативою виступає реляційна БД – *PostgreSQL*.

*PostgreSQL* - Це вільна і відкрита система керування базами даних (СУБД) з відкритим вихідним кодом. Є однією з найпопулярніших СУБД, тому її використовують тисячі організацій по всьому світу. Ця система ґрунтується на некомерційній СУБД *Postgres*, яка почалася як відкритий проект у Каліфорнійському університеті в Берклі. Ініційована 1986 року Майклом Стоунбрейкером, лідером проєкту *Ingres*, вона була названа “*Post Ingres*”. Згодом вона вийшла за межі університету і стала відомою як *PostgreSQL*, розвиваючись і вдосконалюючись до теперішнього часу.

*PostgreSQL* належить до наступного типу СУБД – об’єктно-реляційного (тобто поєднує в собі переваги реляційних баз даних і об’єктно-орієнтованого програмування).

*PostgreSQL* підтримує широкий спектр функцій, які роблять її потужною і гнучкою СУБД. Ось деякі з її ключових особливостей:

* *PostgreSQL* забезпечує транзакції з *ACID*-принципами, що гарантує цілісність даних.
* Підтримує автоматично оновлювані подання, що забезпечує можливість користувачам переглядати дані в базі даних у режимі реального часу.
* Підтримує матеріалізовані подання, які зберігають копію даних подання в базі даних.
* Підтримує тригери, які дають змогу виконувати дії під час зміни даних у таблиці.
* Підтримує зовнішні ключі для створення зв’язку між таблицями.
* Підтримує збережені процедури для виконання складних обчислень на стороні сервера.

Проте для наших потреб вистачить і звичайної *MySQL*.

Висновок до розділу 2

В даному розділі було проаналізовано:

* Вимоги до програми;
* Виконання програм по основним методам.
* Засоби для реалізації програмного продукту

Реалізовано:

* *UML* діаграму моделей БД застосунку;
* Блок-схема роботи програми*.*

Для серверної частини було обрано фреймворк *Node.js*, клієнту – *React.js* та СКБД – *MySQL*.

РОЗДІЛ 3 ОСОБЛИВОСТІ ВИКОРИСТАННЯ ВЕБ-ДОДАТКУ

3.1 Порядок використання програмного додатку

Головна сторінка веб-сайту містить елемент-карусель з картинками та короткою інформацією про бібліотеку, футер з картою сайту, а також блок навігації з кнопкою входу (рисунок 3.1)

Зображення, що містить текст, знімок екрана, полиця, книга

Автоматично згенерований опис

Рисунок 3.1 – Головна сторінка сайту

Всі сторінки проекту контролюються головним файлом з роутами, що зв’язують сторінки з шляхами до них (*App.jsx*).

import "./App.css";

import { BrowserRouter as Router, Route, Routes } from "react-router-dom";

import MainPage from "./components/pages/MainPage/MainPage";

import "bootstrap/dist/css/bootstrap.css";

import NewsPage from "./components/pages/NewsPage/NewsPage";

import LoginPage from "./components/pages/LoginPage/LoginPage";

import ContactsPage from "./components/pages/ContactsPage/ContactsPage";

import BookPage from "./components/pages/BookPage/BookPage";

import GalleryPage from "./components/pages/GalleryPage/GalleryPage";

function App() {

return (

<Router>

<Routes>

<Route path="/" element={<MainPage />} />

<Route path="/login" element={<LoginPage />} />

<Route path="/news" element={<NewsPage />} />

<Route path="/contacts" element={<ContactsPage />} />

<Route path="/books" element={<BookPage/>} />

<Route path="/gallery" element={<GalleryPage/>} />

</Routes>

</Router>

);

}

export default App;

Лістинг програмного коду 3.1 – *App.jsx*

Головна сторінка складається з таких основних компонентів:

* <NavBar /> - Навігаційна панель
* <ControlledCarousel /> - Керована карусель
* <Footer /> - Футер
* Текст між футером та каруселлю

Зображення, що містить текст, знімок екрана, дизайн

Автоматично згенерований описПісля натискання користувачем на кнопку «Увійти» відкривається сторінка авторизації. Де користувач може авторизуватися, що буде підтверджено виданим його клієнту *jwt*-токеном. (рисунок 3.2)

Рисунок 3.2 – Сторінка авторизації

Зображення, що містить текст, знімок екрана, програмне забезпечення, Веб-сторінка

Автоматично згенерований описПісля цього користувач може перейти на сторінку по роботі з бібліотечним фондом. На цій сторінці він може переглядати список книг, що належать бібліотеці:

Рисунок 3.3 – Список книг на сторінці по роботі з бібліотечним фондом

Зображення, що містить текст, знімок екрана, Шрифт, число

Автоматично згенерований описНа цій сторінці користувач може не тільки переглядати детальну інформацію про кожну книгу та її тримача, а й редагувати її:

Рисунок 3.4 – Модальне вікно редагування інформації про книгу на сторінці по роботі з бібліотечним фондом

Також користувач має можливість зареєструвати книгу, що також реалізовано за допомогою форми в модальному вікні:

Зображення, що містить текст, знімок екрана, Шрифт, ряд

Автоматично згенерований опис

Рисунок 3.5 – Модальне вікно для реєстрації книги на сторінці по роботі з бібліотечним фондом

Користувач також має можливість видалити будь-яку книгу натиснувши кнопку «Видалити» на сторінці зі списком книг.  
  
 За основний функціонал цієї сторінки відповідає файл *BookList*.*jsx*:

import React, { useState, useEffect } from 'react';  
import { fetchBooks, addBook, updateBook, deleteBook, fetchVisitors } from '../../services/BookService';  
import './BookList.css';  
  
export default function BookList() {  
 const [books, setBooks] = useState([]);  
 const [loading, setLoading] = useState(true);  
 const [searchQuery, setSearchQuery] = useState('');  
 const [filteredBooks, setFilteredBooks] = useState([]);  
 const [isModalOpen, setIsModalOpen] = useState(false);  
 const [currentBook, setCurrentBook] = useState(null);  
  
 useEffect(() => {  
 async function loadBooks() {  
 try {  
 const data = await fetchBooks();  
 setBooks(data);  
 setFilteredBooks(data);  
 setLoading(false);  
 } catch (error) {  
 *console*.error('Error loading books:', error);  
 setLoading(false);  
 }  
 }  
 loadBooks();  
 }, []);  
  
 useEffect(() => {  
 const filtered = books.filter((book) =>  
 book.title && book.title.toLowerCase().includes(searchQuery.toLowerCase())  
 );  
 setFilteredBooks(filtered);  
 }, [searchQuery, books]);  
  
 const handleAddBook = () => {  
 setCurrentBook(null);  
 setIsModalOpen(true);  
 };  
  
 const handleEditBook = (book) => {  
 setCurrentBook(book);  
 setIsModalOpen(true);  
 };  
  
 const handleCloseModal = () => {  
 setIsModalOpen(false);  
 };  
  
 const handleSaveBook = async (book) => {  
 try {  
 const savedBook = currentBook  
 ? await updateBook(book)  
 : await addBook(book);  
  
 setBooks((prevBooks) =>  
 currentBook  
 ? prevBooks.map((item) =>  
 item.id === savedBook.id ? savedBook : item  
 )  
 : [...prevBooks, savedBook]  
 );  
 } catch (error) {  
 *console*.error('Error saving book:', error);  
 } finally {  
 setIsModalOpen(false);  
 }  
 };  
  
 const handleDeleteBook = async (bookId) => {  
 try {  
 await deleteBook(bookId);  
 setBooks((prevBooks) =>  
 prevBooks.filter((item) => item.id !== bookId)  
 );  
 } catch (error) {  
 *console*.error('Error deleting book:', error);  
 }  
 };  
  
 if (loading) {  
 return <div className="loading">Loading...</div>;  
 }  
  
 return (  
 <div className="book-list-container">  
 <input  
 type="text"  
 placeholder="Пошук по назві книги..."  
 value={searchQuery}  
 onChange={(e) => setSearchQuery(e.target.value)}  
 className="search-input"  
 />  
 <button onClick={handleAddBook} className="add-button">  
 Зареєструвати книгу  
 </button>  
 <div className="book-list">  
 {filteredBooks.map((book) => (  
 <div key={book.id} className="book-card">  
 <h3>{book.title}</h3>  
 <p>{book.description}</p>  
 <div className="book-details">  
 <span><strong>Автор:</strong> {book.author}</span>  
 <span><br/><strong>Жанр:</strong> {book.genre}</span>  
 <span><br/><strong>Статус:</strong> {book.status}</span>  
 <span><br/><strong>Тримач книги:</strong> {book.borrower ? book.borrower.name + " " + book.borrower.surname : 'Бібліотека'}</span>  
 </div>  
 <div className="book-actions">  
 <button onClick={() => handleEditBook(book)} className="edit-button">  
 Редагувати  
 </button>  
 <button onClick={() => handleDeleteBook(book.id)} className="delete-button">  
 Видалити  
 </button>  
 </div>  
 </div>  
 ))}  
 </div>  
 {isModalOpen && (  
 <BookModal  
 book={currentBook}  
 onClose={handleCloseModal}  
 onSave={handleSaveBook}  
 />  
 )}  
 </div>  
 );  
}  
  
function BookModal({ book, onClose, onSave }) {  
 const [title, setTitle] = useState(book ? book.title : '');  
 const [author, setAuthor] = useState(book ? book.author : '');  
 const [genre, setGenre] = useState(book ? book.genre : 'Наукова фантастика');  
 const [description, setDescription] = useState(book ? book.description : '');  
 const [status, setStatus] = useState(book ? book.status : 'В наявності');  
 const [borrowerId, setBorrowerId] = useState(book && book.borrower ? book.borrower.id : 0);  
 const [visitors, setVisitors] = useState([]);  
  
 useEffect(() => {  
 setTitle(book ? book.title : '');  
 setAuthor(book ? book.author : '');  
 setGenre(book ? book.genre : 'Наукова фантастика');  
 setDescription(book ? book.description : '');  
 setStatus(book ? book.status : 'В наявності');  
 setBorrowerId(book && book.borrower ? book.borrower.id : 0);  
 }, [book]);  
  
 useEffect(() => {  
 if (status === 'Видана на руки') {  
 getVisitors();  
 }  
 }, [status]);  
  
 const getVisitors = async () => {  
 try {  
 const data = await fetchVisitors();  
 setVisitors(data);  
 *console*.log('Visitors fetched:', data);  
 } catch (error) {  
 *console*.error('Error fetching visitors:', error);  
 }  
 };  
  
 const handleSave = () => {  
 const newBook = {  
 id: book ? book.id : 0,  
 title,  
 author,  
 genre,  
 description,  
 status,  
 borrowerId: status === 'Видана на руки' && borrowerId !== 0 ? borrowerId : null,  
 createdAt: book ? book.createdAt : new *Date*().toISOString(),  
 updatedAt: new *Date*().toISOString(),  
 };  
  
 onSave(newBook);  
 };  
  
 return (  
 <div className="modal-overlay">  
 <div className="modal-content">  
 <h2>{book ? 'Редагувати інформацію про книгу' : 'Зареєструвати книгу'}</h2>  
 <input  
 type="text"  
 placeholder="Назва книги"  
 value={title}  
 onChange={(e) => setTitle(e.target.value)}  
 />  
 <input  
 type="text"  
 placeholder="Автор"  
 value={author}  
 onChange={(e) => setAuthor(e.target.value)}  
 />  
 <select value={genre} onChange={(e) => setGenre(e.target.value)}>  
 <option value="Наукова фантастика">Наукова фантастика</option>  
 <option value="Фантастика">Фантастика</option>  
 <option value="Мелодрама">Мелодрама</option>  
 <option value="Містика">Містика</option>  
 <option value="Художня література">Художня література</option>  
 </select>  
 <textarea  
 placeholder="Опис"  
 value={description}  
 onChange={(e) => setDescription(e.target.value)}  
 />  
 <select value={status} onChange={(e) => setStatus(e.target.value)}>  
 <option value="В наявності">В наявності</option>  
 <option value="Видана на руки">Видана на руки</option>  
 </select>  
  
 {status === 'Видана на руки' && (  
 <select value={borrowerId} onChange={(e) => setBorrowerId(parseInt(e.target.value))}>  
 <option value="0">Оберіть тримача книги</option>  
 {visitors.map((visitor) => (  
 <option key={visitor.id} value={visitor.id}>  
 {visitor.surname} {visitor.name} {visitor.lastName}  
 </option>  
 ))}  
 </select>  
 )}  
  
 <div className="modal-actions">  
 <button onClick={handleSave} className="save-button">Зберегти</button>  
 <button onClick={onClose} className="cancel-button">Відхилити</button>  
 </div>  
 </div>  
 </div>  
 );  
}

Лістинг програмного коду 3.2 – *BookList*.*jsx*

Ця сторінка використовує кінцеві точки сервера, що дозволяють виконувати повноцінний *CRUD* з моделлю книга:

*const express = require('express');  
const router = express.Router();  
const { getAllBooks, getBook, createBook, updateBook, deleteBook } = require('../controllers/bookController');  
const authMiddleware = require('../middlewares/authMiddleware');  
  
router.get('/', getAllBooks);  
router.get('/:id', getBook);  
router.post('/', authMiddleware, createBook);  
router.put('/:id', authMiddleware, updateBook);  
router.delete('/:id', authMiddleware, deleteBook);  
  
module.exports = router;*

Лістинг програмного коду 3.3 – Файл роутера, що відповідає за книги *booksRoutes.js*

*const Book = require('../models/Book');  
const Visitor = require('../models/Visitor');  
  
exports.getAllBooks = async (req, res) => {  
 try {  
 const books = await Book.findAll({ include: [{ model: Visitor, as: 'borrower' }] });  
 res.json(books);  
 } catch (err) {  
 console.error(err.message);  
 res.status(500).send('Server error');  
 }  
};  
  
exports.getBook = async (req, res) => {  
 const { id } = req.params;  
 try {  
 const book = await Book.findByPk(id, { include: [{ model: Visitor, as: 'borrower' }] });  
 if (!book) {  
 return res.status(404).json({ message: 'Book not found' });  
 }  
 res.json(book);  
 } catch (err) {  
 console.error(err.message);  
 res.status(500).send('Server error');  
 }  
};  
  
exports.createBook = async (req, res) => {  
 const { title, author, genre, description, status, borrowerId } = req.body;  
  
 try {  
 const book = await Book.create({ title, author, genre, description, status, borrowerId });  
 res.json(book);  
 } catch (err) {  
 console.error(err.message);  
 res.status(500).send('Server error');  
 }  
};  
  
exports.updateBook = async (req, res) => {  
 const { id } = req.params;  
 const { title, author, genre, description, status, borrowerId } = req.body;  
  
 try {  
 const book = await Book.findByPk(id);  
 if (!book) {  
 return res.status(404).json({ message: 'Book not found' });  
 }  
  
 book.title = title;  
 book.author = author;  
 book.genre = genre;  
 book.description = description;  
 book.status = status;  
 book.borrowerId = borrowerId;  
  
 await book.save();  
 res.json(book);  
 } catch (err) {  
 console.error(err.message);  
 res.status(500).send('Server error');  
 }  
};  
  
exports.deleteBook = async (req, res) => {  
 const { id } = req.params;  
  
 try {  
 const book = await Book.findByPk(id);  
 if (!book) {  
 return res.status(404).json({ message: 'Book not found' });  
 }  
  
 await book.destroy();  
 res.json({ message: 'Book deleted' });  
 } catch (err) {  
 console.error(err.message);  
 res.status(500).send('Server error');  
 }  
};*

Лістинг програмного коду 3.4 – Файл, що відповідає за логіку роботи з моделлю *Book* *booksController*

Зображення, що містить текст, знімок екрана

Автоматично згенерований опис Наступною сторінкою на яку може перейти користувач є сторінка галереї на якій можна переглянути фотографії бібліотеки (рисунок 3.6):

Рисунок 3.6 – Сторінка галереї

Фотографії з галереї беруться з галереї сервера бібліотеки в реальному часі, що дозволяє підтримувати їх актуальність. Це реалізовано у файлі *GalleryList.jsx*:  
import React, { useState, useEffect } from 'react';  
import { *Container*, *Row*, *Col*, *Card*, *Modal*, *Button*, *Alert* } from 'react-bootstrap';  
  
export default function GalleryList() {  
 const [images, setImages] = useState([]);  
 const [show, setShow] = useState(false);  
 const [selectedImage, setSelectedImage] = useState('');  
  
 useEffect(() => {  
 const getImages = async () => {  
 try {  
 const baseURL = `http://${*process*.env.REACT\_APP\_SERVER\_MAIN\_HOST}:5000/photos/gallery/img`;  
 const imageCount = 6;  
 const generatedImages = *Array*.from({ length: imageCount }, (\_, index) => {  
 if (index % 1 === 0) {  
 return {  
 url: `${baseURL}${index + 1}.png`  
 };  
 }  
 return null;  
 }).filter(Boolean);  
 setImages(generatedImages);  
 } catch (error) {  
 *console*.error('Error fetching images:', error);  
 }  
 };  
  
 getImages();  
 }, []);  
  
 const handleShow = (image) => {  
 setSelectedImage(image);  
 setShow(true);  
 };  
  
 const handleClose = () => {  
 setShow(false);  
 };  
  
 if (images.length === 0) {  
 return (  
 <Container className="mt-3">  
 <Alert variant="warning" className="text-center">  
 Помилка! Жодної фотографії не додано!  
 </Alert>  
 </Container>  
 );  
 }  
  
 return (  
 <Container className="mt-3 gallery-container">  
 <Row>  
 {images.map((image, index) => (  
 <Col key={index} md={4} className="mb-4">  
 <Card className="gallery-card">  
 <Card.Img   
 variant="top"   
 src={image.url}   
 alt="Image"  
 onClick={() => handleShow(image.url)}  
 className="gallery-image"  
 />  
 </Card>  
 </Col>  
 ))}  
 </Row>  
 <Modal show={show} onHide={handleClose} centered>  
 <Modal.Header closeButton>  
 <Modal.Title>Перегляд зображення</Modal.Title>  
 </Modal.Header>  
 <Modal.Body>  
 <img src={selectedImage} alt="Selected" style={{ width: '100%', borderRadius: '10px' }}/>  
 </Modal.Body>  
 <Modal.Footer>  
 <Button variant="secondary" onClick={handleClose}>  
 Закрити  
 </Button>  
 </Modal.Footer>  
 </Modal>  
 </Container>  
 );  
}

Лістинг програмного коду 3.4 – *GalleryList.jsx*

Іншою сторінкою на яку також може перейти користувач є сторінка з новинами бібліотеки (рисунок 3.7), на цій сторінці користувач може дізнатися, про основні події бібліотеки.

Зображення, що містить текст, знімок екрана, Шрифт, Бренд

Автоматично згенерований опис

Рисунок 3.7 – Сторінка з новинами

Ця сторінка використовує дані з сервера, що віддає потрібну інформацію про новини. Отримання інформації з серверу реалізовано на клієнті у файлі *NewsList.jsx*:  
import React, { useState, useEffect } from 'react';

import { Alert } from 'react-bootstrap';

import { fetchNews } from '../../services/NewsService'; // Убедитесь, что путь правильный

import './NewsList.css';

export default function NewsList() {

const [news, setNews] = useState([]);

useEffect(() => {

const getNews = async () => {

try {

const data = await fetchNews();

setNews(data);

} catch (error) {

console.error('Error fetching news:', error);

setNews([]); // Устанавливает пустой массив новостей в случае ошибки

}

};

getNews();

}, []);

if (news.length === 0) {

return (

<div className="news-container">

<Alert variant="warning" className="news-alert">

Помилка! Немає новин.

</Alert>

</div>

);

}

return (

<div className="news-container">

{news.map((item, index) => (

<div key={index} className="news-card">

<img src={`http://${process.env.REACT\_APP\_SERVER\_MAIN\_HOST}:5000/photos/news/${item.photo}`} alt="News" className="news-image"/>

<div className="news-content">

<h3 className="news-title">{item.title}</h3>

<p className="news-text">{item.text}</p>

</div>

</div>

))}

</div>

);

}

Лістинг програмного коду 3.7 – *NewsList.jsx*

На серверній стороні за обробку інформації про новини з БД відповідає *newsController.js*:  
const *News* = *require*('../models/News');  
const *User* = *require*("../models/User");  
  
*exports*.getAllNews = async (req, res) => {  
 try {  
 const news = await *News*.*findAll*();  
 res.json(news);  
 } catch (err) {  
 *console*.error(err.message);  
 res.status(500).send('Server error');  
 }  
};  
  
*exports*.getNews = async (req, res) => {  
 const {id} = req.params;  
 try {  
 const news = await *News*.*findByPk*(id);  
 res.json(news);  
 } catch (err) {  
 *console*.error(err.message);  
 res.status(500).send('Server error');  
 }  
};  
  
*exports*.createNews = async (req, res) => {  
 const {photo, title, text} = req.body;  
  
 try {  
 const news = await *News*.*create*({photo, title, text});  
 res.json(news);  
 } catch (err) {  
 *console*.error(err.message);  
 res.status(500).send('Server error');  
 }  
};  
  
*exports*.updateNews = async (req, res) => {  
 const {id} = req.params;  
 const {photo, title, text} = req.body;  
  
 try {  
 const news = await *News*.*findByPk*(id);  
 if (!news) {  
 return res.status(404).json({message: 'News not found'});  
 }  
  
 news.photo = photo;  
 news.title = title;  
 news.text = text;  
  
 await news.save();  
 res.json(news);  
 } catch (err) {  
 *console*.error(err.message);  
 res.status(500).send('Server error');  
 }  
};  
  
*exports*.deleteNews = async (req, res) => {  
 const {id} = req.params;  
  
 try {  
 const news = await *News*.*findByPk*(id);  
 if (!news) {  
 return res.status(404).json({message: 'News not found'});  
 }  
  
 await news.destroy();  
 res.json({message: 'News deleted'});  
 } catch (err) {  
 *console*.error(err.message);  
 res.status(500).send('Server error');  
 }  
};

Лістинг програмного коду 3.8 – *newsController.js*

Останньою сторінкою додатка є сторінка «Контакти» (рисунок 3.8) - це сторінка з загальною інформацією про бібліотеку, її розташування на мапі та контактну інформацію.

Зображення, що містить текст, знімок екрана, програмне забезпечення, Веб-сторінка

Автоматично згенерований опис

Рисунок 3.8 – Сторінка «Контакти»

Ця сторінка реалізована за допомогою компонента *Contacts.jsx*:

*import React from 'react';  
import './Contacts.css';  
  
export default function Contacts() {  
 return (  
 <div className="contacts-container">  
 <h1 className="contacts-title">Контакти бібліотеки</h1>  
 <section className="contacts-section">  
 <p className="contacts-description">  
 Національна бібліотека — це провідний науково-дослідний і навчальний центр, який забезпечує студентів і дослідників доступом до сучасної літератури. Бібліотека відіграє важливу роль у підготовці фахівців, забезпечуючи інформаційні ресурси та сприяючи їх професійному розвитку.  
 </p>  
 </section>  
 <section className="contacts-section">  
 <h2 className="contacts-subtitle">Контактна інформація</h2>  
 <p className="contacts-info"><strong>Адреса:</strong> вул. Просвітницька, 45, м. Київ, Україна</p>  
 <p className="contacts-info"><strong>Телефон:</strong> +380 (44) 123 4567</p>  
 <p className="contacts-info"><strong>Email:</strong> library@university.ua</p>  
 <p className="contacts-info"><strong>Робочий час:</strong> Пн-Пт 09:00 - 18:00</p>  
 </section>  
 <section className="contacts-mapSection">  
 <h2 className="contacts-subtitle">Наше розташування на карті</h2>  
 <div className="contacts-mapContainer">  
 <iframe  
 title="Google Map"  
 src="https://www.google.com/maps/embed?pb=!1m16!1m12!1m3!1d10164.828074069079!2d30.480668899111762!3d50.43724451793718!2m3!1f0!2f0!3f0!3m2!1i1024!2i768!4f13.1!2m1!1z0LLRg9C7LiDQn9GA0L7RgdCy0ZbRgtC90LjRhtGM0LrQsCwgNDUsINC8LiDQmtC40ZfQsiwg0KPQutGA0LDRl9C90LA!5e0!3m2!1sru!2sua!4v1728328478200!5m2!1sru!2sua"  
 className="contacts-map"  
 allowFullScreen=""  
 loading="lazy"  
 referrerPolicy="no-referrer-when-downgrade"  
 ></iframe>  
 </div>  
 </section>  
 </div>  
 );  
}*

Лістинг програмного коду 3.9 – Компонент *Contacts.jsx*

Висновок до розділу 3

У цьому розділі було розглянуто особливості застосування веб-сайту. Представлено послідовність його використання, яка дозволяє користувачеві ознайомитися з його функціоналом та можливостями, а також освоїти роботу з продуктом. Окрім того, надано програмний код ключових частин проекту із поясненням логіки роботи методів, класів і окремих компонентів.

ВИСНОВКИ

В результаті виконаної роботи було проведено ретельний аналіз та вибір технологій для реалізації веб-застосунку. Серед різних варіантів було обрано фреймворк *Node.js* для серверної частини, *React.js* для клієнтської частини, а також систему керування базами даних *MySQL*. Такий вибір зумовлений необхідністю створення кросплатформного рішення, яке б дозволяло максимально охопити різні типи пристроїв і забезпечувало зручність доступу до веб-застосунку для широкої аудиторії користувачів. Використання цих технологій гарантує високу продуктивність і стабільність, що є критично важливим для функціонування застосунку в режимі реального часу.

У процесі розробки було спроектовано логічну структуру програмного забезпечення, яка включає як серверну, так і клієнтську частини, що взаємодіють між собою за допомогою *API*. Крім того, створено діаграми варіантів використання, які описують можливі сценарії взаємодії користувачів із системою, а також діаграму класів, що наочно демонструє структуру даних та логіку роботи основних компонентів.

Окремо було розглянуто питання інтеграції з базою даних, що дозволило оптимізувати процеси зберігання, пошуку та обробки інформації в межах системи. Також реалізовано інтерфейс користувача, який є інтуїтивно зрозумілим і простим у використанні, дозволяючи швидко навчитися працювати з програмою.

Завдяки проведеній роботі створено повноцінний веб-застосунок, який забезпечує можливість вести облік книг у бібліотечному фонді в реальному часі. Система є масштабованою та готовою до подальшого розвитку, що відкриває перспективи для її подальшого вдосконалення та розширення функціоналу відповідно до потреб користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. David Flanagan - JavaScript: The Definitive Guide. 7th Ed.
2. Robin WieruchRobin Wieruch - The Road to React: Your journey to master plain yet pragmatic React.js
3. Sergey Kuzmichev, Vinicius M. GrippaV - Learning MySQL: Get a Handle on Your Data
4. Jon Duckett - HTML and CSS: Design and Build Websites
5. Jennifer Robbins - Learning Web Design: A Beginner's Guide to HTML, CSS, JavaScript, and Web Graphics
6. Elizabeth Castro, Bruce Hyslop - HTML5 and CSS3: Visual QuickStart Guide
7. Thomas A. Powell - HTML & CSS: The Complete Reference
8. Ben Frain - Responsive Web Design with HTML5 and CSS
9. Marijn Haverbeke - Eloquent JavaScript: A Modern Introduction to Programming
10. Kyle Simpson - You Don't Know JS: Scope & Closures
11. Nicolás Bevacqua - JavaScript Application Design: A Build First Approach
12. Robin Wieruch - The Road to React: Your journey to master plain yet pragmatic React.js
13. Alex Banks, Eve Porcello - Learning React: Functional Web Development with React and Redux
14. Stoyan Stefanov - React: Up & Running: Building Web Applications
15. Cory House - Building React Applications with Idiomatic Redux
16. Mark Tielens Thomas - React in Action
17. Nadia Makarevich, Artemij Fedosejev - React Quickly
18. Adam Boduch - React and React Native
19. Carl Rippon - Learning React: Modern Patterns for Developing React Apps
20. Kirupa Chinnathambi - Learning React: A Hands-On Guide to Building Web Applications Using React and Redux
21. Azat Mardan - React Quickly: Painless Web Apps with React, JSX, Redux, and GraphQL
22. Basarat Syed - Node.js Design Patterns
23. David Herron - Node.js Web Development
24. Sam Newman - Building Microservices: Designing Fine-Grained Systems
25. Ethan Brown - Learning Node.js Development
26. Azat Mardan - Pro Express.js: Mastering Express Framework for Node.js and Io.js
27. Brad Dayley, Brendan Dayley - Node.js, MongoDB, and AngularJS Web Development
28. Jonathan Wexler - Get Programming with Node.js
29. Mario Casciaro, Luciano Mammino - Node.js Design Patterns
30. Andrew Mead, Rob Percival - The Complete Node.js Developer Course
31. Shelley Powers - Learning Node: Moving to the Server-Side
32. Vaughn Vernon - Implementing Domain-Driven Design
33. Eric Evans - Domain-Driven Design: Tackling Complexity in the Heart of Software
34. Martin Fowler - Patterns of Enterprise Application Architecture
35. Sam Newman - Building Microservices: Designing Fine-Grained Systems

ДОДАТОК А

Лістинг коду

import "./App.css";  
import { BrowserRouter as Router, Route, Routes } from "react-router-dom";  
import MainPage from "./components/pages/MainPage/MainPage";  
import "bootstrap/dist/css/bootstrap.css";  
import NewsPage from "./components/pages/NewsPage/NewsPage";  
import LoginPage from "./components/pages/LoginPage/LoginPage";  
import ContactsPage from "./components/pages/ContactsPage/ContactsPage";  
import BookPage from "./components/pages/BookPage/BookPage";  
import GalleryPage from "./components/pages/GalleryPage/GalleryPage";  
  
function App() {  
 return (  
 <Router>  
 <Routes>  
 <Route path="/" element={<MainPage />} />  
 <Route path="/login" element={<LoginPage />} />  
 <Route path="/news" element={<NewsPage />} />  
 <Route path="/contacts" element={<ContactsPage />} />  
 <Route path="/books" element={<BookPage/>} />  
 <Route path="/gallery" element={<GalleryPage/>} />  
 </Routes>  
 </Router>  
 );  
}  
  
export default App;

import React, { useState, useEffect } from 'react';  
import { fetchBooks, addBook, updateBook, deleteBook, fetchVisitors } from '../../services/BookService';  
import './BookList.css';  
  
export default function BookList() {  
 const [books, setBooks] = useState([]);  
 const [loading, setLoading] = useState(true);  
 const [searchQuery, setSearchQuery] = useState('');  
 const [filteredBooks, setFilteredBooks] = useState([]);  
 const [isModalOpen, setIsModalOpen] = useState(false);  
 const [currentBook, setCurrentBook] = useState(null);  
  
 useEffect(() => {  
 async function loadBooks() {  
 try {  
 const data = await fetchBooks();  
 setBooks(data);  
 setFilteredBooks(data);  
 setLoading(false);  
 } catch (error) {  
 *console*.error('Error loading books:', error);  
 setLoading(false);  
 }  
 }  
 loadBooks();  
 }, []);  
  
 useEffect(() => {  
 const filtered = books.filter((book) =>  
 book.title && book.title.toLowerCase().includes(searchQuery.toLowerCase())  
 );  
 setFilteredBooks(filtered);  
 }, [searchQuery, books]);  
  
 const handleAddBook = () => {  
 setCurrentBook(null);  
 setIsModalOpen(true);  
 };  
  
 const handleEditBook = (book) => {  
 setCurrentBook(book);  
 setIsModalOpen(true);  
 };  
  
 const handleCloseModal = () => {  
 setIsModalOpen(false);  
 };  
  
 const handleSaveBook = async (book) => {  
 try {  
 const savedBook = currentBook  
 ? await updateBook(book)  
 : await addBook(book);  
  
 setBooks((prevBooks) =>  
 currentBook  
 ? prevBooks.map((item) =>  
 item.id === savedBook.id ? savedBook : item  
 )  
 : [...prevBooks, savedBook]  
 );  
 } catch (error) {  
 *console*.error('Error saving book:', error);  
 } finally {  
 setIsModalOpen(false);  
 }  
 };  
  
 const handleDeleteBook = async (bookId) => {  
 try {  
 await deleteBook(bookId);  
 setBooks((prevBooks) =>  
 prevBooks.filter((item) => item.id !== bookId)  
 );  
 } catch (error) {  
 *console*.error('Error deleting book:', error);  
 }  
 };  
  
 if (loading) {  
 return <div className="loading">Loading...</div>;  
 }  
  
 return (  
 <div className="book-list-container">  
 <input  
 type="text"  
 placeholder="Пошук по назві книги..."  
 value={searchQuery}  
 onChange={(e) => setSearchQuery(e.target.value)}  
 className="search-input"  
 />  
 <button onClick={handleAddBook} className="add-button">  
 Зареєструвати книгу  
 </button>  
 <div className="book-list">  
 {filteredBooks.map((book) => (  
 <div key={book.id} className="book-card">  
 <h3>{book.title}</h3>  
 <p>{book.description}</p>  
 <div className="book-details">  
 <span><strong>Автор:</strong> {book.author}</span>  
 <span><br/><strong>Жанр:</strong> {book.genre}</span>  
 <span><br/><strong>Статус:</strong> {book.status}</span>  
 <span><br/><strong>Тримач книги:</strong> {book.borrower ? book.borrower.name + " " + book.borrower.surname : 'Бібліотека'}</span>  
 </div>  
 <div className="book-actions">  
 <button onClick={() => handleEditBook(book)} className="edit-button">  
 Редагувати  
 </button>  
 <button onClick={() => handleDeleteBook(book.id)} className="delete-button">  
 Видалити  
 </button>  
 </div>  
 </div>  
 ))}  
 </div>  
 {isModalOpen && (  
 <BookModal  
 book={currentBook}  
 onClose={handleCloseModal}  
 onSave={handleSaveBook}  
 />  
 )}  
 </div>  
 );  
}  
  
function BookModal({ book, onClose, onSave }) {  
 const [title, setTitle] = useState(book ? book.title : '');  
 const [author, setAuthor] = useState(book ? book.author : '');  
 const [genre, setGenre] = useState(book ? book.genre : 'Наукова фантастика');  
 const [description, setDescription] = useState(book ? book.description : '');  
 const [status, setStatus] = useState(book ? book.status : 'В наявності');  
 const [borrowerId, setBorrowerId] = useState(book && book.borrower ? book.borrower.id : 0);  
 const [visitors, setVisitors] = useState([]);  
  
 useEffect(() => {  
 setTitle(book ? book.title : '');  
 setAuthor(book ? book.author : '');  
 setGenre(book ? book.genre : 'Наукова фантастика');  
 setDescription(book ? book.description : '');  
 setStatus(book ? book.status : 'В наявності');  
 setBorrowerId(book && book.borrower ? book.borrower.id : 0);  
 }, [book]);  
  
 useEffect(() => {  
 if (status === 'Видана на руки') {  
 getVisitors();  
 }  
 }, [status]);  
  
 const getVisitors = async () => {  
 try {  
 const data = await fetchVisitors();  
 setVisitors(data);  
 *console*.log('Visitors fetched:', data);  
 } catch (error) {  
 *console*.error('Error fetching visitors:', error);  
 }  
 };  
  
 const handleSave = () => {  
 const newBook = {  
 id: book ? book.id : 0,  
 title,  
 author,  
 genre,  
 description,  
 status,  
 borrowerId: status === 'Видана на руки' && borrowerId !== 0 ? borrowerId : null,  
 createdAt: book ? book.createdAt : new *Date*().toISOString(),  
 updatedAt: new *Date*().toISOString(),  
 };  
  
 onSave(newBook);  
 };  
  
 return (  
 <div className="modal-overlay">  
 <div className="modal-content">  
 <h2>{book ? 'Редагувати інформацію про книгу' : 'Зареєструвати книгу'}</h2>  
 <input  
 type="text"  
 placeholder="Назва книги"  
 value={title}  
 onChange={(e) => setTitle(e.target.value)}  
 />  
 <input  
 type="text"  
 placeholder="Автор"  
 value={author}  
 onChange={(e) => setAuthor(e.target.value)}  
 />  
 <select value={genre} onChange={(e) => setGenre(e.target.value)}>  
 <option value="Наукова фантастика">Наукова фантастика</option>  
 <option value="Фантастика">Фантастика</option>  
 <option value="Мелодрама">Мелодрама</option>  
 <option value="Містика">Містика</option>  
 <option value="Художня література">Художня література</option>  
 </select>  
 <textarea  
 placeholder="Опис"  
 value={description}  
 onChange={(e) => setDescription(e.target.value)}  
 />  
 <select value={status} onChange={(e) => setStatus(e.target.value)}>  
 <option value="В наявності">В наявності</option>  
 <option value="Видана на руки">Видана на руки</option>  
 </select>  
  
 {status === 'Видана на руки' && (  
 <select value={borrowerId} onChange={(e) => setBorrowerId(parseInt(e.target.value))}>  
 <option value="0">Оберіть тримача книги</option>  
 {visitors.map((visitor) => (  
 <option key={visitor.id} value={visitor.id}>  
 {visitor.surname} {visitor.name} {visitor.lastName}  
 </option>  
 ))}  
 </select>  
 )}  
  
 <div className="modal-actions">  
 <button onClick={handleSave} className="save-button">Зберегти</button>  
 <button onClick={onClose} className="cancel-button">Відхилити</button>  
 </div>  
 </div>  
 </div>  
 );  
}

import React, { useState, useEffect } from 'react';  
import image1 from '../../assets/image1.png';  
import image2 from '../../assets/image2.png';  
import image3 from '../../assets/image3.png';  
import './ControlledCarousel.css';  
  
export default function ControlledCarousel() {  
 const images = [image1, image2, image3];  
 const captions = [  
 {  
 title: 'Розташування',  
 text: 'Зручне розташування в центрі міста. До бібліотеки можна дістатись з будь якого кутка міста менш ніж 30 хвилин',  
 },  
 {  
 title: 'Матеріальна база',  
 text: 'Наявність великої кіькості книжок на будь який смак',  
 },  
 {  
 title: 'Атмосфера',  
 text: 'Приємний вайб для зручного читання в затишній атмосфері футуристичнох бібліотеки',  
 }  
 ];  
  
 const [index, setIndex] = useState(0);  
  
 const handlePrev = () => {  
 setIndex(index === 0 ? images.length - 1 : index - 1);  
 };  
  
 const handleNext = () => {  
 setIndex(index === images.length - 1 ? 0 : index + 1);  
 };  
  
 useEffect(() => {  
 const interval = setInterval(() => {  
 handleNext();  
 }, 3000);  
 return () => clearInterval(interval);  
 }, [index]);  
  
 return (  
 <div className="carousel-container">  
 <div className="carousel">  
 <img src={images[index]} alt={captions[index].title} className="carousel-image" />  
 <div className="carousel-caption">  
 <h3>{captions[index].title}</h3>  
 <p>{captions[index].text}</p>  
 </div>  
 <button className="carousel-control prev" onClick={handlePrev}>&#10094;</button>  
 <button className="carousel-control next" onClick={handleNext}>&#10095;</button>  
 </div>  
 <div className="carousel-indicators">  
 {images.map((\_, i) => (  
 <span  
 key={i}  
 className={`indicator ${i === index ? 'active' : ''}`}  
 onClick={() => setIndex(i)}  
 ></span>  
 ))}  
 </div>  
 </div>  
 );  
}

import React from 'react';  
import './Contacts.css';  
  
export default function Contacts() {  
 return (  
 <div className="contacts-container">  
 <h1 className="contacts-title">Контакти бібліотеки</h1>  
 <section className="contacts-section">  
 <p className="contacts-description">  
 Національна бібліотека — це провідний науково-дослідний і навчальний центр, який забезпечує студентів і дослідників доступом до сучасної літератури. Бібліотека відіграє важливу роль у підготовці фахівців, забезпечуючи інформаційні ресурси та сприяючи їх професійному розвитку.  
 </p>  
 </section>  
 <section className="contacts-section">  
 <h2 className="contacts-subtitle">Контактна інформація</h2>  
 <p className="contacts-info"><strong>Адреса:</strong> вул. Просвітницька, 45, м. Київ, Україна</p>  
 <p className="contacts-info"><strong>Телефон:</strong> +380 (44) 123 4567</p>  
 <p className="contacts-info"><strong>Email:</strong> library@university.ua</p>  
 <p className="contacts-info"><strong>Робочий час:</strong> Пн-Пт 09:00 - 18:00</p>  
 </section>  
 <section className="contacts-mapSection">  
 <h2 className="contacts-subtitle">Наше розташування на карті</h2>  
 <div className="contacts-mapContainer">  
 <iframe  
 title="Google Map"  
 src="https://www.google.com/maps/embed?pb=!1m16!1m12!1m3!1d10164.828074069079!2d30.480668899111762!3d50.43724451793718!2m3!1f0!2f0!3f0!3m2!1i1024!2i768!4f13.1!2m1!1z0LLRg9C7LiDQn9GA0L7RgdCy0ZbRgtC90LjRhtGM0LrQsCwgNDUsINC8LiDQmtC40ZfQsiwg0KPQutGA0LDRl9C90LA!5e0!3m2!1sru!2sua!4v1728328478200!5m2!1sru!2sua"  
 className="contacts-map"  
 allowFullScreen=""  
 loading="lazy"  
 referrerPolicy="no-referrer-when-downgrade"  
 ></iframe>  
 </div>  
 </section>  
 </div>  
 );  
}

import React from 'react';  
import { FontAwesomeIcon } from '@fortawesome/react-fontawesome';  
import { *faFacebook*, *faTwitter*, *faInstagram* } from '@fortawesome/free-brands-svg-icons';  
import { *Link* } from "react-router-dom";  
import './Footer.css';  
  
function Footer() {  
 return (  
 <footer className="footer">  
 <div className="footer-container">  
 <div className="footer-section">  
 <h5>Національна бібліотека Ревуцького</h5>  
 <p>  
 Всі права зарезервовані Ревуцький inc. Будь яке викривлення, або зловживання інформацією - заборонено!  
 </p>  
 </div>  
 <div className="footer-section">  
 <h5>Quick Links</h5>  
 <ul className="footer-links">  
 <li><Link to={"/"}>Головна</Link></li>  
 <li><Link to={"/about"}>Про нас</Link></li>  
 <li><Link to={"/gallery"}>Галерея</Link></li>  
 <li><Link to={"/news"}>Новини</Link></li>  
 </ul>  
 </div>  
 <div className="footer-section">  
 <h5>Follow Us</h5>  
 <ul className="footer-socials">  
 <li>  
 <a href="https://facebook.com">  
 <FontAwesomeIcon icon={*faFacebook*} /> Facebook  
 </a>  
 </li>  
 <li>  
 <a href="https://twitter.com">  
 <FontAwesomeIcon icon={*faTwitter*} /> Twitter  
 </a>  
 </li>  
 <li>  
 <a href="https://instagram.com">  
 <FontAwesomeIcon icon={*faInstagram*} /> Instagram  
 </a>  
 </li>  
 </ul>  
 </div>  
 </div>  
 <div className="footer-bottom">  
 <p>&copy; {new *Date*().getFullYear()} Company Name. All rights reserved.</p>  
 </div>  
 </footer>  
 );  
}  
  
export default Footer;

import React, { useState, useEffect } from 'react';  
import { *Container*, *Row*, *Col*, *Card*, *Modal*, *Button*, *Alert* } from 'react-bootstrap';  
  
export default function GalleryList() {  
 const [images, setImages] = useState([]);  
 const [show, setShow] = useState(false);  
 const [selectedImage, setSelectedImage] = useState('');  
  
 useEffect(() => {  
 const getImages = async () => {  
 try {  
 const baseURL = `http://${*process*.env.REACT\_APP\_SERVER\_MAIN\_HOST}:5000/photos/gallery/img`;  
 const imageCount = 6;  
 const generatedImages = *Array*.from({ length: imageCount }, (\_, index) => {  
 if (index % 1 === 0) {  
 return {  
 url: `${baseURL}${index + 1}.png`  
 };  
 }  
 return null;  
 }).filter(Boolean);  
 setImages(generatedImages);  
 } catch (error) {  
 *console*.error('Error fetching images:', error);  
 }  
 };  
  
 getImages();  
 }, []);  
  
 const handleShow = (image) => {  
 setSelectedImage(image);  
 setShow(true);  
 };  
  
 const handleClose = () => {  
 setShow(false);  
 };  
  
 if (images.length === 0) {  
 return (  
 <Container className="mt-3">  
 <Alert variant="warning" className="text-center">  
 Помилка! Жодної фотографії не додано!  
 </Alert>  
 </Container>  
 );  
 }  
  
 return (  
 <Container className="mt-3 gallery-container">  
 <Row>  
 {images.map((image, index) => (  
 <Col key={index} md={4} className="mb-4">  
 <Card className="gallery-card">  
 <Card.Img   
 variant="top"   
 src={image.url}   
 alt="Image"  
 onClick={() => handleShow(image.url)}  
 className="gallery-image"  
 />  
 </Card>  
 </Col>  
 ))}  
 </Row>  
 <Modal show={show} onHide={handleClose} centered>  
 <Modal.Header closeButton>  
 <Modal.Title>Перегляд зображення</Modal.Title>  
 </Modal.Header>  
 <Modal.Body>  
 <img src={selectedImage} alt="Selected" style={{ width: '100%', borderRadius: '10px' }}/>  
 </Modal.Body>  
 <Modal.Footer>  
 <Button variant="secondary" onClick={handleClose}>  
 Закрити  
 </Button>  
 </Modal.Footer>  
 </Modal>  
 </Container>  
 );  
}

import React, { useState } from "react";  
import { *Form*, *Button*, *Alert* } from "react-bootstrap";  
import *Modal* from "react-bootstrap/Modal";  
import *CloseButton* from "react-bootstrap/CloseButton";  
import { useNavigate } from "react-router-dom";  
import "./LoginPage.css";  
import { loginUser } from "../../../services/AuthService";  
  
export default function LoginPage() {  
 const [email, setEmail] = useState("");  
 const [password, setPassword] = useState("");  
 const [error, setError] = useState("");  
 const navigate = useNavigate();  
  
 const handleSubmit = async (event) => {  
 event.preventDefault();  
 const credentials = { email, password };  
 try {  
 await loginUser(credentials);  
 navigate("/");  
 } catch (err) {  
 setError(err.message); // Устанавливаем сообщение об ошибке  
 *console*.error('Login failed:', err);  
 }  
 };  
  
 const handleExit = () => {  
 navigate("/");  
 };  
  
 return (  
 <div className="login-container">  
 <Modal.Dialog className="login-dialog">  
 <Modal.Header>  
 <Modal.Title className="login-title">Вхід в систему</Modal.Title>  
 <CloseButton onClick={handleExit} className="close-button" />  
 </Modal.Header>  
  
 <Modal.Body>  
 {error && <Alert variant="danger">{error}</Alert>}  
 <Form onSubmit={handleSubmit}>  
 <Form.Group controlId="formBasicEmail">  
 <Form.Label>Email</Form.Label>  
 <Form.Control  
 type="email"  
 placeholder="Введіть email"  
 value={email}  
 onChange={(e) => setEmail(e.target.value)}  
 required  
 className="login-input"  
 />  
 </Form.Group>  
  
 <Form.Group controlId="formBasicPassword">  
 <Form.Label>Пароль</Form.Label>  
 <Form.Control  
 type="password"  
 placeholder="Введіть пароль"  
 value={password}  
 onChange={(e) => setPassword(e.target.value)}  
 required  
 className="login-input"  
 />  
 </Form.Group>  
  
 <Button variant="primary" type="submit" className="login-button w-100 mt-3">  
 Увійти  
 </Button>  
 </Form>  
 </Modal.Body>  
 </Modal.Dialog>  
 </div>  
 );  
}

import { *Link*, useNavigate } from "react-router-dom";  
import { useEffect, useState } from "react";  
import { getEmail } from "../../services/AuthService";  
import logo from "../../assets/logo.png";  
import "./NavBar.css"; // Подключаем файл со стилями  
  
export default function NavBar() {  
 const [email, setEmail] = useState(null);  
 const navigate = useNavigate();  
  
 useEffect(() => {  
 const fetchEmail = async () => {  
 try {  
 const res\_email = await getEmail();  
 if (res\_email) {  
 setEmail(res\_email);  
 } else {  
 *console*.log("Email not found");  
 }  
 } catch (error) {  
 *console*.error("Error fetching email:", error);  
 }  
 };  
 fetchEmail();  
 }, []);  
  
 const handleLogout = () => {  
 *localStorage*.removeItem('jwtToken');  
 setEmail(null);  
 navigate('/');  
 };  
  
 return (  
 <nav className="navbar">  
 <div className="navbar-container">  
 <div className="navbar-brand">  
 <Link to="/">  
 <img src={logo} alt="Logo" className="navbar-logo" />  
 </Link>  
 <Link to="/" className="navbar-title">  
 Національна бібліотека Ревуцького  
 </Link>  
 </div>  
 <div className="navbar-links">  
 <Link to="/books" className="navbar-link">Книжки</Link>  
 <Link to="/gallery" className="navbar-link">Галерея</Link>  
 <Link to="/news" className="navbar-link">Новини</Link>  
 <Link to="/contacts" className="navbar-link">Контакти</Link>  
 </div>  
 <div className="navbar-actions">  
 {email ? (  
 <>  
 <span className="navbar-email">Користувач: {email}</span>  
 <button onClick={handleLogout} className="navbar-logout">  
 Вийти  
 </button>  
 </>  
 ) : (  
 <Link to="/login">  
 <button className="navbar-login">Увійти</button>  
 </Link>  
 )}  
 </div>  
 </div>  
 </nav>  
 );  
}

import React, { useState, useEffect } from 'react';  
import { *Alert* } from 'react-bootstrap';  
import { fetchNews } from '../../services/NewsService'; // Убедитесь, что путь правильный  
import './NewsList.css';  
  
export default function NewsList() {  
 const [news, setNews] = useState([]);  
  
 useEffect(() => {  
 const getNews = async () => {  
 try {  
 const data = await fetchNews();  
 setNews(data);  
 } catch (error) {  
 *console*.error('Error fetching news:', error);  
 setNews([]); // Устанавливает пустой массив новостей в случае ошибки  
 }  
 };  
  
 getNews();  
 }, []);  
  
 if (news.length === 0) {  
 return (  
 <div className="news-container">  
 <Alert variant="warning" className="news-alert">  
 Помилка! Немає новин.  
 </Alert>  
 </div>  
 );  
 }  
  
 return (  
 <div className="news-container">  
 {news.map((item, index) => (  
 <div key={index} className="news-card">  
 <img src={`http://${*process*.env.REACT\_APP\_SERVER\_MAIN\_HOST}:5000/photos/news/${item.photo}`} alt="News" className="news-image"/>  
 <div className="news-content">  
 <h3 className="news-title">{item.title}</h3>  
 <p className="news-text">{item.text}</p>  
 </div>  
 </div>  
 ))}  
 </div>  
 );  
}