

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

«Система електронного документообігу»

Виконав студент групи ПЗ-20-2
БАСИСТИЙ Іван Олександрович

Керівник роботи:
ДУНЄВ Сергій Сергійович

Рецензент:
ЄФРЕМОВ Юрій Миколайович

Житомир – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення
Тетяна ВАКАЛЮК
«26» лютого 2024 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **БАСИСТИЙ Іван Олександрович**

Керівник роботи: **ДУНЄВ Сергій Сергійович**

Тема роботи: **«Система електронного документообігу»**,

затверджена Наказом закладу вищої освіти від «23» лютого 2024 р., №74/с

Вихідні дані для роботи: **процес впровадження системи електронного документообігу для цифровізації взаємодії між співробітниками компаній.**

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Дунєв С.С.	02.03.2024р.	02.03.2024р.
2	Дунєв С.С.	04.04.2024р.	04.04.2024р.
3	Дунєв С.С.	02.05.2024р.	02.05.2024р.

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається з веб-додатку системи електронного документообігу та пояснювальної записки.

Пояснювальна записка до випускної роботи містить 55 сторінок, 32 ілюстрацію та 14 таблиць.

Метою цієї роботи є створення системи електронного документообігу з використанням сучасних технологій веб-розробки та обробки даних. Це включає в себе розробку інтуїтивно зрозумілого і зручного інтерфейсу користувача, який забезпечить легкий доступ до необхідних документів та інформації.

У роботі визначено основні завдання, пов'язані з розробкою системи електронного документообігу, проведено огляд існуючих програмних рішень, аналогічних за функціональністю, а також наведено сценарії використання програмного продукту. Представлено аналіз загальної архітектури додатку, описано алгоритми взаємодії між окремими модулями, а також наведено опис об'єктної моделі додатку.

КЛЮЧОВІ СЛОВА: ДОКУМЕНТ, ЦИФРОВІЗАЦІЯ, ДОКУМЕНТООБІГ, ШАБЛОН, АРХІВ.

					ІПЗ.КР.Б – 121 – 24 – ПЗ		
Змн.	Арк.	№ докум.	Підпис	Дата			
Розроб.		І.О. Басистий			Система електронного документообігу Пояснювальна записка	Літ.	Арк.
Керівник		С.С. Дунєв					Аркушів
							3
							55
Рецензент		Ю.М. Єфремов			Житомирська політехніка, Група ІПЗ-20-2		
Зав. каф.		Т.А. Вакалюк					

ABSTRACT

The bachelor's thesis consists of a web application of the electronic document management system and an explanatory note.

The explanatory note to the thesis contains 55 pages, 32 illustrations and 14 tables.

The purpose of this work is to create an electronic document management system using modern web development and data processing technologies. This includes the development of an intuitive and convenient user interface that will provide easy access to the necessary documents and information.

The work defines the main tasks related to the development of an electronic document management system, provides an overview of existing software solutions similar in functionality, and also provides scenarios for the use of the software product. The analysis of the general architecture of the application is presented, the algorithms of interaction between individual modules are described, and the object model of the application is also described.

KEY WORDS: DOCUMENT, DIGITALIZATION, DOCUMENT CIRCULATION, TEMPLATE, ARCHIVES.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ	6
ВСТУП	7
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ	9
1.1. Постановка задачі	9
1.2. Аналіз аналогів програмного продукту	10
1.3. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення	14
Висновки до першого розділу.....	15
РОЗДІЛ 2. ПРОЕКТУВАННЯ ВЕБ-ДОДАТКУ СИСТЕМИ ЕЛЕКТРОННОГО ДОКУМЕНТООБІГУ	16
2.1. Визначення варіантів використання та об'єкто-орієнтованої структури системи	16
2.2. Розробка бази даних системи.....	22
2.3. Проектування та реалізація алгоритмів роботи системи	25
2.4. Реалізація функціоналу системи електронного документообігу ...	26
Висновки до другого розділу	35
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З СИСТЕМОЮ ЕЛЕКТРОННОГО ДОКУМЕНТООБІГУ	36
3.1. Порядок встановлення та налаштування параметрів платформи ..	36
3.2. Структура інтерфейсу системи електронного документообігу	38
3.3. Тестування веб-додатку електронного документообігу.	45
Висновки до третього розділу	51
ВИСНОВКИ.....	52
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	54
ДОДАТКИ	56

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

СЕД – система електронного документообігу;

БД – база даних;

СУБД – система управління базами даних;

API – Application Programming Interface (Інтерфейс програмування додатків);

MERN – MongoDB, ExpressJS, ReactJS та NodeJS;

CRUD – Create, Read, Update, Delete (створення, читання, оновлення, видалення);

UI – User Interface (інтерфейс користувача);

MVC – Model-View-Controller (модель-вигляд-контролер);

JSON – JavaScript Object Notation (нотація об'єктів JavaScript);

JWT – JSON Web Token (веб-токен на основі JSON);

ВСТУП

Актуальність теми

Умови сучасного світу, особливо під час криз і епідемій. Вимагаються гнучкості у роботі та можливості здійснювати обмін документами без прив'язки до офісного простору.

Впровадження системи електронного документообігу дозволяє оптимізувати процеси обробки та обміну документами, що призводить до підвищення продуктивності роботи та скороченням часових затрат.

Заміна традиційного, паперового документообігу на електронний формат також призводить до значної економії ресурсів: паперу, часу на ручну обробку, фізичного простору для зберігання.

Електронні системи дозволяють уникнути помилок, характерних для стандартного документообігу, а також надають можливість забезпечення захисту від втрати документів чи помилок при їх обробці людиною.

Мета роботи

Метою даної кваліфікаційної роботи є розробка системи електронного документообігу з використанням сучасних технологій веб-розробки та обробки даних. Це включає в себе розробку інтуїтивно зрозумілого і зручного інтерфейсу користувача, який забезпечить легкий доступ до необхідних документів та інформації.

Завдання роботи

Встановлена мета обумовлює наступні завдання:

- аналіз теми документообігу, формування вимог до продукту;
- аналіз існуючих програмних рішень для впровадження електронного документообігу, формування порівняльної таблиці;
- визначення логіки роботи продукту, формування вимог до розробки;
- вибір архітектури та засобів реалізації, їх обґрунтування;
- проектування системи, формування діаграм та опис сутностей;

- розробка програмного продукту, реалізація компонентів та моделей продукту, їх тестування.

Об’єкт дослідження

Впровадження електронного документообігу, цифровізація паперових носіїв.

Предмет дослідження

Використання технологій та засобів веб-розробки для розробки системи електронного документообігу для компаній.

Методи дослідження

В роботі використано різні методики аналізу, наприклад:

- аналіз літератури та існуючих рішень;
- вивчення робочих процесів на базі існуючих рішень;
- моделювання та протипування;
- аналіз вимог.

Використання цих методологій дозволило чітко сформулювати задачі та етапи розробки системи. Визначити місця до яких потрібно приділити увагу, а також сформулювати порівняльну характеристику з іншими ресурсами, для проведення тестування розробки та формування документації.

Практичне значення

Розроблена система може бути інтегрованою до існуючих рішень, або використовуватись, як окремий ресурс на підприємствах, де потрібно забезпечити процес цифровізації документообігу.

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

1.1. Постановка задачі

В кваліфікаційній роботі, виходячи з мети потрібно реалізувати веб-орієнтовану систему електронного документообігу, яка дозволить користувачам зберігати, створювати та редагувати цифрові варіанти документів.

Електронний документообіг — це сукупність процесів створення, оброблення, відправлення, передавання, одержання, зберігання, використання та знищення електронних документів, які виконуються із застосуванням перевірки цілісності та у разі необхідності з підтвердженням факту одержання таких документів. [1]

Для розробки системи основними етапами є:

1. Аналіз вимог та формування технічного завдання.
 - 1.1. Аналіз існуючих рішень.
 - 1.2. Аналіз прецедентів використання або впровадження.
 - 1.3. Визначення основних задач та можливостей продукту.
2. Вибір архітектури та технологій розробки.
 - 2.1. Порівняння можливих варіантів реалізації.
 - 2.2. Вибір та обґрунтування технологій.
 - 2.3. Інтеграція необхідних стандартних рішень та фреймворків.
3. Розробка та впровадження.
 - 3.1. Розробка дизайну та UI.
 - 3.2. Формування архітектури серверної частини додатку.
 - 3.3. Формування клієнтської частини додатку.
 - 3.4. Розробка конкретних компонентів та їх інтеграція в клієнтську частину.
 - 3.5. Розробка основних можливостей визначених та етапі аналізу.

4. Тестування.

4.1. Перевірка роботи клієнтської та серверної частини, як окремо одна від одної так і у поєднанні у готовому ресурсі.

4.2. Виправлення недоліків знайдених у процесу тестування.

5. Формування документації та розгортання.

В результаті розробки користувач має отримати готовий ресурс, який буде виконувати основні визначені задачі, відповідати визначеним стандартам вимогам, буде сформована документація до продукту, та надано доступ користувач до демонстраційного варіанту системи.

1.2. Аналіз аналогів програмного продукту

Тема впровадження електронного документообігу є доволі частим питанням у наших реаліях, тому існує певна кількість готових програмних рішень для забезпечення даного процесу, як застарілих так і нових. Для подальшого проектування системи проведено їх аналіз та сформовано порівняльну характеристику по кожному з них.

Розглянемо деякі з них:

1. Edin - сервіси та програмні продукти EDIN, складові програмних продуктів, завдяки яким забезпечується електронний документообіг, а також послуги сервісної підтримки. [2]
2. Vchasno - онлайн-сервіс електронного документообігу (Вчасно) - програмна продукція у вигляді онлайн-сервісу, призначена для автоматизації процесів електронного документообігу між Користувачами та Контрагентами, що передбачає підписання, надсилання, отримання та зберігання електронних документів онлайн. [3]
3. СОТА - сучасний, зручний та надійний веб-сервіс, який має функціональну модульну систему та задовільняє всі потреби вашого бізнесу. СОТА допомагає легко та швидко вирішити питання

електронного документообігу, подання звітності та ведення каси. Однозначною перевагою хмарного сервісу є те, що його не потрібно встановлювати на ваш ПК, а документи можна підписувати навіть зі смартфона. [3]

Розглянемо більш детально кожен з прикладів, переглянемо інтерфейс кожного з прикладів, наведемо переваги та недоліки кожної системи.

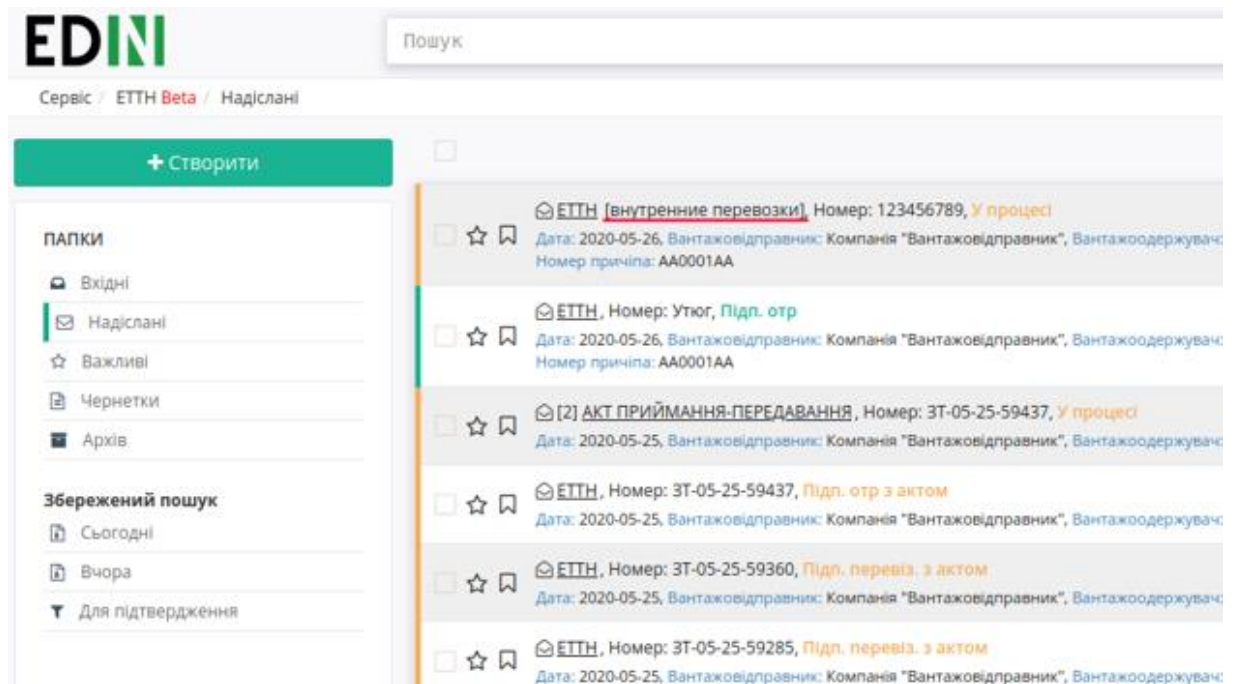


Рис.1.1 - Вигляд інтерфейсу edin

До переваг можна віднести:

1. Модульність сервісу
2. Оплата по кількості опрацьованих документів за місяць
3. Документація сервісу, з наведеним описом присутніх процесів
4. Пробний період – 30 днів.
5. Можлива інтеграція в існуючу облікову систему

Недоліками системи є:

1. Відсутня тарифікація по підпискам

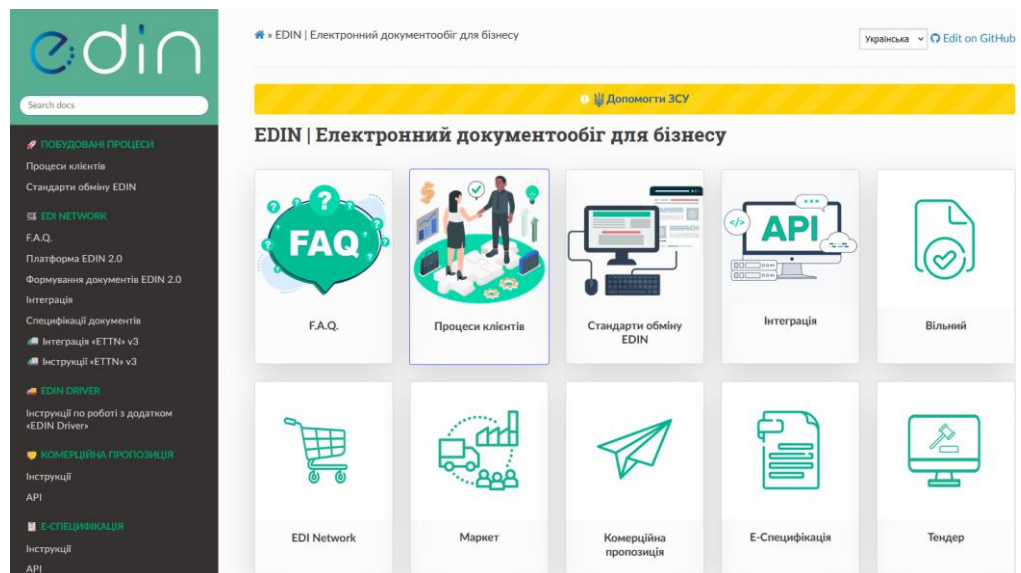


Рис.1.2 - Вигляд документації edin

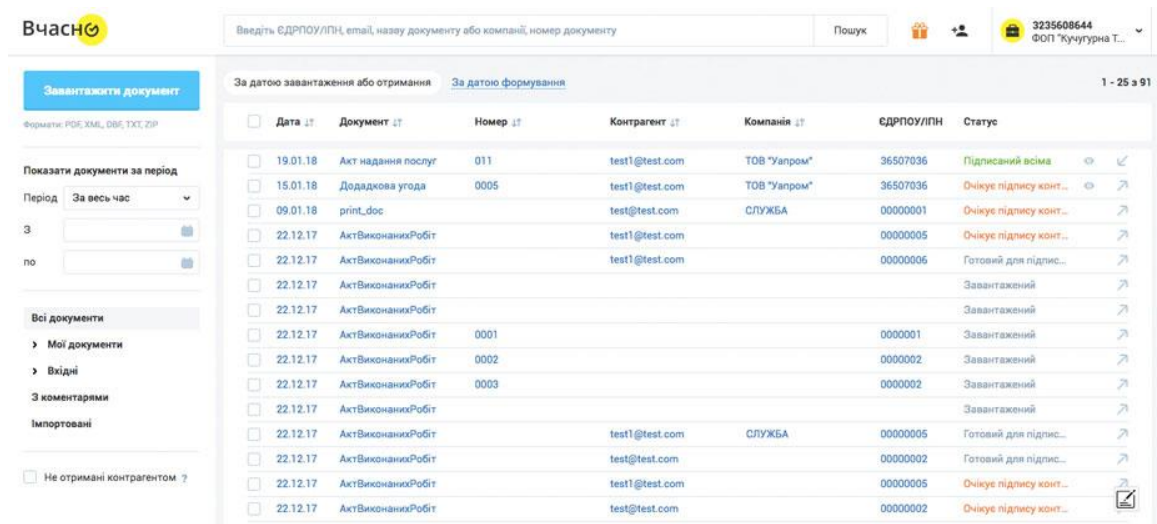


Рис.1.3 - Вигляд інтерфейсу vchasno

Цей сервіс також пропонує модульну роботу, але як окремі сервіси для конкретного процесу, є декілька відмінностей від попереднього ресурсу наведемо перелік переваг та недоліку цього варіанту.

Переваги:

1. Присутні конкретні тарифи для користувачів на рік
2. Можлива інтеграція в існуючі системи, чітко визначені ціни
3. Доступний пробний період
4. Присутня документація по API для інтеграції

Недоліки:

1. Відсутня документація для користувача

Розглянемо останній наведений приклад – COTA (SOTA BUN)

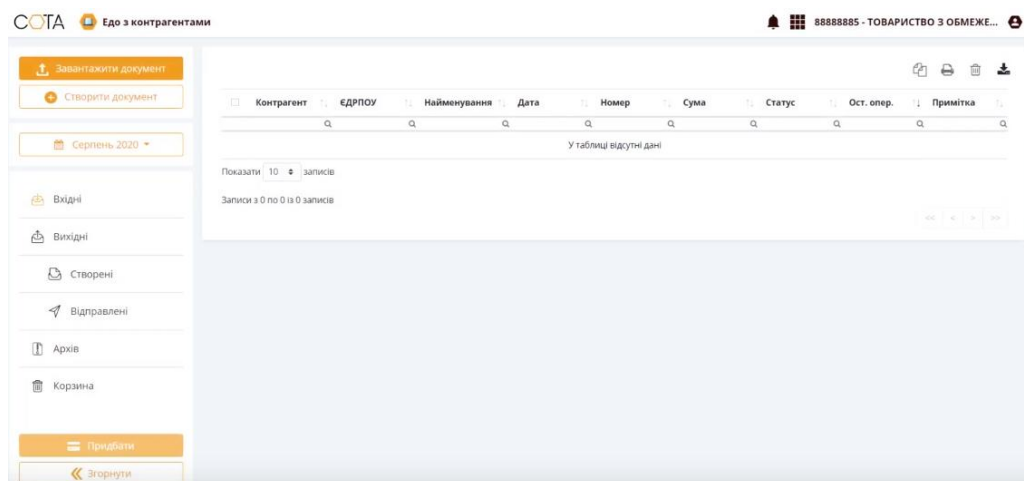


Рис.1.4 - Вигляд інтерфейсу COTA

Переваги:

1. Модульність сервісу
2. Присутні відео-уроки для користувачів
3. Привабливий інтерфейс

Недоліки:

1. Відсутня документація для користувача
2. Відсутня можливість інтеграції в інші системи

Ознайомившись з цими прикладами, складемо порівняльну таблицю (таблиця 1.1.), окресливши основні спільні риси, що формують розуміння, на що першочергово потрібно звернути увагу.

Таблиця 1.1

Таблиця порівняння систем електронного документообігу

	Edin	Вчасно	COTA
1	2	3	4
Основні функції	Створення, редагування, зберігання, передача документів	Створення, редагування, зберігання, передача документів	Створення, редагування, зберігання, передача документів
Розгортання системи	Веб-додаток	Веб-додаток	Веб-додаток

Продовження таблиці 1.1

1	2	3	4
Можливість інтеграції	+	+	-
Модульність	Модулі сервісу	Окремі сервіси	Модулі сервісу
Документація	+	Тільки для API	Відео-уроки
Цінова політика	Передплата	Передплата	Передплата

Провівши дослідження можемо визначити основні задачі, які повинна виконувати система, сформувавши загальне поняття про систему ознайомившись з вимогами від законодавства.

На основі наведених даних, вирішено створити веб-додаток, який буде виконувати такі функції: створення, зберігання, редагування, передача документів. Dodatok повинен мати зручний користувацький інтерфейс, та бути інтуїтивно зрозумілим.

1.3. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

Враховуючи поставлену задачу сервіс буде реалізований у вигляді централізованого додатку[5], в основі буде клієнт-серверна архітектура[6].

Для розробки буде використано MERN стек - це набір технологій, який допомагає розробникам створювати надійні та масштабовані веб-додатки за допомогою JavaScript. Аббревіатура «MERN» означає MongoDB, Express, React і Node.js, причому кожен компонент відіграє свою роль у процесі розробки. MongoDB служить орієнтованою на документи базою даних, яка може ефективно зберігати дані у форматі JSON. Express — це структура веб-програм, яка надає набір функцій для оптимізації збирання веб-програм. React — це інтерфейсна бібліотека JavaScript, яка пропонує великий набір інструментів для створення інтерфейсів користувача. Node.js — це середовище виконання для виконання коду JavaScript на стороні сервера, координуючи зв'язок між зовнішніми та внутрішніми компонентами. [7]

В основу додатку також буде використано концепцію MVC, це - архітектурний шаблон програмування, який використовується для побудови веб-додатків та інших програмних систем. Він допомагає розділити логіку програми на три основні компоненти: Model (Модель), View (Вид) і Controller (Контролер). Кожен з цих компонентів відповідає за виконання певної функції в програмі, а їх взаємодія регулюється певними правилами. [8]

Висновки до першого розділу

Ознайомившись з метою дослідження, сформовано задачі, які потрібно виконати у ході кваліфікаційної роботи.

Для більш глибокого занурення в тему, проведено аналіз існуючих систем, які присутні зараз, чи колись використовувались. Також ознайомились з нормативно правовим забезпеченням по даній темі.

Сформовано порівняльну характеристику аналогів, що дало змогу визначити мінімальні вимоги до реалізації, виділено перелік питань на які потрібно звернути увагу.

Розглянувши всі наведені вимоги, обрано засоби розробки та архітектуру додатку.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ВЕБ-ДОДАТКУ СИСТЕМИ ЕЛЕКТРОННОГО ДОКУМЕНТООБІГУ

2.1. Визначення варіантів використання та об'єкто-орієнтованої структури системи

Для визначення варіантів використання веб-додатку наведемо перелік вимог до системи. Ці вимоги допоможуть визначити основні напрямки взаємодії сервісу з користувачем, допоможе спланувати роботу. Задовільнивши всі умови додаток зможе забезпечити безперебійну роботу та виконання всіх поставлених задач.

Вимоги користувачів:

1. Можливість створення документів.
2. Можливість редагувати створені документи.
3. Можливість надавати доступ, або архівувати документ.
4. Можливість перегляду доступних документів.
5. Можливість створювати підрозділи та взаємодіяти з ними.
6. Можливість погоджувати документи.
7. Можливість переглядати інформацію про свій підрозділ, а також документи створені підрозділом.
8. Можливість додавати модераторів до підрозділів.

Характеристика об'єкта комп'ютеризації:

Користувач сервісу буде мати можливість переглядати інформацію про себе, свій підрозділ, документи, які створив сам користувач, а також ті до яких він отримав доступ. Можливість пошуку, обробки та аналізу документів.

Функціональні вимоги:

1. Автентифікація та авторизація користувачів.
2. Створення, перегляд, редагування та архівація документів.
3. Пошук та фільтрація документів.
4. Створення підрозділів, налаштування доступів до них та документів.

Нефункціональні вимоги:

1. Сприйняття

- 1.1. Час, необхідний для навчання звичайних користувачів – 5-8 годин, а для навчання просунутих користувачів – 4 години.
- 1.2. Час відгуку для типових задач – не більше 3 секунд, для складних завдань – не більше 15 секунд.

2. Надійність.

- 2.1. Доступність – час, потрібний для обслуговування системи не повинен перевищувати 15% від загального часу роботи.
- 2.2. Максимальна норма помилок та дефектів в роботі системи – 1 помилка на 1000 запитів користувача.

3. Продуктивність.

- 3.1. Система повинна підтримувати до 75 одночасно працюючих користувачів, пов'язаних із загальною базою даних.

4. Можливість експлуатації.

- 4.1. Масштабування – сервіс повинен мати можливість збільшувати кількість користувачів, додавання нових підрозділів таким чином, щоб це не вплинуло негативно на його роботу.

Системні вимоги:

1. Вимоги до середовища виконання:

Система повинна задовольняти вимогам на комп'ютері, що знаходиться в наступній мінімальній комплектації:

- 4 ГБ ОЗУ.
- Процесор Intel I5 шостого покоління (або вище) та аналоги, з тактовою частотою від 1,6 GHz.
- Операційна система: будь-яка система з останніми оновленнями.

2. Вимоги до СУБД та доступу до даних:

- У ядрі системи повинна бути представлена СУБД нереляційного доступу.

					ІПЗ.КР.Б – 121 – 24 – ПЗ	
						17

Для візуалізації всіх наведених вимог, наведемо UML [9] діаграму варіантів використання (рис.2.1).

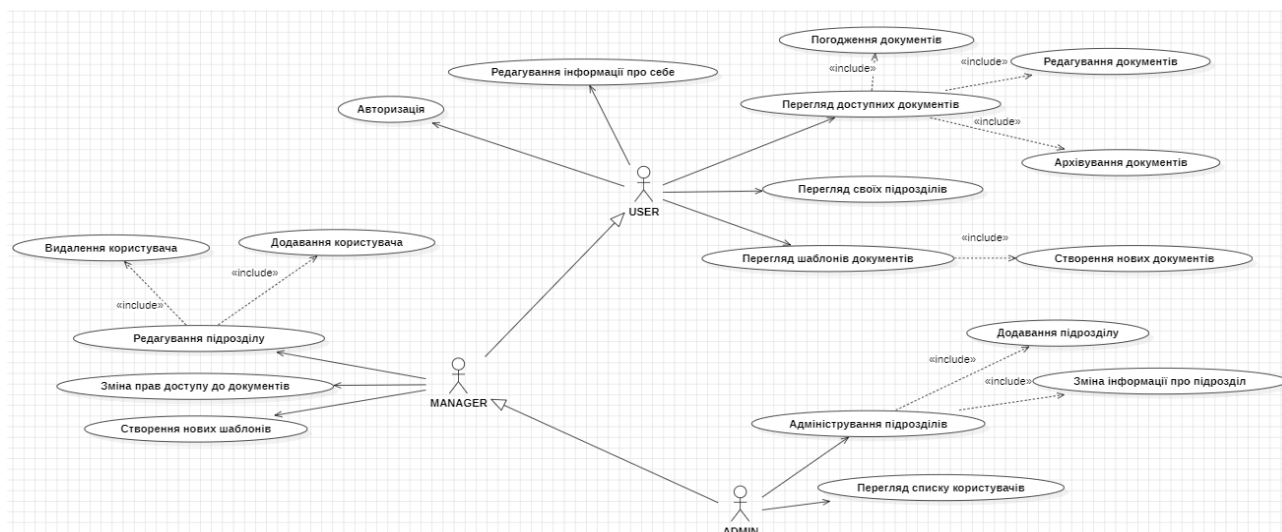


Рис.2.1 - Діаграма варіантів використання веб-додатку.

Опишемо прецеденти, які відображені у діаграмі варіантів використання (таблиця 2.1 – 2.10).

Таблиця.2.1

Опис прецеденту «Реєстрація, авторизація»

Варіант використання	Реєстрація, авторизація
Основні актори	Користувач, Модератор, Адміністратор

Продовження таблиці 2.1

Опис	Користувач може здійснити вхід у свій акаунт для подальшої роботи, або зареєструватись, якщо у нього немає акаунту.
------	---

Таблиця.2.2

Опис прецеденту «Редагування інформації про себе»

Варіант використання	Редагування інформації про себе
Основні актори	Користувач, Модератор, Адміністратор
Опис	Користувач у разі потреби може вносити правки про себе, у своєму профілі.

Таблиця.2.3

Опис прецеденту «Перегляд доступних документів»

Варіант використання	Перегляд доступних документів
Основні актори	Користувач, Модератор, Адміністратор
Опис	Користувач може перейти до списку доступних для нього документів, відкрити їх та погодити, або відредагувати, архівувати.

Таблиця.2.4

Опис прецеденту «Перегляд своїх підрозділів»

Варіант використання	Перегляд своїх підрозділів
Основні актори	Користувач, Модератор, Адміністратор
Опис	Користувач може перейти до списку доступних підрозділів, перейти до кожного з них та переглянути інформацію про конкретний.

Таблиця.2.5

Опис прецеденту «Редагування підрозділу»

Варіант використання	Редагування підрозділу
Основні актори	Модератор, Адміністратор
Опис	Модератор може вносити зміни у підрозділи, додавати та видаляти користувачів з них. Змінювати інформацію про підрозділ.

Таблиця.2.6

Опис прецеденту «Зміна прав доступу до документів»

Варіант використання	Зміна прав доступу до документів
Основні актори	Модератор, Адміністратор
Опис	У документах до яких має доступ модератор він може додавати нових користувачів

Таблиця.2.7

Опис прецеденту «Створення нових шаблонів»

Варіант використання	Створення нових шаблонів
Основні актори	Модератор, Адміністратор
Опис	Модератор може створювати шаблони, з яких в подальшому користувачі зможуть створювати нові документи.

Опис прецеденту «Адміністрування підрозділів»

Варіант використання	Адміністрування підрозділів
Основні актори	Адміністратор
Опис	Адміністратор може створювати нові підрозділи, додавати до них модераторів

Опис прецеденту «Перегляд списку користувачів»

Варіант використання	Перегляд списку користувачів
Основні актори	Адміністратор
Опис	Адміністратор може переглядати список всіх існуючих користувачів, а також підрозділів.

Розглянувши варіанти використання, та функціональні вимоги, можемо навести діаграму класів, які будуть використані при розробці веб-додатку (рис.2.2).

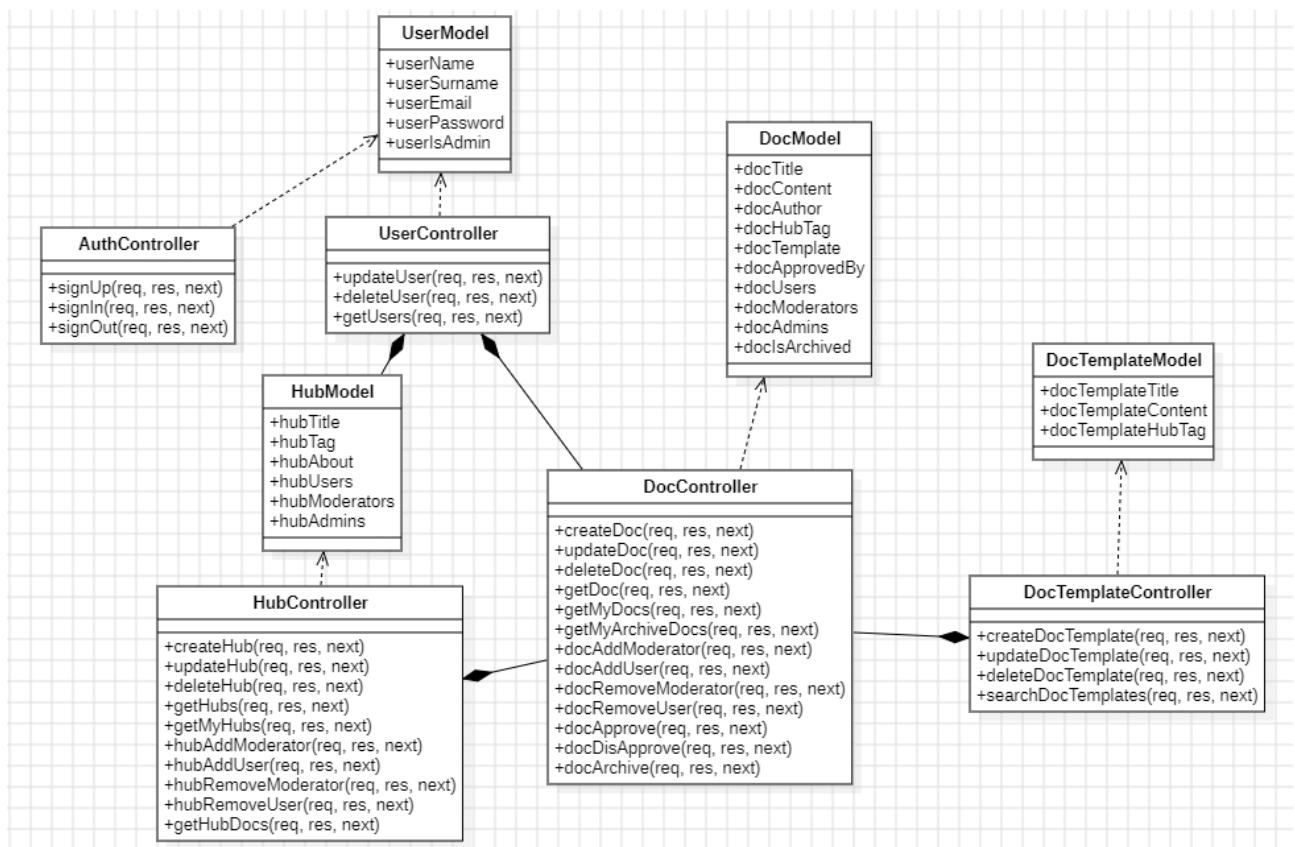


Рис.2.2 - Діаграма класів веб-додатку.

На діаграмі можемо побачити реалізацію MVC, показані моделі бази даних, та контролери, які відповідають за обробку інформації.

Розглянемо більше детально кожну модель та контролери.

Моделі:

1. UserModel – зберігає інформацію про користувача.
2. HubModel – зберігає інформацію про підрозділи.
3. DocModel – зберігає інформацію про документ.
4. DocTemplateModel – зберігає інформацію про створені шаблони.

Контролери:

1. UserController – відповідає за взаємодію з користувачем.
 - 1.1. updateUser() – оновлення даних користувача.
 - 1.2. deleteUser() – видалення користувача з бази.
 - 1.3. getUsers() – отримання списку користувачів
2. AuthController – відповідає за процес реєстрації та авторизації користувача.
 - 2.1. signUp() – реєстрація нового користувача.
 - 2.2. signIn() – авторизація.
 - 2.3. signOut() – вихід користувача з системи.
3. HubController – виконує дії з підрозділами.
 - 3.1. createHub() – створення нового підрозділу.
 - 3.2. updateHub() – оновлення даних про підрозділ.
 - 3.3. deleteHub() – видалення підрозділу.
 - 3.4. getHubs(), getMyHubs() – отримання списку підрозділів.
 - 3.5. hubAddModerator(), hubRemoveModerator() - додавання/видалення модератора.
 - 3.6. hubAddUser(), hubRemoveUser() – додавання та видалення користувачів підрозділу.
 - 3.7. getHubDocs() – отримання списку документів підрозділу.
4. DocTemplateController – виконує дії з шаблонами документів.

- 4.1. createDocTemplate() – створення шаблону для документів.
- 4.2. updateDocTemplate() – редагування вмісту шаблону.
- 4.3. deleteDocTemplate() – видалення шаблону.
- 4.4. searchDocTemplates() – пошук шаблонів для конкретного документу.
- 5. DocController – виконує дії з документами
 - 5.1. createDoc() – створення документу.
 - 5.2. updateDoc() – редагування документу.
 - 5.3. deleteDoc() – видалення документу.
 - 5.4. getDoc() – отримання конкретного документу.
 - 5.5. getMyDocs() – отримання всіх доступних для користувача документів.
 - 5.6. getMyArchiveDocs() – отримання всіх документів в архіві, до яких має доступ користувач.
 - 5.7. docAddModerator(), docRemoveModerator() – додавання/видалення модератора в документі.
 - 5.8. docAddUser(), docRemoveUser() – додавання, видалення користувача в документі.
 - 5.9. docApprove(), docDisApprove() - погодження документу.
 - 5.10. docArchive() - архівування документів.

2.2. Розробка бази даних системи

Зберігання інформації буде відбуватись у базі даних, нереляційної моделі, а саме – MongoDB. [10] Тут інформація зберігається в “колекціях”, а записи в них “документи”. Тобто база даних буде містити 4 колекції для Користувачів, Підрозділів, Документів, Шаблонів, розглянемо загальну схему бази даних (рис.2.3), де показано поля за якими буде відбуватись зв’язок між колекціями, а також опишемо кожен з колекцій (таблиця 2.11-2.14).

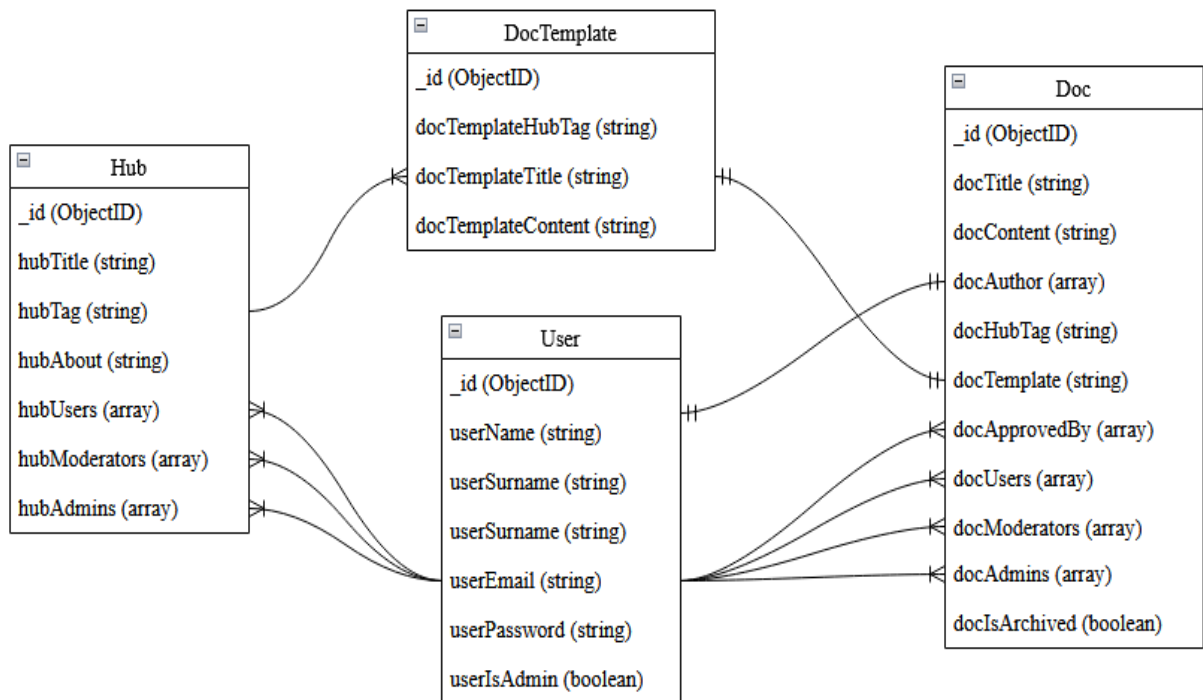


Рис.2.3 – Схема бази даних.

Таблиця.2.11

Опис колекції «Docs»

Поле	Тип даних	Опис
_id	ObjectId	Ідентифікатор документу
docTitle	String	Назва документу
docContent	String	Вміст документу
docAuthor	Array	Автор документу
docHubTag	String	Підрозділ в якому створений документ
docTemplate	String	Шаблон з якого створений документ
docApprovedBy	Array	Перелік користувачів, які погодили документ
docUsers	Array	Перелік користувачів, які мають доступ
docModerators	Array	Перелік модераторів документу
docAdmins	Array	Перелік адміністраторів документу
docIsArchived	Boolean	Перевірка чи документ архівовано

Таблиця.2.12

Опис колекції «Users»

Поле	Тип даних	Опис
_id	ObjectId	Ідентифікатор документу
userName	String	Ім'я користувача
userSurname	String	Прізвище користувача
userEmail	Array	Електронна пошта користувача
userPassword	String	Пароль користувача
userIsAdmin	String	Перевірка чи користувач є адміністратором системи

Таблиця.2.13

Опис колекції «Hubs»

Поле	Тип даних	Опис
_id	ObjectId	Ідентифікатор документу
hubTitle	String	Назва підрозділу
hubTag	String	Абревіатура підрозділу
hubAbout	String	Опис підрозділу
hubUsers	Array	Перелік користувачів підрозділу
hubModerators	Array	Перелік модераторів підрозділу
hubAdmins	Array	Перелік адміністраторів підрозділу

Таблиця.2.14

Опис колекції «DocTemplates»

Поле	Тип даних	Опис
_id	ObjectId	Ідентифікатор документу
docTemplateTitle	String	Назва шаблону
docTemplateContent	String	Вміст шаблону
docTemplateHubTag	String	Абревіатура підрозділу, де створено

2.3. Проектування та реалізація алгоритмів роботи системи

В системі передбачено що існує два типи користувачів, звичайний та адміністратор, роль модератора передбачена в конкретних документах або підрозділах, розглянемо алгоритм роботи кожного з них після авторизації.

Користувач:

1. Перегляд доступних підрозділів, документів, архіву.

1.1. В підрозділі може переглядати список користувачів, модераторів, шаблонів, може створити документ вибравши один з доступних шаблонів.

1.2. В списку документів користувач може знайти необхідний та перейти до сторінки документу.

1.3. В документі користувач може вносити в нього корективи, погодити або перемістити в архів.

2. Перегляд та редагування інформації про себе у профілі.

Модератор, наслідує весь функціонал користувача та отримує додатковий на етапах перегляду підрозділу, а саме, редагування підрозділу, створення шаблонів. На сторінці документу, якщо користувач є модератором цього документу він може додавати нових користувачів до нього.

Адміністратор, має весь функціонал попередніх ролей, а також має доступ до адміністративної панелі, де може переглянути список всіх користувачів, а також підрозділів. Адміністратор може створювати нові підрозділи та додавати модераторів як до підрозділів, так і до документів окремо.

Для опису алгоритму роботи наведемо діаграму активності (рис.2.4), на ній зображено основний цикл, який може пройти користувач, також показано в який момент користувач може відійти від алгоритму, для внесення змін в документи, або підрозділи, якщо він має доступ до цих дій.

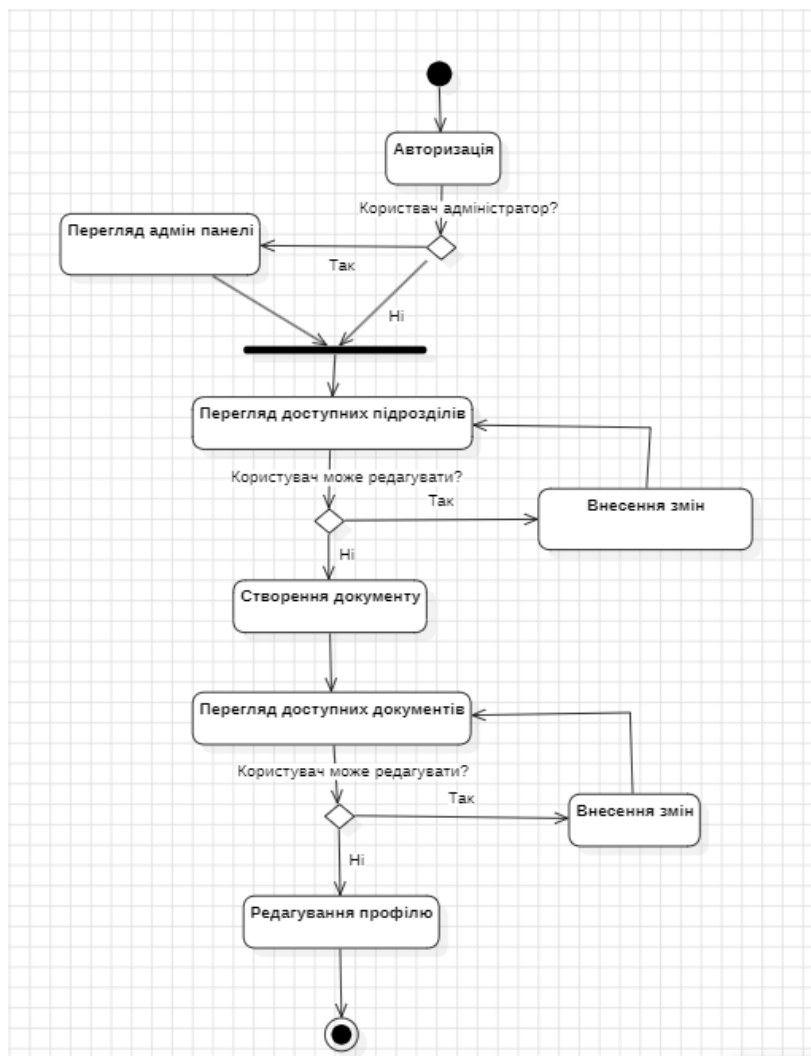


Рис.2.4 - Діаграма активностей.

2.4. Реалізація функціоналу системи електронного документообігу

Робота веб-додатку буде починатись з процесу авторизації користувача, першим кроком є реєстрація нового акаунту, після чого користувач зможе авторизуватись в свій акаунт. В процесі авторизації створюється Json Web Token [11], в якому зберігається інформація про користувача, що на наступних етапах дозволить ідентифікувати його. Наведемо лістинг коду цього контролера.

```

export const signIn = async (req, res, next) => {
  const { userEmail, userPassword } = req.body
  try {
    const validUser = await User.findOne({ userEmail: userEmail })
    if (!validUser) return next(errorHandler(404, 'Користувач не знайдений'))
  }
}

```

```

const validPassword = bcryptjs.compareSync(
  userPassword,
  validUser.userPassword,
)
if (!validPassword) return next(errorHandler(401, 'Невірні облікові дані'))

const token = jwt.sign(
  {
    id: validUser._id,
    userName: validUser.userName,
    userSurname: validUser.userSurname,
    userEmail: validUser.userEmail,
    userIsAdmin: validUser.userIsAdmin,
  },
  process.env.JWT_TOKEN,
)
const { userPassword: pass, ...rest } = validUser._doc

res.cookie('access_token', token, { httpOnly: true }).status(200).json(rest)
} catch (error) {
  next(errorHandler(500, 'Помилка входу в систему'))
}
}

```

З лістингу можна побачити, що значення, яке використовується для формування токена збережено в env файлі, цим самим забезпечується додаткова безпека для даних. [12]

Після авторизації, в cookie файлах користувача [13], буде збережено його токен, в подальшому за цим токеном він зможе отримати доступ до сайту без потреби проходити авторизацію повторно. Для перевірки токена, розробимо окремий елемент, який буде проводити верифікацію токена, який зберігається у користувача.

```

import jwt from 'jsonwebtoken'
import { errorHandler } from './error.js'
import dotenv from 'dotenv'

dotenv.config()

export const verifyToken = (req, res, next) => {
  const token = req.cookies.access_token
  if (!token) return next(errorHandler(401, 'Unauthorized'))
  jwt.verify(token, process.env.JWT_TOKEN, (err, user) => {
    if (err) return next(errorHandler(403, 'Forbidden'))
    req.user = user
    next()
  })
}

```

```
}
```

Після авторизація користувач потрапляє на сторінку з переліком його документів, для цього в контролері реалізовано функцію, яка виконує запит до бази даних для отримання переліку документів доступних для користувача.

```
export const getMyDocs = async (req, res, next) => {
  if (!req.user) return next(errorHandler(401, 'Неавторизовано'))
  try {
    const sortDirection = req.query.sort === 'asc' ? 1 : -1
    const docsList = await Doc.find({
      docIsArchived: false,
      $or: [
        { 'docAdmins._id': new mongoose.Types.ObjectId(req.user.id) },
        { 'docModerators._id': new mongoose.Types.ObjectId(req.user.id) },
        { 'docUsers._id': new mongoose.Types.ObjectId(req.user.id) },
      ],
    }).sort({ createdAt: sortDirection })
    const totalDocs = docsList.length
    res.status(200).json({ docs: docsList, totalDocs })
  } catch (error) {
    next(error)
  }
}
```

В цьому фрагменті коду також наведено використання `mongoose`, це бібліотека для Node.js, яка надає зручний спосіб моделювання даних для використання з MongoDB. Вона дозволяє вам визначати схеми для даних, створювати моделі на їх основі та виконувати операції з базою даних MongoDB з використанням цих моделей, розглянемо одну з них, на прикладі Users.

```
import mongoose from 'mongoose'

const userSchema = new mongoose.Schema(
  {
    userName: {
      type: String,
      required: true,
      unique: false,
    },
    userSurname: {
      type: String,
      required: true,
      unique: false,
    },
    userEmail: {
      type: String,
      required: true,
```

```

    unique: true,
  },
  userPassword: {
    type: String,
    required: true,
    unique: false,
  },
  userIsAdmin: {
    type: Boolean,
    required: true,
    unique: false,
  },
},
{ timestamps: true },
)

```

```
const User = mongoose.model('User', userSchema)
```

```
export default User
```

Для того, щоб відобразити отриману інформацію потрібно спочатку створити маршрутизацію для кожної окремої функції.

```
import express from 'express'
import { signIn, signUp, signOut } from '../controllers/auth.controller.js'
```

```
const router = express.Router()
```

```
router.post('/signup', signUp)
router.post('/signin', signIn)
router.get('/signout', signOut)
```

```
export default router
```

На прикладі контролера авторизації бачимо, як формується маршрутизація для функцій, використовуючи express.Router [14]

Розглянемо реалізацію інших компонентів веб-додатку.

```
import express from 'express'
import mongoose from 'mongoose'
import dotenv from 'dotenv'
import userRouter from './routes/user.routes.js'
import authRouter from './routes/auth.routes.js'
import hubRouter from './routes/hub.routes.js'
import docRouter from './routes/doc.routes.js'
import docTemplateRouter from './routes/docTemplate.routes.js'
import cookieParse from 'cookie-parser'
```

```
dotenv.config()
```

```
try {
  await mongoose.connect(process.env.MONGO_URI)
```

```

    console.log('MongoDB connected')
  } catch (error) {
    console.error('MongoDB connection error:', error)
  }
  const app = express()
  app.use(express.json())
  app.use(cookieParse())

  app.listen(3000, () => {
    console.log('Server started')
  })

  app.use('/api/user', userRouter)
  app.use('/api/auth', authRouter)
  app.use('/api/hub', hubRouter)
  app.use('/api/doc', docRouter)
  app.use('/api/template', docTemplateRouter)

```

Налаштування роботи серверної частини додатку, налаштування шляхів доступу до кожного компоненту.

Для обробки помилок веб-додатку додамо окремий модуль для виведення повідомлення про помилку.

```

export const errorHandler = (statusCode, message) => {
  const error = new Error();
  error.statusCode = statusCode;
  error.message = message;
  return error;
};

```

З огляду на алгоритм роботи додатку, перед створенням документу користувач повинен обрати шаблон з якого він буде створений, опишемо роботу цього контролера. Першим кроком є перевірка отриманого параметру, а саме аббревіатури підрозділу, виконується перевірка чи існує такий підрозділ, після цього виконується пошук по цьому полю в моделі шаблонів, використовуючи засоби mongoose.

```

export const searchDocTemplates = async (req, res, next) => {
  try {
    const { hubTag } = req.body
    if (!hubTag) {
      return next(errorHandler(400, 'Не вказано параметр'))
    }
    const docTemplates = await DocTemplate.find({
      docTemplateHubTag: hubTag,
    })
  }
}

```

```

if (docTemplates.length === 0) {
  return next(errorHandler(404, 'Шаблони документів не знайдено'))
}
res.status(200).json(docTemplates)
} catch (error) {
  next(error)
}
}

```

Розглянемо сам процес створення документу, перш за все виконується перевірка чи необхідні поля присутні в запиті, після цього виконується запит до моделі підрозділу для перевірки чи існує підрозділ з такою аббревіатурою. Наступним кроком є перевірка чи існує заданий шаблон. Далі виконуємо запит для пошуку шаблону та формуємо назву для наступного документу. Виконуємо пошук в базі користувача, який сформував запит та додаємо його як автора документу та надаємо доступ модератора. Виконуємо зберігання документу.

```

export const createDoc = async (req, res, next) => {
  const { docHubTag, docTemplate } = req.body
  if (!docHubTag || !docTemplate) {
    next(errorHandler(500, 'Введіть правильні дані'))
    return
  }
  let allUsers = await Hub.find({ hubTag: docHubTag }).select(
    'hubAdmins hubModerators',
  )
  if (!allUsers) {
    return next(errorHandler(500, 'Такого хабу не існує'))
  }
  const template = await DocTemplate.find({ docTemplateTitle: docTemplate,
    docTemplateHubTag: docHubTag,
  })
  if (template.length === 0) return next(errorHandler(500, 'Шаблону не існує'))
  const docs = await Doc.find({ docTemplate: docTemplate, docHubTag: docHubTag, })
  const titleNum = `${docTemplate}-${docs.length + 1}`
  const author = await User.findById(req.user.id).select('-userPassword')
  const foundAdmin = allUsers[0].hubAdmins.find(
    (admin) => admin._id.toString() === req.user.id.toString(),
  )
  const foundModerator = allUsers[0].hubModerators.find(
    (moderator) => moderator._id.toString() === req.user.id.toString(),
  )
  if (!foundAdmin && !foundModerator) allUsers[0].hubModerators.push(author)
  const newDoc = new Doc({
    docTitle: titleNum,
    docHubTag,
    docAuthor: [author],
  })

```

```

docTemplate: docTemplate,
docContent: template[0].docTemplateContent,
docApprovedBy: [],
docAdmins: allUsers[0].hubAdmins,
docModerators: allUsers[0].hubModerators,
docIsArchived: false,
})
try {
  await newDoc.save()
  res.status(201).json(newDoc._id)
} catch (error) {
  next(errorHandler(500, error.message))
}
}

```

Одним з важливих етапів є оновлення інформації однієї з моделей, розглянемо на прикладі документу.

```

export const updateDoc = async (req, res, next) => {
  let allUsers = await Doc.findById(req.params.id).select(
    'docAdmins docModerators docUsers',
  )
  let canAdd = false
  const foundAdmin = allUsers.docAdmins.find(
    (admin) => admin._id.toString() === req.user.id.toString(),
  )
  if (foundAdmin) canAdd = true
  const foundModerator = allUsers.docModerators.find(
    (moderator) => moderator._id.toString() === req.user.id.toString(),
  )
  if (foundModerator) canAdd = true
  if (!canAdd) return next(errorHandler(401, 'Заборонено, ви не можете це зробити'))
  try {
    const updatedDoc = await Doc.findByIdAndUpdate(req.params.id,
      {
        $set: {
          docTitle: req.body.docTitle,
          docHubTag: req.body.docHubTag,
          docAuthor: req.body.docAuthor,
          docContent: req.body.docContent,
          docUsers: req.body.docUsers,
          docModerators: req.body.docModerators,
          docAdmins: req.body.docAdmins,
          docIsArchived: req.body.docIsArchived,
        },
      }, { new: true },
    )
    res.status(200).json(updatedDoc._doc)
  } catch (error) {
    next(error)
  }
}

```

Перш за все перевіряється право користувача оновлювати документ, після чого виконується запит до бази даних з переліком полів, які будуть оновлені. В результаті успішно виконаної операції у відповідь отримуємо документ вже зі зміненими полям, за це відповідає вказаний параметр `new: true`.

Після реалізації серверної частини додатку, реалізується клієнтська частина додатку для цього використовуються засоби ReactJS - це бібліотека JavaScript для створення інтерфейсів користувача, особливо для односторінкових додатків. Вона підтримується Facebook та спільнотою індивідуальних розробників та компаній. ReactJS дозволяє розробникам створювати повторно використовувані компоненти користувацького інтерфейсу, які можна складати разом для побудови складних інтерфейсів. [15]

Приділимо увагу сторінці профілю користувача, її реалізацію.

```
const dispatch = useDispatch()
const currentUser = useSelector((state) => state.user.currentUser)
const loading = useSelector((state) => state.user.loading)
const error = useSelector((state) => state.user.error)

const [formData, setFormData] = useState({
  userName: currentUser.userName,
  userSurname: currentUser.userSurname,
  userNewPassword: "",
  userNewPasswordConfirm: "",
})
const [updateDoneNotification, setUpdateDoneNotification] = useState(false)
const handleChange = (e) => {
  const { name, value } = e.target
  setFormData({
    ...formData,
    [name]: value,
  })
}
```

В цьому фрагменті коду наведено реалізацію отримання інформації про поточного користувача використовуючи `react-redux` - це бібліотека для керування станом додатка в React-додатках. Вона забезпечує інтеграцію між бібліотеками React та Redux, що дозволяє зберігати стан додатка в одному місці - у дереві стану Redux, і забезпечує легкий доступ до цього стану у всіх компонентах React. [16] Також можна побачити реалізацію функції

handleChange, як відповідає за збереження інформації з форми, яка знаходиться на сторінці, відображення форми показано в лістингу нижче.

Реалізація відображення інформації про поточного користувача на сторінці профілю, для демонстрації показано виведення двох полів, які показують ім'я та прізвище користувача.

```
<div className="mb-4">
  <label className="block text-gray-700 text-sm font-bold mb-2">First Name</label>
  <input
    className="block w-full px-4 py-2 border rounded-lg focus:outline-none focus:border-
blue-500"
    type="text"
    value={formData.userName}
    id="userName"
    name="userName"
    onChange={handleChange}
  />
</div>
<div className="mb-4">
  <label className="block text-gray-700 text-sm font-bold mb-2">Last Name</label>
  <input
    className="block w-full px-4 py-2 border rounded-lg focus:outline-none focus:border-
blue-500"
    type="text"
    value={formData.userSurname}
    id="userSurname"
    name="userSurname"
    onChange={handleChange}
  />
</div>
```

Приділимо увагу функції виходу користувача з системи “signOut” . При цьому буде виконуватись GET запит [17], та обробка storage в redux, з видаленням токена JWT з локального сховища користувача.

```
export const signOut = async (req, res, next) => {
  try {
    res.clearCookie('access_token')
    res.status(200).json('Вихід з системи виконано')
  } catch (error) {
    next(errorHandler(500, 'Помилка виходу з системи'))
  }
}
```

Тепер переглянемо як цей запит буде виглядати на клієнтській частині додатку.

```
const handleSignOut = async () => {
  try {
    dispatch(signOutStart())
    const res = await fetch(`/api/auth/signout`, {
      method: 'GET',
      headers: {
        'Content-Type': 'application/json',
      },
    })
    const data = await res.json()
    if (data.success === false) {
      dispatch(signOutFailure(data.message))
      return
    }
    dispatch(signOutDone(data))
  } catch (error) {
    dispatch(signOutFailure(error.message))
  }
}
```

Висновки до другого розділу

В ході виконання другого розділу сформовано перелік варіантів використання веб-додатку, сформовано остаточний перелік вимог до системи. На основі вищевказаного сформовано таблиці з описом прецедентів використання, описано взаємодію між компонентами системи.

Спроектовано базу даних, чітко описано перелік колекцій бази даних та форму кожної з них, які дані там будуть зберігатись. Перед реалізацією коду додатку створено діаграму активностей, яка визначає шлях, який буде проходити користувач при використанні сервісу.

Була показана реалізація основних функцій веб-додатку, показано фрагменти коду, які показують всі використані технології, описано взаємодію між серверною і клієнтською частиною веб-додатку.

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З СИСТЕМОЮ ЕЛЕКТРОННОГО ДОКУМЕНТООБІГУ

3.1. Порядок встановлення та налаштування параметрів платформи

Встановлення додатку полягає в розгортанні системи на серверному обладнанні, налаштування домену за потреби або інтеграція в вже існуючу систему. Для відображення архітектури додатку було створено діаграму розгортання (рис.3.1).

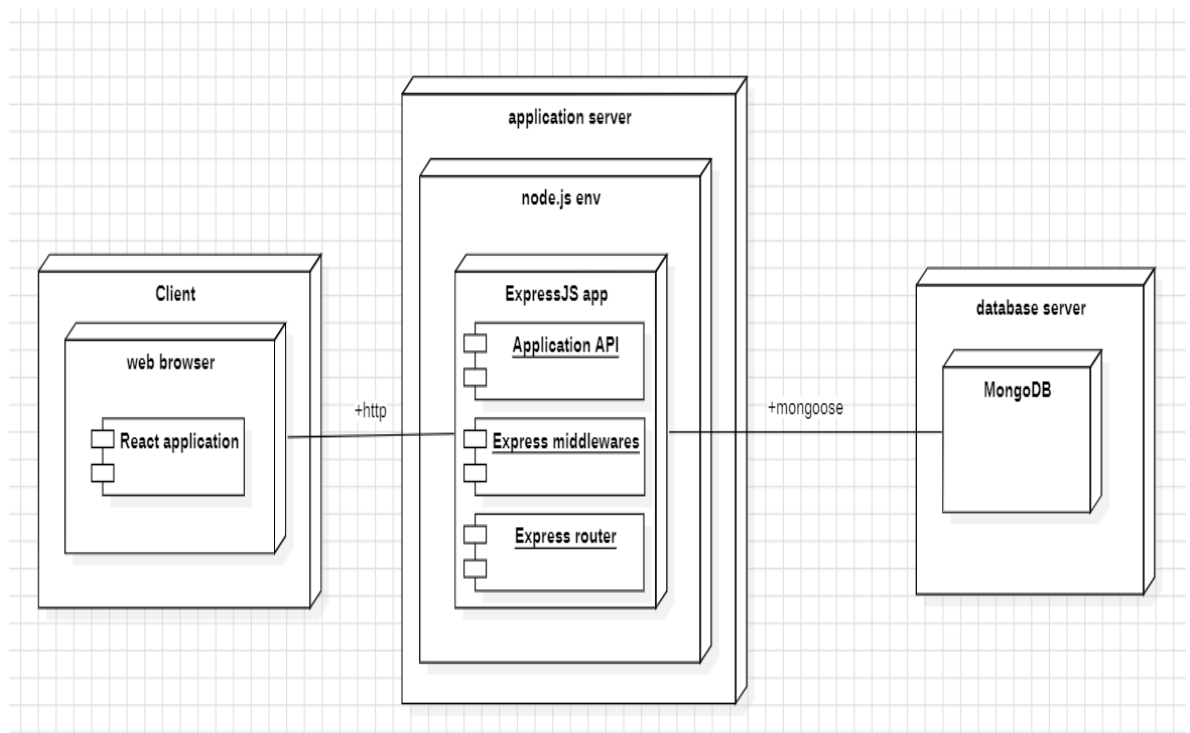


Рис.3.1 - Діаграма розгортання веб-додатку

Для розгортання додатку буде використано два сервери, один для роботи самого додатку, інший для бази даних, можливо використання хмарного рішення для бази даних. Для того щоб користувач отримав доступ до системи він повинен встановити на свій пристрій браузер та мати доступ до мережі інтернет. Якщо більш детально розглянути компонент серверу веб-додатку, на ньому можна побачити такі елементи – Application API, що являє собою всю логіку роботи додатку, Express middlewares, Express router, що відповідають за маршрутизацію та залежності розгорнутого серверу засобами ExpressJS.

Для більш детального опису архітектури додатку створено діаграму компонентів для ApplicationAPI (рис.3.2).

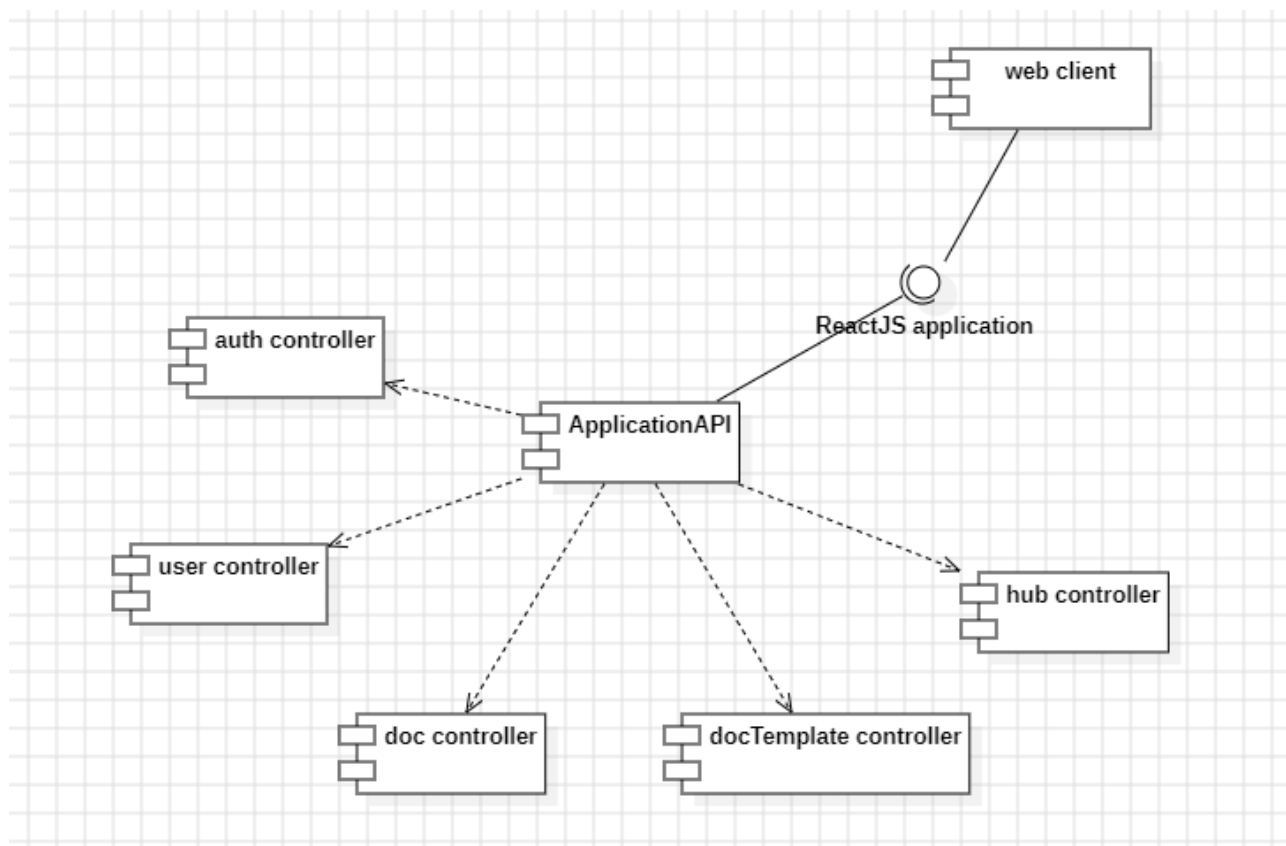


Рис.3.2 діаграма компонентів веб-додатку.

На діаграмі відображено компоненти з яких складається додаток, показані контролери, які відповідають за обробку інформації:

1. auth controller – відповідає за авторизація користувача.
2. user controller – відповідає за обробку користувачів.
3. doc controller – відповідає за роботу з документами.
4. docTemplate controller – відповідає за роботу з шаблонами.
5. hub controller – відповідає за роботу з підрозділами.

Всі ці компоненти забезпечують роботу додатку його взаємодію з сервером бази даних, передачу даних на клієнтську сторону. Засобами ReactJS реалізовано веб-сторінку, де користувач зможе взаємодіяти з даними, які були отримані в ході роботи серверної частини.

3.2. Структура інтерфейсу системи електронного документообігу

Першою сторінкою з якою користувачу необхідно буде взаємодіяти – це сторінка реєстрації, авторизації (рис.3.3 – 3.4).

Головна Реєстрація Авторизація

Sign Up

Name

Surname

Email

Password

Sign Up Sign In

Рис.3.3 - Сторінка реєстрації користувача

Sign In

Email

Password

Sign In Sign Up

Рис.3.4 – Вікно авторизації користувача

Після того як користувач авторизувався в свій акаунт він потрапляє на сторінку з доступними для нього документами (рис.3.5), також він може переглянути перелік доступних підрозділів, архів та свій профіль (рис.3.6), поки йому не надали доступ до документів або підрозділу, йому буде виводитись повідомлення про відсутність доступних елементів.

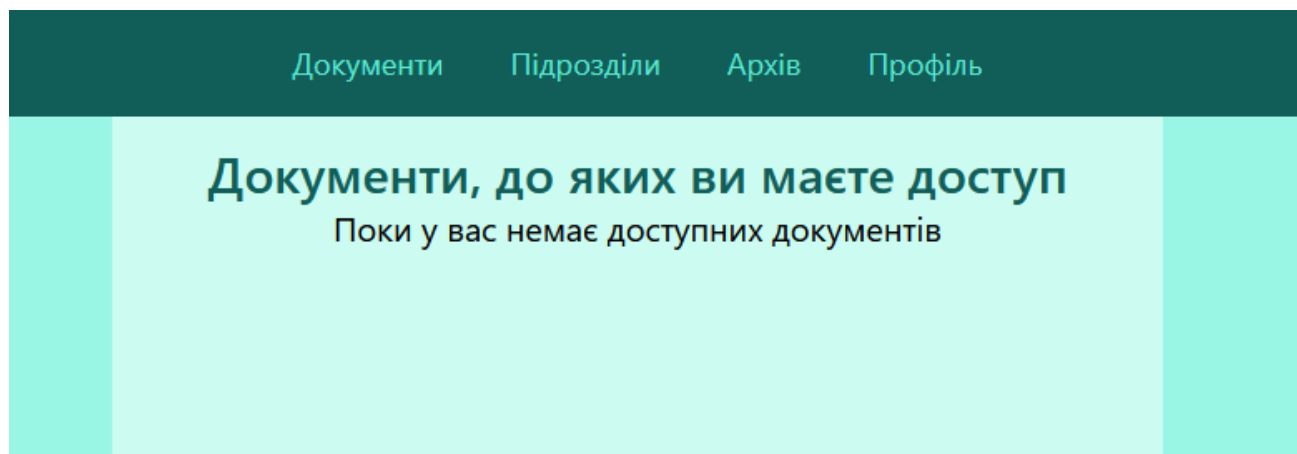


Рис.3.5 – Сторінка доступних документів

Рис.3.6 – Профіль користувача

На сторінці профілю користувач може вносити корективи про себе, а також змінювати пароль до свого акаунту при цьому для кожної дії присутня валідація полів, щоб користувач вводив тільки коректні дані (рис.3.7).

Документи Підрозділи Архів Профіль

Your profile

First Name

Іван

Last Name

Петров

Email

user@gmail.com

Password

•••••

Confirm Password

•••••

Update **Sign Out**

Пароль не може бути менше 5 символів

Рис.3.7 – Спроба користувача змінити свій пароль

Якщо користувач авторизується як адміністратор у нього будуть доступні нові функції, а саме створення підрозділу, а також сторінка керування, де можна переглянути перелік всіх користувачів додатку, а також всі створені підрозділи (рис.3.8).

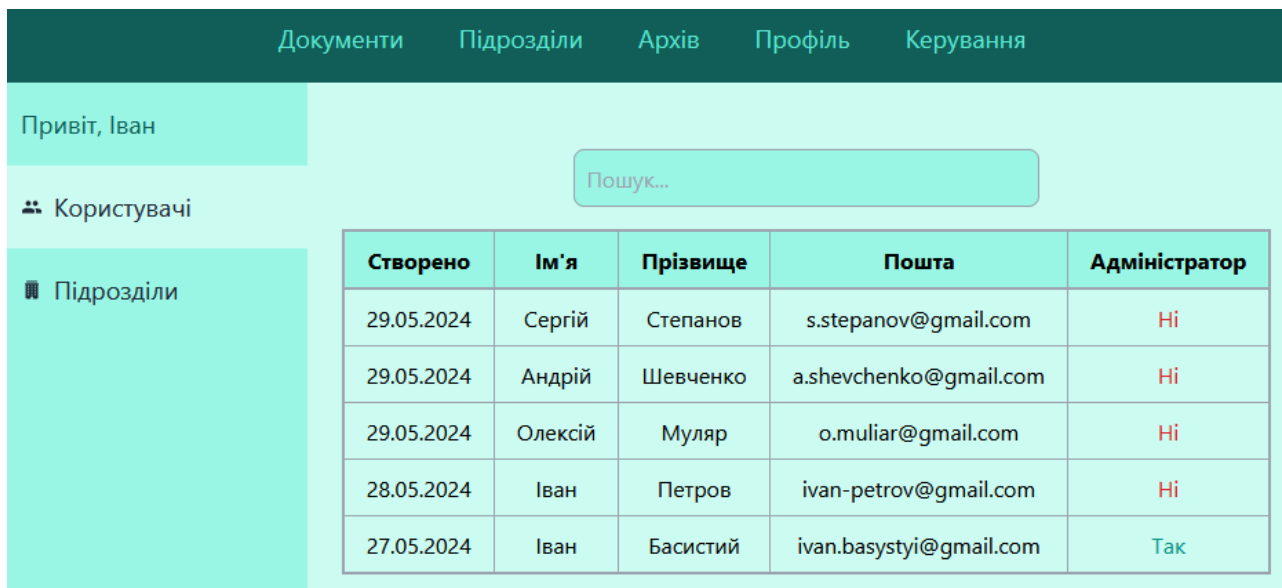


Рис.3.8 – Сторінка керування веб-додатком

Розглянемо функціонал створення підрозділу, якщо користувач має права адміністратора (рис.3.9).

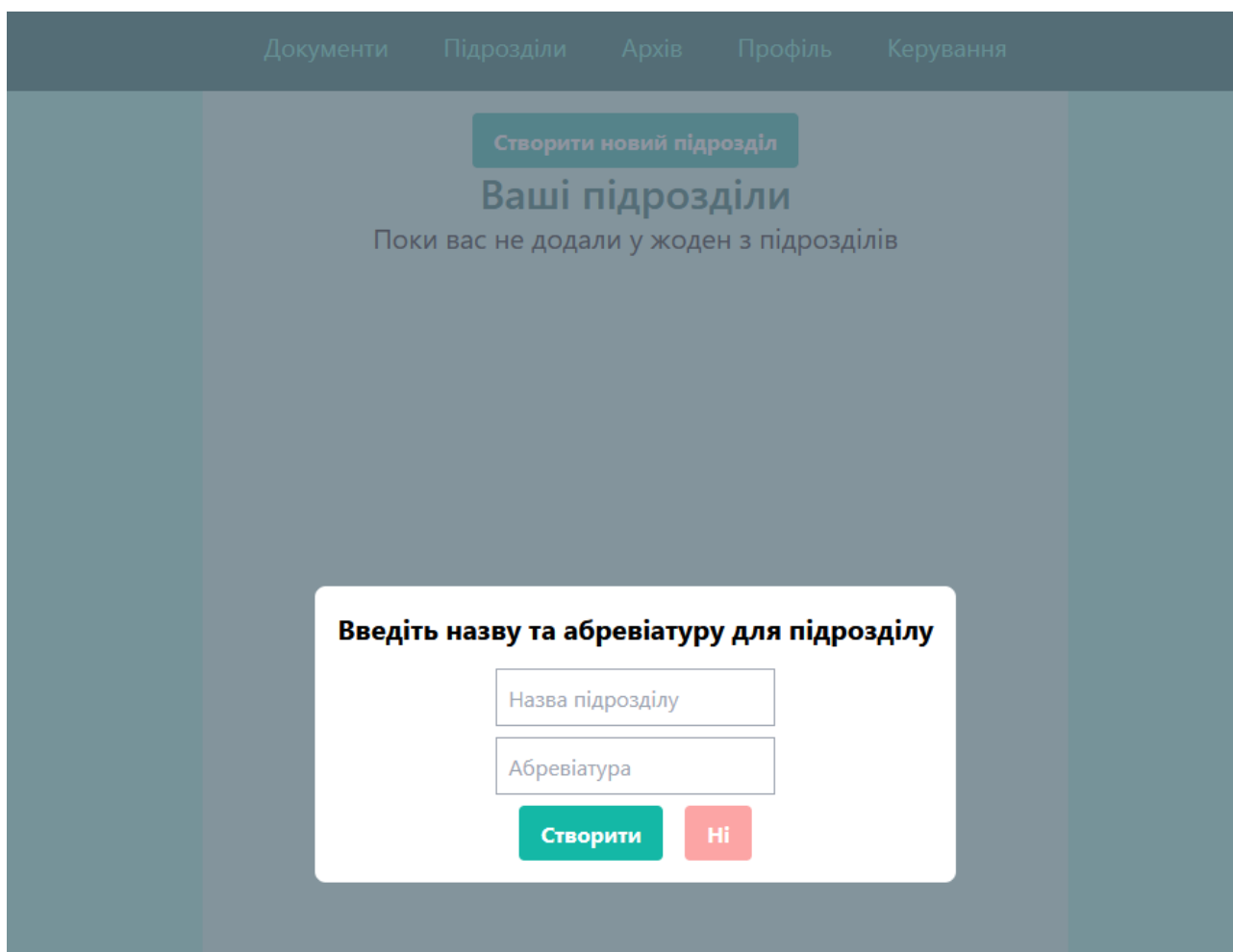


Рис.3.9 – Форма створення нового підрозділу

Коли користувач потрапляє на сторінку підрозділу, якщо він має доступ, він може редагувати інформацію про підрозділ (рис.3.10) , додавати модераторів та користувачів вказавши їх пошту (рис.3.11).

Документи Підрозділи Архів Профіль Керування

Інформація

Назва:
Відділ інформаційних технологій

Абревіатура:
ІТ

Опис:
Підрозділ інформаційних технологій (ІТ) в організації відповідає за управління всіма технологічними ресурсами, необхідними для підтримки та покращення операцій організації. Це включає як апаратні, так і програмні компоненти, мережеву інфраструктуру, дані та всі інші ресурси, пов'язані з ІТ.

Зберегти зміни

Рис.3.10 – Внесення змін до інформації про підрозділ

Хто має доступ

Модератори

Користувачі

Додати модератора до підрозділу

Пошта користувача

Так **Ні**

Рис.3.11 – Додавання нового модератора до підрозділу

Також додамо шаблон нового документу (рис.3.12), для подальшої роботи з ним, та можливості створення документів для користувачів підрозділу.

Введіть назву та вміст шаблону

Переміщення

H1 H2

Sans Serif

Large

B

I

U

”

Склад відправник: _____

Склад отримувач: _____

Накладна на внутрішнє переміщення, від "___" _____ 2024р.

1. _____

2. _____

3. _____

Видав: _____

Одержав: _____

Створити

Ні

Рис.3.12 – Форма створення нового шаблону для документу

Після того як були створені шаблони, користувачі можуть створити документ, використовуючи один з них (рис. 3.13).

Інформація

Назва: Відділ інформаційних технологій

Абревіатура: ІТ

Опис: Підрозділ інформаційних технологій (ІТ) в організації відповідає за управління всіма технологічними ресурсами, необхідними для підтримки та покращення операцій організації. Це включає як апаратні, так і програмні компоненти, мережеву інфраструктуру, дані та всі інші ресурси, пов'язані з ІТ.

Редагувати

Хто має доступ

Модератори

Іван Петров +

Користувачі

Олексій Муляр

Андрій Шевченко

Сергій Степанов +

Оберіть один з шаблонів для вашого підрозділу

Переміщення

Заява

Доповідна записка

Створити

Ні

Рис.3.13 – Створення нового документу

					ІПЗ.КР.Б – 121 – 24 – ПЗ	43
--	--	--	--	--	--------------------------	----

Після створення документу користувач потрапляє на його сторінку (рис.3.14), де він може виконувати його обробку, вносити корективи в текст документу, або погодити його у разі необхідності (рис.3.15), при цьому блокується внесення змін в його текст при наявності хоча б 1 погодження.

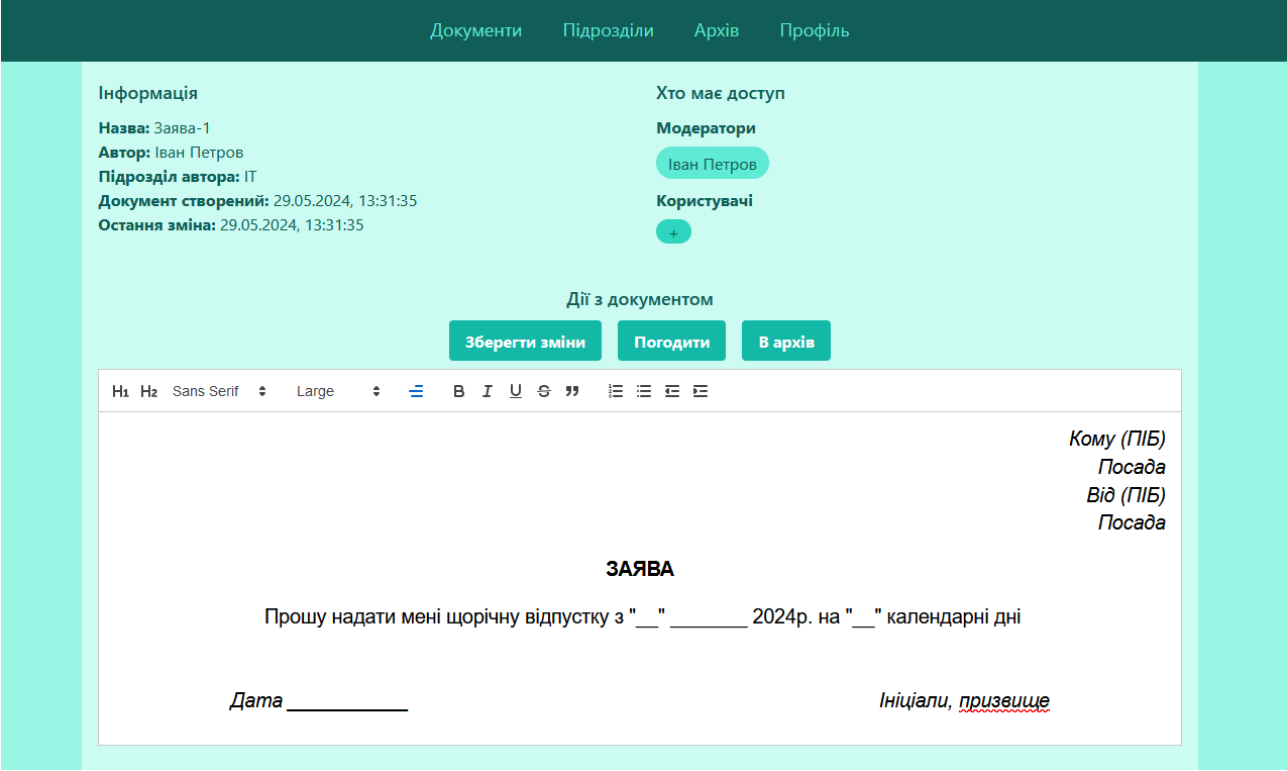


Рис.3.14 – Сторінка документу

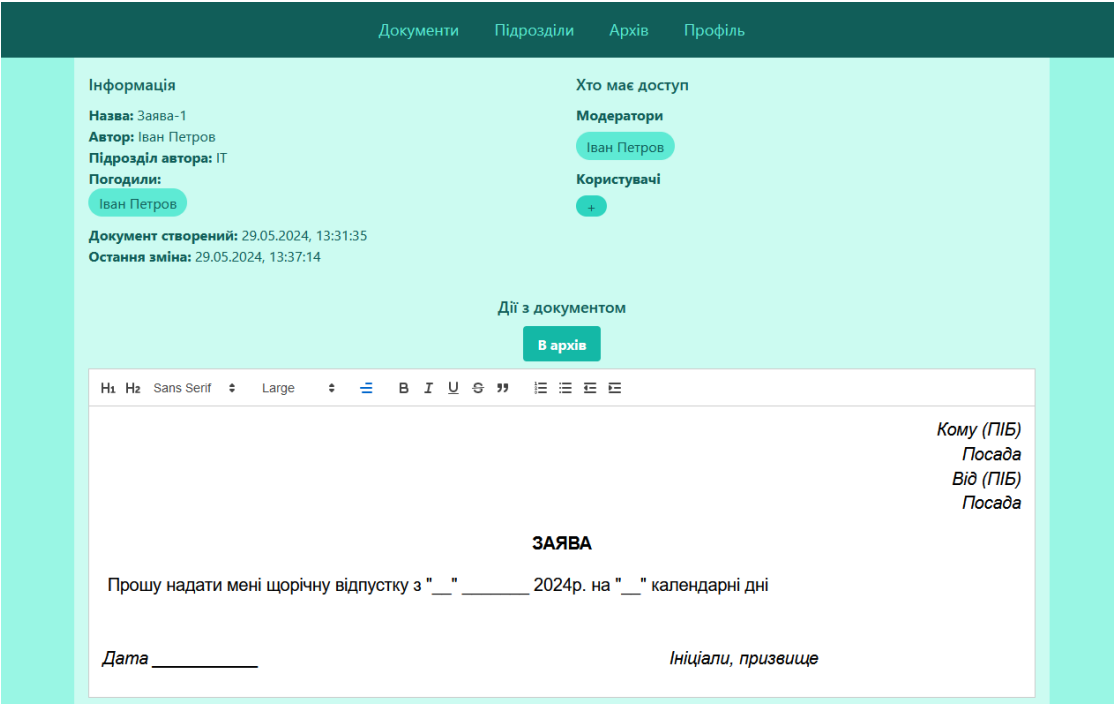


Рис.3.14 – Вигляд документу після погодження

3.3. Тестування веб-додатку електронного документообігу.

Тестування додатку може відбуватись, як шляхом ручного тестування, так і з залученням онлайн сервісів[18]. Для того щоб провести тестування додатку, розгорнемо його в середовищі Vercel - це платформа для розгортання та хостингу додатків. Вона надає розробникам інструменти та сервіси для безперебійного управління процесом розробки, тестування та розгортання веб-проектів. [19]

Спочатку проведемо ручне тестування для перевірки.

Перші етап, який підлягає тестуванню, це авторизація. При спробі ввести некоректні дані користувач отримує повідомлення про неправильність даних (рис.3.15), при спробі перейти на сторінку за адресою, до якої у нього немає доступу, користувача повертає на сторінку авторизації.

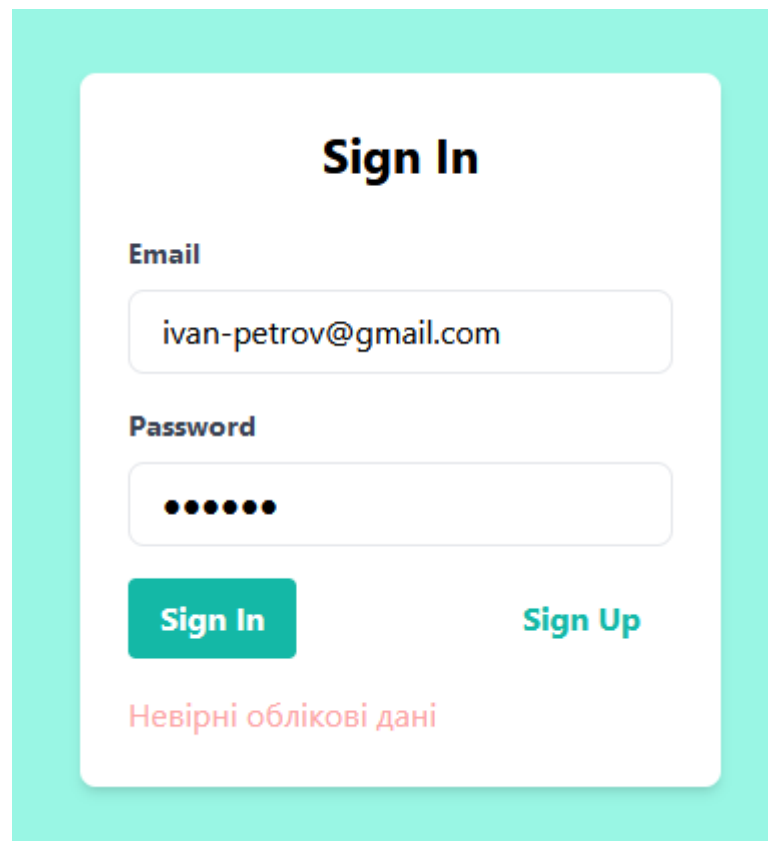
A screenshot of a web application's 'Sign In' form. The form is white with rounded corners and is centered on a light blue background. It features two input fields: 'Email' with the value 'ivan-petrov@gmail.com' and 'Password' with masked characters. Below the fields are two buttons: a blue 'Sign In' button and a teal 'Sign Up' button. At the bottom of the form, a red error message reads 'Невірні облікові дані' (Incorrect login details).

Рис.3.15 – Помилка при спробі авторизації з хибними обліковими даними

Наступний етап, перевірка чи може користувач перейти на сторінку, до якої він не повинен мати доступ, наприклад сторінка керування, куди доступ повинні мати тільки адміністратори. При переході за цією адресою відображається порожня сторінка з повідомленням “Немає доступу” (рис.3.16).

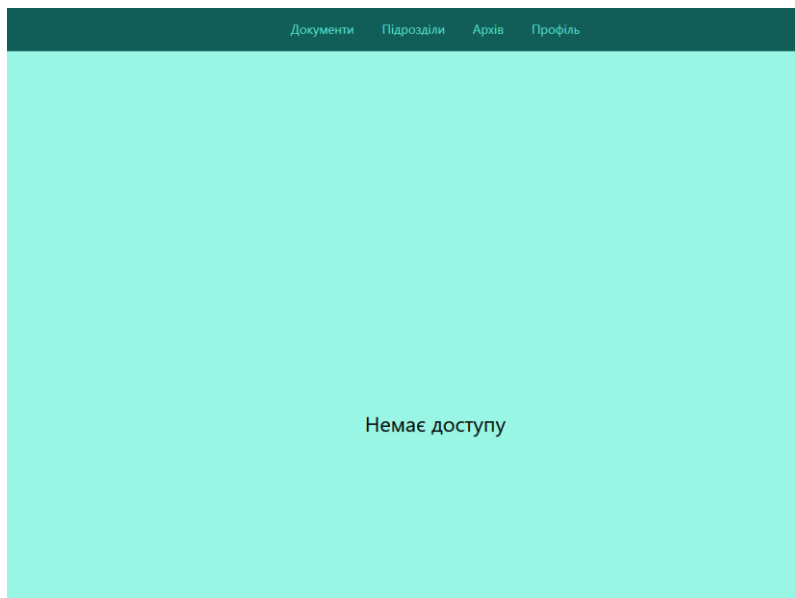


Рис.3.16 – Помилка при переході на сторінку, куди користувач не має доступу

Функціонал додатку передбачає, що користувач може змінити свій пароль, при спробі ввести значення, яке не відповідає стандарту – виникає помилка (рис.3.17).

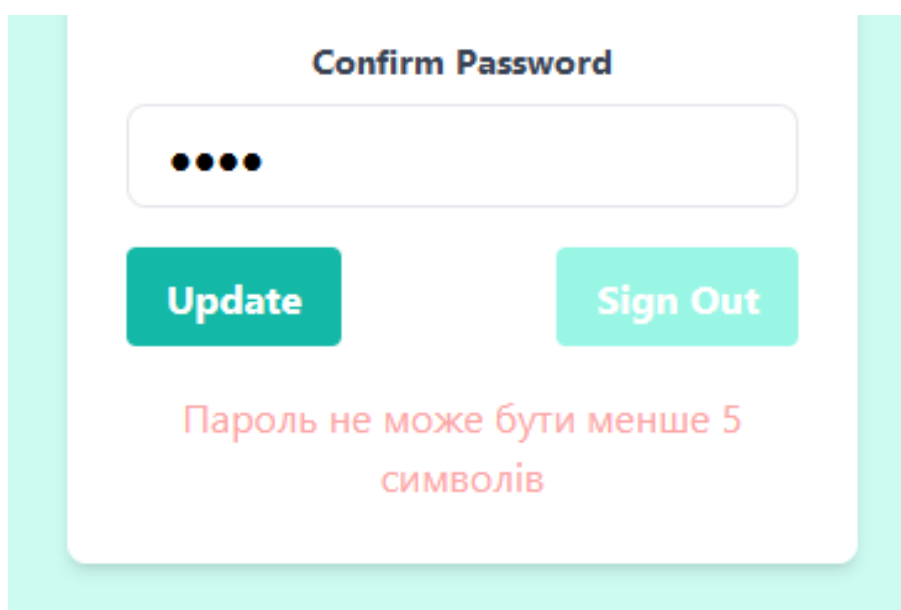


Рис.3.17 – Помилка при спробі оновити пароль

Також протестовано функціонал роботи з документами, якщо користувач який не має права на додавання модератора спробує це зробити – виникне помилка (рис.3.18).

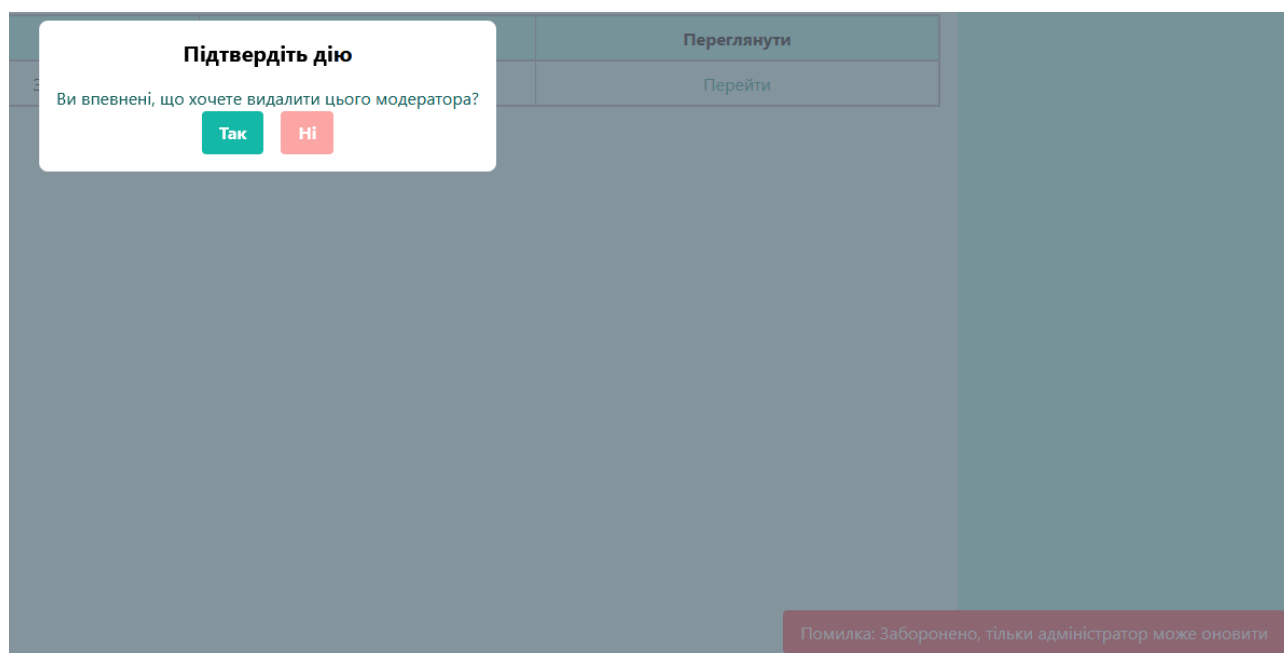


Рис.3.18 – Помилка при спробі видалити модератора

Також при спробі залишити порожнім поле в одній з форм, виникає помилка, що користувач повинен ввести всі дані (рис.3.19).

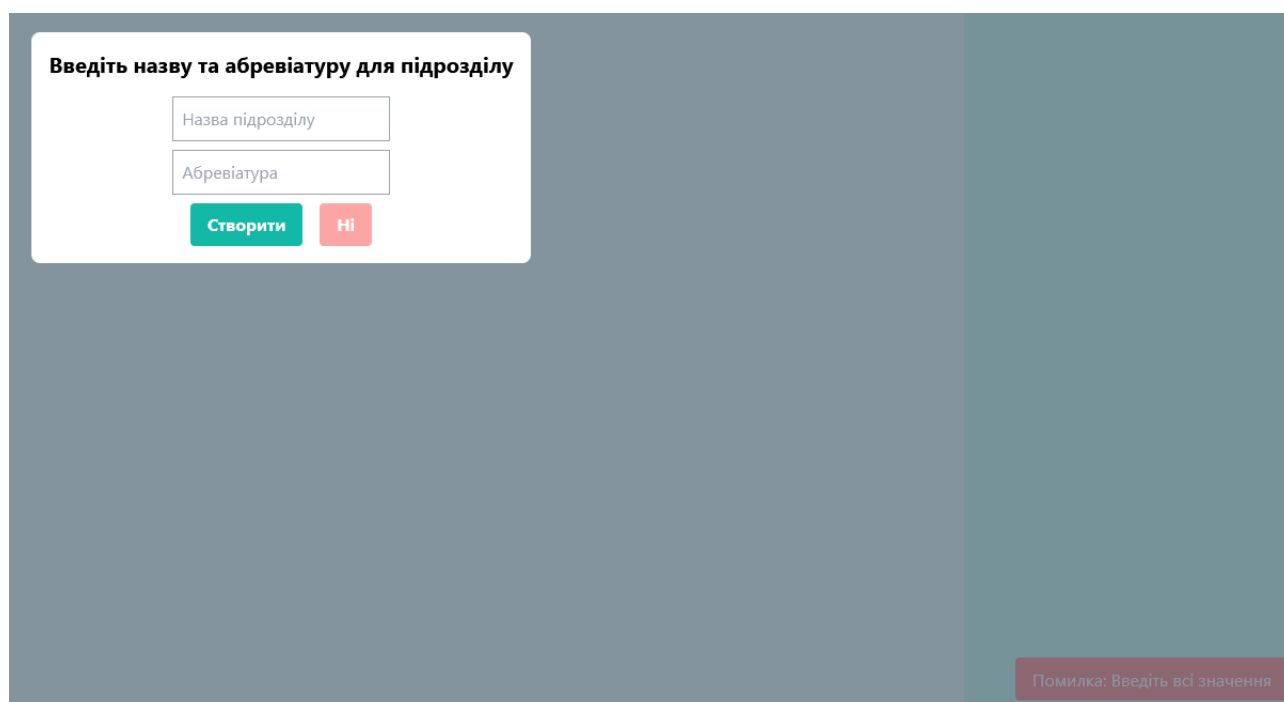


Рис.3.19 – Помилка при спробі залишити поля порожніми

Для подальшого тестування використаємо онлайн сервіси PageSpeed Insights [20], цей ресурс дозволить перевірити швидкість завантаження веб-додатку, а також покаже, де наявні проблеми з швидкодією, також можна перевірити крос-платформеність, результати тестування (рис.3.20 – рис.3.22)

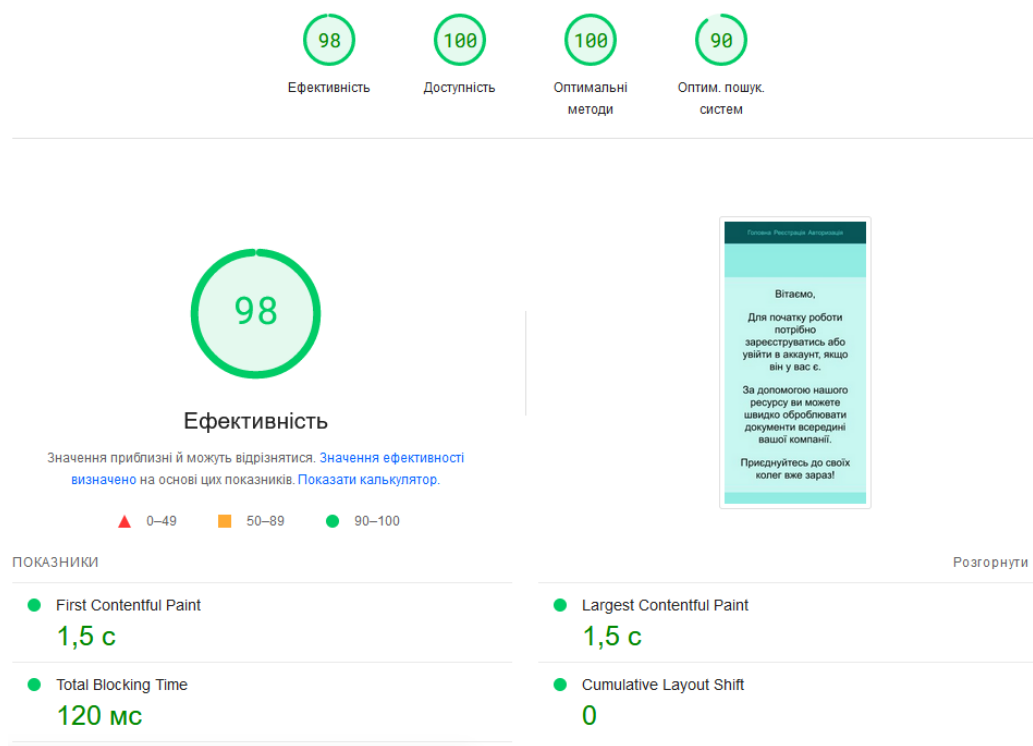


Рис.3.20 – Результати тестування веб-додатку засобами PageSpeed

ДІАГНОСТИКА	
▲	Зменште код JavaScript, який не використовуєте — Можна зменшити на 80 КіБ
■	Не показуйте застарілі елементи JavaScript у сучасних веб-переглядачах — Можна зменшити на 9 КіБ
○	Сервер відповідає швидко — Кореневий документ відповів через 20 мс
○	Уникається великий обсяг даних мережі — Загальний розмір — 145 КіБ
○	Уникається надмірний розмір DOM — 17 елементів
○	Намагайтеся не створювати ланцюжки критичних запитів — Знайдено 2 ланцюжки
○	Час виконання JavaScript — 0,1 с
○	Мінімізується робота основного потоку — 0,1 с
○	Елемент візуалізації великого контенту — 370 мс

Рис.3.21 – Результати тестування веб-додатку засобами PageSpeed

Перелік тестів, які були виконані, засобами сервісу (рис.3.22).

ВИКОНАНІ ПЕРЕВІРКИ (14)

Сховати

● Використовує протокол HTTPS	▼
● Уникає інтерфейсів API, які більше не підтримуються	▼
● Уникає сторонніх файлів cookie	▼
● Дає користувачам змогу вставляти контент із буфера в поля введення	▼
● Уникає надсилання запитів на доступ до геолокації під час завантаження сторінки	▼
● Уникає надсилання запитів на показ сповіщень під час завантаження сторінки	▼
● Показує зображення правильного формату	▼
● Показує зображення з правильною роздільною здатністю	▼
● Має тег <code><meta name="viewport"></code> з атрибутами <code>width</code> або <code>initial-scale</code>	▼
● Сторінка має елемент HTML doctype	▼
● Належно визначає набір символів	▼
● Помилки веб-переглядача не записано в журнал консолі	▼
● На панелі "Issues" в Інструментах розробника Chrome немає помилок	▼
● Сторінка містить дійсні карти джерел	▼

Рис.3.22 – Результати тестування веб-додатку засобами PageSpeed

Ще один сервіс використаний для тестування клієнтської частини додатку GTMetrix - це популярний інструмент для аналізу продуктивності веб-сайтів. Він надає відомості про різні аспекти швидкості та продуктивності веб-сайту, такі як час завантаження сторінки, розмір сторінки та кількість запитів. GTmetrix генерує докладний звіт, в якому наведені рекомендації щодо оптимізації продуктивності веб-сайту, такі як мінімізація CSS та JavaScript файлів, оптимізація зображень, використання кешування браузера та інше. [21], результати тестування даним сервісом (рис.3.23 – 3.25).

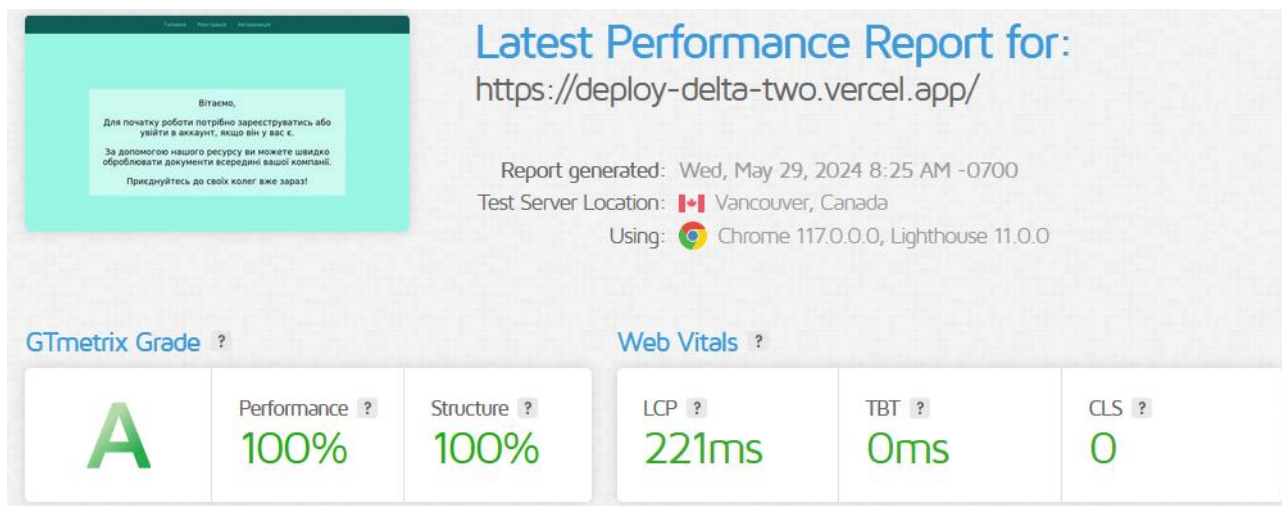


Рис.3.23 – Результати тестування веб-додатку засобами GTMetrix

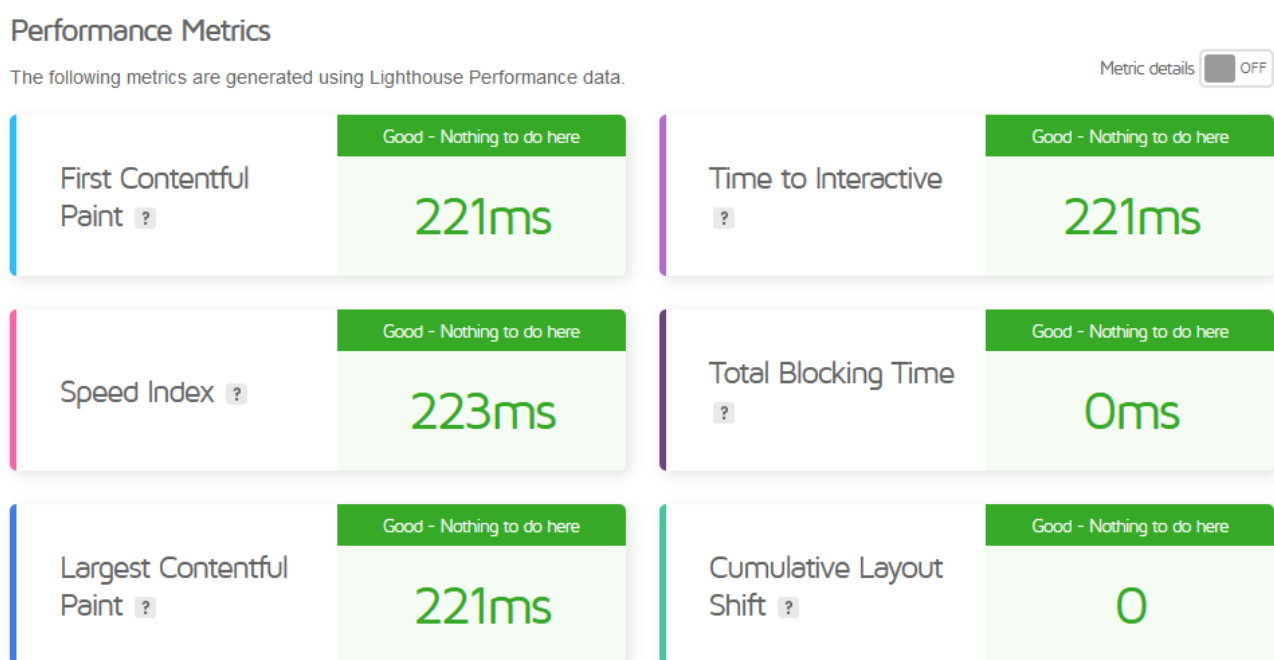


Рис.3.24 – Результати тестування швидкодії веб-додатку засобами GTMetrix

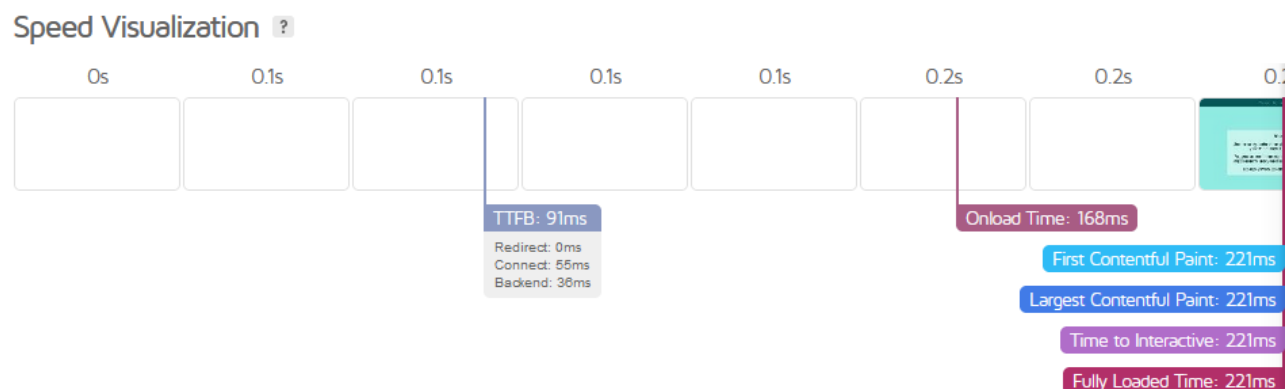


Рис.3.25 – Результати тестування швидкодії веб-додатку засобами GTMetrix

Висновки до третього розділу

У третьому розділі розглянуто інтерфейс та структуру додатку. Спочатку подано опис архітектури розгортання, а також висвітлено компоненти, що входять до складу додатку. В цьому контексті надано діаграми, які ілюструють архітектурну концепцію додатку, зокрема, взаємозв'язки між його ключовими складовими.

Наступна частина розділу показує детальний аналіз інтерфейсу додатку, зокрема надано докладний опис функціоналу, доступного кожному користувачеві. Підкреслено основні можливості, які надаються на різних етапах взаємодії з програмою. Додатково приведено конкретні приклади та сценарії використання, щоб чітко уявити, як кожна функція може бути застосована в реальних ситуаціях.

В завершальній частині проведено тестування додатку, для забезпечення безперебійної роботи, а також виявити проблеми, які можуть виникнути у користувача в ході виконання завдань. Проведено тестування за допомогою сторонніх сервісів для перевірки швидкодії ресурсу.

ВИСНОВКИ

В ході виконання кваліфікаційної роботи досліджено тему впровадження систем електронного документообігу. У результаті були визначені основні вимоги до таких систем, враховуючи потреби та специфіку організаційного середовища, наведено перелік ключових вимог до системи.

Проведено порівняльний аналіз різних систем електронного документообігу, цей аналіз включає порівняння функціональності, ефективності, вартості впровадження та підтримки, а також дослідження діяльності користувачів даних систем.

На основі сформованого переліку вимог користувачів до системи електронного документообігу було обрано оптимальну архітектуру та формат розробки. Враховуючи ці вимоги, був сформований перелік варіантів використання системи, який охоплює різноманітні можливості та функції, які можуть бути доступні для користувачів. Для ілюстрації цього переліку та розуміння послідовності дій була розроблена діаграма активностей.

У другому розділі одним з важливих етапів був вибір бази даних для системи електронного документообігу та реалізація основних функцій. Було обрано базу даних, яка належним чином відповідає вимогам розроблюваної системи.

Для прикладу було наведено програмну реалізацію ключових компонентів системи включає в себе розробку алгоритмів обробки документів, та взаємодії користувачів між собою.

Третій розділ був присвячений налаштуванню системи інтерфейсу та тестуванню системи. Початково були наведені діаграми розгортання та компонентів, які допомагають візуалізувати масштаб системи.

Також був розроблений зручний інтерфейс для користувача, який враховує потреби та вимоги користувачів.

Цей етап дозволив перевірити готовність системи до впровадження та забезпечити якість роботи програмного забезпечення перед його використанням.

Отже, в ході виконання кваліфікаційної роботи був реалізований повноцінний програмний продукт, який готовий до використання користувачами. Цей програмний продукт відповідає всім основним вимогам, що були визначені для системи електронного документообігу на етапі аналізу та проектування.

Реалізований комплекс забезпечує можливість ефективного та безпечного обміну документами, забезпечуючи їх електронне зберігання, обробку та обмін.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Закон України "Про електронні документи та електронний документообіг" Редакція від 31.12.2023 [Електронний ресурс]. – 2023. – Режим доступу до ресурсу: <https://zakon.rada.gov.ua/laws/show/851-15#Text>.
2. EDIN [Електронний ресурс] – Режим доступу до ресурсу: <https://edin.ua/>.
3. Вчасно [Електронний ресурс] – Режим доступу до ресурсу: <https://vchasno.ua/>.
4. COTA [Електронний ресурс] – Режим доступу до ресурсу: <https://sota-buh.com.ua/>.
5. Davis C. Cloud Native Patterns: Designing change-tolerant software / Cornelia Davis., 2019. – 400 с.
6. Subramanian H. Hands-On RESTful API Design Patterns and Best Practices / H. Subramanian, P. Raj., 2019. – 378 с.
7. What Is the MERN Stack? [Електронний ресурс] – Режим доступу до ресурсу: <https://www.oracle.com/cis/database/mern-stack/>.
8. Fowler M. Refactoring: Improving the Design of Existing Code / Martin Fowler., 2018. – 431 с.
9. Sundaramoorthy S. UML Diagramming: A Case Study Approach / Suriya Sundaramoorthy., 2022. – 416 с.
10. Bradshaw S. MongoDB: The Definitive Guide: Powerful and Scalable Data Storage / S. Bradshaw, E. Brazil, K. Chodorow., 2019. – 511 с.
11. Blokdyk G. Open web platform: complete self-assessment guide. / Gerardus Blokdyk., 2018. – 140 с.
12. Casciaro M. Node.js Design Patterns - Third edition: Design and implement production-grade Node.js applications using proven patterns and techniques / M. Casciaro, L. Mammino., 2020. – 664 с.
13. Simpson K. You Don't Know JS Yet: Get Started / Kyle Simpson., 2020. – 143 с.

14. Brown E. Web Development with Node and Express: Leveraging the JavaScript Stack / Ethan Brown., 2020. – 332 с.
15. Roldán S. C. React 17 Design Patterns and Best Practices - Third Edition: Design, build, and deploy production-ready web applications using industry-standard practices / Santana Carlos Roldán., 2021. – 394 с.
16. Accomazzo A. Fullstack React: The Complete Guide to ReactJS and Friends / A. Accomazzo, N. Murray, A. Lerner., 2018. – 836 с.
17. Ludin S. Learning HTTP/2: A Practical Guide for Beginners / S. Ludin, J. Garza., 2017. – 154 с.
18. Forsgren N. Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations / N. Forsgren, J. Humble, G. Kim., 2018. – 288 с.
19. Vercel [Электронный ресурс] – Режим доступа до ресурсу: <https://vercel.com/>.
20. PageSpeed insights [Электронный ресурс] – Режим доступа до ресурсу: <https://pagespeed.web.dev/>.
21. GTmetrix [Электронный ресурс] – Режим доступа до ресурсу: <https://gtmetrix.com/>.

ДОДАТКИ

					ІПЗ.КР.Б – 121 – 24 – ПЗ	
						56

Реалізація контролера “auth”

Лістинг коду

```

import bcryptjs from 'bcryptjs'
import User from '../models/user.model.js'
import { errorHandler } from '../utils/error.js'
import jwt from 'jsonwebtoken'
import dotenv from 'dotenv'

dotenv.config()

export const signUp = async (req, res, next) => {
  const { userName, userSurname, userEmail, userPassword } = req.body
  const hashUserPassword = bcryptjs.hashSync(userPassword, 13)
  const newUser = new User({
    userName,
    userSurname,
    userEmail,
    userPassword: hashUserPassword,
    userIsAdmin: false,
  })

  try {
    await newUser.save()
    res.status(201).json('Користувач створений')
  } catch (error) {
    next(errorHandler(500, 'Помилка створення користувача'))
  }
}

export const signIn = async (req, res, next) => {
  const { userEmail, userPassword } = req.body
  try {
    const validUser = await User.findOne({ userEmail: userEmail })
    if (!validUser) return next(errorHandler(404, 'Користувач не знайдений'))

    const validPassword = bcryptjs.compareSync(
      userPassword,
      validUser.userPassword,
    )
    if (!validPassword) return next(errorHandler(401, 'Невірні облікові дані'))

    const token = jwt.sign(
      {
        id: validUser._id,
        userName: validUser.userName,
        userSurname: validUser.userSurname,
        userEmail: validUser.userEmail,
        userIsAdmin: validUser.userIsAdmin,
      },

```

```

    process.env.JWT_TOKEN,
  )
  const { userPassword: pass, ...rest } = validUser._doc

  res.cookie('access_token', token, { httpOnly: true }).status(200).json(rest)
} catch (error) {
  next(errorHandler(500, 'Помилка входу в систему'))
}
}

export const signOut = async (req, res, next) => {
  try {
    res.clearCookie('access_token')
    res.status(200).json('Вихід з системи виконано')
  } catch (error) {
    next(errorHandler(500, 'Помилка виходу з системи'))
  }
}

```

Реалізація моделі “doc”

Лістинг коду

```
import mongoose from 'mongoose'

const docSchema = new mongoose.Schema(
  {
    docTitle: {
      type: String,
      required: true,
      unique: true,
    },
    docContent: {
      type: String,
      required: false,
      unique: false,
    },
    docAuthor: {
      type: Array,
      required: true,
      unique: false,
    },
    docHubTag: {
      type: String,
      required: true,
      unique: false,
    },
    docTemplate: {
      type: String,
      required: true,
      unique: false,
    },
    docApprovedBy: {
      type: Array,
      required: false,
      unique: false,
    },
    docUsers: {
      type: Array,
      required: false,
      unique: false,
    },
    docModerators: {
      type: Array,
      required: false,
      unique: false,
    },
  },
)
```

```

docAdmins: {
  type: Array,
  required: false,
  unique: false,
},
docIsArchived: {
  type: Boolean,
  required: false,
  unique: false,
},
},
{ timestamps: true },
)

const Doc = mongoose.model('Doc', docSchema)

export default Doc

```

Реалізація компонента “UserDocuments”

Лістинг коду

```

import React, { useEffect, useState } from 'react'
import { useSelector } from 'react-redux'
import { Link } from 'react-router-dom'
import Notification from '../components/Notification'

const UserDocsList = ({ hubTag }) => {
  const currentUser = useSelector((state) => state.user.currentUser)
  const [docsList, setDocsList] = useState([])
  const [searchTerm, setSearchTerm] = useState("")
  const [showNotification, setShowNotification] = useState(false)
  const [notificationMessage, setNotificationMessage] = useState("")
  const [isError, setIsError] = useState(false)

  useEffect(() => {
    const fetchDocuments = async () => {
      try {
        const res = await fetch(`/api/doc/my`, {
          method: 'POST',
          headers: {
            'Content-Type': 'application/json',
          },
        })
        if (res.ok) {
          const data = await res.json()
          setDocsList(data.docs)
        } else {
          const errorResponse = await res.json()
          setNotificationMessage(`Помилка: ${errorResponse.message}`)
          setIsError(true)
          setShowNotification(true)
          setTimeout(() => {
            setShowNotification(false)
          }, 3000)
        }
      } catch (error) {
        setNotificationMessage(`Помилка: ${error.message}`)
        setIsError(true)
        setShowNotification(true)
        setTimeout(() => {
          setShowNotification(false)
        }, 1500)
      }
    }

    fetchDocuments()
  }, [hubTag])

```

```

const filteredDocs =
  docsList &&
  docsList.filter((doc) => {
    return (
      doc.docTitle.toLowerCase().includes(searchTerm.toLowerCase()) ||
      doc.docHubTag.toLowerCase().includes(searchTerm.toLowerCase())
    )
  })

return (
  <div className="mt-1/2 text-center mx-auto">
    {showNotification && <Notification message={notificationMessage} isError={isError} />}
    {docsList.length > 0 ? (
      <>
        <div className="overflow-x-auto">
          <input
            type="text"
            placeholder="Пошук..."
            value={searchTerm}
            onChange={(e) => setSearchTerm(e.target.value)}
            className="w-1/2 p-2 border bg-teal-200 border-gray-400 rounded-lg mt-4 mb-4"
          />
          <table className="w-full bg-teal-100 shadow-md rounded-lg border-2 border-gray-400">
            <thead className="border-b-2 border-gray-400">
              <tr className="bg-teal-200">
                <th className="py-2 px-4 border-r border-gray-400">Створено</th>
                <th className="py-2 px-4 border-r border-gray-400">Назва</th>
                <th className="py-2 px-4 border-r border-gray-400">Підрозділ</th>
                <th className="py-2 px-4 border-r border-gray-400">Переглянути</th>
              </tr>
            </thead>
            <tbody>
              {filteredDocs.map((doc) => (
                <tr key={doc._id} className="border-b border-gray-400">
                  <td className="py-2 px-4 border-r border-gray-400">
                    {new Date(doc.createdAt).toLocaleDateString()}
                  </td>
                  <td className="py-2 px-4 border-r border-gray-400">{doc.docTitle}</td>
                  <td className="py-2 px-4 border-r border-gray-400">{doc.docHubTag}</td>
                  <td className="py-2 px-4 border-r border-gray-400">
                    { ' ' }
                    <Link className="text-teal-600 cursor-pointer" to={`/doc/${doc._id}`}>
                      Перейти
                    </Link>
                  </td>
                </tr>
              ))}
            </tbody>
          </table>
        </div>
      </>
    )
  )
}

```

```

    ): (
      <div className="text-xl">Поки у вас немає доступних документів</div>
    )}
  </div>
)
}

export default UserDocsList

```

Реалізація сторінки “Hubs”

Лістинг коду

```

import React, { useState, useEffect } from 'react'
import { useSelector } from 'react-redux'
import UserHubs from '../components/UserHubs'
import Notification from '../components/Notification'
import { useNavigate } from 'react-router-dom'

const Hubs = () => {
  const currentUser = useSelector((state) => state.user.currentUser)
  const [showModalCreateHub, setShowModalCreateHub] = useState(false)
  const [createHubTitle, setCreateHubTitle] = useState("")
  const [createHubTag, setCreateHubTag] = useState("")
  const [showNotification, setShowNotification] = useState(false)
  const [notificationMessage, setNotificationMessage] = useState("")
  const [isError, setIsError] = useState(false)

  const navigate = useNavigate()

  const handleCreateHub = async () => {
    try {
      const res = await fetch(`/api/hub/create`, {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
        },
        body: JSON.stringify({
          hubTitle: createHubTitle,
          hubTag: createHubTag,
        }),
      })
      if (res.ok) {
        setShowModalCreateHub(false)
        const data = await res.json()
        navigate(`/hub/${data}`)
      } else {
        const errorResponse = await res.json()
        setNotificationMessage(`Помилка: ${errorResponse.message}`)
        setIsError(true)
        setShowNotification(true)
        setTimeout(() => {
          setShowNotification(false)
        }, 3000)
      }
    } catch (error) {

```

```

    setNotificationMessage(`Помилка: ${error.message}`)
    setIsError(true)
    setShowNotification(true)
    setTimeout(() => {
      setShowNotification(false)
    }, 1500)
  }
}

return (
  <div className="flex flex-col min-h-screen bg-teal-200 items-center">
    {showNotification && <Notification message={notificationMessage} isError={isError} />}
    <div className="w-2/3 flex flex-col flex-grow bg-teal-100">
      <div className="flex flex-wrap">
        <div className="w-full lg:w-1/2 flex-grow">
          <div className="p-4 text-center">
            <div className="flex items-center justify-center">
              {currentUser.isAdmin && (
                <button
                  onClick={() => setShowModalCreateHub(true)}
                  className="bg-teal-500 hover:bg-teal-700 text-white mx-2 font-bold py-2 px-4
rounded"
                >
                  Створити новий підрозділ
                </button>
              )}
            </div>
          </div>
          <h2 className="text-3xl font-semibold text-teal-800">Ваші підрозділи</h2>
          <UserHubs />
        </div>
      </div>
    </div>
    {showModalCreateHub && (
      <div className="fixed top-0 left-0 w-full h-full flex items-center justify-center bg-gray-500
bg-opacity-75">
        <div className="bg-white p-4 rounded-lg max-w-full">
          <div className="text-center">
            <h3 className="text-xl font-bold mb-4">Введіть назву та аббревіатуру для
підрозділу</h3>
            <div className="flex flex-col items-center">
              <input
                type="text"
                placeholder="Назва підрозділу"
                value={createHubTitle}
                onChange={(e) => setCreateHubTitle(e.target.value)}
                className="border border-gray-400 p-2 mb-2"
              />

```

```

    <input
      type="text"
      placeholder="Абревіатура"
      value={createHubTag}
      onChange={(e) => setCreateHubTag(e.target.value)}
      className="border border-gray-400 p-2 mb-2"
    />
  </div>
  <div className="flex justify-center items-center">
    <button
      onClick={handleCreateHub}
      className="bg-teal-500 hover:bg-teal-700 text-white font-bold py-2 px-4 rounded
mr-4"
    >
      Створити
    </button>
    <button
      onClick={() => setShowModalCreateHub(false)}
      className="bg-red-300 hover:bg-red-700 text-white font-bold py-2 px-4 rounded"
    >
      Hi
    </button>
  </div>
</div>
</div>
  )}
</div>
)
}

export default Hubs

```