Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних наук
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка сховища нормативних документів Тернопільського національного технічного університету імені Івана Пулюя з веб-інтерфейсом користувача

Виконав: студент IV курсу, групи СНс-42
спеціальності 122 Комп'ютерні науки
(шифр і назва спеціальності)

_____ Демчук В.С.
(підпис) (прізвище та ініціали)

_____ Бекер І.М.
(підпис) (прізвище та ініціали)

Керівник _____ Готович В.А.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Шимчук Г.В
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ Боднарчук І.О.
(підпис) (прізвище та ініціали)

Рецензент _____ Бойко І.В.
(підпис) (прізвище та ініціали)

Тернопіль
2022

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра комп'ютерних наук
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Боднарчук І.О.
(підпис) (прізвище та ініціали)
«__» _____ 2022 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавр
(назва освітнього ступеня)

за спеціальністю 122 Комп'ютерні науки
(шифр і назва спеціальності)

Студенту Бекеру Івану Миколайовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка сховища нормативних документів Тернопільського національного технічного університету імені Івана Пулюя з веб-інтерфейсом користувача

Керівник роботи Готович Володимир Анатолійович, к.т.н.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від « 16 » березня 2021 року № 4/7-161

2. Термін подання студентом завершеної роботи 22.06.2022р.

3. Вихідні дані до роботи Літературні та Інтернет-джерела

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. 1. Аналіз поставленого завдання. 1.1 Вимоги до функціональних характеристик. 1.2 Вибір технологій розробки та мови програмування. 1.3 Вимоги до програмної документації. 1.4 Стадії та етапи розробки.1.7 Висновки до першого розділу 2. Реалізація програмного рішення. 2.1 Розробка загальної структури сервера 2.3 Забезпечення політики безпееки доступу до даних на сервері 2.4 Опис структури запитів 2.8 Система кешування 2.9 Тестування програми. 2.10 Висновки до другого розділу 3. Безпека життєдіяльності, основи охорони праці 3.3 Небезпечні і шкідлтві виробничі фактори, джерелом яких є комп'ютер 3.4 Заходи захисту користувачів ПК від шуму вібрції Та випромінювань. 3.5 Висновки до третього розділу. Перелік джерел. Лістинг файлів клієнтської частини А. Лістинг файлів серверної частини В

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці	Гурик О.Я., к.т.н., доцент, доцент кафедри МТ		

7. Дата видачі завдання 24 січня 2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	24.01.2022	*Виконано*
2.	Підбір джерел про створення програмного забезпечення	04.01.2022-30.01.2022	*Виконано*
3.	Переклад та опрацювання джерел про створення сервера	31.01.2022-06.02.2022	*Виконано*
4.	Виконання дослідження щодо вибору серверної архітектури і реляційних баз даних	07.02.2022-13.02.2022	*Виконано*
	Розроблення серверної частини програми		
5.	Оформлення розділу «Аналіз поствленого завдання»	14.02.2022-06.03.2022	*Виконано*
6.	Оформлення розділу «Реалізація програмного рішення»	07.03.2022-03.04.2022	*Виконано*
7.	Виконання завдання до підрозділу «Безпека життєдіяльності»	04.04.2022-17.04.2022	*Виконано*
8.	Виконання завдання до підрозділу «Основи охрони	18.04.2022-01.05.2022	*Виконано*
	праці»		
9.	Оформлення кваліфікаційної роботи	02.05.2022-15.05.2022	*Виконано*
10.	Нормоконтроль	16.05.2022-22.05.2022	*Виконано*
11.	Перевірка на плагіат	08.06.2022	*Виконано*
12.	Попередній захист кваліфікаційної роботи	09.06.2022	*Виконано*
13.	Захист кваліфікаційної роботи	22.06.2022	

Студент _____ Бекер І.М.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Готович В.А.
(підпис) (прізвище та ініціали)

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра комп'ютерних наук
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Боднарчук І.О.
(підпис) (прізвище та ініціали)
«__» _____ 2022 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавр
(назва освітнього ступеня)

за спеціальністю 122 Комп'ютерні науки
(шифр і назва спеціальності)

Студенту Демчуку Василю Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка сховища нормативних документів Тернопільського національного технічного університету імені Івана Пулюя з веб-інтерфейсом користувача

Керівник роботи Готович Володимир Анатолійович, к.т.н.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від « 16 » березня 2022 року № 4/7-161

2. Термін подання студентом завершеної роботи 22.06.2022р.

3. Вихідні дані до роботи Літературні та Інтернет-джерела

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. 1. Аналіз поставленого завдання. 1.1 Вимоги до функціональних характеристик. 1.2 Вибір технологій розробки та мови програмування. 1.5 Порядок тестування та прийому. 1.6 Складність пошукової оптимізації 1.7 Висновки. 2. Реалізація програмного рішення. 2.2 Розробка структури сайту. 2.5 Створення інтерфейсу користувача. 2.6 Відправлення запитів на сервер. 2.7 Особливості відображення PDF, DOC і DOCX файлів на сторінці. 2.10 Висновки до друго розділу. 3. Безпека життєдіяльності, основи охорони праці 3.1 Характеристи скарг працівників та посадових осіб, які працюють більше половини робочого дня за комп'ютером. 3.2 Причини пожеж у приміщенні з ПК. Висновки Перелік джерел. Лістинг файлів клієнтської частини А. Лістинг файлів серверної частини В

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці	Гурик О.Я., к.т.н., доцент, доцент кафедри МТ		

7. Дата видачі завдання 24 січня 2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	24.01.2022	*Виконано*
2.	Підбір джерел про створення програмного забезпечення	04.01.2022-30.01.2022	*Виконано*
3.	Переклад та опрацювання джерел про стоверення клієнтської частини	31.01.2022-06.02.2022	*Виконано*
4.	Виконання дослідження щодо створення адаптивного і кросбраузерного сайту	07.02.2022-13.02.2022	*Виконано*
	Розроблення клієнтської частини програми		
5.	Оформлення розділу «Аналіз поствленого завдання»	14.02.2022-06.03.2022	*Виконано*
6.	Оформлення розділу «Реалізація програмного рішення»	07.03.2022-03.04.2022	*Виконано*
7.	Виконання завдання до підрозділу «Безпека життєдіяльності»	04.04.2022-17.04.2022	*Виконано*
8.	Виконання завдання до підрозділу «Основи ожрони	18.04.2022-01.05.2022	*Виконано*
	праці»		
9.	Оформлення кваліфікаційної роботи	02.05.2022-15.05.2022	*Виконано*
10.	Нормоконтроль	16.05.2022-22.05.2022	*Виконано*
11.	Перевірка на плагіат	08.06.2022	*Виконано*
12.	Попередній захист кваліфікаційної роботи	09.06.2022	*Виконано*
13.	Захист кваліфікаційної роботи	22.06.2022	

Студент _____ Демчук В.С.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Готович В.А.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Розробка сховища нормативних документів Тернопільського національного технічного університету імені Івана Пулюя з веб-інтерфейсом користувача // Кваліфікаційна робота освітнього рівня «Бакалавр» // Демчук Василь Сергійович, Бекер Іван Миколайович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра комп'ютерних наук, група СНс-42 // Тернопіль, 2022 // С. 69, рис. – 22, табл. – 0, кресл. – , додат. – 2, бібліогр. – 35.

Ключові слова: електронне сховище документів, клієнт-серверна архітектура, веб-інтерфейс, REST API.

Кваліфікаційна робота присвячена розробці клієнт-серверного програмного рішення для реалізації сховища нормативних документів організації з веб-інтерфейсом користувача.

Мета роботи: створення клієнт-серверного застосунку нормативної бази Тернопільського національного технічного університету імені Івана Пулюя.

У першому розділі кваліфікаційної роботи розглянуто теоретичні відомості щодо створення додатку і перелік функціональних вимог серверного та клієнтського застосунків.

У другому розділі кваліфікаційної роботи розглянуто практичну частину розробки клієнт-серверної програми, технології створення додатку.

У третьому розділі кваліфікаційної роботи розглянуто основні питання з охорони праці і безпеки життєдіяльності.

У додатках до роботи прозміщено лістинги файлів клієнтського та серверного додатків.

ANNOTATION

Development of a repository of normative documents of Ternopil National Technical University named after Ivan Pulyuy with web user interface // Qualification work of educational level "Bachelor" // Demchuk Vasyl, Beker Ivan // Ternopil National Technical University named after Ivan Pulyuy, Faculty of Computer and Information Systems and Software Engineering, Department of Computer Science, SNs-42 group // Ternopil, 2022 // P. 69, pic. - 22, table. - 0, drawing. - , addition. - 2, bibliogr. - 35.

Keywords: electronic document repository, client-server architecture, web interface, REST API.

Qualification work is devoted to developing client-server software solutions for the implementation of the repository of regulatory documents of the organization with a web user interface.

Purpose: create a client-server application of the regulatory framework of Ternopil National Technical University named after Ivan Pulyuy.

The first section of the qualification work discusses theoretical information on application development and a list of functional requirements for server and client applications.

In the second section of the qualification work the practical part of client-server program development, and application creation technology is considered.

In the third section of the qualification work, the main issues of labor protection and life safety are considered.

The application appendices contain listings of client and server application files.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API (англ. Application Programming Interface) – опис способів якими одна комп'ютерна програма може взаємодіяти з іншою програмою.

HTTP (англ. Hypertext Transfer Protocol) – протокол прикладного рівня передачі даних.

JSON (англ. JavaScript Object Notation) – текстовий формат обміну даними на основі JavaScript.

JWT (англ. JSON Web Token) – це відкритий стандарт для створення токенів доступу.

MIT (англ. Massachusetts Institute of Technology) – Массачусетський технологічний інститут.

REST (англ. Representational State Transfer) – архітектурний стиль взаємодії компонентів розподіленої програми у мережі.

SOAP (англ. Simple Object Access Protocol) – протокол обміну структурованими повідомленнями у розподіленому обчислювальному середовищі.

SPA (англ. Single page application) – односторінковий додаток.

SQL (англ. Structured query language) – декларативна мова програмування, яка застосовується для створення, модифікації та управління даними в реляційній базі даних.

UI (англ. User interface) – користувакий інтерфейс.

URL (англ. Uniform Resource Locator) – стандартизована адреса певного ресурсу.

UX (англ. User experience) – користувацький досвід.

ОС – Операційна система.

ПЗ – програмне забезпечення.

ШІ – Штучний інтелект.

ЗМІСТ

ВСТУП

Актуальність теми. У наш час інформація є таким самим товаром, як і продукція. Багато підприємств визначають основним джерелом інформації щодо інновацій участь у закордонних виставках, контакти з іноземними фахівцями, стажування на відомих і великих підприємствах, закордонні відрядження. Однак, оскільки такі можливості доступні далеко не всім підприємствам, то наявної інформації недостатньо.

Інформація, особливо її автоматизована обробка, і тепер залишається важливим фактором підвищення ефективності діяльності будь-якого підприємства. Важливу роль у використанні інформації відіграють способи її реєстрації, обробки, нагромадження і передачі, систематизоване збереження інформації і її видача в потрібній формі.

В сучасних умовах у великих організаціях створені і ефективно діють інформаційні системи, які обслуговують процес підготовки і прийняття управлінських рішень і вирішують наступні задачі: обробку інформації, реалізацію інтелектуальної діяльності з метою створення інформації. Інформаційна система являє собою сукупність організаційних, технічних, програмних і інформаційних засобів, об'єднаних в єдину систему з метою збору, зберігання, обробки і видачі необхідної інформації, призначена для виконання заданих функцій.

В наш час є багато різних інформаційних систем, які виконують різні завдання і функції для вирішення тієї чи іншої проблеми .

Однією із таких проблем є зберігання і структуризація документів та інших даних.

Мета і задачі дослідження. Метою даної кваліфікаційної роботи освітнього рівня «Бакалавр» є розробка клієнт-серверного програмного рішення для реалізації сховища нормативних документів організації з веб-інтерфейсом користувача.

Для реалізації мети необхідно розв’язати такі **задачі**:

– провести аналіз відомих систем для зберігання документів;

– провести аналіз UI/UX в подібних програмних застосунках;

– дослідити сучасні методи створення програмного забезпечення;

– розробити архітектуру клієнтської та серверної частин;

– створити програмне рішення швидкодіючого сервера для зберігання нормативних документів;

– розробити зручний користувацький інтерфейс та панель адміністратора для відображення нормативних документів;

– закріпити практичні навички щодо розробки клієнт-серверних застосунків.

Практичне значення отриманих результатів полягає в розробці проекту сховища нормативних документів для університету, який запущено в експлуатацію в тестовому режимі.

Внесок авторів в роботу наступний:

– проведено аналіз відомих систем для зберігання документів (Бекер І.М., п. 1.1.1, 1.1.2);

– проведено аналіз UI/UX в подібних програмних застосунках (Демчук В.С., п. 1.1.3);

– досліджено сучасні методи створення програмного забезпечення (Бекер І.М., п. 1.2.1, 1.3, 1.4);

– розроблено архітектуру клієнтської та серверної частин програмного рішення (Демчук В.С., Бекер І.М., п. 1.2).;

– створено програмне рішення сервера для зберігання нормативних документів (Бекер І.М. п. 2.6, 2.8);

– розроблено зручний інтерфейс користувача та панель адміністратора (Демчук В.С., п. 1.6, 2.2, 2.5, 2.7, 2.8);

– закріплено практичні навички щодо розробки клієнт-серверних застосунків (Демчук В.С., Бекер І.М. п. 2.3, 2.9).

РОЗДІЛ 1. АНАЛІЗ ПОСТАВЛЕНОГО ЗАВДАННЯ

Опишемо результати проведеного аналізу по розробці програмного рішення для реалізації сховища нормативних документів Тернопільського національного технічного університету імені Івана Пулюя з веб-інтерфейсом користувача.

1.1 Вимоги до функціональних характеристик програмного рішення

1.1.1 Управління користувачами

Управління користувачами потрібне для надання можливості іншим людям керувати документами.

Адміністратор може додавати користувачів які можуть редагувати документи. За допомогою їхньої електронної пошти Адміністратор зможе надсилати запрошувані листи. У цих листах буде код для авторизації і у парі з поштою він дозволить користувачу зареєструватись. Ці користувачі зможуть керувати документами.

Також Адміністратор матиме змогу додавати видаляти користувачів або видалити дозвіл користувача на редагування та перегляд.

1.1.2 Управління документами

Управління документами є важливою функцією адміністрування. Управління документами повинне мати такі функції:

- створення категорії документів;
- зміна назви категорії;
- видалення категорії;
- створення запису документа;

– зміна запису документа;
– вивантаження документа;
– видалення документа.

Документ повинен мати такі дані:

– дата створення;
– ім’я;
– ідентифікатор (ID) категорії, до якої належить документ.

1.1.3 Перегляд документів

Перегляд документів повинен бути доступний для усіх користувачів. Окрім архівованих документів, вони повинні відображатись тільки зареєстрованим користувачам. Перегляд повинен мати такі функції:

– отримання запису документу по ID;
– пошук записів документів по частині імені;
– завантаження документу по ID;
– завантаження PDF версії документу по ID.

У даному проекті буде можливість використовувати будьякі формати файлів, максимальний розмір файлів можна буде налаштувати. Тільки певні розширення файлів зможуть використовуватись для конвертації у PDF формат. А саме файли форматів DOCX і DOC.

PDF-версії документів можна буде завантажити тільки тих документів які неможливо відобразити у браузері і які піддаються конвертації у PDF формат, а саме .doc і .docx формати.

Формат PDF (Portable Document Format) – це формат файлів для відображення документів створений компанією Adobe у 1992 році. Даний формат не залежить від програмного забезпечення або операційної системи. PDF побудований на основі PostScript мови і може містити текст, векторну графіку, шрифти, зображення та інші дані.

Файли PDF можуть містити різноманітний вміст, окрім плоского тексту та графіки, включаючи елементи логічної структури, інтерактивні елементи, такі як анотації та поля форми, шари, мультимедійний вміст (включаючи відеовміст), тривимірні об’єкти з використанням U3D або PRC та інші формати даних. Специфікація PDF також передбачає шифрування та цифрові підписи, вкладення файлів і метадані для забезпечення робочих процесів, які вимагають цих функцій.

DOCX – це формат файлів Microsoft Word. Введений у 2007 році з випуском Microsoft Office 2007, структура цього нового формату документа була змінена з простого двійкового на комбінацію XML та двійкових файлів. Файли Docx можна відкривати за допомогою Word 2007 і бічних версій, але не в попередніх версіях MS Word, які підтримують розширення файлів DOC.

Після того, як Microsoft відкрила специфікації формату файлу DOC, його конкурентам було легко перепроектувати формат і забезпечити таку ж підтримку у своїх власних програмах. Крім того, конкуренція з боку Open Office у формі відкритого формату документів змусила Microsoft прийняти більш відкриті та широкі стандарти. Це було на початку 2000 року, коли Microsoft вирішила внести зміни, щоб пристосувати стандарт Office Open XML . Документи згідно з цим новим стандартом отримали розширення .docx, «X» для XML. До 2007 року цей новий формат файлів став частиною Office 2007 і також використовується в нових версіях Microsoft Office. Новий тип файлу має переваги: невеликі розміри файлів, менша кількість пошкоджень і добре відформатовані зображення.

Спочатку формат створювався як заміна колишньому двійковому формату документів, який використовували програми Microsoft Office аж до версії Office 2003 включно, і конкурент стандартизований прямо перед цим OpenDocument. У 2006 році формат Office Open XML був оголошений вільним та відкритим форматом Ecma International. Він є стандартним форматом для програм Microsoft Office 2007 і пізніших.

1.2 Вибір технологій розробки та мови програмування

1.2.1 Технології розробки серверної частини

Для даного проекту було вибрано мову програмування Java з використанням фреймворком Spring [4].

Java (Див. рисунок 1.1) це мова програмування та платформа обчислень, яка була створенна Sun Microsystems у 1995 р. Java розроблялась як спрощена версіямови програмуванн C++ і відрізняється надійністю і високим рівнем захисту.

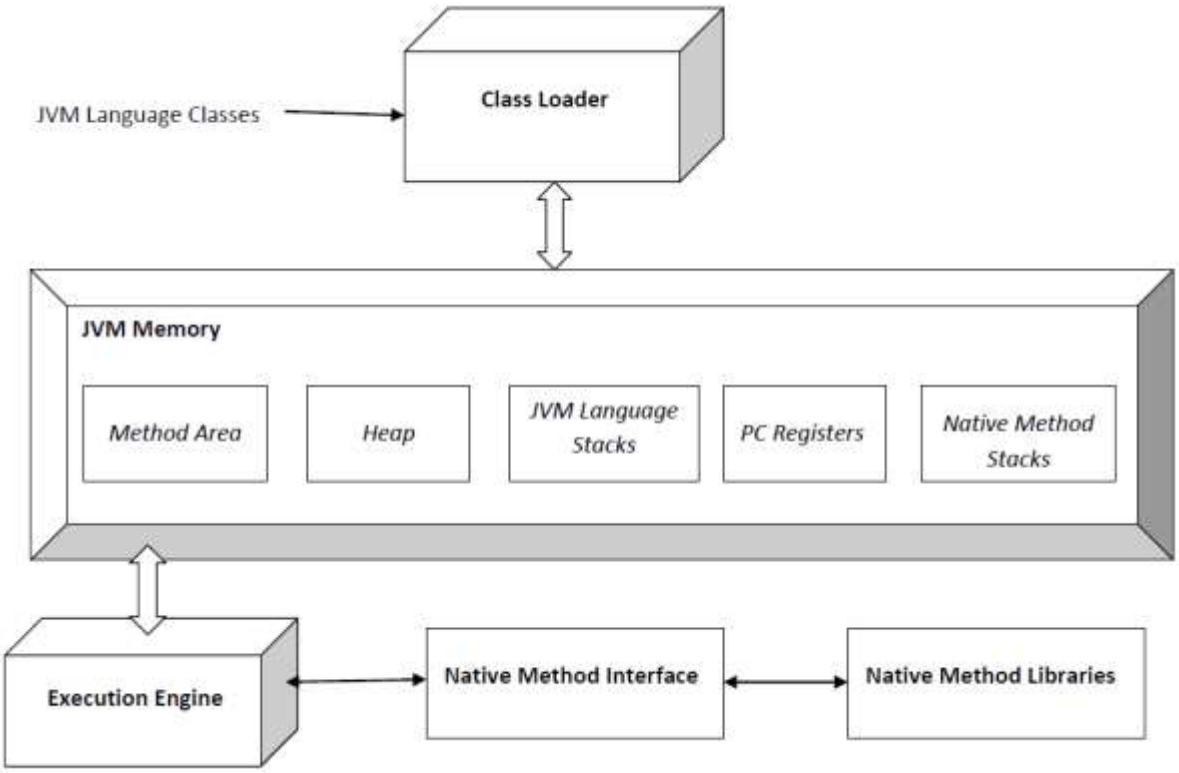

Рисунок 1.1 – Схема роботи JVM

Для роботи з мовою програмування Java потрібно встановити Java Runtime Environment (JRE). JRE складається з

– віртуальної машини (JVM);

– компілятора;

– базових бібліотек та класів;

– допоміжних бібліотек.

Віртуальна машина виконує байт-код створений компілятором Java, цей код є кросплатформенним і може запускатись на будь якому пристрої де встановлений JVM. Тому даний проект можна буде запускати на будь-якому пристрої або JVM [5].

Дана мова програмування є одною з найпопулярніших мов програмування у світі. Найчастіше на Java розробляють Android програми а також складні відмовостійкі системи. Одною з найпопулярніших віток розробки на Java є створення серверних застосунків із використанням фреймворку Spring.

Фреймворк – це платформа, яка містить набір інструментів для розробки певного типу програм а також готові інструменти для розробки структури цих програм. Основна відмінність фреймворку від бібліотеки це інверсія управління. У фреймворку основний код викликає код користувача який може реалізовувати конкретну поведінку ко вбудовується у код фреймворку. Також фреймворк може декларувати саму структуру програми. У даному проекті буде використовуватись Spring framework (Див. рисунок 1.2). Spring Framework (або коротко Spring) – фреймворк із відкритим вихідним кодом для Java-платформи. Початкова версія була написана Родом Джонсоном, який опублікував її разом із виданням своєї книги "Expert One-on-One Java EE Design and Development".

Spring Framework надає комплексну модель програмування та конфігурації для сучасних корпоративних програм на базі Java – на будь-якій платформі розгортання.

Ключовим елементом Spring є інфраструктурна підтримка на рівні програми: Spring зосереджується на «підготовці» корпоративних додатків, щоб команди могли зосередитися на бізнес-логіці на рівні програми без зайвих зв’язків із конкретними середовищами розгортання [6].

Spring надає легкий спосіб реалізації інверсії керування також відомий як ін'єкція залежностей (DI). Це процес, за допомогою якого об’єкти визначають свої залежності (тобто інші об’єкти, з якими вони працюють) лише за

допомогою аргументів конструктора, аргументів заводського методу або властивостей, які встановлюються для екземпляра об’єкта після його створення або повернення з заводського методу. Потім контейнер вводить ці залежності під час створення bean-компонента. Цей процес, є зворотним (звідси назва — інверсія керування) самого bean-компонента, який контролює створення або розташування його залежностей за допомогою прямого конструювання класів або механізму, такого як шаблон Service Locator [7].

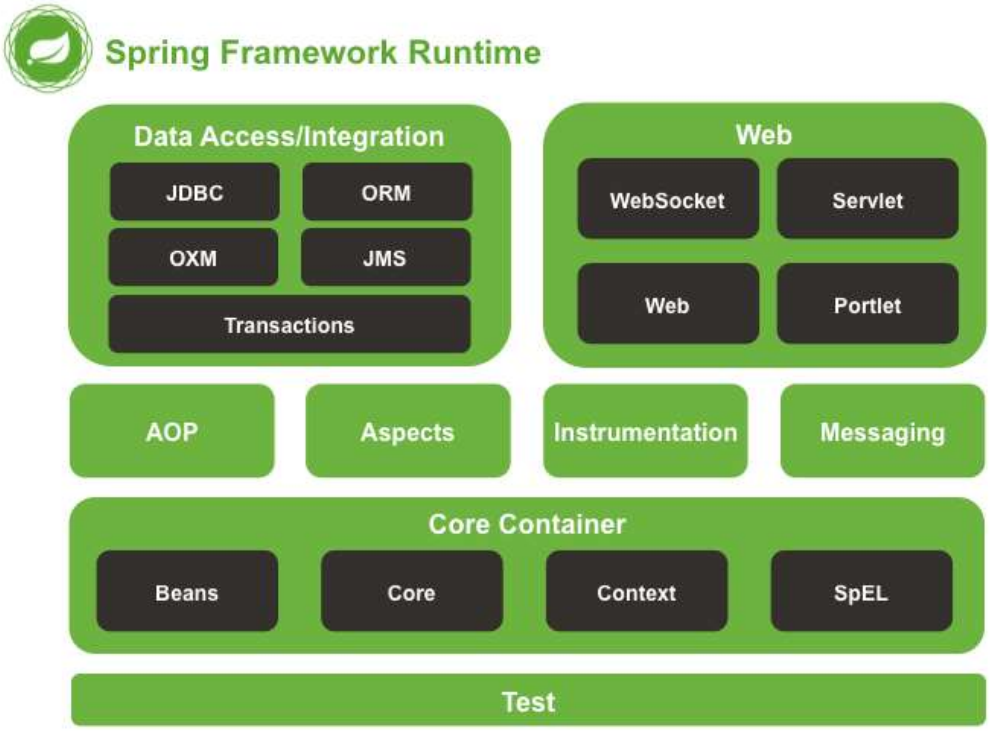

Рисунок 1.2 – Схема Spring Framework

Частиною фреймворку Spring є Spring Web MVC. Це веб-фреймворк, побудований на API Servlet і з самого початку був включений в Spring Framework. Формальна назва «Spring Web MVC» походить від назви вихідного модуля (spring-webmvc), але більш відомий як «Spring MVC».

Паралельно з Spring Web MVC, Spring Framework 5.0 представив веб-фреймворк із реактивним стеком, назва якого «Spring WebFlux» також заснований на вихідному модулі (spring-webflux).

Для забезпечення авторизації використовується Spring Security OAuth. Цей модуль забезпечує підтримку безпеки на основі маркерів, включаючи веб-токен JSON (JWT) і використовує це як механізм аутентифікації у веб-додатках (Див. рисунок 1.3), включаючи взаємодії STOMP через WebSocket, як це описано в попередньому розділі (тобто для підтримки ідентичності через сеанс на основі файлів cookie).

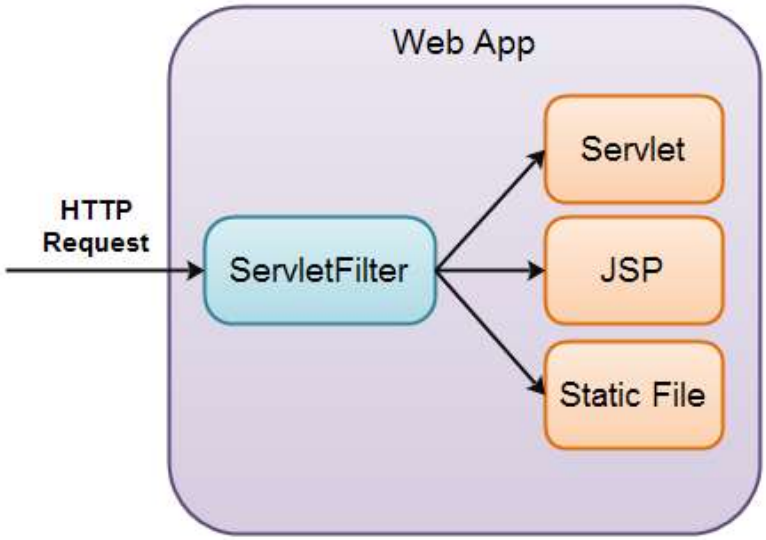

Рисунок 1.3 – Схема API Servlet

У той же час сеанси на основі cookie не завжди найкраще підходять (наприклад, у програмах, які не підтримують сеанс на стороні сервера, або в мобільних додатках, де зазвичай використовують заголовки для аутентифікації).

Розробка усього проекту може вимагати дуже багато залежностей на бібліотеки інших розробників, тому для кращого управління ними буде використано систему збірки Maven та плагін о неї під назвою Spring-Boot.

Система збірки Maven це фреймворк для збірки проекту у виконуваний файл .jar з усіма залежностями проекту на бібліотеки сторонніх розробників. Система використовує описання залежностей та інших данних у файлах на мові POM (Project Object Model).

Авторизація користувачів буде здійснюватись за допомогою OAuth 2.0 протоколу. Протокол OAuth 2.0 – це протокол авторизації, що дозволяє видати одному сервісу (додатку) права на доступ до ресурсів користувача на іншому сервісі. Протокол позбавляє необхідності довіряти додатку логін і пароль, а також дозволяє видавати обмежений набір прав, а не всі відразу.

Як і перша версія, OAuth 2.0 заснований на використанні базових веб-технологій: HTTP-запитах, редиректах тощо. для браузерів. Ключова відмінність від OAuth 1.0 – простота. У новій версії немає громіздких схем підпису, скорочено кількість запитів, необхідних авторизації. Загальна схема роботи програми, що використовує OAuth, є такою (Див. рисунок 1.4):

– отримання авторизації;
– звернення до захищених ресурсів.

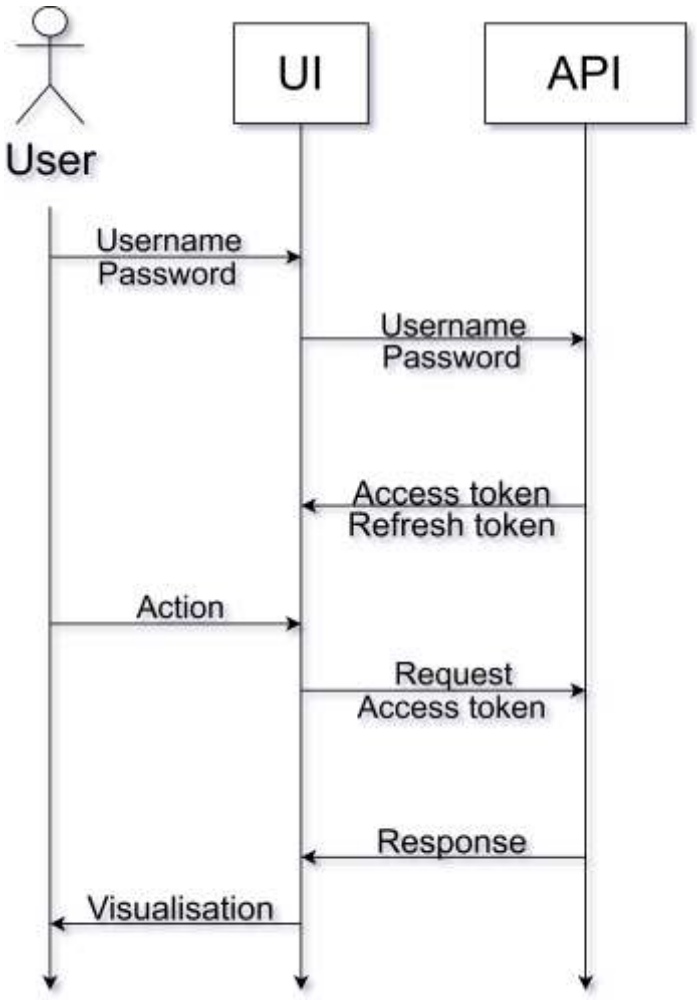

Рисунок 1.4 – Схема роботи OAuth 2.0

Результатом авторизації є access token – будь-який ключ (зазвичай просто набір символів), пред'явлення якого є перепусткою до захищених ресурсів. Зазвичай access token надсилають разом із запитом який вимагає авторизації, у Headers частині запиту. Для Web-Sockets access token надсилається і Query налаштуваннях, для високо рівневих сокет-з’єднань, або напряму у авторизаційному повідомлені, для низькорівневих сокет-з’єднань.

Звернення до них у найпростішому випадку відбувається за HTTPS із зазначенням у заголовках або як один з параметрів отриманого access token'a.

У протоколі описано кілька варіантів авторизації, що підходять для різних ситуацій:

– авторизація для додатків, що мають серверну частину (найчастіше, це сайти та веб-додатки);

– авторизація для повністю клієнтських програм (мобільні та desktop-додатки);

– авторизація за логіном та паролем;

– відновлення попередньої авторизації.

1.2.2 Технології розробки клієнтської частини

Архітектура SPA(single page application) – це буквально одна сторінка, яка постійно взаємодіє з користувачем, динамічно передає поточну сторінку, а не завантажує цілі нові сторінки з сервером.

Особливість архітектури SPA (Див. рисунок 1.5) полягає в тому, що всі елементи, необхідні для роботи програми, знаходяться на одній сторінці. Вони завантажуються при ініціалізації. Також цей вид додатків завантажує додаткові модулі після запиту від користувача. Люба активність користувача фіксується для зручності в навігації. Це дозволяє скопіювати посилання та відкривати додаток у тому самому етапі взаємодії на іншій вкладці, браузері чи пристрої [21].

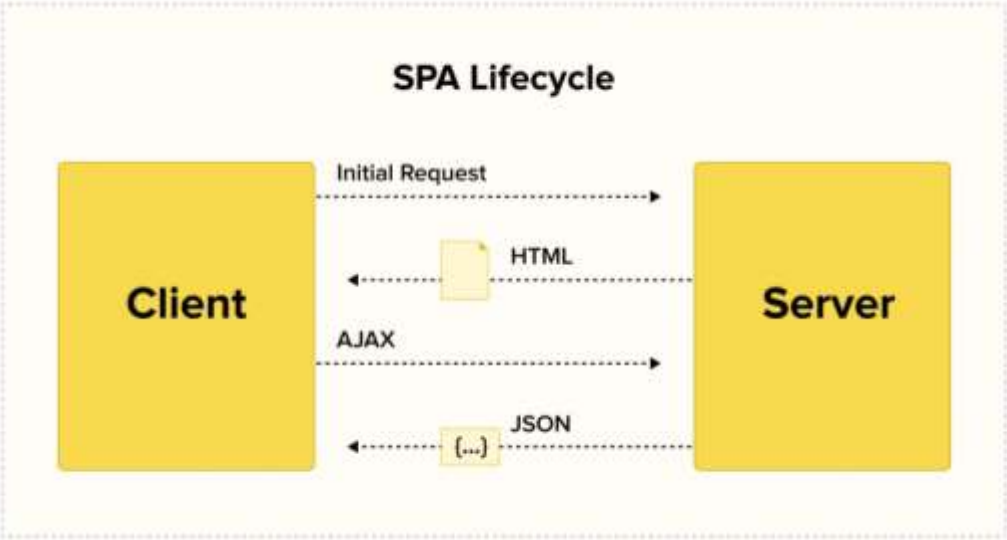

Рисунок 1.5 – Принцип роботи SPA додатка

При завантаженні нових модулів SPA контент на них оновлюється лише частково, так як немає необхідності повторно завантажувати незмінні елементи. Це збільшує швидкість відповіді і скорочує обсяг передавання даних між браузером і сервером [23].

Даний вид програмного забезпечення для способу взаємодії з користувачем більше всього схожий на роботу десктопних додатків, але на сервері.

Переваги Single Page Applications:

– Доступність. Можна отримати моментальний доступ до функціоналу з будь-якого типу пристрою без проблем сумісності, об'єму пам'яті, потужностей або часу на встановлення.

– Універсальність. Використовувати програму можна практично з будь-якого пристрою, якщо на ньому є доступ до інтернету. Якщо при розробці інтерфейсу враховувалися різні роздільні здатності екрану, то використовувати SPA однаково зручно і з ПК, і зі смартфона.

– Можливість використовувати великі обсяги даних. Розмір програми та даних, що використовуються, не обмежений пам'яттю пристрою.

– Швидкість. Одна сторінка з усім необхідним, яка економить час на повторне завантаження даних і підвищує продуктивність роботи.

– Можливості розробки. Розробникам доступні фреймворки, які спрощують створення архітектури проекту та надають чимало готових елементів для роботи.

Приклади зручних та корисних односторінкових програм – Gmail і Google Translate.

Недоліки SPA:

– Необхідність інтернет-з'єднання. Без доступу до мережі використовувати такий додаток неможливо. Але якщо навіть десктопне програмне забезпечення використовує в роботі зовнішні бази даних, то доступ до інтернету необхідний у будь-якому випадку.

– Проблеми з SEO. Особливості SPA ускладнюють або унеможливлюють процес індексації пошуковими системами всіх модулів програми. Це може спричинити труднощі з оптимізацією.

– Не працює у користувачів з відключеною підтримкою JS. Багато хто відключає відображення JS-елементів у себе в браузерах, а Single Page Application використовує їх у роботі, тому додаток може не працювати.

Фреймворк розробки. Оскільки додаток буде використовувати архітектуру SPA потрібно вибрати фраємворк для спрощення процесу розробки клієнтської частини. Серед найпопулярніших JavaScript фраємворків (Див. рисунок 1.6), які підтримують SPA, виділяють наступні:

– Angular.js.
– React.js.
– Meteor.
– Vue.js.
– Backbone.js.
– Ember.js.
– Polimer.js.
– Aurelia.js.

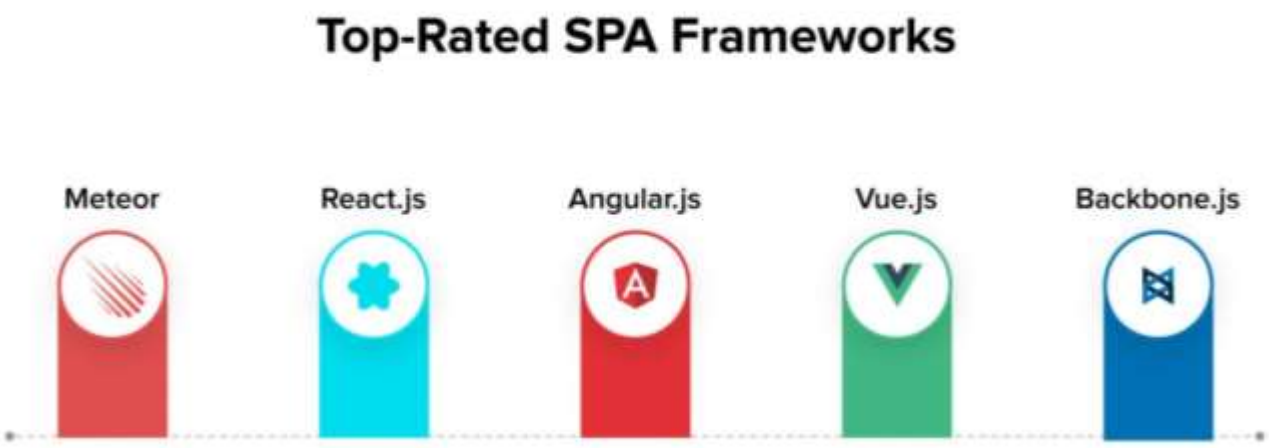

Рисунок 1.6 – Фреймворки для SPA

Великої популярності для створення SPA додатків набула бібліотека React.js, через яку можна будувати складні користувацькі інтерфейси. React забезпечує гнучкість розробки завдяки використанню "компонентів" - коротких ізольованих ділянок коду, які допомагають розробникам створювати складну логіку та UI. React взаємодіє з HTML через Virtual DOM. – копію реального DOM-дерева елементів сторінки. У копії всі елементи представлені як JavaScript. Ці елементи, разом із декларативним стилем програмування React та одностороннім зв'язуванням даних, спрощують та прискорюють розробку. Ключові особливості React:

– Декларативність. За допомогою React розробник визначає, як компоненти інтерфейсу виглядають у різних станах. Декларативний підхід скорочує код та робить його зрозумілим.

– Компонентний підхід. React базується на компонентах, це ще одна ключова особливість бібліотеки. Кожен компонент повертає частину інтерфейсу користувача зі своїм станом. Поєднуючи компоненти, програміст створює завершений інтерфейс веб-програми.

– JSX. Важливою особливістю React це використання JSX. Це розширення синтаксису JavaScript, яке зручно використовувати для опису інтерфейсу. JSX схожий на HTML, проте це все-таки JavaScript. Приклад JSX можна побачити на рисунку 1.7. JSX дозволяє писати JavaScript код за

допомогою готових компонентів, які практично повністю повторюють HTML. Це спрощує розробку.

```
const header = text ? <h1>{text}</h1> : null;

const vdom = (
  <div>
    {header}
    <Hello />
  </div>
);
```

Рисунок 1.7 – Синтаксис JSX

– Virtual DOM. Віртуальний DOM — це об'єкт, в якому зберігається інформація про стан інтерфейсу. При зміні стану, наприклад, після надсилання форми або натискання кнопки, React розраховує різницю між старим та новим станом. Після цього бібліотека малює новий стан. Використання віртуального DOM дає змогу бібліотеці ефективно оновлювати реальний DOM.

Об'єктна модель документа (DOM) визначає деревоподібну структуру HTML-документа, що надсилається клієнту сервером після відповідного запиту (Див. рисунок 1.8). DOM представляє веб-сторінку в об'єктно-орієнтованому форматі, щоб мови програмування могли взаємодіяти з нею.

Браузер регулярно перевіряє будь-які зміни в DOM, оновлюючи її належним чином. Зміна об'єктної моделі документа провокується безліччю факторів, наприклад введенням даних, HTTP-запитом, отриманням даних від API і так далі. Браузер оновлює DOM кожного разу, коли сторінка змінюється. Оскільки DOM сучасних сайтів величезні, оновлення займає багато часу, уповільнюючи загальну продуктивність веб-програми.

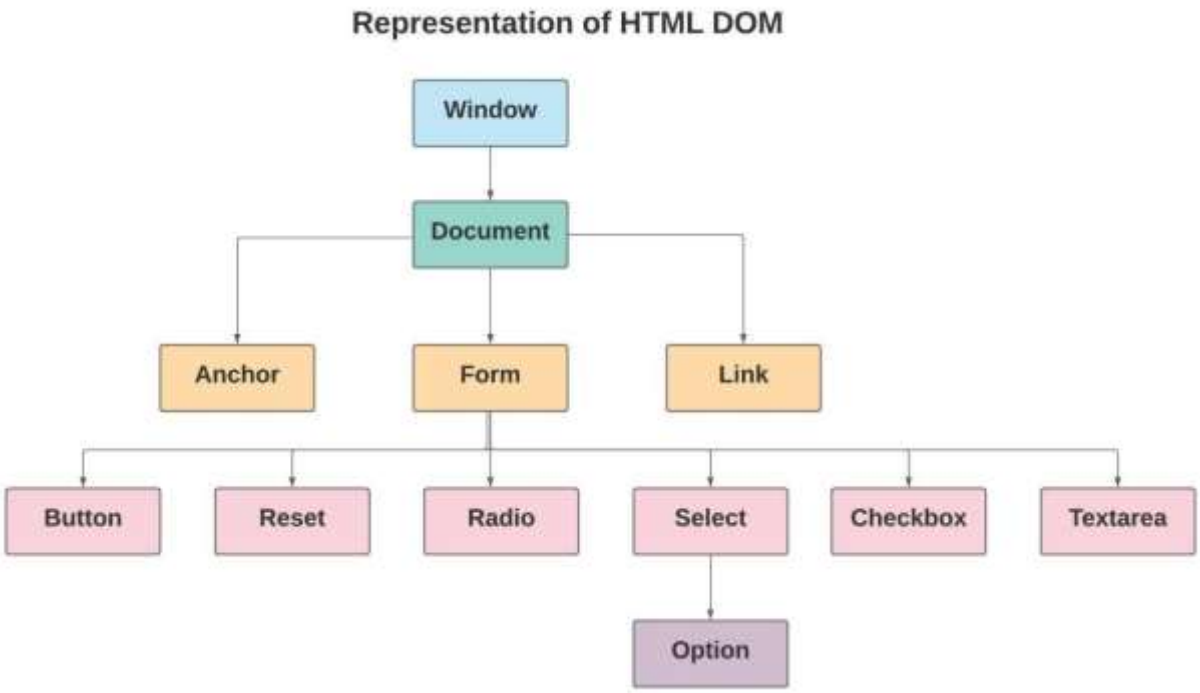

Рисунок 1.8 – DOM-дерево

Дана система має схожість з старою моделлю UI у Android програмах. Де також була система із загальним станом і оновленням його через окремі «безпечні» потоки виконнання програми, і при оновленні глобального стану UI елементи перестворювались [22].

Замість повільної та незручної взаємодії безпосередньо з реальною об'єктною моделлю документа, React взаємодіє з її полегшеною копією — віртуальної DOM, тому реальна DOM оновлюється лише після взаємодії з віртуальною DOM (Див. рисунок 1.9).

Оновлення всього DOM, щоб зробити веб-сторінку “реактивною” — вкрай неефективним, оскільки споживає занадто багато ресурсів. Тому, власне, при зміні веб-сторінки (наприклад, в результаті запиту або дії користувача) React оновлює віртуальний спеціальний DOM.

React зберігає дві версії віртуального DOM: оновлений віртуальний DOM та його резервну копію, створену до оновлення. Після оновлення React порівнює обидві версії між собою, щоб знайти змінені елементи, а потім оновлює частину реального DOM, що виключно змінилася.

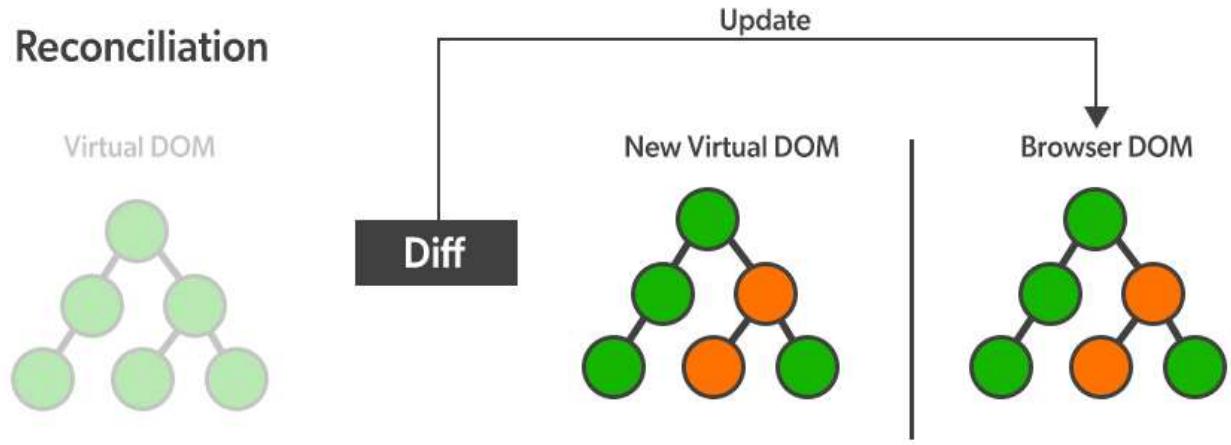

Рисунок 1.9 – Принцип роботи Virtual DOM

Подібний процес на перший погляд здається переускладненим і трудомістким, проте займає набагато менше часу, ніж оновлення реальної об'єктної моделі документа цілком, отже, він оптимізує роботу з DOM.

Віртуальна DOM, на відміну від реального DOM, займає мало місця та швидко оновлюється, тим самим підвищуючи продуктивність програми.

Віртуальна DOM дозволяє сторінці негайно отримувати відповіді від сервера та відображати оновлення. Наприклад, соціальна мережа Facebook застосовує технологію віртуального DOM для оновлення чатів та стрічок користувачів без перезавантаження сторінки.

1.3 Вимоги до програмної документації

Для даного проекту потрібно проста і зрозуміла документація, тому зсилаючись на попередні пункти було вибрано Swagger.

Swagger – це фреймворк для специфікації RESTful API. Його принадність полягає в тому, що він дає можливість не лише інтерактивно переглядати специфікацію, а й надсилати запити – так званий Swagger UI.

Також можна згенерувати безпосередньо клієнта або сервер специфікації API Swagger, для цього знадобиться Swagger Codegen.

Swagger має два підходи до написання документації:

1. Документація пишеться на підставі коду:

– Достатньо додати кілька залежностей у проект, додати конфігурацію і можна отримати потрібну документацію, хоч і не настільки описану.

– Код проекту ставатиме більш читабельним від анотацій та опису в них.

– Вся документація буде вписана в коді (всі контролери та моделі перетворюються на Java Swagger Code).

2. Документація пишеться окремо від коду:

– Цей підхід вимагає знати синтаксис Swagger Specification.

– Документація пишеться або у YAML/JSON файлі, або у редакторі Swagger Editor.

Оскільки Spring Framework має підтримку Swagger не потрібно буде робити усю документацію з нуля, а лише описати функції у коді програми за допомогою анотацій (Див. лістинг 1.1).

Лістинг 1.1 – Затосування анотацій при роботі із Swagger

```
@RestController
@Controller
@ApiOperation("Controller for test")
@RequestMapping("/v1/test")
@Transient
public class TestController {

    @ApiOperation("Function for test")
    @PutMapping(value = "/test-function/{testData}")
    public ResponseEntity<?> testFunction(
           @PathVariable String testData)
    throws DocsException {
        return ResponseEntityFactory.createOk();
    }
}
```

Код буде прочитаний бібліотекою Swagger і перетворений у зрозумілий UI варіант (Див. рисунок 1.10).

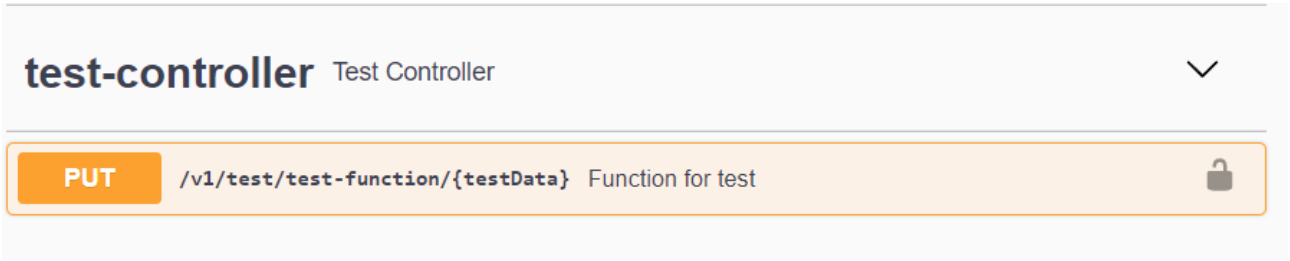

Рисунок 1.10 – UI варіант документації

Цей підхід значно спростить написання документації а також створить простий UI інтерфейс для тестування.

1.4 Стадії та етапи розробки

Загальна структура даного проекту побудована використовуючи багатошарову архітектуру [2]. Увесь підхід працює за принципом поділу відповідальності. ПЗ розділено на шари, що лежать один на одному, і кожен з них виконує певний обов'язок [3].

Архітектура ділить на наступні шари:

– шар представлення (Presentation Layer).Містить інтерфейс користувача і відповідає за забезпечення хорошого користувальницького досвіду;

– шар бізнес-логіки (Business Logic Layer).Містить бізнес-логіку програми. Він відокремлює UI/UX від обчислень, пов'язаних із бізнесом. Це дозволяє легко змінювати логіку залежно від постійно мінливих бізнес-вимог, ніяк не впливаючи на інші шари;

– шар передачі даних (Data Link Layer).Відповідає за взаємодію Космосу з постійними сховищами, такими як бази даних, та іншу обробку інформації, яка не пов'язана з бізнесом. Дані та елементи управління проходять через кожен шар у дизайні та передаються від одного до іншого. Ця система також підвищує рівень абстракції та певною мірою навіть стабільність ПЗ.

Переваги вибраної архітектури:

– простіша реалізація порівняно з іншими підходами;

– пропонує абстракцію завдяки розподілу відповідальності між рівнями;

– ізолювання захищає одні шари від інших змін;

– підвищує керованість програмного забезпечення з допомогою слабкої пов'язаності.

Недоліки вибраної архітектури:

– не пропонує великої масштабованості;

– програмне забезпечення, створене з таким підходом, матиме монолітну структуру, що ускладнює внесення модифікацій;

– дані повинні проходити по кожному шару, навіть якщо не потрібно передавати їх з певних шарів.

Перейдемо до тестування проекту тобто, до перевірки його працездатності на практиці.

1.5 Порядок тестування та прийому

Для тестування даного проекту можна використовувати документацію Swagger яка саме створена для комфортного тестування системи (Див. рисунок 1.11).

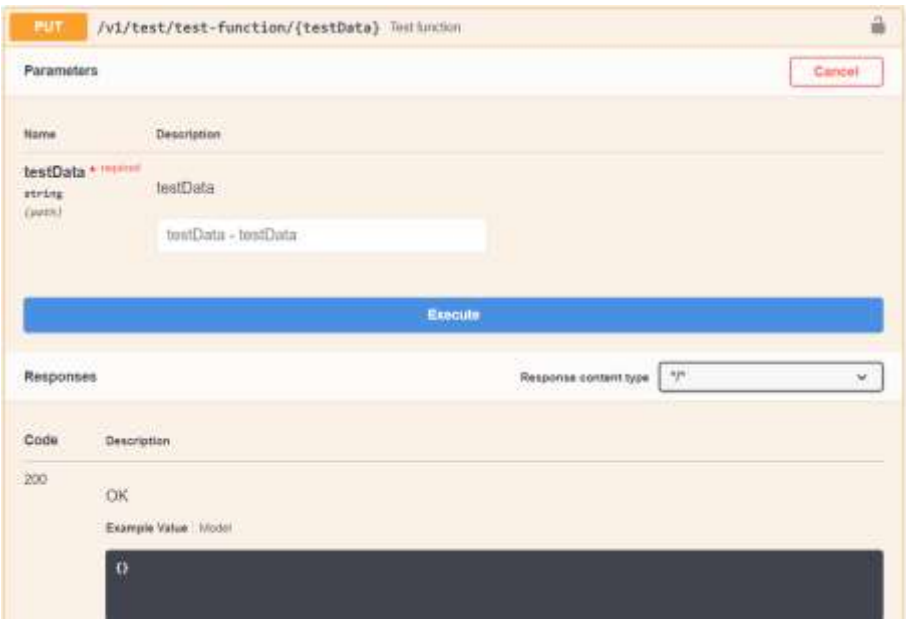

Рисунок 1.11 – Swagger інтерфейс для тестування виконання REST запитів

Swagger створює зручний інтерфейс для виконання REST запитів і водночас показує опис даних які він використовує. Також він попередньо перевіряє дані на правильність що дозволяє швидко знайти помилки про тестуванні [27].

Також використана в даній роботі мова програмування Java і фреймворк Spring надають можливість написання тест-скриптів. Ці тест-скріпти можна використовувати як і при виконанні програми так і підчас запуску. Також система збірки Maven дозволяє налаштувати виконуваний файл .jar так щоб при не проходжені усіх або певних тестів серверний застосунок не змін запуститись. Це створено для того щоб при некоректному запуску серверного застосунку адміністратор не зміг використовувати це програмне забезпечення [28].

1.6 Складність пошукової оптимізації

Пошукова оптимізація (SEO) є вкрай важливою, адже веб-додатку потрібно отримувати трафік та залучати нових клієнтів. Для кращого розуміння складності пошукової оптимізації програми на React потрібно зрозуміти, як Google індексує веб-сторінки.

Веб-сторінки індексуються спеціальними пошуковими роботами Google (Див. рисунок 1.12).

Ці роботи сканують вміст веб-сторінки і зберігають інформацію в індексі Google. Коли користувач запитує певну інформацію, Google перевіряє дані, що зберігаються в індексі, щоб надати найбільш релевантні джерела інформації.

Боти Google можуть легко індексувати HTML-сторінки. Проте з JavaScript-сторінками все вже не так гладко, адже динамічні веб-програми, у тому числі створені на React, проходять ускладнену процедуру індексації, на відміну від статичних веб-сторінок.

Таким чином, сторінка програми React може бути проіндексована неправильно або індексування може зайняти занадто багато часу. Що один випадок, що інший – обидва змусять ботів Google залишити сторінку сайту.

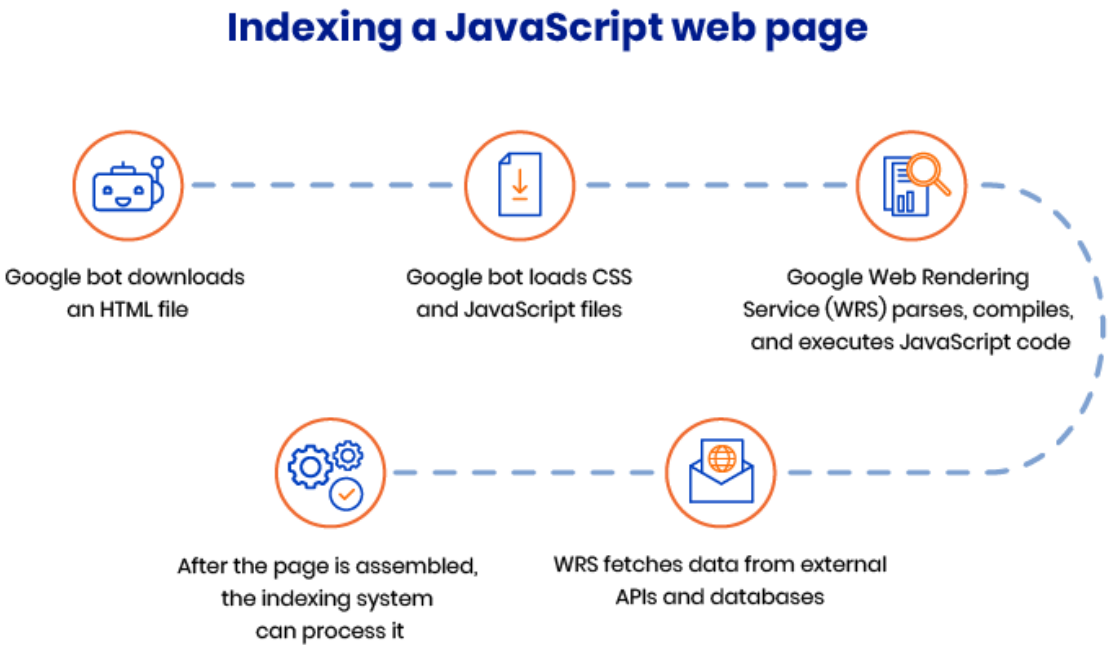

Рисунок 1.12 – Індексація JavaScript сторінок

Існує кілька підходів, які роблять React-сайт більш SEO-дружнім, наприклад, попередній рендеринг або рендеринг на стороні сервера.

1.7 Висновки до першого розділу

У даному розділі розглянуто вимоги до проекту, вибрано технології розробки для клієнтської та серверної частини, поставлено вимоги до програмної документації і складено план та етапи розробки.

Завдяки правильно підібраним технологіям розробки таким як Spring та ReactJs даний проект повинен бути легко та швидко реалізованим а також повинен мати хороші швидкісні і стресостійкі характеристики. Оскільки дані

фреймфорки створені для швидкої та якісної розробки кліент-серверних рішень.

Також правильно підібрана багатошарова архітектура серверного застосунку дозволяє швидко реалізовувати нові функцій та легко покращувати реалізацію. Документація Swagger є одним з найкращих рішень документації а також дозволить легко тестувати програму. Система збірки Maven дозволить легко контролювати залежності на біблотеки сторонніх розробників. Бібліотека JPA дасть змогу не залежати від конкретної реалізації бази данних.

Важливою роллю у розробці є безпека, тому для даного проекту було вибрано мову програмування Java для розробки серверної частини. Дану мову в основному використовують для банківських та Fintatech серверних застосунків.

Також вибір мови програмування Java дозвонить запускат серверний застосунок на практично будь-якій операційній системі. Вибір способів авторизації таких як Bearer дозволяє легко та безпечно авторизовувати користувача, а також не вимагати авторизацію дость довгий час без ніяких ризиків. Також дозволяє адміністраторам визначити період обов’язкової авторизації.

РОЗДІЛ 2. РЕАЛІЗАЦІЯ ПРОГРАМНОГО РІШЕННЯ

Опишемо результати реалізації проекту, використані технології, архітектурні прийоми. У даному розділі розглянуто:

– порядок виконання роботи;
– загальну структуру серверної програми;
– загальну структуру клієнтської програми;
– заходи щодо безпеки данних;
– опис реалізації функціональних характеристик.

2.1 Розробка загальної структури сервера

2.1.1 Структура

Структура проекту буде складатись із трьох шарів (Див. рисунок 2.1):

– ядро;
– контролери;
– репозиторії.

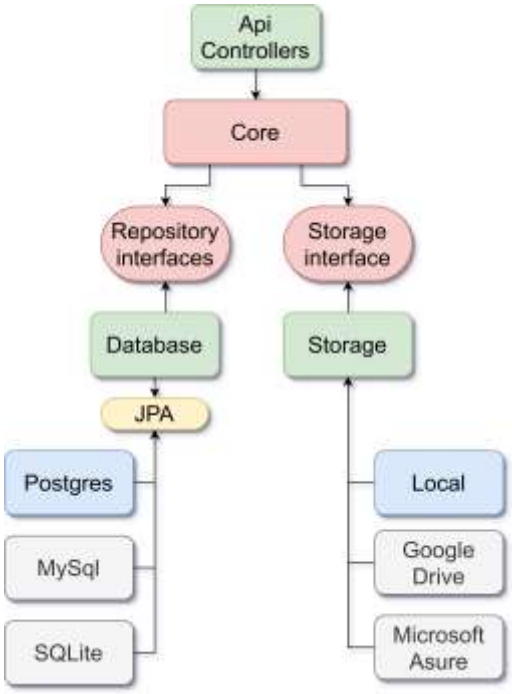

Рисунок 2.1 – Загальна структура класів проекту

Даний підхід дозволяє легко маштабувати та тестувати систему. А також дозволяє додавати будь які нові модулі, технології та підсистеми.

2.1.2 Ядро програмного рішення

Розробка даного проекту буде виконуватись починаючи із ядра програми. Першочергово будуть створені моделі даних, далі інтерфейси репозиторіїв для отримання, зберігання, видалення або зміни даних. Дані репозиторії не будуть мати конкретної реалізації у ядрі, їхню реалізацію буде містити інший шар а самі об’єкти у залежності підставить Spring Framework використовуючи механізм Dependency Injection.

Впровадження залежності (Dependency injection) – це процес надання зовнішньої залежності програмному компоненту. Є специфічною формою «інверсії управління» коли застосовується до управління залежностями. У повній відповідності до принципу єдиної відповідальності об'єкт турбується про побудову необхідних йому залежностей зовнішньому, спеціально призначеному для цього загальному механізму [7].

Далі будуть створені сервіси які будуть реалізацією функціональних характеристик проекту. Сервіси використовуватимуть моделі, інтерфейси репозиторіїв а також інші сервіси для реалізації функціональних характеристик проекту. Ядро повинне залишатися незалежним шаром тому не повинно зсилатись на інші шари програми.

2.1.3 Контролери

Наступним шаром створено шар контролерів. Даний шар буде відповідати за створення RESTful API і опрацювання API запитів.

REST (Representational state transfer) – це стиль архітектури програмного забезпечення для розподілених систем, таких як World Wide Web, який

зазвичай використовується для побудови веб-служб. Термін REST був запроваджений у 2000 році Роєм Філдінгом, одним із авторів HTTP-протоколу. Системи, що підтримують REST, називаються RESTful-системами [11].

У випадку REST є дуже простим інтерфейсом управління інформацією без використання якихось додаткових внутрішніх прошарків. Кожна одиниця інформації однозначно визначається глобальним ідентифікатором, таким, як URL. Кожна URL, у свою чергу, має строго заданий формат.

Як відбувається управління інформацією сервісу – це цілком грунтується на протоколі передачі. Найбільш поширений протокол звичайно ж HTTP.
Для HTTP дія над даними визначається за допомогою методів:

– GET (отримати);
– PUT (додати, замінити);
– POST (додати, змінити, видалити);
– DELETE (видалити);
– PATCH (оновити);
– OPTIONS (список підтримуваних операцій);

Таким чином, дії CRUD (Create-Read-Updtae-Delete) можуть виконуватися як з усіма чотирма методами, так і лише за допомогою GET та POST.

Формат передачі данних у API буде JSON. JSON (JavaScript Object Notation) – це текстовий формат обміну даними, що базується на JavaScript. Але при цьому формат незалежний від JS може використовуватися в будь-якій мові програмування.

Також в такому шарі знаходяться власні сервіси і структури запитів і відповідей. У структурах запитів знаходяться перевірки даних запитів, ці перевірки побудовані використовуючи Spring Validation бібліотеку.

2.1.4 Реалізація репозиторіїв

Останнім шаром буде створено реалізацію репозиторіїв ядра. У даному шарі буде використано JPA Hibernate для роботи з базою даних і з’єднаннями

до неї.

Hibernate – це бібліотека для роботи об'єктно-реляційного відображення баз данних на мові програмування Java. Бібліотека є реалізацією специфікації JPA і є безплатною у використанні [8].

JPA (Java Persistence API) це специфікація Java EE і Java SE, що описує систему управління збереженням Java об'єктів у таблиці реляційних баз даних у зручному вигляді (Див. рисунок 2.2). Сама Java не містить реалізації JPA, однак існує багато реалізацій даної специфікації від різних компаній (відкритих і ні)].

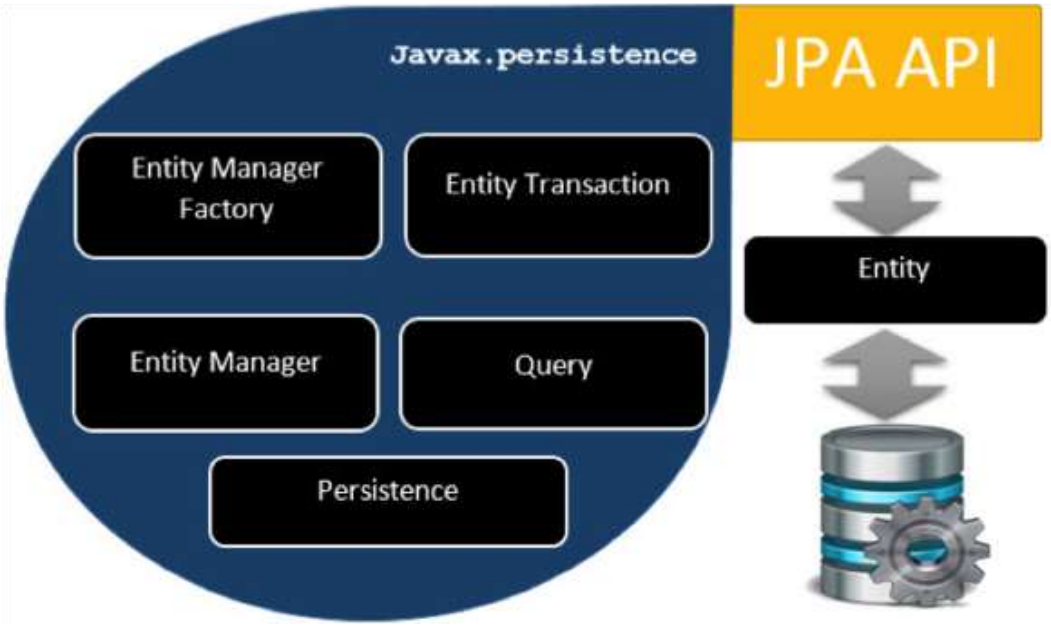

Рисунок 2.2 – Структура JPA

Hibernate зв’язує класи і типи даних мови програмування Java з таблицями і типами даних бази даних. Також спрощує побудову запитів та отримання даних, автоматизує генерацію SQL-запитів та звільняє розробника від ручної обробки результуючого набору даних та перетворення об'єктів, максимально полегшуючи перенесення програми на будь-які бази даних SQL.

2.2 Розробка структури сайту

Оскільки сайт буде складатися з окремих сторінок між якими буде

здійснюватись перехід через бібліотеку react-router-dom, яка допомагає розробляти SPA додаток, оновлюючи не всю сторінку, а лише потріібні елементи, не перезавантажуючи її.

React Router це стандартна бібліотека маршрутизації у React. Він зберігає інтерфейс програми, синхронізованим з URL на браузері. React Router дозволяє маршрутизувати "потік даних" у вашому додатку зрозумілим способом. Він подібний до твердження, якщо у вас є даний URL, він буде подібний до цього Route.

Для розробки програми React потрібно написати багато компонентів і route-сторінок, але потрібний лише один файл для обслуговування цих сторінок і компонентів це index.html (Див. рисунок 2.3).

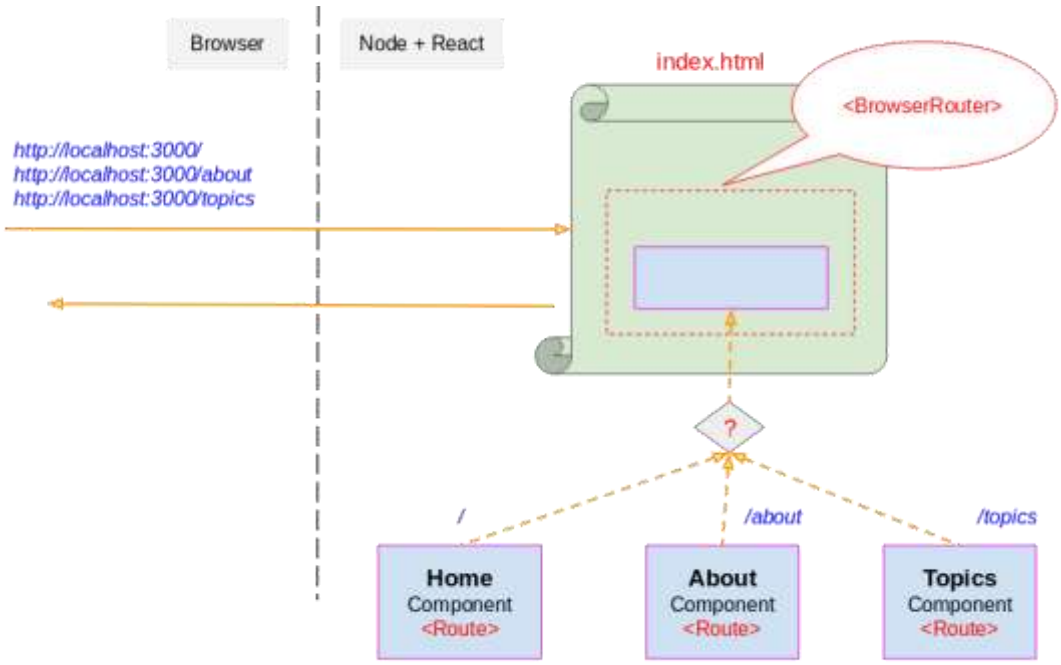

Рисунок 2.3 – Принцип роботи React Route

Список сторінок сайту:

– «/» – домашня сторінка сайту;
– «/category/:id» – список документів вибраної категорії;
– «/document/:id» – сторінка з виводом документа;
– «/login» – сторінка для входу в адмін-панель;

– «/search/:name» – сторінка пошуку документів по імені;

– «/management/document» – сторінка управління документами;

– «/management/document/create» – сторінка створення документа;

– «/management/document/update/:id» – сторінка оновлення документа;

– «/management/category» – сторінка управління категоріями;

– «/management/category/create» – сторінка створення категорій;

– «/management/category/update/:id» – сторінка оновлення категорій.

Перейдемо до питання реалізації політики безпеки доступу до даних на сервері розроблюваного програмного рішення.

2.3 Реалізації політики безпеки доступу до даних на сервері

У даному проекті будь які зміни стану серверу вимагають авторизації користувача який ініціює зміну. Авторизовані користувачі мають доступ до зміни, створення та редагування документів та категорій документів, тому для авторизації користувача вимагається JWT Token. У даному проекті використовується JWT token як токен авторизації OAuth 2.0 [25].

JSON Web Token (JWT) - це відкритий стандарт для створення токенів доступу на основі JSON. Він вважається одним із безпечних способів передачі інформації між двома учасниками. Для створення необхідно визначити заголовок (header) із загальною інформацією по токену, корисні дані (payload), такі як id користувача, його роль тощо. та підписи (signature). В основному, JWT використовується для авторизації у клієнт-серверних застосунках. Токени генеруються сервером, підписуються ключем і надсилаються клієнту, який надалі надсилає цей токен для підтвердження своєї особи.

Токен JWT складається з трьох частин:

– заголовка (header);

– корисного навантаження (payload);

– підпису або даних шифрування.

Перші два елементи це JSON об'єкти певної структури. Третій елемент обчислюється на основі перших і залежить від обраного алгоритму (у разі використання непідписаного JWT може бути опущений). Токени можуть бути перекодовані в компактне уявлення (JWS/JWE Compact Serialization): до заголовка та корисного навантаження застосовується алгоритм кодування Base64-URL, після чого додається підпис і всі три елементи розділяються точками «.».

Будь які запити редагування без JWT токена будуть ігноровані. Також авторизовані запити дозволяють отримувати список записів документі разом із документами які знаходяться у архівованому стані.

Для більшої безпеки пошук по частині імені документу обмежений буквами українського та латинського алфавіту а також символами «(», «)», «.».

2.4 Опис структури запитів

2.4.1 Авторизовані запити

Аутентифікація Bearer (також називається автентифікацією за маркером) - це схема автентифікації HTTP, яка включає маркери безпеки, які називаються маркерами носія. Маркер носія - це рядок, який зазвичай генерується сервером у відповідь на запит на вхід. Клієнт повинен надіслати цей маркер у заголовку авторизації під час виконання запитів до захищених ресурсів (Див. лістинг 2.1):

Лістинг 2.1 – Приклад заголовку авторизації

```
Authorization: Bearer <token>
```

Схема автентифікації Bearer спочатку була створена як частина OAuth2.0, але іноді також використовується як сама по собі. Подібно до базової автентифікації, автентифікацію Bearer слід використовувати лише через HTTPS (SSL, TLS). Secure Sockets Layer (SSL) був найпопулярнішим криптографічним

протоколом для інтернет-комунікацій, поки в 1999 на зміну йому не прийшов TLS (Transport Layer Security). SSL забезпечує безпечний канал зв'язку між двома машинами або пристроями, що працюють через Інтернет чи внутрішню мережу. Одним із найпоширеніших прикладів є використання SSL для безпечного зв'язку між веб-браузером та веб-сервером. Це перетворює адресу веб-сайту з HTTP на HTTPS, а літера "S" означає "безпечний" (secure).

2.4.2 Запити авторизації

В лістингу 2.2 наведено зразок запиту (відповіді) авторизації

Лістинг 2.2 – Запит авторизації

```
// Шлях: /v1/auth/login
// Метод: POST
// Тіло запиту:
{
  "email": "Пошта користувача",
  "password": "Пароль користувача"
}
// Відповідь:
{
  "userId": "Id користувача",
  "accessToken": "JWT токен для авторизації",
  "refreshToken": "JWT токен для оновлення accessToken",
  "expiration": "Дата закінчення дії токена"
}
```

В лістингу 2.3 наведено зразок запиту (відповіді) оновлення токена

Лістинг 2.3 – Запит на оновлення токена

```
// Шлях: /v1/auth/refresh
// Метод: POST
// Тіло запиту:
{
  "token": " JWT токен для оновлення accessToken"
}
// Відповідь:
{
  "userId": "Id користувача",
  "accessToken": "JWT токен для авторизації",
```

```
  "refreshToken": "JWT токен для оновлення accessToken",
  "expiration": "Дата закінчення дії токена"
}
```

У даних запитах показано формати а також типи данних для авторизації користувача.

2.4.3 Запити реєстрації

В лістингу 2.4 наведено зразок запиту (відповіді) початку реєстрації

Лістинг 2.4 – Запит на початок реєстрації користувачів

```
//Шлях: /v1/registration/start
//Метод: POST
//Тіло запиту:
{
  "roleName": "Назва ролі",
  "usersEmails": [
    "Пошта нового користувача"
  ]
}
```

В лістингу 2.5 наведено зразок запиту (відповіді) реєстрації користувача

Лістинг 2.5 – Запит на реєстрацію користувача

```
//Шлях: /v1/registration
//Метод: POST
//Тіло запиту:
{
  "code": "Код авторизації",
  "password": "Пароль",
  "username": "Ім’я користувача"
}
//Відповідь:
{
  “Id”:”ID користувача”,
  “username”:”Ім’я користувача”,
  “email”:”Пошта користувача”,
  “deleteTimestamp”:”Час видалення користувача”,
  “validTokenTimestamp”:”Час активності токена”
}
```

У даних запитах показано формати а також типи данних для реєстрації користувача.

2.4.4 Запити перегляду документів

В лістингу 2.6 наведено зразок запиту (відповіді) знаходження документа по імені.

Лістинг 2.6 – Запит на знаходження документа по імені

```
// Шлях: /v1/docs/find/{Частина імені документу}
// Метод: GET
// Відповідь:
[
 {
  “id”:”Id документу”,
  “sectionId”:”Id секції”,
  “name”:”Ім’я документу”,
  “createTime”:”Дата створення”,
  “status”:”Статус документа”,
  “fileName”:”Ім’я файлу”
 }
]
```

В лістингу 2.7 наведено зразок запиту (відповіді) знаходження документа по ID.

Лістинг 2.7 – Запит на знаходження документа по ID

```
// Шлях: /v1/docs/get/{ID документу}
// Метод: GET
// Відповідь:
{
  “id”:”Id документу”,
  “sectionId”:”Id секції”,
  “name”:”Ім’я документу”,
  “createTime”:”Дата створення”,
  “status”:”Статус документа”,
  “fileName”:”Ім’я файлу”
}
```

В даних запитах показано формати а також типи данних для перегляду документів.

2.4.5 Запити управління документами

В лістингу 2.8 наведено зразок запиту (відповіді) створення документу.

Лістинг 2.8 – Запит на створення документа

```
// Шлях: /v1/docs
// Метод: POST
// Тіло запиту:
{
  "createTime": "Дата створення",
  "name": "Імя документу",
  "sectionId": “ID секції”,
  "status": "Статус документу"
}
```

В лістингу 2.9 наведено зразок запиту (відповіді) вигрузки файлу.

Лістинг 2.9 – Запит на завантаження файла

```
// Шлях: /v1/docs/{ID документу}
// Метод: PUT
// Тіло запиту: Файл
```

В лістингу 2.10 наведено зразок запиту (відповіді) видалення документу.

Лістинг 2.10 – Запит на видалення документа

```
// Шлях: /v1/docs/{ID документу}
// Метод: DELETE
```

В лістингу 2.11 наведено зразок запиту (відповіді) оновлення документу.

Лістинг 2.11 – Запит на оновлення документа

```
// Шлях: /v1/docs/{ID документу}
// Метод: PATCH
// Тіло запиту:
```

```
{
  "createTime": "Дата створення",
  "name": "Імя документу",
  "sectionId": "ID секції",
  "status": "Статус документу"
}
```

В даних запитах показано формати а також типи данних для управління документами.

2.5 Створення інтерфейсу користувача

Інтерфейс користувача (UI) – це точка, на якій люди взаємодіють з комп'ютером, веб-сайтом або програмою. Метою ефективного інтерфейсу є зробити роботу користувача простою та інтуїтивно зрозумілою, вимагаючи мінімальних зусиль з боку користувача для отримання максимального бажаного результату.

Інтерфейс користувача створюється у вигляді шарів взаємодії, які звертаються до органів чуття людини (зір, дотик, слух тощо). Вони включають обидва пристрої введення, такі як клавіатура, миша, трекпад, мікрофон, сенсорний екран, сканер відбитків пальців, електронне перо та камера, а також пристрої виведення, такі як монітори, колонки та принтери. Пристрої, які взаємодіють з кількома органами чуття, називаються «мультимедійними інтерфейсами користувача». Наприклад, повсякденний інтерфейс використовує комбінацію тактильного введення (клавіатура та миша) та візуального та слухового виходу (монітор і динаміки) [20].

Для спрощення написання сайту і його UI елементів буде використовуватись CSS-бібліотека React Bootstrap 5. Вона допомагає швидко будувати зручний, адаптивний і кросплатформерний інтерфейс.

Bootstrap – це вільний набір інструментів для створення сайтів та веб-застосунків (Див. рисунок 2.4). Включає HTML- і CSS-шаблони оформлення

для типографіки, веб-форм, кнопок, міток, блоків навігації та інших компонентів веб-інтерфейсу, включаючи JavaScript-розширення.

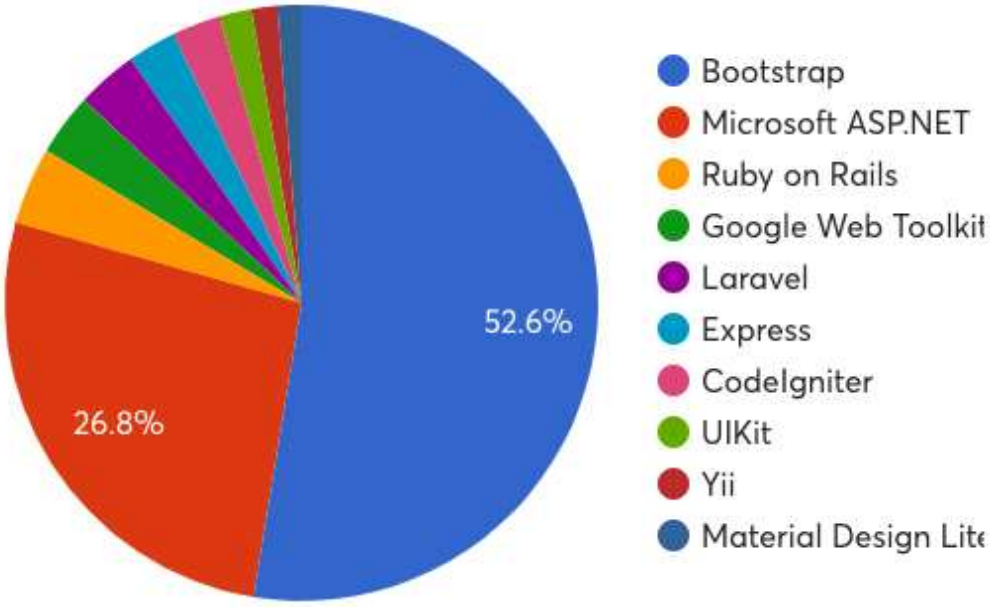

Рисунок 2.4 – Популярність Bootstrap

Особливості Bootstrap:

– бібліотека Bootstrap надає три готові до використання файли і може бути використаний для створення прототипів чи додатків, реалізований як Reset і для верстки за допомогою Grid;

– у Bootstrap використовується власна бібліотека SVG з відкритим вихідним кодом, спеціально розроблена для наших компонентів і документації;

– іконки призначені для найкращої роботи з компонентами Bootstrap, а також вони чудово працюватимуть у будь-якому проекті. Це SVG-файли, тому вони швидко та легко масштабуються, можуть бути підключені різними способами та стилізовані за допомогою CSS [1, 19].

Переваги Bootstrap перед аналогічними біліотеками:

– висока швидкість створення якісної адаптивної верстки;

– наявність зрозумілтх назв класів;

– наявність великої кількості рішень для стовреннz дизайну проекту;

– гнучкість для налаштування стилів під проект;

– кросбраузерність та кросплатформність;

– наявність великої кількості готових добре продуманих компонентів;

– можливість налаштування під свій проект;

– однорідність дизайну та його узгодженість між різними компонентами.

React Bootstrap легко використовувати в JSX, що покращує загальну зрозумілість коду. Щоб створити список не потрібно вказувати клас CSS елемента (Див. рисунок 2.5), а можна просто внести в JSX синтаксис React Bootstrap (Див. рисунок 2.6).

Методи і події з використанням jQuery виконуються імперативно шляхом прямого маніпулювання DOM. На відміну від цього, React використовує оновлення стану для оновлення віртуального DOM. прикладів того, як компоненти React Bootstrap відрізняються від Bootstrap [14].

```
2
3   <ul class="list-group">
4     <li class="list-group-item">An item</li>
5     <li class="list-group-item">A second item</li>
6     <li class="list-group-item">A third item</li>
7     <li class="list-group-item">A fourth item</li>
8     <li class="list-group-item">And a fifth one</li>
9   </ul>
10
```

Рисунок 2.5 – Bootstrap синтаксис

```
2
3   <ListGroup>
4     <ListGroup.Item>Cras justo odio</ListGroup.Item>
5     <ListGroup.Item>Dapibus ac facilisis in</ListGroup.Item>
6     <ListGroup.Item>Morbi leo risus</ListGroup.Item>
7     <ListGroup.Item>Porta ac consectetur ac</ListGroup.Item>
8     <ListGroup.Item>Vestibulum at eros</ListGroup.Item>
9   </ListGroup>
10
```

Рисунок 2.6 – React Bootstrap синтаксис

Оскільки React Bootstrap побудований з використанням React Javascript, стан може бути переданий у компонентах React Bootstrap як опору (Див. рисунок 2.7).

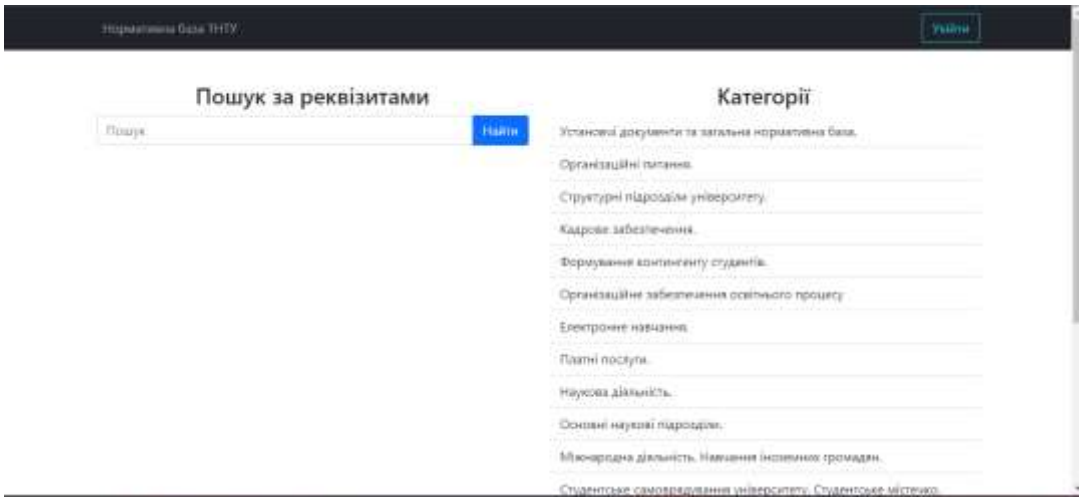

Рисунок 2.7 – Головна сторінка сайту

Це також спрощує керування станом, оскільки оновлення виконуються за допомогою стану React замість прямого керування станом DOM. Це також дає велику гнучкість при створенні складніших компонентів і окремих сторінок сайту.

2.6 Відправлення запитів на сервер

Головне завдання клієнта це прийом і відправлення HTTP-запитів на сервер. У JavaScript існує нативний метод для опрацювання HTTP-запитів Fetch API.

Fetch API надає інтерфейс для отримання ресурсів із браузера. Відповідно до документації, Fetch дає загальне визначення об'єктів для запиту та відповіді. Цей API також використовує проміси та надає глобальний метод fetch(). Цей метод вимагає як необхідний аргумент лише шлях до потрібного ресурсу. У відповідь він повертає проміс, який дозволяє відповісти на цей запит, незалежно від того, успішний він був чи ні [18].

Приклад запиту для створення документу (лістинг 2.12):

Лістинг 2.12 – Приклад featch запиту

```
fetch(serverUrl + "v1/docs", {
        method: "POST",
        body: JSON.stringify({
          createTime: document.getElementById("date").value +
"T15:15:46.001Z",
          name: document.getElementById("name").value,
          sectionId:
document.getElementById("category").options[indexCategory].id,
         status:
document.getElementById("status").options[indexStatus].value,
        }),
        headers: {
          Authorization: "Bearer " +
localStorage.getItem("accsess_token"),
          accept: "*/*",
          "Content-Type": "application/json",
        },
      })
        .then((res) => res.json())
        .then((result) => {
          fetch(serverUrl + "v1/docs/" + result.data.id, {
            method: "PUT",
            body: formData,
            headers: {
              Authorization: "Bearer " +
localStorage.getItem("accsess_token"),
              accept: "*/*",
            },
          }).then(function (res) {
            if (res.status !== 200) {
              alert(res.status);
            } else {
              document.getElementById("name").value = "";
            }
          });
        });
```

Response надає кілька методів, що базуються на промісах, для доступу до тіла відповіді в різних форматах:

– response.text() читає відповідь та повертає як звичайний текст;

– response.json() декодує відповідь у форматі JSON;

– response.formData() повертає відповідь як об'єкт FormData;

– response.blob() повертає об'єкт як Blob (бінарні дані з типом).

Перейдемо до особливостей відображення файлів різних форматів на веб-сторінці.

2.7 Особливості відображення PDF, DOC і DOCX файлів на сторінці

HTML, також відомий як мова гіпертекстової розмітки, є найпоширенішим форматом для завантаження вмісту на веб-сайт. Зазвичай, якщо розмістити на веб-сторінці довгий PDF-файл або кілька PDF-файлів одночасно, найкращим рішенням є вбудовування PDF-файлу в HTML-файл, який не тільки легко опублікувати, але також може містити всі важливі дані. Щоб вбудувати PDF-файл у HTML-файл необхідно перетворити PDF-файл на HTML-файл, який можна легко додати до іншого HTML-файлу. Для Windows і Mac OS X є безліч конвертерів для вбудовування PDF у веб-сторінку, і більшість з них працюють за одним і тим же принципом.

Відображення всіх файлів на сторонці реалізується в PDF, тому що жоден браузер в даний час не має коду, необхідного для візуалізації документів Word, і в даний час не існує бібліотек на стороні клієнта для їхнього рендерингу. Тому файли DOC і DOCX для відобоження конвертуютья в PDF, але для скачування доступні в тому форматі в якому вони були завантаженні спочатку. Завантажувати на сервер можлтво 3 типа файлів: PDF, DOCX, DOC.

Для стабільної роботи і відображення PDF файлів на сторонці використовується бібліотека react-pdf, яка відображає файли стабільно на всіх пристроях навідміну від нативних тегів HTML iframe, embed і object, що на деяких браузерах і при об’ємному файлі можуть не відображатись на сторінці. React-pdf експортує набір примітивів React, які дозволяють дуже легко відображати речі у вашому документі. Він також має API для їх стилізації, використовуючи властивості CSS і макет Flexbox.

React-pdf дозволяє відображати документ у двох різних середовищах: веб та сервер. Процес, по суті, той самий, але відповідає потребам кожного середовища.

2.8 Система кешування

У даному проекті використовується кеш система для файлів к форматі .doc і .docx. Під час розробки серверного застосунку було виявлено що пряме конвертування форматів .doc і .docx займає дуже багато часу, тому було вирішено створити сисему котра буде зберігати конвертовані файли певний час, що дозволить зменшити навантащення на сервер. Також дана сисема повинна коректно працювати з багатьма потоками і легко настроюватись. Тому для виконання даних умов було використано бібліотеку багатопоточних колекцій у мові програмування Java java.util.concurrent, а саме клас ConcurrentHashMap.

Класи java.util.concurrent це класи створенні для вирішення проблем багатопотокового програмування. Дані класи спрощують роботу з кодом який може вконуватись у багатьох потоках, а також тут є засоби для синхронізації, безпечного блокування та використання потоків. У тому числі тут є багатопотокові реалізації колекцій такі як ConcurrentHashMap, ConcurrentLinkedDeque, ConcurrentSkipListMap, ConcurrentBag (Див. рисунок 2.8).

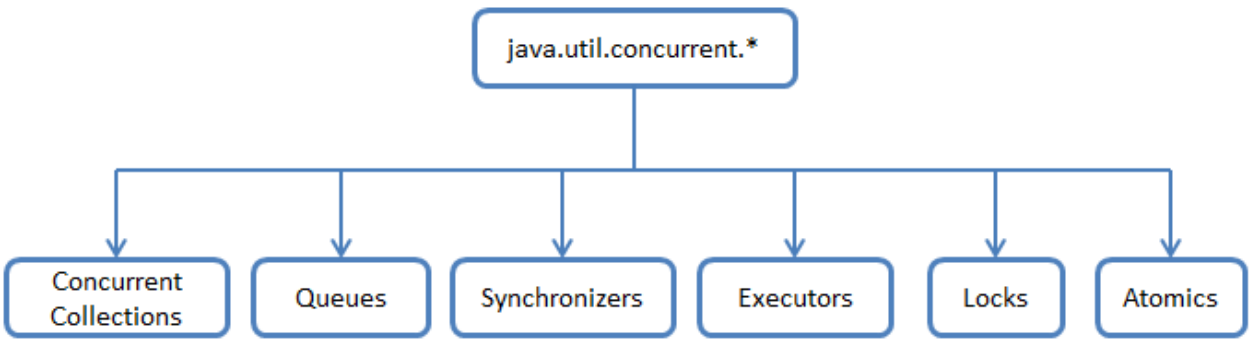

Рисунок 2.8 – Concurrent collections

ConcurrentHashMap спеціально створений для використання у багатьох потоках. Для конвертування фалів формату .doc і .docx використовується бібліотека xdocreport, оскільки бібліотеки які на пряму конвертують формати .doc і .docx у .pdf є платними, було прийняте рішення використовувати дану бібліотеку. Бібліотека xdocreport не конвертує файли на пряму, вона це робить використовуючи OpenSource проекти такі як OpenOffice, LibreOffice та інші, запускаючи їх у HeadLess режимі, що дозволяє конверкувати майже будь які формати файлів [10].

HeadLess режим це режим запуску прикладної програми програми без використання графічної частини програми. Код який запускає програму у HeadLess режимі не використовує графічну оболоку тому це дозволяє використовувати даний режм практично на будь яких операційних системах. Спілкування з запущеною програмою може відбуватись через різні види API.

Логіка кешування така: Користувач робить запит на .pdf формат файлу, серверний застосунок конвертує файл і зберігає його у оперативній пам’яті, створюючи відкладене завдання на видалення цього файлу із системи на час налаштований у системних конфігураціях, і повертає користувачу створений файл. Далі якщо будь-який користувач і ще раз зробить запит на той самий файл – конвертація не відбуватиметься а просто повернеться збережений файл. На той випадок якщо файл, який збережений, буде проредагований адміністратором, даний файл буде видалено з кешу.

Даний підхід дозволяє не навантажуючи сервер надмірними конвертаціями віконувати запити швидко та безпечно, а система віддаленого видалення не створить проблем для серверного застосунку у майбутньому.

2.9 Тестування програми

Тестування проходило з використанням документації Swagger UI а також із використанням програми Postman.

Postman (Див. рисунок 2.9) – це інструмент для роботи з API, який дозволяє тестувальнику надсилати запити до сервісів та працювати з їхніми відповідями. З його допомогою можна протестувати бекенд та переконатися, що він коректно працює.

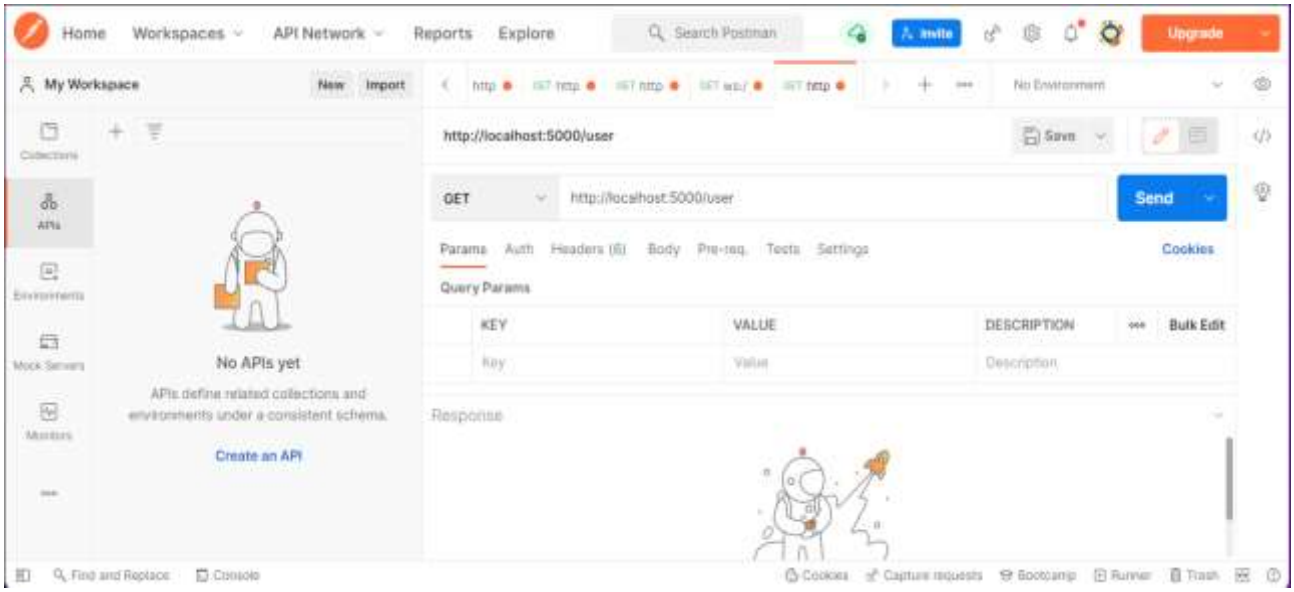

Рисунок 2.9 – Інтерфейс програмного інструмента Postman

Postman має такі переваги:

– інтуїтивно-зрозумілий і простий у використанні, не вимагає складного налаштування або знання мов програмування;

– безкоштовний;

– підтримує різні API (REST, SOAP, GraphQL);

– розширюється під будь-які потреби за допомогою Postman API;

– легко інтегрується в CI/CD за допомогою Newman – консольної утиліти для запуску тестів;

– запускається на будь-яких ОС;

– легко експортує та імпортує дані;

– підтримує ручне та автоматизоване тестування.

Використовуючи Swagger UI було протестовано базові сценарії використання API. Swagger UI має зручний інтерфейс для таких завдань.

Можна побачити що на UI елементах є коментарі та підказки задекларовані у програмному коді а також поля для вводу даних.

Дані у цих полях проходять перевірку на правильність ще до надсилання на сервер і якщо вони мають не правильний формат Swagger UI покаже цю помилку (Див. рисунок 2.10).

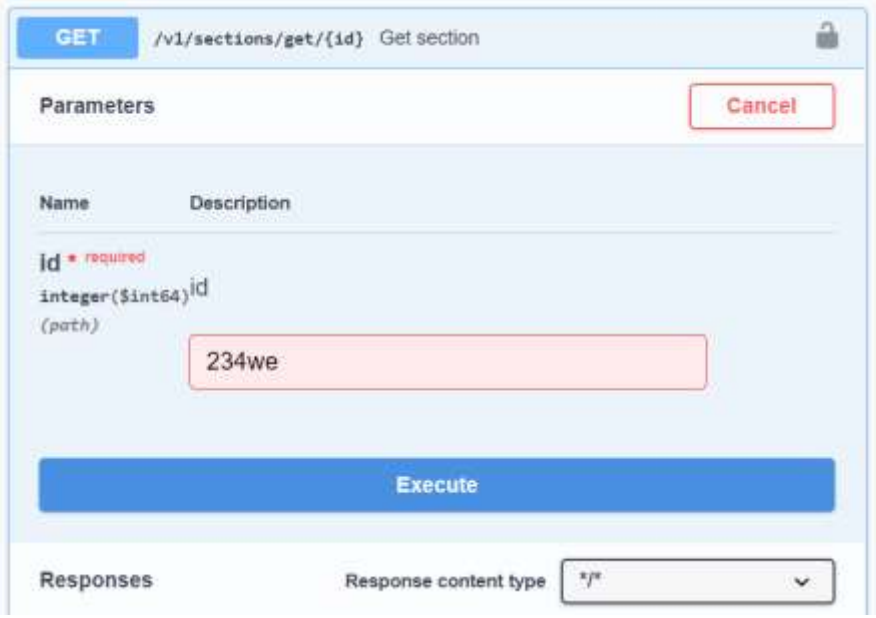

Рисунок 2.10 – Swagger UI

Це дозволяє краще тестувати систему та дозволяє зрозуміти у які дані потрібно надавати і у якому форматі. Також це дозволяє завжди мати приклад запиту який буде працювати.

2.10 Висновки до другого розділу

В результаті практичої частиги було описано сам процес створення клієнтської і серверної частини додатка, технології розробки, підходи до написання колу і взаємодію клієнта з сервером через RESTful API.

Розглянуто особливості розробки сервера на мові програмування Java з використанням фраємворку Spring:

– написання запитів та їх опрацювання;

– робота з JWT токенами;

– робота з базою даних.

Проаналізовано методи створення клієнтської частини на основі фраємворку React такі як:

– надсилання запитів на сервер та їх обробка на клієнті;

– маршрутизаця сайту;

– верстка сайту за допомогою бібліоткеки react-bootstrap.

Тестування розробленого програмного рішення показало його високу продуктивність в практичному використанні.

РОЗДІЛ 3. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

3.1 Характеристика скарг працівників та посадових осіб, які працюють більше половини робочого дня за комп’ютером.

Надійність системи «людина – комп'ютер» значною мірою визначається функціональним станом людини. Психофізіологічні та емоційні перенапруження, втома людини-оператора можуть призвести в комп'ютеризованих системах керування до помилок і як наслідок – до значних економічних втрат.

Помилки працівників, що працюють з комп'ютером в адміністративноуправлінській сфері, викликають, великі за масштабами наслідки. Проте незадовільний функціональний стан користувачів комп'ютерів може викликати небажані наслідки (професійні та професійно-зумовлені захворювання), що також пов'язано зі значними соціальними та економічними втратами враховуючи стрімке зростання кількості комп'ютеризованих робочих місць.

Визначення та вивчення факторів, що впливають на функціональний стан користувачів комп'ютерів дозволить виділити основні причини виникнення станів напруженості, стомлення, стресу і здійснити відповідні профілактичні заходи.

Дослідження, проведені фахівцями всесвітніх організацій охорони здоров'я (ВООЗ) показали, що у професійних операторів та канцелярських службовців, які у своїй діяльності використовують ВДТ, частіше зустрічаються порушення органів зору, опорно-рухового апарату, центральної нервової, серцево-судинної, імунної та статевої систем, захворювання шкіри. Необхідно зазначити, що вже в перші роки впровадження ВДТ в Європі та США була зафіксована значна кількість скарг операторського персоналу на загальне недомагання, передчасне стомлювання, головний біль, порушення функцій

органів зору, які здійснювали несприятливий психофізіологічний вплив на самопочуття та працездатність операторів. Однак, в той час основна увага приділялась розвитку техніки, а людина залишалась без необхідного захисту.

В умовах сучасного виробництва, яке характеризується масовим характером та широким застосуванням комп'ютерної техніки попередні пріоритети зазнали суттєвої трансформації. У центрі уваги вітчизняних та зарубіжних фахівців є питання щодо визначення характеру та умов праці користувачів комп'ютерів, функціональних змін у динаміці виконання трудових завдань, захворюваності та стану здоров'я, розробки засобів захисту.

Дослідження медиків-гігієністів, психологів, світлотехніків та фахівців з охорони праці та ергономіки показали, що сучасна професія користувача ВДТ належить до розумової праці, яка характеризується: високою напруженістю зорових функцій; одноманітною позою; великою кількістю стереотипних високо координованих рухів, що виконуються лише м'язами кистей рук на фоні малої загальної рухової активності; значним нервовоемоційним компонентом, особливо в умовах дефіциту часу; роботою з великими масивами інформації, що викликає активізацію уваги та інших вищих психічних функцій. Крім того, при роботі з дисплеями на електроннопроменевих трубках виникає вплив на користувача цілої низки факторів фізичної природи – електростатичні поля, радіочастотне та рентгенівське випромінювання тощо [31].

При роботі з ВДТ основне навантаження припадає на елементи зорового аналізатора. Ще в перші роки експлуатації комп'ютерів із відеотерміналами з'явились масові скарги на порушення зору, під яким і розуміють здатність сприймати величину, форму та колір предметів, їх взаємне розміщення та відстань між ними. Проведені у сімдесяті роки обстеження у США встановили, що майже у половини професійних операторів ВДТ є різноманітні порушення зорової функції. Враховуючи виняткову важливість даного питання, з огляду на масовий характер сучасної професії користувача комп'ютера, в різних країнах світу були проведені фундаментальні дослідження щодо впливу відеотермінала комп'ютера на очі та зір користувача. Однак, необхідно зазначити, що

опубліковані результати численних досліджень не завжди відповідним чином корелюються між собою. Так в опублікованому в 1985 році звіт Національної ради з науки (США) зроблено висновки про те, що такі захворювання операторів комп'ютерів, як глаукома, катаракта, запалення райдужної оболонки ока не пов'язані з роботою за ВДТ. В той же час, за іншими даними електромагнітне випромінювання від ВДТ може викликати катаракту, тобто помутніння кришталика ока.

Сучасні медичні обстеження Кількох десятків тисяч професійних користувачів комп'ютерів, проведені у Німеччині та Італії, показали, що частота порушень зору в них на 15 – 20% більша ніж серед працівників, які у своїй діяльності не використовують ВДТ. Наукова група Національної ради наукових досліджень США сформувала термін «астенопія», який визначається як будь-які суб'єктивні зорові симптоми чи емоційний дискомфорт, що є результатом зорової діяльності. Симптоми астенопії були класифіковані на «очні» (біль, печія та різь в очах, почервоніння повік та очних яблук, ломоти у надбрівній частині) та «зорові» (пелена перед очима, подвоєння предметів, мерехтіння, швидка втома під час зорової роботи).

Більшість досліджень показує, що у операторів ВДТ «очні» симптоми зустрічали частіше, ніж «зорові», причому частота проявів астенопії вища у жінок, ніж у чоловіків. Відмічено також, що порушення функцій зору корелюють із віком операторові ВДТ. Астенопія більш виражена у операторів старшого та середнього віку.

3.2 Причини пожеж у приміщеннях з ПК

Вогонь, що вийшов із під контролю, здатний викликати значні руйнівні та смертоносні наслідки. До таких проявів вогняної стихії належать пожежі. Пожежа – це неконтрольований процес знищування або пошкодження вогнем

майна, під час якого виникають чинники, небезпечні для істот та навколишнього природного середовища;

Залежно від розмірів матеріальних збитків пожежі поділяються на особливо великі (коли збитки становлять від 10000 і більше розмірів мінімальної заробітної плати), великі (збитки сягають від 1000 до 10000 розмірів мінімальної заробітної плати). Проте наслідки пожеж не обмежуються суто матеріальними втратами, пов’язаними зі знищенням або пошкодженням основних виробничих та невиробничих фондів, товарно1матеріальних цінностей, особистого майна населення, витратами на ліквідацію пожежі та її наслідків, на компенсацію постраждалим і т. ін. Найвідчутнішими, безперечно, є соціальні наслідки, які, передусім, пов’язуються з загибеллю і травмуванням людей, а також пошкодженням їх фізичного та психологічного стану, зростанням захворюваності населення, підвищенням соціальної напруги у суспільстві внаслідок втрати житлового фонду, позбавленням робочих місць тощо.

Не слід забувати й про екологічні наслідки пожеж, до яких, у першу чергу, можна віднести забруднення навколишнього середовища продуктами горіння, засобами пожежогасіння та пошкодженими матеріалами, руйнування озонового шару, втрати атмосферою кисню, теплове забруднення, посилення парникового ефекту.

Цілком природно, що існує безпосередня зацікавленість у зниженні вірогідності виникнення пожеж і зменшенні шкоди від них. Досягнення цієї мети є досить актуальним і складним соціально-економічним завданням, вирішенню якого повинні сприяти системи пожежної безпеки.

Щоб перешкодити появленню пожеж потрібно розібратися в основних причинах виникнення. Класифікують два види причин: електричного та неелектричного походження, у приміщеннях з ПК частіше виникають пожежі саме електричного походження [32].

До причин електричного походження належать:

– іскріння в електричних апаратах, машинах, електростатичні розряди і розряди блискавки;

– трум короткого замикання, який нагріває провідник до високої температури, при якій може виникнути запалення ізоляції, а також значні електричні перевантаження проводу і обмоток електричних апаратів та машин;

– погані контакти в місцях з'єднання проводу, коли внаслідок великого перехідного опору виділяється велика кількість тепла;

– електрична дуга, що виникає в результаті помилкових операцій з комутаційною апаратурою при перемиканнях у електроустановках або при дуговому електричному зварюванні, яке може викликати запалення розташованих поблизу горючих матеріалів та апаратів, наповнених маслом;

– аварія масляного вимикача при вимкненні струму коротким замиканням, якщо його розривна сила потужності нижча вимкненої потужності струму, що може призвести до викиду пари, масла в навколишнє середовище і до утворення вибухонебезпечної суміші з повітрям;

– в акумуляторних приміщеннях при заряджанні акумуляторів з електроліту виділяється кисень і водень, які змішуються з повітрям і при недостатній вентиляції концентрація водню може бути вищою нижньої границі вибуховості. Випадкова іскра може стати причиною вибуху.

Причинами пожеж і вибухів неелектричного походження можуть бути саме такі;

– необережне поводження з вогнем при газозварювальних роботах;

– неправильне поводження з апаратурою газової зварки і паяльними лампами, а також неправильне розігрівання кабельної маси і просочувальних сполук;

– пошкодження котельних і виробничих печей, опалювальних приладів і порушення режимів їхньої роботи;

– пошкодження виробничого устаткування і порушення технологічного процесу, в результаті якого можливе виділення горючих газів, пари або пилу в повітряне середовище;

– паління в пожежо- та вибухонебезпечних приміщеннях;

– самоспалахування деяких матеріалів.

Перейдемо до наступного питання даного розділу.

3.3 Небезпечні і шкідливі виробничі фактори, джерелом яких є комп'ютер

До небезпечних психофізіологічних та шкідливих виробничих чинників належать фізичні: статичні, динамічні та гіподинамічні, а також нервово-психічні (розумове, зорове, емоційне) перевантаження.

Праця економістів, фінансистів, працівників банківських установ, науково-дослідних та інших установ, а також інших працівників невиробничої сфери характеризується тривалою багатогодинною (8 год і більше) працею в одноманітному напруженому положенні, малою руховою активністю при значних локальних динамічних навантаженнях.

Робоче положення "сидячи" супроводжується статичним навантаженням значної кількості м'язів ніг, плечей, шиї та рук, що дуже втомлює. М'язи перебувають довгий час у скороченому стані і не розслабляються, що погіршує кровообіг. В результаті, виникають больові відчуття в руках, шиї, верхній частині ніг, спині та плечових суглобах.

Внаслідок динамічного навантаження на кістково-м'язовий апарат кистей рук виникають больові відчуття різної сили в суглобах та м'язах кистей рук; оніміння та уповільнена рухливість пальців; судоми м'язів кисті; ниючий біль в ділянці зап'ястя.

В результаті виникають локальні м'язові перенапруження хронічні розтягнення м'язів травматичного характеру, що можуть викликати професійні

захворювання: дисоціативні моторні розлади, захворювання периферійної нервової та кістково-м'язової систем. Ці захворювання увійшли до Переліку професійних захворювань, затвердженого постановою Кабінету Міністрів України від 8 листопада 2000 р., № 1662.

Крім того, робота "сидячи" призводить до зниження м'язової активності - гіподинамії. За браком рухів відбувається зниження споживання кисню тканинами організму, сповільнюється обмін речовин. Це сприяє розвитку атеросклерозу, ожиріння, може стати причиною дистрофії міокарда, хронічного головного болю, запаморочення, безсоння, роздратування.

Помірними гімнастичними вправами можна викликати активізацію обміну речовин в організмі, посилити виділення отруйних продуктів життєдіяльності.

Трудова діяльність працівників невиробничої сфери належить до категорії робіт, які пов'язані з використанням великих обсягів інформаці, із застосуванням комп'ютеризованих робочих місць, із частим прийняттям відповідальних рішень в умовах дефіциту Часу, безпосереднім контактом із людьми різних типів темпераменту тощо. Це зумовлює високий рівень нервовопсихічного перевантаження, знижує функціональну активність центральної нервової системи, призводить до розладів в її діяльності, розвитку втоми, перевтоми, стресу.

Тривала робота на комп'ютеризованому робочому місці призводить до значного навантаження на всі елементи зорової системи і зумовлює втому та перевтому зорового аналізатора. Напружена зорова робота викликає "очні" (біль, печія та різь в очах, почервоніння повік та очей, ломота у надбрівній частині) та "зорові" (пелена перед очима, подвоєння предметів, мерехтіння, швидка втома під час зорової роботи) порушення органів зору, що може викликати головний біль, посилення нервово-психічного напруження, зниження працездатності.

Надмірні фізичні та нервово-психічні перевантаження зумовлюють зміни у фізіологічному та психічному станах працівника, призводять до розвитку

втоми та перевтоми. Втома – це сукупність тимчасових змін у фізіологічному та психологічному стані людини, які з'являються внаслідок напруженої чи тривалої праці і призводять до погіршення її кількісних і якісних показників, нещасних випадків. Втома буває загальною, локальною, розумовою, зоровою, м'язовою. Оскільки організм - єдине ціле, то межа між цими видами втоми умовна і нечітка.

Хід збільшення втоми та її кінцева величина залежать від індивідуальних особливостей працюючого, трудового режиму, умов виробничого середовища тощо. Залежно від характеру вихідного функціонального стану працівника втома може досягати різної глибини, переходити у хронічну втому або перевтому.

Перевтома – це сукупність стійких несприятливих для здоров'я працівників функціональних зрушень в організмі, які виникають внаслідок накопичення втоми. Основною відмінністю втоми від перевтоми є зворотність зрушень при втомі і неповна зворотність їх при перевтомі [33].

Відомо, що розвиток втоми та перевтоми веде до порушення координації рухів, зорових розладів, неуважності, втрати пильності та контролю реальної ситуації. При цьому працівник порушує вимоги технологічних інструкцій, припускається помилок та неузгодженості в роботі; у нього знижується відчуття небезпеки. Крім того, перевтома супроводжується хронічною гіпоксією (кисневою недостатністю), порушенням нервової діяльності.

Проявами перевтоми є головний біль, підвищена стомлюваність, дратівливість, нервозність, порушення сну, а також такі захворювання як вегетосудинна дистонія, артеріальна гіпертонія, виразкова хвороба, ішемічна хвороба серця, інші професійні захворювання. Втома характеризується фізіологічними та психічними показниками її розвитку. Фізіологічними показниками розвитку втоми є високий артеріальний кров'яний тиск, велика частота пульсу, зміни у складі крові.

Психічними показниками розвитку втоми є: погіршення сприйняття подразників, внаслідок чого працівник окремі подразники зовсім не сприймає, а

інші сприймає із запізненням; зменшення здатності концентрувати увагу, свідомо її регулювати; посилення мимовільної уваги до побічних подразників, які відволікають працівника від трудового процесу; погіршення запам'ятовування та труднощі пригадування інформації, що знижує ефективність професійних знань; сповільнення процесів мислення, втрата їх гнучкості, широти, глибини і критичності; підвищення дратівливості, поява депресивних станів; порушення сенсомоторної координації, збільшення часу реакцій на подразники; зміни частоти слуху, зору.

3.4 Заходи захисту користувачів ПК від шуму, вібрації та випромінювань

Звук – це розповсюдження звукової хвилі в пружному середовищі. Він характеризується частотою звукових коливань, амплітудою та часовими змінами коливань. Звуковий спектр поділяється та інфразвук, частота коливань звукової хвилі якого знаходиться в межах від 0 до 20 Гц - людина цих звуків не відчуває. Звуки з частотою від 20 до 20 000 Гц - звуковий діапазон, який людина чує . Частота від 20 000 Гц до 109 Гц - ультразвук, від 109 і вище - гіперзвук – людське вухо їх не сприймає [34].

Шум – це коливання звукової хвилі в звуковому діапазоні, що характеризується змінною частотою і амплітудою, непостійні в часі, які не несуть корисної інформації людині.

Вібрація - це механічні коливання, що призводять до розладу життєвих функцій людини, шкідливо впливають на роботу обладнання та руйнують будівельні конструкції.

Існують такі способи боротьби з шумом механічного походження та вібрацією:

– зменшення шуму та вібрації безпосередньо в джерелах їх виникнення, застосовуючи обладнання, що не утворює шуму, замінюючи

ударні технологічні процеси безударними, застосовуючи деталі із недзвінких матеріалів (пластмаса, гума, деревина та ін), підшипники ковзання замість кочення, косозубі та шевронні зубчасті передачі замість прямозубих, проводячи своєчасне обслуговування та ремонт елементів, що створюють шум та ін.;

– зменшення шуму та вібрації на шляхах їх розповсюдження заходами звуко- та віброізоляції, а також вібро- та звукопоглинання;

– зменшення шкідливої дії шуму та вібрації, застосовуючи індивідуальні засоби захисту та запроваджуючи раціональні режими праці та відпочинку. Способи зменшення шумів аеродинамічного та гідродинамічного походження;

– зменшення швидкості руху повітря та рідин, що забезпечує їх ламінарний режим течії; - встановлення глушників, що вміщують звукопоглинаючі матеріали і поглинають звукову та коливальну енергію, що потрапляє на них;

– встановлення глушників, що подрібнюють потоки, зменшуючії таким чином їх енергію; спрямування потоку у зворотньому напрямку, що дає змогу взаємопоглинатися енергіям потоків прямого та зворотнього напрямків, які контактують через перетинку.

Одним із найпростіших та економічно доцільних способів зниження шуму є застосування методів звукоізоляції та звукопоглинання.

Звукоізоляція. Звукоізолюючі кожухи, екрани, стіни, перетинки виготовляють із щільних твердих матеріалів, здатних добре відбивати звукові хвилі, запобігаючи їх розповсюдженню (метал, пластмаса, бетон, цегла). Ефективність звукоізоляції характеризується коефіцієнтом відбиття Р, який чисельно дорівнює долі енергії звукової хвилі, відбитої від поверхні огородження, що ізолює джерело шуму ($\beta = 0,8...0,9$).

Для звукоогороджуючих конструкцій визначається коефіцієнт звукопровідності. Для дифузійного звукового поля, в якому всі напрямки розповсюдження прямих і відбитих звукових хвиль

Звукопоглинання. Пористі конструкції та матеріали, здатні поглинати падаючу на них енергію звукових хвиль, яка в цьому випадку витрачається на приведення в рух повітря в масі конструкції. Долю енергії звукової хвилі, що поглинає пористий матеріал, називають коефіцієнтом звукопоглинання а. Звукопоглинаючими матеріалами є поліуретан, мінеральна вата, супертонке скловолокно, пористий бетон, перфоровані гіпсові плити - акміґран та ін., що мають коефіцієнт звукопоглинання (α = 0,2...0,9). Звукопоглинаючі та звукоізолюючі матеріали, зазвичай, використовують разом. Для захисту від шуму, що випромінюється в діапазоні високих та середніх звукових частот, застосовуються акустичні екрани. Це щити, облицьовані зі сторони джерела шуму звукопоглинаючим матеріалом товщиною не менше 50-60 мм. Їх призначення - зниження інтенсивності прямого звуку або відбитого шуму, що спрямовується на працівника. Екран є перепоною, за якою утворюється акустична тінь із низьким рівнем звукового тиску.

Захист від шуму будівельно-акустичним методом потрібно проектувати на підставі акустичного розрахунку, який дозволяє визначити в розрахункових точках очікувані рівні звукового тиску і зіставити з нормованими. Визначення рівня звукового тиску в розрахункових точках проводять згідно з будівельними нормами і правилами «Нормы проектирования. Защита от шума». Для зниження шуму всередині промислових приміщень проводять їх акустичну обробку, яка полягає в розміщенні на внутрішніх поверхнях приміщень звукопоглинаючих матеріалів. Ефект від їх використання досягається за рахунок зменшення енергії звукових хвиль.

Боротьба з аеродинамічним та гідродинамічним шумом. Для поглинання аеродинамічних та гідродинамічних шумів застосовують такі типи глушників: активні й реактивні. Активні глушники застосовують у вигляді облицювальних матеріалів зсередини повітро- та рідинопроводів, які поглинають імпульсні коливання повітря та рідин, що виникають при їх турбулентній течії. Реактивні глушники налаштовуються на найбільш інтенсивну складову шуму за частотою шляхом розрахунку та розміщення елементів глушника, які відбивають

енергію. При цьому досягається зниження шуму на 20-30 дБ. Для отримання ефективного зниження шуму в широкому діапазоні частот застосовують комбіновані глушники.

Боротьба з електромагнітним шумом. Електромагнітний шум виникає при взаємодії феромагнітних мас і змінних магнітних полів. Цей шум характерний для обладнання із електроприводом. Зниження шуму електромагнітного походження досягається шляхом конструктивних змін в електричних машинах.

3.5 Висновки до третього розділу

В результаті виконання кваліфікаційної роботи в розділі про безпеку життєдіяльності та основи охорони праці розглянуто питання, які безпосередньо стосуються предметної області кваліфікаційної роботи:

– досліджено скарги працівників, які працюють за монітором більше половини робочого дня, їх проблематику і методи вирішення. Описано дослідження амереканських науковців на цю тему;

– описано головні терміни і поняття про пожежу, причини винекнення вогню і пожежі в приміщеннях з персональним ком’ютером і засоби запобігання пожежам;

– розглянуто небезпечні шкідливі фактори джерелом яких є комп’ютер, зокрема про втому, її види та причини її появи при роботі за ком’ютером;

– проаналізовано фактори шуму і вібрацій при роботі з персональним комп’ютером.

Дотримання правил техніки безпеки та належних умов праці є запорукою збереження високої працездатності працівників.

ВИСНОВКИ

Автоматизація роботи з документами є дуже відповідальною задачею, тому у даному проекті було використано найефективніші структурні та теоретичні прийоми з метою отримати стабільний, швидкий та надійний в експлуатації програмний продукт.

В першому розділі кваліфікаційної роботи освітнього рівня «Бакалавр»:

– чітко визначено функціональні можливості та характеристики програного рішення відповідно до завдання на розробку;

– розглянуто та описано інструменти для виконання та тестування даної роботи;

– висвітлено майбутні характеристики та особливості проекту;

– проаналізовано найкращі методи підтримки проекту.

В другому розділі кваліфікаційної роботи:

– розроблено план порядку роботи та реалізації функціональних характеристик програмного рішення;

– запропоновано найкращі методи для реалізації поставлених задач;

– спроєктовано глобальну інфраструктуру проекту та конкретні шляхи вирішення деяких проблем;

– випробувано проекту на відповідність поставленим вимогам та характеристикам;

– протестовано функціонування розробленого проекту.

У розділі «Безпека життєдіяльності, основи хорони праці» висвітлено базові правила щодо розробки, експлуатації проекту, роботи з ПК.

ПЕРЕЛІК ДЖЕРЕЛ

1. Хоган Б. С. HTML5 и CSS3. Веб-розробка по стандартам нового покоління/ Б.С. Хоган - М.: Махаон, 2019 – 280 с.

2. Список паттернів розробки програмного забезпечення [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/post/136766

3. Список паттернів розробки програмного забезпечення. [Електронний ресурс] – Режим доступу до ресурсу: https://refactoring.guru/uk/design-patterns

4. Уроки по програмуванню на Java. [Електронний ресурс] – Режим доступу до ресурсу: https://metanit.com/java

5. Документація Java [Електронний ресурс] – Режим доступу до ресурсу: https://java.net

6. Документація Spring [Електронний ресурс] – Режим доступу до ресурсу: https://spring.io/quickstart

7. Робота з файлами у Spring [Електронний ресурс] – Режим доступу до ресурсу: https://www.baeldung.com/spring-file-upload

8. Робота з JPA у Spring [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/post/435114/

9. Робота з базами данних у Spring [Електронний ресурс] – Режим доступу до ресурсу: https://www.baeldung.com/the-persistence-layer-with-spring-data-jpa

10. Робота з файлами у Java [Електронний ресурс] – Режим доступу до ресурсу:https://javarush.ru/groups/posts/2275-files-path

11. Робота з REST запитами у Spring [Електронний ресурс] – Режим доступу до ресурсу:https://javarush.ru/groups/posts/2488-obzor-rest-chastjh-3-sozdanie-restful-servisa-na-spring-boot

12. Робота з Spring MVC [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/post/336816/

13. Документація Spring configuration [Електронний ресурс] – Режим доступу до ресурсу: https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/context/annotation/Configuration.html

14. Документація до Resct- Bootstrap [Електронний ресурс] – Режим доступу до ресурсу: https://react-bootstrap.netlify.app/

15. Туторіал по CSS [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/CSS

16. Туторіал по HTML [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/HTML

17. Туторіал по JavaScript [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/JavaScript

18. Туторіал по Web API [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/API

19. Документація по таблицях Bootstrap [Електронний ресурс] – Режим доступу до ресурсу: https://getbootstrap.com/docs/4.0/content/tables/

20. Розробка інтрерфейсу кроистувача [Електронний ресурс] – Режим доступу ресурсу: https://www.techtarget.com/searchapparchitecture/definition/user-interface-UI

21. Документація по SPA [Електронний ресурс] – Режим доступу до ресурсу: https://viking.com.ua/blog/scho-take-spa-protseduri-ta-scho-take-spa-spa

22. Туторіал по React [Електронний ресурс] – Режим доступу до ресурсу: https://metanit.com/web/react/1.1.php

23. Туторіал по SPA [Електронний ресурс] – Режим доступу до ресурсу: https://single-spa.js.org/docs/ecosystem-react/

24. Розробка програмного забезпечення з використання JPA [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/post/265061/

25. Спецтфікація JSON WEB token [Електронний ресурс] – Режим доступу до ресурсу: https://cyberpolygon.com/ru/materials/security-of-json-web-tokens-jwt/

26. Туторіал по REST [Електронний ресурс] – Режим доступу до ресурсу: https://systems.education/what-is-rest

27. Специфікацйія Swagger UI [Електронний ресурс] – Режим доступу до ресурсу: https://swagger.io/resources/open-api/

28. Використання бібліотеки Swagger у Java програмах: вебсайт. URL: https://habr.com/ru/post/536388/

29. Туторіал по Spring MVC [Електронний ресурс] – Режим доступу до ресурсу: https://betacode.net/10129/spring-mvc-tutorial-for-beginners

30. Розробка програмних застосунків на Spring frameworks [Електронний ресурс] – Режим доступу до ресурсу: https://www.tutorialspoint.com/spring/spring_web_mvc_framework.htm

31. Одарченко М. С. Основи охорони праці : підручник / М. С. Одарченко, А. М. Одарченко, В. І. Степанов, Я. М. Черненко, 2017. – 334 с.

32. Гандзюк М. П., Желібо Є. П., Халімовський М. О. Основи охорони праці: Підруч. для студ. вищих навч. закладів. За ред. М. П. Гандзюка. - К.: Каравела, 2004. - 408 с.

33. Конспект лекцій з курсу «Охорона праці в галузі» / Укладачі: Яскілка В.Я., Олійник М.З. – Тернопіль: Вид-во ТНТУ імені Івана Пулюя, 2018. – 56 с.

34. Основи охорони праці: Підручник. 21ге видання, доповнене та перероблене. / К. Н. Ткачук, М. О. Халімовський, В. В. Зацарний, Д. В. Зеркалов, Р. В. Сабарно, О. І. Полукаров, В. С. Коз’яков, Л. О. Мітюк. За ред. К. Н. Ткачука і М. О. Халімовського. — К.: Основа, 2006 — 448 с.

ДОДАТКИ

Додаток А

Лістинг файлів клієнтської частини А

Лістинг файлу «index.js»

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  document.getElementById('root')
);

reportWebVitals();
```

Лістинг файлу «app.js»

```
import React, { useState } from "react";
import "./App.css";
import "bootstrap/dist/css/bootstrap.min.css";
import Header from "./components/Header";
import {
  Route,
  BrowserRouter as Router,
  Switch,
  Redirect,
} from "react-router-dom";
import Home from "./pages/Home";
import Login from "./pages/Login";
import Category from "./pages/Category";
import Documents from "./pages/Documents";
import managemenetDocument from "./pages/managemenetDocument";
import CreateDocument from "./pages/CreateDocument";
import UpdateDocument from "./pages/UpdateDocument";
import createCategory from "./pages/createCategory";
import managementCategory from "./pages/managementCategory";
import searchDocument from "./pages/searchDocument";
import changeCategoty from "./pages/changeCategoty";
import notFound from "./pages/notFound";

function App() {
  const [isLogin, setLogin] = useState(false);

  const logout = () => {
    setLogin(false);
  };

  const login = () => {
```

```
    setLogin(true);
  };

  const PrivateRoute = ({ component: Component, ...rest }) => (
    <Route
      {...rest}
      render={(props) =>
        localStorage.getItem("accsess_token") ? (
          <Component {...props} />
        ) : (
          <Redirect to={{ pathname: "/" }} />
        )
      }
    />
  );

  return (
    <div className="App">
      <Router>
        <Header isLogin={isLogin} logout={logout} />
        <br></br>
        <br></br>
        <br></br>
        <Switch>
          <Route exact path="/" component={Home} />
          <Route
            exact
            path="/login"
            component={Login}
            isLogin={isLogin}
            login={login}
          />
          <Route exact path="/category/:id" component={Category} />
          <Route exact path="/document/:id" component={Documents} />
          <Route exact path="/search/:name"
component={searchDocument} />
          <PrivateRoute
            exact
            path="/management/document"
            component={managemenetDocument}
          />
          <Route
            exact
            path="/management/document/create"
            component={CreateDocument}
          />
          <Route
            exact
            path="/management/document/update/:id"
            component={UpdateDocument}
          />
          <PrivateRoute
            exact
            path="/management/category"
            component={managementCategory}
          />
          <Route
            exact
```

```
                  path="/management/category/create"
                  component={createCategory}
                />
                <Route
                  exact
                  path="/management/category/update/:id"
                  component={changeCategoty}
                />
                <Route exact={true} path="*" component={notFound} />
              </Switch>
            </Router>
          </div>
        );
      }

export default App;
```

Лістинг файлу «Home.js»

```
import React, { Component } from "react";
import {
  Container,
  Tab,
  Row,
  Col,
  ListGroup,
  Button,
  InputGroup,
  FormControl,
} from "react-bootstrap";
import { serverUrl } from "../config.json";
import { Link } from "react-router-dom";
export default class Home extends Component {
  constructor(props) {
    super(props);
    this.state = {
      error: null,
      isLoaded: false,
      items: [],
    };
  }

  handleSubmit(event) {
    document.location.href =
      "/search/" + document.getElementById("documentName").value;
    event.preventDefault();
  }

  componentDidMount() {
    fetch(serverUrl + "v1/sections/get", {
      method: "GET",
      headers: {
        accept: "*/*",
      },
    })
      .then((res) => res.json())
```

```
      .then(
        (result) => {
          this.setState({
            isLoaded: true,
            items: result.data,
          });
        },
        (error) => {
          this.setState({
            isLoaded: true,
            error,
          });
        }
      );
  }
  render() {
    const { error, isLoaded, items } = this.state;
    if (error) {
      return <h1>Error</h1>;
    } else if (!isLoaded) {
      return <h1></h1>;
    } else {
      return (
        <Container>
          <Row>
            <Col sm={6}>
              <br></br>
              <h3>Пошук за реквізитами</h3>
              <InputGroup className="mb-3">
                <FormControl
                  id="documentName"
                  placeholder="Пошук"
                  aria-label="Recipient's username"
                  aria-describedby="basic-addon2"
                />
                <Button variant="primary"
onClick={this.handleSubmit}>
                  Найти
                </Button>
              </InputGroup>
            </Col>
            <Col sm={6}>
              <br></br>

              <h3>Категорії</h3>
              <ListGroup variant="flush">
                {items.map((items) => (
                  <ListGroup.Item className="listCategory"
id={items.id}>
                    <Link className="link"
to={`/category/${items.id}`}>
                      {items.name}
                    </Link>
                  </ListGroup.Item>
                ))}
              </ListGroup>
            </Col>
          </Row>
```

```
            </Container>
          );
        }
      }
}
```

Лістинг файлу «Login.js»

```
import React, { Component } from "react";
import { Container, Row, Col, Form, Button, Breadcrumb } from
"react-bootstrap";
import axios from "axios";
import { serverUrl } from "../config.json";
import { Link } from "react-router-dom";
export default class Login extends Component {
  constructor(props) {
    super(props);
    this.state = {
      email: "",
      password: "",
      formErrors: { email: "", password: "" },
      emailValid: false,
      passwordValid: false,
      formValid: false,
    };
  }

  handleUserInput = (e) => {
    const name = e.target.name;
    const value = e.target.value;
    this.setState({ [name]: value });
  };

  handleSubmit(event) {
    axios
      .post(serverUrl + "v1/auth/login", {
        password: document.forms.auth.elements.password.value,
        email: document.forms.auth.elements.login.value,
      })
      .then(function (response) {
        localStorage.setItem("accsess_token",
response.data.accessToken);
        localStorage.setItem("refresh_token",
response.data.refreshTokrn);
        document.location.href = "/";
      })
      .catch(function (error) {
        alert("Невірний логін або пароль");
      });

    event.preventDefault();
  }

  componentDidMount() {
    document.title = "Вхід";
  }

  render() {
```

```
        return (
          <>
            <Container>
              <Breadcrumb>
                <Breadcrumb.Item>
                  <Link to={"/"}>Головна</Link>
                </Breadcrumb.Item>
                <Breadcrumb.Item active>Вхід</Breadcrumb.Item>
              </Breadcrumb>
              <Row className="justify-content-md-center">
                <Col sm={4}>
                  <Form name="auth">
                    <Form.Group controlId="formBasicEmail">
                      <Form.Label>Електронна адреса</Form.Label>
                      <Form.Control
                        name="login"
                        type="email"
                        placeholder="Введіть електронну адресу"
                      />
                    </Form.Group>

                    <Form.Group controlId="formBasicPassword">
                      <Form.Label>Пароль</Form.Label>
                      <Form.Control
                        name="password"
                        type="password"
                        placeholder="Введіть пароль"
                      />
                    </Form.Group>
                    <Link to={"/"}>
                      <Button
                        variant="primary"
                        type="submit"
                        onClick={(e) => this.handleSubmit(e)}
                      >
                        Увійти
                      </Button>
                    </Link>
                  </Form>
                </Col>
              </Row>
            </Container>
          </>
        );
      }
}
```

Лістинг файлу «Category.js»

```
import React, { Component } from "react";
import {
  Col,
  Container,
  ListGroup,
  Row,
  Table,
```

```
  Breadcrumb,
} from "react-bootstrap";
import { serverUrl } from "../config.json";
import { Link } from "react-router-dom";

export default class Category extends Component {
  constructor(props) {
    super(props);
    this.state = {
      error: null,
      isLoaded: false,
      categories: [],
      section: {},
    };
  }

  changeCategory(event, id) {
    fetch(serverUrl + "v1/sections/get/" + id, {
      method: "GET",
      headers: {
        accept: "*/*",
      },
    })
      .then((res) => res.json())
      .then(
        (result) => {
          this.setState({
            isLoaded: true,
            section: result.data,
          });
        },
        (error) => {
          this.setState({
            isLoaded: true,
            error,
          });
        }
      );
    event.preventDefault();
  }
  componentDidMount() {
    fetch(serverUrl + "v1/sections/get/" + this.props.match.params.id, {
      method: "GET",
      headers: {
        accept: "*/*",
      },
    })
      .then((res) => res.json())
      .then(
        (result) => {
          this.setState({
            isLoaded: true,
            section: result.data,
          });
        },
        (error) => {
          this.setState({
            isLoaded: true,
```

```
          error,
        });
      }
    );

  fetch(serverUrl + "v1/sections/get/", {
    method: "GET",
    headers: {
      accept: "*/*",
    },
  })
    .then((res) => res.json())
    .then(
      (result) => {
        this.setState({
          isLoaded: true,
          categories: result.data,
        });
      },
      (error) => {
        this.setState({
          isLoaded: true,
          error,
        });
      }
    );
}
render() {
  const { error, isLoaded, categories, section } = this.state;
  if (error) {
    return <h1></h1>;
  } else if (!isLoaded) {
    return <h1></h1>;
  } else {
    return (
      <Container>
        <Breadcrumb>
          <Breadcrumb.Item>
            <Link className="breadCrumbLink" to={"/"}>
              Головна
            </Link>
          </Breadcrumb.Item>
          <Breadcrumb.Item className="breadCrumbLink">
            <Link className="readCrumbLink" to={"/"}>
              Категорії
            </Link>
          </Breadcrumb.Item>
          <Breadcrumb.Item active>{section.name}</Breadcrumb.Item>
        </Breadcrumb>
        <Row>
          <Col sm={3}>
            <h3>Категорії</h3>
            <ListGroup className="listCategory" variant="flush">
              {categories.map((categories) => (
                <ListGroup.Item
                  action
                  id={`category_${categories.id}`}
                  onClick={window.scrollTo(0, 0)}
```

```
                  >
                    <Link
                      className="link"
                      onClick={(e) => this.changeCategory(e, categories.id)}
                      to={`/category/${categories.id}`}
                    >
                      {categories.name}
                    </Link>
                  </ListGroup.Item>
                ))}
              </ListGroup>
            </Col>
            <Col sm={9}>
              <h3>{section.name}</h3>
              <Table responsive size="sm">
                <thead>
                  <tr>
                    <th>Назва</th>
                    <th>Дата</th>
                  </tr>
                </thead>
                <tbody>
                  {section.documents?.map((documents) => (
                    <tr>
                      <td>
                        <Link
                          className="tableLink"
                          to={`/document/${documents.id}`}
                        >
                          {documents.name}
                        </Link>
                      </td>
                      <td>{documents.createTime.split("T")[0]}</td>
                    </tr>
                  ))}
                </tbody>
              </Table>
            </Col>
          </Row>
        </Container>
      );
    }
  }
}
```

Лістинг файлу «Documents.js»

```
import React, { Component } from "react";
import "../App.css";
import AllPagesPDFViewer from "../components/pdf-pages";
import {
  Col,
  Container,
  Row,
  Breadcrumb,
  Table,
```

```
  Button,
} from "react-bootstrap";
import { serverUrl } from "../config.json";
import { Link } from "react-router-dom";

export default class Documents extends Component {
  constructor(props) {
    super(props);
    this.state = {
      error: null,
      isLoaded: false,
      documentsItem: [],
      status: null,
      categoryName: null,
      pdf: null,
    };
  }
  downloadFile(event) {
    window.open(serverUrl + "v1/docs/load/" + this.state.documentsItem.id);
    event.preventDefault();
  }
  componentDidMount() {
    const checkPdf = (nameFile, id) => {
      if (nameFile.includes("pdf") === true) {
        this.setState({ pdf: `${serverUrl}v1/docs/load/${id}` });
      } else {
        this.setState({ pdf: `${serverUrl}v1/docs/load/${id}/pdf` });
      }
    };
    const id = this.props.match.params.id;
    fetch(serverUrl + "v1/docs/get/" + id, {
      method: "GET",
      headers: {
        accept: "*/*",
        Authorization: "Bearer " + localStorage.getItem("accsess_token"),
      },
    })
      .then((res) => res.json())
      .then(
        (result) => {
          document.title = result.data.name;
          checkPdf(result.data.fileName, result.data.id);
          this.setState({
            isLoaded: true,
            documentsItem: result.data,
            status: result.data.status,
          });
          if (this.state.status === "ACTIVE")
            this.setState({ status: "Діючий" });
          if (this.state.status === "INOPERATIVE")
            this.setState({ status: "Припинений" });
          if (this.state.status === "ARCHIVED")
            this.setState({ status: "Архівнмй" });
          fetch(
            serverUrl + "v1/sections/get/" +
this.state.documentsItem.sectionId,
            {
              method: "GET",
```

```
            headers: {
              accept: "*/*",
              Authorization:
                "Bearer " + localStorage.getItem("accsess_token"),
            },
          }
        )
          .then((res) => res.json())
          .then(
            (result) => {
              this.setState({
                categoryName: result.data.name,
              });
            },
            (error) => {
              this.setState({
                isLoaded: true,
                error,
              });
            }
          );
      },
      (error) => {
        this.setState({
          isLoaded: true,
          error,
        });
      }
    );
}
render() {
  const { error, isLoaded, documentsItem, status, categoryName, pdf } =
    this.state;
  if (error) {
    return <h1>Error</h1>;
  } else if (!isLoaded) {
    return <h1></h1>;
  } else {
    return (
      <Container>
        <Breadcrumb>
          <Breadcrumb.Item>
            <Link to={"/"}>Головна</Link>
          </Breadcrumb.Item>
          <Breadcrumb.Item>
            <Link to={"/category/" + documentsItem.sectionId}>
              {categoryName}
            </Link>
          </Breadcrumb.Item>
          <Breadcrumb.Item active>Документи</Breadcrumb.Item>
        </Breadcrumb>
        <br></br>
        <Row>
          <Col sm={3}>
            <Table className="table">
              <tbody>
                <tr>
                  <td>
```

```
                    <b>Назва:</b>
                  </td>
                  <td clas>{documentsItem.name}</td>
                </tr>
                <tr>
                  <td>
                    <b>Категорія:</b>
                  </td>
                  <td>
                    <Link
                      className="tableLink"
                      to={"/category/" + documentsItem.sectionId}
                    >
                      {categoryName}
                    </Link>
                  </td>
                </tr>
                <tr>
                  <td>
                    <b>Статус:</b>
                  </td>
                  <td>{status}</td>
                </tr>
                <tr>
                  <td>
                    <b>Дата:</b>
                  </td>
                  <td>{documentsItem.createTime.split("T")[0]}</td>
                </tr>
              </tbody>
            </Table>
            <Link>
              <Button
                variant="primary"
                size="sm"
                style={{ width: "100%" }}
                onClick={(e) => this.downloadFile(e)}
              >
                Завантажити файл
              </Button>
            </Link>
          </Col>
          <Col sm={1}></Col>
          <Col sm={8}>
            <h4>{documentsItem.name}</h4>
            <AllPagesPDFViewer pdf={pdf} />
          </Col>
        </Row>
      </Container>
    );
  }
}
```

Додаток Б

Лістинг файлів серверної частини Б

Лістинг файлу «DocsApplication.java»

```
package com.tntu.server.docs;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.autoconfigure.domain.EntityScan;
import
org.springframework.data.jpa.repository.config.EnableJpaRepositories
;
import
org.springframework.transaction.annotation.EnableTransactionManageme
nt;

@SpringBootApplication(scanBasePackages =
{"com.tntu.server.docs.*"})
@EnableJpaRepositories(basePackages =
{"com.tntu.server.docs.db.repositories.db"})
@EntityScan(basePackages = {"com.tntu.server.docs.db.entities"})
@EnableTransactionManagement
public class DocsApplication {

    public static void main(String[] args) {
        SpringApplication.run(DocsApplication.class, args);
    }

}
```

Лістинг файлу «DocumentService.java»

```
package com.tntu.server.docs.core.services;

import com.tntu.server.docs.core.data.exceptions.DocsException;
import
com.tntu.server.docs.core.data.exceptions.docs.DocumentAlreadyExists
Exception;
import
com.tntu.server.docs.core.data.exceptions.docs.DocumentNotExistsExce
ption;
import
com.tntu.server.docs.core.data.exceptions.section.SectionNotExistsEx
ception;
import
com.tntu.server.docs.core.data.exceptions.storage.file.FileNotExists
Exception;
import com.tntu.server.docs.core.data.models.docs.DocumentModel;
import com.tntu.server.docs.core.repositories.DocumentRepository;
import com.tntu.server.docs.core.utils.Updater;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
```

```
import org.springframework.web.multipart.MultipartFile;

import java.util.List;
import java.util.stream.Collectors;

@Service
public class DocumentService {

    @Autowired
    private DocumentRepository documentRepository;

    @Autowired
    private SectionService sectionService;

    @Autowired
    private FilesService filesService;

    public DocumentModel getDocument(long id) throws
DocumentNotExistsException {
        return documentRepository.getDocument(id)
                .orElseThrow(DocumentNotExistsException::new);
    }

    public MultipartFile loadFile(long id) throws DocsException {

        var document = getDocument(id);
        var section =
sectionService.getSection(document.getSectionId());

        var sectionName = section.getName();
        var originalFileName = document.getFileName();

        if (originalFileName == null || originalFileName.isEmpty())
            throw new FileNotExistsException();

        var location = filesService.combine(sectionName,
originalFileName);
        var fileName = document.getName();
        if (originalFileName.contains("."))
            fileName +=
originalFileName.substring(originalFileName.lastIndexOf("."));
        return filesService.getFile(location, fileName);
    }

    public List<DocumentModel> getDocumentsBySection(long sectionId)
throws SectionNotExistsException {
        if (!sectionService.isExists(sectionId))
            throw new SectionNotExistsException();

        return documentRepository.getAllBySection(sectionId);
    }

    public DocumentModel createDocument(DocumentModel document)
throws DocsException {
        if (documentRepository.exists(document.getName()))
            throw new DocumentAlreadyExistsException();

        return save(document);
```

```
    }

    public void updateDocument(long id, DocumentModel newDocument)
throws DocsException {
        var document = getDocument(id);
        var oldSectionId = document.getSectionId();
        var newSectionId = newDocument.getSectionId();

        if (newSectionId != null &&
!newSectionId.equals(oldSectionId))
            moveDocument(document, newSectionId);

        var updater = new Updater<>(document, newDocument);
        updater.update(DocumentModel::getSectionId,
DocumentModel::setSectionId);
        updater.update(DocumentModel::getName,
DocumentModel::setName);
        updater.update(DocumentModel::getCreateTime,
DocumentModel::setCreateTime);

        save(document);
    }

    public void delete(long id) throws DocsException {
        var document = getDocument(id);
        var sectionName =
sectionService.getSection(document.getSectionId()).getName();

        var fileLocation = filesService.combine(sectionName,
document.getName());
        documentRepository.delete(id);
        filesService.deleteFile(fileLocation);
    }

    public List<DocumentModel> findDocuments(String name) {
        return documentRepository.find(name);
    }

    private DocumentModel save(DocumentModel document) throws
SectionNotExistsException {
        if (!sectionService.isExists(document.getSectionId()))
            throw new SectionNotExistsException();
        document = documentRepository.save(document);

        return document;
    }

    public void uploadFile(long id, MultipartFile file) throws
DocsException {
        if (file == null || file.isEmpty())
            return;

        var document = getDocument(id);

        var section =
sectionService.getSection(document.getSectionId());
        var sectionName = section.getName();
```

```
        if (!filesService.isExists(sectionName))
            filesService.createDirectory(sectionName);

        var originalFilename = file.getOriginalFilename();
        var fileName = String.valueOf(id);
        if (originalFilename != null &&
originalFilename.contains("."))
            fileName +=
originalFilename.substring(originalFilename.lastIndexOf("."));

        filesService.saveOrRewrite(sectionName, fileName, file);
        document.setFileName(fileName);
        documentRepository.save(document);
    }

    private void moveDocument(DocumentModel documentModel, Long
newSectionId) throws DocsException {
        var oldSectionId = documentModel.getSectionId();

        var oldSection = sectionService.getSection(oldSectionId);
        var newSection = sectionService.getSection(newSectionId);

        var oldSectionName = oldSection.getName();
        var newSectionName = newSection.getName();

        var documentName = documentModel.getFileName();
        var oldFileLocation = filesService.combine(oldSectionName,
documentName);

        if (filesService.isExists(newSectionName))
            filesService.createDirectory(newSectionName);

        filesService.moveFile(oldFileLocation, newSectionName);
    }

}
```

Лістинг файлу «FilesService.java»

```
package com.tntu.server.docs.core.services;

import com.tntu.server.docs.core.data.exceptions.DocsException;
import
com.tntu.server.docs.core.data.exceptions.storage.file.CanNotMoveExc
eption;
import
com.tntu.server.docs.core.data.exceptions.storage.file.CanNotWriteFi
leException;
import
com.tntu.server.docs.core.data.exceptions.storage.file.DeleteFileExc
eption;
import
com.tntu.server.docs.core.data.exceptions.storage.file.FileAlreadyEx
istsException;
import
com.tntu.server.docs.core.data.exceptions.storage.resource.CanNotCre
ateDirectoryException;
```

```
import
com.tntu.server.docs.core.data.exceptions.storage.resource.InvalidRe
sourceException;
import
com.tntu.server.docs.core.data.exceptions.storage.resource.ResourceN
otExistsException;
import
com.tntu.server.docs.core.data.models.file.BytesMultipartFile;
import com.tntu.server.docs.core.services.storage.StorageService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.web.multipart.MultipartFile;

@Service
public class FilesService {

    @Autowired
    private StorageService storageService;

    public String combine(String root, String... paths) throws
InvalidResourceException {
        return storageService.combine(root, paths);
    }

    public boolean isExists(String resource) throws
InvalidResourceException {
        return storageService.isExists(resource);
    }

    public void createDirectory(String resource) throws
InvalidResourceException, CanNotCreateDirectoryException {
        storageService.createDirectory(resource);
    }

    public void saveFile(String location, MultipartFile file)
            throws CanNotWriteFileException,
FileAlreadyExistsException, InvalidResourceException {
        storageService.saveFile(location, file);
    }

    public MultipartFile getFile(String location, String name)
throws DocsException {
        var fileBytes = storageService.loadFile(location);
        return new BytesMultipartFile(name, fileBytes);
    }

    public void moveFile(String fileLocation, String pathLocation)
            throws InvalidResourceException, CanNotMoveException {
        var isExistsPath = storageService.isExists(fileLocation);
        var isExistsPathTo = storageService.isExists(pathLocation);
        if (!isExistsPath || !isExistsPathTo)
            throw new InvalidResourceException();

        storageService.moveObject(fileLocation, pathLocation);
    }

    public void deleteFile(String location)
            throws InvalidResourceException,
```

```
ResourceNotExistsException, DeleteFileException {
        if (!storageService.isExists(location))
            throw new ResourceNotExistsException();
        storageService.deleteFile(location);
    }

    public synchronized void saveOrRewrite(String location, String
name, MultipartFile file)
            throws InvalidResourceException,
CanNotWriteFileException, FileAlreadyExistsException,
DeleteFileException {
        var resource = storageService.combine(location, name);
        if (storageService.isExists(resource)) {
            try {
                storageService.deleteFile(resource);
            }catch (Exception ignored) {}
        }
        storageService.saveFile(location, name, file);
    }


}
```

Лістинг файлу «RegistrationService.java»

```
package com.tntu.server.docs.core.services;

import com.tntu.server.docs.core.data.exceptions.DocsException;
import
com.tntu.server.docs.core.data.exceptions.auth.CanNotCreateUserExcep
tion;
import
com.tntu.server.docs.core.data.exceptions.auth.RegistrationCodeNotFo
undException;
import
com.tntu.server.docs.core.data.exceptions.auth.RegistrationProblemsE
xception;
import
com.tntu.server.docs.core.data.exceptions.user.ActionOnAdminRoleExce
ption;
import
com.tntu.server.docs.core.data.exceptions.user.RoleNotFoundException
;
import
com.tntu.server.docs.core.data.exceptions.user.UserAlreadyRegistered
Exception;
import com.tntu.server.docs.core.data.models.user.RegistrationModel;
import com.tntu.server.docs.core.data.models.user.RoleModel;
import com.tntu.server.docs.core.data.models.user.UserModel;
import com.tntu.server.docs.core.options.SecureOptions;
import
com.tntu.server.docs.core.repositories.RegistrationRepository;
import com.tntu.server.docs.core.services.mail.MailService;
import net.bytebuddy.utility.RandomString;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
```

```
import java.util.ArrayList;
import java.util.List;

@Service
public class RegistrationService {

    @Autowired
    private SecureOptions secureOptions;

    @Autowired
    private RegistrationRepository registrationRepository;

    @Autowired
    private SecureRandomService secureRandomService;

    @Autowired
    private MailService mailService;

    @Autowired
    private RoleService roleService;

    @Autowired
    private UserService userService;

    @Autowired
    private PasswordEncoder passwordEncoder;


    public void startUserRegistration(String roleName, List<String>
userEmails)
            throws RoleNotFoundException,
ActionOnAdminRoleException, RegistrationProblemsException {
        var exceptions = new ArrayList<DocsException>();
        var role = roleService.getByName(roleName);
        if (role.getName().equals(RoleModel.ADMIN))
            throw new ActionOnAdminRoleException();
        for (String userEmail : userEmails) {
            try {
                startUserRegistration(role, userEmail);
            } catch (DocsException e) {
                exceptions.add(e);
            }
        }
        if (!exceptions.isEmpty())
            throw new RegistrationProblemsException(exceptions);
    }

    public void startUserRegistration(RoleModel roleModel, String
userEmail) throws DocsException {
        if (userService.existsByEmail(userEmail)) {
            throw new UserAlreadyRegisteredException();
        }
        if (registrationRepository.existsByEmail(userEmail)) {
            registrationRepository.deleteByEmail(userEmail);
        }
        var registrationCode = generateRegistrationCode();
        mailService.sendRegistrationMessage(registrationCode,
userEmail, roleModel);
```

```
        var roleId = roleModel.getId();
        registrationRepository.save(userEmail, registrationCode,
roleId);
    }

    private String generateRegistrationCode() {
        var length = secureOptions.getRegistrationCodeLength();
        try {
            return secureRandomService.generateAlphaNumeric(length);
        } catch (Exception e) {
            return RandomString.make(length);
        }
    }

    public UserModel register(RegistrationModel registrationModel)
            throws CanNotCreateUserException,
RegistrationCodeNotFoundException {
        var reg = registrationRepository

.getRegistrationModelByCode(registrationModel.getCode())

.orElseThrow(RegistrationCodeNotFoundException::new);
        UserModel userModel = new UserModel();
        userModel.setUsername(registrationModel.getUsername());
        userModel.setEmail(reg.getEmail());
        var passwordHash =
passwordEncoder.encode(registrationModel.getPassword());
        userModel.setPasswordHash(passwordHash);
        userModel = userService.createNewUser(userModel);
        registrationRepository.deleteByEmail(reg.getEmail());
        return userModel;
    }


}
```

Лістинг файлу «RoleService.java»

```
package com.tntu.server.docs.core.services;

import
com.tntu.server.docs.core.data.exceptions.user.RoleNotFoundException
;
import com.tntu.server.docs.core.data.models.user.RoleModel;
import com.tntu.server.docs.core.repositories.RoleModelRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;


@Service
public class RoleService {

    @Autowired
    private RoleModelRepository roleModelRepository;
```

```
    public List<RoleModel> getAll() {
        return roleModelRepository.getAll();
    }

    public long findId(String name) throws RoleNotFoundException {
        return roleModelRepository
                .findId(name)
                .orElseThrow(() -> new RoleNotFoundException(name));
    }

    public RoleModel getById(long id) throws RoleNotFoundException {
        return
roleModelRepository.findById(id).orElseThrow(RoleNotFoundException::
new);
    }

    public RoleModel getByName(String name) throws
RoleNotFoundException {
        return roleModelRepository
                .findByName(name)
                .orElseThrow(() -> new RoleNotFoundException(name));

    }

    public void ensureExists(long roleId) throws
RoleNotFoundException {
        if (!roleModelRepository.isExists(roleId))
            throw new RoleNotFoundException();
    }

}
```

Лістинг файлу «SectionService.java»

```
package com.tntu.server.docs.core.services;

import com.tntu.server.docs.core.data.exceptions.DocsException;
import
com.tntu.server.docs.core.data.exceptions.section.SectionAlreadyExis
tsException;
import
com.tntu.server.docs.core.data.exceptions.section.SectionNotExistsEx
ception;
import com.tntu.server.docs.core.data.models.docs.SectionModel;
import com.tntu.server.docs.core.repositories.SectionRepository;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import javax.transaction.Transactional;
import java.util.ArrayList;
import java.util.List;

@Service
public class SectionService {
```

```
    private final Logger LOG =
LoggerFactory.getLogger(this.getClass());

    @Autowired
    private SectionRepository sectionRepository;

    @Autowired
    private DocumentService documentService;

    @Autowired
    private FilesService filesService;

    public List<SectionModel> getAllSections() {
        return sectionRepository.getAllSections();
    }

    public SectionModel getSection(long id) throws
SectionNotExistsException {
        return sectionRepository.getSection(id)
                .orElseThrow(SectionNotExistsException::new);
    }

    public SectionModel getSection(String name) throws
SectionNotExistsException {
        return sectionRepository.getSection(name)
                .orElseThrow(SectionNotExistsException::new);
    }

    public SectionModel updateSection(long id, String name) throws
SectionNotExistsException {
        var section = getSection(id);
        section.setName(name);

        return sectionRepository.save(section);
    }

    public SectionModel createSection(SectionModel model) throws
DocsException {
        var name = model.getName();
        if (sectionRepository.exists(name))
            throw new SectionAlreadyExistsException();

        if (!filesService.isExists(name))
            filesService.createDirectory(name);

        return sectionRepository.save(model);
    }

    @Transactional
    public void deleteSection(long id) throws
SectionNotExistsException {
        if (!sectionRepository.exists(id))
            throw new SectionNotExistsException();

        var section = getSection(id);
        for (var document : section.getDocuments()) {
            var documentId = document.getId();
            try {
```

```
                documentService.delete(documentId);
            } catch (DocsException e) {
                LOG.warn("Can not delete document");
            }
        }

        sectionRepository.deleteSection(id);
    }

    public boolean isExists(long id) {
        return sectionRepository.exists(id);
    }

}
```

Лістинг файлу «UserRolesService.java»

```
package com.tntu.server.docs.core.services;

import
com.tntu.server.docs.core.data.exceptions.user.ActionOnAdminRoleExce
ption;
import
com.tntu.server.docs.core.data.exceptions.user.RoleNotFoundException
;
import
com.tntu.server.docs.core.data.exceptions.user.UserNotFoundException
;
import com.tntu.server.docs.core.data.models.user.RoleModel;
import com.tntu.server.docs.core.data.models.user.UserRoleModel;
import
com.tntu.server.docs.core.repositories.UserRolesModelRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;

@Service
public class UserRolesService {

    @Autowired
    private UserRolesModelRepository userRolesModelRepository;

    @Autowired
    private UserService userService;

    @Autowired
    private RoleService roleService;

    public List<UserRoleModel> streamUserRoles(long userId) throws
UserNotFoundException {
        userService.ensureExists(userId);
        return userRolesModelRepository.findByUserId(userId);
    }
```

```
    public void assignUserRole(long userId, long roleId)
            throws UserNotFoundException, RoleNotFoundException,
ActionOnAdminRoleException {
        userService.ensureExists(userId);
        roleService.ensureExists(roleId);
        ensureAllowed(roleId);
        if (!userRolesModelRepository.isExists(userId, roleId))
            userRolesModelRepository.saveAssign(userId, roleId);
    }

    public void removeAssignUserRole(long userId, long roleId)
            throws UserNotFoundException, RoleNotFoundException,
ActionOnAdminRoleException {
        userService.ensureExists(userId);
        roleService.ensureExists(roleId);
        ensureAllowed(roleId);
        if (userRolesModelRepository.isExists(userId, roleId))
            userRolesModelRepository.removeAssign(userId, roleId);
    }

    public List<RoleModel> getUserRoles(long userId) throws
UserNotFoundException, RoleNotFoundException {
        var roleIdList = streamUserRoles(userId)
                .stream()
                .map(UserRoleModel::getRoleId)
                .collect(Collectors.toList());
        var userRoles = new ArrayList<RoleModel>();

        for (Long id : roleIdList) {
            var role = roleService.getById(id);
            userRoles.add(role);
        }
        return userRoles;
    }

    private void ensureAllowed(long roleId)
            throws RoleNotFoundException, ActionOnAdminRoleException
{
        var role = roleService.getById(roleId);
        if (role.getName().equals(RoleModel.ADMIN))
            throw new ActionOnAdminRoleException();
    }

}
```

Лістинг файлу «RoleService.java»

```
package com.tntu.server.docs.core.services;

import
com.tntu.server.docs.core.data.exceptions.auth.CanNotCreateUserExcep
tion;
import
com.tntu.server.docs.core.data.exceptions.auth.LoginFailedException;
import
com.tntu.server.docs.core.data.exceptions.user.UserNotFoundException
```

```
;
import com.tntu.server.docs.core.data.models.user.UserModel;
import com.tntu.server.docs.core.repositories.UserModelRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import javax.mail.AuthenticationFailedException;
import java.util.List;

@Service
public class UserService {

    @Autowired
    private UserModelRepository userModelRepository;

    @Autowired
    private PasswordEncoder passwordEncoder;


    public UserModel login(String email, String password) throws
LoginFailedException {
        var user = userModelRepository
                .findLive(email)
                .orElseThrow(LoginFailedException::new);

        if (!passwordEncoder.matches(password,
user.getPasswordHash())) {
            throw new LoginFailedException();
        }
        return user;
    }

    public UserModel findActiveUser(long userId) throws
AuthenticationFailedException {
        return userModelRepository
                .findActive(userId)
                .orElseThrow(AuthenticationFailedException::new);
    }

    public UserModel findActiveUser(String username) throws
UserNotFoundException {
        return userModelRepository
                .findActive(username)
                .orElseThrow(UserNotFoundException::new);
    }

    public List<UserModel> getUsers() {
        return userModelRepository.getAll();
    }


    public boolean existsByEmail(String email) {
        return userModelRepository.existsByEmail(email);
    }

    public UserModel getUser(long userId) throws
UserNotFoundException {
        return userModelRepository
```

```
                .getUser(userId)
                .orElseThrow(UserNotFoundException::new);
    }

    public void ensureExists(long id) throws UserNotFoundException {
        if (!userModelRepository.existsById(id))
            throw new UserNotFoundException();
    }

    public UserModel createNewUser(UserModel newUserModel)
            throws CanNotCreateUserException {

        var user = userModelRepository.createUser(newUserModel);
        return user.orElseThrow(CanNotCreateUserException::new);
    }

}
```

Лістинг файлу «RoleService.java»

```
package com.tntu.server.docs.core.services;

import org.springframework.stereotype.Service;

import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;
import java.util.stream.Collectors;
import java.util.stream.Stream;

@Service
public final class SecureRandomService {
    private static final String ALGORITHM = "SHA1PRNG";


    public String generateDigits(int length) throws
NoSuchAlgorithmException {
        var random = SecureRandom.getInstance(ALGORITHM);
        var bound = (int) Math.pow(10, length) - 1;
        var code = random.nextInt(bound);

        return String.format("%0" + length + "d", code);
    }

    public String generateAlphaNumeric(int length) throws
NoSuchAlgorithmException {
        var random = SecureRandom.getInstance(ALGORITHM);
        var bound = (int) 'z' + 1;

        return Stream
                .generate(() -> (char) random.nextInt(bound))
                .filter(x -> (x >= '0' && x <= '9') || x >= 'a' ||
(x >= 'A' && x <= 'Z'))
                .limit(length)
                .map(String::valueOf)
                .collect(Collectors.joining());
    }
}
```