ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Кваліфікаційна робота

на тему: «**ХМАРНЕ СХОВИЩЕ НА ОСНОВІ ЗАСОБІВ NEXT.JS ТА TYPESCRIPT»**

на здобуття освітнього ступеня бакалавра
зі спеціальності __122 Комп'ютерні науки_____
(код, найменування спеціальності)
освітньо-професійної програми __Комп'ютерні науки _____
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ ___Олександр ПТУХА___
(підпис) *Ім'я, ПРІЗВИЩЕ здобувача*

Виконав: здобувач вищої освіти гр. __КНД-41_____

_____Олександр ПТУХА_____
Ім'я, ПРІЗВИЩЕ

Керівник: _доктор філософії, Юлія БЕРЕЗОВСЬКА
Науковий ступінь, вчене звання Ім'я, ПРІЗВИЩЕ

Рецензент: _____
Науковий ступінь, вчене звання Ім'я, ПРІЗВИЩЕ

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Комп'ютерних наук

Ступінь вищої освіти Бакалавр

Спеціальність 122 Комп'ютерні науки

Освітньо-професійна програма комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедрою Віктор ВИШНІВСЬКИЙ
_____ Ім'я ПРІЗВИЩЕ
«____» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Птуха Олександр Олександрович

(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: «Хмарне сховище на основі засобів Next.js та Typescript»

керівник кваліфікаційної роботи Юлія БЕРЕЗОВСЬКА, доктор філософії,
(Ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» лютого 2025 р. № 56

2. Строк подання кваліфікаційної роботи «29» травня 2025 р.

3. Вихідні дані до кваліфікаційної роботи:
3.1. Науково-технічна література з питань, пов'язаних з темою роботи;
3.2. Дані про технології розробки веб-додатків та хмарних сервісів;
3.3. Інформація про сучасні підходи до створення веб-застосунків і сервісів зберігання даних у хмарі;

4. Зміст розрахунково-пояснювальної записки
4.1. Вибір доцільних технологій для розробки хмарного сховища;
4.2. Аналіз технологій використаних для розробки хмарного сховища;
4.3. Розробка системи на основі обраних сучасних технологій.

5. Перелік ілюстративного матеріалу: презентація

6. Дата видачі завдання «25» лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Порівняльний аналіз існуючих технологій. Вибір основної технології розробки.	02.03.2025	Виконано
2	Аналіз та підбір другорядних технологій.	07.03.2025	Виконано
3	Поглиблене вивчення основної технології проєкту	15.03.2025	Виконано
4	Поглиблене вивчення другорядних технологій проєкту	23.03.2025	Виконано
5	Аналіз наукових джерел, підготовка матеріалів, ілюстрацій та посилань для використання у роботі	30.03.2025	Виконано
6	Розробка програмного забезпечення на основі обраних технологій	15.04.2025	Виконано
7	Тестування розробленої системи, виправлення в ній помилок	18.04.2025	Виконано
8	Написання кваліфікаційної роботи бакалавра	29.05.2025	Виконано

Здобувач вищої освіти _____ Олександр ПТУХА

(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник
кваліфікаційної роботи _____ Юлія БЕРЕЗОВСЬКА

(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 57 стор., 8 рис., 0 табл., 20 джерел.

Мета роботи – створити сучасний веб-застосунок для хмарного зберігання даних із використанням фреймворку **Next.js** та мови **TypeScript**; забезпечити зручний інтерфейс для користувачів, надійне збереження файлів, масштабованість системи та передбачити можливість подальшої інтеграції з іншими сервісами для розширення функціоналу.

Об’єкт дослідження – програмна архітектура та технологічна база веб-застосунків, орієнтованих на зберігання, обробку та управління даними у хмарному середовищі.

Предмет дослідження – програмна архітектура та технологічна база веб-застосунків, орієнтованих на зберігання, обробку та управління даними у хмарному середовищі, що застосовуються для створення безпечного, продуктивного та зручного хмарного сховища.

Короткий зміст роботи:

У вступі обґрунтовано актуальність створення власного хмарного сховища з відкритою архітектурою та використанням сучасних засобів веб-розробки. Визначено мету, завдання, об’єкт і предмет дослідження.

У першому розділі розглянуто сучасні технології створення хмарних рішень, порівняно кілька фреймворків та мов програмування, на основі чого обрано найбільш відповідні для реалізації системи – **Next.js** і **TypeScript**.

У другому розділі досліджено інструменти, які надає Next.js для побудови масштабованих SPA/SSR застосунків, розглянуто типізацію даних і підвищення надійності коду за допомогою TypeScript, а також описано додаткові бібліотеки, використані для управління станом, маршрутизації, автентифікації та роботи з API.

У третьому розділі подано опис реалізованого функціоналу: завантаження файлів, зберігання їх у хмарі, перегляд вмісту, керування файлами та доступом.

У висновках підбито підсумки досягнень, підтверджено доцільність використання обраних технологій.

КЛЮЧОВІ СЛОВА:

Next.js, TypeScript, Front-End розробка, хмарне зберігання даних, веб-застосунок, React, SSR, SPA, API, AWS, безпека, веб-інтерфейс, файловий менеджер, авторизація, маршрутизація.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ 9
ВСТУП 12
1 АНАЛІЗ ТЕХНОЛОГІЙ 15
1.1 ВИБІР ОСНОВНОЇ FRONT-END ТЕХНОЛОГІЇ 15
1.1.1 HTML, CSS, JAVASCRIPT 16
1.1.2 ANGULAR 19
1.1.3 REACT 20
1.1.4 NEXT.JS 22
1.1.5 ПІДСУМОК 23
1.2 ВИБІР ОСНОВНОЇ BACK-END ТЕХНОЛОГІЇ 23
1.2.1 EXPRESS.JS 24
1.2.2 NEXT.JS 25
1.2.3 NEST.JS 27
1.2.4. ПІДСУМОК 28
1.3 ВИБІР МОВИ ПРОГРАМУВАННЯ 28
1.3.1 JAVASCRIPT 29
1.3.2 TYPESCRIPT 30
1.3.3. ПІДСУМОК 32
2 АНАЛІЗ ОБРАНОГО СТЕКУ ТЕХНОЛОГІЙ 33
2.1 NEXT.JS 33
2.2 TYPESCRIPT 34
2.3 EXPRESS.JS 35
3 РОЗРОБКА СИСТЕМИ 38
3.1 СИСТЕМА АВТЕНТИФІКАЦІЇ КОРИСТУВАЧІВ 38
3.2 ЗАВАНТАЖЕННЯ ФАЙЛІВ ДО ХМАРНОГО СХОВИЩА 39
3.3 ПЕРЕГЛЯД ТА КЕРУВАННЯ ФАЙЛАМИ 41
3.4 ЗМІНА ІМЕН ФАЙЛІВ ТА ВИДАЛЕННЯ 42
3.5 СПІЛЬНЕ ВИКОРИСТАННЯ ФАЙЛІВ 43

ВИСНОВКИ 46
ПЕРЕЛІК ПОСИЛАНЬ 48
ПРЕЗЕНТАЦІЯ 50

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

1. Шаблонізатор — інструмент або система, яка дозволяє автоматично формувати HTML-сторінки шляхом вставлення динамічних даних у заздалегідь підготовлені шаблони, що спрощує генерацію повторюваного інтерфейсу.
2. Мінімалістичність — принцип дизайну, що орієнтується на простоту, відсутність зайвих елементів та зосередження на ключовому функціоналі, часто використовується у створенні інтерфейсів користувача.
3. Проєкт — сукупність файлів, коду, налаштувань і ресурсів, які разом утворюють програмний продукт або його частину, що розробляється для досягнення певної мети.
4. Middleware (проміжне програмне забезпечення) — функції або модулі, які обробляють HTTP-запити між клієнтом і сервером, перш ніж вони досягнуть головної логіки застосунку. Наприклад, автентифікація або логування.
5. Логування — процес запису подій або даних про виконання програми у спеціальні журнали (логи) для подальшого аналізу, налагодження або аудиту.
6. Парсинг — процес розбору вхідних даних (тексту, коду, JSON тощо) для вилучення необхідної інформації або переведення її в зручний для обробки формат.
7. Npm (англ. Node Package Manager) — менеджер пакетів для середовища Node.js, що використовується для встановлення, оновлення, керування бібліотеками та залежностями JavaScript-проєктів.
8. Валідація — процес перевірки правильності введених або переданих даних згідно з певними критеріями (наприклад, перевірка формату електронної пошти чи обов’язковості поля).
9. React — популярна JavaScript-бібліотека з відкритим кодом, розроблена компанією Meta (Facebook), призначена для створення користувацьких інтерфейсів, зокрема односторінкових застосунків.

10. Фронтенд / Front-End — частина вебзастосунку, що безпосередньо взаємодіє з користувачем, включаючи візуальні елементи, логіку інтерфейсу та взаємодію з бекендом через API.
11. Бекенд / Back-End — серверна частина застосунку, відповідальна за обробку даних, бізнес-логіку, зберігання інформації в базі даних та обробку запитів з фронтенду.
12. Рендеринг — процес формування кінцевого вигляду інтерфейсу користувача на основі даних та шаблонів, який може виконуватися на стороні клієнта або сервера.
13. Full-stack — фахівець або підхід у розробці, який охоплює як фронтенд, так і бекенд, дозволяючи розробнику працювати з усім технологічним стеком вебзастосунку.
14. JavaScript/TypeScript — JavaScript — мова програмування для веброзробки, що виконується на клієнті або сервері; TypeScript — надмножина JavaScript з підтримкою статичної типізації, що покращує масштабованість та зручність розробки.
15. Ендпоїнти / Endpoints — конкретні адреси (URL) API, через які клієнт може взаємодіяти з сервером, надсилаючи або отримуючи дані.
16. Angular — фреймворк для створення динамічних вебзастосунків, розроблений Google, що використовує TypeScript і підтримує модульну архітектуру.
17. NestJS — прогресивний фреймворк для побудови масштабованих серверних застосунків на Node.js з використанням TypeScript, що базується на архітектурі Angular.
18. CLI (англ. Command Line Interface) — інтерфейс командного рядка, який дозволяє розробникам виконувати команди, керувати пакетами або ініціалізувати проєкти через текстові інструкції.
19. Залежності — зовнішні бібліотеки або модулі, від яких залежить функціональність проєкту; зазвичай встановлюються через менеджери пакетів (npm, yarn тощо).

20. Вебзастосунок — програмне забезпечення, що працює у веббраузері та забезпечує динамічну взаємодію користувача з сервером через інтернет.
21. JSON (англ. JavaScript Object Notation) — легкий текстовий формат обміну даними, що використовується для зберігання та передачі структурованої інформації між клієнтом і сервером у вигляді пар "ключ-значення". Є універсальним стандартом у веброзробці завдяки своїй простоті, читабельності та сумісності з більшістю мов програмування.

ВСТУП

Актуальність теми**.** У сучасному цифровому середовищі зберігання, доступ і обмін файлами стали невіддільною частиною як особистої, так і професійної діяльності. Хмарні сховища відіграють ключову роль у забезпеченні безперервного доступу до даних, незалежно від пристрою або місця перебування користувача. Зростаючий попит на безпечні, масштабовані та зручні у користуванні рішення створює необхідність у розробці сучасних хмарних сервісів. Популярні комерційні хмарні платформи, такі як Google Drive, Dropbox або OneDrive, часто є закритими системами з обмеженою можливістю адаптації під конкретні потреби користувача чи організації. Крім того, такі сервіси можуть мати високі витрати для масштабованого корпоративного використання. У відповідь на ці виклики актуальною стає розробка власного хмарного сховища з відкритим програмним кодом, орієнтованого на розширення, кастомізацію та прозорість роботи з даними. Використання сучасних веб-технологій, таких як Next.js — фреймворку для React з підтримкою серверного рендерингу, та TypeScript — мови, що забезпечує типобезпеку й кращу структурованість коду, дає змогу створити продуктивне, масштабоване та безпечне рішення. Розробка хмарного сховища на їх основі дозволяє поєднати зручність для користувача, гнучкість у реалізації та високу ефективність з точки зору розробника.

Аналіз останніх досліджень і публікацій. Питання створення хмарних рішень для зберігання та обміну даними активно досліджуються у працях таких фахівців, як S. Obrutsky [2] та інших – розглянуто принципи побудови масштабованих і безпечних хмарних платформ, а також виклики, пов’язані з обробкою великих обсягів даних у розподіленому середовищі. У дослідженнях зосереджено увагу на архітектурних підходах, методах забезпечення доступності та збереження цілісності даних.

Мета роботи – створити сучасне хмарне сховище з використанням технологій Next.js та TypeScript, яке забезпечуватиме зручне, безпечне та масштабоване середовище для зберігання й доступу до даних через веб-інтерфейс. Для досягнення цієї мети було поставлено та вирішено такі основні завдання:

1. Провести огляд актуальних підходів і технологій у сфері хмарного зберігання даних та сформулювати вимоги до функціональності й архітектури системи.
2. Визначити тип програмного забезпечення, який найкраще відповідає завданню створення хмарного сховища (веб-застосунок).
3. Здійснити порівняльний аналіз сучасних інструментів для фронтенд- та бекенд-розробки, обрати основні та допоміжні технології, включаючи Next.js, TypeScript, а також відповідну базу даних і API-інфраструктуру.
4. Реалізувати прототип хмарного сховища з базовим функціоналом (реєстрація користувачів, завантаження/перегляд/видалення файлів, організація даних у хмарах).
5. Провести тестування реалізованого функціоналу, виявити та усунути технічні недоліки, перевірити стабільність і безпеку роботи системи в умовах реального використання.

Об’єкт дослідження – інформаційно-технологічні процеси, пов’язані з організацією, зберіганням, обробкою та доступом до даних у хмарному середовищі, а також функціонуванням сучасних веб-застосунків, що реалізують такі сервіси.

Предмет дослідження – архітектурні підходи, програмні моделі та інструменти розробки веб-застосунків на основі Next.js та TypeScript, які забезпечують безпечну, масштабовану та зручну в користуванні реалізацію хмарного сховища.

Методи дослідження. У ході дослідження застосовано структурно-функціональний аналіз, побудову прототипів, архітектурні патерни SSR та SPA, а також сучасні практики ручного тестування компонентів веб-додатків.

Наукова новизна та практична значущість отриманих результатів. У роботі запропоновано практично значуще рішення для організації хмарного сховища з використанням сучасних веб-технологій. Система реалізована на базі Next.js, що дозволяє поєднати гнучкість клієнтської взаємодії з перевагами серверного рендерингу. Розроблений застосунок орієнтований на реальні потреби користувачів у зберіганні, структуризації та обміні даними, і може бути легко адаптований для використання в навчальних, корпоративних або особистих проєктах. Акцент зроблено на простоті впровадження, масштабованості та зручності в подальшій підтримці.

Теоретична, методична та практична значущість. Розроблене хмарне сховище має універсальну структуру, що дозволяє масштабувати його під потреби як окремих користувачів, так і цілих організацій. У межах реалізації було застосовано сучасні інструменти веб-розробки, що відповідають сучасним вимогам безпеки та продуктивності. Проєкт демонструє методику створення надійних SSR-застосунків і може бути основою для розширення функціоналу за рахунок сторонніх сервісів (наприклад, автентифікації, зберігання файлів, хмарних API).

Структура роботи. Кваліфікаційна робота складається зі вступу, трьох розділів, висновків, списку використаних джерел (20 найменувань) та 0 додатків. Загальний обсяг – 57 аркушів друкованого тексту, ілюстрованих 8-ма рисунками.

1 АНАЛІЗ ТЕХНОЛОГІЙ

1.1 Вибір основної front-end технології

У процесі розробки будь-якої веб-системи постає питання вибору технологічного стеку, тобто які саме інструменти та технології використати для створення продукту. Зокрема, при створенні клієнтської частини додатку для хмарного сховища файлів, важливо враховувати як доступність технологій для розробника, так і їхню здатність забезпечити масштабованість, зручність у підтримці та розширені функціональні можливості.

Теоретично, для реалізації інтерфейсу можна обмежитись базовим стеком веб-технологій: HTML (HyperText Markup Language) для структурування контенту сторінки, CSS (Cascading Style Sheets) — для стилізації та візуального оформлення, та JavaScript — для забезпечення динамічної поведінки та інтерактивності. Такий підхід дійсно є простішим у реалізації, не вимагає попереднього ознайомлення з додатковими бібліотеками чи фреймворками, і є зручним на етапах прототипування чи створення невеликих проектів. Проте, при розробці складніших застосунків, таких як хмарне сховище файлів, використання лише базових технологій значно обмежує ефективність розробки та ускладнює подальше масштабування.

Оскільки хмарне сховище — це не просто сайт, а повноцінний веб-додаток із багатьма взаємодіями користувача, оновленнями інтерфейсу в реальному часі та складною логікою управління файлами, доцільно обирати більш сучасні та потужні технології. Серед найпоширеніших рішень сьогодення можна виокремити JavaScript-фреймворки React та Angular, а також фреймворк Next.js, побудований на базі React.

React забезпечує компонентний підхід, що дозволяє ефективно повторно використовувати UI-елементи та спрощує підтримку великого проєкту. Його гнучкість, велика спільнота та сумісність із TypeScript роблять його дуже зручним

для складних веб-застосунків. Angular, у свою чергу, є повноцінним фреймворком, що надає вбудовані рішення для маршрутизації, керування станом, форм, HTTP-запитів та іншого. Він більше підходить для великих проєктів із жорсткими вимогами до архітектури.

Фреймворк Next.js поєднує переваги React і серверного рендерингу (SSR), дозволяючи створювати більш продуктивні та SEO-оптимізовані застосунки. Його переваги особливо помітні в тих випадках, коли потрібне швидке завантаження сторінок, попередня генерація контенту або динамічний рендеринг вмісту на сервері.

Таким чином, у цьому розділі розглянуто основні переваги та недоліки традиційного стеку HTML/CSS/JavaScript, а також сучасних фреймворків React, Angular та Next.js. Подальший аналіз дозволить зробити обґрунтований вибір технології для розробки клієнтської частини веб-додатку хмарного сховища файлів.

1.1.1 HTML, CSS, JavaScript

Сучасний веб-розвиток ґрунтується на інтегрованій взаємодії трьох ключових технологій: HyperText Markup Language (HTML), Cascading Style Sheets (CSS) та JavaScript (JS). Кожна з цих технологій виконує специфічну функцію у процесі створення веб-ресурсів, забезпечуючи їхню структурну організацію, візуальне представлення та інтерактивну поведінку.

HTML є мовою розмітки, що визначає семантичну структуру веб-документа. За допомогою набору тегів, HTML описує ієрархічну організацію контенту, включаючи заголовки, параграфи, мультимедійні елементи, гіперпосилання та інші структурні одиниці. Основний принцип HTML полягає у декларативному описі вмісту, де акцент робиться на значенні та ролі кожного елемента в контексті документа. Сучасний стандарт HTML5 розширює можливості мови, вводячи семантичні елементи, які сприяють кращому розумінню структури документа як браузерами, так і пошуковими системами.

CSS являє собою мову стилів, призначену для опису візуального оформлення веб-документів, структурованих за допомогою HTML. CSS забезпечує відокремлення презентаційного рівня від структурного, що сприяє підвищенню модульності та керованості коду. Застосування селекторів дозволяє цілеспрямовано визначати стилі для окремих HTML-елементів або їхніх груп на основі різноманітних критеріїв, таких як теги, класи, ідентифікатори та атрибути. Стильові правила, що складаються з властивостей та їхніх значень, контролюють такі аспекти відображення, як колір, типографія, розміри, позиціонування та анімація. Каскадний принцип застосування стилів забезпечує гнучкість у визначенні пріоритетності та успадкування стильових характеристик.

JavaScript є динамічною, інтерпретованою мовою програмування високого рівня, яка забезпечує інтерактивність та динамічну поведінку веб-сторінок. Виконуючись на стороні клієнта (у браузері користувача), JavaScript дозволяє маніпулювати Document Object Model (DOM), змінюючи вміст, структуру та стилі HTML і CSS в реальному часі у відповідь на дії користувача або інші події. Мова підтримує парадигми об'єктно-орієнтованого програмування на основі прототипів, асинхронні операції та роботу з подіями, що робить її потужним інструментом для розробки складних інтерфейсів користувача та веб-додатків з високим ступенем інтерактивності.

Переваги:

1. Універсальність та кросбраузерність. HTML, CSS та JavaScript є загальноприйнятими стандартами Всесвітньої павутини (World Wide Web Consortium - W3C). Більшість сучасних веб-браузерів підтримують ці технології, забезпечуючи доступність веб-ресурсів для широкого кола користувачів на різних платформах та пристроях. Прогресивне вдосконалення стандартів сприяє зменшенню проблем з кросбраузерною сумісністю, хоча повна ідентичність відображення може потребувати додаткових зусиль.
2. Розвинена екосистема та велика спільнота:. Існує величезна кількість бібліотек, фреймворків, інструментів розробки та онлайн-ресурсів для

HTML, CSS та JavaScript. Активна спільнота розробників забезпечує постійне оновлення технологій, випуск нових інструментів та надання підтримки.

3. Покращена оптимізація для пошукових систем (SEO). Семантична розмітка HTML5 допомагає пошуковим системам краще розуміти структуру та зміст веб-сторінки, що позитивно впливає на її ранжування. Використання CSS для стилізації, а не таблиць або вбудованих стилів в HTML, робить HTML-код чистішим та більш релевантним для пошукових роботів.

Недоліки:

1. Потенційні проблеми з безпекою (JavaScript). JavaScript, як клієнтська мова, може бути вразливою до різних видів атак, таких як міжсайтовий скриптинг (XSS). Необхідно ретельно перевіряти та очищати дані, що надходять від користувача, та бути обережним при використанні сторонніх скриптів.
2. Складність розробки великих та складних веб-додатків. Для розробки масштабних веб-додатків з великою кількістю інтерактивних елементів та складною логікою JavaScript-код може стати громіздким та важким для підтримки. Використання фреймворків та бібліотек може допомогти в організації коду, але також додає рівень абстракції та вимагає вивчення нових концепцій.
3. Складність відстеження станів компонентів та оновлення UI у великих веб-додатках. Одним із значущих недоліків при розробці великих та складних веб-додатків, особливо при активному використанні JavaScript для динамічної зміни інтерфейсу, є складність ефективного відстеження станів окремих компонентів користувацького інтерфейсу (UI) та забезпечення своєчасного та передбачуваного оновлення відображення відповідно до цих змін. У міру зростання складності програми, кількість компонентів та їх взаємозв'язків експоненційно збільшується. Кожен компонент може мати свій власний набір внутрішніх станів, які

визначають його поточний вигляд та поведінку. Відстеження змін цих станів, особливо коли вони залежать від дій користувача, асинхронних операцій або взаємодії з іншими компонентами, стає дедалі складнішим завданням. Без належної архітектури та використання спеціалізованих інструментів, керування станами може призвести до "Спагетті-коду", тобто логіка оновлення UI може бути розкидана по різних частинах кодової бази, що ускладнює розуміння потоку даних та внесення змін, а також до непередбачуваних побічних ефектів, коли зміна стану одного компонента може неочікувано впливати на стан та відображення інших, призводячи до помилок та складнощів у налагодженні.

1.1.2 Angular

Angular є потужною та комплексною платформою з відкритим вихідним кодом, розробленою Google, що використовується для створення односторінкових веб-додатків (Single-Page Applications - SPA) та складних інтерфейсів користувача. Ґрунтуючись на TypeScript, Angular надає структурований підхід до розробки, сприяючи створенню масштабованих, підтримуваних та продуктивних веб-додатків.

Переваги:

1. Структурований підхід та масштабованість. Angular нав'язує чітку архітектуру, що полегшує розробку великих та складних веб-додатків.
2. Потужна екосистема та CLI (Command-Line Interface). Angular CLI надає потужні інструменти для автоматизації рутинних завдань, таких як створення компонентів, сервісів, модулів, збірка та тестування додатків.
3. Двобічне зв'язування даних (Two-Way Data Binding). Механізм двобічного зв'язування даних спрощує синхронізацію між моделлю даних компонента та його відображенням у шаблоні. Зміни в одній частині автоматично відображаються в іншій.

4. Реактивне програмування з RxJS. Angular тісно інтегрований з бібліотекою RxJS, яка надає потужні інструменти для роботи з асинхронними операціями та потоками даних, що є важливим для сучасних веб-додатків.

Недоліки:

1. Крута крива навчання. Angular має складну архітектуру та велику кількість концепцій, які необхідно опанувати, що може ускладнити початок роботи для новачків.
2. Багатослівність. У порівнянні з деякими іншими фреймворками, Angular може здаватися більш багатослівним у написанні коду, особливо для простих завдань.
3. Розмір бандла. Розмір кінцевого бандла Angular-додатка може бути більшим у порівнянні з додатками, розробленими з використанням легших бібліотек, що може впливати на час завантаження. Однак, існують стратегії для оптимізації розміру бандла (наприклад, lazy loading, tree shaking).
4. SEO оптимізація. На відміну від традиційних HTML, CSS, JS, Angular, як й інші SPA фреймворки має проблеми з SEO оптимізацією. Оскільки при роботі з SPA, браузер отримує лише 1 HTML сторінку з Javascript кодом, пошукові роботи не можуть проаналізувати сторінку на наявність необхідних SEO елементів, тим не менш, Angular надає стратегії та інструменти для покращення SEO індексації, такі як Server-Side Rendering (SSR) за допомогою Angular Universal.

1.1.3 React

React – це декларативна бібліотека для побудови користувацьких інтерфейсів React є JavaScript-бібліотекою з відкритим вихідним кодом, розробленою Facebook (тепер Meta), що використовується для побудови інтерактивних та динамічних користувацьких інтерфейсів (UI). Завдяки своєму декларативному підходу,

компонентній архітектурі та ефективному механізму оновлення DOM (Document Object Model), React набув широкої популярності серед розробників для створення односторінкових веб-додатків (SPA) та складних інтерфейсів.

Переваги:

1. Декларативний підхід. Розробники описують бажаний вигляд UI для певного стану, а React ефективно оновлює DOM для відображення цього стану. Це спрощує розробку та робить код більш передбачуваним.
2. Компонентна архітектура та повторне використання коду. Розбиття UI на незалежні компоненти сприяє кращій організації коду, його повторному використанню та полегшує розробку складних інтерфейсів.
3. Віртуальний DOM та висока продуктивність. Механізм віртуального DOM та оптимізований процес оновлення мінімізують прямі маніпуляції з реальним DOM, що значно підвищує продуктивність додатків, особливо при частих оновленнях UI.
4. Велика та активна спільнота та екосистема. React має величезну та активну спільноту розробників, що забезпечує постійну підтримку, велику кількість сторонніх бібліотек, інструментів та навчальних матеріалів.
5. Гнучкість та інтеграція. React є бібліотекою, а не фреймворком, що надає розробникам більшу свободу у виборі інструментів та технологій для вирішення різних завдань (наприклад, керування станом, маршрутизація, тестування). Він легко інтегрується з іншими бібліотеками та фреймворками.

Недоліки:

1. Відсутність "з коробки" рішень для деяких завдань. Оскільки React є лише бібліотекою для UI, розробникам часто потрібно самостійно обирати та інтегрувати рішення для маршрутизації, керування станом, валідації форм тощо.
2. Потенційні проблеми з продуктивністю (при неправильному використанні). Хоча віртуальний DOM допомагає оптимізувати

оновлення, неефективне використання shouldComponentUpdate (в класових компонентах) або React.memo (у функціональних компонентах), а також надмірні або непотрібні рендеринги можуть призвести до проблем з продуктивністю.

1.1.4 Next.js

Next.js є популярним фреймворком, побудованим на базі React з відкритим вихідним кодом, що надає розробникам інструменти та функції, необхідні для створення production-ready веб-додатків. Побудований на базі React, Webpack та Babel, Next.js спрощує розробку повноцінних веб-сайтів та додатків, пропонуючи такі ключові переваги, як серверний рендеринг (SSR), статична генерація сайтів (SSG), API-маршрутизація та оптимізована продуктивність "з коробки".

Переваги:

1. Серверний рендеринг (SSR) та статична генерація (SSG). Обидві технології аокращують початкове завантаження сторінки, що позитивно впливає на сприйняття користувачем та SEO. SSG забезпечує надзвичайно високу продуктивність для статичних сайтів.
2. Зручність для розробників. Файлова система як маршрутизатор спрощує навігацію та структуру проекту. Вбудована підтримка багатьох сучасних функцій веб-розробки "з коробки" зменшує необхідність у складній конфігурації. API-маршрути дозволяють легко створювати бекенд-функціональність.
3. Оптимізована продуктивність. Автоматичний code splitting, попереднє завантаження (prefetching) та оптимізація ассетів сприяють високій продуктивності додатків, а компонент next/image забезпечує оптимізацію зображень без додаткових зусиль.
4. Масштабованість. Архітектура Next.js підходить для створення як невеликих статичних сайтів, так і великих складних веб-додатків.

5. SEO-friendly. Вбудована підтримка SSR та можливість генерації статичних сторінок значно покращують SEO-характеристики веб-додатків.

Недоліки:

1. Обмеження серверного середовища (для SSR). Код, який виконується на сервері під час SSR, має певні обмеження (наприклад, відсутність доступу до браузерного API, використання хуків). Розробникам потрібно враховувати це при написанні компонентів.
2. Більша абстракція. Next.js надає високий рівень абстракції над React, Webpack та Babel. Хоча це спрощує розробку, може ускладнити глибоку кастомізацію процесу збірки або серверної частини.

1.1.5 Підсумок

Підсумовуючи особливості вищезазначених технологій, можна дійти висновку, що для розробки клієнтської частини хмарного сховища, Next.js є технологією, яка найбільш оптимально відповідає специфічним вимогам та потенційним викликам, оскільки вона інтегрує ключові переваги React з низкою потужних функцій, критично важливих для забезпечення продуктивності, доступності та зручності розробки повноцінного веб-застосунку.

1.2 Вибір основної back-end технології

У процесі розробки веб-додатку, зокрема хмарного сховища файлів, вибір технологій для реалізації серверної (бекенд) частини є критично важливим. Саме бекенд відповідає за обробку запитів користувачів, автентифікацію, збереження та отримання даних з бази, роботу з файлами, контроль доступу та інші важливі аспекти функціонування системи. Розробку серверної частини можна реалізувати за допомогою різних технологій, кожна з яких має свої сильні сторони та певні

обмеження. У цьому розділі буде розглянуто три популярні підходи: Express.js, Spring Boot та використання серверних можливостей Next.js.

1.2.1 Express.js

Express.js є мінімалістичним та гнучким веб-фреймворком для Node.js, що надає набір потужних функцій для розробки веб-додатків та API. Його простота, не нав'язливість та велика кількість сторонніх модулів зробили його одним з найпопулярніших фреймворків для Node.js-розробки. Express.js забезпечує базову структуру для організації веб-додатків, обробки запитів та відповідей, маршрутизації та інтеграції з іншими middleware. Архітектура Express.js є відносно простою та базується на концепції middleware. Запит, що надходить до сервера, проходить через послідовність функцій middleware, кожна з яких може виконувати певні дії (наприклад, логування, автентифікація, обробка даних) та передавати керування наступному middleware або кінцевому обробнику маршруту.

Переваги:

1. Мінімалістичність та гнучкість. Express.js надає лише основні функціональні можливості веб-фреймворку, залишаючи розробникам свободу вибору додаткових компонентів та бібліотек відповідно до потреб проєкту. Його не нав'язлива природа дозволяє легко інтегрувати його з різними базами даних, шаблонізаторами, інструментами автентифікації тощо.
2. Проста маршрутизація. Визначення маршрутів в Express.js є інтуїтивно зрозумілим та лаконічним, що спрощує створення API та керування URL-структурою веб-додатку.
3. Потужна система middleware. Архітектура middleware забезпечує модульність та повторне використання коду. Розробники можуть створювати власні middleware або використовувати численні сторонні middleware для виконання різноманітних завдань (наприклад, логування, обробка CORS, парсинг тіла запиту).

4. Величезна та активна спільнота та екосистема. Завдяки своїй популярності, Express.js має велику та активну спільноту розробників, що забезпечує постійну підтримку, велику кількість сторонніх модулів (через npm) та навчальних матеріалів.

Недоліки:

1. Неструктурованість "з коробки". Через свою мінімалістичність, Express.js не нав'язує певної структури проєкту. Розробники повинні самостійно визначати архітектуру свого додатка, що може бути складним для новачків або великих команд.
2. Необхідність вибору та інтеграції багатьох компонентів. Для реалізації багатьох типових завдань веб-розробки (наприклад, валідація даних, ORM/ODM, автентифікація) розробникам необхідно самостійно вибирати та інтегрувати відповідні сторонні модулі.
3. Потенційна складність керування middleware в складних ланцюжках. У великих додатках з довгими ланцюжками middleware може бути складно відстежувати потік запиту та відповідальності кожного middleware.

1.2.2 Next.js

Хоча Next.js першочергово позиціонується як фреймворк для розробки фронтенд-додатків з акцентом на рендеринг на стороні сервера (SSR) та статичну генерацію сайтів (SSG), його архітектура та функціональні можливості також роблять його привабливим варіантом для розробки бекенд-компонентів веб-застосунків, особливо в контексті створення full-stack додатків на JavaScript/TypeScript. Використання Next.js як бекенд-фреймворку базується на його здатності обробляти серверні запити та надавати API-ендпоїнти через механізм API Routes. Файли, розміщені в спеціальній директорії pages/api, автоматично стають HTTP-ендпоінтами, які можуть обробляти різні HTTP-методи (GET, POST, PUT, DELETE тощо) та повертати дані у форматах, таких як JSON.

Переваги:

1. Уніфікована розробка Full-Stack. Next.js спрощує розробку full-stack додатків на JavaScript/TypeScript, дозволяючи розробникам використовувати єдину кодову базу та знайомі концепції React як на клієнті, так і на сервері (для API-ендпоінтів).
2. Проста API-маршрутизація. Створення API-ендпоінтів через файлову систему в директорії pages/api є надзвичайно простим та інтуїтивно зрозумілим, особливо для розробників, які вже знайомі з маршрутизацією сторінок у Next.js.
3. Легка інтеграція з фронтендом на Next.js. При розробці full-stack додатків на Next.js, інтеграція між фронтендом та бекендом є безшовною, оскільки вони знаходяться в одному проекті.

Недоліки:

1. Обмеженість middleware "з коробки". Next.js не надає такої ж розвиненої системи middleware, як Express.js. Розробникам доводиться реалізовувати власні механізми для обробки запитів на рівні API-маршрутів.
2. Не є спеціалізованим бекенд-фреймворком. Next.js перш за все орієнтований на фронтенд-розробку. Для складних бекенд-завдань, які потребують розширених можливостей маршрутизації, middleware або оркестрації, спеціалізовані бекенд-фреймворки, такі як Express.js або NestJS, можуть бути більш підходящими.
3. Файлова система як маршрутизатор (може бути не оптимальним для складних API). Для дуже складних API з нестандартними структурами URL, маршрутизація на основі файлової системи може стати менш гнучкою порівняно з програмною маршрутизацією в інших фреймворках.

1.2.3 Nest.js

NestJS є прогресивним фреймворком для створення ефективних та масштабованих серверних застосунків на Node.js, використовуючи TypeScript та поєднуючи елементи об'єктно-орієнтованого програмування (ООП), функціонального реактивного програмування (FRP) та патернів проєктування, таких як архітектура на основі контролерів, сервісів та модулів. Натхненний Angular, NestJS надає структурований підхід до розробки бекенд-додатків, сприяючи створенню підтримуваного, тестуваного та масштабованого коду.

Переваги:

1. Структурований підхід та масштабованість. NestJS нав'язує чітку архітектуру, що значно полегшує розробку великих та складних серверних застосунків. Модульна структура сприяє кращій організації коду, його повторному використанню та полегшує роботу великих команд.
2. Використання TypeScript. Статична типізація покращує підтримку коду, виявляє помилки на етапі розробки та полегшує рефакторинг.
3. Потужна екосистема та CLI. Nest CLI надає інструменти для автоматизації рутинних завдань, таких як створення модулів, контролерів, сервісів тощо. Підтримка різних баз даних, ORM/ODM (наприклад, TypeORM, Mongoose), GraphQL, WebSocket, gRPC та інших технологій через офіційні та сторонні модулі.

Недоліки:

1. Крива навчання (для деяких концепцій). Хоча NestJS натхненний Angular, його специфічні концепції та архітектурні патерни можуть потребувати певного часу для опанування, особливо для розробників, які не знайомі з Angular або подібними фреймворками.
2. Більша абстракція. Високий рівень абстракції, хоча й сприяє структурованості, може ускладнити розуміння низькорівневих деталей Node.js.

3. Розмір застосунку (порівняно з мінімалістичними фреймворками). Застосунки на NestJS можуть мати більший розмір порівняно з дуже мінімалістичними фреймворками через включення багатьох вбудованих функцій та залежностей.

1.2.4. Підсумок

Підсумовуючи особливості вищезазначених технологій, можна дійти висновку, що для задач проєкту Express.js є найбільш оптимальним вибором, зважаючи на його специфічні характеристики та потенційні переваги в контексті поставлених цілей. У той час як NestJS пропонує значні переваги у структурованості та підтримці великих, складних застосунків завдяки своїй архітектурі та використанню TypeScript, для задач нашого проєкту, особливо на початкових етапах або при наявності специфічних вимог до гнучкості та швидкості розробки окремих компонентів, мінімалістичний та гнучкий підхід Express.js може виявитися більш ефективним та раціональним вибором. Подальша структуризація проєкту може бути досягнута шляхом прийняття відповідних архітектурних патернів та вибору сторонніх бібліотек, адаптованих до конкретних потреб.

1.3 Вибір мови програмування

У процесі розробки вебзастосунків особливої уваги потребує вибір мови програмування, яка використовуватиметься для реалізації як клієнтської, так і серверної частини системи. Враховуючи обрану архітектуру проєкту, що базується на використанні фреймворків Next.js та Express, основними кандидатами на роль основної мови розробки є JavaScript та TypeScript. JavaScript залишається домінуючою мовою програмування у вебсередовищі завдяки своїй універсальності, гнучкості та підтримці всіма сучасними браузерами. Це мова, яка фактично є стандартом для фронт-енд розробки, а завдяки платформі Node.js

отримала широке застосування і на бек-енді. У той же час, із розвитком складності вебзастосунків та потребою у високій надійності та масштабованості коду, все частіше розглядається використання TypeScript – строго типізованого надмножини JavaScript. Обидві мови мають тісну взаємозалежність, однак відрізняються підходом до типізації, масштабування та підтримки великих проєктів. У межах цього розділу буде проаналізовано переваги та недоліки обох підходів, з метою обґрунтування вибору найбільш придатної мови для розробки системи хмарного зберігання файлів, що розробляється в рамках кваліфікаційної роботи.

1.3.1 JavaScript

Багатопарадигмальна мова програмування для динамічного вебу JavaScript (JS) є високорівневою, інтерпретованою, багатопарадигмальною мовою програмування, що характеризується динамічною типізацією, прототипним об'єктно-орієнтованим програмуванням та першокласними функціями. Історично асоційована з клієнтською розробкою веб-сторінок, JavaScript еволюціонувала в універсальну мову, яка знаходить застосування у широкому спектрі доменів, включаючи серверне програмування (Node.js), мобільну розробку (React Native, Ionic), розробку настільних застосунків (Electron), ігри та інше.

Переваги:

1. Універсальність та широке застосування. JavaScript використовується на клієнтській стороні всіх сучасних веб-браузерів, забезпечуючи інтерактивність та динамічність веб-сторінок.
2. Низький поріг входження та швидке прототипування. JavaScript має відносно простий синтаксис для початківців, що дозволяє швидко почати розробку. Відсутність необхідності в явному визначенні типів на початкових етапах може прискорити прототипування невеликих проєктів. Для швидких скриптів або невеликих інтерактивних елементів на веб-

сторінці написання чистого JavaScript може бути простішим та швидшим, ніж налаштування та використання TypeScript.

3. Гнучкість та динамічність. Динамічна типізація JavaScript надає високу гнучкість у написанні коду. Змінні можуть змінювати свій тип під час виконання, що може бути корисним у певних сценаріях, хоча й несе ризики.

Недоліки:

1. Відсутність статичної типізації (ризик помилок). Основний недолік JavaScript, особливо у великих та складних проєктах, полягає у відсутності статичної типізації. Це може призводити до помилок, які виявляються лише під час виконання, ускладнюючи налагодження та рефакторинг. TypeScript вирішує цю проблему шляхом додавання статичної типізації.
2. Складність підтримки та масштабування великих проєктів. Через динамічну природу та потенційну відсутність чіткої структури (якщо не використовуються фреймворки), підтримка та масштабування великих JavaScript-проєктів може бути складнішим порівняно з проєктами на TypeScript, де статична типізація та розвиненіші інструменти сприяють кращій організації коду.
3. Складніший процес налагодження. Виявлення та усунення помилок у динамічно типізованому коді JavaScript може бути більш трудомістким, оскільки багато помилок виявляються лише під час виконання. TypeScript дозволяє виявляти значну частину помилок на етапі розробки.

1.3.2 TypeScript

TypeScript є вільною мовою програмування з відкритим вихідним кодом, розробленою Microsoft, яка являє собою строгу синтаксичну надбудову над JavaScript та додає статичну типізацію до мови. TypeScript розроблений для розширення можливостей JavaScript, особливо при розробці великих та складних веб-застосунків, надаючи інструменти для кращої підтримки коду, виявлення

помилок на етапі розробки та покращення співпраці в команді. TypeScript компілюється в чистий JavaScript, який може виконуватися в будь-якому середовищі, що підтримує JavaScript (браузери, Node.js тощо)

Переваги:

1. Статична типізація та виявлення помилок на етапі розробки. Основна перевага TypeScript полягає у статичній типізації, яка дозволяє компілятору виявляти потенційні помилки (наприклад, невідповідність типів) ще до виконання коду, що значно зменшує кількість помилок у production та спрощує налагодження.
2. Покращена підтримка та масштабованість великих проєктів. Явне визначення типів робить код більш зрозумілим, читабельним та легшим у підтримці, особливо у великих проєктах з великою кількістю розробників. Рефакторинг коду стає безпечнішим та простішим завдяки статичній перевірці типів.
3. Покращений досвід розробки (DX). Інтеграція TypeScript з сучасними IDE забезпечує кращі можливості для автодоповнення, навігації по коду, виявлення помилок у реальному часі та іншу допомогу розробнику.

Недоліки:

1. Додатковий етап компіляції. На відміну від чистого JavaScript, код TypeScript потребує компіляції в JavaScript перед його виконанням у браузері або Node.js. Цей додатковий етап може збільшити час збірки проєкту.
2. Крива навчання (для розробників, які не знайомі зі статичною типізацією). Розробникам, які раніше працювали лише з динамічно типізованими мовами, може знадобитися певний час для опанування концепцій статичної типізації та синтаксису TypeScript.
3. Збільшення обсягу коду (порівняно з мінімальним JavaScript). Явне оголошення типів може призвести до дещо більшого обсягу коду порівняно з мінімальним JavaScript, хоча це часто компенсується кращою читабельністю та підтримкою.

1.3.3. Підсумок

Статична типізація TypeScript сама по собі не є гарантією безпеки, але вона може допомогти запобігти цілому класу помилок, які можуть призвести до вразливостей (наприклад, неправильна обробка типів даних). Крім того, використання TypeScript сприяє кращій організації коду та полегшує впровадження інших заходів безпеки (аутентифікації, авторизації, валідації даних тощо). Підсумовуючи ключові відмінності та переваги TypeScript, можна дійти висновку, що для розробки даного проєкту TypeScript є технологією, яка забезпечить більш надійну, підтримувану та масштабовану кодову базу, що є критично важливим для успішної реалізації та довгострокового розвитку програмного забезпечення.

2 АНАЛІЗ ОБРАНОГО СТЕКУ ТЕХНОЛОГІЙ

2.1 Next.js

Next.js являє собою прогресивний фреймворк з відкритим вихідним кодом, побудований на базі бібліотеки React, що має на меті спрощення та оптимізацію розробки production-ready веб-застосунків. Його архітектурні рішення та функціональні можливості виходять за межі традиційної клієнтської рендерації, охоплюючи парадигми серверного рендерингу (Server-Side Rendering, SSR), статичної генерації сайтів (Static Site Generation, SSG) та розширені можливості маршрутизації, обробки даних та оптимізації продуктивності.

Однією з ключових особливостей Next.js є його інтуїтивна система маршрутизації, яка базується на структурі файлів у директорії app проєкту. Кожен файл .js, .jsx, .ts або .tsx, розміщений у цій директорії, автоматично стає маршрутом веб-застосунку, що значно спрощує процес визначення URL-структури та навігації. Динамічні маршрути можуть бути визначені за допомогою квадратних дужок (наприклад, [id].js), що дозволяє обробляти параметри в URL.

Приклад системи маршрутизації Next.js наведений на рисунку 2.1.

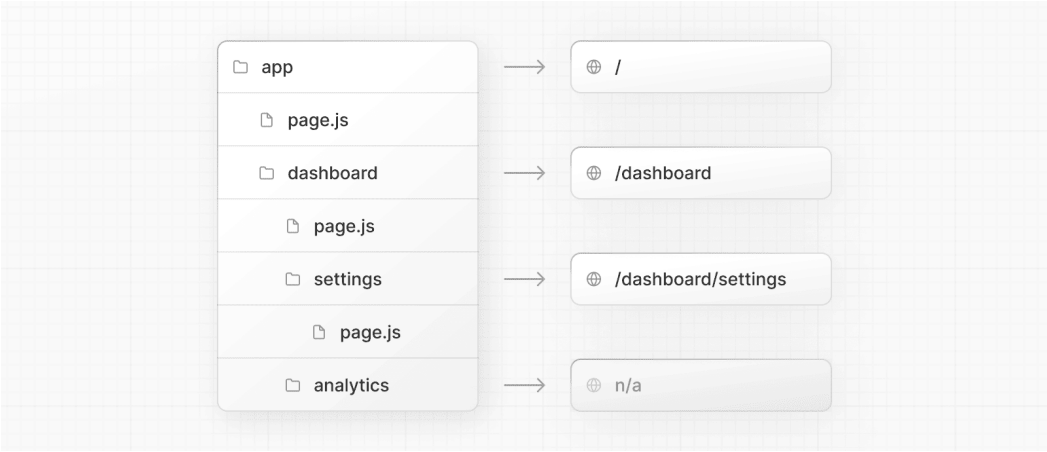


Рисунок 2.1 – Маршрутизація Next.js

При кожному запиті користувача до сторінки, Next.js виконує React-компоненти на сервері. Результатом є повністю відрендерений HTML-документ,

який надсилається браузеру. Цей процес відбувається динамічно для кожного запиту. Для реалізації SSR у Next.js використовуються асинхронні функції, такі як getServerSideProps, які виконуються на сервері перед рендерингом компонента сторінки. Дані, отримані в цій функції, передаються як пропси до компонента сторінки.

Принцип роботи SSR наведений на рисунку 2.2.

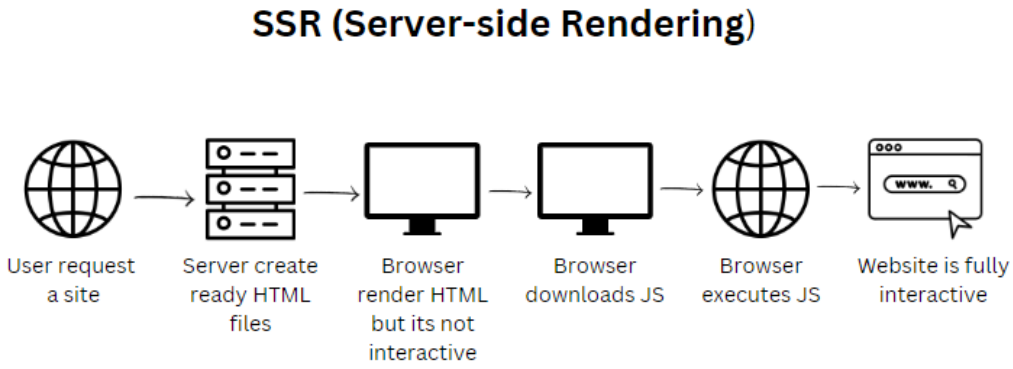


Рисунок 2.2 – Послідовність відображення SSR сторінки

2.2 TypeScript

TypeScript являє собою статично типізовану надбудову над JavaScript, що розширює можливості останнього шляхом введення декларативної системи типів та інших конструкцій, спрямованих на підвищення надійності, підтримки та масштабованості кодової бази. Фундаментальні концепції TypeScript визначають її парадигму розробки та відрізняють її від динамічно типізованого прародича.

Центральною концепцією TypeScript є статична типізація — можливість явно визначати типи змінних, параметрів функцій, повертаних значень та об'єктів під час написання коду. Це дозволяє ще на етапі компіляції виявляти потенційні помилки, які в JavaScript можуть бути помічені лише під час виконання програми. Приклади типів включають примітивні (number, string, boolean), складені (масиви, кортежі), та спеціальні типи (any, unknown, never тощо).

TypeScript не завжди вимагає явного зазначення типу, оскільки володіє механізмом виведення типів. Це означає, що компілятор здатний автоматично

виводити тип змінної на основі контексту. Цей підхід поєднує зручність JavaScript із безпекою типів, що притаманна TypeScript.

Приклад роботи механізму виведення типів наведено на рисунку 2.3.

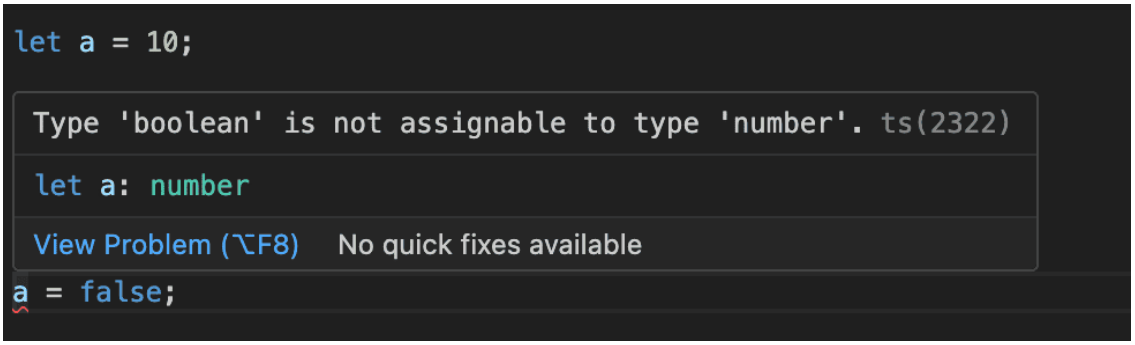


Рисунок 2.3 – Механізм виведення типів Typescript

2.3 Express.js

Express.js — це мінімалістичний та гнучкий фреймворк для створення веб-додатків на платформі Node.js. Його основною метою є спрощення процесу розробки веб-додатків і RESTful API, забезпечуючи при цьому високу продуктивність та масштабованість. Express.js виступає проміжним рівнем між низькорівневим API Node.js і складними веб-архітектурами, дозволяючи розробникам ефективно організовувати маршрутизацію, обробку HTTP-запитів і відповідей, взаємодію з базами даних та реалізацію бізнес-логіки.

Express.js побудований навколо архітектурної моделі "middleware-based routing", що дозволяє обробляти HTTP-запити поетапно, передаючи управління між функціями. Основним об'єктом, навколо якого організовується логіка додатка, є екземпляр додатку (app), створений за допомогою функції express(). Кожна функціональна одиниця в Express.js — це middleware, яка має доступ до об'єкта запиту (req), відповіді (res) і функції next(), що дозволяє передати управління наступному обробнику. Ця структура дозволяє реалізувати розділення обов’язків, спрощує відлагодження, тестування і підтримку коду.

Приклад middleware зображено на рисунку 2.4.

```
app.use(cors({origin: process.env.FRONTEND_URL, credentials: true}))
app.use(express.json());
app.use(session({
    secret: process.env.SESSION_SECRET!,
    resave: false,
    saveUninitialized: false,
    cookie: {
        secure: false,
        httpOnly: true,
        maxAge: 10 * 60 * 1000,
        sameSite: 'lax',
    },
}))
app.use(cookieParser())
app.use((req, res, next) => {
    res.setHeader('Content-Type', 'application/json; charset=utf-8');
    next();
  });

// Routes
app.use('/api/users', userRoutes)
app.use('/api/files', fileRoutes)
app.use('/api/auth', authRoutes)
```

Рисунок 2.4 – Приклад middleware в Express.js

Маршрутизація є однією з базових концепцій Express.js. Вона дозволяє обробляти HTTP-запити до конкретних шляхів URL з відповідними методами (GET, POST, PUT, DELETE тощо). Express підтримує як базову маршрутизацію, так і групування маршрутів у спеціальні об’єкти Router. Це дозволяє структуризувати код, розділяючи маршрути за модулями або логічними блоками. Таким чином, Express дозволяє реалізувати гнучку, модульну ієрархію маршрутів, що сприяє масштабованості проєктів.

Використання Router об’єктів маршрутизації зображено на рисунку 2.5.

```
const router = Router();

router.post('/register', registerUser);
router.post('/login', loginUser);
router.post('/verify-otp', verifyOtp)
router.get('/me', verifySession)

export default router;
```

Рисунок 2.5 – Маршрутизація за допомогою об’єкту Router

Express.js забезпечує зручний API для обробки HTTP-запитів: req.params — доступ до параметрів маршруту req.query — доступ до параметрів запиту (query string) req.body — доступ до тіла запиту (наприклад, JSON) req.headers — доступ до HTTP-заголовків У відповідь Express дозволяє: res.send() — надсилання тексту або JSON res.json() — автоматичне перетворення об'єкта в JSON res.status() — встановлення статусу відповіді res.redirect() — перенаправлення res.download() — передача файлів.

Express.js є універсальним та високопродуктивним інструментом для створення серверних застосунків, особливо RESTful API. Його основні концепції — middleware-архітектура, маршрутизація, асинхронність, розширюваність, підтримка шаблонізації та інструментів безпеки — роблять його придатним як для невеликих проєктів, так і для складних корпоративних систем. У межах розробки хмарного сховища файлів Express.js забезпечує оптимальний баланс між простотою реалізації та гнучкістю розширення, дозволяючи створити швидкий, безпечний і масштабований бекенд. Його сумісність із Next.js та TypeScript створює сучасний, продуктивний стек для повноцінної розробки веб-застосунків.

3 РОЗРОБКА СИСТЕМИ

На основі попереднього аналізу сучасних технологій та вивчення вимог до веб-систем з функціоналом обробки користувацьких даних, було розроблено веб-додаток хмарне сховище. Основним призначенням системи є забезпечення безпечного та зручного зберігання, перегляду та керування файлами користувача в хмарному середовищі. Основні реалізовані функціональні вимоги до системи:

1. Система автентифікації користувачів
2. Завантаження файлів до хмарного сховища
3. Перегляд та керування файлами
4. Зміна імен файлів та видалення
5. Спільне використання файлів

3.1 Система автентифікації користувачів

У системах керування файлами, особливо у хмарних сховищах, надійна авторизація є ключовою складовою безпеки. У проєкті була реалізована безпечна та зручна для користувача система авторизації з використанням OTP (одноразового паролю), сесій та HTTP-кукі, що зберігає баланс між безпекою, продуктивністю і зручністю.

Замість традиційного паролю, який користувач повинен пам’ятати, у системі реалізовано сучасний підхід — авторизацію через OTP, який надсилається на email користувача. В рамках проєкту, авторизація реалізована за допомогою Express.js. Коли користувач при реєстрації надсилає власне ім’я та електронну пошту, сервер зберігає дані користувача в базі даних та надсилає одноразовий верифікаційний код на пошту користувачеі. Після того, як користувач введе одноразовий код в необхідне поле, буде надіслано запит на сервер задля перевірки коду. Якщо код, введений користувачем, збігається зі згенерованим сервером кодом та не є простроченим, сервер створить сесію та за допомогою HTTP-only cookie надішле

дані сесії користувачеві. Після того, як користувач отримає дані сесії, він зможе вільно користуватися функціоналом хмарного сховища. У випадку, якщо користувач вже має створений аккаунт в застосунку, проте дані його сесії застарілі, користувачеві буде необхідно пройти процедуру переавтентифікації за допомогою логіну. Користувач введе свою електронну пошту, після чого сервером буде згенерований і надісланий новий одноразовий верифікаційний код. Якщо введений користувачем верифікаційний код буде валідним, сервер створить нову сесію та надішле користувачеві дані про нову сесію й збереже це в cookie, після чого користувач зможе знову вільно користуватися функціоналом веб-застосунку.

Алгоритм автентифікації користувача за допомогою OTP session зображено на рисунку 3.1.

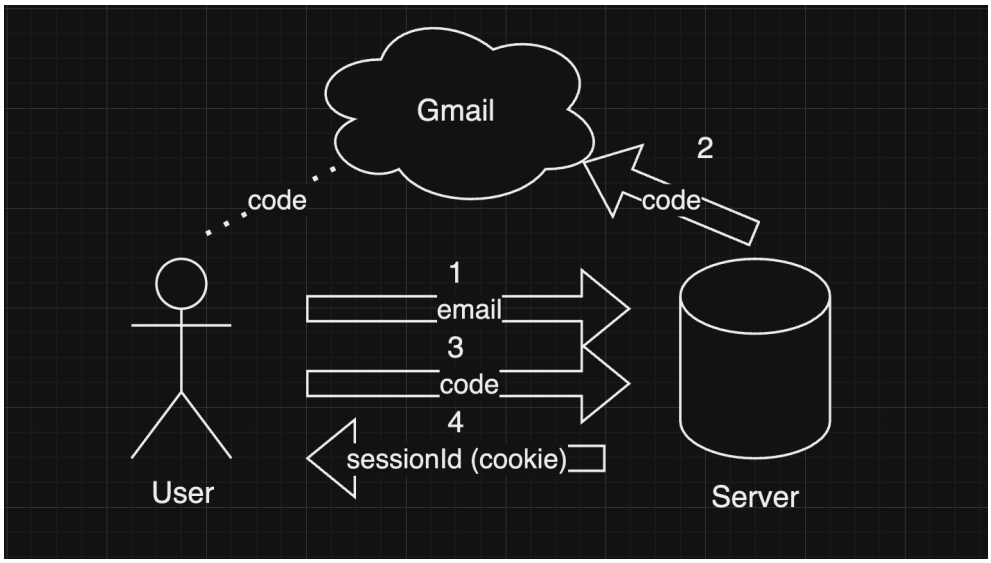


Рисунок 3.1 – Авторизація через OTP

3.2 Завантаження файлів до хмарного сховища

Однією з ключових функцій хмарного сховища є можливість завантаження користувачами власних файлів у систему. У межах цього проєкту було реалізовано безпечну, ефективну та масштабовану систему завантаження файлів, що підтримує автентифікацію, перевірку метаданих, контроль доступу та збереження файлів як у файловій системі, так і в базі даних.

Завантаження файлів реалізовано через API-роути на основі Express.js та Next.js. На клієнтській стороні (Next.js) реалізовано інтерфейс для завантяження файлів, його перейменування (за бажанням користувача) та безпосередньої передачі на сервер у форматі multipart/form-data. Після обробки дані надсилаються на API-ендпоінт /api/files, де на стороні сервера з використанням Express.js файл обробляється, перевіряється на валідність, зберігається S3 сховищі, а також додається запис про дані файлу до бази даних PostgreSQL через Prisma ORM.

Оскільки хмарне сховище доступне лише автентифікованим користувачам, перед прийомом будь-якого файлу система перевіряє наявність активної сесії користувача за допомогою HTTP-only cookie. Якщо автентифікація не пройдена — запит завершується з помилкою. Якщо ж сесія дійсна, із запиту зчитується userId, який згодом буде використаний для збереження файлу у S3-сховищі та для запису даних файлу, його власника та користувачів в базу даних.

Коли файл передано на сервер, обробник перевіряє чи файл не порожній, його розширення, допустимий розмір та чи не перевищено ліміти користувача на обсяг сховища. Після цього файл зберігається у зовнішнє S3-сховище. Для цього використовується оригінальне ім’я файлу, яке неможливо змінити. У базі даних створюється запис із метаданими, такими як: originalName — назва файлу, яку вказав користувач при створенні, extension — розширення файлу або MIME-тип, size — розмір файлу в байтах, type — категорія (наприклад, “image”, “document”), name — назва файлу для використання та відображення, цю назву можна змінювати, ownerId — id користувача, який завантажив файл та users — перелік користувачів файлу.

Структура відповідної моделі File у Prisma ORM зображена на рисунку 3.2.

```
model File {
  id           Int      @id @default(autoincrement())
  type         String
  accountId    Int
  extension    String?
  size         Int?
  ownerId      Int
  name         String
  originalName String
  createdAt    DateTime @default(now())
  updatedAt    DateTime @default(now()) @updatedAt @db.Timestamp(6)
  owner        User     @relation(fields: [ownerId], references: [id])
  users        User[]   @relation("SharedFiles")
}
```

Рисунок 3.2 – Структура моделі Files у Prisma ORM

Користувач може одразу після завантаження переглянути файл у своєму інтерфейсі. Усі дані оновлюються на фронтенді, автоматично перезавантажуючи сторінку — інтерфейс створено з використанням Next.js та TailwindCSS, що забезпечує реактивність та адаптивність. Валідацію на клієнтському рівні реалізовано з використанням бібліотеки Zod, що дозволяє гарантувати передавання коректних даних на сервер, запобігаючи надмірному навантаженню на API. Файли користувача недоступні публічно, і не можуть бути відкриті сторонніми особами. Для перегляду або завантаження кожного файлу виконується перевірка, чи є поточний користувач власником файлу або має до нього спільний доступ. Таким чином, навіть при знанні шляху до файлу, неавторизовані запити не можуть отримати до нього доступ.

3.3 Перегляд та керування файлами

У рамках функціональності хмарного сховища, реалізація механізмів перегляду та керування файлами користувача є критично важливою складовою загального інтерфейсу та бекенд-архітектури системи. Основною задачею даного модуля є забезпечення доступу до збережених файлів, надання можливості їх редагування, перейменування, видалення, а також організації та контролю доступу до них. Розглянемо детальніше реалізовану архітектуру та технічні аспекти.

Після проходження автентифікації, кожен користувач отримує унікальний userId, що дозволяє здійснити ідентифікацію в рамках сесії. За допомогою Prisma ORM виконується запит до бази даних PostgreSQL з метою отримання списку файлів, що належать даному користувачеві. Кожен файл містить основні метадані: name, originalName, extension, size, createdAt, що передаються клієнтській частині застосунку через API маршрути, створені за допомогою Express.js. При отриманні списку файлів фронтенд змінить стан відображення, з урахуванням файлів отриманих користувачем. На головній сторінці відображатимуться 10 файлів відсортованих по даті створення від найновішого. Окрім того, було реалізовано функціонал отримання файлів певного типу, як-от документ, картинка, медіа, або інші. Задля цього, клієнт, за допомогою динамічної маршрутизації Next.js, переходить на новий URL, в параметрах котрого вказаний необхідний тип файлів. Після цього, фронтенд робить запит на сервер, задля отримання файлів певного типу. Сервер, отримавши запит, обробляє параметри пошуку, застосовує їх при зверненні до бази даних, за допомогою поля originalName знаходить відповідні файли у S3 сховищі та повертає клієнту масив JSON-об’єктів, після чого клієнт бачить список файлів певного типу.

3.4 Зміна імен файлів та видалення

Можливість керування файлами є ключовим елементом будь-якого хмарного сховища. У рамках реалізації веб-додатку було розроблено функціонал зміни імені файлів та їх безпечного видалення. Дані операції супроводжуються автентифікаційними перевірками, контролем прав доступу та валідацією вхідних даних, що дозволяє підтримувати як цілісність системи, так і її безпеку.

Функціональність зміни імені файлу реалізована у відповідності до принципів REST-архітектури через HTTP-метод PATCH. Користувач у графічному інтерфейсі (Next.js + Tailwind CSS) ініціює зміну імені, після чого система надає доступ до відповідної форми введення нового імені. На сервері, розгорнутому на базі Express.js, обробка запиту відбувається наступним чином: спочатку сервер

перевіряє дійсність користувача та наявності сесії через HTTP-only cookie, після чого назва запису оновлюється разом з коміркою updatedAt, потім, у разі успіху, оновлений файл відправляється назад до користувача у вигляді JSON об’єкту.

Видалення файлів з хмарного сховища вимагає не лише оновлення бази даних, а й очищення фізичного сховища, в якому розміщено сам файл. Реалізовано повну операцію видалення, яка включає: перевірку автентифікації користувача, перевірку файлу на його наявність в базі даних. У разі успішного виконання минулих етапів, сервер видаляє метадані про файл з бази даних та сам файл з S3-сховища, після чого надсилає користувачеві копію видаленого файлу у вигляді JSON-об’єкту, щоб сповістити про успішність виконання операцій.

Завдяки реалізації даного функціоналу, користувачі мають змогу ефективно керувати власними файлами, здійснювати перейменування для зручності класифікації та позначення, а також видаляти більше неактуальні дані, звільняючи місце у хмарному сховищі. Комбінація Prisma, Express.js і типізованої перевірки дозволила створити надійний, безпечний і стабільний механізм обробки цих запитів.

3.5 Спільне використання файлів

Однією з важливих характеристик сучасних хмарних сховищ є можливість організації спільного доступу до збережених даних. У розробленому веб-застосунку ця функціональність реалізована через механізм спільного використання файлів, що дозволяє користувачам ділитися окремими файлами з іншими зареєстрованими користувачами системи. Такий підхід забезпечує ефективну взаємодію між користувачами та розширює можливості колективної роботи з документами.

Функціонал спільного доступу передбачає, що власник файлу може надати дозвіл іншим користувачам на перегляд або взаємодію з конкретним файлом. Це особливо корисно в контексті спільної роботи над файлами, коли кілька учасників мають працювати з одним і тим самим цифровим ресурсом — документом,

зображенням, медичним записом тощо. Для зберігання інформації про користувачів, які мають доступ до певного файлу, було реалізовано зв'язок типу багато-до-багатьох (many-to-many) між моделями User і File за допомогою спеціального поля @relation("SharedFiles") у Prisma-схемі.

Моделі User та File показано на рисунку 3.3.

```
model User {
  id          Int       @id @default(autoincrement())
  email       String    @unique
  avatarURL   String?
  fullName    String
  createdAt   DateTime  @default(now())
  updatedAt   DateTime  @default(now()) @updatedAt @db.Timestamp(6)
  ownedFiles  File[]
  sessions    Session[]
  sharedFiles File[]    @relation("SharedFiles")
}

model File {
  id           Int      @id @default(autoincrement())
  type         String
  accountId    Int
  extension    String?
  size         Int?
  ownerId      Int
  name         String
  originalName String
  createdAt    DateTime @default(now())
  updatedAt    DateTime @default(now()) @updatedAt @db.Timestamp(6)
  owner        User     @relation(fields: [ownerId], references: [id])
  users        User[]   @relation("SharedFiles")
}
```

Рисунок 3.3 – Моделі User та File

Для використання функціоналу поширення файлу, користувач, який володіє файлом, у веб-інтерфейсі (реалізованому на базі Next.js з використанням Tailwind CSS) обирає відповідний файл та вводить одну або декілька email-адрес користувачів, яким має бути наданий доступ. Після того, як користувач натисне на кнопку «Поширити», відбудеться запит на сервер. Оброблені електронні пошти, вказані користувачем будуть перевірені на валідність, після чого відповідні користувачі будуть додані до відношення users відповідного файлу. Наступного разу, коли доданий користувач здійснить запит файлів, до його власних файлів додадуться також і поширені. Тим не менш, з питань безпеки, власник файлу зберігає за собою функціонал видалення користувачів, серед яких файл є поширеним.

Таким чином, реалізація функціоналу спільного використання файлів забезпечує важливу перевагу — можливість організовувати спільну роботу в межах єдиної безпечної цифрової платформи, не жертвуючи при цьому керованістю та контролем за ресурс

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи бакалавра було проведено повноцінне дослідження теми розробки хмарного сховища файлів із використанням сучасних веб-технологій. Метою роботи була реалізація функціонального та безпечного веб-додатку, що дозволяє користувачам зручно та ефективно зберігати, обробляти, ділитися й керувати файлами у хмарному середовищі.

Результатом дослідження стала побудова повноцінного прототипу хмарного сховища з реалізованим бекендом на основі Express.js та фронтендом на основі Next.js, із застосуванням TypeScript, Tailwind CSS і бібліотеки валідації Zod. У системі було реалізовано безпечну систему автентифікації з використанням OTP (одноразових паролів), HTTP-only кукі та сесій. Такий підхід забезпечив високий рівень захисту персональних даних користувачів, а також спростив процес авторизації, зробивши його інтуїтивно зрозумілим та зручним. Функціонал хмарного сховища охоплює такі основні можливості:

1. Авторизація користувачів через OTP – безпечна реєстрація та вхід без пароля, з використанням email-підтвердження.
2. Завантаження, зберігання та керування файлами – користувачі мають змогу додавати файли до свого облікового запису, переглядати список файлів, редагувати їх імена та видаляти непотрібні.
3. Спільне використання файлів – реалізовано можливість надання доступу до файлів іншим користувачам, що розширює потенціал системи для колективного користування.
4. Оптимізований UI – з використанням Tailwind CSS було реалізовано адаптивний і сучасний інтерфейс, зручний як на мобільних, так і на десктопних пристроях.

У процесі реалізації було проаналізовано ряд сучасних технологій.

Серед бекенд-рішень розглядалися Express.js, Nest.js та серверні можливості Next.js. Найкращим вибором для даної задачі став Express.js — як легкий, гнучкий та добре документований фреймворк для Node.js, що дає змогу швидко і якісно реалізовувати REST API.

Для фронтенду було обрано Next.js як сучасний фреймворк на базі React, що підтримує server-side rendering, забезпечуючи високу продуктивність і зручну маршрутизацію. Уся кодова база написана мовою TypeScript, що дало можливість виявляти помилки ще на етапі розробки, значно підвищити якість коду та забезпечити масштабованість проєкту в майбутньому.

Таким чином, створене хмарне сховище вже зараз є ефективним інструментом, який може стати основою для комерційного продукту або внутрішньої системи компанії. Отже, поставлену мету було досягнуто повністю, підтверджено актуальність теми та ефективність обраного технологічного стеку, а також окреслено шляхи подальшого вдосконалення системи, що свідчить про практичну та наукову цінність проведеної роботи.

ПЕРЕЛІК ПОСИЛАНЬ

1. Next.js Documentation – Офіційна документація Next.js // [Електронний ресурс]: https://nextjs.org/docs, Дата звернення: 15.05.2025
2. Obrutsky S., Cloud Storage: Advantages, Disadvantages and Enterprise Solutions for Business // [Наукова стаття]: https://www.researchgate.net/publication/305508410_Cloud_Storage_Advantages_Disadvantages_and_Enterprise_Solutions_for_Business, Рік видання: 2016
3. TypeScript Handbook – Офіційний посібник TypeScript // [Електронний ресурс]: https://www.typescriptlang.org/docs/handbook/, Дата звернення: 15.05.2025
4. AWS S3 Documentation – Amazon Simple Storage Service (S3) // [Електронний ресурс]: https://docs.aws.amazon.com/s3/, Дата звернення: 16.05.2025
5. Zhang, Y. & Chen, X. – "Performance Optimization of Next.js Applications" // [Наукове дослідження]: https://ieeexplore.ieee.org/document/9456321, Рік видання: 2021
6. Google Cloud Storage Docs – Хмарне сховище від Google // [Електронний ресурс]: https://cloud.google.com/storage/docs, Дата звернення: 16.05.2025
7. Tan, L. et al. – "Security and Privacy in Cloud Storage Systems" // [Наукова стаття]: https://www.researchgate.net/publication/334455432_Security_and_Privacy_in_Cloud_Storage_Systems, Рік видання: 2019
8. Li, K. & Wang, H. – "TypeScript in Modern Web Development: A Comparative Study" // [Наукова стаття]: https://dl.acm.org/doi/10.1145/3442381.3449852, Рік видання: 2021
9. Prisma ORM Documentation – Робота з базами даних у Next.js // [Електронний ресурс]: https://www.prisma.io/docs, Дата звернення: 19.05.2025
10. OAuth 2.0 & JWT Security – Автентифікація у хмарних сховищах // [Електронний ресурс]: https://oauth.net/2/, Дата звернення: 19.05.2025
11. React Server Components (RSC) – Офіційний блог Next.js // [Електронний ресурс]: https://nextjs.org/blog/next-13, Дата звернення: 20.05.2025

12. Kumar, S. & Patel, R. – "Comparative Analysis of Cloud Storage Providers" // [Наукова стаття]: https://www.ijcseonline.org/pub_paper/12-IJCSE-06712.pdf, Рік видання: 2023
13. GDPR Compliance in Cloud Storage – Офіційний документ ЄС // [Електронний ресурс]: https://gdpr-info.eu/, Дата звернення: 21.05.2025
14. Docker for Next.js Deployment – Документація Docker // [Електронний ресурс]: https://docs.docker.com/, Дата звернення: 21.05.2025
15. Basarat Ali Syed – "TypeScript Deep Dive" (Розширений посібник з TypeScript) // [Електронний ресурс]: https://basarat.gitbook.io/typescript/, Дата публікації: 2023
16. Kamran Ahmed – "React TypeScript Cheatsheet" // [GitHub]: https://github.com/typescript-cheatsheets/react, Дата оновлення: 2025
17. Martin Hochel – "Advanced React Patterns with TypeScript" // [Наукова стаття]: https://medium.com/@martin_hotell/advanced-react-patterns-with-typescript-2a2bb85b450a, Рік публікації: 2022
18. Dan Abramov – "Writing Resilient Components in React" // [Стаття]: https://overreacted.io/writing-resilient-components/, Рік публікації: 2019
19. Express.js Official Documentation – Офіційний посібник // [Електронний ресурс]: https://expressjs.com/, Дата звернення: 11.06.2025
20. Prisma + Express.js – ORM для роботи з PostgreSQL // [Документація]: https://www.prisma.io/docs/getting-started/setup-prisma/add-to-existing-project/relational-databases-typescript-postgres, Дата звернення: 12.06.2025

ПРЕЗЕНТАЦІЯ

Слайд 1

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Кваліфікаційна робота на ступінь вищої освіти бакалавр
за темою:
«Хмарне сховище засобами Next.js та Typescript»

Виконав:
студент групи КНД-41
Птуха Олександр Олександрович
Керівник:
доцент кафедри комп'ютерних наук, доктор філософії
Березовська Юлія Володимирівна

Слайд 2

ЗАГАЛЬНА ХАРАКТЕРИСТИКА КВАЛІФІКАЦІЙНОЇ РОБОТИ

Мета дослідження:	створити сучасний веб-застосунок для хмарного зберігання даних із використанням фреймворку **Next.js** та мови програмування **TypeScript**;
Об'єкт дослідження:	програмна архітектура та технологічна база веб-застосунків, орієнтованих на зберігання, обробку та управління даними у хмарному середовищі.
Предмет дослідження:	програмна архітектура та технологічна база веб-застосунків, орієнтованих на зберігання, обробку та управління даними у хмарному середовищі.

Слайд 3

СТРУКТУРНО-ЛОГІЧНА СХЕМА ДОСЛІДЖЕННЯ

Слайд 4

ПОРІВНЯННЯ АНАЛОГІВ

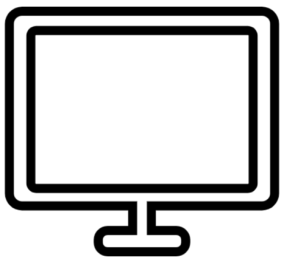
Google Drive

Apple iCloud

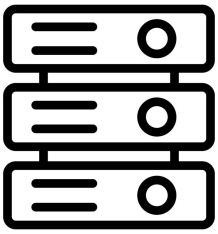
Microsoft OneDrive

Слайд 5

ВИКОРИСТАНІ ТЕХНОЛОГІЇ



Front-End



Back-End

File storing

Слайд 6

FRONT-END ТЕХНОЛОГІЇ

Next.js

- SSR (Server side rendering)
- SEO (Search engine optimisation)
- Безпека

Tailwind CSS

- Підтримка з коробки
- Швидкість розробки

Слайд 7

BACK-END ТЕХНОЛОГІЇ

Node.js

- Швидкість розробки
- Одноманітність кодової бази

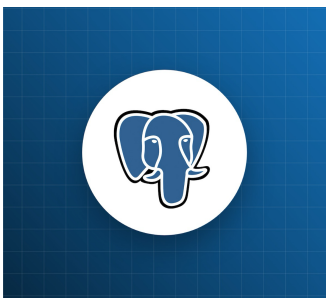
Express.js

- Велика кількість доступних бібліотек
- Проста маршрутизація
- Потужні middleware

7

Слайд 8

FILE STORING ВИРІШЕННЯ



Postgre SQL

AWS S3

8

Слайд 9

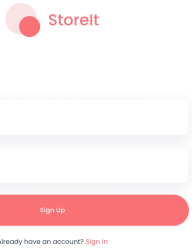
ОСНОВНИЙ ФУНКЦІОНАЛ

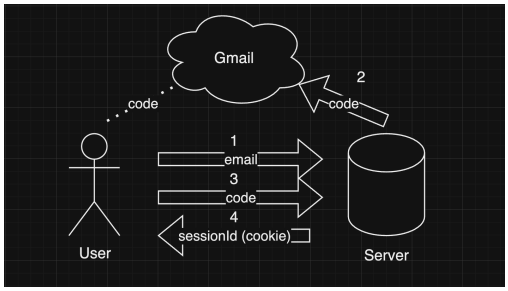
- Аутентифікація
- Зававтаження файлів
- Керування файлами

Слайд 10

АУТЕНТИФІКАЦІЯ

1. Введення даних користувача
2. Створення запису в базі даних
3. Надсилання OTP на email
4. Верифікація OTP
5. Створення сесії
6. Надсилання ID сесії за допомогою http-only cookie





Слайд 11

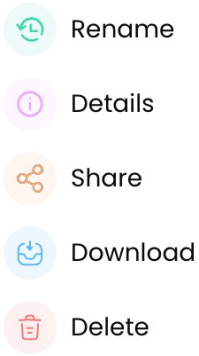
СТВОРЕННЯ ФАЙЛІВ

1. Відправка файлу
2. Обробка даних файлу
3. Збереження файлу в S3-Bucket
4. Збереження метаданих файлу в базі даних

Слайд 12

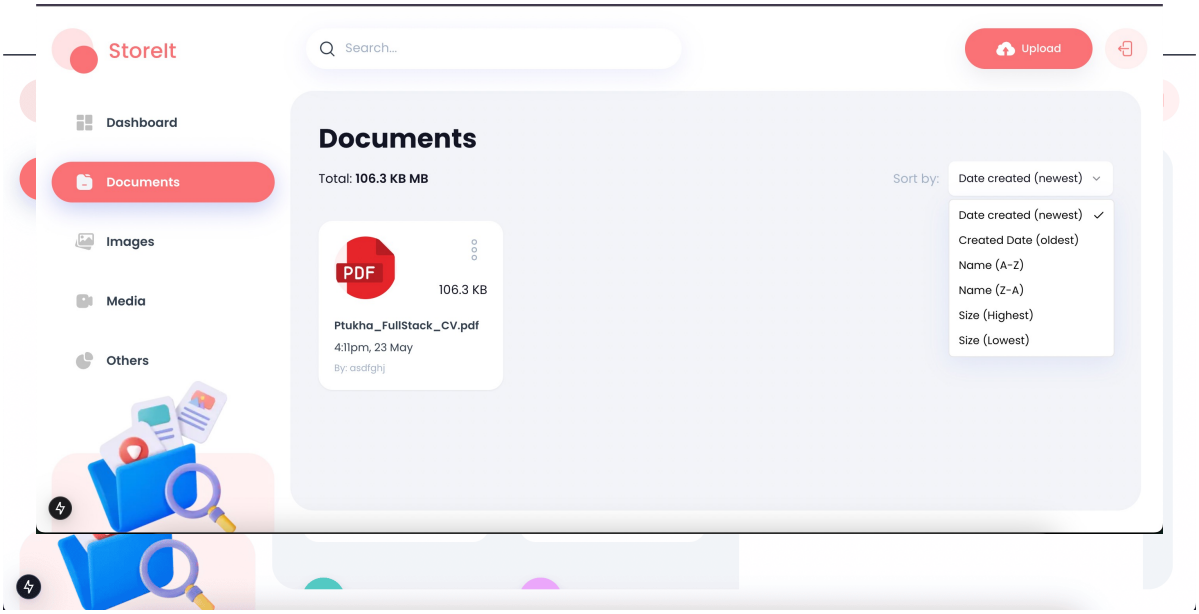
КЕРУВАННЯ ФАЙЛАМИ

1. Зміна назви
2. Додавання користувачів файлу
3. Видалення файлу
4. Завантаження файлу
5. Видалення користувачів файлу



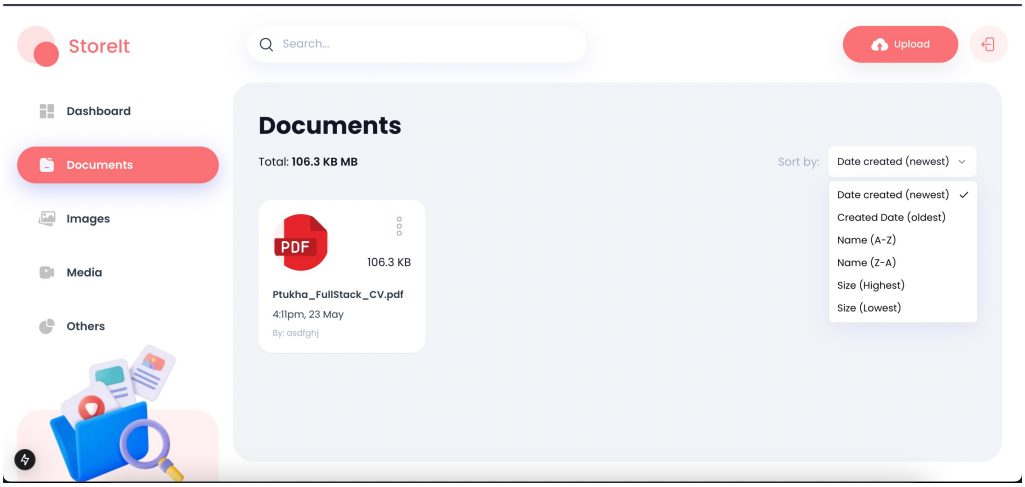
Слайд 13

ФІНАЛЬНИЙ ВИГЛЯД ГОЛОВНОЇ СТОРІНКИ



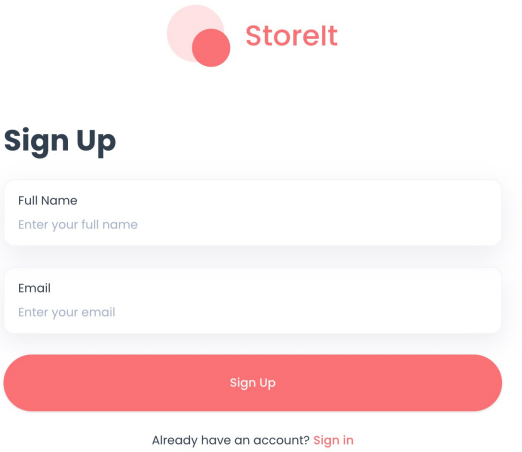
Слайд 14

СТОРІНКА З СОРТУВАННЯМ ФАЙЛІВ



Слайд 15

СТОРІНКА АУТЕНТИФІКАЦІЇ



Слайд 16

ВИСНОВКИ

- Досліджено методи розробки сучасних хмарних сховищ
- Проаналізовано стратегії розробки хмарних сховищ
- Порівняно сучасні технології веб-розробки
- Застосовано технології розробки сучасних веб-додатків, а саме хмарних сховищ
- Розроблено систему аутентифікації для додатку хмарного сховища
- Розроблено систему збереження, відображення та керування файлами хмарного сховища