Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня
магістра
(назва освітньо-кваліфікаційного рівня)

студента	*Йолкіна Єгора Вадимовича*
	(ПІБ)
академічної групи	*122м-22-2*
	(шифр)
спеціальності	*122 Комп'ютерні науки*
	(код і назва спеціальності)
освітньої програми	*«122 Комп'ютерні науки»*
	(назва освітньої програми)
на тему:	*Дослідження та оптимізація бізнес-процесів з використанням хмарних сховищ*

_____ *Йолкін Єгор Вадимович*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
розділів кваліфікаційної роботи	*Доц. Приходченко С.Д.*			
спеціальний	*Доц. Приходченко С.Д.*			
економічний				
Рецензент				
Нормоконтролер	*проф. Лактіонов І.С.*			

Дніпро
2023

Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
Завідувач кафедри
Програмного забезпечення комп'ютерних систем
(повна назва)

_____ М.О. Алексєєв
(підпис) (прізвище, ініціали)

« » _____ 20 23 Року

ЗАВДАННЯ
на виконання кваліфікаційної роботи

спеціальності *122 Комп'ютерні науки*
(код і назва спеціальності)

студенту *122м-22-2* *Йолкін Єгор Вадимович*
(група) (прізвище та ініціали)

Тема кваліфікаційної роботи *Дослідження та оптимізація бізнес-процесів з використанням хмарних сховищ*

1 ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Наказ ректора НТУ «Дніпровська політехніка» від 31.10.2023 р. № 1227-с

2 МЕТА ТА ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Об'єкт досліджень – бізнес-процеси, що включають в себе використання хмарних сховищ

Предмет досліджень – аналіз впливу хмарних сховищ на продуктивність та ефективність бізнес-процесів

Мета НДР – оптимізація бізнес-процесів і підвищення ефективності завдяки використанню хмарних сховищ.

Вихідні дані для проведення роботи – теоретичні та експериментальні дослідження, дані про вже існуючі бізнес-процеси та хмарні сховища.

3 ОЧІКУВАНІ НАУКОВІ РЕЗУЛЬТАТИ

Новизна запропонованих рішень полягає в розробці та застосуванні сучасних методів реалізації програмного забезпечення для удосконалення вже існуючих засобів для створення хмарних сховищ, за рахунок підвищення надійності та безпечності даних, з дотриманням всіх вимог щодо захисту особистих даних.

Практична цінність полягає в тому, що хмарні сховища можуть допомогти бізнесу оптимізувати використання ресурсів шляхом підвищення доступності даних та можливостей працювати з ними у будь-який час, з будь-якого місця та пристрою. Крім того, хмарні сховища допоможуть зменшити ризик втрати конфіденційних даних та запобігатимуть ризику порушень безпеки особистої інформації.

4 ВИМОГИ ДО РЕЗУЛЬТАТІВ ВИКОНАННЯ РОБОТИ

Результати досліджень мають бути подані у вигляді, що дозволяє побачити та оцінити безпосереднє використання серверної та клієнтської сторін розробки. В результаті роботи повинне бути розроблене хмарне сховище, яке забезпечує надійну роботи з даними та є безпечне для користувачів.

5 ЕТАПИ ВИКОНАННЯ РОБІТ

Найменування етапів робіт	**Строки виконання робіт (початок – кінець**
Аналіз теми та постановка задачі	12.09.2023-30.09.2023
Збір, дослідження та систематизація інформації щодо проектування та розробки хмарного сховища	01.10.2023-31.10.2023
Розробка і тестування програмного забезпечення для хмарного сховища	01.11.2023-12.12.2023

6 РЕАЛІЗАЦІЯ РЕЗУЛЬТАТІВ ТА ЕФЕКТИВНІСТЬ

Економічний ефект від реалізації результатів роботи очікується позитивним завдяки розробці програмного забезпечення для хмарного сховища очікується зниження витрат на інфраструктуру та обладнання, що забезпечить більшу ефективність та конкурентоспроможність бізнесу.

Соціальний ефект від реалізації результатів роботи очікується позитивним, завдяки підвищенню доступності роботи, яке забезпечує хмарне сховище, це буде сприяти розвитку роботи з віддалених робочих місць та дозволить складати більш гнучкий графік, що покращить якість життя працівникам і дозволить значній кількості людей працювати більш зручніше.

7 ДОДАТКОВІ ВИМОГИ

Завдання видав _____ *Приходченко С.Д*
(підпис) (прізвище, ініціали)

Завдання прийняв до виконання _____ *Йолкін Є.В*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *12.09.2023 р.*

Термін подання кваліфікаційної роботи до ЕК *12.12.2023*

РЕФЕРАТ

Пояснювальна записка: 88 стор., 29 рис., 3 таблиць, 4 додатка, 25 джерел.

Об'єкт дослідження: вплив та використання хмарних сховищ для оптимізації бізнес-процесів.

Предмет дослідження: хмарні сховища та їхній вплив на оптимізацію бізнес-процесів.

Мета магістерської роботи: аналіз та дослідження поточного стану бізнес-процесів у контексті використання хмарних сховищ, а також пошук можливостей для оптимізації бізнес-процесів з метою підвищення ефективності, продуктивності підприємства.

Методи дослідження. Для досягнення поставленої були запроваджені такі методи дослідження як: аналіз поточного стану бізнес-процесів, SWOT-аналіз, вивчення можливостей хмарних платформ, міграція даних, розробка архітектури хмарної системи.

Новизна отриманих результатів полягає в дослідженні та створенні оптимізованої хмарної системи, яка сприяє збільшенню ефективності бізнес-процесів та зменшенню витрат.

Практична цінність полягає в можливості використання розробленої хмарної системи для покращення роботи організації та оптимізації витрат.

Область застосування: За результатом кваліфікаційної роботи розроблена хмарна система може бути використана для підвищення ефективності та конкурентоспроможності бізнесу у сучасних умовах.

Значення роботи та висновки. Впровадження розробленої хмарної системи може призвести до значного покращення ефективності та продуктивності бізнес-процесів, що відобразиться на збільшенні прибутковості та конкурентоспроможності підприємства.

Список ключових слів: хмарне сховище, бізнес-процеси, база даних, оптимізація бізнес-процесів, міграція даних.

ABSTRACT

Explanatory note: 88 pp., 29 Fig., 4 app., 25 sources.

Object of research: the impact and use of cloud storage for business process optimization.

Subject of research: cloud storage and its impact on business process optimization.

Purpose of the master's thesis: to analyze and study the current state of business processes in the context of using cloud storage, as well as to find opportunities for optimizing business processes in order to increase the efficiency and productivity of the enterprise.

Research methods. To achieve this goal, the following research methods were introduced: analysis of the current state of business processes, SWOT analysis, study of the capabilities of cloud platforms, data migration, development of the cloud system architecture.

The novelty of the obtained results lies in the study and creation of an optimized cloud system that helps to increase the efficiency of business processes and reduce costs.

The practical value lies in the possibility of using the developed cloud system to improve the organization's performance and optimize costs.

Scope of application: Based on the results of the qualification work, the developed cloud system can be used to improve the efficiency and competitiveness of business in modern conditions.

Significance of the work and conclusions. The implementation of the developed cloud system can lead to a significant improvement in the efficiency and productivity of business processes, which will be reflected in the increase in the profitability and competitiveness of the enterprise.

Keywords: cloud storage, business processes, database, business process optimization, data migration.

ЗМІСТ

ВСТУП

Сучасний бізнес відчуває гостру потребу у вдосконаленні та оптимізації своїх процесів в умовах стрімкого розвитку технологій. Однією з потужних і перспективних технологій, що мають суттєвий вплив на різні сфери бізнесу, є використання хмарних сховищ. Ця технологія дає змогу ефективно та безпечно зберігати, опрацьовувати й обмінюватися даними та відкриває не обмежені можливості для оптимізації бізнес-процесів.

Зацікавленість у даній темі пов'язана, з реальної необхідності у впровадженні ефективних інструментів для підвищення продуктивності та зменшення витрат підприємства. Аналіз та дослідження показали, що хмарні сховища дійсно можуть стати вирішальним елементом для досягнення цих цілей. Саме досягнення цілей дослідження дозволить підприємствам оптимізувати свою діяльність та підвищити конкурентоспроможність на світовому ринку.

Метою цього дослідження є визначення потенціалу використання хмарних сховищ для бізнес-процесів та розробка оптимальних стратегій їх впровадження та використання. Оригінальність даної роботи буде практична спрямованість отриманих результатів та можливість їх впливу на підвищення ефективності та конкурентоспроможності підприємств в сучасних умовах глобального ринку.

Сучасні підприємства не можуть залишатися непереможними, якщо не вдосконалювати та оптимізувати свої процеси. Зі стрімким зростанням обсягу створюваних даних, який за прогнозами IDC, зросте в десять разів до 2025 року, нагальна потреба в ефективному зберіганні, обробці й обміні даними буде більшою, ніж будь-коли.

За даними RightScale, аналітичні звіти показують, що 94% організацій вже використовують хмарні сервіси, усвідомлюючи та підтримуючи цю потребу [1]. Це свідчить про широке використання хмарних технологій у сучасному бізнесі. Враховуючи постійний пошук ефективності, дані Gartner показують, що понад 60% підприємств визнають, що оптимізація та підвищення ефективності бізнес-процесів є головним пріоритетом [2].

Тому важливим фактором є економія ресурсів. Використання хмарного сховища дозволяє компаніям заощаджувати витрати на інфраструктуру, оскільки їм не потрібні власні великі центри обробки даних. В умовах глобалізації та мобільності бізнесу централізоване зберігання та обмін даними стало надзвичайно важливим для успішної роботи компаній у різних частинах світу. Також потрібно враховувати високий рівень безпеки, який пропонує хмарне сховище, і можливість автоматичного резервного копіювання даних. Це дозволяє уникнути втрати важливої інформації в найвідповідальніші моменти.

Крім того, різні галузі, такі як фінанси, охорона здоров'я та освіта, мають особливі потреби у зберіганні й обробці даних, які можна ефективніше задовольнити за допомогою хмарних технологій.

У сукупності ці аналітичні факти свідчать про те, що оптимізація бізнес-процесів за допомогою хмарних сховищ є актуальною та критичною задачею для багатьох компаній у сучасних умовах.

В даній кваліфікаційній роботі метою є дослідження та впровадження хмарних технологій з метою підвищення ефективності бізнес-процесів.

У першому розділі проведено аналіз сучасного стану предметної області та розглянуто основні проблеми, які виникають у бізнес-процесах. Особлива увага була приділена аналізу можливостей та переваг використання хмарних сховищ для оптимізації бізнес-процесів.

У другому розділі було проаналізовано підходи, методи та інструменти, які можна використовувати для впровадження хмарних технологій у бізнес-процесах. Особливу увагу приділили вибору та конфігурації хмарних рішень для конкретних потреб підприємства.

У третьому розділі розроблена та впроваджена хмарна система, яка стала прикладом ефективної оптимізації бізнес-процесів. Ця система дозволяє зберігати, обробляти та обмінюватися даними з використанням хмарних технологій, що прискорює робочі процеси та зменшує витрати.

Результати досліджень і розробок хмарних систем підкреслюють важливість впровадження сучасних технологій для оптимізації бізнес-процесів.

Створена система є яскравим прикладом успішної оптимізації та підвищення продуктивності в сучасних умовах ведення бізнесу.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Загальні відомості з предметної галузі

Хмарні технології є інноваційним підходом до обчислень і зберігання даних, який використовує Інтернет для надання послуг і ресурсів через віртуальну хмару. Ця концепція дозволяє користувачам отримувати доступ до таких ресурсів, як сервери, сховища, програмне забезпечення та дані через Інтернет, не обмежуючись фізичною інфраструктурою.

Ідея хмарних технологій з'явилася в 1960-х роках, в момент коли інженери досліджували концепцію “довжини хвилі”. В основі цього дослідження була ідея, що обчислення та доступ до комп'ютерних ресурсів та даних мають надаватись так само, як послуги електроенергії та водопостачання – за вимірюванням та за необхідністю. Проте на той час технології та інфраструктура не біли на стільки розвинені для впровадження такого підходу. З часом ідея концепції “довжини хвилі” стала більш практичною [3].

Значний прорив відбувся у 2006 році, коли компанія Amazon розпочала еру хмарних обчислень, створивши перший великий комерційний хмарний сервіс Amazon Web Services (AWS). За перші п'ять років існування AWS приніс дохід у розмірі близько 5,6 мільярдів доларів США, що свідчить про величезний попит на хмарні сервіси. Пізніше Microsoft та Google також запустили свої хмарні платформи.

Згідно зі звітом Flexera 2023 State of the Cloud Report, 93% організацій використовують публічні хмарні сервіси, а 87% - принаймні одну приватну хмару [4].

Один з ключових висновків цього звіту полягає в тому, що управління витратами в хмарі стало головною проблемою для організацій. Вперше за останнє десятиліття управління витратами випередило важливість безпеки.

Частково це пов'язано з економічною невизначеністю, яка змушує організації уважніше придивлятися до своїх хмарних бюджетів. Цікавою

тенденцією є розвиток мультихмарних стратегій. У звіті показано, що компанії все частіше використовують кілька хмарних провайдерів. Це може бути викликано бажанням підвищити гнучкість, знизити ризики або знайти оптимальне співвідношення між ціною і продуктивністю.

Дослідження та оптимізація бізнес-процесів з використанням хмарних сховищ є надзвичайно актуальною та важливою темою в сучасному бізнес-середовищі. Насамперед, хмарні технології дають змогу підприємствам ефективно та безпечно зберігати й обробляти великі обсяги даних. Це дуже важливо в сучасному світі постійно зростаючих обсягів даних.

Згідно з дослідженням, проведеним International Data Corporation (IDC) у 2021 році, близько 60% підприємств у світі вже використовують хмарні сховища для зберігання та обробки даних. І ці показники продовжують зростати. Хмарні рішення дають компаніям можливість масштабувати свою діяльність і скорочувати витрати на інфраструктуру [5].

Ще однією причиною актуальності дослідження та оптимізації бізнес-процесів у хмарних сховищах є підвищення продуктивності та зниження кількості помилок. За даними Forbes Insights, 81% компаній вважають, що завдяки хмарним обчисленням була підвищення продуктивність підприємства. Це стосується як великих, так і малих підприємств. Оптимізація бізнес-процесів у хмарі дає змогу автоматизувати рутинні завдання, забезпечити доступ до даних із будь-якої точки та підвищити оперативність реагування на зміни ринку.

Використання хмарних сховищ для оптимізації бізнес-процесів стало важливим напрямком розвитку сучасного бізнесу. Попередні дослідження вже підтвердили, що такий підхід може зробити значущий внесок у підвищення ефективності та зниження витрат.

Наприклад за допомогою хмарних сховищ, підприємства можуть забезпечити доступність даних з будь-якого місця та пристрою. Згідно з IDC, до 2025 року 60% світових даних буде зберігатися в хмарних сховищах, що дасть підприємствам більше можливостей для оптимізації робочих процесів.

Не менш важливим аспектом є використання хмарних сховищ для захисту конфіденційності та цілісності даних. За допомогою розвинутих засобів шифрування та захисту, хмарні послуги стають надійними. За дослідженням McAfee, 52% організацій відзначають покращення безпеки завдяки переходу до хмарних сховищ [6].

У загальному зв'язок між хмарним сховищем та оптимізацією бізнес-процесів дуже важливий і може призвести до значних переваг для підприємств та компаній. Хмарне сховище перш за все забезпечує зручний доступ до даних зберігаючи самі дані в онлайн-середовищі та дає можливість отримувати ці дані з любого місця чи пристрою.

Також хмарні сховища легко масштабуються в залежності від потреби підприємства. Можливість збільшувати або зменшувати обсяг сховища дозволяє підприємству ефективно використовувати ресурси та уникати надмірного вкладення коштів в апаратне забезпечення.

Слід зауважити що аспект безпеки даних в хмарному сховищі є одним з найважливіших речей. Якщо повернутися до звіту Flexera 2023 State of the Cloud Report, то актуальність безпеки виноситься на перший план у сфері хмарних сховищ. Багато хмарних сховищ надають різні рівні захисту даних. Наприклад, це включає в себе шифрування даних, впровадження механізмів автентифікації та системи контрою доступу. Забезпечивши належну безпеку даних, хмарні сховища допомагають запобігти витоку інформації [7].

1.2. Огляд та аналіз існуючих аналогів хмарних сховищ

У сучасному світі використання хмарних сховищ стало невід'ємною частиною цифрового життя. Безліч платформ дозволяє користувачам зберігати, керувати та спільно використовувати свої дані та файли в інтернеті. На світовому ринку існує безліч аналогів хмарних сховищ та деякі із них ми розглянемо, а також проведемо порівняльний аналіз їхніх основних характеристик та можливостей [8].

Один з найвідоміших і поширених сервісів хмарного сховища є Dropbox. Далі ми розглянемо переваги та недоліки Dropbox як хмарного сховища, а також окреслимо деякі цікаві технології, які роблять Dropbox особливим на фоні його аналогів.

Dropbox дійсно має безліч переваг, як приклад можна виділити такі:

- Спрощена синхронізація та доступ до даних: Dropbox надає користувачам можливість синхронізувати файли між різними пристроями, що робить доступ до даних зручним та доступним у будь-який час і в будь-якому місці. Користувач може легко отримувати доступ до своїх файлів через веб-інтерфейс або спеціальні додатки для різних платформ.
- Спільна робота над файлами: Dropbox дозволяє користувачам спільно працювати над документами та проектами в реальному часі. Є можливість додавати коментарі, відгуки та редагувати файли разом з іншими користувачами, що робить його ідеальним інструментом для командної роботи та колаборації.
- Захист даних і безпека: Dropbox використовує передові методи шифрування для захисту даних під час їх передачі та зберігання. Крім того, сервіс надає можливість використовувати двоетапну аутентифікацію для додаткового рівня безпеки. Користувач завжди впевнений в конфіденційності своїх даних.
- Спрощена інтеграція з іншими додатками: Dropbox інтегрується з багатьма популярними додатками та сервісами, що дозволяє користувачеві зручно працювати зі своїми даними та автоматизувати різні завдання. Це робить Dropbox відмінним вибором для користувачів, які використовують різноманітні програми.
- Можливості роботи офлайн: Dropbox дозволяє користувачеві працювати з файлами в офлайн-режимі, що особливо корисно для тих, хто потребує доступ до даних без Інтернет-з'єднання.

Також можна виділити декілька значущих недоліків які є у Dropbox, а саме:

- Обмежена безкоштовна версія: Однією з основних проблем Dropbox є обмеження безкоштовної версії, яка обмежує обсяг зберігання даних. Для користувачів, які потребують більше простору для зберігання, вони можуть бути змушені перейти на платний план, що може призвести до додаткових витрат.
- Висока ціна платних планів: Платні плани Dropbox можуть бути відносно дорогими в порівнянні з іншими аналогічними хмарними сховищами. Особливо для бізнес-користувачів і великих компаній витрати на платний план можуть стати значними.
- Обмеженість функцій без інтернет-з'єднання: Використання Dropbox в офлайн-режимі обмежене, і ви не можете редагувати файли або синхронізувати дані без доступу в Інтернет. Це може бути нестачею для користувачів, які часто потребують робити зміни в документах поза мережею.
- Конфлікти в спільній роботі: Під час спільної роботи над файлами у Dropbox можуть виникати конфлікти в разі одночасних змін одним чи декількома користувачами. Це може призвести до втрати даних або надмірного часу на вирішення конфліктів.
- Синхронізація може займати багато часу: Залежно від обсягу та швидкості Інтернет-з'єднання, синхронізація великих файлів у Dropbox може займати значний час, що може призвести до затримок та неефективності.

Dropbox також використовує декілька особливих технологій, що відрізняє його від аналогів. Наприклад, Dropbox використовує Btrfs (B-tree (рис.1.1) файлову систему) для збереження та керування даними користувачів. Btrfs є сучасною файловою системою, яка дозволяє виявляти та виправляти пошкоджені файли, зменшує ризик втрати даних та забезпечує надійність збережених даних. Також Btrfs підтримує створення снапшотів файлової системи, які дозволяють зберегти стан системи в певний момент часу. Це корисно для резервного копіювання та відновлення даних [9].

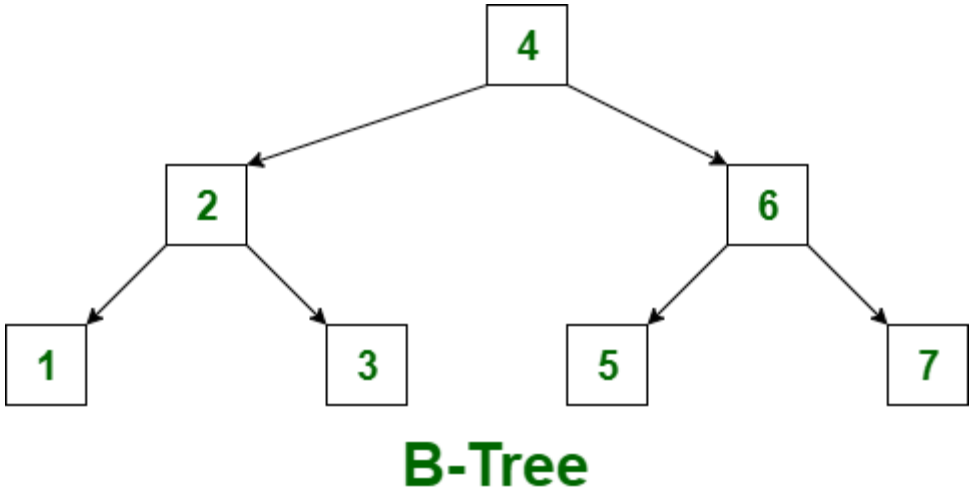

Рис.1.1. Файлова система B-Tree

Ще однією цікавою технологією є Smart Sync. Ця технологія дозволяє користувачам зберігати файли в Dropbox, не витрачаючи простір на своєму пристрої. Файли можуть бути доступні для перегляду, але завантажуються лише тоді, коли вони фактично потрібні. Це допомагає економити місце на пристроях та зберігати дані в безпеці. Smart Sync дозволяє командам спільно працювати над великими обсягами даних, не перевантажуючи пристрої зберігання, і забезпечує швидкий доступ до ресурсів. Подібна технологія робить користування Dropbox більш ефективним та зручним для користувачів, особливо тих, які працюють з обмеженим диском на своїх пристроях або потребують доступ до великих обсягів даних у хмарі.

Microsoft OneDrive є ще однією найвідомішою системою хмарного сховища. Ця платформа надає зручний і надійний доступ до файлів та даних через хмарне сховище. Значною особливістю Microsoft OneDrive є тісна інтеграція з екосистемою Microsoft, а також значному спектрі функціональних можливостей. Якщо брати до уваги переваги Microsoft OneDrive, то можна виділити такі:

– Інтеграція з Microsoft 365: Однією з найважливіших переваг OneDrive є його інтеграція з іншими продуктами компанії Microsoft, наприклад такими як Word, Excel, PowerPoint і Outlook. Це дозволяє

користувачам зручно маніпулювати документами, редагувати їх, а також працювати над проектами у зручному інтерфейсу.

- Доступність на різних платформах: Microsoft OneDrive підтримує роботу на багатьох операційних системах, включаючи Windows, macOS, Android і iOS. Це робить його доступним для користувачів різних пристроїв.
- Завантаження великих файлів: У Microsoft OneDrive є можливість завантажувати великі файли та навіть папки. Така система спрощує спільну роботу над великим обсягом даних.
- Зручний доступ через веб-інтерфейс: Доступ до файлів можна отримати через веб-інтерфейс навіть з комп'ютерів без встановленого спеціального додатку.

Також розглянемо недоліки які є в Microsoft OneDrive.

- Надмірна інтеграція з Windows: Користувачам які не використовують операційну систему Windows, деякі функції OneDrive можуть бути менш зручними або не так доступними.
- Захист даних і шифрування: OneDrive використовує шифрування даних в покладеному стані (data at rest) та під час передачі даних (data in transit). Однак, ключі шифрування управляються самою Microsoft, що означає, що компанія має технічну можливість розшифрувати дані користувачів за необхідності. Це може створювати певні ризики, особливо для корпоративних клієнтів.

Microsoft OneDrive використовує сучасні та інноваційні технології у своїй системі хмарного сховища.

Наприклад Files On-Demand, це функція в системі хмарного сховища Microsoft OneDrive, яка пропонує зручну роботу з файлами, дозволяючи користувачам ефективно використовувати локальний обсяг диска та зменшувати потребу в завантаженні файлів на локальний пристрій. Основна ідея Files On-Demand полягає в тому, щоб файли в хмарі OneDrive не завантажувалися автоматично на локальний пристрій користувача. Замість цього, файли

залишаються у хмарі, а їх іконки та імена відображаються на локальному комп'ютері.

Також Microsoft OneDrive використовує штучний інтелект у своїй системі для покращення функціональності та полегшення роботи користувачів з файлами. Завдяки штучному інтелекту OneDrive автоматично індексує та класифікує файли, що надає можливість користувачам швидко знаходити потрібні файли.

ШІ може розпізнавати зображення та відео, індексувати їх і навіть створювати ключові слова або описи для полегшення пошуку.

Треба зауважити що Microsoft OneDrive призначений переважно для зберігання та обміну файлами, а ось наприклад Google Cloud - це хмарна платформа, яка дозволяє створювати та розгортати різні хмарні додатки та обчислення.

Google Cloud також є однією з популярних платформ для хмарних обчислень, а також сховищ в інфраструктурі хмарних обчислень. Google Cloud пропонує безліч різноманітних послуг для корпорацій та звичайних користувачів.

У Google Cloud є значна кількість переваг в порівнянні зі своїми аналогами, наприклад як:

- Широкий вибір послуг: Google Cloud надає багато різноманітних послуг, включаючи обчислення, сховище даних, бази даних, інструменти розробки, машинне навчання, аналітику даних та інше. Це дає можливість користувачам обирати саме ті послуги, які відповідають їхнім потребам.
- Інновації: У Google завжди шукають шлях до покращення своїх технологій або пошуку інноваційних рішень в сфері інформаційних технологій. Google Cloud включає в себе інноваційні можливості, такі як BigQuery для аналізу великих обсягів даних, а також Anthos для оркестрації контейнерів, що робить його дуже привабливим для сучасних бізнесів.

- Безпека: Google приділяє значні ресурси для забезпечення належної безпеки для даних користувачів. Google Cloud використовує передові технології для захисту даних користувачів, включаючи різноманітні інструменти для автентифікації та шифрування.

Google Cloud також має недоліки, наприклад:

- Вартість: Google Cloud може бути дорожчим, ніж інші хмарні платформи, особливо при використанні великої кількості ресурсів і додаткових послуг. Важливо пам'ятати про витрати.
- Локалізація даних: У деяких регіонах можуть виникнути питання щодо вимог до зберігання даних у межах їхніх національних кордонів. Для деяких користувачів це може бути суттєвим обмеженням.
- Залежність від інтернет-з'єднання: Хмарні сервіси, зокрема Google Cloud, потребують стабільного з'єднання з Інтернетом. Якщо воно недоступне, доступ до даних і сервісів може бути ускладнений.

Google Cloud є новаторами у своїй сфері, вони продовжують розвивати свої функції та послуги, щоб задовольняти потреби різних користувачів і галузей. Google Cloud визначається своїми досягненнями в галузі штучного інтелекту та машинного навчання. Також Google Cloud надає широкий спектр інструментів для розробки та розгортання моделей штучного інтелекту та машинного навчання, наприклад послуга AI Platform дозволяє розробляти, навчати та розгортати моделі машинного навчання. Вона надає інструменти для роботи з даними, навчання моделей та автоматизації процесу.

Google Cloud AutoML надає доступ до машинного навчання без необхідності поглиблених знань про машинне навчання. Ви можете створювати власні моделі без глибоких навичок програмування.

Також є інноваційним рішенням від Google Cloud, яке спрямоване на оркестрацію контейнерів та управління додатками у різних хмарних середовищах та локальних центрах обробки даних. Anthos (рис.1.2) розширює можливості контейнерної оркестрації, основаної на Kubernetes, на гібридні та багатохмарні інфраструктури.

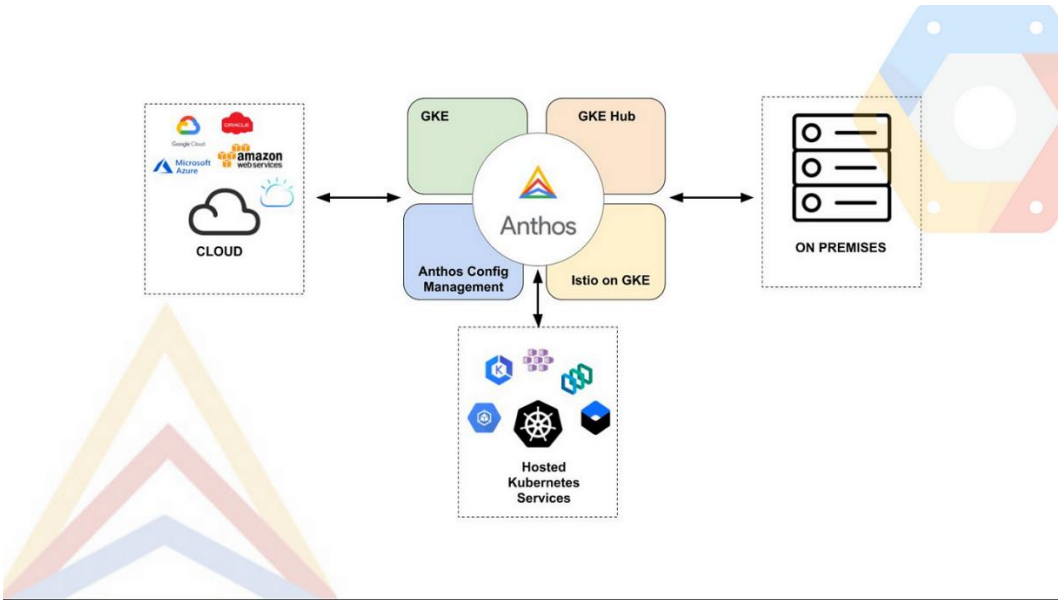

Рис.1.2. Мультихмарне та мультикластерне управління — Anthos

1.3. Постановка задачі

Задача полягає у проведенні дослідження та подальшій оптимізації бізнес-процесів вдаючись до використанням хмарних сховищ. Основна мета полягає у розробці та впровадженні хмарних сховищ як конкретного рішення для оптимізації бізнес-процесів. Розроблена хмарна система повинна надавати користувачеві наступні можливості: можливість реєстрації та авторизації користувача у хмарній системі, можливість авторизованим користувачам завантажувати, вивантажувати та видаляти файли у хмарному сховищі, для забезпечення більш ефективної та зручною роботи з хмарним сховищем у користувача повинна бути можливість фільтрувати файл по їх розширенню. Належна увага повинна приділятись також і безпеці даних у хмарному сховищ, завантажених файлів, даних про користувача.

1.4. Вимоги до програми або програмного виробу

1.4.1. Вимоги до функціональних характеристик

Одною з головних умов якій повинна відповідати хмарне сховище є забезпечення можливості зберігати та керувати даними в хмарній системі.

Система повинна надавати можливість користувачам вивантажувати, завантажувати, видаляти різні типи файлів таки як текстові документи, зображення, відео, архіви тощо.

Не менш важливою є вимога забезпечення користувачеві зручного та інтуїтивно зрозумілого інтерфейсу, де буде можливість фільтрації даних, а саме забезпечення процесу вибору певних файлів або даних з хмарного сховища на основі певних критеріїв. Це забезпечить користувачу більш ефективнішу роботу з хмарним сховищем, шляхом економії часу.

Забезпечення хмарного сховища функціоналом реєстрації та авторизації також є невід'ємною вимогою до будь-якої хмарної системи. Це важливо для забезпечення безпеки для даних та запобігає несанкціонованому доступу до особистих файлів користувача. Кожен користувач повинен мати унікальній обліковий запис з власними правами доступу. Реєстрація дає можливість ідентифікувати користувача у хмарній системі, що є важливим для створення особистого облікового запису та що допомагає пов'язати користувача з його особистими даними, які є також конфіденційні.

1.4.2. Вимогу до інформаційної безпеки

Важливим аспектом у розробці хмарної системі є забезпечення захисту конфіденційності, цілісності та доступності даних. Головною вимогою є забезпечення аутентифікацію користувача за допомогою JWT (JSON Web Token). Це гарантує, що лише автентифіковані користувачі можуть завантажувати файли. Для кожного збереженого файлу повинна бути функція яка генерує унікальний ідентифікатор. Це допоможе запобігти перезапису файлів.

1.4.3. Вимоги до складу та параметрів технічних засобів

Технічними умовами для роботи фінальної версії програмного продукту, а саме хмарного сховища є:

- обсяг оперативної пам’яті не менш за 2 гб;

- операційна система Windows, Linux, MacOS, IOS, Android;
- наявність сучасного браузера, наприклад, Google Chrome, Opera, Safari;
- підтримка високошвидкісного Інтернет з’єднання.

1.4.4. Вимоги до інформаційної та програмної сумісності

Для програмної сумісності обов’язковою вимогою є наявність останньої версії мережевого драйверу. Також важливим фактором є наявність стабільного підключення до високошвидкісного Інтернет з'єднання. Та наявність будь-якого сучасного браузера на девайсі. Наприклад: Opera, Safari, Google Chrome тощо.

РОЗДІЛ 2

АНАЛІЗ МЕТОДІВ РОБОТИ ТА ХМАРНИХ ТЕХНОЛОГІЙ

2.1. Аналіз сучасних хмарних рішень в бізнес-процесах

Впровадження хмарних сховищ в бізнес-процеси є необхідністю у сучасному світі. Завдяки хмарним сховищам підприємство може забезпечити ефективну і зручну роботу, збільшивши при цьому свою конкурентоспроможність на ринку та знизити тиск, витрати на IT-інфраструктуру. Впровадження хмарних рішень в бізнес надають можливість підприємствам зберігати велику кількість даних, не переймаючись за їх фізичний стан [10].

Розглянемо основні аспекти впровадження хмарних сховищ та визначимо основну їх роль у сучасних бізнес-процесах, а також зробимо огляд різних видів хмарних сховищ, включаючи публічні, гібридні та приватні.

Як і локальне мережеве сховище, хмарне сховище використовує сервери для зберігання даних, однак дані будуть надіслані на сторонні сервери. Більшість серверів, якими ми користуєтеся, є віртуальними машинами, розміщеними на фізичних серверах. У міру збільшення попиту на сховище постачальники створюватимуть нові віртуальні сервери, щоб задовольнити попит.

Як правило, ми підключаємося до хмарного сховища через Інтернет або спеціальне приватного з’єднання за допомогою веб-порталу, веб-сайту або мобільного додатка. Сервер, до якого ми підключаємося, передає наші дані на пул серверів, розташованих в одному або кількох центрах обробки даних, залежно від масштабу операцій хмарного провайдера.

У рамках послуги постачальники зазвичай зберігають однакові дані на кількох машинах для резервування. Таким чином, якщо сервер вимкнено на технічне обслуговування або зазнає збою, ми все одно зможете отримати доступ до своїх даних.

У загальному є три види хмарних сховищ, а саме публічні, приватні та гібридні.

Публічні хмарні сховища: у цій моделі ми підключаємося через Інтернет до хмарного сховища, яке підтримується постачальником хмарних технологій і використовується іншими компаніями. Вони зазвичай базуються на масштабованих інфраструктурах, які спільно використовуються багатьма клієнтами. Прикладами публічних хмарних сховищ є Amazon Web Services (AWS), Microsoft Azure та Google Cloud Platform (GCP) [11].

Приватні хмарні сховища: Приватні хмарні сховища - це інфраструктура хмарного сховища, що належить і управляється однією компанією або організацією. Вони зазвичай надають більшу контроль над даними та безпекою, але можуть бути дорогими у вдосконаленні та обслуговуванні. Приватні хмарні сховища дозволяють компаніям зберігати дані в інфраструктурі, яку вони керують.

Гібридні хмарні сховища: Гібридні хмарні сховища поєднують в собі як публічні, так і приватні хмарні ресурси, щоб надавати користувачам більшу гнучкість і можливість керування. Це дозволяє компаніям зберігати чутливі дані в приватних хмарних сховищах, а менш критичні дані - в публічних хмарних сховищах. Наприклад, суворо регульовані дані, які підлягають суворим вимогам до архівування та реплікації, зазвичай більше підходять для приватного хмарного середовища, тоді як менш конфіденційні дані (такі як електронна пошта, яка не містить комерційної таємниці) можуть зберігатися в загальнодоступній хмарі. Деякі організації використовують гібридні хмари, щоб доповнити свої внутрішні мережі зберігання загальнодоступними хмарними сховищами.

Бізнес-процеси в цілому можна класифікувати на: людино-центричні та системно-центричні з поєднанням: людина-людина (P2P), людина-додаток (P2A) та додаток-додаток (A2A).

Розглянемо процеси A2A в програмних системах, які охоплюють: системи обробки транзакцій, платформи інтеграції архітектури підприємства (EAI) та

веб-інтеграційні сервери. Ця програмна архітектура дозволяє отримати доступ до всіх різних бізнес-процесів динамічно через протокол, відомий як сервіс-орієнтована архітектура (SOA) і пристосований до бізнес-потреб компанії. Використання веб-орієнтованої SOA в BPM включає: розробку інструментів для користувачів для особистого визначення моделей з основними компонентами; інструменти управління ефективністю бізнесу для управління всіма функціями як різними процесами, а також для моніторингу IT-систем і бізнес-процесів операціями IT-систем та бізнес-процесів.

Хмарні управління бізнес-процесами(BPM) означає використання інструментів BPM, які надаються як програмна послуга [12]. За даними Gartner, до 2016 року понад 20% усіх бізнес-процесів у всьому світі будуть підтримуватися хмарними BPM-платформами. Хмарні технології BPM надає користувачам хмарних технологій цінні можливості використовувати хмарне програмне забезпечення на основі оплати за кожного користувача. Застосування моделі хмарного сервісу в організації (рис.2.1) можна підсумувати наступним чином:

Рис.2.1. Застосування моделі хмарного сервісу в організації

Інфраструктура як послуга (IaaS) - це послуга хмарного клієнта, яка надає IT-інфраструктуру з процесором, оперативною пам'яттю, сховищем, пропускною спроможністю та іншими, інші конфігурації. Ці компоненти використовуються для побудови віртуальних комп'ютерів. На віртуальний

комп'ютер може бути встановлена операційна система та додатки за потребою. Перевагою послуги IaaS є те, що не потрібно купувати комп'ютери, що дозволяє створити економічно ефективне рішення. Конфігурацію віртуального комп'ютера також можна змінювати за потреби.

Платформа як послуга (PaaS) - це послуга, яка надає обчислювальну платформу. Зазвичай це операційні системи, бази даних, веб-сервери та фреймворк для запуску додатків. PaaS дозволяє розробникам зосередитися на додатках, які вони створюють, не думаючи про обслуговування самої обчислювальної платформи.

Програмне забезпечення як послуга (SaaS) - це послуга хмарних обчислень сервісів, де надаються додатки. Постачальник послуг керує інфраструктурою та платформою на якій працює програма. Прикладами сервісів електронної пошти є: Gmail, Yahoo та Outlook, в той час як прикладами додатків соціальних мереж є Twitter, Facebook. Ця послуга дозволяє користувачам використовувати додаток без необхідності купувати ліцензію. Користувачам потрібен лише пристрій, підключений до інтернету.

2.2. Архітектура хмарних сховищ

Хмарне сховище складається з тисяч пристроїв зберігання даних, згрупованих мережею, розподіленою файловою системою та іншим проміжним програмним забезпеченням для зберігання, щоб надати користувачам хмарне сховище [13].

Сховище може приймати форму автономних масивів, конвергентної інфраструктури, гіперконвергентної інфраструктури, програмно визначеного сховища або публічного хмарного сховища в цілому. Зберігання також може бути у формі блоку, файлу або об'єкта. Багато з цих систем зберігання даних використовують таку мережеву інфраструктуру, як оптоволокно, iSCSI, NFS і SMB. Ці мережеві інфраструктури об’єднують системи зберігання даних, такі як масиви на основі NVMe, гібридні масиви, HCI, загальнодоступна хмара для

основного та резервного копіювання та контейнерне сховище. Пули ресурсів зберігання, розподілені файлові системи, угоди про рівень обслуговування та інтерфейси послуг є типовими структурами хмарного сховища. Бачимо що п'ятирівнева модель хмарного сховища (рис.2.2), складається з рівня мережі та інфраструктури зберігання, рівня керування сховищем, рівня керування метаданими, рівня накладання сховища та рівня інтерфейсу служби, тоді як на наступному малюнку зображено архітектуру хмарного сховища (рис.2.3).

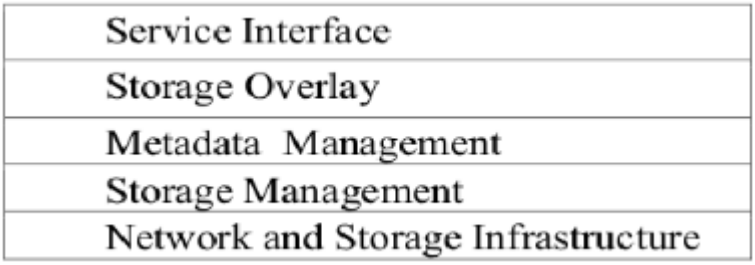

Рис.2.2. П'ятирівнева модель хмарного сховища

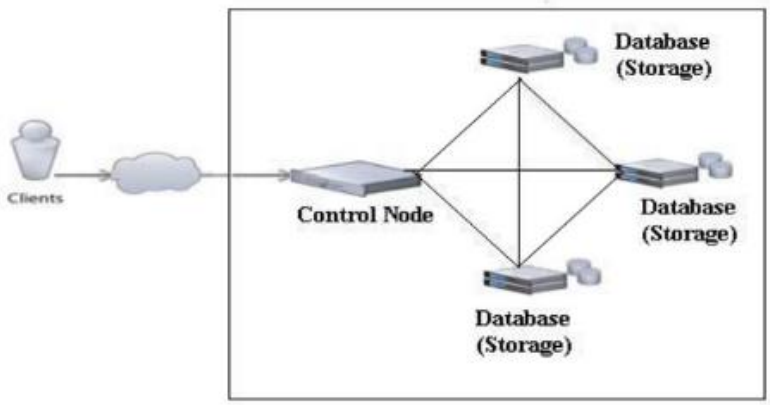

Рис.2.3. Архітектура хмарного сховища

2.3. Аналіз технологій та мов програмування у створенні хмарного сховища

У сучасну цифрову епоху хмарне сховище стало невід'ємною частиною нашого особистого та професійного життя. Зі збільшенням обсягу даних, які генеруються, потреба в ефективному сховищі та доступності ніколи не була

такою важливою. Одним із підходів до досягнення цього є створення веб-орієнтованої інформаційної системи, яка пропонує численні переваги як для користувачів, так і для підприємств. Ми розглянемо переваги побудови хмарної системи зберігання з веб-орієнтованим підходом.

Однією з найважливіших переваг веб-орієнтованої інформаційної системи хмарного сховища є можливість доступу до даних з будь-якого місця та в будь-який час за допомогою будь-якого пристрою з підключенням до Інтернету. Це забезпечує безперебійну співпрацю та продуктивність, дозволяючи користувачам отримувати свої файли та документи з кількох пристроїв, усуваючи потребу у фізичних пристроях зберігання. Це забезпечує віддалену роботу, підтримує мобільність і покращує загальну ефективність роботи. Також веб-орієнтовані хмарні системи пропонують масштабовані та гнучкі варіанти зберігання, що дозволяє користувачам збільшувати або зменшувати ємність зберігання відповідно до своїх потреб. Це усуває необхідність інвестувати в додаткове обладнання або сервери, оскільки постачальники хмарних сховищ можуть легко пристосуватися до мінливих вимог до сховища. Завдяки веб-орієнтованій моделі підприємства можуть швидко адаптуватися до зростаючих вимог, забезпечуючи можливість масштабування своєї системи відповідно до потреб [14].

Вибираючи веб-хмарне сховище, бізнес може значно скоротити свої капітальні витрати. Традиційні локальні системи зберігання вимагають початкових інвестицій в апаратне забезпечення, центри обробки даних і обслуговування. В цьому випадку навпаки, постачальники хмарних сховищ покривають ці витрати, дозволяючи користувачам платити лише за ресурси, які вони використовують. Крім того, ця розрахункова модель також може зменшити експлуатаційні витрати, оскільки технічне обслуговування та оновлення здійснюється постачальником послуг.

Веб-хмарна система зберігання даних характеризується своєю архітектурою, яка складається з кількох фундаментальних частин: серверної та клієнтської.

Основною точкою контакту для користувачів є веб-інтерфейс. Він робить можливим керування даними та доступ до них через веб-браузер, забезпечуючи нейтральний для платформи та зручний досвід. Здатність веб-інтерфейсу забезпечувати універсальну доступність є однією з його відмінних рис. Якщо користувачів є веб-браузер, а також підключення до Інтернету, потенційні користувачі можуть безпечно отримувати доступ до своїх даних практично з будь-якого місця [15].

Веб-сервери мають головну роль у веб-системах хмарних сховищ, яка є життєво важливою ланкою між користувачами та основною інфраструктурою зберігання. Ці сервери відповідають за керування та маршрутизацію запитів, розширення доступу до ресурсів і забезпечення безперебійного зв’язку. Також треба виділити те, що веб-сервери відіграють значну роль у процесах автентифікації та авторизації користувачів. Вони перевіряють облікові дані та дозволи користувача, перш ніж дозволити доступ до даних. Це має гарантувати, що особисті дані користувачів завжди будуть в безпеці та що лише авторизовані користувачі можуть виконувати певні дії з ними [16].

Узагальнюючи саме веб підхід у питанні створення хмарного сховище може забезпечити швидкість і гнучкість, дозволяючи підприємствам швидко розгортати нові служби та технології.

Не можна переоцінити важливість правильного вибору технологій та мов програмування під час створення хмарного сховища. Цей вибір на пряму впливає на продуктивність, масштабованість, безпеку та загальну ефективність системи. Також не менш важливим є зручність обраних технологій, які допомагають розробнику створювати більш ефективнішу систему для короткий проміжок часу.

Роздивимось технології для серверної частини хмарного сховища. Перш за все для забезпечення ефективності системи може бути використаний фреймворк NestJS. У загальному це фреймворк для побудови ефективних, масштабованих сервер-сайд додатків. У ньому використовується прогресивний JavaScript, він розроблений на TypeScript і повністю його підтримує і комбінує в

собі елементи ООП (Об'єктно орієнтоване програмування), ФП (Функціональне програмування) та ФРП (Функціонально-реактивне програмування). Головною зручністю є те, що у під капотом використовує надійні HTTP фреймворки такі як Express (за замовчуванням) і за бажанням може бути налаштований на використання Fastify. Nest надає рівень абстракції над цими Node.js фреймворками (Express/Fastify), але також надає API для розробника. Це дозволяє розробникам вільно використовувати велику кількість сторонніх модулів, які доступні для базової платформи. Головною проблемою попередників NestJS, наприклад таких як Angular, Vue, React є те, що не один із них не вирішує ефективну проблему архітектури проекту. В той час NestJS вже з старту надає архітектуру програми яка дозволяє розробникам та командам створювати високо тестовані, розширювані, слабозв'язані та легко підтримувані додатки [17].

NestJS забезпечує модульну архітектуру, це сприяє розділенню умовного додатка на модулі, що дозволяє розробнику легко керувати та організувати роботу над кодом. Кожен модуль має свій контролер, провайдер та сервіс, це спрощує структуру та зрозумілість проекту. Крім того NestJS побудований на TypeScript, це забезпечує написання коду з використанням строгої типізації, підвищуючи надійність проекту, а також допомагає у відлагодженні.

Ще однією унікальністю NestJS є те, що в нього вбудовано Swagger, що дозволяє автоматично генерувати документацію API. Це полегшує співпрацю з іншими розробниками та забезпечить чіткість структури API. Крім того, він також допомагає розробнику тестувати своє API, наприклад на коректність роботи REST-запитів.

Одною з головних функцій в процесі створення хмарного сховища є забезпечення обробки файлових завантажень. Існує чимало npm-пакетів, призначених для аналізу та обробки multipart/form-data запитів на Node.js-сервери. Кожен із них спроектований по-особливому. Деякі призначені для використання з Express.js, інші розраховані на автономне використання. Деякі зберігають проміжні файли на жорсткому диску або пам'яті, а деякі ні.

Busboy - це парсер потокових даних, робота якого ґрунтується на подіях. Цей пакет не пов'язаний з Express.js. Busboy не зберігає проміжні файли. Натомість вхідні дані обробляються в потоковому режимі. Ядро реалізації обробки multipart/form-data-запитів виділено окремий модуль dicer.

Multer – це програмне забезпечення проміжного шару для Express.js. Він зберігає проміжні файли в пам'яті або на диску та заповнює об'єкт res.files для отримання файлів. Користувачі цього пакета можуть керувати полями, призначеними для завантаження. Є можливість обмеження кількості файлів, що вивантажуються на сервер. Усередині Multer використовується Busboy.

Multiparty за функціоналом це те ж саме, що і Formidable, але, крім того, цей пакет підтримує потокову обробку файлів. Але у документації до Multiparty рекомендується використовувати для потокової обробки даних Busboy як швидшу альтернативу цього пакета.

Formidable є однією з найстаріших бібліотек. Її перша версія вийшла у 2011 році. Це автономна бібліотека, для роботи якої не потрібен фреймворк Express.js. Вона використовує тимчасову директорію на жорсткому диску для зберігання файлів.

Таблиця 2.1

Загальна інформація по пакетам

Назва пакету	Busboy	Formidable	Multer	Multiparty
1	2	3	4	5
Щотижневі завантаження з npm	1.2 мільйона	2.1 мільйона	0.6 мільйона	0.3 мільйона
Рік створення	2013	2010	2015	2013
ПЗ проміжного шару для Express.js	Ні	Ні	Так	Ні
Підтримка потоків	Так	Ні	Ні	Так

Продовження таблиці 2.1

1	2	3	4	5
Збереження проміжних даних на диску	Ні	Так	Так	Так
Збереження проміжних даних у пам'яті	Ні	Ні	Так	Ні

Аналізуючи загальну інформацію по пакетам з табл. 2.1, робимо висновок що саме для розробки веб-орієнтованого хмарного сховища, де буде використано фреймворк Express.js доцільно буде використовувати саме програмне забезпечення Multer.

Для взаємодії с СУБД (системами управління базами даних), такими як PostgreSQL, MySQL, SQLite, та іншими в даному контексті доцільніше використовувати TypeORM, яка є безперечно найдосконалішим Object Relational Mapper (ORM), доступним у світі node.js. Вона дозволяє розробникам використовувати об'єктно-реляційне відображення, щоб працювати з базами даних, використовуючи об'єктно-орієнтований підхід. Оскільки він написаний на TypeScript, він досить добре працює з фреймворком NestJS. TypeORM має можливість створювати та застосовувати міграції, що дозволяє нам оновлювати структури бази даних та залишати її сумісними зі змінами в моделях.

Як систему управління базами даних в цій системі можна використовувати PostgreSQL. PostgreSQL - це вдосконалена реляційна база даних корпоративного класу з відкритим вихідним кодом, яка підтримує як SQL (реляційні), так і JSON (нереляційні) запити. Це дуже стабільна система керування базами даних, яка підтримується більш ніж 20-річним розвитком спільноти, що сприяло її високому рівню стійкості, цілісності та коректності. PostgreSQL використовується як основне сховище даних або сховище даних для багатьох веб-, мобільних, геопросторових і аналітичних програм. PostgreSQL має надійні набори функцій, включаючи Multi-Version Concurrency Control (MVCC),

відновлення на певний момент часу, деталізований контроль доступу, табличні простори, асинхронну реплікацію, вкладені транзакції, онлайнове/гаряче резервне копіювання, удосконалений планувальник/оптимізатор запитів і ведення журналу наперед. Він підтримує міжнародні набори символів, багатобайтові кодування символів, Unicode, а також ураховує локалізацію для сортування, чутливості до регістру та форматування. PostgreSQL має високу масштабованість як щодо кількості даних, якими він може керувати, так і щодо кількості одночасних користувачів, яких він може прийняти [18].

Роздивимось технології для клієнтської частини хмарного сховища. Для створення швидкого та ефективного веб-додатку важливо приділити увагу питанню оптимізації, а також зручності розробки. Виходячи з вимог, для реалізації подібного функціоналу доцільно використовувати фреймворк NextJS, як створювався "поверх" фреймворку React. Під капотом Next.js також абстрагує та автоматично налаштовує інструменти, необхідні для React, як групування, компіляція тощо. Це дозволяє вам зосередитися на створенні програми замість того, щоб витрачати час на налаштування. Next.js забезпечує структуру та кращі можливості рендерингу для React. Як написано вище, він працює поверх React, оскільки називає себе "React Framework for Production". Таким чином, він працює як двигун для можливостей React, використовуючи багато інструментів і ресурсів, які вже використовуються React, такі як Redux або Hooks. Next.js керує даними для оптимальної швидкості, маючи два типи попереднього рендерингу. Рендеринг на стороні сервера (SSR) дозволяє отримувати дані та рендерувати їх під час запиту. Інший тип - Static Generation (рис.2.4), який використовує дані, вже доступні під час складання до виконання запиту, що дуже корисно в ситуаціях, коли дані можуть бути публічно кешовані (не для конкретного користувача) або заздалегідь відрендеровані для SEO. Крім того, в NextJS вбудована підтримка TypeScript дозволяє розробникам писати типізований код, що полегшує відлагодження та підтримку додатків.

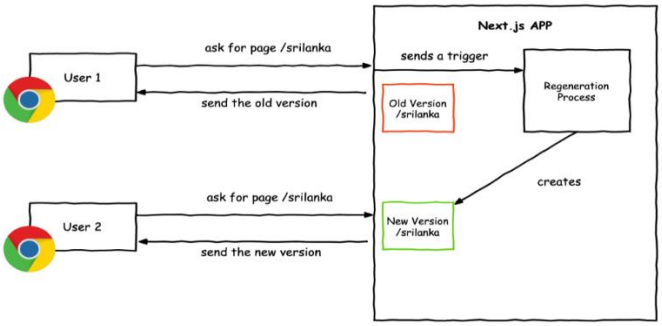

Рис.2.4. Інкрементальна статична генерація

Для здійснення запитів з клієнтської частини на серверну можна використати бібліотеку Axios, яка також підтримує TypeScript. Перевага Axios в тому, що він має чистий и легко зрозумілий синтаксис, що робить процес розробки більш зручним. Крім того Axios можна використовувати як на стороні клієнта, так і на стороні сервера, що робить його універсальним інструментом для рендерингу на сторінці [19].

Для збереження кукі(cookie), тобто наприклад токену, доцільно використовувати бібліотеку Nookies, яка зроблена на основі Node.js, зокрема в контексті серверного рендерингу та роботи зі стороною серверу. Також nookies дозволяє зберігати та передавати інформацію клієнту для роботи зі станом аутентифікації, сесіями та іншими даними.

2.4. Безпека в системі хмарного сховища

Безпека в системі хмарного сховища є одним з головних питань під час розробки. Хмарні сховища, такі як Amazon, Google Cloud Storage, або Microsoft Azure Storage, надають зручний та ефективний спосіб зберігання, проте важливість конфіденційності та безпеки особистих даних користувача є справжнім викликом та важливим питанням для розробника. Насамперед у питаннях безпеки треба забезпечити захист своїй системі від несанкціонованого вторгнення. Розробка системи авторизації та реєстрації є важливим етапом забезпечення безпеки. Для налаштування простої та гнучкої аутентифікації

можна використовувати PassportJS спільно з JWT (JSON Web Tokens). Ця зв'язка добре підходить для стартапів або невеликих проектів та в той же час є простим и безпечним рішенням.

Технологія PassportJS спільно з JWT (JSON Web Tokens) використовується для автентифікації та авторизації користувачів у веб-додатках. PassportJS спрощує важливі задачі аутентифікації, такі як перевірка ідентичності користувача та перенаправлення на сторінки входу та виходу, в той час як JWT часто використовується для передачі інформації про аутентифікацію та дозволи між клієнтом та сервером. Після успішної аутентифікації PassportJS генерує JWT, який містить інформацію про користувача та підписаний ключем сервера. Згенерований токен передається клієнту для зберігання та передачі під час подальших запитів. Під час кожного запиту клієнта PassportJS перевіряє наявність та валідність токену, розшифровує його та використовує інформацію для авторизації. Ця комбінація дозволяє забезпечити безпеку та зручність при аутентифікації та авторизації в веб-додатках [20].

Таким чином, PassportJS використовує обрану стратегію аутентифікації для перевірки облікових даних користувача, а JWT токени використовуються для передачі інформації про користувача між клієнтом та сервером. PassportJS забезпечує захист маршрутів та надає зручний спосіб керування автентифікацією у вашому додатку.

2.5. Аналіз ефективності хмарних технологій у бізнес-середовищі

Хмарні обчислення розглядаються як метод взаємодії з клієнтами, і деякі організації, які використовують їх, викликали великий інтерес у клієнтів, особливо після того, як цю технологію використовували для обслуговування клієнтів. Це питання доповнює процес взаємодії між клієнтом і самою організацією. Однак є деякі організації, які все ще не можуть використовувати хмарні обчислення для надання швидших, більш високотехнологічних послуг і ведення бізнесу [21].

Крім того, цим підприємствам важко встигати за всіма змінами та трансформаціями, які є ескалацією сил змін у багатьох частинах світу, а також проблемами, з якими стикаються організації в досягненні прогресу в результаті цих подій, що вимагає встановлення координації між організаційною безперервністю та врахуванням розвитку, вимог до змін і стресу, а також внутрішніх і зовнішніх викликів, які впливають на його здатність виживати, рости і продовжувати свою діяльність.

Підприємство завжди прагне досягти своїх цілей, незалежно від того, чи є ці цілі прибутком або певними темпами зростання, що спонукає організації час від часу вимірювати свою ефективність та результативність у досягненні цих цілей, і це робиться для того, щоб підвищити організаційну ефективність діяльності [22]. Складність пошуку фіксованих і точних показників для визначення стандарту або моделі, на які можна було б орієнтуватися при оцінці ефективності та результативності організації, враховуючи, що кожна установа має свою власну реальність, а також складність вимірювання ступеня застосування професійної етики всіма сторонами організації [23].

Буде використано описово-аналітичний метод з метою визначення рівня використання хмарних обчислень та його впливу на досягнення організаційної ефективності. Вибірка дослідження складалася з усіх адміністраторів та працівників консалтингової компанії "Консалт-Практик". Вибірка дослідження складалася з 100 працівників, серед яких було розповсюджено анкети. Було отримано 83 анкету, тоді як 12 анкет, які не підходили для статистичного аналізу, були виключені, таким чином, кількість анкет, придатних для статистичного аналізу, становила 83. Для оцінки внутрішньої узгодженості питань у нашій анкеті ми використали інструмент Кронбаха-альфа. Цей інструмент дозволяє нам переконатися, наскільки питання, включені в анкету, дійсно вимірюють одне й те саме поняття. Високий коефіцієнт Кронбаха-альфа свідчить про хорошу внутрішню узгодженість питань, що підвищує надійність та точність наших даних. Такий аналіз допомагає нам створити інструмент, який дає надійні результати і може бути основою для обґрунтованих висновків та прийняття

рішень у рамках нашого дослідження. Формула для розрахунку Кронбаха-Альфа являє собою відношення суми дисперсії всіх елементів шкалі до суми загальної дисперсії та дисперсії, виявленої випадковими помилками. Формула виглядає таким чином(2.1)

$$\alpha = \frac{k}{k-1}\left(1 - \frac{\sum_{i=1}^{k} \sigma_i^2}{\sigma_T^2}\right) \quad (2.1)$$

- k - кількість елементів (питань) у шкалі,
- σ_i^2 - дисперсія відповідей кожного елемента,
- σ_T^2 - загальна дисперсія відповідей.

Таблиця 2.2

Коефіцієнт Кронбаха-альфа

Сфера	Ca
Хмарні обчислення	0.78
Організаційна ефективність	0.82

Рівняння стабільності інструменту Кронбаха-альфа було застосовано до всіх областей дослідження та інструменту в цілому, і це показано в табл. 2.2.

З таблиці 1 видно, що коефіцієнт стабільності для сфери хмарних обчислень склав 0,80, а організаційна ефективність - 0,84, що є високими та прийнятними значеннями для цілей дослідження.

Було отримано коефіцієнт кореляції застосування хмарних обчислень на організаційну ефективність, а також застосовано множинну регресію для виявлення впливу застосування хмарних обчислень на організаційну ефективність, і результати представлені в табл. 2.3.

Таблиця 2.3

Коефіцієнт кореляції впливу застосування хмарних обчислень на ефективність організації

Організаційна ефективність		Хмарні обчислення	
		Кореляція	Значимість
Досягнення цілей		0.75	0.00
Продуктивність		0.72	0.00
Адаптування до робочого середовища		0.78	0.00

Табл. 2.3 показує, що коефіцієнти кореляції між застосуванням хмарних обчислень та організаційною ефективністю коливаються в межах 0,71-0,75, що є позитивним зв'язком, який свідчить про позитивну кореляцію між застосуванням хмарних обчислень та організаційною ефективністю.

У підсумку проведення аналізу впливу використання хмарних сховищ на оптимізацію різноманітних бізнес-процесів було виявлено, що використання хмарних обчислень чинить статистично значущий вплив на досягнення цілей, підвищення продуктивності та адаптацію до бізнес-середовища. Базуючись на цих висновках бачимо що потенціалі хмарних обчислень у поліпшенні комунікаційних процесів усередині та поза організацією, сприяють координації та співпраці на різних рівнях адміністративної структури [24].

РОЗДІЛ 3
ПРОЕКТУВАННЯ ТА РОЗРОБКА ДЕМОНСТРАЦІЙНОГО ХМАРНОГО СХОВИЩА

3.1. Опис структури системи та алгоритмів її функціонування

Демонстраційний веб-додаток хмарного сховища являє собою дві частини: серверну, а також клієнтську.

Загальна архітектура серверної частини дотримується принципів модульності та поділу відповідальності. Кожен модуль має власну область відповідальності та взаємодіє з іншими модулями у вигляді сервісів. Це забезпечує високий ступінь ізоляції та повторного використання коду. Архітектура бази даних в цьому проекті являє собою дві таблиці: user та files, які пов'язані між собою зв'язком "багато-то-одного"(рис.3.1). Після впровадження проект nest.js отримуємо файли (рис.3.2), які є фундаментальними, на яких будується вся серверна частина [25].

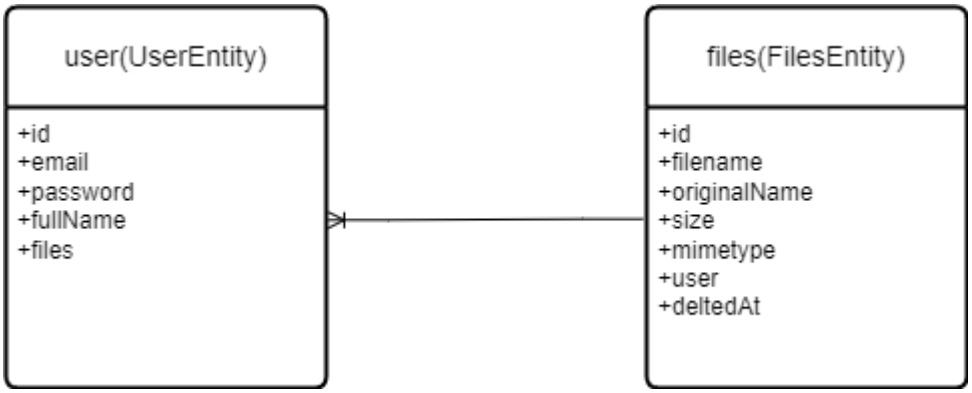

Рис.3.1. Зв'язок files з user

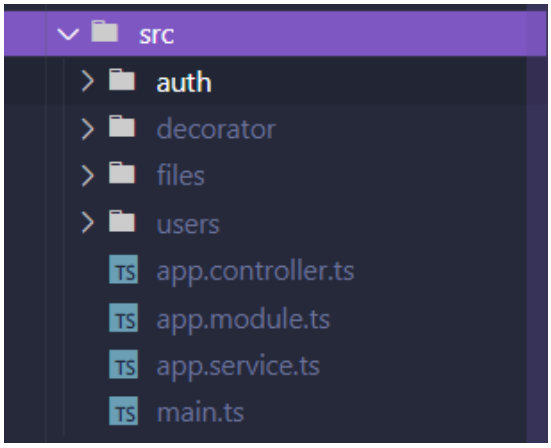

Рис.3.2. – Структура файлів серверної частини

Основним файлом програми є main.ts, який запускає сервер порту 7777. Він налаштовує Swagger для документації API, а також для тестування коректної роботи функцій бази даних.

В app.controller створено для обробки запитів, таких як: GET, POST, DELETE тощо. З цих запитів вивантажуватимуться дані та передаватимуться в app.service. В свою чергу в app.service буде зберігатися бізнес-логіка, завдяки якій додаток може взаємодіяти з користувачем. Для зв'язку між контролером та сервісом було створено app.module, він є головним модулем серверної частини, який імпортує та поєднує всі інші модулі нашої програми.

Папка src/users (рис.3.3) містить файли, які належать до модуля користувачів у програмі. Цей модуль обробляє всі операції, пов'язані з користувачами, такі як створення, отримання інформації користувача і т.д.

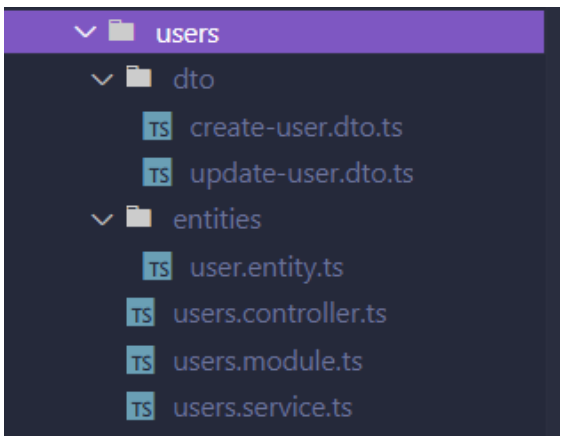

Рис.3.3. Структура папки src/users

Файл users.controller містить клас UsersController, який обробляє запити HTTP, пов'язані з користувачами. Контролери в NestJS відповідають за обробку вхідних запитів та повернення відповідей клієнтам. У даному випадку UsersController містить метод getMe, який повертає інформацію про поточного користувача. Цей метод використовує сервіс UsersService для отримання даних користувача.

UsersService - це сервіс, що надає методи роботи з користувачами. Цей сервіс використовує репозиторій User, який є інтерфейсом для взаємодії з базою даних користувачів. Ось опис методів у цьому сервісі:

- findByEmail: Цей метод приймає email як параметр і повертає користувача, у якого відповідний email. Він використовує метод findOneBy репозиторію для пошуку користувача.
- findById: Цей метод приймає id як параметр і повертає користувача з відповідним id. Він також використовує метод findOneBy репозиторію для пошуку користувача.
- Create: Цей метод приймає об'єкт CreateUserDto як параметр і створює нового користувача. Він використовує метод save репозиторію для збереження нового користувача в базі даних.

Усі ці методи повертають Promise, оскільки вони асинхронні та взаємодіють з базою даних.

Файл users.module.ts містить клас UsersModule, який є модулем NestJS. Модулі в NestJS – це спосіб організації коду, вони інкапсулюють провайдери, контролери, імпорти та експорти в одну функціональну групу. UsersModule імпортує TypeOrmModule.forFeature([UserEntity]), що дозволяє використовувати репозиторій UserEntity усередині провайдерів цього модуля.

Файл create-user.dto містить клас CreateUserDto, який використовується для передачі даних під час створення нового користувача. DTO (Data Transfer Object) – це об'єкт, який використовується для інкапсуляції даних та їх передачі між клієнтом та сервером.

Усередині entity ми описуємо структуру нашої таблиці бази даних, у цьому випадку користувачів, і вже з допомогою user.entity ми можемо змінювати дані всередині нашої таблиці. Файл user.entity містить клас UserEntity, який представляє сутність користувача базі даних.

Також на серверній частині був створений окремий CRUD для файлів(рис.3.4.), для можливості створення окремих ресурсів, так само, як це було зроблено для користувачів.

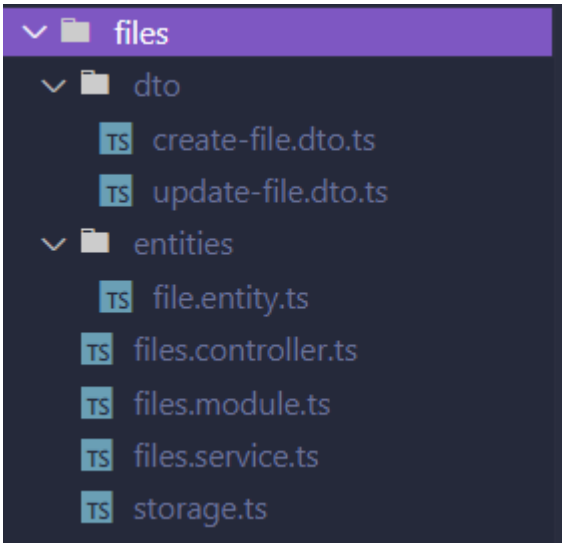

Рис.3.4. Структура папки src/files

Файл files.module містить модуль FilesModule, який являє собою модуль NestJS. У цьому модулі оголошено контролери та провайдери, які використовуються у додатку. Зокрема, він використовує FilesController і FilesService і імпортує FileEntity через TypeOrmModule.

Files.contoller містить контролер FilesController, який обробляє запити HTTP, пов'язані з файлами. У ньому визначено такі методи:

– downloadFile: Цей метод обробляє GET-запити на маршрут :id/download, де :id - це ідентифікатор файлу. Він використовує сервіс filesService для отримання файлу за ідентифікатором і, якщо файл знайдено, відправляє його у відповіді. Якщо файл не знайдено, надсилається відповідь зі статусом 404 і повідомленням 'File not found'.

- findAll: Цей метод обробляє GET-запити на маршрут /files. Він приймає userId і fileType як параметри запиту і повертає всі файли, що належать користувачеві із зазначеним userId і типом файлу fileType. Цей метод захищений за допомогою JwtAuthGuard, тому доступ до нього можливий тільки після аутентифікації.
- create: Цей метод обробляє POST-запити на маршрут /files. Він приймає файл і userId як параметри і створює новий файл за допомогою сервісу filesService. Файл має бути меншим за 5 МБ, інакше буде викинуто виняток.
- remove: Цей метод обробляє DELETE-запити на маршрут /files. Він приймає userId і ids як параметри запиту і видаляє файли із зазначеними ids, що належать користувачеві із зазначеним userId.

Файл files.service являє собою сервіс FilesService у фреймворку NestJS, який надає методи для роботи з файлами в базі даних. Якщо розбирати докладний опис кожного методу, він буде виглядати так:

- findAll: Цей метод приймає userId і fileType як параметри і повертає всі файли, що належать користувачеві із зазначеним userId і типом файлу fileType. Він використовує createQueryBuilder для створення запиту до бази даних.
- create: Цей метод приймає файл і userId як параметри і створює новий файл у базі даних. Він зберігає файл за допомогою методу save сховища.
- getFileById: Цей метод приймає id як параметр і повертає файл із зазначеним id з бази даних. Він використовує метод findOne сховища для пошуку файлу.
- remove: Цей метод приймає userId і ids в якості параметрів і видаляє файли із зазначеними ids, що належать користувачеві із зазначеним userId. Він використовує createQueryBuilder для створення запиту до бази даних і метод softDelete для м'якого видалення файлів.

М'яке видалення (Soft Delete) - це підхід до видалення даних, за якого записи не видаляються фізично з бази даних. Замість цього, вони позначаються

як видалені, зазвичай за допомогою спеціального поля, такого як deletedAt у нашому випадку.

Коли запис позначено як видалений, він не відображається під час звичайних запитів до бази даних, але все ще зберігається і може бути відновлений за необхідності. Це корисно, коли ви хочете зберегти історію даних або запобігти втраті даних через випадкове видалення.

Сервіс files.service використовує репозиторій TypeORM для роботи з базою даних і впроваджує його за допомогою декоратора @InjectRepository.

Storage визначає конфігурацію сховища для файлів, що завантажуються, з використанням бібліотеки multer, яка широко використовується в Node.js для обробки multipart/form-data, зазвичай використовуваних для завантаження файлів.

У цьому файлі визначено такі елементи:

- genID: Це функція, яка генерує випадковий ідентифікатор із 20 символів. Вона створює масив із 20 елементів, заповнює його випадковими числами від 0 до 18, перетворює ці числа в рядки з основою 18 і об'єднує їх в один рядок.
- genFileName: Це функція, яка генерує ім'я файлу для завантажуваного файлу. Вона бере оригінальне ім'я файлу, витягує його розширення, генерує новий ідентифікатор за допомогою функції genID і об'єднує ідентифікатор і розширення в нове ім'я файлу.
- fileStorage: Це конфігурація сховища multer, яка використовує diskStorage для збереження файлів на диску. Вона визначає, що файли мають бути збережені в директорії ./uploads і що ім'я кожного файлу має бути згенероване функцією genFileName.

Цей файл забезпечує функціональність для збереження завантажених файлів на диску з унікальними іменами файлів.

Files.entity визначає сутність File у TypeORM, фреймворку для роботи з базами даних у Node.js. У FileType перераховується два можливих типи файлів, а саме PHOTO(фото) і TRASH(сміття).

Клас File, який представляє сутність File у базі даних. Він має такі поля:

- id: Це первинний ключ, який автоматично генерується для кожного нового файлу. Декоратор @PrimaryGeneratedColumn() вказує, що це поле автоматично генерується і унікальне для кожного файлу.
- filename: Це ім'я файлу, яке може бути null. Декоратор @Column({ nullable: true }) вказує, що це поле може бути нульовим у базі даних.
- originalname: Це оригінальне ім'я файлу. Декоратор @Column() вказує, що це поле є стовпцем у таблиці бази даних.
- size: Це розмір файлу в байтах, який може бути null. Декоратор @Column вказує, що це поле може бути нульовим у базі даних.
- mimetype: Це MIME-тип файлу. MIME-тип - це стандарт для ідентифікації типів даних, який зазвичай використовується в інтернеті.
- user: Це зв'язок між файлом і користувачем. Декоратор @ManyToOne вказує, що це поле є відношенням "багато-до-одного" між файлами та користувачами. Кожен файл належить одному користувачеві.
- deletedAt: Це дата і час, коли файл було видалено. Якщо файл не було видалено, це поле буде null. Декоратор @DeleteDateColumn() вказує, що це поле використовується для відстеження часу видалення файлу в базі даних.

Клас File декорований за допомогою @Entity('files'), що вказує на те, що цей клас має бути пов'язаний із таблицею files у базі даних. Це дає змогу TypeORM автоматично зіставляти об'єкти цього класу із записами в таблиці files.

Папка auth (рис.3.5) у загальному значенні відповідає за авторизацію та аунтефікацію в додатку.

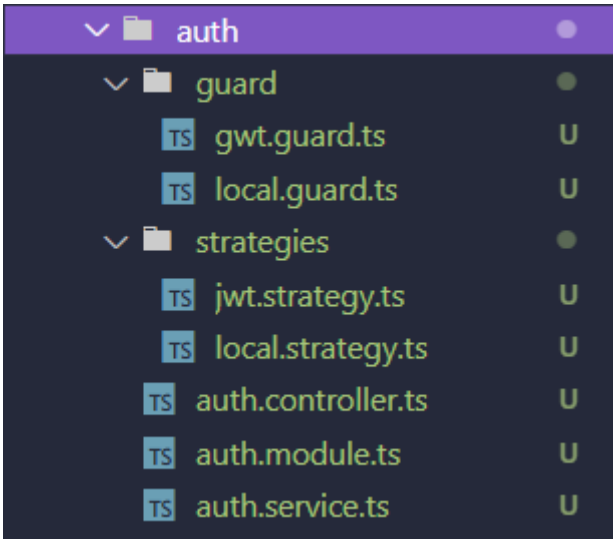

Рис.3.5. Структура папки src/auth

Файл auth.controller, так само як і попередні контролери, містить контролер AuthController, який обробляє запити HTTP, пов'язані з автентифікацією. У ньому є два методи: login та register. Метод login використовує охоронця LocalJwtAuthGuard для перевірки вхідного запиту. Якщо перевірка відбувається успішно, метод login викликає метод login із сервісу AuthService, передаючи йому об'єкт користувача. Метод register просто викликає метод register із сервісу AuthService, передаючи йому DTO (Data Transfer Object) для створення нового користувача.

Сервіс AuthService, що містить бізнес-логіку, пов'язану з автентифікацією. У ньому є три методи: validateUser, register і login. Метод validateUser перевіряє, чи існує користувач із зазначеною адресою електронної пошти та паролем. Метод register створює нового користувача та повертає JWT. Метод login просто повертає JWT для вказаного користувача.

Модуль AuthModule, який поєднує всі класи, пов'язані з автентифікацією, в одну одиницю. Модуль імпортує JwtModule, UsersModule та PassportModule, а також реєструє AuthController, AuthService, LocalStrategy та JwtStrategy як провайдери.

У папці src/auth є два файли: jwt.guard.ts та local.guard.ts. Ці файли містять охоронців (guards), які використовуються для автентифікації та авторизації в додатку.

JwtAuthGuard розширює AuthGuard із стратегією 'jwt'. Це означає, що він використовуватиме стратегію JWT (JSON Web Token) для автентифікації користувачів. JWT - це спосіб безпечної передачі між двома сторонами як JSON-объекта. Ця інформація може бути перевірена та довірена, оскільки вона підписана. В даному випадку JwtAuthGuard буде використовуватися для перевірки JWT токенів при аутентифікації.

LocalAuthGuard також розширює AuthGuard, але з стратегією 'local'. Це означає, що він буде використовувати стратегію локальної аутентифікації, яка зазвичай включає перевірку імені користувача і пароля.

Обидва охоронці є @Injectable, що означає, що їх можна впровадити до інших класів у додатку.

Папка strategies містить стратегії автентифікації, які використовуються у додатку. Стратегії автентифікації - це способи, якими програма може перевірити облікові дані користувача та встановити його ідентичність. У нашому додатку використовуються дві стратегії: LocalStrategy і JwtStrategy.

JwtStrategy, який успадковує PassportStrategy. Ця стратегія використовується для аутентифікації за допомогою JSON Web Tokens (JWT). У конструкторі вказуються параметри стратегії, включаючи метод вилучення JWT із запиту та секретний ключ для верифікації токена. Метод validate викликається автоматично за умови успішної верифікації токена. Він перевіряє, чи існує користувач з ID, зазначеним у корисному навантаженні (payload) токена, і повертає об'єкт користувача або викидає виняток.

LocalStrategy, який також успадковує PassportStrategy. Ця стратегія використовується для аутентифікації за допомогою локальної стратегії (зазвичай це ім'я користувача та пароль). У конструкторі вказується поле, яке використовуватиметься як ім'я користувача. Метод validate викликається автоматично за успішної аутентифікації. Він перевіряє, чи існує користувач із

зазначеним ім'ям користувача та паролем, і повертає об'єкт користувача або викидає виняток.

Файл user-id.decorator.ts містить декоратор UserId, який використовується для отримання ідентифікатора користувача з запиту. Декоратори в NestJS - це функції, які дозволяють додавати додаткові функції до класів, методів, властивостей або параметрів. У цьому випадку декоратор UserId використовується для вилучення ідентифікатора користувача запиту.

Клієнтська частина (рис.3.6) побудована на основі фреймворку next.js, який надає функції серверного та статичного рендерингу.

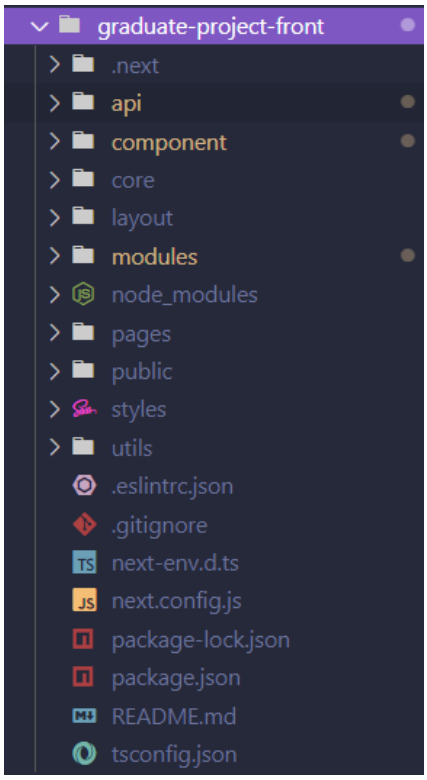

Рис.3.6. Структура файлів клієнтської частини

Папка api (рис.3.7) у проекті використовується для організації коду, який взаємодіє із зовнішніми API або сервером. Це включає функції для надсилання HTTP-запитів, визначення форми даних (DTO), обробки помилок тощо.

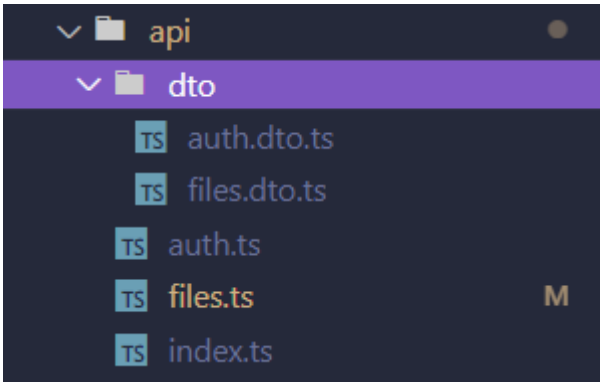

Рис.3.7. Структура папки api

Файли з розширенням .dto.ts (Data Transfer Objects), визначають структуру даних, що використовуються під час взаємодії з API. Це допомагає забезпечити типобезпеку та передбачуваність у коді.

Файли auth.ts і files.ts, які містять функції взаємодії з конкретними частинами API (в цьому випадку автентифікація і файли). Ці функції використовують бібліотеку axios для надсилання запитів HTTP.

Файл index.ts, який експортує всі функції з auth.ts і files.ts, щоб вони могли легко імпортуватися в інші частини програми.

Папка component (рис.3.8) містить різні компоненти React, які використовуються у проекті. Кожен компонент має власний каталог, який містить файли з кодом компонента та стилями.

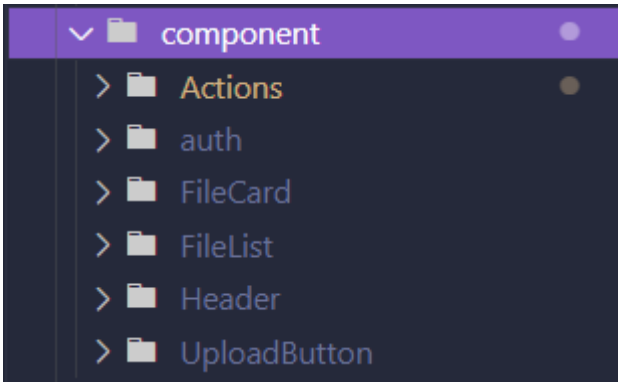

Рис.3.8. Структура папки component

Компонент Actions є набором дій, які можна виконати з файлом. Він використовує компоненти Button та Popconfirm з бібліотеки "antd".

Компонент UploadButton є кнопкою завантаження. Він використовує компоненти Button, Upload, UploadFile та notification із бібліотеки "antd", а також іконку CloudUploadOutlined.

Компонент FileCard відображає картку файлу з ім'ям файлу, розширенням та можливо зображенням, якщо файл є зображенням.

Компонент FileList відображає список файлів, використовуючи компонент FileCard для кожного файлу. Він також використовує компонент Selecto для вибору файлів.

Файл LoginForm.tsx у папці auth містить компонент LoginForm, який є формою для входу користувача. У цьому компоненті визначено функцію onSubmit, яка викликається при надсиланні форми. Ця функція приймає значення форми як аргумент і намагається виконати вхід користувача за допомогою цих значень. Якщо вхід успішно, функція встановлює cookie з токеном користувача, відображає повідомлення про успіх та перенаправляє користувача на сторінку /dashboard. Якщо вхід не вдалося, функція відображає повідомлення про помилку. Сама форма містить два поля: "E-Mail" та "Пароль", і кнопку "Увійти" для надсилання форми.

Файл RegisterForm.tsx у папці auth містить компонент RegisterForm, який є формою для реєстрації користувача. Ця функція приймає значення форми як аргумент і намагається зареєструвати користувача за допомогою цих значень. Якщо реєстрація успішна, функція встановлює cookie з токеном користувача, відображає повідомлення про успіх та перенаправляє користувача на сторінку /dashboard. Також як і на LoginForm якщо реєстрація не вдалася, функція відображає повідомлення про помилку.

Файл axios.ts у папці core (рис.3.9) відіграє ключову роль у налаштуванні та експорті модуля axios, який використовується у додатку для надсилання HTTP-запитів. axios - це обіцянка на основі HTTP-клієнта для браузера та Node.js, який надає єдиний API для роботи із запитами та відповідями HTTP.

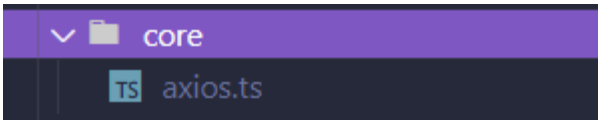

Рис.3.9. Структура папки core

На початку файлу відбувається імпорт модуля axios та функції parseCookies із бібліотеки nookies. nookies - це зручна бібліотека для роботи з cookies на сервері та на клієнті у додатках Next.js. Далі, у файлі встановлюється базова URL-адреса для всіх запитів, що надсилаються через axios. Це означає, що всі запити, надіслані за допомогою axios, будуть надіслані на адресу "http://localhost:7777", якщо не вказано іншу URL-адресу. В кінці файлу налаштований екземпляр axios експортується, щоб його можна було використовувати в інших частинах програми. Це означає, що замість налаштування axios у кожному файлі, де він використовується, ці налаштування винесені в окремий файл, а потім імпортуються звідти.

Файл Files.tsx в папці modules (3.10) є компонентом React, який управляє списком файлів, надаючи функціональність вибору, видалення та обміну файлами.

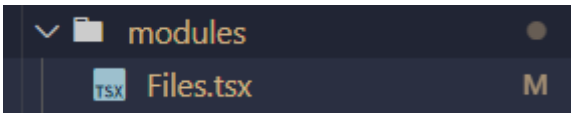

Рис.3.10. Структура папки modules

У проекті, папка pages (3.11) є особливою. Вона використовується для визначення маршрутів вашої програми. Кожен файл у цій папці стає окремою сторінкою у вашій програмі.

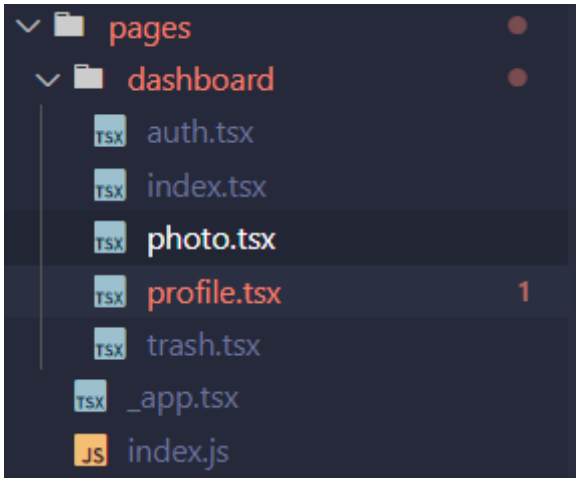

Рис.3.11. Структура папки pages

Файл index визначає головну сторінку панелі керування. Він приймає як властивість масив елементів файлу і відображає їх за допомогою компонента Files.

Файл trash визначає сторінку "Кошик" на панелі керування. Він також приймає масив елементів файлу та відображає їх. Важливо, що він використовує функцію Api.files.getAll("trash") для отримання елементів файлу, що вказує на те, що він відображає лише видалені файли.

Файл photo визначає сторінку "Фото" на панелі керування. Він також приймає масив елементів файлу та відображає їх. Він використовує функцію Api.files.getAll("photo") для отримання елементів файлу, що вказує на те, що він відображає лише файли фотографій.

Файл auth визначає сторінку автентифікації у програмі. Він відображає форму входу та форму реєстрації, які користувач може перемикати за допомогою вкладок.

Файл profile є сторінкою профілю користувача в програмі. Ця сторінка служить для відображення особистої інформації користувача та надає можливість виходу із системи.

Важливо, що кожен із цих файлів використовує функцію getServerSideProps для отримання властивостей сторінки на сервері перед

рендерингом. Це означає, що дані для цих сторінок завантажуються на сервері перед надсиланням сторінки до браузера, що може покращити продуктивність.

Папка utils (рис.3.12) містить функції, які допомагають з автентифікацією, визначенням типу файлу та вибором способу його відображення. Ці функції можуть бути використані в різних частинах програми, тому їх зручно зберігати в централізованому місці, доступному для проекту.

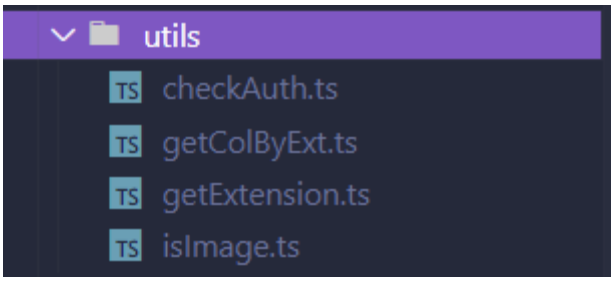

Рис.3.12. Структура папки utils

Файл checkAuth містить функцію checkAuth, яка використовується для перевірки автентифікації користувача. Ця функція приймає контекст GetServerSidePropsContext як аргумент, витягує токен із кука і встановлює його в заголовки запиту axios. Потім вона робить запит до API, щоб отримати інформацію про поточного користувача. Якщо запит успішний, функція повертає об'єкт із порожніми властивостями. Якщо запит невдалий, функція повертає об'єкт, який перенаправляє користувача на сторінку аутентифікації.

Файл getColByExt містить функцію getColByExt, яка приймає розширення файлу як аргумент і повертає колір, асоційований із цим розширенням. Цей колір потім може бути використаний для візуалізації файлу в користувацькому інтерфейсі.

Файл getExtension містить функцію getExtension, яка приймає ім'я файлу як аргумент і повертає його розширення. Це може бути корисно, коли необхідно визначити тип файлу або вибрати спосіб його обробки.

Файл isImage містить функцію isImage, яка приймає розширення файлу і перевіряє, чи є файл зображенням. Це може бути корисно, коли необхідно

визначити, чи слід відображати файл як зображення або використовувати інший спосіб представлення.

3.2. Обґрунтування та організація вхідних та вихідних даних програми

Організація вхідних та вихідних даних у цьому проекті ґрунтується на принципах модульності. Код проекту організований у модулі та компоненти, кожен з яких має свою специфічну функцію. Наприклад, api/files.ts містить функції для роботи з файлами, включаючи отримання всіх файлів, видалення файлів і завантаження файлів. Це полегшує управління кодом і робить його більш зрозумілим.

Проект використовує API для взаємодії з сервером та обміну даними. Це забезпечує централізований спосіб обробки вхідних та вихідних даних. Наприклад, в api/index.ts експортуються модулі auth та files, які надають функції для взаємодії з відповідними API.

Вхідні та вихідні дані в інтерфейсі користувача управляються за допомогою компонентів React. Наприклад, у components/FileActions/index.tsx визначено компонент FileActions, який приймає вхідні дані та повертає JSX-елементи як вихідні дані.

У pages/dashboard/profile.tsx використовується функція getServerSideProps для отримання даних на сервері перед рендерингом сторінки. Це дозволяє завантажувати дані до того, як сторінка буде відправлена до браузера, що покращує продуктивність та SEO.

Організація вхідних і вихідних даних в проекті забезпечує чистоту і структурованість коду, спрощує його підтримку та розвиток, а також забезпечує ефективну взаємодію з сервером та інтерфейсом користувача.

3.3. Опис розробленої системи

3.3.1. Використані технічні засоби

Для створення та тестування інформаційної системи була використана персональна ЕОМ, з наступними системними характеристиками:

- Операційна система: Windows 10 Домашняя;
- Оперативна пам'ять: 16Гб;
- Процесор: AMD Ryzen 5 1600 Six-Core Processor 3.20 GHz;
- 500Мб вільного місця на жорстокму диску;
- Монітор;
- Клавіатура та комп’ютерна миша.

3.3.2. Використані програмні засоби

Для розробки інформаційної системи хмарного сховища я обрав Visual Studio Code. Visual Studio Code- це простий у використанні редактор вихідного коду, який працює на багатьох операційних системах, включаючи macOS, Windows і Linux, з вбудованою підтримкою JavaScript, TypeScript і Node.js, Важливим фактором, який робить Visual Studio Code корисним, є можливість встановлення розширень для інших мов і фреймворків. Це робить процес написання коду більш зручним для професіоналів, а кінцевий код має більш цілісну структуру.

В якості інструмента для перевірки роботи справності серверної частини був використаний Swagger. Swagger дає змогу описувати схему даних для запитів і відповідей із використанням JSON Schema або OpenAPI Specification (раніше відомий як Swagger Specification). Ця схема визначає типи даних, обов'язкові поля, формати й обмеження даних, яких потрібно дотримуватися в запитах і відповідях API.

Коли схему даних визначено, Swagger інструменти можуть використовувати її для автоматичної валідації запитів і відповідей. Наприклад,

під час надсилання запиту до API, Swagger може перевірити, що всі обов'язкові поля присутні, типи даних відповідають очікуваним і значення задовольняють певним обмеженням.

Валідація запитів і відповідей за допомогою Swagger сприяє підвищенню надійності та безпеки API, оскільки допомагає уникнути передачі некоректних або невідповідних даних. Це також спрощує процес розроблення, оскільки дає змогу швидко виявити та виправити помилки в запитах і відповідях до їхнього надсилання або оброблення.

3.3.3. Виклик та завантаження програми

Для роботи з інформаційною системою хмарного сховища достатньо спочатку запустити серверну частину, командою у терміналі. Після успішного запуску серверної частини, треба запустити клієнтську, також командою у терміналі.

3.3.4. Опис інтерфейсу користувача

Після запуску серверної та клієнтської частини хмарного сховища перш ніж почати працювати з даними варто здійснити автентифікацію (рис.3.13) або реєстрацію (рис.3.14) в системі. Після успішної реєстрації (рис.3.15) користувач з'являється в базі даних.

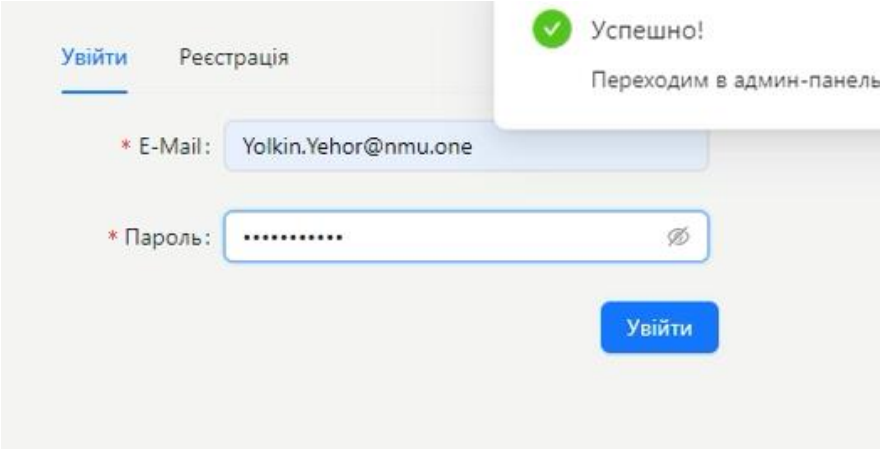

Рис.3.13. Демонстрація успішної аунтефікації

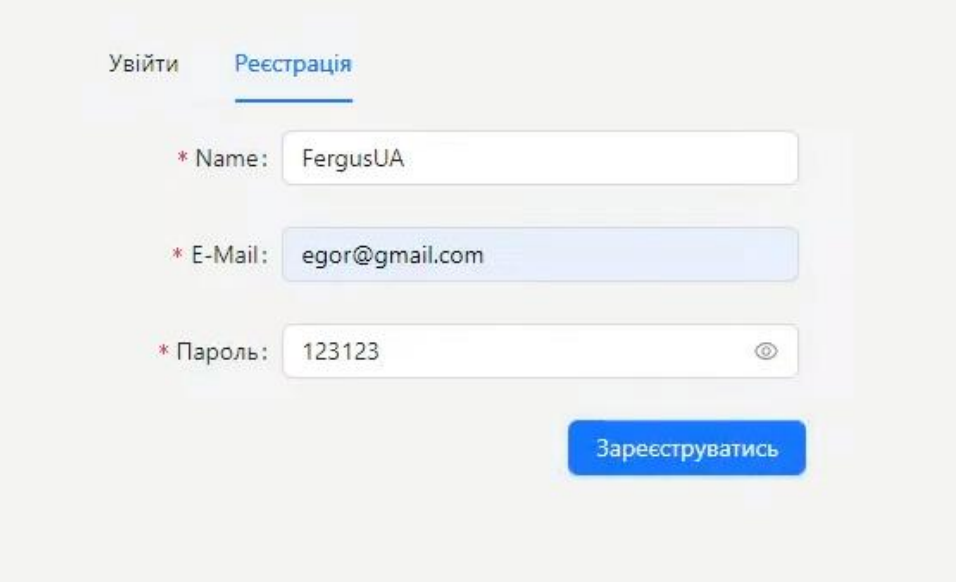

Рис.3.14. Демонстрація реєстрації

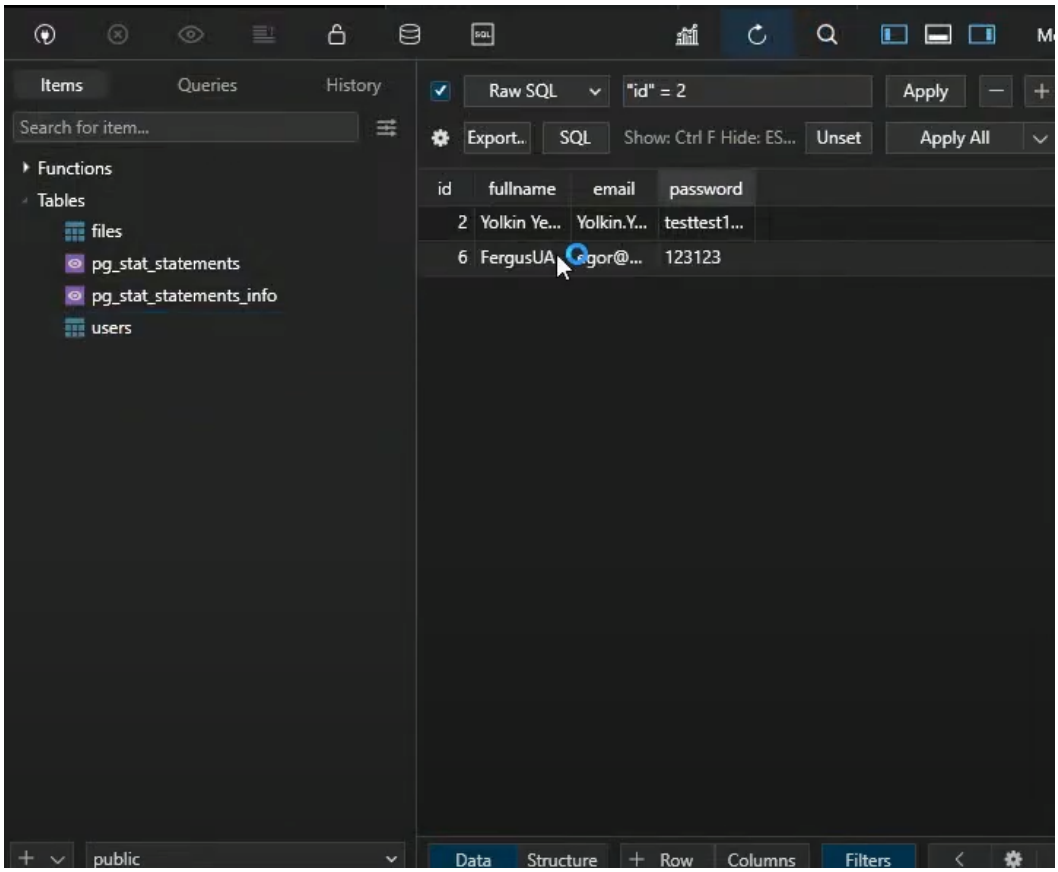

Рис.3.15. Демонстрація зареєстрованого користувача в базі даних

Основний компонент (рис.3.16) розділений на два основні блоки: бічну панель (sidebar) і контейнер (container). Бокова панель містить компоненти UploadBtn і Menu. Menu містить набір елементів, які ведуть на різні сторінки

панелі керування, це формує наш фільтр (рис.3.17). Контейнер слугує областю для відображення дочірніх компонентів, які передаються в HomeLayout як children.

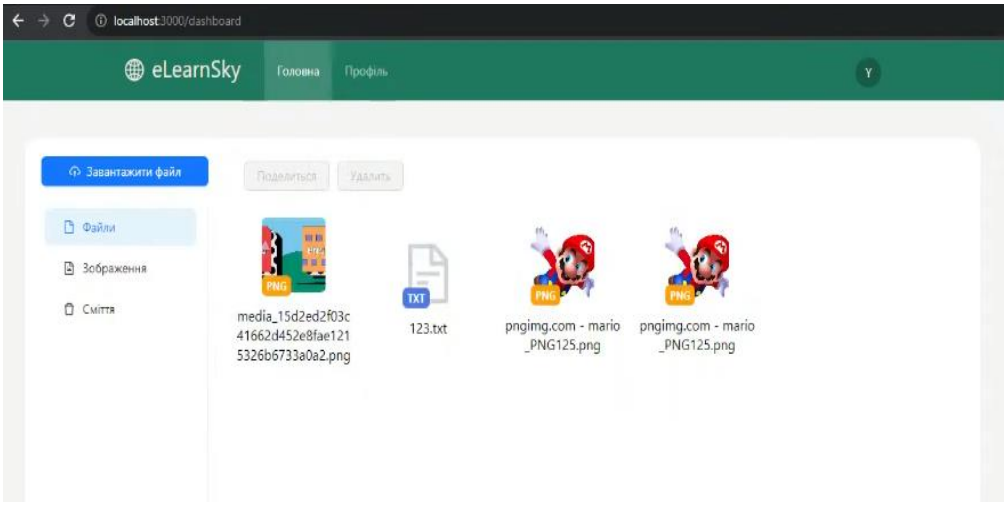

Рис.3.16. Демонстрація основного компоненту

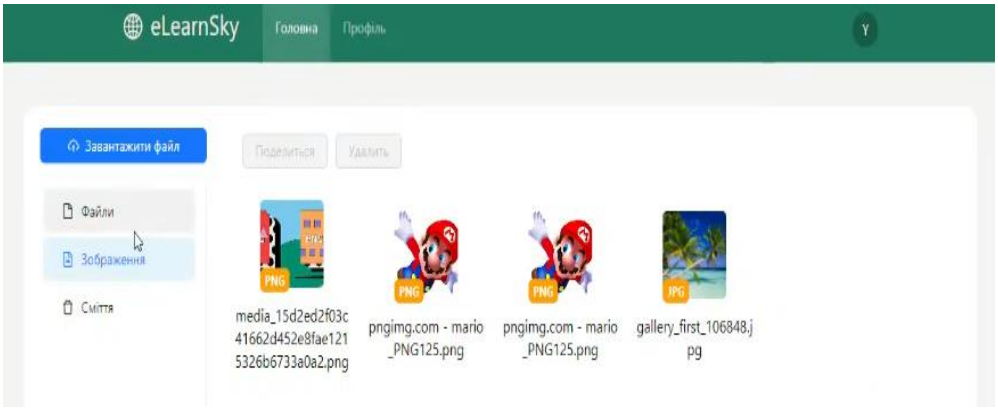

Рис.3.17. Демонстрація роботи фільтру по зображенню

Візуально компонент UploadBtn (рис.3.18) являє собою кнопку з іконкою хмари і текстом "Завантажити файл". Коли користувач натискає на цю кнопку, відкривається системне вікно вибору файлів, що дає змогу користувачеві вибрати один або кілька файлів для завантаження. Якщо завантаження проходить успішно (рис.3.19), список fileList очищається, і сторінка перезавантажується, щоб відобразити нові файли (рис.3.20). Якщо під час завантаження відбувається помилка, відображається повідомлення з повідомленням про помилку "Неможливо завантажити файл".

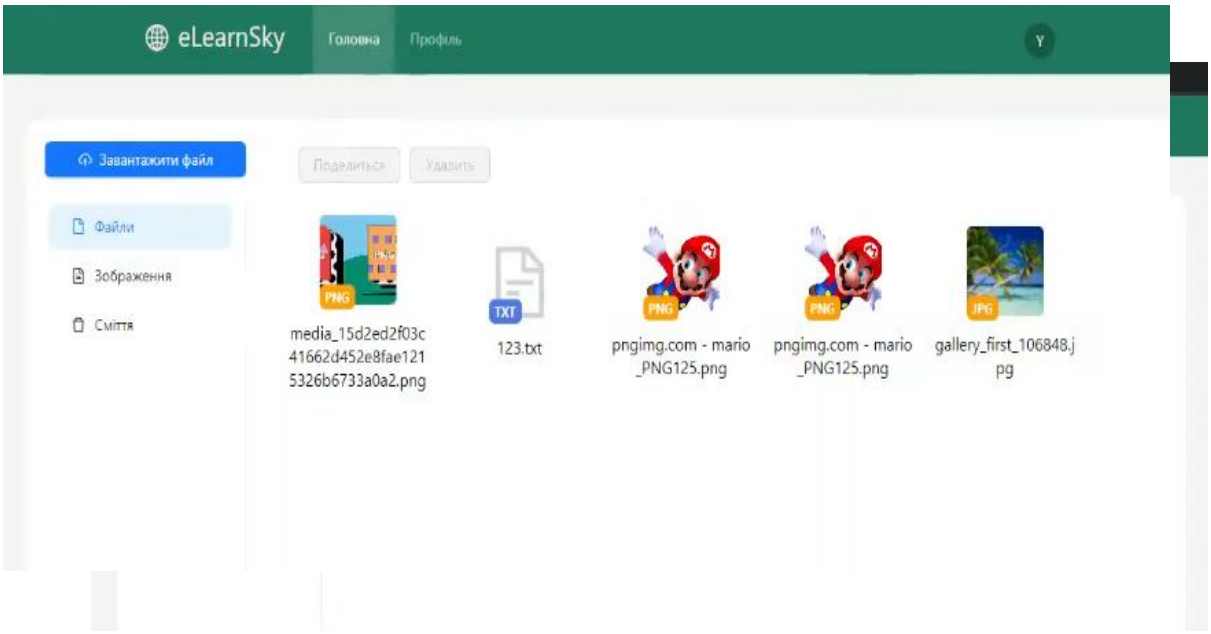

Рис.3.18. Демонстрація вибору файлу для завантаження

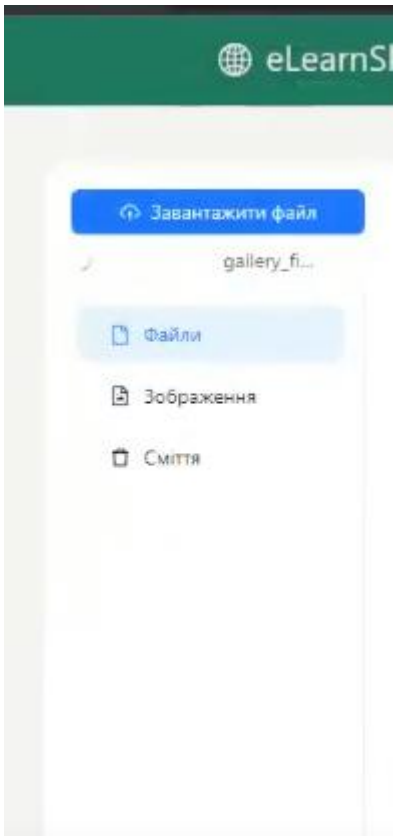

Рис.3.19. Демонстрація завантаження файлу

Рис.3.20. Демонстрація успішного завантаження файлу

Коли користувач натискає на кнопку "Видалити" (рис.3.21), спочатку з'являється спливаюче вікно з підтвердженням. Це реалізовано за допомогою компонента Popconfirm з бібліотеки antd. У цьому спливаючому вікні користувачеві пропонується підтвердити свій намір видалити файл. Якщо користувач підтверджує видалення, файл з'являється у розділі "Сміття" (рис.3.22), але не видаляється з бази даних. Після натискання кнопки "Завантажити" обраний файл завантажується на систему користувача.

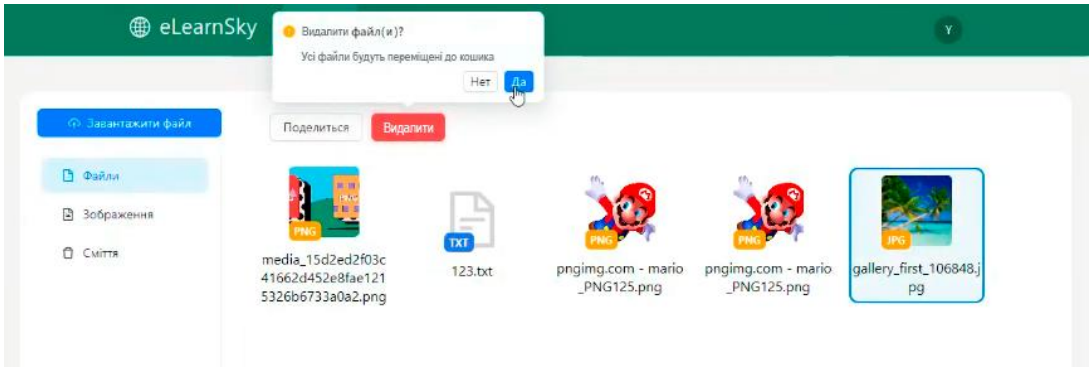

Рис.3.21. Демонстрація видалення файлу

Рис.3.22. Демонстрація розділу видалених файлів

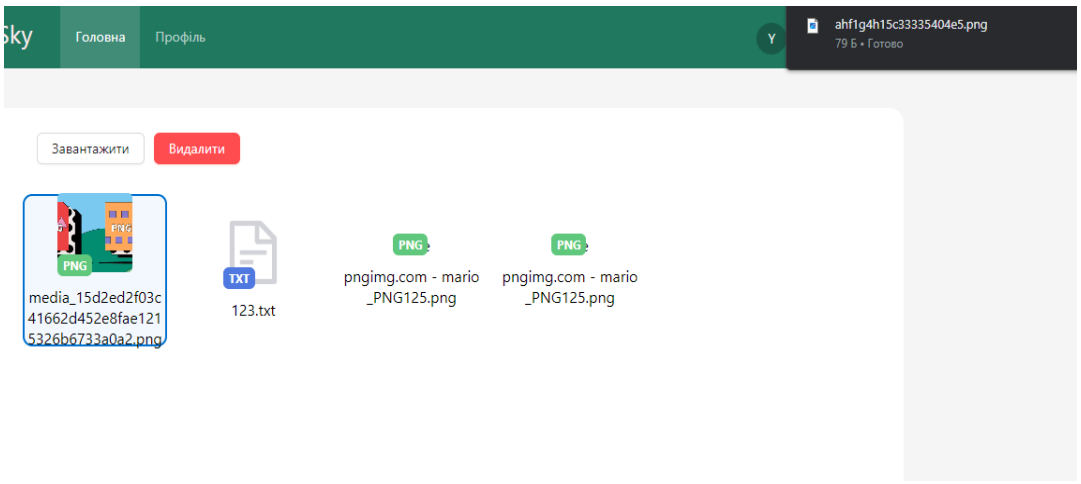

Рис.3.23. Демонстрація завантаження файлу на систему користувача

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було проаналізовано вплив хмарних сховищ на продуктивність та ефективність бізнес-процесів, у підсумку була розроблена демонстраційна інформаційна веб орієнтована система хмарного сховища. Проведене дослідження підтверджує актуальність та критичність впровадження хмарних сховищ для підприємств, оскільки вони мають потребу в ефективних інструментах для підвищення продуктивності та зменшення витрат. В ході дослідження було розглянуто різні підходи до впровадження хмарних технологій в "екосистему" підприємств. Було визначено що впровадження хмарних технологій в бізнес, є одним з надпотужних і перспективних технологій у сучасному світі.

Дослідження також звертає увагу на важливість врахування етичних аспектів застосування хмарних технологій. Використання описово-аналітичного методу та інструменту Кронбаха-альфа для вимірювання ступеня застосування професійної етики виявило високу внутрішню узгодженість та стабільність результатів, підтверджуючи надійність отриманих даних. Було проаналізовано як сильно впливають хмарні технології на організаційну ефективність, шляхом використання коефіцієнтів кореляції та множинної регресії. Як підсумок підтвердив статично значущий позитивний зв'язок між застосуванням цих технологій та підвищенням продуктивності і адаптації до робочого середовища.

У третьому розділі була розроблена демонстраційна веб орієнтована інформаційна система хмарного сховища. В ході розробки було підібрано стек технологій, для серверної та клієнтської частини. Було реалізовано завантаження та вивантаження файлів на сервер, як елемент безпеки було реалізовано систему реєстрації та авторизації. Для зручнішого користування системою була реалізована система фільтрації файлів щодо його розширення.

Дослідження в даній роботі виявило потенціал хмарних сховищ для підвищення ефективності бізнес-процесів, а впроваджена хмарна система стала конкретним прикладом успішної оптимізації та підвищення продуктивності в

умовах сучасного бізнес-середовища. Отже, можна визначити, що використання хмарних технологій стає необхідністю для сучасних підприємств, які прагнуть залишатися конкурентоспроможними та високоефективними в умовах постійних змін та розвитку.

СПИСОК ВИКОРИСТАННИХ ДЖЕРЕЛ

1. RightScale 2019 State of the Cloud Report – [Електронний ресурс] – Режим доступу: https://resources.flexera.com/web/media/documents/rightscale-2019-state-of-the-cloud-report-from-flexera.pdf (дата звернення: 25.10.23)

2. Why Data and Analytics Are Key to Digital Transformation – [Електронний ресурс] – Режим доступу: https://www.gartner.com/smarterwithgartner/why-data-and-analytics-are-key-to-digital-transformation (дата звернення: 26.10.23)

3. McHaney R. Cloud Technologies: An Overview of Cloud Computing Technologies for Managers. Wiley & Sons, Incorporated, John, 2021. – 288 p.

4. Flexera 2023 State of the Cloud Report Press Release – [Електронний ресурс] – Режим доступу: https://www.flexera.com/about-us/press-center/flexera-2023-state-of-the-cloud-report (дата звернення: 29.10.23)

5. Organizations Rely on Cloud Storage to Optimize Cost, Increase Agility, and Drive Innovation – [Електронний ресурс] – Режим доступу: https://pages.awscloud.com (дата звернення: 01.11.23)

6. Faynberg I., Lu H.-L., Skuler D. Cloud Computing: Business Trends and Technologies. Wiley & Sons, Limited, John, 2015. – 376 с.

7. Huawei Technologies Co., Ltd. Cloud Computing Technology. Singapore: Springer Nature Singapore, 2023. – [Електронний ресурс] – Режим доступу: https://doi.org/10.1007/978-981-19-3026-3 (дата звернення: 12.11.2023).

8. Erl T. Cloud Computing: Concepts, Technology & Architecture. Pearson, 2013. – 489 с.

9. Webb A. Big Nine: How the Tech Titans and Their Thinking Machines Could Warp Humanity. Public Affairs, 2019. – 336 с.

10. Modi R. Azure for Architects: Implementing cloud design, DevOps, IoT, and serverless solutions on your public cloud. Packt Publishing, 2017. 358 с.

11. Kavis M. J. Architecting the cloud: design decisions for cloud computing service models. Wiley & Sons, Incorporated, John, 2014. – 224 с.

12. Kim G., Humble J., PhD N. F. Accelerate: the science of lean software and devops: building and scaling high performing technology organizations. IT Revolution Press, 2018. – 288 с.

13.Weinman J. Cloudonomics: The Business Value of Cloud Computing. Wiley & Sons, Incorporated, John, 2012. – 416 с.

14. Dutt D. G. Cloud Native Data Center Networking: Architecture, Protocols, and Tools. O'Reilly Media, 2019. – 486 с.

15. Tigani J. Google BigQuery Analytics. Wiley, 2014. – 528 с.

16. Inmon W. H., Linstedt D. Data Architecture: Primer for the Data Scientist: Big Data, Data Warehouse and Data Vault. Elsevier Science & Technology Books, 2014. – 378 с.

17. Офіційна документація NestJS. – [Електронний ресурс] – Режим доступу: https://docs.nestjs.com (дата звернення: 02.10.2023).

18. Офіційна документація PostgreSQL. – [Електронний ресурс] – Режим доступу: https://www.postgresql.org/docs (дата звернення: 02.10.23).

19.Офіційна документація Axios. – [Електронний ресурс] – Режим доступу: https://axios-http.com/en/docs/intro (дата звернення: 05.10.2023).

20. Офіційна документація PassportJS. – [Електронний ресурс] – Режим доступу: https://www.passportjs.org/docs/ (дата звернення: 05.10.2023).

21. Gilbert J. Cloud Native Development Patterns and Best Practices: Practical architectural patterns for building modern, distributed cloud-native systems. Packt Publishing, 2018. – 316 с.

22. Zou S.-H., Fang N.-S., Gao W.-J. Research on online cloud storage technology. 2020 19th International Symposium on Distributed Computing and Applications for Business Engineering and Science Xuzhou, 2020. – [Електронний ресурс] – Режим доступу: https://doi.org/10.1109/dcabes50732.2020.00025 (дата звернення: 05.10.2023).

23. Bogataj Habjan K., Pucihar A. The Importance of Business Model Factors for Cloud Computing Adoption: Role of Previous Experiences. Organizacija. 2017. –

[Електронний ресурс] – Режим доступу: https://doi.org/10.1515/orga-2017-0013 (дата звернення: 08.10.2023).

24. Kassner E., Briggs B. Enterprise Cloud Strategy. Microsoft Press, 2016. – 156с.

25. Weinman J. Cloudonomics: The Business Value of Cloud Computing. Wiley & Sons, Incorporated, John, 2015. 416 с.

ДОДАТОК А

ЛІСТИНГ ПРОГРАМИ

graduate-project\src\auth\guard\gwt.guard.ts

```
import { Injectable } from '@nestjs/common';
import { AuthGuard } from '@nestjs/passport';
@Injectable()
export class JwtAuthGuard extends AuthGuard('jwt') {}
```

graduate-project\src\auth\guard\local.guard.ts

```
import { Injectable } from '@nestjs/common';
import { AuthGuard } from '@nestjs/passport';
@Injectable()
export class LocalJwtAuthGuard extends AuthGuard('local') { }
```

graduate-project\src\auth\strategies\jwt.strategy.ts

```
import { ExtractJwt, Strategy } from 'passport-jwt';
import { PassportStrategy } from '@nestjs/passport';
import { Injectable, UnauthorizedException } from '@nestjs/common';
import { UsersService } from 'src/users/users.service';
@Injectable()
export class JwtStrategy extends PassportStrategy(Strategy) {
   constructor(private readonly userService: UsersService) {
      super({
         jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),
         ignoreExpiration: false,
         secretOrKey: process.env.SECRET_KEY,
      });
   }
   async validate(payload: any) {
      const user = await this.userService.findById(+payload.id)

      if (!user) {
         throw new UnauthorizedException('Ви не маєте доступу')
      }
      return {
         id: user.id,
      }
   }
}
```

graduate-project\src\auth\strategies\local.strategy.ts

```
import { Strategy } from 'passport-local';
```

```
import { PassportStrategy } from '@nestjs/passport';
import { Controller, Injectable, UnauthorizedException } from '@nestjs/common';
import { AuthService } from '../auth.service';
@Injectable()
export class LocalStrategy extends PassportStrategy(Strategy) {
   constructor(private authService: AuthService) {
      super({
         usernameField: 'email',
      });
   }
   async validate(username: string, password: string): Promise<any> {
      const user = await this.authService.validateUser(username, password);
      if (!user) {
         throw new UnauthorizedException('Некоректний логін чи пароль');
      }
      return user;
   }
}
```

graduate-project\src\auth\auth.controller.ts

```
import { Controller, Post, UseGuards, Request, Body } from '@nestjs/common';
import { AuthGuard } from '@nestjs/passport';
import { ApiBody } from '@nestjs/swagger';
import { CreateUserDto } from 'src/users/dto/create-user.dto';
import { AuthService } from './auth.service';
import { User } from 'src/users/entities/user.entity';
import { LocalJwtAuthGuard } from './guard/local.guard';
@Controller('auth')
export class AuthController {
   constructor(private readonly authService: AuthService) { }
   @UseGuards(LocalJwtAuthGuard)
   @Post('login')
   @ApiBody({ type: CreateUserDto })
   async login(@Request() req) {
      return this.authService.login(req.user as User)
   }
   @Post('/register')
   register(@Body() dto: CreateUserDto) {
      return this.authService.register(dto)}}
```

graduate-project\src\auth\auth.module.ts

```
import { Module } from '@nestjs/common';
import { AuthController } from './auth.controller';
import { AuthService } from './auth.service';
import { UsersModule } from 'src/users/users.module';
import { LocalStrategy } from './strategies/local.strategy';
import { PassportModule } from '@nestjs/passport';
```

```
import { JwtModule } from '@nestjs/jwt';
import { ConfigModule, ConfigService } from '@nestjs/config';
import { JwtStrategy } from './strategies/jwt.strategy';
@Module({
  imports: [
    JwtModule.registerAsync({
      imports: [ConfigModule],
      inject: [ConfigService],
      useFactory: async (configService: ConfigService) => {
        return {
          secret: configService.get('SECRET_KEY'),
          signOptions: { expiresIn: configService.get('EXPIRES_IN') },
        }
      }
    }),
    UsersModule, PassportModule],
  controllers: [AuthController],
  providers: [AuthService, LocalStrategy, JwtStrategy]
})
export class AuthModule { }
```

graduate-project\src\auth\auth.service.ts

```
import { ForbiddenException, Injectable } from '@nestjs/common';
import { JwtService } from '@nestjs/jwt';
import { CreateUserDto } from 'src/users/dto/create-user.dto';
import { User } from 'src/users/entities/user.entity';
import { UsersService } from 'src/users/users.service';
@Injectable()
export class AuthService {
    constructor(private usersService: UsersService, private jwtService: JwtService) { }
    async validateUser(email: string, password: string): Promise<any> {
        const user = await this.usersService.findByEmail(email)
        if (user && user.password == password) {
            const { password, ...result } = user;
            return result;
        }
      return null
    }
    async register(dto: CreateUserDto) {
        try {
            const userData = await this.usersService.create(dto)
            return {
                token: this.jwtService.sign({ id: userData.id })
            };
        }
        catch (err) {
            console.log(err);
            throw new ForbiddenException('Помилка при реєстрації')
        }
```

```
    }
    async login(user: User) {
        return {
            token: this.jwtService.sign({ id: user.id }),
        }
    }
}
```

graduate-project\src\decorator\userId.decorator.ts

```
import { createParamDecorator, ExecutionContext } from '@nestjs/common';
export const UserId = createParamDecorator(
    (_: unknown, ctx: ExecutionContext): number | null => {
        const request = ctx.switchToHttp().getRequest();
        return request.user?.id ? Number(request.user.id) : null;
    },
);
```

graduate-project\src\files\files.controller.ts

```
import { Controller, Get, Post, Body, Patch, Param, Delete, UseInterceptors, UploadedFile,
ParseFilePipe, MaxFileSizeValidator, UseGuards, Query, Res } from '@nestjs/common';
import { FilesService } from './files.service';
import { CreateFileDto } from './dto/create-file.dto';
import { UpdateFileDto } from './dto/update-file.dto';
import { ApiBearerAuth, ApiBody, ApiConsumes, ApiTags } from '@nestjs/swagger';
import { FileInterceptor } from '@nestjs/platform-express';
import { fileStorage } from './storage';
import { JwtAuthGuard } from 'src/auth/guard/gwt.guard';
import { UserId } from 'src/decorator/userId.decorator';
import { FileType } from './entities/file.entity';
import { Response } from 'express';
@Controller('files')
@ApiTags('files')
export class FilesController {
  constructor(private readonly filesService: FilesService) { }
  @Get(':id/download')
  async downloadFile(@Param('id') id: number, @Res() res: Response) {
    const file = await this.filesService.getFileById(id);
    if (file && file.filename) {
      res.download(`uploads/${file.filename}`);
    } else {
      res.status(404).send('File not found');
    }
  }
  @UseGuards(JwtAuthGuard)
  @ApiBearerAuth()
  @Get()
  findAll(@UserId() userId: number, @Query('type') fileType: FileType) {
    return this.filesService.findAll(userId, fileType)
```

```
  }
  @Post()
  @UseInterceptors(
    FileInterceptor('file', {
      storage: fileStorage,
    })
  )
  @ApiConsumes('multipart/form-data')
  @ApiBody(
    {
      schema: {
        type: 'object',
        properties: {
          file: {
            type: 'string',
            format: 'binary'
          }
        }
      }
    }
  )
  create(@UploadedFile(
    new ParseFilePipe({
      validators: [new MaxFileSizeValidator({ maxSize: 1024 * 1024 * 5 })]
    })
  ) file: Express.Multer.File, @UserId() userId: number,) {
    return this.filesService.create(file, userId);
  }
  @Delete()
  remove(@UserId() userId: number, @Query('ids') ids: string) {

    return this.filesService.remove(userId, ids);
  }
}
```

graduate-project\src\files\files.module.ts

```
import { Module } from '@nestjs/common';
import { FilesService } from './files.service';
import { FilesController } from './files.controller';
import { TypeOrmModule } from '@nestjs/typeorm';
import { File } from './entities/file.entity';
@Module({
  controllers: [FilesController],
  providers: [FilesService],
  imports: [TypeOrmModule.forFeature([File])],
})
export class FilesModule {}
```

graduate-project\src\files\files.service.ts

```
import { Injectable } from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import { File, FileType } from './entities/file.entity';
import { Repository } from 'typeorm';
@Injectable()
export class FilesService {
  constructor(
    @InjectRepository(File)
    private repository: Repository<File>,
  ) { }
  findAll(userId: number, fileType: FileType) {
    const qb = this.repository.createQueryBuilder('file');

    qb.where('file.userId = :userId', { userId });

    if (fileType === FileType.PHOTO) {
      qb.andWhere('file.mimetype ILIKE :type', { type: '%image%' });
    }

    if (fileType === FileType.TRASH) {
      qb.withDeleted().andWhere('file.deletedAt IS NOT NULL');
    }

    return qb.getMany();
  }
  create(file: Express.Multer.File, userId: number) {
    return this.repository.save({
      filename: file.filename,
      originalname: file.originalname,
      size: file.size,
      mimetype: file.mimetype,
      user: { id: userId },
    });
  }
  async getFileById(id: number) {
    return this.repository.findOne({ where: { id } });
  }
  async remove(userId: number, ids: string) {
    const idsArray = ids.split(',');
    const qb = this.repository.createQueryBuilder('file');
    qb.where('id IN (:...ids) AND userId = :userId', {
      ids: idsArray,
      userId,
    });
    return qb.softDelete().execute();
  }
}
graduate-project\src\files\storage.ts
import { diskStorage } from "multer";
```

```
const genID = () =>
   Array(20)
      .fill(null)
      .map(() => Math.round(Math.random() * 18).toString(18))
      .join('')


const genFileName = (req, file, callback) => {
   const fileExtName = file.originalname.split('.').pop();

   callback(null, `${genID()}.${fileExtName}`)
}
export const fileStorage = diskStorage({
   destination: './uploads',
   filename: genFileName,})
```

graduate-project\src\users\entities\user.entity.ts

```
import { File } from "src/files/entities/file.entity";
import { Column, Entity, OneToMany, PrimaryGeneratedColumn } from "typeorm";
@Entity('users')
export class User {
   @PrimaryGeneratedColumn()
   id: number;
   @Column()
   fullname: string;
   @Column()
   email: string;
   @Column()
   password: string;
   @OneToMany(() => File, file => file.user)
   files: File[];
}
```

graduate-project\src\users\users.controller.ts

```
import { Controller, Get, Post, Body, Patch, Param, Delete, UseGuards } from '@nestjs/common';
import { UsersService } from './users.service';
import { CreateUserDto } from './dto/create-user.dto';
import { UpdateUserDto } from './dto/update-user.dto';
import { ApiBearerAuth, ApiTags } from '@nestjs/swagger';
import { JwtAuthGuard } from 'src/auth/guard/gwt.guard';
import { UserId } from 'src/decorator/userId.decorator';
@Controller('users')
@ApiTags('users')
@ApiBearerAuth()
export class UsersController {
  constructor(private readonly usersService: UsersService) { }
  @Get('/me')
  @UseGuards(JwtAuthGuard)
  getMe(@UserId() id: number) {
```

```
    return this.usersService.findById(id)
  }
}
```

graduate-project\src\users\users.module.ts

```
import { Module } from '@nestjs/common';
import { UsersService } from './users.service';
import { UsersController } from './users.controller';
import { TypeOrmModule } from '@nestjs/typeorm';
import { User } from './entities/user.entity';

@Module({
  controllers: [UsersController],
  providers: [UsersService],
  imports: [TypeOrmModule.forFeature([User])],
  exports:[UsersService],
})
export class UsersModule {}
```

graduate-project\src\users\users.service.ts

```
import { Injectable } from '@nestjs/common';
import { CreateUserDto } from './dto/create-user.dto';
import { UpdateUserDto } from './dto/update-user.dto';
import { InjectRepository } from '@nestjs/typeorm';
import { User } from './entities/user.entity';
import { Repository } from 'typeorm';
@Injectable()
export class UsersService {
  constructor(
    @InjectRepository(User)
    private repository: Repository<User> //Возможно ошибка
  ) { }
  async findByEmail(email: string) {
    return this.repository.findOneBy({
      email,
    })
  }
  async findById(id: number) {
    return this.repository.findOneBy({
      id,
    })
  }
  create(dto:CreateUserDto){
    return this.repository.save(dto)
  }
}
```

graduate-project\src\app.controller.ts

```
import { Controller, Get } from '@nestjs/common';
import { AppService } from './app.service';

@Controller()
export class AppController {
  constructor(private readonly appService: AppService) {}

  @Get()
  getHello(): string {
    return this.appService.getHello();
  }
}
```

graduate-project\src\app.module.ts

```
import { Module } from '@nestjs/common';
import { AppController } from './app.controller';
import { AppService } from './app.service';
import { UsersModule } from './users/users.module';
import { FilesModule } from './files/files.module';
import { TypeOrmModule } from '@nestjs/typeorm';
import { User } from './users/entities/user.entity';
import { File } from './files/entities/file.entity';
import { ConfigModule } from '@nestjs/config';
import { AuthModule } from './auth/auth.module';
@Module({
  imports: [
    ConfigModule.forRoot(),
    TypeOrmModule.forRoot({
    type: 'postgres',
    host: process.env.DB_HOST,
    port: Number(process.env.DB_PORT) || 5432,
    username: process.env.DB_USER,
    password: process.env.DB_PASSWORD,
    database: process.env.DB_NAME,
    entities: [User, File],
    synchronize: true,
  }),
    UsersModule,
    FilesModule,
    AuthModule],
  controllers: [AppController],
  providers: [AppService],
})
export class AppModule { }
```

graduate-project\src\app.service.ts

```
import { Injectable } from '@nestjs/common';
@Injectable()
export class AppService {
  getHello(): string {
    return 'Hello World!!';
  }
}
```

graduate-project\src\main.ts

```
import { NestFactory } from '@nestjs/core';
import { AppModule } from './app.module';
import { DocumentBuilder, SwaggerModule } from '@nestjs/swagger';
import * as express from 'express'
import { join } from 'path';
async function bootstrap() {
  const app = await NestFactory.create(AppModule, { cors: false });
  app.enableCors({ credentials: true, origin: true })
  app.use('/uploads', express.static(join(__dirname, '..', '/uploads')))
  const config = new DocumentBuilder()
    .setTitle('Cloud')
    .setVersion('1.0')
    .addBearerAuth()
    .build();
  const document = SwaggerModule.createDocument(app, config);
  SwaggerModule.setup('swagger', app, document, {
    swaggerOptions: {
      persistAuthorization: true,
    }
  });
  await app.listen(7777);
}
bootstrap();
```

graduate-project-front\api\dto\auth.dto.ts

```
export interface LoginFormDTO {
    email: string;
    password: string;
}
export interface LoginResponseDTO {
    token: string;
}
export type RegisterFormDTO = LoginFormDTO & { fullname: string };
export type RegisterResponseDTO = LoginResponseDTO;
export interface User {
    id: number;
    fullname: string;
    email: string;
```

```
}
```

graduate-project-front\api\dto\files.dto.ts

```
import { User } from "../dto/auth.dto";
export interface FileItem {
  filename: string;
  originalname: string;
  size: number;
  mimetype: string;
  user: User;
  deletedAt: string | null;
  id: number;
}
```

graduate-project-front\api\auth.ts

```
import { destroyCookie } from "nookies";
import axios from "../core/axios";
import { LoginFormDTO, LoginResponseDTO, RegisterFormDTO, RegisterResponseDTO, User }
from "./dto/auth.dto";
export const login = async (
  values: LoginFormDTO
): Promise<LoginResponseDTO> => {
  return (await axios.post("/auth/login", values)).data;
};
export const register = async (
  values: RegisterFormDTO
): Promise<RegisterResponseDTO> => {
  return (await axios.post("/auth/register", values)).data;
};
export const getMe = async (): Promise<User> => {
  return (await axios.get("/users/me")).data;
};
export const logout = () => {
  destroyCookie(null, "_token", { path: "/" });
};
```

graduate-project-front\api\files.ts

```
import axios from '../core/axios'
import { FileItem } from './dto/files.dto'
type FileType = "all" | "photo" | "trash";
export const getAll = async (type: FileType = "all"): Promise<FileItem[]> => {
    return (await axios.get("/files?type=" + type)).data;
};
export const remove = (ids: number[]): Promise<void> => {
    return axios.delete("/files?ids=" + ids);
};
```

```
export const get = (ids: number[]): Promise<FileItem[]> => {
   return axios.get("/files?ids=" + ids)
      .then(response => response.data); // Извлекаем данные из ответа
};
export const uploadFile = async (options: any) => {
   const { onSuccess, onError, file, onProgress } = options;
   const formData = new FormData();
   formData.append("file", file);
   const config = {
      headers: { "Content-Type": "multipart/form-data" },
      onProgress: (event: ProgressEvent) => {
         onProgress({ percent: (event.loaded / event.total) * 100 });
      },
   };
   try {
      const { data } = await axios.post("files", formData, config);
      onSuccess();
      return data;
   } catch (err) {
      onError({ err });
   }
};
```

graduate-project-front\api\index.ts

```
export * as auth from "./auth";
export * as files from "./files";
```

graduate-project-front\component\Actions\index.tsx

```
import React from "react";
import styles from "./Actions.module.scss";
import { Button, Popconfirm } from "antd";

interface FileActionsProps {
  onClickRemove: VoidFunction;
  onClickShare: VoidFunction;
  isActive: boolean;
}
export const Actions: React.FC<FileActionsProps> = ({
  onClickRemove,
  onClickShare,
  isActive,
}) => {
  return (
    <div className={styles.root}>
      <Button onClick={onClickShare} disabled={!isActive}>
        Завантажити
      </Button>
      <Popconfirm
```

```
      title="Удалить файл(ы)?"
      description="Все файлы будут перемещены в корзину"
      okText="Да"
      cancelText="Нет"
      disabled={!isActive}
      onConfirm={onClickRemove}
    >
      <Button disabled={!isActive} type="primary" danger>
       Видалити
      </Button>
    </Popconfirm>
  </div>
 );
};
```

graduate-project-front\component\auth\LoginForm.tsx

```
import React from "react";
import styles from './LoginForm.module.scss'
import { setCookie } from "nookies";
import { Button, Form, Input, notification } from "antd";
import { LoginFormDTO } from "../../api/dto/auth.dto";
import * as Api from "../../api";
export const LoginForm: React.FC = () => {
  const onSubmit = async (values: LoginFormDTO) => {
    try {
      const { token } = await Api.auth.login(values);
      notification.success({
        message: "Успішна авторизація",
        description: "Переходимо до адмін панелі",
        duration: 2,
      });
      setCookie(null, "_token", token, {
        path: "/",
      });
      location.href = "/dashboard";
    } catch (err) {
      console.warn("LoginForm", err);
      notification.error({
        message: "Помилка!",
        description: "Невірний логін або пароль",
        duration: 2,
      });
    }
  };
  return <div className={styles.formBlock}>
    <Form
      name="basic"
      labelCol={{
        span: 6,
```

```
        }}
        onFinish={onSubmit}>
        <Form.Item
          label="E-Mail"
          name="email"
          rules={[{ required: true, message: 'Введіть свій e-mail' }]}
        >
          <Input></Input>
        </Form.Item>
        <Form.Item
          label="Пароль"
          name="password"
          rules={[{ required: true, message: 'Введіть свій пароль' }]}
        >
          <Input.Password
            type="password"
            placeholder="Password"></Input.Password>
        </Form.Item>

        <Form.Item
          wrapperCol={{
            span: 1,
            offset: 20,
          }}>
          <Button type="primary" htmlType="submit">
            Увійти
          </Button>
        </Form.Item>
      </Form>
    </div>
}
```

graduate-project-front\component\auth\RegisterForm.tsx

```
import React from "react";
import styles from './RegisterForm.module.scss'
import { Button, Form, Input, notification } from "antd";
import { RegisterFormDTO } from "../../api/dto/auth.dto";
import { setCookie } from "nookies";
import * as Api from "../../api";
export const RegisterForm: React.FC = () => {
  const onSubmit = async (values: RegisterFormDTO) => {
    try {
      const { token } = await Api.auth.register(values);
      notification.success({
        message: "Успешно!",
        description: "Переходим в админ-панель...",
        duration: 2,
      });
      setCookie(null, "_token", token, {
```

```
            path: "/",
        });
        location.href = "/dashboard";
    } catch (err) {
        console.warn(err);

        notification.error({
            message: "Помилка!",
            description: "Помилка під час реєстрації",
            duration: 2,
        });
    }
};
return (
    <div className={styles.root}>
        <Form
            name="basic"
            labelCol={{
                span: 6,
            }}
            onFinish={onSubmit}
        >
            <Form.Item
                label="Name"
                name="fullname"
                rules={[{ required: true, message: "Введіть своє ім'я" }]}
            >
                <Input></Input>
            </Form.Item>
            <Form.Item
                label="E-Mail"
                name="email"
                rules={[{ required: true, message: 'Введіть свій e-mail' }]}
            >
                <Input></Input>
            </Form.Item>
            <Form.Item
                label="Пароль"
                name="password"
                rules={[{ required: true, message: 'Введіть свій пароль' }]}
            >
                <Input.Password
                    type="password"
                    placeholder="Password"></Input.Password>
            </Form.Item>

            <Form.Item
                wrapperCol={{
                    span: 1,
                    offset: 16,
```

```
            }}>
            <Button type="primary" htmlType="submit">
              Зареєструватись
            </Button>
          </Form.Item>
        </Form>
      </div>
    )
}
```

graduate-project-front\component\FileCard\index.tsx

```
import React from "react";
import styles from "./FileCard.module.scss";
import { getExtension } from "../../utils/getExtension";
import { isImage } from "../../utils/isImage";
import { getColByExt } from "../../utils/getColByExt";
import { FileTextOutlined } from "@ant-design/icons";
interface FileCardProps {
  filename: string;
  originalname: string;
}
export const FileCard: React.FC<FileCardProps> = ({
  originalname,
  filename,
}) => {
  const ext = getExtension(filename);
  const imageUrl =
    ext && isImage(ext) ? "http://localhost:7777/uploads/" + filename : "";
  const color = getColByExt(ext);
  const classColor = styles[color];
  return (
    <div className={styles.root}>
      <div className={styles.icon}>
        <i className={classColor}>{ext}</i>
        {isImage(ext) ? (
          <img className={styles.image} src={imageUrl} alt="File" />
        ) : (
          <FileTextOutlined />
        )}
      </div>
      <span>{originalname}</span>
    </div>
  );
};
```

graduate-project-front\component\FileList\index.tsx

```
import React from "react";
import styles from "./FileList.module.scss"
import { FileItem } from "../../api/dto/files.dto";
import { FileCard } from "../FileCard/";
import Selecto from "react-selecto";
export type FileSelectType = "select" | "unselect";
interface FileListProps {
    items: FileItem[];
    onFileSelect: (id: number, type: FileSelectType) => void;
  }
export const FileList: React.FC<FileListProps> = ({ items, onFileSelect }) => {
  return (
    <div className={styles.root}>
      {items.map((item) => (
        <div data-id={item.id} key={item.id} className="file">
          <FileCard filename={item.filename} originalname={item.originalname} />
        </div>
      ))}
      <Selecto
        selectableTargets={[".file"]}
        selectByClick
        hitRate={10}
        selectFromInside
        toggleContinueSelect={["shift"]}
        continueSelect={false}
        onSelect={(e) => {
          e.added.forEach((el) => {
            el.classList.add("active");
            onFileSelect(Number(el.dataset["id"]), "select");
          });
          e.removed.forEach((el) => {
            el.classList.remove("active");
            onFileSelect(Number(el.dataset["id"]), "unselect");
          });
        }}
      />
    </div>
  );
};
```

graduate-project-front\component\Header\index.tsx

```
import { Avatar, Button, Layout, Menu, Popover } from "antd"
import React from "react"
import styles from "./Header.module.scss"
import { GlobalOutlined } from "@ant-design/icons";
import { useRouter } from "next/router";
import * as Api from "../../api"
export const Header: React.FC = () => {
   const router = useRouter();
   const selectMenu = router.pathname;
   const onClickLogout = () => {
      if (window.confirm("Ви справді хочете вийти?")) {
         Api.auth.logout();
         location.href = "/";
      }
   };
   return (
      <Layout.Header className={styles.root}>
         <div className={styles.headerInner}>
            <div className={styles.headerLeft}>
               <h2>
                  <GlobalOutlined />
                  eLearnSky
               </h2>
               <Menu
                  className={styles.topMenu}
                  theme="dark"
                  mode="horizontal"
                  defaultSelectedKeys={[selectMenu]}
                  onSelect={({ key }) => router.push(key)}
                  items={[
                     { key: "/dashboard", label: "Головна" },
                     { key: "/dashboard/profile", label: "Профіль" },
                  ]}></Menu>
            </div>
            <div className={styles.headerRight}>
               <Popover trigger="click" content={
                  <Button onClick={onClickLogout} type="primary" danger>Вийти</Button>
               }>
                  <Avatar>Y</Avatar>
               </Popover>
            </div>
         </div></Layout.Header)}
```

graduate-project-front\component\UploadButton\index.tsx

```
import React from "react";
import styles from "../../styles/Home.module.scss"
import { Button, Upload, UploadFile, notification } from "antd";
import { CloudUploadOutlined } from "@ant-design/icons";
import * as Api from "../../api";
```

```
export const UploadBtn: React.FC = () => {
   const [fileList, setFileList] = React.useState<UploadFile[]>([]);
   const onUploadSuccess = async (options) => {
      try {
         await Api.files.uploadFile(options);
         setFileList([]);
         window.location.reload()
         window.location.reload();
      } catch (err) {
         notification.error({
            message: "Помилка!",
            description: "Неможливо завантажити файл",
            duration: 2,
         });
      }
   };
   return (
      <Upload
         onChange={({ fileList }) => setFileList(fileList)}
         customRequest={onUploadSuccess}
         fileList={fileList}
         className={styles.upload}>
         <Button type="primary" icon={<CloudUploadOutlined />}>Завантажити файл</Button>
      </Upload>
   )
}
```

graduate-project-front\core\axios.ts

```
import axios from "axios";
import { parseCookies } from "nookies";
axios.defaults.baseURL = "http://localhost:7777";
axios.interceptors.request.use((config) => {
  if (typeof window !== "undefined") {
    const { _token } = parseCookies();
    config.headers.Authorization = "Bearer " + _token;
  }  return config;}); export default axios;
```

graduate-project-front\layout\Layout.tsx

```
import Head from "next/head";
import { Header } from "../component/Header";
import React from "react";
import styles from "../styles/Home.module.scss";
interface LayoutProps {
   title: string;
}
export const Layout: React.FC<React.PropsWithChildren<LayoutProps>> = ({
   title,
   children,
}) => {
```

```
  return (
    <>
      <Head>
        <title>{title}</title>
      </Head>
      <main>
        <Header />
        <div className={styles.main}>
          <div className={styles.layout}>{children}</div>
        </div>
      </main>
    </>
  );
};
```

graduate-project-front\modules\Files.tsx

```
import React from "react";
import { FileItem } from "../api/dto/files.dto";
import { Actions } from "../component/Actions";
import { FileList, FileSelectType } from "../component/FileList";
import { Empty } from "antd";
import * as Api from "../api";
interface FilesProps {
 items: FileItem[];
 withActions?: boolean;
}
export const Files: React.FC<FilesProps> = ({ items, withActions }) => {
 const [files, setFiles] = React.useState(items || []);
 const [selectedIds, setSelectedIds] = React.useState<number[]>([]);
 const onFileSelect = (id: number, type: FileSelectType) => {
  if (type === "select") {
   setSelectedIds((prev) => [...prev, id]);
  } else {
   setSelectedIds((prev) => prev.filter((_id) => _id !== id));
  }
 };
 const onClickRemove = () => {
  setSelectedIds([]);
  setFiles((prev) => prev.filter((file) => !selectedIds.includes(file.id)));
  Api.files.remove(selectedIds);
 };
 const onClickShare = async () => {
  if (selectedIds) {
   const fileId = selectedIds[0];
   const fileItems = await Api.files.get([fileId]);
   if (fileItems.length > 0) {
    const fileItem = fileItems[0];
    const response = await fetch(`http://localhost:7777/files/${fileId}/download`);
    const blob = await response.blob();
```

```
      const url = window.URL.createObjectURL(blob);
      const a = document.createElement('a');
      a.href = url;
      a.download = fileItem.filename; filename одного объекта FileItem
      document.body.appendChild(a);
      a.click();
      document.body.removeChild(a);
    }
  }
 };
 return (
  <div>
   {files.length ? (
    <>
     {withActions && (
      <Actions
       onClickRemove={onClickRemove}
       onClickShare={onClickShare}
       isActive={selectedIds.length > 0}
      />
     )}
     <FileList items={files} onFileSelect={onFileSelect} />
    </>
   ) : (
    <Empty className="empty-block" description="Список файлов пуст" />
   )}
  </div>
 );
};
```

graduate-project-front\pages\dashboard\index.tsx

```
import { GetServerSidePropsContext, NextPage } from "next";
import { checkAuth } from "../../utils/checkAuth"
import { Header } from "../../component/Header";
import React from "react";
import { FileOutlined, FileImageOutlined, DeleteOutlined } from "@ant-design/icons";
import { Layout } from "../../layout/Layout";
import styles from "../../styles/Home.module.scss"
import { Button, Menu } from "antd";
import { useRouter } from "next/router";
import { UploadBtn } from "../../component/UploadButton";
import * as Api from "../../api";
import { FileItem } from "../../api/dto/files.dto";
import { FileList } from "../../component/FileList";
import { HomeLayout } from "../../layout/HomeLayout";
import { Actions } from "../../component/Actions";
import { Files } from "../../modules/Files";
interface Props {
```

```
    items: FileItem[];
}
const PDashboard: NextPage<Props> = ({ items }) => {
    const router = useRouter();
    const selectMenu = router.pathname;
    return (
        <HomeLayout>
            <Files items={items} withActions/>
        </HomeLayout>
    )
}
PDashboard.getLayout = (page: React.ReactNode) => {
    return <Layout title="Dashboard">{page}</Layout>;
};
export const getServerSideProps = async (ctx: GetServerSidePropsContext) => {
    const authProps = await checkAuth(ctx);
    if ("redirect" in authProps) {
        return authProps;
    }
    try {
    const items = await Api.files.getAll();
        return {
            props: {
                items,
            },
        };
    } catch (err) {
        console.log(err);
        return {
            props: {},
        }
    }
}
export default PDashboard;
```

graduate-project-front\pages\dashboard\auth.tsx

```
import { NextPage } from "next";
import Head from "next/head";
import { LoginForm } from "../../component/auth/LoginForm";
import { Tabs } from "antd";
import { RegisterForm } from "../../component/auth/RegisterForm";
const AuthPage: NextPage = () => {
    return (
        <>
            <Head>
                <title>Auth</title>
            </Head>
            <main style={{ width: 400, margin: '50px auto' }}>
                <Tabs
```

```
            items={[
              {
                label: 'Увійти',
                key: '1',
                children: <LoginForm />
              },
              {
                label: 'Реєстрація',
                key: '2',
                children: <RegisterForm />
              }
            ]}
          />
        </main>
      </>  )}export default AuthPage;
```

graduate-project-front\utils\checkAuth.ts

```
import { GetServerSidePropsContext } from "next";
import nookies from "nookies";
import axios from "../core/axios";
import * as Api from "../api";
export const checkAuth = async (ctx: GetServerSidePropsContext) => {
  const { _token } = nookies.get(ctx);
  axios.defaults.headers.Authorization = "Bearer " + _token;
  try {
    await Api.auth.getMe();
    return {
      props: {},
    };
  } catch (err) {
    return {
      redirect: {
        destination: "/dashboard/auth",
        permanent: false,
      },
    };
  }
};
```

Додаток Б

ПЕРЕЛІК ДОКУМЕНТІВ НА ОПТИЧНОМУ НОСІЇ

Ім’я файла	**Опис**
Пояснювальні документи	
Диплом.doc	Пояснювальна записка кваліфікаційної роботи. Документ Word.
Диплом.pdf	Пояснювальна записка кваліфікаційної роботи в форматі PDF
Програма	
Program.rar	Архів. Містить коди програми і откомпільовану програму
Презентація	
Презентація.ppt	Презентація до магістерської роботи