КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра теорій та технологій програмування

Кваліфікаційна робота
на здобуття ступеня бакалавра

за спеціальністю 122 Комп'ютерні науки
на тему:

РОЗРОБКА ВЕБ-ОРІЄНТОВАНОГО СЕРВІСУ ЗБЕРІГАННЯ ФАЙЛІВ

Виконав студент 4-го курсу

Максим ОШИЙКО _____
(підпис)

Науковий керівник
доцент, кандидат технічних наук
Олексій ТКАЧЕНКО _____
(підпис)

Засвідчую, що в цій курсові роботі немає запозичень
з праць інших авторів без відповідних
посилань.

Студент ___
(підпис)

Роботу розглянуто й допущено до захисту
на засіданні кафедри теорій та технологій програмування
«5» червня 2023р.,
протокол № 18
Завідувач кафедри
Микола НІКІТЧЕНКО _____
(підпис)

Київ – 2023

РЕФЕРАТ

Обсяг роботи 55 сторінки, 33 ілюстрацій, 10 таблиць, 23 джерел посилань.

ВЕБ-СЕРВІС, СХОВИЩЕ, ЗБЕРЕЖЕННЯ ДАНИХ, ОБМІН ІНФОРМАЦІЄЮ, СПІЛЬНИЙ ДОСТУП ДО ФАЙЛІВ, ПРИКЛАДНИЙ ПРОГРАМНИЙ ІНТЕРФЕЙС, ASP.NET CORE, MS SQL SERVER, REACT JS.

Предметом роботи є веб-сервіси для збереження файлів з функціоналом, що дозволяє користувачам зберігати, створювати, редагувати, копіювати, переміщати та видаляти файли та папки, а також давати спільний доступ до них. Об’єктом роботи є файлові сховища.

Метою роботи є розробка веб-сервісу для збереження файлів з використанням сучасних технологій та інструментів. Основними завданнями роботи є:

1. Розробка веб-інтерфейсу, який дозволяє користувачам зручно зберігати, керувати та спільно використовувати свої файли та папки.
2. Використання сучасних технологій для забезпечення безпеки даних користувачів та швидкої роботи сервісу.
3. Розробка функціоналу, що дозволяє користувачам створювати та редагувати текстові файли всередині сервісу.
4. Розробка контролю доступу до файлів та папок, що дозволяє давати спільний доступ до своїх файлів іншим користувачам.

Результатом роботи є функціональний веб-сервіс, який включає в себе наступні функції:

1. Реєстрація користувачів у системі.
2. Збереження файлів та папок в сховищі сервісу.
3. Створення та видалення папок.
4. Завантаження файлів та папок зі сховища.
5. Редагування текстових файлів всередині системи.
6. Копіювання, переміщення та видалення файлів.
7. Спільний доступ до файлів та папок.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

FTP – File Transfer Protocol, протокол передавання файлів;

AES – Advanced Encryption Standard; розширений стандарт шифрування;

API – Application Programming Interface; інтерфейс прикладного програмування;

CQRS – Command and Query Responsibility Segregation; розподіл відповідальності на команди та запити;

CSS – Cascading Style Sheets; каскадні таблиці стилів;

DOM – Document Object Model; об'єктна модель документа;

Drag-and-drop – це жест вказівного пристрою, під час якого користувач вибирає об’єкт, «схоплюючи» його та перетягуючи в інше місце або на інший об’єкт.

EF – Entity Framework; інструмент для об’єктно-реляційного відображенн;

ER – Entity–relationship; відношення між сутностями;

HTML – HyperText Markup Language; мова розмітки гіпертексту;

HTTP – HyperText Transfer Protocol; протокол передачі гіпертексту;

IDE – Integrated development environment; інтегроване середовище розробки;

ORM – Object-Relational Mapping; об'єктно-реляційне відображення;

SSMS – SQL Server Management Studio; утиліта для конфігурації, управління та адміністрування всіх компонентів Microsoft SQL Server;

СУБД – система управління базами даних;

ВСТУП

Оцінка сучасного стану об’єкта дослідження або розробки. За останні роки спостерігається значне зростання потреби в збереженні файлів в Інтернеті. Користувачі все більше залежать від зберігання та обміну файлами в хмарних сервісах, що вимагає розробки ефективних та безпечних рішень. На сьогоднішній день існує декілька популярних веб-сервісів для збереження файлів, таких як Google Drive, Dropbox, OneDrive та інші. Ці сервіси забезпечують збереження даних користувачів в хмарі та можливість роботи з цими даними з будь-якого пристрою, що підключений до Інтернету.

Актуальність роботи та підстави для її виконання. З огляду на ріст популярності хмарних технологій та необхідність доступу до файлів з різних пристроїв, досить актуальною є розробка веб-сервісу для збереження файлів. Актуальність роботи базується на зростаючій кількості даних, які потрібно зберігати та обробляти. Компанії зберігають все більше і більше даних в Інтернеті, і роблять це в тому числі й для забезпечення доступу до них з різних місць та пристроїв. Також, велика кількість компаній шукають способи замінити старі методи зберігання файлів (наприклад, використання локальних серверів) більш сучасними, ефективнішими та безпечними.

Мета й завдання роботи. Метою роботи є розробка веб-сервісу для збереження файлів з можливістю створення та редагування текстових файлів, керуванням папками та файлами, наданням спільного доступу. Такий веб-сервіс буде дозволяти користувачам зберігати та організовувати свої файли в хмарному сховищі та отримувати до них доступ з будь-якого місця з підключенням до Інтернету. Для досягнення поставленої мети треба вирішити такі завдання:

1. Дослідити технології веб-сервісів для зберігання файлів.
2. Зробити огляд існуючих систем зберігання файлів, визначити їхні переваги та недоліки.
3. Розробити сервіс-орієнтовану схему зберігання файлів.
4. Розробити серверну та клієнтську частини.

5. Протестувати систему.

Об’єкт, методи й засоби дослідження або розроблення. Об’єктом дослідження є файлові сховища. Предметом дослідження є веб-сервіси для зберігання файлів, що дозволяють користувачам зберігати, створювати, редагувати, копіювати, переміщати та видаляти файли та папки, а також надавати доступ до них. Для дослідження та розроблення використовувалися метод моделювання та об’єктно орієнтований підхід. Розробці системи передувало визначення цілей та потреб користувачів. Перед тим, як розпочати розробку, потрібно визначити, що саме має бути розроблено та які потреби користувачів має задовольняти розроблюваний продукт.

Для вибору найбільш підходящих засобів та технологій для розробки був проведений попередній аналіз популярних та сучасних фреймворків та бібліотек, які змогли з пришвидшити розробку проекту та зробити систему надійною.

Можливі сфери застосування. Можливі сфери застосування веб-сервісу для зберігання файлів безліч, від особистого використання до бізнесу та державних організацій. Користувачі можуть використовувати цей сервіс для збереження фотографій, відео та документів, з доступом до них з будь-якого місця і пристрою. Бізнес-клієнти можуть використовувати його для зберігання і обміну даними з колегами та клієнтами, а також для забезпечення резервного копіювання.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ТЕХНОЛОГІЙ РОЗРОБКИ ВЕБ-СЕРВІСУ

1.1 Історія розвитку веб-сервісів для зберігання файлів

Історія виникнення веб-сервісів для збереження файлів починається з появи перших електронних пристроїв для збереження даних. У минулому, користувачі зберігали свої файли на дискетах або жорстких дисках. FTP був одним з перших протоколів, що дозволяв передавати файли через мережу[1]. Він забезпечував зручний доступ до віддалених серверів і дозволяв керувати файлами на них. Проте з розвитком інтернету з'явилася потреба в збереженні даних в онлайн-сховищах, які можна було б відкривати з будь-якого пристрою з підключенням до мережі Інтернет.

Веб-сервіси для збереження файлів почали з'являтись на початку 2000-х років, коли компанії, такі як Dropbox, Box та Google Drive, почали надавати такі послуги. З того часу ринок сервісів збереження файлів став все більш насиченим, і сьогодні він є досить конкурентним.

Розвиток інтернет-технологій та збільшення доступності до високошвидкісного Інтернету також допомогли виростити популярність цих сервісів. Крім того, зростання популярності мобільних пристроїв, таких як смартфони та планшети, дало змогу зберігати та отримувати файли з будь-якого місця та в будь-який час.

Ідея веб-сервісів для збереження файлів була сприйнята дуже позитивно користувачами, адже вони дозволяли зберігати файли в безпечному місці, віддаленому від ризиків втрати даних через віруси або поломку техніки. Також це дозволяло легко ділитися файлами з іншими користувачами, без необхідності надсилати їх по електронній пошті або іншими способами. Тому, нині зберігання файлів в Інтернеті є домінуючим.

Основними задачами, які веб-сервіси для збереження файлів мали вирішувати, є:

1. Зберігання файлів в безпечному і надійному середовищі: веб-сервіси забезпечували зберігання файлів на серверах, що мають високі рівні надійності та безпеки.
2. Доступ до файлів з будь-якого місця та пристрою: завдяки веб-сервісам користувачі могли зберігати свої файли в хмарі та мати доступ до них з будь-якого пристрою з Інтернет-з'єднанням.
3. Синхронізація даних між пристроями: веб-сервіси забезпечували синхронізацію даних між різними пристроями, що дозволяло користувачам мати актуальну версію своїх файлів на кожному пристрої.
4. Підтримка спільної роботи з файлами: веб-сервіси надавали можливість спільної роботи над файлами, що дозволяло користувачам спільно редагувати, коментувати та обговорювати файли в режимі реального часу.

1.2 Існуючі рішення

Основними лідерами веб-сервісів для збереження файлів є Dropbox, Google Drive та Microsoft OneDrive[2].

Dropbox - це сервіс зберігання і обміну файлами на хмарній платформі, який дозволяє користувачам зберігати, отримувати доступ та обмінюватися файлами з будь-якого пристрою з підключенням до Інтернету. Запущений у 2008 році, він швидко став однією з найпопулярніших платформ зберігання в хмарі, і тепер має більше 600 мільйонів зареєстрованих користувачів.

Однією з ключових особливостей Dropbox є його простота використання. Після реєстрації на сервісі користувачі можуть завантажити додаток Dropbox для робочого столу або мобільного пристрою, який автоматично синхронізує їх файли на всіх пристроях. Це означає, що користувачі можуть отримати доступ до своїх файлів з будь-якого місця, чи то вдома, в офісі, чи в дорозі.

Ще однією важливою функцією Dropbox є його безпека. Платформа використовує шифрування AES з довжиною ключа 256 біт для захисту файлів користувачів під час передачі та зберігання. Dropbox також пропонує

різноманітні функції безпеки для бізнесу, такі як продвинуте управління користувачами та інструменти управління даними.

Крім зберігання та обміну файлами, Dropbox також пропонує функції співпраці. Користувачі можуть ділитися папками з іншими користувачами і встановлювати дозволи, щоб керувати тим, хто може переглядати та редагувати їх файли. Dropbox також інтегрується з різноманітними інструментами та сервісами, такими як Microsoft Office, Slack та Zoom, що робить його популярним вибором для бізнесу та віддалених команд.

Загалом, Dropbox є надійною та користувачів-дружньою платформою зберігання в хмарі, яка надає безпечне зберігання, обмін та функції співпраці. Його легкість використання та потужні функції безпеки зробили його популярним вибором для окремих користувачів та бізнесів[3].

Google Drive - це хмарна послуга для зберігання та спільного використання файлів, розроблена компанією Google. Запущена в 2012 році, вона швидко стала однією з найпопулярніших платформ зберігання в хмарі, налічуючи понад 2 мільярди активних користувачів.

Одна з головних переваг Google Drive - це інтеграція з іншими сервісами Google, такими як Google Docs, Sheets та Slides. Це дозволяє користувачам створювати, редагувати та співпрацювати над документами, електронними таблицями та презентаціями прямо в платформі, не потребуючи завантаження або відвантаження файлів. Крім того, Google Drive підтримує широкий спектр форматів файлів, що робить його універсальним інструментом для всіх видів даних.

Google Drive також пропонує потужні засоби захисту файлів користувачів. Він пропонує двофакторну автентифікацію та аудит-логування для підвищення безпеки. Крім того, Google Drive відповідає різноманітним стандартам та регулятивним вимогам, таким як HIPAA та GDPR, що робить його підходящим для бізнесу в різних галузях.

Ще однією важливою функцією Google Drive є можливості співпраці. Користувачі можуть ділитися файлами та папками з іншими, встановлювати

дозволи для контролю доступу до перегляду, редагування та коментування файлів. Крім того, користувачі можуть спільно працювати в режимі реального часу над документами, таблицями та презентаціями, що дозволяє ефективно та безперебійно працювати в команді.

Google Drive також пропонує ряд планів зберігання, які задовольнять різні потреби, від безкоштовних планів на 15 ГБ для окремих користувачів до бізнес-планів з необмеженим зберіганням та розширеними функціями безпеки. Крім того, він доступний на кількох пристроях, включаючи настільні комп'ютери, ноутбуки, смартфони та планшети, що забезпечує можливість доступу до файлів в будь-який час та в будь-якому місці з доступом до Інтернету[4].

Загалом, Google Drive є потужною та універсальною платформою зберігання в хмарі, яка надає надійні функції зберігання файлів та співпраці. Його інтеграція з іншими сервісами Google, надійні функції безпеки та різноманітні плани зберігання зробили його популярним вибором для користувачів.

OneDrive є хмарним сховищем, розробленим компанією Microsoft, який надає можливість користувачам зберігати, редагувати та ділитися своїми файлами в хмарі. Однією з головних переваг OneDrive є його безпека та конфіденційність даних. Платформа використовує технологію шифрування файлів під час їх транспортування та збереження на сервері, що робить надійним і безпечним зберігання даних[5].

OneDrive також забезпечує інтеграцію з іншими продуктами Microsoft, такими як Microsoft Office, що дає можливість прямо з платформи OneDrive створювати, редагувати та зберігати документи в різних форматах, включаючи Word, Excel, PowerPoint тощо. Крім того, OneDrive інтегрується зі Skype, що дає можливість викликати інших користувачів OneDrive та спільно працювати з документами, що значно полегшує комунікацію між колегами.

OneDrive також пропонує функцію Personal Vault, яка надає додатковий рівень безпеки для конфіденційних файлів. Ця функція дозволяє

користувачам зберігати свої найважливіші файли в захищеній папці, яка вимагає двохфакторної аутентифікації або сильного методу аутентифікації, такого як відбиток пальця або розпізнавання обличчя для доступу. Крім того, функція Files On-Demand дозволяє користувачам отримати доступ до всіх своїх файлів, не завантажуючи їх на пристрій, що зберігає важливий простір для зберігання.

Кожен з сервісів має свої переваги (рисунок 1) над іншими і користувачам слід обирати між цими ресурсами в залежності від потреб.

The Comparison of File Sharing			
Cloud Storage / Aspects	Dropbox	Google Drive	OneDrive
Share by Creating a Link	✓	✓	✓
Share by Emailing	✓	✓	✓
Permit Files Viewed or Edited	✓	✓	✓
Set Passwords and Expiration Dates	✓	✗	✓
Review Shared Files	✓	✓	✓
Review shared Files Activities	✗	✗	✓

Рисунок 1 – Порівняння існуючих рішень[6]

1.3 Технології розробки

1.3.1 Інтегроване середовище розробки Microsoft Visual Studio

Microsoft Visual Studio - це інтегроване середовище розробки (IDE) з великим переліком функцій (рисунок 2), яке створене компанією Microsoft. Воно дозволяє програмістам розробляти програми для різних платформ, таких як Windows, Android, iOS, та веб-додатки.

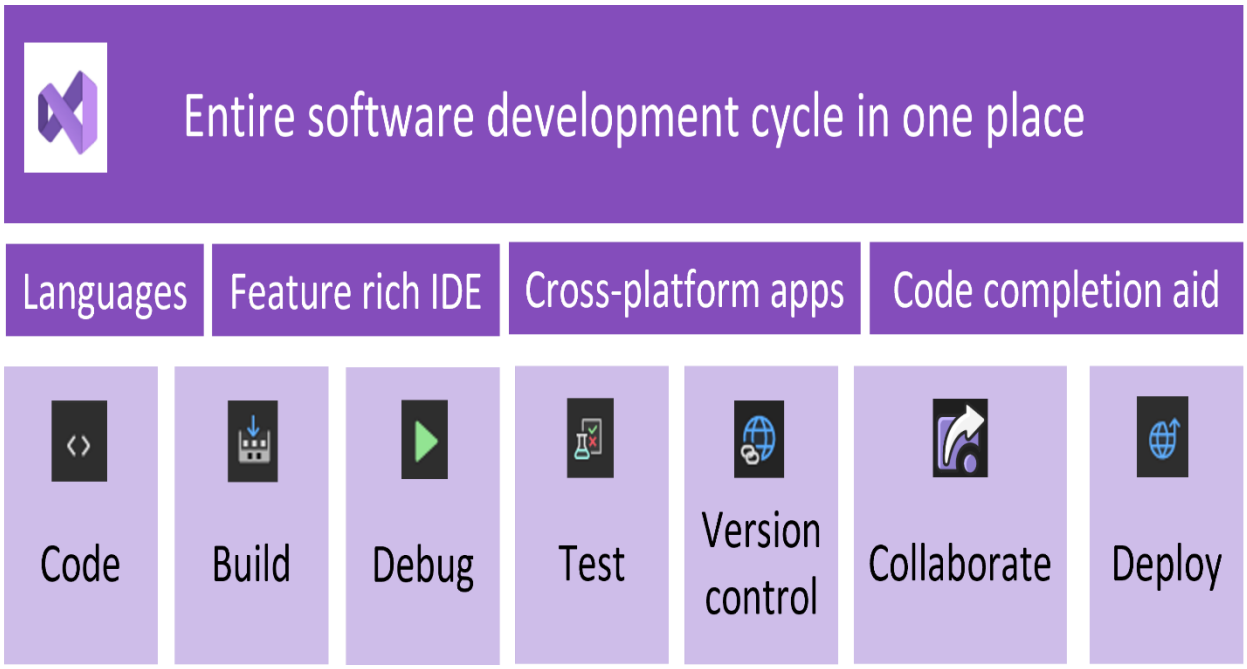

Рисунок 2 – Функції Microsoft Visual Studio[7]

Основна функція Visual Studio - це створення програмних продуктів з високою якістю. Воно має усі інструменти, необхідні для розробки програмного забезпечення, від написання коду до відлагодження, тестування та розгортання. З його допомогою розробники можуть працювати з різними мовами програмування, такими як C++, C#, Visual Basic та інші.

Visual Studio має ряд корисних функцій, таких як IntelliSense, який допомагає програмістам писати код швидше та ефективніше, та CodeLens, який надає користувачам контекстну інформацію про їх код.

Однією з основних переваг Visual Studio є його розширюваність. Це означає, що користувачі можуть додавати до IDE свої власні розширення, що дозволяє робити його більш функціональним та відповідним до вимог користувачів.

Окремими компонентами Microsoft Visual Studio є редактори коду, візуальні дизайнери, налагоджувачі та пакети розширень, які дозволяють збільшити функціональність редактора та прискорити розробку.

Крім того, Visual Studio має різноманітні інструменти, які дозволяють проводити тестування та аналіз коду, що дозволяє виявляти та виправляти

помилки на ранніх етапах розробки. Це дозволяє розробникам зберігати якість коду[7].

Узагалі, Microsoft Visual Studio є потужним інструментом для розробки програмного забезпечення, який надає розробникам можливість працювати з багатьма мовами програмування та різними платформами, та дозволяє покращити продуктивність та якість розробки завдяки вбудованим інструментам.

1.3.2 Серверна частина

Для розробки серверної частини веб-сервісу використовувалися мова програмування C#, фреймворк для створення веб-додатків ASP.NET Core та база даних Microsoft SQL Server.

Мова програмування C#

C# є сучасною об'єктно-орієнтованою мовою програмування, яка була розроблена компанією Microsoft на початку 2000-х років як частина її ініціативи .NET. Мова широко використовується для розробки Windows-додатків, ігор, веб-додатків та мобільних додатків, серед іншого. C# розроблено з метою забезпечення простоти використання, потужності та масштабованості, а також пропонує широкий спектр функцій, що робить його однією з найбільш популярних мов програмування, які використовуються сьогодні[8].

Однією з ключових переваг C# є його здатність до керування управлінням пам'яті. C# є мовою з автоматичним збором сміття, що означає, що розробники не потребують вручну виділяти та звільняти пам'ять, оскільки система виконання це робить автоматично. Це допомагає знизити ризик виникнення помилок, пов'язаних з пам'яттю, таких як переповнення буфера, витік пам'яті та залишкові покажчики[9].

Ще однією перевагою C# є його підтримка сучасних парадигм програмування, таких як об'єктно-орієнтоване програмування та функціональне програмування. C# є мовою зі статичною типізацією, що означає, що кожна змінна має певний тип, який не можна змінити в ході

виконання програми. Це дозволяє компілятору виявити багато помилок на етапі компіляції, а не в ході виконання програми, що допомагає уникнути помилок та забезпечити надійність коду.

C# також надає широкий набір функцій, що дозволяє розробляти масштабні застосунки. Наприклад, C# підтримує інтерфейси, які дозволяють розробникам визначати контракт, який клас повинен реалізувати. Це сприяє перевикористанню коду та спрощує підтримку великих кодових баз. C# також підтримує властивості, які надають можливість інкапсулювати внутрішній стан об'єкту, що спрощує його управління та оновлення.

C# також має багатий набір бібліотек та фреймворків, які полегшують розробку додатків. Наприклад, фреймворк .NET містить широкий набір бібліотек для роботи з інтерфейсами користувача, базами даних, мережами та іншими. Крім того, є багато сторонніх бібліотек та фреймворків, які можна легко інтегрувати в проекти C#, таких як популярний відкритий фреймворк ASP.NET для розробки веб-додатків.

Ще однією перевагою C# є його сильна інтеграція з Microsoft Visual Studio, який є потужним інтегрованим середовищем розробки (IDE) для створення та налагодження додатків[10].

Отже, C# - це універсальна та потужна мова програмування, яка стала популярним вибором для розробки різноманітних додатків. Її підтримка сучасних парадигм програмування, сильна система управління пам'яті та продуктивність роблять її чудовим вибором для великих додатків. Її інтеграція з фреймворком .NET та IDE Visual Studio, а також активна спільнота роблять розробку, тестування та розгортання додатків більш простими. Незалежно від того, чи ви розробляєте додатки для Windows, веб-додатки або мобільні додатки, C# надає надійну та гнучку платформу для створення високоякісного програмного забезпечення.

Фреймворк ASP.NET Core

ASP.NET Core є безкоштовним веб-фреймворком з відкритим кодом, який дозволяє розробникам будувати сучасні, високопродуктивні веб-додатки

для різноманітних платформ, таких як Windows, Linux та macOS. Це наступне покоління фреймворку ASP.NET від Microsoft, і його було розроблено з урахуванням модульності, кросплатформеності та готовності до хмарних технологій[11].

Однією з ключових переваг ASP.NET Core є його можливість працювати на різних платформах. Це означає, що розробники можуть будувати та розгортати веб-додатки на будь-якій операційній системі, не будучи пов'язаними з певною платформою або технологічним стеком. Це спрощує розробку додатків, які можуть використовуватися ширшою аудиторією та працювати в різних середовищах, від локальних серверів до хмари.

Ще однією перевагою ASP.NET Core є його модульна архітектура. Фреймворк був розроблений з урахуванням легкості та гнучкості, з модульною архітектурою, яка дозволяє використовувати лише необхідні компоненти. Це дозволяє будувати додатки, які є швидкими та ефективними, без зайвих накладних витрат або зайвого коду.

Крім цих функцій, ASP.NET Core містить широкий спектр бібліотек і інструментів, що дозволяють працювати з базами даних, API та іншими веб-технологіями. Наприклад, у фреймворку є вбудована підтримка Entity Framework Core, який був використаний для розробки системи[12].

Загалом, ASP.NET Core є потужним, гнучким і кросплатформним веб-фреймворком, який добре підходить для створення сучасних веб-додатків з високою продуктивністю. Його модульна архітектура, підтримка кросплатформовості та широкий спектр функцій роблять його популярним серед розробників, а інтеграція з екосистемою .NET та Visual Studio полегшує розробку, тестування та розгортання додатків.

Об’єктно-реляційний маппер Entity Framework

Entity Framework є потужним інструментом для роботи з базами даних у програмних додатках. Це об'єктно-реляційний маппер (ORM), який

дозволяє розробникам працювати з даними у вигляді об'єктів, замість безпосередньої роботи з SQL-запитами та таблицями бази даних.

Entity Framework надає зручну модель програмування, що базується на об'єктах, властивостях і зв'язках між ними. Розробники можуть описати структуру бази даних у вигляді класів (сутностей) і використовувати їх для створення, читання, оновлення та видалення даних. Він автоматично перетворює ці операції на відповідні SQL-запити, що дозволяє зосередитись на бізнес-логіці додатку[13].

Entity Framework підтримує різні типи баз даних, включаючи Microsoft SQL Server, MySQL, PostgreSQL та багато інших. Він надає можливість легко створювати, модифікувати та синхронізувати схеми баз даних за допомогою міграцій. EF також надає підтримку для виконання складних запитів, включаючи фільтрацію, сортування, з'єднання та агрегацію даних[14].

Іншою корисною функцією Entity Framework є його можливість кешування даних, яка допомагає покращити продуктивність додатка шляхом зменшення кількості запитів до бази даних.

Загалом, Entity Framework є потужним інструментом для роботи з базами даних, який спрощує розробку додатків і забезпечує зручний спосіб взаємодії з даними. Використання EF дозволяє розробникам швидко і ефективно створювати додатки з можливістю простої масштабованості і підтримки різних типів баз даних.

Реляційна база даних Microsoft SQL Server

Microsoft SQL Server є однією з найпопулярніших систем управління базами даних (СУБД) в світі. Розроблена корпорацією Microsoft, вона надає розширені можливості для зберігання, управління і обробки даних.

Microsoft SQL Server - це потужна реляційна база даних, яка надає широкий спектр можливостей для зберігання та обробки даних. Вона має вбудовану підтримку інших типів зберігання даних та обробки, таких як XML-дані, просторові дані та повнотекстовий пошук, що робить її універсальною платформою для різноманітних даних-інтенсивних додатків.

SQL Server також включає низку інструментів для управління та моніторингу продуктивності баз даних. Ці інструменти дозволяють адміністраторам баз даних моніторити активність системи, відстежувати продуктивність запитів та оптимізовувати налаштування системи для покращення загальної продуктивності[15].

Також, для керування та адміністрування базою даних було використано інтрумент Microsoft SQL Server Management Studio. Він дозволяє створювати, змінювати та керувати об'єктами бази даних, такими як таблиці, процедури, функції, індекси та інше.

Інтерфейс користувача Microsoft SQL Server Management Studio є досить зручним і легким у використанні, що дозволяє ефективно використовувати всі його можливості. Крім того, з його допомогою можна керувати декількома базами даних одночасно, що забезпечує ефективність та продуктивність[16].

SSMS надає користувачам можливість виконувати запити SQL, редагувати дані, створювати звіти та керувати конфігурацією SQL Server. Крім того, він має вбудовану підтримку для інструментів адміністрування, таких як SQL Server Agent, Database Engine Tuning Advisor та SQL Server Configuration Manager.

Загалом, Microsoft SQL Server - це потужна, надійна та універсальна платформа для управління та обробки даних. Її багаті можливості, надійність та широкий спектр інструментів управління роблять її популярним вибором серед розробників.

1.3.3 Клієнтська частина

Для розробки веб-інтерфейсу використовувалися мова розмітки документів HTML, мова стилю сторінок CSS та мова програмування JavaScript, фреймворк Bootstrap та JavaScript бібліотека React.

Фреймворк Bootstrap

Bootstrap є одним з найпопулярніших фреймворків для розробки фронтенду веб-сайтів і додатків. Випущений в 2011 році з боку Twitter, Bootstrap надав зручні та прості засоби для швидкої інтернет-розробки[17].

Основна мета Bootstrap полягає в тому, щоб зменшити час та зусилля, необхідні для розробки веб-сторінок і додатків, забезпечуючи багато готових компонентів та різноманітних інструментів для розробки інтерфейсів. Таким чином, це дозволило зменшити час на розробку інтерфейсу і зосередитись на розробці складної бізнес логіки на серверній частині веб-сервісу.

Bootstrap містить широкий набір CSS класів і JavaScript компонентів, які можуть бути використані для розробки будь-якої веб-сторінки з базової до складної. Бібліотека містить готові компоненти, такі як кнопки, форми, меню, картки, модальні вікна та багато іншого (рисунок 3).

Рисунок 3 – Готовий набір компонент фреймворку Bootstrap[18]

Ще одна велика перевага Bootstrap полягає в його адаптивності, що означає, що веб-сайт або додаток, розроблений з використанням цієї бібліотеки, виглядатиме добре на будь-якому розмірі екрану. Bootstrap містить класи, які можуть змінювати вигляд компонентів в залежності від розміру екрану, що дозволяє створювати респонсивні інтерфейси. Це є важлимим фактором для веб-сервісу для зберігання файлів, оскільки дозволить зручно користуватися інтерфейсом з телефонів та планшетів.

Загалом, Bootstrap є потужним інструментом для швидкої інтернет-розробки, що надає розробникам зручні інструменти, готові компоненти і можливості адаптивності.

Бібліотека React

React (або React.js) - це бібліотека JavaScript для створення користувацьких інтерфейсів веб-додатків. Розробка бібліотеки React була почата Facebook, а перший публічний реліз відбувся в 2013 році[19].

Головними принципами React є компонентний підхід та декларативний підхід до опису інтерфейсу користувача. Компоненти в React - це незалежні, перевикористовувані елементи інтерфейсу, які складаються з HTML-подібного коду, стилів та логіки поведінки. Це дозволяє писати більш організований, підтримуваний та легко змінюваний код.

React дозволяє розробникам працювати з віртуальним DOM (Document Object Model), який є абстрактною копією реального DOM. Це дозволяє React здійснювати швидкі перерендеринги інтерфейсу користувача, зменшуючи кількість маніпуляцій з реальним DOM і покращуючи продуктивність додатка.

React також має вбудовану підтримку стану компонентів, що дозволяє розробникам легко керувати даними та їхнім відображенням на екрані. Крім того, в React є багато сторонніх бібліотек і плагінів, які розширюють його можливості та дозволяють створювати різноманітні інтерактивні інтерфейси.

React використовують для створення високопродуктивних веб-додатків, що відрізняються відповідністю до найсучасніших вимог до користувацького досвіду та можливостями розширення.

РОЗДІЛ 2. РОЗРОБКА СИСТЕМИ ЗБЕРІГАННЯ ФАЙЛІВ

Перед початком розробки були сформульовані вимоги до проекту. Для цього були створені сценарії використання, які описують поведінку системи на зовнішні запити.

2.1 Основні сценарії використання

Таблиця 1 – Реєстрація користувача

Дійові особи	Користувач, Система.
Мета	Користувач: зареєструватися в системі. Система: створити новий обліковий запис.
Успішний сценарій: 1. Користувач запускає систему. Система відкриває форму реєстрації, пропонує ввести логін та пароль. 2. Користувач вводить email, логін та пароль. 3. Система перевіряє на валідність email, логін та пароль. 4. Система створює новий обліковий запис. 5. На пошту вказану користувачем надходить лист із підтвердженням реєстрації. 6. Система видає користувачеві повідомлення щодо успішної реєстрації.	
Результат	Користувач успішно зареєструвався і може авторизуватися.
Розширення:	3а – валідація не є успішною, система показує відповідне повідомлення про помилку валідації (неправильний формат email, слабкий пароль і т.д).

Таблиця 2 – Завантаження файлу в систему

Дійові особи	Користувач, Система.
Мета	Користувач: завантаження файлу або папки в систему. Система: зберегти елемент у базі даних.
Передумова	Користувач повинен бути авторизований.
Успішний сценарій: 7. Користувач завантажує файл у сховище за допомогою кнопки або	

функції drag-and-drop. Якщо завантажується папка, то така ж структура папок має утворитися в сховищі. 8. Файл буде завантажений у папку, в якій користувач перебуває в момент завантаження. 9. Після того, як файл завантажений, він повинний відобразитися у списку без перезавантаження сторінки. 10. Система видає нотифікацію, що завантаження файлу або папки успішно завершено.	
Результат	Файл або папка успішно завантажені
Розширення:	1а - якщо розмір файлу не валідний, система повідомляє користувача.

Таблиця 3 – Спільний доступ

Дійові особи	Користувач 1, Користувач 2, Система.
Мета	Користувач 1: надати доступ до файлу Користувачу 2. Користувач 2: отримати доступ до файлу. Система: надати файл Користувачу 2, якому надали доступ до файлу.
Передумова	Користувач 1 повинен бути авторизований.
Успішний сценарій: 1. Користувач 1 натискає на відповідну кнопку надання доступу. 2. Поява модального вікна, в якому Користувач 1 вводить логін Користувача 2 з яким хоче поділитися файлом. 3. Після того, як Користувач 1 підтвердив дію надання доступу, Користувачу 2 на пошту має надіслатися повідомлення, що він отримав доступ до файлу. 4. Усі користувачі, які отримали доступ, зможуть знайти цей файл у ресурсах загального доступу.	
Результат	Користувач 2 отримав доступ до файлу.

Таблиця 4 – Завантаження файлу або папки з системи

Дійові особи	Користувач, Система.
Мета	Пользователь: завантаження файлу або папки з системи. Система: надає файл для завантаження.
Передумова	Користувач повинен бути авторизований.
Успішний сценарій: 1. Користувач натискає на кнопку для завантаження файлу чи папки. 2. При завантаженні кількох файлів або папки програма створює zip архів і завантажує його з вибраним вмістом.	
Результат	Файл або папка успішно завантажені з системи.

Таблиця 5 – Створення текстових файлів та папок

Дійові особи	Користувач, Система.
Мета	Користувач: створення текстового файлу або папки. Система: зберегти елемент у базі даних.
Передумова	Користувач повинен бути авторизований.
Успішний сценарій: 1. Користувач натискає кнопку створити файл/папку. 2. Файл/папка створюється у тому місці, де користувач знаходився.	
Результат	Успішне створення текстового файлу або папки.

2.2 База даних

2.2.1 ER-модель

Наступним кроком перед розробкою було проектування схеми бази даних. Для цього було створено ER-модель (рисунок 4).

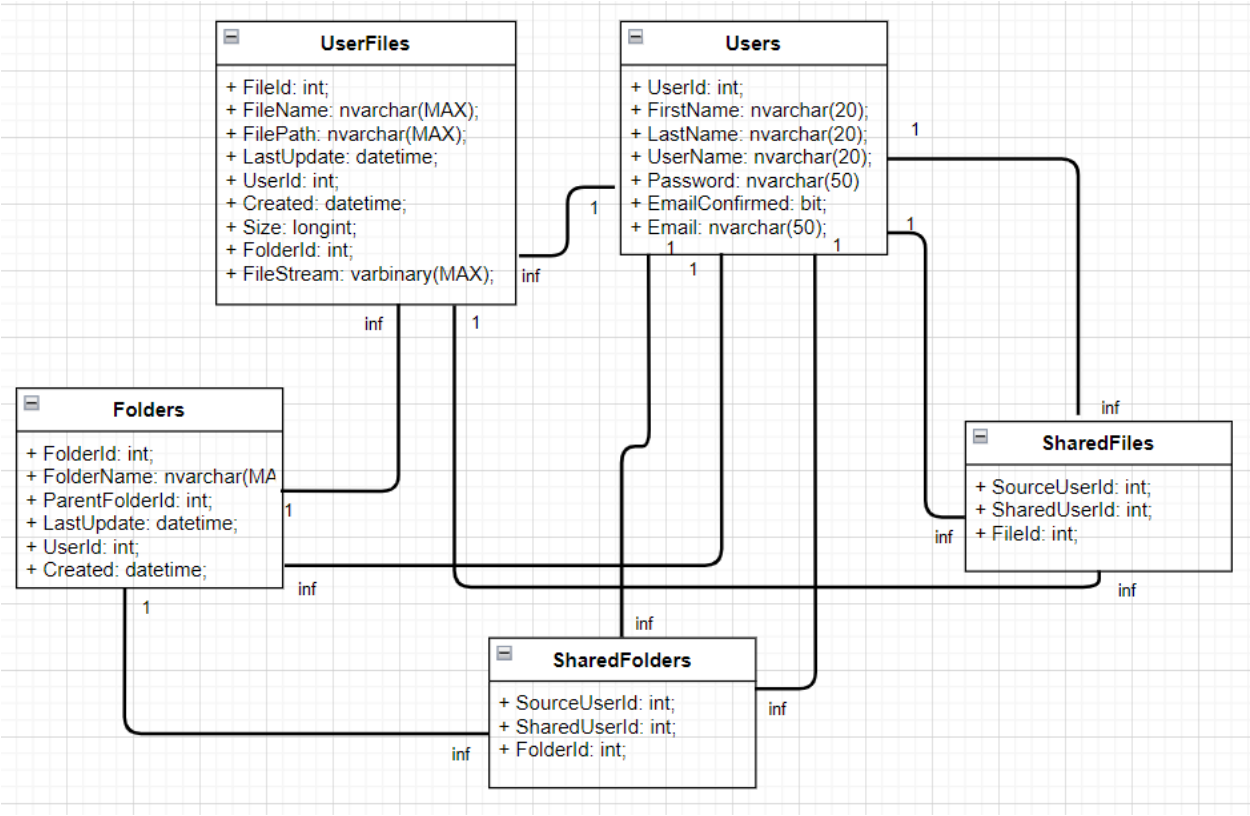

Рисунок 4 – ER-модель

ER-модель - це модель даних, яка використовується для розробки баз даних. Ця модель дозволяє зображувати різні об'єкти та взаємозв'язки між ними, що допомагає у більш точному визначенні взаємодії між об'єктами та розумінні логіки додатка.

ER-модель дозволяє візуалізувати сутності (таблиці), атрибути (колонки) та зв'язки між ними. Завдяки цьому можна чітко визначити взаємозв'язки між таблицями та визначити їхній тип. Наприклад, зв'язок один-до-багатьох, багато-до-багатьох тощо.

ER-модель допомогла зробити більш точну оцінку складності системи та розробити кращий план для її реалізації. Вона дозволила зосередитись на потребах користувачів та забезпечити ефективне використання бази даних.

2.2.2 Опис таблиць

Таблиця 6 – Таблиця UserFiles для збереження інформації про файли

FileId	Ідентифікатор файлу
FileName	Ім’я файлу
FilePath	Шлях до файлу
LastUpdate	Дата останньої модифікації файлу
UserId	Ідентифікатор користувача, якому належить файл
Created	Дата створення файлу
Size	Розмір файлу в байтах
FolderId	Ідентифікатор папки, в якій знаходиться файл
FileStream	Вміст файлу збережений у вигляді масиву байтів

Таблиця 7 – Таблиця Folders для збереження інформації про папки

FolderId	Ідентифікатор папки
FolderName	Ім’я папки
ParentFolderId	Ідентифікатор батьківської папки
LastUpdate	Дата останньої модифікації папки
UserId	Ідентифікатор користувача, якому належить папка
Created	Дата створення файлу

Таблиця 8 – Таблиця Users для збереження інформації про користувачів

UserId	Ідентифікатор користувача
FirstName	Ім’я користувача
LastName	Прізвище користувача
UserName	Логін користувача
Password	Пароль користувача
EmailConfirmed	Логічне значення, що відповідає стану про підтвердження пошти користувачем
Email	Пошта користувача

Таблиця 9 – Таблиця SharedFiles для збереження інформації про файли, доступ до яких був наданий іншим користувачам

SourceUserId	Ідентифікатор користувача, який надав доступ до файлу
SharedUserId	Ідентифікатор користувача, який отримав доступ до файлу
FileId	Ідентифікатор файлу, до якого надано доступ

Таблиця 10 – Таблиця SharedFiles для збереження інформації про файли, доступ до яких був наданий іншим користувачам

SourceUserId	Ідентифікатор користувача, який надав доступ до файлу
SharedUserId	Ідентифікатор користувача, який отримав доступ до файлу
FolderId	Ідентифікатор папки, до якої надано доступ

2.3 Архітектура веб-сервісу

В якості архітектурного підходу до реалізації було обрано Clean Architecture (Чиста архітектура)[20].

Clean Architecture - це підхід до розробки програмного забезпечення, який спрямований на підтримку високої чистоти, модульності і розширюваності коду. Цей підхід ставить акцент на розділення програмного забезпечення на незалежні компоненти з явно визначеними відповідальностями.

Clean Architecture пропонує розбити систему на логічні рівні (рисунок 5), кожен з яких виконує певні функції і має чітко визначені межі.

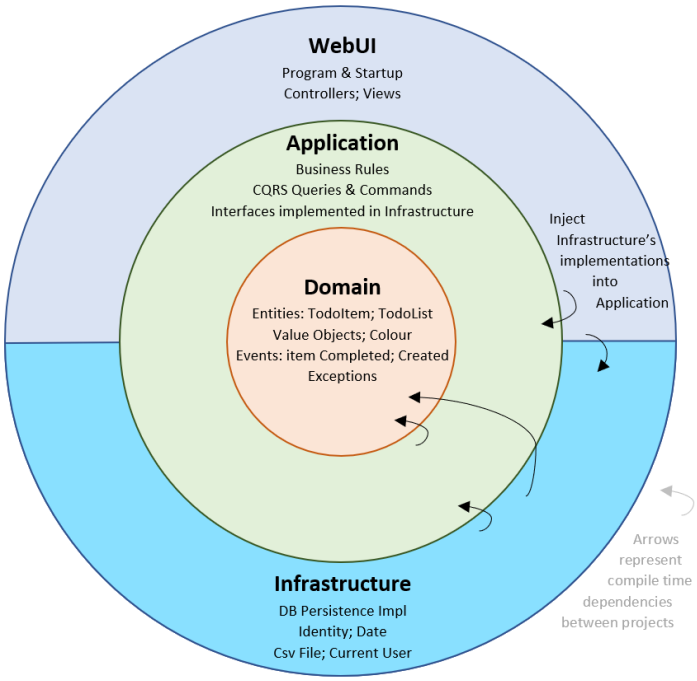

Рисунок 5 – Рівні чистої архітектури[20]

Найважливішими рівнями є:

1. Presentation Layer (Рівень презентації): Відповідає за представлення даних користувачу і взаємодію з ним. Цей рівень може містити елементи інтерфейсу користувача, контролери, презентери або вьюмоделі.
2. Application Layer (Рівень додатку): Містить бізнес-логіку, яка не залежить від фреймворків або зовнішніх факторів. Цей рівень виконує

обробку запитів, координує взаємодію між різними компонентами і визначає правила бізнес-поведінки.

3. Domain Layer (Рівень домену): Містить ядро бізнес-логіки і визначає модель домену. Цей рівень не залежить від деталей реалізації і може бути перенесений до іншого проекту або платформи без змін.
4. Infrastructure Layer (Рівень інфраструктури): Забезпечує зв'язок з зовнішніми ресурсами, такими як бази даних, API, зовнішні служби та інші.

Clean Architecture допомагає зробити систему більш гнучкою і тестованою. Він розділяє відповідальності і забезпечує зв'язок між компонентами через чітко визначені інтерфейси. Це сприяє підтримці коду в чистому стані, легкому для розуміння і змінення[21].

Відповідно до архітектурного підходу, систему було розділено на чотири проекти (рисунок 6):

1. Application – рівень додатку
2. Domain – рівень домену, містить основні
3. Infrastructure – рівень інфраструктури
4. WebUI – рівень презентації

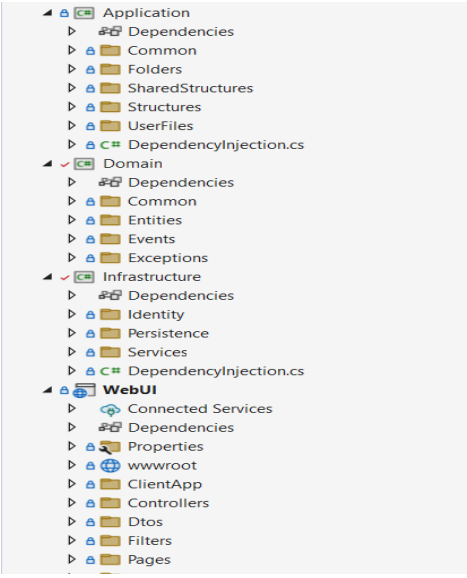

Рисунок 6 – Розділення на проекти

Крім цього, було використано підхід CQRS (Command Query Responsibility Segregation). CQRS – це архітектурний підхід, який розділяє операції запису (команди) та операції читання (запити) в додатку.

У CQRS команди і запити виконуються незалежно один від одного. Команди відповідають за зміну стану додатку і виконують дії, такі як створення, оновлення або видалення даних. Запити, з іншого боку, призначені для отримання даних без їх зміни.

Цей підхід дозволив більш гнучко керувати логікою додатку, оскільки команди та запити можуть мати власну модель даних і є незалежними[20].

2.4 Рівень домену

В домені зберігаються сутності, виключення та типи, які відносяться до домену збереження файлів. Сюди були додані класи (рисунок 7), які представляють моделі, що були визначені на етапі проектування бази даних за допомогою ER-моделі.

```
public class UserFile
{
    public int Id { get; set; }

    public string FileName { get; set; }

    public string FilePath { get; set; }

    public long Size { get; set; }

    public Folder Folder { get; set; }

    public int FolderId { get; set; }

    public ICollection<SharedFile> SharedFiles { get; set; } = new HashSet<SharedFile>();

    public byte[] FileStream { get; set; }
}
```

Рисунок 7 – Клас UserFile

2.5 Рівень додатку

Цей рівень містить всю основну бізнес-логіку для отримання, збереження, модифікації, видалення, завантаження файлів та папок. Оскільки, при розробці системи використовувався CQRS, то логіка була

розділена на команди, які модифікують стан системи, та запити, які слугують тільки для отримання даних.

Прикладом запита є отримання збереженого файлу з системи (рисунок 8).

```
public class GetUserFileQueryHandler : IRequestHandler<GetUserFileQuery, UserFile>
{
    private readonly IApplicationDbContext _context;

    public GetUserFileQueryHandler(IApplicationDbContext context)
    {
        _context = context;
    }

    public async Task<UserFile> Handle(GetUserFileQuery request, CancellationToken cancellationToken)
    {
        var entity = await _context.UserFiles
            .Where(l => l.Id == request.Id)
            .SingleOrDefaultAsync(cancellationToken);

        if (entity == null)
        {
            throw new NotFoundException(nameof(UserFile), request.Id);
        }

        return entity;
    }
}
```

Рисунок 8 – Запит отримання файлу

Основна логіка запиту міститься в методі Handle. В ньому є три логічні блоки:

1. Отримання даних про файл з бази даних за допомогою ідентифікатора файлу. Слід зазначити, що для цього використовується контекст Entity Framework, який дозволив не писати SQL-запит і зробити код зрозумілим та стійким до помилок.
2. Перевірка існування шуканого файлу з базі даних. Якщо файл не був знайдений, система створює виключення, яке сигналізує, про виключну ситуацію в системі та дозволяє її опрацювати.
3. Повернення шуканого файлу, якщо він існує в базі даних.

Оскільки це запит, то він ніяк не модифікує систему, а лише отримує дані про неї.

Прикладом команди є зміна імені файлу (рисунок 9).

```
public async Task<StructureDto> Handle(UpdateUserFileCommand request, CancellationToken cancellationToken)
{
    var entity = await _context.UserFiles.FindAsync(request.Id);

    if (entity == null)
    {
        throw new NotFoundException(nameof(UserFile), request.Id);
    }
    var userFile = await _context.UserFiles
        .Where(x => x.FolderId == entity.FolderId)
        .Where(x => x.FileName == request.FileName)
        .Where(x => x.Id != request.Id)
        .FirstOrDefaultAsync(cancellationToken);

    if (userFile != null)
    {
        throw new BadRequestException($"File {request.FileName} already exist");
    }

    entity.FileName = request.FileName;

    await _context.SaveChangesAsync(cancellationToken);

    return new StructureDto
    {
        Id = entity.Id,
        Name = entity.FileName,
        Path = entity.FilePath,
        Created = entity.Created,
        CreatedBy = entity.CreatedBy,
        LastModified = entity.LastModified,
        LastModifiedBy = entity.LastModifiedBy,
        Size = entity.Size,
        Type = "file"
    };
}
```

Рисунок 9 – Команда зміни імені файлу

Можна зазначити, що в кожному методі є обробка виключних ситуацій. В випадку зміни імені є обробка таких помилок:

1. Шуканий файл не знайдено
2. Нове ім’я файлу недоступне, оскільки в папці вже існує файл з таким іменем.

Це свідчить про те, що система має гарний рівень надійності, оскільки виключення завжди опрацьовуються та система надає відповідне повідомлення користувачеві.

Також, у рівні додатку визначаються інтерфейси (рисунок 10), які реалізуються в зовнішніх рівня. Наприклад, якщо додатку потрібен доступ до бази даних, в додатковому рівні додається новий інтерфейс, а реалізація створюється в інфраструктурному рівні. Це робить систему більш незалежною, оскільки бізнес-логіка не прив'язана до деталей інфраструктури.

```
public interface IApplicationDbContext
{
    DbSet<Folder> Folders { get; set; }

    DbSet<UserFile> UserFiles { get; set; }

    DbSet<SharedFile> SharedFiles { get; set; }

    DbSet<SharedFolder> SharedFolders { get; set; }

    DbSet<ErrorLog> ErrorLogs { get; set; }

    Task<int> SaveChangesAsync(CancellationToken cancellationToken);
}
```

Рисунок 10 – Інтерфейс контексту бази даних

2.6 Рівень інфраструктури

Основну частину інфраструктури займає функціонал для зв'язку з базою даних. Для роботи з базою даних було обрано підхід Code First з Entity Framework.

Entity Framework Code First є підхідом розробки бази даних, який дозволяє створювати моделі даних та бази даних за допомогою коду C# без необхідності ручного створення або внесення змін до схеми бази даних[22].

За допомогою підходу Entity Framework Code First можна визначити класи, які представляють таблиці бази даних, а також властивості цих класів, які відображають колонки бази даних. При запуску додатка Entity Framework автоматично аналізує ці класи та створює відповідну схему бази даних.

Даний підхід надав гнучкість та зручність при роботі з базою даних. Для внесення змін в схему бази даних використовувалися міграції.

Загалом, підхід Entity Framework Code First допоміг ефективно працювати з базою даних, спростив розробку та підтримку схеми бази даних і пришвидшив розробку системи.

Використовуючи Entity Framework, головним елементом є клас контексту бази даних (рисунок 11). Всередині нього було задано поля, які представляють таблиці в базі даних.

```
public class ApplicationDbContext : IdentityDbContext<AppUser, IdentityRole<int>, int,
    IdentityUserClaim<int>, IdentityUserRole<int>, IdentityUserLogin<int>,
    IdentityRoleClaim<int>, IdentityUserToken<int>>, IApplicationDbContext
{
    private readonly ICurrentUserService _currentUserService;
    private readonly IDateTime _dateTime;
    private readonly IDomainEventService _domainEventService;

    public ApplicationDbContext(
        DbContextOptions options,
        ICurrentUserService currentUserService,
        IDomainEventService domainEventService,
        IDateTime dateTime) : base(options)
    {
        _currentUserService = currentUserService;
        _domainEventService = domainEventService;
        _dateTime = dateTime;
    }

    public ApplicationDbContext(DbContextOptions options) : base(options) { }

    public DbSet<Folder> Folders { get; set; }

    public DbSet<UserFile> UserFiles { get; set; }

    public DbSet<SharedFile> SharedFiles { get; set; }

    public DbSet<SharedFolder> SharedFolders { get; set; }
```

Рисунок 11 – Контекст бази даних

Також, для встановлення зв'язку багато-до-багатьох було задано окремі налаштування (рисунок 12), оскільки такі зв'язки не розпізнаються автоматично і схема бази даних може створитися неправильною.

```
class SharedFileConfiguration : IEntityTypeConfiguration<SharedFile>
{
    public void Configure(EntityTypeBuilder<SharedFile> builder)
    {
        builder.HasKey(e => new { e.SourceUser, e.SharedUser, e.FileId });

        builder.Property(e => e.SourceUser)
            .IsRequired();

        builder.Property(e => e.SharedUser)
            .IsRequired();

        builder.Property(e => e.FileId)
            .IsRequired();

        builder.HasOne(e => e.File)
            .WithMany(e => e.SharedFiles)
            .HasForeignKey(e => e.FileId)
            .HasConstraintName("FK_SharedFiles_UserFiles");
    }
}
```

Рисунок 12 – Налаштування атрибутів сутностей

Крім того, за допомогою таких налаштувань можна окремо задавати інші налаштування, такі як максимальну та мінімальну довжина рядка, тип атрибута, значення за замовчуванням.

Також, на рівні інфраструктури реалізовано реєстрацію та авторизацію користувачів (див. додаток А).

2.7 Рівень презентації

Цей рівень є інтерфейсом системи, який складається з двох частин:

1. API (Application Programming Interface), розроблений за допомогою ASP.NET Core Web API, який залежить від рівнів додатку та інфраструктури та використовує їхній функціонал.
2. Клієнтська частина, створена за допомогою JavaScript бібліотеки React.

React взаємодіє з ASP.NET Core Web API за допомогою HTTP-запитів, отримуючи та відправляючи дані на сервер.

2.7.1 API

Основними компонентами API є контролери (рисунок 13), в яких об’явлені кінцеві точки, які використовуються клієнтською частиною для отримання та відправлення даних на сервер.

```
public class FileController : ApiControllerBase
{
    [HttpGet("get-text-file-content/{id}")]
    public async Task<ActionResult<UserTextFileDto>> GetUserTextFile(int id)
    {
        return await Mediator.Send(new GetUserTextFileQuery { Id = id });
    }


    [HttpPost("create-text")]
    public async Task<ActionResult<StructureDto>> CreateTextFile(CreateUserTextFileCommand command)
    {
        return await Mediator.Send(command);
    }

    [HttpPut("update-text")]
    public async Task<ActionResult<StructureDto>> UpdateTextFile(UpdateUserTextFileCommand command)
    {
        if (command.Id == 0)
        {
            return BadRequest();
        }
        return await Mediator.Send(command);
    }

    [HttpPut("update-file-name")]
    public async Task<ActionResult<StructureDto>> UpdateFileName(UpdateUserFileCommand command)
    {
        if (command.Id == 0)
        {
            return BadRequest();
        }

        return await Mediator.Send(command);
    }
}
```

Рисунок 13 – Контролер файлів

Варто зазначити, що всередині них майже не міститься бізнес-логіки. Контролер слугує як вхідна точка, а бізнес-логіка розташована на рівні додатку та інфраструктури. Це зробило компоненти системи більш незалежними та розділило відповідальності кожної з них. Крім того, було використано патерн "Медіатор", який у поєднанні зі структурою CQRS є відмінним патерном, який допомагає зменшити залежності та зберегти чистоту коду в рамках програми. Він дозволяє повторно використовувати компоненти та дотримуватися принципу єдиної відповідальності.

2.7.2 Клієнтська частина

Для розробки клієнтської частини було використано бібліотеку React. Вона відповідає за відображення користувацького інтерфейсу та обробку взаємодії з користувачем.

Також, для зручної інтеграції React з Bootstrap було додано бібліотеку React-Bootstrap, яка дозволила використовувати Bootstrap стилі та класи безпосередньо в React-компонентах. Це дозволило зручно комбінувати стандартні компоненти React з готовими стилями Bootstrap, забезпечуючи швидку і просту розробку.

Для налаштування навігації використовувалася бібліотека react-router, яка дозволяє створювати React компоненти для опису сторінок та шляхів до них (рисунок 14).

```
const AppRouter = observer(() => {
  const { user, storage } = useContext(Context);

  return (
    <Switch>
      {user.isAuth &&
        authRoutes.map(({ path, Component }) => (
          <Route key={path} path={path} component={Component} exact />
        ))}
      {publicRoutes.map(({ path, Component }) => (
        <Route key={path} path={path} component={Component} exact />
      ))}
      {user.isAuth ? (
        <Redirect from="*" to={STORAGE_ROUTE + '/' + storage.rootId} />
      ) : (
        <Redirect from="*" to={LOGIN_ROUTE} />
      )}
    </Switch>
  );
});
```

Рисунок 14 - Налаштування маршрутизації

Оскільки система містить сторінки, які недоступні не авторизованим користувачам, то система це перевіряє і забороняє доступ до них. В випадку, якщо користувач не має доступу, то відбувається переадресація на сторінку авторизації.

При реалізації форм, наприклад, для реєстрації та авторизації були задані валідаційні схеми, які дозволили виводити відповідне повідомлення користувачеві під полем форми (див. додаток Б).

Для форми реєстрації задані такі обмеження:

1. Ім’я є обов’язковим.
2. Прізвище є обов’язковим.
3. Пошта має бути коректною та обов’язковою.
4. Логін має містити мінімум 3 символи.
5. Пароль має містити мінімум один символ верхнього регістру, мінімум одну цифру та один спеціальний символ.

Також, після відправки форми, на сервері перевіряється чи існує вже такий логін. Якщо логін існує, то повертається відповідь з сервера з кодом 400 та

повідомленням про те, що такий логін існує. На клієнтській частині ця помилка оброблюється і користувачеві показується відповідне повідомлення на інтерфейсі (рисунок 15).

```
const register = async (firstName, lastName, email, username, password,setSubmitting) => {
  setError(false);
  try {

    await signUp(firstName, lastName, email, username, password);

    user.setIsRegister(true);
    setError(false);
    setSubmitting(true);
    history.push(LOGIN_ROUTE);
    toast.info('For successfully authorization you must confirm your email');
  }
  catch (e) {
    setError(true);
    setSubmitting(false);
  }
};
```

Рисунок 15 – Відправка форми на сервер

Для взаємодії з сервером були створені сервіси, які містять усі необхідні функції (рисунок 16).

```
export const signUp = async (
  firstName,
  lastName,
  email,
  username,
  password
) => {
  const { data } = await $host.post('api/account/register', {
    firstName,
    lastName,
    email,
    username,
    password,
  });

  return data;
};

export const login = async (username, password) => {
  const { data } = await $host.post('api/account/login', {
    username,
    password,
  });

  localStorage.setItem('token', data.token);
  return jwt_decode(data.token);
};
```

Рисунок 16 – Функції для реєстрації та авторизації

Крім того, для обробки помилок був доданий перехоплювач запитів, який в разі помилкової відповіді з сервера, показує відповідне повідомлення на інтерфейсі користувачеві (рисунок 17).

```
function (error) {
  if (error) {
    switch (error.response.status) {
      case 400:
        if (error.response.data.errors) {
          let validationError = '';
          for (const key in error.response.data.errors) {
            if (error.response.data.errors[key]) {
              validationError = error.response.data.errors[key][0];
              break;
            }
          }
          toast.error(validationError);
        } else if (error.response.data.detail) {
          toast.error(error.response.data.detail);
        } else {
          toast.error(error.response.data);
        }
        break;
      case 401:
        Router.replace(LOGIN_ROUTE);
        toast.error(`${error.response.status}: Unauthorized`);
        break;
      case 403:
        toast.error(`${error.response.status}: Forbidden access`);
        break;
      case 404:
        toast.error(`${error.response.status}: Not found`);
        break;
      default:
        toast.error('Something unexpected went wrong');
        break;
    }
  }
```

Рисунок 17 – Обробка помилок на клієнтській частині

РОЗДІЛ 3. ТЕСТУВАННЯ

3.1 Вибір технології

Процесу тестування передувало порівняння технологій для тестування, які б найбільше підійшли для серверної частини. Оскільки серверна частина використовує мову C#, то розглядались такі варіанти[23]:

1. XUnit є легким і простим у використанні фреймворком, який пропонує широкий набір функцій для створення тестів. Він підтримує концепцію "тест-класів" і "тест-методів", де кожен метод представляє окремий тестовий сценарій. XUnit надає багато вбудованих ассертів для перевірки умов і очікуваних результатів.
2. NUnit є ще одним популярним фреймворком для тестування, який пропонує багато функціональності для створення і виконання тестів. Він має широкі можливості для налаштування тестових сценаріїв і підтримує такі концепції, як "тест-класи", "тест-методи" і "тест-фікстури". NUnit надає багато вбудованих ассертів та можливостей для налаштування запуску тестів.
3. MSTest є фреймворком для модульного тестування, який постачається разом з Visual Studio. Він надає інтегровану підтримку для створення, виконання і аналізу тестів без необхідності встановлення додаткових компонентів. MSTest підтримує схожі концепції "тест-класів" і "тест-методів", а також має вбудовані ассерти і засоби для організації тестів.

Для системи був обраний XUnit по наступним перевагам відносно інших технологій:

1. Простота використання: XUnit пропонує простий і зрозумілий синтаксис для написання тестів. Він використовує атрибути для маркування тестових методів і не вимагає додаткового налаштування. Це робить процес створення і виконання тестів швидким і безпроблемним.
2. Параметризовані тести: XUnit дозволяє створювати параметризовані тести, що дозволяє виконувати один і той самий тестовий метод з

різними вхідними параметрами. Це дозволяє провести багато тестів з використанням мінімального написання коду.

3. Компактність тестового коду: XUnit надає можливість використовувати анонімні функції і лямбда-вирази для скорочення розміру тестового коду. Це дозволяє зосередитись на самому тестуванні і зменшити кількість зайвих вузлів коду.
4. Розширюваність: XUnit має розширювану архітектуру, що дозволяє створювати власні власні ассерти, атрибути, фікстури і інші компоненти для власних потреб тестування.

3.2 Розробка тестів

Основними тестами в системі є модульні тести, які покривають рівень додатку, бо вони містять основну бізнес-логіку та неправильність їхньої роботи може містити критичні наслідки.

Оскільки основною частиною коду на рівні додатку є команди та запити, то було створено два базових класи для даних типів тестів, щоб уникнути дублювання в тестах (рисунок 18). В них створюються об'єкти-заглушки, які імітують роботу з базою даних та отримання інформації про поточного користувача.

```
public class CommandTestBase : IDisposable
{
    public CommandTestBase()
    {
        Context = ApplicationDbContextFactory.Create();

        var currentUserServiceMock = new Mock<ICurrentUserService>();
        currentUserServiceMock
            .Setup(m => m.UserId)
            .Returns("1");

        CurrentUserService = currentUserServiceMock.Object;
    }

    public ApplicationDbContext Context { get; }

    public ICurrentUserService CurrentUserService { get; }

    public void Dispose()
    {
        ApplicationDbContextFactory.Destroy(Context);
    }
}
```

Рисунок 18 – Базовий клас для тестів команд

Розглянемо приклад тесту на основі тестового класу для операції зміни вмісту текстового файлу (рисунок 19). В ньому присутні два тести:

1. Handle_ShouldUpdateTextFile – перевіряє успішний результат виконання операції
2. Handle_GivenInvalidId_ThrowsException – перевіряє помилковий результат операції

Розглянемо перший тест. Він ділиться на три логічні блоки:

1. Ініціалізація тестових даних – створюється об’єкт команди UpdateUserTextFileCommand, який має бути переданим в функцію
2. Виклик функції, яка тестується – виклик функції Handle;
3. Перевірка справжнього результату з очікуваним – перевірка, що текст, який був переданий в команду UpdateUserTextFileCommand в першому блоці, зберігся та повернувся в результаті виклику функції з другого блоку.

```
[Fact]
public async Task Handle_ShouldUpdateTextFile()
{
    var command = new UpdateUserTextFileCommand
    {
        Id = 1,
        Text = "Qwerty123"
    };

    var sut = new UpdateUserTextFileCommandHandler(Context);

    var result =  await sut.Handle(command, CancellationToken.None);

    var entity = Context.UserFiles.Find(result.Id);

    entity.FileStream.ShouldBe(Encoding.UTF8.GetBytes(command.Text));
}
```

Рисунок 19 – Перевірка успішної модифікації вмісту текстового файлу

В другому тесті (рисунок 20) ситуація схожа, окрім того, що замість перевірки результату функції, перевіряється, що всередині функції виникло виключення NotFoundException, яке сигналізує систему, що файл з шуканим ідентифікатором не був знайдений.

```
[Fact]
public async Task Handle_GivenInvalidId_ThrowsException()
{
    var command = new UpdateUserTextFileCommand
    {
        Id = 99999,
        Text = "Qwerty123"
    };

    var sut = new UpdateUserTextFileCommandHandler(Context);

    await Should.ThrowAsync<NotFoundException>(() =>
         sut.Handle(command, CancellationToken.None));
}
```

Рисунок 20 - Перевірка створення виключення при не існуючому ідентифікаторі файлу

Результати виконання тестів надані у додатку В.

РОЗДІЛ 4. РОБОТА З СИСТЕМОЮ

4.1 Реєстрація

Для використання функцій сервісу користувачу потрібно зареєструватися.

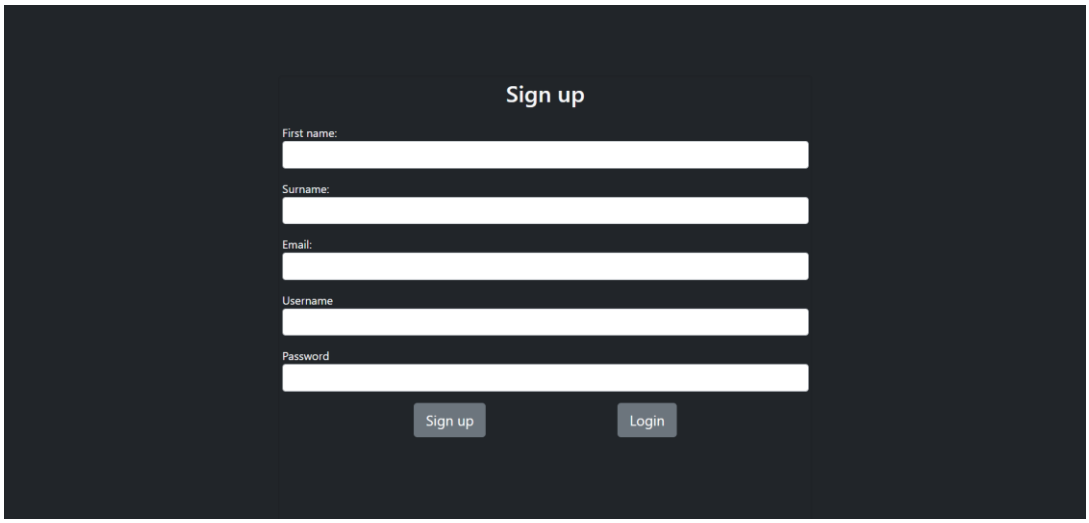

Рисунок 21 – Сторінка реєстрації користувача

Після введення потрібної для реєстрації інформації (рисунок 21), користувачу на пошту повинен прийти відповідний лист, в якому він має перейти за посиланням для активації облікового запису. Після цього користувач отримує можливість ввійти в свій акаунт та має повний доступ до функцій сервісу.

4.2 Збереження файлів

Після введення логіну та паролю на сторінці авторизації, користувача перенаправляє на основну сторінку системи (рисунок 22).

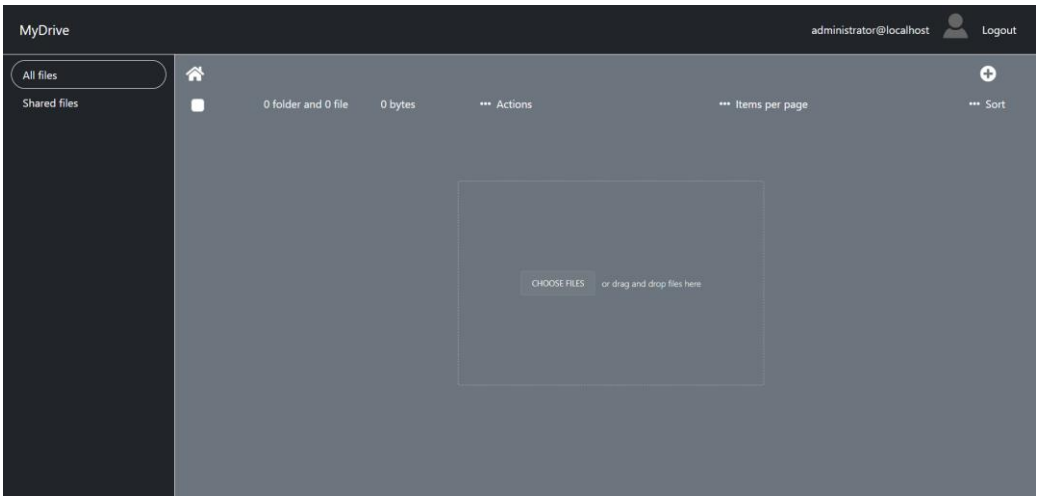

Рисунок 22 – Основна сторінка системи

Для збереження файлу в системі можна натиснути на кнопку “Choose files”, яка відкриє вікно з вибором файлів з файлової системи, або перетягнути файл в зону дії drag-and-drop елемента.

4.3 Завантаження та видалення файлів та папок

Для видалення або завантаження файлу з сервісу потрібно натиснути на іконку з трьома крапками для виклику додаткового меню і обрати відповідно “Delete” або “Download” (рисунок 23).

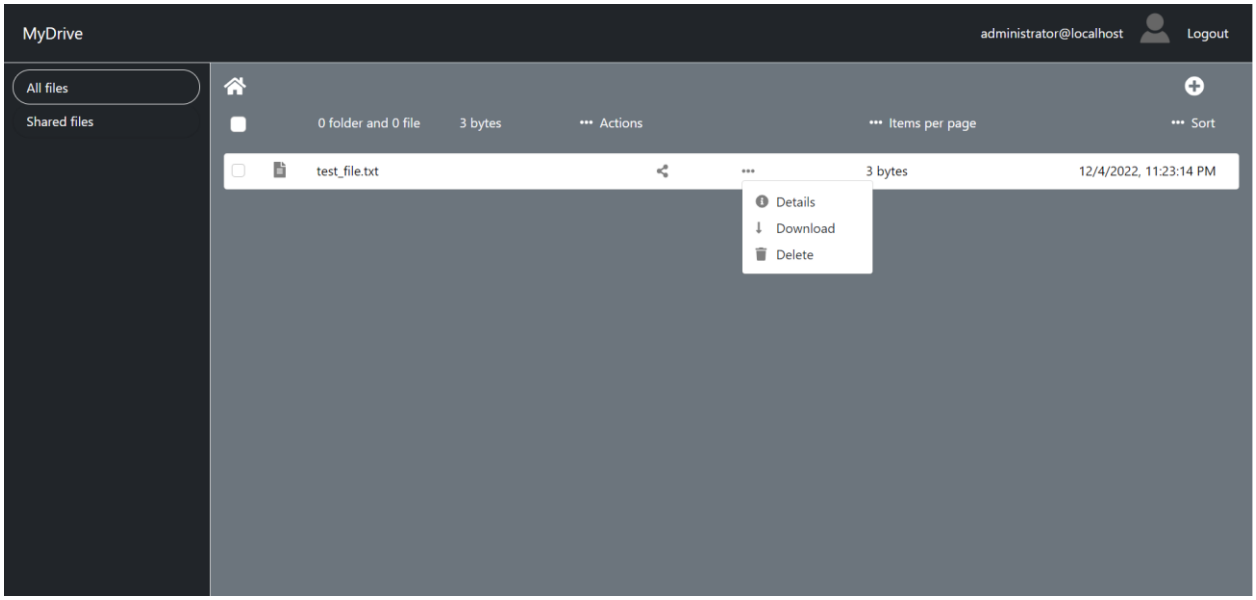

Рисунок 23 – Завантаження та видалення файлів

4.4 Створення папок

Для створення папки потрібно натиснути на іконку з знаком плюс і обрати “Folder”. Натиснувши на папку, можемо перейти всередину неї.

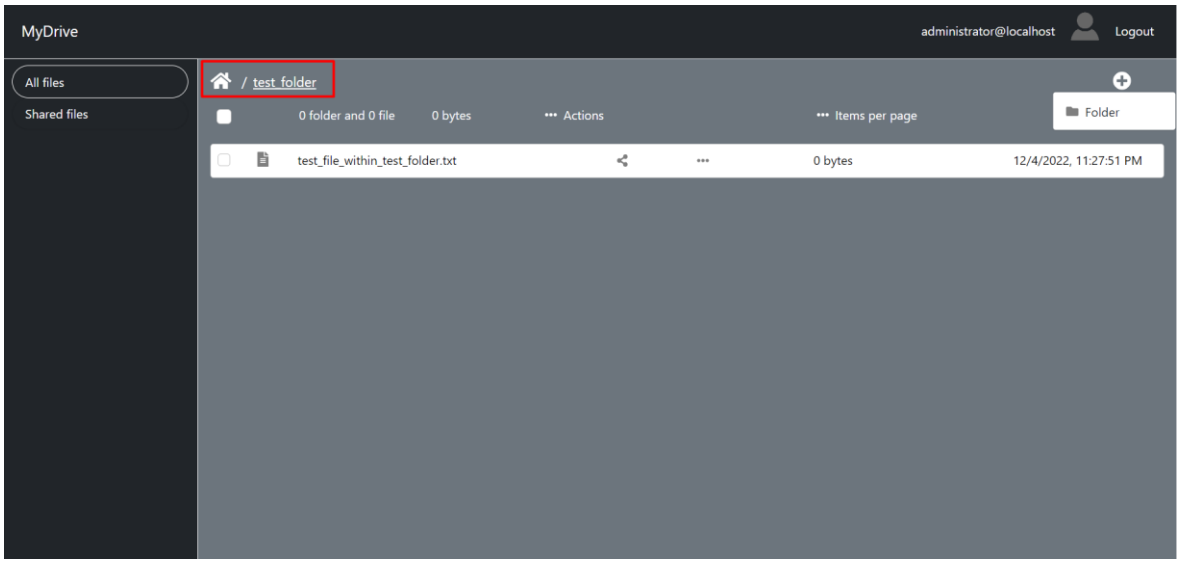

Рисунок 24 – Створення папки та перехід всередину папки

На рисунку 24 можна побачити, що перейшовши всередину папки “test_folder”, система показує, в якій папці ми знаходимося, а також надає можливість переходити назад, натиснувши на іконку будинка.

4.5 Створення текстових файлів

Однією з зручних особливостей сервісу є можливість створення текстових файлів (рисунок 25). Для цього потрібно натиснути на іконку з знаком плюс і обрати “Text File”.

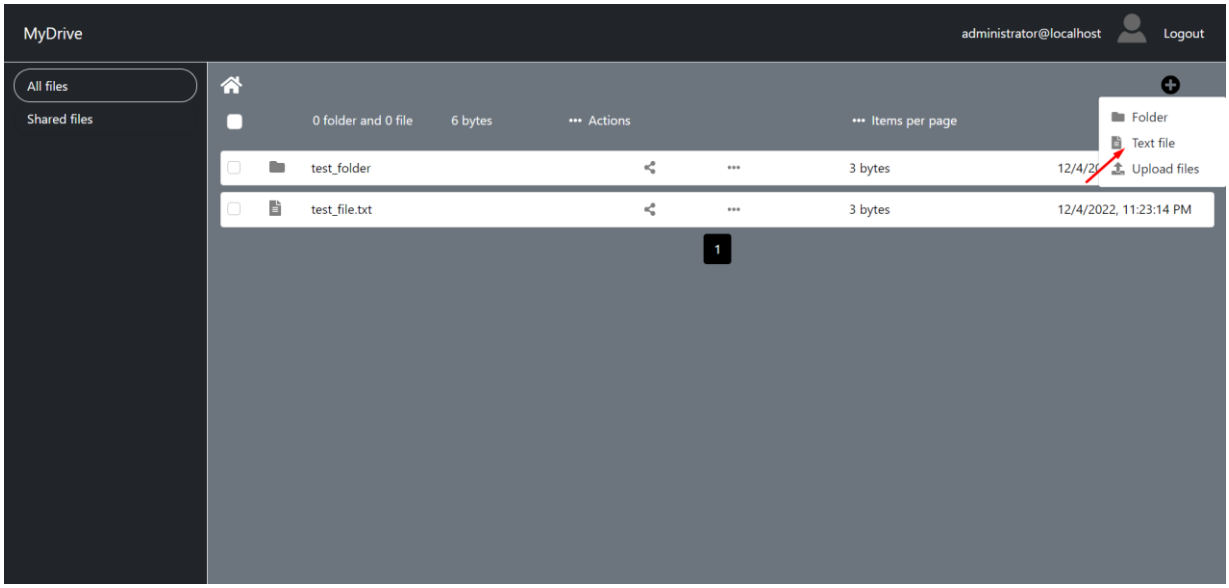

Рисунок 25 – Створення текстового файлу

Після цього відкриється модальне вікно (рисунок 26), в якому потрібно вказати ім’я файлу і натиснути на кнопку “Create”.

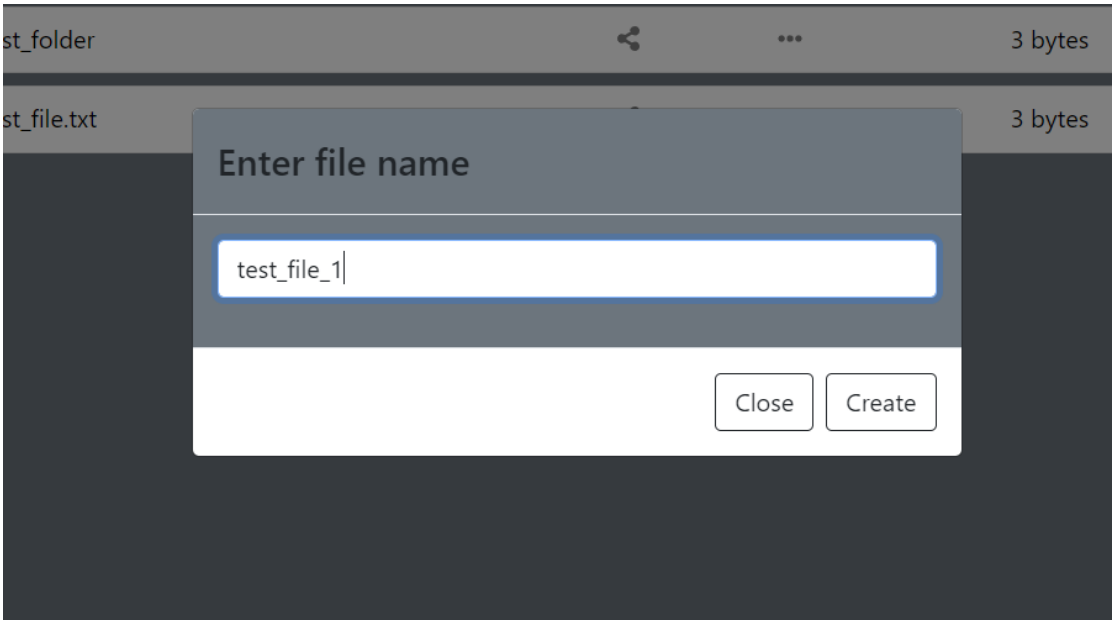

Рисунок 26 – Введення ім’я для текстового файлу

Результатом буде поява текстового файлу в списку без перезавантаження сторінки та нотифікації про успішне створення (рисунок 27).

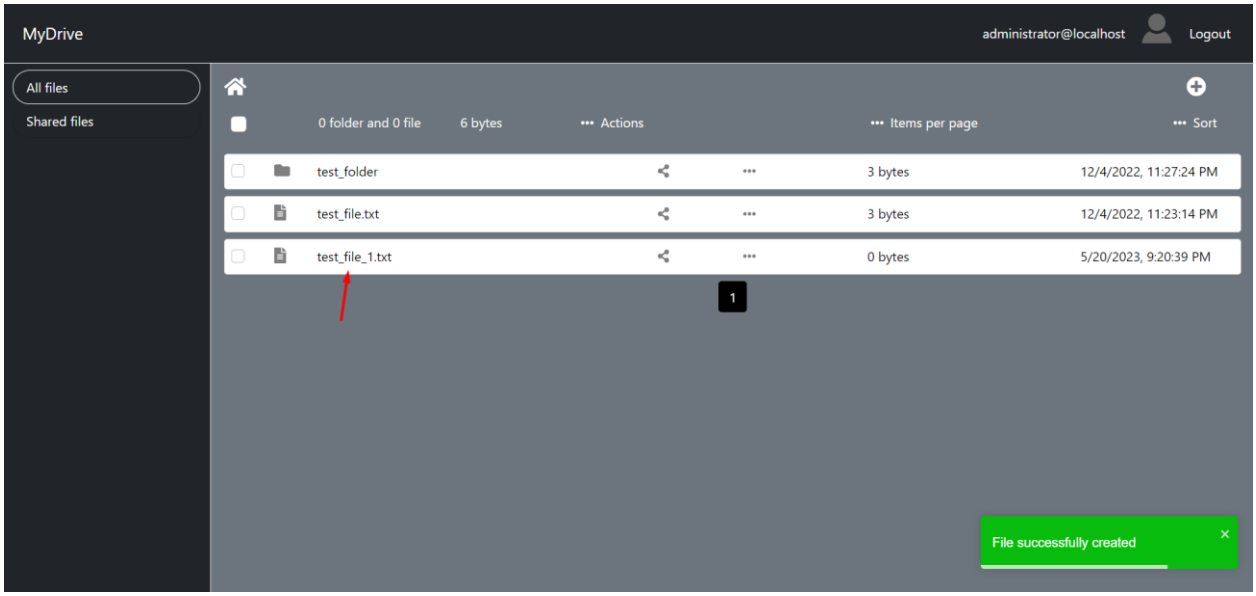

Рисунок 27 – Успішне створення текстового файлу

4.6 Редагування текстових файлів

Для редагування текстового файлу потрібно натиснути на файл, після чого відкриється модальне вікно (рисунок 28) з вмістом файлу, який можна редагувати.

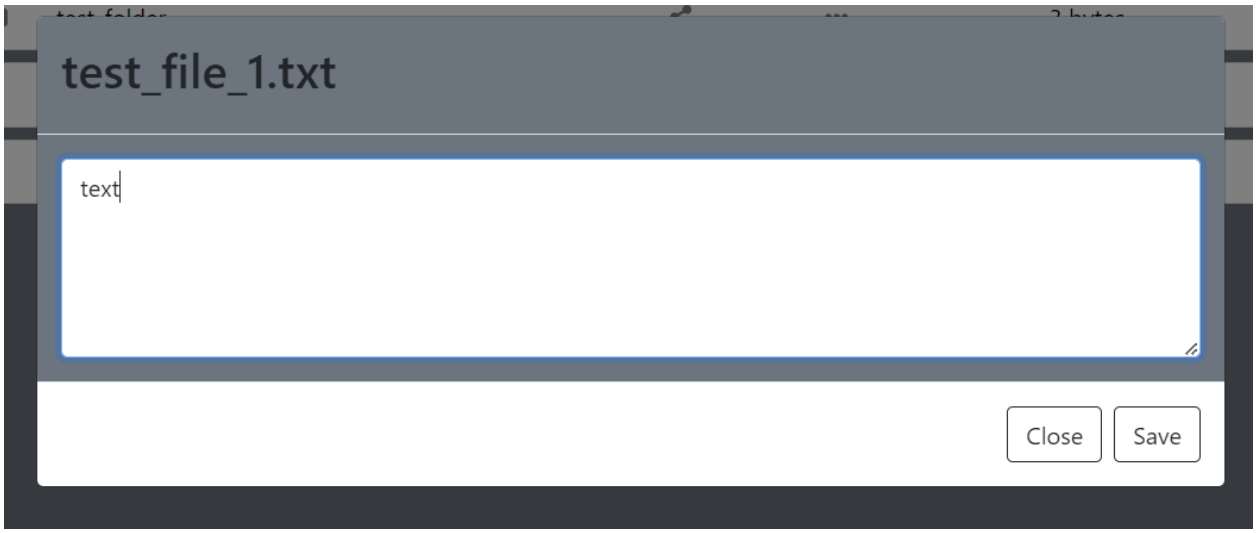

Рисунок 28 – Редагування вмісту текстового файлу

4.7 Переміщення та копіювання файлів та папок

Переміщати та копіювати файли та папки можна двома шляхами: по одному або по декілька. Для переміщення або копіювання одного файлу або

однієї папки потрібно натиснути на іконку з трьома крапками та обрати там “Cut or copy”. Після цього відкриється модальне вікно (рисунок 29), в якому можна буде обрати папку до якої буде переміщено елемент.

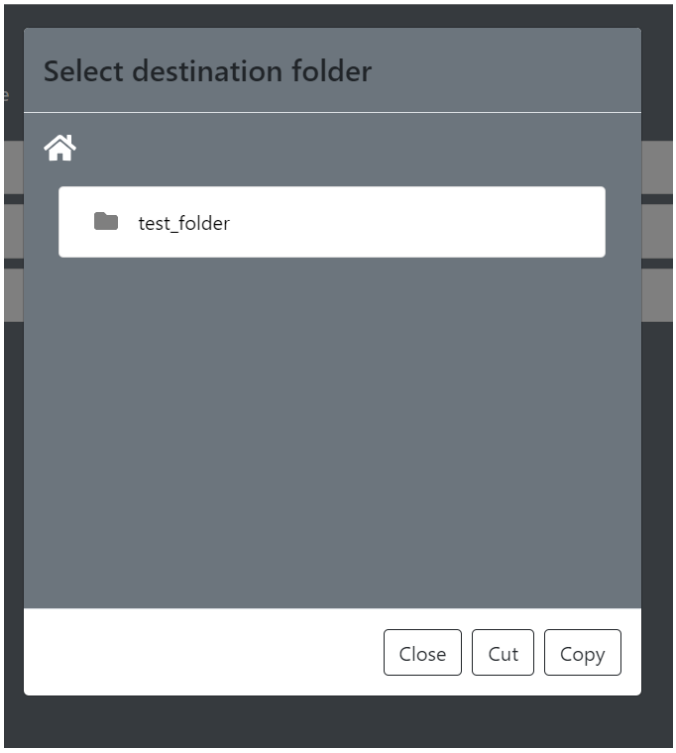

Рисунок 29 – Модальне вікно для переміщення або копіювання файлу або папки

У випадку, якщо потрібно перемістити або скопіювати декілька файлів або папок, то потрібно обрати їх за допомогою прапорців (рисунок 30). Після цього потрібно натиснути на “Actions” та обрати дію “Cut or copy”. Як результат буде відкрито таке ж модальне вікно, як при переміщенні чи копіюванні одного елементу.

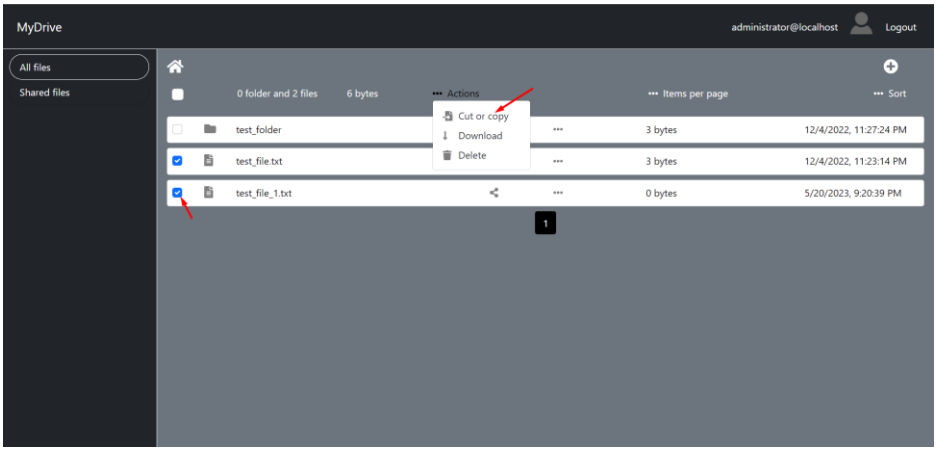

Рисунок 30 – Переміщення та копіювання декількох елементів

4.8 Спільний доступ

Щоб надати доступ до файлу чи папки іншому користувачу потрібно натиснути на іконку поширення (рисунок 31).

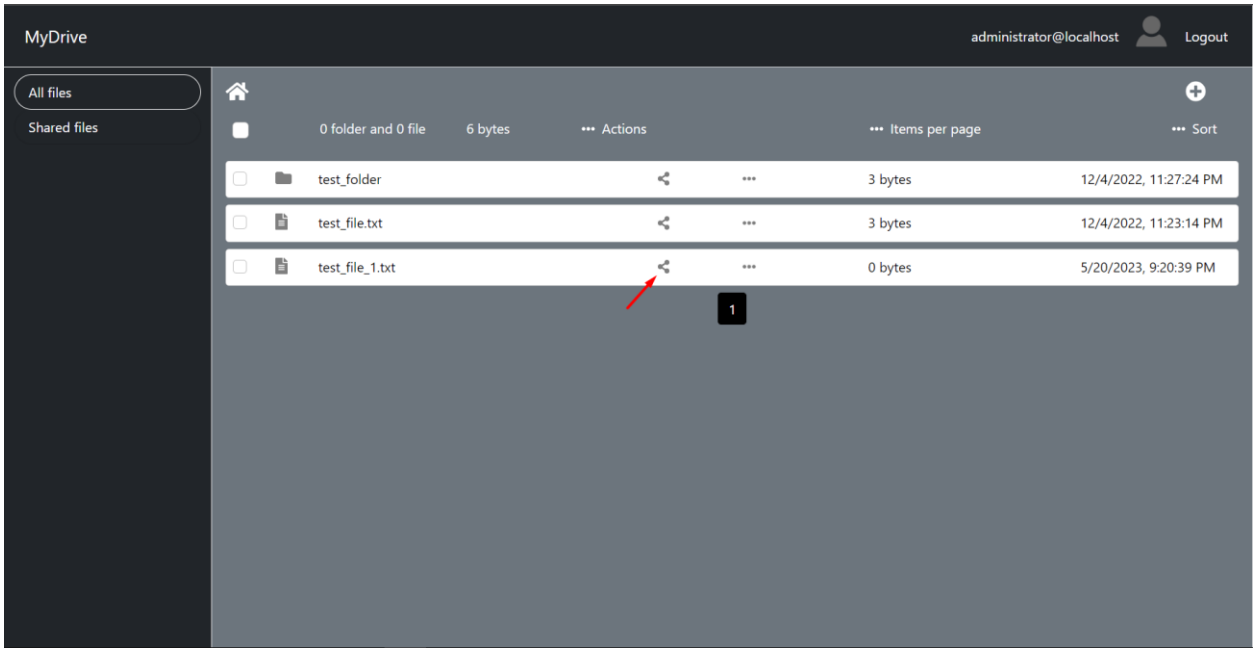

Рисунок 31 – Надання доступу до файлу

Після цього відкриється модальне вікно (рисунок 32), в якому можна ввести логін користувача, якому потрібно надати доступ. Достатньо ввести декілька літер, після чого буде показано всіх користувачів, які в логіні містять ці літери. Далі треба вибрати користувача за допомогою прапорця та натиснути на кнопку “Share”.

Рисунок 32 – Пошук та вибір користувачів для надання доступу

У результаті користувач, якому було надано доступ, зможе переглянути поширену йому файл чи папку в меню “Shared files” (рисунок 33).

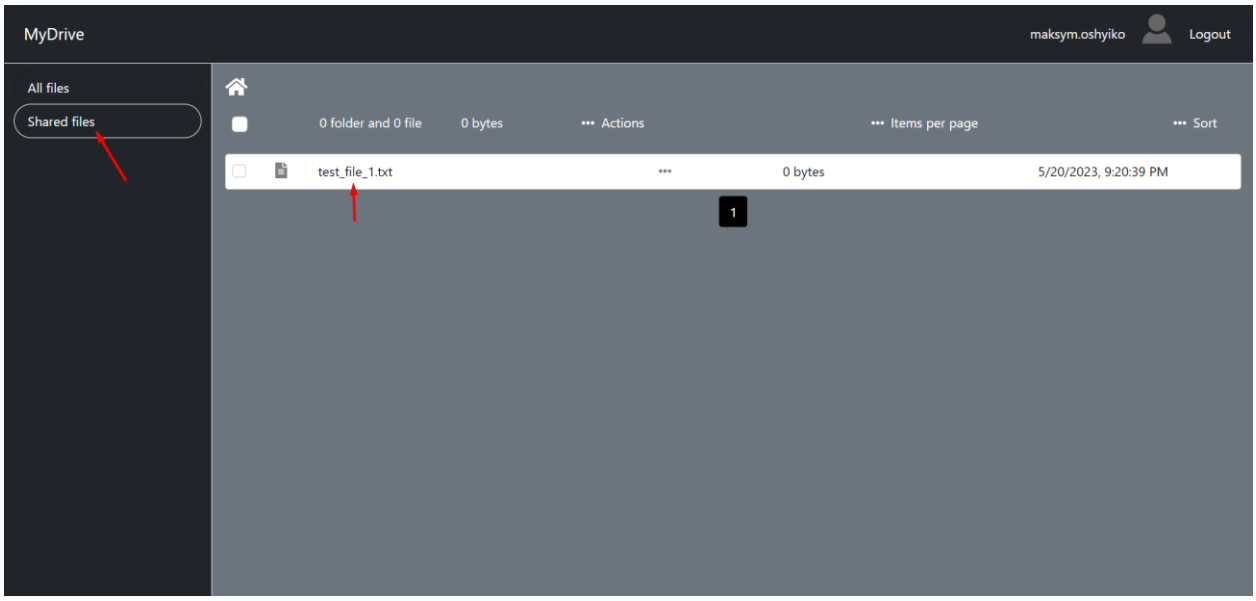

Рисунок 33 – Меню спільного доступу

ВИСНОВКИ

Розроблено веб-сервіс для збереження файлів з можливістю завантаження, створення, копіювання та переміщення файлів та папок.

Досліджено та використано сучасні технології та методи, які використовуються для розробки веб-сервісів для зберігання файлів та зроблено огляд існуючих веб-орієнтованих рішень. Було сформульовано вимоги до системи, розроблено схему бази даних та архітектуру веб-сервісу, яка поділяє сервіс на 4 логічних рівні: домену, додатку, інфраструктури та презентації.

Розроблено веб-інтерфейс, який дозволяє зберігати, керувати та спільно використовувати файли та папки за допомогою контролю доступу до файлів та папок, що дозволяє надавати доступ до власних файлів іншим користувачам системи.

Розроблена система має можливість створювати текстові файли всередині системи та змінювати їхній вміст. Це дає змогу швидко зберегти певний текст без потреби його створення та подальшого завантаження поза системою.

Серверна частина системи покрита модульними тестами, що підвищує надійність та перевіряє основні сценарії використання системи.

Перспективні напрями розробки:

1. Перегляд зображень та PDF документів.
2. Надання спільного доступу на читання або читання та модифікацію.
3. Видалення файлів через кошик та можливість їх відновлення.
4. Встановлення обмеження на розмір сховища для користувача.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. FTP Overview, History and Standards. URL: http://www.tcpipguide.com/free/t_FTPOverviewHistoryandStandards.htm (дата звернення: 25.05.2023).
2. The Best Cloud Storage and File-Sharing Services for 2023. PCMAG. URL: https://www.pcmag.com/picks/the-best-cloud-storage-and-file-sharing-services (дата звернення: 25.05.2023).
3. Dropbox - Experience. URL: https://experience.dropbox.com (дата звернення: 25.05.2023).
4. Top 10 Google Drive Features. NoBlue. URL: https://noblue.co.uk/news-updates/b2b/top-ten-google-drive-features (дата звернення: 25.05.2023).
5. Benefits of OneDrive | Why Microsoft OneDrive? | ramsac. ramsac Ltd. URL: https://www.ramsac.com/blog/benefits-of-onedrive (дата звернення: 25.05.2023).
6. Comparison: Dropbox VS Google Drive VS OneDrive. URL: https://www.isunshare.com/blog/comparison-dropbox-vs-google-drive-vs-onedrive/ (дата звернення: 25.05.2023).
7. What is Visual Studio? Microsoft Learn: Build skills that open doors in your career. URL: https://learn.microsoft.com/en-us/visualstudio/get-started/visual-studio-ide?view=vs-2022 (дата звернення: 25.05.2023).
8. A tour of C# - Overview. URL: https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/ (дата звернення: 25.05.2023).
9. C# Features. URL: https://www.javatpoint.com/csharp-features (дата звернення: 25.05.2023).
10. Greene J., Stellman A. Head First C#. O'Reilly Media, Incorporated, 2013.
11. Overview of ASP.NET Core. Microsoft Learn. URL: https://learn.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-7.0 (дата звернення: 25.05.2023).
12. Freeman A. Pro ASP. NET MVC 5. Springer, 2014.
13. Entity Framework Core Tutorials. Entity Framework Tutorial. URL: https://www.entityframeworktutorial.net/efcore/entity-framework-core.aspx (дата звернення: 25.05.2023).
14. Overview of Entity Framework Core - EF Core. Microsoft Learn. URL: https://learn.microsoft.com/en-us/ef/core/ (дата звернення: 25.05.2023).
15. Editions and supported features of SQL Server 2019 - SQL Server. Microsoft Learn. URL: https://learn.microsoft.com/en-us/sql/sql-server/editions-and-components-of-sql-server-2019 (дата звернення: 25.05.2023).
16. SQL Server Management Studio (SSMS). TutorialsTeacher - Learn Technologies. URL: https://www.tutorialsteacher.com/sqlserver/sql-server-management-studio (дата звернення: 25.05.2023).
17. Get started with Bootstrap. Bootstrap · The most popular HTML, CSS, and JS library in the world. URL: https://getbootstrap.com/docs/4.6/getting-started/introduction/ (дата звернення: 25.05.2023).

18. Explain the components of Bootstrap - GeeksforGeeks. URL: https://www.geeksforgeeks.org/explain-the-components-of-bootstrap/ (дата звернення: 25.05.2023).
19. ReactJS - Overview. Online Courses and eBooks Library. URL: https://www.tutorialspoint.com/reactjs/reactjs_overview.htm(дата звернення: 25.05.2023).
20. Clean Architecture for ASP.NET Core Solution: A Case Study - NDepend. URL: https://blog.ndepend.com/clean-architecture-for-asp-net-core-solution/ (дата звернення: 25.05.2023).
21. Мартін Роберт. Чиста архітектура: Мистецтво розроблення програмного забезпечення / пер. з англ. І. Бондар-Терещенко. – Харків : Вид-во «Ранок» : Фабула, 2021. – 368 с.
22. What is Code-First?. Entity Framework Tutorial. URL: https://www.entityframeworktutorial.net/code-first/what-is-code-first.aspx (дата звернення: 25.05.2023).
23. NUnit vs. XUnit vs. MSTest: Comparing Unit Testing Frameworks In C#. LambdaTest. URL: https://www.lambdatest.com/blog/nunit-vs-xunit-vs-mstest/ (дата звернення: 25.05.2023).

ДОДАТОК А

Реалізація реєстрації та авторизації на сервері

```
public async Task CreateUserAsync(string username, string password, string email,
    string firstName, string lastName)
{
    var user = new AppUser
    {
        UserName = username,
        Email = email,
        FirstName = firstName,
        LastName = lastName
    };

    var result = await _userManager.CreateAsync(user, password);

    if (result.Succeeded)
    {
        var token = await _userManager.GenerateEmailConfirmationTokenAsync(user);

        var callbackLink = _generator.GetUriByAction("ConfirmEmail", "Account",
            new { userId = user.Id, token }, _httpContextAccessor.HttpContext.Request.Scheme,
            _httpContextAccessor.HttpContext.Request.Host);

        await _emailService.SendEmailAsync(email, "Confirm your account",
                            $"Thank you for registering MyDrive account." +
                            $" In order to verify your account click here: <a href='{callbackLink}'>link</a>");
    }
    else
    {
        throw new IdentityException(result.Errors.Select(x => x.Description).ToArray());
    }
}
```

```
public async Task<IIdentityResult> AuthorizeAsync(string username, string password)
{
    var user = await _userManager.FindByNameAsync(username);

    if (user == null)
    {
        throw new UnauthorizedAccessException();
    }

    if (!user.EmailConfirmed)
    {
        throw new BadRequestException("Email is not confirmed.");
    }

    var result = await _signInManager.CheckPasswordSignInAsync(user, password, false);

    if (result.Succeeded)
    {
        var userDto = new UserDto
        {
            Id = user.Id,
            UserName = username,
            Token = await _jwtService.CreateToken(username, user.Id.ToString())
        };


        return userDto;
    }

    throw new UnauthorizedAccessException();
}
```

ДОДАТОК Б

Валідація полей форми реєстрації та авторизації на клієнтській частині

```
validationSchema={yup.object().shape({
  firstName: yup.string().required('Name is required'),
  lastName: yup.string().required('Last name is required'),
  email: yup
    .string()
    .email('Incorrect email')
    .required('Email is required'),
  username: yup
    .string()
    .min(3, 'Too short username')
    .required('Username is required'),
  password: yup
    .string()
    .minUppercase(1, 'Password must have 1 uppercase letter')
    .minNumbers(1, 'Password must have at least 1 number')
    .minSymbols(1, 'Password must have at least 1 special symbol')
    .required('Password is required'),
})}
```

```
validationSchema={Yup.object().shape({
  username: Yup.string()
    .min(3, 'Too short username')
    .required('Username is required'),
  password: Yup.string()
    .min(6, 'Too short password')
    .required('Password is required'),
})}
```

ДОДАТОК В

Результати виконання тестів для серверної частини