

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

ПОЯСНЮВАЛЬНА ЗАПИСКА

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

на тему:

«Сервіс для обміну файлами»

Виконав студент групи ПЗ-20-2
ЗАГЛАДА Мирослав Русланович

Керівник роботи:
КУБРАК Юрій Олександрович

Консультант:
МОРОЗОВ Андрій Васильович

Рецензент:
ВЛАСЕНКО Олег Васильович

Житомир – 2024

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

«ЗАТВЕРДЖУЮ»

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

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: ЗАГЛАДА Мирослав Русланович

Керівник роботи: КУБРАК Юрій Олександрович

Тема роботи: **«Сервіс для обміну файлами»**,

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

Вихідні дані для роботи: використання Web-орієнтованих технології для розробки сервісу для обміну файлами та інформаційні технології для розміщення і контролю доступу до файлів.

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

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Кубрак Ю.О., Морозов А.В.	27.02.2024р.	27.02.2024р.
2	Кубрак Ю.О., Морозов А.В.	27.02.2024р.	27.02.2024р.
3	Кубрак Ю.О., Морозов А.В.	27.02.2024р.	27.02.2024р.

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається з вебсервісу для обміну файлами та пояснювальної записки. Пояснювальна записка до випускної роботи містить 94 сторінок, 37 ілюстрацій та 5 таблиць.

Метою роботи є створення сервісу для обміну файлами з використанням сучасних технологій та методів забезпечення безпеки файлів які зберігаються.

Для зберігання файлів були використані сучасні практики що до шифрування та стискання. Відносини того кому належить файл або має доступ були змодельовані та зберігаються за допомогою графів.

Сервіс має простий та зрозумілий інтерфейс, який дозволяє користувачам легко завантажувати файли на сервіс, завантажувати файли на їх пристрій та ділитися ними з іншими користувачами.

КЛЮЧОВІ СЛОВА: ВЕБСЕРВІС, ФАЙЛ, ОБМІН, ШИФРУВАННЯ.

					ІПЗ.КР.Б — 121 — 24 — ПЗ						
Змн.	Арк.	№ докум.	Підпис	Дата	Сервіс для обміну файлами			Літ.	Арк.	Аркушів	
Розроб.		Заглада М.Р.									
Керівник		Кубрак Ю.О.							3	94	
								Житомирська політехніка група ІПЗ-20-2			
Н. контр.		Тичина О.П.									
Затвердив		Вакалюк Т.А.									

ABSTRACT

The bachelor's thesis consists of a web service for file sharing and an explanatory note. The explanatory note to the final thesis contains 94 pages, 37 illustrations and 5 tables.

The goal of the work is to create a service for exchanging files using modern technologies and methods of ensuring the security of stored files.

Modern encryption and compression practices were used to store the files. Relationships of who owns a file or has access to have been modeled and stored using graphs.

The service has a simple and clear interface that allows users to easily upload files to the service, upload files to their device and share them with other users.

KEYWORDS: WEBSERVICE, FILE, SHARE, ENCRYPTION.

		Заглада М.Р.			ІІЗ.КР.Б — 121 — 24 — ІІЗ	Арк.
		Кубрак Ю.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		4

ЗМІСТ

РЕФЕРАТ.....	3
ABSTRACT.....	4
ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ.....	6
ВСТУП.....	7
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ СЕРВІСУ ДЛЯ ОБМІНУ ФАЙЛАМИ.....	9
1.1 Постановка задачі.....	9
1.2 Аналіз аналогів програмного продукту.....	11
1.3 Вибір архітектури сервісу для обміну файлами.....	16
1.4 Обґрунтування вибору інструментальних засобів та вимоги до програмного забезпечення.....	17
Висновки до першого розділу.....	19
РОЗДІЛ 2. ПРОЕКТУВАННЯ СЕРВІСУ ДЛЯ ОБМІНУ ФАЙЛАМИ.....	21
2.1 Визначення варіантів використання та об'єктно-орієнтованої структури системи.....	21
2.2 Розробка структури бази даних системи.....	24
2.3 Проектування та реалізація алгоритмів роботи системи.....	27
2.4 Реалізація додатку сервісу для обміну файлами.....	30
Висновки до другого розділу.....	46
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З СЕРВІСОМ ДЛЯ ОБМІНУ ФАЙЛАМИ.....	47
3.1 Порядок встановлення та налаштування параметрів системи.....	47
3.2 Структура інтерфейсу та порядок роботи.....	48
3.3 Тестування роботи сервісу.....	58
Висновки до третього розділу.....	61
ВИСНОВКИ.....	62
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	64
ДОДАТКИ.....	67

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ПЗ — програмне забезпечення

БД — база даних

СУБД – система управління базами даних

API – прикладний програмний інтерфейс

CRUD – основні операції, що використовуються для управління даними
(Create , Read , Update , Delete)

REST – акронім для репрезентативної передачі стану та представляє собою архітектурний стиль проектування для розподілених гіпермедійних систем [1].

		Заглада М.Р.			ІПЗ.КР.Б — 121 — 24 — ПЗ	Арк.
		Кубрак Ю.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		6

ВСТУП

Актуальність теми. Багато хто стикався з ситуацією коли потрібно поділитися з кимось певним файлом будь-то документи, програмні файли, тощо. Для реалізації цього існує безліч сервісів від звичайних месенджерів до онлайн хмарних сховищ. Але у всіх них є певні недоліки такі як надто маленький ліміт для завантаження файлу на сервер чи не зручний процес відкриття доступу цього файлу для інших користувачів. Всі ці проблеми в певній мірі можна вирішити шляхом розробки сервісу який буде спеціалізуватися на зручному обміні файлами між користувачами. Завдяки спеціалізації він може мати більший ліміт на завантаження файла і простий та зручний інтерфейс який дозволить користувачу за декілька натискань налаштувати параметри доступу до файлу.

Актуальність таких сервісів особливо очевидна в наш час, особливо коли глобалізація та збільшення кількості віддаленої роботи створюють потребу у зручних інструментах для передачі великих об'ємів даних між співробітниками, клієнтами та партнерами. Це особливо важливо для компаній з міжнародними командами, де швидкість і зручність доступу до файлів можуть впливати на продуктивність і ефективність роботи.

Розвиток технологій і збільшення обсягу цифрового контенту вимагають надійних рішень для зберігання та обміну файлами. Від фотографій і відео до професійних документів і програмного коду, сучасні користувачі та організації потребують інструментів, які забезпечують безпечний і швидкий доступ до цих даних.

Таким чином, зручність використання таких сервісів сприяє їх популярності. Вони надають можливість легкого та швидкого обміну файлами між користувачами, незалежно від їхнього місця перебування, що особливо корисно для командної роботи чи дистанційного навчання. Багато з них мають інтуїтивно зрозумілі інтерфейси, які дозволяють легко завантажувати на сервіс, завантажувати на локальний пристрій та ділитися файлами без необхідності в

		Заглада М.Р.			ІПЗ.КР.Б — 121 — 24 — ПЗ	Арк.
		Кубрак Ю.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		7

технічних знаннях. Це робить їх доступними для широкого кола користувачів, від студентів і фрілансерів до великих корпорацій.

Метою випускної кваліфікаційної роботи бакалавра є проектування алгоритмів роботи та розробка сервісу для обміну файлами.

Встановлена мета обумовлює наступні завдання:

1. Провести аналіз потреб користувачів що до сервісів для обміну файлами.
2. Визначити архітектуру та узагальнену структуру системи.
3. Обґрунтувати та вибирати засоби реалізації сервісу.
4. Проектування системи та алгоритмів роботи сервісу.
5. Розробити сервіс для обміну файлами.

Об'єктом дослідження є процес розміщення і контролю доступу до файлів.

Предметом дослідження є використання веборієнтованих технологій, засобів та алгоритмів для розробки сервісу для обміну файлами.

В роботі було використано методи системного аналізу методи об'єктно орієнтованого проектування, веб-програмування та дизайну.

Даний сервіс дозволить користувачам швидко та зручно передавати документи, фотографії, відео та інші файли між різними пристроями та користувачами незалежно від їхнього місцезнаходження. Це особливо корисно для роботи та навчання, оскільки спрощує спільну роботу над проектами, забезпечуючи легкий доступ до необхідних матеріалів у будь-який час.

Структура та обсяг проекту (роботи). Випускна робота складається з вступу, 3-х розділів, висновків, списку використаних джерел із 33 найменувань, 1 додатку, 5 таблиць та 37 ілюстрацій по тексту. Повний обсяг роботи становить 94 сторінок, з них 63 сторінки основного тексту.

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ СЕРВІСУ ДЛЯ ОБМІНУ ФАЙЛАМИ

1.1 Постановка задачі

Мета кваліфікаційної роботи бакалавра є впровадження сервісу для обміну файлами, який спростить та забезпечить безпеку процесу обміну файлами між користувачами.

Що до переліку задач які необхідно вирішити під час розробки сервісу то можна зазначити наступні задачі:

1. Аналіз вимог та проектування системи:

- аналіз вимог користувачів: визначити цільову аудиторію, їх потреби та сценарії використання;
- вимоги до функціоналу: описати основні функції сервісу, такі як завантаження, зберігання, обмін та управління файлами;
- вимоги до безпеки: визначити вимоги до безпеки, включаючи аутентифікацію, авторизацію та шифрування даних;
- архітектурне проектування: розробити архітектуру системи, визначити основні компоненти та їх взаємодію;
- вибір технологій: вибрати мови програмування, бібліотеки, бази даних та інші технології для реалізації сервісу.

2. Розробка серверної частини:

- розробка API: створити REST API для взаємодії клієнтської частини з серверною;
- зберігання файлів: реалізувати функціонал для завантаження та зберігання файлів на сервері;
- база даних: спроектувати та реалізувати базу даних для зберігання метаданих про файли, користувачів та їх зв'язків;
- аутентифікація та авторизація: реалізувати систему реєстрації та входу в обліковий запис;

- логування та моніторинг: впровадити системи логування та моніторингу для відстеження роботи сервісу та виявлення помилок.

3. Розробка клієнтської частини:

- інтерфейс користувача: розробити макети та прототипи інтерфейсу користувача;
- реалізація вебінтерфейсу: створити інтерфейс для взаємодії користувачів із сервісом;
- адаптивний дизайн: забезпечити коректне відображення інтерфейсу на різних пристроях та розмірах екранів.

4. Безпека:

- захист даних: реалізувати шифрування файлів під час зберігання та передачі;
- захист від атак: впровадити захист від поширених атак, таких як xss, csrf.

5. Тестування:

- модульне тестування: написати тести для перевірки окремих компонентів системи.

6. Документація та підтримка:

- користувацька документація: підготувати інструкції для користувачів сервісу;
- технічна документація: описати архітектуру, API та модулі системи для розробників.

7. Юридичні аспекти:

- політика конфіденційності: розробити та впровадити політику конфіденційності даних користувачів;
- умови використання: створити умови використання сервісу, які мають прийняти користувачі перед початком використання сервісу;
- відповідність законам: забезпечення відповідності сервісу місцевим та міжнародним законам про захист даних (gdpr, cсра тощо).

		Заглада М.Р.			ІПЗ.КР.Б — 121 — 24 — ПЗ	Арк.
		Кубрак Ю.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		10

У результаті виконання цих задач буде створений зручний та ефективний сервіс, який дозволить користувачам легко надсилати та отримувати файли без необхідності в складних налаштуваннях який зможе забезпечити високу швидкість передачі, захист конфіденційності даних та надійність зберігання файлів.

1.2 Аналіз аналогів програмного продукту

Для проведення дослідження потрібно було обрати та проаналізувати існуючі аналогічні програмні продукти для того щоб дізнатись про те як такий сервіс повинен функціонувати, які вимоги мають користувачі до них, їх інтерфейс, тощо, для того щоб визначити подальший напрямок розвитку сервісу. В даному розділі наведено порівняльний аналіз трьох сервісів-аналогів разом з їх короткою характеристикою.

WeTransfer - це онлайн-платформа для обміну файлами, яка дозволяє користувачам відправляти великі файли через Інтернет (рис. 1.1). Сервіс був заснований у 2009 році в Нідерландах і швидко здобув популярність завдяки своїй простоті та зручності. WeTransfer дозволяє відправляти файли об'ємом до 2 ГБ безкоштовно, а для користувачів, які потребують більше можливостей, пропонується платна версія WeTransfer Pro.

До основних переваг можна віднести:

- простий та інтуїтивний інтерфейс, а також відсутність необхідності реєстрації для відправки файлів, що робить його зручним для користувачів будь-якого рівня технічної підготовки;
- швидке завантаження та відправка файлів, що економить час користувачів;
- використання шифрування для забезпечення безпеки файлів під час передачі;
- можливість відправляти файли об'ємом до 2 ГБ безкоштовно, що є великою перевагою для багатьох користувачів;

- для тих, хто потребує додаткових функцій, WeTransfer Pro пропонує збільшений ліміт до 200 ГБ можливість створювати власні канали та захищати файли паролями.

Але в ньому також наявні деякі недоліки:

- безкоштовна версія дозволяє відправляти файли лише до 2 ГБ, що може бути недостатнім для деяких користувачів;
- безкоштовні файли зберігаються лише протягом 7 днів, після чого вони автоматично видаляються;
- в безкоштовній версії відсутні деякі функції управління файлами, такі як захист файлів паролем;
- деяким користувачам вартість платної версії може здатися високою, враховуючи доступність безкоштовних альтернатив;

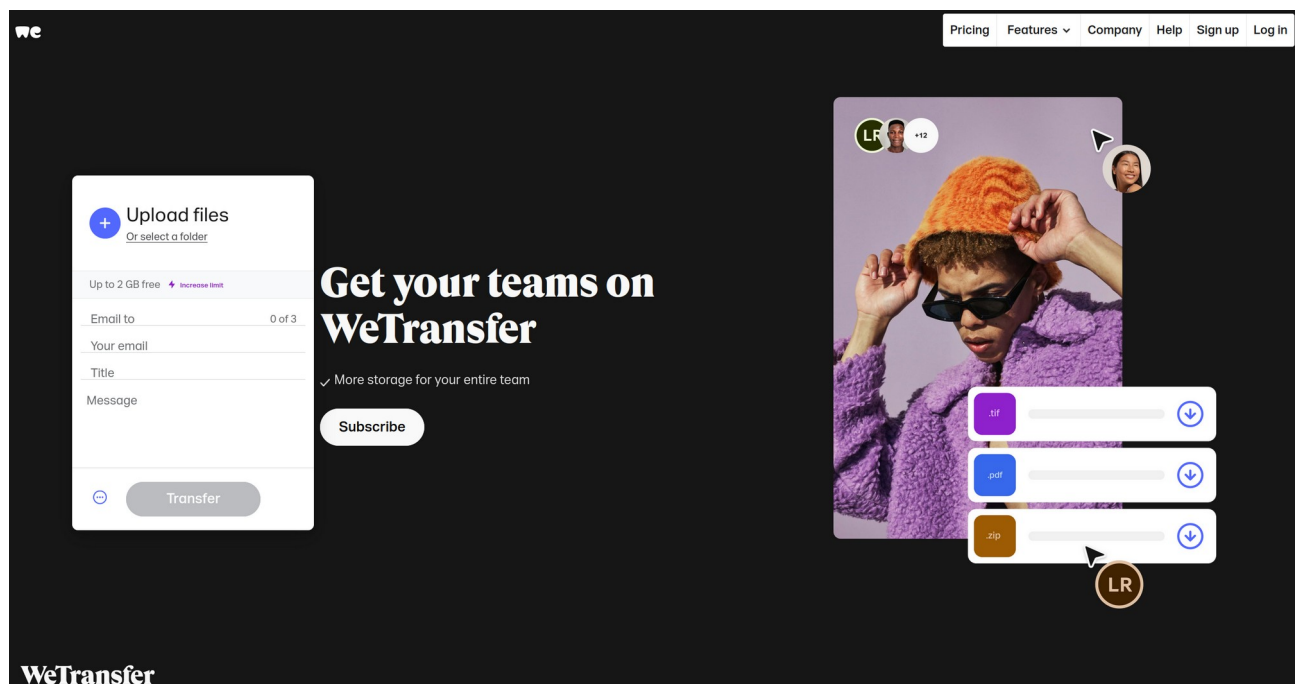


Рис 1.1. Інтерфейс WeTransfer [2]

SendAnywhere - це сервіс, який дозволяє користувачам передавати файли будь-якого розміру між різними пристроями та платформами (рис. 1.2). Він підтримує передачу файлів між смартфонами, планшетами, комп'ютерами, а також через веб-додаток. Основною характеристикою сервісу є його простота

		Заглада М.Р.			ІПЗ.КР.Б — 121 — 24 — ПЗ	Арк.
		Кубрак Ю.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		12

та швидкість передачі файлів без необхідності реєстрації чи входу в обліковий запис.

Основні переваги включають:

- інтерфейс сервісу дуже простий та інтуїтивно зрозумілий, що робить процес передачі файлів зручним навіть для користувачів без технічних знань;
- даний доступний для багатьох пристроях та має нативні додатки для мобільних платформ;
- для використання сервісу не потрібна реєстрація або вхід в обліковий запис, що спрощує процес передачі файлів;
- використання прямих підключень дозволяє швидко передавати файли без затримок, особливо якщо пристрої знаходяться в одній мережі;
- сервіс дозволяє передавати файли набагато більшого розміру в безкоштовній версії (до 10 ГБ), що є важливою перевагою порівняно з багатьма іншими сервісами, які мають обмеження на розмір файлів;
- для забезпечення безпеки файлів користувач може обрати опцію передачі файлу за одноразовим кодом, який потрібно ввести на пристрої-отримувачі.

До основних недоліків можна віднести:

- файли які завантажені в даний сервіс, зберігаються протягом обмеженого часу (лише 48 годин), що може бути незручно для користувачів, які хочуть зберігати файли довше;
- у безкоштовній версії сервісу може бути обмежена швидкість передачі файлів порівняно з преміум версією;
- безкоштовна версія сервісу містить рекламу, що може бути дратівливим для деяких користувачів;
- деякі додаткові функції такі як захист файлів паролем доступні тільки в платній версії.

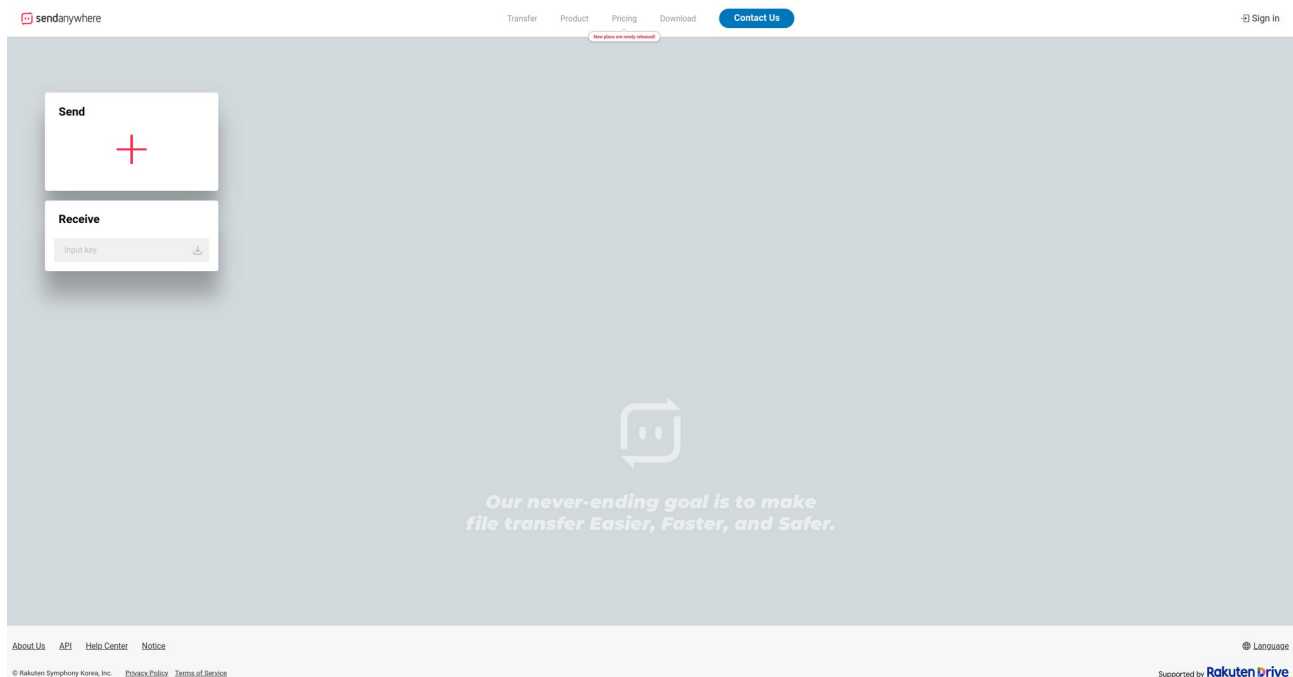


Рис 1.2. Інтерфейс SendAnywhere [3]

TransferNow - платформа для передавання та обміну файлами різних типів, включаючи великі файли, медіа-контент та документи (рис. 1.3).

Переваги TransferNow:

- сервіс пропонує інтуїтивно зрозумілий і легкий у використанні інтерфейс де користувачі можуть швидко зрозуміти, як завантажувати та надсилати файли без особливих труднощів;
- сервіс дозволяє відправляти великі файли (до 5 ГБ в безкоштовній версії) швидко завдяки швидкісним серверам, що підвищує зручність використання для користувачів з високими вимогами до часу передачі;
- основні функції цього сервісу доступні безкоштовно, що робить його доступним для широкого кола користувачів без необхідності великих витрат;

Недоліки TransferNow:

- безкоштовна версія сервісу має обмеження на максимальний обсяг передачі даних та швидкість завантаження, що може бути недостатнім для великих обсягів роботи;

		Заглада М.Р.			ІПЗ.КР.Б — 121 — 24 — ПЗ	Арк.
		Кубрак Ю.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		14

- після того як файли відправлені, даний не надає засобів для подальшого керування ними в безкоштовній версії (наприклад, можливість видалення або зміни доступу).

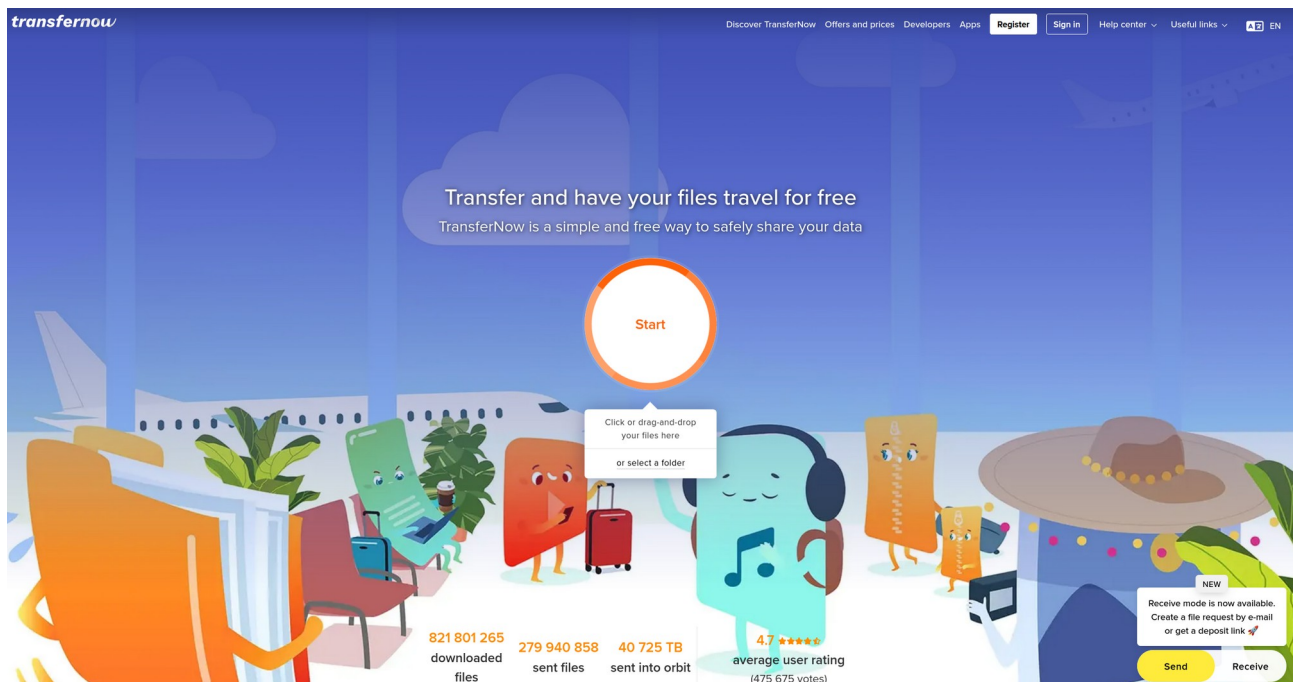


Рис 1.3. Інтерфейс TransferNow [4]

Враховуючи аналіз сервісів наведений вище, можна скласти порівняльну таблицю вищезгаданих аналогів (табл. 1.1).

Таблиця 1.1

Таблиця порівняння сервісів для обміну файлів

Характеристика	WeTransfer	SendAnywhere	Transfernow
Зручність інтерфейсу	+	+	+
Кросплатформенність	+	+	+
Захист паролем	+/-	+/-	+
Термін зберігання файлів	+	-	+
Відсутність обмежень на швидкість передачі	+	-	-
Інструменти для керування файлами	+	+	-

Переглянувши порівняльну таблицю можна визначити яким повинен бути сервіс та який функціонал він повинен мати: простий та зрозумілий інтерфейс, можливість створення акаунту де користувач може переглядати всі завантажені ним файли та виконувати певні дії з ними, можливість захисту паролем файлів від сторонніх очей, швидку передачу файлів та можливість доступу до них з будь-якого пристрою.

1.3 Вибір архітектури сервісу для обміну файлами

При проектуванні сервісу для обміну файлами, розглядались кілька основних архітектурних патернів, кожен з яких має свої переваги та недоліки.

Серед них можна виділити:

1. Клієнт-серверна архітектура:

- переваги: централізоване управління даними, сервер може обслуговувати велику кількість клієнтів одночасно, а також може бути розширений за рахунок додавання додаткових ресурсів, простий в розгортанні та оновленні;
- недоліки: якщо сервер виходить з ладу, весь сервіс перестає працювати і сервер може стати вузьким місцем при високому навантаженні.

2. Монолітна архітектура:

- переваги: проста розробка та розгортання, легке тестування;
- недоліки: складність в масштабуванні, складність у підтримці великих кодових баз, обмежена гнучкість.

3. Мікросервісна архітектура:

- переваги: кожен сервіс виконує окрему функцію, що полегшує розробку, тестування та масштабування, а також вони можуть бути оновлені незалежно один від одного;
- недоліки: складність у налаштуванні комунікацій між сервісами, підвищена складність розгортання та тестування.

Проаналізувавши надане порівняння архітектур вище було прийнято рішення обрати клієнт-серверна архітектуру. Дана архітектура передбачає розподіл обчислювальних задач між клієнтськими пристроями користувачів і центральним сервером, що забезпечує централізоване управління і зберігання файлів, що дозволяє підтримувати високу рівень безпеки та ефективності при обробці і передачі даних.

Сервер виконує основну роль у системі: він відповідає за зберігання файлів, управління правами доступу, автентифікацію користувачів, а також за обробку запитів на завантаження та вивантаження файлів. Сервер забезпечує централізовану базу даних, в якій зберігаються всі метадані про файли та користувачів, що дозволяє легко відстежувати і управляти активністю в системі, в той час як клієнти, з іншого боку, взаємодіють із сервером за допомогою спеціалізованого вебінтерфейсу.

Вибір клієнт-серверної архітектури обумовлений рядом переваг. По-перше, централізоване управління дозволяє швидко і ефективно впроваджувати оновлення системи і забезпечувати її масштабованість. По-друге, така архітектура полегшує захист даних, оскільки всі критичні процеси і зберігання інформації знаходяться під контролем центрального сервера. По-третє, клієнт-серверна модель забезпечує високу продуктивність і стабільність роботи системи, оскільки обробка даних розподіляється між сервером і клієнтськими пристроями.

Таким чином, клієнт-серверна архітектура була обрана як оптимальне рішення для розробки сервісу обміну файлами, забезпечуючи надійність, безпеку та зручність використання.

1.4 Обґрунтування вибору інструментальних засобів та вимоги до програмного забезпечення

Node.js: слугує середовищем виконання програми. Його модель вводу/виводу, керована подіями, що не блокується, робить його добре придатним для створення масштабованих і високопродуктивних вебдодатків та

		Заглада М.Р.			ІПЗ.КР.Б — 121 — 24 — ПЗ	Арк.
		Кубрак Ю.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		17

дозволяють обробляти велику кількість одночасних запитів, що в даному випадку особливо важливо. Крім того, широка екосистема модулів і пакетів дозволяє розробникам використовувати широкий спектр інструментів і бібліотек для оптимізації процесів розробки [5].

TypeScript: обрано як мову програмування для бекенду, щоб підвищити зручність обслуговування коду та продуктивність розробників. Завдяки статичній типізації TypeScript дозволяє виявляти потенційні помилки на ранніх стадіях і забезпечує кращу навігацію в коді та можливості рефакторингу, що призводить до зменшення кількості помилок і підвищення якості коду [6].

Koa.js: мінімалістичний веб-фреймворк для Node.js, розроблений як заміна Express.js тими ж самими розробниками, що надає потужний набір функцій для створення вебдодатків та API. Він використовує сучасний підхід з використанням `async/await`, що спрощує роботу з асинхронним кодом і робить його більш зручним в розробці ніж його попередник. Простота та гнучкість роблять його ідеальним вибором для швидкого створення прототипів та розробки REST API, обробки HTTP-запитів та маршрутизації [7].

ArangoDB: мультимодальна NoSQL база даних, яка поєднує в собі можливості графової, ключ-значення та документної моделей, що дозволяє використовувати найефективніший підхід для різних частин сервісу. Вона надає потужні можливості для індексування, зберігання та опрацювання даних в залежності від потреб проекту та має гарні можливості щодо розширення як вертикально так і горизонтально [8].

Nuxt.js: мета фреймворк який дозволяє розробникам створювати універсальні додатки та використовується для створення інтерфейсу сервісу. Він є відмінним вибором завдяки своїй підтримці серверного рендерингу (SSR), що покращує SEO та швидкість завантаження сторінок, забезпечуючи кращий користувацький досвід. Цей фреймворк на базі Vue.js дозволяє створювати як серверні, так і клієнтські вебдодатки, забезпечуючи універсальність розробки. Крім того, Nuxt.js надає добре організовану архітектуру проекту з чіткою структурою файлів, що значно полегшує розробку та підтримку коду [9].

		Заглада М.Р.			ІПЗ.КР.Б — 121 — 24 — ПЗ	Арк.
		Кубрак Ю.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		18

Daisy UI: бібліотека компонентів інтерфейсу користувача для додатку яка базується на Tailwind CSS, що надає заздалегідь розроблені та оформлені компоненти для будь-яких сценаріїв оскільки вона є незалежною від використання конкретних бібліотек для відмальовки інтерфейсу. Вона дозволяє швидко створювати стильні та адаптивні інтерфейси. Інтеграція з Tailwind CSS дозволяє використовувати прості класи для створення елементів інтерфейсу без написання великої кількості CSS-коду. Це забезпечує швидкість розробки та гнучкість, оскільки розробники можуть легко налаштовувати та розширювати стильові компоненти відповідно до потреб проекту [10].

Visual Studio Code (VSCode): середовище розробки (IDE) для написання коду, що було обрано через свою високу продуктивність, легку вагу та широку підтримку розширень для TypeScript, Node.js, Vue.js та Nuxt.js. Воно надає потужні інструменти для відладки коду, що допомагають швидко знаходити та виправляти помилки. Вбудована підтримка системи контролю версій Git дозволяє ефективно керувати версіями коду та співпрацювати з іншими розробниками, що є важливим для командної роботи над великими проектами [11].

Postman: використовується для тестування та налагодження кінцевих точок API під час розробки. Його інтуїтивно зрозумілий інтерфейс дозволяє розробникам надсилати HTTP-запити, перевіряти відповіді та автоматизувати робочі процеси тестування, забезпечуючи надійність та продуктивність бекенд API [12].

Висновки до першого розділу

В ході виконання першого розділу було проаналізовано завдання на кваліфікаційну роботу, основне призначення та мету створення сервісу.

Було розглянуто три основних аналога до даного сервісу, проаналізовано їх переваги та недоліки на основі яких було складено їх порівняльну характеристику на основі якої було побудовано подальший план розвитку

		Заглада М.Р.			ІПЗ.КР.Б — 121 — 24 — ПЗ	Арк.
		Кубрак Ю.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		19

сервісу в якому зазначено необхідний функціонал який будуть очікувати користувачі.

Був виконаний комплексний аналіз та вибір оптимальної архітектури де було проведено оцінку різних архітектурних підходів, зокрема монолітної архітектури, мікросервісної архітектури та клієнт-серверної архітектури. Кожен підхід був проаналізований на предмет переваг та недоліків у контексті вимог до системи. Це дозволило обґрунтовано вибрати клієнт-серверної архітектуру як найкращу, тому що вона найкраще відповідала потребам проекту, забезпечуючи необхідний рівень гнучкості, масштабованості та надійності.

Також було проаналізовано та обґрунтовано інструментарій який буде використовуватись для розробки сервісу та встановлено вимоги до апаратного забезпечення.

		Заглада М.Р.			ІПЗ.КР.Б — 121 — 24 — ПЗ	Арк.
		Кубрак Ю.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		20

РОЗДІЛ 2. ПРОЕКТУВАННЯ СЕРВІСУ ДЛЯ ОБМІНУ ФАЙЛАМИ

2.1 Визначення варіантів використання та об'єктно-орієнтованої структури системи

Аналіз вимог до проєкту дав змогу виокремити двох основних користувачів: авторизований користувач та неавторизований користувач. На основі них ми можемо визначити варіанти використання сервісу та побудувати діаграму використання програмної системи для відповідних користувачів (рис. 2.1).

Для створення цієї та наступних діаграм був використаний вебдодаток Draw.io оскільки він є потужним та простим у використанні для створення різноманітних діаграм онлайн [13].

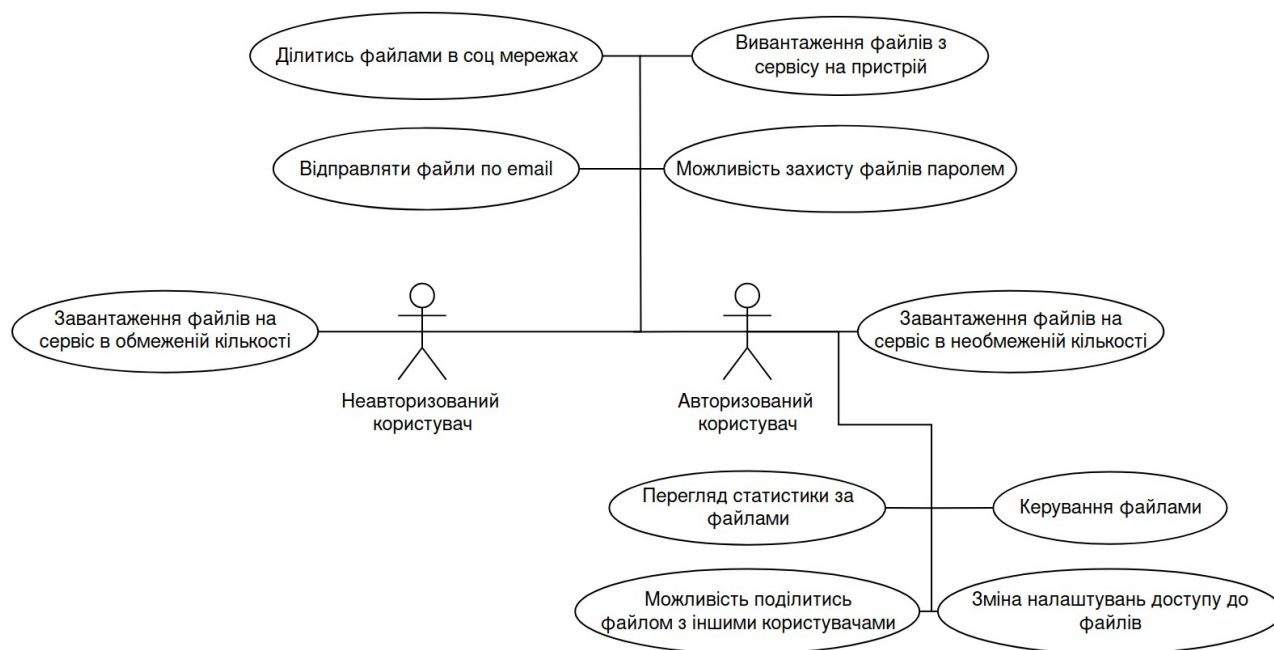


Рис. 2.1. Варіанти використання сервісу для обміну файлами

З основного що можна побачити на цих діаграмах, система буде мати 2 основних типи користувачів: неавторизований і авторизований користувач. Неавторизований користувач має найменші можливості серед трьох і має лише базовий функціонал сервісу такі як можливість завантаження обмеженої кількості файлів на сервер в день, відправки файлів по email та захист файлів

паролем. Для доступу до повного функціоналу сервісу користувачу потрібно зареєструватись та увійти в свій обліковий запис, оскільки даний функціонал вимагає прив'язку файлу до конкретного користувача, але оскільки у випадку якщо він не зареєстрований, то після завантаження файлу на сервер, файл не прив'язується до конкретного користувача, тобто чіткого власника він мати не буде, через це він цим функціоналом скористатись на зможе.

Після виконання цих дій він стає авторизованим користувачем. Авторизований користувач вже має доступ до повного функціоналу сервісу такий як особистий кабінет де він може переглядати свої файли, виконувати певні дії над ними такі як пошук, фільтрація, зміна рівнів доступу та інші, можливість ділитись файлами з іншими користувачами, безлімітну можливість завантаження файлів на сервер та зручний інтерфейс для контролю усіх файлів які завантажив користувач.

Тепер, коли були проаналізовані основні варіанти використання, можна приступити до побудови діаграми класів основних компонентів системи (рис. 2.2). Для її побудови було використано інструмент TsUML2 [14], що дозволяє з легкістю представити візуально структуру системи шляхом автоматичної побудови діаграми виходячи з наявного коду.

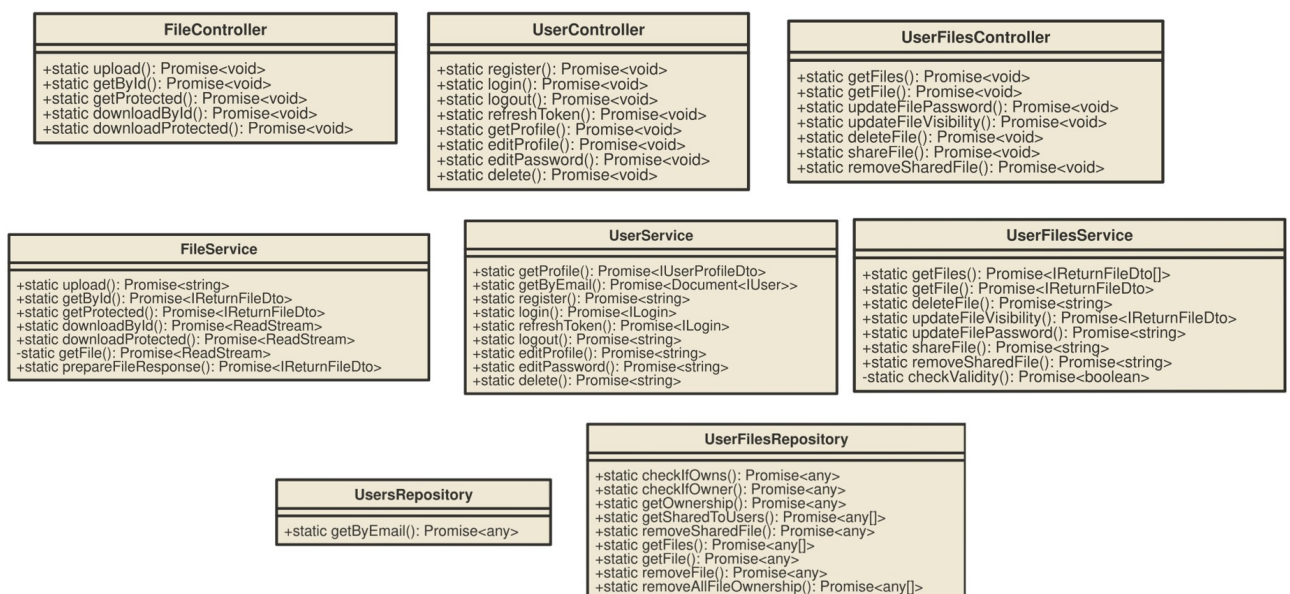


Рис. 2.2. Діаграма класів

Під час проектування системи був використаний паттерн контроллер-сервіс-репозиторій, що дає змогу розбити великі шматки коду на окремі модулі для кращої читабельності та підтримки в майбутньому.

На даній діаграмі присутні три основні типи класів:

- контроллер: відповідає за обробку HTTP-запитів і відповіді на них, Не містить жодної бізнес-логіки через що спілкується з сервісним шаром для виконання завдань;
- сервіс: містить майже всю бізнес-логіку програми та використовує репозиторій для доступу до даних;
- репозиторій: надає абстракцію для доступу до даних та спілкується з базою даних або іншим сховищем даних.

Для простоти сприйняття далі дані три типи класів будуть описані як один загальний модуль.

Модуль User — містить основні методи для роботи з користувачами.

Методи:

- register(): реєстрація користувача;
- login(): вхід в обліковий запис;
- logout(): вихід з облікового запису;
- refreshToken(): оновлення токена автентифікації для користувача;
- getProfile(): отримання профілю користувача;
- editProfile(): редагування профілю користувача;
- editPassword(): редагування паролю користувача;
- delete(): видалення профілю користувача.

Модуль Fiels — містить основні методи для роботи з файлами.

Методи:

- upload(): завантаження файлу на сервіс;
- getById(): отримання інформації про файл за ідентифікатором;
- getProtected(): отримання інформації про файл за ідентифікатором (тільки для файлів які захищені паролем);
- downloadById(): завантаження файлу на пристрій за ідентифікатором;

		Заглада М.Р.			ІПЗ.КР.Б — 121 — 24 — ІЗ	Арк.
		Кубрак Ю.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		23

- downloadProtected(): завантаження файлу на пристрій за ідентифікатором (тільки для файлів які захищені паролем).

Модуль UserFiles — містить основні методи для роботи з функціоналом пов'язаним з менеджментом файлів.

Методи:

- getFiles(): отримання інформації про всі файли користувача;
- getFile(): отримання інформації про файл користувача за ідентифікатором;
- updateFilePassword(): редагування паролю до файлу (тільки для файлів які захищені паролем);
- updateFileVisibility(): редагування рівню доступу до файлу;
- shareFile(): можливість поділитись файлом з іншим користувачем;
- removeSharedFile(): відміна можливість поділитись файлом з іншим користувачем;

2.2 Розробка структури бази даних системи

Для збереження інформації сервісу була обрана СУБД ArangoDB, це гнучка, масштабована та швидка база даних, яка підтримує моделі документів, графів та ключ-значення, що прийде в нагоді під час моделювання структури збереження даних.

База даних сервісу має чотири колекції:

- Users – колекція, яка зберігає інформацію про користувачів;
- Files – колекція, яка зберігає інформацію про файли;
- UserFiles – колекція яка відображає ребро графа. Вона зберігає зв'язки між користувачами та їх файли;
- UploadLimit – допоміжна колекція, яка зберігає IP адреси користувачів для відслідковування їх поточного ліміту на завантаження файлів (тільки не авторизовані користувачі).

Опис кожної з колекцій бази даних наведений у таблицях 2.1 – 2.4

		Заглада М.Р.			ІПЗ.КР.Б — 121 — 24 — ІЗ	Арк.
		Кубрак Ю.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		24

Таблиця 2.1

Опис таблиці Users

Поле	Тип	Призначення
username	string	Ім'я
email	string	Email користувача
password	string	Пароль користувача

Таблиця 2.2

Опис таблиці Files

Поле	Тип	Призначення
name	string	Назва файлу
path	string	Шлях в файловій системі де зберігається файл
privacy	string	Рівень доступу до файлу
password	string	Пароль файлу (тільки для файлів які захищені паролем)
uploadDate	string	Дата завантаження на сервіс
deletionDate	string	Дата коли файл буде автоматично видалено
viewCount	number	Кількість переглядів
downloadCount	number	Кількість завантажень
authTag	string	Допоміжне поле для збереження мітки автентифікації яка використовується для розшифрування файлів

Таблиця 2.3

Опис таблиці UserFiles

Поле	Тип	Призначення
_from	string	Ідентифікатор користувача, який має доступ до цього файлу

Продовження таблиці 2.3

_to	string	Файл яким володіє користувач
owner	boolean	Мітка того чи є даний користувач власником цього файлу
grantedBy	string	Ім'я користувача який надав доступ до цього файлу (тільки якщо користувач не є власником цього файлу)

Таблиця 2.4

Опис таблиці UploadLimit

Поле	Тип	Призначення
_key	string	ІР адреса користувача
uploadCount	string	Поточна кількість завантажених файлів користувачем
expireAt	string	Дата коли дане обмеження анулюється (1 день за замовчуванням)

Для автоматичного видалення записів про файли та обмеження завантажених файлів були використані Time To Live (TTL) [15] індекси які забезпечують зберігання даних лише протягом визначеного часу, після якого записи автоматично видаляються. Для того щоб скористатись ними потрібно додати до кожного запису поле, яке визначає час створення або закінчення життя запису, в даному випадку, поле deletionDate або expireAt. Потім на цьому полі створюється новий індекс після чого подальший процес видалення записів автоматизується самою СУБД.

Однак, дані індекси також мають деякі обмеження. Наприклад, видалення може не відбуватися точно у заданий момент через особливості реалізації механізмів очистки баз даних, тому що масова очистка може вплинути на

продуктивність системи. В даному випадку ArangoDB проводить очистку поступово, щоб знизити навантаження на ресурси.

TTL індекси надають ефективний спосіб автоматичного видалення записів через певний час після їх створення, знижуючи навантаження на розробників і дозволяючи уникнути створення складних механізмів видалення старих даних.

2.3 Проектування та реалізація алгоритмів роботи системи

Всі функції додатку представлені в простій формі які можуть бути зрозумілими для звичайного користувача. Загальний алгоритм роботи системи сервісу можна побачити на діаграмі активності (рис. 2.3).

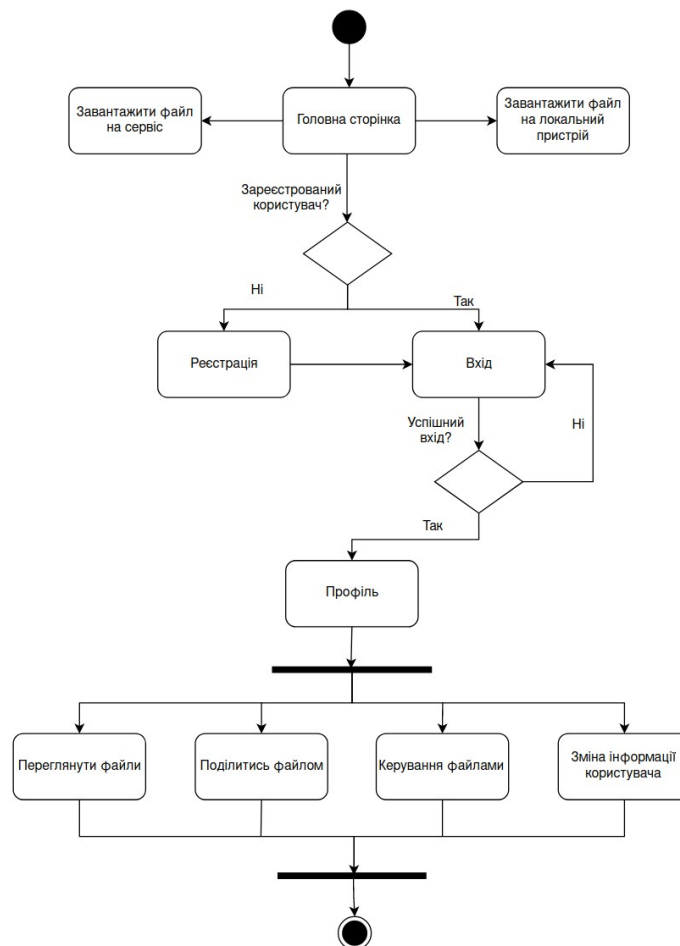


Рис. 2.3. Діаграма активності

Користувач починає роботу з сервісом на головній сторінці, де він може або завантажити файл на сервіс або вивантажити його на свій пристрій. Для

доступу до повного функціоналу сервісу користувачу потрібно бути зареєстрованим і якщо він не зареєстрований то повинен спочатку створити обліковий запис, і після цього він повинен увійти в нього. Після виконання цих кроків користувач потрапляє на сторінку профілю де він вже буде мати змогу скористатись всім наявним функціоналом сервісу.

Клієнтська частина сервісу була написана з використанням Nuxt.js, що дозволяє нам скористатись технологією серверної побудови сторінок (SSR). Серверна побудова сторінок — це підхід до побудови вебсторінок, при якому вміст сторінки генерується на сервері перед відправкою користувачу. Замість того, щоб браузер користувача самостійно завантажував JavaScript та збирав вебсторінку, сервер робить цю роботу заздалегідь.

Наприклад, коли користувач відправляє запит на сторінку сервісу, цей запит надходить на сервер. Сервер приймає цей запит і починає обробляти його. Спочатку він завантажує відповідний компонент або сторінку, яку запитав користувач. Потім він збирає всі необхідні дані для цієї сторінки, звертаючись до відповідних API або баз даних. Це відбувається на сервері до того, як будь-який HTML-код буде відправлено користувачеві. Після отримання всіх даних він генерує сторінку з уже заповненими даними. Цей HTML-код містить повний контент сторінки, що робить його готовим до відображення у браузері користувача без необхідності додаткових запитів для початкового відображення контенту.

Окрім того, разом з HTML-кодом сервер відправляє на клієнтську сторону JavaScript-код додатка. Коли браузер завантажує цю сторінку, він також завантажує та виконує JavaScript-код, що дозволяє повністю побудувати весь функціонал сервісу на клієнтській стороні. Цей процес називається "гідратація" (hydration). Під час гідратації JavaScript-код бере вже відображену сервером HTML-сторінку і перетворює її в повністю інтерактивний Vue додаток. Таким чином, Nuxt.js забезпечує швидке початкове завантаження за рахунок серверного рендерингу, а потім надає всі переваги клієнтського рендерингу, такі як інтерактивність та динамічні оновлення.

Завдяки використанню цього підходу покращується продуктивність додатка та пошукова оптимізація (SEO), оскільки пошукові роботи отримують повністю відмальовану сторінку без необхідності виконувати JavaScript. Це особливо важливо в нашому випадку, де час завантаження та SEO мають критичне значення.

Далі можна побачити діаграму діяльності для одного з головних видів функціоналу з яким буде взаємодіяти користувач на сервісу — завантаження файлів на сервіс (рис. 2.4).

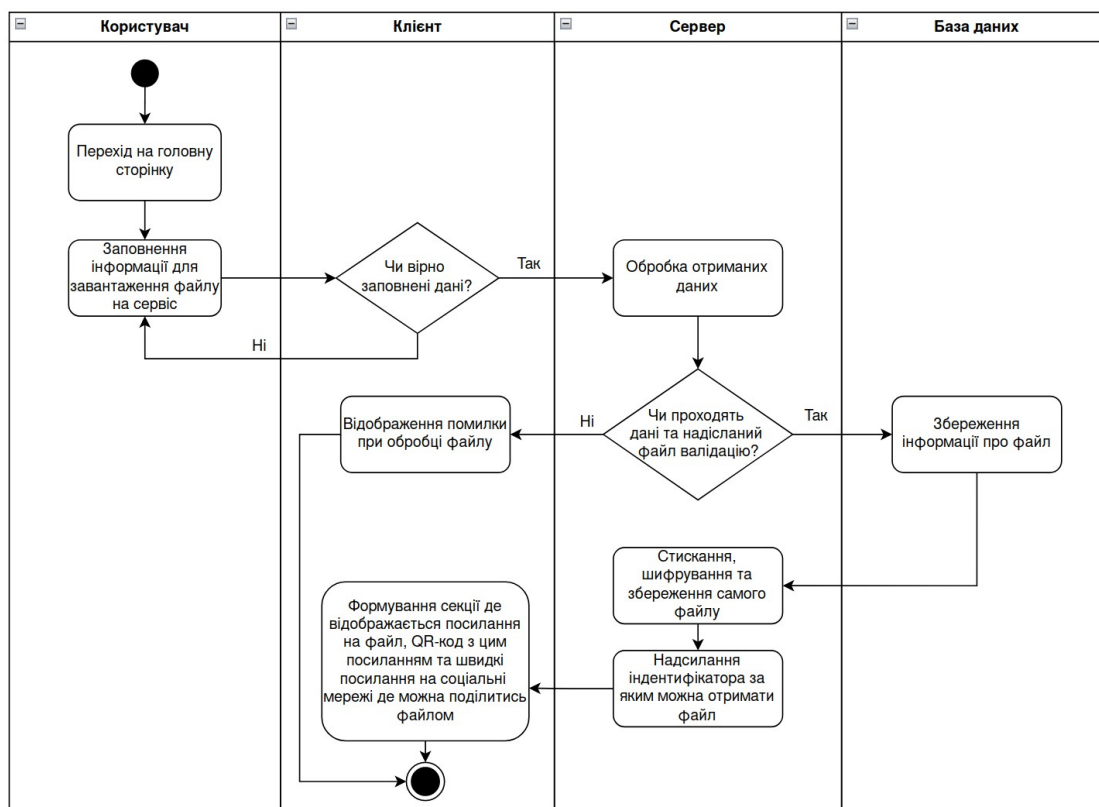


Рис. 2.4. Діаграма діяльності для завантаження файлів на сервіс

Користувач починає роботу після переходу на головну сторінку де він може завантажити файл та вказати відповідні для нього параметри. Далі відбувається валідація даних на стороні клієнта і якщо вона не була пройдена то користувачу потрібно знову ввести коректні дані. Після цього клієнт надсилає запит на сервер для збереження файлу. Він в свою чергу також виконує валідацію як даних так і файлу і якщо якась частина не пройшла її успішно то сервер відправляє відповідь з помилкою, після чого клієнт

відображає її для користувача. При успішній валідації сервер надсилає запит для збереження інформації про файл, стискає, шифрує та зберігає сам файл після чого надсилає ідентифікатор за яким можна отримати сам файл. Клієнт в свою чергу на основі нього формує секцію нову секцію на сторінці де може побачити посилання на свій файл.

2.4 Реалізація додатку сервісу для обміну файлами

Клієнтська та серверна частина сервісу були розбиті на дві окремих програми для полегшення розуміння та підтримки сервісу в майбутньому.

Серверну частину було розроблено за допомогою мінімалістичного фреймворка Koa.js, що дало змогу зменшити використання ресурсів для оперування серверної частини.

В фалі index.ts відбувається ініціалізація об'єкту сервера де до нього у вигляді проміжних функцій (middleware) передаються налаштування такі як параметри безпеки, функціонал логування та парсингу даних які передаються в запиті, а також роутери які являють собою колекції шляхів за якими користувач може звертатись для роботи з сервером.

```
const app = new Koa();

app.use(
  koaHelmet({
    contentSecurityPolicy: {
      directives: {
        defaultSrc: ['"self"'],
        scriptSrc: ['"self"', '"unsafe-inline"', 'cdnjs.cloudflare.com'],
        styleSrc: [
          '"self"',
          '"unsafe-inline"',
          'cdnjs.cloudflare.com',
          'fonts.googleapis.com',
        ],
        fontSrc: ['"self"', 'fonts.gstatic.com'],
        imgSrc: [
          '"self"',
          'data:',
          'online.swagger.io',
          'validator.swagger.io',
        ],
      },
    },
  }),
);
```

		Заглада М.Р.			ІПЗ.КР.Б — 121 — 24 — ІЗ	Арк.
		Кубрак Ю.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		30

```

app.use(
  cors({
    credentials: true,
  }),
);
app.use(logger);
app.use(bodyParser());

app.use(indexRouter.routes()).use(indexRouter.allowedMethods());
app.use(authRouter.routes()).use(authRouter.allowedMethods());
app.use(userRouter.routes()).use(userRouter.allowedMethods());
app.use(fileRouter.routes()).use(fileRouter.allowedMethods());
app.use(userFilesRouter.routes()).use(userFilesRouter.allowedMethods());

```

Трохи нижче в тому ж файлі також відбувається перевірка на підключення до БД, що можна побачити в повному коді файлу в додатку А.

Саме підключення відбувається за допомогою допоміжного класу, де створюється єдиний об'єкт підключення до БД, після чого він використовується кожен раз коли сервер робить запит до БД. Таким чином цей клас гарантує, що буде існувати лише одне підключення для всіх запитів, що дасть змогу не перевантажувати БД постійно створюваними підключеннями.

```

class DatabaseInstance {
  database: Database;

  constructor() {
    this.database = new Database({
      url: process.env.DATABASE_URL,
      dbName: process.env.DATABASE_NAME,
      auth: {
        username: process.env.DATABASE_USERNAME,
        password: process.env.DATABASE_PASSWORD,
      },
    } as Config);
  }

  async checkConnection() {
    return await this.database.exists();
  }
}

export const databaseInstance = new DatabaseInstance();

```

Реалізація логування було здійснено за допомогою бібліотеки winston [16] яка надає можливість логування як в консоль так і в окремий файл для логів. Дана функція також дозволяє використовувати її напряду коли потрібно вивести певне службове повідомлення як можна було побачити в прикладі при перевірці підключення до БД.

		Заглада М.Р.			ІПЗ.КР.Б — 121 — 24 — ПЗ	Арк.
		Кубрак Ю.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		31

```

export const loggerInstance = createLogger({
  level: process.env.NODE_ENV == 'development' ? 'debug' : 'info',
  transports: [
    new transports.File({
      filename: path.resolve(__dirname, '../app.log'),
    }),
    new transports.Console(),
  ],

  format: format.combine(
    format.colorize(),
    format.timestamp(),
    format.printf(({ timestamp, level, message }) => {
      return `[${timestamp}] ${level}: ${message}`;
    }),
  ),
});

```

Але найчастіше вона використовується в проміжній функції (middleware) для логування кожного запиту до серверу, де вказується час коли він відбувся, шлях запиту, HTTP-код статусу а також при помилці відповідь сервера з помилкою та час того скільки зайняла обробка запиту.

```

export async function logger(
  ctx: Context,
  next: () => Promise<any>,
): Promise<void> {
  const start = new Date().getTime();
  try {
    await next();
  } catch (err: any) {
    ctx.status = err.status || 500;
    ctx.body = err.message;
  }
  const ms = new Date().getTime() - start;

  let logLevel: string;
  if (ctx.status >= 400) {
    logLevel = 'error';
  } else {
    logLevel = 'info';
  }

  const msg = `${ctx.method} ${ctx.originalUrl} ${ctx.status}${logLevel} ===
  'error' ? ` ${ctx.response.body}` : ''
  } ${ms}ms`;
  loggerInstance.log(logLevel, msg);
}

```

Для автентифікації користувачів використовуються JSON веб токени (JWT) які передаються через HTTP заголовок (header) який називається Authorization. Для автоматичного парсингу цих токенів була використана

		Заглада М.Р.			ІПЗ.КР.Б — 121 — 24 — ІЗ	Арк.
		Кубрак Ю.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		32

бібліотека koa-jwt [17] яка передається в якості проміжної функції (middleware) та автоматично робить розшифровані дані з токена доступними в контексті запиту. Для генерації та ручного парсингу токенів використовується допоміжний клас. Ручний парсинг використовується у випадках коли автентифікація є опціональною, наприклад при завантаженні файлу на сервер користувач може опціонально увійти в свій обліковий запис, для того щоб кожен файл який він завантажує був прив'язаний саме до нього.

```
export class AuthUtils {
  public static verifyToken(ctx: Context, type: string) {
    try {
      switch (type) {
        case 'auth':
          return jsonwebtoken.verify(
            ctx.get('Authorization').slice(7) as string,
            process.env.JWT_SECRET as string,
          );
        case 'refresh':
          return jsonwebtoken.verify(
            ctx.request.body.refreshToken,
            process.env.JWT_SECRET as string,
          );
        default:
          ctx.throw(400, 'invalid token type');
      }
    } catch (err: any) {
      ctx.throw(400, err.message);
    }
  }

  public static generateToken(payload: object, expiresIn: string): string {
    return jsonwebtoken.sign(payload, process.env.JWT_SECRET as string, {
      algorithm: 'HS512',
      expiresIn: expiresIn,
    });
  }
}
```

Після надходження запиту від користувача, роутер викликає метод відповідного контролера який був прив'язаний до цього шляху. Самі контролери майже не містять коду але в них розписана документація всіх шляхів по якими користувач може звернутись. Реалізована вона було за допомогою Swagger [18] що визначає стандарти специфікації документації для API, а також бібліотеки koa-swagger-decorator [19] яка дозволяє описувати методи та класи за допомогою декораторів. Повний код контролеру UserController можна побачити в додатку А.

```
@tagsAll(['User'])
```

		Заглада М.Р.			ІПЗ.КР.Б — 121 — 24 — ІЗ	Арк.
		Кубрак Ю.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		33

```

export class UserController {
  @request('post', '/auth/register')
  @summary('Register user')
  @description('Registers user based on provided info')
  @body(userSchema)
  @responses({
    201: { description: 'registration successful' },
    400: { description: 'uploaded data has validation errors' },
    500: { description: 'something went wrong on the server' },
  })
  public static async register(ctx: Context): Promise<void> {
    ctx.body = await UserService.register(ctx);
  }
  ...

  @request('get', '/user/profile')
  @summary('User profile')
  @description('Get user profile')
  @middlewares([auth])
  @responses({
    200: { description: 'user profile request successful' },
    400: { description: 'user with this email doesnt exist' },
    401: { description: 'authentication required' },
    500: { description: 'something went wrong on the server' },
  })
  public static async getProfile(ctx: Context): Promise<void> {
    ctx.body = await UserService.getProfile(ctx);
  }
  ...

```

Кожен раз коли користувач робить запит де потрібно ввести певні дані то вони валідуються як на клієнті так і на сервері, для того щоб мінімізувати ризик потрапляння неправильних даних до БД. Для валідації було використано бібліотеку Joi [20] яка дозволяє задавати схеми для валідації об'єктів, як наприклад показано нижче.

```

export const userLoginValidatorSchema = Joi.object({
  email: Joi.string()
    .email({
      minDomainSegments: 2,
    })
    .required(),

  password: Joi.string().min(7).max(100).required(),
});

export const userValidatorSchema = userLoginValidatorSchema.keys({
  username: Joi.string().alphanum().min(3).max(30).required(),
});

export const userPasswordEditValidatorSchema = Joi.object({
  oldPassword: Joi.string().min(7).max(100).required(),

  password: Joi.string().min(7).max(100).required(),
});

```

		Заглада М.Р.			ІПЗ.КР.Б — 121 — 24 — ІЗ	Арк.
		Кубрак Ю.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		34

```
});
```

Після чого валідацію об'єктів можна проводити асинхронно як показано на прикладі реєстрації користувача. Повний код сервісу користувача **UserService** наведений в додатку А.

```
public static async register(ctx: Context): Promise<string> {
  const userTemplate: IUserDto = {
    username: ctx.request.body.username,
    email: ctx.request.body.email,
    password: ctx.request.body.password,
  };

  try {
    const user = await userValidatorSchema.validateAsync(userTemplate);
    user.password = await argon2.hash(user.password);

    await usersCollection.save(user);
    ctx.status = 201;
    return `registration successful`;
  } catch (err: any) {
    if (err instanceof ValidationError) {
      ctx.throw(400, err.message);
    } else if (err.message.includes('unique constraint violated')) {
      ctx.throw(400, 'user with this email already exist');
    } else {
      ctx.throw(500, err.message);
    }
  }
}
```

Для парсингу файлів які завантажує користувач на сервіс було використано бібліотеку **formidable** [21] через те що вона підтримує завантаження файлів через потоки (**Stream**) [22] що дає змогу завантажувати файли на сервер частинами та одразу робити операції над ними навіть коли вони ще повністю не завантажились. Використання потоків в даному випадку дозволяє значно зменшити використання пам'яті, зберігаючи файли безпосередньо на диск під час завантаження. Також вона пропонує широкий спектр налаштувань для точного налаштування процесу завантаження, таких як обмеження розміру файлів, обробка помилок та фільтрація файлів за їх типом.

Дана бібліотека використовується в проміжній функції для автоматичного парсингу та зберігання файлу на диск.

```
export async function upload(ctx: Context, next: () => Promise<any>) {
  await checkUploadLimit(ctx);
  const form = formidable({
    uploadDir: path,
```

```

maxFiles: 1,
maxFileSize: Number(process.env.MAX_FILE_SIZE),
maxFields: 3,
hashAlgorithm: process.env.HASH_ALGORITHM,
keepExtensions: true,
filename: (name, ext) => {
  return `${name}-${Date.now()}-${Math.round(Math.random() * 1e9)}${ext}`;
},
filter: ({ mimetype }): boolean => {
  try {
    fileTypeValidatorSchema.validateAsync({ type: mimetype });
    return true;
  } catch (err: any) {
    ctx.throw(400, err.message);
  }
},
});
try {
  if (Number(ctx.get("content-length")) > Number(process.env.MAX_FILE_SIZE))
    ctx.throw(413, "File too large");
  const [fields, files] = await form.parse(ctx.req);
  if (!files?.file?.length) {
    ctx.throw(400, "no file was provided");
  }
  ctx.request.body.fields = fields;
  ctx.request.body.file = files!.file![0];
  await next();
} catch (err: any) {
  ctx.throw(err.statusCode || err.status || 500, err.message);
}
}
}

```

Далі розглянемо сам процес завантаження файлів. Спочатку відбувається валідація самого файлу: чи припустимий в нього розмір та розширення і потім він зберігається на диск. Далі метод сервісу створює шаблон для завантаження файлу, збираючи інформацію про рівень доступу та пароль файлу з тіла запиту якщо він є. Потім відбувається валідація вхідних даних, де перевіряється чи вірний тип рівня доступу і якщо він дорівнює «Захищений паролем» то додатково перевіряє наявність паролю. Якщо рівня доступу позначений як приватний, але користувач не авторизований, то користувачу повертається помилка 401 з проханням увійти в свій обліковий запис, оскільки якщо файл приватний то тільки сам користувач та користувачі з якими він поділився цим файлом можуть його переглядати, але якщо він буде не авторизованим то файл не буде доступний нікому.

Далі підготовлюються метадані файлу, такі як ім'я, шлях, рівень доступу, хеш пароля, дати завантаження та видалення, кількість переглядів та завантажень, а також допоміжне поле для тегу автентифікації для шифрування. Метадані файлу зберігаються в колекцію, після чого файл стискається та шифрується за допомогою пароля користувача або новоствореного ключа файлу, і зберігається тег автентифікації до цього шифру.

Якщо користувач авторизований, то відбувається прив'язка файлу з користувачем, де також зазначається, що користувач є власником цього файлу. Після цього встановлюється статус відповіді 201 і повертається ідентифікатор файлу за яким його можна отримати.

У випадку виникнення помилки, файл видаляється, і відправляється відповідний статус помилки з повідомленням.

```
public static async upload(ctx: Context): Promise<string> {
    const fileTemplate: IFileUploadDto = {
        privacy: ctx.request.body.fields.privacy[0],
        password: ctx.request.body.fields.password
            ? ctx.request.body.fields.password[0]
            : undefined,
    };

    try {
        await fileValidatorSchema.validateAsync(fileTemplate);
        if (fileTemplate.privacy === PrivacyType.Private && !ctx.state.user) {
            ctx.throw(401, "please log in to use this visibility modifier");
        }

        const date = new Date(ctx.request.body.file.lastModifiedDate);
        date.setDate(date.getDate() + 30);
        const fileDto: IFileDto = {
            name: ctx.request.body.file.originalFilename,
            path: `${ctx.request.body.file.filepath}.enc`,
            privacy: fileTemplate.privacy,
            password: fileTemplate.password
                ? await argon2.hash(fileTemplate.password)
                : "",
            uploadDate: ctx.request.body.file.lastModifiedDate.toISOString(),
            deletionDate: date.toISOString(),
            viewCount: 0,
            downloadCount: 0,
        };

        const file = await filesCollection.save(fileDto, { returnNew: true });
        let authTag: string;
        if (fileDto.password) {
            authTag = await EncryptionUtils.encryptFile(
                ctx.request.body.file.filepath,
```

```

        fileTemplate.password!,
    );
} else {
    authTag = await EncryptionUtils.encryptFile(
        ctx.request.body.file.filepath,
        file.new._key,
    );
}
await filesCollection.update(file.new._key, { authTag: authTag });

if (ctx.state.user) {
    const user = await UserService.getByEmail(ctx.state.user.email);
    if (user)
        await userFilesCollection.save({
            _from: user._id,
            _to: file._id,
            owner: true,
        });
}

ctx.status = 201;
return file.new._key;
} catch (err: any) {
    await unlink(ctx.request.body.file.filepath);
    if (err instanceof ValidationError) {
        ctx.throw(400, err.message);
    } else {
        ctx.throw(err.status || 500, err.message);
    }
}
}
}

```

Для отримання файлів існує 2 типи методів: отримання інформації про файл та отримання самого файлу. В даному випадку ми сфокусуємось на отриманні самого файлу оскільки ці типи методів майже не відрізняються між собою окрім інформації, що вони повертають.

Коли користувач надсилає запит на завантаження файлу то викликається відповідний метод сервісу. Спочатку він намагається асинхронно отримати документ з колекції файлів за переданим ідентифікатором. Якщо файл не знайдено, генерується помилка з кодом 404 про відсутність даного файлу.

Далі йде перевірка рівня доступу файлу. Якщо файл має публічну приватність, збільшується лічильник завантажень файлу і повертається потік для читання файлу де відбувається розшифрування та розтискання файлу, а також встановлення відповідних HTTP заголовків для клієнта, щоб він міг почати завантаження файлу.

Якщо файл має рівень доступу «Захищений паролем» то генерується помилка з кодом 401, оскільки для завантаження таких файлів існує окремий метод де користувачу потрібно додатково надати пароль до файлу.

У випадку якщо файл є приватним перевіряється, чи є користувач автентифікованим, збереженим у стані контексту. Якщо користувач не автентифікований, генерується помилка з кодом 401 з проською автентифікації. Далі отримується користувач за електронною адресою, збереженою у стані контексту, і перевіряється, чи володіє він файлом через репозиторій файлів користувача. Якщо користувач володіє файлом, збільшується лічильник завантажень файлу і повертається потік для читання файлу за такою ж схемою як і у випадку публічного файлу. Якщо користувач не має доступу до файлу, генерується помилка з кодом 403. Якщо приватність файлу має некоректне значення, генерується помилка з відповідним повідомленням.

У випадку виникнення будь-якої іншої помилки в процесі виконання функції, генерується помилка з кодом 500 або іншим відповідним кодом і повідомленням.

```
public static async downloadById(ctx: Context): Promise<ReadStream> {
  try {
    const file = await filesCollection.document(ctx.params.id, {
      graceful: true,
    });
    if (!file) ctx.throw(404, "file doesnt exist");

    switch (file.privacy) {
      case PrivacyType.Public:
        await filesCollection.update(file._key, {
          downloadCount: ++file.downloadCount,
        });
        ++file.downloadCount;
        return await this.getFile(ctx, file, file._key);

      case PrivacyType.Protected:
        ctx.throw(401, "this file is password protected");

      case PrivacyType.Private: {
        if (!ctx.state.user)
          ctx.throw(401, "please log in to view this file");

        const user = await UserService.getByEmail(ctx.state.user.email);
        if (await UserFilesRepository.checkIfOwns(user._id, file._id)) {
          await filesCollection.update(file._key, {
            downloadCount: ++file.downloadCount,
          });
        }
      }
    }
  }
}
```

```

    });
    ++file.downloadCount;
    return await this.getFile(ctx, file, file._key);
  }

  ctx.throw(403, "this file visibility is set to private");
}

default:
  ctx.throw("broken file visibility setting");
}
} catch (err: any) {
  ctx.throw(err.status || 500, err.message);
}
}
}

```

Для шифрування/розшифрування використовується допоміжний клас з відповідними методами. Дана функція виконує шифрування файлу з використанням алгоритму AES-256-GCM та пароля, наданого користувачем.

Даний шифр був обраний через низку переваг:

- безпека: AES-256-GCM забезпечує високий рівень захисту даних завдяки використанню ключів довжиною 256 біт. Це значно ускладнює злом алгоритму за допомогою грубої сили, оскільки кількість можливих ключів дуже велика;
- продуктивність: Шифрування за допомогою AES-256-GCM є дуже ефективним, особливо на сучасних процесорах, які мають апаратну підтримку для AES-інструкцій. Це забезпечує швидке шифрування та дешифрування, що є важливим для додатків, які обробляють великі обсяги даних у реальному часі;
- автентифікація: AES-256-GCM поєднує шифрування з механізмом автентифікації, що гарантує цілісність і достовірність даних. Це дозволяє виявляти будь-які зміни або підробки даних, що особливо важливо для захисту від атак, таких як підміна повідомлень;
- захист від атак: AES-256-GCM є стійким до багатьох видів атак, включаючи атаки на цілісність повідомлень. Механізм аутентифікації забезпечує додатковий захист від спроб підробки або модифікації даних;
- широке використання: AES-256-GCM є стандартом де-факто для багатьох протоколів безпеки, таких як TLS (Transport Layer Security) і IPsec

		Заглада М.Р.			ІПЗ.КР.Б — 121 — 24 — ПЗ	Арк.
		Кубрак Ю.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		40

(Internet Protocol Security). Це забезпечує високу сумісність з багатьма програмними і апаратними засобами.

В самому метод спочатку генерується випадковий вектор ініціалізації розміром 16 байтів як набір ентропії для шифру. Далі створюється ключ шифрування на основі пароля користувача або ідентифікатора файлу, використовуючи метод `generateKey` який перетворює рядковий ключ в байтовий хеш для подальшого шифрування. Створюється потік для читання вхідного файлу за вказаним шляхом і паралельно створюється потік для стиснення файлу у форматі `gzip`. Після цього створюється шифрувальний потік з використанням алгоритму, ключа та вектора ініціалізації. Вектор ініціалізації за допомогою додаткового класу `AppendIv` додається на початок файлу. Потім створюється потік для запису зашифрованих даних у новий файл з розширенням `".enc"`.

Далі відбувається послідовне з'єднання потоків - читання файлу, стиснення, шифрування, додавання IV та запису зашифрованих даних у новий файл. Після успішного шифрування виконується видалення оригінального файлу. На завершення, метод `getAuthTag` повертає тег аутентифікації шифрування у вигляді рядка у форматі `hex` який потім буде збережено в БД. Повний код даного класу можна знайти в додатку А.

```
public static async encryptFile(filePath: string, password: string) {
    const algorithm: CipherGCMTypes = "aes-256-gcm";
    const iv = randomBytes(16);
    const key = this.generateKey(password);

    const encryptStream = createReadStream(filePath);
    const gzip = createGzip();
    const cipher = createCipheriv(algorithm, key, iv);
    const appendIv = new AppendIv(iv);
    const writeStream = createWriteStream(path.join(filePath + ".enc"));

    await StreamPromises.pipeline(
        encryptStream,
        gzip,
        cipher,
        appendIv,
        writeStream,
    );
    await unlink(filePath);
    return cipher.getAuthTag().toString("hex");
}
```

		Заглада М.Р.			ІПЗ.КР.Б — 121 — 24 — ПЗ	Арк.
		Кубрак Ю.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		41

Розглянемо, що відбувається коли користувач хоче поділитись файлом з іншим користувачем. Спочатку метод отримує email з тіла запиту і створює шаблон для користувача з ким треба поділитись файлом. Робиться виклик до сервісу користувача, щоб отримати користувача за його email. Далі відбувається валідація отриманого email та перевірка на валідність виклику де перевіряється чи є користувач власником файлу.

Далі відбувається виклик на отримання файлу користувача і потім перевіряється його наявність. Якщо файл не знайдено, генерується помилка з кодом 404. Також перевіряється, чи не спробує користувач поділитися файлом сам собі. Якщо співпадають ідентифікатори поточного користувача та користувача, якому потрібно поділитися файлом, генерується помилка з кодом 400. Далі перевіряється, чи не поділився вже користувач цим файлом з цим користувачем. Якщо такий користувач знайдений, генерується помилка з кодом 400.

Якщо всі перевірки пройдені успішно, функція зберігає в БД зв'язок цього файлу і користувача з приміткою того що він не є його власником. Видалення статусу, що користувач поділився файлом з цим користувачем відбувається за аналогічним принципом. Повний код даного сервісу можна знайти в додатку А.

```
public static async shareFile(ctx: Context): Promise<string> {
    const emailTemplate: IShareFileDto = {
        email: ctx.request.body.email,
    };

    try {
        const user = await UserService.getByEmail(ctx.state.user.email);
        await shareFileEmailValidatorSchema.validateAsync(emailTemplate);
        await this.checkValidity(ctx, user._id, ctx.params.id, true);
        const file = await UserFilesRepository.getFile(
            user._id,
            `Files/${ctx.params.id}`,
        );
    }
    if (!file) {
        ctx.throw(404, "file not found");
    }

    const sharedToUser = await UserService.getByEmail(emailTemplate.email);
    if (user._id === sharedToUser._id) {
        ctx.throw(400, "you cant share files to yourself");
    }
}
```

```

const sharedToUsers = (
  await UserFilesRepository.getSharedToUsers(user._id, file._id)
)[0];
if (sharedToUsers?.includes(sharedToUser.email)) {
  ctx.throw(400, "you already shared this file to this user");
}

await userFilesCollection.save({
  _from: sharedToUser._id,
  _to: file._id,
  owner: false,
  grantedBy: user.username,
});

return `successfully shared file ${file._key} to user $
{sharedToUser._key}`;
} catch (err: any) {
  if (err instanceof ReferenceError) {
    ctx.throw(400, err.message);
  }
  ctx.throw(err.status || 500, err.message);
}
}

```

Для видалення застарілих файлів використовується окрема функція яка перевіряє дати записані в імені файлу та БД.

```

async function clearOldFiles() {
  try {
    const files = await readdir(path);
    let removed = 0;

    const re = /^.+-(\d+)-/;
    for (const file of files) {
      const date = re.exec(file);
      const dateCheck = new Date(+date[1]);
      dateCheck.setDate(dateCheck.getDate() + 30);

      if (dateCheck.getTime() < Date.now()) {
        await unlink(`${path}/${file}`);
        ++removed;
      }
    }
    console.log(`Removed ${removed} old files`);
  } catch (err) {
    console.log(err);
  }
}

await clearOldFiles();

```

Для автоматичного виконання цієї функції було використано програму `fcron` [23], що базується на специфікації `cron`, що являє собою систему універсального планування завдань у UNIX-подібних операційних системах.

		Заглада М.Р.			ІПЗ.КР.Б — 121 — 24 — ПЗ	Арк.
		Кубрак Ю.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		43

Для того щоб дана функція виконувалась кожну годину потрібно прописати таку конфігурацію: @hourly node clearOldFiles.js, після чого скрипт буде сам виконувати очищення старих файлів.

Також для неавторизованих користувачів діє метод по обмеженню завантажень файлів на день. Спочатку визначається IP за допомогою бібліотеки request-ip [24] користувача та потім перевіряється відповідний запис з такою адресою на те скільки за нею вже було завантажено файлів.

```
export async function checkUploadLimit(ctx: Context) {
  const userIp = requestIp.getClientIp(ctx.request);
  const uploadLimitCollection =
    databaseInstance.database.collection("UploadLimit");

  if (userIp && !ctx.state.user) {
    const uploadLimitRecord: IUploadLimit =
      await uploadLimitCollection.document(userIp, {
        graceful: true,
      });

    if (uploadLimitRecord) {
      if (
        uploadLimitRecord.uploadCount >= 1 &&
        uploadLimitRecord.uploadCount < 3
      ) {
        await uploadLimitCollection.update(uploadLimitRecord._key, {
          uploadCount: uploadLimitRecord.uploadCount + 1,
        });
      } else {
        ctx.throw(429, "daily upload limit reached");
      }
    } else {
      const date = new Date();
      date.setDate(date.getDate() + 1);
      await uploadLimitCollection.save({
        _key: userIp,
        uploadCount: 1,
        expireAt: date.toISOString(),
      });
    }
  } else if (!ctx.state.user) {
    ctx.throw(400, "cant get request ip");
  }
}
```

Також для тестування функціоналу API присутній файл конфігурації Postman який можна знайти в додатку А.

Що до клієнтської частини, там було використано декілька бібліотек для покращення досвіду роботи користувача.

		Заглада М.Р.			ІПЗ.КР.Б — 121 — 24 — ПЗ	Арк.
		Кубрак Ю.О.				44
Змн.	Арк.	№ докум.	Підпис	Дата		

Nuxt Color Mode [25] — це бібліотека яка надає можливість зміни тем інтерфейсу таких як біла та темна. Також вона надає можливість зберігання значення теми в локальному сховищі браузера, що дозволяє йому одразу відмальовувати інтерфейс з потрібною темою коли користувач робить новий запит на сторінку.

Nuxt i18n [26] — бібліотека для інтернаціоналізації інтерфейсу. Вона надає можливість динамічно підставляти правильний переклад сторінки в залежності від того яку мову обрав користувач або обрав в самому браузері. Зберігання значення вибраної мови в куках дозволяє серверу одразу відмальовувати сторінку з правильною мовою не витрачаючи час користувача на повторне вибирання мови кожен раз коли він робить новий запит.

Turnstile [27] — сервіс який надає послуги перевірки користувачів на роботів і є альтернативою до Google ReCaptcha. Використовується коли користувач завантажує файл на сервер та на пристрій, а також коли він реєструється та входить в обліковий запис. Приклад використання наведений нижче.

```
<NuxtTurnstile
  v-model="turnstileToken"
  class="flex justify-center mt-3"
/>
```

Notivue [28] — система сповіщень для інтерфейсу. Вона використовується для індикації певних дій користувача, таких як копіювання посилання, реєстрація та зміна параметрів файлу. Приклад використання наведений нижче.

```
push.success({
  title: t("copySuccess"),
});
```

Nuxt Auth [29] — це офіційний модуль для автентифікації та авторизації в Nuxt.js, який значно спрощує реалізацію процесу менеджменту користувацьких сесій. Він підтримує JWT токенами для забезпечення безпечної автентифікації і авторизації. Також він надає можливість налаштовувати правила і поведінку сесій, такі як таймаут сесій, відновлення токенів і т. д., що дозволяє точно керувати безпекою і зручністю автентифікації.

Nuxt Fonts [30] — офіційний модуль для завантаження шрифтів з Інтернету на клієнт. Має різні налаштування що до вибору провайдерів шрифту та розумний підбір шрифту в залежності від роздільної здатності екрану кінцевого користувача.

Vee validate [31] — бібліотека для валідації форм на клієнтській частині. Використовується разом в поєднанні з Joi для простої валідації та виводу повідомлень про її помилки.

Auto animate [32] — бібліотека для автоматичної анімації компонентів які з'являються або зникають. Факт того, що її використання не потребує ніякої конфігурації робить її дуже привабливою для покращення досвіду користувача.

Nuxt social share [33] — невелика бібліотека яка автоматично генерує посилання на різні соціальні мережі. Використовується для швидких посилань за якими користувач може поділитись файлом.

Висновки до другого розділу

В ході виконання другого розділу було проаналізовано основні вимоги до системи та вимоги користувачів. На основі отриманих результатів було побудовано діаграму класів системи та діаграму варіантів використання.

Потім було спроектовано модель БД та було визначено основні колекції для зберігання даних.

Також була розроблена діаграми активності користувача яка показує загальний принцип роботи користувача з додатком, та діаграму діяльності для завантаження файлу на сервіс.

В кінці було представлено реалізацію самого сервісу, та подано пояснення для ключових фрагментів коду.

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З СЕРВІСОМ ДЛЯ ОБМІНУ ФАЙЛАМИ

3.1 Порядок встановлення та налаштування параметрів системи

Оскільки сервіс для обміну файлами використовує клієнт-серверну архітектуру то кожен його компонент потрібно розгортати окремо. Для простоти розуміння була побудована діаграма розгортання (рис. 3.1).

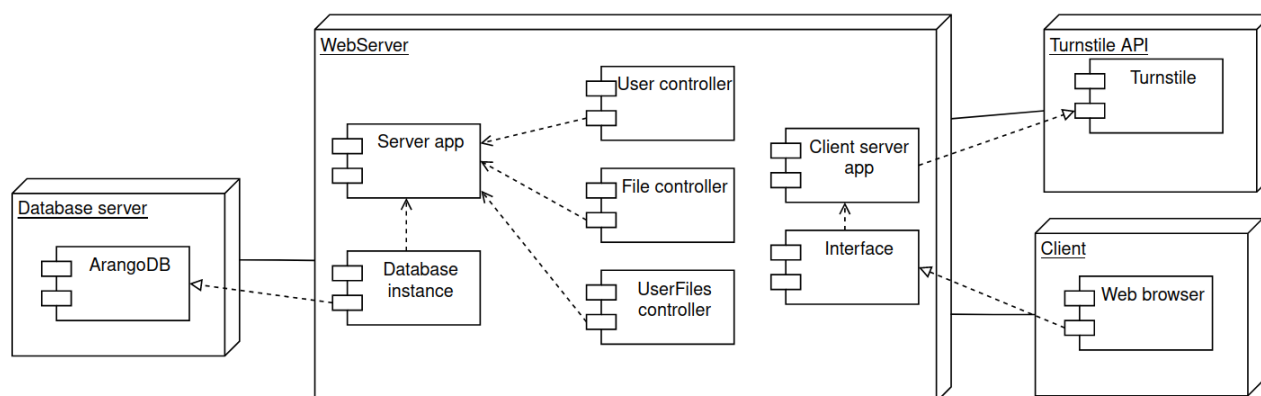


Рис. 3.1. Діаграма розгортання

На даній діаграмі зображені основні компоненти з яких складається сервіс: Вебсервер, сервер БД, API для сервісу Turnstile та кінцевого користувача.

Вебсервер тут є ключовим компонентом, оскільки він поєднує в собі функціонал всього сервісу, як клієнтську так і серверну частину. Для його розгортання потрібно встановити ОС Debian 12, потім встановити Node.js версії 20 та прописати в файлі середовища всі потрібні налаштування, такі як порти для сервера, БД та ін.

Сервер БД використовується для зберігання всіх даних які використовує сервіс. Підключення до нього відбувається через TCP/IP, що забезпечує цілісність та захист даних при передачі. Його розгортання відбувається через Dockerfile для того щоб забезпечити ізолюваність БД від зовнішньої системи, щоб уникнути можливості запитів ззовні до БД на пряму.

Turnstile API використовується як провайдер сервісу для перевірки користувачів на роботів. Даний сервіс належить Cloudflare і для його використання потрібно отримати токен який потрібно потім прописати в конфігурації середовища сервера.

Кінцевий користувач людина, яка взаємодіє безпосередньо з сервісом. Для роботи з сервісом кінцевий користувач повинен використовувати сучасний браузер, щоб отримати доступ до функцій, які надаються сервісом через API.

3.2 Структура інтерфейсу та порядок роботи

Після переходу на сервіс користувач буде бачити головну сторінку (рис. 3.2). На ній користувач може завантажити файл на сервіс, зареєструватись або увійти в свій обліковий запис.

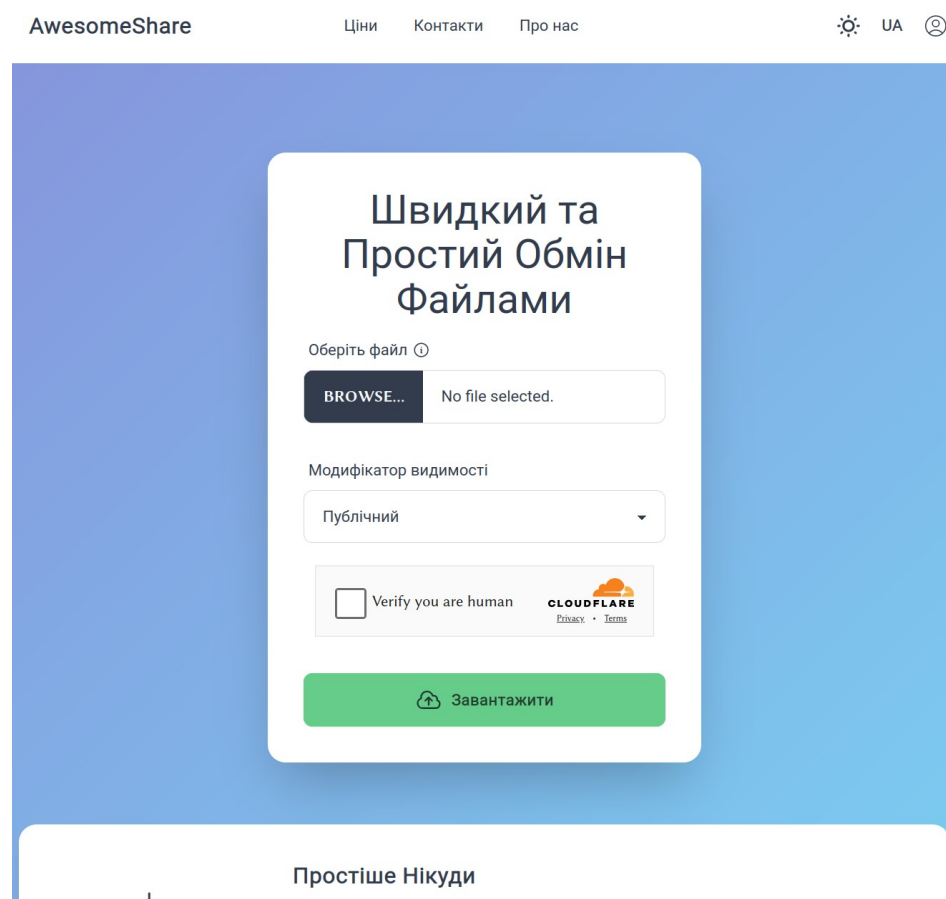


Рис. 3.2. Головна сторінка

В шапці сторінки присутні кнопки для зміни теми сторінки (рис. 3.3) та мови інтерфейсу (рис. 3.4) для зручності користування сервісом.

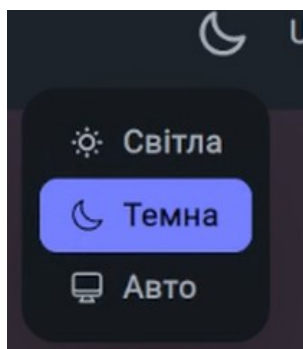


Рис. 3.3. Меню для вибору теми сторінки

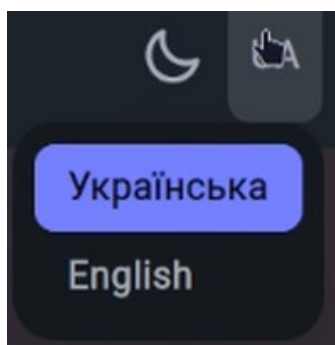


Рис. 3.4. Меню для вибору теми сторінки

Далі користувач може почати заповнювати форму для завантаження файлу на сервіс (рис 3.5).

Рис. 3.5. Заповнена форма для завантаження файлу

У випадку якщо користувач хоче захистити файл паролем, в формі з'являться додаткові поля для введення паролю до файлу (рис 3.6).

Рис. 3.6. Додаткові поля в формі якщо користувач хоче захистити файл паролем

Потім, якщо не було помилок валідації, файл завантажується на сервер та під формою з'являється секція де користувач може побачити посилання на його файл, QR код який він може відсканувати для швидкого завантаження файлу та різні посилання на соціальні мережі, натиснувши по яким він може з легкістю поділитись файлом з іншими людьми (рис 3.7).

Рис. 3.7. Успішне завантаження файлу

Далі користувач може перейти за цим посиланням де він може побачити загальну інформацію про файл, а також кнопку по якій почнеться завантаження файлу на пристрій користувача (рис 3.8).

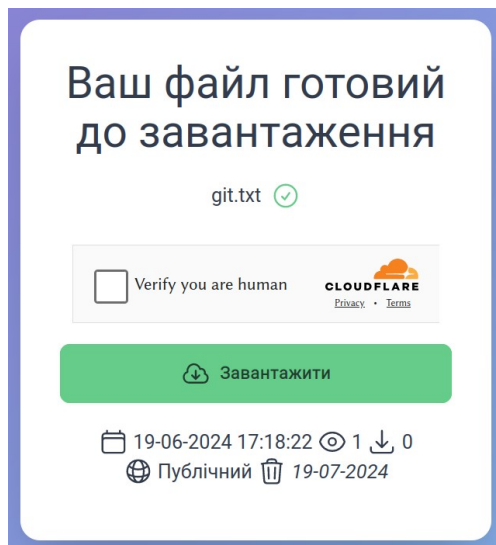


Рис. 3.8. Сторінка завантаження файлу на пристрій

Для доступу до повного функціоналу додатку користувачу потрібно зареєструватись, що він може зробити натиснувши на іконку профілю в шапці та обравши пункт «Реєстрація» (рис 3.9).

Рис. 3.9. Сторінка реєстрації

Після реєстрації користувач буде перенаправлений на сторінку входу в профіль, де йому потрібно заповнити дані які він використав під час реєстрації (рис 3.10).

Рис. 3.10. Сторінка реєстрації

Якщо він ввів свої дані вірно, то він буде перенаправлений на сторінку профілю, де він може побачити його дані та бічну панель для сторінок профілю, його файлів та зміни його інформації (рис 3.11).

Рис. 3.11. Сторінка профілю

На сторінці з його файлами користувач може побачити список усіх файлів які він завантажив на сервіс (рис 3.12). Самі файли представлені у вигляді карток та мають найбільш потрібну інформацію написану на них, такі як ім'я, дата видалення, рівень доступу та кнопка для швидкого отримання посилання на цей файл. Також зверху присутній фільтр (рис 3.13) та поле для пошуку за іменем файлів для зручності (рис 3.14).

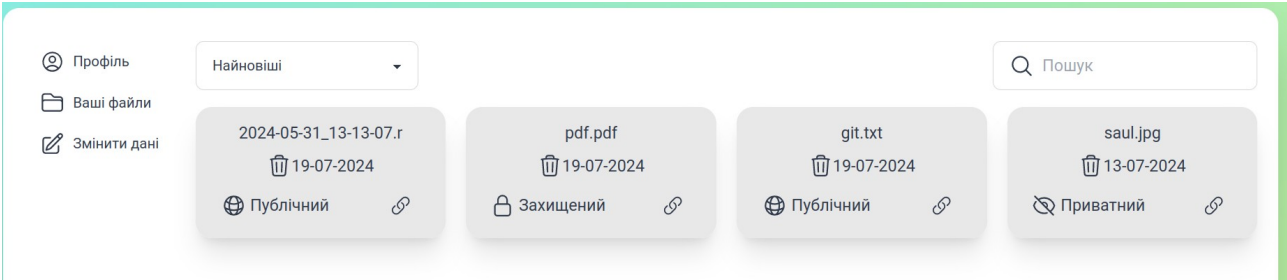


Рис. 3.12. Сторінка з файлами користувача

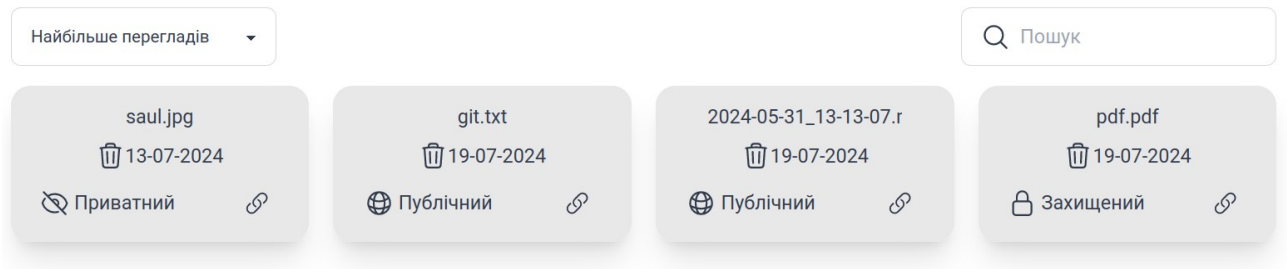


Рис. 3.13. Використання фільтра на сторінці з файлами користувача

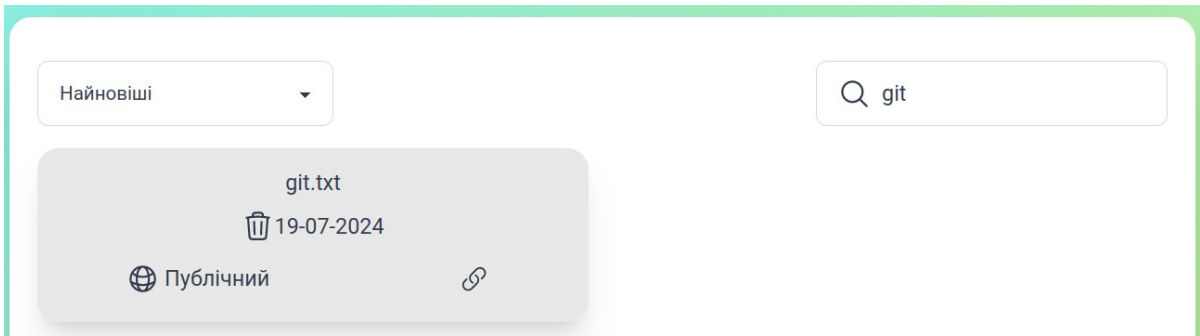


Рис. 3.14. Використання пошуку за назвою на сторінці з файлами користувача

Користувач може натиснути на картку файла, після чого відкриється бічна панель де він може переглянути повну інформацію про файл, а також секції з діями які він може виконувати над файлом (рис 3.15).

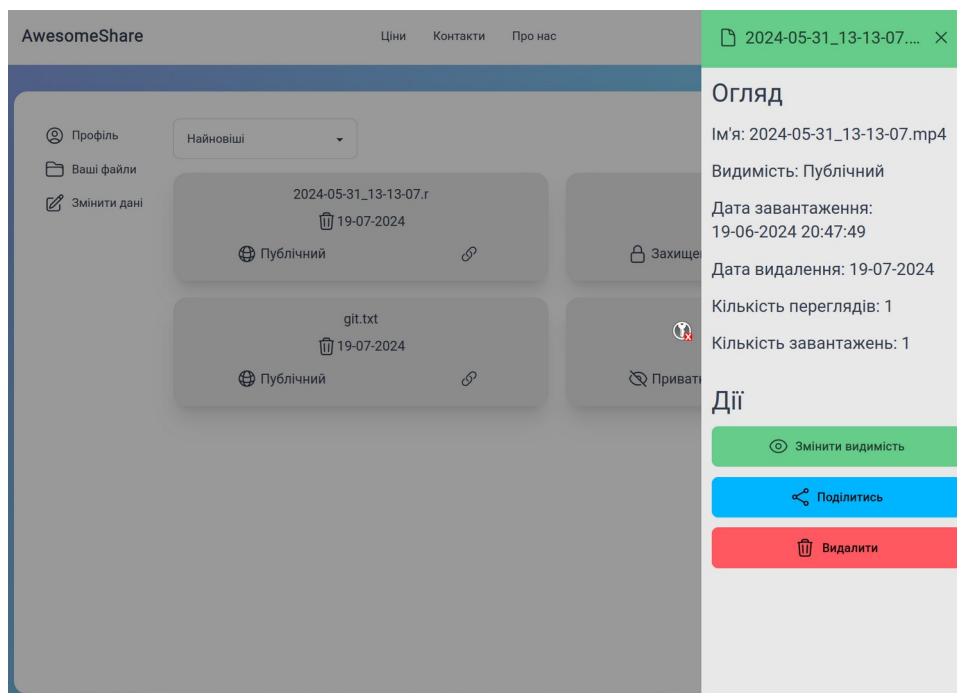


Рис. 3.15. Бічна панель з інформацією та діями над файлом

При натисканні на кнопку змінити видимість користувачу висвітиться вікно яке його попросить обрати новий рівень доступу до файлу. В списку доступних рівнів немає поточного рівню доступу і додатково якщо користувач обере «Захищений паролем» то з’явиться відповідні поля для вводу пароля (рис 3.16).

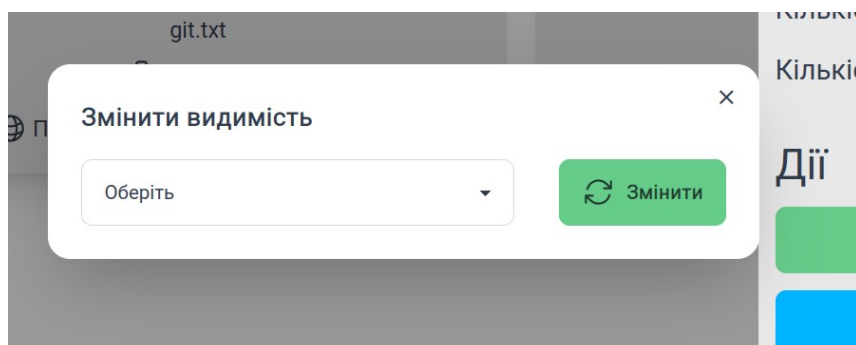


Рис. 3.16. Зміна рівню доступу до файлу в новому вікні

На тій же панелі, користувач може керувати тим хто має доступ до файлу натиснувши на кнопку «Поділитись». В новому вікні користувач може побачити форму де він може поділитись файлом з користувачем та трохи нижче список користувачів які мають доступ до цього файлу (рис 3.17).

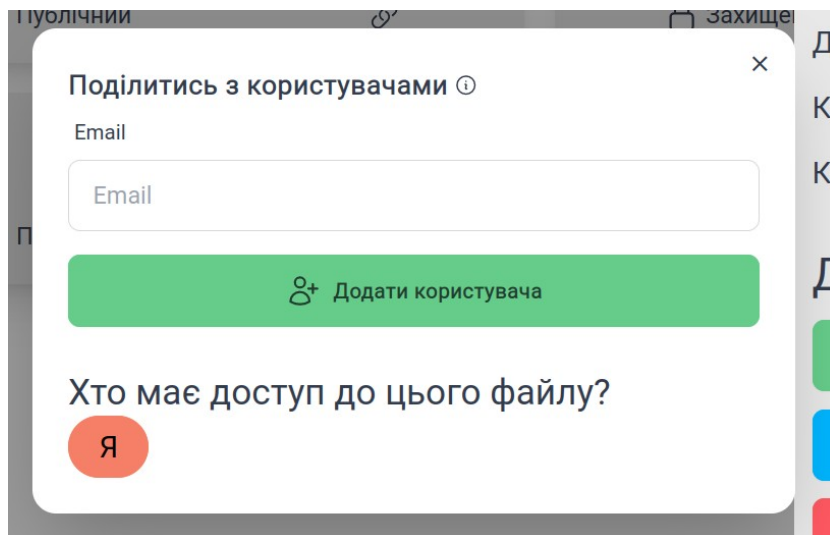


Рис. 3.17. Вікно для керування доступу до файлу

При успішному додаванні нового користувача він буде відображений знизу в секції «Хто має доступ до цього файлу» де біля його імені буде кнопка за якою його також можна легко видалити (рис 3.18).

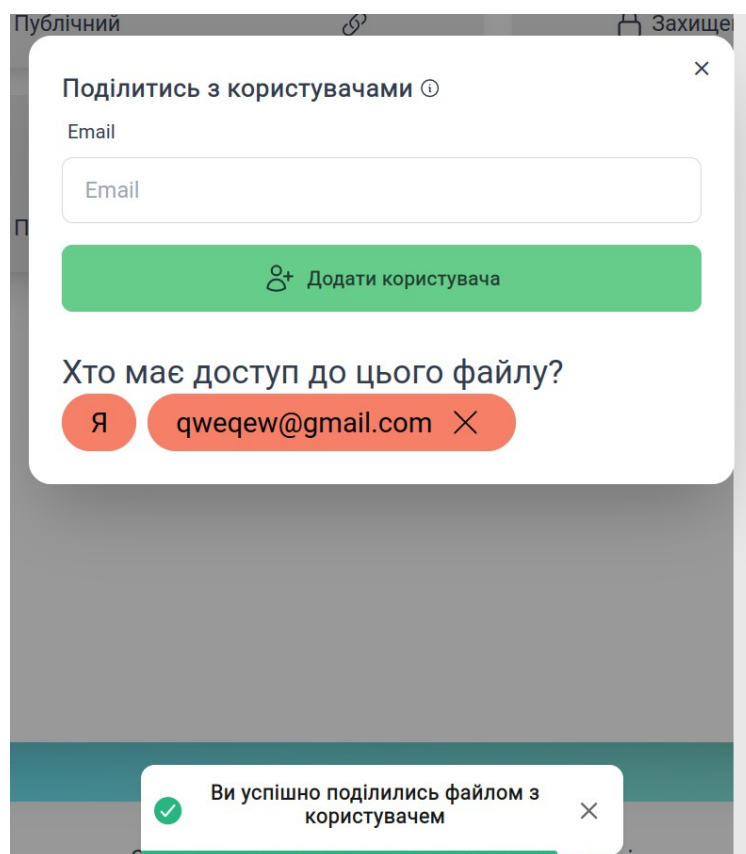


Рис. 3.18. Успішне додавання нового користувача

Для користувача файл який з ним поділився власник буде відображатись в спеціальній рамці (рис 3.19), а в бічній панелі замість дій буде виведена інформації про те що даний власник поділивсь з ним цим файлом (рис 3.20).

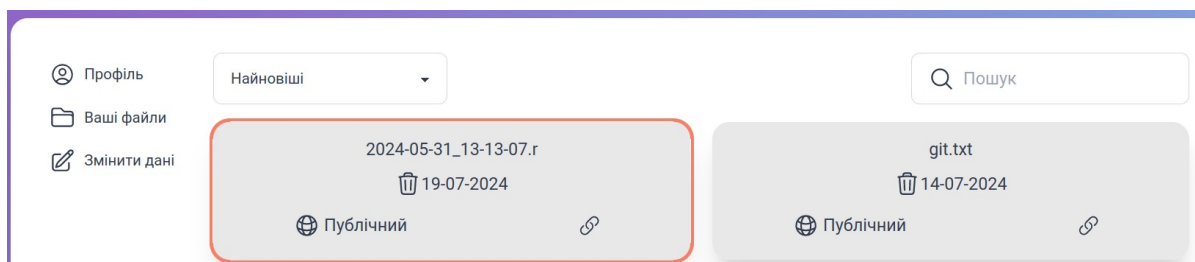


Рис. 3.19. Спеціальна рамка для картки файлу для користувачів з ким його поділились

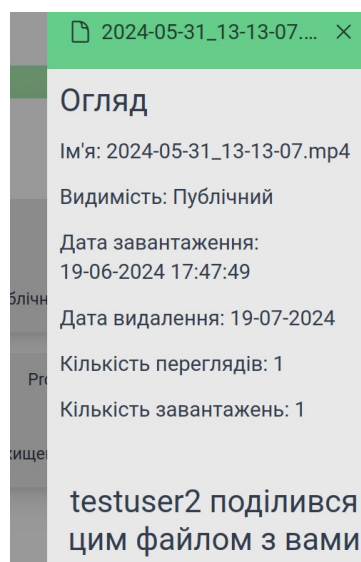


Рис. 3.20. Інформація про власника файлу для користувачів з ким його поділились

Якщо у файла рівень доступу «Захищений паролем» то в діях також додатково буде відображена опція для зміни поточного паролю. При натисканні відповідної кнопки, з’явиться нове вікно де користувачу потрібно ввести старий та новий пароль (рис 3.21).

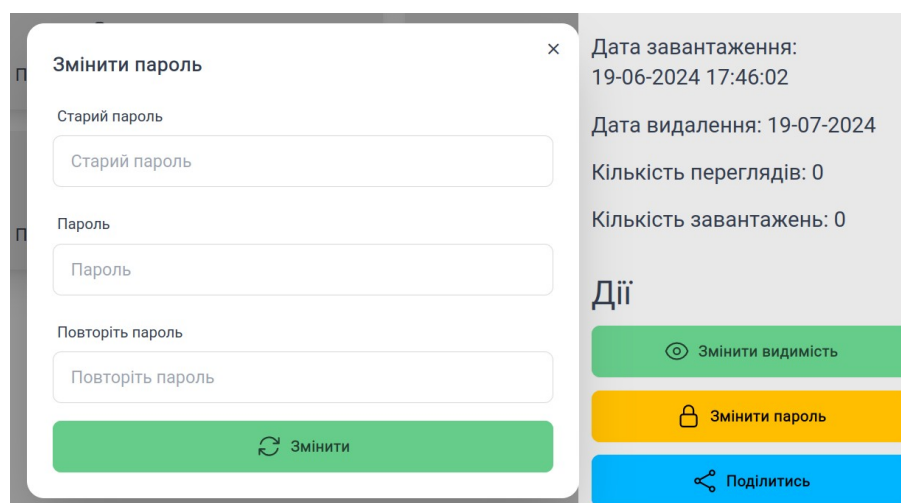


Рис. 3.21. Вікно для зміни паролю

Якщо користувач захоче видалити файл то він може це зробити натиснувши на кнопку «Видалити». Перед ним з'явиться вікно яке буде просити підтвердження на операцію і після нього файл повністю видаляється (рис 3.22).

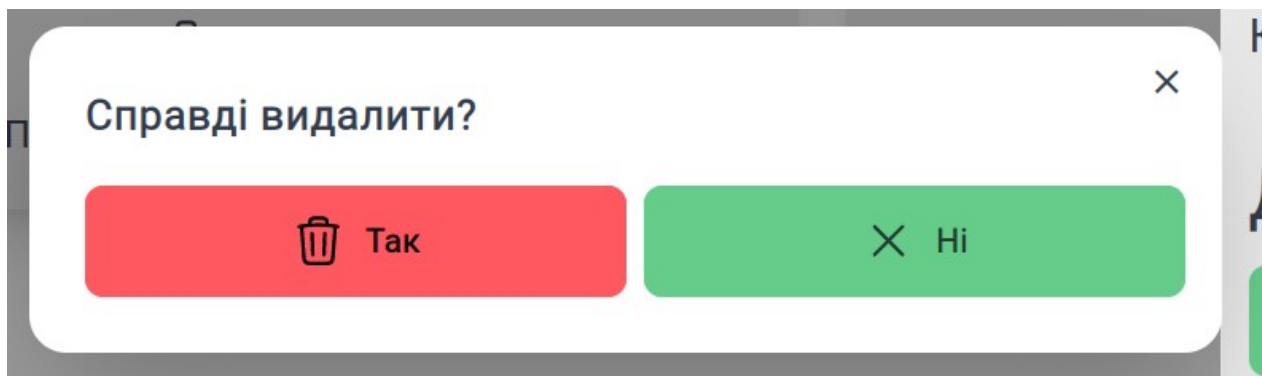


Рис. 3.22. Вікно для видалення файлу

На останок, якщо користувач захоче змінити свої дані то він може це зробити на вкладці «Змінити дані». На ній присутні три секції: змінити інформацію, змінити пароль та видалити профіль. Користувач може натиснути на будь-яку з опцій і під нею з'явиться відповідне вікно для дії (рис 3.23).

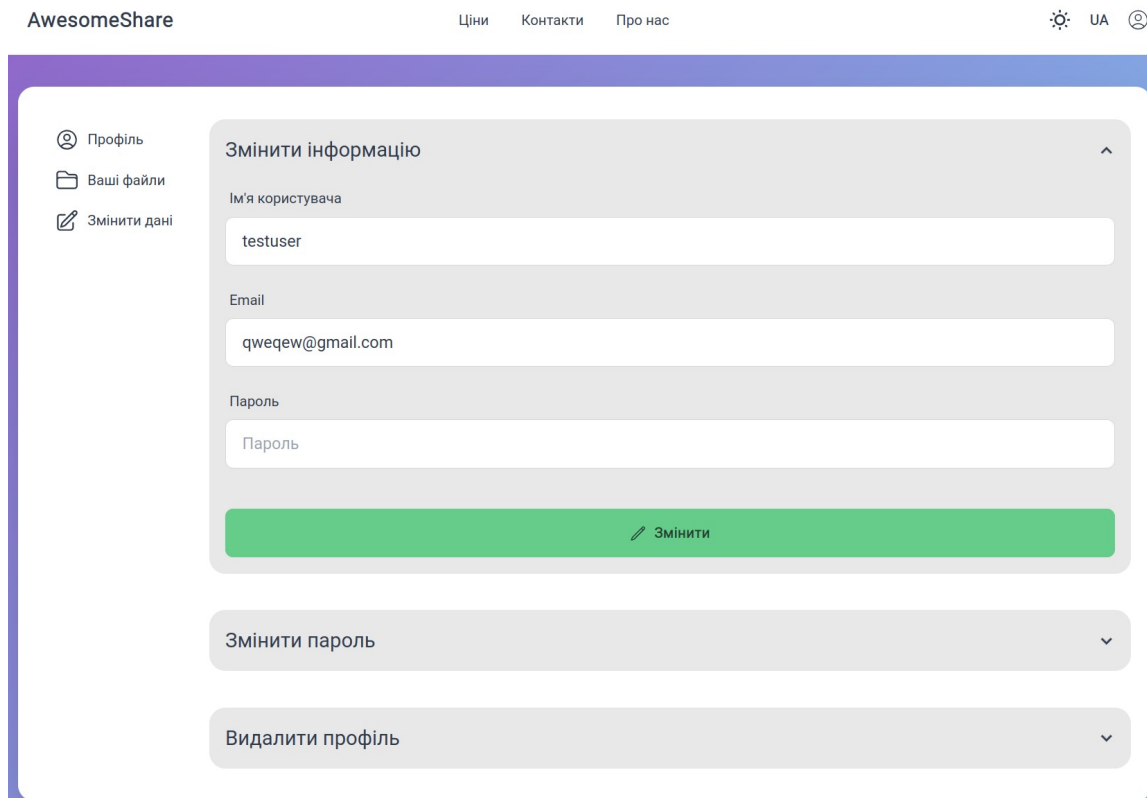


Рис. 3.23. Сторінка зміни даних користувача

3.3 Тестування роботи сервісу

Для тестування клієнтської частини було використане ручне тестування. Спочатку було протестовано адаптацію сторінки до різних розмірів екрану (рис 3.24).

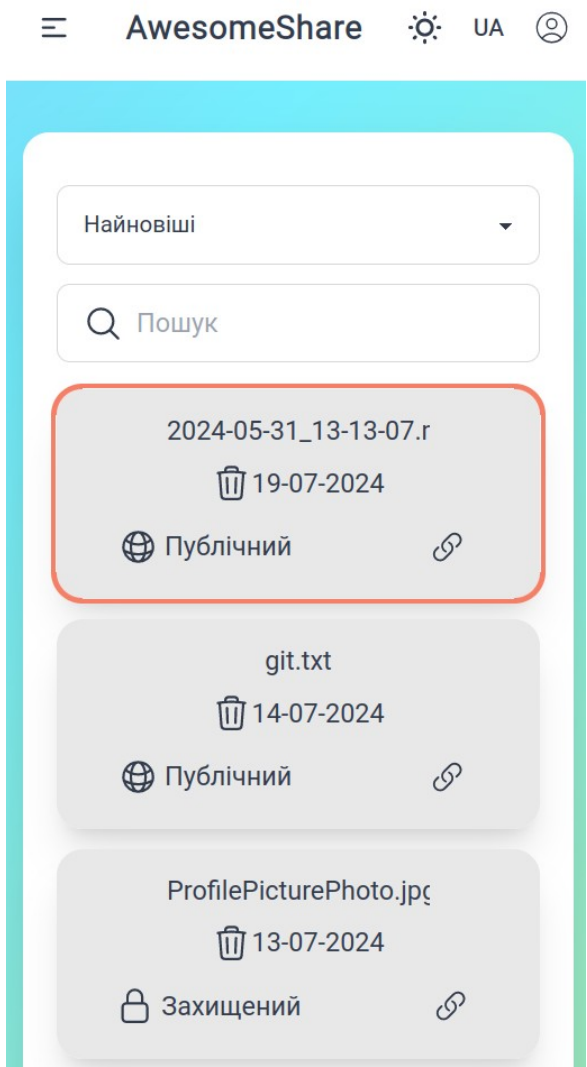


Рис. 3.24. Тестування роботи сторінки при малих розмірах екрану

Як можна побачити на рисунку, результат позитивний. Всі сторінки були розроблені з підходом до адаптації спочатку для мобільних пристроїв і потім до більших розмірів.

Для всіх даних які вводить користувач на сторінках відбувається валідація як на клієнті так і на сервері для того щоб запобігти потраплянню неправильних даних в систему. Наприклад, присутня валідація типу файла який завантажується (рис 3.25) та валідація для вводу електронної пошти (рис 3.26).

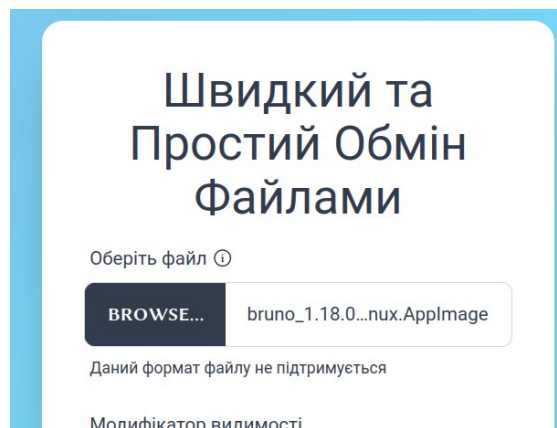


Рис. 3.25. Валідація типу файла

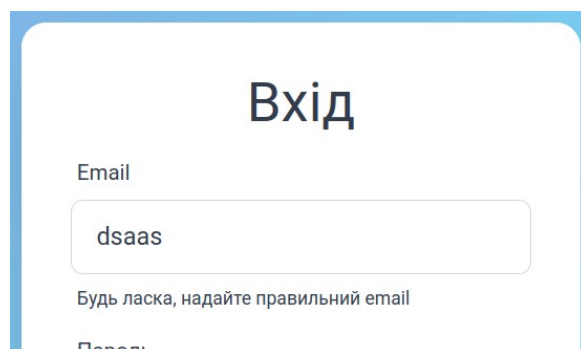


Рис. 3.26. Валідація електронної пошти

Також для кожної помилки яка відбувається на сервері, клієнт має відповідне вікно яке буде автоматично показуватись користувачу з поясненням того що сталося. Більшість з цих повідомлень мають конкретне пояснення для користувача того що саме пішло не так (рис 3.27), але в деяких випадках коли помилка є не стандартною, то для них є окреме повідомлення яке просто говорить, що щось пішло не так (рис 3.28).

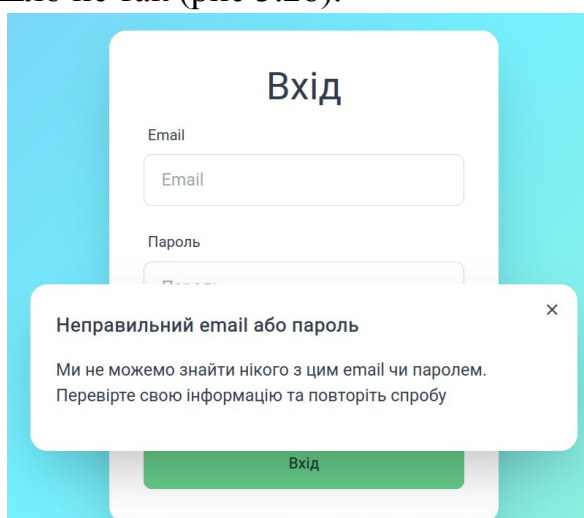


Рис. 3.27. Відображення конкретної помилки на клієнті

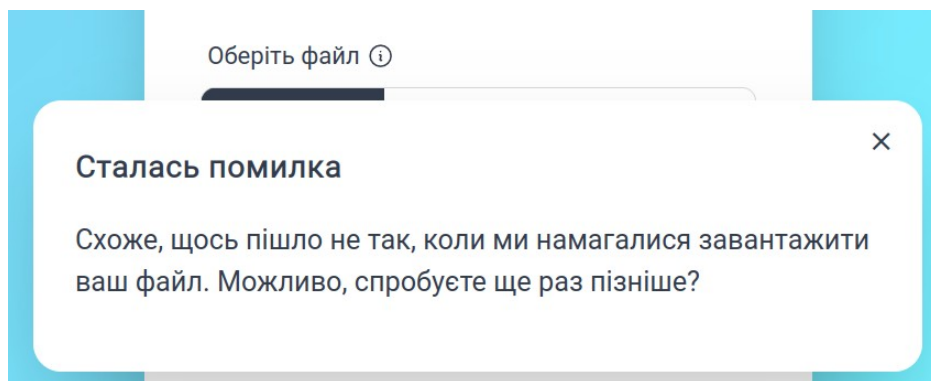


Рис. 3.28. Відображення загальної помилки на клієнті

При спробі користувача навігації на неіснуючу сторінку, існує спеціальна сторінка з відповідною помилкою. Вона також використовується коли відбувається якась помилка зі сторони сервера де буде відображено відповідну помилку користувачу (рис 3.29).

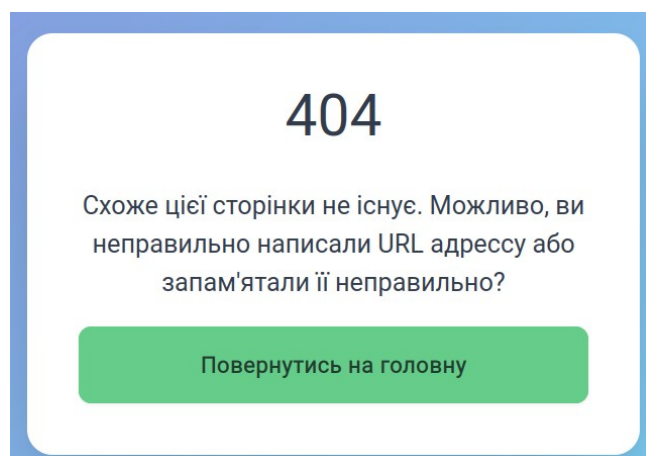


Рис. 3.29. Сторінка з помилкою

Для тестування серверної частини було написано пакет модульних тестів які покривають весь існуючий функціонал сервісу (рис 3.30).

```
==== Coverage summary ====
Statements   : 61.08% ( 350/573 )
Branches     : 70.75% ( 75/106 )
Functions    : 56.94% ( 41/72 )
Lines        : 60.57% ( 338/558 )

Test Suites: 17 passed, 17 total
Tests:       87 passed, 87 total
Snapshots:   0 total
Time:        14.632 s
Ran all test suites.
Done in 15.02s.
```

Рис. 3.30. Модульне тестування серверної частини

Висновки до третього розділу

В третьому розділі було представлено алгоритм та діаграму розгортання сервісу, де було детально описано основні компоненти системи та як їх конфігурацію.

Потім було описано інтерфейс сервісу, його основні компоненти та вигляд. Також було надано детальну інструкцію для користувача для типових дій в сервісі для забезпечення користування сервісу для різних типів користувачів.

Також було проведено низку тестів, які всі були пройдені успішно. Було доведено, що інтерфейс відповідає стандартам адаптивності під різні розміри екранів, має працюючу валідацію даних та коректну обробку різних помилок. Функціонал сервера було протестовано окремо за допомогою модульних тестів які було також успішно пройдені.

		Заглада М.Р.			ІПЗ.КР.Б — 121 — 24 — ПЗ	Арк.
		Кубрак Ю.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		61

ВИСНОВКИ

В першому розділі було проведено детальний аналіз поставленої задачі, проаналізовано конкурентів та обрано технології, які будуть використовуватись для розробки сервісу. Було розглянуто архітектури додатків та сучасні засоби для розробки сервісу. Особлива увага приділена архітектурі системи, яка передбачає використання клієнт-серверного підходу, що дозволяє легко масштабувати окремі компоненти сервісу. Використання даної архітектури сприяє легкості масштабування, проектування сервісу з урахуванням високих навантажень забезпечує надійність роботи, а впровадження чітко визначених API полегшує інтеграцію з іншими системами. На основі цього дослідження були визначені основні вимоги від користувачів до сервісу та обґрунтовано вибір засобів реалізації сервісу.

В другому розділі було проведено аналіз вимог користувачів, на основі якого було побудовано діаграму варіантів використання та описано основні компоненти сервера на основі яких було побудовано діаграму класів. Також було проведено проектування схеми зберігання даних в БД, описано основні процеси та дії користувача на відповідних діаграмах та описані основні частини коду сервісу.

В третьому розділі було описано процес розгортання сервісу, де визначено основні компоненти системи: серверну частину, клієнтську частину, базу даних, різні API та користувача який який буде взаємодіяти з ними. Також було проведено презентацію користувацького інтерфейсу та опис порядку роботи з сервісом. Інтерфейс орієнтований на зручність користувачів та забезпечує інтуїтивно зрозумілу навігацію. Продемонстровано функції завантаження, перегляду та обміну файлами, а також налаштування рівня доступу для забезпечення конфіденційності. В кінці було проведено тестування сервісу як на клієнті так і на сервері, які дали позитивні результати.

В результаті виконання кваліфікаційної роботи було спроектовано та розроблено сервіс для обміну файлами. Додаток має весь функціонал який потрібний користувачу, має зручний інтерфейс, використовує сучасні

		Заглада М.Р.			ІПЗ.КР.Б — 121 — 24 — ПЗ	Арк.
		Кубрак Ю.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		62

технології та відповідає стандартам захисту та продуктивності. Відповідність вимогам було протестовано як автоматично так і вручну.

		Заглада М.Р.			ІПЗ.КР.Б — 121 — 24 — ПЗ	Арк.
		Кубрак Ю.О.				63
Змн.	Арк.	№ докум.	Підпис	Дата		

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. What is REST? [Електронний ресурс] – Режим доступу до ресурсу: <https://restfulapi.net/>
2. WeTransfer [Електронний ресурс] – Режим доступу до ресурсу: <https://wetransfer.com/>
3. SendAnywhere [Електронний ресурс] – Режим доступу до ресурсу: <https://send-anywhere.com>
4. TransferNow [Електронний ресурс] – Режим доступу до ресурсу: <https://www.transfERNOW.net/en>
5. Node.js [Електронний ресурс] – Режим доступу до ресурсу: <https://nodejs.org>
6. TypeScript [Електронний ресурс] – Режим доступу до ресурсу: <https://www.typescriptlang.org>
7. Koa.js [Електронний ресурс] – Режим доступу до ресурсу: <https://koajs.com>
8. ArangoDB [Електронний ресурс] – Режим доступу до ресурсу: <https://arangodb.com>
9. Nuxt.js [Електронний ресурс] – Режим доступу до ресурсу: <https://nuxt.com>
10. Daisy UI [Електронний ресурс] – Режим доступу до ресурсу: <https://daisyui.com>
11. VSCode [Електронний ресурс] – Режим доступу до ресурсу: <https://code.visualstudio.com>
12. Postman [Електронний ресурс] – Режим доступу до ресурсу: <https://www.postman.com>
13. Draw.io [Електронний ресурс] – Режим доступу до ресурсу: <https://www.drawio.com>
14. TsUML2 [Електронний ресурс] – Режим доступу до ресурсу: <https://github.com/demike/TsUML2>
15. TTL Indexes [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.arangodb.com/3.11/index-and-search/indexing/working-with-indexes/ttl-indexes/>

16. Winston [Електронний ресурс] – Режим доступу до ресурсу:
<https://github.com/winstonjs/winston>
17. koa-jwt [Електронний ресурс] – Режим доступу до ресурсу:
<https://www.npmjs.com/package/koa-jwt>
18. Swagger [Електронний ресурс] – Режим доступу до ресурсу:
<https://swagger.io>
19. koa-swagger-decorator [Електронний ресурс] – Режим доступу до ресурсу:
<https://www.npmjs.com/package/koa-swagger-decorator>
20. Joi [Електронний ресурс] – Режим доступу до ресурсу: <https://joi.dev/>
21. Formidable [Електронний ресурс] – Режим доступу до ресурсу:
<https://www.npmjs.com/package/formidable>
22. Stream [Електронний ресурс] – Режим доступу до ресурсу:
<https://nodejs.org/api/stream.html>
23. fcron [Електронний ресурс] – Режим доступу до ресурсу: <http://fcron.free.fr/>
24. request-ip [Електронний ресурс] – Режим доступу до ресурсу:
<https://www.npmjs.com/package/request-ip>
25. Nuxt Color Mode [Електронний ресурс] – Режим доступу до ресурсу:
<https://color-mode.nuxtjs.org>
26. Nuxt i18n [Електронний ресурс] – Режим доступу до ресурсу:
<https://i18n.nuxtjs.org>
27. Nuxt Turnstile [Електронний ресурс] – Режим доступу до ресурсу:
<https://nuxt.com/modules/turnstile>
28. Notivue [Електронний ресурс] – Режим доступу до ресурсу:
<https://notivue.smastron.io>
29. Nuxt Auth [Електронний ресурс] – Режим доступу до ресурсу:
<https://sidebase.io/nuxt-auth/getting-started>
30. Nuxt Fonts [Електронний ресурс] – Режим доступу до ресурсу:
<https://nuxt.com/modules/fonts>
31. Vee validate [Електронний ресурс] – Режим доступу до ресурсу: <https://vee-validate.logaretm.com>

32. Auto animate [Електронний ресурс] – Режим доступу до ресурсу: <https://auto-animate.formkit.com/>

33. Nuxt social share [Електронний ресурс] – Режим доступу до ресурсу: <https://nuxt.com/modules/nuxt-social-share>

		Заглада М.Р.			ІПЗ.КР.Б — 121 — 24 — ПЗ	Арк.
		Кубрак Ю.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		66

ДОДАТКИ

		Заглада М.Р.			ІПЗ.КР.Б — 121 — 24 — ПЗ	Арк.
		Кубрак Ю.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		67

Лістинг файлу index.ts

```

import Koa from 'koa';
import bodyParser from '@koa/bodyparser';
import koaHelmet from 'koa-helmet';
import cors from '@koa/cors';
import { indexRouter } from './routes/indexRoutes';
import { userRouter } from './routes/userRoutes';
import { authRouter } from './routes/authRoutes';
import { fileRouter } from './routes/fileRoutes';
import { userFilesRouter } from './routes/userFilesRoutes';
import { databaseInstance } from './util/db';
import { loggerInstance } from './logger';
import { logger } from './middleware/loggerMiddleware';
import 'dotenv/config';

const app = new Koa();

app.use(
  koaHelmet({
    contentSecurityPolicy: {
      directives: {
        defaultSrc: ['"self"'],
        scriptSrc: ['"self"', '"unsafe-inline"', 'cdnjs.cloudflare.com'],
        styleSrc: [
          '"self"',
          '"unsafe-inline"',
          'cdnjs.cloudflare.com',
          'fonts.googleapis.com',
        ],
      },
      fontSrc: ['"self"', 'fonts.gstatic.com'],
      imgSrc: [
        '"self"',
        'data:',
        'online.swagger.io',
        'validator.swagger.io',
      ],
    },
  }),
);
app.use(
  cors({
    credentials: true,
  }),
);
app.use(logger);
app.use(bodyParser());

app.use(indexRouter.routes()).use(indexRouter.allowedMethods());
app.use(authRouter.routes()).use(authRouter.allowedMethods());
app.use(userRouter.routes()).use(userRouter.allowedMethods());
app.use(fileRouter.routes()).use(fileRouter.allowedMethods());
app.use(userFilesRouter.routes()).use(userFilesRouter.allowedMethods());

```

		Заглада М.Р.			ІІЗ.КР.Б — 121 — 24 — ІІЗ	Арк.
		Кубрак Ю.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		68

```

databaseInstance
  .checkConnection()
  .then(() => {
    loggerInstance.info({
      message: `Established connection to DB at ${process.env.DATABASE_URL}`,
    });

    app.listen(process.env.PORT, () => {
      loggerInstance.info({
        message: `Server is running on port ${process.env.PORT}`,
      });
    });
  })
  .catch((err) => {
    loggerInstance.error({
      message: `Failed to establish connection to DB: ${err.code}, shutting down
api`,
    });
  });
});

```

Лістинг файлу controller/user.ts

```

import { Context } from 'koa';
import {
  description,
  request,
  summary,
  tagsAll,
  body,
  responses,
  middlewares,
} from 'koa-swagger-decorator';
import j2s from 'joi-to-swagger';
import { UserService } from '../service/user';
import {
  userValidatorSchema,
  userPasswordEditValidatorSchema,
} from '../schema/user';
import { AuthUtils } from '../util/authUtils';
import { auth } from '../middleware/auth';

const userSchema = j2s(userValidatorSchema).swagger.properties;
const { username, ...loginSchema } = userSchema;

const userPasswordEditSchema = j2s(userPasswordEditValidatorSchema).swagger
  .properties;

@tagsAll(['User'])
export class UserController {
  @request('post', '/auth/register')
  @summary('Register user')
  @description('Registers user based on provided info')
  @body(userSchema)
  @responses({
    201: { description: 'registration successful' },
    400: { description: 'uploaded data has validation errors' },
    500: { description: 'something went wrong on the server' },
  })

```

```

    })
    public static async register(ctx: Context): Promise<void> {
        ctx.body = await UserService.register(ctx);
    }

    @request('post', '/auth/login')
    @summary('User login request')
    @description('Creates sample user')
    @body(loginSchema)
    @responses({
        200: { description: 'login successful' },
        400: { description: 'provided data has validation errors' },
        401: { description: 'invalid email or password' },
        500: { description: 'something went wrong on the server' },
    })
    public static async login(ctx: Context): Promise<void> {
        ctx.body = await UserService.login(ctx);
    }

    @request('get', '/auth/logout')
    @summary('User logout request')
    @description('Logout current user')
    @middlewares([auth])
    @responses({
        200: { description: 'logout successful' },
        401: { description: 'authentication required' },
        500: { description: 'something went wrong on the server' },
    })
    public static async logout(ctx: Context): Promise<void> {
        ctx.body = await UserService.logout(ctx);
    }

    @request('post', '/auth/refresh-token')
    @summary('Refresh access token')
    @description('Refresh user access token with the refresh token')
    @responses({
        200: { description: 'token refresh successful' },
        400: { description: 'provided user in refresh token doesnt exist' },
        401: { description: 'authentication required' },
        500: { description: 'something went wrong on the server' },
    })
    public static async refreshToken(ctx: Context): Promise<void> {
        if (ctx.request.body.refreshToken) {
            ctx.state.user = AuthUtils.verifyToken(ctx, 'refresh');
            ctx.body = await UserService.refreshToken(ctx);
        } else {
            ctx.throw(401, 'Authentication error');
        }
    }

    @request('get', '/user/profile')
    @summary('User profile')
    @description('Get user profile')
    @middlewares([auth])
    @responses({
        200: { description: 'user profile request successful' },
        400: { description: 'user with this email doesnt exist' },
    })

```

```

    401: { description: 'authentication required' },
    500: { description: 'something went wrong on the server' },
  })
  public static async getProfile(ctx: Context): Promise<void> {
    ctx.body = await UserService.getProfile(ctx);
  }

  @request('put', '/user/edit')
  @summary('Edit user info')
  @description('Update user info with provided info')
  @body(userSchema)
  @middlewares([auth])
  @responses({
    200: { description: 'user profile edit successful' },
    400: { description: 'provided data has validation errors' },
    401: { description: 'not authenticated' },
    500: { description: 'something went wrong on the server' },
  })
  public static async editProfile(ctx: Context): Promise<void> {
    ctx.body = await UserService.editProfile(ctx);
  }

  @request('put', '/user/update-password')
  @summary('Update user password')
  @description('Update user password with provided one')
  @body(userPasswordEditSchema)
  @middlewares([auth])
  @responses({
    200: { description: 'user password update successful' },
    400: { description: 'provided data has validation errors' },
    401: { description: 'not authenticated' },
    500: { description: 'something went wrong on the server' },
  })
  public static async editPassword(ctx: Context): Promise<void> {
    ctx.body = await UserService.editPassword(ctx);
  }

  @request('delete', '/user/delete')
  @summary('Delete user')
  @description('Delete current user')
  @middlewares([auth])
  @responses({
    200: { description: 'user successfully deleted' },
    401: { description: 'not authenticated' },
    500: { description: 'something went wrong on the server' },
  })
  public static async delete(ctx: Context): Promise<void> {
    ctx.body = await UserService.delete(ctx);
  }
}

```

Лістинг файлу service/user.ts

```

import { Context } from 'koa';
import { ValidationError } from 'joi';
import * as argon2 from 'argon2';
import type { Document } from 'arangojs/documents';
import {

```

		Заглада М.Р.			ІПЗ.КР.Б — 121 — 24 — ІЗ	Арк.
		Кубрак Ю.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		71

```

    IUser,
    IUserLoginDto,
    IUserDto,
    IUserEditPasswordDto,
    IUserProfileDto,
} from '../interface/user';
import { ILogin } from '../interface/auth';
import {
    userValidatorSchema,
    userLoginValidatorSchema,
    userPasswordEditValidatorSchema,
} from '../schema/user';
import { databaseInstance } from '../util/db';
import { UsersRepository } from '../repository/user';
import { AuthUtils } from '../util/authUtils';
import 'dotenv/config';
import { UserFilesRepository } from '../repository/userFiles';

const usersCollection = databaseInstance.database.collection('Users');

export class UserService {
    public static async getProfile(ctx: Context): Promise<IUserProfileDto> {
        try {
            const user = await this.getByEmail(ctx.state.user.email);
            const { _id, _key, _rev, password, ...profile } = user;
            return profile;
        } catch (err: any) {
            if (err instanceof ReferenceError) {
                ctx.throw(400, err.message);
            } else {
                ctx.throw(500, err.message);
            }
        }
    }

    public static async getByEmail(email: string): Promise<Document<IUser>> {
        const user = await UsersRepository.getByEmail(email);
        if (user) return user;
        throw ReferenceError('user doesnt exist');
    }

    public static async register(ctx: Context): Promise<string> {
        const userTemplate: IUserDto = {
            username: ctx.request.body.username,
            email: ctx.request.body.email,
            password: ctx.request.body.password,
        };

        try {
            const user = await userValidatorSchema.validateAsync(userTemplate);
            user.password = await argon2.hash(user.password);

            await usersCollection.save(user);
            ctx.status = 201;
            return `registration successful`;
        } catch (err: any) {
            if (err instanceof ValidationError) {

```

```

        ctx.throw(400, err.message);
    } else if (err.message.includes('unique constraint violated')) {
        ctx.throw(400, 'user with this email already exist');
    } else {
        ctx.throw(500, err.message);
    }
}
}

public static async login(ctx: Context): Promise<ILogin> {
    const userTemplate: IUserLoginDto = {
        email: ctx.request.body.email,
        password: ctx.request.body.password,
    };

    try {
        await userLoginValidatorSchema.validateAsync(userTemplate);

        const user = await this.getByEmail(ctx.request.body.email);
        if (await argon2.verify(user.password, ctx.request.body.password)) {
            return {
                token: AuthUtils.generateToken(
                    { email: user.email, username: user.username },
                    '2m',
                ),
                refreshToken: AuthUtils.generateToken({ userId: user._id }, '30d'),
            };
        }
    } catch (err: any) {
        if (err instanceof ValidationError) {
            ctx.throw(400, err.message);
        } else if (err instanceof ReferenceError) {
            ctx.throw(401, 'invalid email or password');
        } else {
            ctx.throw(500, err.message);
        }
    }
    ctx.throw(400, 'invalid email or password');
}

public static async refreshToken(ctx: Context): Promise<ILogin> {
    const user = await usersCollection.document(ctx.state.user.userId, {
        graceful: true,
    });
    if (!user) ctx.throw(400, 'user doesnt exist');
    try {
        return {
            token: AuthUtils.generateToken(
                { email: user.email, username: user.username },
                '2m',
            ),
            refreshToken: AuthUtils.generateToken({ userId: user._id }, '30d'),
        };
    } catch (err: any) {
        ctx.throw(500, err.message);
    }
}
}

```

```

public static async logout(ctx: Context): Promise<string> {
    return `logout successful`;
}

public static async editProfile(ctx: Context): Promise<string> {
    const dbUser = await this.getByEmail(ctx.state.user.email);

    const userToValidate: IUserDto = {
        username: ctx.request.body.username,
        email: ctx.request.body.email,
        password: ctx.request.body.password,
    };

    try {
        await userValidatorSchema.validateAsync(userToValidate);

        if (await argon2.verify(dbUser.password, userToValidate.password)) {
            const { password, ...userTemplate } = userToValidate;
            const updatedUser = await usersCollection.update(
                dbUser._key,
                userTemplate,
                {
                    returnNew: true,
                },
            );

            return AuthUtils.generateToken(
                { email: updatedUser.new.email, username: updatedUser.new.username },
                '2m',
            );
        }
    } catch (err: any) {
        if (err instanceof ValidationError) {
            ctx.throw(400, err.message);
        } else {
            ctx.throw(500, err.message);
        }
    }
    ctx.throw(401, 'invalid password');
}

public static async editPassword(ctx: Context): Promise<string> {
    const dbUser = await this.getByEmail(ctx.state.user.email);

    const passwordToValidate: IUserEditPasswordDto = {
        oldPassword: ctx.request.body.oldPassword,
        password: ctx.request.body.password,
    };

    try {
        await userPasswordEditValidatorSchema.validateAsync(passwordToValidate);

        if (
            await argon2.verify(dbUser.password, passwordToValidate.oldPassword)
        ) {
            const { oldPassword, ...userTemplate } = passwordToValidate;

```

```

        userTemplate.password = await argon2.hash(userTemplate.password);
        await usersCollection.update(dbUser._key, userTemplate);

        return `user password updated successfully`;
    }
} catch (err: any) {
    if (err instanceof ValidationError) {
        ctx.throw(400, err.message);
    } else {
        ctx.throw(500, err.message);
    }
}
ctx.throw(401, 'invalid password');
}

public static async delete(ctx: Context): Promise<string> {
    const user = await this.getByEmail(ctx.state.user.email);

    try {
        await UserFilesRepository.removeAllFileOwnership(user._id);
        await usersCollection.remove(user._id);
        return `profile successfully deleted`;
    } catch (err: any) {
        ctx.throw(500, err.message);
    }
}
}
}

```

Лістинг файлу encryption.ts

```

import {
    createCipheriv,
    createDecipheriv,
    createHash,
    randomBytes,
} from "crypto";
import type { CipherGCMTypes } from "crypto";
import { createReadStream, createWriteStream } from "fs";
import { unlink } from "node:fs/promises";
import { createGzip, createUnzip } from "zlib";
import path from "node:path";
import { AppendV } from "./iv";
import * as StreamPromises from "stream/promises";
import { PassThrough } from "stream";

export class EncryptionUtils {
    public static async encryptFile(filePath: string, password: string) {
        const algorithm: CipherGCMTypes = "aes-256-gcm";
        const iv = randomBytes(16);
        const key = this.generateKey(password);

        const encryptStream = createReadStream(filePath);
        const gzip = createGzip();
        const cipher = createCipheriv(algorithm, key, iv);
        const appendV = new AppendV(iv);
        const writeStream = createWriteStream(path.join(filePath + ".enc"));

        await StreamPromises.pipeline(

```

```

        encryptStream,
        gzip,
        cipher,
        appendIv,
        writeStream,
    );
    await unlink(filePath);
    return cipher.getAuthTag().toString("hex");
}

public static async decryptFile(
    filePath: string,
    password: string,
    authTag: string,
) {
    const readIv = createReadStream(filePath, { end: 15 });

    let iv: any;
    readIv.on("data", (chunk) => {
        iv = chunk;
    });

    const returnStream = new PassThrough();
    readIv.on("close", async () => {
        const key = this.generateKey(password);
        const readStream = createReadStream(filePath, { start: 16 });
        const decipher = createDecipheriv("aes-256-gcm", key, iv);
        decipher.setAuthTag(Buffer.from(authTag, "hex"));
        const unzip = createUnzip();
        await StreamPromises.pipeline(readStream, decipher, unzip, returnStream);
    });
    return returnStream;
}

public static generateKey(password: string): Buffer {
    return createHash("sha256").update(password).digest();
}
}

```

Лістинг файлу service/userFiles.ts

```

import { Context } from "koa";
import { ValidationError } from "joi";
import * as argon2 from "argon2";
import { unlink } from "fs/promises";
import {
    IFileEditDto,
    IFileEditPasswordDto,
    IReturnFileDto,
    IShareFileDto,
    PrivacyType,
} from "../interface/file";
import { databaseInstance } from "../util/db";
import { UserService } from "../user";
import { FileService } from "../file";
import { UserFilesRepository } from "../repository/userFiles";
import {
    filePasswordEditValidatorSchema,

```

		Заглада М.Р.			ІІЗ.КР.Б — 121 — 24 — ІІЗ	Арк.
		Кубрак Ю.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		76

```

    fileValidatorSchema,
    shareFileEmailValidatorSchema,
} from "../schema/file";

const filesCollection = databaseInstance.database.collection("Files");
const userFilesCollection = databaseInstance.database.collection("UserFiles");

export class UserFilesService {
    public static async getFiles(ctx: Context): Promise<IReturnFileDto[]> {
        const user = await UserService.getByEmail(ctx.state.user.email);

        try {
            const userFiles = await UserFilesRepository.getFiles(user._id);
            const returnUserFiles: IReturnFileDto[] = [];
            for (const userFile of userFiles) {
                returnUserFiles.push(
                    await FileService.prepareFileResponse(userFile, user._id),
                );
            }
            return returnUserFiles;
        } catch (err: any) {
            ctx.throw(err.status || 500, err.message);
        }
    }

    public static async getFile(ctx: Context): Promise<IReturnFileDto> {
        const user = await UserService.getByEmail(ctx.state.user.email);

        try {
            await this.checkValidity(ctx, user._id, ctx.params.id);
            const userFile = await UserFilesRepository.getFile(
                user._id,
                `Files/${ctx.params.id}`,
            );
            return await FileService.prepareFileResponse(userFile, user._id);
        } catch (err: any) {
            ctx.throw(err.status || 500, err.message);
        }
    }

    public static async deleteFile(ctx: Context): Promise<string> {
        const user = await UserService.getByEmail(ctx.state.user.email);

        try {
            await this.checkValidity(ctx, user._id, ctx.params.id, true);
            const file = await UserFilesRepository.getFile(
                user._id,
                `Files/${ctx.params.id}`,
            );

            if (file) {
                await unlink(file.path);
                await UserFilesRepository.removeFile(user._id, file._id);
                await UserFilesRepository.removeAllFileOwnership(file._id);
                return `file deleted successfully`;
            }
            ctx.throw(404, `file doesnt exist`);
        }
    }
}

```

```

    } catch (err: any) {
        ctx.throw(err.status || 500, err.message);
    }
}

public static async updateFileVisibility(
    ctx: Context,
): Promise<IReturnFileDto> {
    const user = await UserService.getByEmail(ctx.state.user.email);
    const fileTemplate: IFileEditDto = {
        privacy: ctx.request.body.privacy,
    };
    if (ctx.request.body.password) {
        fileTemplate.password = ctx.request.body.password;
    }

    try {
        await this.checkValidity(ctx, user._id, ctx.params.id, true);
        await fileValidatorSchema.validateAsync(fileTemplate);
        const file = await UserFilesRepository.getFile(
            user._id,
            `Files/${ctx.params.id}`,
        );

        if (!file) {
            ctx.throw(404, "file not found");
        }
        if (file.privacy === fileTemplate.privacy) {
            ctx.throw(400, "please select new visibility modifier");
        }
        if (fileTemplate.privacy === PrivacyType.Protected) {
            fileTemplate.password = await argon2.hash(fileTemplate.password!);
        } else {
            fileTemplate.password = "";
        }

        const updatedFile = await filesCollection.update(
            file._key,
            fileTemplate,
            {
                returnNew: true,
            },
        );
        return await FileService.prepareFileResponse(updatedFile.new, user._id);
    } catch (err: any) {
        if (err instanceof ValidationError) {
            ctx.throw(400, err.message);
        }
        ctx.throw(err.status || 500, err.message);
    }
}

public static async updateFilePassword(ctx: Context): Promise<string> {
    const user = await UserService.getByEmail(ctx.state.user.email);
    const fileTemplate: IFileEditPasswordDto = {
        fileId: ctx.request.body.fileId,
        oldPassword: ctx.request.body.oldPassword,
    };

```

```

        password: ctx.request.body.password,
    };

    try {
        await filePasswordEditValidatorSchema.validateAsync(fileTemplate);
        await this.checkValidity(ctx, user._id, fileTemplate.fileId, true);
        const file = await UserFilesRepository.getFile(
            user._id,
            `Files/${fileTemplate.fileId}`,
        );

        if (!file) {
            ctx.throw(404, "file not found");
        }
        if (file.privacy !== PrivacyType.Protected) {
            ctx.throw(400, "file visibility is not protected");
        }

        if (await argon2.verify(file.password, fileTemplate.oldPassword)) {
            const { fileId, oldPassword, ...passwordToUpdate } = fileTemplate;
            passwordToUpdate.password = await argon2.hash(
                passwordToUpdate.password,
            );
            await filesCollection.update(file._key, passwordToUpdate);
            return `file password updated successfully`;
        }
        ctx.throw(401, "invalid password");
    } catch (err: any) {
        if (err instanceof ValidationError) {
            ctx.throw(400, err.message);
        }
        ctx.throw(err.status || 500, err.message);
    }
}

public static async shareFile(ctx: Context): Promise<string> {
    const emailTemplate: IShareFileDto = {
        email: ctx.request.body.email,
    };

    try {
        const user = await UserService.getByEmail(ctx.state.user.email);
        await shareFileEmailValidatorSchema.validateAsync(emailTemplate);
        await this.checkValidity(ctx, user._id, ctx.params.id, true);
        const file = await UserFilesRepository.getFile(
            user._id,
            `Files/${ctx.params.id}`,
        );
        if (!file) {
            ctx.throw(404, "file not found");
        }

        const sharedToUser = await UserService.getByEmail(emailTemplate.email);
        if (user._id === sharedToUser._id) {
            ctx.throw(400, "you cant share files to yourself");
        }
    }
}

```

```

const sharedToUsers = (
  await UserFilesRepository.getSharedToUsers(user._id, file._id)
)[0];
if (sharedToUsers?.includes(sharedToUser.email)) {
  ctx.throw(400, "you already shared this file to this user");
}

await userFilesCollection.save({
  _from: sharedToUser._id,
  _to: file._id,
  owner: false,
  grantedBy: user.username,
});

return `successfully shared file ${file._key} to user $
{sharedToUser._key}`;
} catch (err: any) {
  if (err instanceof ReferenceError) {
    ctx.throw(400, err.message);
  }
  ctx.throw(err.status || 500, err.message);
}
}

public static async removeSharedFile(ctx: Context): Promise<string> {
  const emailTemplate: IShareFileDto = {
    email: ctx.request.body.email,
  };

  try {
    const user = await UserService.getByEmail(ctx.state.user.email);
    await shareFileEmailValidatorSchema.validateAsync(emailTemplate);
    await this.checkValidity(ctx, user._id, ctx.params.id, true);
    const file = await UserFilesRepository.getFile(
      user._id,
      `Files/${ctx.params.id}`,
    );
    if (!file) {
      ctx.throw(404, "file not found");
    }

    const sharedToUser = await UserService.getByEmail(emailTemplate.email);
    if (user._id === sharedToUser._id) {
      ctx.throw(400, "you cant unshare file that you own");
    }

    const sharedToUsers = (
      await UserFilesRepository.getSharedToUsers(user._id, file._id)
    )[0];
    if (!sharedToUsers.includes(sharedToUser.email)) {
      ctx.throw(400, "you never shared this file to that user");
    }

    await UserFilesRepository.removeSharedFile(sharedToUser._id, file._id);
    return `successfully removed shared file ${file._key} from user $
{sharedToUser._key}`;
  } catch (err: any) {
    if (err instanceof ReferenceError) {

```

		Заглада М.Р.			ІІЗ.КР.Б — 121 — 24 — ІІЗ	Арк.
		Кубрак Ю.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		80

```

        ctx.throw(400, err.message);
    }
    ctx.throw(err.status || 500, err.message);
}

private static async checkValidity(
    ctx: Context,
    userId: string,
    fileId: string,
    ownerCheck: boolean = false,
): Promise<boolean> {
    if (
        ownerCheck &&
        !(await UserFilesRepository.checkIfOwner(userId, `Files/${fileId}`))
    ) {
        ctx.throw(
            403,
            `you cant perform this operation because you arent the owner of this
file`,
        );
    } else if (
        !(await UserFilesRepository.checkIfOwns(userId, `Files/${fileId}`))
    ) {
        ctx.throw(403, `you dont have access to this file`);
    }

    return true;
}
}

```

Лістинг файлу postman/file-sharing-service-api.postman_collection.json

```

{
  "info": {
    "_postman_id": "866bfe8d-a443-4da9-a554-9faedcfe8a0c",
    "name": "file-sharing-service-api",
    "schema":
"https://schema.getpostman.com/json/collection/v2.1.0/collection.json",
    "_exporter_id": "33066872"
  },
  "item": [
    {
      "name": "User routes",
      "item": [
        {
          "name": "Register",
          "request": {
            "method": "POST",
            "header": [],
            "body": {
              "mode": "urlencoded",
              "urlencoded": [
                {
                  "key": "username",
                  "value": "newuser",
                  "type": "text"
                }
              ]
            }
          }
        }
      ]
    }
  ]
}

```

```

        {
            "key": "email",
            "value":
"rwewerwrerwe@gmail.com",
            "type": "text"
        },
        {
            "key": "password",
            "value": "1234567",
            "type": "text"
        }
    ],
    },
    "url": {
        "raw": "{{base_url_auth}}/register",
        "host": [
            "{{base_url_auth}}"
        ],
        "path": [
            "register"
        ]
    },
    "response": []
},
{
    "name": "Login",
    "request": {
        "method": "POST",
        "header": [],
        "body": {
            "mode": "urlencoded",
            "urlencoded": [
                {
                    "key": "email",
                    "value":
"rwewerwrerwe@gmail.com",
                    "type": "text"
                },
                {
                    "key": "password",
                    "value": "1234567",
                    "type": "text"
                }
            ]
        },
        "url": {
            "raw": "{{base_url_auth}}/login",
            "host": [
                "{{base_url_auth}}"
            ],
            "path": [
                "login"
            ]
        }
    },
    "response": []
}

```

		Заглада М.Р.			ІІЗ.КР.Б — 121 — 24 — ІІЗ	Арк.
		Кубрак Ю.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		82

```

    },
    {
        "name": "Profile",
        "request": {
            "auth": {
                "type": "bearer",
                "bearer": [
                    {
                        "key": "token",
                        "value":
"eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJlbWFnbnBhbmFpbC5jb2
0iLCJ1c2VybmFtZSI6Im5ld3VzZXIiLCJpYXQiOiE3MTcyNTkyMzEsImV4cCI6MTcxNzI1OTM1MX0.X7
9GY79YVuDlMsWKCDKz7wsxaDtQRZ3bXMWHpnaR7E1ENPdN_IXAzXH1FQUl4Renv7gCTXjam5QYLKvll3
a_Sg",
                        "type": "string"
                    }
                ]
            },
            "method": "GET",
            "header": [],
            "url": {
                "raw": "{{base_url_user}}/profile",
                "host": [
                    "{{base_url_user}}"
                ],
                "path": [
                    "profile"
                ]
            }
        },
        "response": []
    },
    {
        "name": "Refresh token",
        "request": {
            "method": "POST",
            "header": [],
            "body": {
                "mode": "urlencoded",
                "urlencoded": [
                    {
                        "key": "refreshToken",
                        "value":
"eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOiJvc2Vycy8xIiwiaWF0IjoxNzE3Mj
U4MDQzLCJleHAiOiE3MTk4NTAwNDN9.LZvxx5N20RfTHcHQgFon96kad2RxtujKgL4F9NUJMHuDNbKO
9Yj1wwxdpRWjngp80s_3EQuEkSmGteNgp6j_Q",
                        "type": "text"
                    }
                ]
            },
            "url": {
                "raw": "{{base_url_auth}}/refresh-
token",
                "host": [
                    "{{base_url_auth}}"
                ],
                "path": [

```

		Заглада М.Р.			ІІЗ.КР.Б — 121 — 24 — ІІЗ	Арк.
		Кубрак Ю.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		83


```

        "value":
"eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJlbWFnbnCI6InJ3ZXdlcndyZXJ3ZUBnbWFpbC5jb2
0iLCJ1c2VybmFtZSI6Im5ld3VzZXIiLCJpYXQiOiJlMTczMjc4MDUsImV4cCI6MTcxNzMyNzkyNX0.R9
5u1MKvtA0WozyxHoMzkWBixpnUTTbVXibv4n1YsnDfWYw0r9oL8HqfVE2RPGbh-
yiQKxq9f2TQIMVz7DNoUg",

        "type": "string"

    }

    ],

    },
    "method": "DELETE",
    "header": [],
    "url": {
        "raw": "{{base_url_user}}/delete",
        "host": [
            "{{base_url_user}}"
        ],
        "path": [
            "delete"
        ]
    }
    },
    "response": []
},
{
    "name": "Logout",
    "request": {
        "auth": {
            "type": "bearer",
            "bearer": [
                {
                    "key": "token",
                    "value":
"eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJlbWFnbnCI6InJ3ZXdlcndyZXJ3ZUBnbWFpbC5jb2
0iLCJ1c2VybmFtZSI6Im5ld3VzZXIiLCJpYXQiOiJlMTczMjc4MDUsImV4cCI6MTcxNzMyNzkyNX0.R9
5u1MKvtA0WozyxHoMzkWBixpnUTTbVXibv4n1YsnDfWYw0r9oL8HqfVE2RPGbh-
yiQKxq9f2TQIMVz7DNoUg",

                    "type": "string"

                }
            ]
        },
        "method": "GET",
        "header": [],
        "url": {
            "raw": "{{base_url_auth}}/logout",
            "host": [
                "{{base_url_auth}}"
            ],
            "path": [
                "logout"
            ]
        }
    },
    "response": []
}
}
},
{

```



```

    },
    "method": "GET",
    "header": [],
    "body": {
      "mode": "urlencoded",
      "urlencoded": []
    },
    "url": {
      "raw": "{{base_url_file}}/d1863c9a-11c4-46c9-b3b0-d58c54834f06",
      "host": [
        "{{base_url_file}}"
      ],
      "path": [
        "d1863c9a-11c4-46c9-b3b0-d58c54834f06"
      ]
    }
  },
  "response": []
},
{
  "name": "Download file",
  "protocolProfileBehavior": {
    "disableBodyPruning": true
  },
  "request": {
    "auth": {
      "type": "bearer",
      "bearer": [
        {
          "key": "token",
          "value": "eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJlbWFpbCI6InJ3ZXdlcndyZXJ3ZUBnbWFpbC5jb20iLCJ1c2VybmFtZSI6Im5ld3VzZXIiLCJpYXQiOiJlMjE3MTczMjc4MDUsImV4cCI6MTcxNzMyNzkyNX0uRjU1MKvtAQWozyxHoMzkWBixpnUTTbVXibv4n1YsnDfWywOr9oL8HqfVE2RPGbh-yiQKxq9f2TQIMVz7DNoUg",
          "type": "string"
        }
      ]
    },
    "method": "GET",
    "header": [],
    "body": {
      "mode": "urlencoded",
      "urlencoded": []
    },
    "url": {
      "raw": "{{base_url_file}}/download/30835fc1-9f51-463a-bc29-77823e558f29",
      "host": [
        "{{base_url_file}}"
      ],
      "path": [
        "download",

```

		Заглада М.Р.			ІІЗ.КР.Б — 121 — 24 — ІІЗ	Арк.
		Кубрак Ю.О.				88
Змн.	Арк.	№ докум.	Підпис	Дата		

77823e558f29"

"30835fc1-9f51-463a-bc29-

]

}

},

"response": []

},

{

"name": "Get protected file",

"request": {

"method": "POST",

"header": [],

"body": {

"mode": "urlencoded",

"urlencoded": [

{

"key": "password",

"value": "1234567",

"type": "text"

}

]

},

"url": {

"raw":

"{{base_url_file}}/d1863c9a-

11c4-46c9-b3b0-d58c54834f06",

"host": [

"{{base_url_file}}"

],

"path": [

"d1863c9a-11c4-46c9-b3b0-

d58c54834f06"

],

"query": [

{

"key": "",

"value": null,

"disabled": true

}

]

}

},

"response": []

},

{

"name": "Download protected file",

"request": {

"method": "POST",

"header": [],

"body": {

"mode": "urlencoded",

"urlencoded": [

{

"key": "password",

"value": "1234567",

"type": "text"

}

]

}

]

```

        },
        "url": {
            "raw":
"{{base_url_file}}/download/d1863c9a-11c4-46c9-b3b0-d58c54834f06",
            "host": [
                "{{base_url_file}}"
            ],
            "path": [
                "download",
                "d1863c9a-11c4-46c9-b3b0-
d58c54834f06"
            ],
            "query": [
                {
                    "key": "",
                    "value": null,
                    "disabled": true
                }
            ]
        },
        "response": []
    }
},
{
    "name": "User files routes",
    "item": [
        {
            "name": "Get user files",
            "request": {
                "auth": {
                    "type": "bearer",
                    "bearer": [
                        {
                            "key": "token",
                            "value":
"eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJlbWFPbCI6InJ3ZXdlcndyZXJ3ZUBnbWFPbC5jb2
0iLCJ1c2VybmFtZSI6Im5ld3VzZXIiLCJpYXQiOiE3MTczMjc3ImV4cCI6MTcxNzMyNzg0N30.ZL
ZQyEjzjJ4IezH0Jt9-0NXcVsMjF32juHgCe910xiamPphXabDxLRCG80VUBM-
VeaYTSqlgGpXz2Z0QSi_PWg",
                            "type": "string"
                        }
                    ]
                },
                "method": "GET",
                "header": [],
                "url": {
                    "raw": "{{base_url_user-files}}",
                    "host": [
                        "{{base_url_user-files}}"
                    ]
                }
            },
            "response": []
        }
    ],
    {

```

		Заглада М.Р.			ІІЗ.КР.Б — 121 — 24 — ІІЗ	Арк.
		Кубрак Ю.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		90

```

"name": "Delete user file",
"request": {
  "auth": {
    "type": "bearer",
    "bearer": [
      {
        "key": "token",
        "value":
"eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJlbWFnZW50InR5cGU6ImV4cCI6MTcxNzMyNzkyNX0uR9
5u1MKvtA0WozyxHoMzkWBixpnUTTbVXIbV4n1YsnDfWYw0r9oL8HqfVE2RPGbh-
yiQKxq9f2TQIMVz7DNoUg",
        "type": "string"
      }
    ]
  },
  "method": "DELETE",
  "header": [],
  "url": {
    "raw": "{{base_url_user-files}}/49",
    "host": [
      "{{base_url_user-files}}"
    ],
    "path": [
      "49"
    ]
  }
},
"response": []
},
{
  "name": "Change user file visibility",
  "request": {
    "auth": {
      "type": "bearer",
      "bearer": [
        {
          "key": "token",
          "value":
"eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJlbWFnZW50InR5cGU6ImV4cCI6MTcxNzMyNzkyNX0uR9
5u1MKvtA0WozyxHoMzkWBixpnUTTbVXIbV4n1YsnDfWYw0r9oL8HqfVE2RPGbh-
yiQKxq9f2TQIMVz7DNoUg",
          "type": "string"
        }
      ]
    },
    "method": "PATCH",
    "header": [],
    "body": {
      "mode": "urlencoded",
      "urlencoded": [
        {
          "key": "privacy",
          "value": "public",
          "type": "text"
        }
      ]
    }
  }
}

```

		Заглада М.Р.			ІІЗ.КР.Б — 121 — 24 — ІІЗ	Арк.
		Кубрак Ю.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		91

```

        {
            "key": "password",
            "value": "1234567",
            "type": "text",
            "disabled": true
        }
    ],
    },
    "url": {
        "raw": "{{base_url_user-files}}/52",
        "host": [
            "{{base_url_user-files}}"
        ],
        "path": [
            "52"
        ]
    }
},
"response": []
},
{
    "name": "Update user file password",
    "request": {
        "auth": {
            "type": "bearer",
            "bearer": [
                {
                    "key": "token",
                    "value":
"eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJlbWFnZWVudCI6InJ3ZXdlcnR5ZXJ3ZUBnbWFnZWVudC5jb2
0iLCJ1c2VybmFtZSI6Im5ld3VzZXIiLCJpYXQiOiE3MTczMjc4MDUsImV4cCI6MTcxNzMyNzkyNX0.R9
5u1MKvtA0WozyxHoMzkWBixpnUTTbVXibv4n1YsnDfWYw0r9oL8HqfVE2RPGbh-
yiQKxq9f2TQIMVz7DNoUg",
                    "type": "string"
                }
            ]
        },
        "method": "PATCH",
        "header": [],
        "body": {
            "mode": "urlencoded",
            "urlencoded": [
                {
                    "key": "fileId",
                    "value": "34",
                    "type": "text"
                },
                {
                    "key": "oldPassword",
                    "value": "1234567",
                    "type": "text"
                },
                {
                    "key": "password",
                    "value": "12345678",
                    "type": "text"
                }
            ]
        }
    }
}

```

		Заглада М.Р.			ІІЗ.КР.Б — 121 — 24 — ІІЗ	Арк.
		Кубрак Ю.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		92


```

        "host": [
            "{{base_url}}"
        ],
        "path": [
            ""
        ]
    }
},
"response": []
}
],
"variable": [
    {
        "key": "base_url",
        "value": "http://localhost:3000"
    },
    {
        "key": "base_url_user",
        "value": "http://localhost:3000/user"
    },
    {
        "key": "base_url_auth",
        "value": "http://localhost:3000/auth"
    },
    {
        "key": "base_url_file",
        "value": "http://localhost:3000/file"
    },
    {
        "key": "base_url_user-files",
        "value": "http://localhost:3000/user-files"
    }
]
}

```

		Заглада М.Р.			ІІЗ.КР.Б — 121 — 24 — ІІЗ	Арк.
		Кубрак Ю.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		94