ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

Пояснювальна записка

до бакалаврської роботи
на ступінь вищої освіти бакалавр

на тему: «РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ПІДТРИМКИ ПРОЦЕСУ СТРУКТУРУВАННЯ ДОКУМЕНТІВ НА КОМП'ЮТЕРІ МОВОЮ C#»

Виконав: студент 4 курсу, групи ПД–43
спеціальності
121 Інженерія програмного забезпечення
(шифр і назва спеціальності/спеціалізації)

Гуж О.С.
(прізвище та ініціали)

Керівник Коба А.Б.
(прізвище та ініціали)

Рецензент _____
(прізвище та ініціали)

Київ – 2022

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти -«Бакалавр»

Спеціальність підготовки – 121 «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідувач кафедри

Інженерії програмного забезпечення

_____ Негоденко О.В.

“____” _____2023 року

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТА

ГУЖ ОЛЕКСАНДРУ СВЯТОСЛАВОВИЧУ

(прізвище, ім’я, по батькові)

1. Тема роботи: «Розробка програмного забезпечення для підтримки процесу структурування документів на комп’ютері мовою C#»

Керівник роботи: ст. викладач Коба А.Б.

(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом вищого навчального закладу від «24» лютого 2023 року №26.

2. Строк подання студентом роботи «1» червня 2023 року

3. Вхідні дані до роботи

Процес структурування документів; технічна література програмного забезпечення для структурування документів, технічна література C#/.NET, технічні засоби розробки Visual Studio, графічна підсистема WPF, Entity Framework Core;

4. Зміст розрахунково-пояснювальної записки(перелік питань, які потрібно розробити):

4.1 Аналіз предметної області

4.2 Аналіз та порівняння існуючих рішень ПЗ.

4.3 Розробка та тестування ПЗ.

4.4 Висновки.

5. Перелік демонстраційного матеріалу (назва основних слайдів)

5.1. Титульний слайд

5.2. Мета, об’єкт та предмет дослідження

5.3.Аналіз аналогів

5.4.Технічне завдання

5.5.Програмні засоби реалізації

5.6. Діаграма прецедентів

5.7. Діаграми діяльності

5.8. Архітектура застосунку

5.9. Діаграма бази даних

5.10. Діаграма класів

5.11. Криптографічна схема

5.12. Екранні Форми

5.13. Апробація результатів дослідження

5.14. Висновки

6. Дата видачі завдання «025»лютого 2023

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури	19.04	Виконано
2	Вимоги до ПЗ	20.04	Виконано
3	Дослідження програмних засобів	21.04-22.04	Виконано
4	Розробка функціоналу ПЗ	23.04-27.04	Виконано
5	Висновки, оформлення роботи	27.04-30.04	Виконано
6	Розробка обов‘язкових демонстраційних матеріалів	30.04	Виконано
7	Попередній захист роботи	24.05	Виконано
8	Здача роботи	01.06	Виконано

Студент _____ Гуж О.С.

(підпис) (прізвище та ініціали)

Керівник роботи _____ Коба А.Б.

(підпис) (прізвище та ініціали)

РЕФЕРАТ

Текстова частина бакалаврської роботи 100 с., 27 рис., 14 джерел.

СТРУКТУРУВАННЯ ДОКУМЕНТІВ, ЗАСТОСУНОК, DESKTOP, .NET, C#, WPF, MVVM, ENTITY FRAMEWORK

Об'єкт дослідження – Процес структурування документів на комп'ютері.

Предмет дослідження – Програмне забезпечення для підтримки процесу структурування документів мовою C#.

Мета роботи – Покращення процесу роботи з документами за рахунок розширення функціональності програмного забезпечення для підтримки процесу структурування документів.

В роботі розглянуті вимоги до ПЗ для структурування документів та існуючі аналоги.

Проаналізовано можливості середовища розробки .NET, мови програмування C# та графічної підсистеми WPF.

Також був спроектований та розроблений застосунок для підтримки процесу структурування документів.

Галузь використання – застосунок може бути використаним будь-яким користувачем що працює з документами.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ORM – Object-Relation Mapping

ОС – Операційна Система

БД – База даних

SDK – software development kit

API - Application Programming Interface

LINQ – Language-Integrated Query

MVVM – Model View View-Model

WPF – Windows Presentation Foundation

XAML – Extensible Application Markup Language

UI –User Interface

QA – Quality Assurance

ПЗ – програмне забезпечення

СУБД – Система Управління Базами Даних

ВСТУП

Сучасний світ вимагає все більш швидкого та ефективного оброблення великого обсягу документів. Однак, процес структурування документів може бути складним та часоємним завданням, особливо в умовах зростаючої кількості електронних документів та потреби в організації інформації для забезпечення ефективного управління та пошуку даних.

Недостатньо ефективне управління документами може мати серйозні наслідки для будь-якої організації. Наприклад:

– Якщо працівники не можуть легко знайти необхідні документи, це може призвести до затримок та витрат часу на пошук та організацію даних.

– Якщо документи не зберігаються належним чином та не доступні для потрібних людей, можливі втрати важливої інформації.

– Недоступність важливих документів може призвести до помилок у процесах прийняття рішень та роботі з клієнтами.

– Якщо документи не можна знайти, можливе дублювання роботи та витрати на її повторне виконання.

Процес структурування документів є важливою частиною багатьох процесів, пов'язаних з роботою над документами. Цей процес може бути дуже часо та ресурсозатратним, особливо коли розмір документів великий, а їх кількість значна.

Розробка програмного забезпечення для підтримки процесу структурування документів може значно спростити цей процес. Таке програмне забезпечення може надавати користувачам зручний інтерфейс для створення та редагування документів, інструменти для їх структурування та організації, а також безпечного зберігання.

У зв'язку з цим, розробка програмного забезпечення для підтримки процесу структурування документів є важливою та актуальною задачею в галузі інформаційних технологій.

Об’єкт дослідження – Процес структурування документів на комп’ютері.

Предмет дослідження – Програмне забезпечення для підтримки процесу структурування документів мовою C#.

Мета роботи – Покращення процесу роботи з документами за рахунок розширення функціональності програмного забезпечення для підтримки процесу структурування документів.

Завдання роботи – розробка програмного забезпечення для забезпечення процесу структурування документів.

1 ТЕОРЕТИЧНІ АСПЕКТИ СТРУКТУРУВАННЯ ДОКУМЕНТІВ

1.1. **Процес структурування документів**

Збільшення обсягу електронних документів, що зберігаються на комп'ютерах, може призвести до збільшення часу, необхідного для пошуку необхідного документу, або до втрати документів, що може призвести до серйозних проблем. Для ефективної роботи з документами на комп'ютері необхідна система організації та структурування документів.

Структурування документів є особливо важливою для бізнесів та організацій, де зберігаються великі обсяги документів, таких як контракти, фінансові звіти, підручники та інші. Ефективна система структурування та організації документів може забезпечити економію часу та ресурсів організації, а також збільшити її продуктивність.

Структурування документів - це процес організації документів та інших файлів на комп'ютері з метою полегшення доступу до них та забезпечення зручності їх подальшої обробки. Цей процес передбачає упорядкування документів за категоріями, темами, типами або будь-якими іншими параметрами, що їх визначають, і створення системи логічних зв'язків між ними.

Організація документів може включати створення папок, розміщення файлів у відповідних папках, використання міток, тегів або ключових слів для класифікації документів, а також створення зв'язків між документами та іншими файлами.

Структурування файлів документів допомагає зменшити час, який потрібний для пошуку інформації, полегшує взаємодію з документами, забезпечує зручність їх зберігання та резервного копіювання, а також сприяє підвищенню продуктивності та ефективності роботи з документами.

Структурування файлів документів важливо не лише для збереження порядку і організації на комп'ютері, але й для забезпечення ефективності та продуктивності

роботи. Із зростанням кількості документів, що зберігаються на комп'ютері, пошук необхідного файлу може стати дуже складним, а також може виникнути ризик втрати документу в морі файлів.

Структурування файлів документів може включати не лише використання папок та міток, але й створення ієрархії документів, визначення прав доступу, підтримку відстеження версій документів та їхню автоматичну синхронізацію між різними пристроями.

Структурування файлів документів має особливу важливість для бізнесу та організацій, де велика кількість документів зберігається на комп'ютерах та в мережах. Організація документів у таких випадках може включати створення системи документообігу, де документи проходять різні етапи роботи та зберігаються відповідно до їхнього статусу. Це забезпечує збереження порядку та контроль над роботою з документами в організації.

1.2. **Ефективна організація документів на комп’ютері**

Ефективна організація документів передбачає раціональне та логічне розташування їх на комп'ютері або в хмарному сховищі. Для цього можна використовувати наступні принципи:

Створення структури папок та директорій. Найбільш ефективним способом організації документів є створення структури папок та директорій на комп'ютері. Кожна папка повинна мати назву, що відображає її вміст, та розташовуватися у відповідній категорії документів.

Використання ключових слів та тегів. Ключові слова та теги допомагають легко знайти потрібний документ серед великої кількості файлів. Для цього можна використовувати назви файлів та папок, ключові слова в описах файлів та теги.

Регулярне оновлення та сортування документів. Для збереження ефективності організації документів необхідно регулярно проводити оновлення та

сортування файлів. За допомогою цих дій можна видалити застарілі документи, перейменувати або перемістити файли та папки у відповідні категорії.

Створення резервних копій. Для забезпечення безпеки важливих документів необхідно створювати резервні копії файлів та зберігати їх на зовнішньому носії чи в хмарному сховищі.

Використання системи автоматизації організації документів. Для підвищення ефективності організації документів можна використовувати спеціальні програми, які автоматизують процеси збереження та організації файлів на комп'ютері.

2 АНАЛІЗ НАЯВНИХ ЗАСОБІВ ТА ТЕХНОЛОГІЙ ДЛЯ ПІДТРИМКИ ПРОЦЕСУ СТРУКТУРУВАННЯ ДОКУМЕНТІВ

2.1. Аналіз існуючих засобів структуризації документів

Засоби для підтримки процесу структурування документів на комп’ютері можна розділити на структурування за допомогою:

- файлового менеджеру
- хмарного сховища
- спеціалізованого програмного забезпечення

Структурування файлів документів за допомогою файлового менеджеру передбачає створення логічної структури для зберігання та організації файлів на комп'ютері. Для цього можна використовувати вбудовані в операційну систему файлові менеджери, такі як Windows Explorer, macOS Finder або Dolphin.

Основною одиницею структурування є папка (directory), яка може містити інші папки та файли. Папки можна організовувати в ієрархічну структуру, де вище розташовані папки є батьківськими, а нижче - дочірніми.

Наприклад, для організації документів можна створити папку "Документи", а вже в ній створити дочірні папки для різних типів документів, наприклад, "Накази", "Впровадження", "Дипломи". В кожній з цих дочірніх папок можуть міститися файли, що відповідають її назві та типу документів, наприклад, "Наказ№2313_УрядУкраїни.docx", "Впровадження№23112_Олексій.С.Ф.docx", "Диплом_ГужО.С.-ПД43.docx" тощо.

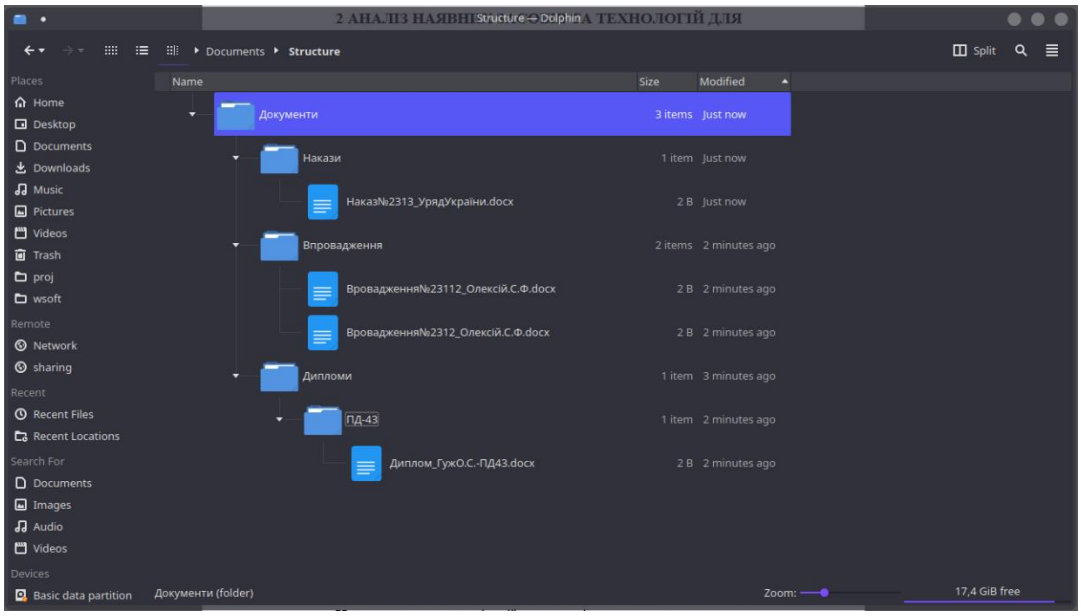


Рис. 2.1. – приклад структуризації за допомогою файлового менеджеру Dolphin

В загальному, структурування файлів документів за допомогою файлового менеджеру є індивідуальним підходом, який залежить від потреб користувача та характеру документів. Однак, важливо дотримуватися логічної та послідовної структури для зручності організації та пошуку необхідних файлів.

Структурування документів за допомогою файлового менеджеру має серйозні недоліки. Коли структура стає складною, важко зберігати логічну послідовність та відслідковувати зміни в структурі. Якщо велика кількість файлів знаходиться в одній папці, то знайти потрібний файл може бути складно. Також пошук файлів за певними критеріями може займати багато часу та зусиль, особливо якщо потрібно шукати у багатьох папках. Якщо файли не зберігаються в правильній папці, або не виконується регулярне резервне копіювання, то існує ризик втрати даних у разі несподіваної помилки або вірусного нападу. Якщо файли містять конфіденційну інформацію, то може бути важко забезпечити їх захист від несанкціонованого доступу, зокрема, якщо користувачі використовують один комп'ютер з загальним доступом до файлової системи.

Хмарні сховища дозволяють організувати та структурувати файли документів у віртуальному просторі, доступному з будь-якого пристрою та з будь-

якого місця з Інтернет-підключенням. Це можна зробити за допомогою створення папок та під-папок, розташування файлів відповідно до їх тематики та категорії, а також надання їм описів та ключових слів для зручного пошуку.

У порівнянні з файловим менеджером, який працює лише на локальному пристрої, хмарні сховища мають ряд переваг. По-перше, вони дозволяють зберігати великий обсяг даних, оскільки віртуальний простір не обмежується місткістю локального диска. По-друге, хмарні сховища забезпечують резервне копіювання та захист даних від видалення або пошкодження, що робить їх більш надійними для зберігання важливих даних. По-третє, хмарні сховища дають можливість легко ділитись документами з іншими користувачами, дозволяючи надавати доступ до файлів за посиланням або через спільний доступ до папки.

Однак, у хмарних сховищах можуть бути недоліки. Наприклад, вони можуть бути повільнішими для роботи з файлами, особливо якщо вони знаходяться великому обсязі. Також, якщо відсутнє інтернет-підключення, доступ до даних буде обмеженим. Крім того, залежно від використовуваного хмарного сховища, можуть бути обмеження на розмір файлів та обсяг збереження даних. Крім того безпека при зберіганні даних на сторонніх сервісах може бути сумнівною

Спеціалізовані програми для управління документами надають більш розширені можливості для структурування та керування документами, ніж звичайні файлові менеджери або хмарні сховища. Основною метою таких програм є полегшення доступу до даних та спрощення процесу роботи з документами в командній роботі.

Структурування файлів документів в програмах для управління документами може включати створення окремих проектів, кожен з яких може містити декілька папок, що відповідають певним тематикам або процесам. Кожна папка може містити документи та файли, що стосуються конкретної теми, з можливістю розширеного пошуку та фільтрації. Крім того, такі програми дозволяють встановлювати права доступу до файлів та папок, що забезпечує контроль над змінами та обмежує доступ до конфіденційної інформації.

Окрім цього, програми для управління документами можуть включати інструменти для контролю версій документів, що дозволяє стежити за змінами та історією документів. Це робить можливим відновлення попередніх версій документів в разі потреби. Розглянемо деякі з них детальніше.

Microsoft SharePoint - це платформа для управління документами та спільної роботи, що дозволяє організаціям створювати веб-сайти та портали для обміну інформацією, спільної роботи над документами та керування процесами бізнесу.

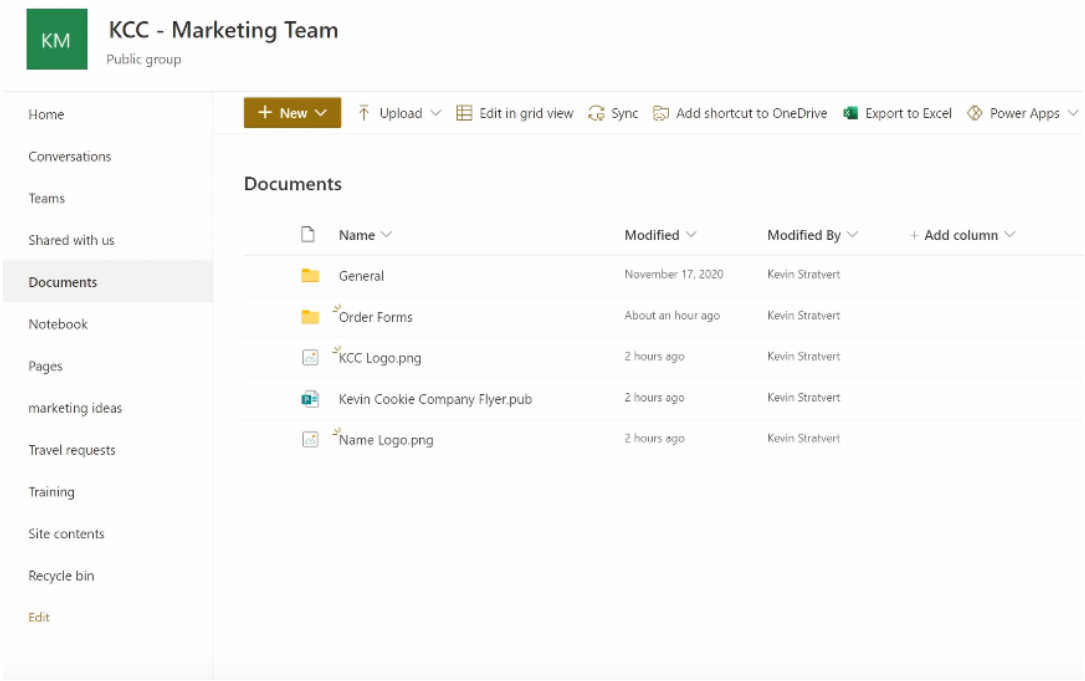


Рис. 2.2. – структура документів в SharePoint

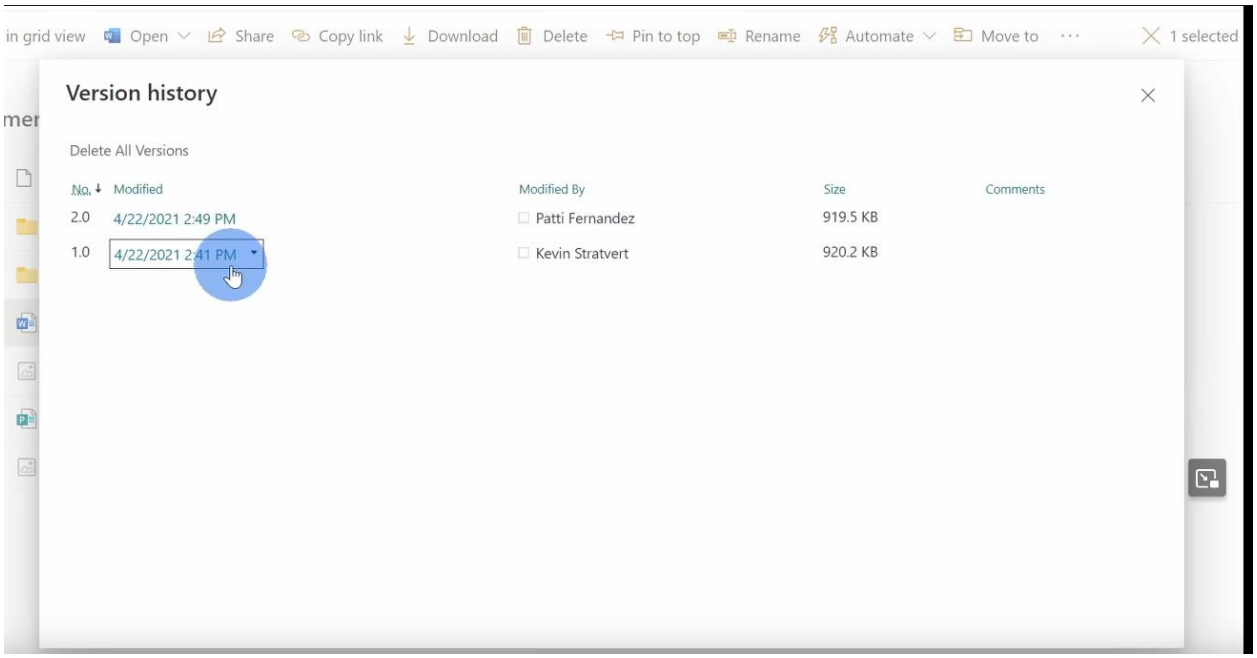


Рис. 2.3. – версії документів в SharePoint

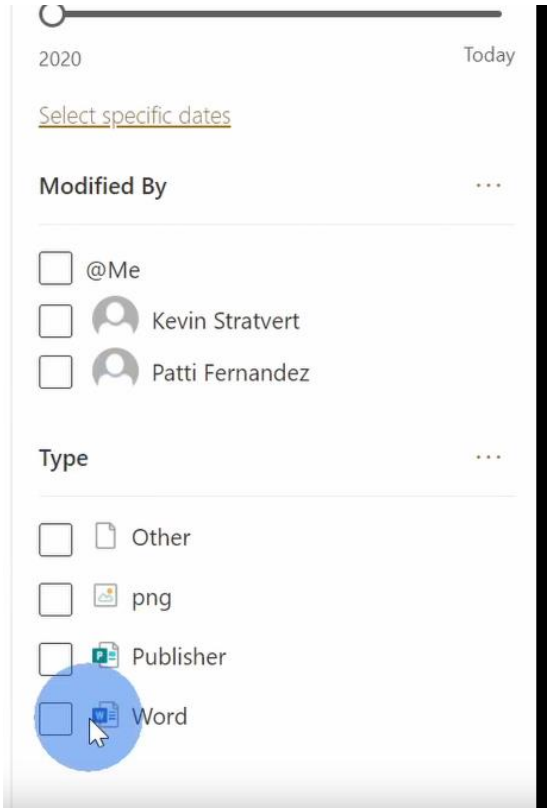


Рис. 2.4. – Пошук документів в SharePoint

Основними можливостями Microsoft SharePoint є:

Управління документами: SharePoint дозволяє зберігати, організовувати та керувати документами в онлайн-середовищі. Користувачі можуть легко знайти необхідні документи, скориставшись пошуковими функціями та фільтрами.

Керування версіями: SharePoint забезпечує можливість контролювати версії документів та відстежувати зміни, внесені до них. Кожен новий варіант документу зберігається в системі з можливістю повернення до будь-якої попередньої версії.

Спільна робота: SharePoint надає можливість спільної роботи над документами та проектами в команді. Користувачі можуть легко ділитися документами, коментувати їх та вести дискусії в контексті документів.

Незважаючи на свої переваги, Microsoft SharePoint має недоліки, серед яких:

– Складність: Хоча SharePoint може бути дуже корисним для організації, він може бути досить складним у використанні, оскільки є дуже комплексним рішенням яке включає в себе безліч функцій які відносяться до бізнес процесів

– Обмеження роботи в офлайн режимі: SharePoint підтримує доступ до документів тільки в Інтернеті

– Функціональні обмеження: Немає можливості структурування вже існуючих не структурованих документів, оскільки їх кількість може бути достатньо велика це може бути проблемою для деяких користувачів.

– Безпека: Оскільки файли зберігаються на серверах сторонньої організації питання безпеки документів може бути спірним.

File Juggler - це програмне забезпечення для автоматизації обробки файлів на Windows, яке дозволяє користувачам створювати складні правила для перетворення, перейменування, переміщення та видалення файлів на основі різних параметрів, таких як розмір файлу, дата створення, тип файлу та інші.

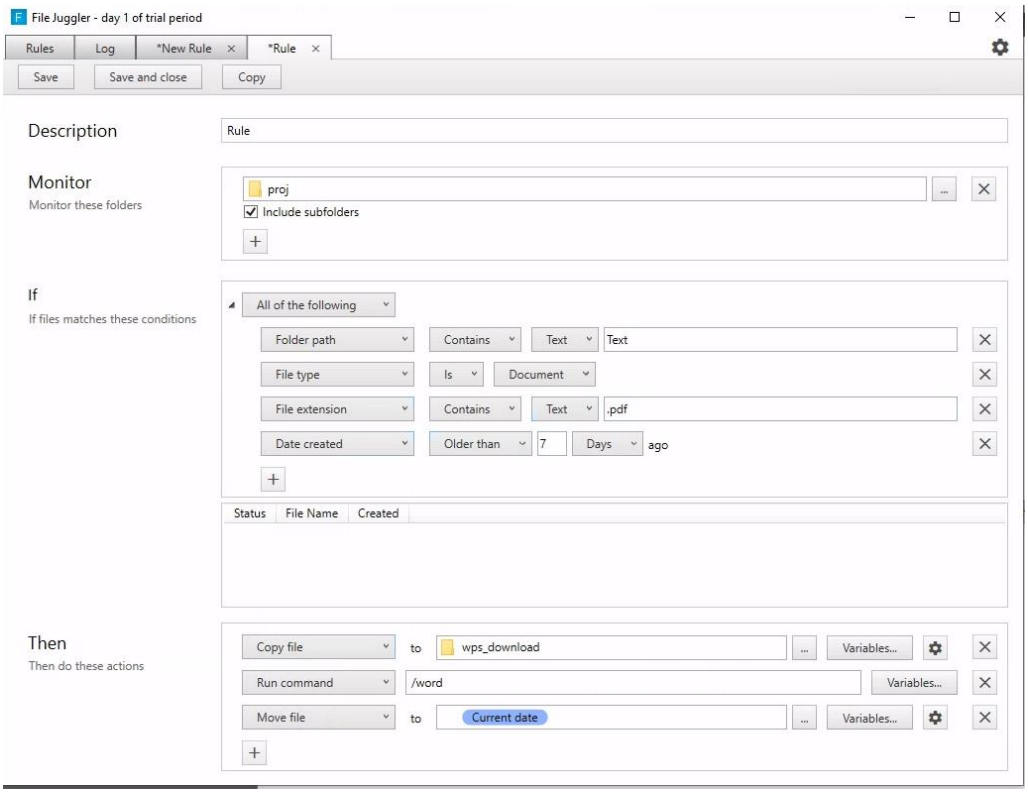


Рис. 2.5. – налаштування правил у File Jugger

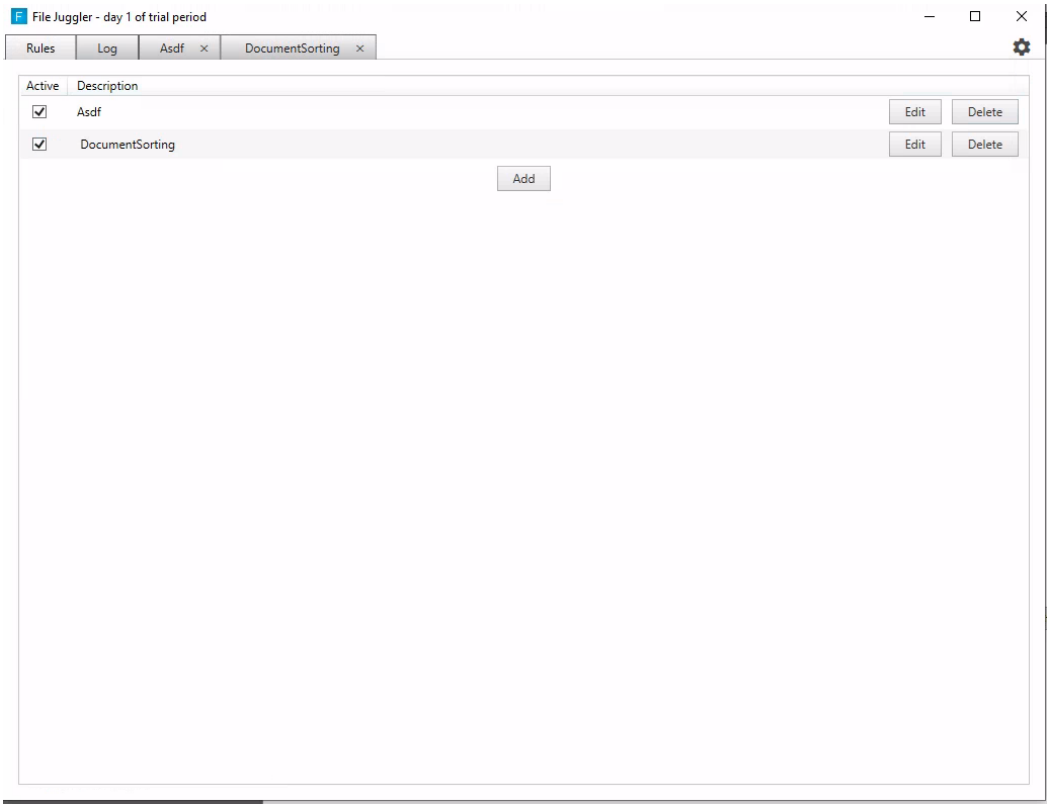


Рис. 2.6. – головне меню File Jugger

File Juggler дозволяє користувачам налаштовувати правила, щоб автоматично виконувати різні дії з файлами, такі як переміщення, перейменування, копіювання та видалення. Це дозволяє забезпечити більш ефективне управління файлами та економію часу.

Користувачі можуть створювати складні правила на основі різних параметрів, таких як розмір файлу, дата створення, тип файлу, певні ключові слова в іменах файлів та інші.

File Juggler підтримує більшість типів файлів, включаючи документи, зображення, аудіо та відео. Це дозволяє користувачам обробляти різні типи файлів за допомогою однієї програми.

До недоліків можна віднести складне налаштування правил структуризації, відсутність резервного копіювання, відсутність захищеного сховища, відсутність версій документів, не має вбудованої структури документів.

DropIt - це безкоштовна програма для автоматичного сортування і обробки файлів на комп'ютері. Вона дозволяє користувачам створювати правила, які будуть автоматично застосовуватися до файлів, які попадають в задані папки. DropIt дозволяє настроювати обробку файлів з використанням шаблонів і ключових слів, що спрощує процес сортування і пошуку файлів.

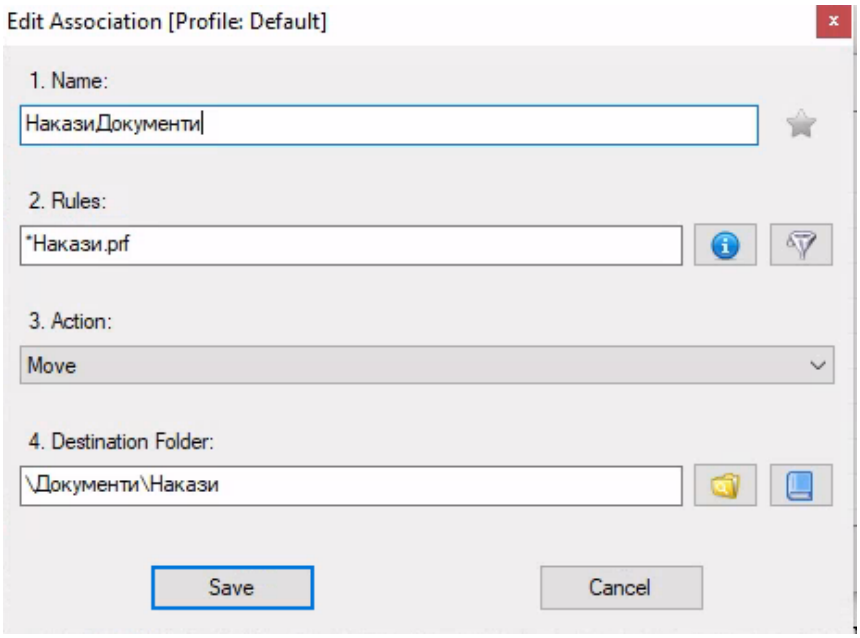


Рис. 2.7. – налаштування правил в DropIt

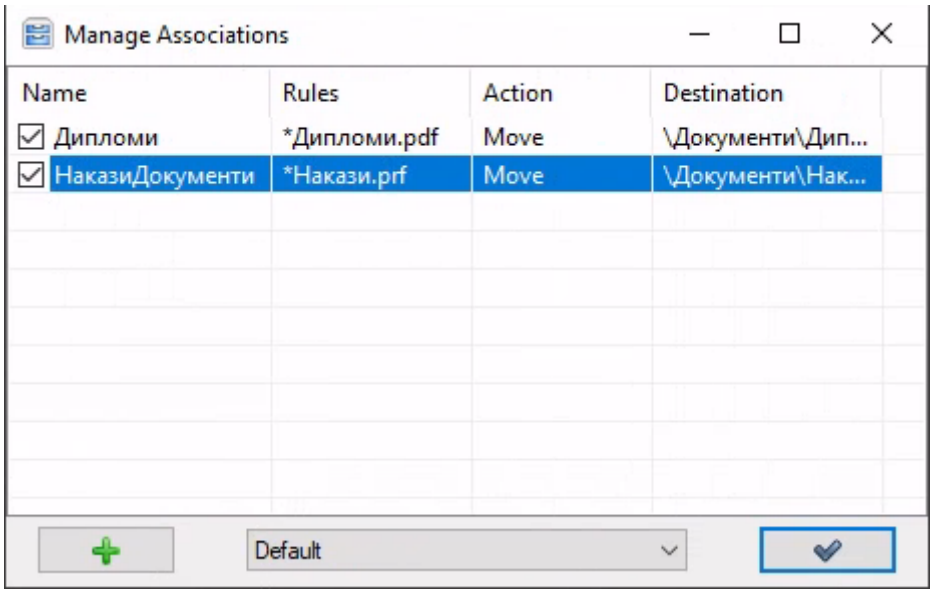


Рис. 2.8. – головне меню DropIt

DropIt дозволяє створювати правила для автоматичного сортування файлів в папки, використовувати ключові слова та шаблони для обробки файлів. DropIt підтримує використання регулярних виразів, що дозволяє створювати більш складні правила для обробки файлів, підтримує сортування і обробку різних типів файлів, таких як текстові, зображення, відео, аудіо та інші.

Незважаючи на те, що програма DropIt має декілька переваг, вона також має деякі недоліки серед них: немає можливості резервного копіювання, відсутні версії документів, складність налаштування правил та відсутність шифрованого сховища, не забезпечує вбудовану структуру документів.

Таблиця 2.1. – Порівняння аналогів

Показник	**SharePoint**	**File Juggler**	**DropIt**
Платформа	Web	Windows	Windows
Відображення структури документів	+	-	-
Створення категорій, папок, тегів документів	+	+	+
Створення правил для автоматичного структурування	-	+	+
Резервне копіювання	+	-	-
Версії файлів	+	-	-
Редагування файлів	+	-	-
Зашифроване сховище	-	-	-

2.2. **Вибір технологій та засобів розробки програмного забезпечення**

Під час розробки програмного забезпечення для підтримки процесу структурування, важним етапом є вибір необхідних інструментів. Для виконання

вимог до ПЗ потрібно вибрати такі інструменти: мову програмування, інтегроване середовище розробки, засіб розробки UI, засіб роботи з БД.

2.2.1. **Об’єктно-орієнтована мова програмування C#**

C# - це мова програмування, яку створила компанія Microsoft у 1999 році в рамках ініціативи розробки платформи .NET. Вона є об'єктно-орієнтованою і часто використовується для створення програм під операційну систему Windows.

C# є мовою програмування загального призначення, яка дозволяє розробникам створювати програми для різних цілей, включаючи веб-розробку, розробку додатків для настільних комп'ютерів та мобільних пристроїв, розробку ігор та різноманітних програмних продуктів.

Синтаксис C# схожий на синтаксис інших мов програмування з родини мов C, таких як C++ та Java, тому для програмістів, які вже володіють досвідом роботи з цими мовами, вивчення C# не буде складним. Однак, C# має свої особливості та можливості, які роблять його унікальним.

Основні особливості C#:

Об'єктно-орієнтований підхід: C# дозволяє програмістам створювати класи, які описують об'єкти, що використовуються в програмі. Це дає можливість створювати більш складні програми з більшою кількістю функцій та операцій.

Система керування пам'яттю: C# використовує систему автоматичного керування пам'яттю, що дозволяє програмістам не звертати увагу на процеси виділення та звільнення пам'яті.

Бібліотека класів .NET: C# має доступ до широкого спектру класів, що забезпечують виконання різноманітних завдань, таких як робота з мережевим протоколами, робота з файлами та базами даних, взаємодія з операційною системою та багато іншого. Ці класи збираються в бібліотеку класів .NET, яка є частиною платформи .NET Framework.

Розширені можливості безпеки: C# має розширені можливості безпеки, що дозволяють програмістам розробляти програми, які захищені від вразливостей та атак.

Підтримка мультиплатформеності: C# може бути виконана на різних операційних системах, включаючи Windows, macOS та Linux. Крім того, C# є основною мовою для розробки програм для платформи .NET Core, що дозволяє створювати програми, які можуть бути виконані на різних пристроях та операційних системах.

Підтримка паралельності: C# має вбудовану підтримку паралельного програмування, що дозволяє програмістам розробляти програми, які можуть виконуватись паралельно на багатьох процесорах або ядрах.

Строга типізація: C# має строгу типізацію, що дозволяє програмістам визначати типи даних, що використовуються у програмі. Це дозволяє запобігти багам, що пов'язані з неправильним використанням даних.

У загальному, C# є мовою програмування з великою кількістю можливостей та переваг. Це дозволяє програмістам розробляти програми для різних платформ та завдань, забезпечуючи високу продуктивність та безпеку.

2.2.2. **IDE Visual Studio 2022**

Visual Studio 2022 - це комплексне програмне середовище, розроблене компанією Microsoft, яке дозволяє програмістам ефективно розробляти програми для платформи Windows. Це високоефективний інструмент, який надає широкі можливості для створення високоякісного програмного забезпечення.

Visual Studio 2022 має декілька важливих функцій та особливостей, які дозволяють зручно та швидко розробляти програмне забезпечення. Деякі з них:

– Керування проектами: Visual Studio 2022 надає можливість керувати проектами та їх залежностями. Це дозволяє зручно керувати структурою проекту та його компонентами.

– Розширені можливості відлагодження: Visual Studio 2022 має розширені можливості відлагодження, що дозволяють програмістам зручно відлагоджувати код та знаходити помилки.

– Вбудована документація: Visual Studio 2022 має вбудовану документацію, що дозволяє програмістам швидко знайти необхідну інформацію про функції, класи та методи.

– Розширені можливості роботи з Git: Visual Studio 2022 має розширені можливості роботи з Git, що дозволяють зручно керувати версіями коду та співпрацювати з іншими членами команди.

– Підтримка віддаленої розробки: Visual Studio 2022 має підтримку віддаленої розробки, що дозволяє програмістам розробляти програмне забезпечення на віддалених серверах та віртуальних машинах.

– Розширені можливості тестування: Visual Studio 2022 має розширені можливості тестування, що дозволяють програмістам зручно тестувати свій код та виявляти помилки.

– Вбудований редактор графічного інтерфейсу: Visual Studio 2022 має вбудований редактор графічного інтерфейсу, що дозволяє програмістам швидко та зручно створювати графічний інтерфейс для свого програмного забезпечення.

– Розширені можливості підключення додаткових інструментів: Visual Studio 2022 має розширені можливості підключення додаткових інструментів та плагінів, що дозволяє програмістам зручно розширювати можливості IDE та підлаштовувати його до своїх потреб.

2.2.3. **Платформа .NET Standard**

.NET Standard - це специфікація, що визначає набір API (Application Programming Interface) для розробки програмного забезпечення на платформі .NET. Ця специфікація встановлює мінімальний набір API, які повинні бути підтримані всіма імплементаціями .NET, що підписались на цю специфікацію.

Основною метою створення .NET Standard є забезпечення сумісності між різними імплементаціями .NET. Завдяки цьому, розробники можуть створювати бібліотеки класів, які можуть використовуватися на різних імплементаціях .NET без потреби їх переписування.

.NET Standard встановлює загальну базу класів, яка містить типи, що повинні бути підтримані всіма імплементаціями .NET. Ці типи включають базові класи та інтерфейси, такі як System.Object, System.String, System.Collections.Generic та інші. Крім того, .NET Standard визначає спеціальні набори API, які повинні бути підтримані для різних версій імплементацій .NET. Наприклад, .NET Standard 2.0 містить API, які підтримуються в .NET Framework 4.6.1, .NET Core 2.0, Xamarin.iOS 10.14, Xamarin.Android 7.5 та інших.

.NET Standard використовується для розробки бібліотек класів, які можуть використовуватися на різних платформах .NET, таких як .NET Framework, .NET Core та Xamarin. Розробники можуть створювати бібліотеки класів з використанням .NET Standard, які будуть працювати на всіх платформах, що підтримують цю специфікацію.

Однією з переваг використання .NET Standard є забезпечення сумісності між різними версіями імплементацій .NET. Розробники можуть створювати бібліотеки класів з використанням .NET Standard, які підтримують різні версії .NET Framework, .NET Core, Xamarin і т.д. Це дозволяє зменшити залежність від конкретної версії платформи .NET і забезпечує більш широку сумісність бібліотек.

Крім того, .NET Standard дозволяє розробникам створювати бібліотеки класів, які будуть працювати на різних платформах .NET, що дозволяє створювати більш уніфіковану кодову базу. Це дозволяє збільшити швидкість розробки і підтримки коду, що покращує якість розробки програмного забезпечення.

Також, .NET Standard дозволяє розробникам використовувати більш широкий набір функцій, що реалізовані на різних імплементаціях .NET, в тому числі .NET Framework, .NET Core, Xamarin і т.д. Це дозволяє розробникам використовувати бібліотеки класів з використанням більш широкого набору функцій та API, що забезпечує більш гнучкий підхід до розробки.

Більш того, .NET Standard підтримує автоматичне завантаження залежностей, що дозволяє забезпечити більш просту та зручну розробку програмного забезпечення. Крім того, ця специфікація підтримує пакування та розповсюдження бібліотек класів, що забезпечує більш простий та зручний процес розгортання.

2.2.4. **UI підсистема WPF**

Windows Presentation Foundation (WPF) є технологією розробки графічного інтерфейсу користувача (GUI) для десктопних додатків на платформі Windows. Вона дозволяє розробникам створювати багатофункціональні інтерактивні додатки з використанням різноманітних елементів управління, таких як кнопки, поля введення, вікна і т.д.

Одним з найпоширеніших підходів до розробки додатків на WPF є патерн MVVM (Model-View-ViewModel). MVVM дозволяє розділити логіку додатку на три компоненти: Model, View та ViewModel.

Model - це складова, яка містить логіку додатку і зберігає дані. Вона не має відношення до GUI і може використовуватися незалежно від інтерфейсу користувача.

View - це компонента, яка відображає інтерфейс користувача. Вона має взаємодіяти з ViewModel, щоб відображати дані та обробляти події.

ViewModel - це компонента, яка дозволяє зв'язувати Model і View. Вона відповідає за зберігання та обробку даних, які повинні відображатися в View, а також за обробку подій, які спричинені взаємодією користувача з інтерфейсом.

MVVM дозволяє зробити код більш організованим та ефективним, зменшуючи залежність між компонентами і забезпечуючи взаємодію між ними через інтерфейс. Це дозволяє розробникам швидко змінювати функціональність додатку, збільшувати масштаб проектів та забезпечувати більш ефективну підтримку і розширення.

Крім того, WPF і MVVM підтримують зв'язування даних, яке дозволяє автоматично оновлювати дані, які відображаютьться в інтерфейсі користувача при їх зміні. Зв'язування даних також забезпечує більш простий і зрозумілий код, оскільки розробникам не потрібно вручну оновлювати дані в інтерфейсі користувача.

Також, WPF підтримує використання стилів та шаблонів, які дозволяють створювати багатофункціональні елементи управління та забезпечують єдиний стиль для всієї програми. Це дозволяє розробникам створювати власні шаблони для елементів управління, щоб вони відповідали специфічним потребам їх додатку.

Крім того, WPF дозволяє розробникам створювати додатки з використанням векторної графіки, що дозволяє забезпечити кращу якість відображення, зменшити розмір програми та забезпечити її більш гнучке управління.

Узагалі, WPF та MVVM - це потужні інструменти для розробки додатків з високоякісним інтерфейсом користувача. Вони дозволяють розробникам створювати багатофункціональні додатки, які легко розширюються і підтримуються, а також забезпечують відмінну якість відображення інтерфейсу користувача.

2.2.5. **ORM EntityFramework**

ORM (Object-Relational Mapping) - це технологія, яка дозволяє програмістам працювати з базами даних, як з об'єктами в коді, не звертаючись безпосередньо до SQL запитів. Один з найбільш популярних ORM-ів для роботи з базами даних в .NET - це Entity Framework.

Entity Framework (EF) - це ORM, який дозволяє програмістам взаємодіяти з базами даних з використанням об'єктів в .NET. Entity Framework підтримує різноманітні провайдери баз даних, такі як MS SQL Server, MySQL, Oracle і т.д.

Основна перевага використання Entity Framework полягає в тому, що програмісти можуть працювати з базами даних в термінах об'єктів, що спрощує розробку, управління та підтримку додатків, що використовують бази даних.

Замість написання складних SQL запитів, програмісти можуть використовувати LINQ (Language-Integrated Query) - мову запитів, яка дозволяє програмістам здійснювати запити до баз даних з використанням різноманітних методів та операцій LINQ.

Крім того, Entity Framework дозволяє легко реалізувати взаємозв'язки між таблицями в базі даних з використанням концепції Code First, коли програміст може створювати сутності (об'єкти), які будуть відображені на таблиці в базі даних.

Entity Framework підтримує транзакції, що дозволяє програмістам виконувати групу операцій в одній транзакції, що забезпечує цілісність бази даних і захист від помилок, які можуть виникнути при паралельному виконанні запитів.

Entity Framework також забезпечує механізми кешування даних, що дозволяє збільшити швидкодію додатків, що взаємодіють з базами даних. Кешув ання даних в Entity Framework реалізовано за допомогою механізму відстеження змін (Change Tracking). Кешування даних в EF забезпечує можливість повторного використання вже завантажених даних, що дозволяє запитувати базу даних менше разів і зменшити навантаження на сервер.

Entity Framework підтримує технологію Migrations, яка дозволяє програмістам керувати структурою бази даних з використанням коду. Міграції в Entity Framework дозволяють створювати, змінювати та видаляти таблиці в базі даних з використанням коду.

Одна з головних переваг Entity Framework - це висока абстракція бази даних. При роботі з EF програмісти не повинні вручну писати складні SQL запити та використовувати складні алгоритми. EF надає API для взаємодії з базою даних, що дозволяє легко розширювати функціональність бази даних та спрощує розробку додатків.

Завдяки Entity Framework програмісти можуть ефективно використовувати концепцію Model-View-ViewModel (MVVM), що дозволяє відокремлювати логіку додатку від його інтерфейсу та даних. MVVM забезпечує поділ додатку на три складові: модель (Model), представлення (View) та модель представлення (ViewModel).

3 МОДЕЛЮВАННЯ ТА ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ ДЛЯ СТРУКТУРУВАННЯ ДОКУМЕНТІВ

3.1. **Модель предметної галузі**

Діаграма предметної галузі, також відома як діаграма сутностей-зв'язків, є інструментом моделювання, що використовується для візуалізації та опису структури і зв'язків в предметній галузі. Вона допомагає розуміти потреби та вимоги до системи або проекту.

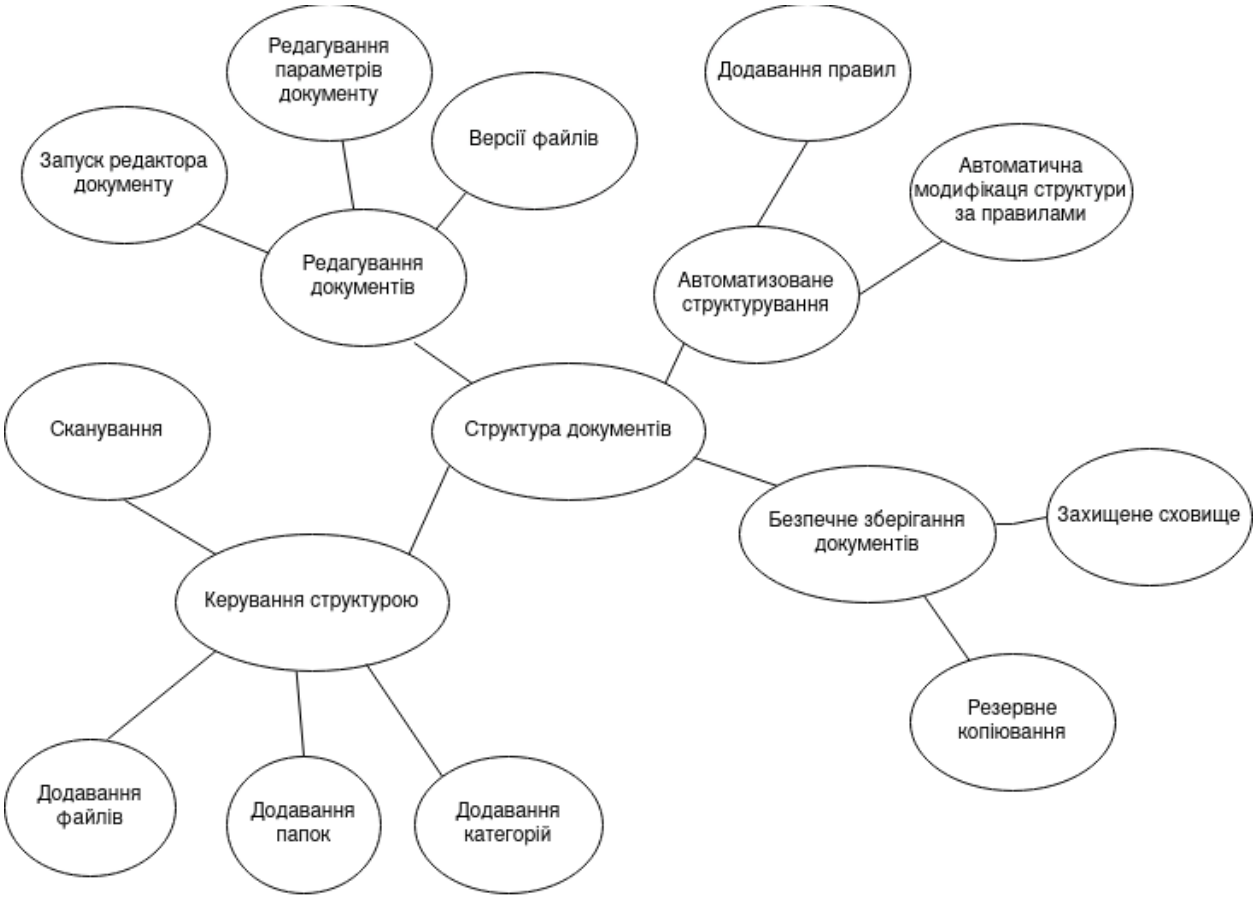


Рис. 3.1. – Діаграма предметної галузі

За даною діаграмою, можна помітити що функція «Структура документів» є головною, оскільки на ній базується вся робота з документами. Вона має похідні функції, які забезпечують продуктивну роботу з структурою документів.

«Керування структурою» є одною з найважливіших функцій оскільки відповідає за коригування структури що напряму впливає на ефективність роботи з документами. Містить наступний спектр підфункцій:

– Додавання файлів;
– Додавання категорій;
– Додавання папок;
– Сканування комп’ютеру;

«Безпечне зберігання документів» визначає спектр підфункцій, які відповідають за безпечне зберігання документів. Містить наступні пункти:

– Резервне копіювання;
– Захищене сховище;

Функція «Редагування документів» також є одною з визначних, оскільки дозволяє реалізувати весь потенціал структури документів за рахунок швидкого доступу до редактору документів. Таким чином визначений мінімальний набір підфункцій:

– Запуск редактора документу;
– Редагування параметрів документу;
– Версії документу;

«Автоматизоване структурування» є функцією котра забезпечує автоматизацію організації великої кількості документів, містить наступні підфункції:

– Додавання правил;
– Автоматична модифікація структури за правилами;

3.2. **Модель прецедентів**

UML-діаграма прецедентів, також відома як Use-Case Diagram, моделює поведінку системи. Ця діаграма відображає взаємодії між системою та її

учасниками, які називаються акторами. На діаграмах варіантів використання та дійових осіб описується, що система робить і як учасники її використовують, але не відображається внутрішня робота системи.

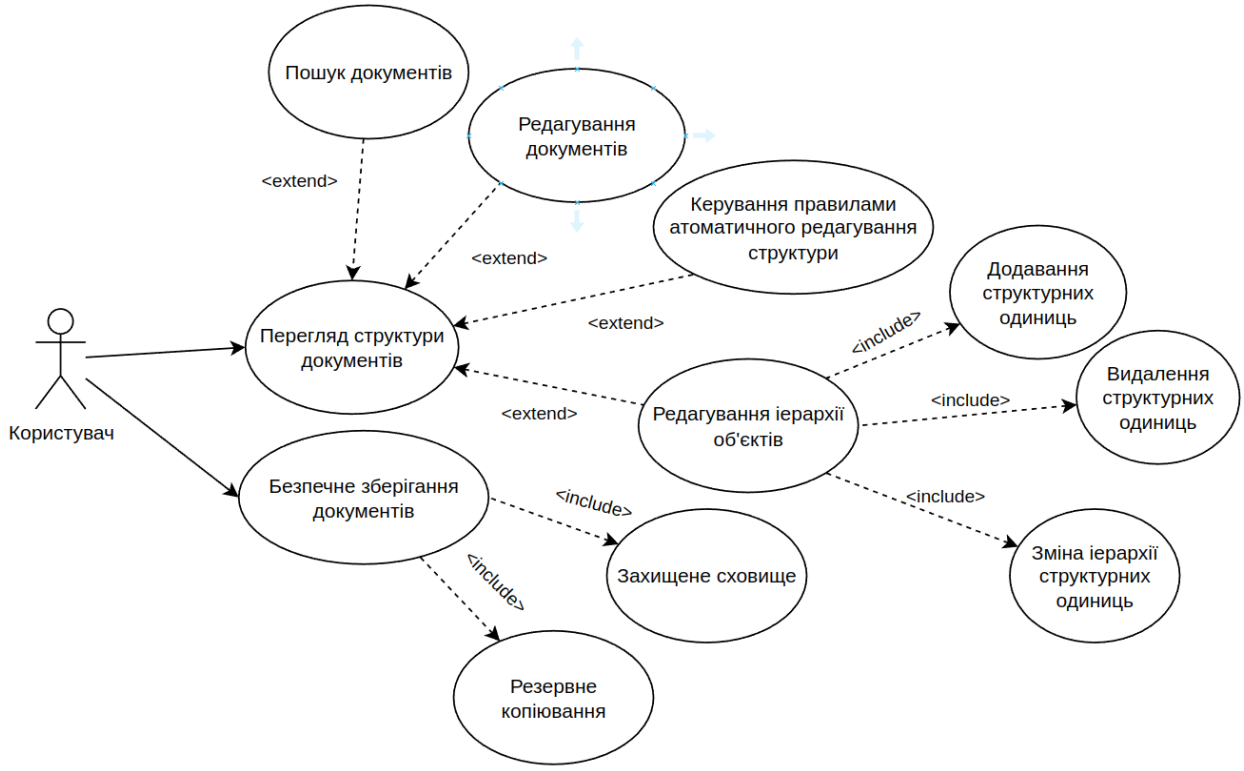


Рис. 3.2. – Діаграма прецедентів

За даною діаграмою можна побачити, що для сутності користувача основні сценарії використання це «Безпечне зберігання документів» та «Перегляд структури документів».

Сценарій «Безпечне зберігання документів» котрий включає в собі такі можливості як «Резервне копіювання» та «Захищене сховище».

Головним сценарієм є «Перегляд структури документів», він містить в собі можливість «Редагування структури документів» яка має сценарії з модифікації «Додавання структурних одиниць», «Видалення структурних одиниць», «Додавання структурних одиниць» та можливість «Керування правилами автоматичного редагування» що містить в собі звичайні сценарії модифікації.

Також головний сценарій містить в собі допоміжні можливості «Пошуку документів» та «Редагування документів»

3.3. **Модель діяльності**

На наступних діаграмах відображається процес роботи з застосунком

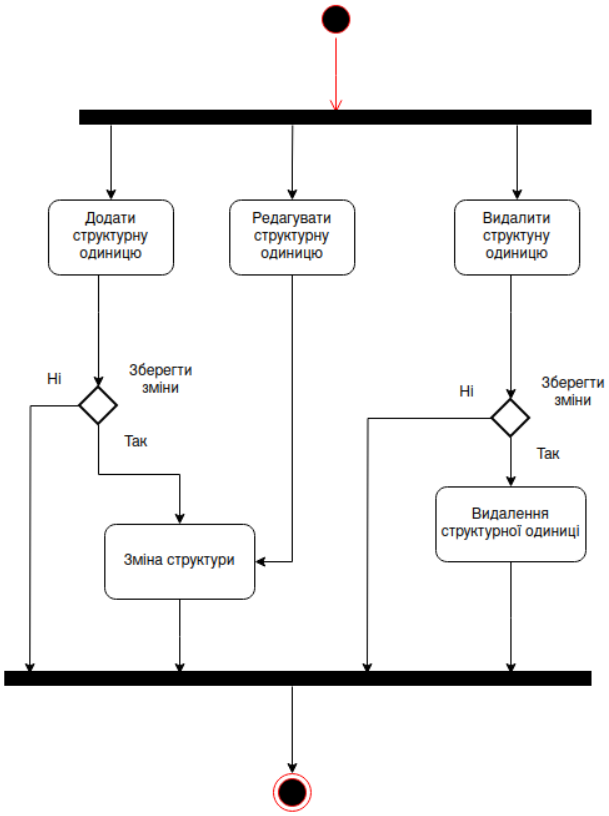


Рис. 3.3. – Діаграма діяльності, коригування структури документів

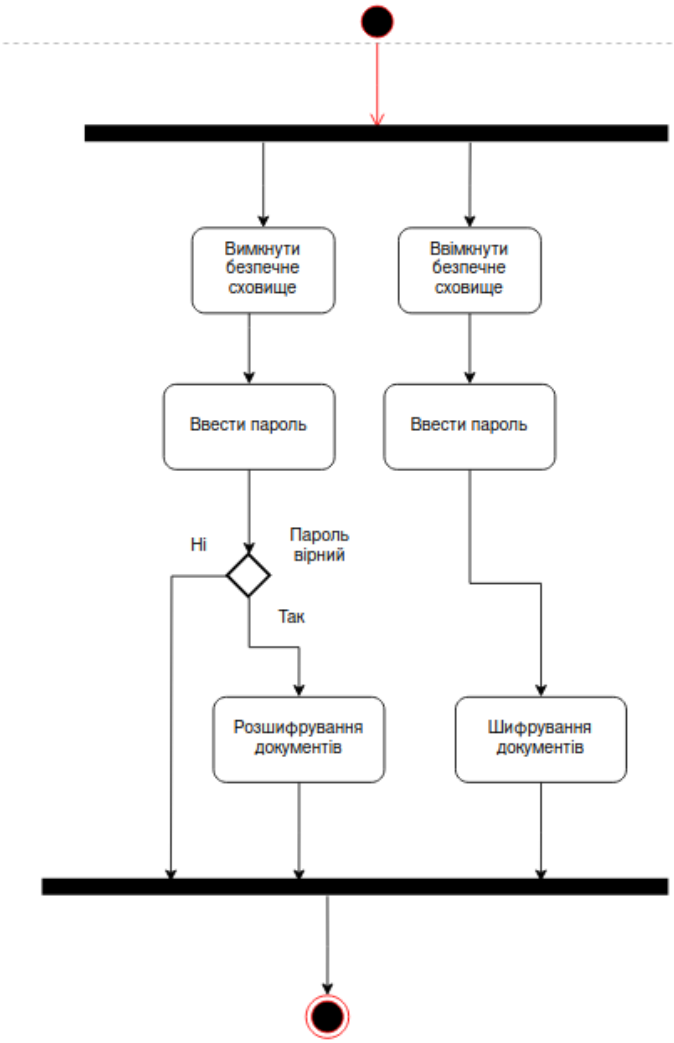


Рис. 3.4. – Діаграма діяльності, зашифроване зберігання документів

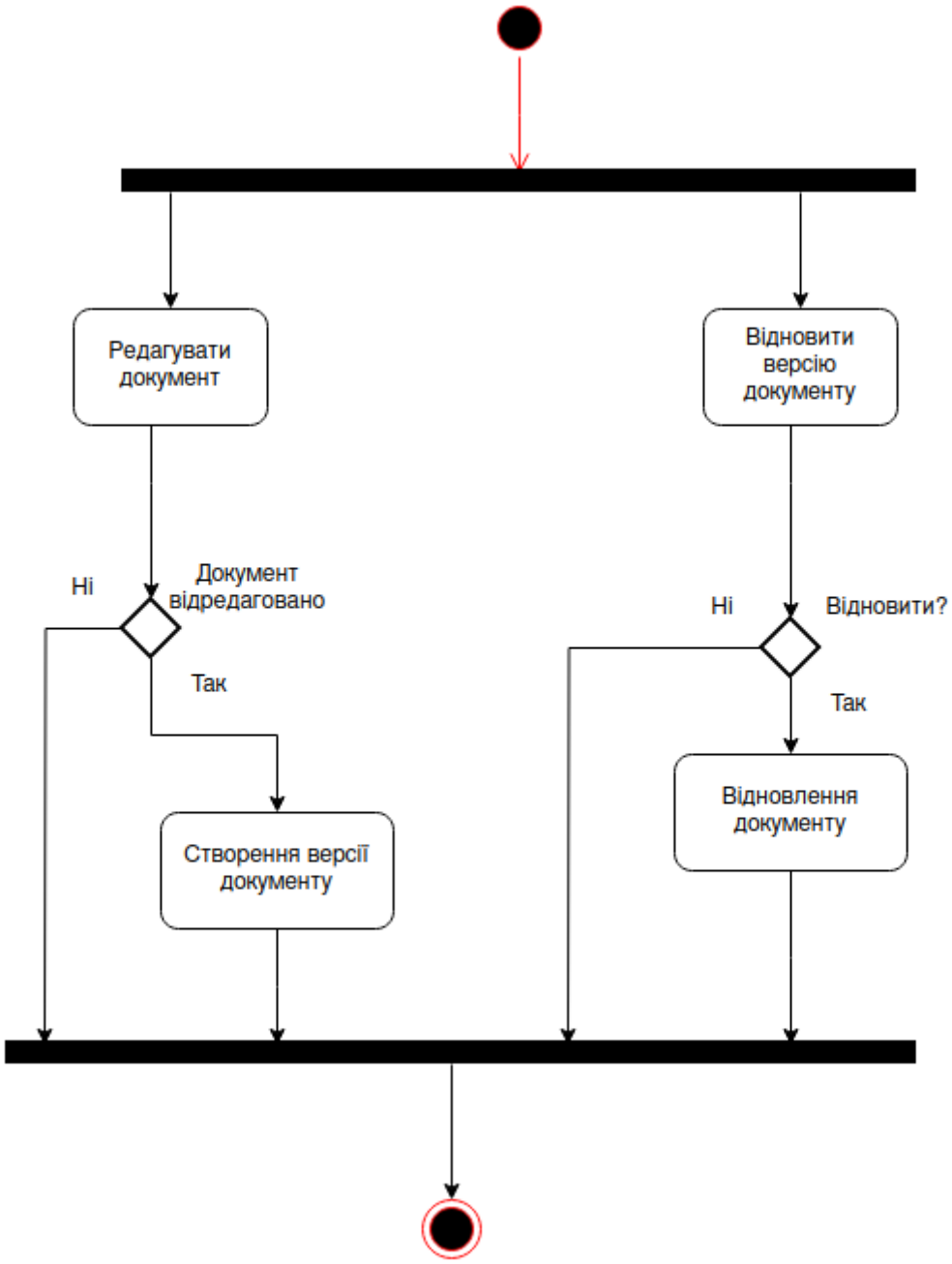


Рис. 3.5. – Діаграма діяльності редагування документу

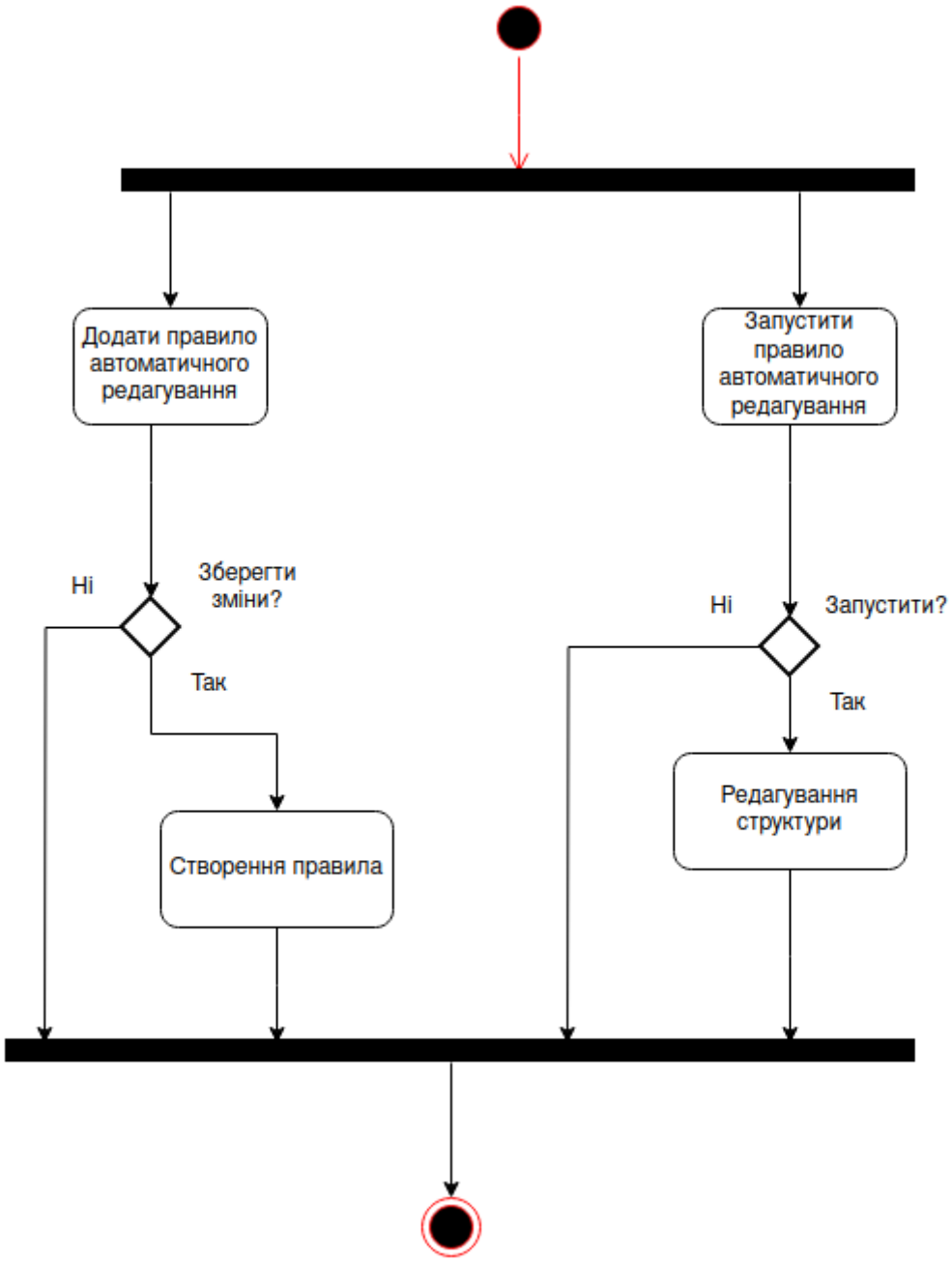


Рис. 3.6. – Діаграма діяльності автоматичного редагування структури

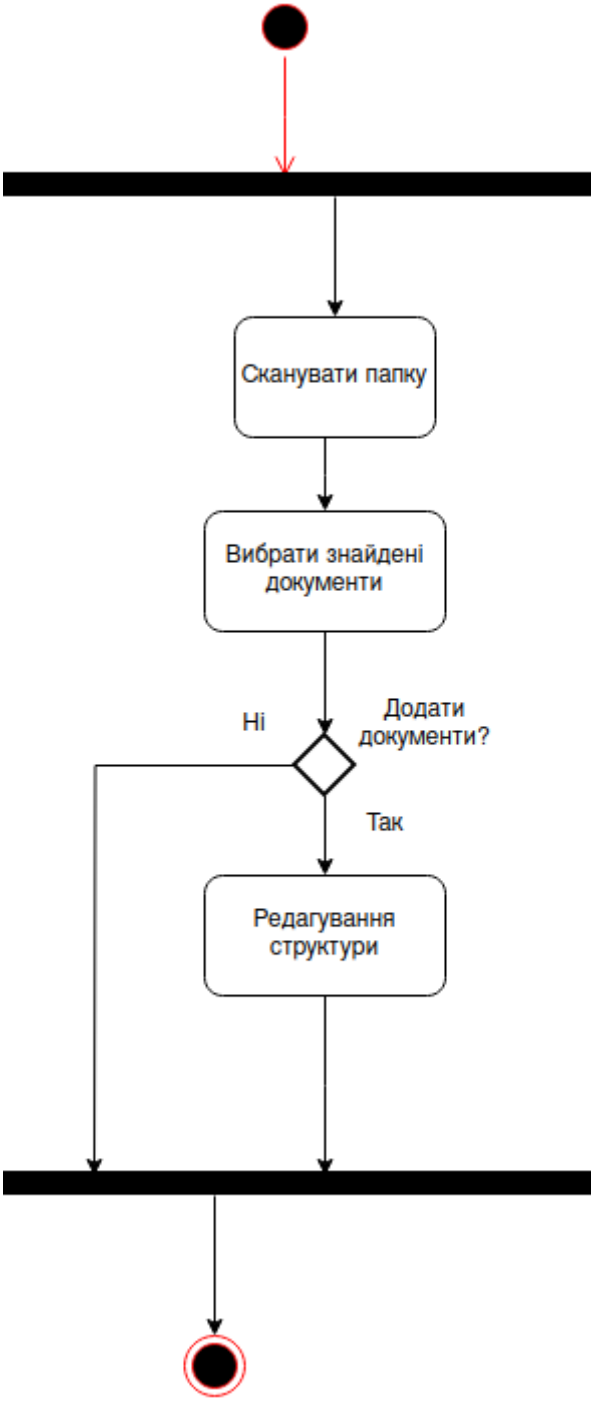


Рис. 3.7. – Діаграма діяльності сканування папки

3.4. **Нефункціональні вимоги**

ПЗ повинно задовольняти наступні нефункціональні вимоги: зручність використання, безпека, продуктивність, надійність, можливість модифікації, системні вимоги.

Зручність використання. Інтерфейс застосунку повинен відповідати таким вимогам:

– Використання нейтральних кольорів
– Ергономічне розташування елементів управління
– Розмір тексту, що зручно читається

Безпека. Застосунок повинен мати можливість безпечного зберігання документів, на випадок несанкціонованого доступу до комп’ютеру

Надійність. Застосунок повинен відповідати таким вимогам надійності:

– Застосунок повинен продовжувати працювати при виникненні неопрацьованих помилок
– Застосунок повинен мати можливість відновлюватися з останньої резервної копії

Продуктивність. Застосунок повинен швидко відповідати на запити користувача

Можливість модифікації. Програмні модулі системи повинні використовувати шаблони проектування та слідувати принципам SOLID, KISS, DRY. Повторюванні елементи системі повинні бути винесені в окремі методи, класи або бібліотеки.

Системні вимоги. Для коректної роботи застосунку, система на котрій запускається ПЗ повинна мати: операційну систему Windows 10 або вище, набір бібліотек .NET Standard Runtime 6.8 або вище, редактори для таких типів файлів як docx, pdf, xlsx.

3.5. **Архітектура програмного забезпечення**

У цій інформаційній системі відповідно до задач використовується монолітна архітектура.

Монолітна архітектура є однією з найпоширеніших архітектур програмного забезпечення, яка використовується для створення desktop-програм. У цій

архітектурі весь код програми знаходиться в одному файлі або наборі файлів, що взаємодіють між собою, тобто весь програмний код знаходиться в моноліті, тому і отримала таку назву. Монолітна архітектура є простою та прямолінійною, оскільки зазвичай працює на пряму з апаратними ресурсами комп'ютера.

Основна перевага монолітної архітектури полягає в тому, що вона забезпечує простоту розробки програм та зменшує час, необхідний для створення та розгортання програми. Всі компоненти програми розташовані в одному місці, тому їх легко знайти та відлагодити. Зазвичай, монолітна архітектура є найкращим варіантом для створення малих та середніх програм, таких як текстові редактори, ігри та додатки для роботи з базами даних.

Окрім цього, монолітна архітектура забезпечує низький рівень складності. Підтримка монолітних програм є простою, оскільки все знаходиться в одному місці. Це дозволяє ефективно виявляти та виправляти помилки, що може знизити час зупинки програми та забезпечити користувачам більш якісне досвід користування.

Монолітна архітектура також забезпечує більшу продуктивність в порівнянні з децентралізованою архітектурою, оскільки немає додаткових витрат на обмін повідомленнями між компонентами програми, як у більш складних архітектурах.

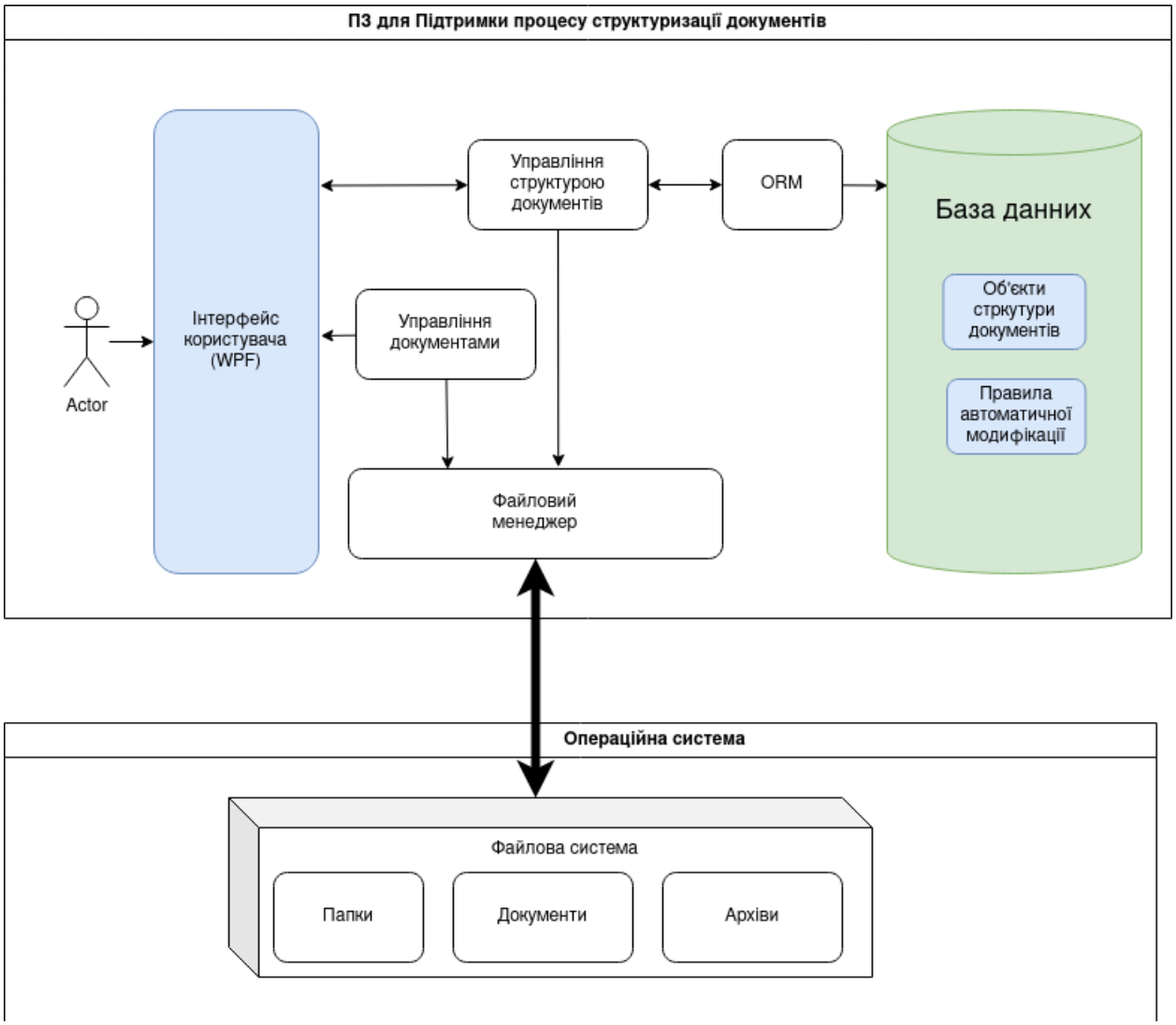


Рис. 3.8. – Архітектура застосунку

3.6. **Модель бази даних**

Схему бази даних представлено через ERD (Entity-Relationship diagram), яка відображає сутності та зв’язки між ними

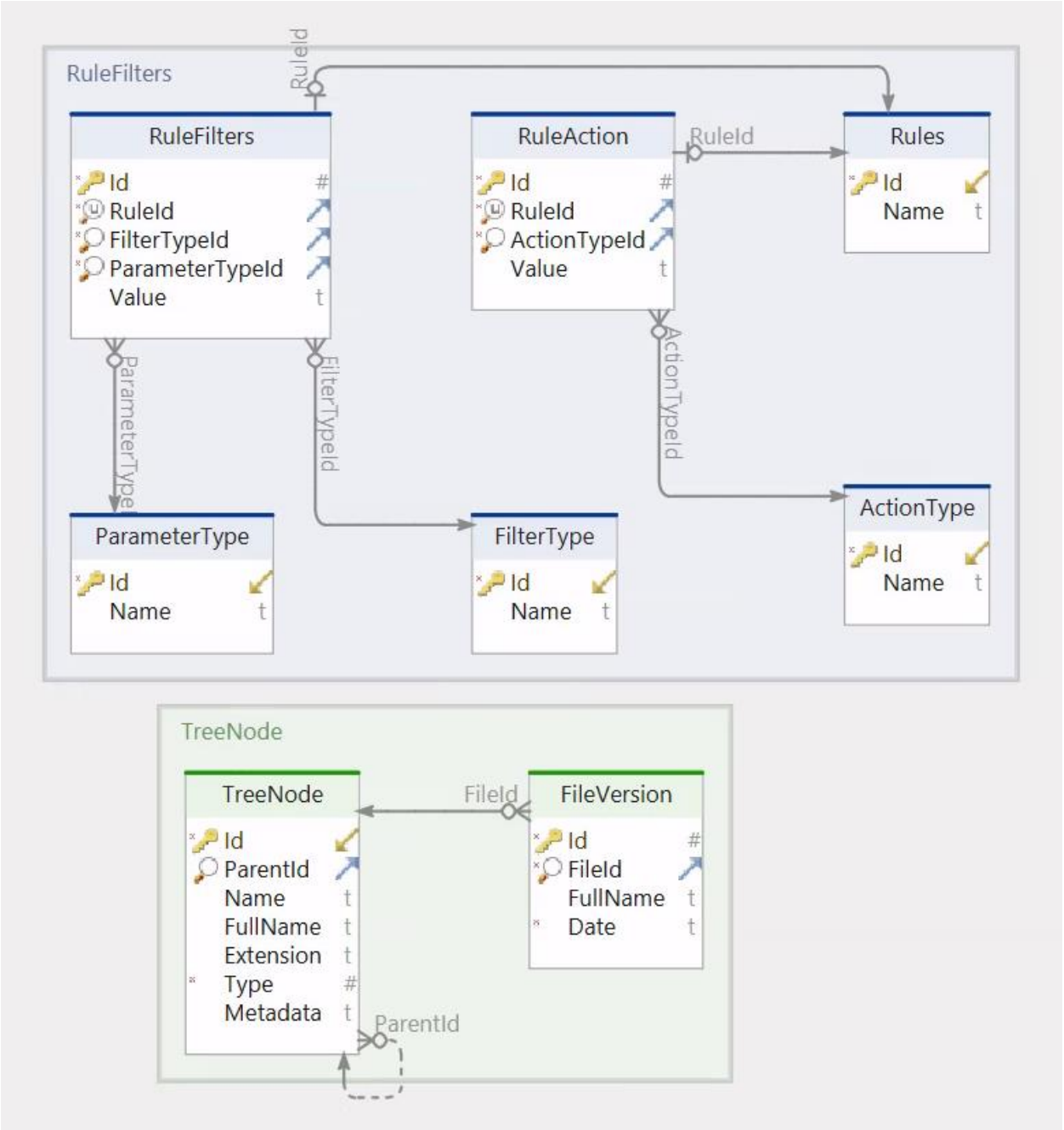


Рис. 3.9 – Модель бази даних

3.7. **Діаграма класів**

Діаграма класів - це структурна діаграма, яка відображає класи програмної системи, їх взаємозв'язки та основні атрибути та методи класу. Вона дозволяє моделювати структуру системи та розуміти взаємозв'язки між класами, допомагаючи у процесі аналізу та проектування програмного забезпечення.

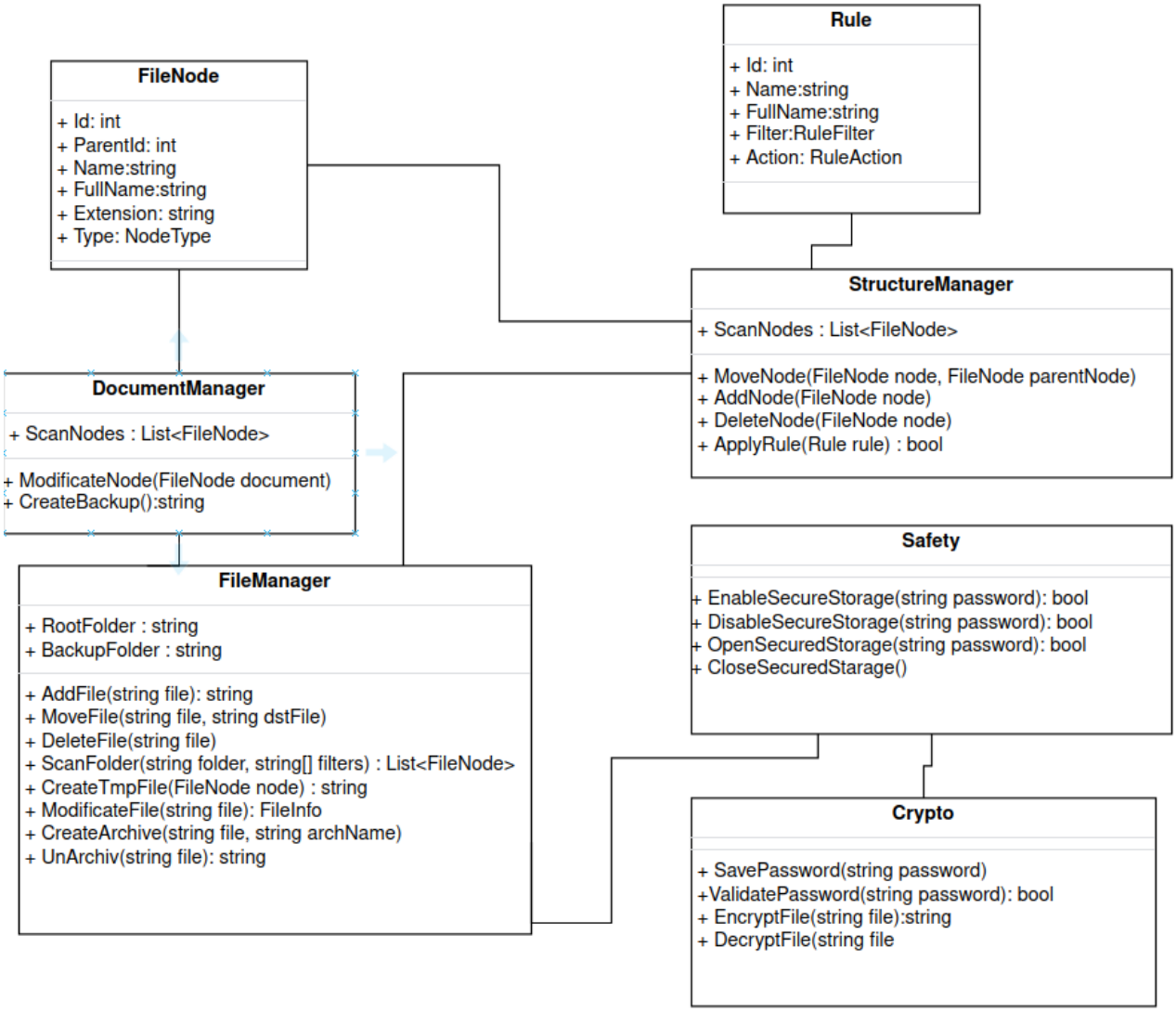


Рис. 3.10 – Діаграма класів

4 РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ ДЛЯ АВТОМАТИЗАЦІЇ СКЛАДСЬКОГО ОБЛІКУ

4.1. **Реалізація сутностей EF Core**

Сутності (Entities) в Entity Framework Core (EF Core) - це класи, які відображають таблиці бази даних та зв'язки між ними в об'єктно-орієнтованому програмуванні. Сутності керуються EF Core і можуть виконувати CRUD-операції (створення, читання, оновлення та видалення) даних у базі даних.

EF Core дозволяє визначити сутності за допомогою анотацій або відповідної конфігурації у класі контексту бази даних. Кожна сутність повинна мати принаймні одне властивість, яка буде відображатися на первинний ключ таблиці бази даних.

EF Core підтримує різноманітні типи відносин між сутностями, такі як один до одного, один до багатьох та багато до багатьох. За допомогою відносин EF Core автоматично створює зв'язки між таблицями у базі даних та надає зручний API для роботи з даними.

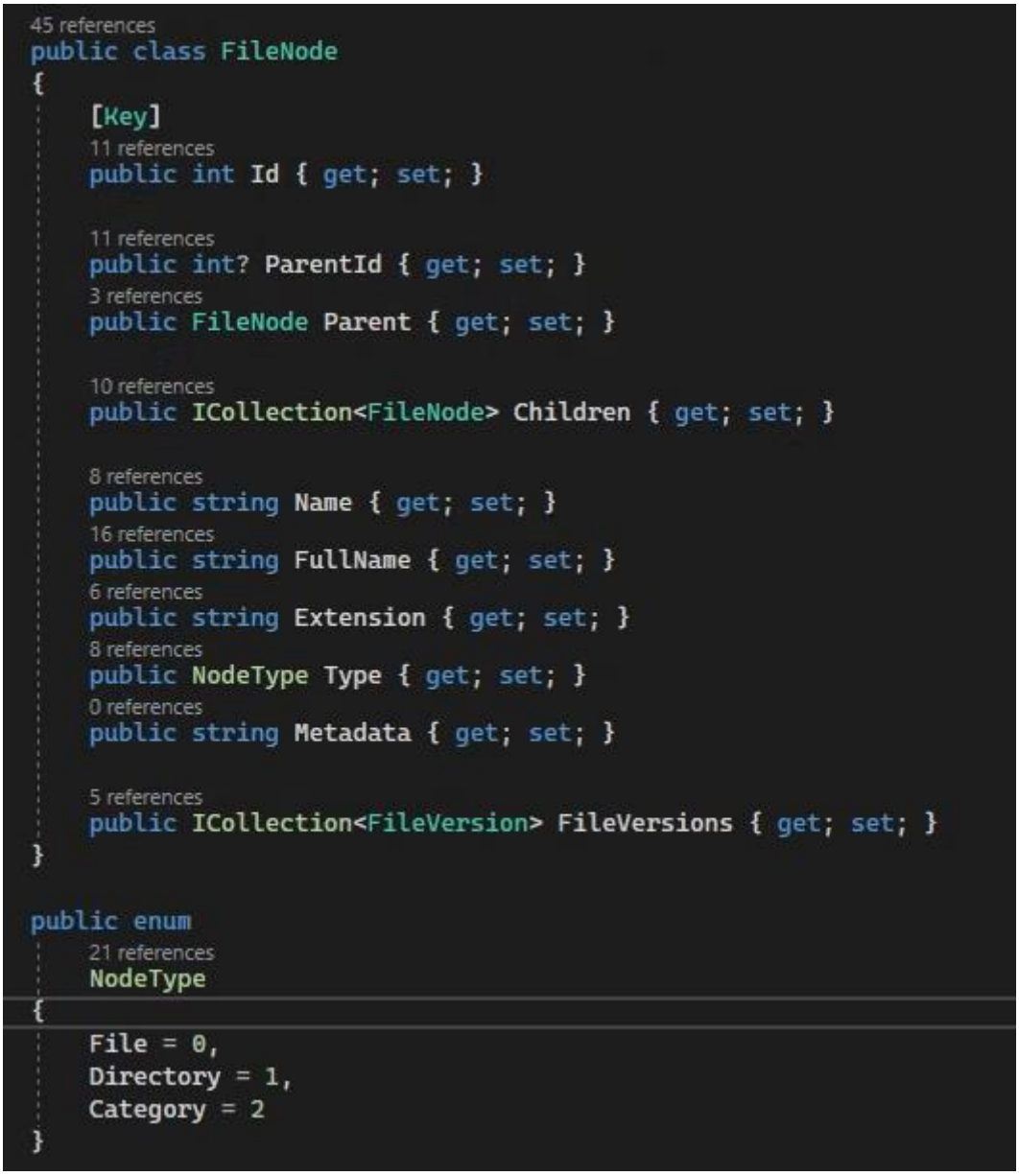


Рис. 4.1 – Сутність об’єкту структурної одиниці у EF Core

Для реалізації складних зв’язків між сутностями у EF Core використовується, який дозволяє налаштувати поведінку фреймворку за допомогою ланцюжків методів.

Fluent API дозволяє виконувати більш точну настройку конфігурації, ніж можливо за допомогою атрибутів. Наприклад, Fluent API дозволяє визначати зв'язки між таблицями бази даних за допомогою методів типу "HasOne", "WithMany", "HasForeignKey" тощо. Також можна виконувати налаштуваня

поведінки каскадного видалення, включаючи поведінку видалення дочірніх записів при видаленні батьківського запису.

Fluent API дозволяє виконувати конфігурацію в окремому класі, який відповідає за контекст бази даних, що дозволяє додатково забезпечувати розділення поведінки між класами та модулями програми.

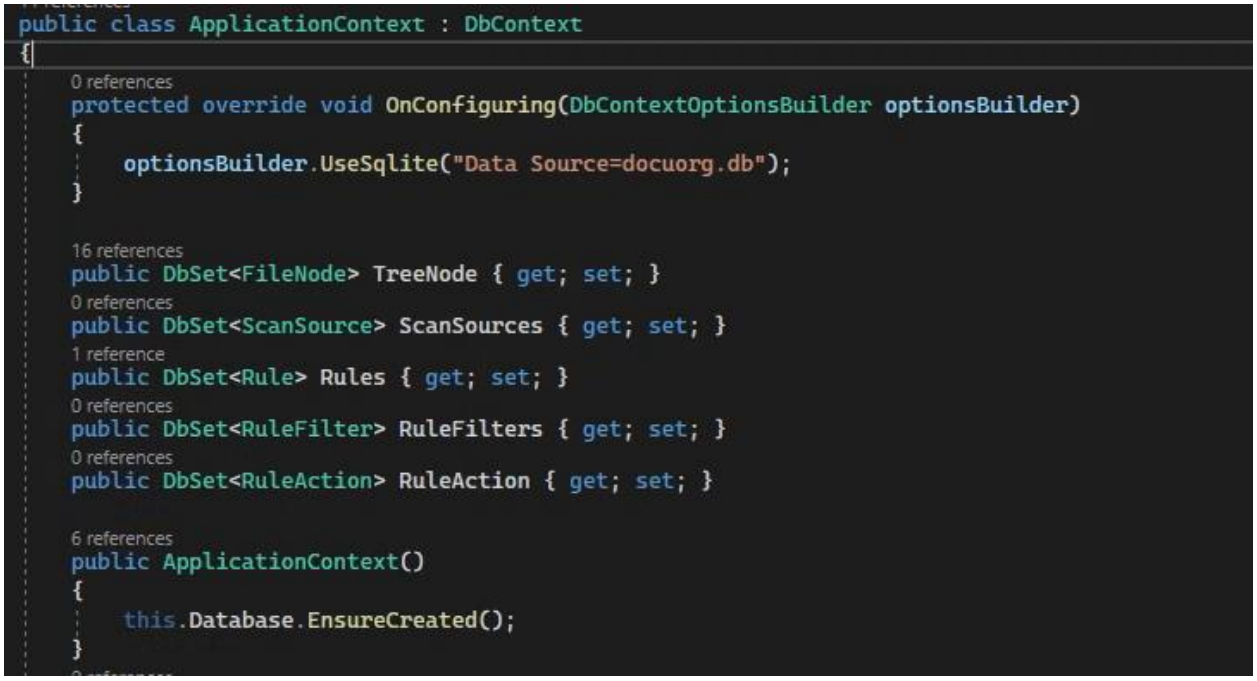
Крім того, Fluent API дозволяє виконувати настройку багатьох інших аспектів EF Core, таких як використання рядків SQL, настройка звязування типів даних та використання сховища даних.

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<FileNode>()

        .HasOne(c => c.Parent)
        .WithMany(p => p.Children)
        .HasForeignKey(a => a.ParentId)
        .OnDelete(DeleteBehavior.Cascade);

    modelBuilder.Entity<FileNode>()
        .HasMany(p => p.FileVersions)
        .WithOne(c => c.File)
        .HasForeignKey(a => a.FileId)
        .OnDelete(DeleteBehavior.Cascade);

    modelBuilder.Entity<FilterType>()
        .Property(e => e.Id)
        .HasConversion<int>();

    modelBuilder.Entity<FilterType>()
        .HasData(Enum.GetValues(typeof(FilterTypeId))
        .Cast<FilterTypeId>()
        .Select(e => new FilterType()
        {
            Id = e,
            Name = e.ToString()
        }));

    modelBuilder.Entity<ParameterType>()
        .HasData(Enum.GetValues(typeof(ParameterTypeId))
        .Cast<ParameterTypeId>()
        .Select(e => new ParameterType()
        {
            Id = e,
            Name = e.ToString()
        }));

    modelBuilder.Entity<ActionType>()
        .Property(e => e.Id)
        .HasConversion<int>();

    modelBuilder.Entity<ActionType>()
        .HasData(Enum.GetValues(typeof(ParameterTypeId))
        .Cast<ActionTypeId>()
        .Select(e => new ActionType()
        {
            Id = e,
            Name = e.ToString()
        }));

    modelBuilder.Entity<FilterType>()
        .Property(e => e.Id)
        .HasConversion<int>();

    modelBuilder.Entity<RuleFilter>()
        .HasOne(f => f.FilterType)
        .WithMany(t => t.Filters)
        .HasForeignKey(x => x.ParameterTypeId);

    modelBuilder.Entity<RuleAction>()
        .HasOne(f => f.ActionType)
        .WithMany(t => t.Actions)
        .HasForeignKey(x => x.ActionTypeId);

    modelBuilder.Entity<Rule>()
        .HasOne(c => c.Action)
        .WithOne(x => x.Rule)
        .HasForeignKey<RuleAction>(a => a.RuleId)
        .IsRequired(true);
```

Рис. 4.2 – Використання FluentAPI

4.2. **Підключення до бази даних**

Для підключення до бази даних використовується провайдер SqliteProvider, який дозволяє легко створювати бази даних у вигляді файлу, що суттєво спрощує управління базою даних.



```
public class ApplicationContext : DbContext
{
    0 references
    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        optionsBuilder.UseSqlite("Data Source=docuorg.db");
    }

    16 references
    public DbSet<FileNode> TreeNode { get; set; }
    0 references
    public DbSet<ScanSource> ScanSources { get; set; }
    1 reference
    public DbSet<Rule> Rules { get; set; }
    0 references
    public DbSet<RuleFilter> RuleFilters { get; set; }
    0 references
    public DbSet<RuleAction> RuleAction { get; set; }

    6 references
    public ApplicationContext()
    {
        this.Database.EnsureCreated();
    }
```

Рис. 4.3 – підключення до бази даних

4.3. **Користувацький інтерфейс**

Для реалізації візуальної частини застосунок використовує комбінацію MVVM, WPF та EF Core, що дозволяє створювати додатки з відокремленим представленням даних, що полегшує їхню розробку та тестування.

За допомогою MVVM можна легко відокремити логіку додатку від його представлення та моделей даних, що дозволяє забезпечити більше можливостей для повторного використання коду. WPF надає зручний інструментарій для розробки графічних інтерфейсів, який підтримує шаблон MVVM, тим самим дозволяючи забезпечити розділення між логікою додатку та його представленням. EF Core дозволяє створювати сутності які можна прив’язати до представлень що суттєво зменшує кількість коду, для візуальної реалізації таблиць, дерев та списків.

```
<TreeView Margin="0,31,699,5" RenderTransformOrigin="0.5,0.5" Name="FileTree">
    <TreeView.Resources>
        <Converters:NodeTypeConverter x:Key="NodeTypeConverter"/>
    </TreeView.Resources>
    <TreeView.ItemTemplate>
        <HierarchicalDataTemplate ItemsSource="{Binding Path=Children}">
            <StackPanel Orientation="Horizontal">
                <Image Source="{Binding Type,Converter={StaticResource NodeTypeConverter}}" Width="20" Height="20"/>
                <TextBlock Text="{Binding Path=Name}"/>
            </StackPanel>
        </HierarchicalDataTemplate>
    </TreeView.ItemTemplate>
</TreeView>
```

Рис. 4.4 – Дерево структурних елементів з прив’язкою до сутності EF Core

```
using (var context = new ApplicationContext())
{
    var nodes = context.TreeNode.ToList();

    Nodes = context.TreeNode.Where(t => t.ParentId == null).Include(a => a.Children).ToList();
    treeView = new CollectionView(Nodes);

    FileTree.ItemsSource = treeView;

}
```

Рис. 4.5 – Прив’язка сутності БД до View

4.4. **Редагування документів**

Для ефективної роботи з документами користувачу важливо мати можливість легко змінювати документи, тому для реалізації такої можливості використаний самий простий але в той же час най зручніший та най функціональніший метод це використання вбудованих можливостей ОС Windows.

Застосунок створює новий процес, у ньому виконується системна консольна команда яка запускає відкриття файлу, це дозволяє користувачу вибирати якою програмою він хоче відкрити той чи інший тип файлу. Також даний спосіб не потребує встановлення додаткових програм та сумісний з будь-якими редакторами що встановлені на комп’ютері.

Для збереження версій файлів застосунок створює тимчасовий файл, який в свою чергу редагується користувачем, після закінчення операції якщо файл було змінено змінюється і час модифікацію файлу. Застосунок порівнює дату до

модифікації і після, якщо вони відрізняються на тимчасовий файл стає основним, а основний одною з версій файлу.

```
1 reference
public static void EditFile(FileNode file)
{
    try
    {
        var info = new FileInfo(file.FullName);
        var mDate = info.LastWriteTime;
        var tmpFile = FileManager.AddTempFile(file.FullName);

        var proc = new Process();
        proc.StartInfo = new ProcessStartInfo(file.FullName)
        {
            UseShellExecute = true
        };
        proc.Start();
        proc.WaitForExit();

        info.Refresh();
        var newDate = info.LastWriteTime;
        if (mDate != newDate)
        {
            var fileVersion = FileManager.TempToNewVersionFile(file, tmpFile);
            using (var context = new ApplicationContext())
            {
                context.TreeNode.FirstOrDefault(x => x.Id == file.Id)?.FileVersions?.Add(fileVersion);
                context.SaveChanges();
            }
        }
        else
        {
            File.Delete(tmpFile);
        }
    }
    catch
    {
    }
}
```

Рис. 4.6 – Редагування документів

4.5. **Захищене сховище**

Для безпечного зберігання документів застосунок використовує хеш-функцію PKDBF2 у комбінації зі стандартом AES.

PKDBF2 (Password Key Derivation Function 2) - це криптографічна хеш-функція, яка використовується для посилення паролів перед збереженням їх у базі даних.

Хеш-функція PKDBF2 використовує ітеративний процес для створення хеш-значення з вхідного пароля та "солі" - випадкової додаткової інформації, що додається до пароля для підвищення безпеки.

Алгоритм PKDBF2 має параметри, які можна налаштувати, щоб зробити його більш безпечним та більш складним для атак.

Алгоритм PKDBF2 базується на HMAC (Hash-based Message Authentication Code), який поєднує в собі хеш-функцію та ключ для створення підпису повідомлення. Це допомагає забезпечити високу стійкість до атак на підміну.

AES - "Advanced Encryption Standard" – один з найбільш поширених криптографічних алгоритмів, що використовуються для шифрування даних.

Стандарт AES був створений заміною старого стандарту шифрування DES (Data Encryption Standard) в результаті того, що стандарт DES став занадто слабким для використання в сучасних системах.

AES працює за принципом заміни і перестановки (substitution-permutation network), де кожен блок шифрується шляхом заміни кожного байту на інший байт згідно з таблицею заміни, а потім переставляється згідно з фіксованим алгоритмом перестановки.

AES має декілька режимів роботи, таких як ECB (Electronic Codebook), CBC (Cipher Block Chaining), CFB (Cipher Feedback), OFB (Output Feedback) та інші, що дає можливість використовувати його в різних випадках.

Для шифрування використовується криптографічна система з відкритими та закритими ключами. У стандарті AES застосовується хешований ключ розміром 256 біт.

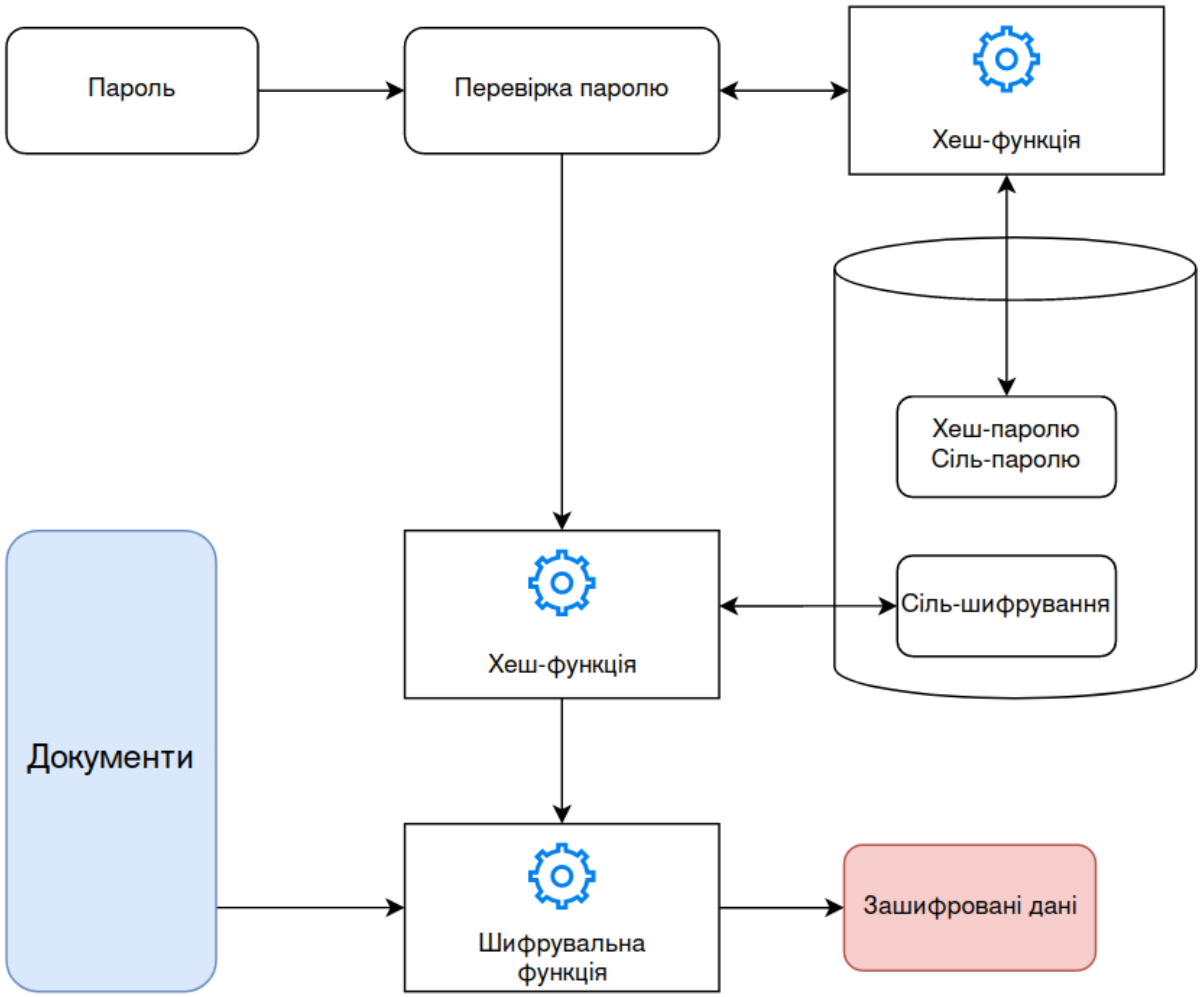


Рис. 4.7 – Схема криптографічної системи

```
1 reference
public static string FileEncrypt(string inputFile, string password)
{
    FileStream fsCrypt = new FileStream(inputFile + ".aes", FileMode.Create);

    byte[] passwordBytes = System.Text.Encoding.UTF8.GetBytes(password);
    byte[] salt = GetAESSalt();

    RijndaelManaged AES = new RijndaelManaged();
    AES.KeySize = 256;
    AES.BlockSize = 128;
    AES.Padding = PaddingMode.PKCS7;

    var key = new Rfc2898DeriveBytes(passwordBytes, salt, 50000);
    AES.Key = key.GetBytes(AES.KeySize / 8);
    AES.IV = key.GetBytes(AES.BlockSize / 8);

    AES.Mode = CipherMode.CFB;

    fsCrypt.Write(salt, 0, salt.Length);

    CryptoStream cs = new CryptoStream(fsCrypt, AES.CreateEncryptor(), CryptoStreamMode.Write);

    FileStream fsIn = new FileStream(inputFile, FileMode.Open);

    byte[] buffer = new byte[1048576];
    int read;

    try
    {
        while ((read = fsIn.Read(buffer, 0, buffer.Length)) > 0)
        {
            cs.Write(buffer, 0, read);
        }

        fsIn.Close();
    }
    catch (Exception ex)
    {
        Console.WriteLine("Error: " + ex.Message);
    }
    finally
    {
        cs.Close();
        fsCrypt.Close();
    }
    return fsCrypt.Name;
}
```

Рис. 4.7 – Реалізація шифрування файлу

4.6. **Резервне копіювання**

Для реалізації резервнисх копій документів використано спосіб архівування документів. Для цього використаний ZIP, формат архівування файлів, який дозволяє зменшити обсяг файлів і зберігати їх в одному файлі з розширенням .zip.

Формат ZIP дозволяє зберігати кілька файлів в одному архіві та стискувати їх, що дозволяє зменшити обсяг файлів та спростити їх передачу чи збереження.

Файли у форматі ZIP можуть бути розпаковані за допомогою різноманітного програмного забезпечення для роботи з архівами, таких як WinZip, 7-Zip, WinRAR та інші. При розпакуванні ZIP-архіву всі вміст архіву буде відновлено у вихідний вигляд.

Рис. 4.8. – Архівування документів

ВИСНОВКИ

В процесі виконання роботи було проведено аналіз процесу структурування документів, на його основі виявлено базові потреби користувача та створено модель предметної галузі.

Виявлені основні недоліки прямих і не прямих аналогів, такі як:

– Відсутність безпечного зберігання документів

– Резервне копіювання

– Значна вартість програмного забезпечення

– Комплексність функціоналу для ефективного структурування документів

На основі опрацьованих даних розроблено модель прецедентів, діаграми діяльності, нефункціональні вимоги.

Було спроектовано внутрішньої структуру програмного забезпечення зображену у вигляді архітектурної модель та модель бази даних що відображає сутності застосунку та зв’язки між ними.

Розроблено desktop застосунок під операційну систему Windows для структурування документів. В якості платформи для розробки використано фреймворк .NET у комбінації з мовою програмування C#. Для створення користувацького інтерфейсу застосована графічна підсистема WPF з шаблоном MVVM. Також для зберігання даних використаний ORM фреймворк Entity Framework Core з базою даних SQLite.

Для забезпечення без безпечного зберігання документів використано хеш-функцію PBKDF2 та алгоритм шифрування AES з розміром ключа 256 біт

Також в процесі роботи використано базові бібліотеки .NET для роботи з ОС, архівацію документів за допомогою формату ZIP.

ПЕРЕЛІК ПОСИЛАНЬ

1. Unified search for finding workplace content [Електронний ресурс]. – Режим доступу до ресурсу:

https://www.elastic.co/pdf/unified-search-for-finding-workplace-content

2. Digital Transformation Statistics and Trends [Електронний ресурс]. – Режим доступу до ресурсу: https://quixy.com/blog/top-digital-transformation-statistics-trends-forecasts/

3. Jordan S. Document Management System – A Way to Digital Transformation [Електронний ресурс] / Sandra Jordan, Simona Sternad Zabukovšek, Irena Šišovska Klančnik // Naše gospodarstvo/Our economy. – 2022. – Т. 68, № 2. – С. 43–54. – Режим доступу: https://doi.org/10.2478/ngoe-2022-0010

4. Warraich N. F. Keeping found things found [Електронний ресурс] / Nosheen Fatima Warraich, Irfan Ali, Shazia Yasmeen // Information and Learning Science. – 2018. – Т. 119, № 12. – С. 712–720. – Режим доступу: https://doi.org/10.1108/ils-07-2018-0064

5. Assimakopoulos N. A. Systems approach to document management [Електронний ресурс] / Nikitas A. Assimakopoulos, Alexandros Miaris, Elias Sakellaris // Acta Europeana Systemica. – 2020. – Т. 7. – С. 31–50. – Режим доступу: https://doi.org/10.14428/aes.v7i1.56623

6. DropIt: Personal Assistant to Automatically Manage Your Files [Електронний ресурс]. – Режим доступу: http://www.dropitproject.com/.

7. SharePoint [Електронний ресурс]. – Режим доступу: https://www.microsoft.com/en-us/microsoft-365/sharepoint/collaboration/.

8. File Juggler [Електронний ресурс]. – Режим доступу: https://www.filejuggler.com/

9. .NET documentation [Електронний ресурс] – Режим доступу: https://learn.microsoft.com/en-us/dotnet/

10. C# docs - get started, tutorials, reference. [Електронний ресурс] – Режим доступу: https://learn.microsoft.com/en-us/dotnet/csharp/.

11. Windows Presentation Foundation. [Електронний ресурс] – Режим доступу: https://learn.microsoft.com/en-us/dotnet/desktop/wpf/overview/?view=netdesktop-7.0

12. Microsoft Visual Studio 2022. [Електронний ресурс]– Режим доступу: https://visualstudio.microsoft.com/

13. Entity Framework Core. [Електронний ресурс]– Режим доступу: https://learn.microsoft.com/en-us/ef/core/

14. Overview of XAML. [Електронний ресурс]– Режим доступу: https://learn.microsoft.com/en-us/visualstudio/xaml-tools/xaml-overview?view=vs-2022

ДОДАТОК А

Демонстраційні матеріали

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ПІДТРИМКИ ПРОЦЕСУ СТРУКТУРУВАННЯ ДОКУМЕНТІВ НА КОМП'ЮТЕРІ МОВОЮ C#

Виконав студент 4 курсу
Групи ПД-43
Гуж Олександр Святославович
Керівник роботи
Старший викладач Коба Андрій Борисович

Київ – 2023

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи** Покращення процесу роботи з документами за рахунок розширення функціональності програмного забезпечення для підтримки процесу струкутурування документів.
- **Об'єкт дослідження** Процес структурування документів на комп'ютері
- **Предмет дослідження** Програмне забезпечення для підтримки процесу структурування документів мовою C#

АНАЛІЗ АНАЛОГІВ

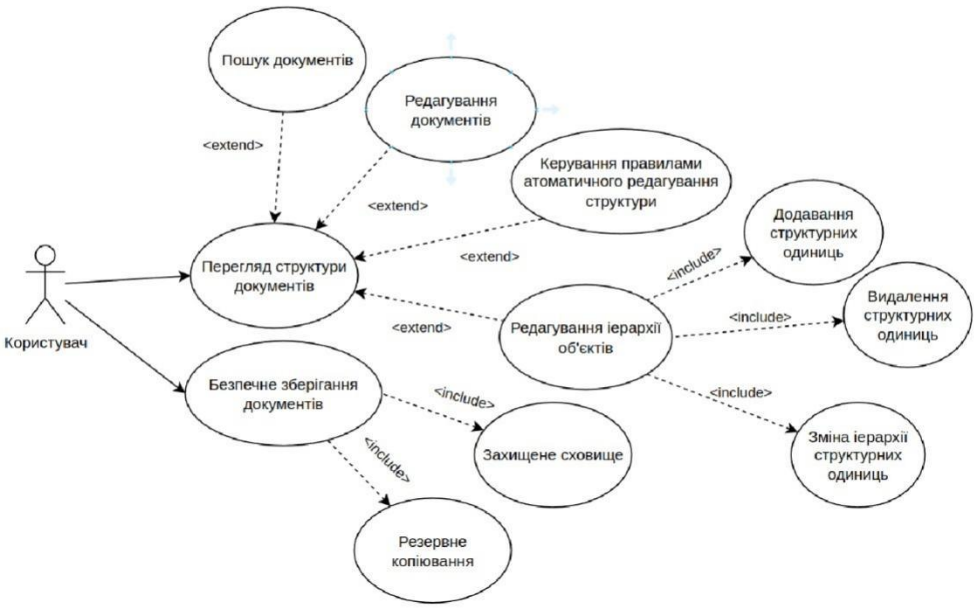
	SharePoint	File Juggler	DropIt	Розроблений застосунок
Відображення структури документів	+	-	-	+
Редагування структури документів	+	-	-	+
Правила автоматичного структурування	-	+	+	+
Резервне копіювання	-	-	-	+
Версії документів	+	-	-	+
Редагування документів	-	-	-	+
Зашифроване сховище	-	-	-	+
Платформи	Web-додаток	Desktop	Desktop	Desktop

Технічне завдання

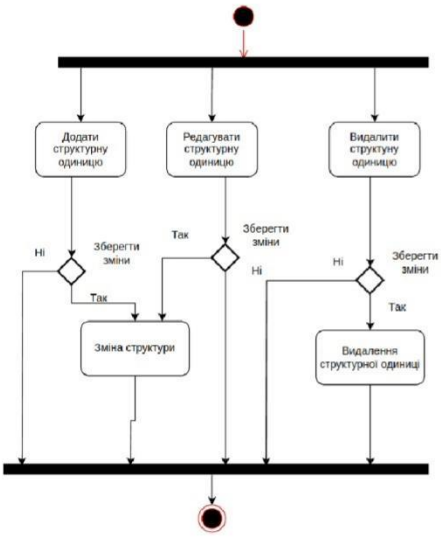
1. Забезпечити можливість перегляду структури документів в застосунку
2. Реалізувати можливість коригування структури документів
3. Реалізувати можливість редагування документів
4. Реалізувати зберігання версій документів при їх редагуванні
5. Реалізувати функцію резервного копіювання
6. Реалізувати можливість зберігання документів у зашифрованому вигляді

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

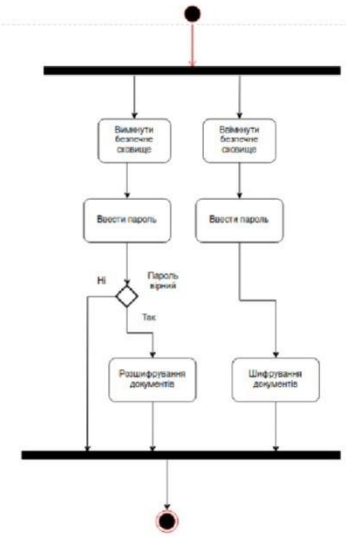
Діаграма прецендентів



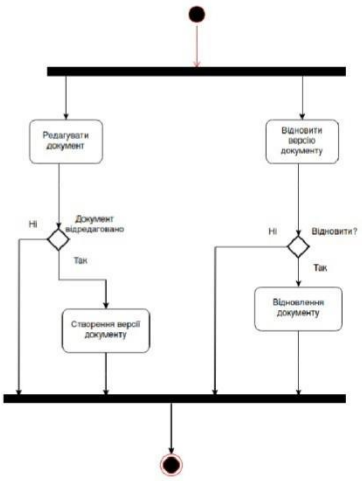
Діаграми діяльності



Зміна структури документів

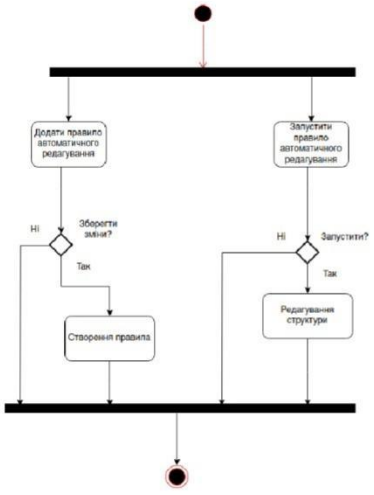


Налаштуваннння зашифрованого сховища документів



Редагування документу

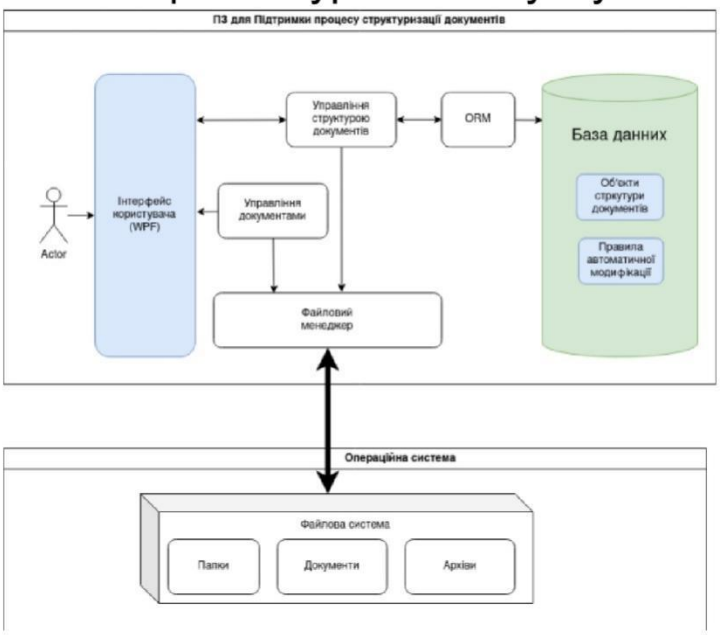
Діаграми діяльності



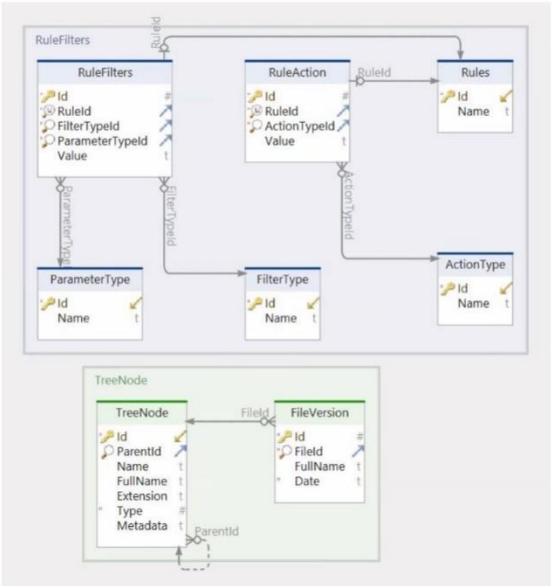
Правила автоматичного редагування

Сканування папки

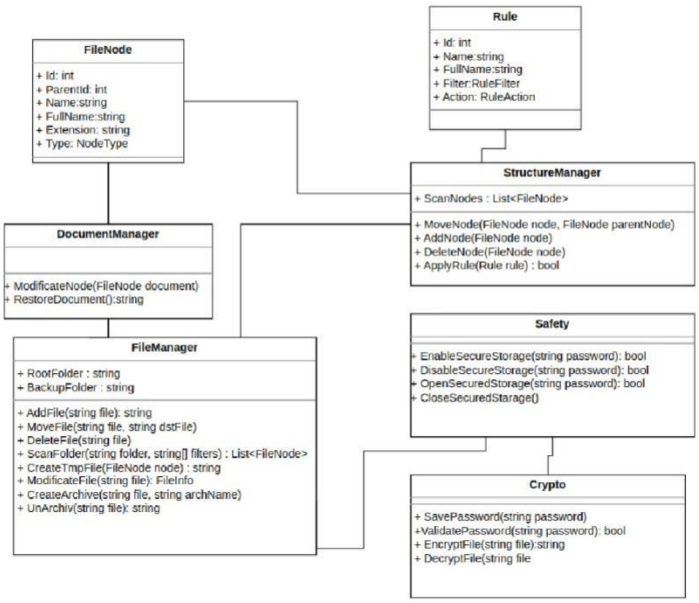
Архітектура застосунку



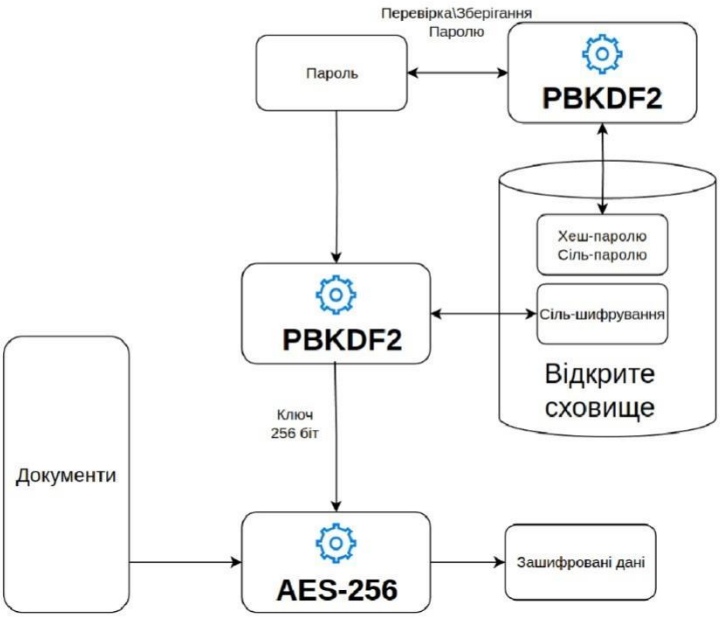
Діаграма бази даних



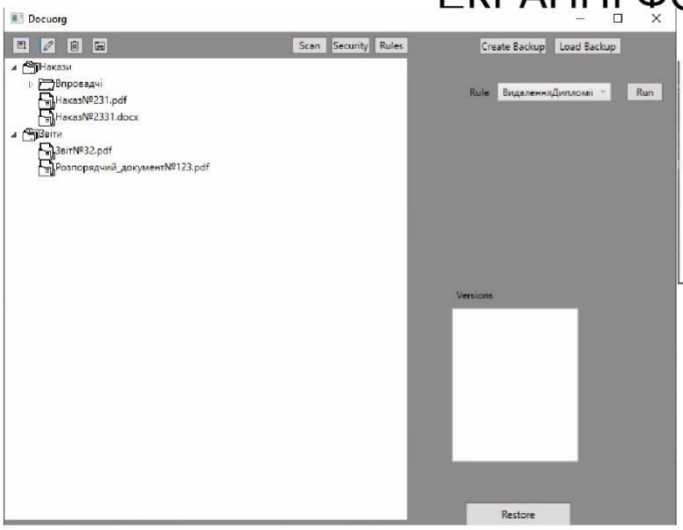
Діаграма класів



Криптографічна схема



ЕКРАННІ ФОРМИ



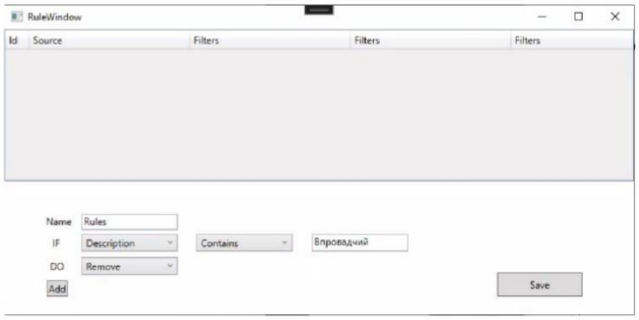
Головне меню

Додавання структурної одиниці

Налаштування захисту

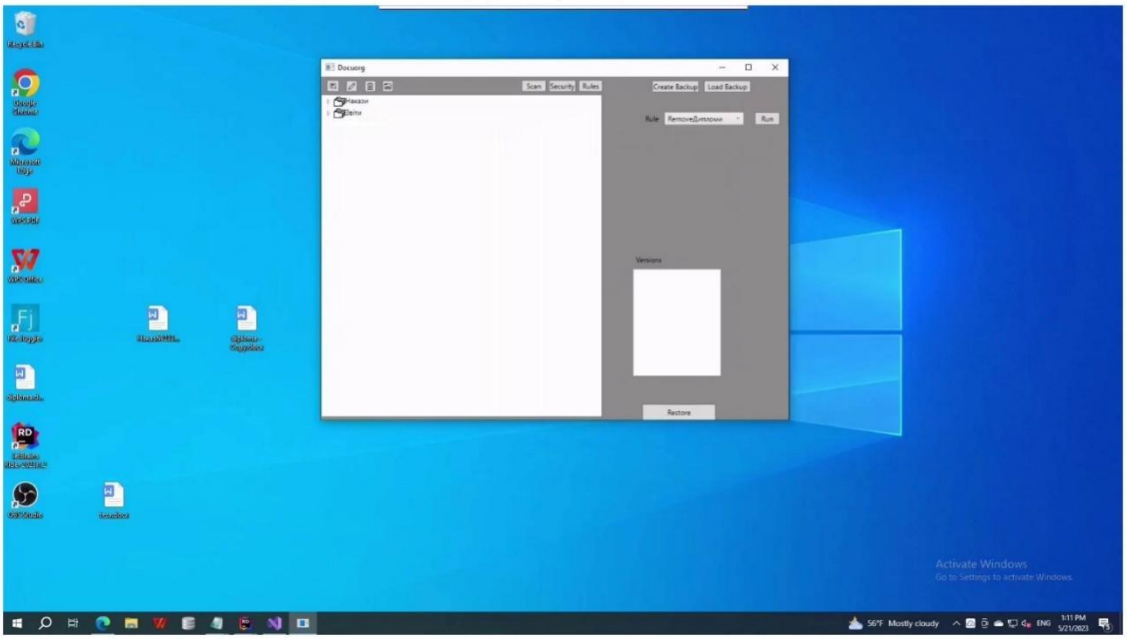
ЕКРАННІ ФОРМИ

Додавання сканованих документів



Додавання правил автоматичного редагування

Відео роботи



АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Гуж О.С. Актуальність програмного забезпечення для підтримки процесу структурування документів / Гуж О.С. Коба А.Б. // Розробка програмного забезпечення для підтримки процесу структурування документів на комп'ютері мовою C#: Матеріали третьої міжнародної науково-технічної конференції. Збірник тез. 04.06.2017, ДУТ, м. Київ — К.: ДУТ, 2023.
2. Гуж О.С. Розробка програмного забезпечення для підтримки процесу структурування документів / Гуж О.С. Коба А.Б. // Розробка програмного забезпечення для підтримки процесу структурування документів на комп'ютері мовою C#: Матеріали третьої міжнародної науково-технічної конференції. Збірник тез. 04.06.2017, ДУТ, м. Київ — К.: ДУТ, 2023.

ВИСНОВКИ

1. Проаналізовано процес структурування документів, виявлено основні потреби користувача
2. На базі аналізу аналогів визначено функціональні та нефункціональні вимоги до ПЗ.
3. Описані програмні засоби та інструменти, за допомогою яких було реалізовано програмне рішення
4. Спроектовано архітектуру та розроблено desktop застосунок для підтримки процесу структуризації документів на комп'ютері

ДЯКУЮ ЗА УВАГУ!

ДОДАТОК Б

```
using System;
using System.Configuration;
using System.IO;
using System.Linq;
using System.Runtime.InteropServices;
using System.Security;
using System.Security.Cryptography;
using System.Windows.Forms;

namespace Docuorg.Core
{
    public static class Crypto
    {
        static Crypto()
        {
            _pwdSalt =
Convert.FromBase64String(ConfigurationManager.AppSettings.Get("pwdSalt") ??
string.Empty);
            _pwdHash =
Convert.FromBase64String(ConfigurationManager.AppSettings.Get("pwdHash") ??
string.Empty);
        }
        [DllImport("KERNEL32.DLL", EntryPoint = "RtlZeroMemory")]
        public static extern bool ZeroMemory(IntPtr Destination, int Length);

        private static byte[] _asSalt;
        private static byte[] GetAESSalt()
        {
            if (_asSalt == null)
            {
                var salt =
Convert.FromBase64String(ConfigurationManager.AppSettings.Get("asSalt") ?? string.Empty);
                if (salt.Length == 0)
                {
                    _asSalt = GenerateRandomSalt();
                    ConfigurationManager.AppSettings.Set("asSalt",
Convert.ToBase64String(_asSalt));
                }
                else
                {
                    _asSalt = salt;
                }
            }
            return _asSalt;
        }

        private static byte[] _pwdHash;
        private static byte[] _pwdSalt;

        private static void SaveSalt()
        {
            ConfigurationManager.AppSettings.Set("asSalt",
Convert.ToBase64String(_asSalt));
        }

        public static byte[] GenerateRandomSalt()
        {
            byte[] data = new byte[32];

            using (RNGCryptoServiceProvider rng = new RNGCryptoServiceProvider())
            {
```

```
                for (int i = 0; i < 10; i++)
                {
                    // Fille the buffer with the generated data
                    rng.GetBytes(data);
                }
            }

            return data;
        }

        public static void SavePassword(string password)
        {
            _pwdSalt = GenerateRandomSalt();
            byte[] passwordBytes = System.Text.Encoding.UTF8.GetBytes(password);
            var key = new Rfc2898DeriveBytes(passwordBytes, _pwdSalt, 50000);
            _pwdHash = key.GetBytes(256);
            ConfigurationManager.AppSettings.Set("pwdSalt",
Convert.ToBase64String(_pwdSalt));
            ConfigurationManager.AppSettings.Set("pwdHash",
Convert.ToBase64String(_pwdHash));
        }

        public static bool ValidatePassword(string password)
        {
            byte[] passwordBytes = System.Text.Encoding.UTF8.GetBytes(password);
            var key = new Rfc2898DeriveBytes(passwordBytes, _pwdSalt, 50000);
            var bytes = key.GetBytes(256);
            return bytes.SequenceEqual(_pwdHash);
        }

        public static string FileEncrypt(string inputFile, string password)
        {
            FileStream fsCrypt = new FileStream(inputFile + ".aes", FileMode.Create);

            byte[] passwordBytes = System.Text.Encoding.UTF8.GetBytes(password);
            byte[] salt = GetAESSalt();

            RijndaelManaged AES = new RijndaelManaged();
            AES.KeySize = 256;
            AES.BlockSize = 128;
            AES.Padding = PaddingMode.PKCS7;

            var key = new Rfc2898DeriveBytes(passwordBytes, salt, 50000);
            AES.Key = key.GetBytes(AES.KeySize / 8);
            AES.IV = key.GetBytes(AES.BlockSize / 8);

            AES.Mode = CipherMode.CFB;

            fsCrypt.Write(salt, 0, salt.Length);

            CryptoStream cs = new CryptoStream(fsCrypt, AES.CreateEncryptor(),
CryptoStreamMode.Write);

            FileStream fsIn = new FileStream(inputFile, FileMode.Open);

            byte[] buffer = new byte[1048576];
            int read;

            try
            {
                while ((read = fsIn.Read(buffer, 0, buffer.Length)) > 0)
                {
                    cs.Write(buffer, 0, read);
                }
```

```
                fsIn.Close();
            }
            catch (Exception ex)
            {
                Console.WriteLine("Error: " + ex.Message);
            }
            finally
            {
                cs.Close();
                fsCrypt.Close();
            }
            return fsCrypt.Name;
        }

        public static void FileDecrypt(string inputFile, string outputFile, string
password)
        {
            byte[] passwordBytes = System.Text.Encoding.UTF8.GetBytes(password);
            byte[] salt = GetAESSalt();

            FileStream fsCrypt = new FileStream(inputFile, FileMode.Open);
            fsCrypt.Read(salt, 0, salt.Length);

            RijndaelManaged AES = new RijndaelManaged();
            AES.KeySize = 256;
            AES.BlockSize = 128;
            var key = new Rfc2898DeriveBytes(passwordBytes, salt, 50000);
            AES.Key = key.GetBytes(AES.KeySize / 8);
            AES.IV = key.GetBytes(AES.BlockSize / 8);
            AES.Padding = PaddingMode.PKCS7;
            AES.Mode = CipherMode.CFB;

            CryptoStream cs = new CryptoStream(fsCrypt, AES.CreateDecryptor(),
CryptoStreamMode.Read);

            FileStream fsOut = new FileStream(outputFile, FileMode.Create);

            int read;
            byte[] buffer = new byte[1048576];

            try
            {
                while ((read = cs.Read(buffer, 0, buffer.Length)) > 0)
                {
                    Application.DoEvents();
                    fsOut.Write(buffer, 0, read);
                }
            }
            catch (CryptographicException ex_CryptographicException)
            {
                Console.WriteLine("CryptographicException error: " +
ex_CryptographicException.Message);
            }
            catch (Exception ex)
            {
                Console.WriteLine("Error: " + ex.Message);
            }

            try
            {
                cs.Close();
            }
            catch (Exception ex)
            {
                Console.WriteLine("Error by closing CryptoStream: " + ex.Message);
```

```
            }
            finally
            {
                fsOut.Close();
                fsCrypt.Close();
            }
        }
    }
}

using System;
using System.DirectoryServices.ActiveDirectory;
using System.IO;
using System.Linq;
using System.Runtime.InteropServices;
using System.Windows.Forms;
using System.Windows.Markup;
using Docuorg.Objects;
using Microsoft.EntityFrameworkCore;

namespace Docuorg.Core
{
    public class DatabaseContext : DbContext
    {
        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
            optionsBuilder.UseSqlite("Data Source=docuorg.db");
        }

        public DbSet<FileNode> TreeNode { get; set; }

        public DbSet<FileVersion> FileVersions { get; set; }
        public DbSet<Rule> Rules { get; set; }

        public DatabaseContext()
        {
            Database.EnsureCreated();
        }
        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.Entity<FileNode>()
                .HasOne(c => c.Parent)
                .WithMany(p => p.Children)
                .HasForeignKey(a => a.ParentId)
                .OnDelete(DeleteBehavior.Cascade);

            modelBuilder.Entity<FileNode>()
                .HasMany(p => p.FileVersions)
                .WithOne(c => c.File)
                .HasForeignKey(a => a.FileId)
                .OnDelete(DeleteBehavior.Cascade);

            modelBuilder.Entity<Rule>()
                .HasOne(c => c.Action)
                .WithOne(x => x.Rule)
                .HasForeignKey<RuleAction>(a => a.RuleId)
                .IsRequired(true);

            modelBuilder.Entity<Rule>()
                .HasOne(c => c.Filter)
                .WithOne(x => x.Rule)
                .HasForeignKey<RuleFilter>(a => a.RuleId)
                .IsRequired(true);

            base.OnModelCreating(modelBuilder);
```

```
        }
    }
}

using System;
using System.Collections.Generic;
using System.Configuration;
using System.Diagnostics;
using System.IO;
using System.Linq;
using Docuorg.Objects;

namespace Docuorg.Core
{
    public static class DocumentManager
    {
        public static void ModificateDocument(FileNode node)
        {

            var tmpFile = FileManager.CreateTempFile(node.FullName);
            var info = FileManager.GetFileInfo(node.FullName);
            var modDate = info.LastWriteTime;
            FileManager.EditFile(node.FullName);
            info.Refresh();
            if (modDate != info.LastWriteTime)
            {
                var fileVersion = node.FullName.Replace(node.Extension,
$"_{_getStringDate(info.LastWriteTime)}{node.Extension}");
                FileManager.ToNewFile(tmpFile, fileVersion, true);
                using (var context = new DatabaseContext())
                {
                    var newVersion = new FileVersion() { FileId = node.Id, FullName =
fileVersion, Date = info.LastWriteTime };
                    context.FileVersions.Add(newVersion);
                    context.SaveChanges();
                }
            }
            else
            {
                FileManager.DeleteFile(tmpFile);
            }
        }

        private static string _getStringDate(DateTime dateTime)
        {
            return dateTime.ToString("yyyy_MM_dd_h_mm_ss");
        }
        public static void RestoreDocument(FileNode node, FileVersion fileVersion)
        {
            var tmpFile = FileManager.CreateTempFile(node.FullName);
            var date = DateTime.Now;
            var newVersion = node.FullName.Replace(node.Extension,
$"_{_getStringDate(date)}{node.Extension}");
            FileManager.ToNewFile(fileVersion.FullName, node.FullName, true);
            FileManager.ToNewFile(tmpFile, newVersion, true);
            using (var context = new DatabaseContext())
            {
                context.FileVersions.Add(new FileVersion { FullName  = newVersion, Date =
date, FileId = node.Id});
                context.FileVersions.Remove(context.FileVersions.FirstOrDefault(c => c.Id
== fileVersion.Id));
                context.SaveChanges();
            }
        }
    }
```

```
}

using Docuorg.Objects;
using System;
using System.Collections.Generic;
using System.Configuration;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using Microsoft.EntityFrameworkCore;

namespace Docuorg.Core
{
    public class FileManager
    {
        public static string RootFolder { get; set; }

        public static string BackupFolder { get; set; }

        static FileManager()
        {
            RootFolder = ConfigurationManager.AppSettings.Get("FilesFolder");
            BackupFolder = $"{RootFolder}\\Backup";
        }

        public static void ScanFolder(ref List<FileNode> newFiles, string[] folders,
List<string> filters)
        {
            foreach (var folder in folders)
            {
                if (folder == RootFolder) continue;
                try
                {
                    var files = Directory.GetFiles(folder);
                    _addTreeNodesWithFilter(ref newFiles, files, filters);
                    var scanFolders = Directory.GetDirectories(folder);
                    ScanFolder(ref newFiles, scanFolders, filters);
                }
                catch
                {
                }
            }
        }

        public static FileNode AddFile(FileNode fileNode, bool isMove, FileNode
parentNode = null)
        {
            var path = parentNode?.FullName ?? RootFolder;
            var newFile = string.Format("{0}\\{1}", path, fileNode.Name);
            if (isMove)
            {
                File.Move(fileNode.FullName, newFile);
            }
            else
            {
                File.Copy(fileNode.FullName, newFile);
            }

            fileNode.FullName = newFile;
            return fileNode;
        }
```

```
        public static FileNode AddFolder(FileNode fileNode, FileNode parentNode = null)
        {
            var path = parentNode?.FullName ?? RootFolder;
            var newFolder = string.Format("{0}\\{1}", path, fileNode.Name);
            Directory.CreateDirectory(newFolder);
            fileNode.FullName = newFolder;
            return fileNode;
        }

        public static void RemoveFileFromTree(FileNode fileNode)
        {
            File.Delete(fileNode.FullName);
            foreach (var vFile in fileNode.FileVersions)
            {
                File.Delete(vFile.FullName);
            }
        }

        public static void EditFile(string file)
        {
            try
            {
                var proc = new Process();
                proc.StartInfo = new ProcessStartInfo(file)
                {
                    UseShellExecute = true
                };
                proc.Start();
                proc.WaitForExit();
            }
            catch
            {
            }
        }

        public static FileInfo GetFileInfo(string file)
        {
            return new FileInfo(file);
        }

        private const string dbName = "docuorg.db";

        public static void CreateBackup(string fileName)
        {
            string tmpFolder = AppDomain.CurrentDomain.BaseDirectory + "\\tmpBackup";
            Directory.CreateDirectory(tmpFolder);
            CopyDirectory(FileManager.RootFolder, tmpFolder + "\\Documents", true);
            var context = new DatabaseContext();
            context.Database.CloseConnection();
            File.Copy(AppDomain.CurrentDomain.BaseDirectory + "\\" + dbName, tmpFolder +
"\\" + dbName);

            System.IO.Compression.ZipFile.CreateFromDirectory(tmpFolder, fileName);
            Directory.Delete(tmpFolder, true);
        }

        public static bool LoadBackup(string fileName)
        {
            string tmpFolder = AppDomain.CurrentDomain.BaseDirectory + "\\tmpBackup";
            Directory.CreateDirectory(tmpFolder);
            Directory.Delete(RootFolder, true);

            System.IO.Compression.ZipFile.ExtractToDirectory(fileName, tmpFolder);
            var dbPath = tmpFolder + "\\" + dbName;
            if (!File.Exists(dbPath))
```

```
                return false;
            var context = new DatabaseContext();
            context.Database.CloseConnection();
            context.Database.EnsureDeleted();
            File.Move(dbPath, AppDomain.CurrentDomain.BaseDirectory +"\\"+ dbName, true);
            Directory.Move(tmpFolder + "\\Documents", RootFolder);

            Directory.Delete(tmpFolder, true);
            return true;
        }

        public static string CreateTempFile(string fullName)
        {
            var tmpFile = fullName + ".tmp";
            ToNewFile(fullName, tmpFile, false);
            return tmpFile;
        }

        public static void DeleteFile(string fullName)
        {
            File.Delete(fullName);
        }

        public static void MoveFolder(string folder, string newFolder)
        {
            Directory.Move(folder, newFolder);
        }

        public static void ToNewFile(string file, string newFile, bool isMove)
        {
            if (isMove)
            {
                File.Move(file, newFile, true);
            }
            else
            {
                File.Copy(file, newFile, true);
            }
        }


        public static FileNode GetFile(string fileName)
        {
            return _getTreeNodesFiltered(new string[] { fileName }).FirstOrDefault();
        }

        private static void _addTreeNodesWithFilter(ref List<FileNode> list, string[]
files, List<string> extFilter)
        {
            list.AddRange(_getTreeNodesFiltered(files.ToArray(), extFilter));
        }

        private static IEnumerable<FileNode> _getTreeNodesFiltered(string[] files,
List<string> extFilter = null)
        {
            foreach (var file in files)
            {
                var info = new FileInfo(file);
                if (extFilter != null && !extFilter.Contains(info.Extension))
                    continue;
                yield return new FileNode()
                {
                    Children = null,
                    Type = NodeType.File,
                    Extension = info.Extension,
```

```
                    Name = info.Name,
                    FullName = info.FullName
                };
            }
        }

        private static void CopyDirectory(string sourceDir, string destinationDir, bool
recursive)
        {
            // Get information about the source directory
            var dir = new DirectoryInfo(sourceDir);

            // Check if the source directory exists
            if (!dir.Exists)
                throw new DirectoryNotFoundException($"Source directory not found:
{dir.FullName}");

            // Cache directories before we start copying
            DirectoryInfo[] dirs = dir.GetDirectories();

            // Create the destination directory
            Directory.CreateDirectory(destinationDir);

            // Get the files in the source directory and copy to the destination
directory
            foreach (FileInfo file in dir.GetFiles())
            {
                string targetFilePath = Path.Combine(destinationDir, file.Name);
                file.CopyTo(targetFilePath, true);
            }

            // If recursive and copying subdirectories, recursively call this method
            if (recursive)
            {
                foreach (DirectoryInfo subDir in dirs)
                {
                    string newDestinationDir = Path.Combine(destinationDir, subDir.Name);
                    CopyDirectory(subDir.FullName, newDestinationDir, true);
                }
            }
        }
    }
}

using Docuorg.Objects;
using System;
using System.Collections.Generic;
using System.Configuration;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using Microsoft.EntityFrameworkCore;

namespace Docuorg.Core
{
    public class FileManager
    {
        public static string RootFolder { get; set; }

        public static string BackupFolder { get; set; }

        static FileManager()
```

```
        {
            RootFolder = ConfigurationManager.AppSettings.Get("FilesFolder");
            BackupFolder = $"{RootFolder}\\Backup";
        }

        public static void ScanFolder(ref List<FileNode> newFiles, string[] folders,
List<string> filters)
        {
            foreach (var folder in folders)
            {
                if (folder == RootFolder) continue;
                try
                {
                    var files = Directory.GetFiles(folder);
                    _addTreeNodesWithFilter(ref newFiles, files, filters);
                    var scanFolders = Directory.GetDirectories(folder);
                    ScanFolder(ref newFiles, scanFolders, filters);
                }
                catch
                {
                }
            }
        }

        public static FileNode AddFile(FileNode fileNode, bool isMove, FileNode
parentNode = null)
        {
            var path = parentNode?.FullName ?? RootFolder;
            var newFile = string.Format("{0}\\{1}", path, fileNode.Name);
            if (isMove)
            {
                File.Move(fileNode.FullName, newFile);
            }
            else
            {
                File.Copy(fileNode.FullName, newFile);
            }

            fileNode.FullName = newFile;
            return fileNode;
        }

        public static FileNode AddFolder(FileNode fileNode, FileNode parentNode = null)
        {
            var path = parentNode?.FullName ?? RootFolder;
            var newFolder = string.Format("{0}\\{1}", path, fileNode.Name);
            Directory.CreateDirectory(newFolder);
            fileNode.FullName = newFolder;
            return fileNode;
        }

        public static void RemoveFileFromTree(FileNode fileNode)
        {
            File.Delete(fileNode.FullName);
            foreach (var vFile in fileNode.FileVersions)
            {
                File.Delete(vFile.FullName);
            }
        }

        public static void EditFile(string file)
        {
            try
            {
                var proc = new Process();
```

```
                proc.StartInfo = new ProcessStartInfo(file)
                {
                    UseShellExecute = true
                };
                proc.Start();
                proc.WaitForExit();
            }
            catch
            {
            }
        }

        public static FileInfo GetFileInfo(string file)
        {
            return new FileInfo(file);
        }

        private const string dbName = "docuorg.db";

        public static void CreateBackup(string fileName)
        {
            string tmpFolder = AppDomain.CurrentDomain.BaseDirectory + "\\tmpBackup";
            Directory.CreateDirectory(tmpFolder);
            CopyDirectory(FileManager.RootFolder, tmpFolder + "\\Documents", true);
            var context = new DatabaseContext();
            context.Database.CloseConnection();
            File.Copy(AppDomain.CurrentDomain.BaseDirectory + "\\" + dbName, tmpFolder +
"\\" + dbName);

            System.IO.Compression.ZipFile.CreateFromDirectory(tmpFolder, fileName);
            Directory.Delete(tmpFolder, true);
        }

        public static bool LoadBackup(string fileName)
        {
            string tmpFolder = AppDomain.CurrentDomain.BaseDirectory + "\\tmpBackup";
            Directory.CreateDirectory(tmpFolder);
            Directory.Delete(RootFolder, true);

            System.IO.Compression.ZipFile.ExtractToDirectory(fileName, tmpFolder);
            var dbPath = tmpFolder + "\\" + dbName;
            if (!File.Exists(dbPath))
                return false;
            var context = new DatabaseContext();
            context.Database.CloseConnection();
            context.Database.EnsureDeleted();
            File.Move(dbPath, AppDomain.CurrentDomain.BaseDirectory +"\\"+ dbName, true);
            Directory.Move(tmpFolder + "\\Documents", RootFolder);

            Directory.Delete(tmpFolder, true);
            return true;
        }

        public static string CreateTempFile(string fullName)
        {
            var tmpFile = fullName + ".tmp";
            ToNewFile(fullName, tmpFile, false);
            return tmpFile;
        }

        public static void DeleteFile(string fullName)
        {
            File.Delete(fullName);
        }
```

```
        public static void MoveFolder(string folder, string newFolder)
        {
            Directory.Move(folder, newFolder);
        }

        public static void ToNewFile(string file, string newFile, bool isMove)
        {
            if (isMove)
            {
                File.Move(file, newFile, true);
            }
            else
            {
                File.Copy(file, newFile, true);
            }
        }


        public static FileNode GetFile(string fileName)
        {
            return _getTreeNodesFiltered(new string[] { fileName }).FirstOrDefault();
        }

        private static void _addTreeNodesWithFilter(ref List<FileNode> list, string[]
files, List<string> extFilter)
        {
            list.AddRange(_getTreeNodesFiltered(files.ToArray(), extFilter));
        }

        private static IEnumerable<FileNode> _getTreeNodesFiltered(string[] files,
List<string> extFilter = null)
        {
            foreach (var file in files)
            {
                var info = new FileInfo(file);
                if (extFilter != null && !extFilter.Contains(info.Extension))
                    continue;
                yield return new FileNode()
                {
                    Children = null,
                    Type = NodeType.File,
                    Extension = info.Extension,
                    Name = info.Name,
                    FullName = info.FullName
                };
            }
        }

        private static void CopyDirectory(string sourceDir, string destinationDir, bool
recursive)
        {
            // Get information about the source directory
            var dir = new DirectoryInfo(sourceDir);

            // Check if the source directory exists
            if (!dir.Exists)
                throw new DirectoryNotFoundException($"Source directory not found:
{dir.FullName}");

            // Cache directories before we start copying
            DirectoryInfo[] dirs = dir.GetDirectories();

            // Create the destination directory
            Directory.CreateDirectory(destinationDir);
```

```
            // Get the files in the source directory and copy to the destination
directory
            foreach (FileInfo file in dir.GetFiles())
            {
                string targetFilePath = Path.Combine(destinationDir, file.Name);
                file.CopyTo(targetFilePath, true);
            }

            // If recursive and copying subdirectories, recursively call this method
            if (recursive)
            {
                foreach (DirectoryInfo subDir in dirs)
                {
                    string newDestinationDir = Path.Combine(destinationDir, subDir.Name);
                    CopyDirectory(subDir.FullName, newDestinationDir, true);
                }
            }
        }
    }
}

using Docuorg.Objects;
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;

namespace Docuorg.Core
{
    public static class StructureManager
    {
        public static bool MoveNode(FileNode node, FileNode dstNode = null)
        {
            if (node?.Parent == null && dstNode == null)
                return false;
            var dstFolder = dstNode?.FullName ?? FileManager.RootFolder;
            using var context = new DatabaseContext();
            var nodeDb = context.TreeNode
                .Include(x => x.FileVersions)
                .Include(c => c.Children)
                .Include(p => p.Parent)
                .FirstOrDefault(n => n.Id == node.Id);
            if (nodeDb == null)
                return false;

            var targetStr = nodeDb.Parent?.FullName ?? FileManager.RootFolder;
            var newPath = nodeDb.FullName.Replace(targetStr, dstFolder);
            if (node.Type == NodeType.File)
            {
                FileManager.ToNewFile(nodeDb.FullName, newPath, true);
                nodeDb.FullName = newPath;

                foreach (var ver in node.FileVersions)
                {
                    var verPath = ver.FullName.Replace(targetStr, dstFolder);
                    FileManager.ToNewFile(ver.FullName, verPath, true);
                    ver.FullName = verPath;
                }
            }
            else
            {
                FileManager.MoveFolder(nodeDb.FullName, newPath);
                nodeDb.FullName = newPath;
                _recursivePathReplace(nodeDb.Children, targetStr, dstFolder);
```

```
            }

            if (dstNode != null)
                nodeDb.ParentId = dstNode.Id;
            context.SaveChanges();

            return true;
        }

        private static void _recursivePathReplace(IEnumerable<FileNode> nodes, string
targetStr, string replaceStr)
        {
            foreach (var node in nodes)
            {
                node.FullName = node.FullName.Replace(targetStr, replaceStr);
                if (node.Type == NodeType.File && (node?.FileVersions.Count ?? 0) == 0)
                {
                    foreach (var fileVersion in node.FileVersions)
                    {
                        fileVersion.FullName = fileVersion.FullName.Replace(targetStr,
replaceStr);
                    }
                }

                if (node.Children != null)
                    _recursivePathReplace(node.Children, targetStr, replaceStr);
            }
        }

        public static void AddNode(FileNode fileNode)
        {
            FileNode parentNode = null;
            using (var context = new DatabaseContext())
            {
                if (fileNode.ParentId != null || fileNode.ParentId != 0)
                    parentNode = context.TreeNode.FirstOrDefault(n => n.Id ==
fileNode.ParentId);

                if (fileNode.Type == NodeType.File)
                {
                    var newFile = FileManager.AddFile(fileNode, false, parentNode);
                    context.TreeNode.Add(newFile);
                }
                else
                {
                    var newFolder = FileManager.AddFolder(fileNode, parentNode);
                    context.TreeNode.Add(newFolder);
                }

                context.SaveChanges();
            }
        }

        public static void RemoveNode(FileNode fileNode)
        {
            using (var context = new DatabaseContext())
            {
                if (fileNode.Type == NodeType.File)
                {
                    FileManager.RemoveFileFromTree(fileNode);
                }
                else
                {
                    Directory.Delete(fileNode.FullName, true);
                }
```

```
                context.TreeNode.Remove(context.TreeNode.FirstOrDefault(n => n.Id ==
fileNode.Id));
            }
        }

        public static int ApplyRule(Rule applyRule)
        {
            Dictionary<FileNode, string> values;
            List<FileNode> nodes;
            List<FileNode> filteredNodes;
            using var context = new DatabaseContext();

            nodes = context.TreeNode.ToList();
            var rule = context.Rules.Include(x => x.Action).Include(c => c.Filter)
                .FirstOrDefault(r => r.Id == applyRule.Id);
            var parameter = (ParameterType)Enum.GetNames(typeof(ParameterType)).ToList()
                .FindIndex(x => x == rule.Filter.ParameterType);
            switch (parameter)
            {
                case ParameterType.Name:

                    values = nodes.Select(x => new { x, x.Name }).ToDictionary(p => p.x,
c => c.Name);
                    break;
                case ParameterType.Extension:
                    values = nodes.Select(x => new { x, x.Extension }).ToDictionary(p =>
p.x, c => c.Extension);
                    break;
                default:
                    return 0;
            }

            var filter = (FilterType)Enum.GetNames(typeof(FilterType)).ToList()
                .FindIndex(x => x == rule.Filter.FilterType);

            switch (filter)
            {
                case FilterType.Is:
                    filteredNodes = values.Where(c => c.Value ==
rule.Filter.Value).Select(x => x.Key).ToList();
                    break;
                case FilterType.IsNot:
                    filteredNodes = values.Where(c => c.Value !=
rule.Filter.Value).Select(x => x.Key).ToList();
                    break;
                case FilterType.Contains:
                    filteredNodes = values.Where(c =>
c.Value.Contains(rule.Filter.Value)).Select(x => x.Key).ToList();
                    break;
                case FilterType.NotContains:
                    filteredNodes = values.Where(c =>
!c.Value.Contains(rule.Filter.Value)).Select(x => x.Key).ToList();
                    break;
                case FilterType.Empty:
                    filteredNodes = values.Where(c =>
string.IsNullOrWhiteSpace(c.Value)).Select(x => x.Key).ToList();
                    break;
                default:
                    return 0;
            }

            var action = (ActionType)Enum.GetNames(typeof(ActionType)).ToList()
                .FindIndex(x => x == rule.Action.ActionType);
```

```
            switch (action)
            {
                case ActionType.Move:
                    var targetNode = nodes.FirstOrDefault(x => x.Name ==
rule.Action.Value);
                    if (targetNode == null)
                        return 0;
                    filteredNodes.ForEach(c => MoveNode(c, targetNode));
                    break;
                case ActionType.Remove:
                    filteredNodes.ForEach(RemoveNode);
                    break;
            }

            return filteredNodes.Count;
        }
    }
}

using System.Collections.ObjectModel;
using System.ComponentModel.DataAnnotations;

namespace Docuorg.Objects
{
    public class FileNode
    {
        [Key]
        public int Id { get; set; }

        public int? ParentId { get; set; }
        public FileNode Parent { get; set; }

        public ObservableCollection<FileNode> Children { get; set; }

        public string Name { get; set; }
        public string FullName { get; set; }
        public string Extension { get; set; }
        public NodeType Type { get; set; }
        public string Metadata { get; set; }

        public ObservableCollection<FileVersion> FileVersions { get; set; }
    }



    public enum
        NodeType
    {
        File = 0,
        Directory = 1,
        Category = 2
    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Docuorg.Objects
{
    public class FileVersion
    {
        public int Id { get;set; }
```

```
        public int FileId { get; set; }
        public FileNode File { get; set; }

        public string FullName { get; set; }
        public string Name { get; set; }

        public DateTime Date { get; set; }
    }
}
using System.Collections.Generic;

namespace Docuorg.Objects
{
    public class Rule
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public RuleFilter Filter { get; set; }
        public RuleAction Action { get; set; }
    }
}

using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Docuorg.Objects
{
    public class RuleAction
    {
        [Key]
        public int Id { get; set; }

        public int RuleId { get; set; }
        public Rule Rule { get; set; }

        public string ActionType { get; set; }
        public string Value { get; set; }

    }
    public enum ActionType
    {
        Move,
        Remove
    }
}
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using System.Windows.Documents;

namespace Docuorg.Objects
{
    public class RuleFilter
    {
        [Key]
        public int Id { get; set; }

        public int RuleId { get; set; }
        public Rule Rule { get; set; }
```

```
        public string FilterType { get; set; }

        public string ParameterType { get; set; }
        public string Value { get; set; }
    }


    public enum FilterType
    {
        Is,
        IsNot,
        Contains,
        NotContains,
        Empty
    }

    public enum ParameterType
    {
        Name,
        Extension
    }
}

using GongSolutions.Wpf.DragDrop;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.ComponentModel.DataAnnotations.Schema;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using Docuorg.Core;
using Docuorg.Objects;
using GongSolutions.Wpf.DragDrop.Utilities;
using DragDropEffects = System.Windows.DragDropEffects;
using IDropTarget = GongSolutions.Wpf.DragDrop.IDropTarget;

namespace Docuorg.ViewModels
{
    public class MainViewModel
    {
        public static event EventHandler OnTreeChange;
        public static IDropTarget NestedDropHandler { get; set; } = new
NestedDropHandler();

        public static void OnOnTreeChange()
        {
            OnTreeChange?.Invoke(null, EventArgs.Empty);
        }
    }

    public class NestedDropHandler : IDropTarget
    {
#if !NETCOREAPP3_1_OR_GREATER
        /// <inheritdoc />
        public void DragEnter(IDropInfo dropInfo)
        {
            // nothing here
        }
#endif

        /// <inheritdoc />
        public void DragOver(IDropInfo dropInfo)
        {
```

```
            if (dropInfo.TargetItem?.ToString().StartsWith("Root",
StringComparison.OrdinalIgnoreCase) == true)
            {
                dropInfo.Effects = DragDropEffects.Move;
                dropInfo.DropTargetAdorner = DropTargetAdorners.Highlight;
            }
        }

#if !NETCOREAPP3_1_OR_GREATER
        /// <inheritdoc />
        public void DragLeave(IDropInfo dropInfo)
        {
            // nothing here
        }
#endif

        /// <inheritdoc />
        public void Drop(IDropInfo dropInfo)
        {
            var srcItem = dropInfo.DragInfo.SourceItem as FileNode;

            var targetItem = dropInfo?.TargetItem as FileNode;
            using (var context = new DatabaseContext())
            {
                StructureManager.MoveNode(srcItem, targetItem);
            }

            MainViewModel.OnOnTreeChange();
        }
    }
}
<Window x:Class="Docuorg.Views.AddWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:Docuorg"
        xmlns:StyleAlias="clr-namespace:Docuorg.Objects"
        mc:Ignorable="d"
        Title="AddWindow" Height="170" Width="439">
    <Grid>
        <Button Content="Add" HorizontalAlignment="Left" Margin="354,107,0,0"
VerticalAlignment="Top" Click="Button_Click" Height="24" Width="61"/>
        <Label Content="Name" HorizontalAlignment="Left" Margin="10,51,0,0"
VerticalAlignment="Top" Height="26"/>
        <ComboBox HorizontalAlignment="Left" Margin="295,55,0,0" VerticalAlignment="Top"
Width="120" Name="nType" SelectionChanged="Type_SelectionChanged"/>
        <TextBox HorizontalAlignment="Left" Margin="79,57,0,0" TextWrapping="Wrap"
Text="" VerticalAlignment="Top" Width="186" Name ="nName"/>
        <Button Content="..." HorizontalAlignment="Left" Margin="270,56,0,0"
VerticalAlignment="Top" Width="20" Name="fPath" Visibility="Hidden"
Click="GetPath_Click"/>
        <TextBox HorizontalAlignment="Left" Margin="79,80,0,0" TextWrapping="Wrap"
Text="" VerticalAlignment="Top" Width="336" Name="fName" IsReadOnly="True"
Visibility="Hidden"/>
        <Label Content="FileName" HorizontalAlignment="Left" Margin="10,76,0,0"
VerticalAlignment="Top" Height="26" Width="68" Visibility="Hidden" Name="fNLabel"/>

    </Grid>
</Window>


using Docuorg.Objects;
using Microsoft.Win32;
using System;
```

```
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;

namespace Docuorg.Views
{
    /// <summary>
    /// Interaction logic for AddWindow.xaml
    /// </summary>
    public partial class AddWindow : Window
    {
        public AddWindow()
        {
            InitializeComponent();
            nType.ItemsSource = Enum.GetValues(typeof(NodeType)).Cast<NodeType>();
        }

        private void Button_Click(object sender, RoutedEventArgs e)
        {
            if (string.IsNullOrWhiteSpace(nName.Text))
            {
                MessageBox.Show("Empty node name");
                return;
            }

            this.DialogResult = true;
        }

        private void Type_SelectionChanged(object sender, SelectionChangedEventArgs e)
        {
            switch ((NodeType)nType.SelectedIndex)
            {
                case NodeType.File:
                    fPath.Visibility = fName.Visibility = fNLabel.Visibility =
Visibility.Visible;
                    break;
                case NodeType.Directory:
                case NodeType.Category:
                    fPath.Visibility = fName.Visibility = fNLabel.Visibility =
Visibility.Hidden;
                    break;
            }
        }

        private void GetPath_Click(object sender, RoutedEventArgs e)
        {
            var openFileDialog = new OpenFileDialog();
            if (openFileDialog.ShowDialog() == true)
            {
                fName.Text = openFileDialog.FileName;
                nName.Text = new FileInfo(fName.Text).Name;
            }
        }
    }
}
```

```
<Window x:Class="Docuorg.Views.PasswordRequest"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:local="clr-namespace:Docuorg.Views"
        mc:Ignorable="d"
        Title="PasswordRequest" Height="139" Width="424">
    <Grid RenderTransformOrigin="0.5,0.5">
        <Grid.RenderTransform>
            <TransformGroup>
                <ScaleTransform/>
                <SkewTransform/>
                <RotateTransform Angle="0.354"/>
                <TranslateTransform/>
            </TransformGroup>
        </Grid.RenderTransform>
        <Label Content="Password" HorizontalAlignment="Left" Margin="37,34,0,0"
VerticalAlignment="Top"/>
        <PasswordBox HorizontalAlignment="Left" Margin="126,38,0,0"
VerticalAlignment="Top" Width="120" Name="password"/>
        <Button Content="Log In" HorizontalAlignment="Left" Margin="315,72,0,0"
VerticalAlignment="Top" Width="60"/>

    </Grid>
</Window>
using System.Windows;

namespace Docuorg.Views;

public partial class PasswordRequest : Window
{
    public PasswordRequest()
    {
        InitializeComponent();
    }
}

<Window x:Class="Docuorg.Views.RuleWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:Docuorg"
        mc:Ignorable="d"
        Title="RuleWindow" Height="394" Width="800">
    <Grid>
        <Label Content="IF" HorizontalAlignment="Left" Margin="55,253,0,0"
VerticalAlignment="Top" Height="24" Width="22"/>
        <Label Content="DO" HorizontalAlignment="Left" Margin="52,282,0,0"
VerticalAlignment="Top"/>
        <ComboBox HorizontalAlignment="Left" Margin="96,255,0,0" VerticalAlignment="Top"
Width="120" Name="parameterType" SelectedValue="0"/>
        <TextBox HorizontalAlignment="Left" Margin="383,255,0,0" TextWrapping="Wrap"
Text="" VerticalAlignment="Top" Width="120" Height="22" Name="filterValue"/>
        <ComboBox HorizontalAlignment="Left" Margin="239,255,0,0" VerticalAlignment="Top"
Width="120" SelectedValue="0" Name="filterType"/>
        <DataGrid Grid.Row="0" Grid.Column="0" AutoGenerateColumns="False"
CanUserAddRows="False" Name="dgContent" Margin="0,0,0,166" ItemsSource="{Binding View}">
            <DataGrid.Columns>
                <DataGridTextColumn Header="Id" Width="30" Binding="{Binding Id}"/>
                <DataGridTextColumn Header="Name" Width="100" Binding="{Binding Name}"/>
                <DataGridTextColumn Header="Parameter" Width="100" Binding="{Binding
Filter.ParameterType}"/>
```

```
                <DataGridTextColumn Header="Filter" Width="100" Binding="{Binding
Filter.FilterType}"/>
                <DataGridTextColumn Header="Value" Width="100" Binding="{Binding
Filter.Value}"/>
                <DataGridTextColumn Header="Action" Width="100" Binding="{Binding
Action.ActionType}"/>
                <DataGridTextColumn Header="Value" Width="100" Binding="{Binding
Action.Value}"/>
                <DataGridTemplateColumn Header="Delete" Width="100">
                    <DataGridTemplateColumn.CellTemplate>
                        <DataTemplate>
                            <Button Content="X" Click="RemoveSource_Click"/>
                        </DataTemplate>
                    </DataGridTemplateColumn.CellTemplate>
                </DataGridTemplateColumn>
            </DataGrid.Columns>
        </DataGrid>
        <Label Content="Name" HorizontalAlignment="Left" Height="27" Margin="47,226,0,0"
VerticalAlignment="Top" Width="44"/>
        <TextBox HorizontalAlignment="Left" Height="20" Margin="96,230,0,0"
TextWrapping="Wrap" Text="" VerticalAlignment="Top" Width="120" Name="ruleName"/>
        <ComboBox HorizontalAlignment="Left" Margin="96,284,0,0" VerticalAlignment="Top"
Width="120" Name="actionType" SelectedValue="0" />
        <Button Content="Add" HorizontalAlignment="Left" Margin="53,313,0,0"
VerticalAlignment="Top" Click="Add_Click"/>
        <Button Content="Save" HorizontalAlignment="Left" Height="30"
Margin="614,303,0,0" VerticalAlignment="Top" Width="106" Click="Save_Click"/>
        <TextBox HorizontalAlignment="Left" Margin="239,284,0,0" TextWrapping="Wrap"
Text="" VerticalAlignment="Top" Width="120" Height="22" Name="actionValue"/>

    </Grid>
</Window>
```

```
using Docuorg.Objects;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;
using Docuorg.Core;

namespace Docuorg.Views
{
    /// <summary>
    /// Interaction logic for RuleWindow.xaml
    /// </summary>
    public partial class RuleWindow : Window
    {
        public List<Rule> Rules = new List<Rule>();
        public CollectionView View;

        public RuleWindow()
        {
            InitializeComponent();
            View = new CollectionView(Rules);
```

```
            dgContent.ItemsSource = View;
            filterType.ItemsSource =
Enum.GetValues(typeof(FilterType)).Cast<FilterType>();
            parameterType.ItemsSource =
Enum.GetValues(typeof(ParameterType)).Cast<ParameterType>();
            actionType.ItemsSource =
Enum.GetValues(typeof(ActionType)).Cast<ActionType>();
        }

        private void Save_Click(object sender, RoutedEventArgs e)
        {
            DialogResult = true;
        }

        private void Add_Click(object sender, RoutedEventArgs e)
        {
            using (var context = new DatabaseContext())
            {
                var newRule = new Rule
                {
                    Name = ruleName.Text
                };
                context.Rules.Add(newRule);
                context.SaveChanges();
                newRule.Action = new RuleAction
                {
                    ActionType = actionType.SelectedValue.ToString(), Value =
actionValue.Text, RuleId = newRule.Id
                };
                newRule.Filter = new RuleFilter
                {
                    RuleId = newRule.Id,
                    FilterType = filterType.SelectedValue.ToString(),
                    ParameterType = parameterType.SelectedValue.ToString(),
                    Value = filterValue.Text
                };
                context.SaveChanges();
                Rules.Add(newRule);
            }

            View.Refresh();
        }

        private void RemoveSource_Click(object sender, RoutedEventArgs e)
        {
            using (var context = new DatabaseContext())
            {
                var rule = (Rule)dgContent.SelectedItem;
                context.Remove(context.Rules.FirstOrDefault(x => x.Id == rule.Id));
                context.SaveChanges();
                Rules.Remove((Rule)dgContent.SelectedItem);
                View.Refresh();
            }
        }
    }
}

<Window x:Class="Docuorg.Views.ScanWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:Docuorg"
        mc:Ignorable="d"
        Title="ScanWindow" Height="437" Width="764">
```

```
    <Grid>
        <ListBox Height="287" HorizontalAlignment="Center" Margin="0,10,0,0"
Name="listFiles" VerticalAlignment="Top" Width="728" Background="#0B000000"
RenderTransformOrigin="0.5,0.5">
            <ListBox.RenderTransform>
                <TransformGroup>
                    <ScaleTransform/>
                    <SkewTransform AngleX="-0.501"/>
                    <RotateTransform/>
                    <TranslateTransform X="-1.491"/>
                </TransformGroup>
            </ListBox.RenderTransform>
            <ListBox.ItemTemplate>
                <DataTemplate>
                    <StackPanel>
                        <CheckBox Content="{Binding Name}" IsChecked="{Binding
IsChecked}"/>
                    </StackPanel>
                </DataTemplate>
            </ListBox.ItemTemplate>
        </ListBox>
        <Button Content="Add" HorizontalAlignment="Left" Margin="627,316,0,0"
VerticalAlignment="Top" Height="45" Width="99" Click="Button_Click"/>
    </Grid>
</Window>
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;
using System.Windows.Forms;
using System.Collections.ObjectModel;
using Docuorg.Objects;
using Docuorg.Core;

namespace Docuorg.Views
{
    /// <summary>
    /// Interaction logic for ScanWindow.xaml
    /// </summary>
    public partial class ScanWindow : Window
    {
        private ObservableCollection<FileList> files = new
ObservableCollection<FileList>();
        public List<FileNode> Nodes;
        public ScanWindow()
        {
            InitializeComponent();
            listFiles.ItemsSource = files;
            foreach (var file in Nodes)
            {
                files.Add(new FileList() { Name = file.FullName, IsChecked = true, Node =
file });
            }
        }
```

```
        private void Button_Click(object sender, RoutedEventArgs e)
        {
            Nodes = files.Select(x => x.Node).ToList();
            this.DialogResult = true;
        }
    }

    public class FileList
    {
        public string Name { get; set; }

        public bool IsChecked { get; set; }

        public FileNode Node { get; set; }
    }
}
```

```
<Window x:Class="Docuorg.Views.SecurityWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:Docuorg"
        mc:Ignorable="d"
        Title="SecurityWindow" Height="108" Width="365">
    <Grid>
        <PasswordBox HorizontalAlignment="Left" Margin="112,47,0,0"
VerticalAlignment="Top" Width="120" Name="password"/>
        <Label Content="Password" HorizontalAlignment="Left" Margin="38,43,0,0"
VerticalAlignment="Top" Name="pLabel"/>
        <CheckBox Content="Enable Secured Storage" HorizontalAlignment="Left"
Margin="38,17,0,0" VerticalAlignment="Top" Checked="CheckBox_Checked"
Name="EnableSecuredStorage"/>
        <Button Content="Save" HorizontalAlignment="Left" Margin="292,45,0,0"
VerticalAlignment="Top" Width="52" Click="Button_Click"/>

    </Grid>
</Window>
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;
using Docuorg.Core;

namespace Docuorg.Views
{
    /// <summary>
    /// Interaction logic for SecurityWindow.xaml
    /// </summary>
    public partial class SecurityWindow : Window
    {
        public SecurityWindow()
        {
            InitializeComponent();
            pLabel.Visibility = password.Visibility = Visibility.Hidden;
```

```
            EnableSecuredStorage.IsChecked = Safety.IsSecureStorageEnabled;
        }

        private void CheckBox_Checked(object sender, RoutedEventArgs e)
        {
            pLabel.Visibility = password.Visibility = EnableSecuredStorage.IsChecked !=
Safety.IsSecureStorageEnabled ? Visibility.Visible : Visibility.Hidden;
        }

        private void Button_Click(object sender, RoutedEventArgs e)
        {
            if (string.IsNullOrEmpty(password.Password))
            {
                MessageBox.Show("Empty password");
                return;
            }

            if (EnableSecuredStorage.IsChecked != Safety.IsSecureStorageEnabled)
            {
                if(EnableSecuredStorage.IsChecked ?? false)
                {
                    Safety.EnableSecureStorage(password.Password);
                }
                else
                {
                    if(!Safety.DisableSecureStorage(password.Password))
                    {
                        MessageBox.Show("Invalid password");
                    }
                }
            }

            DialogResult = true;
        }
    }
}
```

```
<Window x:Class="Docuorg.MainWindow"
        xmlns:Converters="clr-namespace:Docuorg.Converters"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:viewModels="clr-namespace:Docuorg.ViewModels"
        xmlns:views="clr-namespace:Docuorg.Views"
        d:DataContext="{d:DesignInstance viewModels:MainViewModel}"
        xmlns:dd="urn:gong-wpf-dragdrop"
        xmlns:local="clr-namespace:Docuorg"
        mc:Ignorable="d"
        Title="Docuorg" Height="628" Width="818">

    <Grid Background="#FF8E8D8D">
        <TreeView Margin="0,31,320,5" RenderTransformOrigin="0.5,0.5" Name="FileTree"
dd:DragDrop.DropHandler="{Binding NestedDropHandler}" dd:DragDrop.IsDragSource="True"
                  dd:DragDrop.IsDropTarget="True"
dd:DragDrop.UseDefaultEffectDataTemplate="True">
            <TreeView.Resources>
                <Converters:NodeTypeConverter x:Key="NodeTypeConverter" />
            </TreeView.Resources>
            <TreeView.ItemTemplate>
                <HierarchicalDataTemplate ItemsSource="{Binding Path=Children}">
                    <StackPanel Orientation="Horizontal">
                        <Image Source="{Binding Type,Converter={StaticResource
NodeTypeConverter}}" Width="20"
```

```
                            Height="20" />
                        <TextBlock Text="{Binding Path=Name}" />
                    </StackPanel>
                </HierarchicalDataTemplate>
            </TreeView.ItemTemplate>
        </TreeView>
        <Button FontFamily="Segoe MDL2 Assets" Content="&#xED0E;"
HorizontalAlignment="Left" Margin="10,6,0,0"
                VerticalAlignment="Top" Click="Add_Click" Height="20" Width="21" />
        <Button FontFamily="Segoe MDL2 Assets" Content="&#xE70F;"
HorizontalAlignment="Left" Margin="43,7,0,0"
                VerticalAlignment="Top" Height="19" Width="19" Click="Edit_Click" />
        <Button FontFamily="Segoe MDL2 Assets" Content="&#xE74D;"
HorizontalAlignment="Left" Margin="75,7,0,0"
                VerticalAlignment="Top" Height="19" Width="19" Click="Delete_Click" />
        <Button Content="Rules" HorizontalAlignment="Left" Margin="442,6,0,0"
VerticalAlignment="Top" Width="41"
                Click="Rules_Click" />
        <Button Content="Security" HorizontalAlignment="Left" Margin="391,6,0,0"
VerticalAlignment="Top"
                Click="Security_Click" />
        <Button Content="Scan" HorizontalAlignment="Left" Margin="344,6,0,0"
VerticalAlignment="Top" Click="Scan_Click"
                Width="42" />
        <ListBox Margin="535,330,116,75" Name="versionList" ItemsSource="{Binding
versionView}" >
            <ListBox.ItemTemplate>
                <DataTemplate>
                    <TextBlock Text="{Binding Path=Date}" />
                </DataTemplate>
            </ListBox.ItemTemplate>
        </ListBox>
        <Label Content="Versions" HorizontalAlignment="Left" Margin="535,301,0,0"
VerticalAlignment="Top" />
        <Button Content="Restore" HorizontalAlignment="Left" Margin="551,562,0,0"
VerticalAlignment="Top" Height="28"
                Width="126" Click="restore_Click" />
        <Label Content="Rule" HorizontalAlignment="Left" Margin="551,59,0,0"
VerticalAlignment="Top" />
        <ComboBox HorizontalAlignment="Left" Margin="589,61,0,0" VerticalAlignment="Top"
Width="136" Name="rulesList">
            <ComboBox.ItemTemplate>
                <DataTemplate>
                    <TextBlock Text="{Binding Path=Name}" />
                </DataTemplate>
            </ComboBox.ItemTemplate>
        </ComboBox>
        <Button Content="Run" HorizontalAlignment="Left" Margin="744,61,0,0"
VerticalAlignment="Top" Width="43"
                Height="22" Click="ruleRun_Click"/>
        <Button Content="Create Backup" HorizontalAlignment="Left" Margin="568,7,0,0"
VerticalAlignment="Top" Click="createBackup_Click"/>
        <Button Content="Load Backup" HorizontalAlignment="Left" Margin="657,7,0,0"
VerticalAlignment="Top" Width="80" Click="loadBackup_Click"/>

    </Grid>
</Window>
```

```
using System.Windows;
using System.Configuration;
using System;
using System.Collections.Generic;
using Docuorg.Objects;
```

```
using System.Linq;
using System.Windows.Data;
using System.IO;
using Microsoft.EntityFrameworkCore;
using System.Collections.ObjectModel;
using System.Globalization;
using System.Windows.Forms;
using Docuorg.Core;
using Docuorg.ViewModels;
using Docuorg.Views;
using GongSolutions.Wpf.DragDrop.Utilities;
using MessageBox = System.Windows.MessageBox;
using OpenFileDialog = Microsoft.Win32.OpenFileDialog;
using RuleFilter = Docuorg.Objects.RuleFilter;

namespace Docuorg
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow
    {
        ObservableCollection<FileNode> Nodes;
        CollectionView treeView;

        List<FileVersion> versions = new List<FileVersion>();
        CollectionView versionView;

        private List<Rule> Rules = new List<Rule>();
        CollectionView ruleView;


        public MainWindow()
        {
            try
            {
                if (Safety.IsSecureStorageEnabled)
                {
                    Safety.OpenSecuredStorage();
                }

                InitializeComponent();
                MainViewModel.OnTreeChange += MainViewModelOnOnTreeChange;
                FileTree.SetDropHandler(MainViewModel.NestedDropHandler);
                FileTree.SelectedItemChanged += selectLoad;
                versionView = new CollectionView(versions);
                versionList.ItemsSource = versionView;
                var isFirstStart = ConfigurationManager.AppSettings.Get("IsFirstStart");
                if (string.IsNullOrWhiteSpace(isFirstStart))
                {
                    var newDir = $"{AppDomain.CurrentDomain.BaseDirectory}Documents";
                    //if (Directory.Exists(newDir))
                    //Directory.Delete(newDir, true);
                    Directory.CreateDirectory(newDir);
                    ConfigurationManager.AppSettings.Set("FilesFolder", newDir);
                    ConfigurationManager.AppSettings.Set("IsFirstStart", "false");
                }

                using (var context = new DatabaseContext())
                {
                    var nodes = context.TreeNode.ToList();

                    Nodes = new ObservableCollection<FileNode>(context.TreeNode.Where(t
=> t.ParentId == null)
                        .Include(a => a.Children)
```

```
                        .Include(p => p.Parent)
                        .Include(v => v.FileVersions).ToList());
                    treeView = new CollectionView(Nodes);
                    FileTree.ItemsSource = treeView;

                    Rules = context.Rules.ToList();
                    ruleView = new CollectionView(Rules);
                    rulesList.ItemsSource = ruleView;
                }
            }
            catch (Exception ex)
            {
                System.Windows.MessageBox.Show(ex.Message);
            }
        }

        private void MainViewModelOnOnTreeChange(object sender, EventArgs e)
        {
            using var context = new DatabaseContext();
            Nodes.Clear();
            context.TreeNode.Where(t => t.ParentId == null).Include(a =>
a.Children).Include(p => p.Parent)
                .Include(v => v.FileVersions).ToList().ForEach(c => Nodes.Add(c));
            treeView.Refresh();
        }

        private void selectLoad(object sender, RoutedPropertyChangedEventArgs<object> e)
        {
            var node = FileTree.SelectedItem as FileNode;
            using var context = new DatabaseContext();
            var fileVersions = context.FileVersions.Where(x => x.FileId ==
node.Id).ToList();
            if (fileVersions.Count == 0)
                return;

            versions.Clear();
            if (node.Type == NodeType.File)
            {
                versions.AddRange(fileVersions);
            }

            versionView.Refresh();
        }

        private void Add_Click(object sender, RoutedEventArgs e)
        {
            var wAdd = new Views.AddWindow();
            wAdd.Owner = this;
            if (!(wAdd.ShowDialog() ?? false))
            {
                return;
            }

            FileNode newNode;
            var parent = _getParentFromTree();

            if ((NodeType)wAdd.nType.SelectedIndex == NodeType.File)
            {
                newNode = FileManager.GetFile(wAdd.fName.Text);
                newNode.Name = wAdd.nName.Text;
                newNode.ParentId = parent?.Id;
            }
            else
            {
                newNode = new FileNode()
```

```
                {
                    FullName = FileManager.RootFolder + "\\" + wAdd.nName.Text,
                    Name = wAdd.nName.Text,
                    Type = (NodeType)wAdd.nType.SelectedIndex
                };
                if ((NodeType)wAdd.nType.SelectedIndex == NodeType.Directory)
                {
                    newNode.ParentId = parent?.Id;
                }
                else
                {
                    parent = null;
                }
            }

            using (var contex = new DatabaseContext())
            {
                StructureManager.AddNode(newNode);
                contex.SaveChanges();

                if (parent == null)
                    Nodes.Add(newNode);
                else
                {
                    if (parent.Children == null)
                        parent.Children = new ObservableCollection<FileNode>();
                    parent.Children.Add(newNode);
                }
            }

            treeView.Refresh();
        }

        private void Edit_Click(object sender, RoutedEventArgs e)
        {
            var item = FileTree.SelectedItem as FileNode;
            if (item == null)
                return;
            DocumentManager.ModificateDocument(item);
            selectLoad(null, null);
        }

        private void Delete_Click(object sender, RoutedEventArgs e)
        {
            var item = FileTree.SelectedItem as FileNode;
            if (item == null)
                return;
            if (System.Windows.MessageBox.Show($"Delete {item.Name}. This cannot be
undone, Are you sure?", "Question",
                    MessageBoxButton.YesNo, MessageBoxImage.Warning) ==
MessageBoxResult.No)
            {
                return;
            }
            else
            {
                using (var contex = new DatabaseContext())
                {
                    var rmNode = contex.TreeNode.FirstOrDefault(n => n.Id == item.Id);
                    if (rmNode == null)
                        return;

                    contex.TreeNode.Remove(rmNode);
                    contex.SaveChanges();
                }
```

```
            }

            MainViewModelOnOnTreeChange(null, null);
        }

        private void Security_Click(object sender, RoutedEventArgs e)
        {
            var wSecurity = new Views.SecurityWindow();
            wSecurity.Owner = this;
            wSecurity.ShowDialog();
        }

        private void Rules_Click(object sender, RoutedEventArgs e)
        {
            var wRule = new Views.RuleWindow();
            wRule.Owner = this;
            using (var context = new DatabaseContext())
            {
                wRule.Rules.AddRange(context.Rules.Include(x => x.Action).Include(c =>
c.Filter).ToList());
                if (wRule.ShowDialog() ?? false)
                {
                    Rules.Clear();
                    Rules.AddRange(wRule.Rules);
                    ruleView.Refresh();
                }
            }
        }

        private void Scan_Click(object sender, System.Windows.RoutedEventArgs e)
        {
            var scan = new Views.ScanWindow();
            using (var fbd = new System.Windows.Forms.FolderBrowserDialog())
            {
                var result = fbd.ShowDialog();

                if (result == System.Windows.Forms.DialogResult.OK &&
!string.IsNullOrWhiteSpace(fbd.SelectedPath))
                {
                    var nodes = new List<FileNode>();
                    FileManager.ScanFolder(ref nodes, new string[] { fbd.SelectedPath },
new List<string>() { ".pdf" });
                    scan.Nodes = nodes.Where(x => !Nodes.Any(c => c.Name ==
x.Name)).ToList();
                    if (scan.Nodes.Count == 0)
                    {
                        System.Windows.MessageBox.Show("No documents were found");
                        return;
                    }
                }
            }}

            scan.Owner = this;
            if (scan.ShowDialog() ?? false)
            {
                var item = FileTree.SelectedItem as FileNode;

                var parent = _getParentFromTree();
                foreach (var node in scan.Nodes)
                {
                    node.ParentId = parent.ParentId;
                    StructureManager.AddNode(node);
                }

                scan.Nodes.ForEach(x => parent.Children.Add(x));
```

```
    }
}

private FileNode _getParentFromTree()
{
    var item = FileTree.SelectedItem as FileNode;
    if (item != null)
    {
        switch (item.Type)
        {
            case NodeType.Category:
            case NodeType.Directory:
                return item;
            case NodeType.File:
                return item.Parent;
        }
    }

    return null;
}

private void restore_Click(object sender, RoutedEventArgs e)
{
    var node = FileTree.SelectedItem as FileNode;
    var selected = versionList.SelectedValue as FileVersion;
    if (node == null || selected == null)
        return;

    using var context = new DatabaseContext();

    var ver = context.FileVersions.FirstOrDefault(x => x.Id == selected.Id);
    if (ver == null)
        return;
    DocumentManager.RestoreDocument(node, ver);
    selectLoad(null, null);
}

private void ruleRun_Click(object sender, RoutedEventArgs e)
{
    var rule = rulesList.SelectedItem as Rule;
    if(rule == null)
        return;

    var result = StructureManager.ApplyRule(rule);
    MessageBox.Show($"Rule applied to {result} nodes");
    MainViewModelOnOnTreeChange(null, null);
}

private void createBackup_Click(object sender, RoutedEventArgs e)
{
    var sFileDlg = new SaveFileDialog();
    sFileDlg.Title = "Create Backup";
    sFileDlg.Filter = "Archive|*.zip";
    sFileDlg.ShowDialog();
    if(string.IsNullOrEmpty(sFileDlg.FileName))
        return;

    FileManager.CreateBackup(sFileDlg.FileName);
}

private void loadBackup_Click(object sender, RoutedEventArgs e)
{
    var openFileDialog = new OpenFileDialog();
    if (openFileDialog.ShowDialog() != true) return;
```

```
            FileManager.LoadBackup(openFileDialog.FileName);
            MainViewModelOnOnTreeChange(null, null);
        }
    }
}

namespace Docuorg.Converters
{
    public class NodeTypeConverter : IValueConverter
    {
        public object Convert(object value, Type targetType,
            object parameter, CultureInfo culture)
        {
            switch (value.ToString())
            {
                case "File":
                    return "/Images/file.png";
                case "Directory":
                    return "/Images/folder.png";
                case "Category":
                    return "/Images/category.png";
            }

            return "/Images/category.png";
        }

        public object ConvertBack(object value, Type targetType,
            object parameter, CultureInfo culture)
        {
            throw new NotImplementedException();
        }
    }
}
```