

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

магістр

(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ ТЕХНОЛОГІЙ
ОПЕРАЦІЙНОГО ПЕРЕТВОРЕННЯ ДЛЯ СИСТЕМИ СПІЛЬНОГО
РЕДАГУВАННЯ ДОКУМЕНТІВ В РЕАЛЬНОМУ ЧАСІ
RESEARCH AND SOFTWARE IMPLEMENTATION OF OPERATIONAL
TRANSFORMATION TECHNOLOGIES FOR THE SYSTEM OF REAL-TIME
COLLABORATIVE EDITING OF DOCUMENTS

Виконав(ла): студент(ка) 2 курсу, групи КНТ-112м
Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

БОРТНІК В.С.

(ПРИЗВИЩЕ та ініціали)

Керівник ГЛАДКОВА О.М.

(ПРИЗВИЩЕ та ініціали)

Рецензент ЗЕЛЕНЬОВА І.Я.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти магістр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
“ ” 2023 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

БОРТНИКА Владислава Сергійовича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Дослідження та програмна реалізація технологій операційного перетворення для системи спільного редагування документів в реальному часі. Research and Software Implementation of Operational Transformation Technologies for the System of Real-Time Collaborative Editing of Documents

керівник проєкту (роботи) к.т.н., ГЛАДКОВА Ольга Миколаївна
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “ 14 ” листопада 2023 року № 437

2. Строк подання студентом проєкту (роботи) 01 грудня 2023 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області та постановка завдань дослідження. 2. Матеріали і методи. 3. Розробка програми. 4. Розробка архітектури програми. 5. Експлуатація та тестування розробленої програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-5 Основна частина	ГЛАДКОВА О.М., доцент		
Нормоконтролер	БЄЛОВА А.В., асистент		

7. Дата видачі завдання “ 08 ” вересня 2023 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	2-3 тижні	Розділ 1
3	Розробка та удосконалення методів, моделей й алгоритмів вирішення задачі.	4-5 тижні	Розділ 2
4	Розробка архітектури програми.	6 тиждень	Розділ 3
5	Розробка програми.	7-8 тижні	Розділ 4
6	Тестування та експериментальне дослідження програмного забезпечення.	9 тиждень	Розділ 5
7	Оформлення пояснювальної записки та документів до неї.	10-11 тижні	Додатки
8	Нормоконтроль та рецензування.	12 тиждень	
9	Захист роботи.	12 тиждень	

Студент(ка)

(підпис) Владислав БОРТНІК
(Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

(підпис) Ольга ГЛАДКОВА
(Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра:
109 с., 4 табл., 31 рис., 3 дод., 18 джерел.

ТЕХНОЛОГІЇ ОПЕРАЦІЙНОГО ПЕРЕТВОРЕННЯ, СИСТЕМИ
СПІЛЬНОГО РЕДАГУВАННЯ ДОКУМЕНТІВ, JAVASCRIPT, EXPRESS.JS.

Об'єкт дослідження – процес редагування документів в реальному часі.

Предмет дослідження – програмне забезпечення для системи спільного редагування документів в реальному часі.

Мета роботи – розробка програмного забезпечення для ефективного управління та редагування документів у спільному режимі реального часу.

Матеріали, методи та технічні засоби: мова програмування JavaScript, бібліотека React для розробки інтерфейсу, фреймворк Express.js для створення серверної частини, система управління базами даних MongoDB, бібліотека Socket.io для реалізації взаємодії в реальному часі, текстовий редактор Quill для забезпечення спільного редагування.

Результати. Розроблено програмне забезпечення, що забезпечує можливість спільного редагування документів в реальному часі, користувачі можуть одночасно вносити зміни в документ, спостерігаючи за їхніми діями у режимі реального часу. Реалізовано систему операційного перетворення для вирішення конфліктів редагування.

Висновки. Реалізований вебзастосунок надає зручні та ефективні інструменти для спільного редагування документів в реальному часі, забезпечуючи високий рівень взаємодії користувачів та уникнення конфліктів при одночасному редагуванні.

Галузь використання. Спільне редагування документів для забезпечення ефективної співпраці в реальному часі в різних галузях.

ABSTRACT

Explanatory note to the diploma qualifying work of the master: 109 pages, 4 tables, 31 figures, 3 appendixes, 18 sources.

OPERATIONAL TRANSFORMATION TECHNOLOGIES, DOCUMENT COLLABORATIVE EDITING SYSTEM, JAVASCRIPT, EXPRESS.JS.

The research focuses on the real-time collaborative document editing system.

The study delves into the software designed to facilitate simultaneous editing of documents, ensuring seamless collaboration in real-time.

The primary objective is the development of software dedicated to managing and optimizing the collaborative editing of documents in real-time.

Materials, methods, and technical tools employed include the JavaScript programming language, React library for interface development, Express.js framework for the server-side, MongoDB as the database management system, Socket.io library for real-time interaction, and the Quill text editor for collaborative editing functionalities.

Outcomes. The developed software provides a robust platform for collaborative document editing in real-time, enabling multiple users to make concurrent changes to a document while observing each other's actions in real-time. The system incorporates operational transformation techniques to handle and resolve editing conflicts effectively.

Conclusions. The resulting web application offers convenient and efficient tools for real-time collaborative document editing, ensuring high levels of user interaction and preventing conflicts during simultaneous edits.

Field of Application. Facilitating collaborative document editing across various domains, such as team collaboration, scientific research, and editorial activities.

ЗМІСТ

	С.
Перелік скорочень та умовних позначок	8
Вступ.....	9
1 Аналіз предметної області та постановка завдань дослідження.....	11
1.1 Вимоги та функціонал системи.....	11
1.2 Огляд існуючих аналогів	12
1.3 Огляд методів.....	14
1.4 Порівняння методів	15
1.5 Висновки за розділом 1	17
2 Матеріали і методи	19
2.1 Метод Операційного перетворення	19
2.2 Безпека та Аутентифікація з використанням JWT та Bcrypt.....	20
2.3 Висновки за розділом 2	21
3 Розробка програми	22
3.1 Функціональні вимоги до програмного забезпечення.....	22
3.2 Вибір засобів розробки програмного продукту.....	23
3.3 Проектування бази даних	34
3.4 Обмеження моделі спілкування «запит-відповідь»	35
3.5 Способи реалізації real-time сервісу	36
3.6 Вебсокети	38
3.7 Висновки за розділом 3	39
4 Розробка архітектури програми	41
4.1 Алгоритм функціонування програмного забезпечення	41
4.2 Структура реалізованого програмного забезпечення	43
4.3 Опис прийнятих під час реалізації рішень.....	46
4.4 Висновки за розділом 4.....	47
5 Експлуатація та тестування розробленої програми	48
5.1 Призначення програми.....	48

5.2 Умови виконання програми.....	48
5.3 Виконання програми	48
5.4 Висновки за розділом 5	54
Висновки.....	55
Перелік джерел посилання	57
Додаток А Технічне завдання.....	59
Додаток Б Текст програми.....	63
Додаток В Слайди презентації	102

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

- API – Application Programming Interface;
- CSS – Cascading Style Sheets;
- JSON – JavaScript Object Notation;
- HTML – HyperText Markup Language;
- MERN – (MongoDB, Express.js, React, Node.js);
- OT – Operational Transformation (операційне перетворення);
- URL – Uniform Resource Locator;
- БД – база даних;
- ПЗ – програмне забезпечення.

ВСТУП

У світі сучасного бізнесу, де ефективність та швидкість є ключовими факторами успіху, важливо мати в арсеналі інструменти, що спрощують та оптимізують робочі процеси. Для досягнення цієї мети вивчається тема "Дослідження та програмна реалізація технологій операційного перетворення для системи спільного редагування документів в реальному часі".

У сучасних умовах використання системи спільного редагування документів стає важливим елементом, що розробляється з урахуванням потреб сучасних підприємств. Ці системи не лише дозволяють зручно та ефективно обробляти документи, але й автоматизують ряд бюрократичних процесів.

Основна мета систем спільного редагування документів полягає в полегшенні та покращенні взаємодії у колективі, проте їхнє значення не обмежується цим. Вони стають ключовими інструментами для керування внутрішніми процесами та ресурсами підприємства.

Такі системи дозволяють зберігати інформацію про проекти, історію взаємодії з документами, враховувати внески та потреби учасників. Це покращує якість співпраці та створює більш інтерактивний досвід. Крім того, вони сприяють зберіганню, організації та обміну документами всередині підприємства. Це спрощує управління інформацією та зменшує ризики втрати важливих документів, сприяючи автоматизації бюрократичних процесів, таких як відправка повідомлень, обробка замовлень, планування завдань тощо, зменшуючи тим самим трудомісткість для співробітників та ризики помилок.

Системи спільного редагування документів є надійними інструментами управління на різних рівнях та використовуються у всіх сферах бізнесу, допомагаючи підвищити продуктивність, поліпшити співпрацю та зекономити час та ресурси. Вибір конкретного типу системи залежить від структури та технічного забезпечення підприємства.

Обираючи вебсистему спільного редагування документів для підприємства, створюється можливість для розвитку та масштабування

бізнес–процесів. Рішення призначене для оптимізації та спрощення багатьох аспектів діяльності. Незалежно від розміру підприємства та галузі, впровадження системи спільного редагування документів може стати ключовим елементом економічного успіху, підвищуючи продуктивність бізнесу, роблячи його більш конкурентоспроможним і забезпечуючи ефективне управління ресурсами.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАНЬ ДОСЛІДЖЕННЯ

У даному розділі буде розглянуто необхідні кроки, які передують обробці тексту в режимі реального часу. Описано те яким чином буде реалізовано обробку даних та їх синхронізацію. Буде порівняно методи реалізації синхронізації тексту та обрано один із них.

1.1 Вимоги та функціонал системи

Цей розділ присвячений визначенню ключових вимог та функціональних можливостей, які повинна виконувати розроблювана система спільного редагування документів. Враховуючи потреби сучасних малих та середніх бізнесів, система повинна бути гнучкою та адаптивною до різних стилів управління та організаційних структур. Нижче перелічені основні вимоги до системи:

Простий та інтуїтивно зрозумілий інтерфейс. Система повинна мати зрозумілий користувацький інтерфейс, що знижує поріг входження для нових співробітників та сприяє швидкому ознайомленню з основним функціоналом.

Закритість системи. Система повинна передбачати механізми авторизації та додавання нових користувачів через спеціальний відділ.

Інтерфейси перегляду профілів. Система повинна надавати можливість перегляду профілів користувачів для ідентифікації та уточнення їхніх даних.

Операційне перетворення та автоматизація. Система спрощує та автоматизує операції, пов'язані зі створенням, обігом та зберіганням документів в режимі реального часу. Це включає в себе автоматичне оновлення та синхронізацію змін усіх учасників, сприяючи ефективному обміну та спільній роботі.

Гнучкість та масштабованість. Система розроблена з урахуванням гнучкості, щоб вона могла легко адаптуватися під різні вимоги бізнесу. Вона

також масштабована, забезпечуючи зручне зростання разом із розвитком компанії, не втрачаючи ефективності.

Ці вимоги та функціональні особливості є основою для розробки системи спільного редагування документів в реальному часі, яка відповідає вимогам і надає сучасним підприємствам інструмент для оптимізації робочих процесів та підвищення ефективності управління.

1.2 Огляд існуючих аналогів

На ринку існує ряд комерційних реалізацій систем спільного редагування документів, що використовують технології операційного перетворення для забезпечення ефективної співпраці та редагування в реальному часі. Розглянемо кілька провідних платформ з цієї категорії.

Google Docs [1] (рис. 1.1) – одна з популярних платформ для спільного редагування документів в режимі реального часу. *Google Docs* дозволяє користувачам одночасно вносити зміни до документів, автоматично вирішуючи конфлікти та забезпечуючи синхронізацію змін.

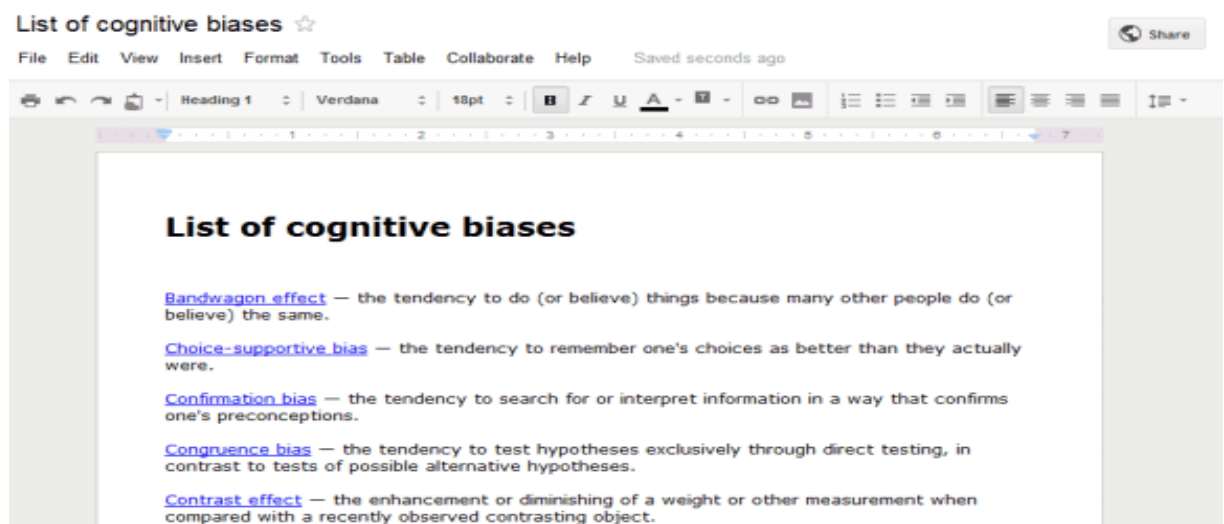


Рисунок 1.1 – Google docs

Основні функції редагування текстових документів, аркушів та презентацій в режимі реального часу, можливість спільної роботи з

документами через Інтернет, а також коментування та взаємодію в чаті. Це забезпечує безперервну комунікацію та співпрацю між користувачами незалежно від їхнього місцезнаходження.

Microsoft 365 [2] (рис. 1.2) – Microsoft 365, включаючи інструменти, такі як Word, Excel та PowerPoint, також використовує технології операційного перетворення для сприяння спільному редагуванню документів в реальному часі. Забезпечуючи інтеграцію з хмарними службами, Microsoft 365 дозволяє користувачам спільно працювати над проектами та обмінюватися ідеями, надаючи доступ до актуальних даних у режимі реального часу.

Основні функції включають спільне редагування текстових документів, створення таблиць та презентацій, а також можливість проводити віртуальні наради та обговорення за допомогою вбудованих інструментів.

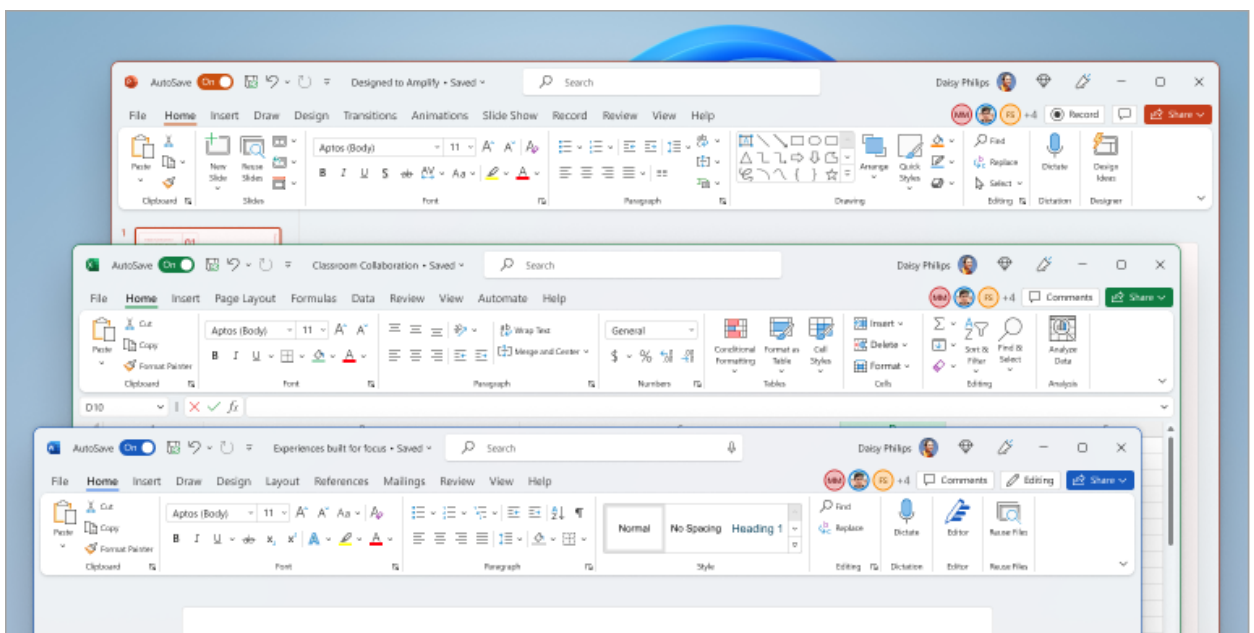


Рисунок 1.2 – Microsoft 365

Notion [3] (рис. 1.3) – це комплексна платформа для спільного редагування документів, яка використовує передові технології операційного перетворення. На відміну від традиційних офісних пакетів, Notion дозволяє користувачам створювати структуровані документи, організовувати завдання та проекти, а також спільно працювати над ними в режимі реального часу.

Основні функції включають можливість створення спільних дошок, баз даних та нотаток, а також інтеграцію з іншими інструментами та додатками. Notion надає можливість ефективно використовувати технології операційного перетворення для організації спільної роботи над різними типами документів та завдань.

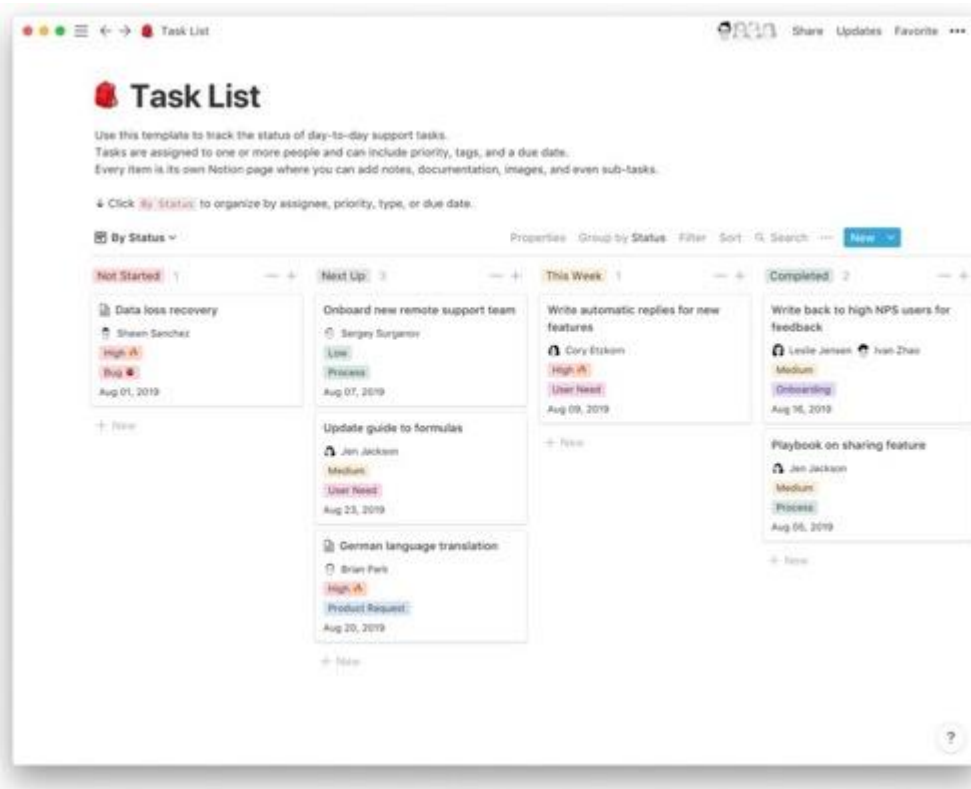


Рисунок 1.3 – Notion

1.3 Огляд методів

Сучасні редактори тексту для систем спільного редагування документів в реальному часі використовують різноманітні методи та техніки для ефективної синхронізації та управління змінами, внесеними різними користувачами. Деякі з основних методів включають.

Операційне перетворення (Operational Transformation, OT) [4;5]: важливий метод для вирішення конфліктів та синхронізації змін. Кожен користувач має локальну репліку документа та історії змін. При внесенні змін

користувач вирішує конфлікти та використовує ОТ для синхронізації своєї історії змін з іншими користувачами. ОТ визначає порядок застосування змін до копій документів усіх учасників, щоб уникнути конфліктів. Крім того, він використовується для ефективного керування курсорами та відзначеннями користувачів.

Техніка "Купюра" (Causal Trees): Використання каузальних дерев дозволяє визначати порядок змін у документі та ефективно обробляти конфлікти. Кожна зміна має унікальний ідентифікатор та вказує на попередні події, що їх спричинили. Це допомагає системі ефективно визначати каузальний порядок подій та обробляти конфлікти.

Системи з Прогнозуванням Конфліктів: використовують алгоритми прогнозування для визначення можливих конфліктів та їх уникнення. Алгоритми прогнозування можуть виявити можливі конфлікти та намагатися їх уникнути, що забезпечує ефективну синхронізацію.

Ці методи та техніки спільно забезпечують роботу редакторів тексту в реальному часі, надаючи високу ефективність та коректність при спільному редагуванні документів.

1.4 Порівняння методів

Усі розглянуті методи мають як сильні так і слабкі сторони. Нижче, у таблиці 1.1 наведено порівняння цих методів.

Таблиця 1.1 – Порівняння методів

Критерії/Методи	ОТ	Техніка "Купюра"	Техніка "Лямбда– архітектура"	Системи з Прогнозуванням Конфліктів
Швидкість синхронізації змін	Висока	Помірна	Висока	Помірна
Забезпечення цілісності даних	Так	Так	Так	Так
Складність реалізації	Висока	Помірна	Висока	Помірна
Підтримка оптимістичної синхронізації	Так	Так	Так	Так
Метод вирішення конфліктів	За правилами	За версіями	—	—
Продуктивність при великій кількості користувачів	Висока	Помірна	Висока	Помірна
Популярність у веб–розробці	Висока	Обмежена	Обмежена	Обмежена

Беручи до уваги всі переваги і недоліки перелічених вище методів, я зупинився на методі Операційних перетворень, нижче наведені переваги такого рішення:

Реальний час та спільне редагування. ОТ спеціально розроблене для сценаріїв реального часу та спільного редагування. З огляду на те, що MERN–застосунки можуть включати функціонал редакторів тексту або колаборативних платформ, де багато користувачів можуть одночасно редагувати дані, ОТ надає ефективний механізм управління конфліктами та забезпечення консистентності даних в режимі реального часу.

Ефективне керування конфліктами. ОТ дозволяє ефективно вирішувати конфлікти при одночасному редагуванні декількох користувачів. Враховуючи порядок введення операцій, ОТ автоматично вирішує конфлікти та забезпечує правильність результатів.

Легка інтеграція. ОТ може бути легко інтегроване в MERN–застосунки завдяки своїй гнучкості та можливості адаптації до конкретних потреб проекту. Наявність багатьох бібліотек та рішень, що використовують ОТ, у сфері JavaScript полегшує його впровадження в середовищі MERN.

Підтримка оптимістичної синхронізації. ОТ може бути ефективно використане для оптимістичної синхронізації, що покращує відчуття реакції та продуктивність для користувачів. Цей підхід дозволяє вносити зміни локально та пізніше синхронізувати їх з сервером.

Застосування в колаборативних інструментах. ОТ є популярним вибором для колаборативних інструментів, таких як графічні або текстові редактори, де важливо забезпечити спільне редагування та уникнути конфліктів при одночасних змінах.

Використання ОТ у MERN–застосунках дозволяє створювати ефективні та гнучкі системи для спільного редагування даних в режимі реального часу.

1.5 Висновки за розділом 1

У цьому розділі був проведений аналіз ринку спрямований на визначення стратегічних напрямків для досягнення поставленої мети – створення розподіленої системи спільного редагування документів в реальному часі. Аналіз дозволив виокремити ключові принципи дизайну та технічної підтримки подібних систем, які будуть впроваджені для поліпшення інтерфейсу користувачів та оптимізації адміністративних процесів у розроблюваному програмному продукті.

Основні завдання дипломної роботи включають:

- глибокий аналіз предметної області;
- розробку концепції програмного забезпечення;
- створення функціонального програмного продукту.

Отже, на основі проведеного аналізу було визначено конкретні кроки для подальшого вдосконалення та реалізації програмного продукту. Використання основних принципів дизайну та операційного перетворення сприятиме створенню продукту, який відповідатиме вимогам інноваційного ринку та відповідатиме потребам користувачів у сфері спільного редагування документів в режимі реального часу.

2 МАТЕРІАЛИ І МЕТОДИ

2.1 Метод Операційного перетворення

Операційне Перетворення (Operational Transformation, OT) – це техніка, яка використовується для синхронізації та координації різних версій документів або об'єктів в розподілених системах, де можливі паралельні редагування. Основний принцип полягає у трансформації (зміненні) одних операцій, зроблених в одній копії документа, таким чином, щоб їх можна було правильно застосувати до іншої копії, забезпечуючи збереження семантики та консистентності. Нижче наведені головні принципи його роботи:

- векторні годинники: кожна операція отримує свій власний унікальний "векторний годинник" - це щось схоже на позначку часу для операцій, векторні годинники використовуються для визначення порядку операцій у часі та уникнення непорозумінь;
- трансформації операцій: коли виявляється, що дві операції конфліктують (одночасні редагування), вони не конфліктують з суттєвою інтелектуальністю, замість цього вони "трансформуються", що означає, що їх змінюють так, щоб вони правильно взаємодіяли, що дозволяє уникнути суперечностей;
- аплікація операцій: кожна операція аплікується до локальної копії документа, що робиться в порядку, визначеному векторним годинником, що гарантує відтворення операцій у правильному порядку;
- забезпечення консистентності: головна мета – зберегти консистентність даних під час паралельних редагувань, що означає, що, навіть якщо багато користувачів редагують документ одночасно, результат буде семантично та логічно консистентним.

Отже, якщо два користувачі редагують одну статтю, вони обидва мають копії документа та роблять зміни. Якщо їх зміни конфліктують, OT перетворює одну з операцій, щоб уникнути суперечностей. Коли обидва користувачі аплікують свої зміни, документ залишається консистентним.

У MERN застосунку, де паралельні редагування можуть відбуватися на клієнті та сервері, OT є доречним механізмом для забезпечення синхронізації даних між всіма складовими системи. Він допомагає уникати конфліктів та зберігає консистентність інформації, що є критичним для реалізації реального часу та спільного редагування документів.

2.2 Безпека та Аутентифікація з використанням JWT та Bcrypt

JWT (JSON Web Tokens) [6] – це потужний механізм для забезпечення безпеки та автентифікації в розподілених застосунках MERN. Основна ідея полягає в тому, щоб створити токен, який може бути використаний для автентифікації користувача при кожному запиті.

Bcrypt [7] – це хеш-функція для хешування паролів із солью (salt) з великими обчислювальними витратами, що робить процес хешування більш безпечним від звичайних хеш-функцій. Bcrypt спеціально розроблений для захисту від атак із використанням "регенерації" (brute-force) і "радужниці" (rainbow table). Його великі обчислювальні витрати ускладнюють процес відновлення пароля, навіть якщо хеш виявиться вкраденим.

В розробці MERN застосунку використання JWT [6] (JSON Web Tokens) та bcrypt грає важливу роль у забезпеченні безпеки та автентифікації.

Після того, як користувач успішно ввійшов на сервері, генерується JWT. Цей токен містить інформацію про користувача та деякі метадані. Щоб гарантувати його валідність, він підписується за допомогою секретного ключа сервера. Після цього, клієнт отримує токен і може зберігати його, наприклад, в localStorage або cookies.

При кожному наступному запиті, клієнт передає токен у заголовок або в cookie. Сервер перевіряє та розшифровує токен, перевіряючи підпис та термін його дії.

Щоб уникнути проблем із токенами, їх треба оновлювати. Клієнт відправляє запит на оновлення, використовуючи попередній токен.

При реєстрації нового користувача його пароль хешується за допомогою bcrypt, і лише хеш зберігається в базі даних, а не сам пароль.

Bcrypt робить обчислення хешу багато разів, ускладнюючи атаки злому шляхом перебору.

Використовуючи унікальний "сільний" хеш для кожного паролю, bcrypt ускладнює атаки рейнбоу-таблицями. Адміністратор може легко оновити хеші користувачів, якщо в майбутньому виходять нові версії bcrypt.

2.3 Висновки за розділом 2

Застосування методу Операційного перетворення та механізмів безпеки на базі JWT [6] та Bcrypt [7] є важливою частиною розробки MERN застосунків. OT дозволяє уникнути конфліктів та зберігати консистентність при паралельних редагуваннях, а JWT з Bcrypt гарантують надійну аутентифікацію та безпеку обробки паролів. Ці техніки в сукупності зроблять ваш MERN застосунок надійним, безпечним та готовим для використання в реальних умовах.

3 РОЗРОБКА ПРОГРАМИ

3.1 Функціональні вимоги до програмного забезпечення

На основі аналізу аналогів та огляду загальноновживаних принципів побудови схожих систем в проєкті не було передбачено виділення ролей користувачів, всі користувачі мають однаковий рівень прав. Нижче наведено функціональні вимоги для користувача:

- можливість перегляду власного профілю;
- можливість оновлення інформації власного профілю;
- можливість перегляду власних документів;
- можливість створення документів;
- можливість оновлення, перейменування та видалення власних документів;
- можливість перегляду власного профілю;
- можливість реєстрації;
- можливість входу у особистий кабінет;
- можливість одночасного редагування документа кількома користувачами;
- можливість автозбереження документа;
- можливість редагування розміру, товщини, стилю шрифту та інший функціонал пов'язаний із редагуванням тексту;
- можливість використання undo, redo під час редагування документа;
- можливість друку документа.

Користувач являється головною сутністю для налаштування та забезпечення ефективної роботи системи. Користувач має розширений доступ до всієї інформації та процесів. Сценарій використання сервісу користувачем вказано на рис 3.1.

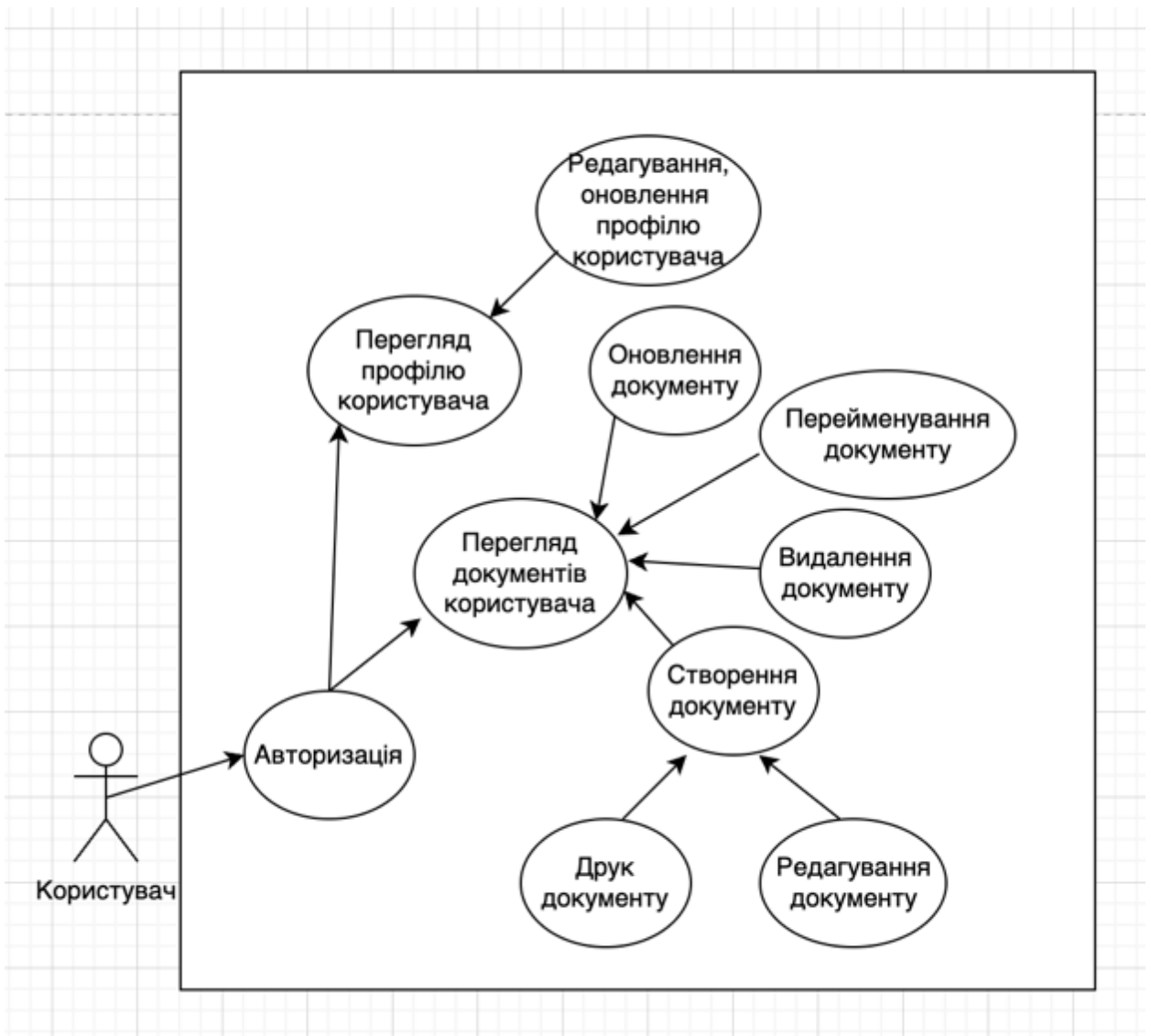


Рисунок 3.1 – Сценарій використання сервісу користувачем

3.2 Вибір засобів розробки програмного продукту

Перед тим, як розпочати розробку, був проведений ретельний аналіз доступних технологій та мов програмування для вибору оптимального варіанта для серверної частини системи спільного редагування документів в реальному часі. В результаті цього аналізу було вирішено використовувати JavaScript та зокрема веб-фреймворк Express для серверної реалізації.

Вибір в підтримку Express обумовлений його високою продуктивністю, широким спектром функціоналу та популярністю у спільноті розробників. Express є легким та гнучким веб-фреймворком, що надає функціонал для

швидкої розробки веб–додатків. Його висока швидкодія та простота використання роблять його ідеальним вибором для мого проекту.

Express [8] є мінімалістичним фреймворком, який надає лише необхідні інструменти для побудови ефективних веб–додатків. Він дозволяє швидко створювати серверні додатки, забезпечуючи при цьому високу продуктивність. Його велика популярність у спільноті розробників забезпечує широку підтримку.

Навпаки, Nest.js, хоч і є потужним фреймворком для розробки серверних додатків на мові TypeScript, має більше важливих компонентів та складніший підхід, що робить його менш оптимальним вибором для даного проекту, орієнтованого на швидку інтеграцію.

Таким чином, використання Express [8] в рамках розробки нашого проекту буде сприяти швидкій інтеграції, ефективній реалізації операційного перетворення та створенню стабільної та продуктивної системи спільного редагування документів в реальному часі. Порівняння технологій подано у таблиці 3.1.

Таблиця 3.1 – Порівняння Express.js та Next.js

Характеристика	Express.js	Nest.js
Тип фреймворку	Мінімалістичний веб–фреймворк	Прогресивний Node.js фреймворк
Залежності від зовнішніх бібліотек	Мінімальні	Залежні від конкретних вимог, може використовувати додаткові модулі
Вбудовані компоненти	Обмежені, що дозволяє вибір додаткових компонентів	Забезпечує структуровану архітектуру з вбудованими компонентами

Продовження таблиці 3.1

Характеристика	Express.js	Nest.js
Підхід до розробки	Гнучкий, підходить для різноманітних завдань	Стандартизований, сприяє швидкій розробці за певними конвенціями
Підтримка баз даних	Підтримка через розширення	Вбудована підтримка від Nest.js з використанням TypeORM або інших ORM
Використання шаблонів	Зазвичай використовується для рендерингу HTML	Більше орієнтований на використання API, але можливість використання шаблонів існує
Форма розробки	Забезпечує вільний вибір технологій та структури	Забезпечує консистентну структуру та підтримує TypeScript
Система адміністрування	Залежить від вибору розширень	Підтримує вбудовану систему адміністрування через Nest.js Admin Module
Розширення та плагіни	Велика кількість розширень доступна	Залежить від вибору, менше плагінів порівняно з Express
Виконання фонових завдань	Через додаткові розширення	Має вбудовану підтримку фонових задач через Nest.js Queues
Специфікації та документація	Велика кількість документації та специфікацій	Добра документація, але менше порівняно з Express
Архітектура	Менш жорстко визначена, залишає вибір архітектури	Забезпечує структуровану архітектуру з використанням модулів та сервісів

Кінець таблиці 3.1

Характеристика	Express.js	Nest.js
Архітектура	Менш жорстко визначена, залишає вибір архітектури розробнику	Забезпечує структуровану архітектуру з використанням модулів та сервісів
Спільнота розробників	Широка та активна спільнота	Активна, але менша порівняно з Express
Швидкодія	Дуже швидкий та легкий	Швидкий, але більше витрати ресурсів порівняно з Express для простих додатків
Інтеграція з іншими сервісами	Легка інтеграція через Middleware	Можливість використання Middleware, але з більшою фокусуванням на структурованій архітектурі

Це порівняння допомогло мені визначити, який з фреймворків (Express чи Nest.js) краще підходить для проекту з реалізації системи спільного редагування документів в реальному часі.

У контексті клієнтської частини проекту, я роздивлявся варіанти використання Express views або спеціалізований фреймворк. Було ухвалено рішення використовувати спеціалізований фреймворк, а саме React. Цей підхід дозволяє забезпечити більш гнучку і універсальну взаємодію з користувачем, враховуючи широкі можливості React у створенні динамічних веб-сторінок. Порівняння наведено у табл 3.2.

Таблиця 3.2 – Порівняння React.js та Express.js views

Характеристика	Express.js Views	React.js
Тип фреймворку	Веб-фреймворк для Node.js	JavaScript бібліотека для UI
Синтаксис та Верстка	HTML та Embedded JavaScript	JSX (JavaScript XML)
Управління Станом	Менше підтримка, більше серверно-орієнтований	Декларативний підхід до стану, однонаправлене потік даних, використання стейту та пропсів
Компоненти	Статичні та динамічні шаблони, менше компонентного підходу	Компоненти як основна одиниця будівельного блоку, реюз та модульність
Взаємодія з DOM	Вбудований в шаблони та контролери	Віртуальний DOM, ефективні зміни через diffing
Життєвий цикл компонентів	Не застосовується, більше структурований на рівні шаблонів та контролерів	Є власний життєвий цикл, методи які викликаються при різних подіях
Спосіб Оптимізації	Кешування та мемоізація шаблонів, асинхронний рендеринг	Віртуальний DOM, оптимізація шляхом часткового оновлення компонентів

Продовження таблиці 3.2

Характеристика	Express.js Views	React.js
Реактивність	Менше реактивний, зазвичай потребує ручного оновлення сторінки	Реактивний за замовчуванням, оновлення відбуваються автоматично при зміні стану
Відокремленість фронтенду та бекенду	Використовується для серверного рендерингу та виведення шаблонів	Рекомендовано для SPA та відокремленого фронтенду
Інструменти та Екосистема	Більше спрямований на бекенд, менше інструментів для реактивного UI	Розширена екосистема для фронтенду, широкий вибір інструментів та бібліотек
Повторне використання компонентів	Менше структурований для реюзу, більше фокус на серверному рендерингу	Великий фокус на повторному використанні компонентів та композиції
Відлагодження	Зазвичай простий, стандартні інструменти для Node.js	Розширена система відлагодження, велика підтримка в DevTools

React.js [9] є важливою бібліотекою для розробки інтерфейсів, яка вносить унікальні функціональні можливості як в клієнтську, так і веб-додатків. Ця бібліотека служить ключем до створення динамічних та ефективних інтерактивних веб-сторінок, які доповнюють основну структуру HTML та стиль CSS. У якості UI-бібліотеки було обрано MUI [10].

React.js дозволяє легко організовувати веб-додатки за допомогою компонентного підходу, де інтерфейс розбивається на невеликі, самостійні компоненти. Це дозволяє впроваджувати взаємодію та динаміку веб-сторінок, забезпечуючи кращу швидкодію та покращений користувацький досвід.

React.js [9] перетворює веб-додатки, дозволяючи створювати захоплюючі функції, такі як реактивні компоненти, віртуальний DOM для оптимізації оновлень інтерфейсу, а також можливості для легкого взаємодії з сервером, наприклад, за допомогою асинхронних запитів. Це робить React.js незамінною частиною сучасної веб-розробки, де важливо надати користувачам захопливий та ефективний веб-досвід.

Під час аналізу систем управління базами даних розглядалися два основних кандидати: PostgreSQL та MongoDB [11]; [12].

PostgreSQL, як відомо, представляє собою об'єктно-реляційну систему управління базами даних з унікальними можливостями, такими як успадкування таблиць та перевантаження функцій, що можуть бути корисні для вирішення різних завдань. Особливо важливою є його строга відповідність стандартам SQL, що може виявитися критичним у певних сценаріях.

MongoDB [12] є інноваційною системою управління базами даних, відомою своєю гнучкістю та спрощеним підходом до зберігання та обробки даних. Вона відзначається нереляційною моделлю даних, що дозволяє зберігати дані у вигляді документів у форматі BSON (Binary JSON). Це надає розробникам велику свободу у створенні та модифікації структури даних без необхідності визначення строгих схем, що робить MongoDB ідеальним рішенням для проектів, які можуть еволюціонувати та змінюватися з часом.

Нижче у табл. 3.3 було порівнянню ці дві системи управління базами даних.

Таблиця 3.3 – Порівняння PostgreSQL та MongoDB

Характеристика	MongoDB	PostgreSQL
Тип бази даних	NoSQL, орієнтована на документи	SQL, реляційна база даних
Мова запитів	JSON-подібний, запити на основі JavaScript	SQL, декларативна мова запитів
Транзакції	Підтримка транзакцій з MongoDB 4.0	Повна підтримка ACID-транзакцій
Структура даних	Гнучка, може мати різні поля для кожного документа	Жорстка, структурована таблиця зі зв'язками
Схема даних	Динамічна схема, може змінюватись з часом	Статична схема, визначається перед розгортанням
Запити JOIN	Не застосовується, дані зазвичай вбудовані в документи	Підтримка JOIN для об'єднання даних з різних таблиць
Горизонтальне масштабування	Легше горизонтально масштабується, додаючи нові сервери	Легше вертикально масштабується, додаючи більш потужне обладнання
Консистентність та Доступність	За замовчуванням більше спрямований на доступність, менше консистентний	За замовчуванням більше спрямований на консистентність, менше доступний при відмовах
Підтримка ACID-властивостей	Менше строга підтримка ACID	Повна підтримка ACID
Індексація та Запити	Добре підтримує індексацію та операції читання	Сильна підтримка індексації та операцій читання/запису

Продовження таблиці 3.3

Характеристика	MongoDB	PostgreSQL
Географічні можливості	Сильна підтримка географічних запитів та індексів	Підтримка географічних об'єктів і операцій
Використання великих об'ємів даних	Добре підходить для великих об'ємів документів	Ефективне використання для великих об'ємів структурованих даних
Вартість та Ліцензія	Відкритий код, безкоштовний, але є комерційні пакети для підтримки	Відкритий код, безкоштовний, з вільною ліцензією
Інтеграція з Іншими Інструментами	Добре інтегрується з JavaScript-фреймворками та іншими NoSQL базами даних	Широкі можливості інтеграції з різними інструментами та платформами

Після порівняння цих двох систем керування базами даних, я надав перевагу саме MongoDB.

MongoDB відмінно працює з даними у форматі JSON, що робить його ідеальним для застосувань, де важливий обмін даними та взаємодія з мовами програмування. У порівнянні з традиційними реляційними базами даних, MongoDB відзначається високою швидкістю завдяки використанню гнучкої індексації та можливості горизонтального масштабування, що дозволяє обробляти великі обсяги даних та запитів.

Однією з ключових переваг MongoDB [12] є його здатність працювати в режимі реального часу та підтримувати велику кількість одночасних записів та читань. Це особливо важливо для застосунків, де швидкість та масштабованість грають вирішальну роль.

Ще однією важливою рисою MongoDB [12] є його можливості в обробці великих обсягів даних та гнучкість в роботі з сучасними великими наборами даних, що включають неструктуровані або напівструктуровані дані.

Окрім того, MongoDB має потужну спільноту та широкий екосистема інструментів, які полегшують розробку та адміністрування баз даних. Серед інших переваг можна відзначити можливість гнучкого горизонтального масштабування за допомогою шардування та можливість роботи в розподілених середовищах.

MongoDB є важливим інструментом для сучасних розробників, які цінують гнучкість, ефективність та можливості роботи з різноманітними типами даних у реальному часі.

MongoDB виходить вперед на ринку баз даних завдяки кільком перевагам, які враховують особливості нереляційних систем управління базами даних (NoSQL):

- гнучкість схеми: MongoDB володіє динамічною схемою, що означає, що документи в колекції можуть мати різні поля, що надає більшу гнучкість у порівнянні з реляційними базами даних, де структура таблиць строго визначена;
- швидкість розробки: MongoDB дозволяє розробникам швидше пристосовуватись до змін в вимогах, оскільки вони можуть додавати нові поля до документів без необхідності змінювати всю схему бази даних;
- гарна масштабованість: MongoDB розроблено для горизонтального масштабування, що означає, що його можна легко розширювати за допомогою додаткових серверів для обробки збільшених обсягів даних;
- повнотекстовий пошук: MongoDB надає можливості повнотекстового пошуку, що дозволяє ефективно здійснювати пошук у текстових полях навіть за допомогою складних запитів;

- гнучкість сховища: MongoDB може легко зберігати різноманітні дані, такі як вбудовані масиви та документи, що відображає природу сучасних додатків з різними типами даних.

Отже, MongoDB надає баланс між гнучкістю, швидкістю розробки та масштабованістю, що робить його привабливим для розробки сучасних додатків, особливо тих, які вимагають швидких змін у структурі даних та обробки великих обсягів даних.

Функціонал вебсайту включає в себе забезпечення управління документами й їхнє редагування. Основна мета це організувати ефективний контроль і редагування документів декількома користувачами одночасно. Це вимагає ефективного впровадження механізму одночасного редагування документів. Були розглянуті наступні технології для реалізації цього функціоналу та перелічено їх переваги:

Переваги Socket.IO–Client:

- двосторонній зв'язок: Socket.IO надає ефективний засіб для встановлення двостороннього зв'язку між сервером і клієнтами, дозволяючи обмінюватися даними в режимі реального часу;
- події та дані: використання подій дозволяє передавати та обробляти дані в режимі реального часу, що є ключовим для синхронізації змін між користувачами;
- надійність: механізм перепідключення та відновлення з'єднання робить його стійким до втрати з'єднання та забезпечує постійну комунікацію.

Переваги Quill [13]:

- інтерактивний редактор: Quill надає потужний та інтуїтивно зрозумілий інтерфейс для редагування текстового контенту з можливістю форматування та вставки різноманітних елементів;
- підтримка змін у реальному часі: забезпечує миттєве відображення змін, що робить його ідеальним інструментом для колективного редагування документів;

– підтримка плагінів: Quill [13] дозволяє легко розширювати його функціонал за допомогою власних плагінів або використання готових рішень.

Інтеграція обох технологій дозволяє створити взаємодію між клієнтами та сервером, забезпечуючи реальний час для спільного редагування.

Socket.IO [14] слугує інструментом для відслідковування та відображення останніх редагувань документа, а Quill [13] – для комфортного внесення змін.

Ці дві технології у зв'язці забезпечують миттєву синхронізацію змін між користувачами, що позитивно сприяє для ефективної та миттєвої взаємодії.

Використання обох технологій разом не лише забезпечує функціональність реального часу для спільного редагування документів, але й гарантує стабільну та інтуїтивно зрозумілу роботу системи управління документами для користувачів.

3.3 Проектування бази даних

Схема бази даних складається з 2 сутностей, пов'язаних 2 зв'язками.

Сутність User (Користувач): Представляє профіль користувача в системі. Має наступні поля: `_id` (унікальний ідентифікатор, який слугує ключем для ідентифікації конкретного документа), `firstName` (поле, що вказує на ім'я), `lastName` (поле, що вказує на прізвище), `email` (поле, що вказує на електронну пошту), `password` (поле, що вказує на пароль).

Сутність Document (Документ): представляє документ, який створений або власноруч змінений користувачем. Має наступні поля: `_id` (унікальний ідентифікатор, який слугує ключем для ідентифікації конкретного документа), `creator_id` (Ідентифікатор користувача, який створив цей документ. Здійснює зв'язок між сутностями Document та User), `name` (поле, що вказує на назв документа, що дозволяє користувачеві ідентифікувати його), `data` (об'єкт, який містить дані документа), `created_at` (поле, що вказує на дату та час створення

документа. За замовчуванням встановлено поточну дату та час), updated_at (поле, що вказує на дату та час останнього оновлення документа. Схема БД представлена на рис. 3.2.

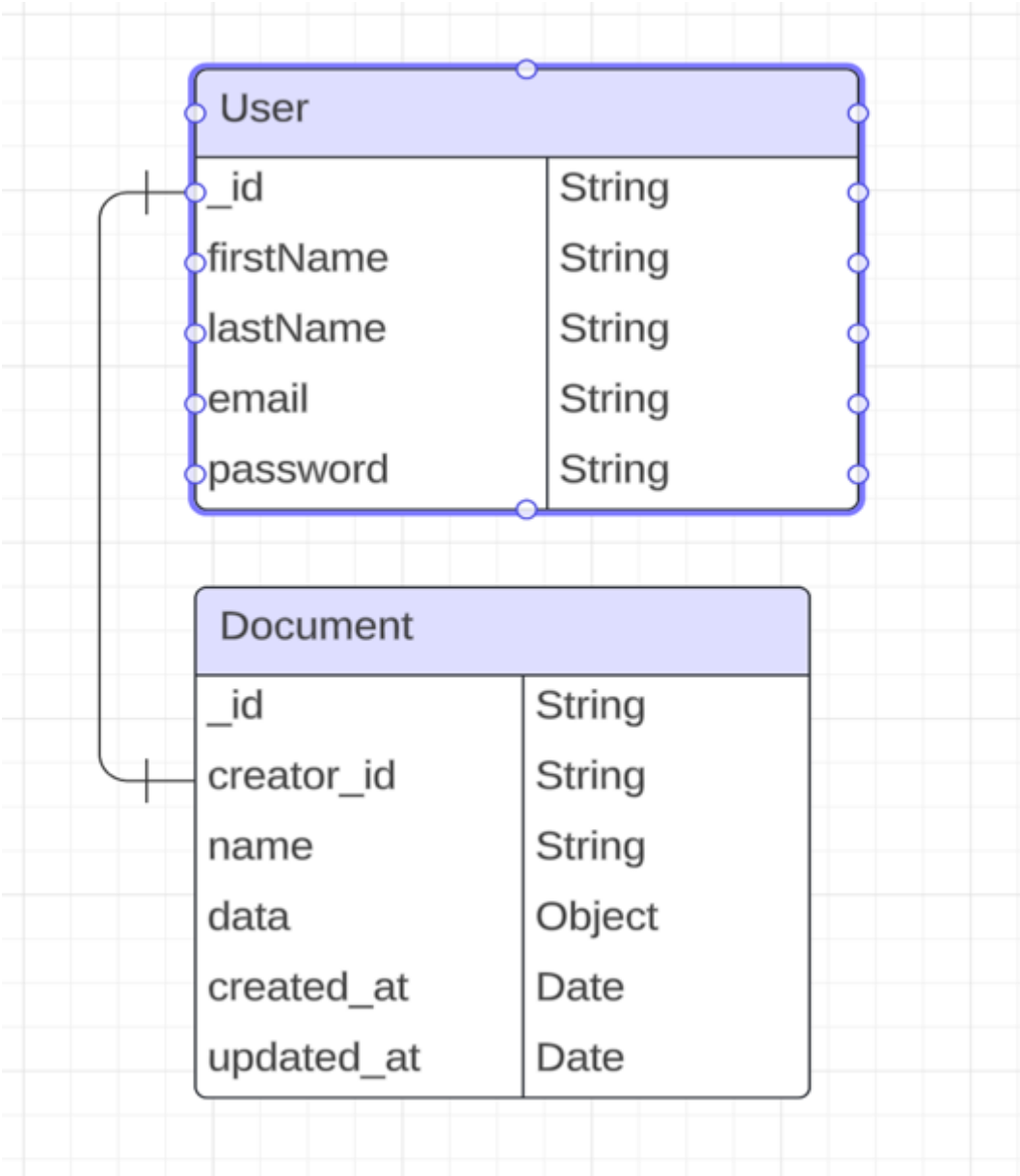


Рисунок 3.2 – Схема бази даних

Зв'язок між User та Document: Один–до–багатьох, оскільки користувач може мати багато документів, але кожен документ може мати лише одного автора.

3.4 Обмеження моделі спілкування «запит–відповідь»

Традиційна архітектура веб–застосунку передбачає, що клієнт ініціює HTTP–запит до сервера, який обробляє запит і відправляє відповідь назад. Клієнт отримує відповідь і оброблює її. Однак для застосунків, які працюють в режимі реального часу, така схема непридатна, оскільки сервер не може ініціювати відправку повідомлень клієнту, якщо відбулися зміни.

3.5 Способи реалізації real–time servісу

Опитування. По суті, спілкування за протоколом HTTP – це процес, коли клієнт просить щось із сервера. Для даного застосунку необхідно, щоб браузер автоматично оновлював зміст документа, щоб відобразити зміни користувачу. Тобто виходить, що коли щось змінюється на сервері, браузер має про це знати.

Перше, що спадає на думку, найпростіший спосіб, – опитування (англ. polling). Він полягає в тому, щоб клієнт періодично запитував сервер, чи відбулися якісь зміни. Для кожного такого опитування створюється HTTP–запит і сервер або повертає оновлення або каже, що нічого не змінилося.

Уявимо, що ми питаємо про оновлення кожну хвилину, у нас буде дуже багато запитів через певний проміжок часу і зайве навантаження на сервер. Також у цього способу є ще проблема – затримка між оновленнями та отриманням їх на клієнті. Сервер буде повідомляти не тоді, коли щось зміниться, а тоді коли прийде черговий запит із браузера

Довге опитування [15]. Щоб зекономити на ресурсах, покращенням попереднього способу, є довге опитування (англ. long polling). Цей підхід відрізняється тільки одним – сервер довше відповідає. Клієнт відправляє запит і сервер не відповідає одразу, а залишає з'єднання відкритим. У кожного такого запиту є час, протягом якого серверу треба відповісти (timeout). Тому є тільки два випадки, коли клієнт отримає відповідь: якщо є оновлення і сервер відповість одразу та якщо з'єднання пора розірвати. Після чого браузер відправить ще один запит.

За такого підходу ми уникаємо частого посилання запитів і відповідно зменшуємо навантаження на сервер та збільшується ймовірність того, що клієнт одразу отримає повідомлення про оновлення. Недоліками даного підходу є те, що при отриманні відповіді, запит переривається і створюється новий.

Наприклад, клієнт відправляє запит, щоб дізнатися чи відбулися якісь зміни у документі. Якщо система зберігає зміни після кожного введенного символу, то клієнт буде дуже часто отримувати повідомлення і відправляти наступний запит. А кожний такий запит – це створення заголовків, за потреби додавання аутентифікації і так далі. Тому виходить, що довге опитування підходить в основному для сервісів, у яких рідко відбуваються оновлення.

Специфікація *server-sent events* [16]. Є ще один спосіб реалізації застосунку в режимі реального часу – створення сервером HTTP-підключення з клієнтом за допомогою *server-sent events*. Ця специфікація описує вбудований клас *EventSource*, який дозволяє підтримувати з'єднання із сервером та отримувати події (потік тексту в кодуванні UTF-8). Цей клас має подію *onMessage* для обробки вхідних повідомлень та події для обробки помилок, які можуть прийти.

Щоб це реалізувати, клієнт має створити об'єкт *EventSource* та вказати URL, з яким буде підтримувати з'єднання: Після цього можна додати прослуховування повідомлень із сервера за допомогою вищезгаданого обробника подій *onMessage*.

Сервер має відповісти зі статусом 200 та використовуючи MIME-тип *text/event-stream*. Після чого підтримує з'єднання відкритим та відправляє повідомлення в особливому форматі. Кожне таке повідомлення відправляється як блок тексту, який закінчується декількома символами нового рядка. Але на практиці більш складні повідомлення відправляють у форматі JSON.

Отже, виходить, що всі події проходять тільки через одне з'єднання, що набагато краще за попередні способи. Але, на жаль, дані можна відправляти тільки в одну сторону (із сервера) та підтримує тільки текстові повідомлення.

Незважаючи на згадані недоліки, розробляти застосунок з використанням специфікації SSE простіше ніж з веб-сокетами та вона підтримує автоматичне перепідключення у разі втрати з'єднання.

3.6 Вебсокети

Протокол веб-сокет (WebSocket) надає можливість обмінюватися даними між сервером і браузером в два напрямки через один TCP сокет, який залишається відкритим, поки потік повідомлень не буде виконаний. Дані передаються як «пакети» без додаткових HTTP-запитів. Даний протокол має дві частини: рукоштовування та передача даних.

WebSocket [17] – це ідеальна транспортна система для реалізації застосунку в реальному часі. Веб-сокети забезпечують найбільш ефективно використання пам'яті сервера, надає повний дуплексний обмін даними між клієнтом і сервером та має найменшу затримку. Але також WebSocket має доволі строгі вимоги до серверу.

Веб-сокети [17] – це новітня технологія, яка добре підходить для застосунків, які потребують постійний обмін даними в реальному часі – ігри, онлайн-майданчики для торгівлі тощо.

Кожний веб-сокет починає своє життя з відправки на сервер HTTP get запиту, який просить оновити сокет. Якщо сервер підтримує WebSocket і погоджується, то далі спілкування починається за протоколом WebSocket, а не HTTP.

Клієнтське рукоштовування – це запит на оновлення. Зазвичай він просить оновитися до веб-сокета. Також він надсилає відкритий ключ – *Sec-WebSocket-Key* у форматі base64.

Варто відмітити, що якщо сервер прийняв запит на оновлення, то версія, яку вказав клієнт, підходить. В іншому випадку відповідь буде вказувати, які версії він підтримує, і тоді клієнт спробує ще раз з однією із них.

Якщо сервер підтримує веб-сокети і версія відходить, він відповідає з кодом статусу 101 і перемикає протокол. Також він повертає ключ у *SecWebSocket-Accept* – хеш, вирахований з ключа, який відправив клієнт.

Коли ж і клієнт і сервер відправили свої рукоштовування і вони були успішними, починається частина передачі даних – двосторонній зв'язок, по якому кожна зі сторін може надсилати дані незалежно від іншої.

Існує декілька типів повідомлень веб-сокетів: текстові дані (UTF-8), двійкові та контрольні повідомлення. Останній тип призначений для сигналізації на рівні протоколу, а не для передачі даних, – *ping/pong* комбінація, яка може використовуватися для перевірки чи з'єднання досі живе, або повідомлення що з'єднання має бути закритим – *close*.

Кожне повідомлення складається з одного або декількох кадрів (*frames*). Кожний такий фрейм одного повідомлення має однаковий тип даних. Всі кадри двійкові, вони мають біти у заголовку, щоб вказати, чи є це останнім фреймом у повідомленні чи ні, і який тип повідомлення це є. Також є біт заголовка, який вказує чи замасковано чи ні це повідомлення. І є біт, щоб встановити довжину корисного навантаження, а вона сама займає іншу частину фрейму.

Зазвичай ми не помічаємо ці фрейми, ми отримуємо тільки все повідомлення у події, а кадри будуть опрацьовані у фоновому режимі. Але важливо знати про них, тому що якщо ми будемо відлагоджувати веб-сокети у браузері, ми побачимо саме їх.

3.7 Висновки за розділом 3

У цій частині було визначено основні функціональні вимоги для створення програмного продукту. Для розробки було вибрано JavaScript як

основну мову програмування, React.js як бібліотеку для розробки веб-інтерфейсів, Express.js як фреймворк для розробки серверної частини функціоналу, а MongoDB використовувався як система управління базами даних.

Також було вирішено використовувати бібліотеку socket.io та Quill для обробки даних документів у режимі реального часу. Крім цього, була розроблена структура бази даних для проєкту. З метою забезпечення реального часу та спільного редагування документів було вирішено використовувати бібліотеку socket.io для встановлення бідирекційного з'єднання між клієнтом та сервером. Для обробки текстового вмісту документів у режимі реального часу була використана бібліотека Quill.

Механізм реалізації включає в себе використання Quill для розробки текстового редактора, який забезпечує багатий функціонал редагування тексту та підтримку форматування. Використання бібліотеки socket.io дозволяє побудувати надійне та ефективне з'єднання між клієнтом та сервером, де сервер може ініціювати відправку змін клієнту в реальному часі, що забезпечує спільне редагування документів та синхронізацію змін між користувачами.

4 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

4.1 Алгоритм функціонування програмного забезпечення

Логіка роботи сервісу не залежить від ролі користувача в системі, оскільки розділення по ролям не передбачено у проекті. Однак, для неавторизованого користувача діють обмеження.

Спільним для всіх користувачів є можливість перегляду головної сторінки сервісу з переліком технологій, використаних для створення продукту. Це було досягнуто за допомогою підходу Guarded Routes, що являють собою обертку над компонентом сторінки. Під час переходу користувача на захищену сторінку, відбувається перевірка наявності токена авторизації.

Сторінка авторизації доступна усім користувачам. Основні функції, що може надати сторінка профіля: вхід у систему для подальшого використання продукту.

Основні функції, що може надати сторінка реєстрації:

- валідація вхідних даних;
- створення облікового запису для подальшого використання продукту.

Сторінка документів доступна лише авторизованим користувачам. Основні функції, що може надати сторінка з документами:

- перегляд особистих документів;
- створення особистих документів;
- видалення особистих документів;
- перейменування особистих документів;
- пошук за особистих документів за іменем.

Сторінка профілю доступна лише авторизованим користувачам. Основні функції, що може надати сторінка профіля:

- перегляд особистої інформації профіля;
- редагування особистої інформації профіля;

Схема функціонування сервісу наведена на рис. 4.1.



Рисунок 4.1 – Схема функціонування сервісу

4.2 Структура реалізованого програмного забезпечення

Файли застосунку згруповані у директорії відповідно до їх зв'язування та зображені на рис. 4.2, 4.3.

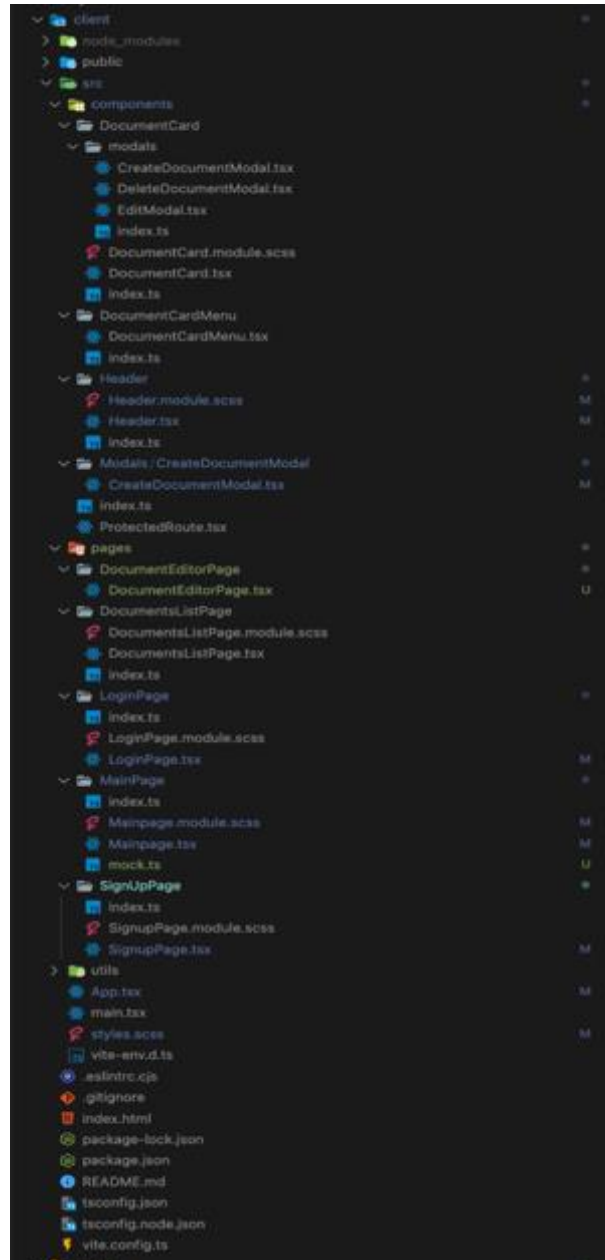


Рисунок 4.2 – Файлова структура клієнтської сторони застосунку у редакторі VS Code

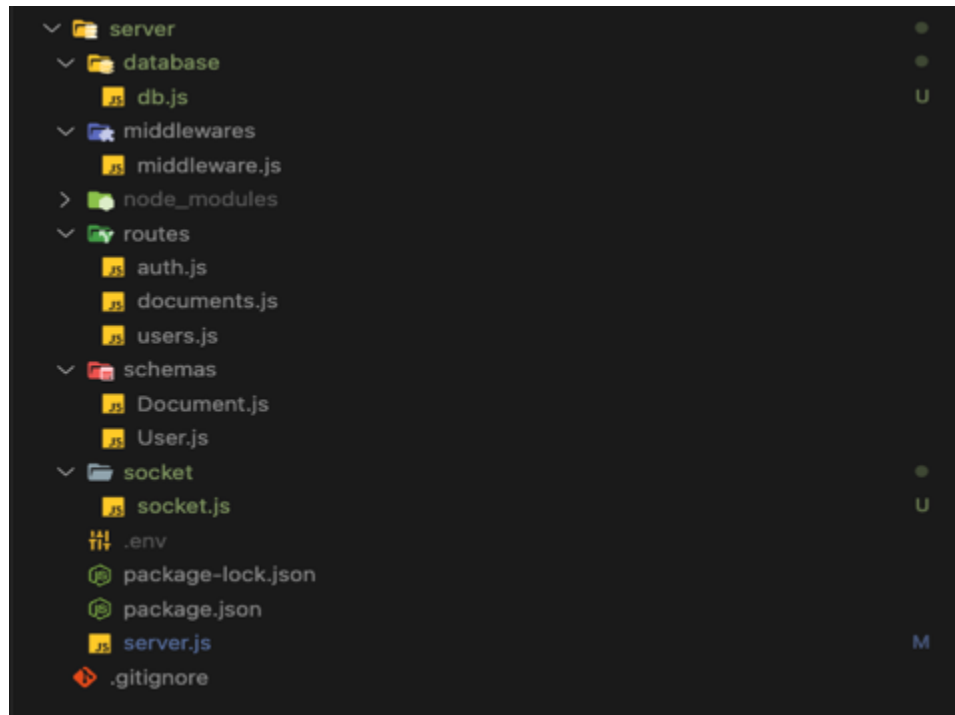


Рисунок 4.3 – Файлова структура серверної сторони застосунку у редакторі VS Code

Структура поточної реалізації програмного забезпечення передбачає умовний розподіл ресурсів та файлів імплементації алгоритмів за директорією відповідних програмної логіки. Було створено розділи client (містить у собі візуальну частину проекту) та server (містить у собі серверну частину проекту).

Розглянемо детальніше серверну частину коду, було створено наступні директорії:

- database (підключення до бази даних та конфігурація з'єднання);
- middlewares (проміжники, котрі використовуються перед виконанням запитів);
- routes (маршрути серверу);
- schemas (схеми для колекцій бази даних);
- socket (забезпечує роботу веб-сокет підключення та оперує даними за допомогою веб-сокетів).

До розділу database належать наступні файли:

- `db.js`, він містить підключення до бази даних та конфігурацію з'єднання для подальшого використання.

До розділу `middlewares` належать файли:

- `middlewares.js`, містить логіку роботи проміжних функцій, що будуть використовуватись у подальшому.

До розділу `routes` належать файли:

- `auth.js` – логіка, що пов'язана із авторизацією;
- `documents.js` – логіка, що пов'язана із роботою з документами;
- `users.js` – логіка, що пов'язана із користувачами;
- До розділу `schemas` належать файли;
- `Document.js` – схема колекції документів;
- `User.js` – схема колекції користувачів.

До розділу `socket` належать файли:

- `socket.js`, містить логіку, що забезпечує роботу веб-сокета підключення та оперує даними за допомогою веб-сокетів.

Розглянемо детальніше візуальну частину коду, було створено наступні директорії:

- `components` (компоненти у проекті, призначені для повторного використання);
- `pages` – (сторінки веб-сайту з прописаною логікою та версткою);
- `utils` – (директорія, що містить допоміжний функціонал);

До розділу `components` належать наступні файли:

- `DocumentCard.tsx` – логіка та верстка для картки;
- `DocumentCardMenu.tsx` – логіка та верстка меню картки;
- `Header.tsx` – логіка та верстка шапки для сайту;
- `CreateDocumentModal.tsx` – логіка та верстка модального вікна картки.

До розділу `pages` належать наступні файли:

- `DocumentEditorPage.tsx` – логіка та верстка сторінки для редагування документів;

- DocumentsListPage.tsx – логіка та верстка сторінки для перегляду та пошуку документів;
 - LoginPage.tsx – логіка та верстка сторінки для авторизації;
 - SignUp.tsx – логіка та верстка сторінки для реєстрації;
 - MainPage.tsx – логіка та верстка головної сторінки.
- До розділу utils належать наступні файли:
- utils.ts – логіка допоміжних функцій у проекті.

4.3 Опис прийнятих під час реалізації рішень

Для повноцінної роботи вебсайту необхідно було реалізувати авторизацію та авторизацію. Авторизація може бути реалізовано декількома популярними способами.

Сесійна авторизація: Клієнт отримує унікальний ідентифікатор сесії після успішного входу. Цей ідентифікатор використовується для ідентифікації користувача на сервері під час кожного запиту. До переваг можна віднести просте впровадження, вбудоване в деякі фреймворки. До недоліків можна віднести наступне: сесії можуть призводити до проблем масштабування та вимагають збереження стану на сервері.

JWT (JSON Web Tokens) [18]: Після входу, сервер генерує JWT, який містить інформацію про користувача та підписує його. Клієнт отримує цей токен і включає його у заголовки для кожного запиту. Сервер перевіряє та розшифровує токен для ідентифікації користувача. До переваг можна віднести легкість впровадження, можливість передачі між різними сервісами, також не вимагає збереження стану на сервері. До недоліків можна віднести наступне: інформація в токені доступна клієнту (але не може бути змінена), токени можуть бути вкрадені, якщо не використовується HTTPS.

OAuth (Open Authorization): Стандартний протокол для делегування авторизації третім службам. Користувачі можуть використовувати облікові записи Google, Facebook, або інших сервісів для входу в застосунок. До

переваг можна віднести зручність для користувачів, необхідність введення пароля відсутня. До недоліків можна віднести залежність від зовнішніх сервісів, складнощі в імплементації.

Було обрано метод JWT для авторизації через легкість імплементації, простоті масштабованості, захищеності та гнучкості.

4.4 Висновки за розділом 4

У цьому розділі було складено і реалізовано алгоритм функціонування програмного забезпечення. Було створено структуру вебсервісу та описано функціонал кожного модуля системи.

5 ЕКСПЛУАТАЦІЯ ТА ТЕСТУВАННЯ РОЗРОБЛЕНОЇ ПРОГРАМИ

5.1 Призначення програми

Розроблений вебсервіс призначений для редагування документів декількома користувачами в режимі реального часу. Цей продукт може бути використаний як корпоративний продукт компанії, призначений для внутрішнього використання, що міститиме необхідний функціонал з можливістю його розширення. Не останню роль займає й питання безпеки, документи будуть надійно збережені та не будуть передаватись стороннім компаніям. Вміст документів зберігається в зашифрованому вигляді.

Програма включає в себе зручний інтерфейс перегляду та редагування документів, пошук по імені, систему автозбереження.

5.2 Умови виконання програми

Технічні вимоги сервера:

- жорсткий диск з вільним простором – 1,5 Гб;
- оперативна пам'ять – 4 Гб;
- процесор: 4 ядра, 2,8 ГГц частота.

У даній технічній реалізації встановлено безкоштовний тариф AWS S3 Storage, що обмежений завантаженням малої кількості даних. Для повноцінної версії знадобиться зміна тарифного плану.

Для справної роботи вебсервісу на стороні користувача, потрібний сучасний браузер з можливістю підключення до мережі Інтернет.

5.3 Виконання програми

Точкою початку роботи із сервісом є головна сторінка. На ній знаходиться опис технологій, що були використані при розробці та короткі

відомості про них. За умови неавторизованого користувача сторінка має такий вигляд (рис. 5.1).

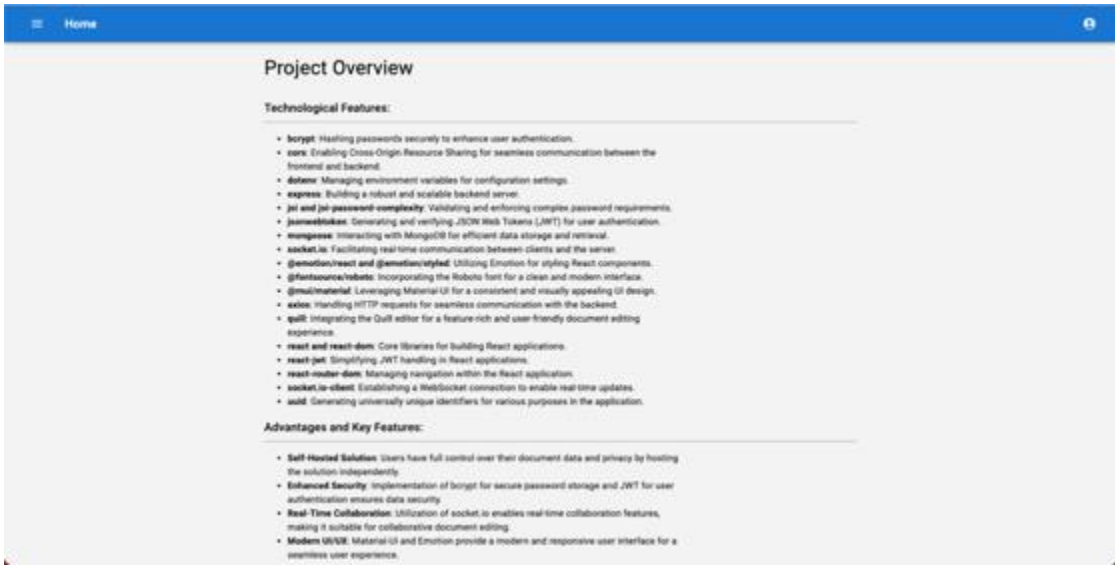


Рисунок 5.1 – Вигляд головної сторінки для неавторизованого користувача

У верхньому правому куті знаходяться кнопки «Log in», при натисканні на яку, користувача переводить на сторінку авторизації (рис.5.2). Авторизація являє собою форму з двома полями: пошта користувача та пароль.



Рисунок 5.2 – Вигляд сторінки авторизації

У тому випадку, якщо користувач ще не має облікового запису, йому надається така можливість, для створення облікового запису треба натиснути «Not a member yet?». Сторінку реєстрації зображено на рис. 5.3.



Рисунок 5.3 – Вигляд сторінки реєстрації

Після того як користувач авторизувався, він може перейти на сторінку з документами (рис 5.4).

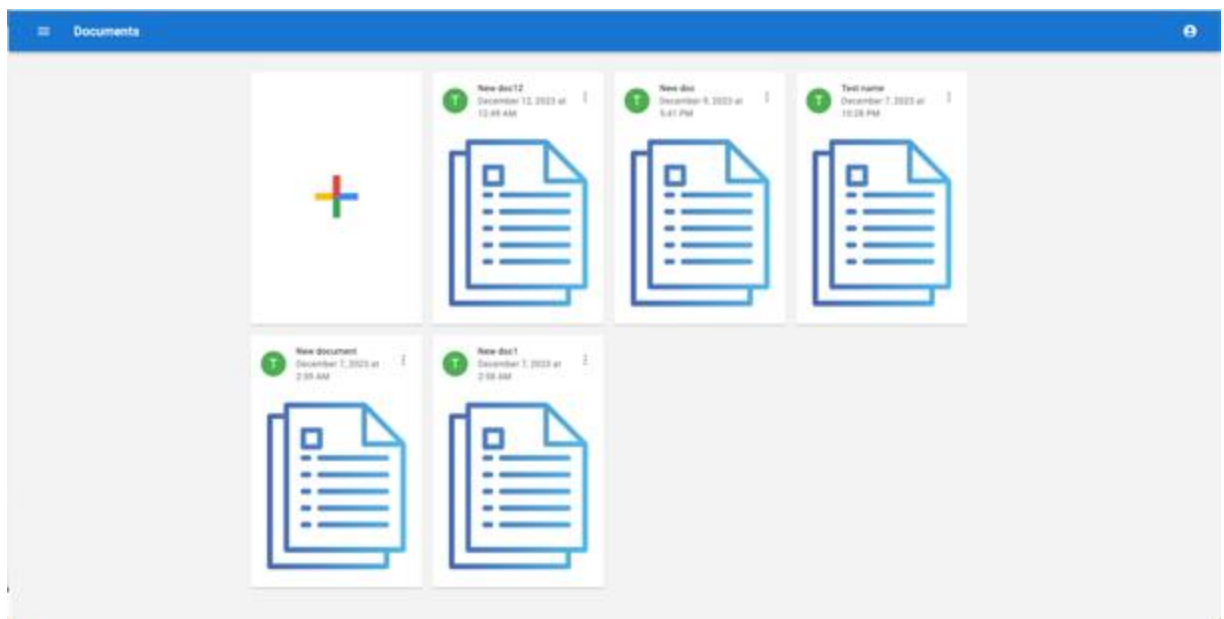


Рисунок 5.4 – Вигляд сторінки для керування документами

На сторінці представлений список існуючих документів, закріплених за авторизованим користувачем. Користувач може скористатись пошуком документу по імені. Кожна картка включає картинку мініатюру документа, мініатюру картинки профілю, що містить в собі першу літеру імені авторизованого користувача, ім'я документа, дату останнього редагування та кнопку дій, що викликає контекстне меню для управління документом (рис. 5.5).

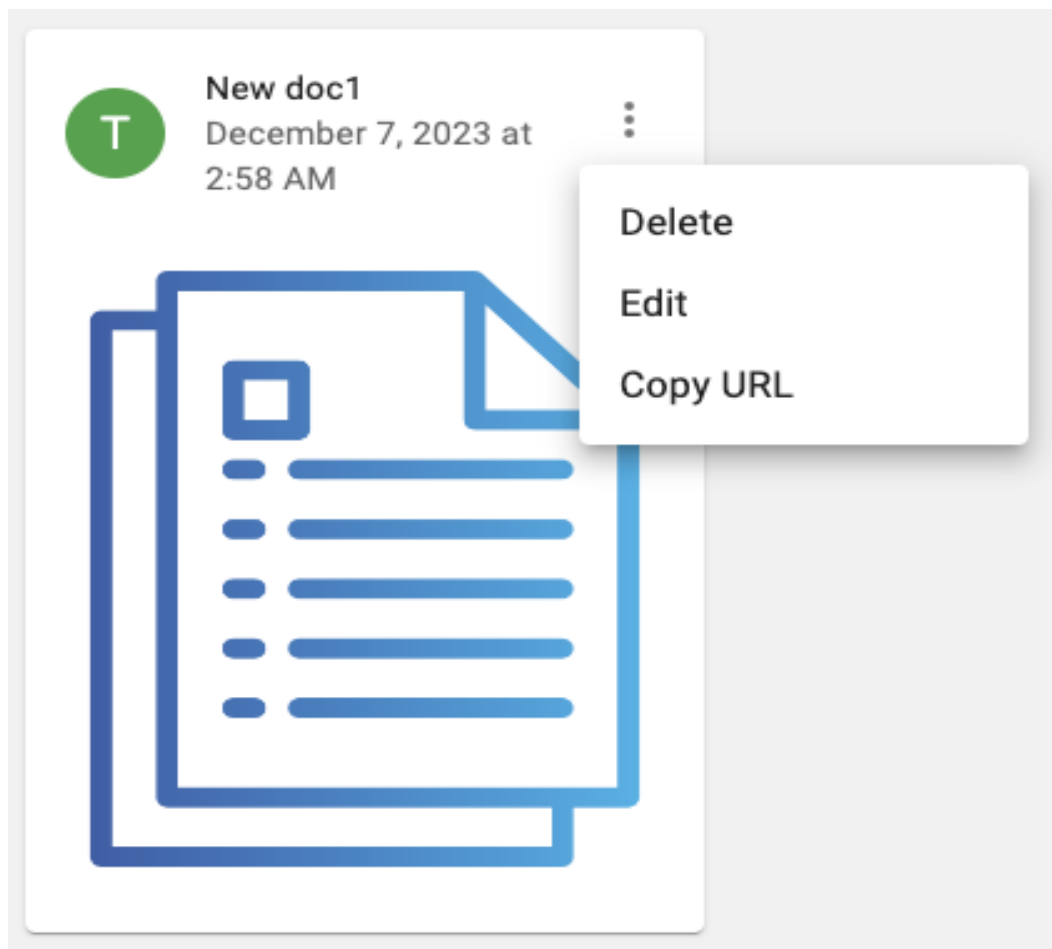


Рисунок 5.5 – Вигляд меню для керування документом

При натисканні на «Delete», з'являється модальне меню для підтвердження видалення документа (рис 5.6)

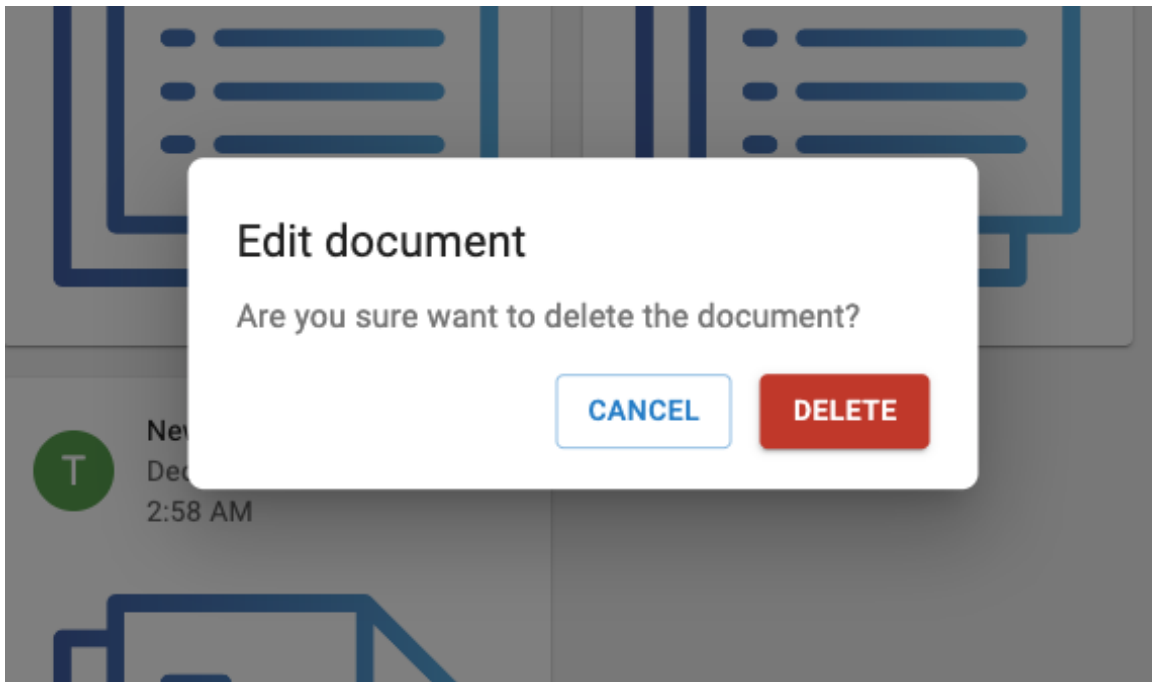


Рисунок 5.6 – Вигляд модального вікна для видалення документа

При натисканні на «Edit», з'являється модальне меню для внесення змін в ім'я документа (рис 5.7)

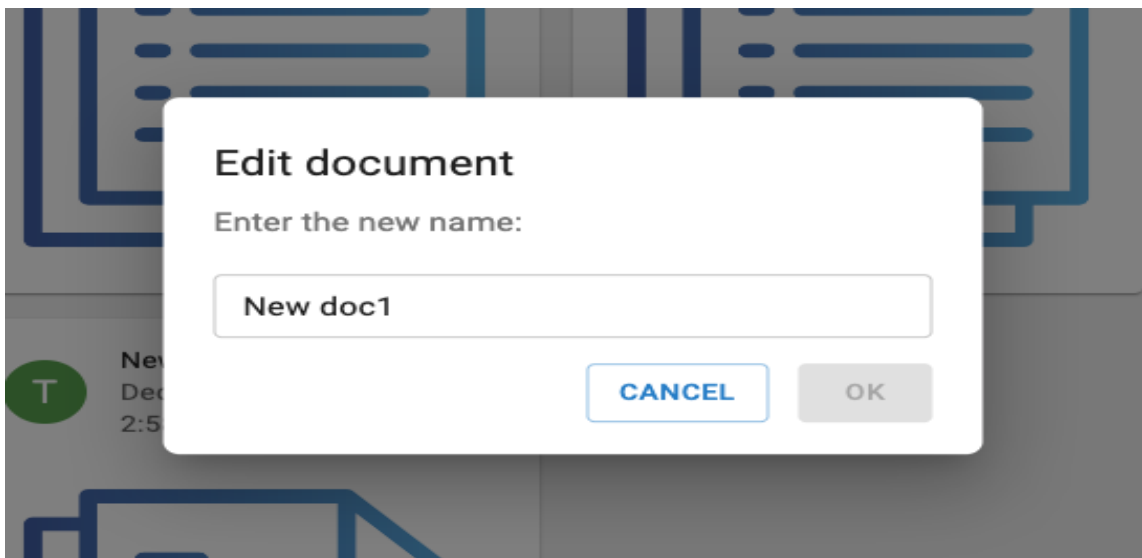


Рисунок 5.7 – Вигляд модального вікна для внесення змін в ім'я документа

При натисканні на документ, користувача переносить у редактор тексту. Тут він може редагувати текст за допомогою звичного інтерфейсу, знайомого з таких програм як Google docs, та інших (рис 5.8).



Рисунок 5.8 – Вигляд інтерфейсу для редагування документа

Також, була передбачена можливість друкувати документи шляхом вбудованої використання у сучасні браузері функції друку (рис.5.9). Було відредаговано CSS стилі для друку документа в точній відповідності до реального документа.

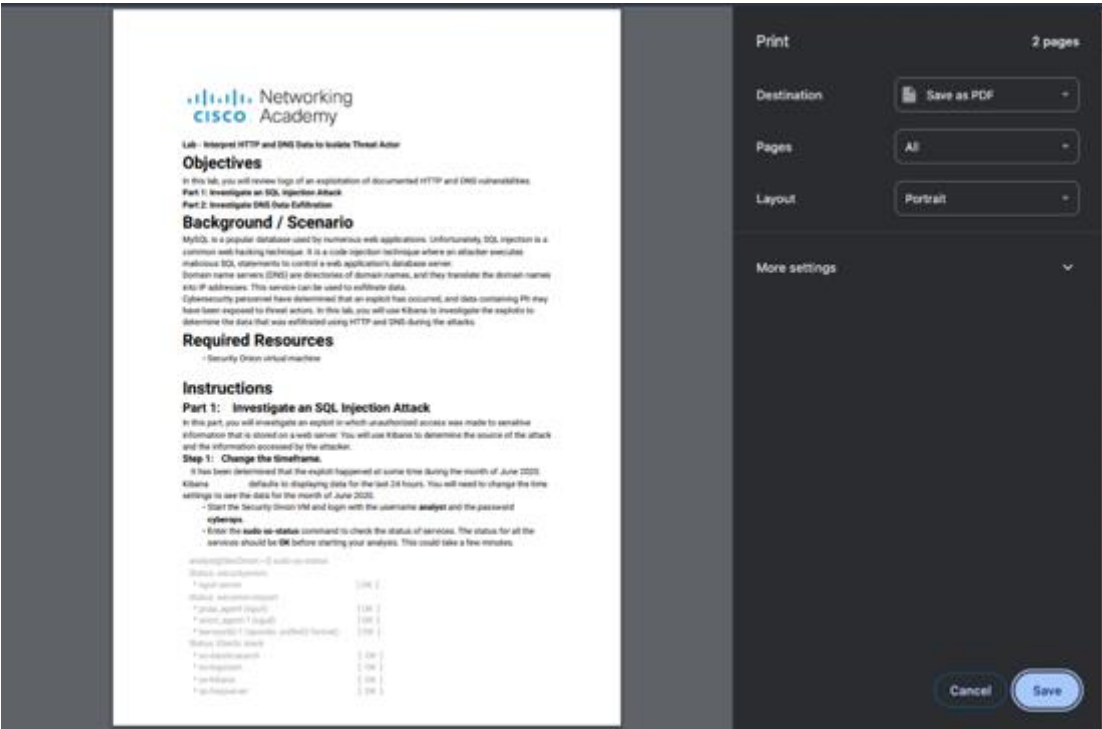


Рисунок 5.9 – Вигляд інтерфейсу для друку документа

5.4 Висновки за розділом 5

В процесі розробки та вдосконалення застосунку я приділяв велику увагу його тестуванню, включаючи мануальні тести, які були проведені з великою увагою до деталей та функціональності. Мій підхід до мануального тестування:

Функціональність та Взаємодія: Крок за кроком було протестовано кожную функціональність, переконуючись, що всі елементи взаємодіють точно та передбачувано. Також, я використовував різноманітні вхідні дані для перевірки забезпечення правильності обробки кожної операції.

Безпека та Захист Даних: Проведено тести, спрямовані на виявлення можливих порушень безпеки та перевірка заходів забезпечення конфіденційності даних користувачів.

Отримані результати тестування відображають високий рівень якості та надійності застосунку.

У результаті тестування всі модулі системи продемонстрували стабільну та надійну роботу. Виявлені помилки були виправлені. Тестування підтвердило, що розроблена система для редагування документів у реальному часі повністю готова до впровадження та використання в реальних умовах.

ВИСНОВКИ

За результатами виконання дипломної кваліфікаційної роботи виконано такі задачі.

Проведено аналіз концепції сумісного редагування документів, розглянуто підходи до реалізації онлайн-редакторів тексту, проведено порівняння з програмними реалізаціями; Визначено функціональні вимоги для реалізації програмного забезпечення, обрано мову програмування JavaScript, фреймворки Express та React, БД MongoDB, створено структуру бази даних проекту.

Реалізовано весь запланований функціонал програмного забезпечення, створено структуру веб-сервісу та описано кожний модуль системи.

Проведено тестування розробленої системи. У результаті тестування всі модулі системи продемонстрували стабільну та надійну роботу. Виявлені помилки були виправлені. Тестування підтвердило, що розроблена система для редагування документів у реальному часі повністю готова до впровадження та використання в реальних умовах.

Розроблений вебсервіс призначений для спільного редагування документів у реальному часі і являє собою веб-сайт з усіма необхідними функціями для використання в реальних умовах роботи команди. Програма включає в себе зручний інтерфейс адміністрування документів, пошук за іменем документів, систему шифрування даних для забезпечення конфіденційності та безпеки. Завдання дипломної кваліфікаційної роботи в результаті повністю виконано.

Науковою новизною є впровадження та оптимізація технологій операційного перетворення в системі спільного редагування документів.

Подолання проблем паралельних редагувань та забезпечення консистентності даних стає ключовим завданням, що вирішується за допомогою операційного перетворення. Введення векторних годинників для визначення порядку операцій, трансформація конфліктуючих операцій та

аплікація їх до локальних копій документів гарантує ефективну синхронізацію та уникнення суперечностей.

Також, новизною є порівняння та модифікація методів роботи системи під час спільного редагування документів. Це відкриває нові можливості для підвищення продуктивності системи.

Практична цінність розробленого застосунку полягає в його здатності забезпечити ефективне та зручне спільне редагування документів в реальному часі. Користувачі можуть одночасно працювати над документами, спостерігати за змінами та уникати конфліктів завдяки використанню технік синхронізації тексту, зокрема Операційного Перетворення.

Важливим аспектом є також безпека та автентифікація, забезпечені за допомогою JWT та Bcrypt. Це гарантує захист конфіденційної інформації користувачів та унеможливорює несанкціонований доступ.

Застосунок дозволяє використовувати сучасні технології та гнучко адаптується до різних типів документів, що робить його універсальним та придатним для різноманітних сценаріїв використання. Можливість отримувати оновлення в реальному часі сприяє покращенню комунікації в командах та підвищує продуктивність спільної роботи.

Такий застосунок не лише впроваджує сучасні технології MERN стеку, а й відповідає на потреби сучасного користувача у зручному та ефективному спільному редагуванні документів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. GOOGLE DOCS [Electronic resource]. – Access mode: <https://docs.google.com/>.
2. MICROSOFT 365 [Electronic resource]. – Access mode: <https://www.office.com/>.
3. NOTION [Electronic resource]. – Access mode: <https://www.notion.so/>.
4. Introduction to SignalR [Electronic resource]. – Access mode: <https://docs.microsoft.com/engb/aspnet/signalr/overview/getting-started/introduction-to-signalr>.
5. Operational Transformation [Electronic resource]. – Access mode: <http://operationaltransformation.github.io/index.html>.
6. JSONWEBTOKEN [Electronic resource]. – Access mode: <https://jwt.io/>.
7. BCrypt [Electronic resource]. – Access mode: <https://www.npmjs.com/package/bcrypt>.
8. EXPRESS.JS [Electronic resource]. – Access mode: <https://expressjs.com/>.
9. REACT.JS [Electronic resource]. – Access mode: <https://react.dev/>.
10. MUI [Electronic resource]. – Access mode: <https://mui.com/>.
11. MONGODB [Electronic resource]. – Access mode: <https://www.mongodb.com/>.
12. Comparing MongoDB vs PostgreSQL [Electronic resource]. – Access mode: <https://www.mongodb.com/compare/mongodb-postgresql>.
13. QUILL.JS [Electronic resource]. – Access mode: <https://quilljs.com/>.
14. SOCKET.IO [Electronic resource]. – Access mode: <https://socket.io/>.
15. Long Polling [Electronic resource]. – Access mode: <https://javascript.info/long-polling>.
16. Server Sent Events [Electronic resource]. – Access mode:

<https://javascript.info/server-sentevents>.

17. Websocket [Electronic resource]. – Access mode:
<https://developer.mozilla.org/en-US/docs/Web/API/WebSocket>.

18. JWT [Electronic resource]. – Access mode: <https://jwt.io/>.

ДОДАТОК А
Технічне завдання

Вступ

Розроблений вебсервіс спрямований на комерційну реалізацію інструменту для спільного редагування документів. Також, веб-сервіс передбачає можливість використання базових інструментів для керування стилями в документі та надає необхідний для цього функціонал.

A.1 Підстави для розробки

Підставою для розробки було завдання на дипломну кваліфікаційну роботу за темою «Дослідження та програмна реалізація технологій операційного перетворення для системи спільного редагування документів в реальному часі» на основі наказу №437 від 14 листопада 2023 р. за Національним університетом «Запорізька політехніка».

A.2 Призначення розробки

Розроблений веб-сервіс призначений для спільного редагування документів у реальному часі і являє собою веб-сайт з усіма необхідними функціями для використання в реальних умовах роботи команди.

Програма включає в себе зручний інтерфейс адміністрування документів, пошук за іменем документів, систему шифрування даних для забезпечення конфіденційності та безпеки.

A.3 Основні вимоги до програми

A.3.1 Вимоги до функціональних характеристик

Сервіс повинен забезпечувати наступний функціонал:

- можливість перегляду власного профілю;
- можливість оновлення інформації власного профілю;

- можливість перегляду власних документів;
- можливість створення документів;
- можливість оновлення, перейменування та видалення власних документів;
- можливість перегляду власного профілю;
- можливість реєстрації;
- можливість входу у особистий кабінет;
- можливість одночасного редагування документа кількома користувачами;
- можливість автозбереження документа;
- можливість редагування розміру, товщини, стилю шрифту та інший функціонал пов'язаний із редагуванням тексту;
- можливість використання undo, redo під час редагування документа;
- можливість друку документа.

A.3.2 Вимоги до надійності

Надійність працездатності сервісу залежить від правильності налаштування сервера бази даних з урахуванням її програмної реалізації у системі. Так само система включає статичні файли, що зберігаються на AWS Cloud серверах, що вимагають налаштування тарифу використання даних.

A.3.3 Умови експлуатації

Для коректної роботи системи потрібен сервер для програмної частини програми, сервер бази даних та налаштування тарифу AWS Cloud.

А.3.4 Вимоги до складу та параметрів технічних засобів

Технічні вимоги сервера:

- жорсткий диск з вільним простором – 1,5 Гб;
- оперативна пам'ять – 4 Гб;
- процесор: 4 ядра, 2,8 ГГц частота.

У даній технічній реалізації встановлено безкоштовний тариф AWS S3 Storage, що обмежений завантаженням малої кількості зображень. Для продакшн версії знадобиться зміна тарифного плану.

Для справної роботи вебсервісу на стороні користувача, потрібний сучасний браузер з можливістю підключення до мережі Інтернет.

А.4 Порядок контролю та приймання

Контроль за виконанням дипломної кваліфікаційної роботи покладається на керівника. Остаточне рішення приймається екзаменаційною комісією під час захисту роботи.

ДОДАТОК Б
Текст програми

Б.1 Текст файла index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml" href="/vite.svg" />
    <meta name="viewport" content="width=device-width, initial-
scale=1.0" />
    <link rel="preconnect" href="https://fonts.googleapis.com" />
    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin
/>
    <link
      rel="stylesheet"
href="https://fonts.googleapis.com/css2?family=Roboto:wght@300;400;500
;600;700&display=swap"
    />
    <link
      rel="stylesheet"
      href="https://fonts.googleapis.com/icon?family=Material+Icons"
    />
    <title>Vite + React + TS</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.tsx"></script>
  </body>
</html>
```

Б.2 Текст файла styles.scss

```
*,
*::before,
*::after {
```

```

    box-sizing: border-box;
    font-family: "Roboto", sans-serif;
}

```

```

html,
body {
    height: 100%;
    margin: 0;
    padding: 0;
    background-color: #f3f3f3;
}

```

```

#root {
    height: 100vh;
}
.App {
    height: 100%;
    // padding: 0 64px;
}

```

```

.container {
    .ql-container {
        &.ql-snow {
            display: flex;
            justify-content: center;
            border: none;
        }
    }
    .ql-toolbar {
        &.ql-snow {
            display: flex;
            justify-content: center;
            position: sticky;
            top: 0;

```

```

        z-index: 1;
        background-color: #f3f3f3;
        border: none;
        box-shadow: 0 0 5px 0 rgba(0, 0, 0, 0.5);
    }
}
 ql-editor {
    width: 8.5in;
    min-height: 11in;
    padding: 1in;
    margin: 1rem;
    box-shadow: 0 0 5px rgba(0, 0, 0, 0.5);
    background-color: white;
}
}

```

```

@page {
    margin: 1in;
    border: none;
    outline: none;
}

```

```

@media print {
    body {
        background: none;
    }
}

```

```

.container ql-editor {
    width: 6.5in;
    height: 9in;
    padding: 0;
    margin: 110px;
    box-shadow: none;
    align-self: flex-start;
}

```

```

}

.container > .ql-toolbar.ql-snow {
  display: none;
}
}

```

Б.3 Текст файла package.json

```

{
  "name": "client",
  "private": true,
  "version": "0.0.0",
  "type": "module",
  "scripts": {
    "dev": "vite",
    "build": "tsc && vite build",
    "lint": "eslint . --ext ts,tsx --report-unused-disable-directives --max-warnings 0",
    "preview": "vite preview"
  },
  "dependencies": {
    "@emotion/react": "^11.11.1",
    "@emotion/styled": "^11.11.0",
    "@fontsource/roboto": "^5.0.8",
    "@mui/icons-material": "^5.14.19",
    "@mui/material": "^5.14.19",
    "axios": "^1.6.2",
    "quill": "^1.3.7",
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-jwt": "^1.2.0",
    "react-router-dom": "^6.18.0",
    "socket.io-client": "^4.7.2",

```

```

    "uuid": "^9.0.1"
  },
  "devDependencies": {
    "@types/quill": "^2.0.14",
    "@types/react": "^18.2.15",
    "@types/react-dom": "^18.2.7",
    "@types/uuid": "^9.0.7",
    "@typescript-eslint/eslint-plugin": "^6.0.0",
    "@typescript-eslint/parser": "^6.0.0",
    "@vitejs/plugin-react": "^4.0.3",
    "eslint": "^8.45.0",
    "eslint-plugin-react-hooks": "^4.6.0",
    "eslint-plugin-react-refresh": "^0.4.3",
    "sass": "^1.69.5",
    "typescript": "^5.0.2",
    "vite": "^4.4.5"
  }
}

```

Б.4 Текст файла main.tsx

```

import React from "react";
import ReactDOM from "react-dom/client";
import App from "./App.tsx";
import "./styles.scss";

ReactDOM.createRoot(document.getElementById("root")!).render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);

```

Б.5 Текст файла App.tsx

```

import {
  Navigate,
  createBrowserRouter,
  RouterProvider,

```

```

} from "react-router-dom";
import { v4 as uuidv4 } from "uuid";
import { SignUpPage } from "../pages/SignUpPage";
import { LoginPage } from "../pages/LoginPage";
import PrivateRoute from "../components/ProtectedRoute";
import { DocumentsListPage } from "../pages/DocumentsListPage";
import { Mainpage } from "../pages/MainPage";
import                                     DocumentEditorPage           from
"../pages/DocumentEditorPage/DocumentEditorPage";
function App() {
  const router = createBrowserRouter([
    {
      path: "/",
      element: <Mainpage />,
    },
    {
      path: "/documents",
      element: (
        <PrivateRoute>
          <DocumentsListPage />
        </PrivateRoute>
      ),
    },
    {
      path: "/blank",
      element: (
        <PrivateRoute>
          <Navigate to={` /documents/${uuidv4()} `} />
        </PrivateRoute>
      ),
    },
    {
      path: "/documents/:id",
      element: (
        <PrivateRoute>
          <DocumentEditorPage />
        </PrivateRoute>
      ),
    },
    {
      path: "/signup",
      element: <SignUpPage />,
    },
    {
      path: "/login",
      element: <LoginPage />,
    },
  ]);
  return (
    <div className="App">
      <RouterProvider router={router}></RouterProvider>
    </div>
  );
}

```

```

    </div>
  );
}

export default App;

```

Б.6 Текст файла SignUpPage.tsx

```

import { Button, TextField, Box, Link } from "@mui/material";
import { blue } from "@mui/material/colors";
import { useState } from "react";
import axios from "axios";
import { makeStyles } from "@mui/styles";
// import styles from "../SignupPage.module.scss";
import { useNavigate } from "react-router-dom";

const useStyles = makeStyles({
  root: {
    display: "flex",
    justifyContent: "flex-start",
    alignItems: "center",
    height: "100vh",
    width: "100vw",
    backgroundImage:
      `url('https://previews.123rf.com/images/orelphoto/orelphoto1401/orelphoto140100143/25867039-blue-security-background-with-hex-code.jpg')`,
    backgroundRepeat: "no-repeat",
    backgroundSize: "cover",
  },
  sidemenu: {
    width: "450px",
    height: "100%",
    display: "flex",
    justifyContent: "center",
    alignItems: "center",
    backgroundColor: blue[700],
  },
  form: {
    display: "flex",
    flexDirection: "column",
    gap: "8px",
    width: "100%",
    margin: "0 24px",
    padding: "20px",
    borderRadius: "8px",
    boxShadow: "0 0 10px rgba(0, 0, 0, 0.1)",
    backgroundColor: "#fff",
  },
  input: {

```

```

    marginBottom: "15px",
  },
  error: {
    color: "red",
    marginBottom: "15px",
  },
  submitButton: {
    width: "100%",
  },
});

export const SignUpPage = () => {
  const classes = useStyles();

  const navigate = useNavigate();
  const [data, setData] = useState({
    firstName: "",
    lastName: "",
    email: "",
    password: "",
  });
  const [error, setError] = useState("");

  const handleSubmit = async (e: any) => {
    e.preventDefault();
    try {
      const url = "http://localhost:5000/api/users";
      const { data: res } = await axios.post(url, data);
      navigate("/login");
      console.log(res.message);
    } catch (error: any) {
      if (
        error.response &&
        error.response.status >= 400 &&
        error.response.status < 500
      ) {
        setError(error.response.data.message);
      }
    }
  };

  const handleChange = ({
    currentTarget,
  }: React.ChangeEvent<HTMLInputElement>) => {
    setData((prevData) => ({
      ...prevData,
      [currentTarget.name]: currentTarget.value,
    }));
  };

  return (
    <div className={classes.root}>

```

```

<Box className={classes.sidemenu}>
  <form className={classes.form} onSubmit={handleSubmit}>
    <TextField
      className={classes.input}
      name="firstName"
      label="First Name"
      type="text"
      value={data.firstName}
      onChange={handleChange}
    />
    <TextField
      className={classes.input}
      name="lastName"
      label="Last Name"
      type="text"
      value={data.lastName}
      onChange={handleChange}
    />
    <TextField
      className={classes.input}
      name="email"
      label="Email"
      type="email"
      value={data.email}
      onChange={handleChange}
    />
    <TextField
      className={classes.input}
      name="password"
      label="Password"
      type="password"
      value={data.password}
      onChange={handleChange}
    />
    {error && <div>{error}</div>}
    <Button
      type="submit"
      variant="contained"
      color="primary"
      className={classes.submitButton}
    >
      Sign up
    </Button>
    <Link
      sx={{
        width: "fit-content",
        cursor: "pointer",
        my: 1,
        textTransform: "uppercase",
        fontSize: 12,
      }}
    />
  </form>
</Box>

```

```

        href="/login"
      >
        Already a member?
      </Link>
    </form>
  </Box>
</div>
);
};

```

Б.7 Текст файла LoginPage.tsx

```

import { Box, Button, Link, TextField } from "@mui/material";
import { useState } from "react";
import axios from "axios";
import { makeStyles } from "@mui/styles";
import { useNavigate } from "react-router-dom";
import { blue } from "@mui/material/colors";

const useStyles = makeStyles({
  root: {
    display: "flex",
    justifyContent: "flex-start",
    alignItems: "center",
    height: "100vh",
    width: "100vw",
    backgroundImage:
      `url('https://previews.123rf.com/images/orelphoto/orelphoto1401/orelph
oto140100143/25867039-blue-security-background-with-hex-code.jpg')`,
    backgroundRepeat: "no-repeat",
    backgroundSize: "cover",
  },
  sidemenu: {
    width: "450px",
    height: "100%",
    display: "flex",
    justifyContent: "center",
    alignItems: "center",
    backgroundColor: blue[700],
  },
  form: {
    display: "flex",
    flexDirection: "column",
    gap: "8px",
    width: "100%",
    margin: "0 24px",
    padding: "20px",
    borderRadius: "8px",
    boxShadow: "0 0 10px rgba(0, 0, 0, 0.1)",
  },
});

```

```

        backgroundColor: "#fff",
      },
      input: {
        marginBottom: "15px",
      },
      error: {
        color: "red",
        marginBottom: "15px",
      },
      submitButton: {
        width: "100%",
      },
    }
  });

export const LoginPage = () => {
  const classes = useStyles();
  const navigate = useNavigate();
  const [data, setData] = useState({
    email: "",
    password: "",
  });
  const [error, setError] = useState("");

  const handleSubmit = async (e: any) => {
    e.preventDefault();
    try {
      const url = "http://localhost:5000/api/auth";
      const { data: res } = await axios.post(url, data);
      localStorage.setItem("token", res.data);
      navigate("/");
      console.log(res.message);
    } catch (error: any) {
      if (
        error.response &&
        error.response.status >= 400 &&
        error.response.status < 500
      ) {
        setError(error.response.data.message);
      }
    }
  };

  const handleChange = ({
    currentTarget,
  }: React.ChangeEvent<HTMLInputElement>) => {
    setData((prevData) => ({
      ...prevData,
      [currentTarget.name]: currentTarget.value,
    }));
  };

```

```

return (
  <div className={classes.root}>
    <Box className={classes.sidemenu}>
      <form className={classes.form} onSubmit={handleSubmit}>
        <TextField
          className={classes.input}
          name="email"
          label="Email"
          type="email"
          value={data.email}
          onChange={handleChange}
          fullWidth
        />
        <TextField
          className={classes.input}
          name="password"
          label="Password"
          type="password"
          value={data.password}
          onChange={handleChange}
          fullWidth
        />
        {error && <div className={classes.error}>{error}</div>}
        <Button
          className={classes.submitButton}
          type="submit"
          variant="contained"
          color="primary"
        >
          Login
        </Button>
        <Link
          sx={{
            width: "fit-content",
            cursor: "pointer",
            my: 1,
            textTransform: "uppercase",
            fontSize: 12,
          }}
          href="/signup"
        >
          Not a member yet?
        </Link>
      </form>
    </Box>
  </div>
);
};

```

Б.8 Текст файла DocumentEditorPage.tsx

```

import Quill, { DeltaOperation, Sources } from "quill";
// import * as Quill from "quill";
import "quill/dist/quill.snow.css";
import { useCallback, useEffect, useState } from "react";
import { io, Socket } from "socket.io-client";
import { useParams } from "react-router-dom";
// import { Header } from "../../components/Header";

type WrapperRefCallback = (
  wrapper: HTMLDivElement | null
) => (() => void) | undefined;

type QuillEditor = Quill | null;
const SAVE_INTERVAL_MS = 5000;
const TOOLBAR_OPTIONS: any = [
  [{ header: [1, 2, 3, 4, 5, 6, false] }],
  [{ font: [] }],
  [{ list: "ordered" }, { list: "bullet" }],
  ["bold", "italic", "underline"],
  [{ color: [] }, { background: [] }],
  [{ script: "sub" }, { script: "super" }],
  [{ align: [] }],
  ["image", "blockquote", "code-block"],
  ["clean"],
];

export default function DocumentEditorPage() {
  const { id: documentId } = useParams<{ id: string }>();
  const [socket, setSocket] = useState<Socket | null>(null);
  const [quill, setQuill] = useState<QuillEditor>(null);
  // const [docName, setDocName] = useState<string>("");
  useEffect(() => {
    const s = io("http://localhost:3001");
    setSocket(s);
    return () => {
      s.disconnect();
    };
  }, []);

  useEffect(() => {
    if (socket == null || quill == null) return;
    socket.once("load-document", (document: any) => {
      // setDocName(document.name);
      quill.setContents(document.data);
      quill.enable();
    });
    socket.emit("get-document", documentId,
    localStorage.getItem("token"));
  }, [socket, quill, documentId]);

```

```

useEffect(() => {
  if (socket == null || quill == null) return;

  const interval = setInterval(() => {
    socket.emit("save-document", quill.getContents());
  }, SAVE_INTERVAL_MS);

  return () => {
    clearInterval(interval);
  };
}, [socket, quill]);

useEffect(() => {
  if (socket == null || quill == null) return;
  const handler: any = (delta: any) => {
    quill.updateContents(delta);
  };

  socket.on("receive-changes", handler);

  return () => {
    socket.off("receive-changes", handler);
  };
}, [socket, quill]);

useEffect(() => {
  if (socket == null || quill == null) return;
  const handler: any = (
    delta: DeltaOperation,
    _oldDelta: DeltaOperation,
    source: Sources
  ) => {
    if (source !== "user") return;
    socket?.emit("send-changes", delta);
  };

  quill?.on("text-change", handler);

  return () => {
    quill.off("text-change", handler);
  };
}, [socket, quill]);

const wrapperRef = useCallback<WrapperRefCallback>((wrapper) => {
  if (wrapper == null) return;

  wrapper.innerHTML = "";
  const editor = document.createElement("div");
  wrapper.append(editor);
  const q = new Quill(editor, {
    theme: "snow",

```

```

    modules: { toolbar: TOOLBAR_OPTIONS },
  });
  q.disable();
  q.setText("Loading...");
  setQuill(q);
  return () => {
    if (wrapper) {
      wrapper.innerHTML = "";
    }
  };
}, []);

return (
  <>
    { /* <Header title={`Document | ${docName}`} /> */ }
    <div className="container" ref={wrapperRef}>
      TextEditor
    </div>
  </>
);
}

```

Б.9 Текст файла ProtectedRoute.tsx

```

import { useEffect, ReactNode } from "react";
import { useJwt } from "react-jwt";
import { useNavigate } from "react-router-dom";

interface PrivateRouteProps {
  children: ReactNode;
}

const PrivateRoute = ({ children }: PrivateRouteProps) => {
  const token = localStorage.getItem("token") || "";
  const navigate = useNavigate();
  const { isExpired, decodedToken } = useJwt(token);
  console.log(decodedToken);
  useEffect(() => {
    if (isExpired) {
      navigate("/Login");
    }
  }, [isExpired, navigate]);

  return !isExpired ? <>{children}</> : null;
};

export default PrivateRoute;

```

Б.10 Текст файла CreateDocumentModal.tsx

```
import { Modal, Box, Typography } from "@mui/material";

interface CreateDocumentModalProps {
  open: boolean;
  handleClose: any;
}

const CreateDocumentModal = ({
  open,
  handleClose,
}: CreateDocumentModalProps) => {
  return (
    <div>
      <Modal
        open={open}
        onClose={handleClose}
        aria-labelledby="modal-modal-title"
        aria-describedby="modal-modal-description"
      >
        <Box>
          <Typography id="modal-modal-title" variant="h6"
component="h2">
            Text in a modal
          </Typography>
          <Typography id="modal-modal-description" sx={{ mt: 2 }}>
            Duis mollis, est non commodo luctus, nisi erat porttitor
ligula.
          </Typography>
        </Box>
      </Modal>
    </div>
  );
};

export default CreateDocumentModal;
```

Б.11 Текст файла Header.tsx

```
import React, { useEffect, useState } from "react";
import styles from "../Header.module.scss";
import AppBar from "@mui/material/AppBar";
import Box from "@mui/material/Box";
import Toolbar from "@mui/material/Toolbar";
import Typography from "@mui/material/Typography";
import IconButton from "@mui/material/IconButton";
import Button from "@mui/material/Button";
```

```

import MenuIcon from "@mui/icons-material/Menu";
import AccountCircle from "@mui/icons-material/AccountCircle";
import MenuItem from "@mui/material/MenuItem";
import Menu from "@mui/material/Menu";
import Drawer from "@mui/material/Drawer";
import List from "@mui/material/List";
import ListItem from "@mui/material/ListItem";
import { useNavigate } from "react-router-dom";
import { blue } from "@mui/material/colors";

interface HeaderProps {
  title: string;
}

export const Header = ({ title }: HeaderProps) => {
  const navigate = useNavigate();
  const [auth, setAuth] = useState<boolean>(true);
  const [anchorEl, setAnchorEl] = useState<null | HTMLInputElement>(null);
  const [isDrawerOpen, setIsDrawerOpen] = useState(false);

  useEffect(() => {
    const isAuthenticated = !!localStorage.getItem("token");
    setAuth(isAuthenticated);
  }, []);

  const handleMenu = (event: React.MouseEvent<HTMLInputElement>) => {
    setAnchorEl(event.currentTarget);
  };

  const handleClose = () => {
    setAnchorEl(null);
  };

  const handleLogout = () => {
    localStorage?.removeItem("token");
    setAuth(false);
    navigate("/login");
  };

  const toggleDrawer = (open: boolean) => {
    setIsDrawerOpen(open);
  };

  return (
    <Box sx={{ flexGrow: 1, width: "100%" }}>
      <AppBar position="static">
        <Toolbar>
          <Button
            size="large"
            // edge="start"
            color="inherit"

```

```

        aria-label="menu"
        sx={{ mr: 2 }}
        onClick={() => toggleDrawer(true)}
      >
        <MenuIcon />
      </Button>
    <Typography variant="h6" component="div" sx={{ flexGrow: 1 }}>
      {title}
    </Typography>

    <div>
      <IconButton
        size="large"
        aria-label="account of current user"
        aria-controls="menu-appbar"
        aria-haspopup="true"
        onClick={handleMenu}
        color="inherit"
      >
        <AccountCircle />
      </IconButton>
      <Menu
        id="menu-appbar"
        anchorEl={anchorEl}
        anchorOrigin={{
          vertical: "top",
          horizontal: "right",
        }}
        keepMounted
        transformOrigin={{
          vertical: "top",
          horizontal: "right",
        }}
        sx={{ minWidth: "100px", mx: -6 }}
        open={Boolean(anchorEl)}
        onClose={handleClose}
      >
        {auth && <MenuItem
onClick={handleClose}>Profile</MenuItem>}
        <MenuItem onClick={handleLogout}>
          {auth ? "Logout" : "Log in"}
        </MenuItem>
      </Menu>
    </div>
  </Toolbar>
</AppBar>

<Drawer
  anchor="left"
  open={isDrawerOpen}
  onClose={() => toggleDrawer(false)}

```

```

        className={styles.root__drawer}
      >
        <List sx={{ background: blue[700] }}>
          <ListItem onClick={() => navigate("/")}>
            <Typography variant="body1">Home</Typography>
          </ListItem>
          <ListItem onClick={() => navigate("/documents")}>
            <Typography variant="body1">Documents</Typography>
          </ListItem>
          <ListItem onClick={handleLogout}>
            <Typography variant="body1">Logout</Typography>
          </ListItem>
        </List>
      </Drawer>
    </Box>
  );
};

export default Header;

```

Б.12 Текст файла DocumentCardMenu.tsx

```

import * as React from "react";
import IconButton from "@mui/material/IconButton";
import Menu from "@mui/material/Menu";
import MenuItem from "@mui/material/MenuItem";
import MoreVertIcon from "@mui/icons-material/MoreVert";

type OptionType = "delete" | "edit" | "share";

type Option = {
  title: string;
  type: OptionType;
};

const options: Option[] = [
  { title: "Delete", type: "delete" },
  { title: "Edit", type: "edit" },
  { title: "Copy URL", type: "share" },
];

const ITEM_HEIGHT = 48;

interface DocumentCardMenuProps {
  setType: (type: OptionType) => void;
}

export function DocumentCardMenu({ setType }: DocumentCardMenuProps) {

```

```

const [anchorEL, setAnchorEL] = React.useState<null |
HTMLElement>(null);
const open = Boolean(anchorEL);
const handleClick = (event: React.MouseEvent<HTMLElement>) => {
  setAnchorEL(event.currentTarget);
};
const handleClose = () => {
  setAnchorEL(null);
};

const handleSelect = (option_type: OptionType) => {
  setType(option_type);
  handleClose();
};

return (
  <div>
    <IconButton
      aria-label="more"
      id="long-button"
      aria-controls={open ? "long-menu" : undefined}
      aria-expanded={open ? "true" : undefined}
      aria-haspopup="true"
      onClick={handleClick}
    >
      <MoreVertIcon />
    </IconButton>
    <Menu
      id="long-menu"
      MenuListProps={{
        "aria-labelledby": "Long-button",
      }}
      anchorEL={anchorEL}
      open={open}
      onClose={handleClose}
      slotProps={{
        paper: { style: { maxHeight: ITEM_HEIGHT * 4.5, width: "20ch"
} }},
    >
      {options.map((option) => (
        <MenuItem
          key={option.title}
          // selected={option === "Pyxis"}
          onClick={() => handleSelect(option.type)}
        >
          {option.title}
        </MenuItem>
      ))}
    </Menu>
  </div>

```

```
);
}
```

Б.13 Текст файла CreateDocumentModal.tsx

```
import { v4 as uuidv4 } from "uuid";
import { Modal, Box, Typography, Card, TextField, Button } from
"@mui/material";
import { useState } from "react";

import axios from "axios";
import { useNavigate } from "react-router-dom";

interface CreateDocumentModalProps {
  open: boolean;
  onClose?: any;
  docName: string;
  docId: string;
  setOpen: any;
}

const CreateDocumentModal = (props: CreateDocumentModalProps) => {
  const navigate = useNavigate();
  const handleClose = () => {
    props.setOpen((prev: boolean) => !prev);
    props.onClose();
  };
  const [name, setName] = useState<string>(props.docName);
  const handleSubmit = async () => {
    const url = "http://localhost:5000/api/documents";
    const request = await axios.put(url, {
      docId: props.docId,
      oldName: props.docName,
      newName: name,
    });
    handleClose();
    console.log("response: ", request);
  };
  return (
    <Modal
      open={props.open}
      onClose={handleClose}
      aria-labelledby="modal-modal-title"
      aria-describedby="modal-modal-description"
    >
      <Card
        sx={{
          width: 391,
          minHeight: 217,
```

```

        background: "white",
        position: "absolute",
        top: "50%",
        left: "50%",
        transform: "translate(-50%, -50%)",
        padding: "24px 24px 20px 24px",
        borderRadius: "8px",
    }}
    elevation={24}
>
<Typography
  id="modal-modal-title"
  variant="h6"
  component="h2"
  fontSize={22}
  fontWeight={400}
>
  Create new document
</Typography>
<form
  onSubmit={async (e: any) => {
    e.preventDefault();
    const url = "http://localhost:5000/api/documents";
    const token = localStorage.getItem("token") || "";
    const documentId = uuidv4();
    await axios.post(
      url,
      {
        doc_name: e.target.name.value,
        doc_id: documentId,
      },
      { headers: { Token: token } }
    );
    navigate(`/documents/${documentId}`);
  }}
>
  <TextField
    value={name}
    onChange={(event: React.ChangeEvent<HTMLInputElement>) => {
      setName(event.target.value);
    }}
    sx={{ width: "100%", mt: 2.5 }}
    size="small"
    variant="outlined"
    label="Name"
    name="name"
    required
  />

  <Box
    sx={{

```

```

        mt: 2,
        display: "flex",
        flexDirection: "row",
        justifyContent: "flex-end",
        gap: "14px",
      }}
    >
    <Button
      size="medium"
      variant="outlined"
      color="info"
      onClick={handleClose}
    >
      Cancel
    </Button>
    <Button
      size="medium"
      variant="contained"
      color="primary"
      onClick={handleSubmit}
      type="submit"
      disabled={
        name === "" ||
        (name === props.docName && props.docName !== "New doc")
      }
    >
      OK
    </Button>
  </Box>
</form>
</Card>
</Modal>
);
};

export default CreateDocumentModal;

```

Б.14 Текст файла DeleteDocumentModal.tsx

```

import { Modal, Box, Typography, Card, Button } from "@mui/material";
import { useState } from "react";
import axios from "axios";

interface DeleteDocumentModalProps {
  open: boolean;
  onClose?: any;
  docId: string;
  docName: string;
}

```

```

const DeleteDocumentModal = (props: DeleteDocumentModalProps) => {
  const [open, setOpen] = useState<boolean>(props.open);
  const handleClose = () => {
    setOpen((prev) => !prev);
    props.onClose();
  };

  const handleDelete = async () => {
    const url = "http://localhost:5000/api/documents";
    console.log(props.docId);
    await axios.delete(url, {
      data: {
        docId: props.docId,
      },
    });
    handleClose();
  };

  return (
    <Modal
      open={open}
      onClose={handleClose}
      aria-labelledby="modal-modal-title"
      aria-describedby="modal-modal-description"
    >
      <Card
        sx={{
          width: 391,
          minHeight: 117,
          background: "white",
          position: "absolute",
          top: "50%",
          left: "50%",
          transform: "translate(-50%, -50%)",
          padding: "24px 24px 20px 24px",
          borderRadius: "8px",
        }}
        elevation={24}
      >
        <Typography
          id="modal-modal-title"
          variant="h6"
          component="h2"
          fontSize={22}
          fontWeight={400}
        >
          Edit document
        </Typography>
        <Typography
          id="modal-modal-label"
          sx={{ mt: 1 }}

```

```

        // component="span"
        fontSize={16}
        fontWeight={400}
        color="InactiveCaptionText"
      >
        Are you sure want to delete the document?
      </Typography>
    <Box
      sx={{
        mt: 2,
        display: "flex",
        flexDirection: "row",
        justifyContent: "flex-end",
        gap: "14px",
      }}
    >
      <Button
        size="medium"
        variant="outlined"
        color="info"
        onClick={handleClose}
      >
        Cancel
      </Button>
      <Button
        size="medium"
        variant="contained"
        color="error"
        onClick={handleDelete}
      >
        Delete
      </Button>
    </Box>
  </Card>
</Modal>
);
};

export default DeleteDocumentModal;

```

Б.15 Текст файла EditModal.tsx

```

import { Modal, Box, Typography, Card, TextField, Button } from
"@mui/material";
import { useState } from "react";
import axios from "axios";

interface EditModalProps {
  open: boolean;
  handleClose?: any;

```

```

    onClose?: any;
    docName: string;
    docId: string;
  }

const EditModal = (props: EditModalProps) => {
  const [open, setOpen] = useState<boolean>(props.open);
  const handleClose = () => {
    setOpen((prev) => !prev);
    props.onClose();
  };
  const [name, setName] = useState<string>(props.docName);
  const handleSubmit = async () => {
    const url = "http://localhost:5000/api/documents";
    const request = await axios.put(url, {
      docId: props.docId,
      oldName: props.docName,
      newName: name,
    });
    handleClose();
    console.log("response: ", request);
  };
  return (
    <Modal
      open={open}
      onClose={handleClose}
      aria-labelledby="modal-modal-title"
      aria-describedby="modal-modal-description"
    >
      <Card
        sx={{
          width: 391,
          minHeight: 217,
          background: "white",
          position: "absolute",
          top: "50%",
          left: "50%",
          transform: "translate(-50%, -50%)",
          padding: "24px 24px 20px 24px",
          borderRadius: "8px",
        }}
        elevation={24}
      >
        <Typography
          id="modal-modal-title"
          variant="h6"
          component="h2"
          fontSize={22}
          fontWeight={400}
        >
          Edit document

```

```

</Typography>
<Typography
  id="modal-modal-label"
  sx={{ mt: 1 }}
  // component="span"
  fontSize={16}
  fontWeight={400}
  color="InactiveCaptionText"
>
  Enter the new name:
</Typography>
<TextField
  value={name}
  onChange={(event: React.ChangeEvent<HTMLInputElement>) => {
    setName(event.target.value);
  }}
  sx={{ width: "100%", mt: 2.5 }}
  size="small"
  variant="outlined"
/>
<Box
  sx={{
    mt: 2,
    display: "flex",
    flexDirection: "row",
    justifyContent: "flex-end",
    gap: "14px",
  }}
>
  <Button
    size="medium"
    variant="outlined"
    color="info"
    onClick={handleClose}
  >
    Cancel
  </Button>
  <Button
    size="medium"
    variant="contained"
    color="primary"
    onClick={handleSubmit}
    disabled={name === "" || name === props.docName}
  >
    OK
  </Button>
</Box>
</Card>
</Modal>
);
};

```

```
export default EditModal;
```

Б.16 Текст файла db.js

```
const mongoose = require("mongoose");
require("dotenv").config();

const connectToDatabase = () => {
  const dbUser = process.env.DB_USER;
  const dbPassword = process.env.DB_PASSWORD;
  const dbCollection = process.env.DB_COLLECTION;

  mongoose
    .connect(
      `mongodb+srv://${dbUser}:${dbPassword}@gdocscluster.l5mmayw.mongodb.net/${dbCollection}?retryWrites=true&w=majority`
    )
    .then(() => console.log("CONNECTED TO MONGO DB"))
    .catch((err) => console.log(err));
};

module.exports = connectToDatabase;
```

Б.17 Текст файла middleware.js

```
const jwt = require("jsonwebtoken");
const secretKey = process.env.JWT_PRIVATE_KEY;

const verifyToken = (req, res, next) => {
  // Get the token from the request headers, query parameters, or cookies
  const token =
    req.headers?.authorization || req.query?.token || req.cookies?.token;

  if (!token) {
    return res.status(401).json({ message: "Unauthorized: No token provided" });
  }

  jwt.verify(token, secretKey, (err, decoded) => {
    if (err) {
      return res.status(401).json({ message: "Unauthorized: Invalid token" });
    }
  });
}
```

```

    // Attach the decoded payload to the request object for further use
    req.user = decoded;
    next();
  });
};

module.exports = { verifyToken };

```

Б.18 Текст файла auth.js

```

const router = require("express").Router();
const { User } = require("../schemas/User");
const jwt = require("jsonwebtoken");
const Joi = require("joi");
const bcrypt = require("bcrypt");
require("dotenv").config();
const secretKey = process.env.JWT_PRIVATE_KEY;

const verifyToken = (token) => {
  return new Promise((resolve, reject) => {
    if (!token) {
      reject(new Error("Unauthorized: No token provided"));
    }

    jwt.verify(token, secretKey, (err, decoded) => {
      if (err) {
        reject(new Error("Unauthorized: Invalid token"));
      }

      resolve(decoded);
    });
  });
};

router.post("/", async (req, res) => {
  try {
    const { error } = validate(req.body);
    if (error)
      return res.status(400).send({ message: error.details[0].message });

    const user = await User.findOne({ email: req.body.email });
    if (!user)
      return res.status(401).send({ message: "Invalid email or password." });

    const validPassword = await bcrypt.compare(
      req.body.password,
      user.password
    );
  }
});

```

```

    if (!validPassword)
      return res.status(401).send({ message: "Invalid email or
password." });
    const token = user.generateAuthToken();
    res.status(200).send({ data: token, message: "Login successful." });
  } catch (error) {
    res.status(500).send({ message: "Internal server error." });
  }
});

router.post("/verify-token", async (req, res) => {
  const token = req.query.token;
  console.log(token, secretKey);
  try {
    const decoded = await verifyToken(token);
    res.status(200).json({ decoded });
  } catch (error) {
    res.status(401).json({ error: error.message });
  }
});

const validate = (data) => {
  const schema = Joi.object({
    email: Joi.string().email().required().label("Email"),
    password: Joi.string().required().label("Password"),
  });
  return schema.validate(data);
};

module.exports = router;

```

Б.19 Текст файла documents.js

```

const router = require("express").Router();
const { User, validate } = require("../schemas/User");
const bcrypt = require("bcrypt");
const Document = require("../schemas/Document");
const jwt = require("jsonwebtoken");
require("dotenv").config();
const secretKey = process.env.JWT_PRIVATE_KEY;

const verifyToken = (token) => {
  return new Promise((resolve, reject) => {
    if (!token) {
      reject(new Error("Unauthorized: No token provided"));
    }

    jwt.verify(token, secretKey, (err, decoded) => {
      if (err) {

```

```

        reject(new Error("Unauthorized: Invalid token"));
    }

    resolve(decoded);
  });
});
};

router.get("/", async (req, res) => {
  try {
    const token = req.query.token;
    const decoded = await verifyToken(token);
    console.log(decoded._id);
    const documents = await Document.find({ creator_id: decoded._id });
    res.status(200).send(documents);
  } catch (error) {
    console.error(error);
    res.status(500).send({ message: "Internal server error." });
  }
});

router.put("/", async (req, res) => {
  try {
    const { newName, docId } = req.body;
    const filter = { _id: docId };
    const update = { name: newName };
    console.log({ newName });
    const updatedDocument = await Document.findByIdAndUpdate(docId,
update);
    console.log({ updatedDocument });

    if (updatedDocument) {
      res.status(200).json({
        message: "Document updated successfully",
        updatedDocument,
      });
    } else {
      res.status(404).json({ message: "Document not found" });
    }
  } catch (error) {
    console.error(error);
    res.status(500).json({ message: "Internal Server Error" });
  }
});

router.delete("/", async (req, res) => {
  try {
    console.log("delete req");
    const { docId } = req.body;
    console.log(req.body);
    const filter = { _id: docId };

```

```

    await Document.deleteOne(filter);
    res.status(200).json({ message: "Document was deleted successfully"
  });
  } catch (error) {
    console.error(error);
    res.status(404).json({ message: "Document not found" });
  }
});

router.post("/", async (req, res) => {
  try {
    const { doc_id, doc_name } = req.body;
    const token = req.headers.token;
    const decoded = await verifyToken(token);
    const userId = decoded._id;
    console.log(userId, doc_id, doc_name);
    await Document.create({
      _id: doc_id,
      creator_id: userId,
      name: doc_name,
      data: "",
    });
    res.status(200).json({ message: "Document was created successfully"
  });
  } catch (error) {
    console.error(error);
    res.status(404).json({ message: "Can't create the document" });
  }
});

module.exports = router;

```

Б.20 Текст файла users.js

```

const router = require("express").Router();
const { User, validate } = require("../schemas/User");
const bcrypt = require("bcrypt");

router.post("/", async (req, res) => {
  try {
    console.log("req.body: ", req.body);
    const { error } = validate(req.body);
    if (error)
      return res.status(400).send({ message: error.details[0].message
  });

    const user = await User.findOne({ email: req.body.email });
    if (user)

```

```

        return res.status(409).send({ message: "User already registered."
    });
    const salt = await bcrypt.genSalt(Number(process.env.SALT));
    const hashPassword = await bcrypt.hash(req.body.password, salt);
    await new User({ ...req.body, password: hashPassword }).save();
    res.status(201).send({ message: "User created successfully." });
  } catch (error) {
    res.status(500).send({ message: "Internal server error." });
  }
});

module.exports = router;

```

Б.21 Текст файла Document.js

```

const { Schema, model } = require("mongoose");

const DocumentSchema = new Schema(
  {
    _id: String,
    creator_id: String,
    name: String,
    data: Object,
    created_at: { type: Date, default: Date.now },
    updated_at: { type: Date, default: Date.now },
  },
  { collection: "gdocs-documents" }
);

module.exports = model("Document", DocumentSchema);

```

Б.22 Текст файла User.js

```

const { Schema, model } = require("mongoose");
const jwt = require("jsonwebtoken");
const Joi = require("joi");
const passwordComplexity = require("joi-password-complexity");

const UserSchema = new Schema(
  {
    firstName: { type: String, required: true },
    lastName: { type: String, required: true },
    email: { type: String, required: true },
    password: { type: String, required: true },
  },
  { collection: "gdocs-users" }
);

```

```

UserSchema.methods.generateAuthToken = function () {
  const token = jwt.sign(
    { _id: this._id, firstName: this.firstName },
    process.env.JWT_PRIVATE_KEY,
    {
      expiresIn: "7d",
    }
  );
  return token;
};

const User = model("user", UserSchema);

const validate = (data) => {
  const schema = Joi.object({
    firstName: Joi.string().required().label("First Name"),
    lastName: Joi.string().required().label("Last Name"),
    email: Joi.string().email().required().label("Email"),
    password: passwordComplexity().required().label("Password"),
  });
  return schema.validate(data);
};

module.exports = { User, validate };

```

Б.23 Текст файла socket.js

```

const socketIO = require("socket.io");
const jwt = require("jsonwebtoken");
const Document = require("../schemas/Document");

const setupSocket = () => {
  const io = socketIO(3001, {
    cors: {
      origin: "*",
      methods: ["GET", "POST"],
    },
  });

  io.on("connection", (socket) => {
    socket.on("get-document", async (documentId, token) => {
      const document = await findOrCreateDocument(documentId, token);
      socket.join(documentId);
      socket.emit("load-document", document);
      socket.on("send-changes", (delta) => {
        socket.broadcast.to(documentId).emit("receive-changes", delta);
        console.log(delta);
      });
    });
    socket.on("save-document", async (data) => {
      console.log(data, Date.now().toString());
    });
  });
};

```

```

        await Document.findByIdAndUpdate(documentId, {
            data,
            updated_at: Date.now().toString(),
        });
    });
});
console.log("connected");
});
};

const defaultValue = "";

async function findOrCreateDocument(id, token) {
    console.log({ id });
    if (id == null) return;
    console.log(token);
    const user = jwt.decode(token);
    console.log(user);
    const document = await Document.findById(id);
    if (document) return document;
    return await Document.create({
        _id: id,
        name: "Test name",
        creator_id: user._id,
        data: defaultValue,
    });
}

module.exports = setupSocket;

```

Б.24 Текст файла server.js

```

const express = require("express");
const cors = require("cors");
const userRoutes = require("./routes/users");
const authRoutes = require("./routes/auth");
const docsRoutes = require("./routes/documents");
const connectToDatabase = require("./database/db");
const setupSocket = require("./socket/socket");
require("dotenv").config();

const app = express();
const PORT = 5000;

app.use(express.json());
app.use(express.urlencoded());
app.use(cors());

connectToDatabase();

```

```

setupSocket();

app.use("/api/users", userRoutes);
app.use("/api/auth", authRoutes);
app.use("/api/documents", docsRoutes);

app.post("/api/userstest", (req, res) => {
  console.log(req.body);
  res.sendStatus(200);
});

app.get("/", (req, res) => {
  res.status(200);
  res.send("Welcome to root URL of Server");
});

app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}`);
});

```

Б.25 Текст файла DocumentsListPage.tsx

```

import { useEffect, useState } from "react";
import axios from "axios";
import styles from "./DocumentsListPage.module.scss";
import { Header } from "../../components/Header";
import { DocumentCard } from "../../components/DocumentCard";
import { Grid, Container } from "@mui/material";
import { useJwt } from "react-jwt";

interface DocumentProps {
  name: string;
  creator_id: string;
  _id: string;
  created_at: string;
  updated_at: string;
}

export const DocumentsListPage = () => {
  const token = localStorage.getItem("token") || "";
  const { decodedToken }: any = useJwt(token);
  const [documents, setDocuments] = useState<DocumentProps[] | null>(null);

  const fetchDocuments = async () => {
    const url = "http://localhost:5000/api/documents";
    const token = localStorage.getItem("token") || "";
    axios.get(url, { params: { token } }).then((res) => {
      if (res.data) {
        setDocuments(res.data);
      }
    });
  };

```

```

    }
  });
};
useEffect(() => {
  fetchDocuments();
  return () => {};
}, []);
return (
  <div className={styles.root}>
    <Header title="Documents" />
    <Container maxWidth="lg" sx={{ my: 4 }}>
      <Grid container spacing={2}>
        {/* <DocumentCard title="Empty document" /> */}
        <Grid item>
          <DocumentCard
            title={"New"}
            id={"new"}
            img="https://ssl.gstatic.com/docs/templates/thumbnails/docs-blank-googlecolors.png"
            type="new"
            fetchDocuments={fetchDocuments}
            createdAt={new Date().toDateString()}
            updatedAt={new Date().toDateString()}
            author={decodedToken?.firstName}
          />
        </Grid>
        {documents &&
          documents
            .sort(
              (a, b) =>
                Number(new Date(b.updated_at)) -
                Number(new Date(a.updated_at))
            )
            .map((doc) => {
              return (
                <Grid item>
                  <DocumentCard
                    title={doc.name || doc._id}
                    id={doc._id}
                    img="https://cdn-icons-png.flaticon.com/512/888/888108.png"
                    type="existing"
                    fetchDocuments={fetchDocuments}
                    createdAt={doc.created_at}
                    updatedAt={doc.updated_at}
                    author={decodedToken?.firstName}
                  />
                </Grid>
              );
            })
        }
      </div>
    );
  // return <div>{doc.name || doc._id}</div>;

```

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

ДОДАТОК В
Слайди презентації

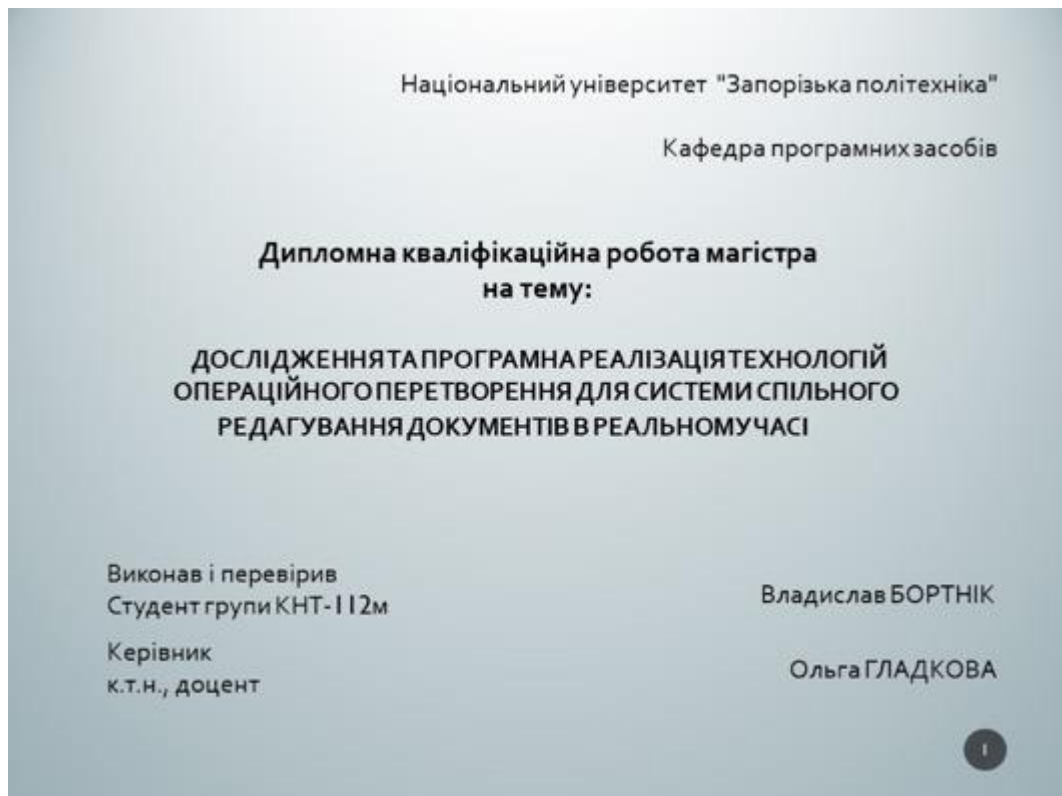


Рисунок В.1 – Слайд № 1

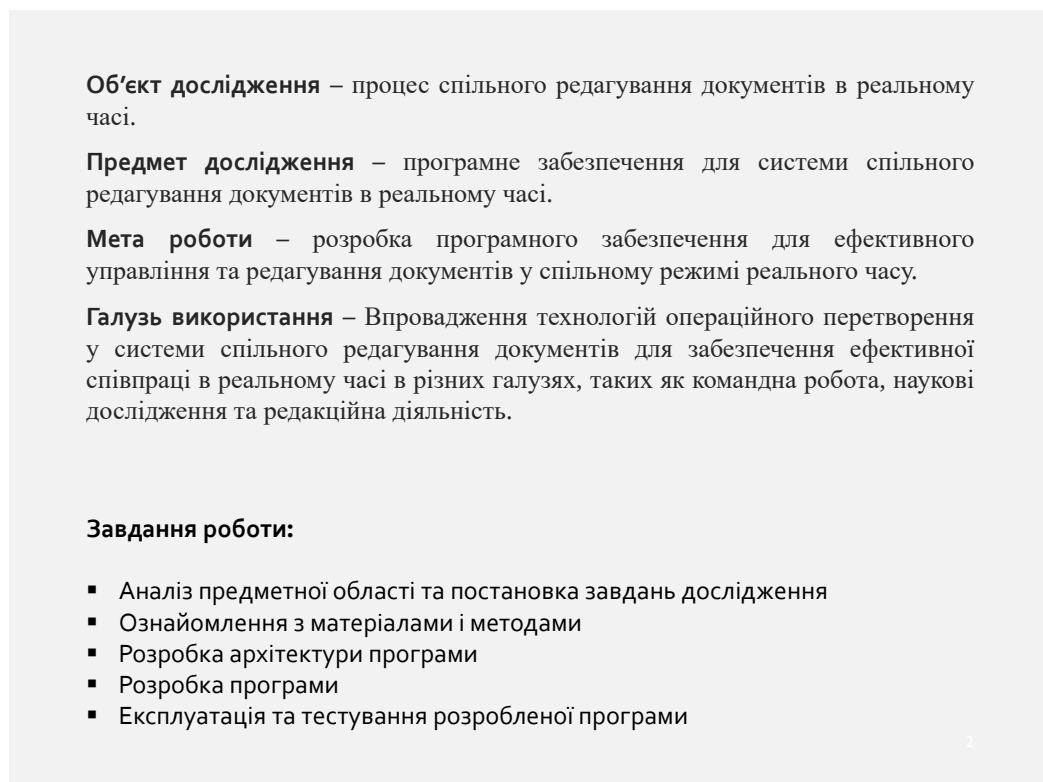
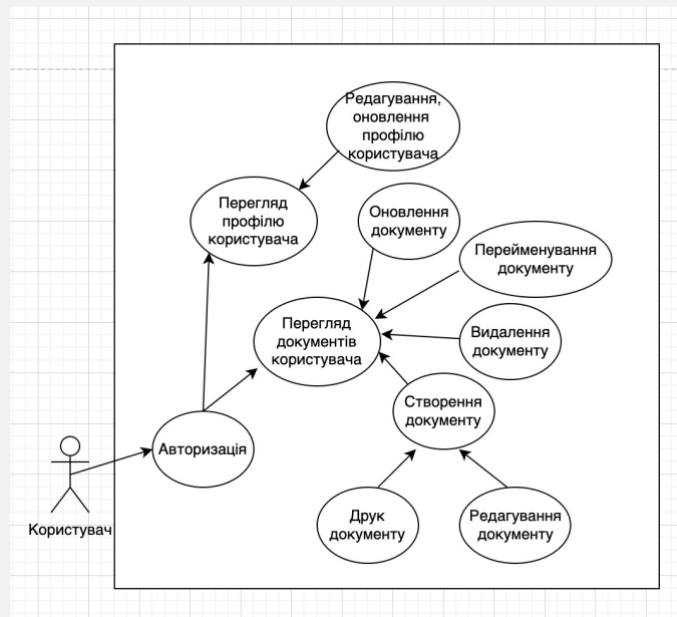


Рисунок В.2 – Слайд № 2

СЦЕНАРІЙ ВИКОРИСТАННЯ ПРОГРАМИ



3

Рисунок В.3 – Слайд № 3

ПОРІВНЯННЯ МЕТОДІВ СИНХРОНІЗАЦІЇ ТА УПРАВЛІННЯ ЗМІНАМИ

Критерії / Методи	ОТ	Техніка "Купюра"	Техніка "Лямбда-архітектура"	Системи з Прогнозуванням Конфліктів
Швидкість синхронізації змін	Висока	Помірна	Висока	Помірна
Забезпечення цілісності даних	Так	Так	Так	Так
Складність реалізації	Висока	Помірна	Висока	Помірна
Підтримка оптимістичної синхронізації	Так	Так	Так	Так
Метод вирішення конфліктів	За правилами	За версіями	-	-
Продуктивність при великій кількості користувачів	Висока	Помірна	Висока	Помірна
Популярність у веб-розробці	Висока	Обмежена	Обмежена	Обмежена

4

Рисунок В.4 – Слайд № 4

ОПЕРАЦІЙНЕ ПЕРЕТВОРЕННЯ (ОТ)

Основна ідея: Механізм синхронізації та координації різних версій документів в розподілених системах.

Мета: Уникнення конфліктів та збереження консистентності даних при паралельних редагуваннях.

Галузь використання: В умовах веб-застосунку, де можливі паралельні редагування на клієнтській та серверній сторонах.

Переваги: Уникнення конфліктів та збереження консистентності при спільному редагуванні документів у реальному часі.

Ключові аспекти:

- **Векторні Годинники:** Кожна операція отримує унікальний "векторний годинник" для визначення порядку в часі та уникнення непорозумінь.
- **Трансформації Операцій:** Конфліктуючі операції трансформуються, а не конфліктують, забезпечуючи семантичну інтеграцію.
- **Аплікація Операцій:** Кожна операція аплікується локально визначеним порядком, забезпечуючи правильну взаємодію.

5

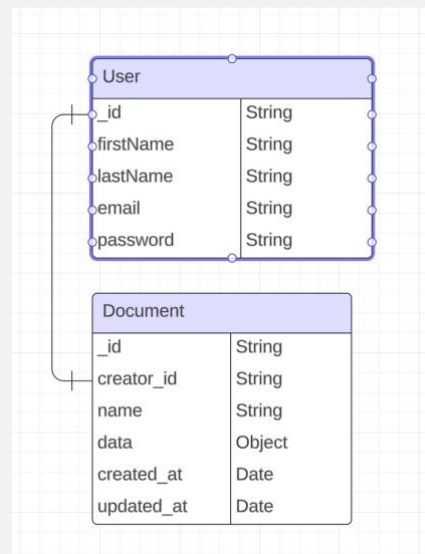
Рисунок В.5 – Слайд № 5

ПОРІВНЯННЯ JAVASCRIPT BACKEND ФРЕЙМВОРКІВ

Характеристика	Express.js	Nest.js	Характеристика	Express.js	Nest.js
Тип фреймворку	Мінімалістичний веб-фреймворк	Прогресивний Node.js фреймворк	Система адміністрування	Залежить від вибору розширень	Вбудована система адміністрування через Nest.js Admin Module
Простота використання	Висока (легкий в освоєнні)	Середня (складніший, але структурований)	Розширення та плагіни	Велика кількість розширень доступна	Залежить від вибору, менше плагінів порівняно з Express
Залежності від зовнішніх бібліотек	Мінімальні	Залежні від конкретних вимог, може використовувати додаткові модулі	Виконання фонових завдань	Через додаткові розширення	Має вбудовану підтримку фонових задач через Nest.js Queues
Вбудовані компоненти	Обмежені, що дозволяє вибір додаткових компонентів	Забезпечує структуровану архітектуру з вбудованими компонентами	Специфікації та документація	Велика кількість документацій та специфікацій	Менша в порівнянні з Express
Підхід до розробки	Гнучкий, підходить для різноманітних завдань	Стандартизований, сприяє швидкій розробці за певними конвенціями	Архітектура	Менш жорстко визначена, залишає вибір архітектури розробнику	Забезпечує структуровану архітектуру з використанням модулів та сервісів
Підтримка баз даних	Підтримка через розширення	Вбудована підтримка від Nest.js з використанням TypeORM або інших ORM	Спільнота розробників	Широка та активна спільнота	Активна, але менша порівняно з Express
Використання шаблонів	Зазвичай використовується для рендерингу HTML	Більше орієнтований на використання API, але можливість використання шаблонів існує	Швидкодія	Дуже швидкий та легкий	Швидкий, але більше витрати ресурсів порівняно з Express для простих додатків
Форма розробки	Забезпечує вільний вибір технологій та структури	Забезпечує консистентну структуру та підтримує TypeScript	Інтеграція з іншими сервісами	Легка інтеграція через Middleware	Можливість використання Middleware, але з більшою фокусуванням на структурованій архітектурі

Рисунок В.6 – Слайд № 6

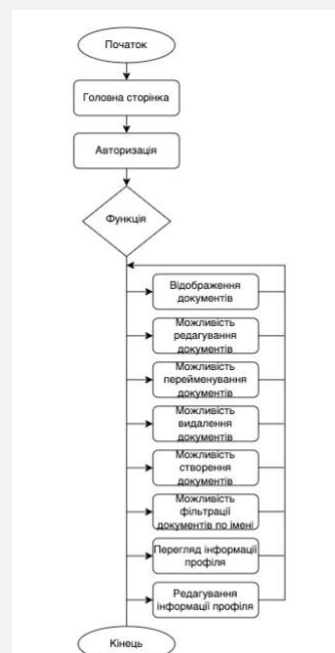
СХЕМА БАЗИ ДАНИХ



7

Рисунок В.7 – Слайд № 7

СХЕМА ФУНКЦІОНУВАННЯ СЕРВІСУ



8

Рисунок В.8 – Слайд № 8

СТРУКТУРНА СХЕМА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

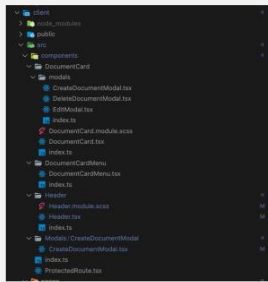


Рис. 1

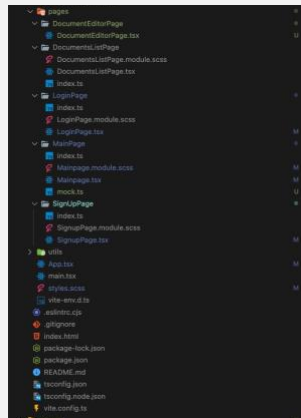


Рис. 2



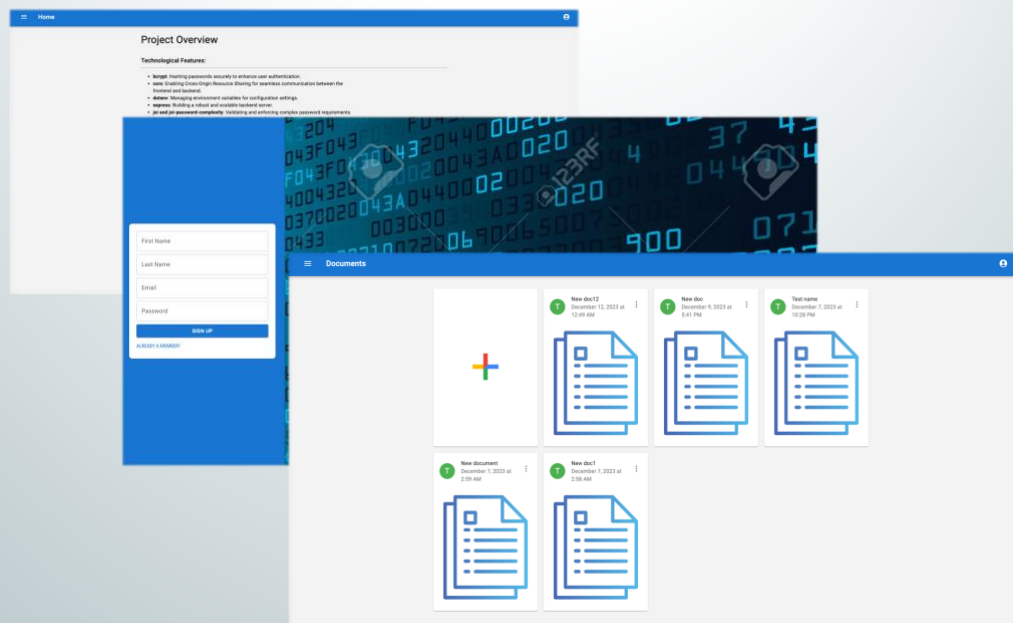
Рис. 3

* Front-end: рисунки 1, 2. Back-end: рисунок 3.

9

Рисунок В.9 – Слайд № 9

ВИГЛЯД СТОРІНОК ВЕБ-САЙТУ



10

Рисунок В.10 – Слайд № 10

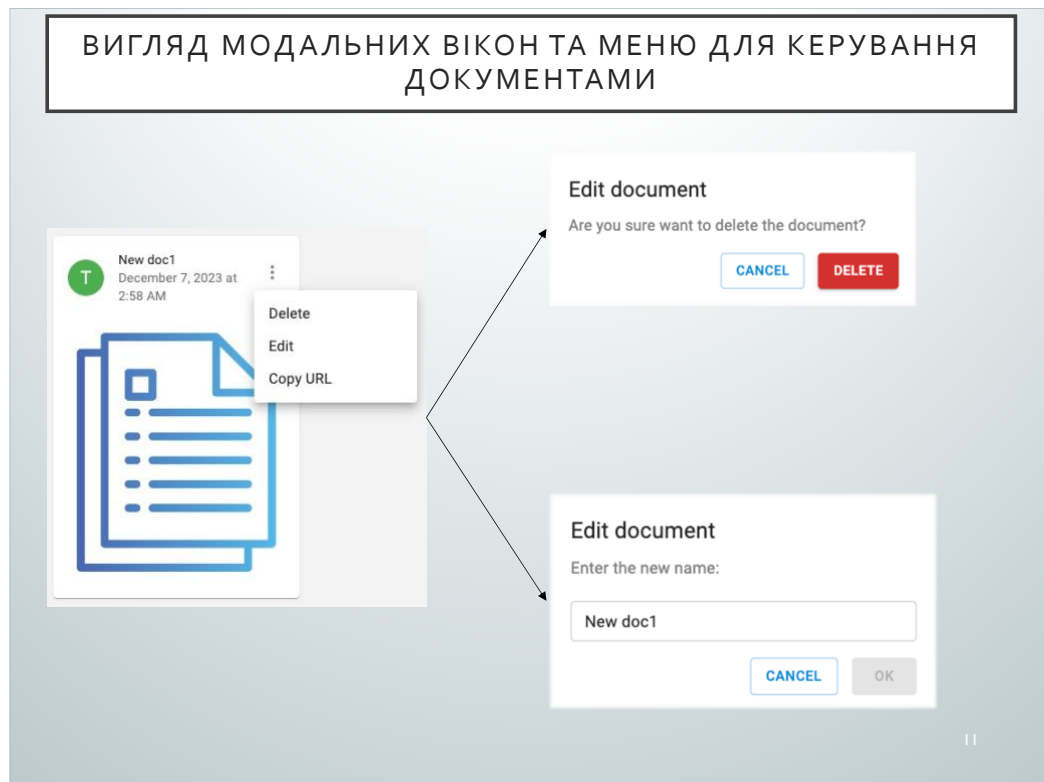


Рисунок В.11 – Слайд № 11

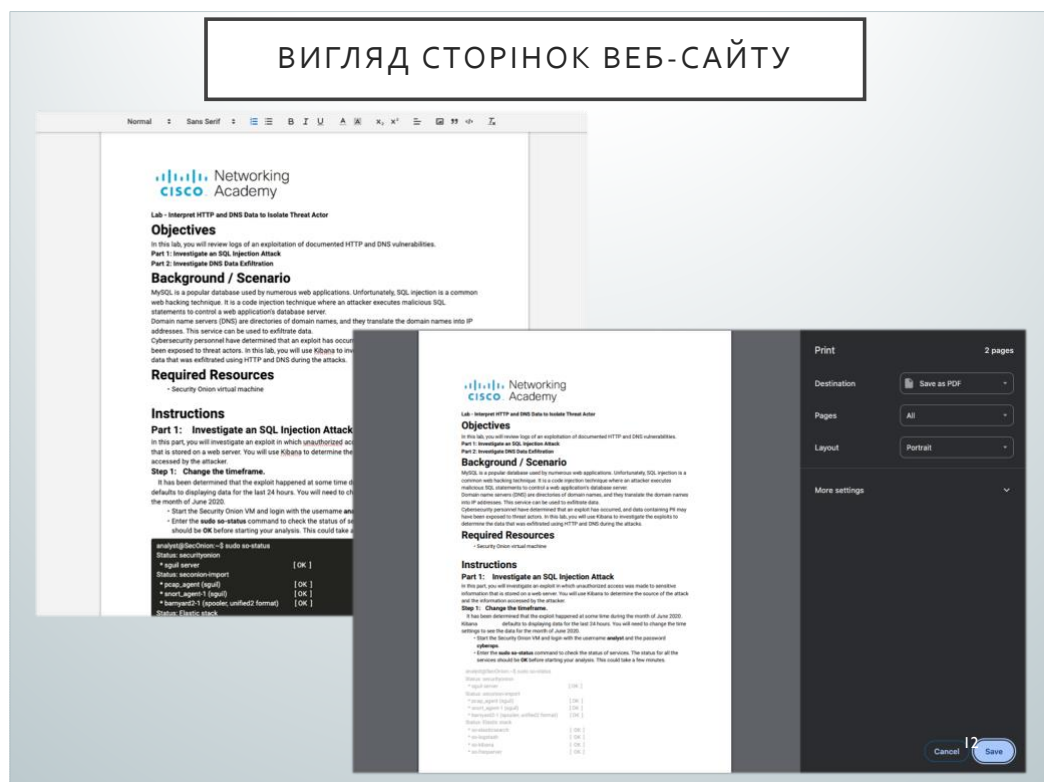


Рисунок В.12 – Слайд № 12

РЕЗУЛЬТАТИ ТЕСТУВАННЯ

В процесі розробки та вдосконалення застосунку я приділяв велику увагу його тестуванню, включаючи мануальні тести, які були проведені з великою увагою до деталей та функціональності.

Мій підхід до мануального тестування:

- *Функціональність та Взаємодія*: Крок за кроком було протестовано кожну функціональність, переконуючись, що всі елементи взаємодіють точно та передбачувано. Також, я використовував різноманітні вхідні дані для перевірки забезпечення правильності обробки кожної операції.
- *Безпека та Захист Даних*: Проведено тести, спрямовані на виявлення можливих порушень безпеки та перевірка заходів забезпечення конфіденційності даних користувачів.

Отримані результати тестування відображають високий рівень якості та надійності застосунку.

У результаті тестування всі модулі системи продемонстрували стабільну та надійну роботу. Виявлені помилки були виправлені. Тестування підтвердило, що розроблена система для редагування документів у реальному часі повністю готова до впровадження та використання в реальних умовах.

13

Рисунок В.13 – Слайд № 13

ВИСНОВКИ

За результатами виконання дипломної кваліфікаційної роботи виконано такі задачі:

- Проведено аналіз концепції сумісного редагування документів, розглянуто підходи до реалізації онлайн-редакторів тексту, проведено порівняння з програмними реалізаціями; Визначено функціональні вимоги для реалізації програмного забезпечення, обрано мову програмування JavaScript, фреймворки Express та React, БД MongoDB, створено структуру бази даних проекту;
- Реалізовано весь запланований функціонал програмного забезпечення, створено структуру веб-сервісу та описано кожний модуль системи.
- Проведено тестування розробленої системи. У результаті тестування всі модулі системи продемонстрували стабільну та надійну роботу. Виявлені помилки були виправлені. Тестування підтвердило, що розроблена система для редагування документів у реальному часі повністю готова до впровадження та використання в реальних умовах.

Розроблений веб-сервіс призначений для спільного редагування документів у реальному часі і являє собою веб-сайт з усіма необхідними функціями для використання в реальних умовах роботи команди. Програма включає в себе зручний інтерфейс адміністрування документів, пошук за іменем документів, систему шифрування даних для забезпечення конфіденційності та безпеки. Завдання дипломної кваліфікаційної роботи в результаті повністю виконано.

14

Рисунок В.14 – Слайд № 14