КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра інтелектуальних програмних систем

Кваліфікаційна робота

на здобуття освітнього рівня бакалавра
за спеціальністю 121 Інженерія програмного забезпечення
на тему:

РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ДИСТАНЦІЙНОГО ЗБЕРЕЖЕННЯ ІНФОРМАЦІЇ

Виконав студент 4-го курсу
Матвій ГОЛЬЦЕВ _____
(підпис)

Науковий керівник:
доцент, кандидат фіз.-мат. наук
Євгеній ІВАНОВ _____
(підпис)

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

Студент _____
(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри інтелектуальних програмних систем
« 29 » травня 2023 р.,
протокол № 11
Завідувач кафедри
Олександр ПРОВОТАР _____
(підпис)

Київ – 2023

РЕФЕРАТ

Обсяг роботи 40 сторінок, 7 ілюстрацій, 8 джерел посилань

РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, ДИСТАНЦІЙНЕ ЗБЕРЕЖЕННЯ ІНФОРМАЦІЇ, ASP.NET CORE, AWS S3, SQLITE, ENTITY FRAMEWORK, ANGULAR, ЧИСТА АРХІТЕКТУРА, АВТЕНТИФІКАЦІЯ JWT, ПРОДУКТИВНІСТЬ, БЕЗПЕКА, МАСШТАБОВАНІСТЬ.

Об'єкт роботи: Розробка програмного забезпечення для дистанційного збереження інформації.

Предмет роботи: Розробка ефективної та безпечної системи збереження та обміну файлами в хмарному сховищі.

Мета роботи: Розробити програмне забезпечення, яке забезпечує користувачам зручний та безпечний доступ до дистанційного збереження файлів.

Методи розроблення: Використання ASP.NET Core, AWS S3, SQLite, Entity Framework, Angular та інших сучасних технологій для створення серверної та клієнтської частини програмного забезпечення.

Результати роботи: Розробка функціонального програмного забезпечення, яке дозволяє завантажувати та зберігати файли в хмарному сховищі з підтримкою безпеки та ефективної роботи.

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

ASP.NET – Active Server Pages .NET, активні серверні сторінки;

AWS – Amazon Web Services, амазон веб-сервіси;

CRUD – Create, Read, Update, Delete, створення, читання, оновлення і вилучення;

API – Application Programming Interface, інтерфейс програмування застосунків;

JWT – JSON Web Token, веб-токен;

JSON – JavaScript Object Notation, запис об'єктів JavaScript;

SQL – Structured Query Language, мова структурних запитів;

HTML – Hypertext Markup Language, мова розмітки гіпертексту;

CSS – Cascading Style Sheets, каскадні таблиці стилів;

SPA – Single Page Application, односторінковий додаток;

ЗМІСТ

ВСТУП

В сучасному світі інформація стала однією з найважливіших ресурсів для людей і організацій. Ми зберігаємо інформацію в різних форматах, таких як документи, фотографії, відео та аудіо файлі. Важливість безпечного зберігання і доступу до цієї інформації ніколи не була вищою, ніж зараз.

Однак зберігання великої кількості інформації може бути викликом, особливо коли вона має бути доступною з різних місць. Традиційні методи зберігання інформації на фізичних носіях, таких як жорсткі диски або SSD, мають обмеження, зокрема обмежений обсяг зберігання, ризик втрати даних через збій обладнання або фізичне пошкодження, а також незручності з доступом до інформації з різних місць.

Хмарне зберігання вирішує багато з цих проблем, пропонуючи масштабоване, безпечне і доступне з будь-де зберігання даних. Однак, розробка програмного забезпечення, яке надає можливість зберігання даних у віддаленому вигляді, вимагає вмілого поєднання різних технологій, а також глибокого розуміння потреб користувачів і найкращих практик у сфері безпеки даних.

Важливість теми обумовлена зростаючими потребами людей і організацій у надійному, безпечному і доступному зберіганні даних. У контексті цифровізації та інформатизації сучасного суспільства обсяг даних, що генеруються і обробляються, постійно зростає. Це створює попит на ефективні рішення для зберігання і доступу до цих даних. Окрім того, під час пандемії COVID-19 багато організацій перейшли на дистанційну роботу, що призвело до зростання попиту на рішення для віддаленого зберігання даних.

Однак, попри наявність комерційних рішень для зберігання даних в хмарі, не завжди існує програмне забезпечення, що відповідає конкретним потребам

користувачів або організацій, або яке є доступним і простим у використанні. Тому розробка програмного забезпечення для дистанційного збереження інформації є актуальною проблемою, яка має великий потенціал для впровадження і комерційного успіху.

Ця кваліфікаційна робота зосереджена на розробці програмного забезпечення для дистанційного збереження інформації, використовуючи сучасні технології та платформи, такі як ASP.NET Core, AWS S3, Sqlite, Entity Framework і Angular, з метою надання рішення, що є масштабованим, безпечним і зручним для користувачів.

Для досягнення цієї мети, основні завдання, які мають бути вирішені включають:

- Проектування і реалізація серверної частини яка обробляє запити від користувачів і взаємодіє з хмарним сховищем для зберігання файлів;
- Використання Sqlite для зберігання метаданих файлів, та інформації про користувачів;
- Розробка клієнтської частини, що дозволяє користувачам легко завантажувати свої файли;
- Тестування і оцінка ефективності програмного забезпечення.

Таким чином, кінцева мета роботи - створити працююче, зручне в користуванні і безпечне програмне забезпечення для віддаленого зберігання інформації, яке може бути адаптоване до потреб різних користувачів і організацій.

РОЗДІЛ 1 ТЕОРЕТИЧНА ЧАСТИНА

1.1 Огляд існуючих методів дистанційного збереження інформації

У контексті швидкого розвитку технологій та збільшення обсягу генерованих даних, методи дистанційного зберігання інформації стали ключовими інструментами для організацій різних рівнів. Різноманітність цих методів демонструє широкий спектр використання, від особистих потреб до корпоративних рішень. Перелічимо декілька основних методів дистанційного збереження інформації:

- хмарні сховища;
- FTP/SFTP сервери;
- хмарні бази даних;
- хмарні системи управління контентом;
- P2P файлообмінні мережі;
- блокчейн-технології.

Хмарні сховища це, можливо, найвідоміший метод дистанційного зберігання інформації, який включає сервіси, такі як Google Drive, Dropbox, Microsoft OneDrive. Ці сервіси надають прості у використанні інтерфейси та дозволяють користувачам зберігати документи, фотографії, відео та інші форми даних в хмарі. Користувачі можуть легко доступатися до своїх файлів з будь-якого місця, де є доступ до Інтернету, що робить ці сервіси дуже зручними.

FTP і SFTP - це стандартні протоколи для передачі файлів між системами по мережі. Вони використовуються для віддаленого зберігання інформації на серверах, доступ до яких здійснюється через спеціальне програмне забезпечення або через вбудовані засоби операційних систем.

Хмарні бази даних це бази даних, що хостуються на віддалених серверах і доступні через Інтернет. Такі бази даних, як Google Firebase, Amazon RDS, Microsoft Azure SQL Database і MongoDB Atlas, дозволяють зберігати структуровані та неструктуровані дані та доступатися до них з будь-якої точки світу.

Хмарні системи управління контентом це системи, що дозволяють користувачам створювати, редагувати та публікувати контент через веб-інтерфейс. Приклади включають WordPress, Joomla, Drupal та інші. Хоча вони переважно використовуються для управління веб-контентом, вони також можуть бути використані для зберігання різних типів інформації.

Прикладом P2P системи може бути BitTorrent, де файли зберігаються на різних вузлах мережі і можуть бути доступними для інших користувачів.

Хоча блокчейн-технології переважно асоціюються з криптовалютами, блокчейн може бути використаний для зберігання будь-якого типу даних. Децентралізований характер блокчейну робить його особливо привабливим для деяких застосувань.

Оглядаючи ці методи, важливо зазначити, що вибір конкретного методу великою мірою залежить від специфічних потреб користувача або організації. Наприклад, хмарні сховища можуть бути ідеальним вибором для особистого використання або для малого бізнесу, тоді як великі корпорації можуть вимагати більш складних рішень, таких як власні хмарні сервери або спеціалізовані бази даних.

1.2 Аналіз проблем і викликів дистанційного збереження інформації

1.2.1 Безпека даних

Забезпечення безпеки даних є надзвичайно важливим аспектом дистанційного збереження інформації. Існує кілька варіантів та стратегій, які можуть бути використані для ефективного вирішення цієї проблеми. Ось декілька з них:

- Аутентифікація та авторизація;
- Шифрування даних;
- Захист від зловмисників;
- Резервне копіювання та відновлення;
- Моніторинг та аудит безпеки.

Використання механізмів аутентифікації та авторизації є ключовим для забезпечення безпеки даних. Встановлення ідентичності користувачів та надання їм відповідних привілеїв та доступу до даних дозволяє уникнути несанкціонованого доступу до інформації. Використання сильних паролів, двофакторної аутентифікації та інших методів аутентифікації може додатково підвищити рівень безпеки.

Застосування шифрування даних є ефективним способом захисту конфіденційності інформації. Шифрування може використовуватися для захисту даних під час їх зберігання та передачі. Використання сильних шифрів та криптографічних алгоритмів забезпечує захист даних від несанкціонованого доступу.

Важливо враховувати потенційні загрози, які можуть бути спрямовані на систему дистанційного збереження інформації. Застосування механізмів захисту від зловмисників, таких як мережеві брандмауери, системи виявлення вторгнень

(IDS), захист від відмови в обслуговуванні (DDoS), може допомогти запобігти атакам та зберегти безпеку даних.

Резервне копіювання даних та механізми відновлення є важливими компонентами безпеки даних. Регулярне створення резервних копій даних та забезпечення можливості відновлення в разі випадку втрати або пошкодження даних допомагає зберегти їх цілісність та доступність.

Ці варіанти сприяють вирішенню проблеми безпеки даних у системі дистанційного збереження інформації. Їх комбінація та використання відповідних методів та технологій допомагають створити надійну та безпечну систему збереження даних.

Встановлення механізмів моніторингу та аудиту безпеки дозволяє виявляти потенційні загрози та вразливості системи. Аналіз логів, моніторинг мережевої активності та виявлення аномальних патернів допомагають вчасно виявляти та реагувати на можливі атаки чи порушення безпеки.

1.2.2 Маштабованість

Масштабування є важливим аспектом при розробці системи дистанційного збереження інформації. Якщо система не здатна ефективно масштабуватися під зростаюче навантаження та обсяг даних, це може призвести до падіння продуктивності та несправностей. Ось декілька варіантів вирішення проблеми масштабованості:

- Розподілена архітектура;
- Горизонтальне масштабування;
- Вертикальне масштабування;
- Кешування даних.

Використання розподіленої архітектури дозволяє розділити навантаження між різними вузлами системи. Це означає, що завдання можуть бути розподілені між багатьма серверами, що дозволяє забезпечити паралельну обробку та збільшити пропускну здатність системи.

Горизонтальне масштабування полягає в додаванні нових серверів або вузлів до системи для розподілу навантаження. Це дозволяє збільшити потужність обробки та швидкодію системи за рахунок розширення ресурсів.

Вертикальне масштабування полягає в збільшенні ресурсів (наприклад, процесора, пам'яті) на існуючих серверах. Це дозволяє системі обробляти більше навантаження без необхідності додавання нових серверів.

Використання кешування даних допомагає знизити навантаження на базу даних та покращити швидкодію системи. Кешування може бути застосоване до часто використовуваних даних або результатів обчислень, що дозволяє забезпечити швидкий доступ до цих даних без необхідності повторних обчислень або запитів до бази даних.

Ці варіанти допомагають вирішити проблему масштабованості у системі дистанційного збереження інформації. Їх використання дозволяє забезпечити ефективну обробку даних та зростання системи під зростаючі потреби користувачів.

1.3 Аналіз технологій, використаних для розробки додатку

Розробка сучасного програмного забезпечення вимагає використання найновіших технологій, які можуть забезпечити надійність, безпеку, продуктивність і зручність використання. У наступному аналізі розглянемо п'ять ключових технологій, які були використані в цій роботі:

- ASP.NET Core;

- AWS S3;
- SQLite;
- Entity Framework;
- Angular.

ASP.NET Core - це крос-платформний, високопродуктивний фреймворк для розробки веб-додатків, створений Microsoft. Він включає в себе набір технологій, що дозволяють швидко розробляти веб-додатки, які можуть масштабуватися для великих навантажень. ASP.NET Core використовує модель MVC, що допомагає структурувати код і забезпечує чітке розмежування між логікою додатку, відображенням даних та обробкою дій користувача. В ASP.NET Core також вбудована підтримка безпеки, включаючи автентифікацію та авторизацію.

Amazon Simple Storage Service - це об'єктне сховище від Amazon Web Services, призначене для зберігання і отримання будь-якої кількості даних в будь-який час. S3 надає надійне, масштабоване та доступне сховище даних. Його використовують компанії різного розміру для зберігання веб-сайтів, мобільних додатків, резервного копіювання та відновлення, архівування, корпоративних програм, IoT-пристроїв та багатьох інших.

SQLite - це вбудована база даних SQL, що не вимагає окремого сервера. Вона надає повноцінну підтримку SQL і може обробляти бази даних будь-якого розміру. SQLite особливо корисна для розробки, тестування та виробництва додатків з невеликими до середніх обсягами даних, де встановлення та управління повноцінним сервером баз даних можуть бути надлишковими.

Entity Framework (EF) - це ORM фреймворк від Microsoft, який дозволяє розробникам взаємодіяти з базою даних за допомогою .NET об'єктів, а не SQL запитів. EF допомагає розробникам швидко розробляти додатки, що працюють з

базами даних, та автоматизує рутинні завдання, такі як створення SQL запитів для зчитування, вставки, оновлення та видалення даних.

Angular - це популярний крос-платформний фреймворк від Google для розробки веб-додатків. Він використовує TypeScript як основну мову програмування і надає ряд потужних функцій, таких як двосторонній data-binding, dependency injection, модульність, і багато інших. Angular також надає структурований підхід до розробки веб-додатків, що допомагає підтримувати і масштабувати великі проекти.

Ці технології були обрані для цієї роботи через їх надійність, продуктивність, підтримку і широке прийняття в індустрії програмного забезпечення. Кожна з них відіграє важливу роль в створенні цілісного рішення для дистанційного зберігання інформації.

РОЗДІЛ 2 ПРАКТИЧНА ЧАСТИНА

2.1 Проектування архітектури програмного забезпечення

В основі нашої системи лежить трикомпонентна архітектура:

- серверна частина;
- база даних;
- клієнтська частина.

Серверна частина, реалізована на основі ASP.NET Core, служить основою для нашої системи. Використовуючи принципи Clean Architecture, ми забезпечуємо гнучкість і легкість супроводу коду. Зокрема, ця архітектура передбачає розділення бізнес-логіки і інфраструктурного коду, що дозволяє легко модифікувати систему і адаптувати її до змінюваних вимог бізнесу. В рамках серверної частини ми обробляємо HTTP запити від клієнтської частини, а також взаємодіємо з AWS S3 і SQLite базою даних. Ключовою складовою цього процесу є використання AWS S3 для зберігання файлів, що завантажуються користувачами. Після того, як файл було завантажено на сервер, ми пересилаємо його до AWS S3 і зберігаємо отриманий від AWS S3 унікальний шлях до файлу.

База даних, створена за допомогою SQLite, служить для зберігання інформації про користувачів та файли, які вони завантажують. Проект використовує Entity Framework для взаємодії з базою даних, що спрощує роботу з даними і забезпечує високий рівень абстракції від низькорівневих деталей роботи з базою даних. Коли користувач завантажує файл, система зберігає інформацію про цей файл в базі даних, включаючи унікальний шлях до файлу в AWS S3, час завантаження, дані про власника файлу тощо. Це дозволяє нам швидко отримувати необхідну інформацію про файли і користувачів, коли це потрібно.

Клієнтська частина, реалізована на Angular, відповідає за інтерфейс користувача. Вона дозволяє користувачам взаємодіяти з системою, включаючи завантаження файлів, перегляд списку завантажених файлів, авторизацію і так далі. В процесі взаємодії з користувачем, клієнтська частина формує HTTP запити, що відправляються на серверну частину. Вона також обробляє відповіді від сервера і представляє отримані дані користувачу в зручному для сприйняття вигляді.

Разом ці три компоненти формують гнучку і надійну систему, яка може бути легко модифікована і масштабована для задоволення змінюваних потреб бізнесу.

У процесі розробки серверної частини даного програмного забезпечення було обрано "Clean Architecture", що створена Робертом Мартіном [1]. Ця архітектура, яку визнано у всьому світі, дозволяє створювати програмне забезпечення, що легко супроводжується, є надійним, тестованим і незалежним від зовнішніх систем. Clean Architecture розділяє програмне забезпечення на чотири рівні:

- Рівень сутностей (Entities): Цей рівень містить бізнес-правила програмного забезпечення. Сутності - це набір об'єктів, що відображають бізнес-моделі, наприклад, файл, користувач тощо. Вони незалежні від конкретних деталей виконання, таких як бази даних чи веб-інтерфейси;

- Рівень використання (Use Cases): На цьому рівні знаходяться конкретні бізнес-правила програмного забезпечення, такі як операции збереження файлів, аутентифікації користувачів та ін. Ці правила описують, що саме робить програмне забезпечення;

- Рівень інтерфейсу (Interface Adapters): Цей рівень включає код, який перетворює дані з формату, придатного для використання, у формат,

придатний для зовнішніх агентів, таких як бази даних, веб-сервери, консолі тощо. Тут знаходяться контролери, презентери, шлюзи та інші адаптери;

- Рівень фреймворків та драйверів (Frameworks and Drivers): Це найбільш зовнішній рівень в архітектурі. Він включає деталі, що специфічні для конкретних технологій, які використовуються, такі як бази даних, веб-сервери та інші системи;

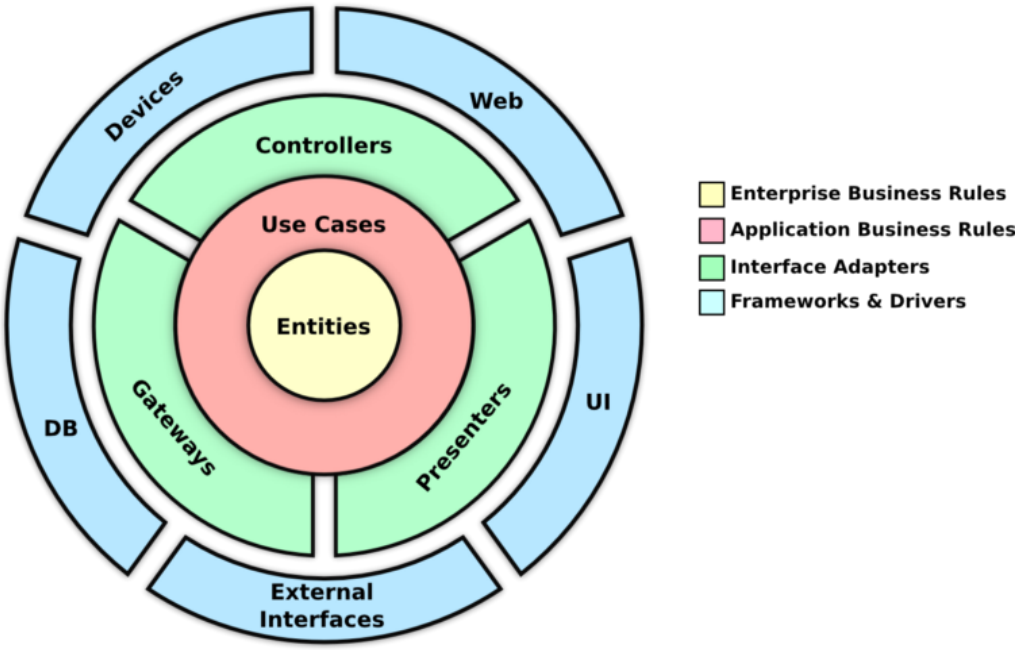

Рисунок 1 - Чиста архітектура (Clean Architecture)

Ця архітектура дозволяє забезпечити гнучкість, легкість в тестуванні та незалежність від окремих технологій [2]. Наприклад, ми можемо легко замінити базу даних або змінити фреймворк веб-додатку без зміни бізнес-логіки програми.

Окрім описаної вище архітектури, існують різні альтернативні варіанти архітектури для подібних систем. Вибір конкретної архітектури залежить від різних факторів, включаючи специфічні вимоги бізнесу, технічні обмеження, доступні ресурси і так далі. Ось декілька альтернативних варіантів:

- Монолітна архітектура;
- Мікросервісна архітектура;
- Serverless архітектура;
- P2P (Peer-to-peer) архітектура;

Кожен з цих варіантів має свої переваги та недоліки, і вибір між ними залежить від конкретних вимог до проекту. У нашому випадку, ми вибрали трирівневу архітектуру, оскільки вона забезпечує гнучкість, надійність, і легкість управління та розширення.

2.2 Розробка серверної частини

ASP.NET Core - це високопродуктивний, універсальний фреймворк від Microsoft, який використовується для побудови веб-додатків, веб-служб і мікросервісів [3]. Він підтримує різні платформи, включаючи Windows, Linux і MacOS, що робить його відмінним вибором для розробки веб-сервера в нашому проекті.

ASP.NET Core оптимізований для високої продуктивності. Сервер, що використовує цей фреймворк, може обробляти велику кількість запитів в секунду, що важливо для системи зберігання даних. Також фреймворк є модульним, що означає, що можна обрати тільки ті компоненти, потрібні для додатка. Це може допомогти зменшити розміру додатку та поліпшити його продуктивність. Фреймворк містить ряд вбудованих функцій безпеки, таких як захист від міжсайтового скриптингу (XSS), міжсайтового підроблення запиту (CSRF), SQL-ін'єкцій і так далі.

У контексті проекту, було використовуємо ASP.NET Core для побудови веб-сервера, що обробляє HTTP запити від клієнтської частини, керує завантаженням файлів на AWS S3, а також обробляє дані користувачів і файлів в SQLite базі даних.

У процесі розробки веб-сервера на ASP.NET Core, спочатку було зосереджено на реалізації автентифікації та реєстрації користувачів. Для цього було створено таблицю користувачів в SQLite базі даних з наступними полями:

- Id: Це унікальний ідентифікатор користувача;
- Login: Це ім'я користувача, яке використовується для входу в систему;
- PasswordSalt: Сіль, яка використовується для хешування пароля. Сіль - це випадкові дані, які використовуються як додатковий вхід при хеш-функції для запобігання атак на основі таблиць "радуги";
- PasswordHash: Це хеш-значення пароля, яке було згенеровано з поєднання пароля та солі;
- MaxBytesAvailable: Це максимальний обсяг даних, який користувач може зберігати;
- BytesUsed: Це поточний обсяг даних, який користувач вже використав.

Для забезпечення безпеки паролів наших користувачів ми використовуємо хешування паролів за допомогою алгоритму PBKDF2 який призначений для створення криптографічних ключів з паролей (рисунок 2).

Алгоритм PBKDF2 використовує ітераційний процес, щоб створити ключ з пароля. Було використано сіль - рядок даних, який передається хеш-функції разом з паролем, щоб забезпечити, що кожен хеш є унікальним, навіть якщо два паролі однакові. Це допомагає забезпечити безпеку наших користувачів, навіть якщо база даних компрометується.

```
public byte[] GenerateSalt()
{
    var salt = new byte[16];
    RandomNumberGenerator.Create().GetBytes(salt);
    return salt;
}

public byte[] EncodePassword(string password, byte[] passwordSalt) ⇒
    KeyDerivation.Pbkdf2(
        password,
        passwordSalt,
        KeyDerivationPrf.HMACSHA256,
        IterationCount,
        SubkeyBitCount / BitCount);
```

Рисунок 2 - Функції генерації солі та хешування пароля

В процесі розробки серверної частини програмного забезпечення було вирішено створити спеціальну таблицю в базі даних для зберігання інформації про файли, які завантажуються користувачами.

Таблиця для файлів має наступні поля:

- Id - унікальний ідентифікатор файлу, який генерується автоматично при завантаженні файлу. Це поле служить первинним ключем таблиці;
- Path - шлях до файлу в хмарному сховищі AWS S3. Це поле дозволяє легко знайти місце зберігання файлу в хмарному сховищі;
- SizeInBytes - розмір файлу в байтах. Це поле допомагає контролювати використання ресурсів хмарного сховища користувачем;
- UploadedAt - дата та час завантаження файлу. Це поле дозволяє відслідковувати історію використання сховища користувачем;

- UserId - ідентифікатор користувача, який завантажив файл. Це поле служить зовнішнім ключем, який зв'язує таблицю файлів з таблицею користувачів.

Таблиця файлів дозволяє зберігати важливі метадані про файли, які завантажуються користувачами, і використовувати ці дані для контролю доступу до файлів, відстеження використання сховища користувачами, реалізації функціональності пошуку файлів та інших можливостей.

Entity Framework - це технологія доступу до даних, яка дозволяє .NET розробникам працювати з даними за допомогою об'єктів домену, не вимагаючи занадто багато уваги до бази даних, з якої ці дані походять [4]. Фреймворк працює за принципом ORM - технології, яка дозволяє працювати з даними на рівні об'єктів, не вдаючись в подробиці роботи з базами даних.

У контексті нашого проекту EF використовується для взаємодії з базою даних SQLite. Ми використовуємо моделі, що представляють наші таблиці, для виконання CRUD операцій. EF володіє такими основними перевагами, як:

- автоматичне створення SQL-запитів, що знижує ризик помилок при ручному формуванні запитів і дозволяє розробникам зосередитися на бізнес-логіці;
- оптимізація запитів до бази даних.
- підтримка LINQ (Language Integrated Query), що дозволяє робити запити до бази даних прямо з C# коду;
- розгортання і міграції бази даних, що полегшує управління структурою бази даних;

Звісно, є й альтернативи EF, такі як Dapper, NHibernate, ADO.NET.

Dapper, наприклад, пропонує більше контролю над SQL-запитами і має вищу продуктивність, але він не надає такого ж рівня абстракції, який дає EF, і вимагає більше ручного кодування.

NHibernate є дуже потужним і гнучким ORM, але має високий поріг входження і може бути занадто складним для невеликих проектів.

ADO.NET - це низькорівнева технологія доступу до даних, яка дає максимальний контроль над процесом, але вимагає більшої кількості ручного кодування та глибоких знань SQL.

Вибір між цими технологіями залежить від специфіки проекту, його вимог до продуктивності, складності бізнес-логіки і рівня впевненості розробника в роботі з SQL. У проекті було вирішено використовувати Entity Framework через його гнучкість, легкість використання і хорошу інтеграцію з іншими технологіями .NET, такими як ASP.NET Core. За його допомогою було написано керування файлами та користувачами в базі даних (рисунок 3).

```
public async Task<IEnumerable<FileRecordResponse>> GetFileRecords(
    CancellationToken cancellationToken = default)
{
    return await _dbContext.Set<FileRecord>()
        .Where(fr => fr.UserId == _identities.CurrentUser!.Id)
        .Select(fr => new FileRecordResponse(fr.Path, fr.SizeInBytes, fr.UploadedAt))
        .ToListAsync(cancellationToken);
}
```

Рисунок 3 - Функція отримання назв файлів користувача за допомогою EF

При взаємодії між серверною та клієнтською частинами програмного забезпечення використовується RESTful API, який є стандартним способом обміну даними між клієнтом і сервером в сучасних веб-застосунках. RESTful

API використовує HTTP методи (GET, POST, PUT, DELETE тощо) для взаємодії з ресурсами, які представлені URL-адресами.

У проекті серверна частина на основі ASP.NET Core використовує RESTful API для обробки запитів від клієнтської частини. Було створено наступні кінцеві точки:

- POST /api/users - використовується для реєстрації нового користувача. Приймає JSON об'єкт з даними користувача (ім'я користувача, пароль) і, якщо дані валідні, створює нового користувача в базі даних;
- POST /api/authenticate - використовується для входу користувача в систему. Приймає JSON об'єкт з ім'ям користувача та паролем, перевіряє ці дані та, якщо вони вірні, повертає токен доступу;
- GET /api/files - повертає список файлів, які належать поточному користувачу;
- POST /api/files/{path} - приймає файл, який потрібно завантажити, зберігає його в базі даних та в AWS S3 за вказаним шляхом;
- DELETE /api/files/{path} - ця кінцева точка видаляє файл за вказаним шляхом;

Цей підхід дозволяє створити чітку структуру API, що полегшує розробку та тестування нашого додатку. Крім того, він дозволяє клієнтській частині зрозуміти, як правильно взаємодіяти з сервером, і, як результат, сприяє створенню стабільної та надійної системи.

Автентифікація користувачів на серверній частині програмного забезпечення є критичною складовою для забезпечення безпеки даних. В проекті використовується аутентифікація на основі JWT для визначення того, ким є користувач, який намагається виконати дію.

JWT є компактним і безпечним способом передачі інформації між двома сторонами в форматі JSON. Ця інформація може бути підтверджена і довірена, оскільки вона цифрово-підписана. JWT може кодувати інформацію, що стосується автентифікації користувача, що робить його особливо корисним для автентифікації користувачів в контексті веб-застосунків.

У побудованій системі, після того, як користувач успішно ввійшов у систему, він отримує JWT, який містить у собі інформацію про його ідентичність. Цей токен потім використовується для авторизації користувача при спробі виконання подальших дій, таких як завантаження або видалення файлу.

JWT використовується у заголовку 'Authorization' HTTP-запитів. Це означає, що при кожному запиті до сервера, клієнт повинен включати JWT в заголовок 'Authorization' для того, щоб сервер міг визначити ідентичність користувача. Наприклад, заголовок 'Authorization' може виглядати так: 'Authorization: Bearer {токен}'.

Якщо JWT відсутній або недійсний, сервер відхиляє запит з повідомленням про помилку, що забезпечує додатковий рівень безпеки.

Така система аутентифікації не тільки надійна, але і масштабована, тому що JWT не вимагає зберігання на сервері, що дозволяє обслуговувати велику кількість користувачів.

2.3 Збереження файлів

Amazon S3 - це одна з найпопулярніших служб облачного зберігання від Amazon Web Services. Ця служба забезпечує безпечне, надійне і гнучке збереження даних в Інтернеті. Переваги використання AWS S3:

- надійність і безпека;
- масштабування;

- широкі можливості інтеграції;

Amazon S3 гарантує 99.999999999% (11 дев'яток) довгострокової надійності зберігання даних [5]. Крім того, S3 надає розширені можливості безпеки, такі як автоматичне шифрування даних і підтримка різних моделей доступу, включаючи політику AWS Identity and Access Management (IAM) для контролю доступу на рівні об'єкта.

AWS S3 дозволяє зберігати необмежену кількість даних, від декількох байтів до петабайт, що робить його відмінним рішенням для додатків будь-якого розміру.

S3 легко інтегрується з іншими службами AWS, такими як Amazon CloudFront для розповсюдження контенту, AWS Lambda для автоматизації завдань на основі подій, а також AWS Glacier для архівації даних.

У контексті проекту, було використано Amazon S3 для збереження файлів користувачів. Завдяки високій надійності, безпеці і масштабованості S3, можна гарантувати, що файли користувачів будуть завжди доступні і безпечні.

Щодо автентифікації в AWS, найчастіше використовуються IAM ролі та політики. IAM дозволяє управляти доступом до служб та ресурсів AWS. Система дозволяє створити IAM користувачів для осіб та додатків, що потребують доступу до S3 bucket. Крім того, можна використовувати IAM ролі, які дозволяють делегувати дозволи на різні служби та користувачів AWS без необхідності передавати постійні AWS ключі доступу.

Як альтернативи AWS S3 можна розглянути Google Cloud Storage від Google Cloud Platform або Azure Blob Storage від Microsoft Azure. Обидві ці служби також надають високу надійність, безпеку і масштабованість для збереження даних в хмарах. Вибір між ними залежить від різних факторів, таких

як вимоги до вартості, локації дата-центрів, наявності регіонального зберігання даних і так далі.

Спочатку було створено S3 бакет через AWS Management Console. Бакет у S3 подібний до директорії, в яку можна завантажувати файли (відомі як об'єкти). При створенні бакету потрібно вказати унікальне ім'я, яке буде використовуватися для ідентифікації бакету.

Однією з особливостей Amazon S3 є те, що він не підтримує рівнів вкладеності, як традиційна файлова система. Однак, можна симулювати ієрархічну структуру, використовуючи символи '/' в іменах об'єктів. Таким чином, можна створити структуру папок всередині S3 бакету.

На проекті було вирішено створити окрему директорію для кожного користувача з назвою його id. Це допомогло забезпечити ізольованість між даними різних користувачів та гарантувати, що користувач може мати доступ лише до своїх власних файлів.

Іншою важливою особливістю S3 є її політика доступу до бакетів і об'єктів. За замовчуванням, всі нові бакети та об'єкти є приватними. Ви можете управляти доступом до бакетів і об'єктів, використовуючи ACL (Access Control Lists) та політику бакетів. Політика безпеки забезпечує, що лише ваш серверний застосунок може завантажувати, отримувати і видаляти файли в S3.

Інтеграція ASP.NET Core з Amazon S3 відбувається за допомогою AWS SDK для .NET. AWS SDK надає .NET-інтерфейси для взаємодії з AWS-службами, включаючи Amazon S3. Наступні кроки описують загальний процес інтеграції:

Перш за все, вам потрібно встановити AWS SDK для .NET у вашому ASP.NET Core проекті. Це можна зробити за допомогою NuGet, менеджера

пакетів для .NET. AWS SDK включає в себе бібліотеки для взаємодії з Amazon S3.

Налаштування доступу до AWS: Для використання AWS-служб, вам потрібно налаштувати облікові дані AWS в вашому проекті. Ви можете зберігати ці облікові дані в конфігураційному файлі або використовувати AWS Identity and Access Management (IAM) для делегування прав доступу.

Використання Amazon S3 API: Після налаштування доступу, ви можете використовувати API Amazon S3 для виконання операцій з S3, таких як завантаження файлів, отримання файлів, видалення файлів і так далі. Зазвичай, це включає створення екземпляру класу AmazonS3Client та використання його методів для взаємодії з S3.

У контексті проекту, було використано AWS SDK для завантаження файлів користувачів на Amazon S3 (рисунок 4), а також для отримання посилань на ці файли, які потім надаються користувачам для доступу до їх файлів.

```
public async Task UploadFile(FileObject file, CancellationToken cancellationToken = default)
{
    using var fileTransferUtility = new TransferUtility(_s3Client);

    var uploadRequest = new TransferUtilityUploadRequest
    {
        InputStream = file.File,
        Key = file.Path,
        BucketName = _s3Options.BucketName,
        CannedACL = S3CannedACL.NoACL,
    };

    try
    {
        await fileTransferUtility.UploadAsync(uploadRequest, cancellationToken);
    }
    catch (AmazonS3Exception e)
    {
        _logger.LogError("Error uploading file to S3: {EMessage}", e.Message);
        throw;
    }
}
```

Рисунок 4 - Функція завантаження файлу до AWS S3

2.4 Розробка клієнтської частини

Клієнтська частина додатку була розроблена за допомогою фреймворку Angular, відомого своєю ефективністю, масштабованістю та модульною архітектурою. Angular допомагає в розробці якісних, розширюваних односторінкових веб-додатків, використовуючи TypeScript та HTML. Це дозволяє легко організувати структуру коду і управляти його.

В основі Angular лежить концепція компонентів. Компоненти представляють окремі частини інтерфейсу користувача, що дозволяє легко виконувати заміну, додавання або видалення елементів інтерфейсу без втрати цілісності [6]. Для реалізації цієї концепції ми створили ряд компонентів, таких як компонент авторизації, компонент завантаження файлів, компонент перегляду файлів тощо.

Angular також надає потужну систему маршрутизації, що дозволяє нам встановлювати різні шляхи до наших компонентів і контролювати, як користувач переходить між ними. Наприклад, ми можемо встановити, що користувач буде перенаправлений на сторінку входу, якщо він не авторизований, або на головну сторінку, якщо він вже ввійшов в систему.

Для розробки інтерфейсу користувача ми використовували Bootstrap - вільно доступну і відкриту бібліотеку для розробки з відгуком. Bootstrap надає великий набір готових до використання компонентів, стилів і шаблонів, які допомагають швидко та ефективно створювати привабливі веб-сторінки.

Bootstrap використовує систему сіток для легкого та гнучкого макетування сторінок, що забезпечує їх адаптивність до різних розмірів екранів. Крім того, він надає набір CSS класів для стилізації різноманітних елементів інтерфейсу, таких як кнопки, форми, навігаційні панелі, модальні вікна і багато іншого.

У проекті буловикористано Bootstrap для створення адаптивного, сучасного інтерфейсу, який би був зручний для користувачів. До прикладу, ми використали класи Bootstrap для створення навігаційної панелі, форми входу, списку файлів тощо.

Однак, Bootstrap не обмежується лише готовими рішеннями. Він дозволяє легко розширювати його стандартні компоненти або створювати свої власні, використовуючи SCSS, що дає великі можливості для кастомізації.

Загалом, комбінація Angular і Bootstrap дозволила швидко і ефективно створити зручний та відповідний веб-інтерфейс для нашого додатку.

В процесі розробки клієнтської частини програмного забезпечення було вирішено почати з компонентів реєстрації та авторизації (рисунок 5). Це стало першим кроком у побудові інтерактивного інтерфейсу користувача.

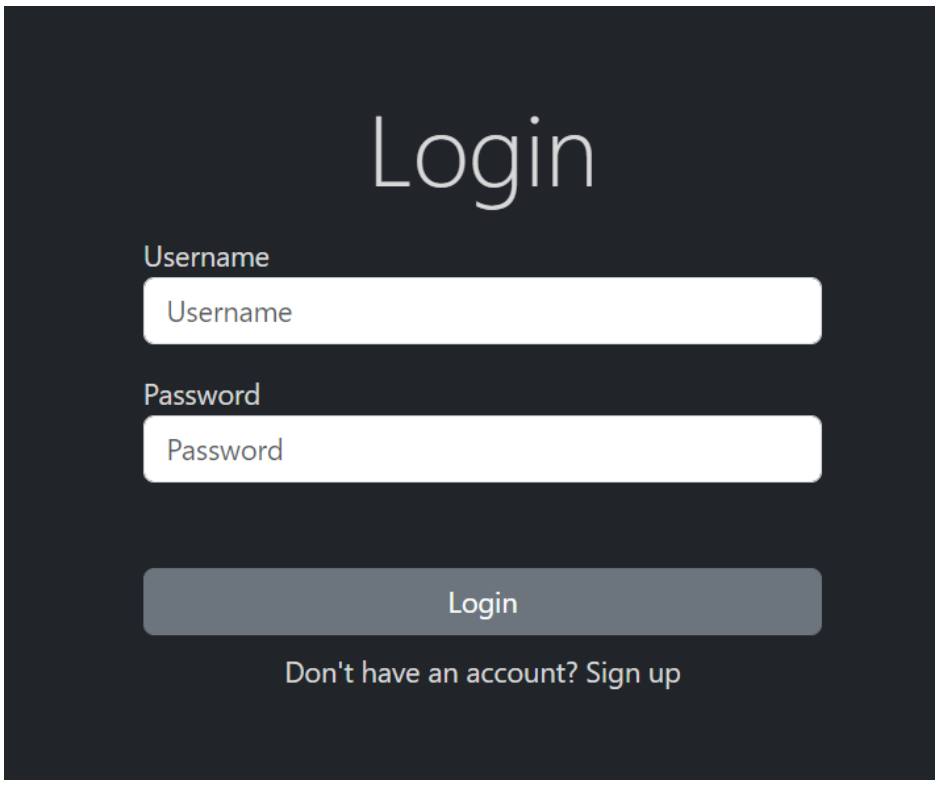

Рисунок 5 – Вікно реєстрації користувача

Компонент реєстрації було розроблено для того, щоб користувач міг створити обліковий запис. Він складається з форми, яка збирає необхідну інформацію від користувача, таку як ім'я користувача, адресу електронної пошти та пароль. Після надсилання форми, дані перевіряються, а потім відправляються на сервер для подальшої обробки.

Компонент авторизації, з іншої сторони, дозволяє користувачам, які вже зареєстровані в системі, увійти до свого облікового запису. Він також містить форму, де користувач може ввести свій логін та пароль. Після надсилання форми, дані також перевіряються і, якщо вони відповідають існуючому обліковому запису, користувачу надається доступ до системи.

Обидва ці компоненти були розроблені з використанням технології двосторонньої прив'язки даних, яка є однією з ключових особливостей Angular.

Ця технологія дозволяє створити пряме зв'язування між моделлю даних (в нашому випадку, даними форми) та елементами інтерфейсу користувача. Таким чином, будь-які зміни, які зробив користувач у формі, автоматично відображаються в моделі даних, і навпаки.

Це не тільки спрощує процес розробки, але також дозволяє створити більш інтуїтивно зрозумілий і зручний для користувача інтерфейс. Наприклад, якщо користувач введе неправильні дані, ми можемо негайно показати повідомлення про помилку без необхідності відправляти форму.

Наступним кроком було створено компонент для завантаження та перегляду директорій та файлів користувача (рисунок 6).

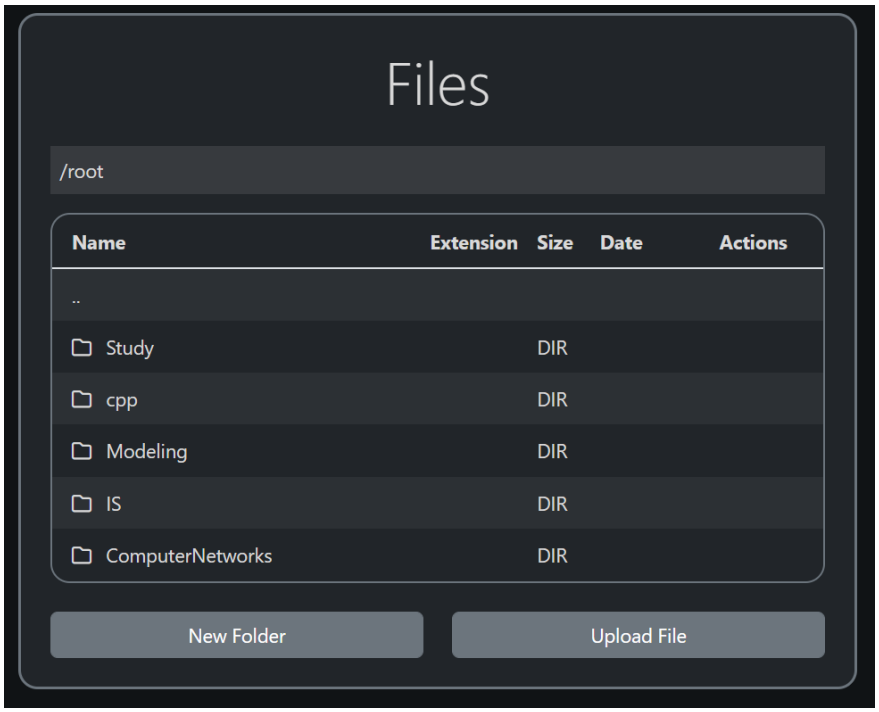

Рисунок 6 – Вікно перегляду файлів та директорій

Також, двостороння прив'язка даних дозволяє нам легко імплементувати функції, такі як "запам'ятати мене", яка зберігає дані користувача в локальному

сховищі браузера, щоб автоматично заповнювати форму входу при наступних відвідуваннях.

Ці компоненти, разом з іншими частинами нашого додатку, були створені з метою забезпечити зручність і ефективність використання нашої системи.

2.5 Тестування програмного забезпечення

Тестування є важливим етапом розробки програмного забезпечення, який допомагає виявити помилки та недоліки у системі, перш ніж вона буде запущена. Процес тестування допомагає гарантувати, що програмне забезпечення відповідає вимогам та очікуванням користувача [7]. У контексті нашої системи дистанційного збереження інформації, тестування включає в себе перевірку роботи всіх компонентів: серверної частини, клієнтської частини та сервісу AWS S3.

Тестування серверної частини здійснювалося за допомогою юніт-тестів та інтеграційних тестів. Юніт-тестування зосереджується на індивідуальних модулях програмного забезпечення, перевіряючи, чи правильно вони виконують свої функції. Наприклад, були створені юніт-тести для перевірки правильної роботи хешування паролів користувачів (рисунок 7). Інтеграційні тести допомагають перевірити, чи правильно модулі взаємодіють один з одним. В нашому випадку це може бути тест, який перевіряє, чи правильно сервер взаємодіє з AWS S3 при завантаженні файлу.

```
[Fact]
public void Encode_ShouldReturnHashedPassword()
{
    // Arrange
    var faker = new Faker();
    var password = faker.Random.String2(10);
    var passwordBytes = Encoding.UTF8.GetBytes(password);
    var salt = faker.Random.Bytes(16);
    var passwordHashService = new PasswordHashService();

    // Act
    var actual = passwordHashService.EncodePassword(password, salt);

    // Assert
    actual.Should().NotBeEquivalentTo(passwordBytes);
}
```

Рисунок 7 – Тест хешування пароля

Тестування клієнтської частини можна здійснити за допомогою фреймворку Jasmine і Angular Testing Library щоб перевірити, чи правильно виконуються дії на стороні клієнта, такі як введення даних у форми, навігація між сторінками та відображення коректних даних від сервера.

Для AWS S3 тестування полягало в перевірці правильної взаємодії з API AWS, включаючи завантаження файлу, його видалення та отримання.

Нарешті, було проведено перевірку коректного відображення інтерфейсу в різних браузерах та на різних екранах.

Всі знайдені недоліки були виправлені в ході процесу тестування, що дозволило підвищити якість кінцевого продукту та забезпечити надійну та ефективну роботу системи дистанційного збереження інформації.

Юніт-тестування - це процес перевірки мінімальних використовуваних одиниць програмного забезпечення, як правило, окремих функцій або методів. Для юніт-тестування серверної частини програмного забезпечення було використано набір інструментів, включаючи xUnit, NSubstitute, Bogus та FluentAssertions.

xUnit - це бібліотека для створення юніт-тестів у .NET. Вона надає велику кількість функцій для написання та створення юніт-тестів, включаючи можливість організації тестів у групи, паралельного виконання тестів та декларативного задання вхідних даних.

NSubstitute - це інструмент для створення макетів (mocks) і замінників (stubs) в .NET. Він дозволяє нам контролювати поведінку об'єктів, що залежать від тестованого коду, тим самим дозволяючи нам зосередитися на тестуванні самого цільового коду. За допомогою NSubstitute ми могли емулювати взаємодію з базою даних і сервісом AWS S3, перевіряючи, що виклики до цих залежностей виконуються з правильними параметрами.

Bogus - це бібліотека для генерації довільних даних. Вона допомагає нам створювати випадкові, але вірогідні вхідні дані для наших юніт-тестів, що забезпечує більше сценаріїв використання.

FluentAssertions - це бібліотека для декларативного створення тверджень в юніт-тестах. Вона дозволяє описувати очікувані результати у вигляді, що є зрозумілим для людини, тобто "результат повинен бути таким-то". Це робить тести більш читабельними і легкими для розуміння.

РОЗДІЛ 3 ОЦІНКА ЕФЕКТИВНОСТІ РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Порівняння продуктивності з іншими існуючими рішеннями

При оцінці продуктивності програмного забезпечення для дистанційного збереження інформації важливо врахувати багато різних факторів. Це включає не тільки швидкість завантаження та вивантаження файлів, але й швидкість відповіді інтерфейсу, зручність використання, надійність, безпеку і багато інших аспектів.

У порівнянні з іншими існуючими рішеннями, такими як Dropbox, Google Drive, OneDrive і інші, розроблене програмне забезпечення володіє декількома перевагами:

- персоналізація;
- безпека;
- швидкість завантаження та вивантаження;
- інтеграція з іншими сервісами.

Унікальна особливість нашої системи полягає в тому, що вона може бути налаштована на потреби конкретного бізнесу чи організації.

Використовуючи інфраструктуру AWS і надійні технології автентифікації та шифрування, розроблена система забезпечує високий рівень безпеки збереження даних.

Багато комерційних систем зберігання файлів мають обмеження на швидкість завантаження або вивантаження. За допомогою AWS S3, наша система надає користувачам високу швидкість передачі даних, незалежно від розміру файлів.

Через використання AWS, наша система може легко інтегруватися з іншими сервісами AWS, що надає додаткову гнучкість при впровадженні системи.

У той же час, важливо врахувати деякі недоліки порівняно з великими комерційними провайдерами. Це включає потенційні додаткові витрати на інфраструктуру AWS, більшу складність управління інфраструктурою та потенційну необхідність власного персоналу для підтримки системи. Однак, ці виклики можуть бути компенсовані перевагами унікальності, гнучкості і контролю, які надає розроблена система.

В цілому, при порівнянні продуктивності нашої системи з іншими існуючими рішеннями важливо врахувати конкретні потреби та обставини. Для деяких організацій, особливо для тих, які потребують високого рівня контролю та налаштувань, наша система може бути найкращим варіантом.

3.2 Обговорення потенційних застосувань та переваг програмного забезпечення

Розроблене програмне забезпечення для дистанційного збереження інформації може мати широкий спектр потенційних застосувань та принести значні переваги різним типам користувачів і організацій. Ось кілька потенційних застосувань і переваг, які можуть бути отримані від цього програмного забезпечення:

- Бізнес-сектор: Для багатьох компаній збереження і обмін файлами є критично важливим аспектом ділових процесів. Наша система надає зручний та безпечний спосіб зберігання і обміну файлами між співробітниками та зовнішніми партнерами. Вона може бути використана для колективної роботи над проектами, збереження важливих документів, резервного копіювання даних та багато іншого.

- Освітні установи: В освітніх установах, де велика кількість документів, робіт студентів та інших матеріалів потребує збереження та обміну, наша система може бути важливим інструментом. Вона дозволяє студентам і викладачам зручно ділитись файлами, зберігати матеріали для вивчення та забезпечувати безпеку цінної інформації.

- Медичний сектор: В медичному секторі, де збереження та обмін медичною інформацією має вирішальне значення для надання якісної медичної допомоги, наша система може бути надійним інструментом. Лікарі можуть зручно зберігати та обмінюватись медичними даними, зображеннями, лабораторними результатами та іншими файлами, що покращує комунікацію та сприяє поліпшенню якості медичних послуг.

- Індивідуальні користувачі: Наша система також може бути корисною для індивідуальних користувачів, які бажають зберігати свої особисті файли в безпечному хмарному сховищі. Вона надає зручний спосіб резервного копіювання важливих даних, збереження фотографій, відео, документів та інших особистих файлів.

Переваги створеного програмного забезпечення включають:

- зручний інтерфейс: інтуїтивно зрозумілий та дружній для користувача інтерфейс, який спрощує роботу з файлами та доступ до них;

- безпека: забезпечення високого рівня безпеки файлів через шифрування та автентифікацію;

- масштабованість: можливість розширення системи залежно від зростаючих потреб користувачів та обсягу даних;

- надійність: забезпечення стабільної та надійної роботи системи, щоб забезпечити доступ до файлів в будь-який час.

Це лише деякі з потенційних застосувань і переваг, які можуть бути отримані від нашого програмного забезпечення для дистанційного збереження інформації. Конкретний вибір залежить від потреб і вимог кожного користувача або організації, але система пропонує широкий спектр можливостей для забезпечення ефективного та безпечного збереження файлів.

3.3 Способи поліпшення та подальшого розвитку програмного забезпечення

Програмне забезпечення для дистанційного збереження інформації завжди потребує постійного вдосконалення та розвитку для забезпечення задоволення потреб користувачів та відповідності новим технологічним та безпековим вимогам. Ось кілька способів поліпшення та подальшого розвитку програмного забезпечення:

- оптимізація продуктивності;
- розширення функціональності;
- покращення безпеки;
- підтримка нових технологій;
- збільшення масштабованості;
- зворотний зв'язок користувачів.

Продуктивність є однією з ключових характеристик програмного забезпечення. Шляхом аналізу та вдосконалення алгоритмів, оптимізації запитів до бази даних та використання кешування можна поліпшити швидкість роботи системи.

Програмне забезпечення може бути розширене додаванням нових функцій та можливостей, які забезпечать більш гнучкі та розширені можливості для користувачів. Наприклад, можна розглянути можливість додавання спільної

роботи над документами, можливостей синхронізації з різними пристроями, аналітичних звітів тощо.

Забезпечення високого рівня безпеки є критичним аспектом програмного забезпечення для збереження конфіденційної інформації користувачів. Можна розглянути вдосконалення механізмів автентифікації, шифрування даних, контролю доступу та моніторингу безпекових подій.

Технологічний прогрес постійно рухається вперед, і важливо враховувати нові технології та інструменти для поліпшення програмного забезпечення. Наприклад, можна розглянути використання більш ефективних алгоритмів шифрування, впровадження розподіленого сховища даних або інтеграцію з іншими хмарними сервісами.

Якщо програмне забезпечення має потенціал зростання кількості користувачів або обсягу даних, важливо забезпечити його масштабованість. Це може включати оптимізацію архітектури для розподіленого оброблення завдань, використання масивних обчислювальних ресурсів та горизонтального масштабування.

Отримання зворотного зв'язку від користувачів є важливим джерелом інформації для покращення програмного забезпечення. Враховуючи побажання та пропозиції користувачів, можна покращити відповідність програмного забезпечення їх потребам і запитам.

Підсумовуючи, поліпшення та подальший розвиток створеного додатку залежить від постійного аналізу вимог, технологічних тенденцій та відгуків користувачів. Зосередження на оптимізації продуктивності, розширенні функціональності, підвищенні безпеки, використанні нових технологій, масштабованості та зворотньому зв'язку користувачів допоможе забезпечити постійний прогрес та вдосконалення програмного забезпечення.

ВИСНОВОК

Було розроблено програмне забезпечення для дистанційного збереження інформації з використанням ASP.NET Core, AWS S3, SQLite, Entity Framework і Angular. Об'єктом роботи було створення ефективної та безпечної системи, яка забезпечує користувачам зручний доступ до збереження та обміну файлами в хмарному середовищі.

Під час розробки були використані передові технології та інструменти, що дозволило створити функціональне та надійне програмне забезпечення. Застосування чистої архітектури на серверній стороні дало можливість впорядкувати код і полегшити його тестування та розширення.

Результати роботи включають розробку серверної та клієнтської частин програмного забезпечення, створення бази даних для збереження метаданих файлів, налаштування дистанційного збереження файлів в AWS S3 та забезпечення безпеки даних через автентифікацію на основі JWT.

Тестування показало, що розроблене програмне забезпечення є ефективним, зручним у використанні та забезпечує високий рівень безпеки. Воно може бути застосоване в різних сферах, включаючи бізнес-сектор, освіту та особисте використання.

Для подальшого розвитку і поліпшення програмного забезпечення можуть бути виконані кроки, такі як оптимізація продуктивності, розширення функціональності, покращення інтерфейсу користувача та інтеграція з іншими сервісами.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Мартін Фаулер, The Clean Coder: A Code of Conduct for Professional Programmers. Prentice Hall. 2011. ISBN 0-13-708107-3. Переклад українською: Чиста архітектура: Мистецтво розроблення програмного забезпечення. «Ранок», Фабула. 2020. с. 368. ISBN 978-617-09-5286-8
2. Мартин Фаулер, Clean Code: A Handbook of Agile Software Craftsmanship. Prentice Hall PTR. 2008. ISBN 0-13-235088-2. Переклад українською: Чистий код: Створення і рефакторинг за допомогою agile. «Ранок», Фабула. 2020. с. 448. ISBN 978-617-09-5285-1
3. Microsoft Docs [Електронний ресурс] - https://docs.microsoft.com/
4. Julia Lerman, Programming Entity Framework, 1st Edition. 2009. — 832 с.
5. Amazon Web Services Documentation [Електронний ресурс] - https://docs.aws.amazon.com/
6. Адам Фримен, Angular для профессионалов 2020. — 800 с. ISBN – 978-5-4461-0451-2
7. Гойко Аджича, Девіда Еванса та Тома Родена, 50 quick ideas to improve your tests. ISBN 978-0201794298
8. Девід Томас, Ендрю Хант, The Pragmatic Programmer, 20th Anniversary .ISBN 9780135957059