Міністерство освіти і науки України
Криворізький національний університет
Факультет інформаційних технологій
Кафедра автоматизації, комп'ютерних наук і технологій

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти – магістр
за освітньо-професійною програмою
«Комп'ютерні науки»

зі спеціальності
122 – Комп'ютерні науки

тема роботи:
«Розробка веб-додатку для управління бібліотекою медіа файлів»

Виконав студент гр. КН-23м _____ Лінський М.П.
Керівник _____ Рубан С. А.
Нормоконтроль _____ Маринич І. А.
Завідувач кафедри _____ Рубан С. А.

Кривий Ріг – 2024

КРИВОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет: *інформаційних технологій*

Кафедра: *автоматизації, комп'ютерних наук і технологій*

Ступінь вищої освіти: *Магістр*

Спеціальність: *122 – Комп'ютерні науки*

ЗАТВЕРДЖУЮ

Зав. кафедри: *доц. Рубан С.А.*

_____

« *5* » *липня* 2024 р.

ЗАВДАННЯ

на кваліфікаційну роботу магістра

студентові групи КН-23м *Лінському Михайлу Павловичу*

1. Тема кваліфікаційної роботи: *«Розробка веб-додатку для управління бібліотекою медіа файлів»*

затверджено наказом по університету № *594с* від *04.07.2024 р.*

2. Термін здачі кваліфікаційної роботи: *01.12.2024 р.*

3. Склад кваліфікаційної роботи: *Пояснювальна записка обсягом 80с., додатки, презентація у Microsoft PowerPoint (13 слайдів) в електронному та друкованому вигляді*

4. Консультанти кваліфікаційної роботи:

Розділ 1-3 доц. Рубан С. А.

_____.

Нормоконтроль доц. Маринич І. А.

5. Календарний план:

№	Етапи роботи	Термін виконання
1	Вступ	10.07.24
2	Розділ 1	15.07.24
3	Розділ 2	18.08.24
4	Розділ 3	19.09.24
5	Висновки	15.10.24
6	Оформлення кваліфікаційної роботи	20.11.24
7	Підготовка презентації та графічного матеріалу	28.11.24
8	Підготовка доповіді до захисту	01.12.23

6. Дата видачі завдання: *28.06.2024р.*

Керівник _____ /Рубан С.А./

7. Запевнення: *Я, Лінський Михайло Павлович, запевняю що ця кваліфікаційна робота виконана самостійно, не містить академічного плагіату, фабрикації, фальсифікації. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.*

Із чинним Положенням про академічну доброчесність Криворізького національного університету ознайомлений.

Чітко усвідомлюю, що в разі виявлення у кваліфікаційної роботі умисних порушень робота не допускається до захисту або оцінюється незадовільно.

Здобувач _____ /Лінський М.П./

АНОТАЦІЯ

Лінський М. П. Розробка веб-додатку для управління бібліотекою медіафайлів.

Кваліфікаційна робота на здобуття ступеня вищої освіти – магістр за спеціальністю 122 Комп’ютерні науки. Криворізький національний університет, Кривий Ріг, 2024.

У межах дослідження було проведено проєктування та реалізацію веб-додатку для управління бібліотекою медіафайлів, призначеного для забезпечення зручного зберігання, організації та управління цифровими медіаресурсами. Для побудови додатку використано сучасні технології та інструменти, такі як веб-фреймворк ASP.NET Core для серверної частини та React з Redux для клієнтської. Система забезпечує масштабованість, інтеграцію із хмарним сервісом Microsoft Azure для зберігання файлів, а також підтримує авторизацію користувачів.

У роботі детально описано архітектурний підхід до розробки, який базується на принципах модульності, підтримки розширюваності та відповідності сучасним стандартам веб-розробки. Для роботи з базою даних застосовано ORM-систему Entity Framework Core, що дозволило спростити доступ до даних і забезпечити їх надійність.

Особливу увагу приділено практичному впровадженню створеного додатку. Проведено тестування функціоналу, яке продемонструвало його коректну роботу в умовах реального використання. Надано рекомендації щодо подальшого вдосконалення системи, зокрема, інтеграції з іншими сервісами та розширення функціональних можливостей.

Ключові слова:

ВЕБ-ДОДАТОК, УПРАВЛІННЯ МЕДІА ФАЙЛАМИ, REACT, AZURE, ENTITY FRAMEWORK CORE, ASP.NET CORE, REDUX.

ANNOTATION

Linskyi M. P. Development of a Web Application for Managing a Media File Library.

Qualification paper for obtaining the master's degree in specialty 122 - Computer Science. Kryvyi Rih National University, Kryvyi Rih, 2024.

The study involved the design and implementation of a web application for managing a media file library, aimed at providing convenient storage, organization, and management of digital media resources. Modern technologies and tools were used to build the application, including the ASP.NET Core web framework for the server side and React with Redux for the client side. The system ensures scalability, integration with the Microsoft Azure cloud service for file storage, and supports user authentication.

The paper provides a detailed description of the architectural approach to development, which is based on the principles of modularity, extensibility, and compliance with modern web development standards. For database operations, the ORM system Entity Framework Core was used, simplifying data access and ensuring reliability.

Particular attention was paid to the practical implementation of the developed application. Functional testing was conducted, demonstrating its correct operation under real-world conditions. Recommendations for further system improvements are provided, including integration with other services and the expansion of functional capabilities.

Keywords:

WEB APPLICATION, MEDIA FILE MANAGEMENT, REACT, AZURE, ENTITY FRAMEWORK CORE, ASP.NET CORE, REDUX.

ЗМІСТ

ВСТУП

У сучасному інформаційному світі управління медіа контентом відіграє ключову роль у багатьох галузях: від маркетингу та освіти до розваг та особистих архівів. З розвитком цифрових технологій кількість медіа файлів стрімко зростає, і виникає необхідність у створенні ефективних інструментів для їх зберігання, організації та доступу. Проблеми, пов'язані з безпекою, структурованістю та швидким пошуком медіа файлів, стають все більш актуальними для користувачів, що працюють з великими обсягами даних. Зазвичай, локальне зберігання файлів на персональних пристроях стає недостатнім через обмежений простір, низьку швидкість доступу та відсутність належних механізмів управління.

Основною проблемою, яку вирішує веб-додаток для управління бібліотекою медіа файлів, є потреба у зручному та надійному інструменті для централізованого управління файлами різних типів. Існуючі рішення, такі як локальні сховища або зовнішні сервіси, часто не відповідають вимогам користувачів, особливо коли мова йде про ефективну роботу з великими колекціями файлів, забезпечення їхньої доступності з будь-якого пристрою, а також підтримку різних рівнів доступу та співпраці між користувачами. Наприклад, компанії, що працюють з великою кількістю мультимедійного контенту, стикаються з труднощами організації файлів, що уповільнює їхню діяльність та впливає на продуктивність.

Метою цієї дипломної роботи є розробка веб-додатку, який дозволить вирішити зазначені проблеми шляхом створення зручної платформи для управління медіа файлами. Такий додаток забезпечить користувачів можливістю зберігати файли в хмарі, організовувати їх за допомогою категорій та тегів, здійснювати швидкий пошук за різними критеріями, а також ділитися медіа файлами з іншими користувачами. Однією з ключових функцій веб-додатку стане забезпечення безпеки збережених файлів, захист їх від

несанкціонованого доступу та можливість регулювання прав користувачів на доступ до певних медіа ресурсів.

Для реалізації такого додатку необхідно застосувати сучасні технології та інструменти веб-розробки. Це включає в себе використання хмарних сховищ для зберігання великих обсягів даних, серверних технологій для обробки запитів користувачів та інтеграцію з системами безпеки для забезпечення захищеного доступу до файлів. Інтерфейс веб-додатку повинен бути інтуїтивно зрозумілим та забезпечувати зручність роботи навіть для недосвідчених користувачів. Крім того, важливою складовою є оптимізація додатку для використання на різних пристроях, таких як персональні комп'ютери, планшети та смартфони.

Розробка веб-додатку для управління медіа файлами стане важливим кроком у вирішенні актуальних проблем, пов'язаних із зберіганням та обробкою мультимедійних даних. Такий додаток не тільки надасть користувачам можливість зручно організовувати свої файли, але й забезпечить надійний доступ до них з будь-якого місця та пристрою, сприяючи більш ефективній роботі з медіа контентом та підвищенню продуктивності користувачів та організацій.

РОЗДІЛ 1
ТЕОРЕТИЧНЕ ОБҐРУНТУВАННЯ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Актуальність розробки навчальної платформи

В епоху сучасних цифрових технологій, кількість інформації, яку продукують і споживають користувачі та організації, зростає з колосальною швидкістю. З новою реальністю пов’язані нові виклики, одним із яких є організація та управління медіа-файлами. Все більше і більше медіа-файлів необхідно управляти бізнесу, освітнім установам, медіа та окремим користувачам. Багато людей і компаній щодня стикаються з невеликим угрупованням деяких файлів: фотографій, відео, аудіо, документів, для яких потрібно прив’язувати конкретну папку для їх компоновки в єдину базу, що, у свою чергу, укоратним підходом, є важкою справою.

Актуальність розвитку платформи для управління бібліотекою медіафайлів визначається кількома основними факторами. По-перше, значний обсяг файлів часто призводить до безладу, втрати важливої інформації та тривалого пошуку необхідного ресурсу. У відсутності будь-якого належного інструменту для централізованого управління ці користувачі стикаються з труднощами в пошуку необхідної інформації, що впливає на продуктивність і збільшує витрати часу.

По-друге, сучасні вимоги до безпеки та доступності роблять обов'язковими для платформ можливість надійного зберігання файлів та контролювання доступу до цих файлів. Як у бізнесі, так і в інших сферах однаково важливо мати контроль не лише над зберіганням файлів, але й над обміном такими файлами з іншими людьми та їх безпекою від несанкціонованого доступу. Традиційні методи зберігання на локальних пристроях або звичайних файлових системах дуже часто не можуть забезпечити такого рівня безпеки та функціонування.

По-третє, зростання дистанційної роботи та моделей колективної роботи в інтернеті актуалізує потребу доступу до медіафайлів з будь-якого регіону та пристрою. Хмарні технології, вбудовані в платформу управління медіафайлами, дозволяють користувачам швидко завантажувати, редагувати, сортувати та переглядати файли в реальному часі, що значно покращує робочий процес. Крім того, актуальність розробки такої платформи полягає в необхідності інтеграції з іншими системами та сервісами. Це дозволяє автоматизувати процеси обміну інформацією, організації контенту та підвищення загальної продуктивності роботи з медіа файлами.

Отже, розробка платформи управління бібліотекою медіа файлів є надзвичайно актуальною в умовах сучасних вимог до зберігання, обробки та доступу до медіа контенту. Така платформа вирішить низку проблем, пов’язаних з ефективністю роботи з файлами, безпекою даних та зручністю співпраці, що робить її важливою складовою успішного управління цифровими ресурсами у багатьох сферах діяльності.

1.2 Аналіз існуючих платформ

Актуальна модель ринку систем управління медіа файлами є надзвичайно різноманітною, і вибрана система може суттєво вплинути на продуктивність організації. У світі, де обробка та зберігання аудіовізуальних матеріалів є послугами, компанії потребують інструментів, які допомогли б їм оптимізувати свою операційну діяльність, підвищити видимість контенту та знизити витрати.

Зручність використання, ефективність та безпека, а також можливості інтеграції є основними характеристиками систем управління медіа. Дуже важливим напрямком також є робота з різними форматами, оскільки в сучасній бізнес-діяльності це відео, зображення, аудіо та документи, які використовуються.

Щороку ці системи стають дедалі популярнішими завдяки розвитку функціональних можливостей та розумній ціні. Вони поєднують широкий спектр інструментів, які дозволяють користувачам зручно та ефективно взаємодіяти з ними з будь-якої географічної локації. Ці платформи різняться за багатьма аспектами: форматом, бізнес-моделями, стратегіями щодо контенту та технологій. У цьому огляді ми зосередимося на основних характеристиках існуючих онлайн-платформ та покажемо приклади провідних компаній, які працюють на цьому ринку.

Google Drive є однією з найбільш затребуваних послуг для зберігання файлів як для окремих осіб, так і для організацій. Це служба, яка забезпечує зберігання файлів для документів, зображень, відео тощо в хмарі і дозволяє отримувати до них доступ з будь-якого пристрою, підключеного до Інтернету, у будь-який час.

- переваги інтеграції та взаємодії з іншими продуктами Google, такими як Gmail і Google Docs, дозволяють Google Drive забезпечити більш захований процес обміну інформацією та співпраці. Користувачі можуть ділитися файлами один з одним, і ця служба є особливо корисною, коли потрібна командна робота. Ще один важливий аспект Google Drive - це захист даних користувачів, які захищені сучасними технологіями шифрування, а також іншими характеристиками компанії;
- підтримує крос-платформену функціональність: Серед пристроїв, які підтримують Google Drive, є комп'ютери, смартфони та планшети. Це дозволяє користувачам отримувати доступ до своїх файлів у будь-який час і з будь-якого місця на глобусі;
- інтегровані інструменти редагування: Користувачі мають не лише доступ до зберігання файлів, але й можуть створювати та редагувати документи, таблиці та презентації в Google Drive;

Співпраця: Функції обміну Google Drive створюють можливості для кількох користувачів редагувати один і той же документ в один і той же час, тим самим підвищуючи ефективність команди.

Приклад використання: Агенція цифрового маркетингу використовує Google Drive для організації своїх проектів. Всі команди мають доступ до спільних папок, де зберігаються матеріали для кампаній, звіти та аналітика. Це дозволяє командам швидко адаптуватися до змін у стратегії, зберігаючи всю важливу інформацію в одному місці.

Недоліки: Хоча Google Drive має багато переваг, він також має деякі недоліки:

- обмежений обсяг зберігання: Безкоштовний план має обмежений обсяг (15 ГБ), що може бути недостатньо для великих компаній з великими медіа-бібліотеками;
- проблеми з конфіденційністю: Зберігання чутливих даних у хмарі може викликати занепокоєння з точки зору безпеки та конфіденційності, оскільки дані можуть бути доступні стороннім особам;

Dropbox - веб-сайт для обміну файлами, який також можна використовувати для зберігання інформації в інтернеті. Його основна мета – надати кінцевому користуву наступне:

- легкість у використанні: Інтуїтивно зрозумілий інтерфейс робить Dropbox простим для використання навіть для людей без технічного досвіду;
- функція «Синхронізації файлів»: все, що користувач зберігає в Dropbox, автоматично оновлюється на всіх пристроях, тому не потрібно турбуватися про перенесення останніх файлів;
- функція «Синхронізації файлів»: все, що користувач зберігає в Dropbox, автоматично оновлюється на всіх пристроях, тому не потрібно турбуватися про перенесення останніх файлів;

Приклад використання: Студія дизайну зберігає всю свою графіку в Dropbox. Дизайнери можуть завантажувати файли і залишати коментарі - це значно скорочує цикл затвердження замовлень.

Недоліки:

- функція «Синхронізації файлів»: все, що користувач зберігає в Dropbox, автоматично оновлюється на всіх пристроях, тому не потрібно турбуватися про перенесення останніх файлів;
- обмеження функціональності для медіа: Dropbox не надає спеціалізованих інструментів для обробки медіа контенту, що може бути важливим для фахівців у цій галузі;
- вартість: Платні плани, що пропонуються компанією для купівлі, можуть бути надзвичайно дорогими для малих підприємств;

OneDrive - це платформа для зберігання, що є частиною екосистеми Microsoft 365. Вона дозволяє користувачам зберігати файли в хмарі та легко ділитися ними.

- глибока інтеграція з Microsoft 365: OneDrive працює в тандемі з такими додатками, як Word, Excel, PowerPoint, що дозволяє користувачам редагувати документи в реальному часі;
- розширені функції безпеки: OneDrive забезпечує різні способи шифрування своїх даних , а також безліч способів управління, що можна робити з файлом, вказуючи, хто має право робити що. На відміну від одноразових паролів, шифрування даних може включати кілька параметрів;
- легке зберігання зображень та відео: В OneDrive зберігаються медіафайли з можливістю їх завантаження, а також організовані за певними критеріями;

Недоліки:

- складність для початківців: Деякі споживачі можуть вважати, що OneDrive має інтерфейс, який важко сприймати;

- обмеження для медіа: Як і Google Drive, OneDrive не є спеціалізованою платформою для управління медіа, тому його можливості в цій галузі обмежені;

Adobe Experience Manager - це система, яка спрямована на управління цифровими активами і найкраще підходить для великих підприємств. AEM має ряд інструментів, які використовуються в робочому процесі медіафайлів.

- інтеграція з Adobe Creative Cloud: AEM дозволяє безшовну інтеграцію з інструментами, такими як Photoshop і Illustrator, що робить маніпуляцію графікою легкою;
- покращена аналітика: Платформа надає надійні аналітичні інструменти, які дозволяють вимірювати ефективність медіаконтенту;
- налаштовувані робочі процеси: AEM дозволяє налаштування робочих процесів, що підвищує ефективність команди;

Приклад використання: Велика рекламна агенція використовує AEM для управління своїми рекламними кампаніями. Інтеграція з Adobe Creative Cloud дозволяє команді швидко редагувати та публікувати графіку і перевіряти їхню ефективність на цільовій аудиторії.

Недоліки:

- висока вартість: AEM є дорогим рішенням, що робить його недоступним для малого бізнесу;
- складність налаштування: Потребує висококваліфікованих фахівців для налаштування та управління;

Bynder - хмарна система управління цифровими активами, яка пропонує прості у використанні рішення для структуризації та створення медіаконтенту з іншими.

- функції категоризації: Bynder дозволяє користувачам створювати та зберігати категорії для легшого пошуку медіафайлів;
- інтуїтивно зрозумілий інтерфейс: Простота використання робить Bynder вибором більшості користувачів;

- безпека даних: Включає функції безпеки для захисту чутливої інформації;

Приклад використання: Компанія, що займається виробництвом одягу, використовує Bynder для збору та управління всіма своїми маркетинговими матеріалами. Кожна колекція має свій набір медіафайлів, що допомагає дизайнерам швидко шукати та отримувати відповідний медіаконтент.

Недоліки:

- вартість: Безумовно, для малого бізнесу може бути важко оформити підписку, оскільки Bynder має високі плани підписки;
- обмеження кастомізації: Деякі користувачі Bynder зазначають, що є обмеження щодо того, наскільки платформа може бути налаштована, що, у свою чергу, може викликати труднощі при впровадженні в існуючу діяльність;

Надана аналітика різних типів програмного забезпечення для управління медіафайлами доводить до консенсусу щодо різноманітності та спеціалізації цих типів платформ. Виглядає так, що вибір платформи повинен бути спрямований від вимог користувачів, розміру медіафайлів, типу бізнесу та бюджету. Основні параметри, які потрібно врахувати при виборі рішення, включають:

- обсяг зберігання: Для компаній з великими бібліотеками медіа необхідні рішення з великим обсягом хмарного зберігання;
- функціональність: Адаптоване рішення, таке як AEM або Bynder, може бути підходящим для великих організацій, готових витратити багато грошей і часу на управління;
- інтеграція: Вибрана система також повинна бути сумісною з іншими існуючими системами, що використовуються, це є важливим і потребує врахування;

У наступному розділі ми розглянемо потенційні поліпшення існуючих рішень і нові способи управління медіафайлами, які можуть стати основою для розробки ефективних та інноваційних рішень для бізнесу.

1.3 Постановка задачі

З огляду на безперервне збільшення обсягу та різноманітності доступного цифрового контенту, існує гостра потреба у ефективних рішеннях, які вирішують питання зберігання, управління та отримання медіафайлів. Веб-застосунок для управління бібліотекою медіафайлів прагне задовольнити ці цілі, надаючи користувачам легкий, безпечний та масштабований інструмент, з яким вони можуть виконувати ряд операцій над медіаконтентом.

Сучасні користувачі часто стикаються з кількома викликами під час управління своїми медіафайлами, включаючи:

- невпорядкованість даних: Неправильна систематизація медіафайлів ускладнює пошук і знаходження відповідних файлів;
- безпека: Проблеми, пов'язані з порушенням авторських прав і конфіденційністю даних, є особливо значними на тлі зростаючої кількості випадків несанкціонованого доступу до чутливих матеріалів;
- обмежена функціональність існуючих рішень: Багато доступних платформ не надають користувачам всіх необхідних інструментів для ефективного управління медіа-контентом, що обмежує їх можливості;

Для подолання вищезазначених проблем, веб-додаток для управління бібліотекою медіа файлів ставить перед собою наступні завдання:

1. Розробка архітектури платформи:

- створення модульної архітектури, яка дозволить інтегрувати нові функції та розширення в майбутньому;
- використання архітектури мікросервісів для підвищення гнучкості та масштабованості;

2. Зберігання медіа файлів:

- розробка надійного механізму зберігання файлів на хмарних сервісах, таких як Azure, що забезпечує автоматичне масштабування та резервне копіювання;

- забезпечення швидкого доступу до файлів з будь-якого місця та з будь-якого пристрою;

3. Управління доступом:

- реалізація системи аутентифікації та авторизації користувачів за допомогою OAuth2 або інших сучасних протоколів безпеки;

- створення ролей користувачів з різними дозволами для доступу до медіа-файлів, щоб обмежити, хто може переглядати, редагувати або видаляти контент;

4. Інтерфейс користувача:

- розробка адаптивного веб-інтерфейсу з акцентом на зручність використання та доступність;

- введення функцій, таких як попередній перегляд медіа-контенту, що дозволяє користувачам легко знаходити необхідні файли;

5. Пошук і фільтрація:

- реалізація потужних механізмів пошуку та фільтрації медіа-файлів за допомогою множинних параметрів (тип, дата завантаження, користувач тощо);

- застосування алгоритмів машинного навчання для покращення рекомендацій та фільтрації контенту;

6. Керування метаданими:

- розробка системи для збору, зберігання та редагування метаданих медіа-файлів для спрощення організації та пошуку контенту;

- включення функціоналу для прикріплення тегів та описів до медіа-файлів для полегшення класифікації;

Зворотний зв’язок та підтримка:

- забезпечення механізмів для отримання відгуків від користувачів для покращення функціональності платформи;
- розробка системи підтримки, яка буде складатися з документації, поширених запитань та можливості зв’язатися з представником служби підтримки;

Завершення викладених завдань дозволить створити потужну платформу, яка відповідає всім актуальним вимогам для управління медіа-файлами. Користувачам будуть надані потужні інструменти для роботи з контентом, що допоможе підвищити ефективність та зручність платформи. Крім того, система зможе реагувати на ринкові вимоги та потреби користувачів, тим самим забезпечуючи свою актуальність впродовж тривалого часу.

Таким чином, проєкт “Веб-застосунок для управління бібліотекою медіа-файлів” стане важливим інструментом для організацій та користувачів, які прагнуть ефективно управляти своїм медіа-контентом у цифровому просторі.

Висновки до розділу:

Перший розділ передбачав детальне дослідження, яке включало теоретичний підґрунтя проблематики та огляд основних ідей і підходів, які будуть застосовані в рамках веб-застосунка. Було проаналізовано актуальні питання, що стосуються створення нової платформи, здатної задовольнити потреби споживачів XXI століття щодо зберігання і обробки даних. Зокрема, були вирішені питання безпеки та зручності користування, швидкості доступу до інформації та інтеграції з іншими сервісами.

У процесі проведення аналізу було також оцінено ряд розповсюджених хмарних платформ, таких як Google Drive, Dropbox тощо. Ці сервіси забезпечують багатий функціонал для користувачів, серед функціоналу можна

виділити – легкість у перегляді та управлінні файлами, розширену спроможність у спільному користуванні даними, й високу надійність зберігання інформації. Але також виявлено і деякі недоліки, які можуть бути усунуті в межах нового продукту.

Дослідження конкурентних рішень дало змогу виявити сильні та слабкі сторони платформ, пропонованих на ринку, а також окреслити основні вимоги щодо створення підходящої системи, що може співвідноситися з останніми напрямами розвитку і революційними принципами. Проведений аналіз дав підстави для прийняття рішення про створення власної системи, яка зможе запропонувати нові переваги, розширити вже відомі можливості.

Досить детально були сформульовані вимоги до функціональності перспективного веб-додатка, а також визначені технічні та організаційні особливості його реалізації. Що більше, розроблений план виконання проєкту, що охоплює ключові етапи розробки, тестування системи та її подальшого впровадження

РОЗДІЛ 2
ПРОЄКТУВАННЯ ВЕБ-ДОДАТКУ ДЛЯ УПРАВЛІННЯ БІБЛІОТЕКОЮ МЕДІА ФАЙЛІВ

2.1 Вибір та обґрунтування обраних технологій для реалізації веб-додатку для управління бібліотекою медіа файлів

З огляду на вимоги до функціональності веб-додатку для управління бібліотекою медіа файлів було прийнято рішення розробляти додаток за допомогою фреймворку ASP.NET Core на базі C# .NET 8.0.

Представлене рішення включає такі переваги як:

- високий рівень гнучкості в плані поєднання багатьох різних підходів розробки, як моноліту так і використання мікро сервісів. Включає підтримку найбільш вживаних моделей передачі даних, зокрема REST (Representational State Transfer), gRPC (Google Remote Procedure Call), WebSockets, GraphQL, SOAP (Simple Object Access Protocol);
- широкий спектр інструментів для розробки інтерфейсів користувача. Найпопулярнішим на сьогоднішній день є Single Page Application із застосуванням фреймворків JS, таких як React, Angular, Vue, архітектурний підхід Model View Controller, також варто відмітити підхід із використанням технології Blazor;
- доступність різноманітних засобів для впровадження залежностей, зокрема через конструктори, властивості та методи, завдяки інтегрованому контейнеру інверсії контролю: IoC;
- має зручний інструмент управління зовнішніми бібліотеками та їх версіями за допомогою менеджеру пакетів Nuget;
- доступ до потужних довідкових систем та ресурсів в Інтернеті. Інформація там оновлюється та доповнюється на постійній основі;

- крос-платформна сумісність як на рівні розробки, так і на рівні розгортання додатків, що дозволяє значно зменшити ресурси для впровадження;
- велика кількість хмарних ресурсів, які підтримають розгортання додатку на цій технології;

Окрім цього проект містить використання різних підходів, платформ, бібліотек, та технологій:

- Docker - платформа, яка дозволяє розробникам автоматизувати процеси створення, тестування і розгортання програмного забезпечення в контейнерах. Контейнеризація є технологією, що дозволяє упакувати програму разом із усіма її залежностями (бібліотеки, середовища виконання тощо) у стандартизовані контейнери, які можна запускати в будь-якому середовищі, що підтримує Docker;
- Microsoft Entity Framework Core - наразі є найпопулярнішою об'єктно-реляційною бібліотекою.NET. Вона підтримує велику кількість СУБД, дуже швидке маніпулювання даними та потужні можливості фільтрації, сортування та проекції;
- Microsoft Identity - бібліотека від Microsoft, що забезпечує зручне управління ідентифікацією, пропонуючи потужні можливості інтеграції з ASP.NET Core. Вона легко налаштовується відповідно до будь-яких вимог, незалежно від їх складності, та може бути використана з будь-яким рішенням для зберігання даних;
- React - популярна бібліотека JavaScript, створена Facebook, яка використовується для розробки користувацьких інтерфейсів (UI) веб-додатків. Вона дозволяє розробникам створювати компоненти, які можуть повторно використовуватися, що сприяє створенню масштабованих і швидко реагуючих інтерфейсів
- Vite - сучасний інструмент для збірки проектів на JavaScript, який надає швидку та ефективну середу розробки;

- Redux - бібліотека для управління станом у веб-додатках, особливо з використанням JavaScript і React. Вона дозволяє зберігати і управляти станом додатку в одному місці, що спрощує розробку та тестування;
- Redux Toolkit - це офіційна бібліотека для спрощення роботи з Redux, яка надає набір інструментів і методів для більш зручного і ефективного управління станом у веб-додатках;
- TailWind - утилітарний CSS-фреймворк, який дозволяє створювати сучасні інтерфейси користувача, використовуючи готові CSS-класи для стилізації;
- Sass (Syntactically Awesome Style Sheets) - препроцесор CSS, який дозволяє писати стилі більш ефективно і організовано. Він розширює можливості стандартного CSS, додаючи потужні функції, які полегшують управління стилями та їх повторне використання;

У цьому проєкті для створення інтерфейсу було вирішено використати технологію React. Завдяки React забезпечується реактивне відображення всіх функціональних можливостей додатку, а також швидка та відносно проста взаємодія з серверною частиною. Цей фреймворк дозволяє динамічно оновлювати лише ті елементи інтерфейсу, що змінилися, без перезавантаження всієї сторінки, що значно покращує досвід користувача.

Для зниження кількості помилок та забезпечення чіткої типізації в коді використовуємо TypeScript. Ця технологія д vає змогу вводити статичну типізацію змінних, що допомагає розробникам уникнути багатьох помилок ще на етапі компіляції, знижуючи ризик виникнення проблем під час роботи додатку. Завдяки TypeScript код стає більш передбачуваним та легшим у підтримці.

Щоб забезпечити швидке і зручне середовище для розробки, ми використовуємо Vite як інструмент для створення збірок. Vite дозволяє оперативно запускати розробницьке середовище та швидко вносити зміни, перезавантажуючи лише змінені частини коду. Це значно прискорює процес

розробки, особливо при роботі з великими проєктами, і дозволяє підтримувати сучасні стандарти ECMAScript. Конфігурація Vite налаштована таким чином, щоб підтримувати стандарт ECMAScript 2020, що дає змогу використовувати найсвіжіші функціональні можливості JavaScript.

Для керування станом додатку використовуємо Redux та Redux Toolkit. Це дозволяє централізувати управління даними, полегшуючи обробку запитів до серверної частини та відповідей від неї. Redux Toolkit надає інструменти для скорочення шаблонного коду та оптимізації роботи з Redux, що дозволяє ефективніше обробляти запити до сервера та керувати станом.

У частині стилізації було обрано комбінацію Tailwind CSS та SCSS. Tailwind CSS пропонує набір утилітарних класів, що дозволяє швидко і зручно створювати адаптивні інтерфейси. Це значно зменшує обсяг CSS-коду, оскільки стилі можуть бути описані безпосередньо в HTML-розмітці за допомогою готових класів Tailwind. Крім того, Tailwind має вбудовану підтримку адаптивного дизайну, що дає змогу швидко налаштувати додаток під різні розміри екранів.

Разом з Tailwind використовується SCSS для створення більш складних стилів та підтримки кастомізації, яка може бути важко досяжною лише за допомогою утилітарних класів. SCSS додає змінні, вкладеність, міксіни та інші можливості, що дозволяють підтримувати чисту та зручну структуру стилів. Завдяки SCSS стилі стають модульними і легко масштабованими, що дуже корисно в великих проєктах, де важливо підтримувати чистоту та послідовність коду.

На рис. 2.1 представлено перелік використаних NuGet-пакетів, які були інтегровані в серверну частину додатку для забезпечення його функціональності. Ці пакети надають додаткові інструменти, бібліотеки та сервіси, що спрощують розробку, дозволяють уникнути повторного написання коду та забезпечують високу продуктивність системи.

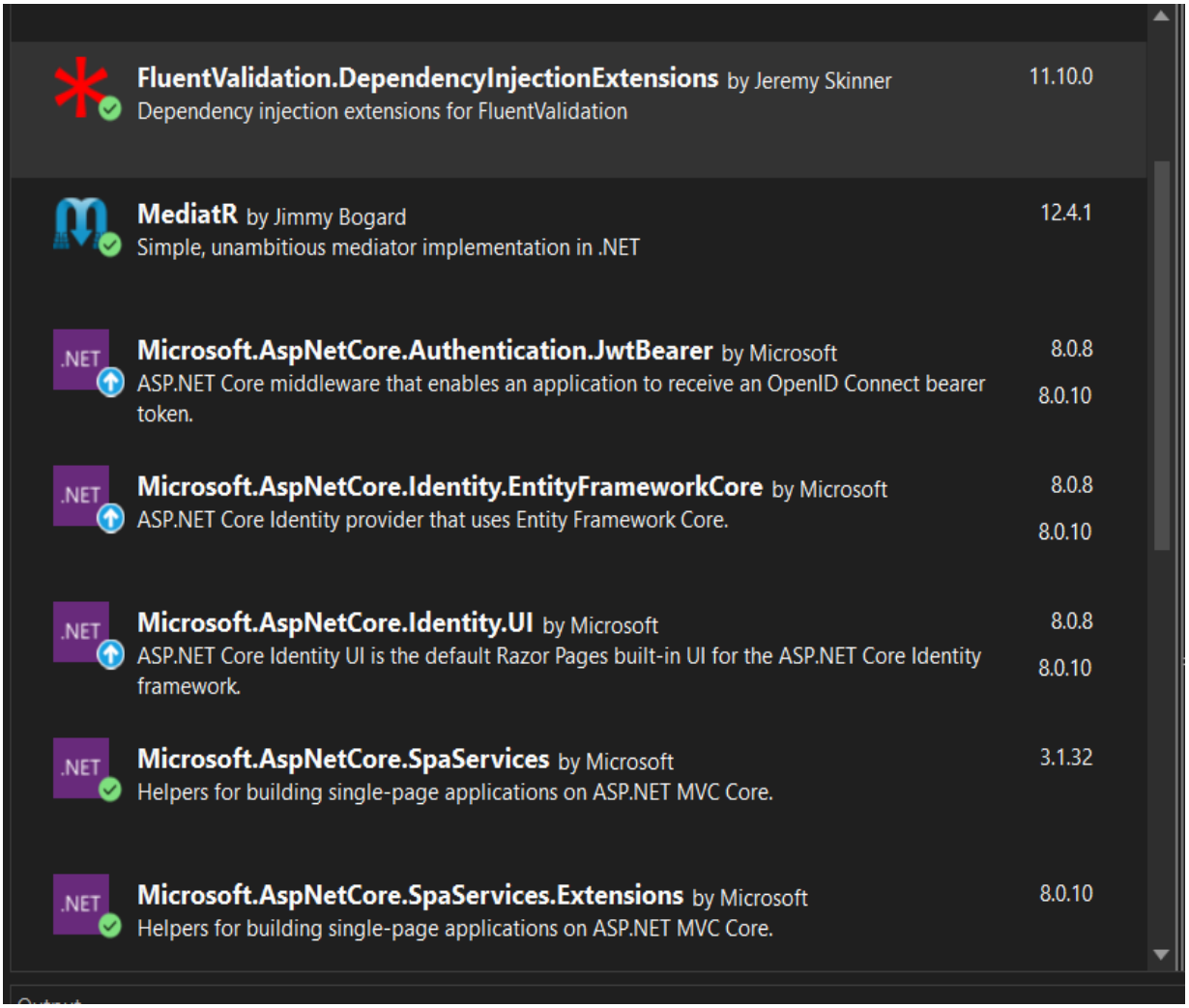


Рисунок 2.1 - Пакети, що використанні у додатку для клієнтської частини

На рис. 2.2 представлено перелік використаних пакетів, інтегрованих у клієнтську частину додатку. Ці пакети є основними інструментами для забезпечення функціональності фронтенд-частини додатку, полегшення розробки, оптимізації продуктивності та покращення користувацького досвіду.

Рисунок 2.2 - Пакети, що використані у додатку для клієнтської частини

2.2 Проєктування структури бази даних веб-додатку для управління бібліотекою медіа файлів

Веб-додаток для управління бібліотекою медіа файлів має включати реалізацію аутентифікації та авторизації кінцевих користувачів, згідно поставлених вимог проєкту. Беручи до уваги, що було обрано використовувати технологію ASP.NET Core, має сенс будувати авторизацію та аутентифікацію на базі Microsoft.AspNetCore.Identity. Вищезгаданий пакет

допомагає налаштувати дані користувачів, керувати ролями користувачів, містить функціонал для інтеграції різних способів аутентифікації (за допомогою паролю, різних зовнішніх OAuth), а також містить інформацію про claims користувача.

Microsoft.AspNetCore.Identity має цілу низку класів для моделювання відповідних сутностей, які потім відображаються у вигляді таблиць у базі даних за допомогою бібліотеки EntityFramework Core. Оскільки у додатку передбачається реалізація аутентифікації та авторизації на основі ролей і клеймів (claims), під час використання Microsoft.AspNetCore.Identity у базі даних будуть створені таблиці, необхідні для роботи цієї бібліотеки:

- AspNetRoleClaims – зберігає клейми, прив'язані до кожної ролі в системі, з аналогічним типом зв’язку "багато до багатьох" (M:M);
- AspNetRoles - зберігає інформацію про всі зареєстровані ролі в системі;
- AspNetUserClaims - містить дані про клейми (claims), пов'язані з кожним користувачем. Ця таблиця також має зв’язок "багато до багатьох" (M:M);
- AspNetUserLogins - зберігає інформацію про логіни користувачів, зокрема ті, що використовуються через сторонні провайдери OAuth, такі як Google, iCloud, Дія, Facebook тощо;
- AspNetUserRoles - використовується для зберігання зв'язків між користувачами та їх ролями в системі. Вона реалізує відношення "багато до багатьох" (M:M), де один користувач може мати кілька ролей, а одна роль може бути призначена багатьом користувачам. Ця таблиця дозволяє ефективно керувати правами доступу, визначаючи, які ролі (а отже, й рівні доступу) мають окремі користувачі;
- AspNetUsers - використовується для зберігання інформації про користувачів у системах, що використовують Identity для аутентифікації та управління користувачами. Вона містить основні дані, такі як логіни, хешовані

паролі, електронну пошту, а також інші властивості, що дозволяють ідентифікувати користувача та керувати його доступом. Ця таблиця є центральною для управління користувачами, підтримуючи операції реєстрації, входу, скидання пароля та інші функції, пов'язані з безпекою та ідентифікацією в системі;

- AspNetUserTokens - використовується для зберігання токенів, пов'язаних із користувачами. Ці токени можуть включати, наприклад, токени доступу (JWT), які використовуються для аутентифікації користувачів у системі. Вона містить інформацію про тип токена, його значення та прив'язку до конкретного користувача. Ця таблиця дозволяє зберігати і керувати тимчасовими або постійними токенами, що забезпечують безпечний доступ до ресурсів та підтримують механізми авторизації;

Було створено дві таблиці для опису ключових сутностей для зберігання медіа файлів Folder (Каталог), File (Файл) (рис. 2.3):

- Folder - таблиця, яка представляє каталоги для організації файлів. Вона містить поля для збереження назви каталогу, посилання на батьківський каталог (якщо є), а також інформацію про користувача, який створив цей каталог. Крім того, таблиця підтримує зв'язки з підкаталогами (реалізуючи ієрархічну структуру каталогів) і файлами, що зберігаються в даному каталозі;
- File - nаблиця для зберігання інформації про файли. Вона включає основні атрибути файлу, такі як ім'я, URL для доступу, розмір файлу та його тип контенту. Файли пов'язані з конкретним каталогом через зовнішній ключ;

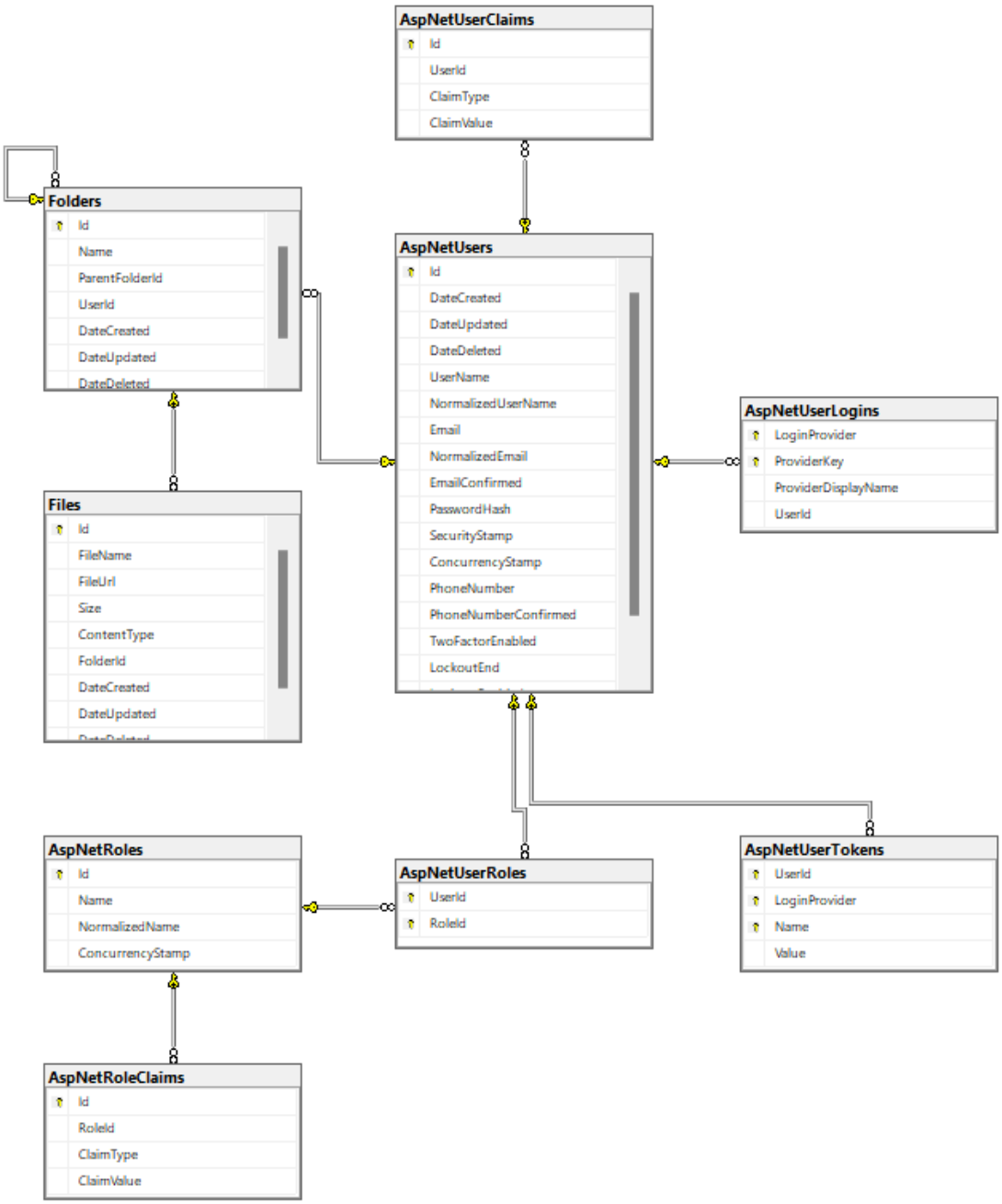


Рисунок 2.3 - Структура бази даних веб-додатку для управління бібліотекою медіа файлів

Сутність User, яка успадкована від базового класу IdentityUser з бібліотеки Microsoft.AspNetCore.Identity створює таблицю AspNetUsers, що є реліційним відображенням сутності. Так як користувачі можуть мати декілька каталогів, було вирішино встановити зв'язок «один-до-багатьох» між користувачами (AspNetUsers) і каталогами (Folders) (1:M).

Таблиця 2.1 – Структура таблиць бази даних «media-library»

Назва таблиці	Назва поля	Тип даних	Атрибут
Folders	Id	uniqueidentifier	Первинний ключ
	Name	nvarchar(255)	
	ParentFolderId	uniqueidentifier	Зовнішній ключ
	UserId	uniqueidentifier	Зовнішній ключ
	DateCreated	datetimeoffset(7)	
	DateUpdated	datetimeoffset(7)	
	DateDeleted	datetimeoffset(7)	
Files	Id	uniqueidentifier	Первинний ключ
	FileName	nvarchar(255)	
	FileUrl	nvarchar(MAX)	
	ParentFolderId	uniqueidentifier	Зовнішній ключ
	ContentType	nvarchar(MAX)	
	Size	Bigint	
	DateCreated	datetimeoffset(7)	
	DateUpdated	datetimeoffset(7)	
	DateDeleted	datetimeoffset(7)	
AspNetUsers	Id	uniqueidentifier	Первинний ключ
	Email	nvarchar(256)	
	PasswordHash	nvarchar (max)	
	DateCreated	datetimeoffset(7)	
	DateUpdated	datetimeoffset(7)	
	DateDeleted	datetimeoffset(7)	

На рис. 2.4 представлено UML діаграму класів, що відображає основні взаємозв'язки між сутностями предметної області, які беруть участь у взаємодії в веб-додатку для управління бібліотекою медіа файлів. Ця діаграма демонструє, як користувачі зберігають папки та файли, і як між ними встановлюються зв'язки.

У системі кожен користувач може створювати одну або декілька папок для організації своїх файлів. Тому сутність User містить навігаційну властивість Folders, що є колекцією типу ICollection<Folder>. Це означає, що користувач може мати багато папок. З іншого боку, кожна папка належить конкретному користувачу, що відображається через зовнішній ключ UserId у

сутності Folder та навігаційну властивість User, яка вказує на користувача, власника папки.

Крім того, система підтримує вкладені папки, що дозволяє користувачам створювати ієрархію папок. Для цього у сутності Folder присутній зовнішній ключ ParentFolderId, що вказує на батьківську папку. Також є навігаційна властивість ParentFolder, яка вказує на об'єкт батьківської папки. Одночасно папка може містити інші папки, тому у сутності Folder передбачено властивість SubFolders типу ICollection<Folder>, яка зберігає дочірні папки.

Кожна папка також може містити один або кілька файлів. Тому сутність Folder має навігаційну властивість Files типу ICollection<File>, що містить файли, які зберігаються в папці. З іншого боку, кожен файл прив'язаний до конкретної папки, що відображається через зовнішній ключ FolderId та навігаційну властивість Folder, яка вказує на папку, в якій цей файл знаходиться.

Таким чином, UML діаграма чітко демонструє такі взаємозв'язки:

- User може мати багато Folder. Це відображає модель, в якій кожен користувач може створювати та зберігати власні папки для організації медіафайлів;
- Folder може містити підпапки (SubFolders), Це забезпечує можливість створення ієрархічної структури папок, де одна папка може бути вкладеною в іншу;
- Folder може містити багато File. Це демонструє, що одна папка може містити кілька файлів одночасно, забезпечуючи гнучкість у організації даних;
- File належить конкретній папці (Folder). Кожен файл чітко асоціюється з однією папкою, що дозволяє легко знайти та керувати файлами в межах певної структури

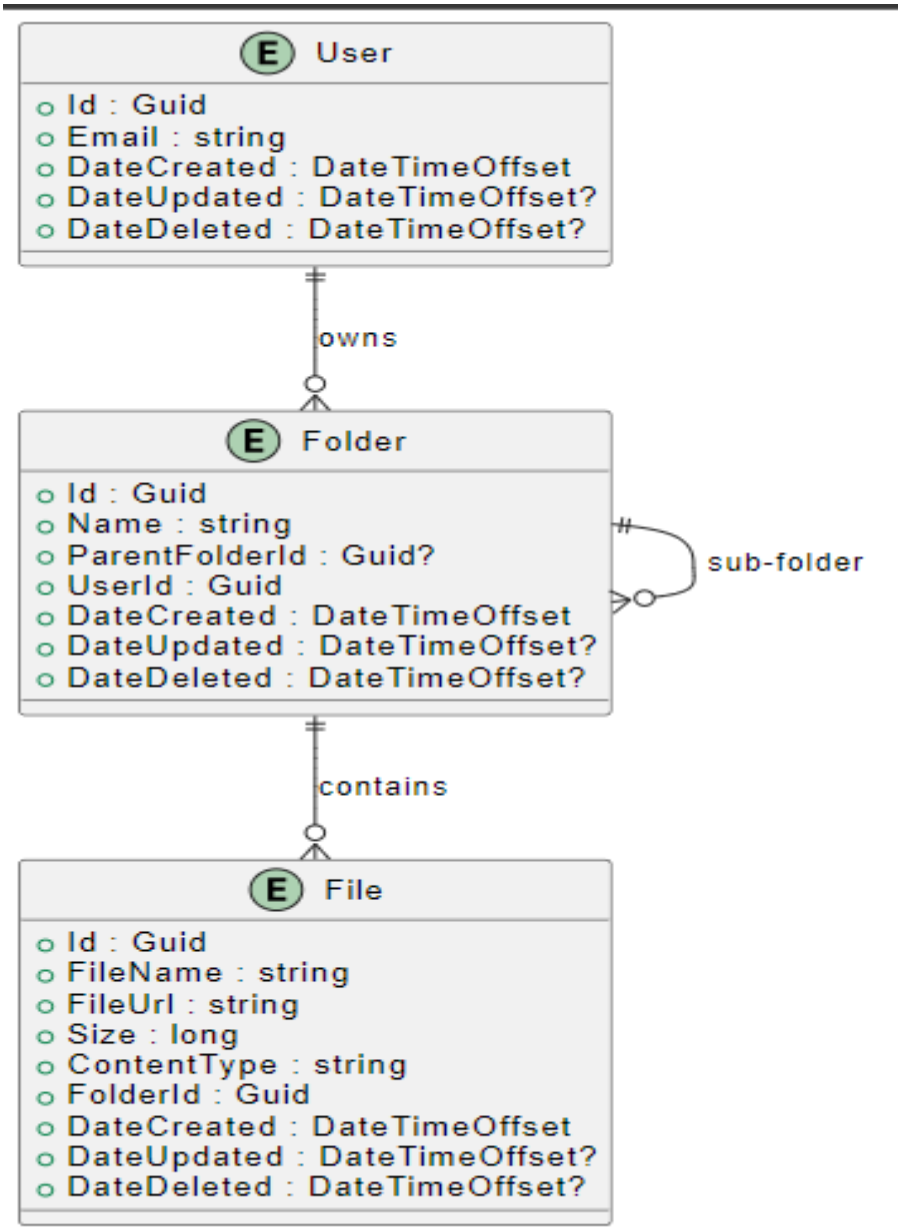


Рисунок 2.4 - Діаграма класів UML основних сутностей веб-додатку для управління бібліотекою медіа файлів

У рамках реалізації об'єктно-орієнтованого відображення за допомогою EntityFramework Core для маніпуляції даними, які зберігаються в базі даних, необхідно створити клас контексту даних, що містить властивості типу DbSet<T>, де T - типи сутностей, для яких створюються таблиці в базі даних. Також, якщо додаток використовує бібліотеку Microsoft.AspNetCore.Identity,

клас контексту може успадковуватися від IdentityDbContext<TUser>, де TUser - це клас, що представляє користувача в системі.

Для розроблюваної інформаційної системи керування медіа-файлами клас контексту даних має наступні властивості:

- DbSet<Folder> Folders - колекція сутностей Folder, яка відображає інформацію про папки користувачів. Папка може містити підпапки та файли, що зберігаються в системі. Ці сутності зберігаються в таблиці бази даних, яка пов'язана з кожним користувачем через зовнішній ключ UserId;
- DbSet<File> Files - колекція сутностей File, яка містить інформацію про файли, що зберігаються в папках. Кожен файл належить до певної папки, що відображається через зовнішній ключ FolderId;

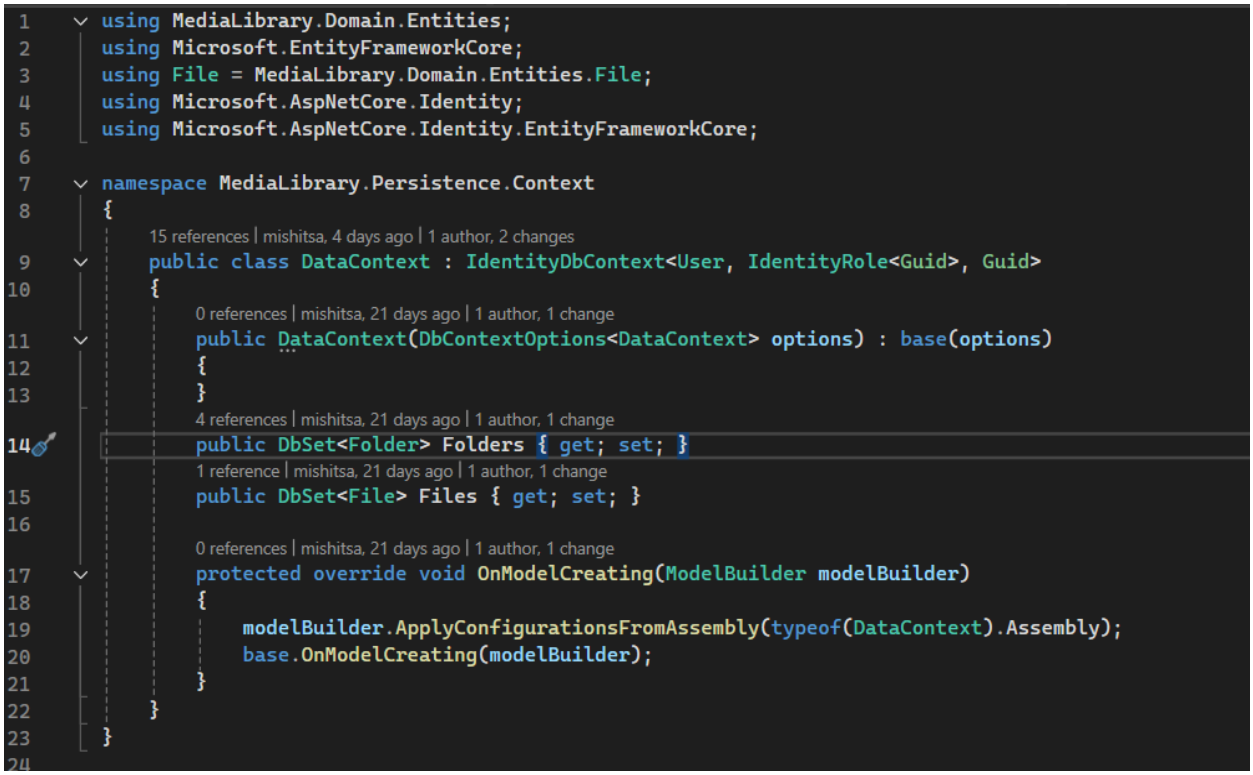
Крім цього, клас контексту DataContext рис (2.5) успадковується від IdentityDbContext<User, IdentityRole<Guid>, Guid>, що дозволяє працювати з сутністю User (користувач системи). Це також означає, що контекст підтримує роботу з системою авторизації та аутентифікації на базі Identity.

Конфігурація сутностей:

Для правильного налаштування відносин між сутностями у базі даних використовуються конфігураційні класи:

- FileConfiguration: У конфігурації для сутності File налаштовані такі параметри рис (2.6);
- файл належить до папки (відношення "багато до одного"), де каскадне видалення забезпечує автоматичне видалення файлів разом з папкою;
- встановлено обов’язкові поля для імені файлу FileName та URL FileUrl з максимальними довжинами значень;
- FolderConfiguration: У конфігурації для сутності Folder налаштовані такі параметри рис (2.7);
- папка належить конкретному користувачу, встановлюючи відношення "багато до одного" з каскадним видаленням;

- підтримується ієрархія папок за допомогою зовнішнього ключа на батьківську папку ParentFolderId, при цьому видалення батьківської папки не призводить до видалення дочірніх (обмежена поведінка видалення);
- поле для імені папки Name є обов’язковим та обмежене за довжиною;
- UserConfiguration: У конфігурації для сутності User налаштовано, що поле для електронної пошти користувача є обов’язковим та має максимальну довжину 100 символів;



```
using MediaLibrary.Domain.Entities;
using Microsoft.EntityFrameworkCore;
using File = MediaLibrary.Domain.Entities.File;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;

namespace MediaLibrary.Persistence.Context
{
    15 references | mishitsa, 4 days ago | 1 author, 2 changes
    public class DataContext : IdentityDbContext<User, IdentityRole<Guid>, Guid>
    {
        0 references | mishitsa, 21 days ago | 1 author, 1 change
        public DataContext(DbContextOptions<DataContext> options) : base(options)
        {
        }
        4 references | mishitsa, 21 days ago | 1 author, 1 change
        public DbSet<Folder> Folders { get; set; }
        1 reference | mishitsa, 21 days ago | 1 author, 1 change
        public DbSet<File> Files { get; set; }

        0 references | mishitsa, 21 days ago | 1 author, 1 change
        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.ApplyConfigurationsFromAssembly(typeof(DataContext).Assembly);
            base.OnModelCreating(modelBuilder);
        }
    }
}
```

Рисунок 2.5 - Клас контекст даних

Під час створення моделей клас контексту викликає класи конфігурацій сутностей зі збірки. На рис. 2.6 зображений клас конфігурації сутності File. Який описує відношення між таблицями File та Folder, описує принцип видалення даних та містить в собі опис полів, які буде мати таблиця. Наприклад назва файлу має бути не більше ніж 100 символів, а також це поле є обов'язковим.

```
using Microsoft.EntityFrameworkCore.Metadata.Builders;
using Microsoft.EntityFrameworkCore;
using File = MediaLibrary.Domain.Entities.File;

namespace MediaLibrary.Persistence.Configurations
{
    0 references | mishitsa, 21 days ago | 1 author, 1 change
    public class FileConfiguration : IEntityTypeConfiguration<File>
    {
        0 references | mishitsa, 21 days ago | 1 author, 1 change
        public void Configure(EntityTypeBuilder<File> builder)
        {
            builder.HasOne(f => f.Folder)
                .WithMany(f => f.Files)
                .HasForeignKey(f => f.FolderId)
                .OnDelete(DeleteBehavior.Cascade);

            builder.Property(f => f.FileName)
                .IsRequired()
                .HasMaxLength(255);

            builder.Property(f => f.FileUrl)
                .IsRequired();
        }
    }
}
```

Рисунок 2.6 - Клас конфігурації сутності File

На рис. 2.7 зображено клас конфігурації сутності Folder, який виконує важливу роль у визначенні відносин між таблицями User та Folder в базі даних. Клас конфігурації описує, як доменна сутність Folder повинна бути мапована на відповідну таблицю, визначаючи зв'язки з іншими сутностями, а також вказує на принципи обробки та видалення даних. Зокрема, цей клас налаштовує зовнішні ключі, що забезпечують коректне відображення зв'язку між користувачем і його папками. Одним із важливих аспектів є опис політики видалення, яка визначає, як мають оброблятися дані при видаленні користувача чи папки, забезпечуючи цілісність даних у базі.

Додатково, клас конфігурації містить опис полів, що визначаються для таблиці Folder. Наприклад, поле Name, яке відображає назву папки, обмежене

максимальним значенням у 255 символів, що відповідає загальноприйнятим вимогам щодо довжини рядкових значень у базах даних. Крім того, це поле є обов'язковим, що гарантує, що кожна папка має бути надана з валідною назвою при її створенні. Завдяки таким налаштуванням, клас конфігурації сутності Folder забезпечує точне відображення бізнес-логіки у структурі бази даних, підтримуючи цілісність і консистентність даних на всіх етапах взаємодії з додатком.

```
using MediaLibrary.Domain.Entities;
using Microsoft.EntityFrameworkCore.Metadata.Builders;
using Microsoft.EntityFrameworkCore;

namespace MediaLibrary.Persistence.Configurations
{
    0 references | mishitsa, 21 days ago | 1 author, 1 change
    public class FolderConfiguration : IEntityTypeConfiguration<Folder>
    {
        0 references | mishitsa, 21 days ago | 1 author, 1 change
        public void Configure(EntityTypeBuilder<Folder> builder)
        {
            builder.HasOne(f => f.User)
                .WithMany(u => u.Folders)
                .HasForeignKey(f => f.UserId)
                .OnDelete(DeleteBehavior.Cascade);

            builder.HasOne(f => f.ParentFolder)
                .WithMany(f => f.SubFolders)
                .HasForeignKey(f => f.ParentFolderId)
                .OnDelete(DeleteBehavior.Restrict);

            builder.Property(f => f.Name)
                .IsRequired()
                .HasMaxLength(255);
        }
    }
```

Рисунок 2.7 - Клас конфігурації сутності Folder

```
using MediaLibrary.Domain.Entities;
using Microsoft.EntityFrameworkCore.Metadata.Builders;
using Microsoft.EntityFrameworkCore;

namespace MediaLibrary.Persistence.Configurations
{
    0 references | mishitsa, 21 days ago | 1 author, 1 change
    public class UserConfiguration : IEntityTypeConfiguration<User>
    {
        0 references | mishitsa, 21 days ago | 1 author, 1 change
        public void Configure(EntityTypeBuilder<User> builder)
        {
            builder.Property(u => u.Email)
                .IsRequired()
                .HasMaxLength(100);
        }
    }
}
```

Рисунок 2.8 - Клас конфігурації сутності User

2.3 Проєктування основної структури веб-додатку

Оскільки було прийнято рішення здійснювати розробку веб-додатку для управління бібліотекою медіафайлів, використовуючи технології ASP.NET Core і React.js, проєкт побудований відповідно до сучасних стандартів проєктування архітектури та вимог написання якісного й підтримуваного коду. Архітектура додатку включає в себе наступні основні компоненти (рис. 2.9):

MediaLibrary.Api

Це шар, який відповідає за побудову API і забезпечує зв'язок між клієнтом і сервером. Основне завдання цього шару полягає в обробці HTTP-запитів і передачі їх для обробки до інших компонентів додатку.

- Client: Містить усі файли, пов'язані з фронтенд-частиною, включаючи компоненти React.js і стилі;

- Controllers: Відповідають за обробку HTTP-запитів, таких як GET, POST, PUT і DELETE. Контролери взаємодіють із сервісами бізнес-логіки, отримують або передають дані, наприклад, завантажують файли чи отримують список папок;
- Extensions: Тут розміщуються методи розширень для конфігурації додатку, middleware, DI-контейнерів, налаштування сервісів та інших компонентів;

MediaLibrary.Application

Цей шар містить бізнес-логіку та основні функціональні можливості для роботи з медіафайлами та їх метаданими. Він забезпечує обробку бізнес-процесів і виконує всі основні операції з даними.

- Common: Загальні допоміжні класи або утиліти, які використовуються в багатьох місцях додатку;
- Features: Відповідає за реалізацію окремих функціональних можливостей, таких як управління файлами, обробка папок, реєстрація та управління користувачами;
- Interfaces: Інтерфейси, які описують контракти для сервісів і репозиторіїв. Це дозволяє реалізувати логіку через залежності та абстракції;
- Models: Містить моделі даних (DTO), які використовуються для передачі інформації між API і клієнтом або між різними шарами додатку;
- FilStorage: Забезпечує інтеграцію з різними сервісами зберігання та роботи з файлами, як-от Azure Blob Storage та Amazon S3 Bucket;
- Services: Реалізують основну бізнес-логіку, включаючи управління файлами, взаємодію з користувачами та обробку запитів до репозиторіїв;

MediaLibrary.Domain

Цей шар фокусується на доменних сутностях і є ядром додатку. У ньому описані основні об’єкти, з якими працює система, а також правила їхньої поведінки.

- Base: Базові класи та інтерфейси, які можуть використовуватися іншими доменними сутностями, наприклад, загальні властивості для сутностей;
- Entities: Містить доменні сутності, які відображають ключові об'єкти додатку, наприклад, користувачі (User), файли (File) і папки (Folder). Ці сутності безпосередньо зберігаються в базі даних;
- Repositories: Визначає репозиторії, які відповідають за доступ до бази даних. Репозиторії надають методи для виконання CRUD-операцій із доменними сутностями, забезпечуючи абстракцію над базовою інфраструктурою даних;

MediaLibrary.Persistence

Шар для роботи з даними, є ключовим компонентом архітектури, який охоплює всі аспекти взаємодії з базою даних. Він забезпечує зручний і надійний доступ до даних, а також управління їхнім збереженням та обробкою. Основні елементи цього шару.

- Configurations: Містить конфігурації для моделей бази даних. Ці конфігурації визначають, як доменні сутності мапуються на таблиці в базі даних, задаючи властивості колонок, індекси, ключі тощо;
- Context: Включає контекст бази даних, наприклад, DataContext, який виступає як головна точка доступу до Entity Framework Core. Контекст керує зв’язком із базою даних і забезпечує роботу з сутностями через DbSet;
- Interfaces: Інтерфейси для шарів доступу до даних, таких як репозиторії, з метою використання залежностей;
- Migrations: Зберігають міграції бази даних, які описують зміни в її схемі. Цей компонент дозволяє додавати нові таблиці, змінювати структуру існуючих або видаляти зайві елементи безпосередньо з коду додатку;
- Models: Охоплюють моделі даних, які використовуються у взаємодії з базою. Наприклад, це можуть бути моделі для зберігання

інформації про файли, метадані, користувачів тощо, які адаптовані до потреб збереження в базі;

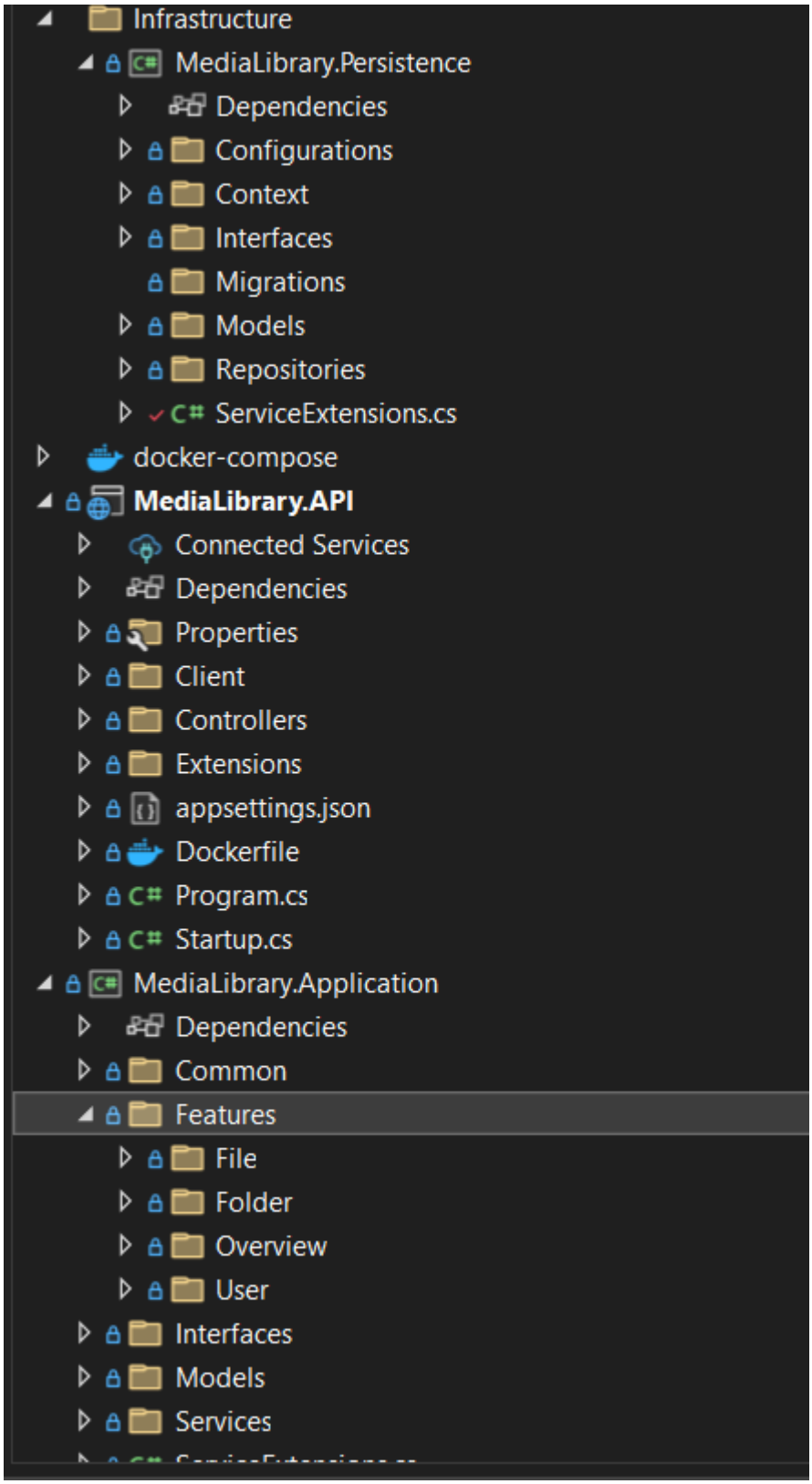


Рисунок 2.9 - Структура веб-додатку

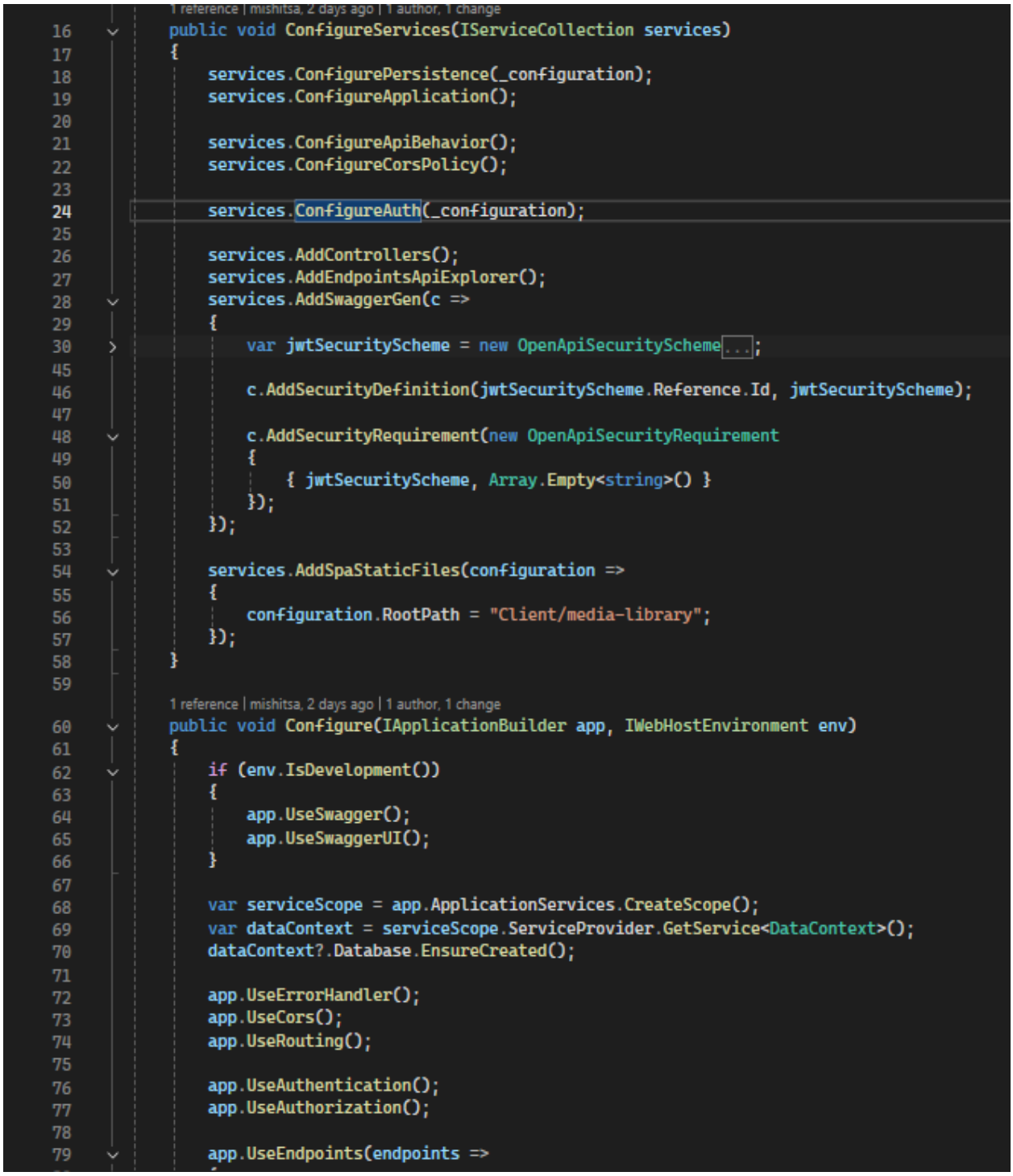
У даному проєкті файл Program.cs є основною точкою входу, яка запускає веб-сервер та ініціює виклик класу Startup для налаштування додатка. Клас Startup, як видно на рис. 2.10, є центральним компонентом, відповідальним за конфігурацію всіх необхідних сервісів і обробку HTTP-запитів. Він включає два основних методи: ConfigureServices та Configure, які забезпечують налаштування середовища виконання програми та визначають логіку обробки запитів.

У методі ConfigureServices реєструються всі сервіси, необхідні для функціонування додатка. Це включає сервіси для збереження даних, що реалізуються через метод ConfigurePersistence, який відповідає за інтеграцію з базами даних чи іншими сховищами даних. Додатково, через метод ConfigureApplication конфігуруються сервіси для реалізації бізнес-логіки додатка. Важливим етапом є налаштування політики CORS (Cross-Origin Resource Sharing), яка дозволяє контролювати доступ до ресурсів веб-додатка з інших доменів. Крім того, у цьому методі налаштовується система автентифікації на основі JWT (JSON Web Tokens), що забезпечує безпечну ідентифікацію користувачів.

У методі Configure здійснюється налаштування конвеєра обробки запитів. Це включає включення Swagger для режиму розробки, обробку помилок, конфігурацію політики CORS, налаштування маршрутизації запитів, а також визначення механізмів автентифікації та авторизації для доступу до різних частин системи. Важливим етапом є конфігурація кінцевих точок для контролерів, що дозволяє визначити, які URL шляхи будуть оброблятися конкретними методами контролерів.

Крім того, через клас Startup налаштовуються сервіси з інших частин проєкту, таких як MediaLibrary.Persistence та MediaLibrary.Application, що дозволяє інтегрувати їх з основним додатком та забезпечити безперебійну роботу всіх його компонентів. Це налаштування сприяє організації чіткої та

зручної для розробників архітектури проєкту, що відповідає сучасним вимогам до масштабованості та підтримки програмного забезпечення.



```
1 reference | mishitsa, 2 days ago | 1 author, 1 change
public void ConfigureServices(IServiceCollection services)
{
    services.ConfigurePersistence(_configuration);
    services.ConfigureApplication();

    services.ConfigureApiBehavior();
    services.ConfigureCorsPolicy();

    services.ConfigureAuth(_configuration);

    services.AddControllers();
    services.AddEndpointsApiExplorer();
    services.AddSwaggerGen(c =>
    {
        var jwtSecurityScheme = new OpenApiSecurityScheme...;

        c.AddSecurityDefinition(jwtSecurityScheme.Reference.Id, jwtSecurityScheme);

        c.AddSecurityRequirement(new OpenApiSecurityRequirement
        {
            { jwtSecurityScheme, Array.Empty<string>() }
        });
    });

    services.AddSpaStaticFiles(configuration =>
    {
        configuration.RootPath = "Client/media-library";
    });
}

1 reference | mishitsa, 2 days ago | 1 author, 1 change
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseSwagger();
        app.UseSwaggerUI();
    }

    var serviceScope = app.ApplicationServices.CreateScope();
    var dataContext = serviceScope.ServiceProvider.GetService<DataContext>();
    dataContext?.Database.EnsureCreated();

    app.UseErrorHandler();
    app.UseCors();
    app.UseRouting();

    app.UseAuthentication();
    app.UseAuthorization();

    app.UseEndpoints(endpoints =>
```

Рисунок 2.10 - Частина класу Startup

У проєкті MediaLibrary.Persistence (рис. 2.11) здійснюється конфігурація сервісів DataContext (через метод AddDbContext) для роботи з базою даних, а також реєструються репозиторії, необхідні для взаємодії з даними.

```
public static class ServiceExtensions
{
    1 reference | mishitsa, 2 days ago | 1 author, 3 changes
    public static void ConfigurePersistence(this IServiceCollection services, IConfiguration configuration)
    {
        var connectionString = configuration.GetConnectionString("SqlServer");
        services.AddDbContext<DataContext>(opt => opt.UseSqlServer(connectionString));

        services.AddScoped<IUnitOfWork, UnitOfWork>();
        services.AddScoped<IUserRepository, UserRepository>();
        services.AddScoped<IOverviewRepository, OverviewRepository>();
        services.AddScoped<IFolderRepository, FolderRepository>();
        services.AddScoped<IFileRepository, FileRepository>();
    }
}
```

Рисунок 2.12 - Конфігурація проєкту MediaLibrary.Application

У проєкті MediaLibrary.Application (рис. 2.12) налаштовується бібліотека для автоматичного співставлення профілів AutoMapper за допомогою методу AddAutoMapper. Крім того, використовується бібліотека Mediatr, яка дозволяє організовувати взаємодію між компонентами (класами, сервісами) без необхідності прямого зв’язку між ними, що реалізується через метод AddMediatR. Також в проєкті додається бібліотека FluentValidation для валідації запитів за допомогою методу AddValidatorsFromAssembly, а також реєструються основні сервіси, відповідальні за бізнес-логіку додатку.

```
public static class ServiceExtensions
{
    1 reference | mishitsa, 5 days ago | 1 author, 2 changes
    public static void ConfigureApplication(this IServiceCollection services)
    {
        services.AddAutoMapper(Assembly.GetExecutingAssembly());
        services.AddMediatR(cfg => cfg.RegisterServicesFromAssembly(Assembly.GetExecutingAssembly()));
        services.AddValidatorsFromAssembly(Assembly.GetExecutingAssembly());
        services.AddTransient(typeof(IPipelineBehavior<,>), typeof(ValidationBehavior<,>));

        ConfigureServices(services);
    }

    1 reference | mishitsa, 5 days ago | 1 author, 1 change
    public static void ConfigureServices(this IServiceCollection services)
    {
        services.AddTransient<IJwtTokenService, JwtTokenService>();
    }
}
```

Рисунок 2.12 - Конфігурація проєкту MediaLibrary.Application

У класі Startup також представлений код для налаштування сервісів аутентифікації та авторизації з використанням бібліотеки Microsoft.AspNetCore.Identity (див. рис. 2.7). Для цього Generic-метод AddIdentity<TUserIdentity, TIdentityRole>() приймає два параметри типу: клас-нащадок TUserIdentity, який описує користувача в системі (в нашому випадку це клас StaffMember), та клас, що використовується для збереження можливих ролей користувачів системи (у нашому випадку - базовий клас IdentityRole).

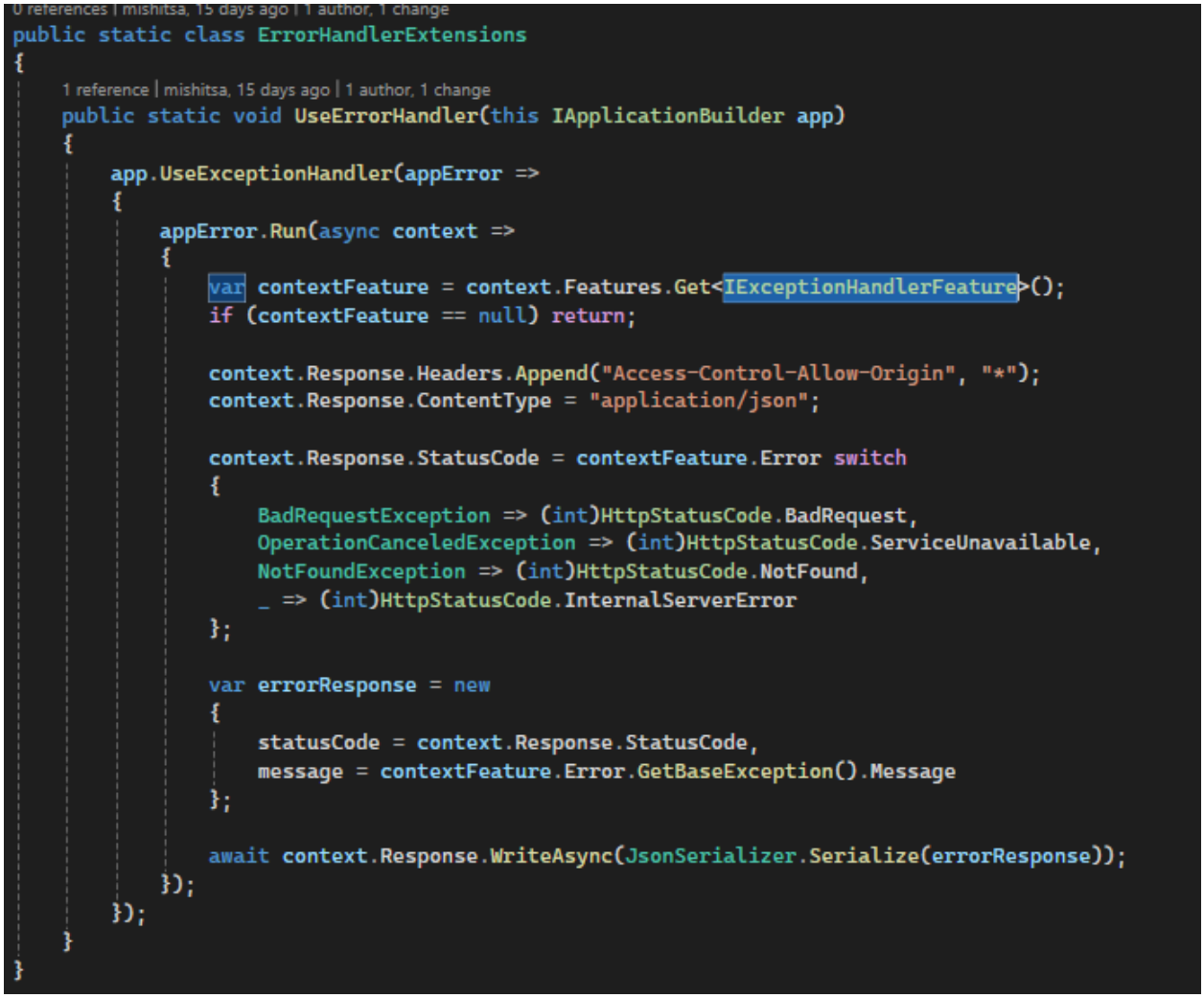
Конфігурація сервісів Microsoft Identity здійснюється через клас контексту бази даних шляхом виклику методу AddEntityFrameworkStores<ApplicationContext>(), в якому передається назва класу контексту, відповідального за управління даними користувачів [20, 21]. Цей клас буде використовуватися для роботи з високорівневими сервісами бібліотеки Microsoft.AspNetCore.Identity, такими як UserManager, RoleManager та SignInManager, для прямого доступу до інформації про користувачів. Виклик методу AddDefaultTokenProviders() забезпечує підключення функціональності генерації токенів аутентифікації для користувачів системи, що дозволяє подальший розвиток системи.

```
services.AddIdentity<User, IdentityRole<Guid>>()
            .AddEntityFrameworkStores<DataContext>()
            .AddDefaultTokenProviders();


services.AddAuthentication(options =>
{
    options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
    options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
})
.AddJwtBearer(options =>
{
    options.TokenValidationParameters = new TokenValidationParameters
    {
        ValidateIssuer = true,
        ValidateAudience = true,
        ValidateLifetime = true,
        ValidateIssuerSigningKey = true,
        ValidIssuer = configuration["Jwt:Issuer"],
        ValidAudience = configuration["Jwt:Audience"],
        IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(configuration["Jwt:Key"]!))
    };
    options.TokenValidationParameters.ValidateIssuer = false;
    options.TokenValidationParameters.ValidateAudience = false;
    options.Events = new JwtBearerEvents...;
});
```

Рисунок 2.12 - Конфігурація автентифікації та авторизації

Обробка винятків (рис. 2.13) налаштована таким чином, щоб перетворювати кастомні помилки, згенеровані застосунком, на відповідний HTTP статус-код з повідомленням та додатковими заголовками. У майбутньому також можна реалізувати логування помилок, щоб мати можливість вдосконалювати застосунок.



```
0 references | mishitsa, 15 days ago | 1 author, 1 change
public static class ErrorHandlerExtensions
{
    1 reference | mishitsa, 15 days ago | 1 author, 1 change
    public static void UseErrorHandler(this IApplicationBuilder app)
    {
        app.UseExceptionHandler(appError =>
        {
            appError.Run(async context =>
            {
                var contextFeature = context.Features.Get<IExceptionHandlerFeature>();
                if (contextFeature == null) return;

                context.Response.Headers.Append("Access-Control-Allow-Origin", "*");
                context.Response.ContentType = "application/json";

                context.Response.StatusCode = contextFeature.Error switch
                {
                    BadRequestException => (int)HttpStatusCode.BadRequest,
                    OperationCanceledException => (int)HttpStatusCode.ServiceUnavailable,
                    NotFoundException => (int)HttpStatusCode.NotFound,
                    _ => (int)HttpStatusCode.InternalServerError
                };

                var errorResponse = new
                {
                    statusCode = context.Response.StatusCode,
                    message = contextFeature.Error.GetBaseException().Message
                };

                await context.Response.WriteAsync(JsonSerializer.Serialize(errorResponse));
            });
        });
    }
}
```

Рисунок 2.12 - Конфігурація обробки винятків

Фінальний етап запуску веб-застосунку включає налаштування конвеєра обробки запитів (рис. 2.14). Це, в основному, типовий код, що додає до конвеєра проміжні компоненти для використання Свагеру (UseSwagger()) коли додаток запускається в середовищі розробки, конфігурується Cors, підключає обробку викинутих помилок, включає обробку статичних файлів (UseStaticFiles()), базових механізмів маршрутизації (UseRouting()), збереження даних сесій користувачів (UseSession()), а також функцій аутентифікації (UseAuthentication()), авторизаці ї (UseAuthorization()) та маршрутизації ендпоінтів (UseEndpoints()).

```
1 reference | mishitsa, 15 days ago | 1 author, 1 change
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseSwagger();
        app.UseSwaggerUI();
    }

    var serviceScope = app.ApplicationServices.CreateScope();
    var dataContext = serviceScope.ServiceProvider.GetService<DataContext>();
    dataContext?.Database.EnsureCreated();

    app.UseStaticFiles();
    app.UseErrorHandler();
    app.UseCors();
    app.UseRouting();

    app.UseAuthentication();
    app.UseAuthorization();

    app.UseEndpoints(endpoints =>
    {
        endpoints.MapControllers();
    });
}
```

Рисунок 2.14 - Конфігурація конвеєра обробки запитів

2.4 Налаштування запуску та розгортання веб-додатку

Для ефективного розгортання та запуску вебдодатку було обрано інструмент Docker, який надає широкі можливості оскільки це засіб для розробки та управління контейнерами – це легковагові, ізольовані середовища, які розміщують додаток та всі його модулі, а також набори їх внесень і конфігурацій. Ці підходи знижують ризики конфліктування між додатками, оскільки застосовуються контейнери, які в свою чергу не конфліктують з іншими процесами або програмами на одному хості.

Однією з ключових переваг використання Docker є здатність забезпечувати консистентність середовищ для роботи додатків. Завдяки цій технології, можна бути впевненим, що додаток працюватиме однаково на будь-якому сервері чи комп'ютері, незважаючи на відмінності в налаштуваннях операційної системи чи іншого програмного забезпечення, що

встановлене на хості. Це суттєво спрощує процес розгортання та тестування, оскільки виключає проблеми, що можуть виникати через несумісність між середовищами розробки та продакшн, і дозволяє розробникам зосередитися безпосередньо на вдосконаленні функціональності додатку, знаючи, що він працюватиме стабільно та надійно в будь-якому оточенні.

Конфігураційний файл, представлений на рисунках 2.15 та 2.16, описує детальні налаштування для запуску вебдодатку "MediaLibrary" та його взаємодії з базою даних SQL Server у Docker. У цьому файлі визначено три основні сервіси, кожен з яких виконує окрему роль в архітектурі додатку: medialibrary.api (бекенд API, що відповідає за обробку запитів та бізнес-логіку), database.server (служба, що відповідає за роботу з базою даних SQL Server) та medialibrary.client (фронтенд, що забезпечує взаємодію користувача з додатком). Така розподілена архітектура забезпечує гнучкість та масштабованість системи, дозволяючи кожному компоненту працювати незалежно, але в тісній взаємодії з іншими частинами додатку.

```yaml
services:
  medialibrary.api:
    environment:
      - ASPNETCORE_ENVIRONMENT=Development
      - ASPNETCORE_HTTP_PORTS=3000
      - ASPNETCORE_HTTPS_PORTS=3001
      - ASPNETCORE_Kestrel__Certificates__Default__Password=123
      - ASPNETCORE_Kestrel__Certificates__Default__Path=/https/media-library.pfx
    container_name: medialibrary.api
    image: medialibrary.api
    build:
      context: .
      dockerfile: MediaLibrary.API/Dockerfile
    ports:
      - "3000:3000"
      - "3001:3001"
    volumes:
      - ~/.aspnet/https:/https:ro
    depends_on:
      database.server:
        condition: service_healthy

  database.server:
    image: "mcr.microsoft.com/mssql/server"
    container_name: database.server
    ports:
      - "1433:1433"
    environment:
      - ACCEPT_EULA=y
      - SA_PASSWORD=SuperPassword123
    volumes:
      - ./sqlserver/data:/var/opt/mssql/data
      - ./sqlserver/log:/var/opt/mssql/log

    healthcheck:
      test: /opt/mssql-tools/bin/sqlcmd -S localhost -U sa -P "SuperPassword123" -Q "SELECT 1" -b -o /dev/null
      interval: 10s
      timeout: 3s
      retries: 10
      start_period: 10s

  medialibrary.client:
    build:
      context: MediaLibrary.API/Client/media-library
      dockerfile: Dockerfile
    image: medialibrary.client
    container_name: medialibrary.client
    ports:
      - "5000:5000"
    depends_on:
      - medialibrary.api
```

Рисунок 2.15 - Файл конфігурації Docker

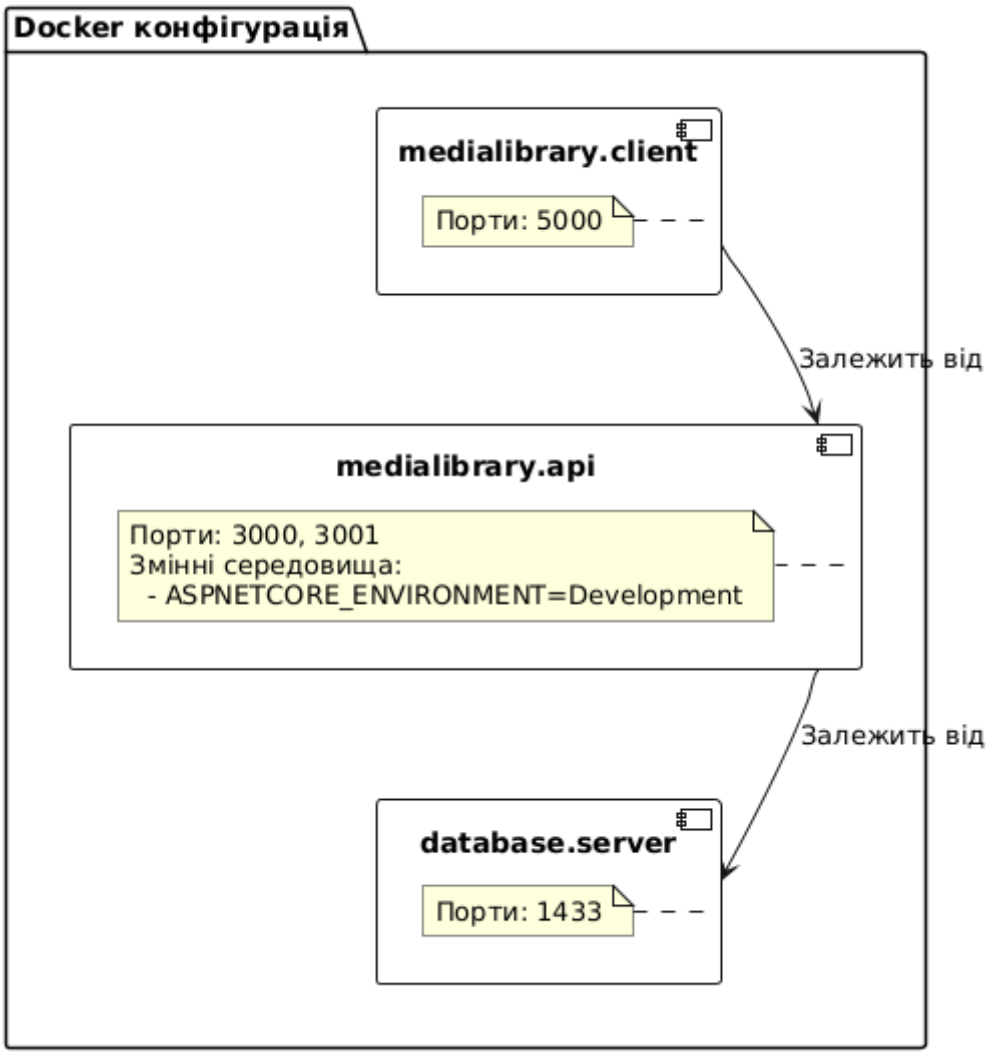


Рисунок 2.16 - UML-діаграма конфігурації Docker

2.5 Розробка базової функціональності веб-додатку

Для реалізації можливостей веб-додатку для управління бібліотеки медіа файлів було прийнято рішення створити REST API контролери для виконання CRUD-операцій з основними сутностями веб-додатку. Це відповідає принципам проєктування веб-додатків на основі фреймворку ASP.NET Core і забезпечує реалізацію основного функціоналу. Також було створено спеціалізований контролер для аутентифікації та авторизації.

Таблиця 2.2 містить інформацію про розроблені контролери та їх ключові функції. Відповідно до вимог, контролери отримують необхідні

сервіси через ін'єкцію залежностей у конструкторі, що дозволяє чітко розподілити обов'язки, забезпечити централізовану заміну реалізацій за потреби та покращити ефективність тестування.

Таблиця 2.2 - Перелік контролерів веб-додатку для управління бібліотекою медіа файлів

№ з/п	Назва контролеру	Призначення
1	UserController	Контролер для реалізації логіки створення аккаунтів(реєстрації), аутентифікації та авторизації
2	OverviewController	Контролер для реалізації CRUD-операцій файлової системи
3	FolderController	Контролер для реалізації CRUD-операцій з папками
4	FileController	Контролер для реалізації CRUD-операцій з файлами. Відповідальний за управління медіа-файлами в зовнішніх сервісах

На рис. 2.17 представлено UML-діаграму, що ілюструє логіку обробки запитів у контролері FileController, який відповідає за реалізацію CRUD-операцій з файлами. З діаграми видно, що контролер використовує патерн проектування Посередник (Mediator). Цей патерн є поведінковим і служить для зменшення зв’язків між численними класами, переміщаючи ці зв’язки до єдиного класу-посередника, що спрощує підтримку та масштабованість коду. За допомогою цього патерна, контролер не має безпосередньої залежності від кожного окремого обробника, що обробляє конкретні запити, а лише звертається до посередника, який, у свою чергу, керує викликами необхідних класів-обробників.

Використання патерна Посередника дозволяє контролеру FileController ефективно знаходити необхідний клас-обробник, який вже реалізує відповідну

бізнес-логіку. Це забезпечує чистоту коду та дозволяє легко змінювати або додавати нові обробники без впливу на інші частини системи. Кожен із зазначених обробників у діаграмі використовує спільні залежності: IUnitOfWork, IFileRepository та IFileStorageService.

IUnitOfWork забезпечує атомарність операцій з базою даних, зменшуючи кількість з'єднань з базою та дозволяючи ефективно обробляти транзакції. IFileRepository відповідає за доступ до даних файлів, виконуючи операції збереження, оновлення, видалення та отримання файлів із бази даних. IFileStorageService є сервісом, що надає можливість зберігати файли на фізичному чи хмарному сховищі, таким чином підтримуючи операції з медіа-файлами.

Обробники мають конструктори, що реалізують механізм впровадження залежностей (Dependency Injection), що значно полегшує тестування та підтримку коду. За допомогою цього механізму, усі необхідні залежності передаються в клас під час його створення, замість того, щоб клас самостійно ініціалізував ці залежності. Такий підхід спрощує модульне тестування, оскільки дозволяє легко замінювати реальні залежності на їхні мок-версії під час тестування.

Всі обробники реалізують метод Task<TResponse> Handle, який приймає модель запиту та токен відміни. Цей метод є асинхронним, що дозволяє ефективно виконувати операції вводу-виводу, не блокуючи потоки. В рамках цього методу обробники виконують валідацію даних, здійснюють CRUD-операції з базою даних, а також взаємодіють із сервісом для зберігання медіа-файлів. Таким чином, усі CRUD-операції і взаємодія з базою даних і зовнішніми сервісами (наприклад, для зберігання файлів) організовані в обробниках, що підвищує ефективність коду та його організацію.

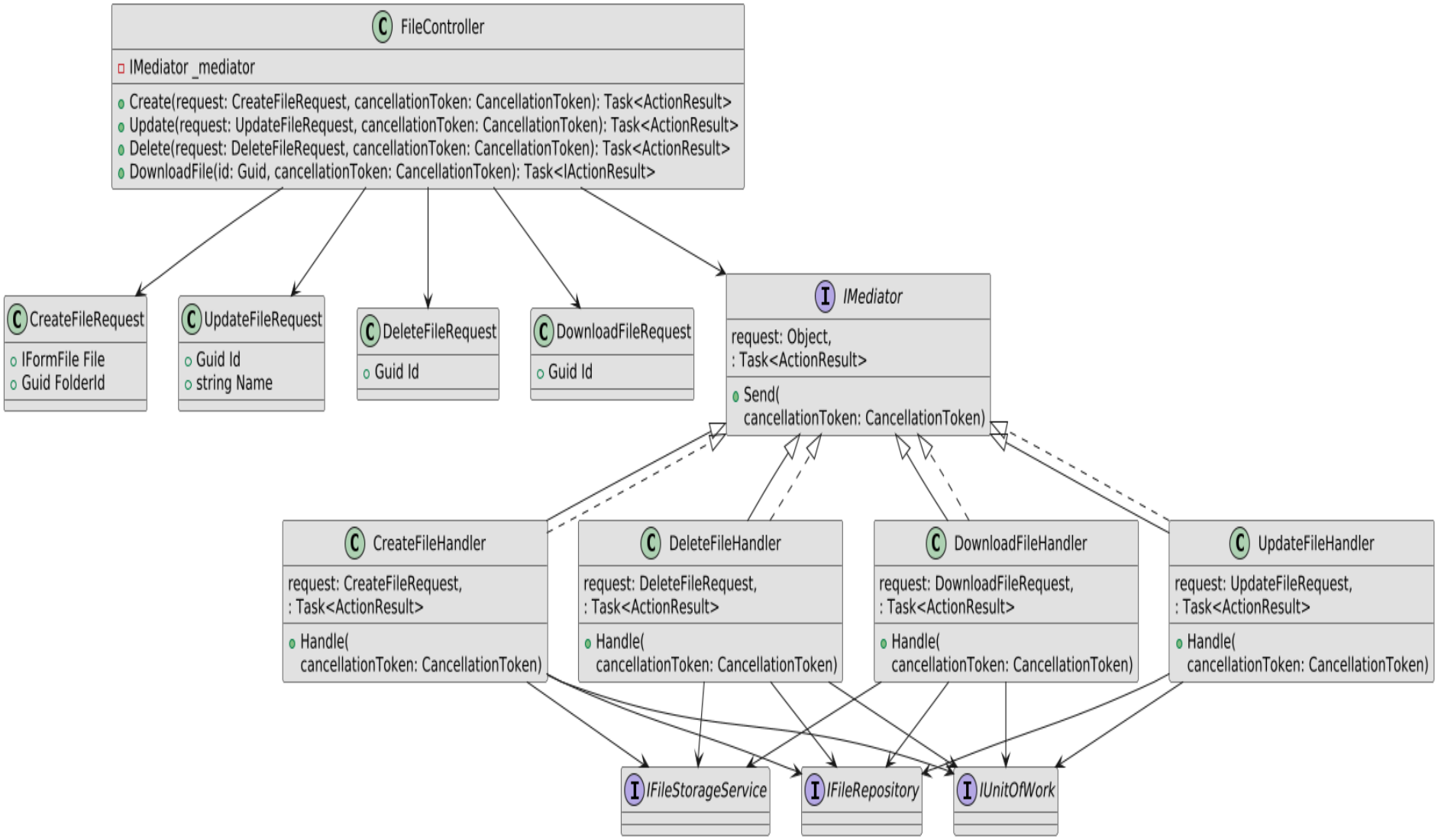


Рисунок 2.17 - UML-діаграма класів контролера FileController

FileController у нашому випадку є контролером, що успадковується від базового класу ControllerBase з фреймворку ASP.NET Core. Він призначений для обробки запитів, пов'язаних із файлами. Контролер має кілька важливих атрибутів:

- AuthorizeAttribute - цей атрибут дозволяє отримувати доступ до контролера лише авторизованим користувачам. Він забезпечує захист, запобігаючи доступу до ресурсів неавторизованим користувачам
- RouteAttribute - встановлює маршрут для всіх кінцевих точок цього контролера. У цьому випадку маршрут визначено як "file/", тому всі кінцеві точки контролера будуть доступні за адресою /file/... .

Кожен контролер містить кінцеві точки API, які обробляють різні типи HTTP-запитів. Ці кінцеві точки також позначаються атрибутами, які можуть вказувати на тип запиту (наприклад, HttpGet, HttpPost, HttpPut, HttpDelete), а також додаткові атрибути, такі як Authorize чи кастомні атрибути. Ось опис основних HTTP-методів:

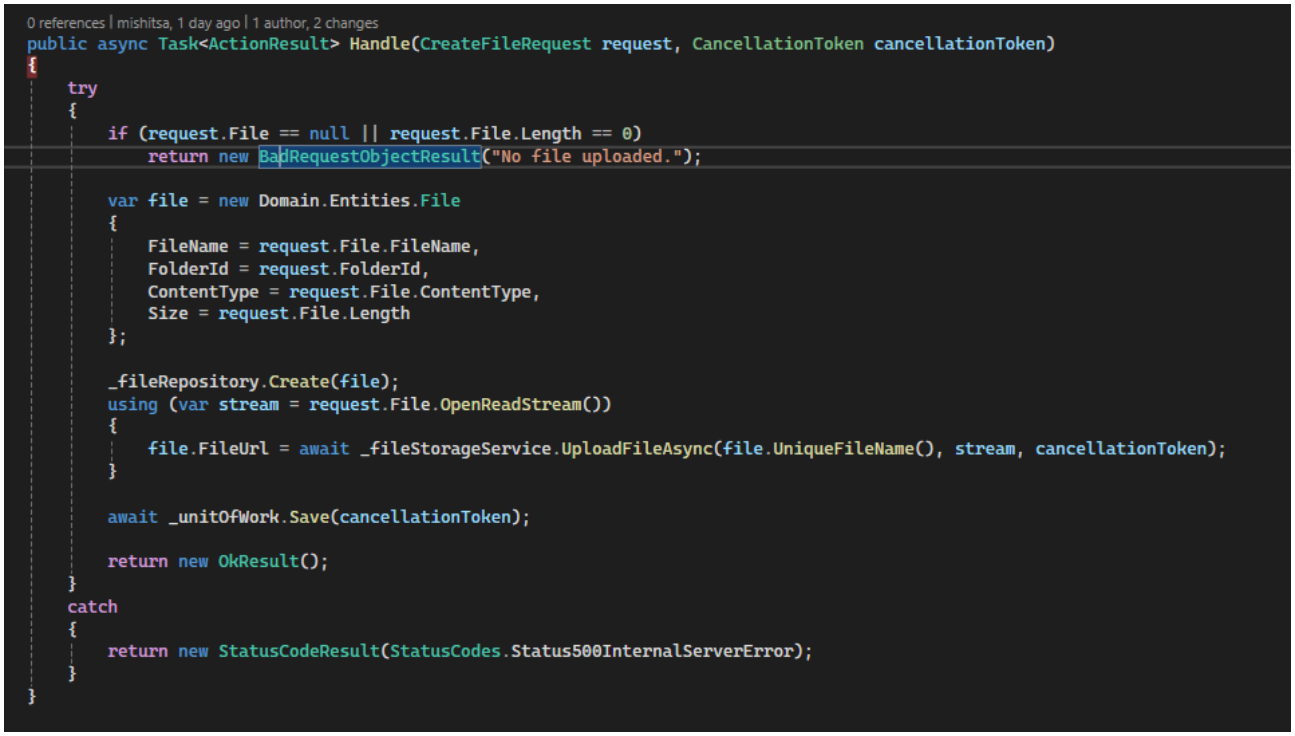
- HttpGet - використовується для отримання інформації. Наприклад, для завантаження файлу або отримання метаданих файлу
- HttpPost - використовується для створення нового ресурсу. Наприклад, для завантаження нового файлу
- HttpPut - використовується для оновлення існуючого ресурсу. Наприклад, для заміни файлу або його метаданих
- HttpDelete - використовується для видалення ресурсу. Наприклад, для видалення файлу

Кожна кінцева точка може приймати об'єкт запиту та токен відміни (CancellationToken). Об'єкт запиту може передаватися різними способами: через рядок запиту (query string), тіло запиту, форму тощо. CancellationToken зазвичай використовується в асинхронних функціях для того, щоб можна було скасувати операцію, якщо результат уже не потрібен.

Для асинхронного виклику обробників контролер використовує конструкцію async/await, що дозволяє ефективно обробляти запити та повертати результат обробки. Результат зазвичай представлений у вигляді задачі Task<T>, типізованої під результат дії, що реалізує інтерфейс IActionResult. Кожен результат має свій статус-код та об'єкт повідомлення. Ось найбільш поширені статус-коди, які повертаються з контролера:

- 200 OK - запит виконаний успішно, результат повернутий у відповіді (наприклад, успішно завантажений файл)
- 204 No Content - запит виконаний успішно, але в відповіді немає вмісту (наприклад, файл видалено)
- 400 Bad Request - сервер не може обробити запит через некоректні дані (наприклад, передано некоректний формат файлу)
- 404 Not Found - ресурс не знайдений (наприклад, файл за вказаним ідентифікатором не існує)
- 500 Internal Server Error - помилка на сервері (наприклад, помилка при збереженні файлу)

На рис. 2.18 зображено код методу, який перевіряє дані, надіслані користувачем у формі. У разі відсутності файлу або якщо файл порожній, повертається статус-код 400 (Bad Request) з повідомленням про те, що файл не завантажено. Далі на основі наданих даних створюється сутність файлу, яка містить інформацію про назву, ідентифікатор папки, тип контенту та розмір файлу. Ця сутність додається до таблиці в базі даних, після чого файл завантажується до відповідного медіа-сервісу для збереження. У випадку, якщо під час виконання коду сталася помилка, користувач отримає відповідь зі статус-кодом 500 (Internal Server Error), що вказує на внутрішню помилку сервера.



```
0 references | mishitsa, 1 day ago | 1 author, 2 changes
public async Task<ActionResult> Handle(CreateFileRequest request, CancellationToken cancellationToken)
{
    try
    {
        if (request.File == null || request.File.Length == 0)
            return new BadRequestObjectResult("No file uploaded.");

        var file = new Domain.Entities.File
        {
            FileName = request.File.FileName,
            FolderId = request.FolderId,
            ContentType = request.File.ContentType,
            Size = request.File.Length
        };

        _fileRepository.Create(file);
        using (var stream = request.File.OpenReadStream())
        {
            file.FileUrl = await _fileStorageService.UploadFileAsync(file.UniqueFileName(), stream, cancellationToken);
        }

        await _unitOfWork.Save(cancellationToken);

        return new OkResult();
    }
    catch
    {
        return new StatusCodeResult(StatusCodes.Status500InternalServerError);
    }
}
```

Рисунок 2.18 - Лістинг методу Handle обробника CreateFileHandler

Рис. 2.19 зображує діаграму послідовності обробки запиту на створення файлу. Діаграма створена з використанням синтаксису PlantUML, який дозволяє наочно відобразити послідовність взаємодії між різними компонентами системи під час обробки запиту.

Процес починається з того, що користувач відправляє HTTP-запит на кінцеву точку контролера для створення файлу. Контролер приймає цей запит і делегує його обробку через патерн посередника (Mediator). Посередник визначає, який обробник (handler) повинен виконати обробку цього запиту, і передає його до CreateFileHandler.

CreateFileHandler спочатку перевіряє, чи був файл дійсно переданий у запиті. У разі відсутності файлу повертається помилка. Якщо файл присутній, створюється об'єкт файлу (File), який передається до репозиторію файлів (IFileRepository) для збереження його метаданих. Далі обробник використовує сервіс зберігання файлів (IFileStorageService) для завантаження файлу до зовнішнього сховища, де для нього генерується унікальний URL.

Після успішного збереження метаданих і завантаження файлу обробник виконує остаточне збереження змін у базі даних через IUnitOfWork. По завершенні всіх операцій обробник повертає результат через посередника до контролера, який надсилає відповідь користувачу. У випадку виникнення помилки під час обробки користувач отримує відповідне повідомлення про помилку.

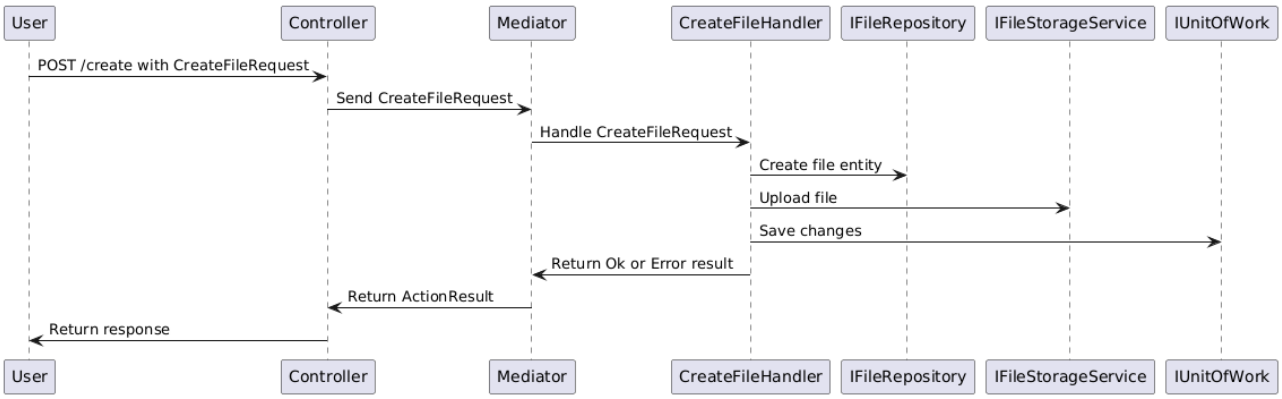


Рисунок 2.19 - UML-діаграма послідовності обробки запиту на створення файлу

У цьому проекті контролери обробляють запити за допомогою пакета MediatR. MediatR – це легка бібліотека, яка реалізує патерн Медіатор, де різні частини програмного забезпечення спілкуються одна з одною, не будучи безпосередньо залежними одна від одної. Патерн Медіатор діє між компонентами так, що ці компоненти не повинні бути безпосередньо пов'язані одне з одним. Розроблений Джиммі Богардом, MediatR став одним із найулюбленіших інструментів серед розробників у .NET, які створюють додатки з складною взаємодією його компонентів.

Використання MediatR дозволяє:

- видалити залежності між модулями;
- сприяти впровадженню чистих архітектурних принципів, таких як CQRS (Сегрегація Запитів та Відповідальностей);
- спростити тестування та підтримку;

- MediatR також пропонує можливість реалізувати патерни запит/відповідь, тобто команди, запити та повідомлення;

Ця бібліотека має наступні переваги:

- розділення відповідальності (Separation of Concerns) - це досягається завдяки спеціалізації обробників запитів та відповідей, у результаті чого класи контролерів та/або самі сервіси не повинні турбуватися про деталі кодування різних операцій, досягнення чистішої та модульної архітектури. Замість того, щоб сервіс безпосередньо викликати інший сервіс, він надсилає свій запит до MediatR, який у свою чергу направляє запит до відповідного обробника. Це усуває безпосередні залежності компонентів і покращує підтримуваність системи;
- CQRS (Сегрегація Запитів та Відповідальностей) - MediatR також широко використовується у додатках, які реалізують патерн CQRS. CQRS передбачає розділення обробки команд (яка модифікує дані) від обробки запитів (яка читає дані), що веде до чіткішої структури коду. У більших додатках це розділення полегшує управління складністю, покращує продуктивність і робить масштабування більш ефективним. MediatR пропонує простий та узгоджений підхід до реалізації операцій CQRS у системах, вказуючи команди та запити як запити з необхідною бізнес-логікою, укладеною в класи обробників;
- поліпшення тестування - медіатор ізолює логіку обробки запитів від основного коду. Це дозволяє обробникам запитів тестуватися в ізоляції від інших компонентів, які здійснюють ці запити, без необхідності мокування сервісів або бази даних, які використовують контролери чи інші вищі рівні;
- масштабованість - оскільки програмна система зростає, завдання підтримки тісного зв’язку різних модулів стає дуже громіздким. Завдяки принципу розділення компонентів, що пропагується патерном Посередник (Mediator), система може зростати еволюційним шляхом. Нові типи запитів можуть бути додані, існуюча функціональність може бути змінена або можуть

бути інтегровані нові сервіси без необхідності великих змін в основній технології;

Медіатор є потужним інструментом, що здатен істотно поліпшити архітектуру додатків, що розробляються на платформі .NET. Дотримуючись патерна «Медіатор», він вносить корективи в відносини між компонентами, впроваджує патерн CQRS, а також дозволяє розробляти більш тестовані та легкі в супроводженні програмні рішення. Єдиною дільницею проекту, що покривається медіатором, будь то проект середньої складності або великий додаток, його інтеграція з MediatR полегшує реалізацію коду в майбутньому.

На рис. 2.20 показано принцип роботи бібліотеки MediatR в даному випадку в додатку .NET Core.

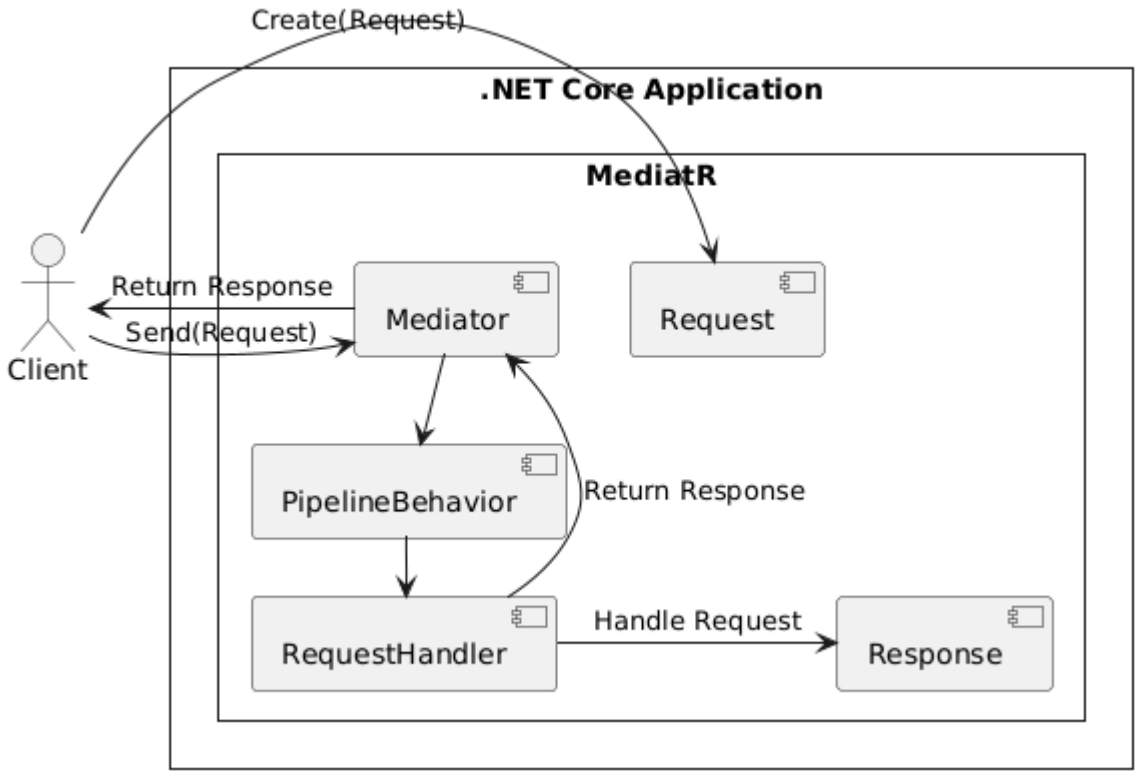


Рисунок 2.20 - UML-*діаграма* принципу роботи бібліотеки MediatR

На рис. 2.21 зображена схема, на якій розташована система, що представляє собою запис файлів на різні сервіси зберігання, реалізуючи патерн "Стратегія". Інтерфейс IFileStorageService описує основні методи завантаження, викладання, видалення файлів і отримання URL. Конкретні

реалізації цього інтерфейсу, такі як AzureFileStorageService, S3FileStorageService та LocalFileStorageService, виконують функції конкретних реалізацій цього інтерфейсу, в залежності від особливостей цього виду сховища.

Коли клієнтський код ініціює запит, система спочатку звертається до конфігурації, щоб отримати тип сховища, вказаний у налаштуваннях. На основі цього вибору, система підключає відповідну реалізацію IFileStorageService. Наприклад, якщо обране сховище – Azure, то використовується AzureFileStorageService для виконання запиту. Якщо ж вибрано S3 чи локальне сховище, то відповідно підключаються S3FileStorageService або LocalFileStorageService.

Після отримання запиту, такого як завантаження або видалення файлу, запити користувача управляються відповідною службою зберігання і, у разі необхідності, URL файлу надсилається назад до клієнта. Такий підхід забезпечує гнучкість, оскільки дозволяє легко змінювати зберігання без переробки логіки в коді клієнта завдяки чітко визначеному інтерфейсу та взаємозамінним стратегіям.

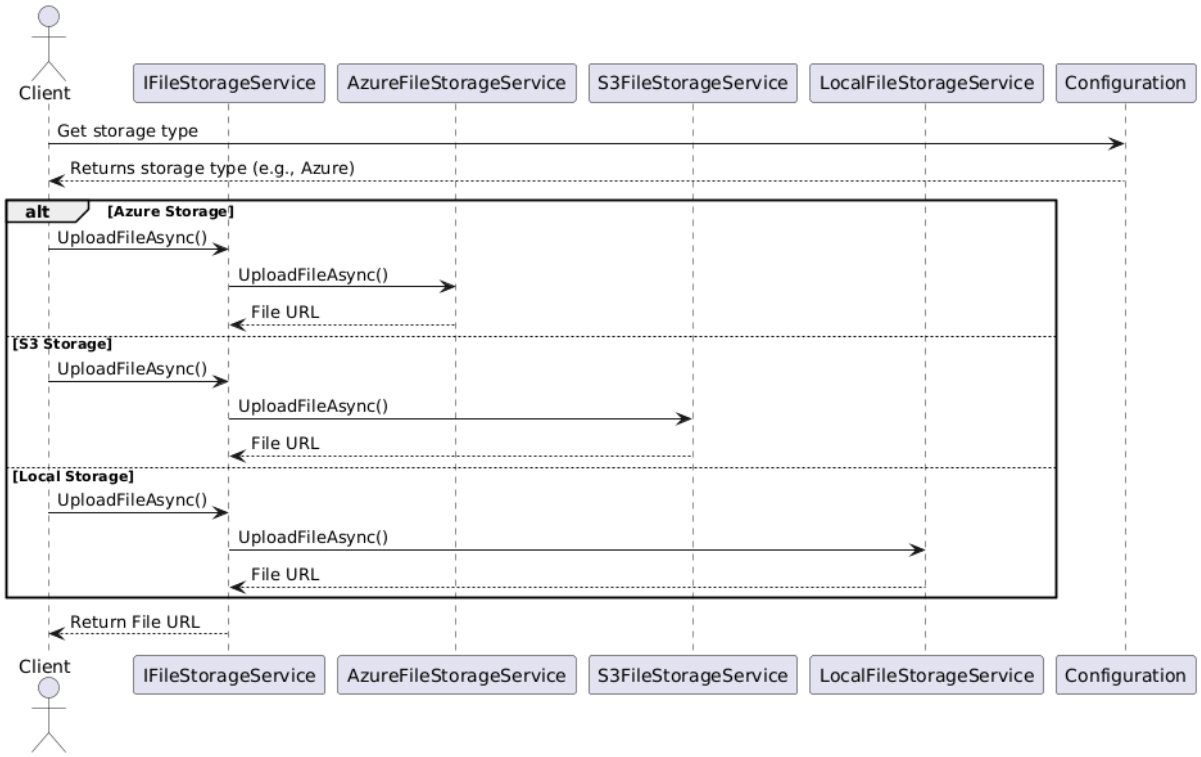


Рисунок 2.21 – UML-*діаграма* взаємодії веб-додатку з сервісами зберігання файлів

Висновки до розділу:

Для розробки веб-додатку для управління бібліотекою медіа файлів було виконано вибір та обґрунтування технологій реалізації. На основі аналізу альтернатив вирішено, що доцільно використовувати веб-фреймворк ASP.NET Core та React з TypeScript, що забезпечить високу продуктивність та достатню зручність у роботі як на серверній, так і на клієнтській стороні. На базі вибору цих технологій вдалося отримати надійний, масштабований та простий в розширенні додаток.

Обґрунтовано застосування бібліотек об’єктно-реляційного відображення (ORM) - Entity Framework, що забезпечує зручну та ефективну роботу з базою даних, з істотним зменшенням написання складних SQL запитів, та забезпечуючи високий рівень абстракції для маніпуляцій з даними. Також вибір був зроблений на користь бібліотек для аутентифікації та авторизації (Microsoft Identity), що дозволяє реалізувати новітні механізми

захисту користувачів. Також використовється бібліотека для побудови графічного інтерфейсу (Tailwind), що забезпечує швидку розробку адаптивних та привабливих інтерфейсів з мінімальними затратами часу.

Було розроблено концептуальну модель сутностей предметної області та зв’язків між ними, що дозволило чітко визначити основні компоненти системи та їх взаємодії. Це дало можливість побудувати ефективну структуру даних, яка відповідає вимогам функціональності веб-додатку. Виконано візуалізацію моделі у вигляді діаграми класів UML, що дозволило наочно представити зв’язки та структуру даних у системі, а також полегшило подальшу реалізацію логіки взаємодії між компонентами.

На основі розробленої моделі було реалізовано проєктування структури бази даних для зберігання необхідної інформації. Окрім того, була врахована можливість масштабування системи в майбутньому, що дозволить додавати нові функціональні можливості та розширювати архітектуру без значних змін в існуючій реалізації.

Таким чином, обрані технології та концептуальна модель дозволяють побудувати стабільний, гнучкий і масштабований веб-додаток для ефективного управління медіафайлами.

РОЗДІЛ 3
ПРАКТИЧНА АПРОБАЦІЯ Веб-ДОДАТКУ ДЛЯ УПРАВЛІННЯ БІБЛІОТЕКОЮ МЕДІА ФАЙЛІВ

3.1 Практична апробація та опис методики використання розробленої панелі управління бібліотекою медіа файлів

З метою продемонструвати функціональність розробленого веб-додатку для управління бібліотекою медіа файлів, розглянемо послідовність операцій та завдань, які вирішує створений застосунок.

Для початку, усі користувачі повинні пройти процедуру реєстрації, яка передбачає заповнення реєстраційної форми (рис. 3.1), за допомогою якої формується обліковий запис у системі. У процесі заповнення форми необхідно ввести електронну пошту, пароль та підтвердити пароль повторним введенням (рисунок 3.1). Якщо користувач вже зареєстрований, йому відкритий доступ до системи через авторизаційну сторінку.

Форма реєстрації надає можливість створити обліковий запис лише за умови, що поле для введення електронної пошти відповідає встановленим стандартам. Зокрема, введена адреса повинна містити символ «@» та крапку, що є обов’язковими елементами валідної електронної адреси. Наступним етапом перевірки є порівняння введених паролів для забезпечення їхньої відповідності один одному. У разі збігу паролів здійснюється додаткова перевірка на відповідність вимогам безпеки. Ці вимоги включають наявність латинських літер, символів верхнього та нижнього регістрів, цифр, а також спеціальних символів.

У випадку, якщо форма заповнена некоректно, користувач отримає повідомлення з описом помилки реєстрації. Це дозволяє забезпечити

зворотний зв’язок і вказати на необхідність виправлення помилок, тим самим підвищуючи зручність користування системою та рівень її захисту.

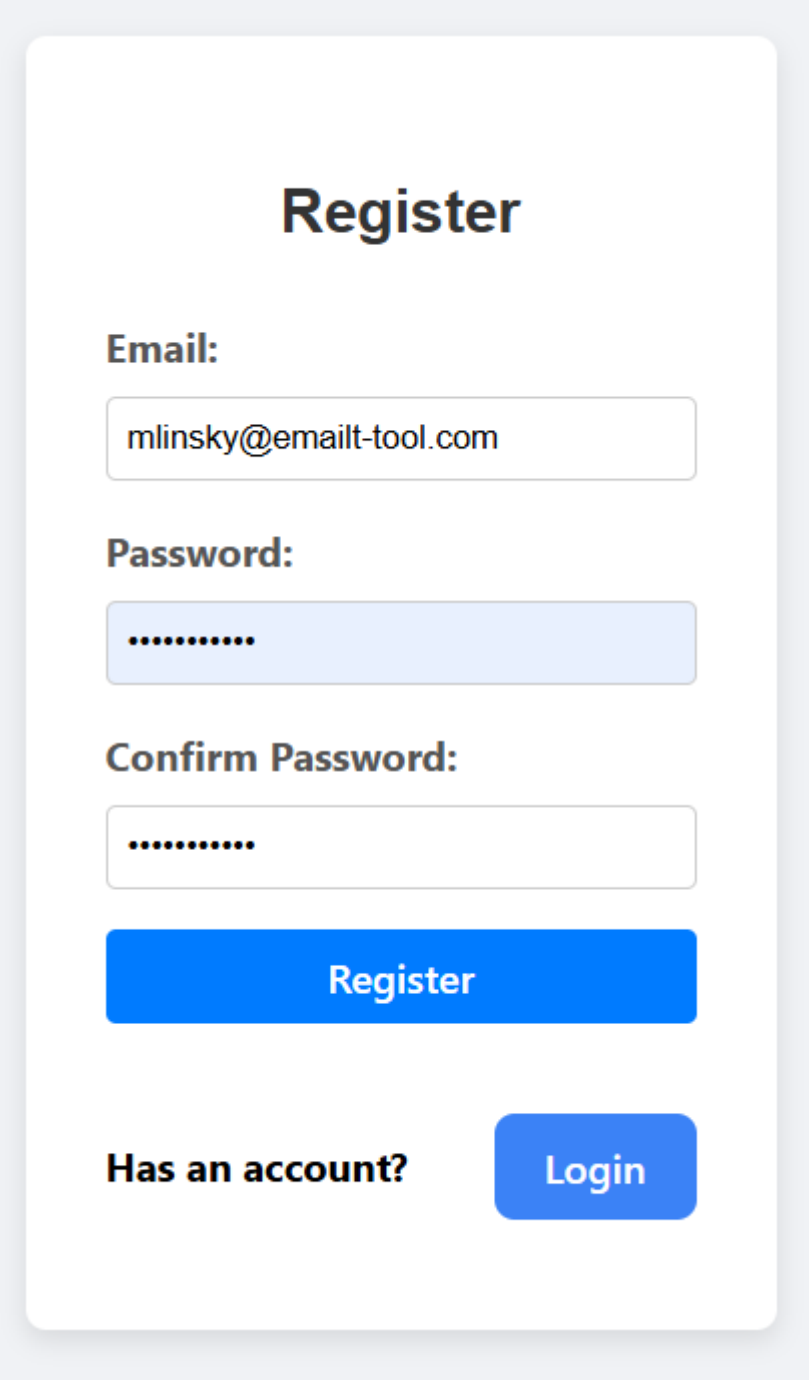


Рисунок 3.1 - Сторінка реєстрації нового аккаунту користувача в веб-додатку

Після успішного створення аккаунту користувач переходить до сторінки авторизації (рис. 3.2). Веб-сторінка входу в систему забезпечує користувачів можливістю повернення до системи за допомогою своєї облікової записи. Для цього надано форму, яка передбачає заповнення двох полей: електронної

пошти та пароля. У разі, якщо користувач ще не має облікового запису, на сторінці передбачено посилання, яке дозволяє перейти до форми реєстрації.

Під час авторизації система перевіряє введені дані на відповідність раніше зареєстрованим обліковим записам. Поле для електронної пошти має відповідати стандартним вимогам, а введений пароль повинен співпадати із тим, що був вказаний під час реєстрації. У разі, якщо будь-який із параметрів введено невірно, користувач отримує відповідне повідомлення про помилку. Це повідомлення інформує про причину відмови у доступі, що може бути корисним для виправлення неточностей, наприклад, у разі помилкового введення паролю.

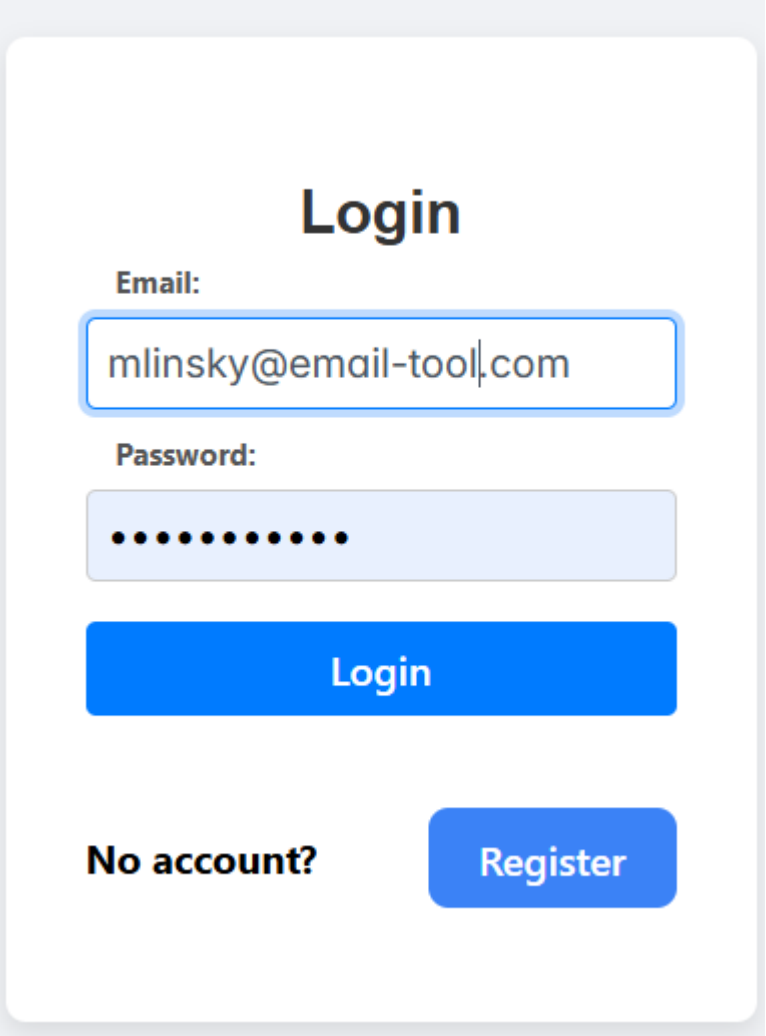


Рисунок 3.2 - Сторінка входу в систему за допомогою вже існуючого аккунту

В результаті успішної авторизації користувач одразу ж переадресовується на головну сторінку(рис. 3.3), котра є доступною виключно для авторизованих користувачів. Дана сторінка є ядром системи, на ній відображаються відомості про завантажені файли, які кореспондують з обраною папкою.

Головна сторінка відкриває перед користувачами можливість переглядати, сортувати та розташовувати файли в межах своїх можливостей в рамках окремих папок. Вона також містить засоби для виконання різних операцій у відношенні до медіафайлів, а саме – завантаження нових, редагування інформації, чи видалення непотрібного. Зовнішній вигляд даної сторінки був спроектований для комфортної і продуктивної роботи з файлами, забезпечуючи ефективність використання даної системи.

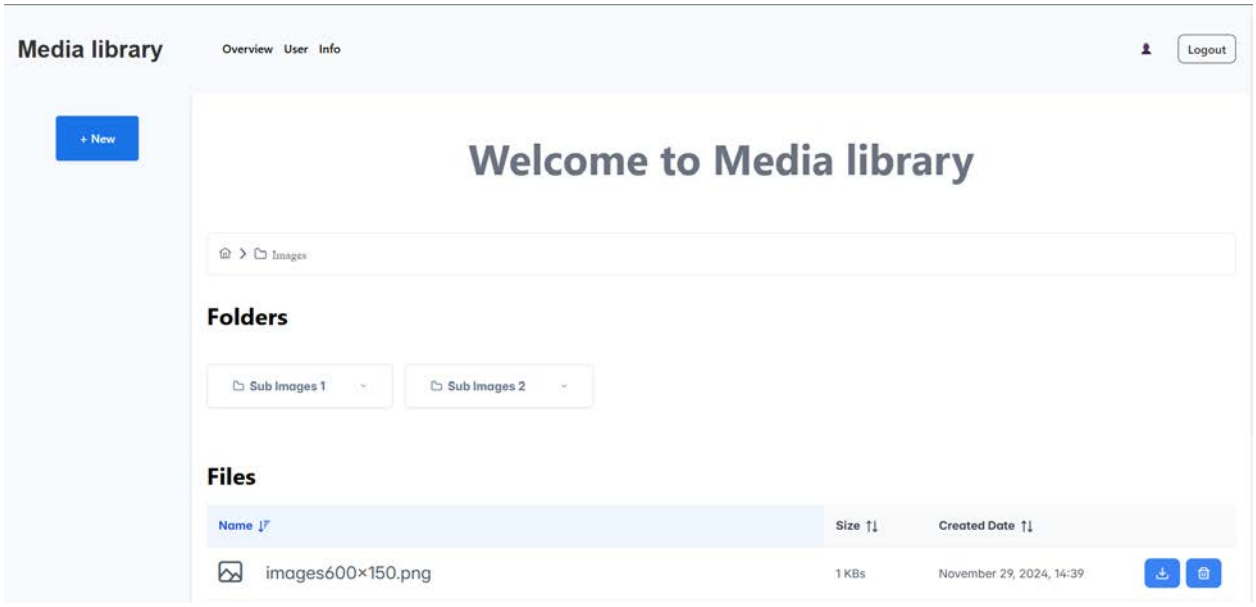


Рисунок 3.3 - Головна сторінка

Всі сторінки веб-додатку, включаючи головну, містять заголовок (рис. 3.4), який виконує кілька важливих ролей. На лівій частині заголовка розташована назва програми, яка виступає як ідентифікатор системи і допомагає в брендуванні. Заголовок є клікабельним і перенаправляє користувача на домашню сторінку.

Поруч із назвою розташоване навігаційне меню, яке дозволяє швидко дістатися до ключових частин програми. Структура меню є простою, щоб навіть нові користувачі не мали проблем із пошуком необхідної інформації.

У верхньому правому куті заголовка розташована кнопка виходу. Вона дозволяє користувачу завершити сесію і, таким чином, захистити обліковий запис користувача від зловмисників.

Аспект заголовка забезпечує однорідність зовнішнього вигляду програми, одночасно роблячи організацію інтерфейсу узгодженою і зручною для користувача.

Рисунок 3.4 - Загальний заголовок сторінок

Компонент огляду папок (рис. 3.5) забезпечує відображення створених папок, а також підпапок, що знаходяться всередині відкритої папки. Цей елемент інтерфейсу виконує ключову роль у навігації файловою структурою, дозволяючи користувачеві швидко орієнтуватися серед наявних каталогів.

Компонент побудований таким чином, щоб відображати ієрархічну структуру, що полегшує пошук необхідної папки чи підпапки. Користувач може легко розгортати та згортати вміст окремих папок для детального перегляду їх вмісту.

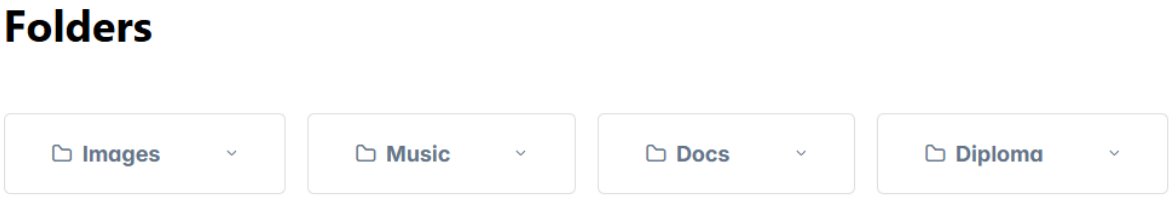


Рисунок 3.5 - Огляд папок

Кожна папка в системі має вбудоване контекстне меню (рис. 3.6), яке надає користувачеві можливість виконувати основні операції, такі як редагування, видалення або інші дії, що стосуються управління цією папкою. Це меню є важливим елементом інтерфейсу, оскільки забезпечує швидкий доступ до найбільш використовуваних функцій без необхідності переходу до інших сторінок або розділів системи. Завдяки контекстному меню, всі необхідні інструменти для роботи з папками та їх вмістом знаходяться під рукою, що суттєво полегшує взаємодію з системою та економить час користувачів.

Користувач може здійснити будь-яку з доступних операцій безпосередньо в контекстному меню, що дозволяє зберегти ефективність і зручність роботи. Так, наприклад, редагування або видалення папки можна здійснити одним кліком, без необхідності здійснювати складні маніпуляції або переходити на інші екрани. Це значно спрощує процес управління файлами та каталогами, роблячи його інтуїтивно зрозумілим і швидким.

Після виконання будь-якої операції користувач одразу отримує відповідне повідомлення про статус виконаної дії. Якщо операція завершена успішно, система відображає повідомлення про успіх, що підтверджує, що всі зміни були внесені коректно та без помилок. Це дозволяє користувачам бути впевненими в тому, що їхні дії виконано правильно, і що всі операції були завершені успішно.

У разі виникнення будь-якої помилки під час виконання дії, система надає користувачеві повідомлення про помилку, яке містить опис проблеми та, якщо це можливо, рекомендації щодо її вирішення. Таке повідомлення не лише інформує про наявність помилки, а й допомагає зрозуміти її причину, що дає змогу користувачу вжити необхідних заходів для виправлення ситуації. Це дозволяє зберегти комфорт користувача при роботі з системою, адже він отримує чітку інформацію щодо стану своїх дій та можливі кроки для вирішення виниклих проблем.

Загалом, контекстне меню кожної папки значно покращує взаємодію з інтерфейсом, забезпечуючи швидкий доступ до функцій редагування та видалення, а також дозволяючи ефективно інформувати користувача про статус виконаних операцій.

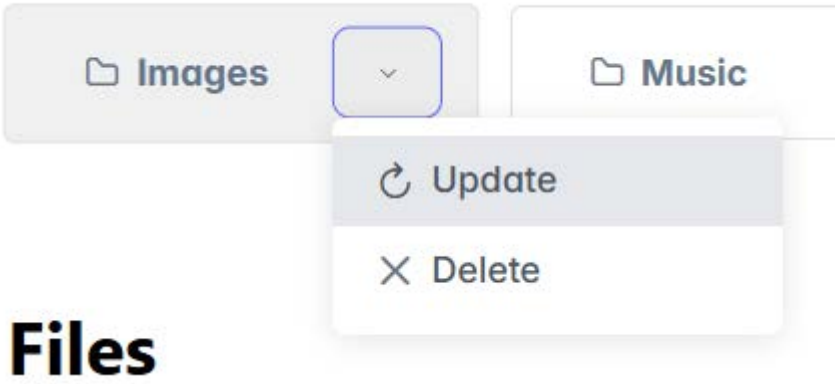


Рисунок 3.6 – Контектсне меню папок

Навігаційне меню відкритих папок (рис. 3.7) є невід'ємною частиною інтерфейсу користувача, яке призначене для відображення ієрархії папок, що в даний момент відкриті, і забезпечує зручність користування системою. Це меню дозволяє користувачам чітко бачити, на якому етапі навігації вони знаходяться і швидко орієнтуватися в структурі даних. Важливою функцією навігаційного меню є відображення структури відкритих папок у вигляді ієрархічного дерева, що включає не лише поточну папку, а й всі папки, що передують їй в ієрархії. Такий підхід дозволяє користувачам краще розуміти контекст, в якому вони працюють, і забезпечує додаткову зручність при роботі з великою кількістю папок і файлів.

Кожен елемент меню відображає відповідає якійсь конкретній папці, починаючи з кореневої і закінчуючи поточною відкритою папки. Це дозволяє користувачам бачити, як побудована система зберігання файлів, і швидко орієнтуватися в ній. Навігаційне меню також інтерактивне, тобто кожен елемент є гіперпосиланням. Натискання на будь-яку з папок дає змогу миттєво

перейти до неї або до файлів, що містяться в ній. Така функціональність забезпечує швидкий доступ до необхідної інформації, дозволяючи користувачам без зусиль переміщатися між різними рівнями структури. Це усуває необхідність вручну шукати потрібні папки або повертатися до попередніх етапів навігації, що значно економить час і робить роботу з системою набагато ефективнішою.

Окрім цього, можливість миттєвого переходу між папками дозволяє уникнути витрат часу на додаткові дії, що призводить до підвищення загальної продуктивності користувача. Якщо користувачі хочуть повернутися до минулих рівнів ієрархії або переключитися між іншими частинами структури даних, вони можуть зробити це з мінімальними витратами часу і зусиль. Така функціональність забезпечує значну гнучкість у навігації та підвищує комфорт використання інтерфейсу, особливо в складних системах з великою кількістю папок і файлів.

Таким чином, навігаційне меню відкритих папок не лише відображає структуру ієрархії файлів, а й активно сприяє зручному і швидкому переходу між папками. Завдяки інтерактивним елементам та чітко структурованій навігації, користувач може без проблем переміщатися по системі, економлячи час і зусилля. Це є важливим аспектом для підтримки високої продуктивності при роботі з великими обсягами даних.

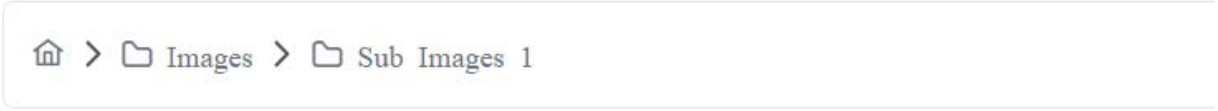


Рисунок 3.7 - Навігаційне меню відкритих папок (Breadcrumb)

Компонент огляду файлів (рис. 3.8) є важливою частиною інтерфейсу системи, який надає користувачам змогу ефективно переглядати та управляти медіафайлами, що були завантажені до конкретної папки. Цей компонент було

реалізовано у вигляді таблиці, яка є засобом для систематизації великих об’ємів файлів, тому, користувачеві надається можливість швидко отримувати потрібну інформацію і вчиняти з файлами певні дії. Таблиця структуровано побудована для того, щоб забезпечити максимальну зручність користування та полегшити процедуру взаємодії з даними.

Таблиця, що містить відомості про файли, має чотири основні колонки, кожна з яких відповідає за відображення певного виду інформації, що є важливою для користувача при роботі з файлами. Перший стовпець - це значок типу файлу, який є візуальним представленням категорії файлу, будь то зображення, відео, аудіофайли, документи тощо. Значок корисний для користувача в розумінні типу файлу, з яким він має справу, без необхідності дивитися на розширення файлу чи його назву..

Наступний стовпець - це назва файлу та його розширення, яке є текстовим описом конкретного файлу. Цей стовпець дозволяє користувачеві легко розрізняти файли, надаючи їм його повну назву разом із розширенням, яке позначає тип файлу, наприклад .jpg, .mp4, .pdf і так далі. Це важливо для користувачів, оскільки спрощує роботу з файлами, особливо коли немає необхідності відкривати файл або вручну визначати його тип.

Третьою колонкою є величина файлу. У цій колонці відображається розмір файлу в кілобайтах або мегабайтах, що дає користувачеві уявлення про обсяг даних.

Четверта колонка зображує дату завантаження, яка дозволяє відстежити, коли файл був доданий до системи. Це може бути корисно для організації файлів та перевірки їх актуальності.

В кінці кожного рядка таблиці знаходяться кнопки для виконання дій над конкретним файлом. Кнопка завантажити дозволяє користувачеві завантажити файл на свій локальний пристрій для подальшої роботи, а кнопка видалити дає змогу прибрати файл з системи. Після виконання будь-якої з цих операцій система надає користувачеві відповідне повідомлення про статус виконаної дії,

інформуючи про її успішне завершення або надаючи відомості про виниклі помилки. Це забезпечує зручність і прозорість у взаємодії з користувачем.

Кожна з колонок таблиці має вбудовану функцію сортування, що дозволяє користувачам впорядковувати дані за різними критеріями. Це може бути сортування за назвою файлу, за розміром чи за датою завантаження. Можливість сортувати файли дозволяє швидко знайти потрібний елемент серед великої кількості даних, що є особливо корисним при роботі з великими обсягами інформації.

Не менш важливою функцією таблиці є можливість поділу файлів на кілька сторінок, так зване, пагінування. Це, безумовно, підвищує зручність перегляду, оскільки без пагінації всі файли начебто розташовані на одній сторінці, що може нести велике навантаження на інтерфейс. Завдяки пагінації користувач може перемикатися між сторінками, які містять обмежену кількість файлів, що забезпечує швидку навігацію та полегшує роботу з великою кількістю даних. Кількість файлів, які відображаються на одній сторінці, може бути налаштована в залежності від потреб користувача.

Таким чином, цей компонент не тільки надає можливість ефективного відображення файлів у таблиці, але й пропонує розширені функції для їх порядкування та маніпулювання. З допомогою сортування, пагінації та інтуїтивно зрозумілих кнопок для завантаження та видалення файлів користувач має можливість швидко і зручно працювати з великими обсягами медіафайлів, що зберігаються в системі.

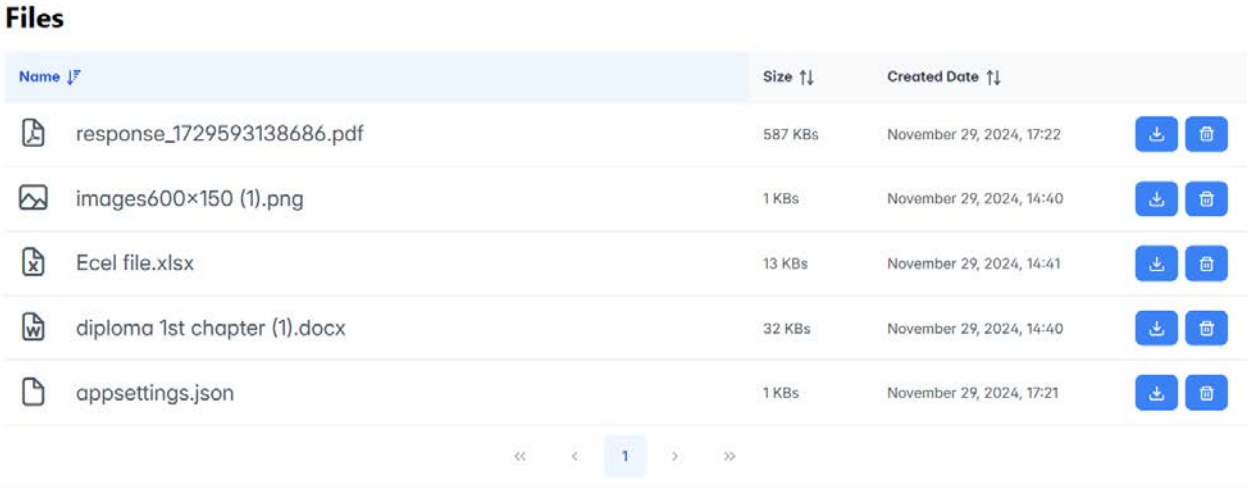


Рисунок 3.8 - Огляд файлів

Збоку інтерфейсу розташована кнопка дії (рис. 3.9), натискання на неї відкриває випадаюче меню з додатковими функціями. Це меню надає користувачеві можливість виконати певні важливі операції, такі як створення нової папки або завантаження медіафайлу. Такий спосіб розташування дозволяє користувачам виконувати певні функції, не перериваючи основний робочий процес, оскільки це легко і зручно.

Кнопка і меню розроблені таким чином, щоб забезпечити інтуїтивно зрозумілу навігацію, дозволяючи користувачам без зайвих зусиль виконувати необхідні дії. Створення нової папки за допомогою цього меню дає можливість користувачу краще структуризувати дані ієрархічно, тоді як варіант завантаження дозволяє додавати матеріали в систему без порушення робочих процесів. Користувач може виконувати ці операції, не переходячи на різні сторінки або вкладки, що заощаджує багато часу і підвищує загальну ефективність взаємодії з інтерфейсом.

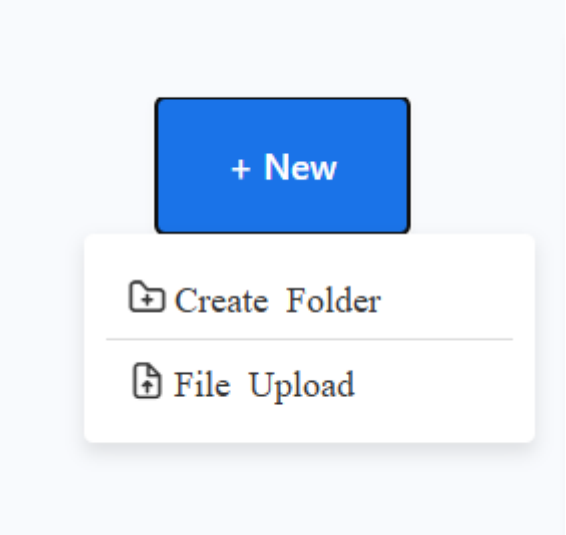


Рисунок 3.9 - Кнопка дії

Кнопка завантаження файлів є важливим елементом в інтерфейсі користувача, бо вона відкриває стандартне діалогове вікно для вибору файла (рис. 3.10), яке забезпечує операційна система. Це вікно є звичною справою для такого роду діяльності, оскільки користувачі його практично зможуть використовувати для пошуку потрібних файлів у системі, їх вибору для подальшого завантаження. Стандартний інтерфейс, що викликається, дає можливість користувачеві швидко переміщуватися по папкам і файлами в тому числі, що полегшує процес пошуку необхідного файла для завантаження швидкого завантаження.

Після того як користувач обирає файл і підтверджує вибір, система ініціює процес завантаження обраного файлу на сервер, де він обробляється та інтегрується в базу даних веб-додатку. Після успішного завантаження дані файлу, такі як назва, розмір і тип, автоматично відображаються в таблиці на відповідній сторінці. Ця таблиця служить наочним засобом для користувача, надаючи йому змогу легко переглядати медіафайли, що знаходяться в обраній папці. Завдяки вказаному підходу користувач може управляти файлами, швидко додавати нові дані та оперативно їх переглядати.

Отже, файли отримують можливість простоти добавлення до системи без додаткових зусиль, а вже в самій системі можна легко ними управляти

виключно в межах заданої структури даних. Це досить зручно для роботи з такими веб-додатками, як було вже зазначено, які активно використовують процес завантаження файлів.

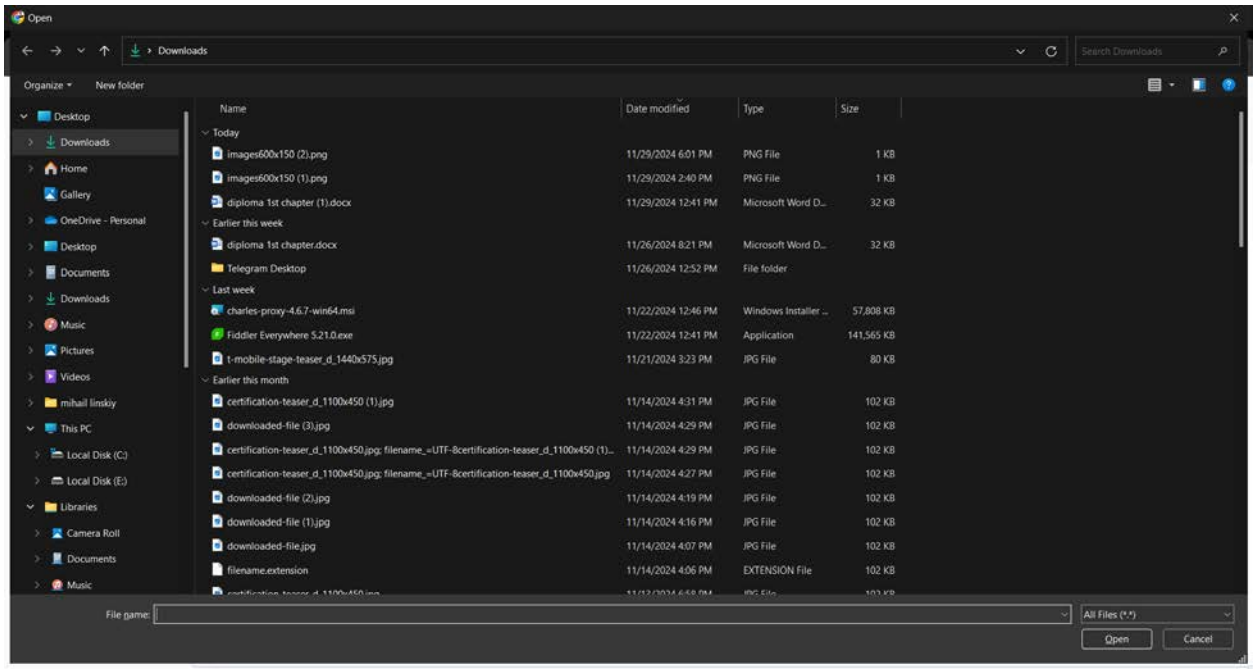


Рисунок 3.10 - Вікно операційної системи для вибору файлу

Модальне вікно для створення нової папки (рис. 3.11) є графічним елементом системи, у якому користувач може створювати нові папки. Вікно, що з’являється у вигляді спливаючого елемента на основному екрані, з’являється при натисканні відповідної кнопки меню, що веде до створення нової папки.

На вікні присутня форма, де користувач вводить назву папки у текстове поле. У разі, коли користувач задовольняється ім’ям заданого текстом папки, є можливість підтвердити його створення, натиснувши на кнопку з відміткою «Створити». Кнопка «Відмінити», що представляє собою X, скасовує дії, які, зрозуміло, буде дієвим, стосовно закриття вікна не створюючи папки.

Після виконання даного сценарія, при натисканні на кнопку “Створити” в разі успішного створення папки система відображає повідомлення про успіх Це повідомлення може з’являтися безпосередньо після закриття всього модального вікна, сповіщаючи про те, що нову папку було успішно створено і

нею можна користуватися. Це повідомлення може з’являтися безпосередньо після закриття всього модального вікна, сповіщаючи про те, що нову папку було успішно створено і нею можна користуватися.

Таким чином, це модальне вікно є важливим елементом для організації роботи з папками, забезпечуючи зручний і зрозумілий процес створення нових структур даних у системі.

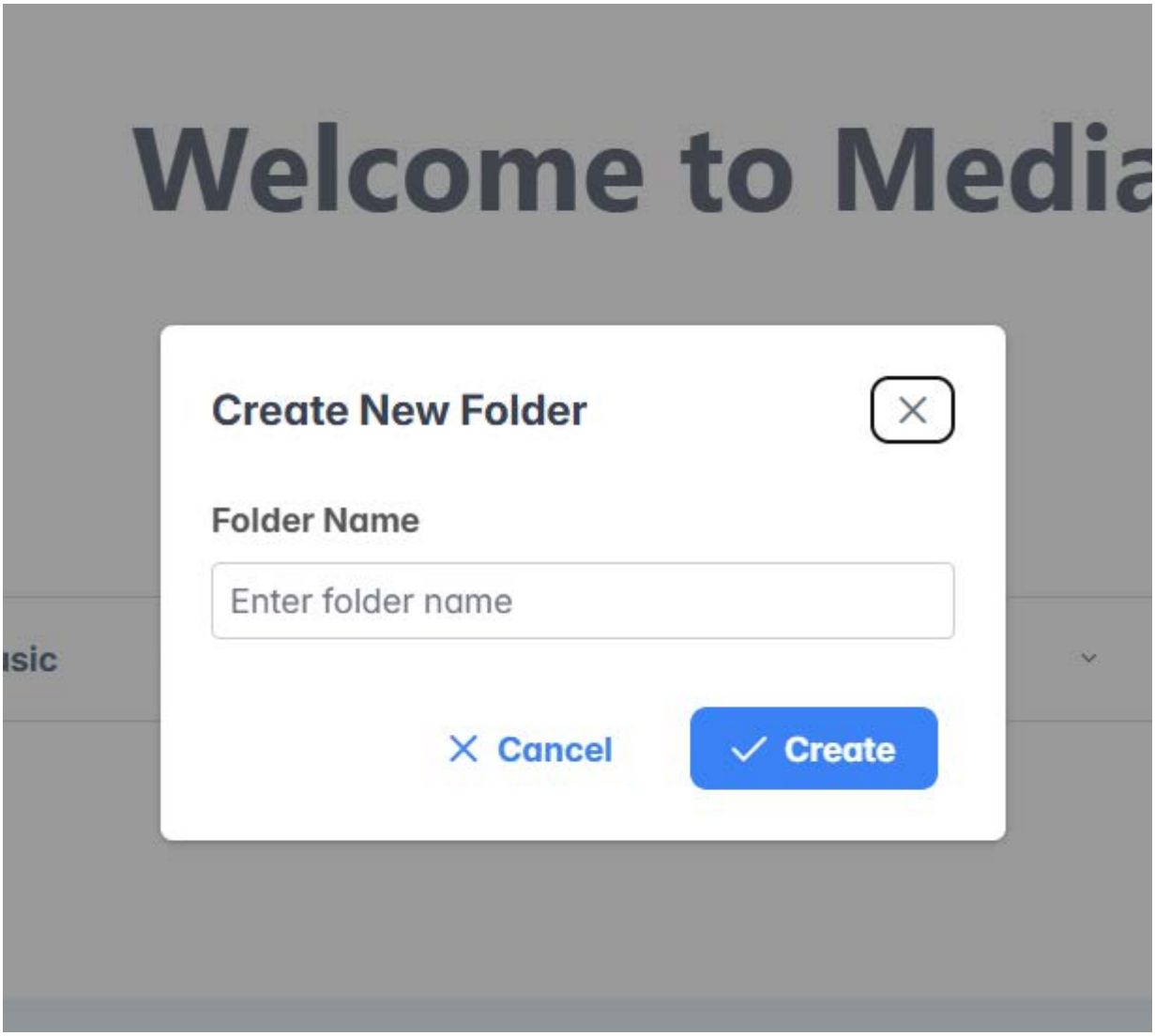


Рисунок 3.11 - Модальне вікно створення папки

Висновки до розділу

Цей розділ розглядає практичну реалізацію веб-додатку для управління медіа-файлами, який включає в себе ряд основних функцій, включаючи навігацію по папках, управління файлами, створення облікових записів та вхід в систему, а також форми управління файлами.

Одним з основних компонентів додатку є таблиця, яка містить усі медіа-контенти, що завантажуються у відповідні папки. Таблиця включає чотири основні колонки: значок типу файла, ім'я та розширення файлу, розмір і дату завантаження. Така структура дозволяє користувачу легко знаходити потрібний файл, шукаючи його за метаданими. Користувачі мають можливість сортувати медіа-контенти за кожною колонкою, що дозволяє організувати файли згідно з різними критеріями, спрощуючи роботу з великими обсягами даних. Крім того, таблиця включає пагінацію, яка дозволяє користувачам переглядати файли групами та зменшує навантаження на інтерфейс, покращуючи продуктивність.

Веб-додаток також надає механізм створення облікових записів та системи входу, які допомагають користувачам отримувати унікальні профілі для роботи з медіа-файлами. Процес реєстрації є простим і зрозумілим, і після реєстрації користувач може увійти, що дозволяє йому входити до особистої бібліотеки файлів. Це не тільки дозволяє зберігати дані окремо для кожного користувача, а отже, обмежує доступ лише для авторизованих користувачів, підвищуючи рівень безпеки даних та конфіденційності, але також є більш ефективним.

Більше того, наявність інтуїтивно зрозумілих елементів інтерфейсу, таких як кнопки для завантаження файлів, створення нових папок та модальні вікна для іменування папок, спрощує та робить управління медіа-файлами без зусиль. Після кожної операції, незалежно від її успіху або невдачі, користувач отримує інформацію про її статус, що додає зручності та прозорості в додатку.

ВИСНОВКИ

Перша частина розглядає та оцінює наявні хмарні сервіси, пов’язані з організацією та управлінням медіа-ресурсами. Їх функції, переваги та недоліки описані. Особливий інтерес становлять можливості реалізації веб-додатку для управління медіа-контентом з використанням хмарних сервісів, таких як Azure та Amazon Web Services. У зв'язку з проведеним аналізом було вирішено розробити власний додаток. Під час реалізації вимоги до програмного забезпечення, а також технічні специфікації для реалізації проекту були чітко визначені

Друга глава презентує вибір і обґрунтування технологій для реалізації веб-додатку, призначеного для управління бібліотекою медіа-ресурсів. Зокрема, обґрунтовані способи реалізації веб-фреймворку ASP.NET Core 8 та архітектурного патерну Односторінковий додаток (SPA). Для реалізації бази даних була використана бібліотека об'єктно-реляційного відображення ORM Entity Framework, а для забезпечення безпеки використовувалися інструменти автентифікації та авторизації з Microsoft Identity. Інтерфейс користувача було реалізовано за допомогою бібліотеки React з мовою TypeScript, тоді як для стилізації компонентів використовувалися Sass і Tailwind CSS.

У рамках роботи розроблена концептуальна модель сутностей предметної області та взаємозв'язків між ними і візуалізована у вигляді UML класової діаграми. На основі цієї моделі була спроектована структура бази даних для забезпечення ефективного збереження та обробки інформації.

Друга частина роботи присвячена реалізації розробленого веб-додатку. Описуються, документуються основні модулі та описується метод їхньої інтеграції з зовнішніми сервісами (AWS, Azure та інші) в обраній архітектурній моделі програми у вигляді односторінкового додатку (SPA). Як засіб спрощення розгортання системи передбачено створення Docker з метою автоматизації процесу інсталяції і конфігурування програми в контейнерах.

У останньому розділі представлені результати апробації розробленого веб-додатку для управління бібліотекою медіа файлів. Описані основні етапи впровадження та реалізації програмного засобу, що дає можливість зосереджено реалізувати ресурсне завантаження, ресурсне зберігання, ресурсну модифікацію та ресурсне управління медіафайлами в інтерактивному просторі. За допомогою цього додатка користувачі мають можливості безпечно та зручно здійснювати основні операції з файлами, які включають в себе, поки що, такі, як додавання, редагування, видалення, та, найголовніше, організацію медіа контенту в папки.

При проектуванні інтерфейсу додатку були враховані фактори для забезпечення максимального зручності та інтуїтивності використання додатку. Користувачам не потрібно витрачати багато часу на освоєння функціоналу, оскільки всі основні операції доступні через прості й зрозумілі елементи керування інтерфейсом. У системі передбачено функціонал для створення папок, нових файлів, метадані файлів, пошук, що дозволяє ефективно знаходити ресурси. Крім того, важливою частиною додатку є система сповіщень, яка простим способом інформує користувачів про виконані дії або про виниклі помилки, тим самим цементуючи зручність і прозорість взаємодії з платформою.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Overview of ASP.NET. url: https://learn.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-8.0

2. Enity Framework Core. url: https://learn.microsoft.com/en-us/ef/core/

3. Фрімен Адам, Pro Entity Framework Core 2 for ASP.NET Core MVC. APress, 2018.

4. Скіт Дж., C# для профессіоналів. Тонкощі программування. Видавництво Вільямс, 2018.

5. Лок Е. Asp.NET Core in Action, Third Edition. Видавництво Manning, 2023

6. Жолткевич Г. М., Кваша В. В. Архітектура та безпека веб-додатків / Промінь.

7. React. url: https://react.dev/learn.

8. ДСТУ 3008:2015. Інформація та документація. Звіти у сфері науки і техніки. Структура і правила оформлення. Київ, ДП «УкрННЦ», 2015. 26с.

9. ДСТУ 3582:2013. Бібліографічний опис. Скорочення слів і словосполучень в українській мові. Загальні вимоги та правила. Київ, ДП «УкрННЦ», 2013. 23 с.

10. Introduction to Identity ASP.NET Core. url: https://learn.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-8.0&tabs=visual-studio

11. ДСТУ 8302:2015. Бібліографічне посилання. Загальні вимоги та правила складання Київ, ДП «УкрННЦ», 2016. 16 с.

12. Тронь В. В., Маринич І. А. Методичні вказівки до виконання магістерської кваліфікаційної роботи для студентів спеціальності 174 - Автоматизація, комп’ютерно-інтегровані технології та робототехніка”. Кривий Ріг: Видавничий центр КНУ, 2022. 50 с.

13. ДСТУ 3651.0-97 Метрологія. Одиниці фізичних величин. Основні одиниці фізичних величин Міжнародної системи одиниць. Основні положення, назви та позначення Київ, Держстандарт України, 1998. 27 с.

14. Plant UML. url: https://plantuml.com// (дата звернення – 10.10.24).

15. Нестеренко Л. М. Основи веб-програмування на мові JavaScript / Л. М. Нестеренко // ЛНУ ім. Івана Франка, 2020. С. 45-57.

16. Cohn Martin. Database Management for Web Applications // Morgan & Claypool

17. Overview of Single Page Apps (SPAs) in ASP.NET Core. url: https://learn.microsoft.com/en-us/aspnet/core/client-side/spa/intro?view=aspnetcore-8.0

18. Рамський Ю.С. Проектування і опрацювання баз даних: Посібник для вчителів / Ю.С. Рамський, Г.Ю. Цибко. – Тернопіль: Навчальна книга – Богдан, 2005. – 116 с.

19. М.В. Грайворонський, О.М. Новіков Безпека інформаційно-комунікаційних систем. Підручник. Видавнича група BHV, К. 2009.

20. Сучасні комп’ютерні технології обробки інформації. Практичний посібник. І.О. Яковлева, О.В. Шматко, Л.В. Гусева, О.О. Паніна., –Х., 2006 – 272 с.

21. Керівництво з безпеки веб-додатків: принципи та практика. Підручник / О.М. Петров, К.М. Василенко. – Київ: Освіта України, 2019. – 180 с.

22. Introduction to Web APIs. url: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side_web_APIs/Introduction

23. Паттерни проектування для розробки веб-додатків / Р.М. Патраков, А.В. Гаврилюк. – Одеса: Політехперспектива, 2020. С. 74-101.

24. Foulds I. Learn Azure in a Month of Lunches. Manning, 2018

25. Azure Blob Storage documentation. url: https://learn.microsoft.com/en-us/azure/storage/blobs/

26. Get started with Azure Blob Storage and .NET. url: https://learn.microsoft.com/en-us/azure/storage/blobs/storage-blob-dotnet-get-started

27. Integrating Azure Blob Storage with .NET. url: https://medium.com/c-sharp-programming/integrating-azure-blob-storage-with-net-b4fc16dfde73

28. Wittig A., Wittig M., Amazon Web Services in Action. Manning Publications, 2018

29. King T.H. AWS: The Ultimate Guide From Beginners To Advanced For The Amazon Web Services (2020 Edition). Independently published, 2019

30. AWS Documentation. url: https://docs.aws.amazon.com/