МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: 123 «Комп'ютерна інженерія»

Освітньо-професійна програма: «Безпека комп'ютерних систем і мереж»

Група: 4КБ-02

Дипломний проєкт

здобувача освіти денної форми навчання
КБ.02.06.000.ДП

КЛЕМЕНЧУК
ЮЛІЇ СЕРГІЇВНИ

м. Одеса
2025 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: **123 «Комп'ютерна інженерія»**

Освітньо-професійна програма: «**Безпека комп'ютерних систем і мереж»**

Група: **4КБ-02**

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломного проєкту на тему:

Розробка додатку для керування файловим сховищем з автентифікацією користувачів в глобальній мережі

Проектний матеріал складається з пояснювальної записки на 86 сторінках та графічного (презентаційного) матеріалу на 11 аркушах (слайдах)

Дипломник _____ (Клеменчук Ю.С.)

Керівник _____ (Гаджиєв М.М.)

Консультанти:

з економічного розділу _____ (Канський М.Ю.)

з розділу охорони праці та техніки безпеки _____ (Чорновол Н.І.)

з нормоконтролю _____ (Петрашова В.І.)

старший консультант _____ (Кривченко Ю.В.)

До захисту допущений

Голова циклової комісії _____ (Кривченко Ю.В.)

Завідувач відділення _____ (Краснокутська К.Г.)

Захист «26» червня 2025 р. Протокол ЕК № 5

Оцінка ЕК 5 (відмінно) / 90 б.

Секретар ЕК _____

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Відділення *комп'ютерних систем* Комісія *КТ та ПІ*

Спеціальність *123 «Комп'ютерна інженерія»*

Освітньо-професійна програма *«Безпека комп'ютерних систем і мереж»*

ЗАТВЕРДЖУЮ:

Заст. дир. з НВР *Беркань І.В.*

"19" 05 2025 р.

ЗАВДАННЯ

на дипломний проєкт

Здобувачеві освіти *Клеменчук Юлії Сергіївни*

(прізвище, ім'я, по батькові)

1. Тема проекту *Розробка додатку для керування файловим сховищем з автентифікацією користувачів в глобальній мережі*

затверджена наказом по коледжу від "14" листопада 2024 р. № 246

2. Термін здачі закінченого проекту

3. Вихідні данні до проекту *1. Реалізувати додаток для керування файлами з авторизацією користувачів за одноразовим кодом. 2 .Забезпечити завантаження, керування та спільний доступ до файлів. 3. Реалізувати інтерфейс користувача для створюваного додатку. 4. Використати фреймворк Next.js ,серверну платформу Appwrite, мови TypeScript, JavaScript. 5. Налаштувати захист(CORS,CSP,HTTP-заголовки). 6. Провести функціональне й безпекове тестування додатку із застосуванням інструменту OWASP ZAP.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які необхідно розробити)

Аналіз предметної області; Наявні рішення-аналоги на ринку; Огляд обраних технологій; Аналіз технічного завдання; Проєктування інтерфейсу користувача; Архітектурне проєктування сховища; Налаштування середовища; Реалізації серверної частини; Реалізація клієнтської частини; Розгортання додатку; Функціональний огляд основних сторінок; Тестування безпеки додатку з OWASP ZAP; Економічний розрахунок; Аспекти охорони праці

5. Перелік графічного (презентаційного) матеріалу (з точним зазначенням обов'язкових креслень, кількості слайдів)

Презентація PowerPoint – 11 слайдів;

Етапи виконання розробки;Узагальнена блок-схема архітектури безсерверного додатку з використанням BaaS; Структурна організація директорій і файлів додатка; Процесс розробки додатка; Фрагменти коду автентифікації; Блок-схема структури бази даних додатку; Алгоритм входу та реєстрації користувача з підтвердженням через одноразовий код (OTP); Скріншоти реалізованого додатку.

6. Консультанти по проєкту, із зазначенням розділів проєкту, що їх стосується

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Основний розділ	*Гаджиєв М.М.*		
Економічний розділ	*Канський М.Ю.*		
Розділ охорони праці	*Чорновол Н.І.*		
Нормоконтроль	*Петрашова В.І.*		
Старший консультант	*Кривченко Ю.В.*		

7. Дата видачі завдання 29.04.2025

Керівник *Гаджиєв М.М.* _____
(підпис)

Завдання прийняв до виконання *Клеменчук Ю.С.* _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/р	Назва етапів дипломного проєкту	Термін виконання етапів дипломного проєкту (роботи)	Відмітка про виконання
1	*Вступ. Постановка задачі проєктування*	29.04	виконано
2	*Аналіз технічного завдання та пошук літератури*	09.05	виконано
3	*Проєктування дизайну додатку*	10.05	виконано
4	*Налаштування середовища розробки*	11.05	виконано
5	*Проєктування архітектури додатку й моделі сховища*	15.05	виконано
6	*Розроблення серверної частини*	19.05	виконано
7	*Розроблення клієнтської частини*	24.05	виконано
8	*Інтеграція функцій завантаження, перегляду,*	29.05	виконано
	пошуку, перейменування й поширення файлів		виконано
9	*Тестування функціоналу та безпеки*	01.06	виконано
10	*Виконання економічних розрахунків*	03.06	виконано
11	*Розробка питань з охорони праці та техніки безпеки*	04.06	виконано
12	*Підготовка мультимедійної презентації проекту*	11.06	виконано
13	*Оформлення пояснювальної записки*	12.06	виконано
14	*Підготовка доповіді для захисту*	16.06	виконано

Дипломник _____
(підпис)

Керівник _____
(підпис)

Формат	Зона	Поз.	Позначення	Назва	Кіл.	Примітка
				Документація		
			КБ 02. 06 000. 00 ДП	Дипломний проєкт		
А4			КБ 02. 06 000. 00 ДП ПЗ	Пояснювальна записка	1	

Зм.	Арк.	№ докум.	Підпис	Дата	КБ 02. 06 000. 00 ДП		
Розробив		Клеменчук Ю.С.		16.06	Розробка додатку для керування файловим сховищем з автентифікацією користувачів в глобальній мережі	Літ.	Аркуш 4 / Аркушів 86
Перевірив		Гаджиєв М.М.				Н Д П	
Н. контр.		Петрашова В.І.		16.06		ВСП "ОТФК ОНТУ" гр.4КБ-02	
Затверд.		Кривченко Ю.В.		16.06			

ЗМІСТ

ВСТУП

У дипломному проєкті розглядається розробка вебдодатку «Storo» для хмарного зберігання й керування цифровими файлами. Актуальність теми зумовлена стрімким зростанням обсягів даних, що генерується користувачами, та необхідністю забезпечити безпечний, швидкий і зручний доступ до інформації з будь-якого пристрою. Сучасні сервіси зберігання даних повинні гарантувати не тільки надійність збереження й автоматичну синхронізацію, а й гнучке управління правами доступу, масштабованість та мінімальний час очікування навіть за великого навантаження.

В умовах високої конкуренції на ринку хмарних рішень вебдодаток «Storo» орієнтований на поєднання інтуїтивно зрозумілого інтерфейсу та потужної серверної логіки без необхідності розгортання власної інфраструктури. Система забезпечує автентифікацію через одноразовий пароль (OTP), структуроване зберігання файлів різних форматів (зображення, документи, аудіо, відео) та швидкий пошук за метаданими. Мета дипломного проєкту — створення повноцінного вебдодатку Storo із чітко окресленим функціоналом, високими показниками продуктивності та багаторівневими механізмами захисту користувацьких даних. Для досягнення цієї мети необхідно провести аналіз існуючих хмарних платформ із виявленням їхніх сильних і слабких сторін. Обґрунтувати вибір стеку технологій із урахуванням вимог до безсерверної архітектури та BaaS-моделі. Реалізувати модулі автентифікації, завантаження й обробки файлів із підтримкою мініатюр і метаданих. Налаштувати розгортання на платформі. Провести тестування продуктивності, безпеки.

У межах проєкту використано такі технології: Next.js (App Router) із підтримкою серверного та клієнтського рендерингу; TypeScript для суворої типізації фронтенду й серверних функцій; Tailwind CSS та ShadCN UI для швидкої розробки адаптивного інтерфейсу; Appwrite як платформа BaaS для автентифікації, бази даних хмарного сховища файлів; Vercel для безсерверного хостингу.

1 ОСНОВНИЙ РОЗДІЛ

1.1 Аналіз предметної області та обґрунтування вибору технологій

В умовах активного розвитку інформаційних технологій та цифровізації суспільства хмарні сервіси стали важливим компонентом для зберігання, обміну й управління цифровими даними. Завдяки постійному доступу до інформації, автоматичній синхронізації між пристроями, резервному копіюванню та зручним функціям спільної роботи, такі сервіси значно підвищують ефективність взаємодії з інформацією. Проте, через різноманітність існуючих рішень вибір конкретної платформи для реалізації задач зберігання й управління даними є складним завданням. Це пов'язано з істотними відмінностями платформ у питаннях архітектури, політики безпеки, вартості, а також рівня інтеграції з іншими системами та сервісами.

Одночасно з активним зростанням хмарних технологій спостерігається стрімкий розвиток інструментів для веб-розробки, які суттєво спрощують і прискорюють процес створення веб-додатків. Серед них можна виділити фреймворки з підтримкою серверного рендерингу, системи стилізації інтерфейсу, а також сервіси типу Backend-as-a-Service (BaaS), які дозволяють значно скоротити час на створення серверної частини додатку. Тому перед початком роботи над власним веб-додатком важливо систематизувати знання про сучасні технології хмарних сховищ, здійснити детальний аналіз сильних та слабких сторін популярних платформ і обґрунтувати вибір тих технологій, які найкраще відповідають поставленим цілям і вимогам проєкту.

1.1.1 Аналіз предметної області

Хмарні сховища даних (англ. cloud storage) – це модель зберігання цифрової інформації, за якої дані розміщуються на віддалених серверах і доступні користувачам через інтернет. Провайдер хмарного сховища забезпечує інфраструктуру, адміністрування та захист даних, тоді як користувачі можуть зберігати й отримувати файли без потреби утримувати власні фізичні носії.

Архітектура хмарних сховищ базується на розподілених серверних ресурсах та високому рівні віртуалізації. Це означає, що дані розподіляються між багатьма серверами (інколи розташованими в різних дата-центрах і регіонах), але система виглядає для користувача як єдине ціле. Такий підхід забезпечує масштабованість на вимогу та багатокористувацький режим. Завдяки дублюванню та резервуванню інформації у різних вузлах хмарне сховище досягає високої відмовостійкості та надійності: навіть у разі виходу з ладу окремих серверів дані залишаються доступними. Обслуговування інфраструктури покладається на провайдера, що знімає ці задачі з кінцевих користувачів. На рис. 1.1 зображена стандартна архітектура хмарного сховища.

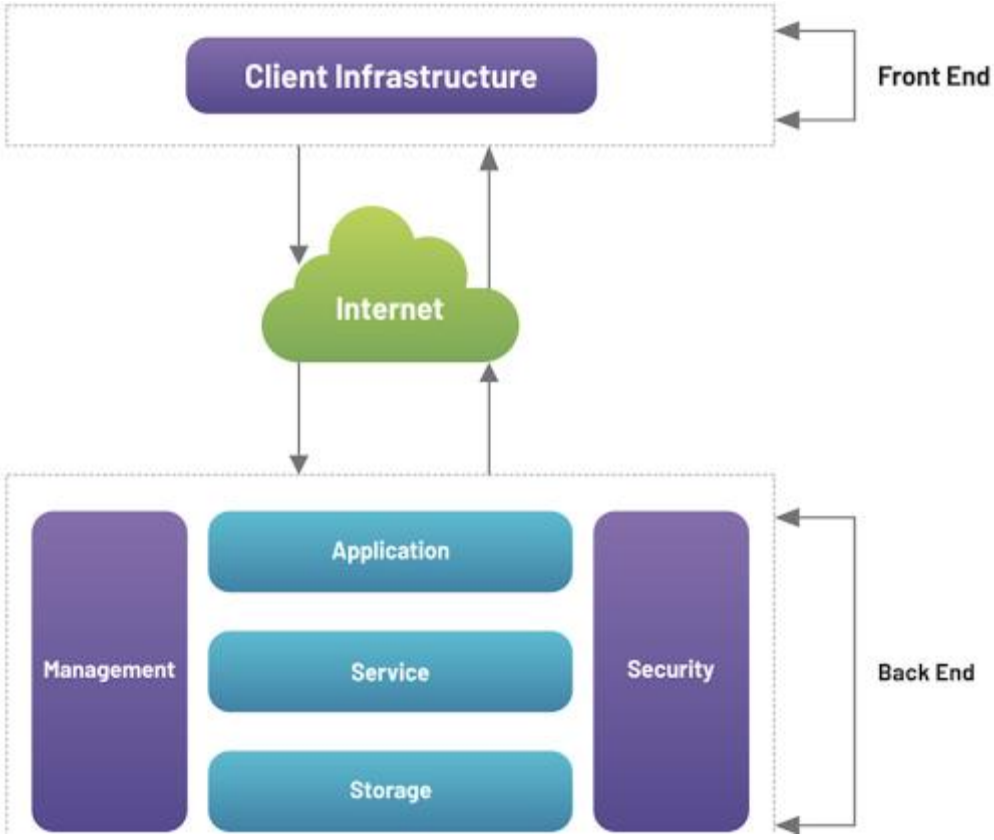


Рис. 1.1 Архітектура стандартного хмарного сховища

Існують різні типи хмарних сховищ, які можна класифікувати за архітектурою та способом впровадження. З погляду розгортання розрізняють публічні, приватні та гібридні хмари. Публічні хмарні сховища надаються сторонніми компаніїями (наприклад, Google Drive, Dropbox) і доступні широкому загалу через інтернет.

Приватні (корпоративні) сховища розгортаються для конкретної організації

– або у власному дата-центрі, або на орендованих серверах – і обслуговують лише внутрішні потреби цієї організації з підвищеним контролем доступу та налаштуваннями безпеки. Гібридні рішення поєднують обидва підходи, дозволяючи частині даних зберігатися в публічній хмарі, а частині – в приватній, щоб збалансувати гнучкість та вимоги до безпеки або відповідності регуляторним нормам. З точки зору архітектури зберігання даних розрізняють об’єктні, блочні та файлові хмарні сховища, а також сховища у вигляді баз даних для структурованої інформації. Об’єктне сховище оперує даними як неструктурованими об’єктами в єдиному адресному просторі, без ієрархії папок – це спрощує масштабування до дуже великих обсягів даних і часто використовується для зберігання резервних копій, медіафайлів, архівів (приклад – Amazon S3).

Блочне сховище надає дисковий простір у вигляді віртуальних томів для серверів; такий підхід застосовується, коли потрібна низька затримка доступу та можливість монтувати віддалений диск, він характерний для корпоративних систем зберігання (наприклад, хмарні диски для віртуальних машин).

Файлове сховище забезпечує зберігання даних у традиційному вигляді файлової системи, каталоги і файли, з доступом через мережеві протоколисуті. Власне це аналог мережевого диска, реалізований на хмарній інфраструктурі. Окремо можна виділити хмарні сховища баз даних, оптимізовані для зберігання та запиту структурованих даних реляційних або NoSQL, через мову запитів – це хмарні СУБД (системи управління базами даних) як служба.

Хмарні сховища пропонують широкий набір функціональних можливостей та переваг для користувачів. Однією з ключових властивостей є доступність даних у будь-який час з будь-якого місця: користувач може отримати файл через вебінтерфейс або клієнтську програму на різних пристроях (персональний комп’ютер, ноутбук, смартфон), маючи підключення до інтернету. При цьому більшість хмарних сервісів підтримують синхронізацію даних між пристроями: зміни у файлі або додавання нового файлу на одному пристрої автоматично розповсюджуються на всі інші пристрої користувача. Це реалізується через фонові клієнтські додатки, що відстежують зміни у визначених папках і завантажують,

вивантажують дані на сервер. Іншою важливою можливістю є спільний доступ до файлів і колаборація: користувач може надати доступ до своїх даних іншим людям – для перегляду або редагування. Деякі хмарні сховища інтегрують інструменти для спільної роботи, наприклад, онлайн-редагування документів декількома користувачами одночасно, коментування, відстеження змін тощо. Більшість сервісів підтримують файли з версіями – зберігання попередніх версій і можливість відновлення випадково видалених або змінених даних. Значна увага приділяється безпеці та конфіденційності: провайдери застосовують шифрування даних при передачі (SSL/TLS) і в стані спокою на серверах, автентифікацію користувачів (паролі, двофакторна автентифікація), а для бізнес-клієнтів - розширені засоби керування доступом, журнали аудиту, відповідність стандартам безпеки.

Таблиця 1.1. Порівняння хмарного та традиційного сховищ

Параметр	**Хмарне сховище**	**Традиційне сховище**
Продуктивність	Вища за рахунок використання масштабованих баз (NoSQL)	Трохи нижча, оскільки операції залежать від локальних дисків
Обслуговування	Просте - за підтримку відповідає провайдер	Вимагає власного адміністрування й інструментів обслуговування
Надійність	Висока - автоматичне резервування та відновлення	Менш надійне без значних початкових вкладень
Спільний доступ	Динамічний - файли можна надавати будь-де, де є мережа	Фізичний - потрібно передавати носії або налаштовувати мережу
Час доступу	Залежить від швидкості Інтернет-з'єднання	Швидкий, оскільки дані на локальному накопичувачі
Безпека	Висока - інтеграція з сучасними системами захисту	Залежить від локальних заходів, ризик вірусів і шкідливого ПЗ
Приклади	Amazon Drive, Dropbox, AutoSync	HDD, SSD, USB-флешки

1.1.2 Наявні рішення-аналоги на ринку

На ринку представлена велика кількість хмарних сервісів для особистого та бізнес-застосування. Серед найпоширеніших платформ для зберігання файлів вирізняються Google Drive, Dropbox та Apple iCloud. Кожен із них пропонує базовий функціонал розміщення інформації у хмарі, але має свої особливості в реалізації інтерфейсу, способах синхронізації, рівнях захисту даних та ціновій політиці. Розглянемо ключові характеристики цих рішень.

Google Drive – це хмарний сервіс для зберігання, створений компанією Google, який інтегровано в екосистему Google Workspace. Користувачам доступний інтуїтивний вебінтерфейс та окремі програми для Windows, macOS, Android і iOS. Особливістю Google Drive є тісна взаємодія з іншими сервісами Google: можна безпосередньо в браузері створювати чи редагувати документи, електронні таблиці, презентації, а також зберігати фотографії через Google Фото. Синхронізація файлів здійснюється через застосунок "Google Drive for Desktop", що дозволяє або дублювати файли на пристрої користувача, або працювати в режимі доступу на вимогу, файли з’являються на комп’ютері лише після відкриття. Усі зміни миттєво синхронізуються з хмарою при наявності інтернет-з'єднання. Мобільний додаток Google Drive також автоматично створює резервні копії фотографій та іншої інформації.

У сфері безпеки Google Drive застосовує шифрування як під час передачі, так і під час зберігання даних на серверах; авторизація відбувається через Google-акаунт з можливістю ввімкнення двофакторної автентифікації. Користувачі корпоративних тарифних планів отримують додаткові опції –управління правами доступу, журнал подій, налаштування політик збереження інформації. Відповідно до політики конфіденційності Google, дані можуть аналізуватися автоматизованими системами для захисту від шкідливого контенту чи для надання додаткових функцій, наприклад, пошуку по файлах. За замовчуванням Google Drive пропонує 15 ГБ безкоштовного простору, який розподіляється між сервісами Диск, Gmail та Фото. Для користувачів, яким потрібно більше місця, доступні платні підписки Google One (100 ГБ, 200 ГБ, 2 ТБ тощо) з щомісячною або річною оплатою. Таким чином, Google Drive приваблює широку аудиторію завдяки інтеграції з іншими сервісами компанії та суттєвому обсягу безкоштовного сховища, хоча існує залежність від екосистеми Google і окремі питання щодо приватності.

Dropbox – один із перших хмарних технологій синхронізації файлів, який з'явився у 2007 році та став стандартом у цій галузі. Dropbox пропонує універсальні програми для настільних операційних систем (Windows, macOS, Linux) і мобільних

пристроїв, а також надає доступ через вебінтерфейс. Його інтерфейс максимально наближений до класичної файлової системи: користувач взаємодіє з папками та файлами, подібно до роботи на локальному диску. Після встановлення спеціальної папки Dropbox на комп'ютері всі файли, розміщені в ній, автоматично синхронізуються з хмарним сховищем і всіма підключеними пристроями.

Однією з головних переваг Dropbox є ефективний механізм синхронізації: використовується блокова (диференційна) передача даних, коли передаються тільки змінені частини файлів, що особливо актуально для великих обсягів інформації. Для спільної роботи Dropbox дозволяє організовувати загальний доступ до окремих папок або файлів, визначати права на редагування чи перегляд, а також залишати коментарі. Сервіс інтегрується з різними сторонніми інструментами, зокрема підтримує онлайн-редагування документів Microsoft Office (через Microsoft 365) і містить власний редактор нотаток "Dropbox Paper". Безпека реалізується шляхом шифрування даних (AES-256 для зберігання, TLS для передачі), а також доступна двофакторна аутентифікація. Для корпоративних клієнтів надаються додаткові функції: віддалене видалення даних з пристрою у випадку втрати, контроль над дозволами учасників і ведення журналу дій. Колись сервіс стикався з інцидентами витоку даних, після чого безпекові заходи були суттєво посилені. Безкоштовний тариф Dropbox пропонує лише 2 ГБ простору, що набагато менше, ніж у конкурентів. Розширити обсяг сховища можна через оформлення платної підписки: для індивідуальних користувачів – плани на 2 ТБ і більше, для бізнес-команд – варіанти з динамічним обсягом з додатковими адміністративними функціями.

Apple iCloud – це хмарна платформа, тісно інтегрована з пристроями Apple, орієнтована насамперед на власників iOS та macOS. Компонент iCloud Drive забезпечує прямий доступ до файлів із системних застосунків Apple: на iPhone чи iPad через додаток "Files", а на Mac – через папку iCloud Drive у Finder. Основна ідея інтерфейсу iCloud – прозора інтеграція хмарного сховища в повсякденний досвід користувача. Документи, які створені у пакеті iWork (Pages, Numbers, Keynote), автоматично зберігаються в iCloud і синхронізуються між усіма

пристроями користувача. Синхронізація в iCloud працює переважно у фоновому режимі: пристрої автоматично завантажують у хмару фотографії, резервні копії, нотатки та іншу інформацію (за умови увімкнення відповідної опції). iCloud Drive підтримує зберігання файлів різних типів і створення власної ієрархії папок; усі зміни миттєво з'являються на всіх пристроях з єдиним Apple ID. Для Windows також доступний окремий клієнт iCloud, який дозволяє синхронізувати фото, закладки й файли з ПК.

Apple значну увагу приділяє захисту даних: інформація шифрується як під час передачі, так і під час зберігання, а для доступу до iCloud обов'язково використовується двофакторна аутентифікація (як правило, увімкнена за замовчуванням). Окремі типи даних (наприклад, паролі в iCloud Keychain чи медичні записи) захищені наскрізним шифруванням, а з 2022 року з'явилася опція Advanced Data Protection, яка поширює наскрізне шифрування на більшість категорій файлів, включно з резервними копіями та iCloud Drive (у такому режимі навіть Apple не має доступу до ключів шифрування користувача). Для забезпечення відновлення даних у разі потреби, за замовчуванням файли iCloud Drive шифруються на сервері, а ключі зберігаються у Apple. Обсяг безкоштовного простору складає 5 ГБ на кожен Apple ID, що зазвичай швидко вичерпується, оскільки використовується для резервних копій пристроїв, фото і документів. Для збільшення сховища можна підключити платну підписку iCloud+ (50 ГБ, 200 ГБ або 2 ТБ).

Таблиця 1.2. Порівняльний аналіз інтерфейсу та безпеки

Показник	**Google Drive**	**Dropbox**	**iCloud**
UX/UI (веб)	Material Design; бічна навігація; семантичний пошук	Картковий вигляд; фільтри; швидкий попередній перегляд	Мінімалізм iOS; базовий файловий менеджер
UX/UI (десктоп)	Локальні папки у файловій системі; офлайн-режим	Smart Sync; відображення без збереження	Інтеграція з Finder; Drag & Drop
Мобільний інтерфейс	Android/iOS: голосовий пошук; автоматична архівація	Android/iOS: автоматичний формат фото; перегляд документів	iOS Files: жести; інтеграція з камерами
Шифрування (спокій)	AES-256 (серверний)	AES-256	AES-128/256 (енд-ту-енд для чутливих даних)

Продовження таблиці 1.2

Шифрування (транспорт)	TLS 1.2/1.3	TLS 1.2	TLS
End-to-End шифрування	немає вбудованого; сторонні плагіни	недоступне	є для ключових даних (Keychain, приватні документи)
Двофакторна автентифікація	Google 2FA	Через дод. параметри; SMS/додаток	Обов'язкова Apple 2FA

1.1.3 Огляд обраних технологій для реалізації веб-додатку

Для реалізації вебдодатку в межах цього проєкту було підібрано сучасний стек технологій, до якого входять фреймворк Next.js, мова програмування TypeScript, система стилізації Tailwind CSS, бекендова хмарна платформа Appwrite та бібліотека UI-компонентів ShadCN UI. Вибір цих інструментів обумовлений їхніми функціональними перевагами, простотою розробки й відповідністю технічним вимогам. Далі подано огляд кожної технології та обґрунтування їх використання з коротким аналізом альтернатив.

Next.js – це система з відкритим вихідним кодом, яка ґрунтується на JavaScript/TypeScript і бібліотеці React, призначена для створення сучасних веб-додатків. До переваг Next.js належать вбудована маршрутизація, механізми попереднього завантаження сторінок, підтримка API-роутів для реалізації бекенд-логіки, а також інтеграція з хмарними середовищами для розгортання. У рамках цього проєкту Next.js виступає базою для клієнтської частини, оскільки забезпечує високу швидкість роботи інтерфейсу й дає змогу ефективно відображати частину контенту на сервері.

TypeScript, мова програмування зі статичною типізацією, яка компілюється у стандартний JavaScript. У даному проєкті TypeScript обрано як основний інструмент для розробки frontend і, за потреби, для серверної частини, оскільки він забезпечує надійність кодової бази, мінімізує кількість помилок і спрощує підтримку в перспективі. Статична типізація дає змогу виявляти дефекти на етапі компіляції, що особливо цінно у великих застосунках. Переваги TypeScript – автодоповнення коду, зручний рефакторинг, полегшена навігація завдяки чітко описаним типам для змінних, функцій та об'єктів. Для проєктів на React TypeScript

став галузевим стандартом, оскільки дозволяє явно вказувати prop-и компонентів і запобігати типових помилок. В якості альтернативи можна було б застосувати JavaScript, що спростило б написання простих сценаріїв, але ускладнило б підтримку й масштабування проєкту в майбутньому. Розглядалися також інші мови, але TypeScript оптимально інтегрується з обраним стеком, зокрема завдяки повній підтримці у Next.js. Застосування TypeScript забезпечує підвищену надійність, структурованість і якість розробки, що особливо важливо для складних або великих рішень.

Tailwind CSS є утилітарним CSS-фреймворк для стилізації інтерфейсу. Замість готових шаблонних стилів чи компонентів як у Bootstrap, Tailwind надає велику кількість дрібних класів, кожен з яких відповідає окремому стилістичному. Розробник складає бажаний вигляд інтерфейсу безпосередньо в розмітці, комбінуючи ці класи, що суттєво прискорює процес верстки й забезпечує гнучкість у створенні дизайну. У цьому проєкті Tailwind CSS використовується для оформлення UI завдяки можливості швидко створювати адаптивні інтерфейси та забезпечувати цілісність стилю. Tailwind підтримує гнучку налаштування дизайну через конфігураційний файл, дозволяє оптимізувати розмір CSS за рахунок видалення невикористаних класів під час збірки, і містить інструменти для адаптивної верстки. На відміну від Bootstrap, Tailwind не накладає жодних обмежень на вигляд елементів, що особливо важливо для унікальних або креативних інтерфейсів. Порівняно із звичайним CSS чи SCSS, Tailwind дає змогу уникнути дублювання стилів і оперативно вносити зміни без необхідності редагувати окремі CSS-файли. Альтернативними підходами можуть бути CSS-in-JS рішення (Styled Components, Emotion) або використання модульних CSS, однак Tailwind вигідно вирізняється швидкістю розробки і зручністю підтримки коду.

Appwrite – це сучасна хмарна бекендова платформа з відкритим кодом (Backend-as-a-Service, BaaS), розроблена для спрощення створення серверної логіки у веб- і мобільних застосунках. Appwrite містить набір готових серверних API і сервісів, що включає автентифікацію користувачів, базу даних, файлове сховище, функції serverless, системи повідомлень та інші важливі інструменти. У

межах цього проєкту Appwrite застосовується для реалізації всіх ключових завдань бекенду: зберігання та обробки файлів, авторизації, керування користувачами й іншої бізнес-логіки. Головна перевага цієї платформи – можливість швидко розгорнути повноцінний сервер без необхідності писати власний бекенд із нуля: достатньо встановити Appwrite на сервер чи у хмарі та налаштувати потрібні модулі. Замість самостійної розробки REST API для реєстрації користувачів, логіну, відновлення пароля або управління файлами, можна використовувати готові API Appwrite, що суттєво економить час і зменшує ризик помилок. Серед альтернатив – комерційні сервіси типу Firebase (Google), AWS Amplify або розробка серверної частини на основі Node.js + Express. Порівняно з Firebase, Appwrite є open-source рішенням, яке можна запускати локально чи на власному сервері для повного контролю над даними, що особливо важливо з позиції конфіденційності.

ShadCN UI – це бібліотека готових інтерфейсних компонентів для React та Next.js, яка повністю сумісна з Tailwind CSS. Вона надає добірку сучасних і продуманих UI-елементів – кнопок, форм, модальних вікон, випадаючих списків, сповіщень тощо, побудованих на основі бібліотеки Radix UI. Головна особливість ShadCN UI полягає у тому, що компоненти імпортуються у вигляді відкритого коду, який можна редагувати й налаштовувати. Застосування ShadCN UI суттєво прискорює створення інтерфейсу: більшість поширених елементів доступні у готовому вигляді, і їх не потрібно розробляти з нуля. Альтернативою можуть бути популярні фреймворки типу Material-UI, Ant Design чи Chakra UI.

1.2 Проєктування вебдодатку

1.2.1 Аналіз технічного завдання на розробку

Розробка вебдодатку розпочалася з чіткого визначення функціональних можливостей і критеріїв якості, яким має відповідати система. На першому етапі основна увага приділялася впровадженню надійної авторизації користувача через одноразовий код (OTP). Такий підхід дає змогу відмовитися від паролів, що суттєво знижує ризик несанкціонованого доступу до облікового запису.

Після проходження автентифікації користувач отримує можливість завантажувати різні типи файлів: зображення, текстові документи, аудіо- та відеоматеріали. Всього на одного користувача надається 2 ГБ, максимальний розмір одного файла для завантажування дорівнює 50 МБ. Вся інформація зберігається у впорядкованій структурі папок. Для кожного файлу доступні ключові метадані – назва, розмір, дата додавання, а також мініатюра для швидкого перегляду та пошуку потрібного контенту.

Програмна платформа розрахована на стабільну роботу при великій кількості одночасних користувачів – система підтримує сотні активних сесій без зниження швидкості реакції. Обробка запитів до API виконується з мінімальною затримкою, зазвичай не перевищуючи двох секунд. Для оптимізації швидкості завантаження файлів впроваджено мережу доставки контенту (CDN), яка розподіляє навантаження між різними вузлами і зменшує час очікування при доступі до ресурсів.

Безпека даних забезпечується багаторівневими механізмами захисту: шифрування інформації на сервері за допомогою алгоритму AES-256, захищені канали зв’язку через SSL/TLS під час передачі даних, налаштування політик доступу до API, зокрема CORS для запобігання несанкціонованим запитам зі сторонніх ресурсів. Крім того, регулярно створюються резервні копії бази даних і файлів, що дає змогу швидко відновити роботу сервісу навіть у разі непередбачуваних технічних проблем.

З технічної точки зору клієнтська частина розроблена із застосуванням Next.js, що дозволяє реалізувати серверний рендеринг і автоматично генерувати статичні сторінки. Це гарантує коректне відображення інтерфейсу на різних типах пристроїв.

Серверна логіка базується на платформі Appwrite, яка надає модулі для аутентифікації, керування базою даних і файлового сховища. Розгортання оновленого коду виконується автоматично через інтеграцію з GitHub та хостингом Vercel, що додатково дозволяє використовувати вбудований CDN для прискорення доступу до ресурсу.

1.2.2 Проєктування інтерфейсу користувача (UI/UX)

При розробці вебдодатку важливим етапом є проєктування його дизайну. Почати можна з аналізу прототипів відомих хмарних сервісів (Google Drive, Dropbox, OneDrive). У кожному з них було видокремлено вдалі рішення – компактну верхню панель навігації, сітку прев'ю для графічних файлів і спрощений потік завантаження – та неефективні підходи, наприклад перевантажені бічні меню та надмірну кількість прихованих команд.

Головна сторінка вебдодатку сформована за модульною схемою, яка об'єднує хедер, ліву навігаційну колонку, центральний інформаційний блок і праву сервісну панель. Така організація забезпечує чітку ієрархію елементів та мінімальне когнітивне навантаження на користувача.

У верхньому полі центрується поле пошуку. Праворуч від поля розміщено кнопку завантаження файлів та піктограму завершення сесії. Концентрація основних команд у межах однієї горизонтальної смуги скорочує час, необхідний для виконання ключових дій.

Ліва вертикальна колонка містить перелік основних розділів Головна, Документи, Зображення, Медіа, Інше. У нижній частині колонки розташовано картку профілю користувача з аватаром та електронною адресою, а також тематичну ілюстрацію.

Центральний інформаційний блок відкриває панель моніторингу місця сховища. Всередині віджета інтегровано текстовий індикатор доступного простору. Нижче формується сітка карток оперативної статистики за категоріями, кожна картка містить іконку, сумарний обсяг і дату останньої дії. Така послідовність загальний показник а далі деталізація забезпечує логічний маршрут перегляду даних.

Праворуч розміщено сервісну панель «Нещодавно завантажені файли». Елементи представлено вертикальним списком, що складається з іконок типів, назв, дат та прихованих контекстних меню дій. На рис. 1.1 зображено інтерфейс головної сторінки Storo.

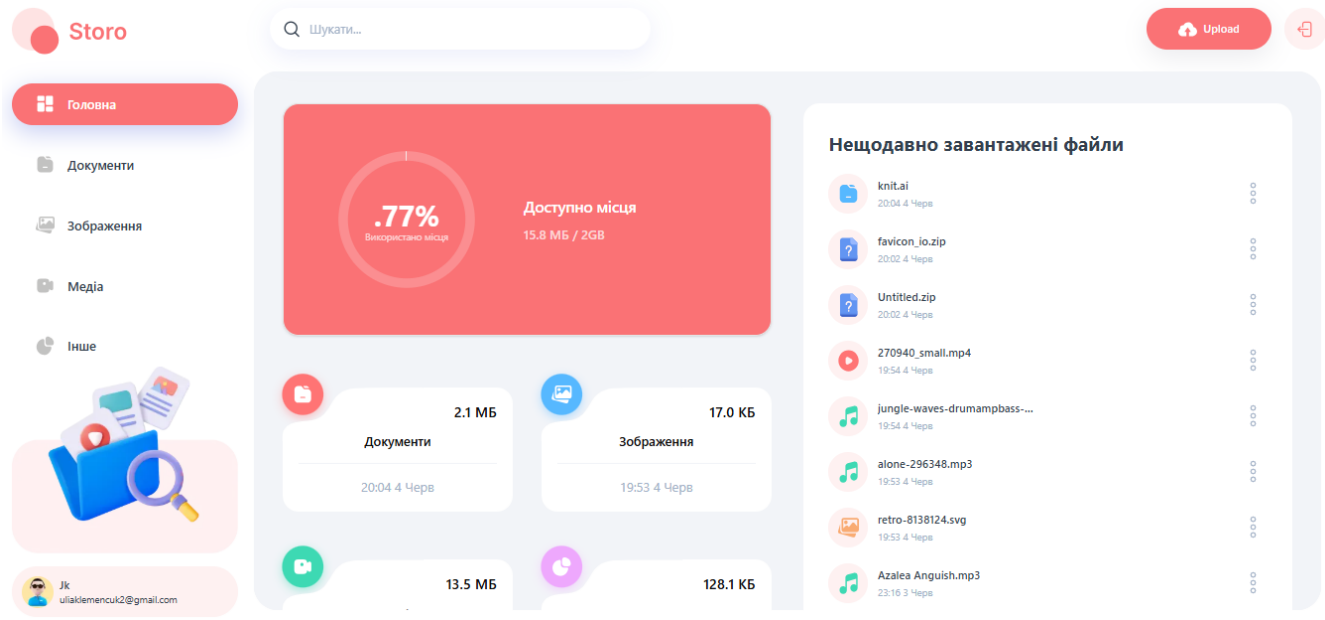


Рисунок 1.2 Головна сторінка вебдодатку Storo

1.2.3 Архітектурне проєктування сховища

Додаток реалізовано на основі безсерверної архітектури (serverless-архітектури), що вказує на Next.js Server Actions – функцій, які виконуються на сервері за запитами клієнта. Фронтенд створено з використанням структури Next.js, який відповідає за відображення інтерфейсу користувача, маршрутизацію та частину клієнтської логіки. Всі серверні обчислення та збереження даних виконуються як serverless-функції в хмарі, що дозволяє автоматично масштабувати ресурси відповідно до навантаження та знижує витрати на підтримку постійно увімкнених серверів.

Backend-функціональність делеговано хмарному сервісу Appwrite як BaaS, що інтегрується через serverless-точки входу. Appwrite надає готові модулі автентифікації, управління користувачами, бази даних і сховища файлів у вигляді викликів API без необхідності власноруч налаштовувати серверну інфраструктуру.

Взаємодія між Next.js і Appwrite відбувається через захищені HTTPS-запити до serverless-функцій. У фронтенд-додатку використовується офіційний Appwrite SDK для Node.js, який являється набором програмних бібліотек, які спрощують ініціалізацію клієнта, створення сессій, завантаження файлів та виконання CRUD-операцій. Конфігураційні змінні середовища, забезпечують безпечне встановлення

з’єднання з Appwrite без прямого зберігання секретів у коді. На рис.1.3 представлена загальна архітектура додатку.

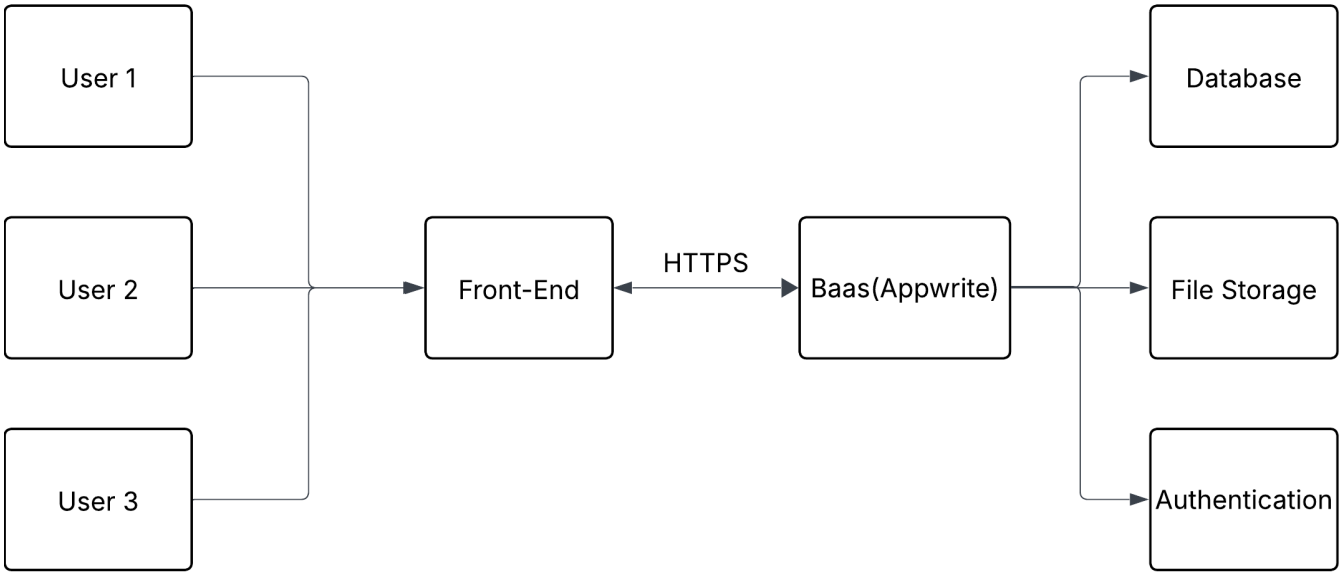


Рисунок 1.3 Блок-схема загальної архітектури додатку

Фізична реалізація обраної архітектури в рамках розробки застосунку чітко структурована у вигляді впорядкованої файлової системи. Проєкт побудований за принципами модульності, що передбачає поділ функціональності на окремі незалежні блоки, кожен з яких відповідає за певний аспект роботи програми — від інтерфейсу користувача до обробки даних і взаємодії з бекендом. Уся бізнес-логіка винесена у спеціальні файли дій (actions), які виконуються на сервері, а компоненти інтерфейсу зосереджені у директорії components, що забезпечує чітке розмежування відповідальностей.

Типізація, реалізована за допомогою TypeScript, підвищує надійність коду та полегшує його супровід. Конфігураційні файли, такі як .env, appwrite.config.ts та tailwind.config.ts, відповідають за параметри середовища виконання, підключення до API, стилізацію та забезпечення захисту чутливих даних через змінні середовища.

Таке структурне впорядкування дозволяє легко масштабувати застосунок, додавати нові функції без ризику порушення існуючої логіки, а також швидко адаптувати кодову базу під різні умови розгортання — локально, на staging-сервері чи у продакшні.

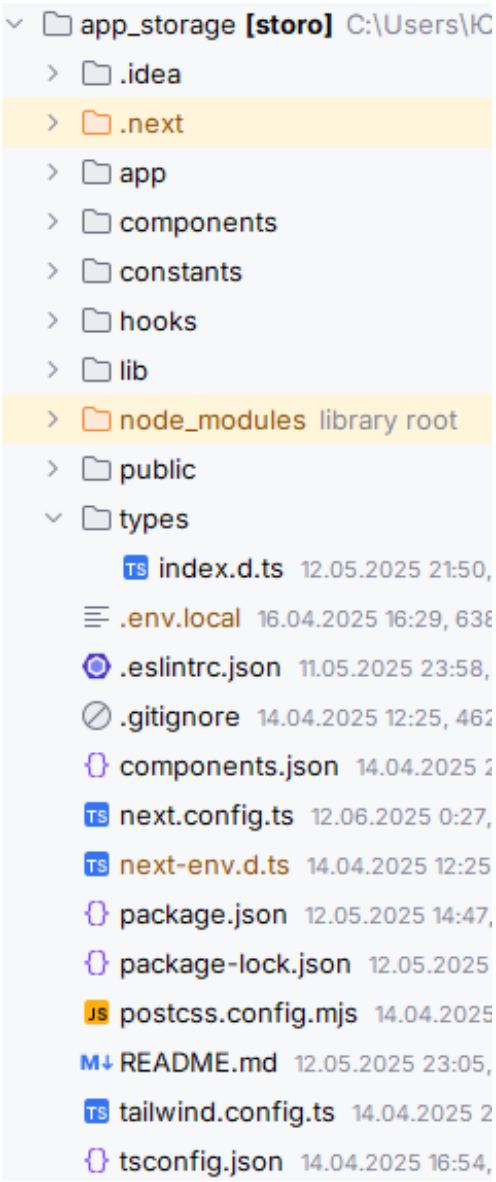


Рисунок 1.4 Структурна організація директорій і файлів вебдодатка

Розглянуто детальну ієрархію директорій, конфігураційних файлів і службових модулів, які формують архітектуру та функціональну логіку вебдодатка Storo.

1. Директорія та файли в корені проєкту:

.eslintrc.json, *.gitignore*, *README.md*, *package.json*, *package-lock.json*, *tsconfig.json*, *next.config.ts*, *tailwind.config.ts*, *postcss.config.mjs* – конфігураційні файли проєкту;

2. Директорія .idea:

Сховище конфігурацій IDE (файли робочого простору, налаштування модулів). Не впливає на роботу самого додатка, використовується для локальних

налаштувань редактора;

3. Директорія app:

layout.tsx та *globals.css* – глобальні компоненти (загальна структура сторінок, стилі, що підключаються на всіх маршрутах);

(auth) – піддиректорія з компонентами та сторінками, пов'язаними з аутентифікацією: сторінка входу, реєстрації, компоненти OTP-моделі;

(root) – піддиректорія з головними маршрутами вебдодатку: Dashboard, File Manager, Upload, Profile;

favicon.ico, fonts - шрифтові ресурси та значок для браузера;

4. Директорія components:

Header.tsx - верхня панель із пошуком, кнопкою завантаження та виходу;

Sidebar.tsx та *MobileNavigation.tsx* – компоненти бокової навігації для десктопної і мобільної версій;

AuthForm.tsx, OTPModal.tsx – форма вводу email та OTP;

ActionDropdown.tsx, ActionsModalContent.tsx – контекстне меню дій над файлом;

FileUploader.tsx – компонент завантажування файлів із drag-and-drop;

Card.tsx, Thumbnail.tsx – карточки та мініатюри для відображення файлів на головній сторінці;

Search.tsx, Sort.tsx – інтерфейс пошуку та сортування;

Chart.tsx – компонент для відображення кругової або іншої діаграми статистики;

FormattedDateTime.tsx – утиліта для форматування дат;

ui/ – набір базових UI-елементів (наприклад, кнопки, поля вводу), упакованих у власну бібліотеку;

5. Директорія constants:

index.ts – файл, у якому зберігаються константи проєкту (наприклад, шляхи до маршрутів, ключі локального сховища, постійні значення);

6. Директорія hooks:

use-toast.ts – кастомний компонент для відображення системних сповіщень у

додатку;

7. Директорія lib:

actions/;

user.actions.ts – функції для взаємодії з об’єктом користувача (реєстрація, вхід, отримання профілю);

file.actions.ts – функції для роботи з файлами: отримання списку, завантаження, перейменування, видалення, шаринг;

appwrite/;

config.ts – конфігурація підключення до сервісів Appwrite (endpoint, projectId);

index.ts – ініціалізація клієнта Appwrite і експорт засобів аутентифікації/бази даних/сховища;

utils.ts – допоміжні функції (наприклад, для обробки помилок, валідації даних тощо);

8. Директорія public:

file.svg, globe.svg, next.svg, vercel.svg, window.svg - піктограми та логотипи, які автоматично доступні за статичними URL;

assets/;

icons/ – набір SVG-іконок для кожного типу файлу (документи, зображення, відео, аудіо, інші), а також системні іконки (видалення, пошук, меню тощо);

images/ – растрові зображення (аватар, ілюстрації, загальні зображення, що використовуються у пустому стані файлового менеджера);

9. Директорія types:

index.d.ts – оголошення глобальних типів TypeScript для проєкту (наприклад, інтерфейси для файлів, користувачів, можливих форматів метаданих).

Для впровадження описаної архітектури у програмній частині вебдодатку застосовуються різні синтаксичні конструкції сучасних мов JavaScript і TypeScript. Їх використання сприяє гнучкості, надійності, безпечній типізації та легкому масштабуванню системи. У таблиці 1.3, наведено основні конструкції, що зустрічаються у коді проєкту, разом із коротким поясненням їхнього призначення.

Таблиця 1.3. Синтаксичні конструкції проєкта

Конструкція	Опис
import	Оператор для завантаження експортованих значень чи функцій із зовнішніх модулів у поточний файл.
export / export default	Ключове слово, яке робить змінні, функції або класи публічно доступними для імпорту в інших модулях. export default позначає єдине значення за замовчуванням модуля.
const	Оголошення константи з блочною областю видимості. Після присвоєння первісного значення воно не може бути змінене.
let	Оголошення змінної з блочною областю видимості. На відміну від const, значення змінної можна переназначувати.
function	Оголошення функції (традиційний синтаксис), що дозволяє створювати іменовані або анонімні функції, придатні для багаторазового виклику.
=> (Arrow Function)	Стрілкова функція – скорочений запис для анонімної функції. Має лексичний контекст this і підходить для виразних функцій.
async / await	async позначає асинхронну функцію, яка повертає Promise. await призупиняє виконання цієї функції до завершення проміса, отримуючи його результат.
try / catch / finally	Блок обробки винятків: код у try виконується, а якщо виникає помилка – вона перехоплюється у блоці catch. finally (опційно) виконується після try/catch у будь-якому разі.
if / else if / else	Умовний оператор. Якщо вираз у if істинний, виконується відповідний блок коду. else if дозволяє перевіряти додаткові умови, else – виконати код за замовчуванням, якщо жодна умова не виконалась.
switch / case / default	Конструкція багаторазового вибору. switch порівнює значення виразу з наведеними у case випадками; коли знаходиться співпадіння, виконується код цього case. Блок default виконується, якщо жоден case не спрацював.
return	Оператор повернення значення з функції. Повертає результат і припиняє подальше виконання функції.
throw	Оператор викидання винятку. Використовується для створення помилки, яку може перехопити блок catch.
Тернарний оператор ?:	Коротка форма умовного виразу. Після умови ? йде вираз для істини, після : – для хибності, що дозволяє в одному виразі вибрати одне з двох значень.
Логічні && та ||	Оператори логічного "І" (&&) та "АБО" (||)
Деструктуризація	Синтаксис для розбору масивів або об’єктів: дозволяє легко витягувати їхні значення у змінні через шаблон { ... } або [...].
TypeScript type / interface	Оголошення власного типу (alias) або інтерфейсу в TypeScript. Не присутні в JS-рантаймі, проте визначають структуру та сигнатуру даних на етапі розробки.
Дженерики (<T>)	Узагальнені типи у TypeScript. Дозволяють визначити інтерфейси чи функції з параметрами типу, працюючи з довільними типами даних.
Утвердження типу (as)	Оператор TypeScript, що навмисно приводить значення до вказаного типудозволяючи обійти перевірку типів компілятором.
Директиви \"use client\" / \"use server\"	Спеціальні директиви Next.js. \"use client\" позначає файл як клієнтський компонент (JS виконується в браузері), а \"use server\" – як серверний (код виконується на сервері).

1.3 Реалізація вебдодатку

1.3.1 Налаштування середовища

У процесі реалізації додатку здійснено налаштування середовища розробки на базі фреймворку Next.js з використанням TypeScript. Налаштування Next.js було проведено у файлі конфігурації *next.config.js*, який включає параметри для ігнорування помилок збірки TypeScript і ESLint, експериментальні функції для підтримки дій на сервері з розміром тіла запиту до 50 МБ, а також визначення зовнішніх джерел зображень для оптимізованого завантаження ресурсів.

Окремо створено конфігурацію Tailwind CSS у файлі *tailwind.config.js*, яка охоплює налаштування шляхів до файлів, що містять класи стилів, розширення базових параметрів, таких як палітра кольорів, шрифти, тіні, анімації, а також підключення додаткових плагінів, зокрема tailwindcss-animate.

Для забезпечення інтеграції з хмарним сервісом було створено файли змінних середовища *.env.local*, що містять ключові параметри, такі як URL-адреса API, ідентифікатори проєкту, бази даних, колекцій користувачів та файлів, ідентифікатор кошика для зберігання, а також секретний ключ доступу. На рис.1.4. продемонстрований файл *.env.local*, який включає ключові параметри для взаємодії з Appwrite.

```
export const appwriteConfig = {  Show usages   Klemjl
  endpointUrl: process.env.NEXT_PUBLIC_APPWRITE_ENDPOINT!,
  projectId: process.env.NEXT_PUBLIC_APPWRITE_PROJECT!,
  databaseId: process.env.NEXT_PUBLIC_APPWRITE_DATABASE!,
  usersCollectionId: process.env.NEXT_PUBLIC_APPWRITE_USERS_COLLECTION!,
  filesCollectionId: process.env.NEXT_PUBLIC_APPWRITE_FILES_COLLECTION!,
  bucketId: process.env.NEXT_PUBLIC_APPWRITE_BUCKET!,
  secretKey: process.env.NEXT_APPWRITE_KEY!,
};
```

Рисунок 1.5 Файл .env.local із параметрами конфігурації конфігурації для роботи з Appwrite endpoint, ідентифікатори, секретний ключ

У проєкті функціонують два серверні клієнти Appwrite. Сесійний клієнт (sessionClient) використовується для доступу до захищених ресурсів на стороні клієнта, авторизуючись через cookie сесії. Адміністративний клієнт (adminClient)

із розширеними правами забезпечує керування акаунтами, роботу з базами даних, файловими сховищами та аватарами, автентифікуючись секретним ключем.

1.3.2 Серверна частина

У проєкті Storo реалізована безсерверна архітектура за допомогою сучасних вебтехнологій: на фронтенді використовується Next.js App Router, а на бекенді - сервіс Appwrite. Appwrite забезпечує основну серверну логіку, включаючи автентифікацію користувачів, управління базою даних та файловим сховищем, що дозволяє значно спростити розробку й обслуговування бекенд-частини додатку.

Безсерверна (serverless) архітектура — це підхід до розробки та розгортання програмних додатків, за якого реалізовані серверні компоненти виконуються в ізольованих середовищах (контейнерах), керованих хмарним провайдером. Розробник не опікується безпосереднім управлінням віртуальними машинами чи серверами, а лише пише логіку функцій або конфігурує сервіси.

Backend (серверна частина) – це частина вебдодатка, яка відповідає за обробку даних, логіку роботи додатку, взаємодію з базою даних та автентифікацію користувачів. Користувач її не бачить напряму.

Frontend (клієнтська частина) – це частина вебдодатка, з якою безпосередньо взаємодіє користувач. Вона забезпечує візуалізацію інтерфейсу, взаємодію з користувачем та надсилає запити до backend.

Next.js App Router є файловим маршрутизатором, який підтримує серверні компоненти та серверні функції React. Це означає, що Next.js дозволяє частину коду виконувати на сервері, а частину - на клієнті. У вебдодатку для звернення до бекенду використовується Appwrite SDK через require(node-appwrite)), а комунікація між клієнтом і сервером здійснюється через REST API Appwrite, для отримання інформації про автентифікованого користувача використовується метод «account.get()» Appwrite SDK, що запитує дані користувача з сервера Appwrite. Реалізація метода «account.get» для отримання даних поточного користувача з сервера на рис. 1.5.

```
const result :User<Preferences>  = await account.get();

const user :DocumentList<Document>  = await databases.listDocuments(
  appwriteConfig.databaseId,
  appwriteConfig.usersCollectionId,
  [Query.equal("accountId", result.$id)],
);
```

Рисунок 1.6 Фрагмент коду account.get для отримання даних поточного користувача з сервера

Next.js виступає клієнтом, який динамічно отримує дані і ресурси від Appwrite та інтерпретує інтерфейс відповідно до серверних або клієнтських операцій.

Next.js підтримує два основні режими генерації: серверний рендеринг (SSR) та клієнтський рендеринг (CSR).

Рендеринг (англ. rendering) – це процес формування та відображення візуального представлення даних на екрані. В контексті веброзробки рендерингом називають створення сторінки або компонента, який бачить користувач.

Препроцесування (англ. preprocessing) - це процес попередньої обробки даних перед їх основним використанням, наприклад, для аналізу, навчання моделей машинного навчання або компіляції.

Під час серверного рендерингу (SSR) HTML-розмітка сторінки формується на сервері в момент кожного запиту користувача, що дозволяє відправляти у браузер вже готовий контент. Такий підхід може суттєво скоротити час початкового завантаження сторінки та підвищити продуктивність, особливо для динамічних або захищених сторінок.

При клієнтському рендерингу (CSR) браузер отримує лише оболонку HTML-сторінки з мінімальною розміткою, а подальше формування інтерфейсу здійснюється шляхом виконання JavaScript-коду на стороні клієнта. У цьому випадку структура сторінки DOM (Document Object Model, модель об'єктів документа) створюється та оновлюється динамічно. DOM є стандартом для опису структури, доступу й маніпулювання HTML- та XML-документами. Порівняння

процесу обробки запиту при CSR і SSR у Next.js на рис. 1.6.

Рисунок 1.7 Порівняння процесу обробки запиту при CSR і SSR у Next.js

У вебсховищі Storo, сторінки, що потребують попередньої підготовки або захищені автентифікацією, можуть формуватися на сервері – при цьому на сервері Next.js виконується перевірка сесії та виклик account.get() для отримання даних користувача.

Натомість взаємодія зі списком файлів getFiles(), uploadFile(), зазвичай здійснюється на клієнті через виклики Appwrite SDK. Такий гібридний підхід дозволяє отримати переваги обох стратегій: SSR гарантує безпечну обробку автентифікації та швидке відображення критичного вмісту, а CSR забезпечує динамічне оновлення інтерфейсу при роботі з файлами.

Поняття CRUD (Create, Read, Update, Delete) є базовим для роботи з даними у будь-яких додатках. CRUD – це акронім для операцій створення, зчитування, оновлення і видалення даних. У проєкті, ці операції реалізовані конкретними функціями. Основою роботи з даними є концепція CRUD (Create, Read, Update,

Delete). Кожна з цих операцій реалізована окремою функцією:

Create - завантаження файлу за допомогою uploadFile(), що викликає storage.createFile(bucketId, file) у Appwrite SDK. Функція приймає файл, ідентифікатори власника й акаунта та шлях сторінки, після чого створює адміністративного клієнта Appwrite і дістає доступ до модулів storage та databases. Вона перетворює отриманий файл у формат InputFile, завантажує його до кошика через storage.createFile і формує об'єкт метаданих: визначає тип і розширення за назвою, додає посилання на файл, розмір, власника й порожній масив користувачів. Якщо будь-який етап переривається помилкою, вона передається до handleError. Завантаження файлу за допомогою функції uploadFile() в файлі *file.actions.ts* на рис. 1.8.

```
export const uploadFile : ({ file, ownerId, accountId, path }: Uplo... = async ({
  file,
  ownerId,
  accountId,
  path,
}: UploadFileProps) :Promise<any> => {
  const { storage, databases } = await createAdminClient();

  try {
    const inputFile = InputFile.fromBuffer(file, file.name);

    const bucketFile = await storage.createFile(
      appwriteConfig.bucketId,
      ID.unique(),
      inputFile,
    );

    const fileDocument = {
      type: getFileType(bucketFile.name).type,
      name: bucketFile.name,
      url: constructFileUrl(bucketFile.$id),
      extension: getFileType(bucketFile.name).extension,
      size: bucketFile.sizeOriginal,
      owner: ownerId,
      accountId,
      users: [],
      bucketFileId: bucketFile.$id,
    };
```

Рисунок 1.8 Фрагменти з файла *file.actions.ts*, функція uploadFile()

Read – отримання відсортованого списку файлів користувача через getFiles() у файлі file.actions.ts, що звертається до databases.listDocuments(files, …). Спочатку функція готує вхідні параметри. Користувач вказує, які типи файлів потрібно показати. Далі система під'єднується до бази даних через createAdminClient, а getCurrentUser з'ясовує, який користувач.

Після авторизації createQueries формує набір фільтрів. З готовими умовами databases.listDocuments звертається до колекції files у Appwrite та повертає документи, що відповідають заданим критеріям. Отриманий список проходить крізь parseStringify, який прибирає зайві дані, серіалізує дати й робить об'єкт безпечним для відправки на клієнт. Якщо виникає помилка, handleError виводить повідомлення. На рис.1.9 представлен фрагмент файлу file.actions.ts, в якому функція getFiles() надає список файлів користувача, який відсортований.

```
export const getFiles = async ({  Show usages  Klemjl
  types = [],
  searchText = "",
  sort = "$createdAt-desc",
  limit,
}: GetFilesProps) => {
  const { databases } = await createAdminClient();

  try {
    const currentUser = await getCurrentUser();

    if (!currentUser) throw new Error("Користувача не знайдено");

    const queries = createQueries(currentUser, types, searchText, sort, limit);

    const files = await databases.listDocuments(
      appwriteConfig.databaseId,
      appwriteConfig.filesCollectionId,
      queries,
    );

    console.log({ files });
    return parseStringify(files);
  } catch (error) {
    handleError(error, "Помилка під час отримання списку файлів");
  }
};
```

Рисунок 1.9 Фрагмент з файлу file.actions.ts, який надає відсортований список файлів користувача через getFiles()

Update – перейменування файлу за допомогою функції renameFile() в файлі file.actions.ts, яке використовує метод storage.updateFile(bucketId, fileId). Функція отримує ідентифікатор файлу, нову назву, розширення та шлях, після чого ініціалізує адміністративний клієнт Appwrite. Вона формує повну нову назву, поєднуючи введений користувачем текст і розширення, а потім оновлює відповідний документ у колекції files, встановлюючи поле name на нове значення. Після успішного оновлення викликається revalidatePath, щоб очистити кеш і одразу відобразити змінену назву у потрібному маршруті. Нарешті функція перетворює відповідь сервера в безпечний для клієнта формат через parseStringify і повертає оновлений об'єкт. Якщо під час будь-якого кроку виникає помилка, вона передається в handleError. Перейменування файлу функцією renameFile() продемонстровано на фрагменті файлу file.actions.ts на рис. 1.10.

```
export const renameFile = async ({  Show usages  Klemjl
  fileId,
  name,
  extension,
  path,
}: RenameFileProps) => {
  const { databases } = await createAdminClient();

  try {
    const newName = `${name}.${extension}`;
    const updatedFile = await databases.updateDocument(
      appwriteConfig.databaseId,
      appwriteConfig.filesCollectionId,
      fileId,
      {
        name: newName,
      },
    );

    revalidatePath(path);
    return parseStringify(updatedFile);
  } catch (error) {
    handleError(error, "Помилка під час перейменування файлу");
  }
};
```

Рисунок 1.10 Фрагмент з файлу реалізація функції renameFile()

Delete – видалення файлу через функцію deleteFile() у файлі file.actions.ts, що

викликає storage.deleteFile(bucketId, fileId). Функція отримує ідентифікатори документа в базі (fileId), самого файлу в бакеті (bucketFileId) і шлях сторінки, після чого підключається до модулів databases та storage за допомогою адміністративного клієнта. Спочатку вона видаляє запис про файл у колекції files. Якщо документ справді видалено, тоді прибирає й сам об'єкт із бакету сховища через storage.deleteFile, щоб не лишати «сирітського» файла. Після успішного видалення викликає revalidatePath, очищаючи кеш і примушуючи сторінку негайно оновити список файлів у UI. У відповідь функція повертає серіалізований об'єкт зі статусом success. Якщо в будь-який момент трапляється помилка, вона передається в handleError. Функція видалення файлу deleteFile() представлено в файлі file.actions.ts на рис. 1.11.

```
export const deleteFile = async ({  Show usages  Klemjl
  fileId,
  bucketFileId,
  path,
}: DeleteFileProps) => {
  const { databases, storage } = await createAdminClient();

  try {
    const deletedFile = await databases.deleteDocument(
      appwriteConfig.databaseId,
      appwriteConfig.filesCollectionId,
      fileId,
    );

    if (deletedFile) {
      await storage.deleteFile(appwriteConfig.bucketId, bucketFileId);
    }

    revalidatePath(path);
    return parseStringify({ status: "success" });
  } catch (error) {
    handleError(error, "Помилка під час видалення файлу");
  }
};
```

Рисунок 1.11 Видалення файлу через deleteFile()

Перед кожним видаленням чи оновленням виконується порівняння власника із поточним session.userId. Структура бази даних серверної платформи організована у вигляді колекцій документів із JSON-подібними записами, проте

фактично дані зберігаються на рівні реалізації в реляційній СУБД. При обробці запитів кожен виклик API колекцій транслюється в SQL-команди, а зв'язки між документами забезпечуються механізмами реляційної моделі.

Кожна колекція слугує контейнером для однорідних даних. У вебсховищі є дві колекції: «users», «files». Колекція «users» містить документи з інформацією про користувачів унікальний ідентифікатор, email, ім'я, а колекція «files» – документи з метаданими файлів ідентифікатор файла, ID користувача-власника, назва файла, розмір, ID bucket. На рис. 1.12 представлена блок-схема структури бази даних додатку Storo.

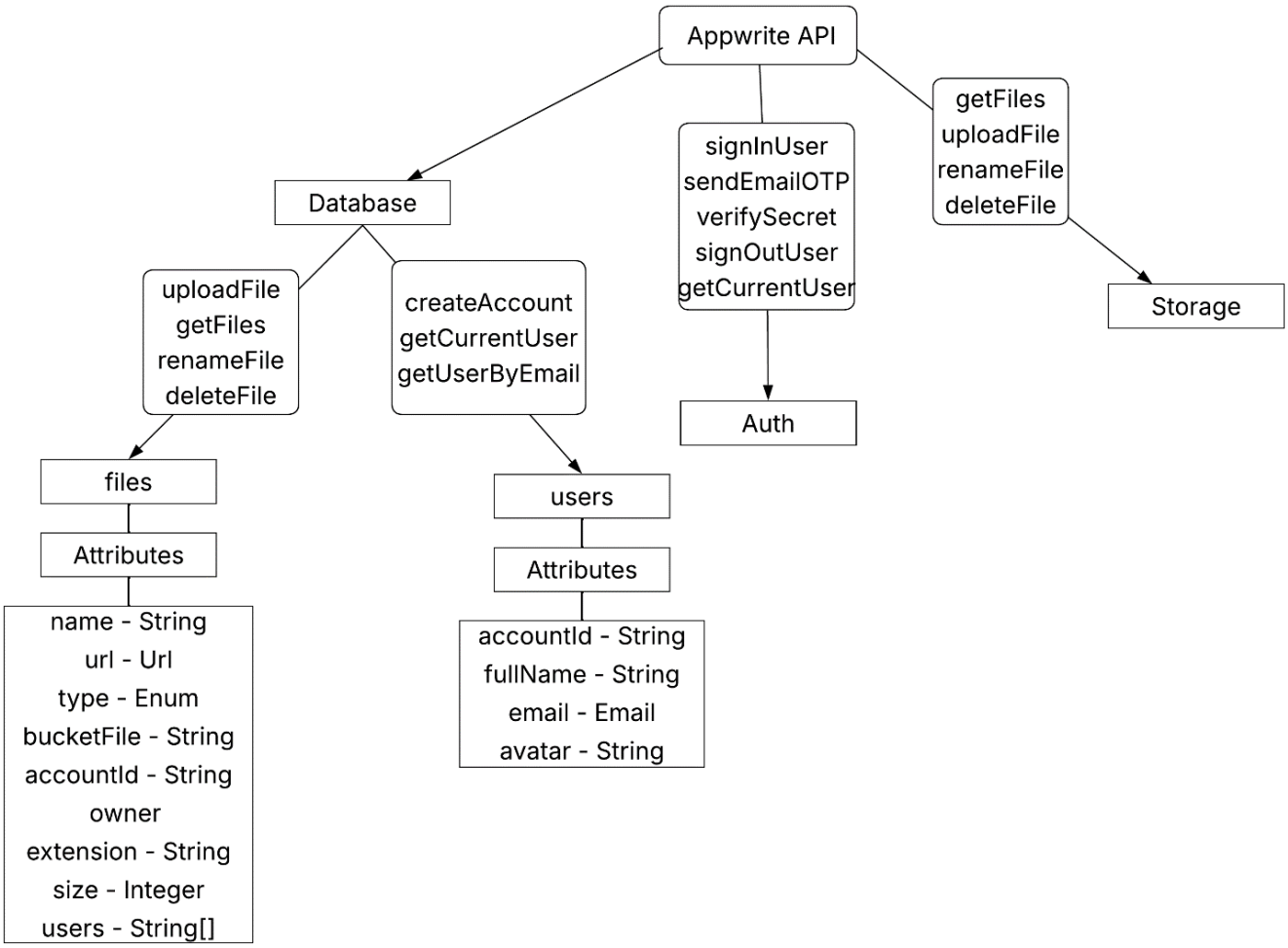


Рисунок 1.12 Блок-схема структури бази даних вебдодатку Storo

Безпека даних у колекціях контролюється ролями та правами доступу. Appwrite дозволяє присвоювати ролі на рівні колекції чи документу, значення ролей: «any», «all users», «all guests», «select users». Це означає, що операції CRUD

у колекціях можуть обмежуватися певними ролями. Запис про файл у «files» може бути доступний тільки власнику файлу або групі користувачів з роллю «власник», а всі інші - отримують відмову в читанні чи зміні.

У проєкті, також розглядаються приклади конкретної реалізації безпекових сценаріїв. Appwrite підтримує File Tokens – спеціальні підписані URL, що дозволяють тимчасово надати публічний доступ до приватного файла. У вебсховищі це може означати, що при необхідності поділитися приватним файлом генерується file token через, після чого формується посилання на файл з цим токеном для перегляду або завантаження.

share: () => updateFileUsers({ fileId: file.$id, emails, path })

Appwrite підтримує безпарольну автентифікацію за допомогою Email-OTP, що значно спрощує процес реєстрації та входу користувачів.

Першим етапом у процесі авторизації користувача – перевірити, чи вже зареєстровано введену електронну адресу в системі. Для цього застосовується функція getUserByEmail, яка звертається до бази даних Appwrite через адміністративний клієнт createAdminClient(). Адміністративний клієнт надає повні права доступу до ресурсів, що дозволяє виконувати запити до бази незалежно від сесії користувача.

У функції виконується запит до певної колекції бази даних, яка містить документи користувачів. Ідентифікатори бази (databaseId) та колекції (usersCollectionId) беруться з об’єкта конфігурації appwriteConfig. Сам запит формулюється з умовою Query.equal("email", [email]), що означає пошук записів, у яких поле email точно збігається з переданим значенням. Усе це свідчить про високий рівень кастомізації та контрольованості, який надає Appwrite для реалізації безпечної і надійної взаємодії з даними. Завдяки використанню таких механізмів, як File Tokens, безпарольна автентифікація через OTP, та прямий доступ до бази даних через адміністративні запити, проєкт забезпечує баланс між безпекою, зручністю для користувача та масштабованістю системи. В результаті, реалізоване рішення не тільки відповідає сучасним вимогам до захисту інформації, а й демонструє ефективний підхід до управління доступом у хмарному середовищі.

На рис. 1.13 наведено фрагмент функції getUserByEmail з файлу *user.actions.ts*, яка виконує пошук користувача за електронною поштою у базі даних.

```
const getUserByEmail : (email: string) => Promise<Document | nul... = async (email: string) : Promise<D
  const { databases } = await createAdminClient();

  const result : DocumentList<Document> = await databases.listDocuments(
    appwriteConfig.databaseId, // ID бази даних
    appwriteConfig.usersCollectionId, // колекція користувачів
    [Query.equal("email", [email])], // умова пошуку
  );
  // Повернення знайденого документа, або null якщо користувач не існує
  return result.total > 0 ? result.documents[0] : null;
};
// Обробка помилок виводить повідомлення у консоль
const handleError : (error: unknown, message: string) => neve... = (error: unknown, message: string)
  console.log(error, message);
  throw error;
};
```

Рисунок 1.13 Фрагмент коду функції getUserByEmail з файлу *user.actions.ts*

Другим кроком у процесі автентифікації є генерація одноразового пароля (OTP), який надсилається користувачу на електронну пошту. Це забезпечує безпечний спосіб входу без потреби вводити постійний пароль. Для виконання цієї задачі використовується функція sendEmailOTP, яка також працює через адміністративний клієнт createAdminClient(), що має відповідні дозволи для створення токенів.

Функція викликається метод account.createEmailToken, який створює унікальний email-токен, прив'язаний до конкретної адреси користувача. Цей токен надсилається на вказану пошту, після чого користувач повинен підтвердити його для входу в систему. Функція повертає userId – ідентифікатор користувача, що буде використаний у наступних етапах авторизації.

У разі виникнення помилки (наприклад, некоректна пошта або проблеми із сервісом) викликається обробник handleError, який виводить повідомлення про невдалу відправку OTP. На рис. 1.14 представлено функцію sendEmailOTP, яка виконує генерацію одноразового коду та його надсилання користувачу.

```
export const sendEmailOTP :({ email }: { email: string }) => Promise... = async ({ email }: {
  const { account } = await createAdminClient();

  try {
    // Створення одноразовий токен (OTP) та надсилається на email
    const session :Token = await account.createEmailToken(ID.unique(), email);
    // Повернення ідентифікатора користувача (accountId)
    return session.userId;
  } catch (error) {
    // Вивод повідомлення у разі помилки
    handleError(error, "Не вдалося надіслати одноразовий код на пошту");
  }
};
```

Рисунок 1.14 Фрагмент з файлу *user.actions.ts,* функції генерації OTP

Третім етапом є створення нового облікового запису користувача за допомогою функції createAccount. Ця функція призначена для первинної реєстрації і поєднує в собі одразу кілька ключових дій. Вона перевіряє, чи вже існує користувач із вказаною електронною адресою, викликаючи допоміжну функцію getUserByEmail. Якщо користувача не знайдено, система вважає, що це новий користувач, і переходить до наступного кроку – надсилання OTP-коду на вказану пошту. Після цього збережений accountId буде використовуватися як унікальний ідентифікатор у системі Appwrite.

Якщо адреса раніше не використовувалась, функція ініціалізує з'єднання з базою даних за допомогою createAdminClient і створює новий документ у колекції користувачів. У цьому документі зберігаються базові дані – ім’я, email, посилання на тимчасовий аватар та отриманий accountId. В кінці, функція повертає accountId у серіалізованому форматі, щоб його можна було використати на клієнтській стороні для підтвердження одноразового коду. Таким чином, функція createAccount виконує роль центрального вузла у процесі створення нового користувача, поєднуючи в собі валідацію, ініціалізацію з’єднання з Appwrite, створення облікового запису та підготовку даних до подальшої верифікації. Завдяки такій послідовній логіці, реалізація забезпечує не лише зручність для кінцевого користувача, а й гарантує цілісність і узгодженість даних у системі з самого початку взаємодії з додатком.На рис. 1.15 наведено фрагмент функції

createAccount, яка реалізує логіку перевірки, реєстрації нового користувача та збереження його профілю в базі.

```
export const createAccount : ({ fullName, email }: { fullName: string;... = async ({ Show
  fullName,
  email,
}: {
  fullName: string;
  email: string;
}) : Promise<any> => {
  // Перевірка чи існує вже користувач
  const existingUser : Document | null = await getUserByEmail(email);
  // Надсилання OTP-код та отримання accountId
  const accountId : string | undefined = await sendEmailOTP({ email });
  if (!accountId) throw new Error("Не вдалося надіслати одноразовий код");

      if (!existingUser) {
        const { databases } = await createAdminClient();
        // Створення нового документа користувача в колекції
        await databases.createDocument(
          appwriteConfig.databaseId,
          appwriteConfig.usersCollectionId,
          ID.unique(), // Генерація унікального ID документа
          {
            fullName,
            email,
            avatar: avatarPlaceholderUrl,
            accountId,
          },
        );
      }
      // Повернення accountId для подальшої авторизації через OTP
      return parseStringify({ accountId });
    };
```

Рисунок 1.15 Фрагмент функції createAccount

Четвертим етапом процесу автентифікації є підтвердження одноразового пароля (OTP), який користувач отримав на електронну пошту. Після того як користувач вводить код, він передається на сервер, де перевіряється за допомогою функції verifySecret.

Усередині функції за допомогою адміністративного клієнта createAdminClient відбувається звернення до Appwrite. Також викликається метод createSession, який перевіряє, чи введений OTP-код відповідає обліковому запису з переданим accountId. Якщо перевірка проходить успішно, створюється нова

сесія. Далі створена сесія зберігається у cookie з прапором httpOnly це означає, що її не зможе прочитати JavaScript на стороні клієнта, а отже забезпечується вища безпека даних. Крім того, встановлюються додаткові параметри: secure (cookie працює лише по HTTPS) та sameSite: "strict" (захист від CSRF-атак). На рис. 1.16 зображено фрагмент функції verifySecret, яка відповідає за валідацію OTP-коду і створення авторизованої сесії користувача.

```
export const verifySecret :({ accountId, password }: { accountId: st... = async ({  Show usages  Klen
  accountId,
  password, // OTP-код введений користувачем
}: {
  accountId: string;
  password: string;
}) :Promise<any>  => {
  try {
    const { account } = await createAdminClient();
    // Перевірка OTP-код та створення серверну сесію
    const session :Session  = await account.createSession(accountId, password);
    // Збереження секрет сесії в httpOnly cookie для безпечної авторизації
    (await cookies()).set("appwrite-session", session.secret, {
      path: "/", // Cookie доступний для всіх маршрутів застосунку
      httpOnly: true, // Cookie доступний лише серверу(захист від XSS)
      sameSite: "strict", // Cookie не буде надіслано сторонніми сайтами (проти CSRF)
      secure: true, // Працює лише через HTTPS(захист від перехоплення)
    });
    // Повернення id створеної сесії для підтвердження авторизації
    return parseStringify({ sessionId: session.$id });
  } catch (error) {
    handleError(error, "Не вдалося перевірити одноразовий код");
  }
```

Рисунок 1.16 Фрагмент функції verifySecret з файлу *user.actions.ts*

П’ятим етапом є отримання даних про поточного авторизованого користувача. Для цього використовується функція getCurrentUser, яка створює сесійний клієнт через createSessionClient. Цей клієнт працює на основі cookie, що містить секрет авторизованої сесії.

Спочатку викликається account.get(), щоб отримати базову інформацію про обліковий запис користувача в Appwrite. Потім, на основі accountId, система звертається до бази даних (usersCollectionId), щоб отримати детальний профіль користувача, який було збережено під час реєстрації (наприклад, повне ім’я, email, аватар). Якщо запис не знайдено, функція повертає null. Інакше, об’єкт користувача обробляється через parseStringify і повертається на клієнт.

На рис. 1.17 наведено фрагмент функції getCurrentUser, яка дозволяє динамічно підвантажувати профіль залогіненого користувача в інтерфейсі застосунку.

```
export const getCurrentUser : () => Promise<any> = async () : Promise<any> => {
  try {
    const { databases, account } = await createSessionClient();
    // Отримано дані залогіненого користувача
    const result : User<Preferences> = await account.get();
    // Отримано додаткову інформацію з бази даних
    const user : DocumentList<Document> = await databases.listDocuments(
      appwriteConfig.databaseId,
      appwriteConfig.usersCollectionId,
      [Query.equal("accountId", result.$id)],
    );

    if (user.total <= 0) return null;

    return parseStringify(user.documents[0]);
  } catch (error) {
    console.log(error);
  }
};
```

Рисунок 1.17 Фрагмент коду *user.actions.ts* функції getCurrentUser

Шостий етап – завершення сесії користувача. Функція signOutUser відповідає за вихід користувача із системи. Через сесійного клієнта викликається метод account.deleteSession("current"), що видаляє поточну авторизаційну сесію в Appwrite. Після цього за допомогою cookies().delete() також очищується cookie, що містить секрет сесії на клієнті. Далі, користувач перенаправляється на сторінку входу (/sign-in), щоб запобігти доступу до захищених сторінок без авторизації. Завдяки чітко визначеній процедурі завершення сесії, гарантується високий рівень безпеки після виходу користувача з системи. Видалення сесії як на стороні Appwrite, так і на клієнтському рівні унеможливлює несанкціонований доступ до облікового запису навіть у випадку перехоплення сесійного cookie. Завдяки чітко визначеній процедурі завершення сесії, гарантується високий рівень безпеки після виходу користувача з системи. Видалення сесії як на стороні Appwrite, так і на клієнтському рівні унеможливлює несанкціонований доступ до облікового запису навіть у випадку перехоплення сесійного cookie. Такий підхід до завершення

автентифікації відповідає сучасним вимогам щодо захисту персональних даних і є важливою складовою загальної моделі безпеки вебдодатку.На рис. 1.18 показано фрагмент коду, який відповідає за безпечний вихід користувача з вебдодатку, з очищенням сесії як на сервері, так і на клієнті.

```
export const signOutUser : () => Promise<never>  = async () : Promise<
  const { account } = await createSessionClient();

  try {
    await account.deleteSession("current");
    (await cookies()).delete("appwrite-session");
  } catch (error) {
    handleError(error, "Не вдалося виконати вихід з системи");
  } finally {
    redirect("/sign-in");
  }
};
```

Рисунок 1.18 Фрагмент коду *user.actions.ts*, реалазація функції signOutUser

Описана послідовність операцій складає комплексну модель авторизації, яка охоплює перевірку даних користувача, створення сесій та їх завершення.На рис. 1.19 подана схема, яка відображає структуру основної логіки вебдодатку із ключовими етапами взаємодії серверної частини з Appwrite та базою даних.

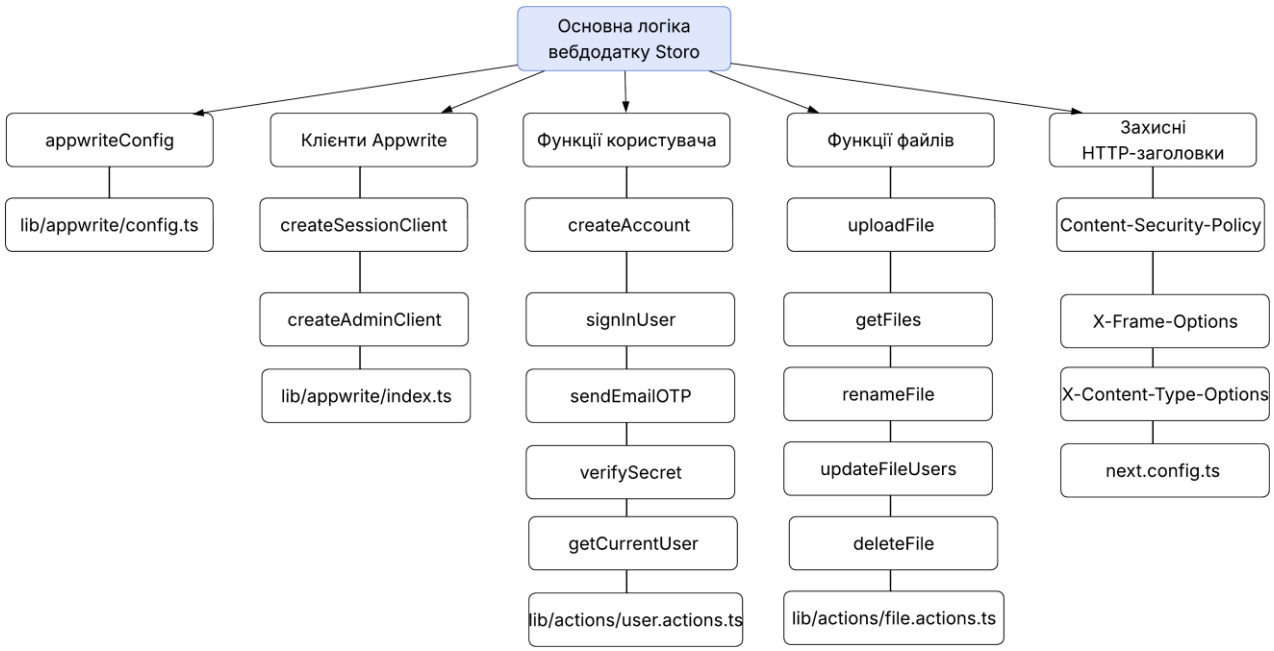


Рисунок 1.19 Схема основної логіки взаємодії серверної частини з базою даних вебдодатку Storo

1.3.3 Клієнтська частина

Фронтенд додатка побудовано з використанням Next.js App Router, React, Tailwind CSS та компонентної бібліотеки ShadCN UI. Для графіків застосовується бібліотека Recharts, а для форм – React Hook Form. Розглянемо структуру основних сторінок і компонентів, а також механізми асинхронних запитів та захисту маршрутів. На головній сторінці відображається кругова діаграма використання дискового простору і список нещодавно завантажених файлів. Серверна функція getTotalSpaceUsed обчислює сумарний об’єм використаного місця і передає його до компонента <Chart>, який формує радіальний графік. Компонент <Chart> будує кольорове кільце, де відображається відсоток використаного простору в центрі – значення у % і підпис «Використано місця». Під графіком виводиться текст «Доступно місця» з обчисленим значенням.

Внизу графіка виводяться картки з підсумковими даними по кожному типу файлів - це список посилань з іконками і датою останнього додавання. Зліва, на сторінці подано блок з заголовком «Нещодавно завантажені файли», де через компонент <Link> формуються елементи списку: компонент <Thumbnail> і деталі файлу назва, дата створення, а також випадаюче меню з діями «Поділитися», «Видалити», «Перейменувати» – файл ActionDropdown.tsx . Усі компоненти інтерфейсу побудовано на принципах повторного використання та модульності. Кожен функціональний блок винесено в окремий компонент, що забезпечує зручність у підтримці та масштабуванні проєкту. Інтерфейс додатку адаптивний, тобто автоматично підлаштовується під різні розміри екранів, що особливо важливо для користувачів мобільних пристроїв.

Для забезпечення асинхронної взаємодії з сервером у Next.js використовуються серверні функції (наприклад, Server Actions), які дозволяють виконувати запити до бази даних та хмарного сховища Appwrite безпосередньо з компонентів. Всі маршрути захищені – користувач не зможе отримати доступ до жодної сторінки додатку, якщо не пройшов автентифікацію. Захист реалізовано за допомогою перевірки наявності валідної сесії Appwrite через middleware та функцію getCurrentUser. У випадку її відсутності користувача автоматично

перенаправляє на сторінку входу.Компонент модального вікна для управління файлами: перейменування, перегляд, спільний доступ, видалення на рис. 1.20.

```
</DialogTitle>
{value === "rename" && (
  <Input
    type="text"
    value={name}
    onChange={(e :ChangeEvent<HTMLInputElement> ) :void => setName(e.target.value)}
  />
)}
{value === "details" && <FileDetails file={file} />}
{value === "share" && (
  <ShareInput
    file={file}
    onInputChange={setEmails}
    onRemove={handleRemoveUser}
  />
)}
{value === "delete" && (
  <p className="delete-confirmation">
    Ви впевнені, що хочете видалити{` `}
    <span className="delete-file-name">{file.name}</span>?
  </p>
)}
```

Рисунок 1.20 Компонент модального вікна для управління файлами: перейменування, перегляд, спільний доступ, видалення

На сторінках «Документи», «Зображення», «Медіа», «Інше» реалізовано перегляд і управління файлами певного типу. Шлях /[type] як динамічний сегмент сторінки зчитує параметр type і передає його у серверну функцію «getFiles», яка повертає список відповідних файлів з бази даних.

Також обробляються параметри пошуку «searchParams.query» і сортування «searchParams.sort». У верхній частині сторінки виводиться заголовок із назвою категорії, а також селектор сортування компонент <Sort>. Селектор <Select> за варіантами сортування,при зміні опції оновлює URL-параметр «sort» . Картка файлу <Card> формує посилання, що веде на файл у хмарі, і містить прев'ю файлу - іконку чи зображення через компонент <Thumbnail>, також блок деталей (назва, дата, автор, розмір). У верхньому правому куті картки розташований компонент <ActionDropdown>, що відкриває меню дій над файлом, такі як перейменувати,

поділитися, завантажити, видалити. Частина лівої панелі містить містить інформацію про поточного користувача: повне ім’я, електронну пошту, аватар.

Вхід в систему виконано через форму одноразового паролю (OTP). Сторінки входу «/sign-in» та реєстрації «/sign-up» створені як клієнтські компоненти з React Hook Form, пакет react-hook-form та валідація через Zod. Інтерфейс карти файлу побудований за принципами UX-орієнтованого дизайну: кожна карта виводить зображення чи іконку попереднього перегляду (<Thumbnail>), назву, дату, автора та розмір файлу, а також інтерактивне меню <ActionDropdown>, що відкриває набір дій для кожного об'єкта. Усі дії виконуються асинхронно, без оновлення сторінки, що підвищує продуктивність і зручність. Ліва панель навігації динамічно відображає інформацію про поточного користувача, включно з аватаром, іменем та email, отриманими з Appwrite-сесії. Код обробки подачі форми для реєстрації/авторизації на рис. 1.21.

```
return (
  <>
    <Form {...form}>
      <form onSubmit={form.handleSubmit(onSubmit)} className="auth-form">
        <h1 className="form-title">
          {type === "sign-in" ? "Увійти" : "Зареєструватися"}
        </h1>
        {type === "sign-up" && (
          <FormField
            control={form.control}
            name="fullName"
            render={({ field } : { field: ControllerRenderProps<{ email: s... ) : Element => (
              <FormItem>
                <div className="shad-form-item">
                  <FormLabel className="shad-form-label">
                    Повне ім'я
                  </FormLabel>

                  <FormControl>
                    <Input
                      placeholder="Введіть ваше повне ім'я"
                      className="shad-input"
                      {...field}
                    />
                  </FormControl>
```

Рисунок 1.21 Код обробки подачі форми для реєстрації/авторизації

```
    try {
      const user :any =
        type === "sign-up"
          ? await createAccount({
              fullName: values.fullName || "",
              email: values.email,
            })
          : await signInUser({ email: values.email });

      setAccountId(user.accountId);
    } catch {
      setErrorMessage(
        "Не вдалося створити акаунт. Будь ласка, спробуйте ще раз.",
      );
    } finally {
      setIsLoading(false);
    }
  };
```

Рисунок 1.22 Код обробки помилки для реєстрації/авторизації

При відправці форми користувач отримує на електронну пошту одноразовий код, після чого відкривається модальне вікно вводу коду – компонент <OtpModal>. У цьому діалоговому вікні є шість полів введення OTP, компонент <InputOTP> із слотами <InputOTPSlot> від ShadCN. Компонент введення одноразового коду підтвердження з шістьма окремими полями на рис.1.23.

```
<InputOTP maxLength={6} value={password} onChange={setPassword}>
  <InputOTPGroup className="shad-otp">
    <InputOTPSlot index={0} className="shad-otp-slot" />
    <InputOTPSlot index={1} className="shad-otp-slot" />
    <InputOTPSlot index={2} className="shad-otp-slot" />
    <InputOTPSlot index={3} className="shad-otp-slot" />
    <InputOTPSlot index={4} className="shad-otp-slot" />
    <InputOTPSlot index={5} className="shad-otp-slot" />
  </InputOTPGroup>
</InputOTP>
```

Рисунок 1.23. Компонент введення одноразового коду підтвердження з

Користувач вводить код, і при натисканні кнопки «Submit» запускається серверна дія verifySecret з Appwrite, яка перевіряє коректність коду та створює

сесію, cookie встановлюється через next/headers. У разі успіху виконується router.push(/) на домашню сторінку. Перевірка введеного OTP-коду та перенаправлення користувача на головну сторінку у разі успіху на рис.1.24.

```
    try {
      const sessionId :any  = await verifySecret({ accountId, password });
      console.log({ sessionId });
      if (sessionId) router.push("/");
    } catch (error) {
      console.log("Не правильний одноразовий пароль", error);
    }
    setIsLoading(false);
  };
```

Рисунок 1.24 Перевірка введеного OTP-коду та перенаправлення користувача на головну сторінку у разі успіху

У модальному вікні також передбачено посилання «Натисніть, щоб надіслати знову», що повторно викликає sendEmailOTP для отримання нового коду. Для асинхронних запитів фронтенд використовує переважно Next.js Server Actions і Appwrite SDK. Всі основні операції з файлами (отримання списку, завантаження, перейменування, надання доступу, видалення) реалізовано як серверні функції uploadFile, getFiles, renameFile, updateFileUsers, deleteFile тощо. Кожна з них виконує виклики до Appwrite (через createAdminClient() або createSessionClient()) для роботи з базою даних та сховищем файлів. Наприклад, getFiles збирає умови запиту з Appwrite на основі поточного користувача, типів, тексту пошуку та сортування, а потім викликає databases.listDocuments. Серверні дії повертають дані у вигляді JSON для використання у відповідних React-компонентах (іноді з викликом parseStringify для безпечної передавання). Асинхронна завантаження файлу (uploadFile) спочатку зберігає файл у Appwrite Storage, потім створює документ у колекції «files», і у кінці викликає revalidatePath(path), аби оновити статично-рендерений контент сторінки. Реалізація входу/реєстрації також використовує серверні дії: відправка OTP (createEmailToken) і верифікація коду (createSession) через createAdminClient(). Отже, взаємодія з бекендом виконується «на сервері» Next.js – фронтендські

компоненти викликають функції, які звертаються до Appwrite, не роблячи прямі fetch до REST API. Це забезпечує безпеку (секретні ключі Appwrite не потрапляють у клієнтський код) і чисту інтеграцію з архітектурою Next.js. Захист маршрутів реалізовано через перевірку сесії на рівні layout-директорії. У компоненті app/(root)/layout.tsx (затримка рендеру динамічна) спочатку викликається getCurrentUser(). Якщо поточний користувач не знайдений (неавторизований запит), виконується redirect("/sign-in"), і доступ до внутрішніх сторінок блокується. Таким чином, всі вкладені сторінки (Dashboard, File Manager, Profile тощо) є захищеними – без активної сесії користувача неможливий перехід на них. Коли ж користувач авторизований, його дані передаються компонентам Sidebar, MobileNavigation і Header для відображення особистої інформації та доступу до виходу з акаунту. Натискання кнопки «Logout» у компоненті <Header> або <MobileNavigation> викликає серверну дію signOutUser(), яка видаляє сесію і робить редирект на сторінку входу. На рис. 1.25 надано фрагмент коду інтерфейсу кнопки виходу з Storo.

```
<FileUploader ownerId={ownerId} accountId={accountId} />
<Button
  type="submit"
  className="mobile-sign-out-button"
  onClick={async () :Promise<never> => await signOutUser()}
>
  <Image
    src="/assets/icons/logout.svg"
    alt="logo"
    width={24}
    height={24}
  />
  <p>Logout</p>
</Button>
```

Рисунок 1.25 Елементи інтерфейсу кнопка виходу з Storo

Розглянуто всі ключові елементи інтерфейсу користувача, що забезпечують повноцінну функціональність і зручність роботи із вебдодатком. Для узагальнення цієї інформації у табл. 1.4 подано стислий опис основних UI-компонентів, реалізованих у проєкті.

Таблиця 1.4. Основні компоненти інтерфейсу вебдодатка Storo

Компонент	Опис та призначення
<Chart>	Візуалізує кругову діаграму заповнення дискового простору, з відображенням відсотків та текстової інформації.
<Layout>	Забезпечує загальну структуру сторінок (зазвичай містить хедер, сайдбар, контент).
<Thumbnail>	Генерує мініатюри файлів (зображення або іконки для файлів різних типів).
<Link>	Формує посилання з деталями файлу (назва, дата створення тощо).
<ActionDropdown>	Випадаюче меню з діями (Rename, Delete, Share) для роботи з файлами.
<Sort>	Селектор для сортування списку файлів за параметрами.
<Select>	Селектор для вибору способу сортування файлів, який оновлює URL-параметр.
<Card>	Картка, що містить прев'ю файлу, посилання, і детальну інформацію (назва, автор, дата, розмір).
<AuthForm>	Форма для входу або реєстрації користувача (email та ім'я).
<OtpModal>	Модальне вікно для введення OTP-коду користувачем.
<InputOTP>	Поле для введення одноразового коду, містить слоти <InputOTPSlot>.
<Dialog>	Модальне вікно для виконання додаткових дій (наприклад, поділитися файлом, підтвердити видалення).
<ShareInput>	Поле введення email для спільного доступу до файлу.
<Toaster>	Виводить сповіщення про помилки та успішні операції.

1. 4 Тестування розробленого вебдодатку

1.4.1 Розгортання вебдодатку на Vercel

В рамках розробки сучасних вебдодатків особливу увагу приділяють не лише функціональності, а й швидкості розгортання, стабільності роботи та безпеці. Саме тому для хостингу і публікації цього вебдодатку було обрано платформу Vercel.

Vercel – це хмарний сервіс, спеціалізований на розміщенні фронтенд-проєктів, зокрема застосунків на основі Next.js. Платформа надає зручний інтерфейс для автоматизації процесів розгортання. Vercel підтримує серверний рендеринг, статичну генерацію сторінок, serverless-функції, а також забезпечує масштабованість у виробничому середовищі.

Однією з ключових переваг Vercel є зручна інтеграція з популярними системами контролю версій, такими як GitHub, GitLab чи Bitbucket. Завдяки цьому процес оновлення проєкту відбувається автоматично: кожний новий коміт у репозиторії запускає збірку, проходження тестів і публікацію актуальної версії

додатку без додаткового втручання розробника.

Важливим аспектом є безпека розгортання та експлуатації вебдодатку. Vercel автоматично забезпечує захищене з'єднання завдяки застосуванню SSL/TLS сертифікатів для всіх опублікованих додатків. Це дозволяє гарантувати конфіденційність та цілісність даних, що передаються між клієнтом і сервером. Платформа не підтримує відкриті порти для прямого зовнішнього доступу, що знижує ймовірність несанкціонованого втручання. Всі змінні середовища зберігаються у захищеному вигляді й недоступні для сторонніх осіб, а кожен деплой ізольований, тому ризик впливу на середовище суттєво мінімізований. Крім цього, впроваджено аудит подій, журналювання та контроль доступу до репозиторію і проєкту, включаючи двофакторну автентифікацію для облікових записів розробників.

1.4.2 Функціональний огляд основних сторінок

У ході тестування після розгортання вебдодатку Storo було перевірено коректність роботи основних сторінок та ключових сценаріїв взаємодії користувача.

На рис. 1.26 представлено вигляд форми створення нового облікового запису, що містить обов'язкові поля для введення електронної пошти та повного імені користувача.

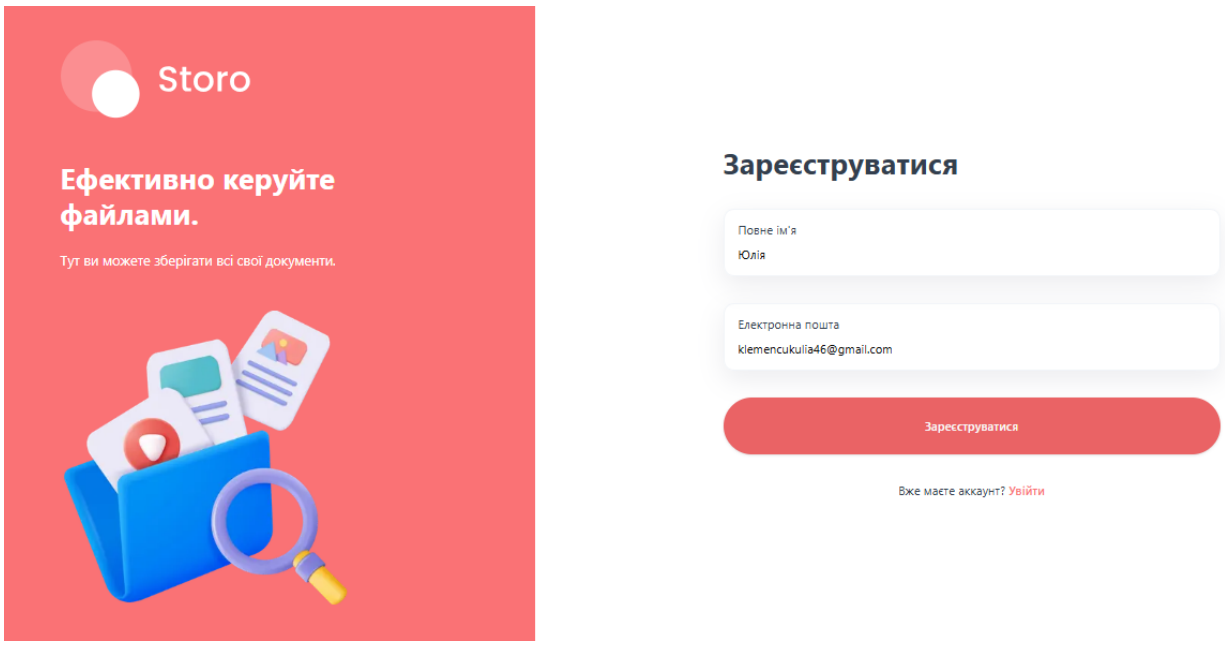


Рисунок 1.26 Сторінка реєстрації

На рис. 1.27 наведено процес отримання OTP-коду на пошту для підтвердження дій користувача.

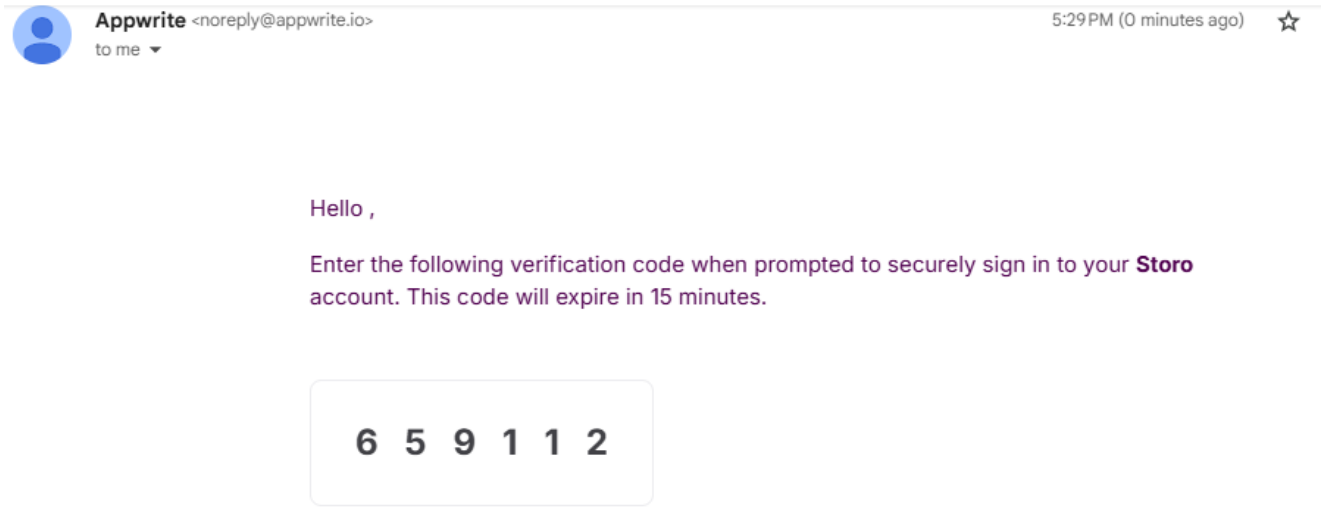


Рисунок 1.27 Електронний лист з одноразовим кодом підтвердження , надісланий автоматичною системою

На рис. 1.28 зображено інтерфейс, у якому користувач повинен ввести отриманий код для завершення авторизації.

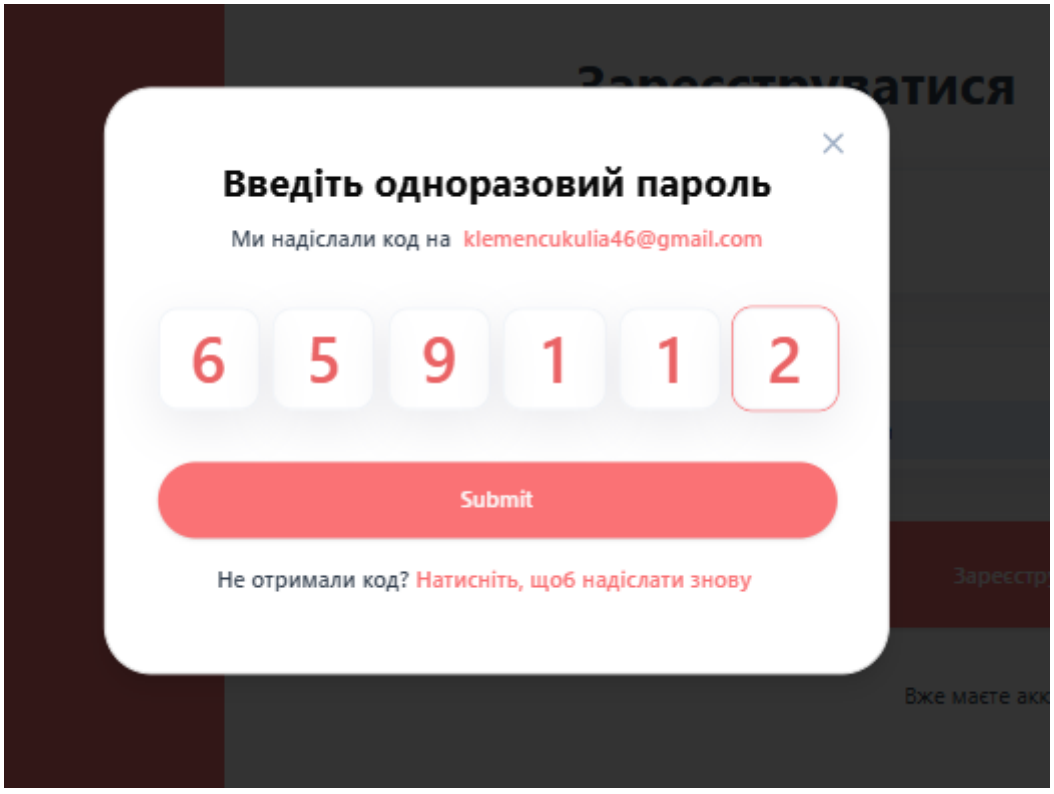


Рисунок 1.28 Вигляд модального вікна для введення одноразового коду

На рис. 1.29 головна сторінка вебсховища демонструє доступ до основної функціональності застосунку: навігацію по категоріях, перегляд файлів,

завантаження нових даних і вихід із системи.

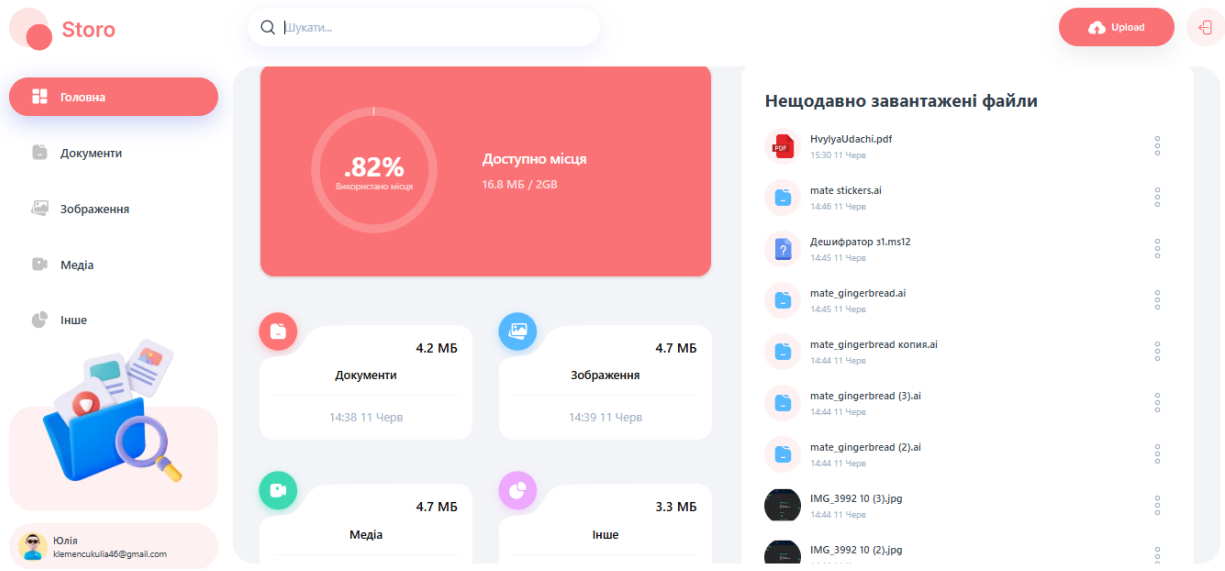


Рисунок 1.29. Головна сторінка вебсховища

На рис. 1.30 зображена сторінка «Документи» з сортуванням за розміром показує, що користувач може впорядковувати документи за різними критеріями для зручного пошуку та роботи з файлами.

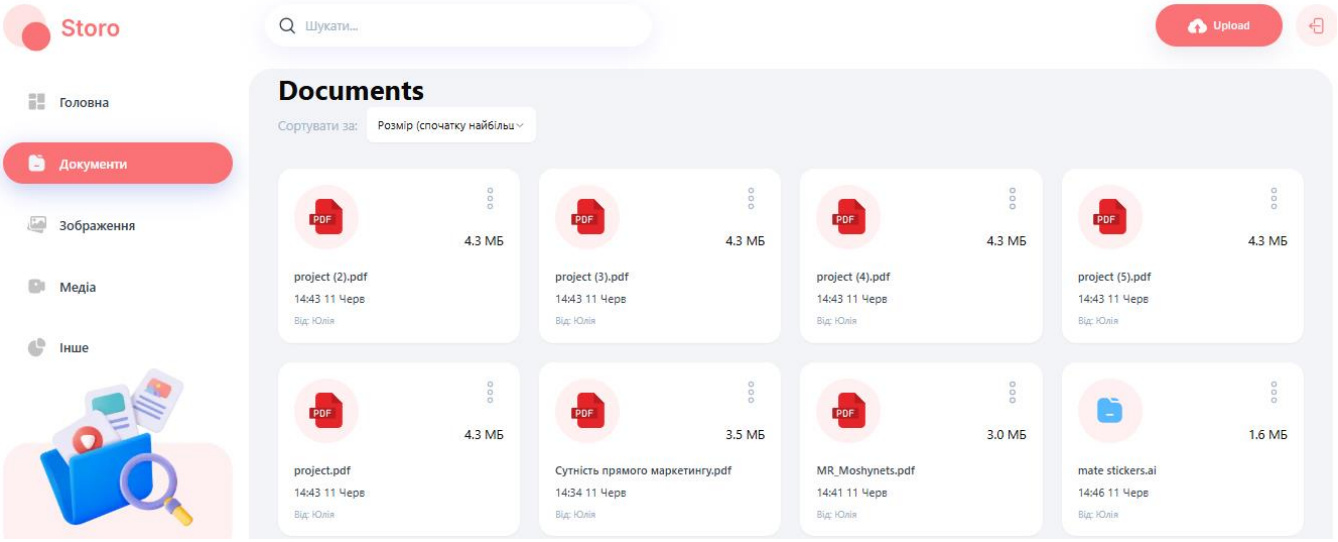


Рисунок 1.31 Сторінка «Документи» з сортуванням файлів за розміром

На рисунку 1.32 представлено алгоритм функціонування пошукового механізму у вебзастосунку. Процес пошуку реалізовано як частину клієнтсько-серверної взаємодії, що забезпечує швидку фільтрацію даних за заданими критеріями. Пошук ініціюється користувачем, який вводить ключове слово або фрагмент назви файлу у відповідне поле введення на інтерфейсі.

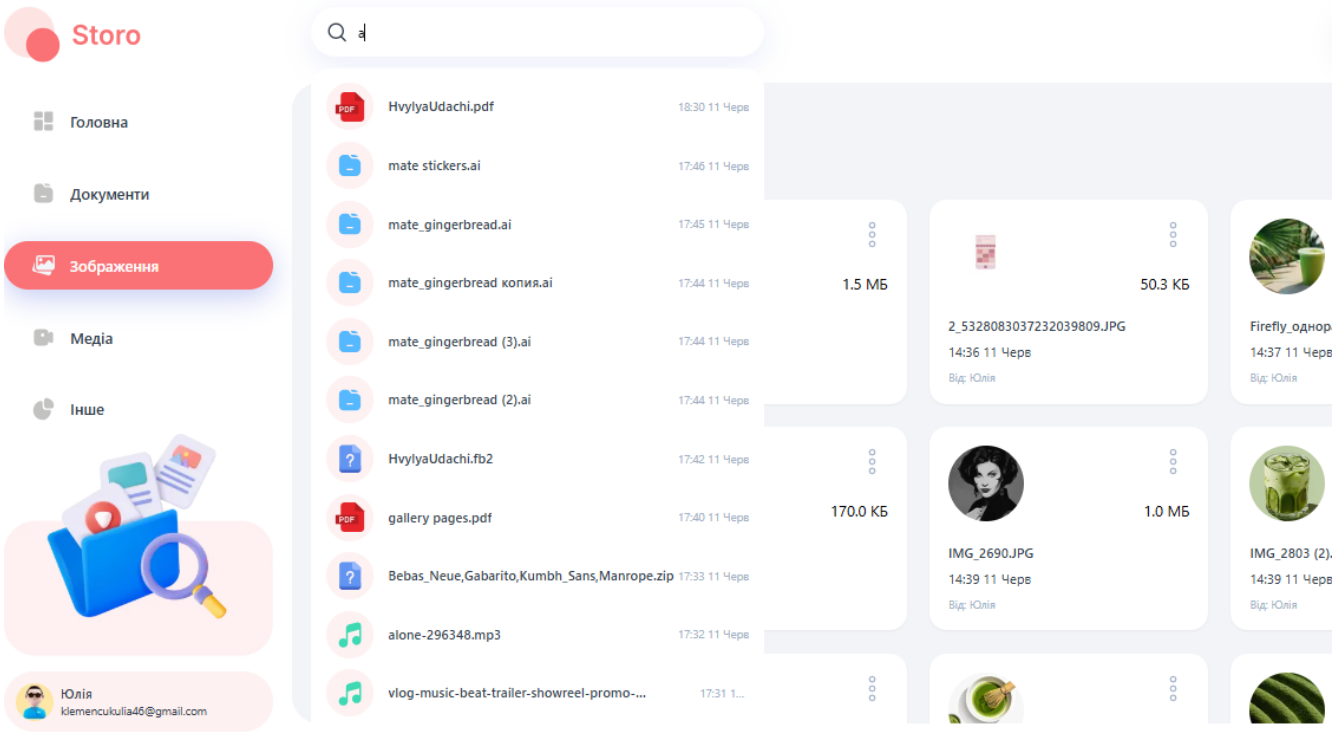


Рисунок 1.32 Пошук документів у системі

На рис. 1.33 зображено блок-схему алгоритму автентифікації користувача через OTP-код, що демонструє послідовність дій під час входу або реєстрації у вебзастосунку.

Автентифікація починається з відкриття сторінки входу. Далі перевіряється, чи є користувач зареєстрованим. Якщо ні – користувач переходить на сторінку реєстрації, вводить ім’я та електронну пошту, після чого натискає кнопку «Зареєструватися» і отримує OTP-код. Якщо користувач уже зареєстрований – він вводить email і натискає «Увійти», після чого також отримує код.

Наступний крок – перевірка поштової скриньки. Якщо код не надійшов, система пропонує повернутися на сторінку входу та натиснути компонент «Натисніть, щоб надіслати знову». Якщо код отримано – його потрібно скопіювати та вставити у відповідне поле, після чого натиснути кнопку «Submit». У разі неправильного коду користувач вводить його повторно. У разі успішного підтвердження система перенаправляє на головну сторінку. На рис. 1.27 зображено послідовність дій користувача під час автентифікації в системі.

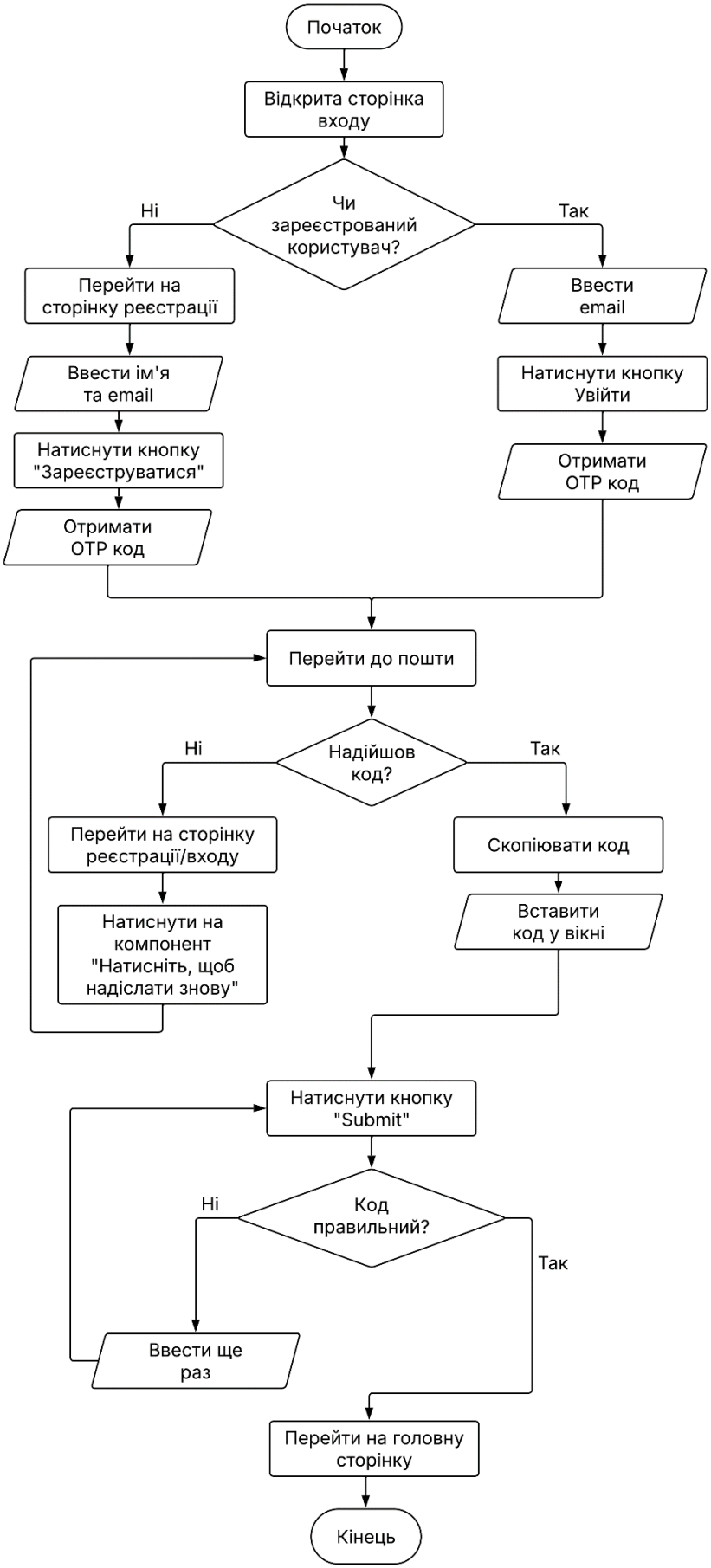


Рисунок 1.34 Блок-схема логіки входу та реєстрації користувача з підтвердженням через одноразовий код (OTP)

На рис. 1.35 представлено функціональні можливості управління файлами, зокрема перегляд деталей, перейменування, видалення та надання спільного доступу.

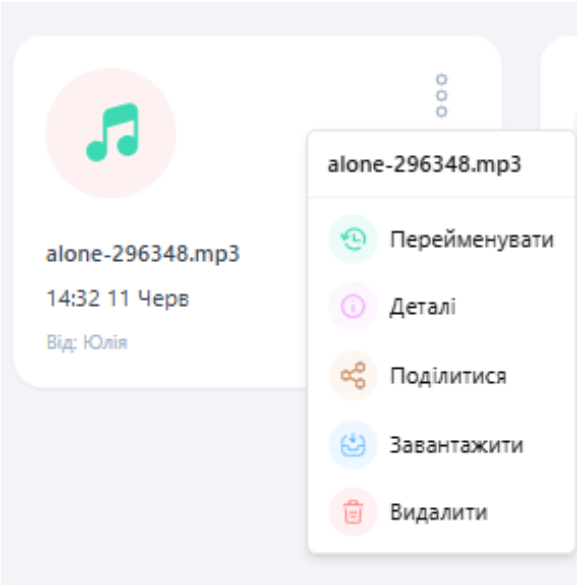


Рисунок 1.35 Контекстне меню керування документом

Детально розглянут рис. 1.36, який ілюструє послідовність дій користувача при керуванні файлами у вебдодатку.

Алгоритм починається з відкриття головної сторінки. Перший крок натискання на іконку «Завантажити» для вибору файлів на пристрої. Після завантаження користувач переходить до відповідного розділу – Документи, Зображення, Медіа або Інше залежно від типу файлу.

Наступним кроком є пошук потрібного файлу за назвою чи частиною назви. Результати відображаються у вигляді списку, в якому користувач може натиснути на контекстне меню окремого файлу.

Після цього з'являється вибір дії:

перейменувати файл – користувач вводить нову назву, після чого файл оновлюється; завантажити файл – копія зберігається на пристрій; поділитися файлом – вказується email-адреса отримувача, після чого генерується посилання; видалити файл – система запитує підтвердження дії; у разі позитивної відповіді файл видаляється зі сховища; переглянути деталі — відкривається інформація про тип, розмір, дату та інші атрибути. На рис. 1.30 розглянуто блок-схему.

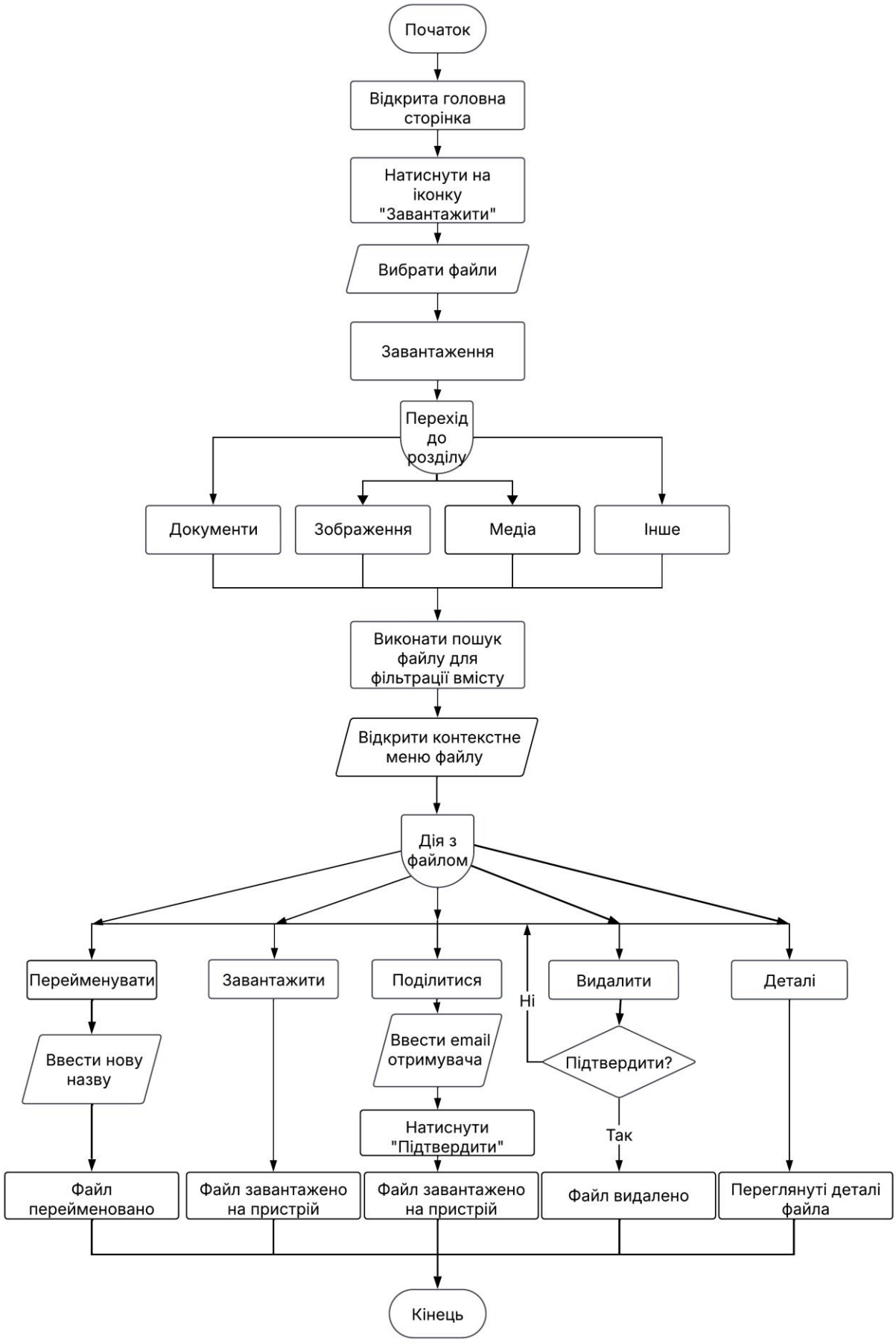


Рисунок 1.36. Блок-схема послідовності дій користувача при відкриті контекстного меню файла

1.4.3 Тестування безпеки вебдодатку за допомогою OWASP ZAP

Для оцінки безпеки розробленого вебдодатку було проведено автоматизоване тестування з використанням інструменту OWASP ZAP (Zed Attack Proxy). OWASP ZAP – це відкрите програмне забезпечення для перевірки вебдодатків на вразливості, що включає сканування на наявність типових загроз, таких як міжсайтовий скриптинг (XSS), помилки конфігурації, незахищені HTTP-заголовки та інші поширені проблеми.

Процес сканування включав автоматичний аналіз розгорнутого вебдодатку на платформі Vercel з наступною детальною інтерпретацією отриманих результатів. Основна увага була спрямована на виявлення потенційних ризиків та вразливостей. У таб. 1.5. наведено резльтати тестування.

Таблиця 1.5. результатів тестування OWASP ZAP

№	Назва загрози	Рівень ризику	Статус після виправлення
1	Відсутність заголовка Content-Security-Policy (CSP)	Середній	Виправлено
2	Міждоменна неправильна конфігурація (CORS)	Середній	Виправлено
3	Відсутність X-Frame-Options (захист від клікджекингу)	Середній	Виправлено
4	Відсутність X-Content-Type-Options	Низький	Виправлено

Як показано у табл. 1.5, під час сканування було виявлено низку типових вебзагроз, більшість з яких мали середній рівень ризику. Кожен з пунктів вказує на відсутність або неправильну реалізацію ключових заголовків безпеки, що можуть бути використані зловмисниками для проведення атак типу XSS, CORS-manipulation або clickjacking.

Далі наведено інтерпретацію кожної виявленої загрози з поясненням її сутності та описом способу усунення. Основна мета впроваджених змін – зменшити площу потенційної атаки та підвищити стійкість вебдодатку до типових загроз, визначених у моделі OWASP Top 10.

1. Content Security Policy (CSP)

CSP – це HTTP-заголовок, який дозволяє вебресурсу чітко визначити перелік дозволених джерел для завантаження різних типів ресурсів (скрипти, стилі,

зображення тощо). Відсутність цього заголовка може призводити до міжсайтового скриптингу (XSS), коли зловмисник вводить у сторінку шкідливий скрипт, який браузер користувача виконує як частину довіреного вебдодатку. Додавши цей заголовок, ми значно знижуємо ризик такої атаки. На рис. 1.37 відображається код політики Content Security Policy.

```
const ContentSecurityPolicy = `
  default-src 'self';
   script-src 'self' 'unsafe-inline';
  style-src 'self' 'unsafe-inline';
  img-src 'self' data: blob: https://cdn.pixabay.com https://img.freepik.com https://
  font-src 'self';
  connect-src 'self' https://fra.cloud.appwrite.io/v1 wss://fra.cloud.appwrite.io/v1;
  base-uri 'self';
  form-action 'self';
  object-src 'none';
  frame-ancestors 'none';
  upgrade-insecure-requests;
`;
```

Рисунок 1.37 Код політики Content Security Policy

5. Cross-Origin Resource Sharing (CORS)

У процесі усунення виявлених ризиків було прийнято рішення замість використання надмірно широкої політики CORS перейти до більш гнучкої та безпечної системи розмежування доступу – Scopes в Appwrite. Згідно з принципом мінімально необхідних привілеїв, додатку були залишені лише ті області доступу, які потрібні для його коректної роботи: Auth (автентифікація користувачів), Storage (доступ до файлів) та Database (робота з базою даних). Усі інші доступи, зокрема до Messaging, Functions, Sites та інших сервісів, було відключено як непотрібні.

3. X-Frame-Options (Захист від клікджекингу)

Заголовок X-Frame-Options запобігає атакам типу «clickjacking», коли сторінка застосунку завантажується у невидимий фрейм, і користувач, не усвідомлюючи цього, виконує небажані дії на атакованому сайті. Встановлення значення цього заголовка на DENY забороняє завантаження сторінки у фреймі іншого вебресурсу. На рис. 1.38 продемонстровано код для захисту від clickjacking

4. X-Content-Type-Options (MIME-sniffing)

Цей заголовок забороняє браузеру «здогадуватися» про тип контенту файлів, що надсилаються сервером, таким чином унеможливлюючи атаки, які базуються на підміні типів файлів. На рис. 1.38 продемонстровано фрагменти коду для зпобігання атаки MIME-sniffing.

```
async headers() : Promise<{ source: string; headers: { key:... {
  return [
    {
      source: "/:path*",
      headers: [
        {
          key: "Content-Security-Policy",
          value: ContentSecurityPolicy.replace(/\s{2,}/g, " ").trim(),
        },
        {
          key: "X-Frame-Options",
          value: "DENY",
        },
        {
          key: "X-Content-Type-Options",
          value: "nosniff",
        },
      ]
```

Рисунок 1.38 Фрагмент коду додавання захисних HTTP-заголовків

Результати тестування підтверджують належний рівень безпеки вебдодатку після усунення виявлених недоліків. Виявлені загрози були типові для вебсередовища, а вжиті заходи з їх усунення суттєво знизили потенційні ризики. Проведена перевірка дозволила оцінити стійкість системи до найбільш поширених вразливостей та забезпечити її відповідність базовим вимогам інформаційної безпеки.

2 ЕКОНОМІЧНИЙ РОЗДІЛ

У даному дипломному проєкті було розроблено програмний продукт вебзастосунок для зберігання та керування користувацькими файлами в хмарному середовищі. Його реалізація базується на сучасних підходах до розробки програмного забезпечення та відповідає актуальним вимогам до безпеки, масштабованості й зручності користування.

Ефективність будь-якого програмного продукту визначається його якістю та ефективністю процесу розробки. Якість програмного забезпечення оцінюється за трьома ключовими критеріями: зручність з позиції користувача, оптимальне використання ресурсів та відповідність функціональним вимогам.

Оцінювання якості програмного продукту передбачає визначення витрат на його створення, зокрема трудомісткості розробки. У цьому розділі буде виконано відповідні розрахунки, що дозволять оцінити витрати часу та ресурсів, необхідних для реалізації вебдодатку.

2.1 Визначення трудомісткості розробки програмного забезпечення

Тривалість створення програмного продукту визначається його складністю, обсягом робіт, трудомісткістю розробки, рівнем підготовки виконавців, а також запланованими строками, обумовленими ринковими вимогами. Для оцінки обсягу програмного забезпечення застосовується метод структурної аналогії, згідно з яким за допомогою спеціалізованих каталогів аналогічних програм визначається орієнтовний обсяг програмного коду в тисячах умовних машинних команд для подібного програмного рішення.

Таблиця 2.1. Каталог аналогів

Найменування ПП	**Обсяг функції ПП – Vo, ум. машинних командах**
1. Комплексні системи ведення БД	950 – 7430
2. ПП введення інформації	1060 – 5750
3. ПП автоматизованих розрахунків	1300 – 860

Обравши аналог ПП, що містить Vo в умовних машинних командах,

визначати трудомісткость на основі табл. 2.2.

Таблиця 2.2. Обсяг команд

Обсяг ПП, тис.умов.машинних команд	**Норма часу, люд/год**
1.00	229
2.00	244
3.00	262
4.00	283
5.00	306
6.00	330
7.00	357
8.00	385
9.00	414
10.00	445
12.00	510
14.00	580
16.00	654
18.00	731

На основі отриманих даних із довідника встановлюється базова норма часу для розробки програмного продукту. Вихідне значення 283(люд.-годин) коригується за допомогою поправочного коефіцієнта, що враховує умови розробки, зокрема роботу в умовах комп'ютера, де коефіцієнт (Кк) може змінюватися від 0,7 до 0,8. В прикладі, який обраний використовується значення 0,7:

$T^a p=283x0,8=226,4$ (люд.- годин)

Трудомісткість програмного продукту визначається по кожному етапу розробки окремо на підставі трудомісткості аналога з урахуванням складності розробки, ступеня новизни і ступеня використання в розробці стандартних модулів:

$$T_{ТЗ} = T^a p \times L_1 \times K_H \qquad (2.1)$$

$$T_{ТП} = T^a p \times L_2 \times K_H \qquad (2.2)$$

$$T_{РП} = T^a p \times L_3 \times K_H \times K_T \qquad (2.3)$$

Для розрахунку необхідні наступні коефіцієнти:

Li – питома вага і-го етапу розробки (див. табл. 2.3);

Кн – поправочний коефіцієнт, що враховує ступінь новизни (див. табл.2.4);

Кт – поправочний коефіцієнт, що враховує ступінь використання в розробці типових програм (див. табл. 2.5).

Таблиця 2.3. Значення питомих коефіцієнтів трудомісткості стадії в загальній трудомісткості розробки ПП

Код стадії	**Ступінь новизни**		
	А	Б	В
ТЗ (L_1)	0,15	0,12	0,12
ТП (L_2)	0,16	0,15	0,11
РП (L_3)	0,55	0,58	0,61

У таблиці 2.4 наведено значення поправочного коефіцієнта новизни (Кн), який враховується при розрахунку техніко-економічних показників програмного продукту. Коефіцієнт Кн відображає ступінь оригінальності або інноваційності розробки.

Таблиця 2.4. Значення поправочного коефіцієнта, що враховує ступінь новизни

Код ступеня новизни	**Ступінь новизни**	**Значення $К_н$**
А	Принципово нове ПП	1,75 – 1,2
Б	ПП – розвиток визначеного параметричного ряду	1,0 – 0,8
В	ПП маючий аналог	0,7

У таблиці 2.5 представлено значення коефіцієнта ступеня використання типових програм (Кт) у процесі розробки програмного продукту. Цей коефіцієнт враховує, наскільки велика частка функціональності майбутнього програмного забезпечення реалізується за допомогою вже наявних, повторно використаних або стандартних рішень.

Чим більша частина функцій покривається типовими програмами, тим менше трудових витрат потребує нова розробка, і, відповідно, значення коефіцієнта Кт є нижчим

Таблиця 2.5. Значення коефіцієнта ступеня використання в розробці типових програм

Ступінь охоплення реалізованих функцій розроблювального ПП типовими програмами, %	**Значення $К_т$**
60 і вище	0,6
40-60	0,7
20-40	0,8
До 20	0,9

Розрахунок трудомісткісті по кожному етапу окремо:

1. Трудомісткість технічного завдання за формулою 2.1

$$T_{ТЗ}=T^{a}p\times L_1\times K_H=226,4\times 0,12\times 0,7=19,02 \text{ (люд.-годин);}$$

2. Трудомісткість розробки технічного проєкту за формулою 2.2

$$T_{ТП}=T^{a}p\times L_2\times K_H=226,4\times 0,11\times 0,7=17,43 \text{ (люд.-годин);}$$

3. Трудомісткість розробки робочого проєкту за формулою 2.3

$$T_{РП}=T^{a}p\times L_3\times K_H\times K_T=226,4\times 0,61\times 0,7\times 0,7=67,67 \text{ (люд.-годин).}$$

Загальний розрахунок трудомісткості ПП зведений у табл. 2.6.

Таблиця 2.6. Розрахунок трудомісткості ПП

Найменування етапів	**Розрахунок, години**		
	Розробка ПП	Контроль керівника	Нормоконтроль
1.Розробка ТЗ	$T_{РТЗ}=19,02$	$T_{КК}=0,7\times N_{ТЗ}=0,7\times 2=1,4$	$T_{НК}=0,15\times N_{ТЗ}=0,15\times 2=0,30$
2.Розробка ТП	$T_{РТП}=17,43$	$T_{КК}=0,7\times N_{ТП}=0,7\times 50=35$	$T_{НК}=0,15\times N_{ТП}=0,15\times 50=7,5$
3.Розробка РП	$T_{РРП}=67,67$	$T_{КК}=0,7\times N_{РП}=0,7\times 8=5,6$	$T_{НК}=0,15\times N_{РП}=0,15\times 8=1,2$
4.Розробка ПП	$T_{ПЗ}=1,5\times Nпз=1,5\times 20=30,0$	$T_{КК}=0,7\times N_{ТЗ}=0,7\times 20=14,00$	$T_{НК}=0,15\times N_{ПЗ}=0,15\times 20=3,00$
Усього Т, в т.ч.:	202,12		
- на розробку	$Tр=171,80$		
- контроль		$T_{КК}=20,21$	
- нормоконтроль			$T_{НК}=10,11$

На основі таблиці розрахуємо тривалість розробки в роках за формулою 2.4:

$$Tпп=T/(8.0\times 0.73\times 360) \qquad (2.4)$$

Де 8,0 – тривалість робочого дня;

0,73 – коефіцієнт перекладу в календарні дні;

$$Tпп=202,12/(8.0\times 0.73\times 360)=0.096 \text{ р}$$

2.2 Розрахунок ціни програмного продукту

У цьому розділі для визначення ціни розраховуємо основну заробітну плату виконавців, матеріальні витрати, вартість машино – години і витрати на розробку

ПП. Відповідно до статті 8 «Закону про Державний бюджет України на 2025» встановлено мінімальну заробітну плату у місячному розмірі з 1 січня 2025 року – 8000 гривень; мінімальну погодинну тарифну ставку – 46 грн.

Таблиця 2.7 Розрахунок основної заробітної плати виконавців.

Найменування робіт	**Трудомісткість робіт, години**	**Погодинна тарифна ставка, грн.**	**Розрахунок, грн.**
1.Розробка ПП	171,80	50,00	8590
2.Контроль керівника	20,21	80,00	1616,80
3.Нормоконтроль	10,11	80,00	808,80
	-	Всього	$З_О$= 11015,60

Для оцінки витрат на документаційне забезпечення проєкту було визначено орієнтовну кількість сторінок, необхідних для кожного етапу розробки. До уваги бралися основні складові, такі як технічне завдання, технічний проєкт, робочий проєкт і пояснювальна записка. Узагальнені дані наведено в табл. 2.8.

Таблиця 2.8. Розрахунок матеріальних витрат на розробку ПО

Найменування матеріальних витрат	**Тип, модель**	**Кількість**	**Ціна одиниці, грн.**	**Вартість, грн.**
Папір	Лист А4	79	5.00	395
Транспортно – заготівельні Витрати (10%)				$Втр\text{-}з = 0,1 \times Вм1$ $=0,1 \times 395=39,5$
			Всього	$Вм=Вмі+Втр\text{-}з$= 434,5

На підставі отриманих даних по окремих статтях витрат складена калькуляція планової собівартості в цілому ПП за формою, приведеною в табл. 2.9.

Таблиця 2.9. Розрахунок статей витрат планової собівартості

Стаття витрат	**Значення, грн.**	**Формула розрахунку**
1. Матеріали	434,5	$В_М$ (див. табл. 2.8)
2. Основна заробітна плата	11015,60	$З_О$ (див. табл. 2.7)
3.Додаткова заробітна плата	1101,56	$З_Д = 0,1 \times Зо = 11015,60 \times 0,1$
4.Відрахування до єдиного фонду соціального внеску	2665,77	$В_{Є.С.В.} = 0,22 \times (Зо + Зд) = 0,22 \times (11015,60+1101,56)$
5. Накладні витрати	4406,24	$Внак. = 0,4 \times Зо = 0,4 \times 11015,60$
6. Повна собівартість	19623,67	$С_{ПОВ}$= $В_М$ +$З_О$ + $З_Д$ +Вє.с.в. + Внак.= 434,5+11015,60+1101,56+2665,77+4406,24

Розмір прибутку, що включається в ціну, визначаємо по наступній формулі 2.4:

$$П = (С_{ПОВ} \times Р)/100 \quad (2.5)$$

$$П = (19623,67 \times 10)/100 = 1962,36 \text{ грн}$$

Де р – плановий рівень рентабельності (10-20%);

Оптова ціна визначається по формулі 2.5:

$$Ц_О = С_П + П \quad (2.6)$$

$$Ц_О = 19623,67 + 1962,36 = 21586,03 \text{ грн}$$

Розрахунок податку на додану вартість за формулою 2.7:

$$ПДВ = Ц_О \times 0,2 \quad (2.7)$$

Виходячи з отриманих даних, ціна реалізації розробленого програмного продукту на основі наступної формули, становитиме:

$$Ц_Р = Ц_О + ПДВ \quad (2.8)$$

$$Ц_Р = 21586,03 + 21586,03 \times 0,2 = 25903,27 \text{ грн}$$

3 РОЗДІЛ ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

Сучасний розвиток технічного та технологічного стану виробництва передбачає постійну автоматизацію та оптимізацію виробничих процесів. Сьогодні, напевно, важко уявити компанію, господарська діяльність в якій здійснювалась б без використання комп'ютерної техніки. Через масовий характер робіт, що виконуються працівниками за допомогою комп'ютера, законодавством України чітко врегульовано норми та вимоги до використання комп'ютерної техніки на підприємстві, безпосередньо й охорона праці при роботі з комп'ютером. В розділі охорони праці дипломного проекту розглядається питання розробки мобільного додатку для обліку особистих фінансів. Тому об'єктом дослідження беремо безпеку праці на робочому місці програміста.

3.1 Аналіз умов праці й забезпечення безпеки при виконання основних видів робіт на об'єкті дипломного проектування

Аналіз умов праці показує, що на працівників можуть негативно впливати наступні фізичні та психофізіологічні фактори:

- підвищені або знижені температура, вологість повітря робочої зони;
- недостатня освітленість робочого місця;
- підвищений рівень шуму на робочому місці;
- підвищені іонізація повітря та рівень електромагнітних випромінювань;
- нервово-психічні та фізичні перевантаження.

3.2 Гігієнічні вимоги до виробничого

Приміщення, призначені для встановлення та експлуатації комп'ютерної техніки, мають відповідати проектним рішенням будівлі. Роботодавець зобов'язаний забезпечити дотримання санітарних норм освітлення, оптимальні параметри мікроклімату – температуру, відносну вологість, допустимий рівень вібрації та шуму, а також вимоги щодо вогнестійкості. Крім того, слід контролювати інтенсивність електромагнітних, ультрафіолетових та інфрачервоних полів.

Розміщувати комп'ютери у підвальних приміщеннях забороняється. Щоб

запобігти аваріям і коротким замиканням, у кімнатах, розташованих над або під робочими зонами з комп'ютерною технікою, не дозволено виконувати технологічні процеси з надмірною вологістю.

3.2.1 Гігієнічні вимоги до параметрів повітря приміщень із ПК

Параметри повітряного довкілля у приміщеннях, де працюють із персональними комп'ютерами, регламентуються вимогами ДСанПіН 3.3.2-007-98. Оскільки така діяльність належить до легких робіт категорій 1а та 1б, варто підтримувати мікрокліматичні показники в межах встановлених норм.

Гранично допустимі величини температури, відносної вологості та швидкості руху повітря для залів із ВДТ, ЕОМ і ПЕОМ подані в таблиці 3.1.

Таблиця 3.1. Норми мікроклімату для приміщень з ВДТ ЕОМ та ПЕМ

Пора року	**Категорія робіт**	**Температура повітря, ^{0}С, не більше**	**Відносна вологість повітря %**	**Швидкість руху повітря, м/с**
Холодна	Легка-1а	22-24	40-60	0,1
	Легка-1б	21-23	40-60	0,1
Тепла	Легка-1а	23-25	40-60	0,1
	Легка-1б	22-24	40-60	0,1

Рівні позитивних і негативних іонів у повітрі приміщень з ВДТ мають відповідати санітарно-гігієнічним нормам № 2152-80.

У таблиці 3.2 наведено санітарно-гігієнічні нормам № 2152-80.

Таблиця 3.2. Санітарно-гігієнічні нормам № 2152-80

Рівні	**Число іонів в 1 см^3 повітря**	**Число іонів в 1 см^3 повітря**
	n+	n-
Мінімально необхідні	400	600
Оптимальні	1500-3000	3000-5000
Максимально допустимі	50000	50000

Щоб підтримувати в приміщенні повітряний склад, який відповідає санітарним нормам, та своєчасно видаляти шкідливі домішки, застосовують вентиляцію. Природний повітрообмін через вікна ґрунтується на різниці

температур, однак залежність від погоди й нерівномірність потоків роблять його малоефективним. Тому в дипломному проєкті передбачено монтаж механізованої припливно-витяжної вентиляції з можливістю додаткового кондиціювання.

3.2.2 Виробниче освітлення

Освітлення в приміщенні поділяється на природне та штучне. Природне освітлення надходить збоку через віконні прорізи. Для забезпечення штучного освітлення використовуються люмінесцентні лампи, які мають низку переваг над традиційними лампами розжарювання: вони характеризуються спектром, наближеним до природного денного світла, забезпечують вищу світлову віддачу та довший термін експлуатації. Рівень освітленості на робочих місцях повинен відповідати нормі 300-500 лк.

3.2.3 Електробезпека та організація робочого місця

Сила струму, що проходить через тіло людини, визначається прикладеною напругою та електричним опором ділянки тіла, з якою виникає контакт. Джерелом живлення є мережа змінного струму з напругою 220 В, яка підпадає під дію вимог ГОСТ 25861-83.

Щоб запобігти ураженню електричним струмом, необхідно суворо дотримуватись правил виконання електромонтажних робіт і технічної експлуатації обладнання. Слід виключити доступ користувача до струмоведучих елементів, що перебувають під небезпечною напругою, а також до неізольованих частин пристроїв, які працюють при низькій напрузі, але не мають підключення до захисного заземлення. Підключення персонального комп’ютера до електромережі має здійснюватися через розетку зі спеціальною вилкою, обладнаною заземлювальним контактом.

Облаштування та організація робочого місця з використанням відеодисплейних терміналів повинні відповідати ергономічним нормам, передбаченим ДСанПіН 3.3.2-007-98, з урахуванням специфіки виконуваних завдань і характеру трудової діяльності. Усі елементи робочого місця — сидіння, засоби керування та пристрої відображення інформації — мають бути розміщені з

дотриманням антропометричних, фізіологічних і психофізіологічних вимог.

Конструкція офісних меблів повинна забезпечувати можливість індивідуального регулювання параметрів — висоти, кута нахилу, глибини тощо – для підтримання зручної та безпечної робочої пози. Поверхня робочого столу повинна мати матове покриття, щоб уникати відблисків. Монітор розміщується так, щоб його верхній край знаходився на рівні очей користувача, на відстані приблизно 70 см, що відповідає нормативному діапазону 60–90 см. Частота оновлення екрана становить 100 Гц, що перевищує мінімально допустимий рівень у 70 Гц і сприяє зменшенню втомлюваності зору. На рисунку 3.1 зображено конструкцію робочого місця користувача ПК.

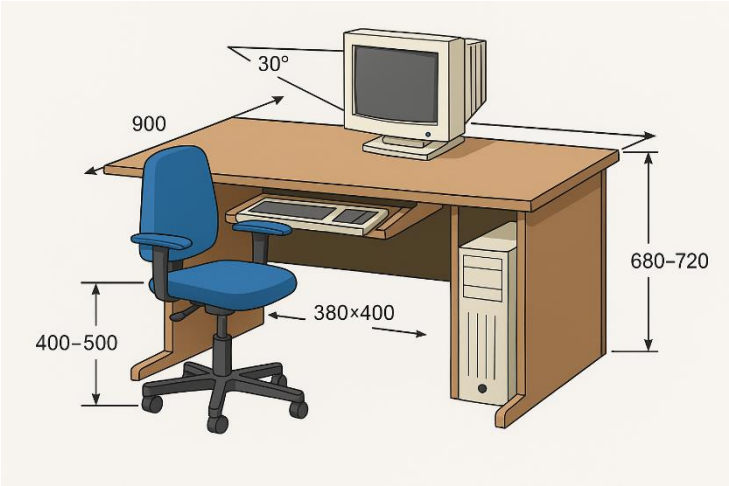


Рисунок 3.1. Конструкція робочого місця

Для зменшення нервово-емоційного навантаження, втомленості, покращення мозкового кровообігу, подолання небажаних наслідків гіподинамії, уникнення втоми доцільно впроваджувати виконання комплексу вправ, що наведені у Державних санітарних правилах і нормах роботи з візуальними терміналами електронно-обчислювальних машин ДСанПіН 3.3.2.007-98.

3.3 Пожежна безпека

Пожежна безпека – це комплекс державних і громадських заходів, що мають на меті захист людей і майна від впливу вогню. Вимоги до пожежної безпеки приміщень з електричними мережами визначені ГОСТ 12.1.033-81 і ГОСТ

12.1.004-85. Робоче місце програміста повинно бути розміщене в приміщенні, що належить до категорії Д пожежної безпеки (використовуються негорючі матеріали в холодному стані).

Усі приміщення мають бути оснащені первинними засобами пожежогасіння: внутрішнім пожежним водопроводом (пожежні крани ПК), пожежними щитами з комплектом інструментів, вуглекислотними або порошковими вогнегасниками. У разі виникнення пожежі необхідно вимкнути електроживлення, зателефонувати за номером 101, евакуювати людей згідно з планом евакуації та розпочати гасіння пожежі.

Будівлі оснащені пожежними щитами з інструментами, поруч із якими розміщено бочки з водою й ящики з піском.

Виробничі приміщення повинні мати запасні виходи, а на дверях має бути світловий напис «Запасний вихід». План евакуації розміщується у доступному місці біля основного виходу з приміщення.

ВИСНОВКИ

У дипломному проєкті було розроблено повнофункціональний вебдодаток «Storo» для хмарного зберігання та керування файлами з автентифікацією за допомогою одноразового паролю. Кожному користувачу надається 2 ГБ, додаток підтримує завантаження файлів різних форматів обсягом до 50 МБ кожний, дає змогу переглядати їхній вміст, виконувати пошук за назвою, сортувати за датою й розміром, а також об’єднує файли за категоріями «Документи», «Зображення», «Медіа» чи «Інше». Кожен файл можна перейменувати, видалити, завантажити на локальний пристрій або надати до нього спільний доступ через введення електронної пошти отримувача. Інтерфейс відображає детальну інформацію про файл, включно з метаданими, мініатюрами, розміром і датою.

Технологічний стек включає клієнтську частину на Next.js (React) і TypeScript із використанням бібліотек Tailwind CSS і ShadCN UI, серверну частину на Appwrite із serverless-функціями Next.js, розгортання на Vercel з впровадженою мережу доставки контенту (CDN) для оптимізації швидкості завантаження файлів.

Архітектура проєкту на базі Next.js і Appwrite у поєднанні з CDN від Vercel забезпечує обробку до 100 одночасних сесій без помітного зниження продуктивності. Завдяки serverless-функціям, автоматичному масштабуванню екземплярів та кешуванню статичного контенту система без проблем витримує пікові навантаження користувачів. Безпеку реалізовано багаторівневими заходами. Автентифікація відбувається за одноразовими кодами (OTP) на основі Appwrite, при цьому жодні постійні паролі не зберігаються, секрет сесії Appwrite зберігається виключно в «HttpOnly-cookie» з атрибутами Secure, що захищає від XSS та CSRF-атак. Всі запити шифруються TLS, політика CORS налаштована на прийом звернень лише з домену фронтенд-застосунку, а HTTP-заголовки Content-Security-Policy, X-Frame-Options та X-Content-Type-Options додано згідно з рекомендаціями OWASP. Проведене автоматизоване сканування OWASP ZAP підтвердило відсутність критичних вразливостей і дало змогу виправити знайдені помилки конфігурації.

ПЕРЕЛІК ВИКОРИСТАНИХ ІНФОРМАЦІЙНИХ ДЖЕРЕЛ

1. Болож О. Огляд serverless архітектури та її переваги – Тернопіль: ТНТУ ім. І. Пулюя, 2023 . –111 с.
2. Тарнавський Ю. А. Технології захисту інформації – Київ: КПІ ім. Ігоря Сікорського, 2018. – 162 с.
3. Олійник С. Г. Інтеграція веб-додатків з використанням сучасних хмарних технологій – Львів: Видавництво ЛНУ, 2019. – 240 с.
4. Pawar P., Naik M., Choudhari M., Tonge S., Adhav P. Analysis of Cloud Storage – International Journal of Research and Analytical Reviews (IJRAR). – May 2022. 362.
5. Документація Appwrite. – [Електронний ресурс]. – Режим доступу: https://appwrite.io/docs.
6. Документація Vercel – [Електронний ресурс]. – Режим доступу: https://vercel.com/klemjls-projects.
7. Документація Next.js – [Електронний ресурс]. – Режим доступу: https://nextjs.org/docs.
8. OWASP ZAP [Вебсайт] URL: https://www.zaproxy.org/docs.
9. TypeScript [Вебсайт] URL: https://www.typescriptlang.org/docs/.
10. Backend as a Service (BaaS). GeeksForGeeks. – [Електронний ресурс]. – Режим доступу: https://www.geeksforgeeks.org/backend-as-a-service-baas/.
11. Одноразовий пароль та інші алгоритми аутентифікації: як вони використовуються в цифровій безпеці. – [Електронний ресурс]. – Режим доступу: https://introserv.com/ua/blog/digital-security-and-one-time-password-algorithms/.
12. Тестування безпеки. – [Електронний ресурс]. – Режим доступу: https://qalight.ua/baza-znaniy/testuvannya-bezpeki/.

ДОДАТОК А. Програмний код основної логіки додатку Storo

```
export const appwriteConfig = {
  endpointUrl: process.env.NEXT_PUBLIC_APPWRITE_ENDPOINT!,
  projectId: process.env.NEXT_PUBLIC_APPWRITE_PROJECT!,
  databaseId: process.env.NEXT_PUBLIC_APPWRITE_DATABASE!,
  usersCollectionId: process.env.NEXT_PUBLIC_APPWRITE_USERS_COLLECTION!,
  filesCollectionId: process.env.NEXT_PUBLIC_APPWRITE_FILES_COLLECTION!,
  bucketId: process.env.NEXT_PUBLIC_APPWRITE_BUCKET!,
  secretKey: process.env.NEXT_APPWRITE_KEY!,
};
"use server";

import { Account, Client, Databases, Avatars, Storage } from "node-
appwrite";
import { appwriteConfig } from "@/lib/appwrite/config";
import { cookies } from "next/headers";

export const createSessionClient = async () => {
  const client = new Client()
    .setEndpoint(appwriteConfig.endpointUrl)
    .setProject(appwriteConfig.projectId);
  const session = (await cookies()).get("appwrite-session");
  if (!session || !session.value) throw new Error("No session");
  client.setSession(session.value);
  return {
    get account() {
      return new Account(client);
    },
    get databases() {
      return new Databases(client);
    },
  };
};
export const createAdminClient = async () => {
  const client = new Client()
    .setEndpoint(appwriteConfig.endpointUrl)
    .setProject(appwriteConfig.projectId)
    .setKey(appwriteConfig.secretKey);
  return {
    get account() {
      return new Account(client);
    },
    get databases() {
      return new Databases(client);
    },
    get storage() {
      return new Storage(client);
    },
    get avatars() {
      return new Avatars(client);
    },
  };
};

"use server";
```

```
import { createAdminClient, createSessionClient } from "@/lib/appwrite";
import { appwriteConfig } from "@/lib/appwrite/config";
import { Query, ID } from "node-appwrite";
import { parseStringify } from "@/lib/utils";
import { cookies } from "next/headers";
import { avatarPlaceholderUrl } from "@/constants";
import { redirect } from "next/navigation";

const getUserByEmail = async (email: string) => {
  const { databases } = await createAdminClient();
  const result = await databases.listDocuments(
    appwriteConfig.databaseId,
    appwriteConfig.usersCollectionId,
    [Query.equal("email", [email])],
  );
  return result.total > 0 ? result.documents[0] : null;
};
const handleError = (error: unknown, message: string) => {
  console.log(error, message);
  throw error;
};
export const sendEmailOTP = async ({ email }: { email: string }) => {
  const { account } = await createAdminClient();
  try {
    const session = await account.createEmailToken(ID.unique(), email);
    return session.userId;
  } catch (error) {
    handleError(error, "Не вдалося надіслати одноразовий код на пошту");
  }
};
export const createAccount = async ({
  fullName,
  email,
}: {
  fullName: string;
  email: string;
}) => {
  const existingUser = await getUserByEmail(email);
  const accountId = await sendEmailOTP({ email });
  if (!accountId) throw new Error("Не вдалося надіслати одноразовий код");

  if (!existingUser) {
    const { databases } = await createAdminClient();
    await databases.createDocument(
      appwriteConfig.databaseId,
      appwriteConfig.usersCollectionId,
      ID.unique(),
      {
        fullName,
        email,
        avatar: avatarPlaceholderUrl,
        accountId,
      },
    );
  }
  return parseStringify({ accountId });
};
export const verifySecret = async ({
```

```
  accountId,
  password,
}: {
  accountId: string;
  password: string;
}) => {
  try {
    const { account } = await createAdminClient();
    const session = await account.createSession(accountId, password);
    (await cookies()).set("appwrite-session", session.secret, {
      path: "/",
      httpOnly: true,
      sameSite: "strict",
      secure: true,
    });
    return parseStringify({ sessionId: session.$id });
  } catch (error) {
    handleError(error, "Не вдалося перевірити одноразовий код");
  }
};
export const getCurrentUser = async () => {
  try {
    const { databases, account } = await createSessionClient();
    const result = await account.get();
    const user = await databases.listDocuments(
      appwriteConfig.databaseId,
      appwriteConfig.usersCollectionId,
      [Query.equal("accountId", result.$id)],
    );
    if (user.total <= 0) return null;
    return parseStringify(user.documents[0]);
  } catch (error) {
    console.log(error);
  }
};
export const signOutUser = async () => {
  const { account } = await createSessionClient();
  try {
    await account.deleteSession("current");
    (await cookies()).delete("appwrite-session");
  } catch (error) {
    handleError(error, "Не вдалося виконати вихід з системи");
  } finally {
    redirect("/sign-in");
  }
};
export const signInUser = async ({ email }: { email: string }) => {
  try {
    const existingUser = await getUserByEmail(email);
    if (existingUser) {
      await sendEmailOTP({ email });
      return parseStringify({ accountId: existingUser.accountId });
    }
    return parseStringify({
      accountId: null,
      error: "Користувача не знайдено",
    });
  } catch (error) {
```

```
    handleError(error, "Не вдалося виконати вхід в систему");
  }
};

"use server";

import { createAdminClient, createSessionClient } from "@/lib/appwrite";
import { InputFile } from "node-appwrite/file";
import { appwriteConfig } from "@/lib/appwrite/config";
import { ID, Models, Query } from "node-appwrite";
import { constructFileUrl, getFileType, parseStringify } from
"@/lib/utils";
import { revalidatePath } from "next/cache";
import { getCurrentUser } from "@/lib/actions/user.actions";

const handleError = (error: unknown, message: string) => {
  console.log(error, message);
  throw error;
};
export const uploadFile = async ({
  file,
  ownerId,
  accountId,
  path,
}: UploadFileProps) => {
  const { storage, databases } = await createAdminClient();
  try {
    const inputFile = InputFile.fromBuffer(file, file.name);
    const bucketFile = await storage.createFile(
      appwriteConfig.bucketId,
      ID.unique(),
      inputFile,
    );
    const fileDocument = {
      type: getFileType(bucketFile.name).type,
      name: bucketFile.name,
      url: constructFileUrl(bucketFile.$id),
      extension: getFileType(bucketFile.name).extension,
      size: bucketFile.sizeOriginal,
      owner: ownerId,
      accountId,
      users: [],
      bucketFileId: bucketFile.$id,
    };
    const newFile = await databases
      .createDocument(
        appwriteConfig.databaseId,
        appwriteConfig.filesCollectionId,
        ID.unique(),
        fileDocument,
      )
      .catch(async (error: unknown) => {
        await storage.deleteFile(appwriteConfig.bucketId, bucketFile.$id);
        handleError(error, "Не вдалося створити запис про файл у базі
даних");
      });
    revalidatePath(path);
    return parseStringify(newFile);
```

```
  } catch (error) {
    handleError(error, "Помилка під час завантаження файлу");
  }
};
const createQueries = (
  currentUser: Models.Document,
  types: string[],
  searchText: string,
  sort: string,
  limit?: number,
) => {
  const queries = [
    Query.or([
      Query.equal("owner", [currentUser.$id]),
      Query.contains("users", [currentUser.email]),
    ]),
  ];
  if (types.length > 0) queries.push(Query.equal("type", types));
  if (searchText) queries.push(Query.contains("name", searchText));
  if (limit) queries.push(Query.limit(limit));
  if (sort) {
    const [sortBy, orderBy] = sort.split("-");
    queries.push(
      orderBy === "asc" ? Query.orderAsc(sortBy) :
Query.orderDesc(sortBy),
    );
  }
  return queries;
};
export const getFiles = async ({
  types = [],
  searchText = "",
  sort = "$createdAt-desc",
  limit,
}: GetFilesProps) => {
  const { databases } = await createAdminClient();
  try {
    const currentUser = await getCurrentUser();
    if (!currentUser) throw new Error("Користувача не знайдено");
    const queries = createQueries(currentUser, types, searchText, sort,
limit);
    const files = await databases.listDocuments(
      appwriteConfig.databaseId,
      appwriteConfig.filesCollectionId,
      queries,
    );
    console.log({ files });
    return parseStringify(files);
  } catch (error) {
    handleError(error, "Помилка під час отримання списку файлів");
  }
};
export const renameFile = async ({
  fileId,
  name,
  extension,
  path,
}: RenameFileProps) => {
```

```
  const { databases } = await createAdminClient();
  try {
    const newName = `${name}.${extension}`;
    const updatedFile = await databases.updateDocument(
      appwriteConfig.databaseId,
      appwriteConfig.filesCollectionId,
      fileId,
      {
        name: newName,
      },
    );
    revalidatePath(path);
    return parseStringify(updatedFile);
  } catch (error) {
    handleError(error, "Помилка під час перейменування файлу");
  }
};
export const updateFileUsers = async ({
  fileId,
  emails,
  path,
}: UpdateFileUsersProps) => {
  const { databases } = await createAdminClient();
  try {
    const updatedFile = await databases.updateDocument(
      appwriteConfig.databaseId,
      appwriteConfig.filesCollectionId,
      fileId,
      {
        users: emails,
      },
    );
    revalidatePath(path);
    return parseStringify(updatedFile);
  } catch (error) {
    handleError(error, "Помилка під час оновлення списку користувачів
файлу");
  }
};
export const deleteFile = async ({
  fileId,
  bucketFileId,
  path,
}: DeleteFileProps) => {
  const { databases, storage } = await createAdminClient();
  try {
    const deletedFile = await databases.deleteDocument(
      appwriteConfig.databaseId,
      appwriteConfig.filesCollectionId,
      fileId,
    );
    if (deletedFile) {
      await storage.deleteFile(appwriteConfig.bucketId, bucketFileId);
    }
    revalidatePath(path);
    return parseStringify({ status: "success" });
  } catch (error) {
    handleError(error, "Помилка під час видалення файлу");
```

```
  }
};
export async function getTotalSpaceUsed() {
  try {
    const { databases } = await createSessionClient();
    const currentUser = await getCurrentUser();
    if (!currentUser) throw new Error("Користувач не автентифікований");
    const files = await databases.listDocuments(
      appwriteConfig.databaseId,
      appwriteConfig.filesCollectionId,
      [Query.equal("owner", [currentUser.$id])],
    );
    const totalSpace = {
      image: { size: 0, latestDate: "" },
      document: { size: 0, latestDate: "" },
      video: { size: 0, latestDate: "" },
      audio: { size: 0, latestDate: "" },
      other: { size: 0, latestDate: "" },
      used: 0,
      all: 2 * 1024 * 1024 * 1024 /* 2GB available bucket storage */,
    };
    files.documents.forEach((file) => {
      const fileType = file.type as FileType;
      totalSpace[fileType].size += file.size;
      totalSpace.used += file.size;
      if (
        !totalSpace[fileType].latestDate ||
        new Date(file.$updatedAt) > new
Date(totalSpace[fileType].latestDate)
      ) {
        totalSpace[fileType].latestDate = file.$updatedAt;
      }
    });
    return parseStringify(totalSpace);
  } catch (error) {
    handleError(error, "Помилка під час підрахунку використаного
простору:, ");
  }
}
import type { NextConfig } from "next";
const ContentSecurityPolicy = `
  default-src 'self';
  script-src 'self' 'unsafe-inline';
  style-src 'self' 'unsafe-inline';
  img-src 'self' data: blob: https://cdn.pixabay.com
https://img.freepik.com https://cloud.appwrite.io;
  font-src 'self';
  connect-src 'self' https://fra.cloud.appwrite.io/v1
wss://fra.cloud.appwrite.io/v1;
  base-uri 'self';
  form-action 'self';
  object-src 'none';
  frame-ancestors 'none';
  upgrade-insecure-requests;
`;
const nextConfig: NextConfig = {
  typescript: {
    ignoreBuildErrors: true,
```

```
  },
  eslint: {
    ignoreDuringBuilds: true,
  },
  experimental: {
    serverActions: {
      bodySizeLimit: "50MB",
    },
  },
  images: {
    remotePatterns: [
      {
        protocol: "https",
        hostname: "cdn.pixabay.com",
      },
      {
        protocol: "https",
        hostname: "img.freepik.com",
      },
      {
        protocol: "https",
        hostname: "cloud.appwrite.io",
      },
    ],
  },
  async headers() {
    return [
      {
        source: "/:path*",
        headers: [
          {
            key: "Content-Security-Policy",
            value: ContentSecurityPolicy.replace(/\s{2,}/g, " ").trim(),
          },
          {
            key: "X-Frame-Options",
            value: "DENY",
          },
          {
            key: "X-Content-Type-Options",
            value: "nosniff",
          },
        ],
      },
    ];
  },
};
export default nextConfig;
```

ДОДАТОК Б. Сторінки мультимедійної презентації

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП “ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ”

Спеціальність: 123 «Комп’ютерна інженерія»
Освітньо-професійна програма: «Безпека комп’ютерних систем і мереж»
Група: 4КБ-02

Дипломний проєкт

здобувачки освіти денної форми навчання
КБ.02.06.000.ДП

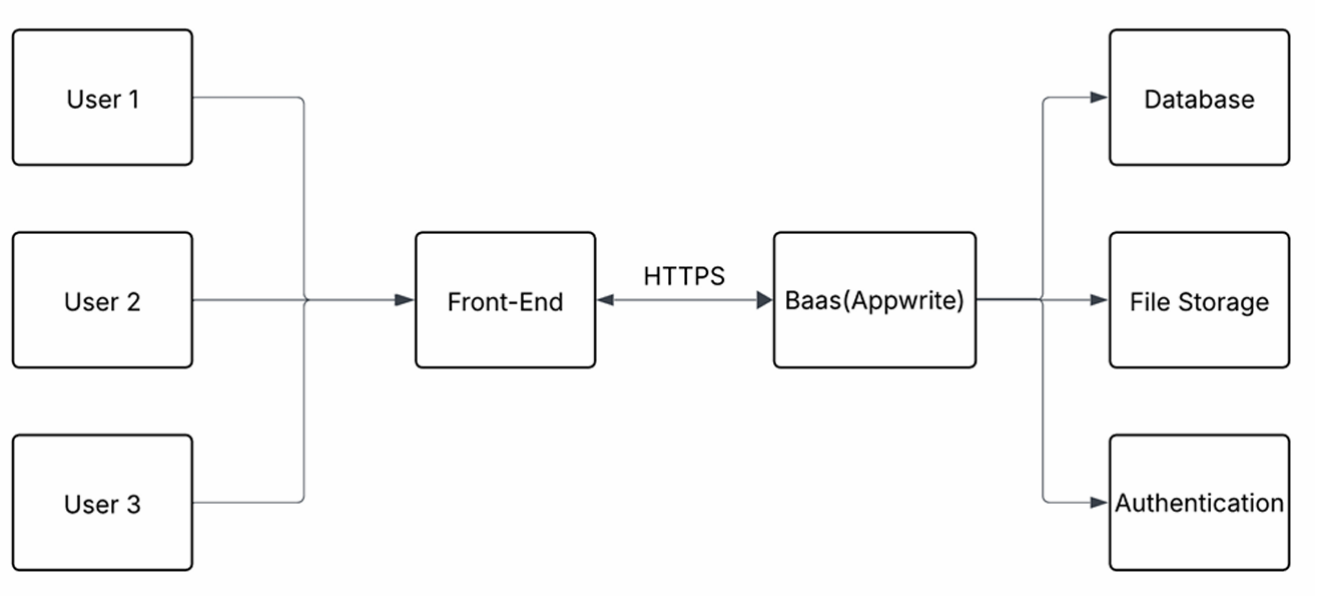
КЛЕМЕНЧУК
ЮЛІЇ СЕРГІЇВНИ

“Розробка додатку для керування файловим сховищем з автентифікацією користувачів в глобальній мережі”

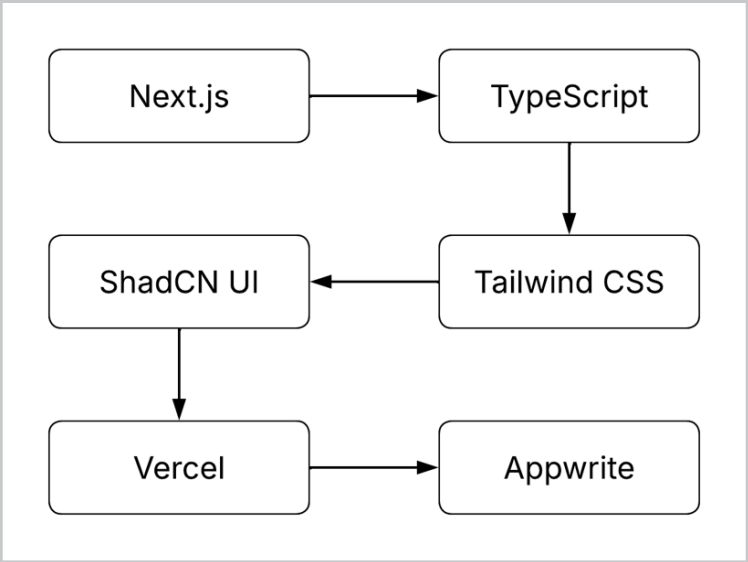
АКТУАЛЬНІСТЬ ТА МЕТА

- Актуальність теми полягає у забезпеченні безпечного та масштабованого зберігання файлів для індивідуальних користувачів без складної інфраструктурної підтримки
- Метою роботи є розробка безсерверного додатку з автентифікацією одноразовим паролем, який відповідатиме сучасним вимогам безпеки, продуктивності й зручності.

УЗАГАЛЬНЕНА БЛОК-СХЕМА АРХІТЕКТУРИ
БЕЗСЕРВЕРНОГО ДОДАТКУ З ВИКОРИСТАННЯМ
BACKEND AS A SERVICE

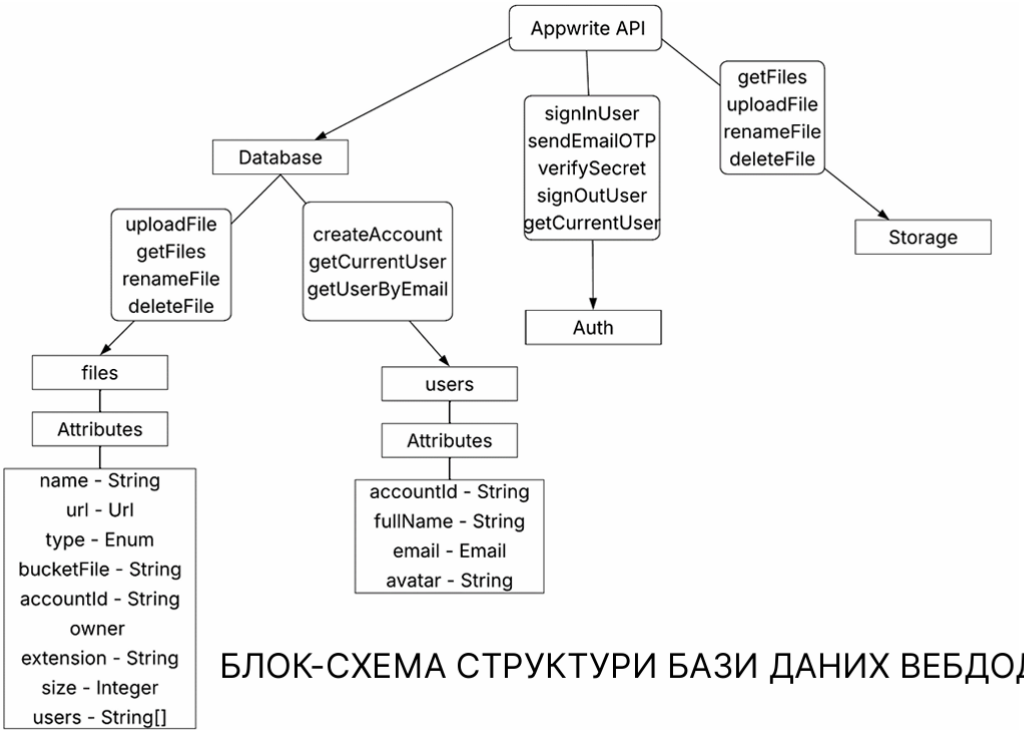


ЗАСОБИ РОЗРОБКИ ВЕБДОДАТКУ



СТРУКТУРНА ОРГАНІЗАЦІЯ ДИРЕКТОРІЙ І ФАЙЛІВ ДОДАТКА STORO

```
Storo/
├── app/ [Клієнт]
│   ├── layout.tsx [Клієнт] - головний макет
│   ├── globals.css [Клієнт] - глобальні стилі
│   ├── page.tsx [Сервер] - Dashboard
│   └── [type]/ [Сервер] - динамічний сегмент
│       └── page.tsx [Сервер] - список файлів за типом
├── components/ [Клієнт] - UI-компоненти
│   ├── Sort.tsx [Клієнт] - сортування
│   ├── Card.tsx [Клієнт] - картка файлу
│   ├── Chart.tsx [Клієнт] - діаграма
│   ├── FileUploader.tsx [Клієнт] - завантажувач файлів
│   ├── Header.tsx [Клієнт] - заголовок
│   ├── Search.tsx [Клієнт] - пошук
│   ├── ActionDropdown.tsx [Клієнт] - меню дій
│   ├── MobileNavigation.tsx [Клієнт] - мобільна навігація
│   ├── Thumbnail.tsx [Клієнт] - мініатюра файлу
│   ├── FormattedDateTime.tsx [Клієнт] - форматована дата
│   └── ui/ [Клієнт] - бібліотека UI-елементів
│       ├── button.tsx [Клієнт]
│       ├── input.tsx [Клієнт]
│       ├── label.tsx [Клієнт]
│       ├── dialog.tsx [Клієнт]
│       ├── dropdown-menu.tsx [Клієнт]
│       ├── sheet.tsx [Клієнт]
│       ├── separator.tsx [Клієнт]
│       ├── card.tsx [Клієнт]
│       ├── chart.tsx [Клієнт]
├── constants/ [Клієнт] - статичні дані
│   └── index.ts [Клієнт] - навігаційні елементи
├── hooks/ [Клієнт] -
│   └── use-toast.ts [Клієнт] - хук для повідомлень
├── lib/ [Сервер] - серверна логіка Appwrite
│   ├── actions/ [Сервер] - дії над даними
│   │   ├── file.actions.ts [Сервер] - завантаження/отримання файлів
│   │   └── user.actions.ts [Сервер] - автентифікація
│   ├── appwrite/ [Сервер] - клієнти Appwrite
│   │   ├── index.ts [Сервер] - createSessionClient, createAdminClien
│   │   └── config.ts [Сервер] - налаштування Appwrite
│   └── utils.ts [Сервер] - утиліти
├── types/ [Клієнт] - оголошення типів TypeScript
│   └── index.d.ts [Клієнт] - типи компонентів
├── public/ [Клієнт] - статичні ресурси
│   └── assets/
│       └── icons/ [Клієнт] - SVG-іконки
├── next.config.ts [Сервер] - конфігурація Next.js
└── postcss.config.mjs [Сервер] - конфігурація PostCSS/TailwindCSS
```



БЛОК-СХЕМА СТРУКТУРИ БАЗИ ДАНИХ ВЕБДОДАТКУ STORO

ПОРІВНЯННЯ ПРОЦЕСУ ОБРОБКИ ЗАПИТУ ПРИ CSR І SSR У NEXT.JS

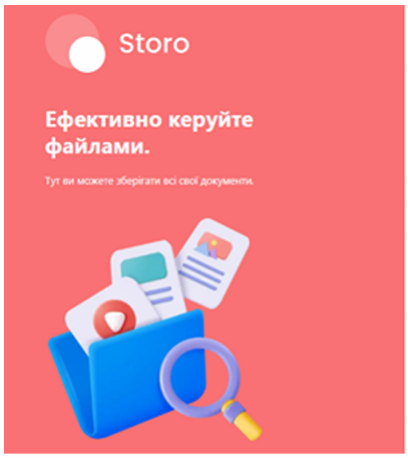

```
onst getUserByEmail : (email: string) => Promise<Document | nul... = async (email: string) : Promise<D
 const { databases } = await createAdminClient();

 const result : DocumentList<Document> = await databases.listDocuments(
   appwriteConfig.databaseId, // ID бази даних
   appwriteConfig.usersCollectionId, // колекція користувачів
   [Query.equal("email", [email])], // умова пошуку
 );
 // Повернення знайденого документа, або null якщо користувач не існує
 return result.total > 0 ? result.documents[0] : null;
;
/ Обробка помилок виводить повідомлення у консоль
onst handleError : (error: unknown, message: string) => neve... = (error: unknown, message: string)
 console.log(error, message);
 throw error;
;
```

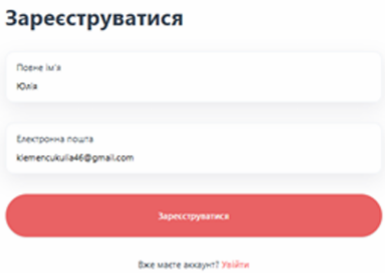
ПОШУК КОРИСТУВАЧА ЗА ЕЛЕКТРОННОЮ ПОШТОЮ У БАЗІ ДАНИХ

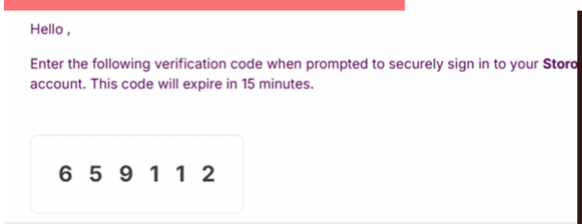
ГЕНЕРАЦІЯ OTP КОДУ

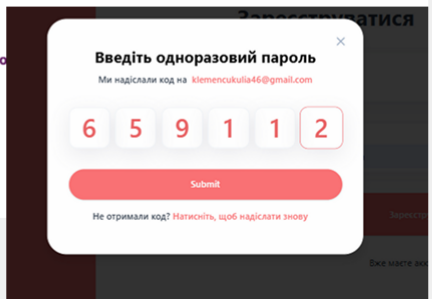
```
export const sendEmailOTP : ({ email }: { email: string }) => Promise... = async ({ email }: {
  const { account } = await createAdminClient();

  try {
    // Створення одноразовий токен (OTP) та надсилається на email
    const session : Token = await account.createEmailToken(ID.unique(), email);
    // Повернення ідентифікатора користувача (accountId)
    return session.userId;
  } catch (error) {
    // Вивод повідомлення у разі помилки
    handleError(error, "Не вдалося надіслати одноразовий код на пошту");
  }
};
```

ЛОГІКА ВХОДУ ТА РЕЄСТРАЦІЇ КОРИСТУВАЧА З ПІДТВЕРДЖЕННЯМ ЧЕРЕЗ ОДНОРАЗОВИЙ КОД (OTP)









РЕЄСТРАЦІЯ КОРИСТУВАЧА З ПІДТВЕРДЖЕННЯМ ЧЕРЕЗ ОДНОРАЗОВИЙ КОД (OTP)

```
const ContentSecurityPolicy = `
  default-src 'self';
   script-src 'self' 'unsafe-inline';
  style-src 'self' 'unsafe-inline';
  img-src 'self' data: blob: https://cdn.pixabay.com https://img.freepik.com https://
  font-src 'self';
  connect-src 'self' https://fra.cloud.appwrite.io/v1 wss://fra.cloud.appwrite.io/v1;
  base-uri 'self';
  form-action 'self';
  object-src 'none';
  frame-ancestors 'none';
  upgrade-insecure-requests;
`;
```

КОД ПОЛІТИКИ
CONTENT SECURITY POLICY ⟵

```
async headers() : Promise<{ source: string; headers: { key:... {
  return [
    {
      source: "/:path*",
      headers: [
        {
          key: "Content-Security-Policy",
          value: ContentSecurityPolicy.replace(/\s{2,}/g, " ").trim(),
        },
        {
          key: "X-Frame-Options",
          value: "DENY",
        },
        {
          key: "X-Content-Type-Options",
          value: "nosniff",
        },
```

ЗПОБІГАННЯ АТАКАМ ⟶
MIME-SNIFFING ТА “CLICKJACKING”

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

РЕЦЕНЗІЯ

на дипломний проект здобувача (здобувачки) освіти

відділення комп'ютерних систем

Клеменчук Юлії Сергіївни

(прізвище, ім`я та по батькові)

Спеціальність *123 «Комп'ютерна інженерія»*

Освітньо-професійна програма *«Безпека комп'ютерних систем і мереж»*

Керівник дипломного проекту (роботи) *Гаджиєв Матін Магсудович*

(прізвище, ім`я та по батькові)

Тема дипломного проекту (роботи) *Розробка додатку для керування файловим сховищем з автентифікацією користувачів в глобальній мережі*

Обсяг розрахунково-пояснювальної записки *81* сторінок

Обсяг графічної (презентаційної) частини *11* аркушів (слайдів)

ХАРАКТЕРИСТИКА ДИПЛОМНОГО ПРОЕКТУ (РОБОТИ)

а) заключення про ступінь відповідності виконаного дипломного проекту завданню

Представлений дипломний проєкт відповідає затвердженій темі та виконаний відповідно технічному завданню. Дипломний проект присвячений розробці додатку для керування файловим сховищем з автентифікацією користувачів в глобальній мережі і складається з пояснювальної записки та мультимедійної презентації з відповідними схемами.

б) характеристика виконання кожного розділу дипломного проекту

Пояснювальна записка складається з основного розділу, економічного розділу, розділу охорони праці та додатків. Перелічені розділи поетапно охоплюють розробку, виконані докладно та обґрунтовано.

в) оцінка якості виконання пояснювальної записки та графічної частини дипломного проекту

Графічна частина складається з 11 слайдів мультимедійної презентації, виконаної у програмному продукті MS PowerPoint, які містять структурні, принципові та функціональні схеми, структурні моделі, блок-схеми алгоритмів, передбачені технічним завданням. Пояснювальна записка виконана акуратно та у відповідності до норм. Якість виконання пояснювальної записки добра, розробку виконано у повному обсязі.

г) перелік позитивних якостей дипломного проекту

Сучасний технологічний стек – Next.js + TypeScript + Tailwind CSS + ShadCN UI + Appwrite + Vercel — вигідне поєднання для швидкої розробки, автоматичного масштабування й безсерверного розгортання. Автентифікація по Email-OTP, збереження сесії в HttpOnly-cookie, додавання CSP, X-Frame-Options, X-Content-Type-Options, сканування OWASP ZAP і виправлення вразливостей.

д) основні недоліки дипломного проекту

Незрозумілим є ліміт на завантаження 50 МБ і місце користувача 2 ГБ. Відсутнє обґрунтування вибору цих порогів з точки зору очікуваного навантаження чи ринку. Згадано підтримку сотень одночасних сесій, але відсутні конкретні замірювання часу відповіді API чи фронтенду під навантаженням.

Оцінка розрахункової частини *Добре*

Оцінка графічної частини *Відмінно*

Загальна оцінка *Відмінно*

Прізвище, ім'я, по батькові рецензента *к.т.н. Шибаєва Наталя Олегівна*

Місце роботи і посада рецензента *Національний університет «Одеська політехніка», доцент кафедри інформаційних технологій*

Підпис:

«___» _____ *2025 р.*

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

ВІДГУК

керівника на дипломний проект здобувачки (здобувача) освіти

відділення комп'ютерних систем

Клеменчук Юлії Сергіївни

(прізвище, ім`я та по батькові)

Спеціальність: *123 "Комп'ютерна інженерія"*

Освітньо-професійна програма: *«Безпека комп'ютерних систем і мереж»*

Тема дипломного проекту: *Розробка додатку для керування файловим сховищем з автентифікацією користувачів в глобальній мережі*

ХАРАКТЕРИСТИКА ДИПЛОМНОГО ПРОЕКТУ

а) обсяг і якість виконання проекту (графічного матеріалу і розрахунково-пояснювальної записки) *Дипломний проект виконано відповідно технічному завданню. Пояснювальна записка містить 81 сторінки. У пояснювальній записці наведено етапи розробки додатку для керування файловим сховищем з автентифікацією користувачів в глобальній мережі. Графічна частина складається з 11 слайдів мультимедійної презентації, які також містять блок-схеми, передбачені технічним завданням. Якість виконання пояснювальної записки та графічної частини добра, розробку виконано в повному обсязі.*

б) самостійність роботи над проектом: *Протягом всього строку дипломного проєктування та переддипломної практики здобувачка освіти Клеменчук Ю.С. поступово та послідовно виконував всі етапи розробки. Всі роботи здобувач освіти виконував самостійно, з оглядом на рекомендації керівника*

в) теоретична підготовка випускника (випускниці): *Здобувач освіти Клеменчук Ю.С. під час роботи над дипломним проєктом вивчила достатню кількість літературних джерел та матеріалів за даною тематикою.*

Вважаю, що теоретична підготовка дипломниці відмінна і вона готова до захисту дипломного проекту

г) вміння розв'язувати виробничі та конструкторські питання *Під час дипломного проєктування здобувачка освіти Кліновський Ю.С. мала змогу самостійно приймати окремі рішення з реалізації програмної частини додатку для керування файловим сховищем та показала вміння організовано працювати над поставленим завданням, розробляти структурні схеми та програмно-апаратні зв'язки із застосуванням сучасних комп'ютерних програмних засобів, таких як Next.js, TypeScript, OWASP ZAP, а також готувати презентаційні та звітні матеріали.*

Оцінка розрахункової частини *Відмінно*

Оцінка графічної частини *Відмінно*

Загальна оцінка *Відмінно*

Прізвище, ім'я, по батькові керівника дипломного проекту

д.т.н. проф. Гаджиєв Матін Магсуд-огли

Місце роботи і посада керівника дипломного проекту

зав. каф. ІПЗ ФІОТ

Підпис

«*30*» *06* 2025 р.

ДОЗВІЛ
НА РОЗМІЩЕННЯ
ВИПУСКНОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
(ДИПЛОМНОГО ПРОЕКТУ)
В ЕЛЕКТРОННОМУ РЕПОЗИТАРІЇ ВСП «ОТФК ОНТУ»

Ми, що нижче підписалися,

Клеменчук Юлія Сергіївна
здобувачки освіти гр. 4КБ-02, та

Гаджиєв Матін Магсуд-огли,
керівник дипломного проекту,

не заперечуємо щодо розміщення електронного варіанту пояснювальної записки до дипломного проекту фахового молодшого бакалавра на тему:

«Розробка додатку для керування файловим сховищем з автентифікацією в глобальній мережі» (автор роботи – Клеменчук.Ю.С. керівник роботи – Гаджиєв М.М.)

виконаного у ВСП «Одеський технічний фаховий коледж Одеського національного технологічного університету» в 2025 році, у повному обсязі в електронному репозитарії ВСП «ОТФК ОНТУ» для вільного доступу через мережу Інтернет.

Несемо відповідальність за ідентичність електронного та друкованого варіантів випускної кваліфікаційної роботи і даємо згоду на обробку персональних даних.

Виконавець _____ / Клеменчук Ю.С. /

Керівник _____ / Гаджиєв М.М. /

«16» червня 2025 р.

ДОВІДКА

циклової комісії КТ та ПІ
про допуск до захисту дипломного проєкту
здобувача (здобувачки) освіти IV курсу
відділення комп'ютерних систем групи 4КБ-02

Клеменчук Юлії Сергіївни

на тему *Розробка додатку для керування файловим сховищем з автентифікацією користувачів в глобальній мережі*

Висновок відповідальної особи за проведення нормоконтролю:
пояснювальна записка до дипломного проєкту виконана з деякими порушеннями ДСТУ та оформлена відповідно до вимог Положення про дипломне проєктування

(підпис) *16.06.2025* (дата) *Петрашова В.І.* (П.І.Б.)

Висновок відповідальної особи за перевірку роботи на наявність академічного плагіату *згідно звіту про перевірку від 16.06.2025 р. значення коефіцієнту подібності в роботі становить 16,45%, коефіцієнт цитування – 2,22%.*

(підпис) *16.06.2025* (дата) *Краснокутська К.Г.* (П.І.Б.)

Попередня експертиза (малий захист) дипломного проєкту

здобувача (здобувачки) освіти *Клеменчук Ю.С.* (П.І.Б.)

проведена «*16*» *червня* 20*25* р.

Висновки *Пояснювальна записка до дипломного проєкту виконана у повному обсязі. Випускна кваліфікаційна робота (дипломний проєкт) відповідає вимогам Положення про дипломне проєктування та рекомендована до захисту.*

Голова ЦК КТ та ПІ (підпис) *Кривченко Ю.В.* (П.І.Б.)

StrikePlagiarism.com

Дата звіту 6/16/2025
Дата редагування 6/16/2025

Документ прийнятий

Звіт подібності

метадані

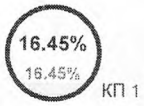
Назва організації
Odesa Technical Professional College of Odesa National University of Technology
Заголовок
Розробка додатку для керування файловим сховищем з автентифікацією користувачів в глобальній мережі

Автор
Клеменчук Юлія Сергіївна
Науковий керівник / Експерт
Гаджиєв Матин Магсудович

підрозділ
Відокремлений структурний підрозділ "Одеський технічний фаховий коледж Одеського національного технологічного університету"

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.



КП 1

КЦ

25
Довжина фрази для коефіцієнта подібності 2

14744
Кількість слів

122111
Кількість символів

Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв		19
Інтервали		0
Мікропробіли		0
Білі знаки		0
Парафрази (SmartMarks)		83

Подібності за списком джерел

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

10 найдовших фраз — Колір тексту

ПОРЯДКОВИЙ НОМЕР	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	Пояснювальна записка [redacted] 6/4/2025 Interregional Academy of Personnel Management (Інститут комп'ютерно-інформаційних технологій та дизайну)	108 0.73 %
2	https://card-file.ontu.edu.ua/server/api/core/bitstreams/a141b658-5fa7-4f90-b0bd-7f0ccaed21e5/content	83 0.56 %
3	https://card-file.ontu.edu.ua/bitstreams/53ed22ad-8700-4162-b97a-082a1ad472d6/download	80 0.54 %

4	https://card-file.ontu.edu.ua/bitstreams/549ee9fe-7574-4ae5-b500-9fe2711f33e6/download	64 0.43 %
5	https://card-file.ontu.edu.ua/bitstreams/11562741-24e6-4201-bc41-a00c8013fca1/download	63 0.43 %
6	https://card-file.ontu.edu.ua/bitstreams/5240e379-7721-49f0-8ee8-27140b0b473a/download	62 0.42 %
7	https://card-file.ontu.edu.ua/bitstreams/035f6436-20b4-4ee6-8e99-bede670e308b/download	58 0.39 %
8	https://card-file.ontu.edu.ua/bitstreams/1dff552d-7200-49b8-ae1d-ba76a1335685/download	50 0.34 %
9	https://card-file.ontu.edu.ua/bitstreams/8999d5af-6274-44f4-ae78-d23e08048d38/download	50 0.34 %
10	https://card-file.ontu.edu.ua/bitstreams/11562741-24e6-4201-bc41-a00c8013fca1/download	42 0.28 %

з домашньої бази даних (0.35 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	Розробка системи авторизації користувача на web-сервері за допомогою nrf-модулю [redacted] 6/15/2025 Odesa Technical Professional College of Odesa National University of Technology (Відокремлений структурний підрозділ "Одеський технічний фаховий коледж Одеського національного технологічного університету")	30 (2) 0.20 %
2	Розробка 3D-гри у жанрі survival-horror з налаштуваннями рівнів складності [redacted] 6/12/2025 Odesa Technical Professional College of Odesa National University of Technology (Відокремлений структурний підрозділ "Одеський технічний фаховий коледж Одеського національного технологічного університету")	11 (1) 0.07 %
3	Створення web-застосунку цифрового помічника з використанням Open AI [redacted] 5/28/2025 Odesa Technical Professional College of Odesa National University of Technology (Відокремлений структурний підрозділ "Одеський технічний фаховий коледж Одеського національного технологічного університету")	10 (1) 0.07 %

з програми обміну базами даних (0.99 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	Пояснювальна записка [redacted] 6/4/2025 **Interregional Academy of Personnel Management (Інститут комп'ютерно-інформаційних технологій та дизайну)**	**113 (2) 0.77 %**
2	Створення веб-сайту для роботи СТО [redacted] 4/28/2025 Rivne Applied College of Information Technologies (Rivne Applied College of Information Technologies)	17 (2) 0.12 %
3	YFCNU/2012/phil/phil_2012_034.pdf [redacted] 10/28/2019 Yuriy Fedkovych Chernivtsi National University(CNU) course papers (Deanery)	10 (1) 0.07 %
4	Інформаційна система управління проєктами. Клієнтська частина [redacted] 3/16/2025 National Technical University of Ukraine Igor Sikorskyi Kyiv Politech Institute (National Technical University of Ukraine Igor Sikorskyi Kyiv Politech Institute)	6 (1) 0.04 %

з Інтернету (15.12 %)

ПОРЯДКОВИЙ НОМЕР	ДЖЕРЕЛО URL	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	**https://card-file.ontu.edu.ua/bitstreams/1dff552d-7200-49b8-ae1d-ba76a1335685/download**	**862 (65) 5.85 %**

2	https://card-file.ontu.edu.ua/server/api/core/bitstreams/a141b658-5fa7-4f90-b0bd-7f0ccaed21e5/content	303 (17) 2.06 %
3	https://card-file.ontu.edu.ua/bitstreams/53ed22ad-8700-4162-b97a-082a1ad472d6/download	127 (5) 0.86 %
4	https://card-file.ontu.edu.ua/bitstreams/11562741-24e6-4201-bc41-a00c8013fca1/download	110 (3) 0.75 %
5	https://card-file.ontu.edu.ua/bitstreams/549ee9fe-7574-4ae5-b500-9fe2711f33e6/download	108 (7) 0.73 %
6	https://card-file.ontu.edu.ua/server/api/core/bitstreams/ead3fa83-2e3d-4cd7-bfbd-1d5ed04c1ce4/content	102 (5) 0.69 %
7	https://card-file.ontu.edu.ua/bitstreams/035f6436-20b4-4ee6-8e99-bede670e308b/download	77 (3) 0.52 %
8	https://card-file.ontu.edu.ua/bitstreams/bbed74c8-2ea7-44c5-8d00-0fe3fd9790ee/download	71 (9) 0.48 %
9	https://card-file.ontu.edu.ua/bitstreams/62baa43e-b968-4993-bb54-8cf8761a89b2/download	69 (6) 0.47 %
10	https://card-file.ontu.edu.ua/bitstreams/29489599-0581-4ce6-8890-c3b13d9f2e0e/download	69 (4) 0.47 %
11	https://card-file.ontu.edu.ua/bitstreams/8999d5af-6274-44f4-ae78-d23e08048d38/download	66 (2) 0.45 %
12	https://card-file.ontu.edu.ua/bitstreams/5240e379-7721-49f0-8ee8-27140b0b473a/download	62 (1) 0.42 %
13	https://www.restack.io/docs/nextjs-knowledge-nextjs-xss-vulnerabilities	50 (4) 0.34 %
14	https://card-file.ontu.edu.ua/bitstreams/0e72a3b9-bdd7-4711-a3c6-dedc1d4287cc/download	36 (4) 0.24 %
15	https://appwrite.io/threads/1285620046605058158	33 (5) 0.22 %
16	https://card-file.ontu.edu.ua/server/api/core/bitstreams/8da72e29-656f-4ee4-9b22-716dedf53ff5/content	31 (1) 0.21 %
17	https://card-file.ontu.edu.ua/server/api/core/bitstreams/3302e08a-9549-43ba-8861-728bff7dc7ff/content	14 (1) 0.09 %
18	https://lektsii.com/2-84147.html	12 (1) 0.08 %
19	https://dkng.net.ua/doc/metod.material/baranchuk/Baranchuk_Standart_pidpryjemstva.pdf.pdf	11 (1) 0.07 %
20	https://card-file.ontu.edu.ua/bitstreams/34a6756b-592f-4b77-a805-183aa03a6a26/download	10 (1) 0.07 %
21	https://card-file.ontu.edu.ua/bitstreams/6cf43324-8f08-4031-ba42-f80b18efbbc8/download	6 (1) 0.04 %

Список прийнятих фрагментів (немає прийнятих фрагментів)

ПОРЯДКОВИЙ НОМЕР	ЗМІСТ	КІЛЬКІСТЬ ОДНАКОВИХ СЛІВ (ФРАГМЕНТІВ)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: 123 «Комп'ютерна інженерія» Освітньо-професійна програма: «Безпека комп'ютерних систем і мереж»
Група: 4 КБ-02

Дипломний проєкт
здобувача освіти денної форми навчання
КБ.02.06.000.ДП

КЛЕМЕНЧУК
ЮЛІЇ СЕРГІЇВНИ

м. Одеса
2025 р. МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»