МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інтелектуальних інформаційних систем

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри інтелектуальних інформаційних систем

_____Юрій КОНДРАТЕНКО

«____»_____ 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА

НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВРА

ІНТЕГРОВАНА ПЛАТФОРМА ДЛЯ ОБРОБКИ ДОКУМЕНТІВ З ФУНКЦІЯМИ ЗВ'ЯЗКУ З ПОШТОВИМИ СЕРВІСАМИ

Спеціальність 122 Комп'ютерні науки

Освітня програма «Комп'ютерні науки»

Здобувач _____ Віталій ТИШИК

«___»_____ 2025 р.

Керівник д-р техн. наук, проф. _____Олександр ГОЖИЙ

«___»_____2025 р.

Миколаїв – 2025

Чорноморський національний університет імені Петра Могили

(повне найменування закладу вищої освіти)

Факультет	Комп’ютерних наук
Кафедра	Інтелектуальних інформаційних систем
Рівень вищої освіти	Перший (бакалаврський)
Освітній ступень	Бакалавр
Спеціальність	122 Комп’ютерні науки
Освітня програма	Комп’ютерні науки

ЗАТВЕРДЖУЮ

Завідувач кафедри інтелектуальних інформаційних систем

_____Юрій КОНДРАТЕНКО

«____» _____ 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу здобувача

Тишик Віталій Ігорович

((прізвище, ім’я, по батькові здобувача)

1. Тема кваліфікаційної роботи: «Інтегрована платформа для обробки документів з функціями зв’язку з поштовими сервісами».

Керівник роботи: Олександр Гожий, професор кафедри інтелектуальних інформаційних систем, д-р техн. наук, професор.

Затверджена наказом ЧНУ ім. Петра Могили від «18» листопада 2024 р. № 321.

2. Строк представлення кваліфікаційної роботи «___» _____ 2025 р.

3. Очікуваним результатом є створення інтегрованої веб-платформи для обробки електронних документів з можливістю їх надсилання через поштові сервіси. Система повинна забезпечувати:

- зберігання, перегляд і редагування документів різних форматів;

– інтеграцію з поштовим сервісом для надсилання документів (наприклад, через Gmail API або SMTP);

– інтерфейс користувача для взаємодії з документами та відправленням;

– базу даних для зберігання метаінформації про документи та історію надсилань.

4. Перелік питань, що підлягають розробці:

– аналіз існуючих рішень для обробки та надсилання електронних документів;

– вибір та обґрунтування архітектури інтегрованої платформи;

– огляд та вибір інструментів для реалізації функцій редагування, зберігання та надсилання документів;

– розробка інтерфейсу користувача для роботи з документами та надсилання їх через поштовий сервіс;

– інтеграція з поштовим сервісом (наприклад, Gmail або SMTP) для надсилання документів.

5. Перелік графічних матеріалів: презентація.

Керівник роботи _____ Олександр ГОЖИЙ

(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Здобувач _____ Віталій ТИШИК

(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Дата видачі завдання «18» грудня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

кваліфікаційної роботи

Тема: Інтегрована платформа для обробки документів з функціями зв’язку з поштовими сервісами.

№	Найменування роботи	Початок	Закінчення	Примітки
1	Отримання завдання на виконання КР	18.12.2024	20.12.2024	
2	Аналіз предметної області та постановка задачі	21.12.2024	30.01.2025	
3	Огляд літературних джерел за темою кваліфікаційної роботи, зокрема огляд публікацій та аналогічних систем, щодо інтегрованої платформи для обробки документів з функціями поштових сервісів	31.01.2025	01.03.2025	
4	Огляд існуючих архітектур інтегрованих платформ здля обробки документів для вирішення поставленої задачі	02.03.2025	01.04.2025	
5	Реалізація обраних технологій з аналізом отриманих результатів	02.04.2025	15.05.2025	
6	Перший попередній захист КР на засіданні комісії кафедри	16.05.2025	16.05.2025	
7	Корегування роботи за результатами попереднього захисту	29.05.2025	29.05.2025	
8	Другий попередній захист КР на засіданні комісії кафедри	30.05.2025	30.05.2025	
9	Доробка та остаточне оформлення КР	31.05.2025	10.05.2025	
10	Подання КР, її електронної копії та інших документів (відгуку, рецензії) до захисту	11.05.2025	17.05.2025	

Керівник роботи _____ Олександр ГОЖИЙ
(Особистий підпис) *(Власне ім’я ПРІЗВИЩЕ)*

Здобувач _____ Віталій ТИШИК
(Особистий підпис) *(Власне ім’я ПРІЗВИЩЕ)*

Дата складання календарного плану
«29» січня 2025 р.

АНОТАЦІЯ

до кваліфікаційної роботи
здобувача (ки) групи 401 ЧНУ ім. Петра Могили

Тишика Віталія Ігоровича

на тему: **“ІНТЕГРОВАНА ПЛАТОФРМА ДЛЯ ОБРОБКИ ДОКУМЕНТІВ З ФУНКЦІЯМИ ЗВ’ЯЗКУ З ПОШТОВИМИ СЕРВІСАМИ”**

Актуальність цієї кваліфікаційної роботи полягає в необхідності автоматизації процесів роботи з електронними документами в межах корпоративного середовища, зокрема створення, редагування, зберігання та надсилання документів. Інтеграція поштових сервісів у єдину платформу дозволяє підвищити ефективність комунікацій, зменшити кількість ручної роботи та забезпечити зручний і безпечний доступ до документації різними категоріями користувачів.

Об’єктом роботи є процес автоматизованого керування електронними документами в організаціях.

Предметом роботи є розробка інтегрованої веб-платформи для створення, редагування, зберігання та надсилання документів із використанням сучасних веб-технологій і поштових сервісів.

Мета роботи: розробити інтегровану платформу для обробки документів із вбудованими функціями поштових сервісів, яка автоматизує процеси створення, редагування, зберігання та надсилання документів у корпоративному середовищі.

Робота складається з чотирьох розділів. У першому розділі здійснено аналіз предметної області, вивчено існуючі рішення та сформульовано основні вимоги до системи. У другому розділі подано огляд інструментів і технологій, які використовуються під час розробки. Третій розділ присвячено моделюванню архітектури системи, підсистем та сценаріїв взаємодії користувача. У четвертому розділі описано процес реалізації програмного

продукту, особливості інтеграції з поштовими сервісами, а також результати тестування системи.

Функціонал системи розділено на підмодулі: модуль обробки документів, модуль поштової інтеграції та модуль керування користувачами. Результати реалізації підтверджують ефективність розробленої платформи для роботи з електронними документами, її масштабованість, зручність та безпеку.

Обсяг кваліфікаційної роботи становить 100 сторінки, містить 19 рисунків, 30 використаних джерел.

Ключові слова: інтегрована платформа, обробка документів, поштові сервіси, клієнт-серверна архітектура, зручний інтерфейс, Next.js, Clerk, CKEditor, Convex, Nodemailer.

ABSTRACT

to the qualification work by the student of the group 401 of Petro Mohyla Black Sea National University

Tyshyk Vitalii

“ INTEGRATED PLATFORM FOR DOCUMENT PROCESSING WITH EMAIL SERVICE INTEGRATION”

The relevance of this qualification work lies in the need to automate electronic document management processes within a corporate environment, particularly in the creation, editing, storage, and sending of documents. The integration of email services into a unified platform improves communication efficiency, reduces manual work, and ensures convenient and secure access to documentation for various categories of users.

The object of this work is the process of automated electronic document management in organizations.

The subject of the work is the development of an integrated web platform for creating, editing, storing, and sending documents using modern web technologies and email services.

The goal of the work is to develop an integrated platform for document processing with built-in email service functions, which automates the processes of creating, editing, storing, and sending documents in a corporate environment.

The work consists of four chapters. The first chapter provides an analysis of the subject area, explores existing solutions, and formulates the main system requirements. The second chapter presents an overview of the tools and technologies used during development. The third chapter is devoted to modeling the system architecture, its subsystems, and user interaction scenarios. The fourth chapter describes the implementation process of the software product, the features of integration with email services, and the results of system testing.

The system's functionality is divided into submodules: a document processing module, an email integration module, and a user management module. The implementation results confirm the effectiveness, scalability, usability, and security of the developed platform for working with electronic documents.

The volume of the thesis is 100 pages, contains 19 figures, and references 30 sources.

Keywords: integrated platform, document processing, email services, client-server architecture, user-friendly interface, Next.js, Clerk, CKEditor, Convex, Nodemailer.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧКИ ... 3

ВСТУП ... 4

1 ПОСТАНОВКА ЗАДАЧІ РОЗРОБКИ ІНТЕГРОВАНОЇ ПЛАТФОРМИ ДЛЯ ОБРОБКИ ДОКУМЕНТІВ З ФУНКЦІЯМИ ЗВ'ЯЗКУ З ПОШТОВИМИ СЕРВІСАМИ ... 5

1.1Опис предметної сфери ... **5**

1.2Огляд та аналіз наявних аналогів та публікацій ... **6**

1.3 Постановка задачі ... **15**

2 ТЕХНОЛОГІЇ ДЛЯ РОЗРОБКИ ІНТЕГРОВАНОЇ ПЛАТФОРМИ ДЛЯ ОБРОБКИ ДОКУМЕНТІВ З ФУНКЦІЯМИ ЗВ'ЯЗКУ З ПОШТОВИМИ СЕРВІСАМИ ... 18

2.1 Технології для вирішення поставленої задачі ... **18**

3 РОЗРОБКА ПРОГРАМНОГО ЗАСТОСУНКУ ТА АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ ... 28

3.1 Опис вхідних даних та структури програми ... **28**

3.2 Проектування/моделювання: ... **29**

3.3. Аналіз отриманих результатів ... **38**

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ... 46

4.1 Опис програмної реалізації ... **46**

4.2 Керівництво користувача ... **66**

ВИСНОВКИ ... 69

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ ... 70

ДОДАТОК Код застосунку ... 72

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧКИ

MVT – Model View Template

SPA – Single-Page Applications

UML – Unified Modeling Language

UI – User Interface

САПР – Система автоматизованого проектування і розрахунку

PDF – Portable Document Format

HTTP – HyperText Transfer Protocol

ПЗ – Програмне Забезпечення

ВСТУП

Останнім часом зростає потреба у швидкій, зручній та ефективній обробці електронних документів, особливо в умовах віддаленої роботи, ділового листування та зростаючих обсягів інформації. Все більше користувачів - як приватні особи, так і представники малого та середнього бізнесу - шукають рішення, які дозволяють не лише створювати та редагувати документи, а й одразу надсилати їх через поштові сервіси без необхідності перемикання між різними платформами. Це зумовлює актуальність створення інтегрованих платформ, які поєднують функції обробки документів з можливістю миттєвої інтеграції з поштовим сервісом.

Більшість існуючих рішень або надто складні у використанні, орієнтовані на досвідчених користувачів, або є частиною комерційних пакетів з високою вартістю. Деякі з них прив'язані до конкретних поштових або офісних екосистем і не підтримують роботу з іншими сервісами. Це створює труднощі для звичайних користувачів, які потребують простого та зрозумілого інструменту для виконання базових задач - створення, редагування, прикріплення та надсилання документів або файлів.

Метою роботи є створення зручної та універсальної платформи, яка дозволить обробляти документи у різних форматах (PDF, DOCX, TXT тощо), а також інтегруватися з поштовим сервісамом, таким як Gmail. Однією з ключових особливостей системи стане простий інтерфейс, орієнтований на пересічного користувача без спеціальних технічних знань. Платформа також надаватиме можливість попереднього перегляду документів, вибору одержувачів, автоматичного прикріплення файлів.

Розробка такої системи дозволить значно спростити обмін документами, зменшити кількість рутинних дій і підвищити ефективність роботи як окремих користувачів, так і цілих організацій.

1 ПОСТАНОВКА ЗАДАЧІ РОЗРОБКИ ІНТЕГРОВАНОЇ ПЛАТФОРМИ ДЛЯ ОБРОБКИ ДОКУМЕНТІВ З ФУНКЦІЯМИ ЗВ’ЯЗКУ З ПОШТОВИМИ СЕРВІСАМИ

Метою роботи є дослідження технологій та методів, які потрібні для розробки інтегрованої програмної платформи для зручної обробки електронних документів з можливістю їх прямого надсилання через поштовий сервіс.

1.1 Опис предметної сфери

Програмне забезпечення - інтегрована система обробки документів із можливістю надсилання через поштові сервіси - вирішує всі основні задачі, пов’язані з підготовкою та передаванням текстових документів. А саме:

– створення та редагування документів DOCX;

– форматування та збереження документів відповідно до потреб користувача;

– миттєве надсилання документів через електронну пошту (Gmail);

– кономія часу завдяки об’єднанню кількох інструментів в одному інтерфейсі;

– мобільність та доступність - система працює у вебсередовищі та не потребує встановлення додаткових програм.

Платформа здатна працювати віддалено: немає необхідності завантажувати окремі програми чи працювати з документацією на локальному пристрої - вся взаємодія здійснюється через браузер, що особливо зручно для тих, хто працює в умовах обмеженого доступу до офісного ПЗ.

Реалізація такої автоматизації враховує тип документа, його структуру, обсяг. Система автоматично оптимізує формат документу для надсилання, забезпечує відповідність технічним вимогам поштових платформ (наприклад, обмеження на розмір вкладень або типи файлів).

Для вирішення задач, з якими стикається користувач у процесі підготовки документації, платформа вимагає від користувача лише мінімального обсягу вхідних даних: змісту документа та електронної адреси одержувача. У разі потреби користувач також може налаштувати тему та текст супровідного листа.

На виході користувач отримує повністю підготовлений до надсилання файл, який може бути негайно відправлений на електронну пошту, або збережений для подальшого використання.

Таким чином, кінцевий результат роботи програмного забезпечення — це зручна, візуально зрозуміла, функціональна система, яка дозволяє значно спростити процес створення, редагування та пересилання документів, не вимагаючи від користувача глибоких знань у галузі офісних технологій або адміністрування електронної пошти.

Існуючі системи [1]:

– DocuWare;
– Egnyte;
– Google Workspace;
– SharePoint;
– Only office Workspace.

1.2 Огляд та аналіз наявних аналогів та публікацій

1.2.1 DocuWare

DocuWare [2] - це потужна хмарна платформа для управління документами та автоматизації бізнес-процесів, що пропонує широкий спектр інструментів для оптимізації зберігання, обробки та обміну документами в організації.

Переваги:

– інтеграція з іншими системами: платформа підтримує інтеграцію з популярними корпоративними системами (CRM, ERP тощо), що дозволяє автоматизувати обмін даними та спростити роботу між різними відділами;

– автоматизація робочих процесів: DocuWare дозволяє автоматизувати ключові бізнес-процеси, такі як погодження документів, обробка заявок та контроль виконання завдань, що суттєво підвищує ефективність компанії;

– безпека даних: платформа гарантує високий рівень захисту даних завдяки використанню шифрування, багатофакторної автентифікації та регулярних резервних копій;

– мобільний доступ: зручний доступ до документів і бізнес-процесів за допомогою мобільного додатку, що дозволяє працювати з будь-якого пристрою в будь-який час, навіть в дорозі;

– колективна робота з документами: користувачі можуть спільно працювати над документами в реальному часі, що покращує командну взаємодію і пришвидшує виконання завдань.

Недоліки:

– складність для нових користувачів: для ефективного використання DocuWare потрібні певні професійні навички, що може стати перешкодою для користувачів без досвіду роботи з подібними системами;

– залишкова залежність від інтернету: оскільки DocuWare є хмарною платформою, для доступу до даних необхідне стабільне з'єднання з Інтернетом, що може стати проблемою при нестабільному з'єднанні;

– вартість: ціна на підписку на DocuWare може бути високою для малих підприємств або стартапів, що робить платформу менш доступною для широкого кола користувачів;

– можливості для малих організацій: для невеликих компаній деякі функції можуть бути надмірними або не зовсім необхідними, що може зробити систему менш ефективною для їхніх потреб.

DocuWare забезпечує зручність, ефективність і безпеку при роботі з документами, дозволяючи організаціям значно спростити свої внутрішні бізнес-процеси, знизити витрати часу на документообіг і підвищити загальну продуктивність.

1.2.2 Egnyte

Egnyte [3] - це потужна хмарна платформа для зберігання, обміну та спільної роботи з документами, що забезпечує високу ефективність управління даними та автоматизацію бізнес-процесів для підприємств усіх розмірів.

Переваги:

– інтеграція з іншими системами: Egnyte підтримує інтеграцію з популярними корпоративними системами (CRM, ERP, Microsoft Office, Google Workspace тощо), що дозволяє забезпечити безперебійний обмін даними між різними відділами та платформами;

– автоматизація робочих процесів: платформа надає інструменти для автоматизації робочих процесів, зокрема управління правами доступу до документів, збереження версій і контролю за виконанням завдань, що значно підвищує ефективність і знижує витрати часу;

– безпека даних: Egnyte гарантує високий рівень захисту даних за допомогою шифрування, багатофакторної автентифікації, ретельного моніторингу активності користувачів і регулярних резервних копій, що робить платформу надійним рішенням для зберігання важливої інформації;

– колективна робота з документами: Egnyte дозволяє декільком користувачам одночасно працювати з документами в реальному часі, що спрощує командну взаємодію та пришвидшує виконання завдань;

– кастомізація доступу: платформа дає змогу налаштовувати рівні доступу та управління правами для кожного користувача, що забезпечує зручне і безпечне співробітництво у команді.

Недоліки:

– складність для нових користувачів: для ефективного використання Egnyte потрібні певні навички в роботі з хмарними платформами;

– залежність від Інтернету: оскільки Egnyte є хмарною платформою, для доступу до документів необхідне стабільне підключення до Інтернету;

– вартість: підписка на Egnyte може бути дорогою для малих підприємств або стартапів, що обмежує доступність платформи для деяких користувачів. Однак, для середніх і великих компаній інвестиція в таку систему є обґрунтованою.

Egnyte - це ідеальне рішення для компаній середнього та великого розміру, які потребують потужного інструменту для зберігання та обміну документами. Платформа дозволяє значно спростити управління даними, забезпечуючи високу безпеку, зручність для користувачів і ефективність процесів. Проте для малих компаній чи тих, хто потребує лише базових функцій зберігання та обміну, вона може бути занадто складною або дорогою.

1.2.3 Google Workspace

Google Workspace [4] - це хмарна платформа для бізнесу, яка включає в себе набір інструментів для роботи з електронною поштою, документами, таблицями, презентаціями, відеоконференціями та іншими корпоративними завданнями. Вона надає функціональні можливості для співпраці та організації робочих процесів.

Переваги:

– інтеграція з іншими інструментами Google: Google Workspace чудово інтегрується з іншими інструментами Google, такими як Google Drive, Google Calendar, Google Meet тощо, що дозволяє спростити робочі процеси. Інтеграція між

цими сервісами забезпечує зручний обмін файлами, автоматичне оновлення календарів і багато інших корисних функцій;

– масштабованість і гнучкість: Google Workspace підходить як для малих, так і для великих компаній завдяки своїй масштабованості. Користувачі можуть додавати нові облікові записи за необхідності, що робить платформу гнучкою для різних організаційних структур;

– безпека та конфіденційність: платформа надає високий рівень захисту даних: шифрування при передачі та зберіганні даних, багатофакторну автентифікацію, а також можливість налаштування прав доступу для користувачів. Це робить її надійним рішенням для збереження корпоративної інформації;

– спільна робота в реальному часі: однією з основних переваг Google Workspace є можливість спільної роботи в реальному часі над документами, таблицями та презентаціями. Це значно спрощує колаборацію між командами, що працюють в різних локаціях;

– мобільність і доступність: інструменти Google Workspace доступні через веб-браузери та мобільні додатки, що дає змогу працювати з документами і комунікувати в будь-який час і з будь-якого пристрою;

– інтуїтивно зрозумілий інтерфейс: платформа має зручний і простий інтерфейс, що дозволяє легко орієнтуватися навіть новачкам. Її інтерфейс є знайомим для користувачів, які вже працюють з продуктами Google.

Недоліки:

– залежність від Інтернету: як і будь-яка хмарна платформа, Google Workspace вимагає стабільного підключення до Інтернету для роботи. У разі нестабільного з'єднання чи відсутності доступу до мережі, робота з документами і поштою стає неможливою;

– ціна для великих команд: хоча Google Workspace має доступні плани для малих компаній, для великих команд вартість підписки може стати суттєвим

чинником, оскільки кожен користувач потребує окремої ліцензії. З урахуванням великої кількості користувачів, це може бути дорогим рішенням;

– обмеження в порівнянні з більш спеціалізованими інструментами: хоча Google Workspace включає багато основних інструментів для роботи з документами і комунікацій, вона не завжди має такі ж спеціалізовані функції, як інші програмні пакети (наприклад, Microsoft Office 365). Це може бути обмеженням для організацій, які потребують більш розширених можливостей;

– обмеження на кастомізацію: у порівнянні з іншими рішеннями, Google Workspace має обмежені можливості для кастомізації. Хоча інтерфейс досить зручний і адаптований для багатьох користувачів, для великих організацій, що потребують повного контролю над своїм робочим середовищем, це може бути недоліком;

– збереження даних у хмарі: Google Workspace зберігає дані на своїх серверах, що може викликати занепокоєння у компаній, які мають особливі вимоги щодо зберігання або обробки конфіденційної інформації (наприклад, у випадку з регламентами збереження даних в окремих країнах);

– залежність від екосистеми Google: якщо організація вже використовує інші інструменти або сервіси, які не інтегруються з Google Workspace, це може створити додаткові проблеми при переході на платформу. Відсутність підтримки сторонніх програм у деяких випадках може обмежити можливості користувачів.

Google Workspace є потужною та гнучкою платформою для управління бізнес-процесами та документами. Вона відмінно підходить для компаній середнього та великого розміру, які вже використовують інші сервіси Google або потребують інтуїтивно зрозумілих інструментів для співпраці. Однак, для організацій з особливими вимогами до кастомізації або без стабільного Інтернет-з'єднання, ця платформа може не бути ідеальним вибором.

1.2.4 SharePoint

SharePoint [5] - це корпоративна платформа від Microsoft для організації зберігання, обміну, пошуку й спільної роботи з документами, а також створення внутрішніх порталів, автоматизації бізнес-процесів і керування контентом в організаціях.

Переваги:

– **глибока інтеграція з Microsoft 365**. Платформа безшовно працює з Word, Excel, Outlook, Teams та іншими сервісами Microsoft, забезпечуючи ефективну екосистему для бізнесу;

– **потужні можливості зберігання й керування документами**. Підтримує контроль версій, історію змін, права доступу на рівні документів і папок, а також інструменти для організації документообігу;

– **гнучке налаштування**. Дозволяє створювати власні сайти, бібліотеки, списки та кастомні рішення відповідно до потреб організації;

– **автоматизація процесів (Power Automate)**. Підтримує створення автоматизованих сценаріїв для рутинних завдань - наприклад, затвердження документів або оповіщення;

– **спільна робота**. Користувачі можуть одночасно редагувати документи, залишати коментарі й працювати в команді через браузер;

– **безпека корпоративного рівня**. Платформа відповідає високим стандартам захисту даних, підтримує аудит, політики доступу, шифрування тощо.

Недоліки:

– **складність впровадження та навчання**. Для налаштування платформи й повного використання її можливостей можуть знадобитися спеціалісти та значний час на навчання користувачів;

– **перевантаженість функціоналом**. Часто для базових потреб малого бізнесу функцій надто багато, що ускладнює користування та адміністрування;

– **вартість**. Повноцінна інтеграція SharePoint із Microsoft 365 або на базі локального сервера може бути витратною для невеликих організацій;

– **залежність від Microsoft**. Екосистема сильно прив’язана до інших продуктів Microsoft, що може обмежувати гнучкість при інтеграції з альтернативними рішеннями;

– **інтерфейс не завжди інтуїтивний**. Хоча дизайн оновлюється, деякі функції залишаються захованими або складними для новачків без досвіду з корпоративними системами.

1.2.5 OnlyOffice Workspace

OnlyOffice Workspace [6] - це комплексне офісне рішення з відкритим кодом, яке об’єднує створення та спільну роботу з документами, керування проєктами, CRM, електронну пошту, календарем і корпоративним порталом. Платформа доступна як у хмарному варіанті, так і для локального розгортання.

Переваги:

– **потужний офісний пакет з підтримкою форматів Microsoft Office**. OnlyOffice має повноцінний набір редакторів для DOCX, XLSX, PPTX, які сумісні з Microsoft Office на високому рівні, збереження структури й стилів гарантоване;

– **підтримка самостійного розгортання (on-premise)**. Можна встановити систему на власному сервері для повного контролю над даними, що важливо для організацій з підвищеними вимогами до безпеки;

– **інтеграція з іншими сервісами**. Платформа підтримує інтеграцію з Nextcloud, ownCloud, Seafile, Alfresco, Moodle, WordPress, Jira та іншими популярними системами;

– **багатофункціональність**. Крім роботи з документами, включає CRM, пошту, календар, керування проєктами, систему завдань і портал співробітників;

– **конкурентна ціна**. Вартість підписки нижча, ніж у більшості західних аналогів, а для особистого чи некомерційного використання є безкоштовна версія;

– **прозорість та відкритий код**. Частина функціоналу доступна як open source, що дає змогу оцінити безпеку, розширювати систему й адаптувати під свої потреби.

Недоліки:

– **менш зручний інтерфейс порівняно з Google Workspace або Microsoft 365**. Інтерфейс хоч і функціональний, але іноді виглядає застарілим або менш інтуїтивним, що може впливати на зручність використання;

– **вища вимога до технічної підтримки при локальному розгортанні**. Якщо обрати варіант встановлення на власний сервер - потрібні базові адміністраторські знання Linux/Windows Server;

– **відносно слабка підтримка мобільних пристроїв**. Хоч мобільні додатки є, їхній функціонал поступається десктопним версіям або аналогам від Google/Microsoft;

– **обмежена екосистема**. Менше сторонніх інтеграцій та модулів у порівнянні з більш розвиненими платформами, як-от Microsoft 365 або Google Workspace;

– **слабша підтримка шаблонів і автоматизації**. У деяких кейсах (наприклад, створення складних шаблонів або форм) можливості поступаються великим конкурентам.

1.2.6 Огляд наукової публікації у журналі IRE Journals

У журналі **IRE Journals[7]** було опубліковано статтю (2024), присвячену порівнянню п’яти провідних систем управління документами, серед яких була і DocuWare. У цьому дослідженні аналізувалися такі аспекти, як безпека інформації, функціональність, інтеграція з іншими сервісами та зручність використання.

Основні результати дослідження щодо DocuWare:

– **безпека:** DocuWare отримала високі оцінки за впровадження багаторівневої автентифікації та шифрування даних, що забезпечує надійний захист інформації;

– **функціональність:** Система відзначена за широкий набір інструментів для автоматизації бізнес-процесів і підтримку колективної роботи;

– **інтеграція:** DocuWare має хорошу сумісність із корпоративними CRM та ERP системами, що полегшує її впровадження у великі організації;

– **зручність:** Оцінена як інтуїтивно зрозуміла, проте має деяку складність для новачків, що потребує навчання користувачів.

Автори статті підкреслюють, що DocuWare є одним із найефективніших рішень для великих підприємств, проте може бути надмірною за функціоналом для малих компаній.

У зв’язку з цим, розроблена в рамках цієї роботи інтегрована платформа для обробки документів орієнтована на створення більш простої та зручної у використанні альтернативи, що задовольняє потреби малого та середнього бізнесу, а також окремих користувачів. Платформа фокусується на мінімізації необхідних навичок для роботи, забезпеченні швидкого редагування, збереження та надсилання документів через поштові сервіси, зберігаючи при цьому достатній рівень безпеки та зручності інтерфейсу. Такий підхід дозволяє значно скоротити час на навчання користувачів та оптимізувати робочі процеси без складних налаштувань та дорогих інтеграцій.

1.3 Постановка задачі

Метою роботи є визначення та дослідження такої теми як інтегрована платформа для обробки документів з функціями зв’язку з поштовими сервісами. Система має забезпечити зручну роботу з документами (PDF, DOCX), можливість завантаження зображень і їх зберігання, а також потрібно інтегрувати функціонал поштового сервісу Gmail для їх надсилання.

Для досягнення цієї мети необхідно реалізувати наступні задачі:

– **розробити інтерфейс користувача**, який дозволяє легко редагувати та переглядати документи;

– **забезпечити інтеграцію з поштовом сервісамом Gmail**, зокрема реалізувати можливість надсилання документів безпосередньо з платформи;

– **реалізувати функції обробки документів**, включаючи збереження, перегляд та форматування тексту;

– **забезпечити збереження та безпеку даних**, зокрема контроль доступу, шифрування та обмеження прав користувача.

Результатом виконання проєкту має стати повнофункціональна веб-платформа, яка дозволить користувачам швидко створювати документи, редагувати їх, додавати вкладення та надсилати безпосередньо з інтерфейсу без залучення сторонніх програм.

Висновки до розділу

У цьому розділі було проаналізовано предметну область, пов'язану з обробкою та надсиланням електронних документів, а також проведено огляд сучасних програмних рішень, що реалізують подібний функціонал. На основі аналізу виявлено, що існуючі платформи, такі як DocuWare, OnlyOffice Workspace, SharePoint, Google Workspace та Egnyte, мають потужний функціонал для корпоративного документообігу, проте здебільшого орієнтовані на великі організації та потребують складного налаштування або мають високу вартість володіння.

Більшість з розглянутих рішень не надають зручних інструментів для інтерактивної обробки вмісту документів перед надсиланням або потребують сторонньої інтеграції з поштовими сервісами, такими як Gmail. Крім того, їх інтерфейси можуть бути перевантажені або не адаптовані для потреб користувачів, які шукають просте й інтуїтивне рішення для роботи з окремими електронними документами.

Таким чином, виявлено необхідність розробки легкої, гнучкої та інтерактивної платформи, яка б поєднувала інструменти для створення, редагування та структурування електронних документів з можливістю безпосереднього їх надсилання через поштові сервіси. Такий підхід дозволить оптимізувати процеси документообігу, зробити їх зручними як для індивідуальних користувачів, так і для малих організацій, не вдаючись до складних корпоративних систем.

2 ТЕХНОЛОГІЇ ДЛЯ РОЗРОБКИ ІНТЕГРОВАНОЇ ПЛАТФОРМИ ДЛЯ ОБРОБКИ ДОКУМЕНТІВ З ФУНКЦІЯМИ ЗВ’ЯЗКУ З ПОШТОВИМИ СЕРВІСАМИ

2.1 Технології для вирішення поставленої задачі

У межах реалізації системи збереження та обробки файлів було використано низку сучасних технологій та інструментів, що забезпечують зручність розробки, гнучкість у роботі з даними та зручний інтерфейс для користувача. Основу клієнтської частини складає Next.js з використанням TypeScript, що забезпечує типобезпечність та покращену підтримку під час розробки. Оформлення інтерфейсу виконане за допомогою HTML, CSS та бібліотеки UI-компонентів **shadcn/ui**, що дозволяє швидко створювати сучасний дизайн. Для стилізації також застосовано **Tailwind CSS**, що забезпечує гнучкий та зручний спосіб створення адаптивного інтерфейсу з мінімальним обсягом коду.

Для аутентифікації користувачів інтегрована система **Clerk**, яка надає простий і безпечний механізм керування сесіями та користувачами. Зберігання даних і логіка обробки реалізовані на базі **Convex** – backend-as-a-service рішення, що дозволяє ефективно працювати з базою даних у реальному часі.

Редагування текстових файлів реалізовано за допомогою вбудованого редактора **CKEditor 5 (classic build)**, який забезпечує багатофункціональне редагування контенту без потреби у сторонніх програмах. Для роботи з датами та часом використовується бібліотека **date-fns**, що надає зручний та легкий API для форматування і обробки дат.

Для надсилання відредагованих файлів поштою використано **SMTP-протокол** у поєднанні з бібліотекою **nodemailer**, що дозволяє інтегрувати функціонал електронної пошти безпосередньо у веб-додаток. Додаткову гнучкість в організації та побудові інтерфейсних елементів забезпечує використання **radix-ui** – низькорівневої бібліотеки доступних та кастомізованих UI-компонентів.

2.1.1 HTML

HTML (HyperText Markup Language)[8] — це стандартизована мова розмітки, призначена для створення структури веб-сторінок. Вона дозволяє визначати різні елементи контенту, такі як заголовки, абзаци, списки, посилання, зображення та інші, за допомогою спеціальних тегів.

Кожен HTML-документ має базову структуру, яка включає такі основні теги:

– <!Doctype html> - визначає тип документа і версію HTML;

– <html> - кореневий елемент, що охоплює весь вміст сторінки;

– <head> - містить метадані документа;

– <body> - містить безпосередній вміст сторінки, який відображається в браузері.

HTML також підтримує використання атрибутів для надання додаткової інформації про елементи. Наприклад, атрибут href у тегу <a> визначає адресу посилання.

Семантичні теги, такі як <header>, <nav>, <main>, <section>, <article>, <footer>, допомагають краще структурувати контент і полегшують розуміння сторінки як для користувачів, так і для пошукових систем.

Для створення більш привабливого та зручного інтерфейсу HTML часто використовується разом з CSS (каскадними таблицями стилів), які відповідають за візуальне оформлення елементів, та JavaScript, який додає інтерактивність.

Знання HTML є фундаментальним для веб-розробників, оскільки воно забезпечує основу для створення структурованих і доступних веб-сторінок.

2.1.2 CSS

CSS (Cascading Style Sheets)[8] - це мова стилів, яка використовується для опису зовнішнього вигляду HTML-документів. Вона дозволяє визначати стилі для елементів веб-сторінки, такі як кольори, шрифти, розміри, відступи та розташування, що сприяє відокремленню структури документа від його

візуального представлення.

Основні переваги використання CSS включають:

– **розділення контенту та дизайну**: HTML відповідає за структуру, тоді як CSS - за стильове оформлення, що полегшує підтримку та оновлення веб-сторінок.

– **можливість повторного використання стилів**: один файл CSS може застосовуватися до кількох HTML-документів, забезпечуючи узгоджений вигляд сайту.

– **покращення доступності та зручності користування**: завдяки гнучкому оформленню елементів інтерфейсу.

CSS дозволяє створювати адаптивні дизайни, які підлаштовуються під різні розміри екранів та пристрої, що є важливим аспектом сучасної веб-розробки.

Використання CSS у поєднанні з HTML є фундаментальним для створення ефективних, естетично привабливих та функціональних веб-сторінок.

2.1.3 TypeScript

TypeScript - це мова програмування, розроблена компанією Microsoft у 2012 році, яка є строго типізованим надмножиною мови JavaScript. Вона додає до JavaScript статичну типізацію, а також підтримку об’єктно-орієнтованого програмування - класи, інтерфейси, модулі, що дозволяє створювати масштабовані та підтримувані веб-додатки.

Основні переваги TypeScript:

– статична типізація - можливість явно задавати типи змінних, параметрів функцій та значень, що повертаються. Це дозволяє виявляти помилки ще на етапі компіляції, що підвищує надійність програмного коду;

– сумісність з ECMAScript - підтримка сучасних стандартів JavaScript, включаючи синтаксис останніх версій, таких як шаблонні рядки, деструктуризація, стрілочні функції тощо;

– підтримка ООП[9] - застосування класів, інтерфейсів, наслідування і модульної структури дає змогу організовувати код у зручний і логічний спосіб;

– сумісність з JavaScript - будь-який коректний код на JavaScript є допустимим у TypeScript, що дозволяє поступово інтегрувати його в існуючі проєкти.

Код на TypeScript компілюється у звичайний JavaScript, що забезпечує його використання як у браузерах, так і на серверних платформах, таких як Node.js.

Застосування TypeScript у веб-проєктах значно підвищує читабельність і підтримуваність коду, полегшує роботу великих команд розробників та дозволяє ефективно масштабувати функціонал програмного продукту.

2.1.4 Clerk

Clerk - це сучасна платформа для автентифікації та управління користувачами, яка надає розробникам повний набір вбудованих інтерфейсів користувача (UI), гнучких API та адміністративних панелей для інтеграції функціоналу входу, реєстрації та керування обліковими записами у веб-додатках. Вона особливо ефективна при розробці додатків на основі React або Next.js.

Основні можливості Clerk:

– вбудовані UI-компоненти: надає готові до використання компоненти для входу, реєстрації, профілю користувача та управління організаціями, які легко інтегруються та налаштовуються відповідно до дизайну додатка;

– підтримка багатьох методів автентифікації: включає автентифікацію за допомогою пароля, соціальних мереж (Google, GitHub тощо), одноразових кодів через електронну пошту або SMS, а також багатофакторну автентифікацію;

– управління сесіями та безпека:[11] Clerk забезпечує повний цикл управління сесіями, включаючи моніторинг активних пристроїв, відкликання сесій та захист від шахрайських дій, таких як блокування тимчасових електронних адрес;

– інтеграція з популярними фреймворками[10]: підтримує інтеграцію з такими фреймворками, як Next.js, React, Vue, Astro, Expo та іншими, що дозволяє швидко додати безпечну автентифікацію до різноманітних додатків.

Використання Clerk у веб-проєктах значно спрощує реалізацію функціоналу автентифікації та управління користувачами, забезпечуючи високий рівень безпеки та зручності для кінцевих користувачів.

2.1.5 Next.js

Next.js[12] - це фреймворк з відкритим вихідним кодом, створений компанією Vercel, який дозволяє розробляти веб-додатки на основі бібліотеки React з підтримкою рендерингу на стороні сервера (SSR) та генерації статичних веб-сайтів (SSG).

Основні можливості Next.js включають [13]:

– рендеринг на стороні сервера (SSR): дозволяє генерувати HTML-код на сервері перед його відправкою клієнту, що покращує продуктивність і SEO-оптимізацію;

– генерація статичних сторінок (SSG): дозволяє створювати сторінки під час компіляції, що забезпечує швидке завантаження та високу продуктивність;

– інкрементальна статична генерація (ISR): дозволяє оновлювати вже згенеровані сторінки без повної перекомпіляції всього сайту;

– підтримка TypeScript: вбудована підтримка мови TypeScript для забезпечення статичної типізації та покращення якості коду;

– автоматичне розділення коду: забезпечує завантаження лише необхідного коду для кожної сторінки, що покращує швидкість завантаження;

– гнучка маршрутизація: на основі файлової структури, що спрощує створення нових сторінок і маршрутів.

Next.js активно використовується для створення високопродуктивних веб-додатків, забезпечуючи розробникам потужні інструменти для реалізації сучасних веб-рішень.

2.1.6 Convex

Convex[14] - це сучасна платформа типу Backend-as-a-Service (BaaS), яка поєднує в собі реактивну базу даних, серверні функції та клієнтські бібліотеки, що дозволяє розробникам створювати повноцінні веб-додатки без необхідності налаштовувати інфраструктуру. Convex надає можливість писати серверний код на TypeScript, який виконується безпосередньо в базі даних, забезпечуючи автоматичну синхронізацію стану між клієнтом і сервером.

Основні особливості Convex[15]:

– реактивна база даних: забезпечує автоматичне оновлення даних на клієнті при зміні стану на сервері без необхідності використання WebSocket або ручного керування кешем;

– серверні функції на TypeScript: дозволяє писати серверну логіку у вигляді функцій, які виконуються в контексті бази даних, забезпечуючи типобезпечність та спрощення розробки;

– інтеграція з фреймворками: підтримує інтеграцію з популярними фреймворками, такими як React та Next.js, що дозволяє легко додавати функціональність бекенду до фронтенд-додатків;

– підтримка масштабування: Convex автоматично масштабується відповідно до навантаження, що забезпечує стабільну роботу додатків при зростанні кількості користувачів.

Використання Convex у веб-проєктах дозволяє розробникам зосередитися на бізнес-логіці додатку, зменшуючи час та зусилля, необхідні для налаштування та підтримки серверної інфраструктури.

2.1.7 SMPT

SMTP[16] (англ. *Simple Mail Transfer Protocol* - Простий протокол передачі пошти) - це стандартний мережевий протокол, призначений для передачі електронної пошти в мережах TCP/IP. Він використовується для надсилання електронних листів від клієнтських програм до поштових серверів, а також для пересилання листів між серверами.

Основні характеристики SMTP:

– **призначення[17]**: SMTP використовується переважно для надсилання електронної пошти. Для отримання листів зазвичай застосовуються інші протоколи, такі як POP3 або IMAP.

– **порти**: за замовчуванням SMTP працює через порт 25. Для безпечної передачі даних часто використовуються порти 465 (з SSL) або 587 (з TLS).

– **розширення ESMTP**: розширений протокол ESMTP (Extended SMTP) додає додаткові можливості, такі як аутентифікація відправника, передача двійкових даних та підтримка різних методів шифрування.

– **безпека**: для забезпечення безпеки передачі електронної пошти SMTP може використовувати шифрування TLS, а також механізми автентифікації, такі як SPF, DKIM та DMARC, що допомагають запобігти спаму та фішингу.

2.1.8 Tailwind CSS

Tailwind CSS[18] - це сучасна утилітарна CSS-фреймворк, орієнтована на швидку та ефективну розробку інтерфейсів без написання власного CSS-коду. Вона дозволяє створювати стилізовані компоненти безпосередньо у HTML-розмітці за допомогою заздалегідь визначених класів.

Основні характеристики**[19]**:

– **принцип утилітарності**: Tailwind працює за принципом «утилітарного CSS», що означає використання великої кількості маленьких класів, кожен з яких

відповідає за окрему властивість (наприклад, p-4 - внутрішній відступ**,** text-center - центрування тексту, bg-blue-500 - фон синього кольору);

– **конфігурованість**: за допомогою файлу tailwind.config.js розробники можуть змінювати тему, розширювати палітру кольорів, додавати власні класи або змінювати існуючі;

– **модульність та масштабованість**: Tailwind ідеально підходить для великих проектів, де важлива уніфікація стилів. Створення дизайн-системи на базі Tailwind дозволяє уникнути дублювання коду;

– **сумісність з сучасними фреймворками[20]**: Tailwind добре інтегрується з React, Next.js, Vue та іншими сучасними бібліотеками й фреймворками;

– **інструменти оптимізації**: під час продакшн-збірки Tailwind автоматично видаляє всі невикористані класи зменшуючи розмір кінцевого CSS-файлу.

Tailwind CSS дозволяє розробникам швидко створювати адаптивні та сучасні інтерфейси, не відволікаючись на написання кастомного CSS-коду, що особливо корисно у прототипуванні та при розробці масштабованих інтерфейсів.

2.1.9 Nodemailer

Nodemailer[21] - це популярна бібліотека для Node.js, яка дозволяє надсилати електронні листи з серверної частини застосунку. Вона підтримує різні транспортні механізми (SMTP, OAuth2, Amazon SES, Sendmail тощо) та є простим і гнучким рішенням для реалізації email-функціоналу в JavaScript/TypeScript проєктах.

Основні характеристики Nodemailer[22]:

– призначення: використовується для відправлення електронної пошти з Node.js-додатків;

– протоколи та транспорти[23]: підтримує SMTP (у тому числі з TLS/SSL), а також інтеграцію з OAuth2 та зовнішніми сервісами на кшталт Gmail або Amazon SES;

– аутентифікація: можлива робота з базовою SMTP-автентифікацією, OAuth2, або навіть без неї (у випадку локального сервера);

– підтримка вкладень: Nodemailer дозволяє прикріплювати файли до листів у вигляді вкладень, зображень, HTML-контенту тощо;

– формати листів: підтримує plain text, HTML-розмітку, а також автоматичне створення альтернативного текстового варіанту HTML-листа.

2.1.10 date-fns

date-fns[24] - це сучасна JavaScript-бібліотека для роботи з датами, яка надає великий набір функцій для форматування, обчислень і маніпуляцій з датами у зручному функціональному стилі. Вона є легкою альтернативою бібліотекам на кшталт Moment.js, забезпечуючи модульність і високу продуктивність.

Основні характеристики date-fns[25]:

– призначення: забезпечує понад 200 функцій для обробки дат - форматування, додавання/віднімання часу, порівняння, парсинг тощо;

– модульність: кожна функція імпортується окремо, що дозволяє зменшити розмір підсумкового JavaScript-коду (tree-shaking);

– функціональний стиль: усі функції є чистими (pure) і не змінюють передані об’єкти дат, що сприяє надійності та передбачуваності коду;

– підтримка локалізації[26]: підтримує десятки мов (у тому числі українську), що дозволяє форматувати дати відповідно до регіональних стандартів;

– сумісність: працює з native об’єктами Date, не вимагаючи спеціального формату чи типу.

2.1.11 radix-ui

Radix UI[27] - це набір низькорівневих, доступних (accessible) компонентів інтерфейсу для React, який дозволяє створювати складні UI-елементи з повним контролем над стилізацією. Radix не нав’язує зовнішній вигляд, натомість

забезпечує готову функціональність і дотримання стандартів доступності (ARIA), що робить його ідеальним для побудови кастомних дизайн-систем.

Основні характеристики Radix UI[28]:

- призначення: надає повністю контрольовані, безстилеві компоненти, такі як модальні вікна, вкладки, спливаючі підказки, меню тощо;
- доступність (Accessibility): усі компоненти відповідають сучасним вимогам WCAG і мають ARIA-атрибути «з коробки»;
- без стилів: компоненти постачаються без CSS-оформлення, що дає змогу застосовувати будь-яку систему стилізації - Tailwind CSS, styled-components, CSS Modules тощо;
- складність і масштабованість: підходить як для простих UI-елементів, так і для складних інтерфейсних рішень;
- платформа: створена спеціально для React і активно підтримується спільнотою.

Висновки до розділу

У цьому розділі було розглянуто основні технології, використані під час розробки вебзастосунку для збереження та обробки електронних документів. Зокрема, Next.js забезпечив ефективну побудову серверно-клієнтської архітектури, TypeScript підвищив надійність коду, а Clerk дозволив реалізувати сучасну систему аутентифікації. Для роботи з текстовим вмістом було інтегровано редактор CKEditor, що надає користувачам інтуїтивний інтерфейс для редагування документів. Збереження даних реалізовано на основі хмарної бази даних Convex, яка забезпечує високу швидкодію та масштабованість. Бібліотеки Tailwind CSS, shadcn/ui та radix-ui дозволили створити сучасний, адаптивний і доступний інтерфейс. Разом усі ці інструменти створили міцну основу для реалізації функціонального, безпечного та зручного у використанні вебзастосунку.

3 РОЗРОБКА ПРОГРАМНОГО ЗАСТОСУНКУ ТА АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ

3.1 Опис вхідних даних та структури програми

У межах розробки інтегрованої веб-платформи для обробки електронних документів основним джерелом вхідних даних є користувачі системи. Система дозволяє зареєстрованим користувачам завантажувати документи, переглядати їх, редагувати, видаляти, а також надсилати на електронну пошту.

До основних типів вхідних даних належать:

- **файли документів**, які завантажуються користувачем (у форматах PDF, DOCX, TXT тощо);
- **файли зображень**, які завантажуються користувачем (у форматах png, jpg тощо);
- **дані користувача**, які зчитуються через сервіс аутентифікації (Clerk), включаючи email, ім’я та унікальний user ID;
- **електронна адреса одержувача**, яка вводиться користувачем для надсилання документа.

Вся інформація зберігається в базі даних у вигляді колекцій, що обробляються за допомогою хмарного сервісу **Convex**, який забезпечує реактивний підхід до роботи з даними.

Загальна структура системи включає наступні компоненти:

- **фронтенд (Next.js)** - інтерфейс користувача, реалізований за допомогою React-компонентів та бібліотеки TailwindCSS у поєднанні з UI-компонентами Shadcn/UI;
- **бекенд (Convex)** - відповідає за зберігання документів, виконання запитів на пошук, фільтрацію, додавання та видалення;
- **система аутентифікації (Clerk)** - забезпечує реєстрацію, авторизацію та керування сесіями користувачів;

– **служба поштової доставки (Nodemailer)** - використовується для надсилання файлів електронною поштою;

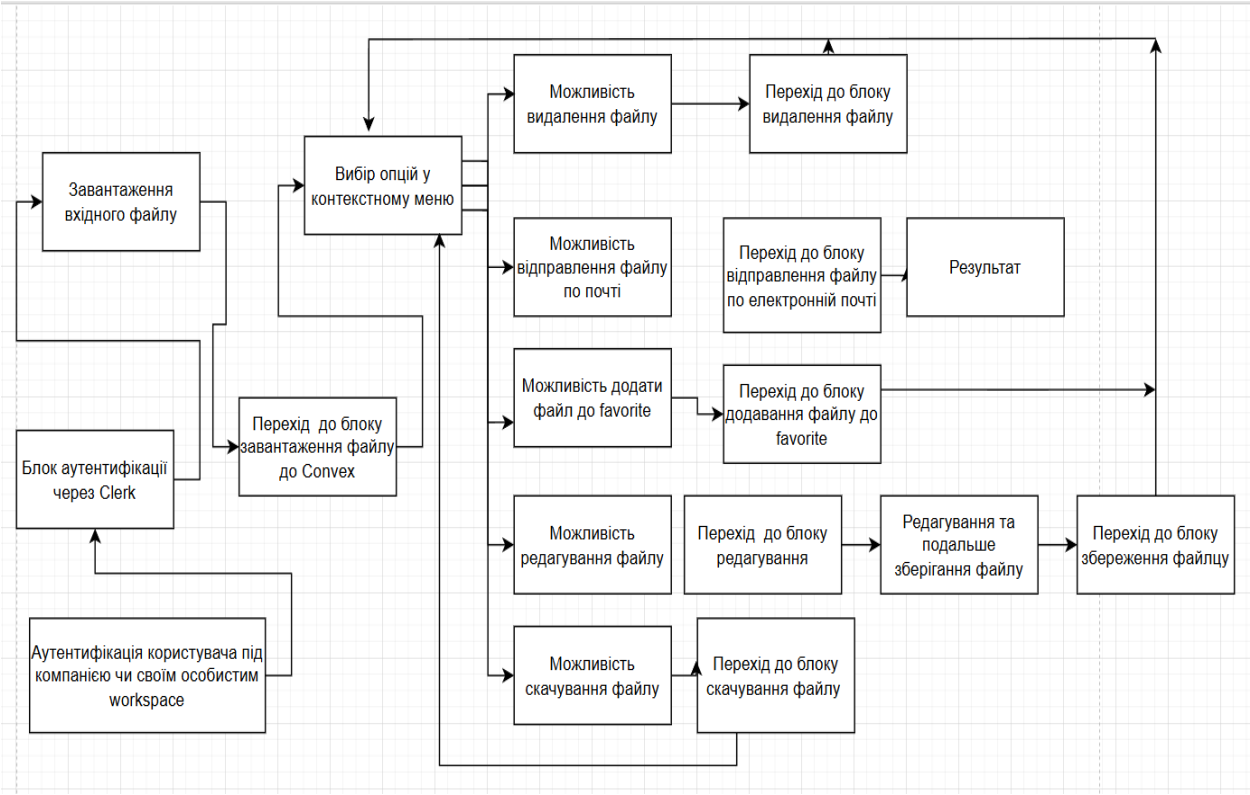

Рисунок 3.1 – Схема структури системи

3.2 Проектування/моделювання:

У процесі розробки інформаційної системи було здійснено проектування її інтерфейсу, внутрішньої структури даних, логіки обробки запитів, а також визначено технологічний стек, який забезпечує ефективну взаємодію між компонентами системи.

3.2.1 Інтерфейс користувача:

Інтерфейс платформи реалізований із використанням бібліотеки **Shadcn/UI** у поєднанні з **Tailwind CSS**, що дозволяє створювати адаптивні та функціональні компоненти без перевантаження стилями. Основні компоненти інтерфейсу:

– форма авторизації та реєстрації (Clerk);

- сторінка завантаження документів;
- таблиця з переліком завантажених файлів;
- кнопка "Надіслати" з діалоговим вікном для введення email-адреси;
- повідомлення про успішне збереження чи надсилання.

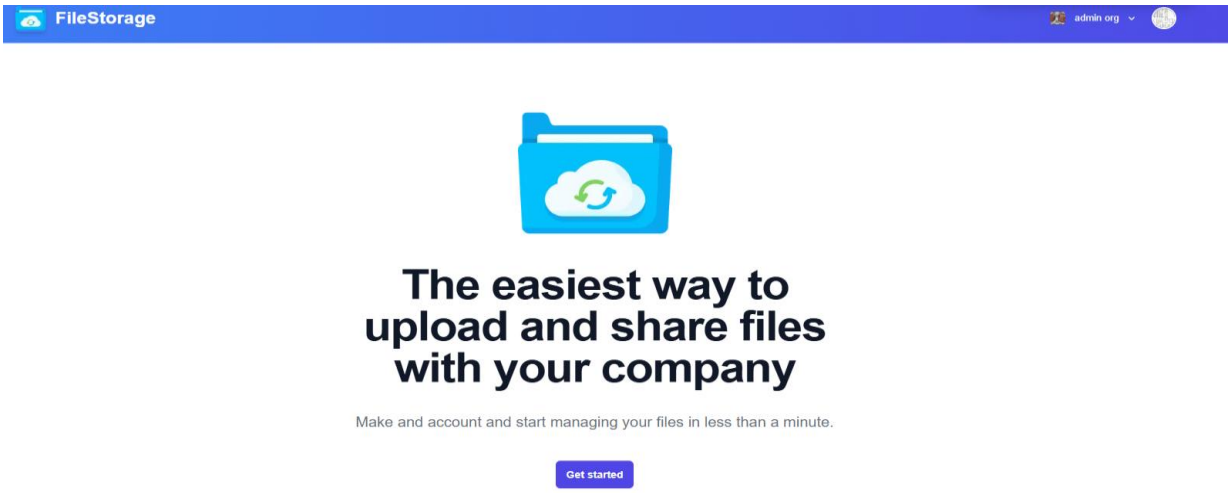

Рисунок 3.2 – Головна сторінка веб-застосунку

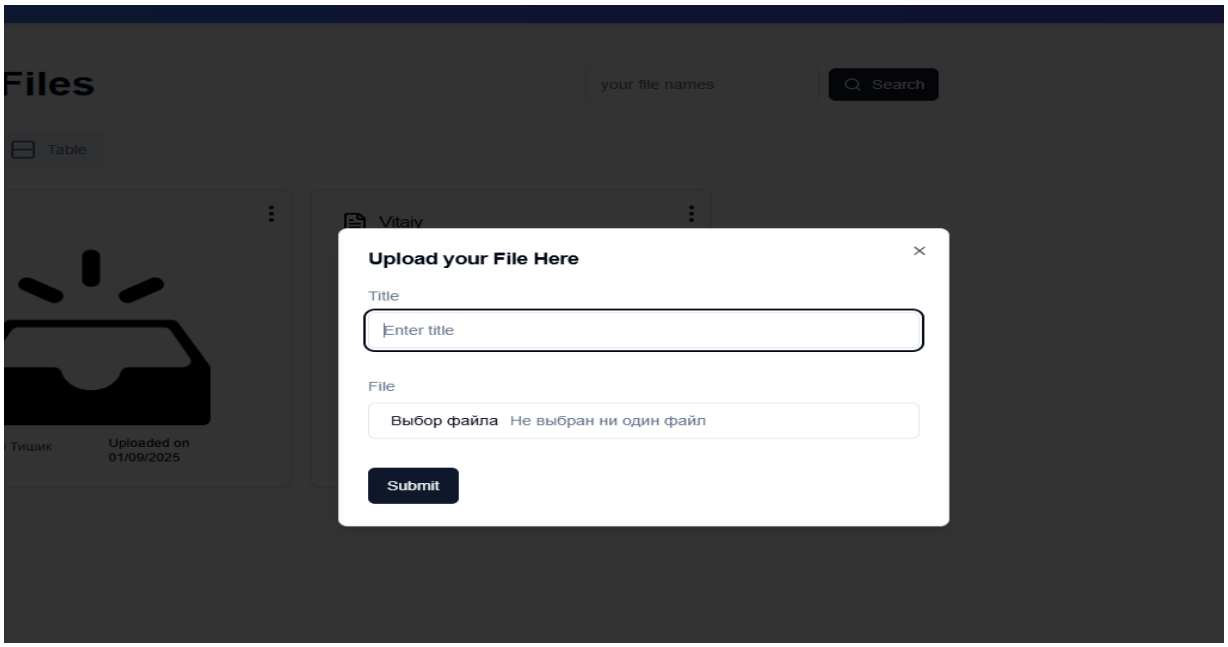

Рисунок 3.3 – Вікно завантаження файлу

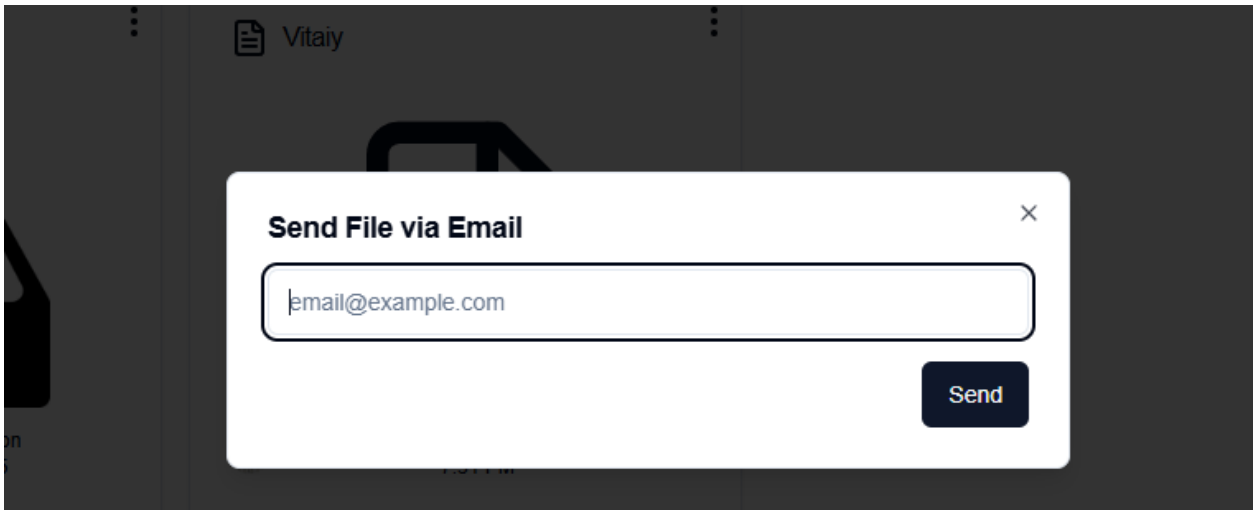

Рисунок 3.4 – Вікно відправлення файлу на електрону пошту

3.2.2 Структура реалізованих схеми у Convex:

Для зберігання даних у системі використовується реактивна база даних **Convex**, у якій було створено три основні колекції: files, favorites, users. Нижче подано опис їх структури:

Колекція Files

Ця таблиця зберігає інформацію про всі завантажені документи користувачами:

- **name** - назва файлу (тип: string);
- **orgId** - ідентифікатор організації, до якої належить користувач (тип: string);
- **shouldDelete** - прапорець, що вказує, чи слід видалити файл (тип: boolean, необов’язковий);
- **fileId** - ідентифікатор файлу у вбудованому сховищі файлів (_storage);
- **userId** - зовнішній ключ, що посилається на таблицю users;
- **type** - тип файлу (визначений у схемі як fileTypes — перелік допустимих типів).

Індекси:

- **by_orgId** - за організацією;

– **by_shouldDelete** - для швидкого фільтрування файлів, що підлягають видаленню;

– **by_fileId** - пошук файлу за його ідентифікатором.

Код схеми на ts:

```
    files: defineTable({
    name: v.string(),
    orgId: v. string(),
    shouldDelete: v.optional(v.boolean()),
    fileId: v.id("_storage"),
    userId: v.id("users"),
    type: fileTypes
    }).index("by_orgId",["orgId"]).index("by_shouldDelete"
,["shouldDelete"]).index("by_fileId", ["fileId"])
```

Колекція favorites:

Ця таблиця містить інформацію про улюблені файли користувачів:

– **fileId** - зовнішній ключ на файл у таблиці files;

– **orgId** - ідентифікатор організації;

– **userId** - зовнішній ключ на користувача у таблиці users.

Індекс:

– **by_userId_orgId_fileId** - дозволяє знайти обрані файли для конкретного користувача в межах певної організації.

Код схеми на ts:

```
    favorites: defineTable({
        fileId: v.id("files"),
      orgId: v.string(),
      userId: v.id("users")
    }).index("by_userId_orgId_fileId", ["userId", "orgId",
"fileId"]),
```

Колеція users:

Ця таблиця містить інформацію про зареєстрованих користувачів:

– **tokenIdentifier** - унікальний ідентифікатор користувача (надається системою Clerk);

– **name** - ім’я користувача (необов’язкове);

– **image** - URL зображення профілю (необов’язкове);

– **orgIds** - список об’єктів, кожен з яких містить: orgId та role.

Індекс:

– **by_tokenIdentifier** - пошук користувача за токеном, що видається Clerk.

Код схеми на ts:

```
    users: defineTable({
       tokenIdentifier: v.string(),
        name: v.optional(v.string()),
       image: v.optional(v.string()),
       orgIds: v.array( v.object({ orgId: v.string(),
   role: roles
    }) ) }).index("by_tokenIdentifier",
["tokenIdentifier"])
```

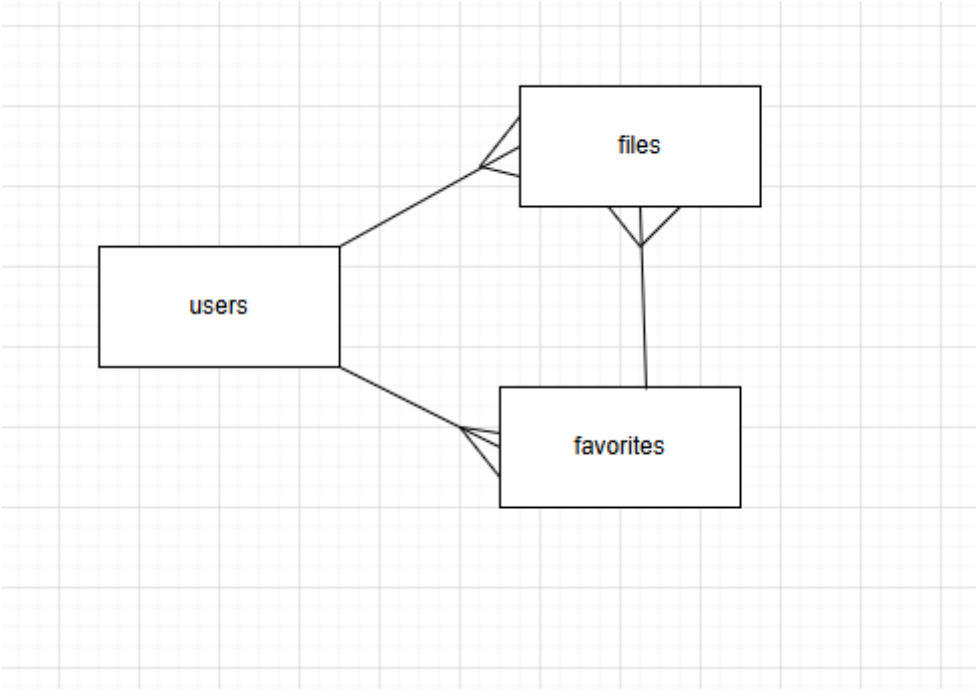

Рисунок 3.5 – ER-diagram спроектованих сутностей

3.2.3 Архітектура застосунку

Архітектура системи побудована за класичною моделлю клієнт-сервер. Згідно з цією моделлю, клієнт ініціює всі запити до сервера, який, у свою чергу,

виконує обробку цих запитів і повертає відповідь. Важливою особливістю такого підходу є те, що сервер не зберігає інформацію про стан клієнта між окремими запитами (принцип stateless). Це означає, що кожен запит від клієнта повинен містити всю необхідну інформацію для його обробки, і сервер не покладається на будь-яку збережену раніше інформацію про користувача або сесію [20].

Такий підхід має кілька суттєвих переваг:

- **масштабованість** - оскільки сервери не зберігають стан, можна легко додавати або видаляти серверні вузли без втрати даних або необхідності синхронізації стану;
- **гнучкість і надійність** — система стає більш відмовостійкою, бо кожен запит незалежний і обробляється окремо.

У нашій системі серверна частина відповідає за реалізацію бізнес-логіки додатку: обробку операцій із документами, управління доступом, фільтрацію, надсилання файлів тощо. Для зберігання даних використовується платформа Convex, яка виконує роль бази даних і бекенд-логіки одночасно, надаючи API для роботи з документами і користувачами.

Для реалізації функцій автентифікації та управління сесіями інтегровано сервіс Clerk, який забезпечує реєстрацію, вхід і безпечний доступ користувачів до системи.

Архітектура додатку є монолітною, тобто всі функціональні компоненти (інтерфейс, логіка, робота з даними) інтегровані в єдине ціле і тісно взаємодіють між собою.Монолітний підхід дозволяє швидше розробляти систему, особливо коли над проєктом працює невелика команда. Проте він має і недоліки: зростання обсягу коду та функціональності може ускладнити масштабування та підтримку додатку у майбутньому [21].

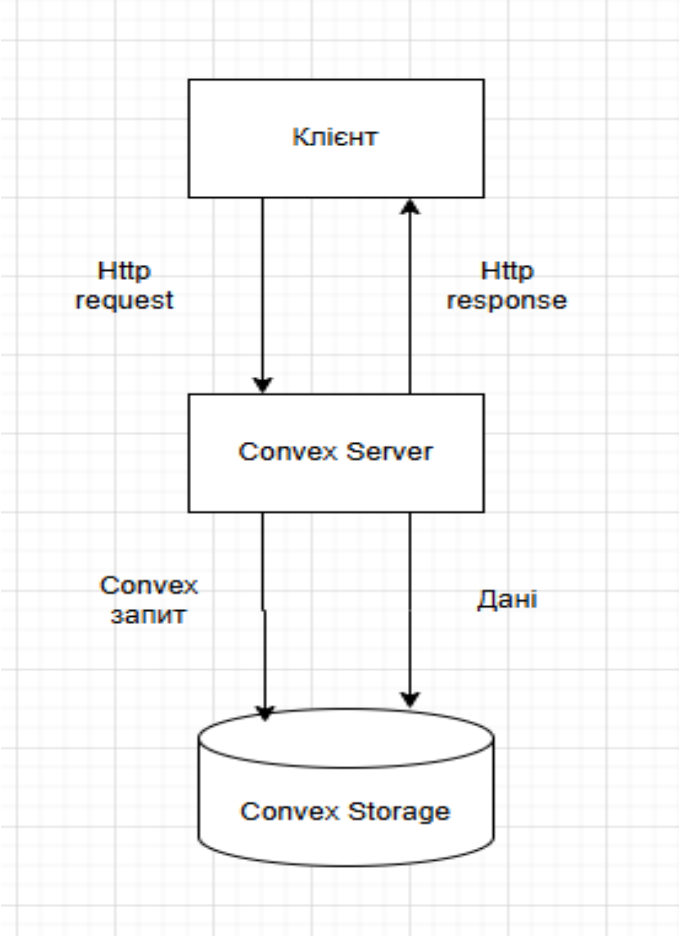

Рисунок 3.6 – Архітектура застосунку

3.2.4 Алгоритми роботи системи

У платформі реалізовано декілька ключових алгоритмів, які забезпечують основні функції: надсилання документів, фільтрацію та перевірку доступу.

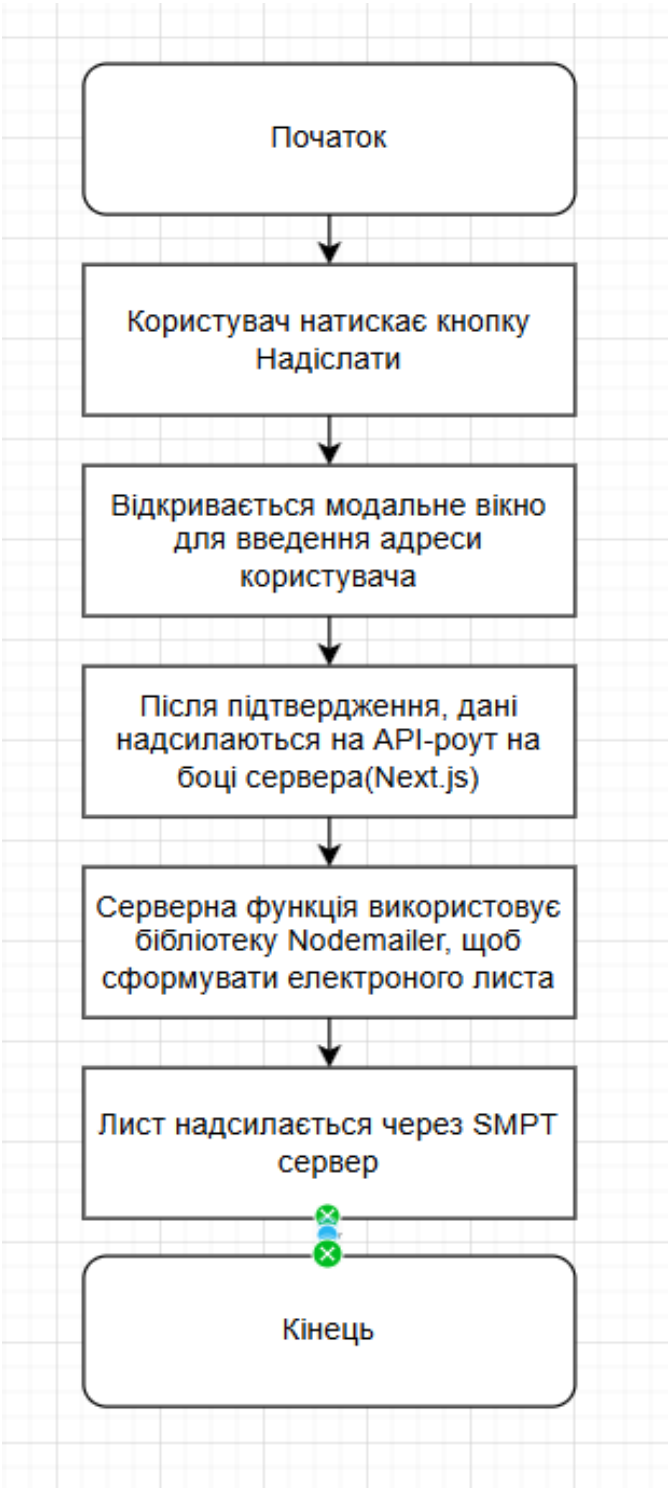

Рисунок 3.7 – Алгоритм надсилання документа на електрону пошту

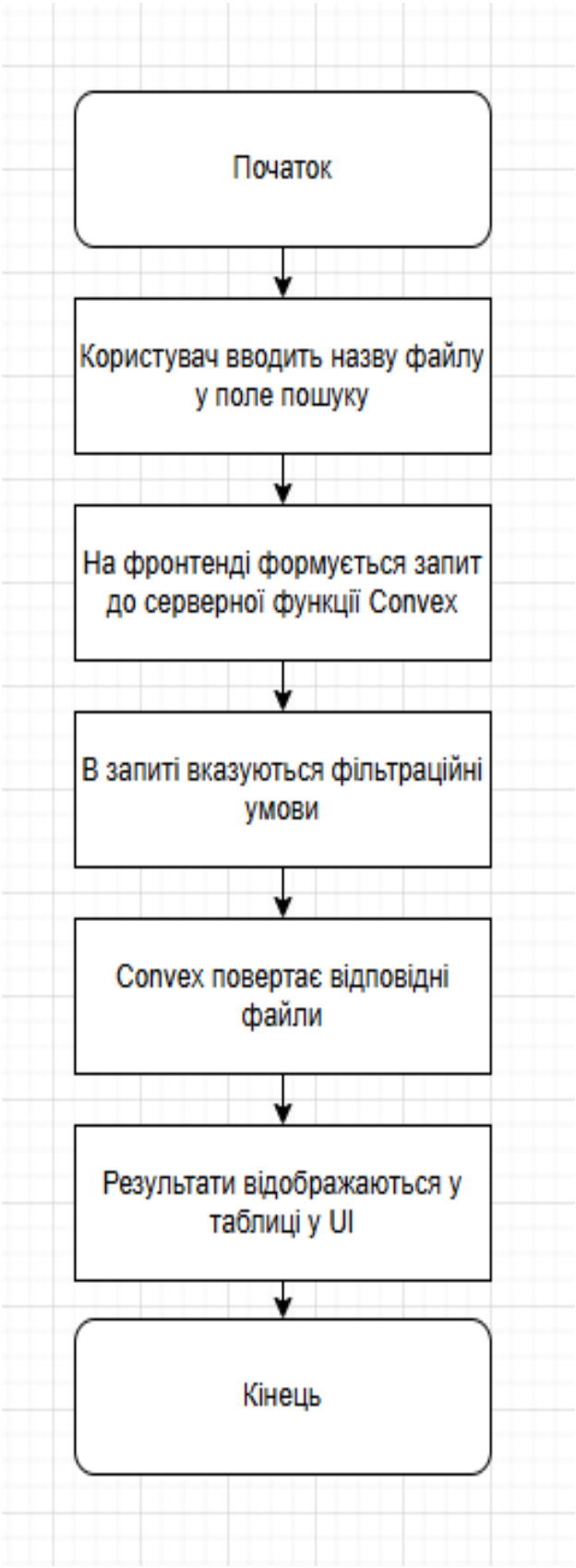

Рисунок 3.8 – Алгоритм пошуку та фільтрації документів

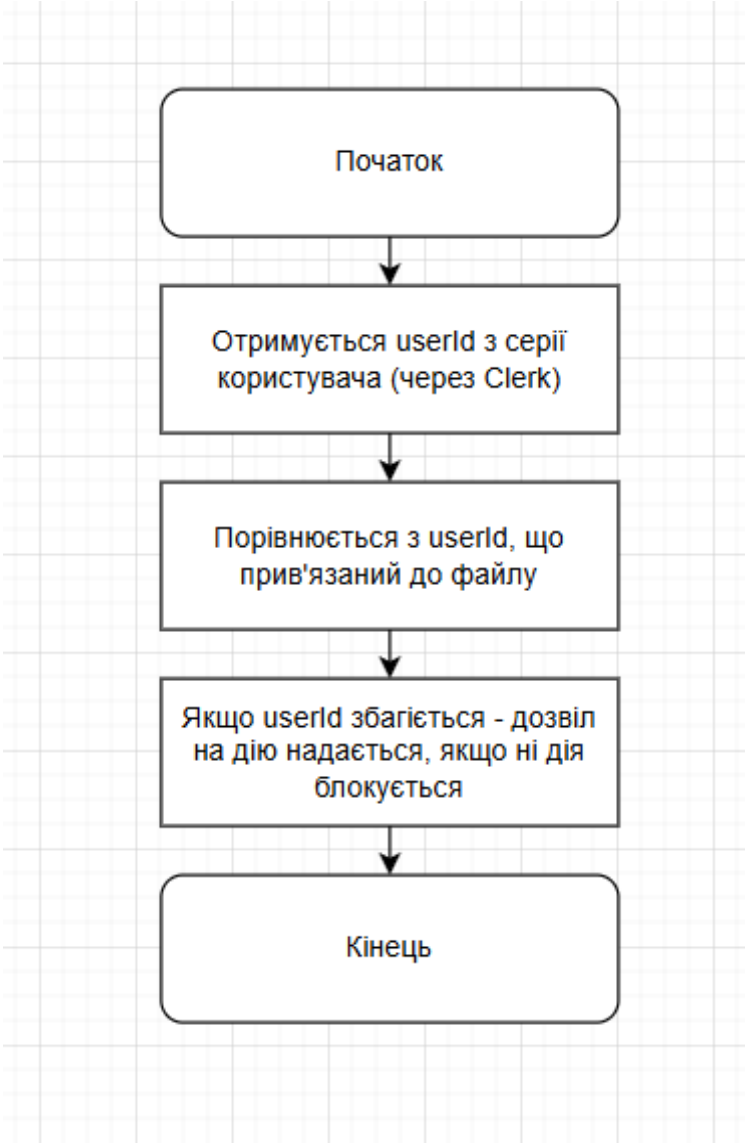

Рисунок 3.10 – Алгоритм перевірки прав доступу до документу

3.3. Аналіз отриманих результатів

У результаті розробки було створено інтегровану веб-платформу для збереження, перегляду, керування та надсилання електронних документів. Всі основні функції були реалізовані з використанням сучасного стеку технологій (Next.js, Convex, Clerk, Nodemailer тощо).

Реалізовано:

– авторизація користувача (реєстрація, вхід, сесії) — через Clerk;

– відображення файлів у таблиці з фільтрацією;
– перегляд, вибір та видалення файлів;
– надсилання обраного документа на email через Nodemailer;
– увесь інтерфейс реалізовано за допомогою Shadcn/UI + Tailwind CSS.

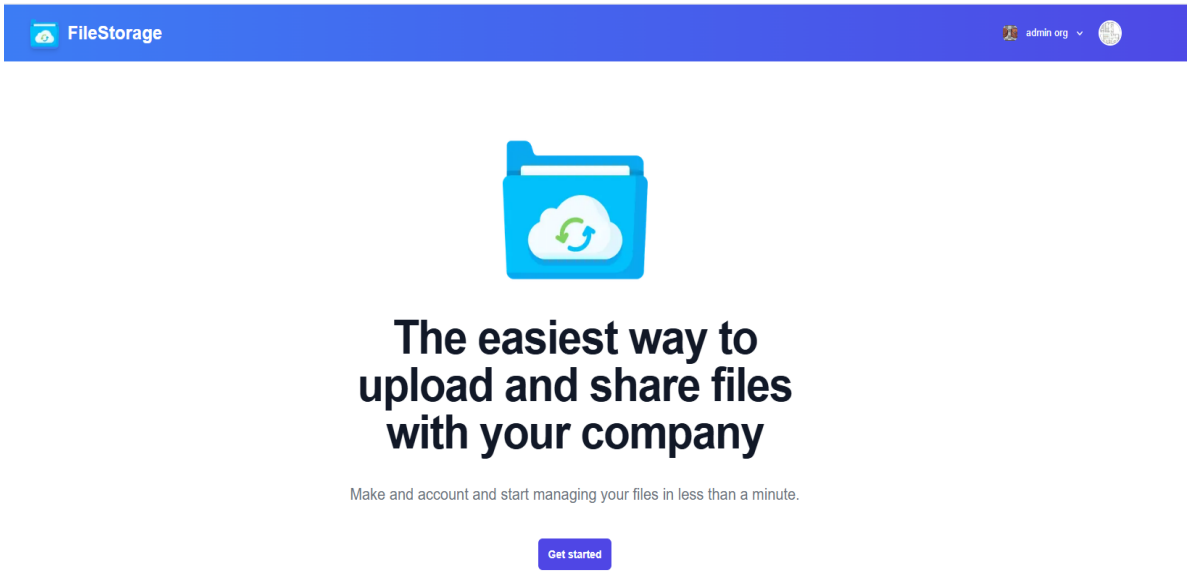

Рисунок 3.11 – Головна сторінка

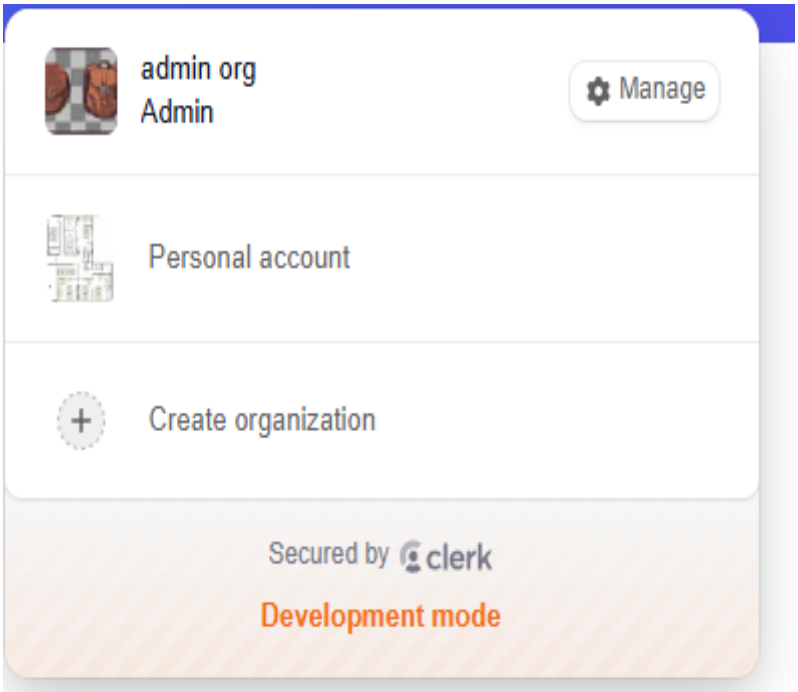

Рисунок 3.12 – Вибір компанії та workspace

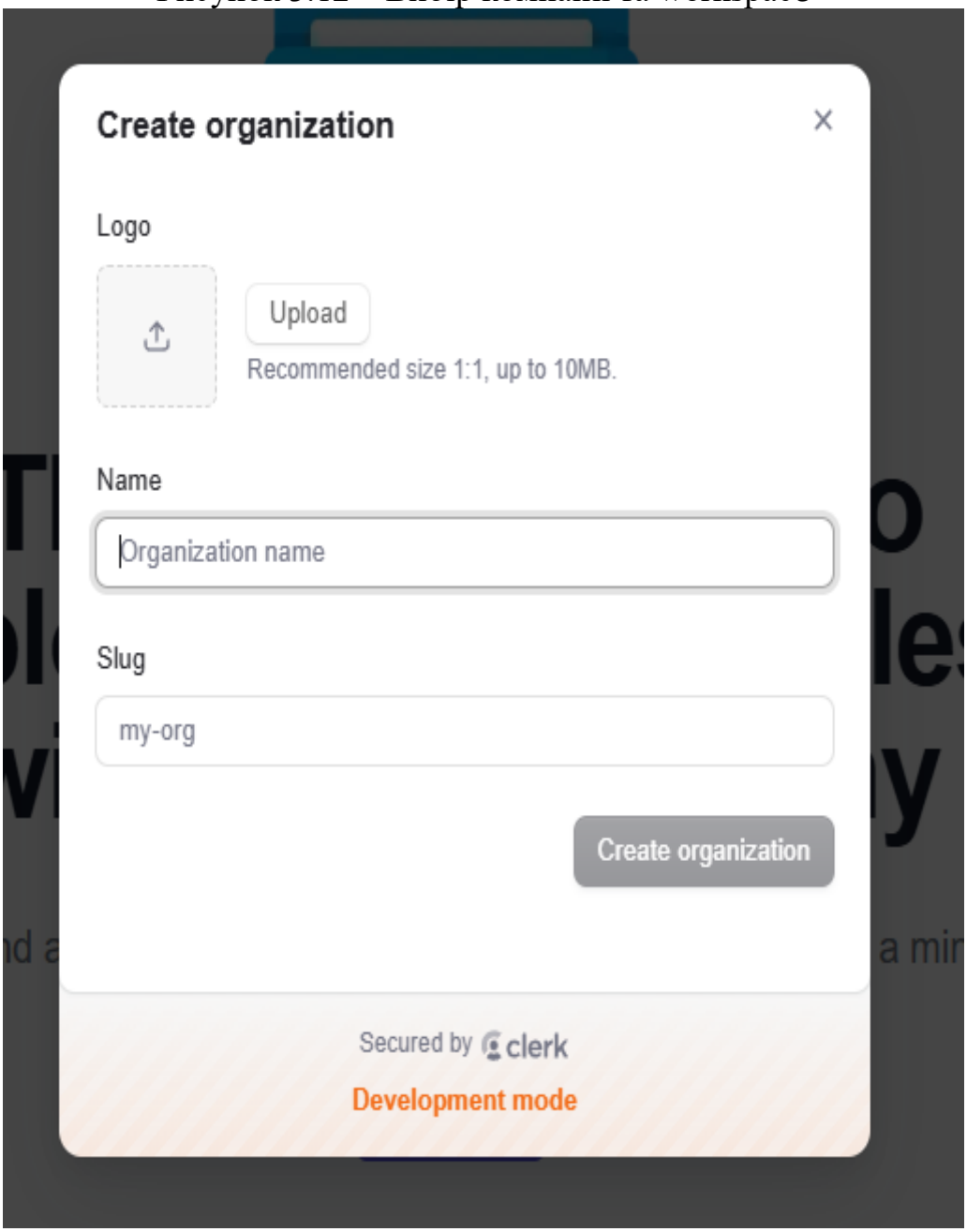

Рисунок 3.13 – Вікно створення компанії

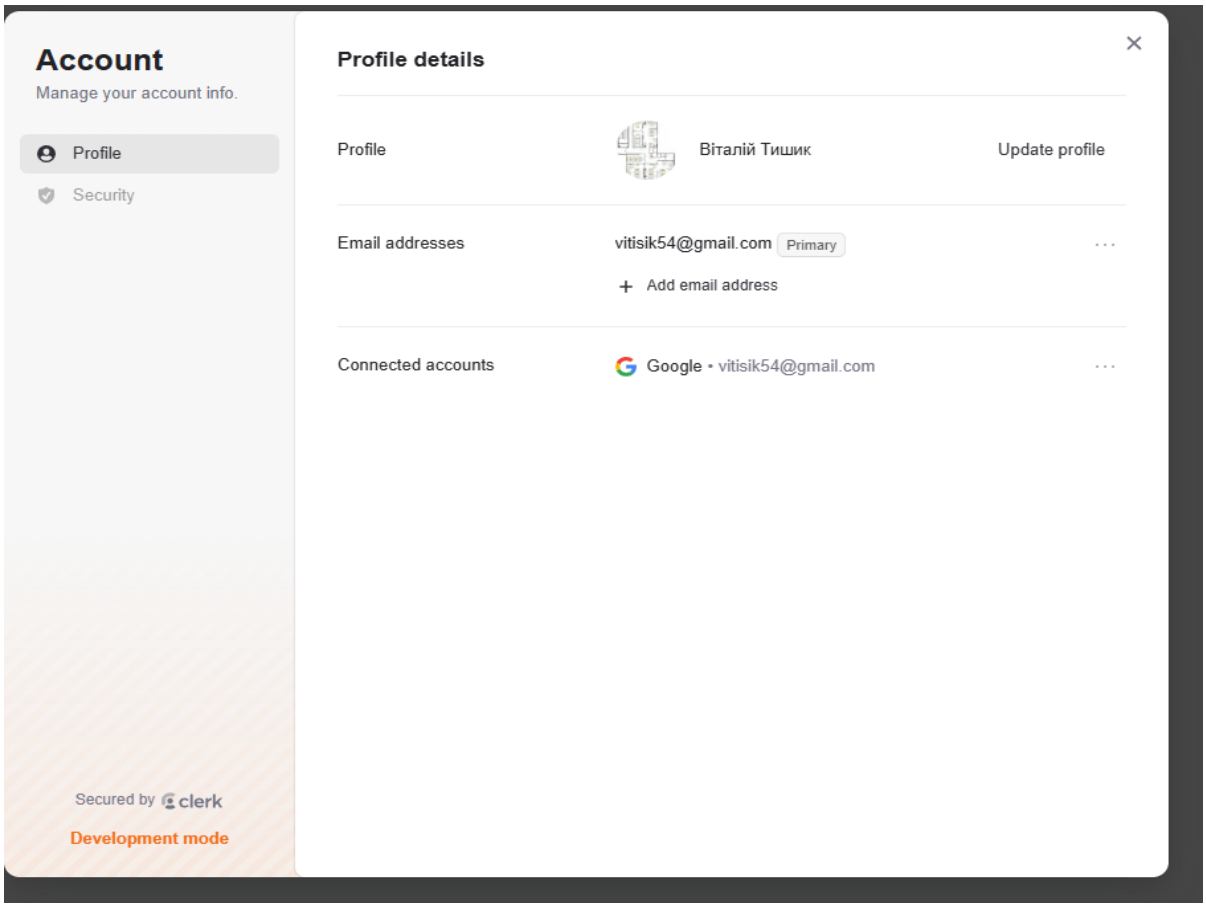

Рисунок 3.14 – Вікно юзера

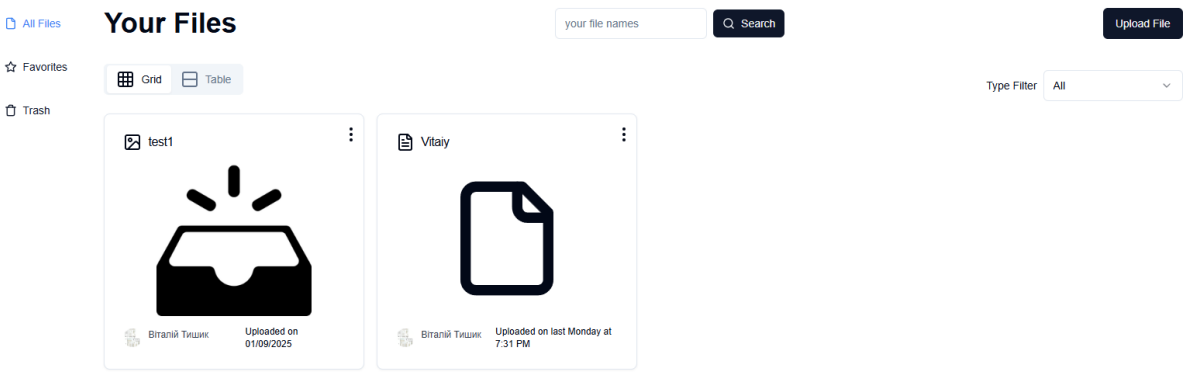

Рисунок 3.15 – Сторінка файлів

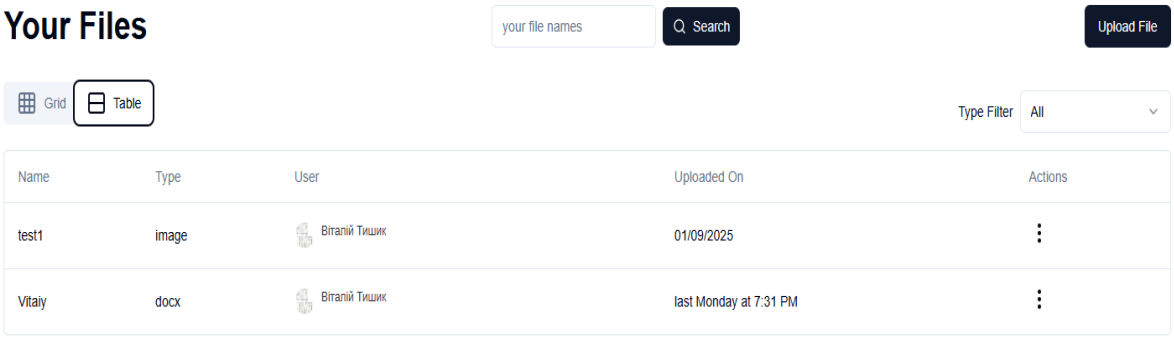

Рисунок 3.16 – Таблиця файлів

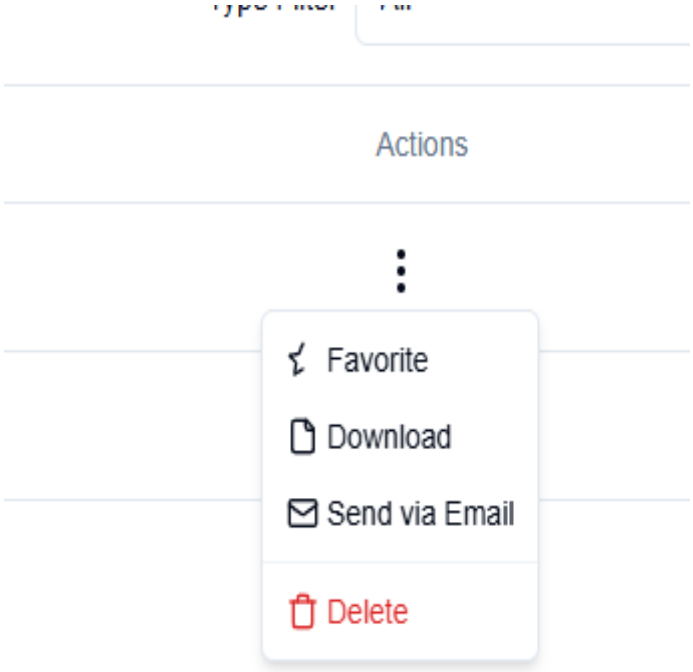

Рисунок 3.17 – Контекстне меню файла

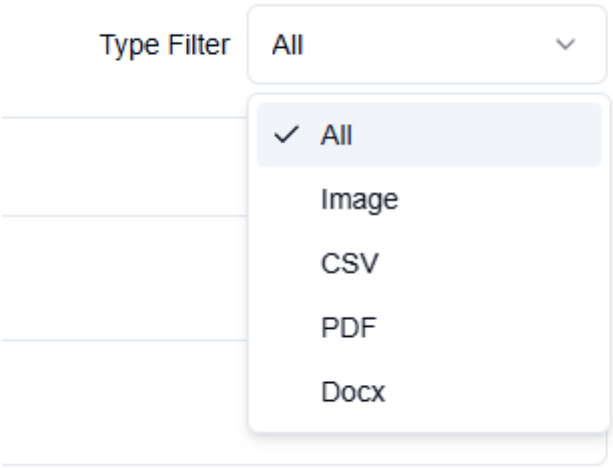

Рисунок 3.18 – Фільтр типу файлів

Рисунок 3.19 – Поле для пошуку файлів

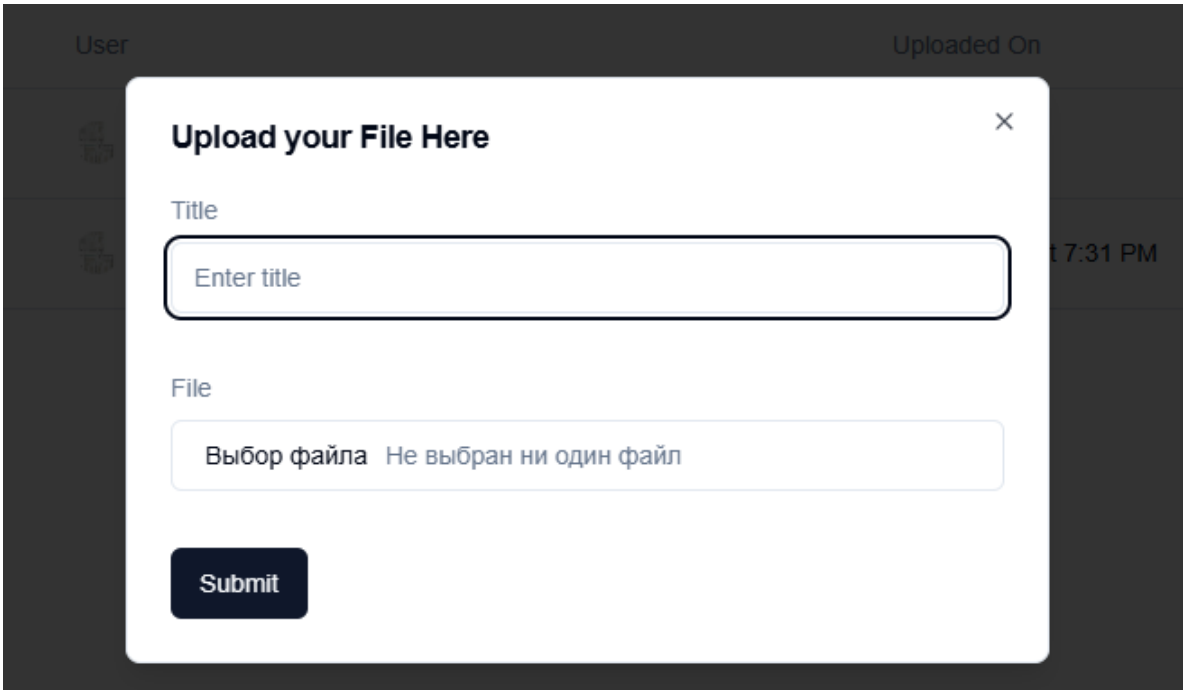

Рисунок 3.20 – Віно для завантаження файлу

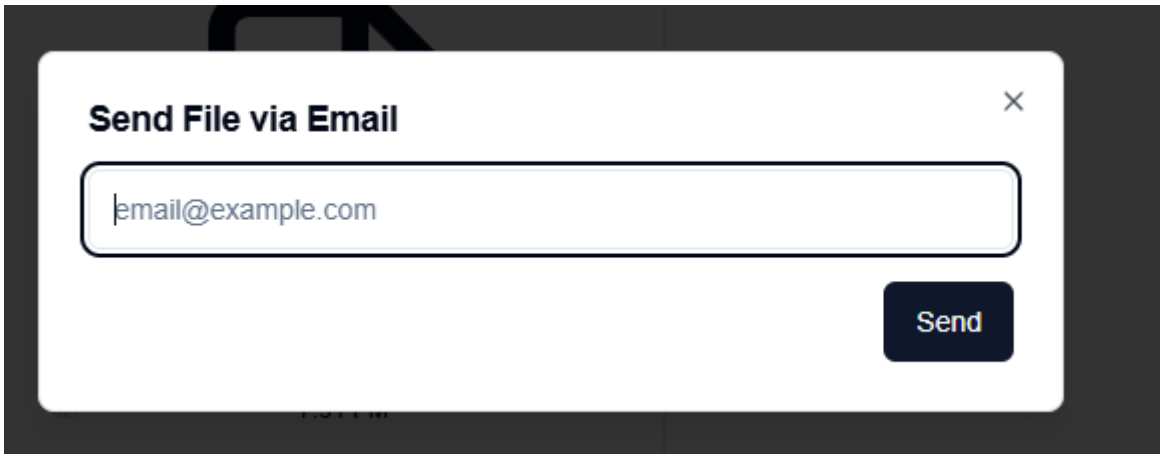

Рисунок 3.21 – Вікно для відправлення файлу на пошту

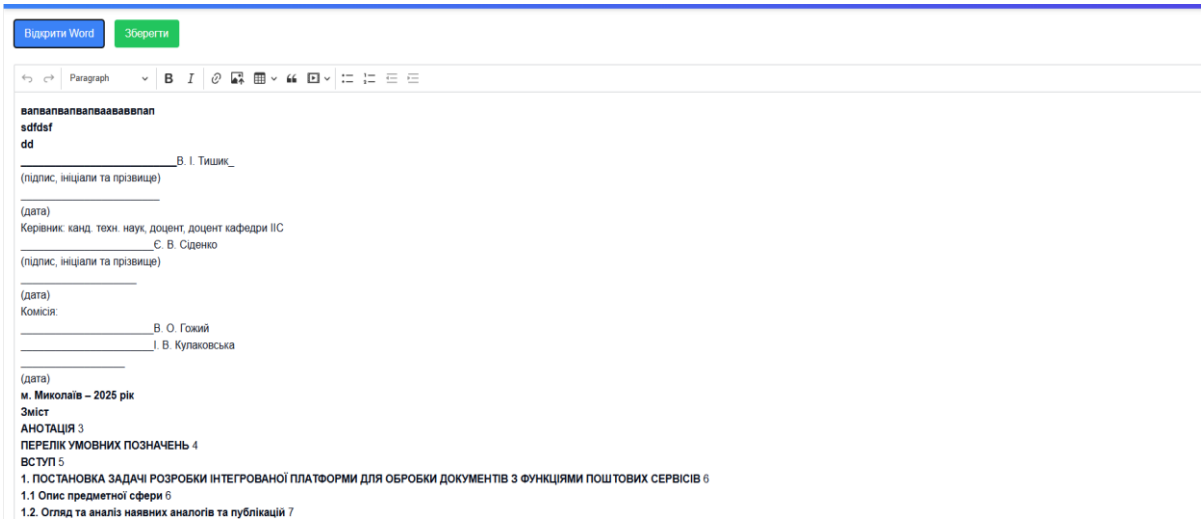

Рисунок 3.22 – Сторінка для редагування файлу

Висновки до розділу

У даному розділі було детально описано структуру та функціонування інтегрованої веб-платформи для обробки електронних документів із функціями поштового сервісу. Було розглянуто вхідні дані, архітектуру системи, реалізовані модулі та їхню взаємодію.

На етапі проєктування було створено інтерфейс користувача із використанням сучасних UI-бібліотек (Shadcn/UI, Tailwind CSS), спроєктовано структуру бази даних у Convex та реалізовано ключову бізнес-логіку. Система

підтримує авторизацію через сервіс Clerk, зберігання файлів у хмарному сховищі, а також надсилання документів на електронну пошту за допомогою Nodemailer.

Таким чином, поставлені завдання були повністю виконані, що підтверджує ефективність обраної архітектури та технологій. Система може бути основою для подальшого розширення функціоналу або впровадження у реальних умовах.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

4.1 Опис програмної реалізації

4.1.1 Сторінка авторизації (Sign In/Sign Up)

Для реалізації автентифікації та управління сесією користувачів у системі було використано сервіс **Clerk**. Цей сервіс надає готові компоненти для входу, реєстрації, керування акаунтом та сесіями. Його інтеграція з фреймворком **Next.js** забезпечує швидку реалізацію без необхідності розробляти власну систему авторизації.

Після налаштування Clerk у проєкті, до маршруту /sign-in було підключено компонент **<SignIn />**, який відображає форму входу користувача. Якщо користувач ще не має облікового запису, він може перейти до реєстрації через інтерфейс Clerk.

Також реалізована перевірка сесії користувача: якщо користувач не авторизований, його автоматично перенаправляє на сторінку входу.

Приклад коду сторінки входу (app/sign-in/[[...sign-in]]/page.tsx):

```
    import { SignIn } from "@clerk/nextjs";

    export default function Page() {
      return (
        <div className="flex justify-center items-center
h-screen">
          <SignIn />
        </div>
      );
    }
```

Перевірка авторизованого доступу:

Для обмеження доступу до внутрішніх сторінок застосовано HOC auth, або перевірку useAuth():

4.1.2 Домашня сторінка (Home)

Після авторизації користувач автоматично перенаправляється на домашню сторінку, яка виконує роль основного робочого середовища. Тут реалізовано відображення списку файлів, що належать поточному користувачу, та можливість створення нового файлу.

Для збереження та отримання документів використовується хмарна база даних **Convex**, яка інтегрується з Next.js через SDK. Кожен документ містить такі поля, як title, content, userId, createdAt, updatedAt.

Для взаємодії з базою Convex використано хуки useQuery для отримання документів та useMutation для створення нового.

Основний функціонал домашньої сторінки:

– виведення списку існуючих документів;

– створення нового документа;

– перехід до сторінки редагування при кліку на документ.

Приклад коду домашньої сторінки:

```
   "use client";
   import {  useOrganization, useUser } from
"@clerk/nextjs";
   import { useQuery } from "convex/react";;
   import Image from "next/image";
   import { GridIcon, Loader2, RowsIcon, TableIcon} from
"lucide-react";
```

```
   import { useState } from "react";
```

```
    import { api } from
"../../../../convex/_generated/api";
    import { SearchBar } from
"@/app/dashboard/_components/search-bar";
    import { UploadButton } from "./UploadButton";
    import { FileCard } from "./file-card";
    import { DataTable } from "./file-table";
    import { columns } from "./columns";
    import { Tabs, TabsContent, TabsList, TabsTrigger }
from "@/components/ui/tabs";
    import { Select, SelectContent, SelectItem,
SelectTrigger, SelectValue } from "@/components/ui/select";
    import { Doc } from
"../../../../convex/_generated/dataModel";
    import { Label } from "@/components/ui/label";

    function Placeholder({
      favorites,
      deletedOnly
    }:{
      favorites?: boolean,
      deletedOnly?: boolean
    }){
      return(<>
        <div className="flex flex-col gap-8 w-full items-
center mt-24">
          <Image
        alt="an image of a picture and directory icon"
            width="300"
            height="300"
            src="/empty.png"
          ></Image>
          <div className="text-2xl">{favorites ? "You have
no favorites files" : "You have no files, upload one
now"}</div>
          {favorites || deletedOnly ? <></> : <UploadButton
/>}
        </div>
      </>)
    }
```

```
    export function FilesBrowser({
      title,
      favorites,
      deletedOnly
    }: {
      title: string,
      favorites?: boolean,
      deletedOnly?: boolean
    }) {
      const organization = useOrganization();
      const user = useUser();
      const [query, setQuery] = useState("");
      const [type, setType] = useState<Doc<"files">["type"]
| "all">('all');

      let orgId: string | undefined = undefined;

      if(organization.isLoaded && user.isLoaded){
        orgId = organization.organization?.id ??
user.user?.id;
      }

      const favoritesFiles = useQuery(
        api.files.getAllFavorites,
        orgId ? {orgId} : "skip"
      );

      const files = useQuery(
        api.files.getFiles,
        orgId ? {orgId, type: type === 'all' ? undefined :
type, query, favorites, deletedOnly} : "skip"
      );

      const isLoading = files === undefined;

      const modifiedFiles = files?.map(file => ({
        ...file,
        isFavorited: (favoritesFiles ?? []).some(
          (favorite) => favorite.fileId === file._id
        )
      })) ?? [];
```

```
    return (
        <div className="w-full">
          <div className="flex justify-between items-center
mb-8">
            <h1 className="text-4xl font-bold">{title}</h1>

            <SearchBar query={query} setQuery={setQuery}/>

            <UploadButton />
          </div>

          <Tabs
            defaultValue="grid"
          >
            <div className="flex justify-between items-
center">
              <TabsList className="mb-4">
                <TabsTrigger value="grid" className="flex
gap-2 items-center">
                  <GridIcon />
                  Grid
                </TabsTrigger>
                <TabsTrigger value="table" className="flex
gap-2 items-center">
                  <RowsIcon />
                  Table
                </TabsTrigger>
              </TabsList>

              <div className="flex gap-2 items-center">
                <Label htmlFor="type-select">Type
Filter</Label>
                <Select
                  value={type}
                  onValueChange={(newType) => {
                    setType(newType as any)
                  }}
                >
                  <SelectTrigger id="type-select"
className="w-[180px]">
                    <SelectValue />
```

```
                  </SelectTrigger>
                  <SelectContent>
                    <SelectItem
value="all">All</SelectItem>
                    <SelectItem
value="image">Image</SelectItem>
                    <SelectItem
value="csv">CSV</SelectItem>
                    <SelectItem
value="pdf">PDF</SelectItem>
                    <SelectItem
value="docx">Docx</SelectItem>
                  </SelectContent>
                </Select>
              </div>
            </div>

            {isLoading && (
              <div className="flex flex-col gap-8 w-full
items-center mt-24">
                <Loader2 className="h-32 w-32 animate-spin
text-gray-500"></Loader2>
                <div className="text-2xl">Loading your
files...</div>
              </div>
            )}

            <TabsContent value="grid">
              <div className="grid grid-cols-4 gap-4">
                {modifiedFiles?.map((file) => {
                  return <FileCard key={file._id}
file={file} />
                })}
              </div>
            </TabsContent>
            <TabsContent value="table">
              <DataTable columns={columns}
data={modifiedFiles}/>
            </TabsContent>
          </Tabs>

          {files?.length === 0 && (
            <Placeholder favorites={favorites}
```

```
deletedOnly={deletedOnly}/>
          )}
        </div>
      );
```

Взаємодія з базою Convex:

```
    import {action, internalMutation, mutation,
MutationCtx, query, QueryCtx} from './_generated/server'
    import {ConvexError, v} from 'convex/values'
    import { fileTypes } from './schema';
    import { Doc, Id } from './_generated/dataModel';
    import { api } from './_generated/api';

    export const generateUploadUrl = mutation(async (ctx)
=> {
        const identity = await ctx.auth.getUserIdentity();

        if(!identity){
            throw new ConvexError('you must be logged in to
upload a file');
        }

        return await ctx.storage.generateUploadUrl();
    })

    export const getUrl = mutation({
        args: {
            fileId: v.id("_storage")
        },
        async handler(ctx, args){
            const indentity = await
ctx.auth.getUserIdentity();

            if(!indentity){
                throw new ConvexError('you must be logged
in to upload a file');
            }

            const fileUrl = await
ctx.storage.getUrl(args.fileId);
```

```
            return fileUrl;
        }
    })

    async function hasAccessToOrg(
        ctx: QueryCtx | MutationCtx,
        orgId: string
    ){
        const indentity = await ctx.auth.getUserIdentity();

        if(!indentity){
           return null;
        }

        const user = await await ctx.db
            .query("users")
            .withIndex("by_tokenIdentifier", (q) =>
                q.eq("tokenIdentifier",
indentity.tokenIdentifier)
            )
            .first();

        if(!user){
            return null;
        }

        const hasAccess = user.orgIds.some((item) =>
item.orgId === orgId) ||
            user.tokenIdentifier.includes(orgId);

        if(!hasAccess){
            return null;
        }

        return {user};
    }

    export const createFile = mutation({
        args: {
            name: v.string(),
            fileId: v.id("_storage"),
            orgId: v.string(),
            type: fileTypes,
```

```
        },
        async handler(ctx, args){
            const hasAccess = await hasAccessToOrg(
                ctx,
                args.orgId,
            );

            if(!hasAccess){
                throw new ConvexError('you do not have
access to this org')
            }

            await ctx.db.insert('files', {
                name: args.name,
                orgId: args.orgId,
                fileId: args.fileId,
                type: args.type,
                userId: hasAccess.user._id,
            })
        }
    })

    export const getFiles = query({
        args: {
            orgId: v.string(),
            query: v.optional(v.string()),
            favorites: v.optional(v.boolean()),
            deletedOnly: v.optional(v.boolean()),
            type: v.optional((fileTypes)),
        },
        async handler(ctx, args){
            const access = await hasAccessToOrg(
                ctx,
                args.orgId,
            );

            if(!access){
                throw [];
            }

            let files = await ctx.db.
                query("files")
                .withIndex("by_orgId", (q) => q.eq("orgId",
```

```
args.orgId))
                .collect();

            const query = args.query;

            if(args.favorites){
                const favorites = await ctx.db
                    .query("favorites")
                    .withIndex("by_userId_orgId_fileId",
(q) =>
                        q.eq("userId",
access.user._id).eq("orgId", args.orgId)
                    )
                    .collect();

                files = files.filter((file) =>
                    favorites.some((favorite) =>
favorite.fileId === file._id)
                )
            }

            if(args.deletedOnly){
                files = files.filter((file) =>
file.shouldDelete);
            } else if(!args.favorites){
                files = files.filter((file) =>
!file.shouldDelete);
            }

            if(query){
                files = files.filter((file) =>
file.name.includes (query))
            }

            if(args.type){
                files = files.filter((file) => file.type
=== args.type);
            }

            return files
        }
    })
```

```
    export const getFileById = query({
        args: {
            orgId: v.string(),
            fileId: v.string(),
        },
        async handler(ctx, args) {
            const access = await hasAccessToOrg(
                ctx,
                args.orgId,
            );

            if (!access) {
                throw [];
            }

            let files = await ctx.db.
                query("files")
                .withIndex("by_orgId", (q) => q.eq("orgId",
args.orgId))
                .collect();

            const file = files.find((f) => f._id ===
args.fileId);

            if (!file) {
                throw new Error('Файл не найден');
            }

            return file;
        }
    });

    export const deletedAllFiles = internalMutation({
        args: {},
        async handler(ctx){
            const files = await ctx.db.query("files")
                .withIndex("by_shouldDelete", (q) =>
q.eq("shouldDelete", true))
                .collect();

            await Promise.all(files.map(async (file) => {
                await ctx.storage.delete(file.fileId);
                return await ctx.db.delete(file._id);
```

```
            }))
        }
    })

    export const deleteFile = mutation({
        args: { fileId: v.id('files') },
        async handler(ctx, args){
            const access = await hasAccessToFile(ctx,
args.fileId);

            if(!access){
                throw new ConvexError('no access to file');
            }

            canDeleteFile(access.user, access.file);

            await ctx.db.patch(args.fileId, {
                    shouldDelete: true
            })
        }
    })

    export const restoreFile = mutation({
        args: { fileId: v.id('files') },
        async handler(ctx, args){
            const access = await hasAccessToFile(ctx,
args.fileId);

            if(!access){
                throw new ConvexError('no access to file');
            }

            canDeleteFile(access.user, access.file);

            await ctx.db.patch(args.fileId, {
                    shouldDelete: false
            })
        }
    })

    export const toggleFavorite = mutation({
        args: { fileId: v.id('files') },
        async handler(ctx, args){
```

```
            const access = await hasAccessToFile(ctx,
args.fileId);

            if(!access){
                throw new ConvexError('no access to file');
            }

            const favourite = await
ctx.db.query("favorites")

    .withIndex("by_userId_orgId_fileId", q =>
                    q.eq('userId',
access.user._id).eq("orgId",
access.file.orgId).eq("fileId", access.file._id)
                )
                .first();

            if(!favourite){
                await ctx.db.insert("favorites", {
                    fileId: access.file._id,
                    userId: access.user._id,
                    orgId: access.file.orgId
                });
            } else{
                await ctx.db.delete(favourite._id);
            }
        }
    })

    export const getAllFavorites = query({
        args: { orgId: v.string() },
        async handler(ctx, args){
            const access = await hasAccessToOrg(ctx,
args.orgId);

            if(!access){
                return [];
            }

            const favourite = await
ctx.db.query("favorites")
                .withIndex("by_userId_orgId_fileId", q =>
                    q.eq('userId',
```

```
access.user._id).eq("orgId", args.orgId)
                )
                .collect();

            return favourite;
        }
    })

    async function hasAccessToFile(ctx: QueryCtx |
MutationCtx, fileId: Id<"files">){
        const file = await ctx.db.get(fileId);

        if(!file){
            return null;
        }

        const hasAccess = await hasAccessToOrg(
            ctx,
            file.orgId
        );

        if(!hasAccess){
           return null;
        }

        return {user: hasAccess.user, file}
    }

    function canDeleteFile(user: Doc<"users">, file:
Doc<"files">){
        const canDelete =
                file.userId === user._id ||
                user.orgIds.find((org) => org.orgId ===
file.orgId)
                ?.role === 'admin';

        if(!canDelete){
            throw new ConvexError("you have no access to
delete this file");
        }
    }

    export const getFileMeta = query({
```

```
        args: {
          fileId: v.id('files'),
        },
        handler: async (ctx, { fileId }) => {
          return await ctx.db.get(fileId);
        },
    });

    export const saveFile = mutation({
        args: { fileId: v.id('files') },
        async handler(ctx, args){
            const access = await hasAccessToFile(ctx,
args.fileId);

            if(!access){
                throw new ConvexError('no access to file');
            }

            await ctx.db.patch(args.fileId, {
                shouldDelete: true
            })
        }
    })

    export const updateFileContent = mutation({
      args: {
        fileId: v.id("_storage"),
        newStorageId: v.id("_storage"),
      },
      async handler(ctx, args) {
        const file = await ctx.db
          .query("files")
          .withIndex("by_fileId", (q) => q.eq("fileId",
args.fileId))
          .first();

        if (!file) {
          throw new ConvexError("File not found");
        }

        await ctx.db.patch(file._id, {
          fileId: args.newStorageId,
        });
```

```
    },
});
```

4.1.3 Сторінка редагування документа

На цій сторінці користувач може редагувати текстовий документ за допомогою **CKEditor**, зберігати зміни, а також надіслати документ на електронну пошту. Редагування відбувається у візуальному форматі (WYSIWYG).

Приклад коду сторінки редагування:

```
    'use client';

    import { useRef, useState } from 'react';
    import { CKEditor } from '@ckeditor/ckeditor5-react';
    import ClassicEditor from '@ckeditor/ckeditor5-build-
classic';
    import mammoth from 'mammoth';
    import { api } from
'../../../../convex/_generated/api';
    import { useMutation } from 'convex/react';
    import { Doc } from
'../../../../convex/_generated/dataModel';
    import { useToast } from '@/hooks/use-toast';
    import { useRouter } from 'next/navigation';

    const HtmlDocx: any = require('html-docx-js/dist/html-
docx');

    export default function WordEditor({
        file,
    }: {
        file: Doc<"files">
    }) {
      const [editorData, setEditorData] =
useState<string>('');
      const editorRef = useRef<any>(null);

      const [docxBuffer, setDocxBuffer] =
useState<Uint8Array | null>(null);
```

```
      const router = useRouter();

      const { toast } = useToast();

      const getFileUrl = useMutation(api.files.getUrl);
      const updateFile =
useMutation(api.files.updateFileContent);
      const generateUploadUrl =
useMutation(api.files.generateUploadUrl);

      const handleOpenDocx = async () => {
        try {
          const url = await getFileUrl({ fileId:
file.fileId });
          if (!url) return;

          const response = await fetch(url);
          const arrayBuffer = await response.arrayBuffer();

          try {
            const uint8 = new Uint8Array(arrayBuffer);
            const { value } = await mammoth.convertToHtml({
arrayBuffer: uint8.buffer });

            if (value && value.trim() !== "") {
              setEditorData(value);
              setDocxBuffer(uint8);
            } else {
              throw new Error("DOCX content empty, fallback
to raw HTML");
            }
          } catch (e) {
            console.warn("Не удалось распарсить как DOCX,
пробуем как HTML", e);

            const cleanedHtml =
cleanUpGarbage(arrayBuffer);
            setEditorData(cleanedHtml);
          }
        } catch (error) {
          console.error("Ошибка загрузки файла:", error);
        }
      };
```

```
     function cleanUpGarbage(arrayBuffer: ArrayBuffer):
string {
       const decoder = new TextDecoder("utf-8");

       const byteArray = new Uint8Array(arrayBuffer);

       const numberOfBytesToRemove = 3375;

       const cleanedArrayBuffer =
byteArray.slice(numberOfBytesToRemove);

       return decoder.decode(cleanedArrayBuffer);
     }

     const handleSaveDocx = async () => {
       const htmlContent = editorRef.current?.getData();

       if (!htmlContent) return;

       const converted = HtmlDocx.asBlob(htmlContent);

       const uploadUrl = await generateUploadUrl();

       await fetch(uploadUrl, {
         method: "POST",
         headers: {
           "Content-Type":
"application/vnd.openxmlformats-
officedocument.wordprocessingml.document"
         },
         body: converted,
       }).then(async res => {
         const { storageId } = await res.json();

         await updateFile({
           fileId: file.fileId,
           newStorageId: storageId,
         });

         toast({
           variant: "success",
           title: "Document save",
```

```
            description: "Документ успешно сохранён."
          });
            router.push(`/dashboard/files`);
        }).catch(() => {
          toast({
            variant: "destructive",
            title: "Document don't save",
            description: "Ошибка при сохранении."
          });
          toast({ title: "Ошибка при сохранении", variant:
"destructive" });
        });
      }

      return (
        <div className="p-4">
          <button
            onClick={handleOpenDocx}
            className="bg-blue-500 text-white px-4 py-2
rounded mr-4"
          >
            Відкрити Word
          </button>
          <button
            className="bg-green-500 text-white px-4 py-2
rounded"
            onClick={handleSaveDocx}
          >
            Зберегти
          </button>
          <div className="mt-6">
            <CKEditor
              editor={ClassicEditor as any}
              data={editorData}
              onReady={(editor) => (editorRef.current =
editor)}
              onChange={(event, editor) =>
setEditorData(editor.getData())}
            />
          </div>
        </div>
      );
    }
```

4.2 Керівництво користувача

Дана інтегрована платформа призначена для зручної роботи з електронними документами та їх обробки з подальшим надсиланням через поштові сервіси. У цьому розділі наведено покрокову інструкцію щодо використання основних можливостей системи.

4.2.1 Запуск системи

Платформа може бути розгорнута у двох режимах:

1) онлайн-режим:

Користувач переходить за посиланням на розміщений веб-додаток, наприклад:

https://documents-platform.vercel.app

2) локальний запуск:

для запуску системи на локальному комп’ютері необхідно:

– клонувати репозиторій:

git clone https://github.com/username/documents-platform.git

cd documents-platform

– встановити залежності:

npm install

– запустити проєкт:

npm run dev

– відкрити браузер і перейти за адресою:

http://localhost:3000

Для повноцінної роботи необхідно створити облікові записи в сервісах **Clerk** (для авторизації) і **Convex** (для збереження даних), а також налаштувати змінні середовища .env.

4.2.2 Реєстрація та авторизація

– користувач переходить на сторінку входу /sign-in;

– можна увійти або зареєструватися, вказавши email;

– після успішної авторизації система автоматично перенаправляє користувача на головну сторінку.

Clerk автоматично керує сесією користувача — повторний вхід не потрібен, поки сесія активна.

4.2.3 Основні дії користувача

4.2.3.1 Перегляд документів

Після входу відкривається домашня сторінка зі списком наявних документів користувача. Тут можна:

- переглянути всі створені раніше документи;
- натиснути контекстне меню документу для відкриття сторінки редагування.

4.2.3.2 Створення нового файлу

Для створення нового файлу:

- натисніть кнопку “**Створити файл”**;
- вести дані у діалогове вікно;

- документ з’явиться на сторінці всіх файлів.

4.2.3.3 Редагування документа

На сторінці редагування відкривається **візуальний редактор** (на базі CKEditor):

- введіть або змініть текст документа;
- зміни зберігаються за кнопкою "Зберегти".

4.2.3.4 Надсилання документа на електронну пошту

Для надсилання документа:

- натисніть кнопку “**Відправити на пошту”**;
- відкриється діалогове вікно з полем для введення email;
- після підтвердження документ буде надіслано на вказану адресу як вкладення.

Висновки до розділу 4:

У даному розділі було здійснено опис програмної реалізації інтегрованої платформи для обробки документів із підтримкою поштових сервісів. Розглянуто основні функціональні компоненти системи, зокрема: сторінку авторизації, домашню сторінку зі списком документів, редактор документів на базі CKEditor, а також механізм надсилання документів на електронну пошту.

Надано детальне керівництво користувача, що описує процес запуску системи, реєстрації, створення та редагування документів, а також їх передачі поштою. Окрему увагу приділено простоті інтерфейсу, зручності взаємодії та автоматичному збереженню змін.

ВИСНОВКИ

У процесі створення веб-застосунку для обробки текстових документів з можливістю їх збереження, редагування та надсилання електронною поштою, було реалізовано низку функціональних рішень, які поєднують переваги сучасних веб-технологій та інструментів. Основу клієнтської частини платформи складає Next.js, що забезпечує високу продуктивність, зручну маршрутизацію та рендеринг сторінок як на клієнті, так і на сервері. Інтерфейс реалізовано з використанням React, TypeScript, HTML, CSS, а також бібліотеки компонентів shadcn/ui, що дозволило досягти інтуїтивного і сучасного вигляду застосунку.

Для автентифікації користувачів інтегровано сервіс Clerk, який гарантує надійний захист доступу до документів. У якості редактора текстових файлів було використано CKEditor 5, що дозволяє зручно створювати та форматувати документи у браузері. Зберігання та обробку даних реалізовано за допомогою хмарної реактивної бази даних Convex, що забезпечує миттєве оновлення вмісту на клієнті без необхідності ручного синхронізування.

Надсилання документів електронною поштою реалізовано через використання SMTP-протоколу, що дозволило забезпечити інтеграцію з поштовими сервісами та можливість обміну файлами безпосередньо з інтерфейсу застосунку.

Особливу увагу приділено безпеці користувацьких даних, масштабованості та стабільній роботі платформи під час навантажень. Інтеграція всіх вищезазначених інструментів дала змогу створити універсальне рішення для обробки документів, яке поєднує зручність використання та функціональну гнучкість.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1) Popular Document Management Systems URL: 10 Popular Document Management Systems (DMS) - Spiceworks

2) DocuWare website URL: Document Management Software & Workflow Solutions | DocuWare

3) Egnyte website URL: Egnyte Content Cloud for Enterprises | Start Free Trial

4) Google Workspace URL: Продукти Google Workspace – від Офіційного Партнера Google в Україні

5) SharePoint website URL: Безпечна спільна робота й доступ до вмісту | Microsoft SharePoint

6) OnlyOffice Workspace website URL: Your virtual digital workspace | ONLYOFFICE

7) Стаття в журналі IRM - A Comparative Review of Five Leading Document Management Systems Based on Information Security

8) Книга дизайн Web-сторінок: HTML+CSS URL: МОВА ОПИСУ ГІПЕРТЕКСТІВ (HTML)

9) Книга Pro Angular: Pro.Angular.6.3rd.Edition.www.EBooksWorld.ir.pdf

10) Стаття про Clerk URL: Create a Simple Authentication System for your Next Application with Clerk - DEV Community

11) Веб-сайт Clerk URL: Clerk | Authentication and User Management

12) Вікіпедія Next.js URL: Next.js — Вікіпедія

13) Sinha S. Learning Next.js: Build modern web applications using React and Next.js. Packt Publishing, 2022. ISBN: 9781803239241

14) Веб сайт Convex URL: Convex | The reactive database for app developers

15) Стаття Announcing Convex 1.0 URL: Stack by Convex

16) Веб сторінка SMPT URL: Що таке SMTP протокол: визначення | SendPulse UA

17) Веб сторінка що таке SMPT URL: Що таке SMTP (Simple Mail Transfer Protocol) - Терміни та визначення кібербезпеки

18) Офіційна документація tailwind URL: Styling with utility classes - Core concepts - Tailwind CSS

19) Офіційний блог Tailwind Labs URL: Blog - Tailwind CSS

20) Wathan A., Schoger S. *Refactoring UI*. Self-published, 2019. ISBN: 9781989603018 URL: Refactoring UI

21) Офіційна сторінка Nodemailer URL: Nodemailer | Nodemailer

22) Ogundipe O. Sending emails using Node.js and Nodemailer. LogRocket Blog URL: Sending emails in Node.js using Nodemailer - LogRocket Blog

23) Nodemailer. GitHub Repository URL: nodemailer/nodemailer: ✉ Send e-mails with Node.JS – easy as cake!

24) Офіційна сторінка date-fns URL: date-fns - modern JavaScript date utility library

25) Date-fns. GitHub Repository.URL: https://github.com/date-fns/date-fns

26) Стаття "Manipulating dates in JavaScript with date-fns" - автор: Michael Wanyoike URL: Managing Dates and Times in JavaScript Using date-fns — SitePoint

27) Офіційна сторінка radix-ui: Introduction – Radix Primitives

28) Radix UI. Radix Primitives GitHub Repository: radix-ui/primitives: Radix Primitives is an open-source UI component library for building high-quality, accessible design systems and web apps. Maintained by @workos.

29) Fielding, Roy T. "Architectural Styles and the Design of Network-based Software Architectures." Dissertation, University of California, Irvine, 2000. URL: Architectural Styles and the Design of Network-based Software Architectures

30) Richardson, C. "Microservices Patterns: With examples in Java." Manning Publications, 2018. URL: Microservices

ДОДАТОК А

Код застосунку

Shema.ts:

```
   import    {    defineSchema,    defineTable    }    from
"convex/server";
   import { v } from "convex/values";

   export const fileTypes = v.union(
     v.literal("image"),
     v.literal("csv"),
     v.literal('pdf'),
     v.literal('docx')
   )

   export   const   roles   =   v.union(v.literal("admin"),
v.literal("member"));

   export default defineSchema({
     files: defineTable({
       name: v.string(),
       orgId: v.string(),
       shouldDelete: v.optional(v.boolean()),
       fileId: v.id("_storage"),
       userId: v.id("users"),
       type: fileTypes
                                        }).index("by_orgId",
["orgId"]).index("by_shouldDelete",
["shouldDelete"]).index("by_fileId", ["fileId"]),
     favorites: defineTable({
```

```
        fileId: v.id("files"),
        orgId: v.string(),
        userId: v.id("users")
      }).index("by_userId_orgId_fileId", ["userId", "orgId",
"fileId"]),
      users: defineTable ({
        tokenIdentifier: v.string(),
        name: v.optional(v.string()),
        image: v.optional(v.string()),
        orgIds: v.array(
          v.object({
            orgId: v.string(),
            role: roles
          })
        )
      }).index("by_tokenIdentifier", ["tokenIdentifier"])
    });
```

Files.ts :

```
    import {action, internalMutation, mutation, MutationCtx,
query, QueryCtx} from './_generated/server'
    import {ConvexError, v} from 'convex/values'
    import { fileTypes } from './schema';
    import { Doc, Id } from './_generated/dataModel';
    import { api } from './_generated/api';

    export const generateUploadUrl = mutation(async (ctx) =>
{
        const identity = await ctx.auth.getUserIdentity();
```

```
        if(!identity){
            throw new ConvexError('you must be logged in to
upload a file');
        }

        return await ctx.storage.generateUploadUrl();
    })

    export const getUrl = mutation({
        args: {
            fileId: v.id("_storage")
        },
        async handler(ctx, args){
                            const    indentity    =    await
ctx.auth.getUserIdentity();

            if(!indentity){
                throw new ConvexError('you must be logged in
to upload a file');
            }

                             const    fileUrl    =    await
ctx.storage.getUrl(args.fileId);

            return fileUrl;
        }
    })

    async function hasAccessToOrg(
```

```
        ctx: QueryCtx | MutationCtx,
        orgId: string
    ){
        const indentity = await ctx.auth.getUserIdentity();

        if(!indentity){
           return null;
        }

        const user = await await ctx.db
            .query("users")
            .withIndex("by_tokenIdentifier", (q) =>
                                    q.eq("tokenIdentifier",
indentity.tokenIdentifier)
            )
            .first();

        if(!user){
            return null;
        }

          const  hasAccess  =  user.orgIds.some((item)  =>
item.orgId === orgId) ||
            user.tokenIdentifier.includes(orgId);

        if(!hasAccess){
            return null;
        }
```

```
        return {user};
    }


    export const createFile = mutation({
        args: {
            name: v.string(),
            fileId: v.id("_storage"),
            orgId: v.string(),
            type: fileTypes,
        },
        async handler(ctx, args){
            const hasAccess = await hasAccessToOrg(
                ctx,
                args.orgId,
            );

            if(!hasAccess){
                throw new ConvexError('you do not have access
to this org')
            }

            await ctx.db.insert('files', {
                name: args.name,
                orgId: args.orgId,
                fileId: args.fileId,
                type: args.type,
                userId: hasAccess.user._id,
            })
        }
```

```
    })

    export const getFiles = query({
        args: {
            orgId: v.string(),
            query: v.optional(v.string()),
            favorites: v.optional(v.boolean()),
            deletedOnly: v.optional(v.boolean()),
            type: v.optional((fileTypes)),
        },
        async handler(ctx, args){
            const access = await hasAccessToOrg(
                ctx,
                args.orgId,
            );

            if(!access){
                throw [];
            }

            let files = await ctx.db.
                query("files")
                 .withIndex("by_orgId", (q) => q.eq("orgId",
args.orgId))
                .collect();

            const query = args.query;

            if(args.favorites){
```

```
                const favorites = await ctx.db
                    .query("favorites")
                    .withIndex("by_userId_orgId_fileId", (q)
=>
                                                q.eq("userId",
access.user._id).eq("orgId", args.orgId)
                    )
                    .collect();

                files = files.filter((file) =>
                                favorites.some((favorite)  =>
favorite.fileId === file._id)
                )
            }

            if(args.deletedOnly){
                            files  =  files.filter((file)  =>
file.shouldDelete);
            } else if(!args.favorites){
                            files  =  files.filter((file)  =>
!file.shouldDelete);
            }

            if(query){
                            files  =  files.filter((file)  =>
file.name.includes(query))
            }

            if(args.type){
```

```
                files = files.filter((file) => file.type ===
args.type);
            }

            return files
        }
    })

    export const getFileById = query({
        args: {
            orgId: v.string(),
            fileId: v.string(),
        },
        async handler(ctx, args) {
            const access = await hasAccessToOrg(
                ctx,
                args.orgId,
            );

            if (!access) {
                throw [];
            }

            let files = await ctx.db.
                query("files")
                 .withIndex("by_orgId", (q) => q.eq("orgId",
args.orgId))
                .collect();
```

```
            const file = files.find((f) => f._id === args.fileId);

            if (!file) {
                throw new Error('Файл не найден');
            }

            return file;
        }
    });

    export const deletedAllFiles = internalMutation({
        args: {},
        async handler(ctx){
            const files = await ctx.db.query("files")
                    .withIndex("by_shouldDelete", (q) => q.eq("shouldDelete", true))
                .collect();

            await Promise.all(files.map(async (file) => {
                await ctx.storage.delete(file.fileId);
                return await ctx.db.delete(file._id);
            }))
        }
    })

    export const deleteFile = mutation({
        args: { fileId: v.id('files') },
        async handler(ctx, args){
```

```
            const access = await hasAccessToFile(ctx,
args.fileId);

            if(!access){
                throw new ConvexError('no access to file');
            }

            canDeleteFile(access.user, access.file);

            await ctx.db.patch(args.fileId, {
                    shouldDelete: true
            })
        }
    })

    export const restoreFile = mutation({
        args: { fileId: v.id('files') },
        async handler(ctx, args){
            const access = await hasAccessToFile(ctx,
args.fileId);

            if(!access){
                throw new ConvexError('no access to file');
            }

            canDeleteFile(access.user, access.file);

            await ctx.db.patch(args.fileId, {
                    shouldDelete: false
```

```
            })
        }
    })

    export const toggleFavorite = mutation({
        args: { fileId: v.id('files') },
        async handler(ctx, args){
                const access = await hasAccessToFile(ctx,
args.fileId);

            if(!access){
                throw new ConvexError('no access to file');
            }

           const favourite = await ctx.db.query("favorites")
                .withIndex("by_userId_orgId_fileId", q =>
                                                q.eq('userId',
access.user._id).eq("orgId",
access.file.orgId).eq("fileId", access.file._id)
                )
                .first();

            if(!favourite){
                await ctx.db.insert("favorites", {
                    fileId: access.file._id,
                    userId: access.user._id,
                    orgId: access.file.orgId
                });
            } else{
```

```
                await ctx.db.delete(favourite._id);
            }
        }
    })

    export const getAllFavorites = query({
        args: { orgId: v.string() },
        async handler(ctx, args){
                const access = await hasAccessToOrg(ctx,
args.orgId);

            if(!access){
                return [];
            }

           const favourite = await ctx.db.query("favorites")
                .withIndex("by_userId_orgId_fileId", q =>
                                            q.eq('userId',
access.user._id).eq("orgId", args.orgId)
                )
                .collect();

            return favourite;
        }
    })

    async function hasAccessToFile(ctx: QueryCtx |
MutationCtx, fileId: Id<"files">){
        const file = await ctx.db.get(fileId);
```

```
        if(!file){
            return null;
        }

        const hasAccess = await hasAccessToOrg(
            ctx,
            file.orgId
        );

        if(!hasAccess){
           return null;
        }

        return {user: hasAccess.user, file}
    }

    function canDeleteFile(user: Doc<"users">, file:
Doc<"files">){
        const canDelete =
                file.userId === user._id ||
                   user.orgIds.find((org) => org.orgId ===
file.orgId)
                ?.role === 'admin';

        if(!canDelete){
             throw new ConvexError("you have no access to
delete this file");
        }
```

```
    }

    export const getFileMeta = query({
        args: {
          fileId: v.id('files'),
        },
        handler: async (ctx, { fileId }) => {
          return await ctx.db.get(fileId);
        },
    });

    export const saveFile = mutation({
        args: { fileId: v.id('files') },
        async handler(ctx, args){
                const access = await hasAccessToFile(ctx,
args.fileId);

            if(!access){
                throw new ConvexError('no access to file');
            }

            await ctx.db.patch(args.fileId, {
                shouldDelete: true
            })
        }
    })

    export const updateFileContent = mutation({
      args: {
```

```
        fileId: v.id("_storage"),
        newStorageId: v.id("_storage"),
      },
      async handler(ctx, args) {
        const file = await ctx.db
          .query("files")
            .withIndex("by_fileId", (q) => q.eq("fileId",
args.fileId))
          .first();

        if (!file) {
          throw new ConvexError("File not found");
        }

        await ctx.db.patch(file._id, {
          fileId: args.newStorageId,
        });
      },
    });
```

Columns.tsx:

```
"use client"

import { ColumnDef } from "@tanstack/react-table"
import { Doc, Id } from "../../../../convex/_generated/dataModel"
import { formatRelative } from "date-fns"
import { useQuery } from "convex/react"
import { api } from "../../../../convex/_generated/api"
```

```
import { Avatar, AvatarFallback, AvatarImage } from "@/components/ui/avatar"
import FileCardActions from "./file-actions"

function UserCell({ userId }: { userId: Id<"users"> }){
    const userProfile = useQuery(api.users.getUserProfile, {
        userId: userId
    });
    return (
        <div className='flex gap-2 text-xs text-gray-700 w-40'>
            <Avatar className='w-6 h-6'>
                <AvatarImage src={userProfile?.image} />
                <AvatarFallback>CN</AvatarFallback>
            </Avatar>

            {userProfile?.name}
        </div>
    )
}

export const columns: ColumnDef<Doc<"files"> & {isFavorited: boolean}>[] = [
  {
    accessorKey: "name",
    header: "Name",
  },
  {
    accessorKey: "type",
    header: "Type",
  },
```

```
        {
          accessorKey: "User",
          cell: ({ row }) => {
            return <UserCell userId={row.original.userId}/>
          }
        },
        {
          header: "Uploaded On",
          cell: ({ row }) => {
              return (
                  <div>
                      {formatRelative(new Date(row.original._creationTime), new Date())}
                  </div>
              )
          }
        },
        {
          header: "Actions",
          cell: ({ row }) => {
              return (
                  <div>
                                                  <FileCardActions      file={row.original}
isFavorited={row.original.isFavorited} />
                  </div>
              )
          }
        }
      ]
```

FileActions.tsx:

```
"use client"

import { ColumnDef } from "@tanstack/react-table"
import { Doc, Id } from "../../../../convex/_generated/dataModel"
import { formatRelative } from "date-fns"
import { useQuery } from "convex/react"
import { api } from "../../../../convex/_generated/api"
import { Avatar, AvatarFallback, AvatarImage } from "@/components/ui/avatar"
import FileCardActions from "./file-actions"

function UserCell({ userId }: { userId: Id<"users"> }){
    const userProfile = useQuery(api.users.getUserProfile, {
        userId: userId
    });
    return (
        <div className='flex gap-2 text-xs text-gray-700 w-40'>
            <Avatar className='w-6 h-6'>
                <AvatarImage src={userProfile?.image} />
                <AvatarFallback>CN</AvatarFallback>
            </Avatar>

            {userProfile?.name}
        </div>
    )
}
```

```
export const columns: ColumnDef<Doc<"files"> & {isFavorited: boolean}>[] = [
  {
    accessorKey: "name",
    header: "Name",
  },
  {
    accessorKey: "type",
    header: "Type",
  },
  {
    accessorKey: "User",
    cell: ({ row }) => {
      return <UserCell userId={row.original.userId}/>
    }
  },
  {
    header: "Uploaded On",
    cell: ({ row }) => {
        return (
            <div>
                {formatRelative(new Date(row.original._creationTime), new Date())}
            </div>
        )
    }
  },
  {
    header: "Actions",
```

```
        cell: ({ row }) => {
          return (
            <div>
                                                <FileCardActions      file={row.original}
isFavorited={row.original.isFavorited} />
            </div>
          )
        }
      }
    ]
```