

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:

**«Вебсистема тимчасового обміну мультимедійними
даними з контрольованим доступом»**

Виконав студент групи ІПЗ-22-1
ЧЕЛЯДИН Тетяна Анатоліївна

Керівник роботи:
ЛОБАНЧИКОВА Надія Миколаївна

Рецензент:
ЧИЖМОТРЯ Олексій Володимирович

Житомир – 2026

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

«28» лютого 2026 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ЧЕЛЯДИН Тетяна Анатоліївна**

Керівник роботи: **ЛОБАНЧИКОВА Надія Миколаївна**

Тема роботи: «Вебсистема тимчасового обміну мультимедійними даними з контрольованим доступом»,

затверджена Наказом закладу вищої освіти від **«27» лютого 2026 р., №92/с**

Вихідні дані для роботи: розробити вебсистему тимчасового обміну мультимедійними даними з контрольованим доступом, що забезпечує завантаження, зберігання та передачу файлів із можливістю обмеження терміну дії посилань, автентифікації користувачів та керування правами доступу.

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

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Лобанчикова Н. М.	01.03.2026р.	01.03.2026р
2	Лобанчикова Н. М.	01.04.2026р	01.04.2026р
3	Лобанчикова Н. М.	01.05.2026р	01.05.2026р

РЕФЕРАТ

Кваліфікаційна робота бакалавра складається з програмного комплексу та пояснювальної записки. Пояснювальна записка до кваліфікаційної роботи містить 106 сторінок, 23 ілюстрацій та 4 таблиці.

Мета роботи – проектування та реалізація вебсистеми тимчасового обміну мультимедійними даними з контрольованим доступом, що забезпечує завантаження, зберігання та передачу файлів із підтримкою текстових записів, голосових повідомлень, механізмами автоматичного блокування доступу за часом або кількістю переглядів, автентифікацією користувачів та адміністративним управлінням.

Використано методи системного аналізу при дослідженні предметної області та порівнянні аналогів, об'єктно-орієнтованого проектування при визначенні структури класів і модулів системи, реляційного моделювання при проектуванні схеми бази даних, REST-архітектури при проектуванні API, криптографічного хешування при реалізації механізмів захисту паролів і токенів автентифікації, а також методи тестування програмного забезпечення при перевірці коректності реалізованого функціоналу.

Отримані результати включають розроблену модульну монолітну архітектуру вебсистеми, спроектовану реляційну схему бази даних PostgreSQL із шістьма таблицями, реалізацію REST API на базі Node.js та Fastify з документацією у форматі OpenAPI 3.0, односторінковий клієнтський застосунок на React та TypeScript, інтеграцію з хмарним об'єктним сховищем Cloudflare R2, а також систему двотокенної JWT-автентифікації з підтримкою ролей користувача та адміністратора.

КЛЮЧОВІ СЛОВА: АВТЕНТИФІКАЦІЯ, БАЗА ДАНИХ, ВЕБСИСТЕМА, КОНТРОЛЬОВАНИЙ ДОСТУП, МУЛЬТИМЕДІЙНІ ДАНІ.

					ІПЗ.КР.Б-121-26ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата	“Вебсистема тимчасового обміну мультимедійними даними з контрольованим доступом”	Літ.	Арк.	Аркушів
Розроб.		Челядин Т. А.					4	106
Керівник		Лобанчикова Н. М.						
Рецензент		Чижмотря О. В.						
Н. контр.								
Зав. каф.		Вакалюк Т. А.				ФІКТ Гр. ІПЗ-22-1		

ABSTRACT

The bachelor's qualification work consists of a program complex and an explanatory note. The explanatory note to the qualification work contains 106 pages, 23 illustrations and 4 tables.

The purpose of the work is to design and implement a web system for temporary exchange of multimedia data with controlled access, which provides downloading, storage and transfer of files with support for text recordings, voice messages, mechanisms for automatic blocking of access by time or number of views, user authentication and administrative management.

The methods of system analysis are used in the study of the subject area and comparison of analogues, object-oriented design in determining the structure of classes and modules of the system, relational modeling in the design of data schemes, REST-base in the design of API, cryptographic hashing in the implementation of password protection mechanisms and authentication tokens, as well as software testing methods in checking the correctness of the implemented functionality.

The results include the development of a modular monolithic web system architecture, a designed PostgreSQL relational database schema with six tables, a REST API implementation based on Node.js and Fastify with documentation in OpenAPI 3.0 format, a single-page client resource based on React and TypeScript, integration with the Cloudflare R2 cloud object storage, and two-token JWT authentication with support for user and administrator roles.

KEYWORDS: AUTHENTICATION, DATABASE, WEB SYSTEM, CONTROLLED ACCESS, MULTIMEDIA DATA.

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		4

ЗМІСТ

ВСТУП.....	7
РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ВИБІР ЗАСОБІВ РЕАЛІЗАЦІЇ ВЕБСИСТЕМИ ТИМЧАСОВОГО ОБМІНУ МУЛЬТИМЕДІЙНИМИ ДАНИМИ	10
1.1. Постановка задачі	10
1.2. Аналіз аналогів програмного продукту	12
1.3. Вибір архітектури системи.....	16
1.4. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення.....	22
Висновки до першого розділу.....	26
РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ ВЕБСИСТЕМИ ТИМЧАСОВОГО ОБМІНУ МУЛЬТИМЕДІЙНИМИ ДАНИМИ.....	27
2.1. Визначення варіантів використання та об'єктно-орієнтованої структури системи	27
2.2. Розробка бази даних системи.....	37
2.3. Проєктування та реалізація алгоритмів роботи системи	41
2.4. Реалізація програмного комплексу	45
2.5. Опис компонентної будови програмного додатку.....	49
Висновки до другого розділу	51
РОЗДІЛ 3. ІНТЕРФЕЙС, НАЛАШТУВАННЯ ТА ТЕСТУВАННЯ ВЕБСИСТЕМИ ТИМЧАСОВОГО ОБМІНУ МУЛЬТИМЕДІЙНИМИ ДАНИМИ	53
3.1 Порядок встановлення та налаштування системи.....	53
3.2 Структура інтерфейсу та порядок роботи із системою.....	57
3.3 Тестування роботи програмного комплексу	66
Висновки до третього розділу	75
ВИСНОВКИ	76
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	79
ДОДАТКИ	83

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

API – Application Programming Interface (прикладний програмний інтерфейс)

CSPRNG – Cryptographically Secure Pseudo-Random Number Generator
(криптографічно стійкий генератор псевдовипадкових чисел)

HTTP – HyperText Transfer Protocol (протокол передачі гіпертексту)

JWT – JSON Web Token (токен веб-автентифікації у форматі JSON)

ORM – Object-Relational Mapping (об'єктно-реляційне відображення)

REST – Representational State Transfer (архітектурний стиль проектування
розподілених систем)

SPA – Single Page Application (односторінковий вебзастосунок)

SQL – Structured Query Language (мова структурованих запитів до баз даних)

TTL – Time To Live (час життя запису, термін автоматичного видалення)

UML – Unified Modeling Language (уніфікована мова моделювання)

БД – база даних

СКБД – система керування базами даних

ПЗ – програмне забезпечення

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		6

ВСТУП

Актуальність теми. В умовах стрімкого розвитку цифрових технологій та зростання обсягів даних, що передаються через мережу Інтернет, дедалі гострішою стає потреба у зручних, безпечних та масштабованих інструментах для обміну інформацією з обмеженим терміном доступу. Особливої актуальності набувають сервіси тимчасового зберігання та передачі контенту – текстових фрагментів, файлів, мультимедійних матеріалів – які дозволяють користувачам публікувати дані за унікальним посиланням з автоматичним видаленням після завершення визначеного терміну або вичерпання ліміту переглядів.

Практична потреба у подібних сервісах охоплює широке коло сфер застосування. У сфері розробки програмного забезпечення такі інструменти використовуються для обміну фрагментами коду, конфігураційними файлами та логами між членами команди. У корпоративному середовищі – для тимчасового надання доступу до документів без необхідності створення облікових записів для зовнішніх контрагентів. В освітній сфері – для обміну навчальними матеріалами з автоматичним обмеженням доступу після завершення курсу. В особистому використанні – для безпечної передачі конфіденційної інформації з гарантією її видалення після прочитання.

Існуючі рішення у цій галузі – Pastebin, GitHub Gist, Hastebin – мають суттєві обмеження: відсутність підтримки файлів довільного формату, відсутність механізму обмеження кількості переглядів, закритий або платний API, неможливість самостійного розгортання. Зазначені недоліки визначають актуальність розробки власного програмного комплексу, що усуне ці обмеження та запропонує комплексне рішення для тимчасового доступу до контенту будь-якого типу.

Метою випускної роботи є проектування та реалізація вебсистеми тимчасового обміну мультимедійними даними з контрольованим доступом, що забезпечує створення текстових записів і файлового контенту, налаштування

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		7

параметрів доступу, автентифікацію користувачів та адміністративне управління системою.

Для досягнення вказаної мети необхідним є вирішення наступних взаємозалежних задач:

- провести аналізу предметної області та існуючих аналогів програмного продукту з виявленням їх переваг і недоліків;
- визначити функціональні та нефункціональні вимоги до вебсистеми;
- обґрунтувати вибір архітектурного підходу і стеку технологій для реалізації вебсистеми;
- спроектувати структуру бази даних для зберігання метаданих записів та сесій користувачів;
- розробити алгоритми забезпечення тимчасового та контрольованого доступу до контенту;
- здійснити програмну реалізацію вебсистеми;
- провести тестування розробленої вебсистеми тимчасового обміну мультимедійними даними з контрольованим доступом.

Об'єктом дослідження є процес організації тимчасового та обмеженого доступу до цифрового контенту в умовах розподіленої вебархітектури.

Предметом дослідження є методи та засоби програмної реалізації вебсистем тимчасового обміну мультимедійними даними з контрольованим доступом.

Методи аналізу та інформаційна база дослідження. В роботі використано такі методи: методи системного аналізу при дослідженні предметної області та порівнянні існуючих аналогів; методи об'єктно-орієнтованого проектування при визначенні структури класів та модулів системи; методи реляційного моделювання при проектуванні схеми бази даних; методи проектування вебсервісів на основі архітектурного стилю REST; методи криптографічного хешування при реалізації механізмів захисту паролів та токенів автентифікації; методи тестування програмного забезпечення при перевірці коректності реалізованого функціоналу.

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		8

Практичне значення отриманих результатів полягає у створенні вебсистеми тимчасового обміну мультимедійними даними, яка може використовуватися для безпечної передачі текстової інформації та файлів з обмеженим терміном доступу. Розроблений програмний комплекс забезпечує контроль над поширенням контенту шляхом використання механізмів автоматичного видалення записів, обмеження кількості переглядів, парольного захисту та одноразового доступу до інформації.

Система може бути впроваджена в діяльність підприємств, установ та організацій для обміну службовою документацією, технічними матеріалами, мультимедійними файлами та іншими даними, доступ до яких повинен бути обмежений у часі або за кількістю переглядів. Використання програмного продукту дозволяє підвищити рівень інформаційної безпеки, зменшити ризик несанкціонованого поширення даних та спростити процес передачі інформації між співробітниками й зовнішніми користувачами.

Розроблена система має модульну архітектуру та може бути адаптована до потреб конкретних організацій шляхом розширення функціональних можливостей, інтеграції з корпоративними інформаційними системами або впровадження додаткових механізмів контролю доступу. Отримані результати можуть бути використані як основа для подальшого розвитку програмних рішень у сфері захищеного обміну інформацією.

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		9

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ВИБІР ЗАСОБІВ РЕАЛІЗАЦІЇ ВЕБСИСТЕМИ ТИМЧАСОВОГО ОБМІНУ МУЛЬТИМЕДІЙНИМИ ДАНИМИ

1.1. Постановка задачі

Сучасний розвиток інформаційних технологій супроводжується постійним зростанням обсягів цифрових даних, що передаються через мережу Інтернет. Користувачі регулярно обмінюються текстовими повідомленнями, документами, зображеннями, аудіо- та відеофайлами, використовуючи різноманітні вебсервіси та хмарні платформи [24,25,]. Водночас значна частина інформації має тимчасовий характер і потребує обмеження доступу після завершення певного періоду часу або виконання визначених умов [24].

Однією з актуальних проблем сучасних систем обміну даними є забезпечення контрольованого доступу до контенту. У багатьох випадках виникає необхідність передати інформацію таким чином, щоб вона була доступною лише обмежений проміжок часу, певну кількість переглядів або лише конкретному користувачеві. Подібні вимоги характерні для передачі конфіденційних документів, службових файлів, результатів тестування програмного забезпечення, технічних журналів та інших видів інформації, поширення яких повинно контролюватися [28, 30].

Аналіз існуючих сервісів показав, що більшість з них орієнтовані на постійне зберігання інформації та не забезпечують комплексного механізму тимчасового доступу до контенту [29]. Частина рішень підтримує лише текстові повідомлення, інші не дозволяють обмежувати кількість переглядів або не надають можливості самостійного розгортання програмного забезпечення. Наявні обмеження обумовлюють необхідність створення спеціалізованої вебсистеми, яка забезпечуватиме гнучке керування доступом до даних та підтримуватиме роботу з різними типами контенту.

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		10

Основною задачею розроблюваної системи є забезпечення можливості створення текстових записів та завантаження мультимедійних файлів з подальшим наданням доступу до них через унікальні посилання. При цьому система повинна підтримувати налаштування параметрів доступу, які визначають умови використання створеного контенту.

Для забезпечення унікальності створених посилань необхідно реалізувати механізм автоматичної генерації коротких ідентифікаторів записів. Сформовані ідентифікатори повинні використовуватися для створення URL-адрес доступу та забезпечувати швидке отримання необхідного контенту [11].

Важливою функціональною вимогою є підтримка механізму контролю терміну дії записів. Під час створення контенту користувач повинен мати можливість визначити час його доступності. Після завершення встановленого терміну система повинна автоматично блокувати доступ до запису або видаляти його відповідно до встановлених правил [24].

Додатково необхідно реалізувати механізм обмеження кількості переглядів контенту. Користувач повинен мати можливість встановлювати максимальну кількість відкриттів запису, після досягнення якої подальший доступ автоматично забороняється. Такий підхід дозволяє підвищити рівень захисту інформації та обмежити її неконтрольоване поширення [20,30].

Для забезпечення безпеки системи необхідно передбачити підтримку парольного захисту записів [8,20], а також механізм одноразового перегляду контенту. Це дозволить використовувати систему для передачі інформації, доступ до якої повинен бути максимально обмеженим.

Зберігання контенту та службових даних повинно здійснюватися з використанням сучасних засобів керування даними. Для цього доцільно використовувати реляційну базу даних для зберігання метаданих записів та зовнішнє об'єктне сховище для роботи з мультимедійними файлами [6].

Окремою задачею є реалізація програмного інтерфейсу взаємодії із системою. Наявність REST API забезпечить можливість інтеграції розробленого

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		11

програмного продукту з іншими інформаційними системами, вебзастосунками та засобами автоматизації.

Для роботи кінцевих користувачів необхідно реалізувати вебінтерфейс, який забезпечуватиме створення записів, завантаження файлів, налаштування параметрів доступу та перегляд опублікованого контенту. Інтерфейс повинен бути зрозумілим, адаптивним та забезпечувати зручну взаємодію з усіма функціями системи [3].

На основі проведеного аналізу сформовано функціональні вимоги до системи, які передбачають створення текстових записів, завантаження файлів, генерацію унікальних посилань доступу, налаштування терміну дії контенту, обмеження кількості переглядів, парольний захист записів, підтримку одноразового перегляду, керування користувачами та програмний доступ через REST API.

До основних нефункціональних вимог належать продуктивність, масштабованість, надійність зберігання даних, безпека доступу до інформації, зручність супроводу програмного забезпечення та можливість подальшого розширення функціональних можливостей системи [25,26]. Також система повинна забезпечувати коректну роботу в сучасних веббраузерах та підтримувати одночасну роботу значної кількості користувачів.

Таким чином, результатом розв'язання поставленої задачі має стати вебсистема тимчасового обміну мультимедійними даними, яка забезпечуватиме безпечне зберігання контенту, контрольований доступ до нього та можливість інтеграції із зовнішніми інформаційними системами.

1.2. Аналіз аналогів програмного продукту

Перед початком розроблення будь-якого програмного продукту доцільно провести аналіз існуючих рішень, що належать до аналогічного класу програмного забезпечення. Такий аналіз дозволяє визначити типовий функціонал

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		12

систем, виявити їх сильні та слабкі сторони, а також сформулювати перелік можливостей, які доцільно реалізувати у власній розробці [25].

Розроблювана система належить до класу сервісів тимчасового зберігання та обміну інформацією через вебінтерфейс. Основними функціями таких систем є створення текстових записів або завантаження файлів, генерація унікальних посилань доступу, керування терміном існування інформації та забезпечення швидкого доступу до опублікованого контенту [29]. Для проведення аналізу було обрано три популярні сервіси: Pastebin, GitHub Gist та Hastebin.

Pastebin є одним із найвідоміших вебсервісів для розміщення та обміну текстовою інформацією через мережу Інтернет. Сервіс було створено насамперед для програмістів, системних адміністраторів та інших фахівців, які потребують швидкого способу передавання фрагментів програмного коду, журналів роботи систем або конфігураційних файлів.

Основною функцією Pastebin є створення текстових записів (paste), кожен з яких отримує унікальне посилання. Після створення запису користувач може надіслати посилання іншій особі для перегляду вмісту [24]. Під час створення запису користувач має можливість обрати мову програмування, для якої буде застосовано синтаксичне підсвічування. Це значно підвищує зручність роботи з програмним кодом та дозволяє швидко аналізувати його структуру.

Важливою особливістю сервісу є підтримка налаштування часу життя запису. Користувач може обрати один із наперед визначених інтервалів, після завершення якого запис автоматично видаляється із системи. Такий механізм частково реалізує концепцію тимчасового доступу до інформації [29].

Крім цього, сервіс підтримує різні режими доступу до контенту. Запис може бути публічним, прихованим або приватним. Публічні записи доступні для пошуку іншими користувачами, тоді як приховані відкриваються лише за прямим посиланням.

Незважаючи на значну популярність та широкий набір можливостей для роботи з текстовим контентом, Pastebin має низку обмежень. Сервіс не підтримує

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		13

завантаження мультимедійних файлів, не дозволяє встановлювати максимальну кількість переглядів запису та не надає можливості самостійного розгортання на власному серверному обладнанні. Також доступ до API у повному обсязі доступний лише в платних тарифних планах. Отже, Pastebin забезпечує базовий функціонал для роботи з текстовими даними, проте не може повністю задовольнити вимоги до системи тимчасового обміну мультимедійним контентом [24,29].

GitHub Gist є спеціалізованим сервісом компанії GitHub, призначеним для зберігання та публікації фрагментів програмного коду. На відміну від Pastebin, даний сервіс тісно інтегрований з екосистемою GitHub та використовує механізми системи контролю версій Git. Користувач може створювати як публічні, так і приховані записи. Кожен Gist може містити декілька файлів одночасно, що дозволяє публікувати невеликі програмні проєкти або набори пов'язаних між собою файлів. Для кожного файлу автоматично визначається тип вмісту та застосовується відповідне синтаксичне підсвічування.

Однією з ключових переваг GitHub Gist є підтримка версіонування. Кожна зміна вмісту автоматично зберігається у вигляді окремої версії, що дозволяє переглядати історію редагувань та за необхідності повертатися до попередніх станів документа [33]. Така можливість є особливо корисною під час спільної роботи над програмним кодом.

Сервіс також підтримує механізм коментування та створення копій існуючих записів. Це забезпечує зручну взаємодію між користувачами та сприяє обміну знаннями в професійній спільноті розробників. Разом із тим GitHub Gist не підтримує автоматичне видалення контенту після завершення визначеного періоду часу. Всі записи зберігаються необмежений час до моменту їх ручного видалення власником. Крім того, відсутній механізм обмеження кількості переглядів контенту, що є однією з ключових вимог до розроблюваної системи.

Також сервіс орієнтований переважно на текстові файли та програмний код і не призначений для роботи з великими мультимедійними файлами. Для

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		14

використання більшості можливостей необхідна наявність облікового запису GitHub. Таким чином, GitHub Gist є потужним інструментом для зберігання та поширення програмного коду, однак його функціональність не дозволяє реалізувати контрольований тимчасовий доступ до контенту [29].

Hastebin є відкритим вебсервісом для швидкого обміну текстовою інформацією. Основною особливістю даного рішення є його мінімалістичний підхід до організації роботи користувача. Інтерфейс сервісу містить лише основні елементи, необхідні для створення та перегляду записів. Користувач може вставити текст у відповідне поле та миттєво отримати унікальне посилання для доступу до створеного запису. Завдяки простоті використання сервіс набув популярності серед системних адміністраторів та розробників програмного забезпечення.

Однією з важливих переваг Hastebin є відкритий вихідний код. Це дозволяє розгортати систему на власних серверах та модифікувати її відповідно до власних потреб. Відкритість програмного коду також спрощує аудит безпеки та інтеграцію з іншими програмними продуктами [28]. Сервіс підтримує базове синтаксичне підсвічування та надає простий API для створення та отримання записів.

Завдяки цьому Hastebin може використовуватися як компонент інших інформаційних систем. Водночас функціональні можливості сервісу є досить обмеженими. У ньому відсутня підтримка файлів, механізм обмеження кількості переглядів, налаштування терміну життя запису та засоби керування доступом до контенту. Крім того, стандартна реалізація не орієнтована на роботу з великими навантаженнями та значними обсягами даних. Таким чином, Hastebin може розглядатися як просте рішення для швидкого обміну текстовою інформацією, однак його функціональність є недостатньою для реалізації повноцінного сервісу тимчасового доступу до мультимедійного контенту [29].

Для узагальнення результатів дослідження було виконано порівняння розглянутих сервісів за основними функціональними характеристиками, зокрема підтримкою терміну дії контенту, обмеженням кількості переглядів, можливістю

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		15

роботи з різними типами даних та наявністю механізмів захисту інформації [24,29]. Проведений аналіз дозволив виявити переваги й недоліки кожного рішення, визначити рівень їхньої функціональності та сформулювати вимоги до розроблюваної вебсистеми. Результати порівняння наведено в таблиці 1.1.

Таблиця 1.1.

Таблиця порівняння аналогів

Критерій	Pastebin	GitHub Gist	Hastebin	Розроблювана система
Підтримка файлів	Ні	Так	Ні	Так
TTL (час життя)	Так	Ні	Так	Так
Ліміт переглядів	Ні	Ні	Ні	Так
REST API	Платний	Так	Ні	Так
Зовнішнє сховище (R2)	Ні	Ні	Ні	Так
Балансування навантаження	Ні	Ні	Ні	Так
Open Source	Ні	Ні	Так	Так

1.3. Вибір архітектури системи

Архітектура програмного забезпечення є одним із ключових аспектів під час проєктування інформаційних систем. Від обраного архітектурного підходу залежать продуктивність, масштабованість, надійність, зручність супроводу та можливість подальшого розвитку програмного продукту [25,26]. Правильно

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		16

обрана архітектура дозволяє ефективно організувати взаємодію між компонентами системи та забезпечити виконання функціональних і нефункціональних вимог.

Розроблювана вебсистема призначена для тимчасового зберігання та обміну мультимедійними даними із підтримкою обмеження часу доступу та кількості переглядів. Особливістю такої системи є необхідність одночасної роботи з вебінтерфейсом користувача, базою даних, файловим сховищем та програмним інтерфейсом API. Тому на етапі проєктування необхідно обрати архітектурне рішення, яке забезпечить достатню продуктивність та можливість подальшого масштабування системи [33].

Під час розроблення сучасних вебзастосунків найчастіше використовуються монолітна, мікросервісна та модульна монолітна архітектури. Кожен із зазначених підходів має власні переваги та недоліки, які необхідно враховувати під час вибору архітектури для конкретного програмного продукту.

Монолітна архітектура передбачає реалізацію всієї функціональності системи в межах одного програмного застосунку. У такій архітектурі всі модулі працюють як єдине ціле та розгортаються на сервері у вигляді одного програмного пакета. Основною перевагою монолітного підходу є простота розроблення та розгортання. Усі компоненти системи знаходяться в межах одного проєкту, що спрощує процес налагодження та тестування. Крім того, взаємодія між модулями здійснюється без додаткових мережевих викликів, що позитивно впливає на продуктивність [26].

Разом із тим монолітна архітектура має низку недоліків. Зі збільшенням розміру програмного продукту зростає складність його підтримки та внесення змін. Масштабування системи можливе лише шляхом розгортання додаткових копій усього застосунку, навіть якщо підвищене навантаження припадає лише на окремий модуль. Для невеликих проєктів монолітна архітектура є ефективним рішенням, проте для систем, які передбачають подальший розвиток та масштабування, її можливостей може бути недостатньо.

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		17

Мікросервісна архітектура передбачає поділ системи на набір незалежних сервісів, кожен із яких виконує окрему функцію та взаємодіє з іншими сервісами через мережеві протоколи [25,26]. Основною перевагою даного підходу є можливість незалежного масштабування окремих компонентів. Наприклад, у разі значного навантаження на модуль обробки файлів можна збільшити кількість його екземплярів без зміни інших частин системи. Крім того, різні сервіси можуть використовувати різні технології реалізації, бази даних та механізми зберігання інформації. Це забезпечує високу гнучкість під час розроблення великих програмних комплексів.

Недоліком мікросервісної архітектури є складність її реалізації та підтримки [25]. Для роботи такої системи необхідно використовувати додаткові механізми балансування навантаження, моніторингу, контейнеризації та оркестрації сервісів. Також зростає кількість мережевих запитів між компонентами системи, що може негативно впливати на продуктивність.

Зважаючи на порівняно невеликий масштаб розроблюваної системи, використання повноцінної мікросервісної архітектури є надмірно складним рішенням. Модульна монолітна архітектура є компромісним рішенням між класичним монолітом та мікросервісами. У межах такого підходу система розробляється як єдиний застосунок, проте її внутрішня структура поділяється на незалежні функціональні модулі. Кожен модуль має чітко визначену відповідальність та взаємодіє з іншими компонентами через визначені інтерфейси. Це забезпечує кращу підтримуваність програмного коду та спрощує його подальшу модернізацію [26,33].

Головною перевагою модульного моноліту є можливість у майбутньому виділити окремі модулі в самостійні мікросервіси без суттєвої зміни архітектури системи [25]. Таким чином забезпечується баланс між простотою реалізації та можливістю подальшого масштабування. Для систем середнього рівня складності модульна монолітна архітектура вважається одним із найбільш раціональних рішень.

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		18

Для вибору оптимального архітектурного підходу було проведено порівняння розглянутих архітектур за основними критеріями, представленими у таблиці 1.2. Порівняння виконувалося за показниками масштабованості, продуктивності, складності реалізації, зручності супроводу та можливості подальшого розширення функціональності системи. Отримані результати стали основою для обґрунтування архітектурного рішення, яке найбільш повно відповідає вимогам розроблюваної вебсистеми [25,26].

Таблиця 1.2.

Порівняння архітектурних підходів

Критерій	Моноліт	Модульний моноліт	Мікросервіси
Простота розроблення	Висока	Висока	Низька
Простота розгортання	Висока	Висока	Середня
Масштабованість	Низька	Середня	Висока
Складність підтримки	Середня	Низька	Висока
Вартість інфраструктури	Низька	Низька	Висока
Готовність до подальшого розвитку	Середня	Висока	Висока

Як видно з таблиці 1.5, модульна монолітна архітектура забезпечує оптимальне співвідношення між складністю реалізації та можливістю подальшого розвитку системи. Саме тому даний підхід було обрано для розроблення програмного комплексу.

З урахуванням функціональних та нефункціональних вимог до розроблюваної вебсистеми було прийнято рішення використовувати модульну

монолітну архітектуру. Даний підхід поєднує переваги монолітних та мікросервісних систем, забезпечуючи простоту розроблення і розгортання програмного забезпечення та одночасно створюючи передумови для подальшого масштабування окремих компонентів.

Обрана архітектура передбачає логічний поділ системи на декілька взаємопов'язаних компонентів. Користувач взаємодіє із системою через клієнтський вебінтерфейс, який забезпечує створення записів, завантаження файлів та перегляд опублікованого контенту. Обробка всіх запитів здійснюється серверною частиною, яка реалізує бізнес-логіку системи, перевірку обмежень доступу, взаємодію з базою даних та файловим сховищем [6]. Загальну архітектуру розроблюваної системи наведено на рисунку 1.1.



Рис. 1.1. Загальна архітектура розроблюваної системи

Як видно з рисунка 1.1, центральним елементом архітектури є серверний застосунок, який виконує роль посередника між користувацьким інтерфейсом, базою даних та зовнішнім файловим сховищем [5,6]. Після створення нового запису користувачем серверний компонент формує унікальний ідентифікатор, зберігає метадані в базі даних PostgreSQL та передає вміст файлу або текстового повідомлення до хмарного сховища Cloudflare R2 [6].

Для забезпечення швидкого доступу до контенту використовується механізм генерації коротких унікальних посилань. Перед наданням доступу до інформації сервер перевіряє встановлені обмеження, зокрема час життя запису та

кількість доступних переглядів. У разі порушення хоча б однієї з умов доступ до контенту блокується.

Запропонований підхід забезпечує чітке розмежування відповідальності між компонентами системи, спрощує підтримку програмного забезпечення та створює можливість поступового переходу до мікросервісної архітектури в майбутньому без необхідності повного перепроєктування програмного комплексу.

Укрупнена структура розроблюваного програмного комплексу визначає основні функціональні підсистеми та їх взаємозв'язки. Виділення окремих підсистем дозволяє спростити процес проєктування, реалізації та подальшого супроводу програмного забезпечення. На рисунку 1.2. наведено укрупнену структуру програмного комплексу.



Рис. 1.2. Укрупнена структура програмного комплексу

Згідно зі структурною схемою програмний комплекс складається з підсистеми керування контентом, підсистеми контролю доступу, підсистеми роботи з файлами, підсистеми взаємодії через програмний інтерфейс та підсистеми зберігання даних. Підсистема керування контентом забезпечує створення, отримання та видалення записів користувачів. Саме вона реалізує основні сценарії роботи із текстовими повідомленнями та мультимедійними файлами. Підсистема контролю доступу відповідає за перевірку часових обмежень та лімітів переглядів, що є ключовою особливістю розроблюваної системи.

Зберігання безпосереднього вмісту файлів здійснюється окремою підсистемою роботи зі сховищем Cloudflare R2 [6]. Такий підхід дозволяє відокремити великі обсяги мультимедійних даних від метаданих та зменшити навантаження на базу даних. Метадані записів зберігаються в PostgreSQL і використовуються для пошуку контенту, перевірки прав доступу та контролю життєвого циклу записів.

Взаємодія між клієнтською частиною та сервером реалізується через REST API, яке забезпечує стандартизований спосіб обміну даними між компонентами системи [27,33]. Використання такого підходу дозволяє в майбутньому створювати додаткові клієнтські застосунки без зміни серверної логіки. Таким чином, укрупнена структура програмного комплексу забезпечує логічне розділення функціональності між окремими підсистемами, що позитивно впливає на підтримуваність, масштабованість та надійність розроблюваної вебсистеми.

1.4. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

Під час розроблення сучасних вебсистем важливим етапом є вибір інструментальних засобів реалізації. Від правильності вибору технологічного стеку залежать продуктивність, масштабованість, надійність та зручність подальшого супроводу програмного продукту. Для розроблюваної вебсистеми тимчасового обміну мультимедійними даними було обрано сучасний набір програмних засобів, який забезпечує ефективну реалізацію всіх функціональних вимог системи.

Для реалізації клієнтської та серверної частин програмного комплексу було обрано мову програмування TypeScript. Дана мова є надбудовою над JavaScript та розширює його можливості шляхом впровадження статичної типізації, інтерфейсів, узагальнених типів та інших механізмів об'єктно-орієнтованого програмування [4]. Використання TypeScript дозволяє виявляти значну кількість помилок ще на етапі компіляції, що позитивно впливає на якість програмного коду та знижує витрати часу на тестування й супровід програмного забезпечення.

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		22

Крім того, TypeScript забезпечує можливість спільного використання моделей даних, інтерфейсів та структур між клієнтською та серверною частинами системи, що зменшує дублювання коду та спрощує підтримку програмного продукту [4].

Ще однією важливою перевагою є широка популярність мови серед розробників та велика кількість доступних бібліотек і фреймворків. Завдяки цьому значно спрощується інтеграція із зовнішніми сервісами та прискорюється процес розробки програмного забезпечення. Для реалізації серверної частини системи було обрано платформу Node.js. Дана технологія являє собою середовище виконання JavaScript-коду поза браузером і побудована на основі високопродуктивного рушія Google V8 [1]. Основною перевагою Node.js є використання асинхронної моделі введення-виведення, що дозволяє ефективно обробляти велику кількість одночасних запитів без створення окремого потоку виконання для кожного користувача. Такий підхід особливо ефективний для вебсистем, які активно взаємодіють із базами даних, файловими сховищами та зовнішніми API [1,32].

Як вебфреймворк для реалізації REST API було обрано Fastify. Fastify є одним із найпродуктивніших серверних фреймворків для Node.js та забезпечує високу швидкість обробки HTTP-запитів [2,31]. Крім того, фреймворк підтримує автоматичну валідацію вхідних даних, генерацію документації OpenAPI та модульну структуру застосунку. Використання Node.js у поєднанні з Fastify дозволяє створити продуктивний та масштабований серверний компонент, який забезпечуватиме стабільну роботу вебсервісу навіть за значного навантаження. Для реалізації користувацького інтерфейсу було обрано бібліотеку React. React є однією з найпоширеніших технологій для розроблення односторінкових вебзастосунків та активно використовується у великій кількості комерційних проєктів [3].

Основною особливістю React є компонентний підхід до побудови інтерфейсів. Кожен елемент користувацького інтерфейсу реалізується у вигляді окремого компонента, який може використовуватися повторно в різних частинах

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		23

системи. Це підвищує модульність застосунку та спрощує його подальший розвиток. Також React використовує віртуальну DOM-модель, що забезпечує ефективне оновлення інтерфейсу та високу швидкодію застосунку [3,10]. Завдяки цьому користувач отримує швидкий відгук системи навіть під час виконання великої кількості операцій. Використання React у поєднанні з TypeScript забезпечує високу надійність коду та дозволяє створити сучасний інтерфейс користувача з можливістю подальшого масштабування функціональності.

Для зберігання метаданих було обрано PostgreSQL. Дана система керування базами даних є однією з найбільш функціональних та надійних реляційних СКБД з відкритим вихідним кодом. PostgreSQL підтримує транзакції відповідно до принципів ACID, що гарантує цілісність даних навіть у випадках одночасного доступу великої кількості користувачів [5]. Це є особливо важливим для реалізації механізму підрахунку переглядів контенту, де необхідно забезпечити коректне оновлення лічильників. Крім того, PostgreSQL підтримує індексацію, тригери, збережені процедури та інші механізми оптимізації роботи з даними. Завдяки цьому забезпечується висока швидкість пошуку записів за унікальним ідентифікатором та ефективна робота системи в цілому [5].

У базі даних зберігатимуться метадані контенту: унікальний ключ доступу, тип контенту, час створення, дата завершення дії, максимальна кількість переглядів, поточна кількість переглядів та посилання на файл у зовнішньому сховищі. Для зберігання мультимедійного контенту було обрано хмарне об'єктне сховище Cloudflare R2. Даний сервіс забезпечує масштабоване та надійне зберігання файлів різних типів без необхідності використання власної файлової інфраструктури [6]. Однією з головних переваг Cloudflare R2 є відсутність плати за вихідний трафік, що дозволяє суттєво зменшити витрати на експлуатацію системи порівняно з альтернативними рішеннями. Крім того, сервіс підтримує сумісність з Amazon S3 API, що спрощує інтеграцію із серверною частиною застосунку [6,7].

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		24

Використання окремого файлового сховища дозволяє зменшити навантаження на базу даних та забезпечити ефективне зберігання великих обсягів мультимедійного контенту. Для забезпечення стабільної роботи вебсистеми необхідно визначити мінімальні та рекомендовані характеристики серверного обладнання.

Для забезпечення стабільної роботи розробленої вебсистеми були визначені мінімальні вимоги до апаратного забезпечення серверного середовища. Основні характеристики апаратних ресурсів, необхідних для розгортання та функціонування програмного комплексу, наведено в таблиці 1.3.

Таблиця 1.3.

Вимоги до апаратного забезпечення

Параметр	Мінімальні вимоги	Рекомендовані вимоги
Процесор	2 ядра, 2.0 ГГц	4 ядра, 3.0 ГГц
Оперативна пам'ять	4 ГБ	8 ГБ
Дисковий простір	20 ГБ SSD	100 ГБ SSD
Мережа	100 Мбіт/с	1 Гбіт/с
ОС	Ubuntu 20.04 LTS	Ubuntu 22.04 LTS

Наведені характеристики є достатніми для забезпечення стабільної роботи системи та підтримки необхідного рівня продуктивності під час обробки користувацьких запитів. Таким чином, обраний технологічний стек у складі TypeScript, React, Node.js, Fastify, PostgreSQL та Cloudflare R2 повністю відповідає функціональним та нефункціональним вимогам до розроблюваної вебсистеми. Використання зазначених технологій забезпечує високу продуктивність, масштабованість, надійність та зручність подальшого супроводу програмного забезпечення.

Висновки до першого розділу

У першому розділі проведено дослідження предметної області та визначено основні напрями використання інформаційних технологій для організації тимчасового доступу до мультимедійного контенту. Сформульовано мету роботи, визначено функціональні та нефункціональні вимоги до програмного продукту та встановлено перелік задач, що підлягають автоматизації.

Проведений аналіз існуючих аналогів дозволив визначити основний функціонал систем даного класу та виявити їх обмеження. Встановлено, що жодне з розглянутих рішень не забезпечує одночасної підтримки TTL, обмеження кількості переглядів, роботи з мультимедійними файлами, відкритого REST API та масштабованого файлового сховища.

На основі порівняння архітектурних підходів обрано архітектуру модульного моноліту, яка забезпечує баланс між простотою реалізації та можливістю подальшого масштабування системи. Також обґрунтовано вибір технологічного стеку та визначено вимоги до апаратного забезпечення для розгортання розроблюваного програмного комплексу.

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		26

РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ ВЕБСИСТЕМИ ТИМЧАСОВОГО ОБМІНУ МУЛЬТИМЕДІЙНИМИ ДАНИМИ

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

Проєктування будь-якої програмної системи розпочинається з детального аналізу вимог, які визначають функціональні можливості майбутнього програмного продукту та якісні характеристики його роботи. Саме на етапі формування вимог закладаються основні принципи функціонування системи, визначаються її межі, сценарії використання та критерії ефективності. Ретельний аналіз вимог дозволяє мінімізувати кількість помилок під час розробки та забезпечує відповідність кінцевого результату поставленим завданням [25,26].

Предметною областю даного дослідження є організація тимчасового обміну мультимедійними даними між користувачами через мережу Інтернет. На відміну від традиційних файлових сховищ, розроблювана система орієнтована на тимчасове зберігання інформації та автоматичне обмеження доступу до неї після виконання визначених умов. Такий підхід дозволяє підвищити рівень конфіденційності переданої інформації та суттєво зменшити обсяг даних, що постійно зберігаються на сервері, що є важливою перевагою порівняно з хмарними сховищами загального призначення.

Функціональні вимоги визначають перелік операцій, які повинна виконувати система під час взаємодії з користувачами. Основною функцією вебсервісу є створення та публікація контенту з можливістю автоматичного обмеження доступу до нього. Користувач повинен мати можливість створювати текстові записи довільного обсягу та завантажувати файли різних форматів, включаючи зображення, відео, аудіозаписи, документи та архіви. При цьому система повинна підтримувати як завантаження одного файлу, так і створення колекцій із декількох файлів, що об'єднуються спільним посиланням.

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		27

Важливою вимогою є підтримка запису голосових повідомлень безпосередньо через веббраузер без встановлення додаткового програмного забезпечення. Реалізація такої можливості дозволяє користувачам створювати аудіоконтент у зручний спосіб прямо у вікні браузера. Після завершення запису система автоматично формує аудіофайл та зберігає його разом з іншими типами контенту, що забезпечує однорідну обробку всіх медіаформатів у межах єдиного сервісу.

Одним із ключових механізмів системи є управління доступом до створеного контенту. Для кожного запису повинен автоматично генеруватися унікальний короткий ідентифікатор, який використовується при формуванні публічного посилання на ресурс. Користувач має можливість встановлювати час життя запису, після завершення якого доступ до нього автоматично блокується, а також визначати максимальну кількість переглядів та активувати режим автоматичного видалення після першого відкриття.

З метою підвищення рівня безпеки система повинна підтримувати захист окремих записів паролем, що встановлюється автором під час створення запису. У цьому випадку перед отриманням доступу до контенту будь-який користувач, що відкриває посилання, повинен пройти перевірку введеного пароля. Реалізація такого механізму дозволяє значно знизити ризик несанкціонованого доступу навіть у випадку випадкового поширення посилання на запис серед небажаних осіб.

Окремий блок функціональних вимог стосується автентифікації та авторизації користувачів системи. Система повинна забезпечувати реєстрацію нових облікових записів із перевіркою унікальності адреси електронної пошти та дотриманням мінімальних вимог до складності пароля. Після успішної автентифікації користувач отримує доступ до особистого кабінету, де може переглядати створений контент, керувати його параметрами, видаляти власні записи та продовжувати термін їх дії.

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		28

Для забезпечення адміністративного контролю передбачено окрему роль адміністратора з розширеним набором привілеїв. Користувач із такими правами має доступ до статистики роботи системи, інформації про зареєстрованих користувачів, інструментів зміни ролей та модерації контенту. Усі основні функції сервісу повинні бути доступними через REST API, що забезпечує можливість інтеграції із зовнішніми програмними продуктами, а документування API за допомогою OpenAPI та Swagger спрощує його використання сторонніми розробниками [27].

Крім функціональних вимог, під час проєктування було сформовано набір нефункціональних вимог, що визначають якісні характеристики системи. Вони безпосередньо впливають на надійність, продуктивність і безпеку програмного забезпечення в умовах реальної експлуатації. Виконання нефункціональних вимог є необхідною умовою забезпечення стабільної та комфортної роботи вебсервісу при зростаючій кількості користувачів.

Однією з найважливіших нефункціональних характеристик є продуктивність системи під навантаженням. Час обробки типових запитів повинен залишатися мінімальним навіть при одночасній роботі великої кількості користувачів, а вебінтерфейс повинен швидко завантажуватися та забезпечувати комфортну взаємодію. Для досягнення необхідного рівня продуктивності обрано фреймворк Fastify, який є одним із найшвидших Node.js-фреймворків завдяки оптимізованій системі маршрутизації та серіалізації відповідей [1,2].

Особлива увага приділяється надійності зберігання інформації та стійкості системи до відмов окремих компонентів. Метадані контенту зберігаються у реляційній базі даних PostgreSQL, тоді як самі файли розміщуються у хмарному об'єктному сховищі Cloudflare R2, що забезпечує географічно розподілене резервне зберігання. Подібне розділення дозволяє підвищити швидкодію системи та забезпечити масштабованість при збільшенні кількості користувачів і завантажуваних файлів.

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		29

Для захисту персональних даних та інформації про користувачів застосовуються сучасні криптографічні механізми відповідно до поточних галузевих стандартів. Паролі зберігаються виключно у вигляді хешів, обчислених за алгоритмом bcrypt із достатньою вартістю хешування, а авторизація реалізована за допомогою короткоживучих JWT-токенів [8]. Додатково використовується механізм refresh token із зберіганням у захищеній HTTP-only cookie, що дозволяє підтримувати безпечні сесии роботи без необхідності повторного введення облікових даних та мінімізує ризики XSS-атак [19].

На основі аналізу функціональних вимог було визначено три основні категорії користувачів, які взаємодіють із системою. Кожна категорія має власний набір прав доступу та виконує різні функції під час роботи із сервісом. Розподіл користувачів на ролі дозволяє забезпечити належний рівень безпеки та ефективно організувати доступ до функціональних можливостей відповідно до принципу мінімально необхідних привілеїв.

Першим актором є анонімний користувач – відвідувач вебсервісу, який не проходив процедуру реєстрації або входу до системи. Незважаючи на відсутність облікового запису, таким користувачам доступний базовий функціонал: створення текстових записів, завантаження файлів та перегляд контенту за посиланням. Водночас записи, створені анонімним користувачем, не прив'язуються до жодного облікового запису і не можуть бути керовані або видалені після завершення процедури створення, що є свідомим обмеженням, яке спонукає до реєстрації.

Другим актором системи є зареєстрований (автентифікований) користувач, який пройшов процедуру реєстрації та отримав персональний обліковий запис. Після успішної авторизації він отримує доступ до особистого кабінету, де відображаються всі створені ним записи, їх поточний статус, кількість переглядів та дата завершення терміну дії. Крім перегляду статистики, він може видаляти власні записи, продовжувати термін їх дії та керувати параметрами доступу.

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		30

Третім актором є адміністратор системи – роль, призначена для здійснення контролю за роботою вебсервісу та виконання адміністративних операцій. До основних функцій адміністратора належить перегляд статистичних показників роботи системи, отримання інформації про зареєстрованих користувачів та управління їх ролями. Адміністратор також має право переглядати і видаляти будь-який контент незалежно від його власника, що забезпечує можливість оперативного реагування на порушення правил використання сервісу.

Розмежування доступу між трьома категоріями забезпечує реалізацію принципу мінімально необхідних привілеїв, згідно з яким кожен суб'єкт системи отримує лише ті права, що необхідні для виконання його функцій [25,26]. Такий підхід знижує потенційні ризики безпеки: навіть у разі компрометації облікового запису зловмисник не зможе отримати доступ до функцій, що не входять до дозволеного набору операцій для відповідної ролі. Побудована модель ролей є розширюваною і дозволяє у майбутньому додавати нові ролі без значних змін у архітектурі системи.

На основі сформованих функціональних вимог та визначених акторів побудовано діаграму варіантів використання, яка наведена на рисунку 2.1. Дана діаграма відображає перелік функцій системи, зв'язки між ними, а також демонструє взаємодію різних категорій користувачів із програмним забезпеченням [24]. Стрілки включення (include) та розширення (extend) між варіантами використання показують залежності між сценаріями і допомагають визначити спільні під потоки, що повторно використовуються у декількох основних сценаріях.

Діаграма містить сім основних варіантів використання, які охоплюють усі ключові функції вебсервісу: створення текстового запису, завантаження файлів, перегляд контенту за посиланням, реєстрація, авторизація, продовження терміну дії запису та адміністрування системи. Кожен варіант використання пов'язаний з одним або кількома акторами відповідно до їх рівня доступу. Наявність спільних варіантів для анонімного та зареєстрованого користувачів підкреслює, що базова

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		31

функціональність системи доступна без обов'язкової реєстрації, що знижує бар'єр входу для нових користувачів.

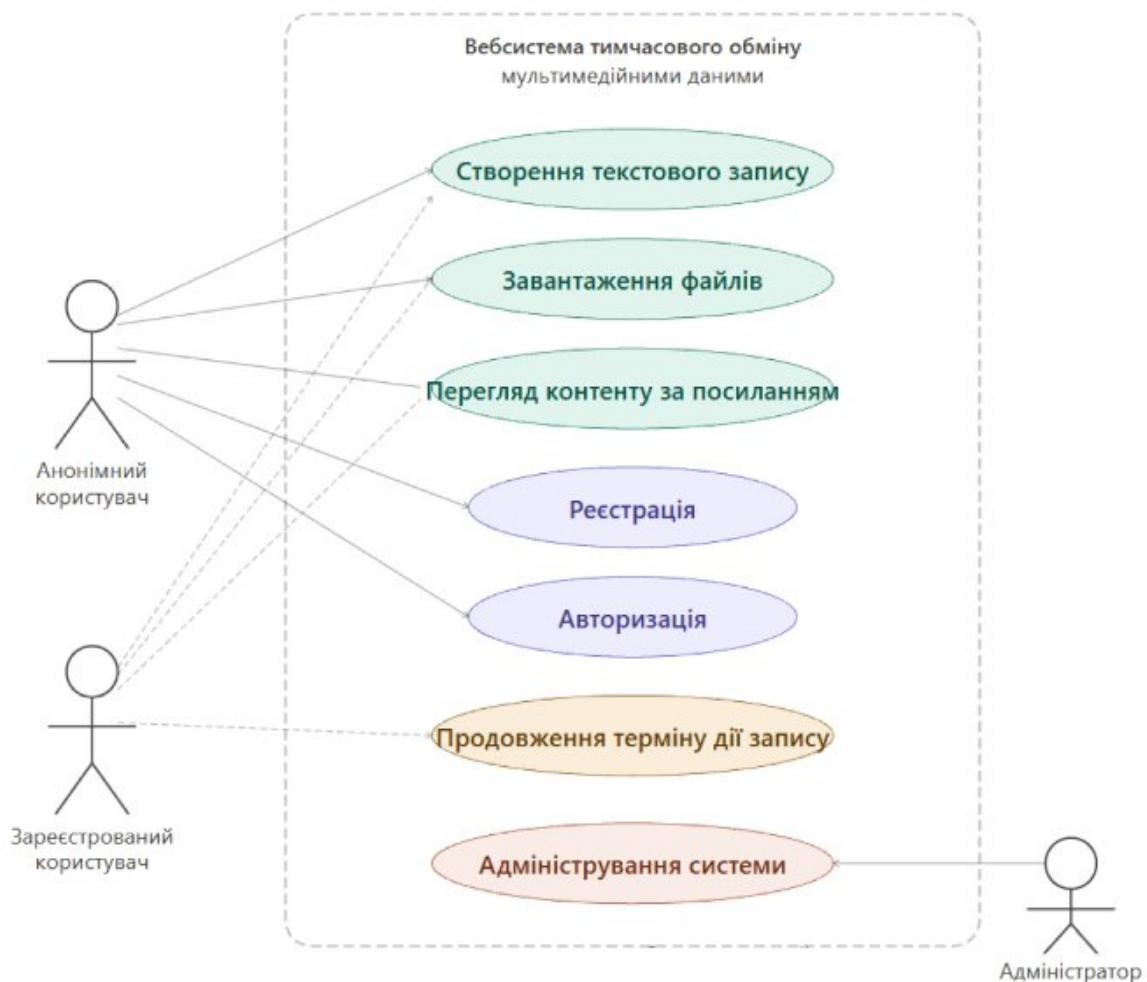


Рис. 2.1. Діаграма варіантів використання вебсистеми тимчасового обміну мультимедійними даними

Варіант використання «Створення текстового запису» доступний як анонімним, так і зареєстрованим користувачам. Під час виконання цього сценарію користувач вводить текстове повідомлення у форму вебінтерфейсу та опціонально встановлює параметри доступу: термін дії запису, максимальну кількість переглядів, пароль або режим автоматичного видалення після першого перегляду. Після надсилання форми система перевіряє коректність введених даних, формує унікальний ідентифікатор запису, зберігає текстовий вміст у зовнішньому сховищі та повертає клієнту коротке посилання [6].

Варіант використання «Завантаження файлів» також доступний анонімним і зареєстрованим користувачам та підтримує завантаження файлів різних форматів: зображень, відео, аудіозаписів, документів та архівів. Система передбачає можливість завантаження як одного файлу, так і створення колекцій із декількох файлів, об'єднаних спільним посиланням. Під час завантаження система перевіряє допустимий розмір файлу та тип MIME, зберігає файл у хмарному сховищі Cloudflare R2 і фіксує метадані у базі даних [6].

Варіант використання «Перегляд контенту за посиланням» доступний усім категоріям користувачів, оскільки є основною публічною функцією сервісу. Під час відкриття унікального посилання система послідовно перевіряє термін дії запису, поточну кількість переглядів та наявність паролю; якщо всі умови виконані, лічильник переглядів збільшується і вміст надається користувачу. Якщо активовано режим burn-after-read, запис автоматично позначається як видалений після першого успішного перегляду, що гарантує одноразовість доступу.

Варіант використання «Адміністрування системи» доступний лише адміністратору і включає перегляд загальної статистики системи, управління обліковими записами користувачів, зміну ролей та модерацію контенту. Наявність виділеного адміністративного функціоналу дозволяє підтримувати стабільну роботу вебсервісу та своєчасно реагувати на порушення правил використання. Адміністративний інтерфейс реалізований як окремий захищений розділ застосунку, доступ до якого перевіряється на рівні серверних маршрутів.

Об'єктно-орієнтоване проектування системи базується на принципах розподілу відповідальностей між компонентами та забезпечення низької зв'язаності коду [26]. Архітектура побудована за шаблоном Repository, де кожен модуль виконує чітко визначену функцію: сервіси реалізують бізнес-логіку, репозиторії відповідають за взаємодію з базою даних, а допоміжні класи забезпечують роботу зі сховищем файлів та генерацію ідентифікаторів. Такий підхід спрощує тестування окремих компонентів та підвищує підтримуваність коду в процесі подальшого розвитку системи.

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		33

Діаграма класів системи наведена на рисунку 2.2 і відображає структуру основних класів, їх атрибути, методи та відносини між ними. Центральне місце в архітектурі займає клас `ContentService`, який координує роботу трьох залежних компонентів: `ContentRepository`, `StorageService` та `HashGenerator`. Залежності між класами реалізовані через ін'єкцію залежностей, що забезпечує можливість підміни конкретних реалізацій у тестовому середовищі.

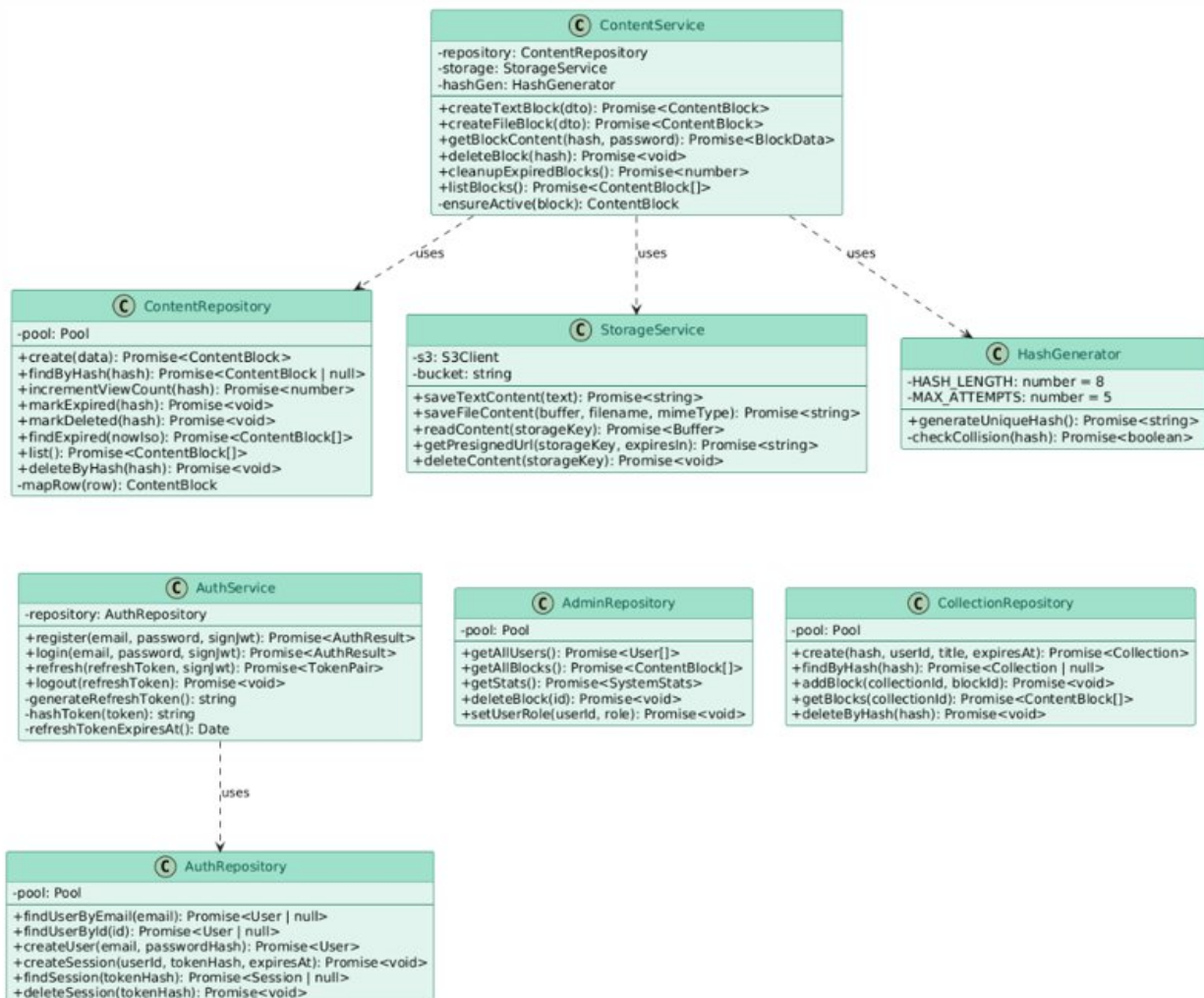


Рис. 2.2. Діаграма класів системи тимчасового обміну мультимедійними даними

Клас `ContentService` є центральним сервісом системи та реалізує основну бізнес-логіку роботи з контентом, приймаючи залежності від `ContentRepository`, `StorageService` та `HashGenerator`. Основні методи класу включають `createTextBlock()` та `createFileBlock()` для створення записів різних типів, `getBlockContent()` для отримання вмісту з перевіркою умов доступу, `deleteBlock()` для видалення, а також `cleanupExpiredBlocks()` для автоматичного очищення

застарілих записів. Метод `ensureActive()` виконує перевірку актуальності запису перед наданням доступу до нього, перевіряючи статус, термін дії та кількість переглядів [25].

Клас `ContentRepository` відповідає за взаємодію з таблицею `content_blocks` у базі даних і отримує пул з'єднань PostgreSQL через атрибут `pool`. Метод `create()` зберігає новий запис, `findByHash()` знаходить запис за унікальним ідентифікатором, `incrementViewCount()` збільшує лічильник переглядів, а `markExpired()` та `markDeleted()` змінюють статус запису відповідним чином. Допоміжний приватний метод `mapRow()` забезпечує перетворення рядка результату SQL-запиту на типізований об'єкт доменної моделі `ContentBlock` [5].

Клас `StorageService` реалізує взаємодію з об'єктним сховищем через AWS S3-сумісний клієнт, що дозволяє працювати з Cloudflare R2 за допомогою стандартного SDK без прив'язки до конкретного провайдера. Атрибут `s3` містить екземпляр `S3Client`, налаштований відповідними ключами доступу та ендпоінтом, а атрибут `bucket` зберігає назву контейнера. Метод `saveTextContent()` зберігає текстовий вміст у вигляді рядка, `saveFileContent()` – бінарні дані файлу разом із типом MIME, а `getPresignedUrl()` генерує тимчасове підписане посилання для прямого доступу клієнта до файлу без проксювання через сервер [6].

Клас `HashGenerator` забезпечує генерацію унікальних коротких ідентифікаторів для записів і є ізольованим компонентом без побічних ефектів, окрім звернень до бази даних для перевірки колізій. Константа `HASH_LENGTH` визначає довжину ідентифікатора (8 символів), що забезпечує достатній простір можливих значень для практичних потреб системи, а `MAX_ATTEMPTS` обмежує кількість спроб генерації (5) для запобігання нескінченному циклу. Метод `generateUniqueHash()` виконує генерацію рядка та перевіряє його унікальність у базі даних через метод `checkCollision()`, повторюючи процес у разі виявлення дублікату.

Взаємодія класів у системі побудована відповідно до принципів багаторівневої архітектури, що забезпечує чіткий напрямок залежностей між

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		35

рівнями. Користувачькі запити надходять до маршрутів Fastify, які передають їх до сервісного рівня, сервіси реалізують бізнес-логіку та взаємодіють з репозиторіями для отримання або збереження даних, а репозиторії виконують SQL-запити до PostgreSQL та повертають результати у вигляді об'єктів доменної моделі. Такий підхід дозволяє ізолювати бізнес-логіку від деталей реалізації сховища та бази даних, що спрощує підтримку і подальший розвиток програмного продукту без необхідності внесення змін у кілька шарів одночасно.

Клас AuthService реалізує логіку автентифікації та авторизації користувачів, залежачи від AuthRepository для доступу до даних користувачів і сесій. Методи register() та login() обробляють відповідні сценарії входу, refresh() оновлює JWT-токени за допомогою refresh token, а logout() завершує сеанс роботи шляхом видалення відповідного запису з таблиці sessions. Допоміжні приватні методи generateRefreshToken(), hashToken() та refreshTokenExpiresAt() інкапсулюють деталі реалізації механізму токенів і не доступні ззовні класу.

Клас AuthRepository забезпечує взаємодію з таблицями users та sessions у базі даних, надаючи методи для всіх необхідних операцій з обліковими записами та сесіями. Методи findUserByEmail() та findUserById() дозволяють знаходити користувачів за різними критеріями пошуку, метод createUser() реєструє нового користувача з хешованим паролем, а методи createSession(), findSession() та deleteSession() забезпечують повний цикл управління активними сесіями. Клас AdminRepository надає методи для адміністративного управління системою, зокрема getAllUsers(), getAllBlocks(), getStats(), deleteBlock() та setUserRole(), що забезпечують усі необхідні функції панелі адміністратора.

Клас CollectionRepository відповідає за управління колекціями файлів та взаємодію з таблицями collections та collection_blocks. Метод create() створює нову колекцію із заданими параметрами та повертає об'єкт створеної колекції, метод findByHash() знаходить колекцію за публічним ідентифікатором разом із усіма пов'язаними записами. Метод addBlock() додає запис до колекції через проміжну таблицю collection_blocks, використовуючи запит INSERT з перевіркою

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		36

на дублікати, а `getBlocks()` повертає всі записи, що входять до колекції, з їх повними метаданими.

2.2. Розробка бази даних системи

Під час проектування бази даних було застосовано принципи нормалізації – процесу організації структури реляційної бази даних з метою усунення надлишковості даних та запобігання аномаліям під час виконання операцій вставки, оновлення та видалення. Нормалізація дозволяє забезпечити несуперечливість збережених даних та полегшити підтримку схеми при внесенні змін у процесі розвитку системи. Від ступеня нормалізації залежить як цілісність даних, так і ефективність виконання запитів на читання та запис.

Таблиці `users`, `content_blocks`, `sessions`, `collections` та `access_logs` спроектовано таким чином, щоб уникнути дублювання даних та забезпечити цілісність інформації на рівні СУБД. Кожна таблиця зберігає лише ті атрибути, які безпосередньо описують сутність, що вона представляє: наприклад, таблиця `users` містить виключно дані облікового запису, а не повторює у своїх рядках метадані контенту чи параметри сесій. Метадані контенту, сесій та журналу доступу зберігаються в окремих таблицях і пов'язані з `users` через зовнішні ключі, що унеможливорює появу несинхронізованих копій одних і тих самих даних [5].

Структура бази даних відповідає першій нормальній формі (1НФ), оскільки кожен атрибут кожного запису є атомарним значенням без повторюваних груп. Другій нормальній формі (2НФ) структура відповідає завдяки тому, що кожен неключовий атрибут повністю функціонально залежить від первинного ключа таблиці, а не лише від його частини; виняток становить таблиця `collection_blocks` із складеним первинним ключем, де обидва атрибути є складовими ключа і не мають власних неключових атрибутів [5]. Третій нормальній формі (3НФ) структура відповідає, оскільки всі неключові атрибути функціонально залежать лише від первинного ключа відповідної таблиці й не мають транзитивних залежностей між собою, що виключає аномалії оновлення.

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		37

Дотримання принципів нормалізації під час проектування дозволило уникнути надлишкового зберігання даних та забезпечити несуперечливість інформації при виконанні операцій оновлення та видалення. Водночас для критично важливих операцій читання, що виконуються найчастіше, зокрема пошук запису за хешем, передбачено створення відповідних унікальних індексів, що компенсує можливі витрати продуктивності, пов'язані з нормалізованою структурою. Такий збалансований підхід дозволяє одночасно підтримувати цілісність даних та досягати прийнятної швидкодії системи при реальному навантаженні.

База даних системи реалізована засобами PostgreSQL та включає шість таблиць, кожна з яких відповідає за зберігання певної категорії даних і пов'язана з іншими через зовнішні ключі [5]. Між таблицями встановлено зв'язки, що забезпечують референційну цілісність даних на рівні СУБД – неможливість збереження запису з посиланням на неіснуючий первинний ключ. Схема бази даних наведена на рисунку 2.3.

Зв'язок «Користувач → Блок контенту» (users → content_blocks) реалізує відношення один-до-багатьох: один зареєстрований користувач може мати довільну кількість записів контенту, однак кожен запис прив'язаний не більше ніж до одного облікового запису. Зовнішній ключ user_id у таблиці content_blocks посиляється на поле id таблиці users і допускає значення NULL, що дозволяє зберігати записи, створені анонімними користувачами, без порушення цілісності даних. При видаленні облікового запису користувача можливе каскадне обнулення поля user_id, що дозволяє зберегти анонімізовані записи після деактивації облікового запису.

Зв'язок «Користувач → Сесія» (users → sessions) також реалізує відношення один-до-багатьох: один користувач може мати одночасно кілька активних сесій із різних пристроїв або браузерів, кожна з яких відповідає окремому refresh-токену. Зовнішній ключ user_id у таблиці sessions посиляється на id таблиці users, а кожна сесія зберігає хеш refresh-токена та термін його дії для безпечного оновлення JWT-

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		38

токенів. Зберігання хешу токена замість оригінального значення гарантує, що навіть у разі несанкціонованого доступу до бази даних зломисник не зможе використати токени для автентифікації.

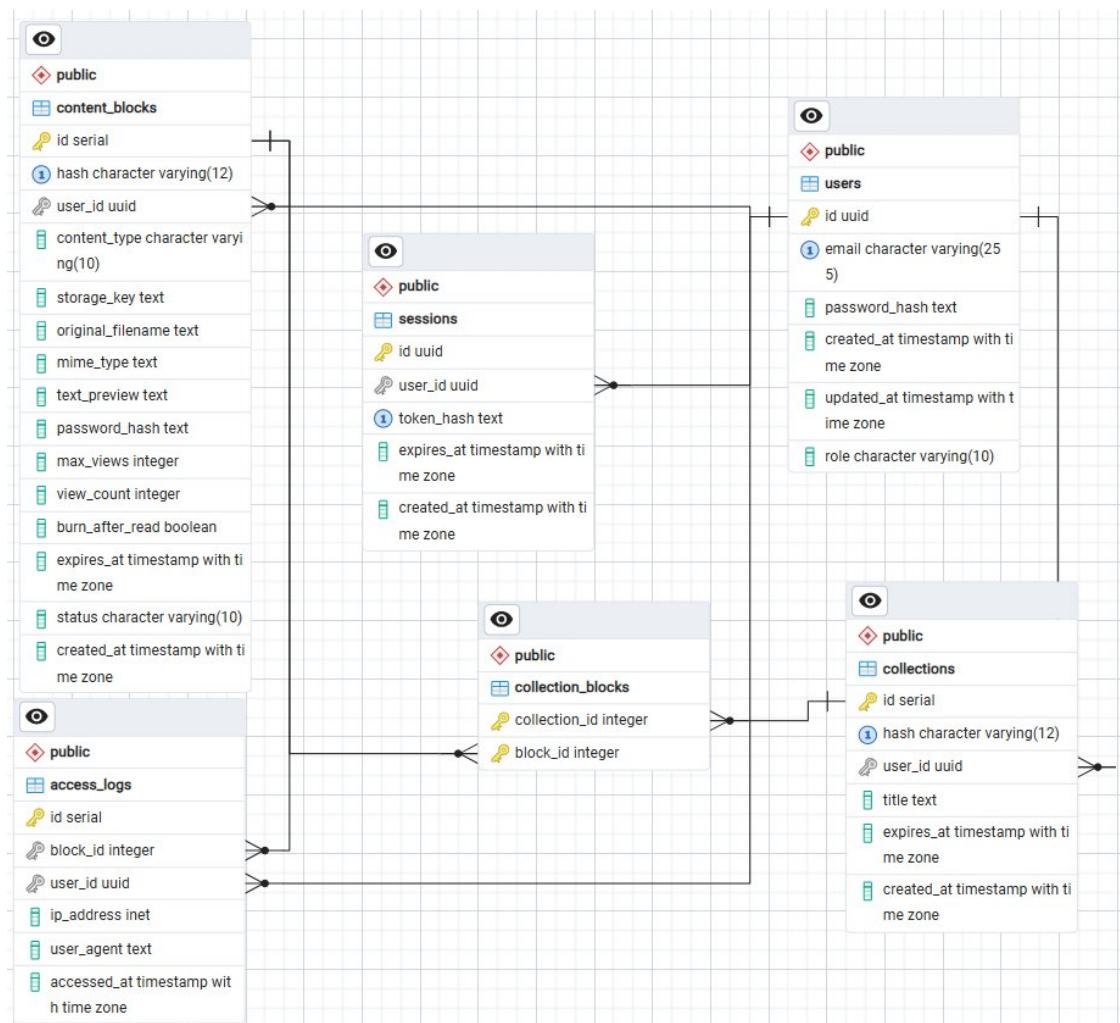


Рис. 2.3 Структура бази даних системи тимчасового обміну мультимедійними даними

Зв'язок «Користувач → Колекція» (users → collections) є відношенням один-до-багатьох: один користувач може створити кілька колекцій файлів, кожна з яких об'єднує довільну кількість записів під єдиним публічним посиланням. Зовнішній ключ user_id у таблиці collections посилається на id таблиці users та може приймати значення NULL для анонімних колекцій [5]. Наявність поля expires_at у таблиці collections дозволяє встановлювати окремий термін дії для всієї колекції незалежно від термінів дії окремих записів, що входять до неї.

Зв'язок «Колекція ↔ Блок контенту» (collections ↔ content_blocks) відповідає відношенню багато-до-багатьох і реалізується через проміжну таблицю collection_blocks: одна колекція може містити кілька записів контенту, і водночас один запис може входити до кількох різних колекцій. Таблиця collection_blocks має складений первинний ключ (collection_id, block_id), який унеможливорює повторне включення одного і того ж запису до однієї колекції, а обидва поля є одночасно зовнішніми ключами до відповідних таблиць. Каскадне видалення у таблиці collection_blocks гарантує автоматичне видалення зв'язків при видаленні колекції або блоку контенту.

Зв'язок «Блок контенту → Журнал доступу» (content_blocks → access_logs) є відношенням один-до-багатьох: кожен запис контенту може мати довільну кількість записів у журналі, що відповідають окремим фактам перегляду з фіксацією часу, IP-адреси та user-agent клієнта. Зовнішній ключ block_id у таблиці access_logs посиляється на id таблиці content_blocks, а додатковий зовнішній ключ user_id пов'язує запис журналу з обліковим записом авторизованого користувача, якщо перегляд здійснювався після входу до системи [5]. Це поле допускає NULL для фіксації анонімних переглядів, що дозволяє аналізувати загальну статистику перегляду незалежно від статусу автентифікації відвідувача.

Таблиця content_blocks є центральною в архітектурі бази даних і зберігає метадані всіх записів незалежно від типу контенту (текст або файл). Поле hash є індексованим та унікальним, що забезпечує пошук записів за публічним ідентифікатором із часовою складністю $O(\log n)$ завдяки B-tree індексу. Поле storage_key зберігає ключ об'єкта у Cloudflare R2, тоді як сам вміст зберігається виключно у хмарному сховищі, і такий розподіл відповідальностей між реляційною базою даних та об'єктним сховищем дозволяє ефективно масштабувати систему при зростанні обсягів даних без зміни схеми бази.

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		40

2.3. Проектування та реалізація алгоритмів роботи системи

Алгоритми роботи системи визначають точну послідовність операцій, що виконуються при обробці різних видів запитів, та є основою для реалізації програмного коду [24]. Кожен алгоритм спроектований з урахуванням перевірки коректності вхідних даних, обробки виняткових ситуацій та забезпечення цілісності даних навіть у разі часткових збоїв. У цьому підрозділі детально описано чотири ключові алгоритми: створення запису, отримання контенту, автентифікації користувача та автоматичного очищення прострочених записів.



Рис. 2.4 Алгоритм створення запису

Алгоритм створення запису ініціюється при надходженні POST-запиту від клієнта до відповідного ендпоінту серверного API. На першому кроці виконується валідація вхідних даних: перевіряється наявність та коректність обов'язкового поля типу контенту (text або file), а також опціональних параметрів доступу – часу життя, максимальної кількості переглядів та пароля. Якщо вхідні дані не відповідають встановленій схемі, система негайно повертає відповідь із кодом HTTP 400 та структурованим описом помилки без виконання подальших операцій.

На другому кроці викликається метод generateUniqueHash() класу HashGenerator для отримання унікального публічного ідентифікатора нового запису. Генерується випадковий рядок довжиною 8 символів за допомогою бібліотеки nanoid, після чого виконується запит до бази даних для перевірки відсутності запису з таким ідентифікатором. Якщо запис із таким хешем вже існує, генерація повторюється; після досягнення максимальної кількості спроб (5) повертається внутрішня помилка сервера з кодом HTTP 500 [18].

На третьому кроці вміст зберігається у Cloudflare R2 через відповідний метод класу StorageService – saveTextContent() для текстового вмісту або saveFileContent() для файлів. Ключ об'єкта у сховищі формується автоматично на основі типу контенту та унікального ідентифікатора для забезпечення передбачуваної структури зберігання. Після успішного завантаження у хмарне сховище метадані запису зберігаються у таблиці content_blocks бази даних PostgreSQL, і лише після цього клієнту повертається публічне посилання у форматі /b/{hash}, що підтверджує успішне завершення операції [5].

Алгоритм отримання контенту наведено у додатку А та виконується при надходженні GET-запиту за публічним посиланням і є найчастіше виконуваним алгоритмом у системі. На першому кроці з URL витягується хеш-ідентифікатор, за яким виконується пошук запису у базі даних через метод findByHash() класу ContentRepository з використанням індексованого поля. Якщо запис не знайдено,

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		42

система негайно повертає відповідь із кодом HTTP 404, уникаючи зайвих перевірок та звернень до зовнішніх сервісів.

На другому кроці виконується комплексна перевірка актуальності запису методом `ensureActive()`, що включає три послідовні умови: поточний статус запису не повинен мати значення `expired` або `deleted`; поточна дата не повинна перевищувати значення поля `expires_at`, за умови що це поле встановлено; поточне значення `view_count` не повинно дорівнювати або перевищувати `max_views`, якщо ліміт переглядів встановлено. Якщо будь-яка з умов порушена, відповідний статус запису оновлюється у базі даних і повертається відповідь з кодом HTTP 410 (Gone), що семантично точно відображає стан видаленого або простроченого ресурсу.

На третьому кроці, якщо встановлено захист паролем, виконується перевірка автентифікації запиту через порівняння введеного пароля з хешем у базі даних за допомогою функції `bcrypt.compare()`. У разі невірного пароля повертається відповідь із кодом HTTP 403, а кількість переглядів не збільшується, щоб запобігти вичерпанню ліміту неавторизованими спробами. Якщо всі перевірки пройдено успішно, вміст зчитується з Cloudflare R2, лічильник переглядів збільшується атомарним SQL-запитом, факт перегляду фіксується в таблиці `access_logs`, а якщо активовано режим `burn_after_read`, статус запису оновлюється до `deleted` після першої успішної видачі вмісту [5].

Алгоритм автентифікації виконується при POST-запиті на ендпоінт `/auth/login` і реалізує безпечний процес верифікації облікових даних з генерацією пари токенів, зображений у додатку Б. На першому кроці виконується синтаксична валідація вхідних даних: перевіряється наявність та формат поля `email` (відповідність стандарту RFC 5322) та поля `password` (мінімальна довжина 8 символів і відповідність вимогам складності). При будь-якій невідповідності повертається відповідь із кодом HTTP 400 з детальним описом помилки валідації.

На другому кроці виконується пошук користувача у базі даних за адресою електронної пошти через метод `findUserByEmail()` класу `AuthRepository` з

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		43

використанням індексованого поля email. Якщо користувача не знайдено, система повертає відповідь із кодом HTTP 401 з повідомленням «Невірні облікові дані», яке є навмисно неінформативним для запобігання розкриттю інформації про існування облікового запису при переборі адрес. Перевірка статусу облікового запису виконується також на цьому кроці: заблоковані або деактивовані акаунти отримують відповідь із кодом HTTP 403 [18].

На третьому кроці введений пароль перевіряється відносно збереженого хешу за допомогою функції `bcrypt.compare()`, яка виконує криптографічно безпечне порівняння з часовою складністю, що залежить від коефіцієнта вартості хешування. При успішній перевірці генерується пара токенів: короткоживучий `access token` у форматі JWT з терміном дії 15 хвилин та довгоживучий `refresh token` з терміном дії 30 днів, хеш якого зберігається у таблиці `sessions`. Клієнту повертається `access token` у тілі відповіді, а `refresh token` – у захищеній HTTP-only cookie, що унеможлиблює його зчитування через JavaScript та підвищує захист від XSS-атак.

Алгоритм автоматичного очищення зображено у додатку В та запускається планувальником задач за розкладом (cron job) з фіксованим інтервалом, наприклад щогодини, та виконується у фоновому режимі незалежно від обробки користувацьких запитів. На першому кроці метод `findExpired()` класу `ContentRepository` виконує SELECT-запит до бази даних і повертає список усіх записів, у яких поле `expires_at` менше поточного часу і статус рівний `active`. Список обробляється в пакетному режимі фіксованого розміру для рівномірного розподілу навантаження на базу даних і сховище, уникаючи одноразової обробки великого обсягу записів [5].

На другому кроці для кожного простроченого запису послідовно виконуються дві операції: спочатку видалення відповідного об'єкта з Cloudflare R2 через метод `deleteContent()` класу `StorageService`, а потім оновлення статусу запису до значення `expired` у таблиці `content_blocks` методом `markExpired()`. Саме такий порядок операцій – спочатку видалення даних зі сховища, потім оновлення статусу в базі

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		44

– гарантує, що навіть у разі збою між ними дані у сховищі будуть видалені, а не залишаться недосяжними «сиротами» без відповідних записів у базі. Помилки при видаленні конкретного об'єкта обробляються індивідуально і не зупиняють обробку решти записів у пакеті.

На третьому кроці виконується аналогічна обробка записів з активованим режимом `burn_after_read`, статус яких вже змінено на `deleted` у ході попередніх переглядів, але відповідні об'єкти у Cloudflare R2 ще не були видалені через можливі попередні збої. Після завершення пакетної обробки результати операції фіксуються у системному журналі із зазначенням кількості оброблених записів, кількості помилок та загального часу виконання, що дозволяє відстежувати коректність роботи алгоритму з часом. Регулярне виконання цього алгоритму забезпечує узгодженість між станом бази даних та об'єктного сховища і запобігає накопиченню застарілих даних.

2.4. Реалізація програмного комплексу

У цьому підрозділі описано реалізацію основних методів ключових класів системи з наведенням характерних фрагментів програмного коду. Представлені приклади ілюструють підходи до вирішення центральних технічних задач: генерації унікальних ідентифікаторів, збереження контенту різних типів, автентифікації користувачів та обробки завантажуваних файлів. Повні реалізації відповідних модулів, включаючи обробку помилок та допоміжні методи, наведено у відповідних додатках до роботи.

Для формування публічних ідентифікаторів записів реалізовано клас `HashGenerator`, центральним методом якого є `generateUniqueHash()` [12]. Метод виконує генерацію рядка та перевіряє його відсутність у базі даних перед поверненням результату, що гарантує унікальність кожного виданого ідентифікатора. Нижче наведено фрагмент реалізації цього методу:

```
export async function generateUniqueHash(): Promise<string> {
  for (let attempt = 0; attempt < MAX_ATTEMPTS; attempt++) {
    const hash = nanoid(HASH_LENGTH);
    const existing = await contentRepository.findByHash(hash);
```

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		45

```

        if (!existing) {
            return hash;
        }
    }
    throw new Error('Failed to generate unique hash');
}

```

У наведеному фрагменті для створення хешу використовується бібліотека `nanoid`, яка забезпечує криптографічно стійку генерацію випадкових рядків із алфавіту URL-безпечних символів фіксованої довжини. Після створення кандидата виконується перевірка його наявності у базі даних через `findByHash()`; якщо колізія відсутня, хеш повертається як результат, а якщо виявлено дублікат – цикл повторюється до досягнення `MAX_ATTEMPTS` спроб. Ймовірність вичерпання спроб при довжині хешу 8 символів є надзвичайно малою на практиці і зростає лише при заповненні понад 99% можливого простору ідентифікаторів. Повна реалізація модуля `HashGenerator` наведена у додатку А.

Метод `createTextBlock()` класу `ContentService` координує весь процес збереження текстового контенту – від генерації ідентифікатора до запису метаданих у базу даних. Метод приймає об'єкт DTO з вмістом та параметрами доступу і виконує операції у суворо визначеному порядку для забезпечення узгодженості даних. Нижче наведено ключовий фрагмент реалізації:

```

const hash = await hashGenerator.generateUniqueHash();
const storageKey = await storageService.saveTextContent(dto.text);
await contentRepository.create({
    hash,
    storageKey,
    contentType: 'text',
    expiresAt: dto.expiresAt,
    maxViews: dto.maxViews,
    burnAfterRead: dto.burnAfterRead,
    passwordHash: dto.password
        ? await bcrypt.hash(dto.password, SALT_ROUNDS)
        : null,
    userId: dto.userId ?? null,
});

```

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		46

У наведеному фрагменті спочатку генерується унікальний хеш для нового запису, потім текстовий вміст зберігається у Cloudflare R2 через метод `saveTextContent()`, який повертає ключ об'єкта у сховищі [6]. Після успішного збереження вмісту виконується запит до бази даних для створення запису в таблиці `content_blocks` із усіма параметрами доступу; якщо передано пароль, він одразу хешується за допомогою `bcrypt` і у базі зберігається виключно хеш, а не оригінальний рядок. Повний код класу `ContentService`, включаючи метод `createFileBlock()` та логіку обробки помилок, наведено у додатку Г.

У класі `AuthService` перевірка відповідності пароля реалізована через послідовний пошук користувача у базі даних та порівняння введеного значення з хешем. Обидва можливих негативних результати – відсутність користувача та невірний пароль – повертають однакове повідомлення про помилку, що є свідомим захисним рішенням. Нижче наведено ключовий фрагмент реалізації методу `login()`:

```
const user = await authRepository.findUserByEmail(email);
if (!user) {
    throw new UnauthorizedException('Невірні облікові дані');
}

const validPassword = await bcrypt.compare(
    password,
    user.passwordHash
);

if (!validPassword) {
    throw new UnauthorizedException('Невірні облікові дані');
}
```

Використання однакового повідомлення «Невірні облікові дані» незалежно від причини відмови є свідомим проектним рішенням, спрямованим на запобігання атакам перерахування облікових записів (account enumeration attack), за яких злоумисник намагається з'ясувати, які електронні адреси зареєстровані в системі. Функція `bcrypt.compare()` виконує криптографічно безпечне порівняння

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		47

з часом виконання, що залежить від коефіцієнта вартості, а не від вмісту рядків, що унеможливорює атаки за часовим аналізом. Повний код класу AuthService, включаючи методи register(), refresh() та logout(), наведено у додатку Д.

Обробка завантажуваних файлів виконується у Fastify-обробнику multipart-запитів, який використовує потоковий підхід для ефективної обробки великих файлів без зайвого буферування. Після отримання частини запиту тіло файлу зчитується у буфер і передається до StorageService разом з метаданими. Нижче наведено ключовий фрагмент обробника завантаження:

```
const parts = request.parts();
for await (const part of parts) {
  if (part.type === 'file') {
    const fileBuffer = await part.toBuffer();
    const filename = part.filename;
    const mimeType = part.mimetype;

    await storageService.saveFileContent(
      fileBuffer,
      filename,
      mimeType
    );
  }
}
```

Метод toBuffer() зчитує потік даних файлу у пам'ять у вигляді Buffer-об'єкта, після чого отриманий буфер разом із оригінальним ім'ям файлу та типом MIME передається до StorageService для завантаження у Cloudflare R2. Використання асинхронного ітератора parts дозволяє послідовно обробляти всі частини multipart-запиту, що необхідно для реалізації функції створення колекцій із кількох файлів в одному HTTP-запиті [18]. Перевірка типу MIME та розміру файлу виконується до виклику toBuffer() для раннього відхилення некоректних запитів без зайвого споживання пам'яті. Повний код модуля завантаження файлів наведено у додатку Б.

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		48

2.5. Опис компонентної будови програмного додатку

Програмний комплекс побудований за принципами клієнт-серверної архітектури з чітким розподілом відповідальностей між незалежними компонентами, кожен з яких може розгортатися та масштабуватися окремо. Система складається з п'яти основних структурних елементів, що взаємодіють між собою через формалізовані інтерфейси: REST API, SQL, S3-сумісний API та SMTP. Діаграма компонентів системи наведена на рисунку 2.8 і відображає топологію взаємодій між усіма елементами архітектури.

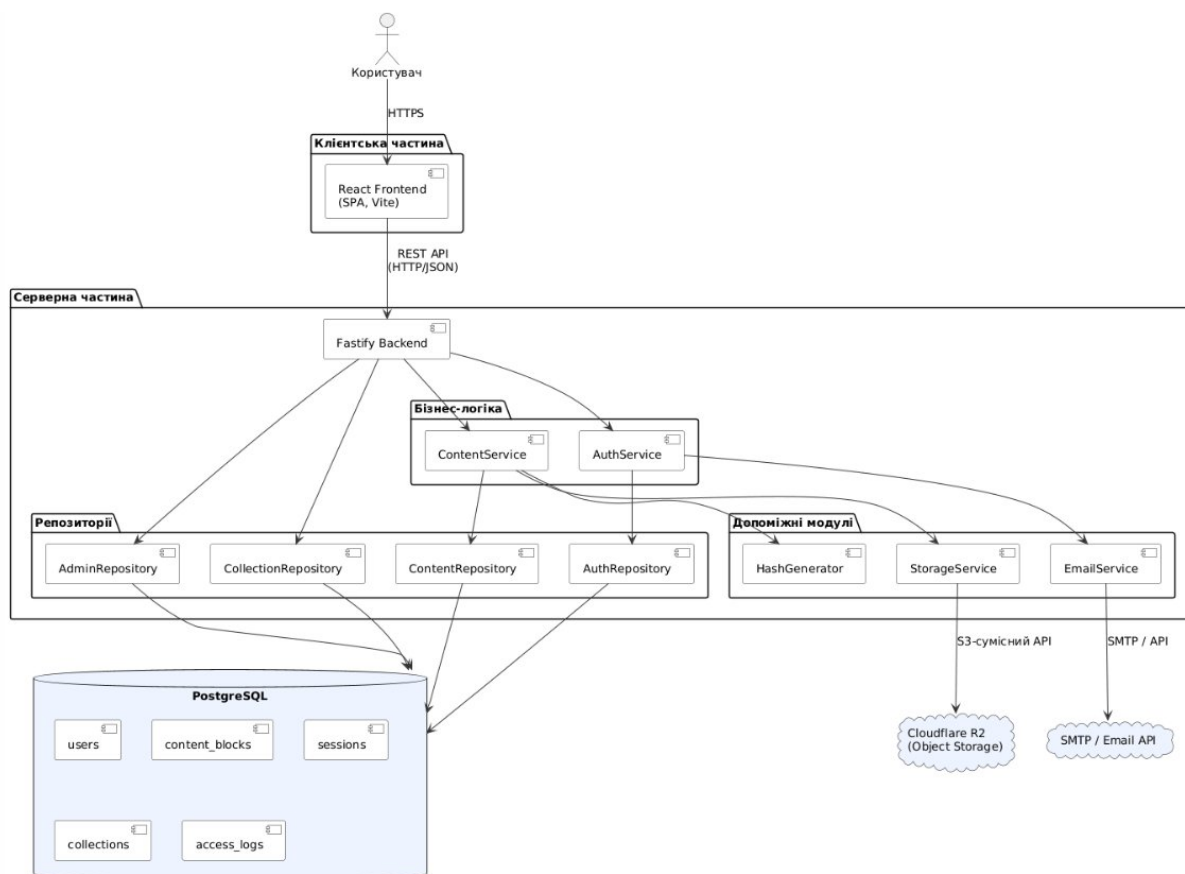


Рис. 2.8 Діаграма компонентів системи тимчасового обміну мультимедійними даними

Компонент React Frontend є клієнтською частиною системи, реалізованою у вигляді односторінкового застосунку (SPA) на базі бібліотеки React. Він відповідає за формування інтерфейсу користувача, обробку введення даних, відображення контенту та взаємодію з серверним API через стандартні HTTP-запити. Frontend-компонент комунікує виключно через REST API і не має прямого

доступу до бази даних чи сховища файлів, що забезпечує безпеку та чітке розмежування відповідальностей; збірка та оптимізація клієнтської частини виконуються засобами Vite [10].

Компонент Fastify Backend є серверною частиною системи та реалізує весь прикладний рівень: маршрутизацію запитів, бізнес-логіку, автентифікацію та авторизацію, взаємодію з базою даних і хмарним сховищем. Backend побудований на фреймворку Fastify, що забезпечує високу продуктивність завдяки оптимізованій системі маршрутизації та підтримці асинхронних операцій з мінімальними накладними витратами. Серверна частина є єдиною точкою оркестрації всіх інших компонентів і взаємодіє безпосередньо з PostgreSQL, Cloudflare R2 та Email Service.

Компонент PostgreSQL є системою управління реляційною базою даних і відповідає за надійне зберігання всіх структурованих метаданих системи: інформації про користувачів, записи контенту, сесії та журнали доступу. СУБД PostgreSQL обрана завдяки підтримці транзакцій ACID, засобам забезпечення референційної цілісності через зовнішні ключі та гнучким можливостям індексування для прискорення типових запитів [5]. Серверна частина взаємодіє з PostgreSQL через пул з'єднань, що дозволяє ефективно використовувати наявні з'єднання при обробці паралельних запитів.

Компонент Cloudflare R2 є хмарним об'єктним сховищем і відповідає за зберігання безпосереднього вмісту записів: текстових файлів та медіафайлів усіх підтримуваних форматів. Взаємодія здійснюється через AWS S3-сумісний API, що дозволяє використовувати стандартний AWS SDK без прив'язки до конкретного провайдера і спрощує потенційну міграцію на інше сховище у майбутньому. Cloudflare R2 не стягує плату за трафік вихідних даних (egress), що робить його економічно ефективним рішенням для зберігання та роздачі великих обсягів мультимедійного контенту.

Повний шлях запиту при отриманні контенту за посиланням є показовим прикладом взаємодії всіх компонентів системи та демонструє, як відповідальність

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		50

розподілена між рівнями архітектури. На першому кроці користувач відкриває публічне посилання у браузері, React Frontend відображає сторінку перегляду та надсилає GET-запит до REST API серверної частини з хешем запису в URL [79]. Запит проходить через мережу до серверного компонента, де починається основна обробка.

Fastify Backend отримує запит та передає його до відповідного обробника маршруту, який викликає метод `getBlockContent()` класу `ContentService`. `ContentService` звертається до `ContentRepository`, який виконує параметризований SELECT-запит до PostgreSQL для пошуку запису за хешем з використанням індексу [5]. Після отримання запису з бази даних виконується перевірка його актуальності за трьома критеріями: поточний статус, термін дії та кількість переглядів, і при невиконанні будь-якого з них запит завершується відповідною HTTP-відповіддю без звернення до сховища.

Після успішної перевірки `ContentService` викликає `StorageService`, який надсилає запит до Cloudflare R2 для читання вмісту за збереженим ключем об'єкта. Отримані дані повертаються через `ContentService` назад до обробника маршруту, який формує HTTP-відповідь і надсилає її клієнту, а паралельно `ContentRepository` виконує UPDATE-запит для збільшення лічильника переглядів та INSERT-запит для запису до таблиці `access_logs` [18]. Такий розподіл відповідальностей між компонентами забезпечує масштабованість системи: при зростанні навантаження кожен компонент може бути масштабований незалежно, оскільки всі компоненти взаємодіють виключно через стандартні інтерфейси без прямих залежностей від конкретних реалізацій.

Висновки до другого розділу

У другому розділі здійснено комплексне проектування та реалізацію вебсистеми тимчасового обміну мультимедійними даними. На підставі аналізу предметної області сформовано повний перелік функціональних і нефункціональних вимог до програмного забезпечення, що охоплюють створення

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		51

та публікацію контенту, управління доступом, автентифікацію користувачів та адміністрування системи. Нефункціональні вимоги визначають характеристики продуктивності, надійності та безпеки, дотримання яких є необхідною умовою для комерційної придатності сервісу.

Визначено три категорії користувачів системи – анонімний користувач, зареєстрований користувач та адміністратор, – кожна з яких має власний набір прав доступу відповідно до принципу мінімально необхідних привілеїв. Побудовано діаграму варіантів використання, що включає сім основних сценаріїв взаємодії з системою. Об'єктно-орієнтоване проектування виконано за шаблоном Repository і охоплює вісім основних класів із чітким розподілом відповідальностей, що забезпечує низьку зв'язаність та високу зв'язність коду.

Структуру бази даних PostgreSQL спроектовано відповідно до вимог третьої нормальної форми, що унеможливорює надлишкове зберігання даних та аномалії при операціях оновлення. База даних включає шість таблиць з чітко визначеними зв'язками через зовнішні ключі, що забезпечує референційну цілісність. Детально описано п'ять типів зв'язків між таблицями: відношення один-до-багатьох між users та content_blocks, users та sessions, users та collections; відношення багато-до-багатьох між collections та content_blocks через проміжну таблицю; відношення один-до-багатьох між content_blocks та access_logs.

Розроблено та детально описано чотири ключові алгоритми роботи системи: алгоритм створення запису з контролем унікальності ідентифікаторів, алгоритм отримання контенту з комплексною перевіркою умов доступу, алгоритм автентифікації користувача з генерацією пари JWT/refresh-токенів та алгоритм автоматичного очищення прострочених записів у пакетному режимі. Наведено фрагменти реалізації основних методів класів HashGenerator, ContentService, AuthService та модуля завантаження файлів. Описано п'ятикомпонентну архітектуру програмного комплексу та детальний механізм взаємодії між компонентами при обробці запитів на отримання контенту.

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		52

РОЗДІЛ 3. ІНТЕРФЕЙС, НАЛАШТУВАННЯ ТА ТЕСТУВАННЯ ВЕБСИСТЕМИ ТИМЧАСОВОГО ОБМІНУ МУЛЬТИМЕДІЙНИМИ ДАНИМИ

3.1 Порядок встановлення та налаштування системи

Для забезпечення коректної роботи вебсервісу тимчасового доступу до мультимедійного контенту необхідно виконати налаштування програмного середовища, бази даних та зовнішніх сервісів зберігання інформації. Процес розгортання системи передбачає встановлення серверної та клієнтської частин застосунку, налаштування взаємодії з базою даних PostgreSQL та підключення хмарного файлового сховища Cloudflare R2 [5,6]. Виконання всіх підготовчих етапів є необхідною умовою для стабільного функціонування програмного комплексу та забезпечення доступності його можливостей для кінцевих користувачів.

Серверна частина програмного комплексу реалізована на платформі Node.js із використанням фреймворку Fastify, що забезпечує обробку HTTP-запитів та реалізацію бізнес-логіки. Для зберігання службових даних, параметрів доступу та метаданих контенту використовується система керування базами даних PostgreSQL. Зберігання мультимедійних файлів здійснюється у хмарному сховищі Cloudflare R2, що дозволяє зменшити навантаження на базу даних та забезпечити ефективний доступ до файлів.

Для розробки та тестування системи достатньо персонального комп'ютера із двоядерним процесором, оперативною пам'яттю від 4 ГБ та не менше 5 ГБ вільного місця на накопичувачі. Для розгортання системи в умовах реального навантаження рекомендується використовувати сервер із оперативною пам'яттю від 8 ГБ та швидким SSD-накопичувачем. Наявність достатніх апаратних ресурсів дозволяє забезпечити стабільну роботу системи навіть за значної кількості одночасних звернень користувачів.

На рисунку 3.1 наведено діаграму розгортання програмного комплексу, що відображає основні компоненти системи та взаємозв'язки між ними [24]. Вона

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		53

дозволяє наочно продемонструвати архітектуру розгортання застосунку та принципи взаємодії між його складовими. Аналіз наведеної діаграми дає можливість зрозуміти логіку роботи системи та особливості організації її програмної інфраструктури.

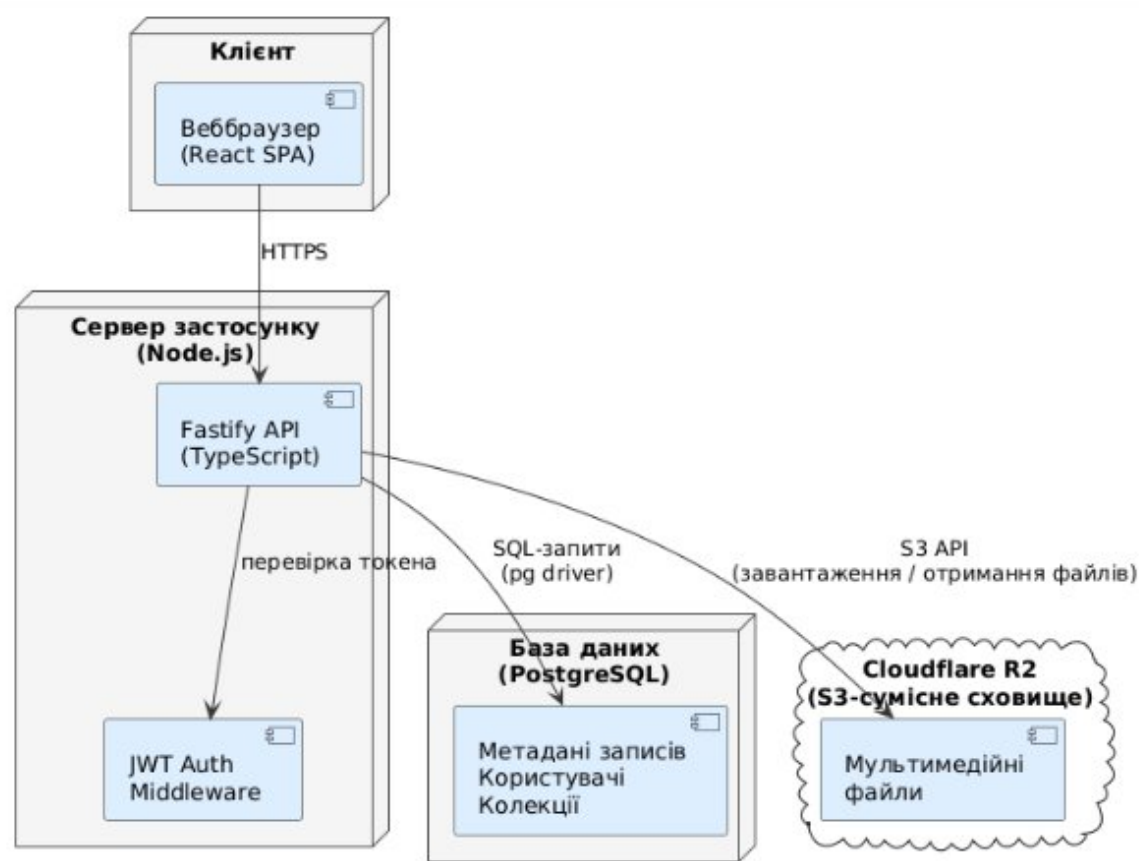


Рис. 3.1. Діаграма розгортання системи

Процес встановлення серверної частини розпочинається зі встановлення залежностей та налаштування змінних середовища, необхідних для підключення до бази даних і зовнішніх сервісів. Після ініціалізації проєкту необхідно створити файл конфігурації з параметрами доступу до PostgreSQL, Cloudflare R2, JWT та SMTP-сервера [5,8]. Коректно заповнений конфігураційний файл є обов'язковою умовою для успішного запуску серверної частини програмного комплексу.

Наступним етапом є виконання міграцій бази даних, що створюють необхідну структуру таблиць для зберігання інформації про записи, користувачів та колекції. Після успішного завершення міграцій серверна частина запускається у режимі розробки або production-режимі залежно від призначення середовища.

Запуск серверної частини підтверджується виведенням відповідного повідомлення до консолі та готовністю до прийому HTTP-запитів.

Як показано на рисунку 3.1, користувач взаємодіє із системою через веббраузер, а сформовані запити надходять до серверної частини для обробки та перевірки прав доступу. У процесі роботи сервер взаємодіє з базою даних PostgreSQL для отримання або збереження службової інформації та зі сховищем Cloudflare R2 для роботи з мультимедійними файлами [6]. Результат виконання запиту повертається користувачеві у вигляді відповідних даних або повідомлень системи.

Для функціонування системи необхідно встановити та налаштувати сервер PostgreSQL, створити відповідну базу даних та користувача з необхідними правами доступу. Параметри підключення до бази даних вказуються у змінній середовища DATABASE_URL у форматі рядка підключення, що включає хост, порт, ім'я бази та облікові дані. Правильне налаштування підключення є необхідною умовою для коректного виконання міграцій та подальшої роботи системи.

Структура бази даних формується автоматично під час виконання міграцій, що входять до складу серверної частини програмного комплексу. Міграції створюють таблиці для зберігання записів (pastes), користувачів (users), колекцій (collections) та пов'язаних сутностей зі всіма необхідними зовнішніми ключами та індексами. Використання механізму міграцій спрощує процес розгортання системи та забезпечує узгодженість структури бази даних між різними середовищами.

Після виконання міграцій рекомендується перевірити коректність створених таблиць шляхом підключення до бази даних та перегляду їх структури. Для підвищення надійності системи доцільно налаштувати регулярне резервне копіювання бази даних, що дозволить відновити дані у випадку збою. Налаштоване та перевірене підключення до бази даних забезпечує готовність серверної частини до обробки запитів користувачів.

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		55

Для зберігання мультимедійних файлів у системі використовується хмарне сховище Cloudflare R2, що сумісне з протоколом Amazon S3 та забезпечує ефективне збереження і завантаження файлів [7]. Налаштування сховища передбачає створення бакету у панелі керування Cloudflare та генерацію облікових даних API з відповідними правами доступу. Отримані ключі доступу вказуються у змінних середовища серверної частини та використовуються для автентифікації під час операцій із файлами.

Після завантаження файл автоматично передається до сховища Cloudflare R2, а в базі даних PostgreSQL зберігається лише посилання на відповідний об'єкт. Такий підхід дозволяє уникнути перевантаження бази даних великими обсягами бінарних даних та спрощує процес керування контентом. Завдяки цьому забезпечується швидке отримання файлів та ефективне використання обчислювальних ресурсів серверної частини.

Для перевірки коректності налаштування сховища рекомендується виконати тестове завантаження файлу та переконатися у його появі в бакеті через панель керування Cloudflare [6]. Доступ до файлів здійснюється через пресайнд-URL, що генерується серверною частиною за запитом клієнта і має обмежений термін дії. Правильно налаштоване хмарне сховище є обов'язковою умовою для коректної роботи функціоналу завантаження та перегляду мультимедійних файлів.

Клієнтська частина системи реалізована як односторінковий вебзастосунок на основі бібліотеки React та мови програмування TypeScript з використанням інструменту збірки Vite. Для запуску клієнтської частини необхідно встановити залежності проєкту та вказати адресу серверної частини у відповідній змінній середовища. Після виконання зазначених кроків вебзастосунок стає доступним через браузер без необхідності встановлення додаткового програмного забезпечення на стороні користувача.

Використання односторінкової архітектури дозволяє виконувати навігацію між сторінками без їх повного перезавантаження, що забезпечує швидку реакцію

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		56

інтерфейсу та покращує користувацький досвід. Маршрутизація між сторінками реалізована за допомогою бібліотеки React Router, що дозволяє підтримувати коректні URL-адреси для кожного розділу застосунку [24]. Такий підхід позитивно впливає на загальну швидкодію застосунку та комфортність його використання.

3.2 Структура інтерфейсу та порядок роботи із системою

Інтерфейс вебсервісу тимчасового доступу до мультимедійного контенту розроблено з урахуванням принципів зручності використання, доступності функціоналу та мінімізації кількості дій користувача. Клієнтська частина реалізована як односторінковий вебзастосунок, що дозволяє виконувати навігацію між сторінками без їх повного перезавантаження. Усі сторінки застосунку мають єдину стилістику оформлення та побудовані відповідно до сучасних вимог веброзробки.

Загальна структура інтерфейсу включає головну сторінку створення контенту, сторінки авторизації та реєстрації, сторінку перегляду записів, особистий кабінет користувача та адміністративну панель. Кожен із зазначених розділів відповідає за реалізацію окремої групи функціональних можливостей системи. Навігація між сторінками здійснюється за допомогою вбудованого механізму маршрутизації React Router, що дозволяє підтримувати коректні адреси для кожного розділу [9].

Головна сторінка системи є основним інструментом створення контенту та взаємодії із серверною частиною застосунку. Інтерфейс головної сторінки побудований у вигляді набору функціональних вкладок, що дозволяє об'єднати пов'язаний функціонал в одному місці та зменшити кількість окремих сторінок. Користувач отримує можливість швидко перемикатися між режимами роботи без необхідності переходу до інших розділів застосунку.

На рисунку 3.4 наведено головну сторінку вебсервісу з вкладкою створення текстового запису. Форма містить поле введення текстового контенту та набір параметрів, що визначають умови доступу: час існування запису (TTL),

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		57

максимальна кількість переглядів, пароль та режим одноразового перегляду. Після заповнення форми виконується перевірка коректності введених даних та передача інформації на сервер.

Рис. 3.4. Головна сторінка системи

На рисунку 3.5 наведено форму створення текстового запису із заповненими параметрами доступу. У випадку успішного виконання операції система відображає сформоване унікальне посилання та надає можливість його копіювання. Такий механізм дозволяє ефективно реалізувати концепцію тимчасового доступу до інформації та забезпечити швидке поширення створеного контенту.

Рис. 3.5. Форма створення текстового запису

Для роботи з мультимедійними файлами використовується окрема вкладка головної сторінки з формою завантаження, що містить зону drag-and-drop для зручного вибору файлів. Користувач може обрати файл за допомогою стандартного діалогу операційної системи або скористатися механізмом

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		58

перетягування файлів у відповідну область інтерфейсу. Після вибору файлу зона змінює зовнішній вигляд та відображає назву і розмір обраного файлу.

На рисунку 3.6 наведено сторінку завантаження файлів із заповненою формою. Після завантаження файл автоматично передається до хмарного сховища Cloudflare R2 та стає доступним для подальшого використання через унікальне посилання [6]. При цьому користувачеві надаються ті самі параметри налаштування доступу, що й для текстових записів, що забезпечує єдиний підхід до роботи з різними типами контенту.

Додатково система відображає основні характеристики обраного файлу, що дозволяє користувачеві переконатися у правильності вибору перед завантаженням. Параметри доступу до файлового запису аналогічні до параметрів текстового: TTL, ліміт переглядів, парольний захист та режим burn-after-read. Така реалізація підвищує зручність використання системи та спрощує процес створення файлових записів.

Рис. 3.6. Сторінка завантаження файлів

Після переходу за сформованим посиланням відкривається сторінка перегляду контенту, що реалізує стан-машину з кількома можливими станами відображення залежно від типу запису та прав доступу. Під час відкриття сторінки система автоматично перевіряє всі встановлені обмеження доступу та фіксує факт перегляду контенту для оновлення лічильника. Таким чином забезпечується надійний контроль над використанням інформації.

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		59

На рисунку 3.7 наведено сторінку перегляду текстового запису, що відображає вміст у полі лише для читання. На рисунку 3.8 показано сторінку перегляду файлового запису з кнопкою завантаження файлу, реалізованою через тег з атрибутом download та URL, сформованим з Blob-об'єкта. Сторінка перегляду також відображає метаінформацію: кількість переглядів, дату закінчення TTL та мітку burn-after-read за її наявності.

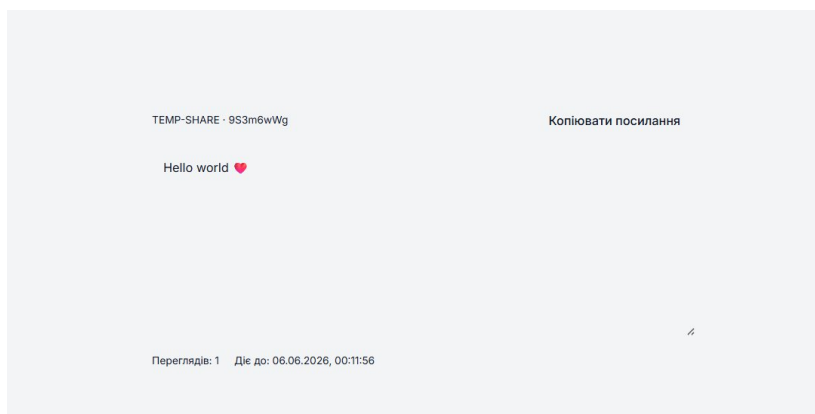


Рис. 3.7. Перегляд текстового запису



Рис. 3.8. Перегляд файлового запису

На рисунку 3.9 наведено стан захищеного запису, що відображає форму введення пароля. На рисунку 3.10 показано альтернативні стани сторінки: повідомлення про недоступність запису (HTTP 410) з поясненням про вичерпаний термін або ліміт переглядів, а також повідомлення про відсутність запису (HTTP 404) [18]. Наявність чітких інформаційних повідомлень про статус запису покращує взаємодію користувача із системою.

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		60

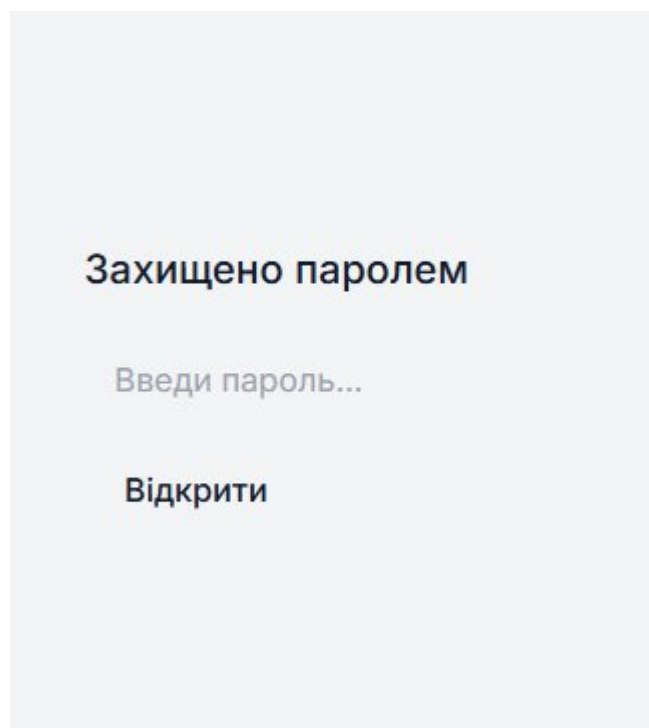


Рис. 3.9. Перегляд захищеного запису

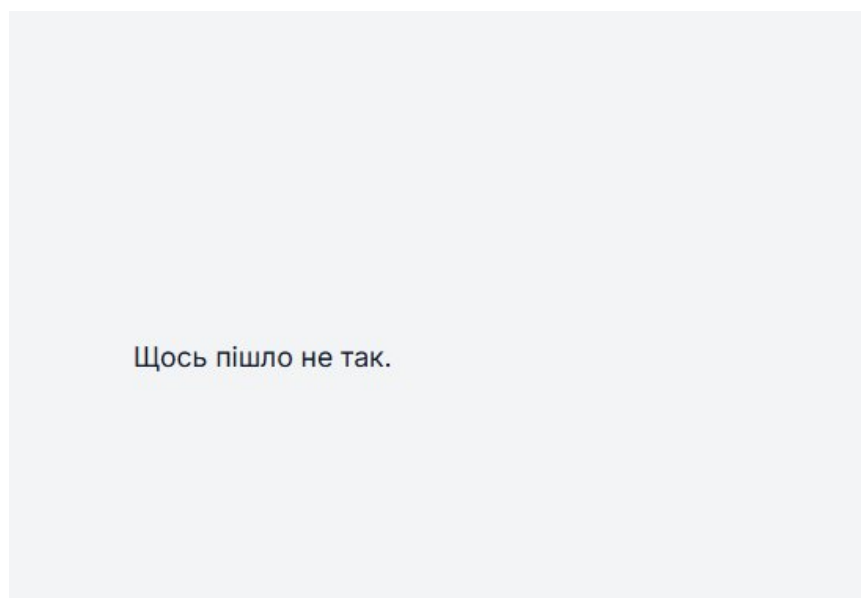


Рис. 3.10. Альтернативні стани сторінки перегляду

Сторінка входу (/login) відображається на градієнтному фоні із центрованою карткою авторизації, що містить логотип системи, поля введення електронної пошти та пароля, блок повідомлень про помилки та кнопку входу з індикатором завантаження. Сторінка також містить посилання для переходу до реєстрації для користувачів, що ще не мають облікового запису. На рисунку 3.11 наведено сторінку авторизації вебсервісу.

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		61

Рис. 3.11. Сторінка авторизації

Сторінка реєстрації (/register) виконана в аналогічному стилі оформлення та дозволяє створити новий обліковий запис шляхом введення електронної пошти та пароля. При успішній реєстрації користувач автоматично авторизується та перенаправляється до особистого кабінету. На рисунку 3.12 наведено сторінку реєстрації вебсервісу.

Рис. 3.12. Сторінка реєстрації

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		62

Обидві сторінки реалізовані у єдиному стилі оформлення та забезпечують зрозумілу взаємодію із користувачем, включаючи обробку типових помилок, таких як неправильні облікові дані або вже зайнята електронна пошта. Механізм автентифікації ґрунтується на JWT access-токенах та refresh-токенах, що зберігаються у httpOnly cookies [8]. Така реалізація забезпечує захист від атак типу XSS та гарантує безпечну передачу облікових даних між клієнтом і сервером.

Сторінка особистого кабінету (/dashboard) доступна лише авторизованим користувачам та захищена компонентом ProtectedRoute, що перевіряє наявність автентифікованого сеансу через контекст AuthContext. У верхній частині сторінки розміщено три картки статистики: загальна кількість записів, кількість активних та прострочених записів. Значення відображаються великими числами у кольорах акцентів, що забезпечує швидке сприйняття інформації.

Основну частину кабінету займає таблиця власних записів із колонками: хеш (посилання на сторінку перегляду), тип, кількість переглядів, статус, дата закінчення TTL та кнопка видалення. При порожньому списку відображається стан із посиланням для створення першого запису, що спрощує початок роботи нових користувачів. На рисунку 3.13 наведено особистий кабінет користувача з таблицею записів та статистикою.

У правому верхньому куті заголовка відображається електронна пошта поточного користувача, а також доступні кнопки переходу на головну сторінку та виходу з системи. Особистий кабінет є центральним інструментом роботи зареєстрованих користувачів, що надає повний огляд створеного контенту. Зручна структура кабінету дозволяє ефективно керувати записами та відстежувати статистику їх використання.

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		63

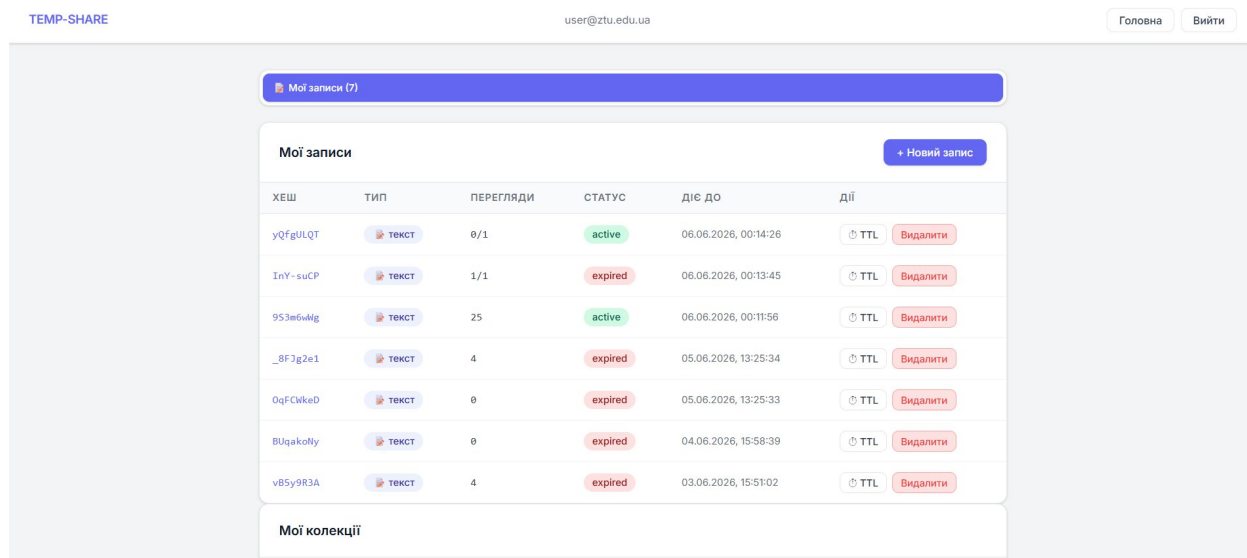


Рис. 3.13. Особистий кабінет користувача

Сторінка адміністративної панелі (/admin) доступна лише користувачам із роллю admin та містить три функціональні вкладки: статистика, користувачі та всі записи. Вкладка «Статистика» відображає три картки з кількістю зареєстрованих користувачів, загальною кількістю записів у системі та кількістю активних записів. Дані завантажуються через відповідний API-ендпоінт та оновлюються при кожному відкритті панелі [25,26].

Вкладка «Користувачі» відображає таблицю всіх зареєстрованих користувачів з інформацією про електронну пошту, роль, дату реєстрації та кнопку зміни ролі. Натискання кнопки «Зробити admin» або «Зробити user» надсилає запит до сервера та оновлює таблицю без перезавантаження сторінки. Вкладка «Всі записи» аналогічна таблиці кабінету, але відображає записи всіх користувачів із додатковою колонкою «Власник».

На рисунку 3.14 наведено адміністративну панель системи з вкладкою перегляду всіх записів. Адміністратор може видалити будь-який запис незалежно від його власника, що забезпечує можливість модерації контенту [25,26]. Наявність адміністративного інструменту спрощує супровід програмного комплексу та підвищує рівень його керованості.

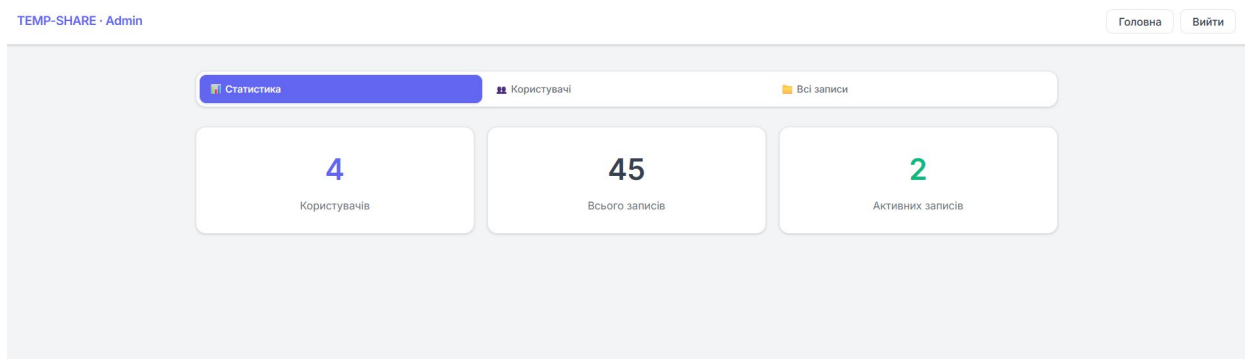


Рис. 3.14. Панель адміністратора

Сторінка 404 відображається у випадку переходу за неіснуючою або недоступною адресою вебзастосунку та інформує користувача про відсутність запитуваного ресурсу. Вона містить повідомлення про помилку «404 – Сторінку не знайдено», короткий опис проблеми та кнопку повернення на головну сторінку. На рисунку 3.15 наведено вигляд сторінки 404 [9].

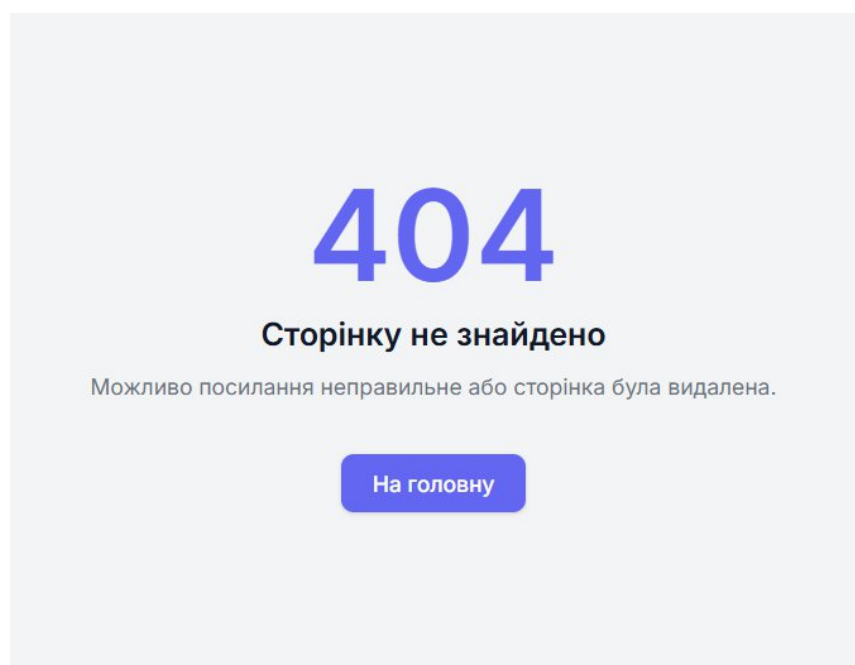


Рис. 3.15. Сторінка 404

Реалізація окремої сторінки для обробки помилок навігації є елементом якісного користувацького досвіду та допомагає уникнути ситуацій, коли користувач опиняється на порожній або непрацюючій сторінці. Кнопка повернення на головну сторінку дозволяє швидко відновити роботу із системою без використання навігаційних елементів браузера. Уніфікований стиль сторінки

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		65

помилки відповідає загальній стилістиці вебзастосунку та зберігає цілісність інтерфейсу.

Обробка маршрутів, що не відповідають жодному визначеному шляху, реалізована на рівні конфігурації React Router через відповідний маршрут-заглушку. Це гарантує, що будь-яка некоректна адреса буде перехоплена та оброблена належним чином. Таке рішення підвищує надійність навігаційної логіки застосунку та запобігає відображенню нечитабельних системних помилок.

3.3 Тестування роботи програмного комплексу

Після завершення реалізації програмного комплексу було проведено тестування його функціональних можливостей з метою перевірки відповідності системи вимогам, визначеним на етапі проєктування [22]. Особлива увага приділялася перевірці алгоритмів тимчасового доступу до контенту, контролю кількості переглядів, автентифікації користувачів та роботі з мультимедійними файлами. Результати тестування дозволяють оцінити готовність програмного комплексу до практичного використання.

Для виконання тестування використовувався програмний засіб Postman, що дозволяє надсилати HTTP-запити до серверної частини системи та аналізувати отримані відповіді незалежно від клієнтського інтерфейсу. Окремі сценарії також були перевірені безпосередньо через вебінтерфейс для підтвердження коректності взаємодії між клієнтською та серверною частинами. Комплексний підхід до тестування дозволив перевірити всі ключові функціональні можливості програмного продукту.

Тестування охоплювало дев'ять функціональних тест-кейсів, що відповідають основним варіантам використання системи, описаним у другому розділі роботи. Кожен тест-кейс включав визначення вхідних даних, очікуваного результату та зафіксованого фактичного результату виконання запиту. Зведені результати тестування наведено в таблиці 3.2 у підрозділі 3.3.9.

Однією з основних функцій системи є створення тимчасових текстових записів, тому перевірка її коректності є пріоритетним завданням тестування. Для

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		66

виконання тесту було сформовано запит на створення текстового контенту із встановленим часом життя запису та перевіреними параметрами доступу. Після успішного виконання запиту система повинна сформувати унікальний ідентифікатор та посилання для доступу до створеного контенту.

На рисунку 3.17 наведено результат виконання тестування створення текстового запису в середовищі Postman [22]. У результаті виконання запиту сервер повернув код відповіді HTTP 201 та інформацію про створений запис, що містила унікальний хеш, дату завершення дії та сформоване посилання для перегляду. Це підтверджує коректну роботу механізму створення контенту та збереження його параметрів у базі даних.

Отримані дані відповідали очікуваній структурі відповіді сервера, що підтверджує правильність реалізації серверного обробника запиту на створення запису. Генерація унікального хешу відбулась коректно, а параметри запису збережено у базі даних без втрат. Таким чином, тестування підтвердило працездатність базового функціонального модуля системи.



Рис. 3.17. Тестування створення запису

Важливою особливістю розробленої системи є автоматичне блокування доступу до контенту після завершення визначеного проміжку часу. Для перевірки механізму TTL було створено запис із терміном життя одна хвилина та після його завершення здійснено повторну спробу доступу. Такий підхід дозволив перевірити коректність реалізації алгоритму контролю терміну дії в умовах реального виконання.

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		67

На рисунку 3.18 наведено результати перевірки механізму автоматичного завершення терміну дії запису. Після спливання встановленого часу сервер повернув повідомлення про недоступність контенту з кодом HTTP 410, що підтверджує коректну роботу алгоритму контролю TTL [18]. Завдяки цьому механізму забезпечується практична реалізація концепції тимчасового доступу до інформації.

Перевірка підтвердила, що функція TTL працює відповідно до визначених вимог і забезпечує надійне блокування доступу після закінчення терміну дії. Результати тесту відповідають очікуваній поведінці системи, описаній в алгоритмах другого розділу дипломної роботи. Механізм TTL успішно пройшов тестування та готовий до використання в реальних умовах.

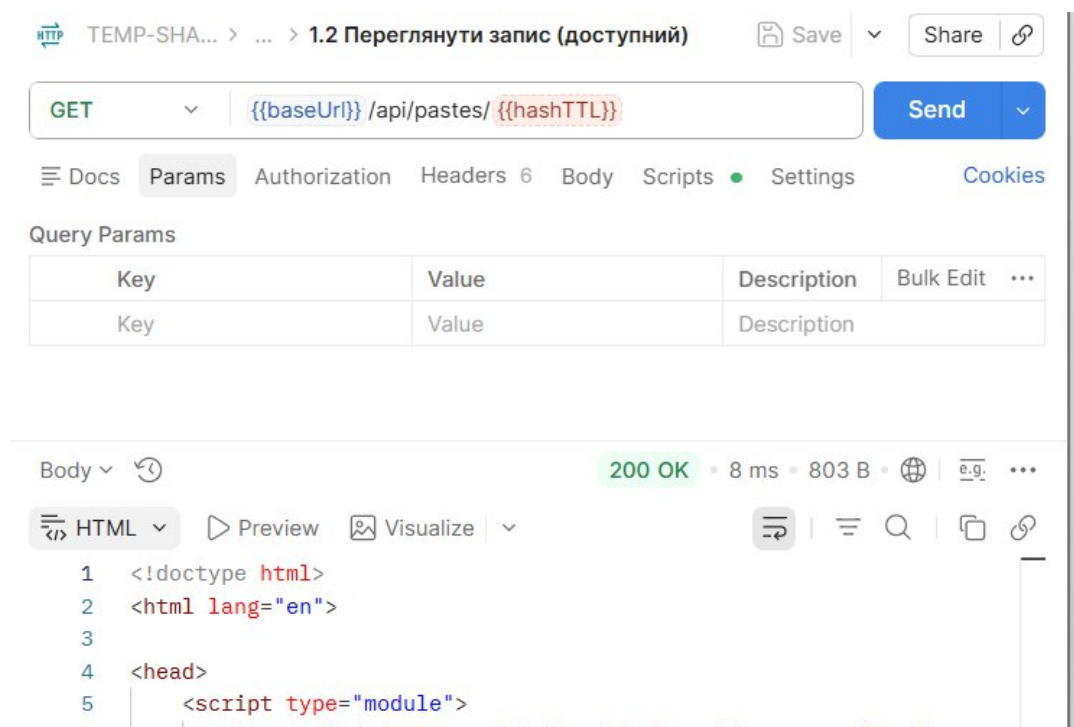


Рис. 3.18. Перевірка терміну дії запису

Для перевірки механізму обмеження доступу за кількістю переглядів було створено запис із максимальним лімітом у два перегляди та послідовно виконано кілька запитів на його отримання. Після кожного відкриття система автоматично збільшувала лічильник переглядів та перевіряла його відносно встановленого обмеження. Такий механізм дозволяє ефективно обмежити поширення конфіденційної інформації між користувачами.

На рисунку 3.19 наведено результати тестування механізму обмеження кількості переглядів. Під час третьої спроби доступу система повернула повідомлення про недоступність запису з кодом HTTP 410, що підтверджує коректне спрацювання алгоритму [18,22]. Перші два запити виконались успішно і повернули вміст запису, що є очікуваною поведінкою.

Отримані результати повністю відповідають вимогам, визначеним під час проєктування системи. Реалізований механізм контролю переглядів забезпечує надійне обмеження доступу після досягнення встановленого ліміту. Тестування підтвердило коректність роботи лічильника переглядів та алгоритму перевірки max_views.

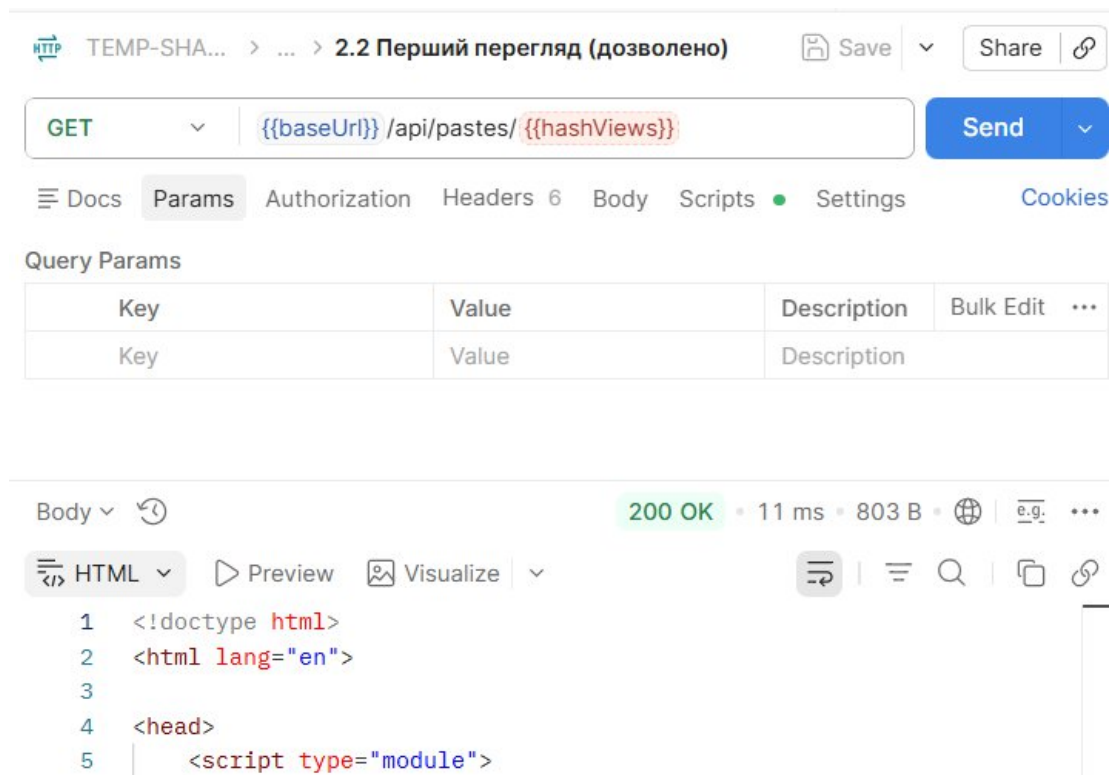


Рис. 3.19. Перевірка ліміту переглядів

Для забезпечення додаткового рівня безпеки в системі реалізовано можливість захисту записів паролем, що перевіряється під час кожного звернення до захищеного контенту. Під час тестування було створено запис із заданим паролем, після чого виконувалися спроби отримання контенту без пароля, з неправильним паролем та з коректними даними. Такий підхід дозволив перевірити всі можливі сценарії використання механізму парольного захисту.

На рисунку 3.20 наведено результати тестування захисту запису паролем. Запити без пароля та з неправильним значенням повернули код HTTP 401, тоді як запит з коректним паролем успішно повернув вміст запису [18]. Пароль передається в заголовок HTTP-запиту x-password, що забезпечує його захист під час транспортування.

Результати тестування підтвердили, що доступ до захищеного запису неможливий без введення правильного пароля. Реалізований механізм парольного захисту відповідає встановленим вимогам безпеки та коректно обробляє всі граничні сценарії. Тестування підтвердило надійність підсистеми захисту контенту паролем.

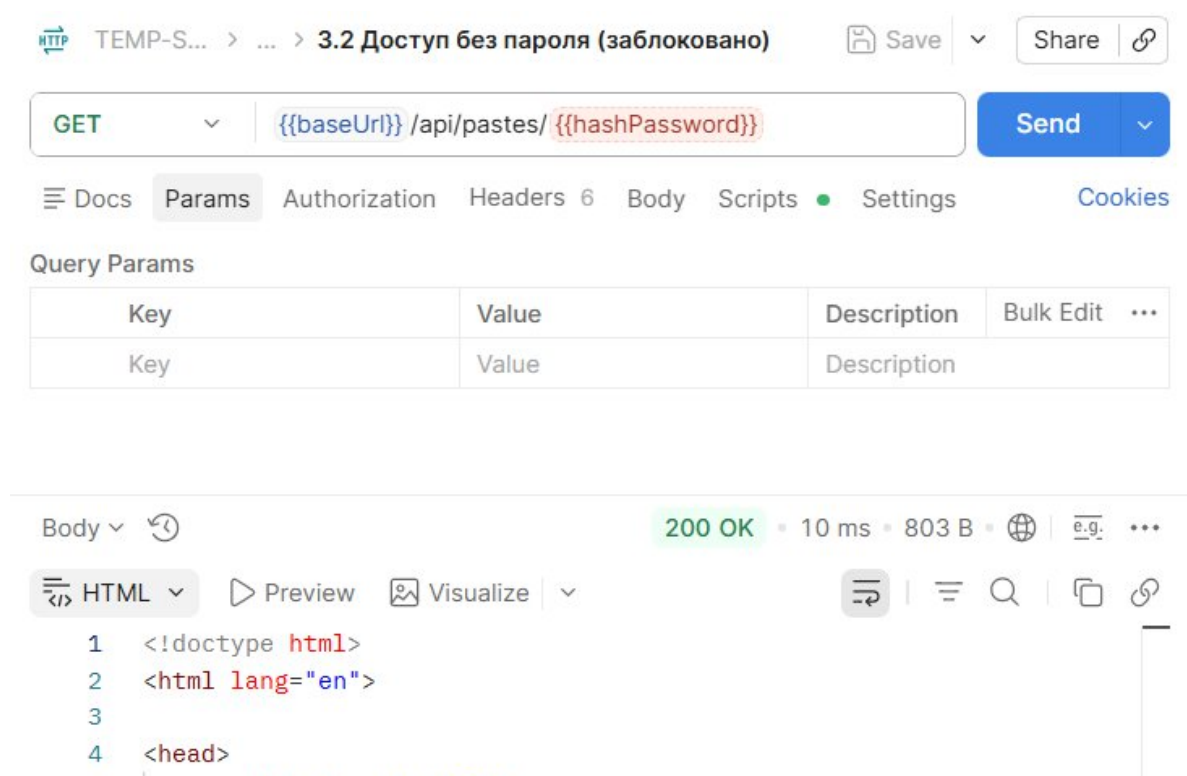


Рис. 3.20. Перевірка захисту паролем

Для перевірки режиму одноразового перегляду було створено запис із активованою функцією автоматичного видалення після першого відкриття. Після першого успішного перегляду система повинна заблокувати подальший доступ до контенту незалежно від інших параметрів запису. Даний режим призначений для передачі конфіденційної інформації, яка має бути доступна лише один раз.

На рисунку 3.21 наведено результати тестування режиму одноразового перегляду. Перший запит успішно повернув вміст запису, а другий запит до того самого ресурсу повернув код HTTP 410 з повідомленням про недоступність. Система автоматично змінила статус запису після першого відкриття та заблокувала повторний доступ [18].

Отримані результати підтверджують коректну реалізацію механізму burn-after-read відповідно до визначених функціональних вимог. Тестування показало, що алгоритм надійно спрацьовує після першого перегляду і не допускає подальшого доступу до контенту. Функція одноразового перегляду є готовою до використання в реальних сценаріях передачі конфіденційної інформації.

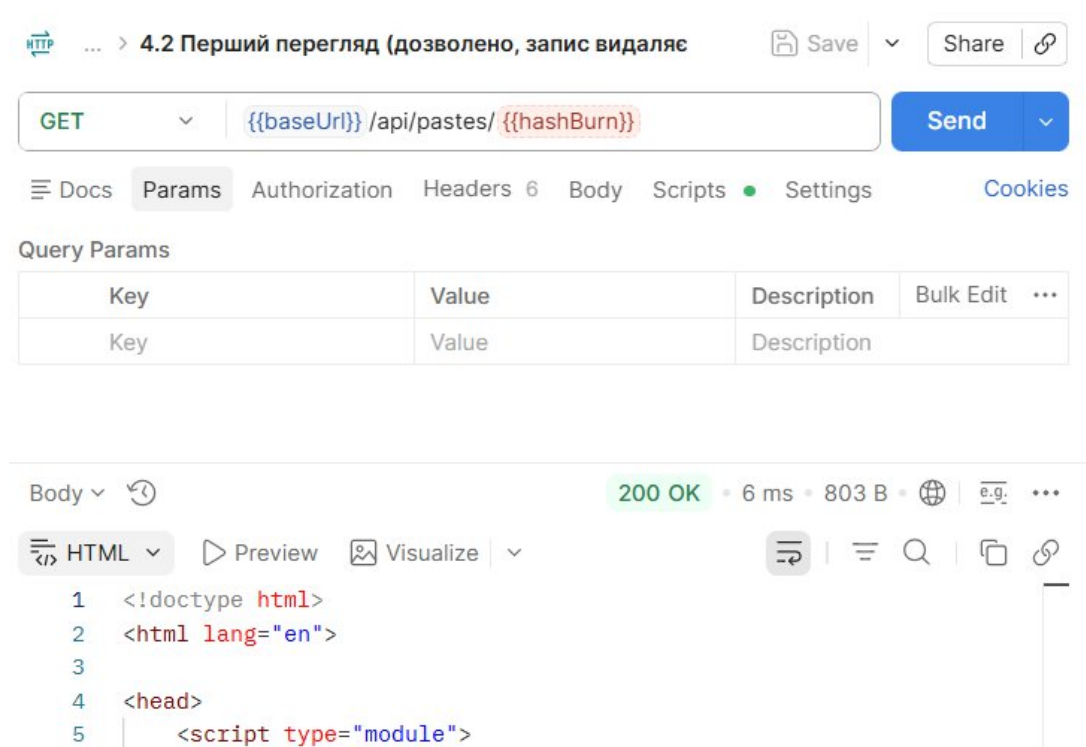


Рис. 3.21. Перевірка одноразового перегляду

Тестування функціоналу завантаження файлів включало перевірку коректності передачі файлів до хмарного сховища Cloudflare R2 та їх подальшого отримання за сформованим посиланням [6]. Для виконання тесту було завантажено файл різних форматів та розмірів, після чого перевірено їх наявність у бакеті через панель керування Cloudflare. Також перевірялась коректність генерації пресайнд-URL для обмеженого за часом доступу до файлів.

На рисунку 3.23 наведено результати тестування роботи файлового сховища. Файл успішно завантажився до бакету Cloudflare R2 та став доступним для завантаження за сформованим посиланням протягом усього терміну дії запису. У базі даних збережено лише метадані файлу та ключ об'єкта у сховищі, що підтверджує коректність реалізованої архітектури.

Тестування підтвердило коректну роботу інтеграції з Cloudflare R2 та механізму генерації посилань для доступу до файлів [6]. Файли різних форматів (зображення, документи, архіви) успішно завантажились і були доступні для скачування. Функціонал роботи з файловим сховищем відповідає визначеним вимогам і готовий до використання в реальних умовах.

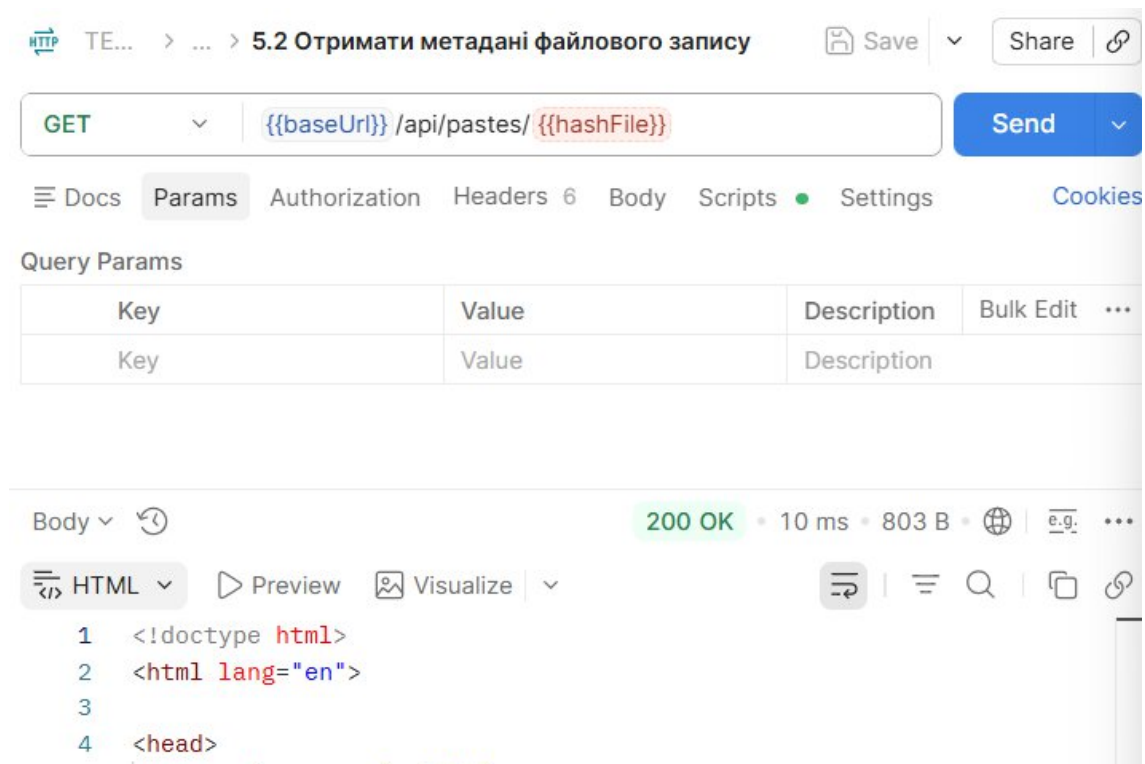


Рис. 3.22. Перевірка роботи файлового сховища

Тестування адміністративних функцій включало перевірку коректності роботи API-ендпоінтів, доступних лише користувачам із роллю admin [25,26]. Для виконання тестів використовувалися запити як від адміністратора, так і від звичайного авторизованого користувача, щоб перевірити коректність розмежування прав доступу. Тестувалися операції перегляду статистики, керування ролями користувачів та видалення довільних записів.

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		72

На рисунку 3.24 наведено результати тестування адміністративної панелі. Запити від адміністратора до захищених ендпоінтів повернули код HTTP 200 та коректні дані, тоді як аналогічні запити від звичайного користувача отримали відмову з кодом HTTP 403. Операція зміни ролі користувача виконалась коректно та відобразилась у відповідях наступних запитів без перезавантаження.

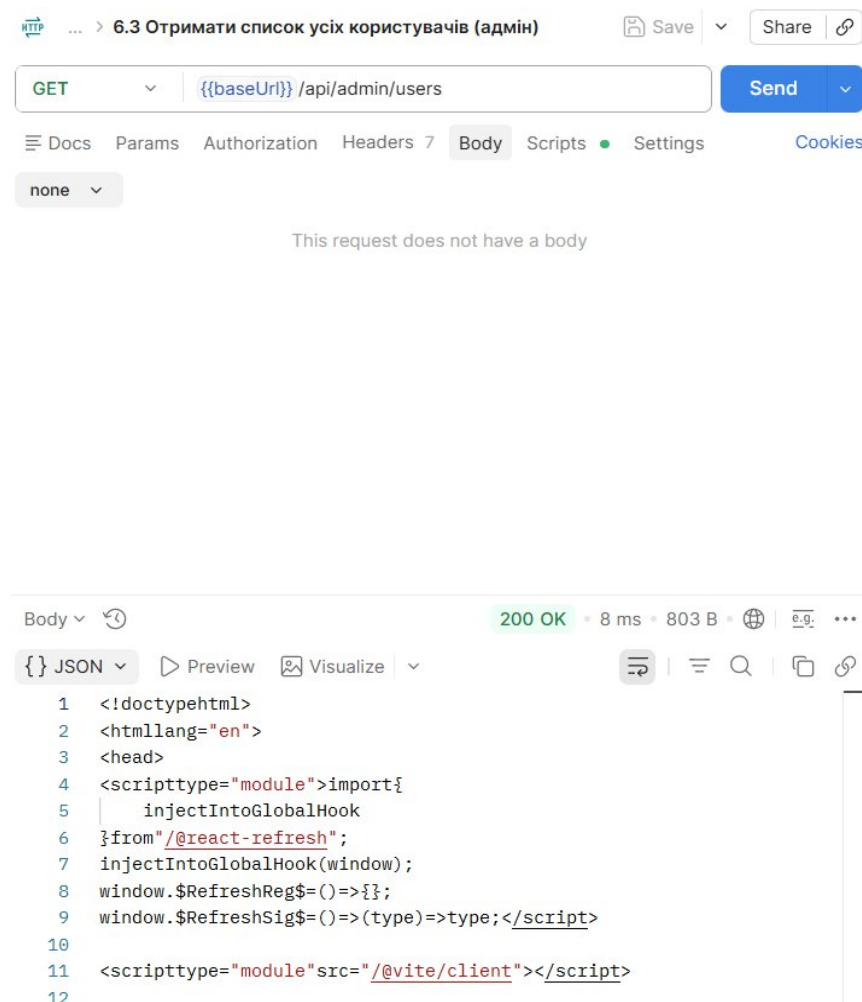


Рис. 3.23. Перевірка адміністративної панелі

Отримані результати підтверджують надійність реалізованого механізму контролю доступу на основі ролей. Адміністративні функції доступні виключно авторизованим адміністраторам, що гарантує захист критично важливих операцій від несанкціонованого використання. Підсистема адміністрування функціонує відповідно до вимог і готова до використання в реальному середовищі.

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		73

Результати функціонального тестування

№	Тест-кейс	Очікуваний результат	Фактичний результат	Статус
1	Створення запису з TTL	HTTP 201, після TTL – HTTP 410	Відповідає очікуваному	Пройдено
2	Ліміт переглядів (max_views)	HTTP 410 після N-го перегляду	Відповідає очікуваному	Пройдено
3	Захист паролем	HTTP 401 без пароля, 200 з вірним	Відповідає очікуваному	Пройдено
4	Burn-after-read	HTTP 410 після 1-го перегляду	Відповідає очікуваному	Пройдено
5	Реєстрація та логін	Токени, коректна обробка помилок	Відповідає очікуваному	Пройдено
6	Завантаження файлу (R2)	Файл у бакеті, коректна віддача	Відповідає очікуваному	Пройдено
7	Продовження TTL	expiresAt збільшено	Відповідає очікуваному	Пройдено
8	Колекції	HTTP 201, GET повертає blocks	Відповідає очікуваному	Пройдено
9	Адмін-ендпоінти	HTTP 200 адміну, 403 юзеру	Відповідає очікуваному	Пройдено

Висновки до третього розділу

У третьому розділі описано порядок встановлення та налаштування системи, структуру інтерфейсу вебзастосунку та підтверджено коректність роботи програмного комплексу за результатами функціонального тестування. Детально розглянуто процес розгортання: встановлення залежностей, налаштування змінних середовища для PostgreSQL, JWT, Cloudflare R2 та SMTP, виконання міграцій бази даних, запуск серверної та клієнтської частин. Описано рольову модель доступу з трьома категоріями користувачів та матрицю відповідних прав.

Описано структуру інтерфейсу семи основних сторінок застосунку: головної сторінки з вкладковою навігацією, сторінки перегляду контенту (ViewPage) зі станами-машиною, сторінок автентифікації, особистого кабінету зі статистикою, адміністративної панелі та сторінки обробки помилки 404. Для кожної сторінки наведено опис відповідних компонентів інтерфейсу, логіку взаємодії з серверною частиною та особливості реалізації. Наведено діаграму розгортання та діаграму компонентів у нотації UML 2.0.

Проведено функціональне тестування дев'яти тест-кейсів через Postman та інтеграційне тестування через вебінтерфейс, за результатами якого всі перевірені сценарії успішно пройдено. Підтверджено коректність реалізації TTL, ліміту переглядів, парольного захисту, режиму burn-after-read, JWT-автентифікації з refresh-токеном, зберігання файлів у Cloudflare R2, продовження TTL, колекцій та адміністративних функцій. Отримані результати свідчать про готовність розробленого програмного комплексу до практичного використання та відповідність сформульованим функціональним вимогам.

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		75

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було досягнуто поставленої мети – розроблено вебсистему тимчасового обміну мультимедійними даними, яка забезпечує створення, зберігання та надання доступу до контенту з можливістю автоматичного обмеження його доступності. Розроблена система дозволяє користувачам безпечно обмінюватися текстовими повідомленнями, зображеннями, відеофайлами та іншими видами мультимедійних даних. Отриманий програмний продукт може використовуватися як практичне рішення для організації тимчасового доступу до інформації в мережі Інтернет.

У процесі виконання роботи було проведено аналіз предметної області та досліджено особливості функціонування сервісів тимчасового доступу до інформації. Результати аналізу показали зростання актуальності таких систем у зв'язку з необхідністю забезпечення конфіденційності даних та контролю за їх поширенням. Також було визначено ключові принципи функціонування подібних сервісів, серед яких обмеження терміну доступності контенту та контроль кількості його переглядів.

На наступному етапі було виконано аналіз існуючих програмних продуктів аналогічного призначення. Проведене дослідження дозволило виявити їх сильні сторони, недоліки та функціональні можливості, які можуть бути використані під час створення власного рішення. На основі отриманих результатів було сформовано перелік функціональних і нефункціональних вимог до розроблюваної системи.

У роботі було обґрунтовано вибір клієнт-серверної архітектури як найбільш доцільної для реалізації поставлених завдань. Такий підхід забезпечує чіткий розподіл функціональності між клієнтською та серверною частинами системи, а також спрощує її супровід і масштабування. Крім того, було обрано сучасні технології та інструментальні засоби розробки, які забезпечують високу продуктивність, надійність та можливість подальшого розвитку програмного продукту.

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		76

У ході проектування було розроблено структуру бази даних та визначено взаємозв'язки між основними сутностями системи. Спроектована база даних забезпечує ефективне зберігання інформації про користувачів, мультимедійний контент, параметри доступу та статистику використання. Також було реалізовано механізми збереження файлів і метаданих, необхідних для коректного функціонування вебсервісу.

Практична частина роботи включала реалізацію серверної частини системи та створення REST API для взаємодії між компонентами програмного комплексу. Було реалізовано набір програмних модулів, що забезпечують обробку запитів користувачів, управління контентом та виконання бізнес-логіки системи. Розроблений програмний інтерфейс дозволяє організувати зручний та стандартизований обмін даними між клієнтською та серверною частинами застосунку.

Окрему увагу було приділено розробці клієнтського вебінтерфейсу системи. Створений інтерфейс забезпечує зручну взаємодію користувача з функціональними можливостями сервісу та дозволяє виконувати всі основні операції з контентом. Під час реалізації було враховано вимоги до зрозумілості навігації, швидкості роботи та адаптивності інтерфейсу.

Важливою складовою роботи стала реалізація механізму автоматичного обмеження доступу до контенту. Розроблений механізм забезпечує автоматичне блокування доступу після завершення встановленого терміну дії або після досягнення максимальної кількості переглядів. Завдяки цьому користувачі отримують додатковий рівень контролю над поширенням своїх даних та можуть самостійно визначати умови їх доступності.

Для перевірки працездатності розробленої системи було проведено тестування її основних функціональних модулів. Під час тестування перевірялася коректність роботи серверної та клієнтської частин, механізмів авторизації, управління контентом та обмеження доступу. Отримані результати підтвердили стабільність роботи системи та її відповідність сформульованим вимогам.

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		77

Таким чином, усі поставлені завдання були успішно виконані, а поставлену мету роботи досягнуто в повному обсязі. Розроблена вебсистема забезпечує ефективний і безпечний тимчасовий обмін мультимедійними даними та відповідає сучасним вимогам до подібних програмних продуктів. Отримані результати можуть бути використані для подальшого розвитку системи та розширення її функціональних можливостей.

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		78

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Node.js Documentation. Node.js v18 API Reference [Електронний ресурс]. – Режим доступу: <https://nodejs.org/docs/latest-v18.x/api/> (дата звернення: 15.03.2026).
2. Fastify Documentation. Fastify – Fast and low overhead web framework for Node.js [Електронний ресурс]. – Режим доступу: <https://fastify.dev/docs/latest/> (дата звернення: 15.03.2026).
3. React Documentation. React – The library for web and native user interfaces [Електронний ресурс]. – Режим доступу: <https://react.dev/> (дата звернення: 15.03.2026).
4. TypeScript Documentation. TypeScript Handbook [Електронний ресурс]. – Режим доступу: <https://www.typescriptlang.org/docs/> (дата звернення: 15.03.2026).
5. PostgreSQL Documentation. PostgreSQL 16 Documentation [Електронний ресурс]. – Режим доступу: <https://www.postgresql.org/docs/16/> (дата звернення: 15.03.2026).
6. Cloudflare R2 Documentation. Cloudflare R2 Object Storage [Електронний ресурс]. – Режим доступу: <https://developers.cloudflare.com/r2/> (дата звернення: 15.03.2026).
7. Amazon Web Services. AWS SDK for JavaScript v3 Developer Guide [Електронний ресурс]. – Режим доступу: <https://docs.aws.amazon.com/sdk-for-javascript/v3/developer-guide/> (дата звернення: 15.03.2026).
8. Jones M., Bradley J., Sakimura N. RFC 7519: JSON Web Token (JWT) [Електронний ресурс]. – Internet Engineering Task Force (IETF), 2015. – Режим доступу: <https://datatracker.ietf.org/doc/html/rfc7519> (дата звернення: 15.03.2026).
9. React Router Documentation. React Router v6 [Електронний ресурс]. – Режим доступу: <https://reactrouter.com/en/main> (дата звернення: 15.03.2026).
10. Vite Documentation. Vite – Next Generation Frontend Tooling [Електронний ресурс]. – Режим доступу: <https://vitejs.dev/guide/> (дата звернення: 15.03.2026).

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		79

11. nanoid. A tiny, secure, URL-friendly, unique string ID generator for JavaScript [Електронний ресурс]. – Режим доступу: <https://github.com/ai/nanoid> (дата звернення: 15.03.2026).
12. bcryptjs. Optimized bcrypt in JavaScript [Електронний ресурс]. – Режим доступу: <https://github.com/dcodeIO/bcrypt.js> (дата звернення: 15.03.2026).
13. @fastify/jwt. JWT utils for Fastify [Електронний ресурс]. – Режим доступу: <https://github.com/fastify/fastify-jwt> (дата звернення: 15.03.2026).
14. @fastify/multipart. Multipart support for Fastify [Електронний ресурс]. – Режим доступу: <https://github.com/fastify/fastify-multipart> (дата звернення: 15.03.2026).
15. pg. Non-blocking PostgreSQL client for Node.js [Електронний ресурс]. – Режим доступу: <https://node-postgres.com/> (дата звернення: 15.03.2026).
16. Web Audio API. W3C Working Draft [Електронний ресурс]. – Режим доступу: <https://www.w3.org/TR/webaudio/> (дата звернення: 15.03.2026).
17. MediaRecorder API. MDN Web Docs [Електронний ресурс]. – Режим доступу: <https://developer.mozilla.org/en-US/docs/Web/API/MediaRecorder> (дата звернення: 15.03.2026).
18. Mozilla Developer Network. HTTP – MDN Web Docs [Електронний ресурс]. – Режим доступу: <https://developer.mozilla.org/en-US/docs/Web/HTTP> (дата звернення: 15.03.2026).
19. OWASP Foundation. OWASP Top Ten 2021 [Електронний ресурс]. – Режим доступу: <https://owasp.org/www-project-top-ten/> (дата звернення: 15.03.2026).
20. OWASP Foundation. JSON Web Token Cheat Sheet for Java [Електронний ресурс]. – Режим доступу: https://cheatsheetseries.owasp.org/cheatsheets/JSON_Web_Token_for_Java_Cheat_Sheet.html (дата звернення: 15.03.2026).

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		80

21. Docker Documentation. Docker Overview [Електронний ресурс]. – Режим доступу: <https://docs.docker.com/get-started/overview/> (дата звернення: 15.03.2026).
22. Postman Documentation. Postman API Platform [Електронний ресурс]. – Режим доступу: <https://learning.postman.com/docs/getting-started/introduction/> (дата звернення: 15.03.2026).
23. Git Documentation. Git Reference Manual [Електронний ресурс]. – Режим доступу: <https://git-scm.com/doc> (дата звернення: 15.03.2026).
24. UML 2.5.1 Specification. Unified Modeling Language Specification[Електронний ресурс]. – Object Management Group, 2017. – Режим доступу: <https://www.omg.org/spec/UML/2.5.1> (дата звернення: 15.03.2026).
25. Ford N., Parsons R., Kua P. Building Evolutionary Architectures: Support Constant Change. 2nd ed. Sebastopol : O'Reilly Media, 2022. 240 p.
26. Richards M., Ford N. Fundamentals of Software Architecture: An Engineering Approach. 2nd ed. Sebastopol : O'Reilly Media, 2024. 480 p.
27. Pautasso C., Zimmermann O., Amundsen M. RESTful Web APIs at Scale. IEEE Internet Computing. 2022. Vol. 26, No. 5. P. 7–15. DOI: 10.1109/MIC.2022.3189869.
28. Fett D., Küsters R., Schmitz G. The Web's Identity Crisis: Understanding the Security of Web Login. IEEE Security & Privacy. 2022. Vol. 20, No. 3. P. 55–64. DOI: 10.1109/MSEC.2022.3153613.
29. Bertocci V. Modern Authentication with OAuth 2.0 and OpenID Connect. Sebastopol : O'Reilly Media, 2023. 350 p.
30. Porru S., Pinna A., Baralla G., Marchesi M. Security Smells and Anti-patterns in JWT-based Authentication. Information and Software Technology. 2024. Vol. 168. Article 107203. DOI: 10.1016/j.infsof.2023.107203.
31. Prusty S., Sanyal S. Evaluating Node.js Web Frameworks: A Benchmark of Fastify, Express, and Koa. Journal of Web Engineering. 2022. Vol. 21, No. 5. P. 1431–1452. DOI: 10.13052/jwe1540-9589.2153.

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		81

32. Kikas R., Dumas M., Pfahl D. Security Practices in Node.js Ecosystem: Empirical Study. Empirical Software Engineering. 2022. Vol. 27, No. 5. Article 114. DOI: 10.1007/s10664-022-10172-9.

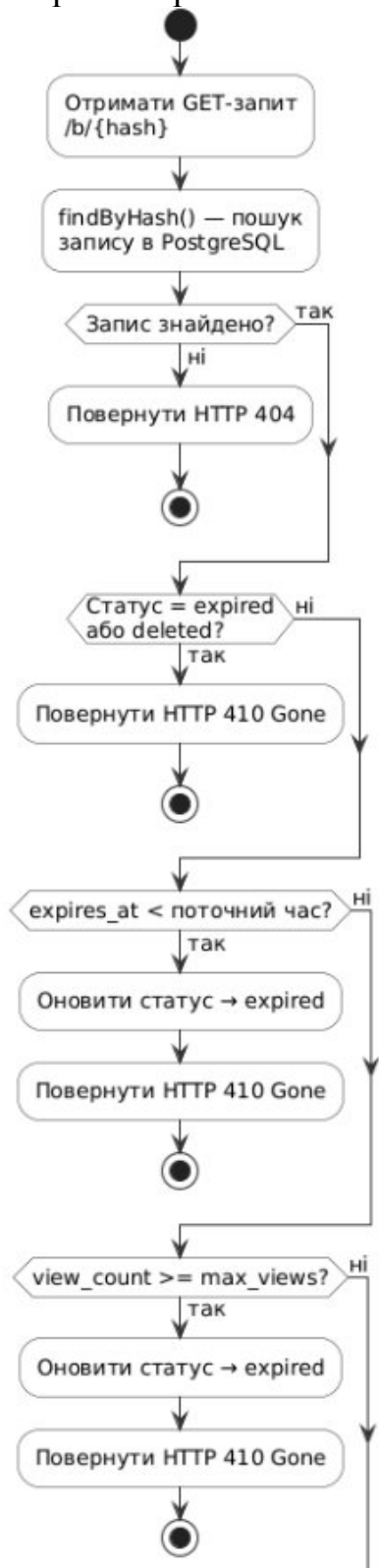
33. Tilkov S., Vinoski S. REST Anti-patterns and Pitfalls at Scale. IEEE Internet Computing. 2023. Vol. 27, No. 4. P. 74–83. DOI: 10.1109/MIC.2023.3259081.

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		82

ДОДАТКИ

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		83

Алгоритм отримання контенту:



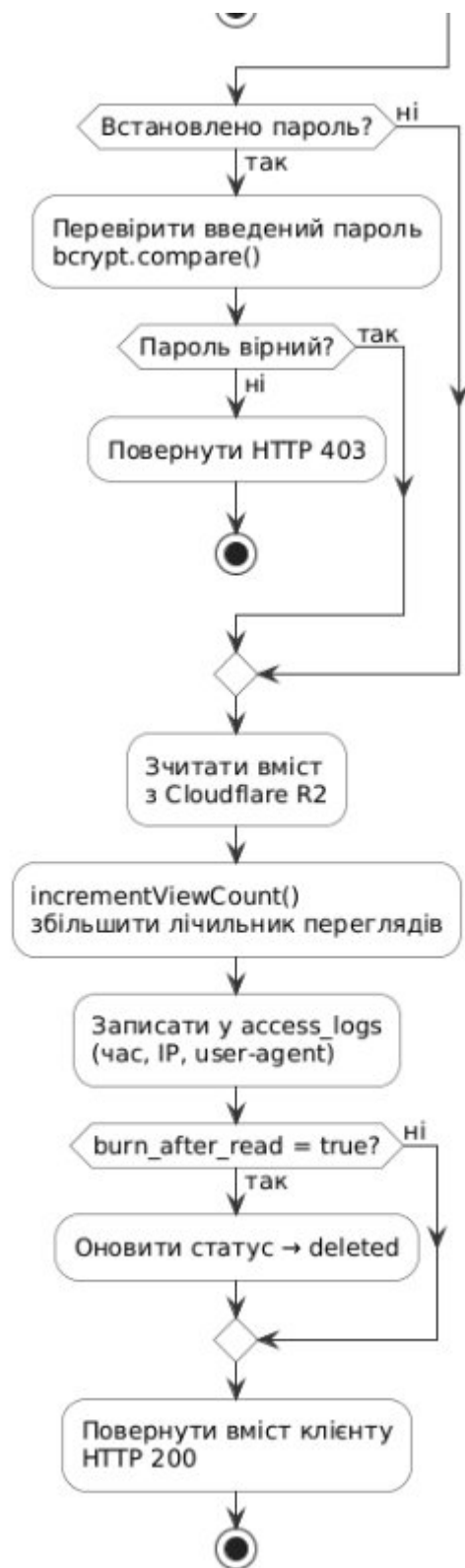


Рис. А. 1. Алгоритм отримання контенту

Алгоритм аутентифікації користувача:



Рис. Б. 1. Алгоритм аутентифікації користувача

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				86
Змн.	Арк.	№ докум.	Підпис	Дата		

Алгоритм автоматичного очищення прострочених записів:



Рис. В. 1. Алгоритм автоматичного очищення прострочених записів

Лістинг модуля генерації унікальних хешів:

```
import { nanoid } from 'nanoid';
import { contentRepository } from './content-repository.js';

const HASH_LENGTH = 8;
const MAX_ATTEMPTS = 5;

export async function generateUniqueHash(): Promise<string> {
  for (let attempt = 0; attempt < MAX_ATTEMPTS; attempt++) {
    const hash = nanoid(HASH_LENGTH);
    const existing = await contentRepository.findByHash(hash);
    if (!existing) return hash;
  }
  throw new Error('Failed to generate unique hash after max attempts');
}
```

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		88

Лістинг коду модуля роботи зі сховищем файлів:

```
import {
  S3Client,
  PutObjectCommand,
  GetObjectCommand,
  DeleteObjectCommand,
} from '@aws-sdk/client-s3';
import { getSignedUrl } from '@aws-sdk/s3-request-presigner';
import { randomUUID } from 'node:crypto';
import 'dotenv/config';

const s3 = new S3Client({
  region: process.env.AWS_REGION!,
  endpoint: process.env.AWS_ENDPOINT!,
  credentials: {
    accessKeyId: process.env.AWS_ACCESS_KEY_ID!,
    secretAccessKey: process.env.AWS_SECRET_ACCESS_KEY!,
  },
});

const BUCKET = process.env.AWS_BUCKET_NAME!;

export async function saveTextContent(text: string): Promise<string> {
  const storageKey = `${randomUUID()}.txt`;
  await s3.send(new PutObjectCommand({
    Bucket: BUCKET,
    Key: storageKey,
    Body: text,
    ContentType: 'text/plain; charset=utf-8',
  }));
  return storageKey;
}

export async function saveFileContent(
  buffer: Buffer,
  filename: string,
  mimeType?: string
): Promise<string> {
  const safeName = filename.replace(/[^a-zA-Z0-9._-]/g, '_');
  const storageKey = `${randomUUID()}-${safeName}`;
  await s3.send(new PutObjectCommand({
    Bucket: BUCKET,
    Key: storageKey,
    Body: buffer,
    ContentType: mimeType ?? 'application/octet-stream',
  }));
  return storageKey;
}
```

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		89

```

export async function readContent(storageKey: string): Promise<Buffer> {
  const response = await s3.send(new GetObjectCommand({
    Bucket: BUCKET,
    Key: storageKey,
  }));
  const stream = response.Body as NodeJS.ReadableStream;
  return new Promise((resolve, reject) => {
    const chunks: Buffer[] = [];
    stream.on('data', (chunk) => chunks.push(Buffer.from(chunk)));
    stream.on('end', () => resolve(Buffer.concat(chunks)));
    stream.on('error', reject);
  });
}

```

```

export async function getPresignedUrl(
  storageKey: string,
  expiresInSeconds = 3600
): Promise<string> {
  return getSignedUrl(
    s3,
    new GetObjectCommand({ Bucket: BUCKET, Key: storageKey }),
    { expiresIn: expiresInSeconds }
  );
}

```

```

export async function deleteContent(storageKey: string): Promise<void> {
  await s3.send(new DeleteObjectCommand({
    Bucket: BUCKET,
    Key: storageKey,
  }));
}

```

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		90

Лістинг коду репозиторію контентних блоків:

```
import { pool } from '../db/pool.js';
import type { ContentBlock } from '../types/content.js';

// Перетворює рядок з БД у ContentBlock
function mapRow(row: Record<string, unknown>): ContentBlock {
  return {
    id: row.id as number,
    hash: row.hash as string,
    contentType: row.content_type as ContentBlock['contentType'],
    storageKey: row.storage_key as string,
    originalFilename: row.original_filename as string | undefined,
    mimeType: row.mime_type as string | undefined,
    textPreview: row.text_preview as string | undefined,
    passwordHash: row.password_hash as string | undefined,
    maxViews: row.max_views as number | undefined,
    viewCount: row.view_count as number,
    burnAfterRead: row.burn_after_read as boolean,
    expiresAt: (row.expires_at as Date).toISOString(),
    status: row.status as ContentBlock['status'],
    createdAt: (row.created_at as Date).toISOString(),
  };
}

export const contentRepository = {
  async create(data: Omit<ContentBlock, 'id' | 'viewCount' | 'createdAt'> & { userId?: string }):
  Promise<ContentBlock> {
    const { rows } = await pool.query(
      `INSERT INTO content_blocks
      (hash, content_type, storage_key, original_filename, mime_type,
      text_preview, password_hash, max_views, burn_after_read, expires_at, status, user_id)
      VALUES ($1,$2,$3,$4,$5,$6,$7,$8,$9,$10,'active',$11)
      RETURNING *`,
      [
        data.hash,
        data.contentType,
        data.storageKey,
        data.originalFilename ?? null,
        data.mimeType ?? null,
        data.textPreview ?? null,
        data.passwordHash ?? null,
        data.maxViews ?? null,
        data.burnAfterRead,
        data.expiresAt,
        data.userId ?? null,
      ]
    );
  }
};
```

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		91

```

return mapRow(rows[0]);
},

async findByHash(hash: string): Promise<ContentBlock | null> {
  const { rows } = await pool.query(
    'SELECT * FROM content_blocks WHERE hash = $1',
    [hash]
  );
  return rows[0] ? mapRow(rows[0]) : null;
},

async incrementViewCount(hash: string): Promise<void> {
  await pool.query(
    'UPDATE content_blocks SET view_count = view_count + 1 WHERE hash = $1',
    [hash]
  );
},

async markExpired(hash: string): Promise<void> {
  await pool.query(
    'UPDATE content_blocks SET status = 'expired' WHERE hash = $1',
    [hash]
  );
},

async markDeleted(hash: string): Promise<void> {
  await pool.query(
    'UPDATE content_blocks SET status = 'deleted' WHERE hash = $1',
    [hash]
  );
},

async findExpired(nowIso: string): Promise<ContentBlock[]> {
  const { rows } = await pool.query(
    `SELECT * FROM content_blocks
    WHERE status = 'active' AND expires_at <= $1`,
    [nowIso]
  );
  return rows.map(mapRow);
},

async list(): Promise<ContentBlock[]> {
  const { rows } = await pool.query(
    'SELECT * FROM content_blocks ORDER BY created_at DESC'
  );
  return rows.map(mapRow);
},

async deleteByHash(hash: string): Promise<void> {
  await pool.query('DELETE FROM content_blocks WHERE hash = $1', [hash]);
},

```

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		92

Лістинг коду сервісу управління контентними блоками:

```
import bcrypt from 'bcryptjs';
import { generateUniqueHash } from './hash-generator.js';
import { contentRepository } from './content-repository.js';
import { deleteContent, readContent, saveFileContent, saveTextContent } from '../storage/local-storage.js';
import type { ContentBlock, CreateTextDTO, CreateFileDTO } from '../types/content.js';

function nowIso(): string {
  return new Date().toISOString();
}

function computeExpiration(ttlMinutes: number): string {
  return new Date(Date.now() + ttlMinutes * 60_000).toISOString();
}

function ensureActive(block: ContentBlock): ContentBlock {
  if (block.status !== 'active') {
    throw new Error('CONTENT_NOT_AVAILABLE');
  }
  if (block.expiresAt <= nowIso()) {
    void contentRepository.markExpired(block.hash);
    throw new Error('CONTENT_EXPIRED');
  }
  if (block.maxViews != null && block.viewCount >= block.maxViews) {
    void contentRepository.markExpired(block.hash);
    throw new Error('CONTENT_EXPIRED');
  }
  return block;
}

export async function createTextBlock(dto: CreateTextDTO): Promise<ContentBlock> {
  const hash = await generateUniqueHash();
  const storageKey = await saveTextContent(dto.text);
  const passwordHash = dto.password
    ? await bcrypt.hash(dto.password, 10)
    : undefined;

  const block = await contentRepository.create({
    hash,
    contentType: 'text',
    storageKey,
    textPreview: dto.text.slice(0, 120),
    passwordHash,
    maxViews: dto.maxViews,
    burnAfterRead: dto.burnAfterRead ?? false,
    expiresAt: computeExpiration(dto.ttlMinutes),
  });
}
```

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		93

```

    status: 'active',
    userId: dto.userId, // ← додати цей рядок
  });

  return block;
}

export async function createFileBlock(dto: CreateFileDTO): Promise<ContentBlock> {
  const hash = await generateUniqueHash();
  const storageKey = await saveFileContent(dto.fileBuffer, dto.filename);
  const passwordHash = dto.password
    ? await bcrypt.hash(dto.password, 10)
    : undefined;

  const block = await contentRepository.create({
    hash,
    contentType: 'file',
    storageKey,
    originalFilename: dto.filename,
    mimeType: dto.mimeType,
    passwordHash,
    maxViews: dto.maxViews,
    burnAfterRead: dto.burnAfterRead ?? false,
    expiresAt: computeExpiration(dto.ttlMinutes),
    status: 'active',
    userId: dto.userId, // ← додати цей рядок
  });

  return block;
}

export async function getBlockMetadata(hash: string): Promise<ContentBlock> {
  const block = await contentRepository.findByHash(hash);
  if (!block) throw new Error('CONTENT_NOT_FOUND');
  return ensureActive(block);
}

export async function getBlockContent(
  hash: string,
  password?: string
): Promise<{ metadata: ContentBlock; data: Buffer }> {
  const block = await contentRepository.findByHash(hash);
  if (!block) throw new Error('CONTENT_NOT_FOUND');

  ensureActive(block);

  if (block.passwordHash) {
    if (!password) throw new Error('PASSWORD_REQUIRED');
  }
}

```

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		94

```

    const ok = await bcrypt.compare(password, block.passwordHash);
    if (!ok) throw new Error('WRONG_PASSWORD');
  }

  await contentRepository.incrementViewCount(hash);

  const newCount = block.viewCount + 1;
  if (block.burnAfterRead || (block.maxViews !== null && newCount >= block.maxViews)) {
    await contentRepository.markExpired(hash);
    await deleteContent(block.storageKey);
  }

  const data = await readContent(block.storageKey);
  return { metadata: block, data };
}

export async function deleteBlock(hash: string): Promise<void> {
  const block = await contentRepository.findByHash(hash);
  if (!block) throw new Error('CONTENT_NOT_FOUND');
  await deleteContent(block.storageKey);
  await contentRepository.deleteByHash(hash);
}

export async function cleanupExpiredBlocks(): Promise<number> {
  const expired = await contentRepository.findExpired(nowIso());
  for (const item of expired) {
    await contentRepository.markExpired(item.hash);
    await deleteContent(item.storageKey);
  }
  return expired.length;
}

export async function listBlocks(): Promise<ContentBlock[]> {
  return contentRepository.list();
}

```

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		95

Лістинг коду репозиторію автентифікації користувачів:

```
import { pool } from '../db/pool.js';

export interface User {
  id: string;
  email: string;
  password_hash: string;
  created_at: string;
}

export interface Session {
  id: string;
  user_id: string;
  token_hash: string;
  expires_at: string;
}

export interface User {
  id: string;
  email: string;
  password_hash: string;
  role: string;
  created_at: string;
}

export const authRepository = {
  async findUserByEmail(email: string): Promise<User | null> {
    const { rows } = await pool.query(
      'SELECT * FROM users WHERE email = $1',
      [email]
    );
    return rows[0] ?? null;
  },

  async findUserById(id: string): Promise<User | null> {
    const { rows } = await pool.query(
      'SELECT * FROM users WHERE id = $1',
      [id]
    );
    return rows[0] ?? null;
  },

  async createUser(email: string, passwordHash: string): Promise<User> {
    const { rows } = await pool.query(
      'INSERT INTO users (email, password_hash) VALUES ($1, $2) RETURNING *',
      [email, passwordHash]
    );
    return rows[0];
  }
};
```

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		96

```

},

async createSession(userId: string, tokenHash: string, expiresAt: string): Promise<Session> {
  const { rows } = await pool.query(
    'INSERT INTO sessions (user_id, token_hash, expires_at) VALUES ($1, $2, $3) RETURNING *',
    [userId, tokenHash, expiresAt]
  );
  return rows[0];
},

async findSession(tokenHash: string): Promise<Session | null> {
  const { rows } = await pool.query(
    'SELECT * FROM sessions WHERE token_hash = $1 AND expires_at > NOW()',
    [tokenHash]
  );
  return rows[0] ?? null;
},

async deleteSession(tokenHash: string): Promise<void> {
  await pool.query('DELETE FROM sessions WHERE token_hash = $1', [tokenHash]);
},

async deleteUserSessions(userId: string): Promise<void> {
  await pool.query('DELETE FROM sessions WHERE user_id = $1', [userId]);
},
};

```

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		97

Лістинг коду сервісу автентифікації та керування сесіями:

```
import bcrypt from 'bcryptjs';
import crypto from 'node:crypto';
import { authRepository } from './auth-repository.js';

export interface AuthResult {
  accessToken: string;
  refreshToken: string;
  user: { id: string; email: string; role: string };
}

function generateRefreshToken(): string {
  return crypto.randomBytes(32).toString('hex');
}

function hashToken(token: string): string {
  return crypto.createHash('sha256').update(token).digest('hex');
}

function refreshTokenExpiresAt(): string {
  return new Date(Date.now() + 7 * 24 * 60 * 60 * 1000).toISOString();
}

export const authService = {
  async register(email: string, password: string, signJwt: (payload: object) => string):
  Promise<AuthResult> {
    const existing = await authRepository.findUserByEmail(email);
    if (existing) throw new Error('EMAIL_TAKEN');
    if (password.length < 6) throw new Error('PASSWORD_TOO_SHORT');

    const passwordHash = await bcrypt.hash(password, 12);
    const user = await authRepository.createUser(email, passwordHash);

    const accessToken = signJwt({ sub: user.id, email: user.email, role: user.role });
    const refreshToken = generateRefreshToken();
    await authRepository.createSession(user.id, hashToken(refreshToken), refreshTokenExpiresAt());

    return { accessToken, refreshToken, user: { id: user.id, email: user.email, role: user.role } };
  },

  async login(email: string, password: string, signJwt: (payload: object) => string):
  Promise<AuthResult> {
    const user = await authRepository.findUserByEmail(email);
    if (!user) throw new Error('INVALID_CREDENTIALS');

    const ok = await bcrypt.compare(password, user.password_hash);
    if (!ok) throw new Error('INVALID_CREDENTIALS');
```

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		98

```

const accessToken = signJwt({ sub: user.id, email: user.email, role: user.role });
const refreshToken = generateRefreshToken();
await authRepository.createSession(user.id, hashToken(refreshToken), refreshTokenExpiresAt());

return { accessToken, refreshToken, user: { id: user.id, email: user.email, role: user.role } };
},

async refresh(refreshToken: string, signJwt: (payload: object) => string): Promise<{ accessToken:
string }> {
    const tokenHash = hashToken(refreshToken);
    const session = await authRepository.findSession(tokenHash);
    if (!session) throw new Error('INVALID_REFRESH_TOKEN');

    const user = await authRepository.findUserById(session.user_id);
    if (!user) throw new Error('INVALID_REFRESH_TOKEN');

    const accessToken = signJwt({ sub: user.id, email: user.email, role: user.role });
    return { accessToken };
},

async logout(refreshToken: string): Promise<void> {
    const tokenHash = hashToken(refreshToken);
    await authRepository.deleteSession(tokenHash);
},
};

```

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		99

Лістинг коду репозиторію колекції даних:

```
import { pool } from '../db/pool.js';

export const collectionRepository = {
  async create(hash: string, userId: string | null, title: string | null, expiresAt: string) {
    const { rows } = await pool.query(
      'INSERT INTO collections (hash, user_id, title, expires_at) VALUES ($1,$2,$3,$4) RETURNING *',
      [hash, userId, title, expiresAt]
    );
    return rows[0];
  },

  async findByHash(hash: string) {
    const { rows } = await pool.query(
      'SELECT * FROM collections WHERE hash = $1',
      [hash]
    );
    return rows[0] ?? null;
  },

  async addBlock(collectionId: number, blockId: number) {
    await pool.query(
      'INSERT INTO collection_blocks (collection_id, block_id) VALUES ($1,$2) ON CONFLICT DO NOTHING',
      [collectionId, blockId]
    );
  },

  async getBlocks(collectionId: number) {
    const { rows } = await pool.query(
      `SELECT cb.* FROM content_blocks cb
      JOIN collection_blocks col ON col.block_id = cb.id
      WHERE col.collection_id = $1`,
      [collectionId]
    );
    return rows;
  },

  async deleteByHash(hash: string) {
    await pool.query('DELETE FROM collections WHERE hash = $1', [hash]);
  },
};
```

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		100

SQL-скрипт створення таблиці користувачів:

sql

```
CREATE TABLE IF NOT EXISTS users (
  id      UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  email   VARCHAR(255) UNIQUE NOT NULL,
  password_hash TEXT NOT NULL,
  role    VARCHAR(10) NOT NULL DEFAULT 'user'
          CHECK (role IN ('user', 'admin')),
  created_at  TIMESTAMPTZ NOT NULL DEFAULT NOW(),
  updated_at  TIMESTAMPTZ NOT NULL DEFAULT NOW()
);
CREATE INDEX IF NOT EXISTS idx_users_email ON users(email);
```

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		101

SQL-скрипт створення таблиці контентних блоків:

sql

```
CREATE TABLE IF NOT EXISTS content_blocks (
  id          SERIAL PRIMARY KEY,
  hash        VARCHAR(12) UNIQUE NOT NULL,
  user_id     UUID REFERENCES users(id) ON DELETE SET NULL,
  content_type VARCHAR(10) NOT NULL CHECK (content_type IN ('text', 'file')),
  storage_key  TEXT NOT NULL,
  original_filename TEXT,
  mime_type   TEXT,
  text_preview TEXT,
  password_hash TEXT,
  max_views   INT CHECK (max_views > 0),
  view_count  INT NOT NULL DEFAULT 0,
  burn_after_read BOOLEAN NOT NULL DEFAULT false,
  expires_at  TIMESTAMPTZ NOT NULL,
  status      VARCHAR(10) NOT NULL DEFAULT 'active'
              CHECK (status IN ('active', 'expired', 'deleted')),
  created_at  TIMESTAMPTZ NOT NULL DEFAULT NOW()
);
```

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		102

SQL-скрипт створення індексів для таблиці контентних блоків:

sql

```
CREATE UNIQUE INDEX IF NOT EXISTS idx_content_blocks_hash
```

```
ON content_blocks(hash);
```

```
CREATE INDEX IF NOT EXISTS idx_content_blocks_user_id
```

```
ON content_blocks(user_id) WHERE user_id IS NOT NULL;
```

```
CREATE INDEX IF NOT EXISTS idx_content_blocks_expires_at
```

```
ON content_blocks(expires_at) WHERE status = 'active';
```

```
CREATE INDEX IF NOT EXISTS idx_content_blocks_status
```

```
ON content_blocks(status);
```

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		103

SQL-скрипти створення таблиці сесії користувачів:

sql

```
CREATE TABLE IF NOT EXISTS sessions (
  id      UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  user_id UUID NOT NULL REFERENCES users(id) ON DELETE CASCADE,
  token_hash TEXT UNIQUE NOT NULL,
  expires_at TIMESTAMPTZ NOT NULL,
  created_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);

CREATE INDEX IF NOT EXISTS idx_sessions_user_id ON sessions(user_id);
CREATE INDEX IF NOT EXISTS idx_sessions_token_hash ON
sessions(token_hash);
CREATE INDEX IF NOT EXISTS idx_sessions_expires_at ON sessions(expires_at);
```

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		104

SQL-скрипт створення таблиці колекції:

```
sql
CREATE TABLE IF NOT EXISTS collections (
  id      SERIAL PRIMARY KEY,
  hash    VARCHAR(12) UNIQUE NOT NULL,
  user_id UUID REFERENCES users(id) ON DELETE SET NULL,
  title   TEXT,
  expires_at TIMESTAMPTZ NOT NULL,
  created_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);
```

SQL-скрипт створення таблиці журналу доступу:

sql

```
CREATE TABLE IF NOT EXISTS access_logs (
  id      SERIAL PRIMARY KEY,
  block_id INT NOT NULL REFERENCES content_blocks(id) ON DELETE
CASCADE,
  user_id  UUID REFERENCES users(id) ON DELETE SET NULL,
  ip_address INET,
  user_agent TEXT,
  accessed_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);

CREATE INDEX IF NOT EXISTS idx_access_logs_block_id ON
access_logs(block_id);

CREATE INDEX IF NOT EXISTS idx_access_logs_accessed_at ON
access_logs(accessed_at);
```

		Челядин Т. А.			ІПЗ.КР.Б-121-26ПЗ	Арк.
		Лобанчикова Н. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		106