МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЖИТОМИРСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА
Фізико-математичний факультет
Кафедра комп'ютерних наук та інформаційних технологій

Реєстраційний №_____96_____

Дата реєстрації___28.11.2024__

РОЗРОБКА АВТОМАТИЗОВАНОЇ СИСТЕМИ КЕРУВАННЯ ВЕРСІЯМИ НАУКОВОГО ВИДАННЯ

Кваліфікаційна (дипломна) робота здобувача вищої освіти
другого (магістерського) рівня вищої освіти
спеціальності 122 Комп'ютерні науки
освітньої програми «Комп'ютерні науки»
26 Мд-Комп групи
Котенка Олександра Дмитровича

Науковий керівник:
доктор технічних наук, доцент,
професор кафедри комп'ютерних наук
та інформаційних технологій
Іванов Дмитро Євгенійович

Рекомендована до захисту
рішенням кафедри комп'ютерних наук та інформаційних технологій

Протокол № 9 від "13" листопада 2024 р.

Зав. кафедри _____ Олена УСАТА
(підпис) (Ім'я Прізвище)

Житомир – 2024

Дата захисту _____

Результат захисту

оцінка за національною шкалою	кількість балів за 100 бальною шкалою	ECTS

Голова ЕК _____ _____

(підпис) (Ім'я Прізвище)

Члени ЕК _____ _____

(підпис) (Ім'я Прізвище)

_____ _____

(підпис) (Ім'я Прізвище)

_____ _____

(підпис) (Ім'я Прізвище)

_____ _____

(підпис) (Ім'я Прізвище)

Секретар ЕК _____ _____

(підпис) (Ім'я Прізвище)

ЗМІСТ

ВСТУП

Сучасний науковий процес характеризується високим рівнем взаємодії між авторами, рецензентами та редакторами. Зважаючи на це, ефективна організація управління науковими статтями та їхніми версіями є критично важливою для забезпечення високої якості публікацій та узгодженості в роботі наукових видань. Проблема керування версіями статей, рецензування та контролю за редагуванням стає все більш актуальною в умовах швидкого зростання кількості наукових матеріалів та складності їх опрацювання.

Актуальність дослідження. Традиційні підходи до управління версіями наукових статей зазвичай базуються на ручному опрацюванні документів, що часто призводить до виникнення помилок та втрати важливої інформації. Відсутність автоматизованих засобів ускладнює контроль за змінами, унеможливлює швидке відстеження прогресу статті та сповільнює процес рецензування. Існуючі програмні рішення для керування науковими публікаціями зазвичай не враховують усіх аспектів взаємодії між авторами, рецензентами та редакторами, що обумовлює необхідність розробки нових, більш комплексних систем. Розробка автоматизованої системи керування версіями наукового видання дозволить зменшити часові та ресурсні витрати на управління статтями, підвищити ефективність рецензування та забезпечити зручність використання системи для всіх учасників наукового процесу.

Об'єкт дослідження – процес автоматизації керування науковими статтями, який включає створення, редагування, рецензування та зберігання версій наукових робіт.

Предмет дослідження – методи та засоби автоматизованого керування версіями наукових видань, що дозволяють оптимізувати роботу з науковими статтями та забезпечують зручний інтерфейс для взаємодії між авторами, рецензентами та редакторами.

Мета дослідження – розробка автоматизованої системи керування версіями наукових видань, яка забезпечить ефективний контроль над

процесом створення та редагування статей, а також дозволить підвищити якість взаємодії між учасниками редакційного процесу.

Завдання дослідження:

— Проаналізувати сучасні підходи та програмні засоби, що використовуються для керування науковими публікаціями.

— Розробити концептуальну модель автоматизованої системи, яка б враховувала специфіку взаємодії між авторами, рецензентами та редакторами.

— Реалізувати основні функціональні модулі системи, зокрема, керування версіями статей, рецензування та публікацію наукових матеріалів.

— Провести тестування розробленої системи на предмет функціональності, зручності використання та ефективності управління версіями статей.

Методологія дослідження базується на системному підході до розробки інформаційних систем, що включає етапи аналізу, проєктування, розробки, тестування та впровадження програмного продукту. Для досягнення поставлених завдань використано методи структурного аналізу та об’єктно-орієнтованого програмування, зокрема, мову програмування C# та платформу .NET, що забезпечує надійність, масштабованість та гнучкість системи.

Наукова новизна отриманих результатів полягає у створенні автоматизованої системи керування версіями наукових видань, яка включає комплексний підхід до зберігання та управління версіями наукових статей, враховуючи взаємодію між авторами та рецензентами, що забезпечує більш ефективний контроль за редагуванням та публікацією наукових матеріалів.

Теоретичне та практичне значення дослідження полягає у розробці та впровадженні методів керування версіями наукових статей, які можуть бути використані у подальших дослідженнях з автоматизації наукових видань, а також у реальних редакційних процесах для підвищення ефективності роботи наукових журналів.

Апробація результатів дослідження здійснювалася шляхом участі у Всеукраїнській науково-практичній конференції Інтернет-конференції

"Автоматизація та комп’ютерно-інтегровані технології у виробництві та освіті: стан, досягнення, перспективи розвитку" (м. Черкаси, 11-17 травня 2024 року). Результати роботи отримали позитивні відгуки від наукової спільноти, що підтверджує доцільність та значущість запропонованих підходів.

Кваліфікаційна робота складається із вступу, трьох розділів, висновків та списку використаних джерел. Загальний обсяг роботи становить 56 сторінок, включаючи 18 рисунків, а список використаних джерел налічує 39 найменувань.

РОЗДІЛ 1. ОГЛЯД СУЧАСНИХ СИСТЕМ КЕРУВАННЯ ВЕРСІЯМИ

1.1. Аналіз існуючих систем керування версіями

Системи керування версіями (СКВ) є важливою складовою частиною сучасного процесу розробки програмного забезпечення та наукової діяльності, де контроль за змінами документів має вирішальне значення. Основне завдання СКВ полягає в зберіганні різних версій файлів, контролі над змінами, які вносяться, та забезпеченні можливості спільної роботи над проєктами. Існує широкий спектр систем керування версіями, які можуть бути класифіковані на централізовані та розподілені.

Централізовані системи керування версіями. У таких системах існує єдине центральне сховище, де зберігаються всі версії файлів та історія змін. Прикладом централізованих СКВ є Subversion (SVN) та CVS (Concurrent Versions System).

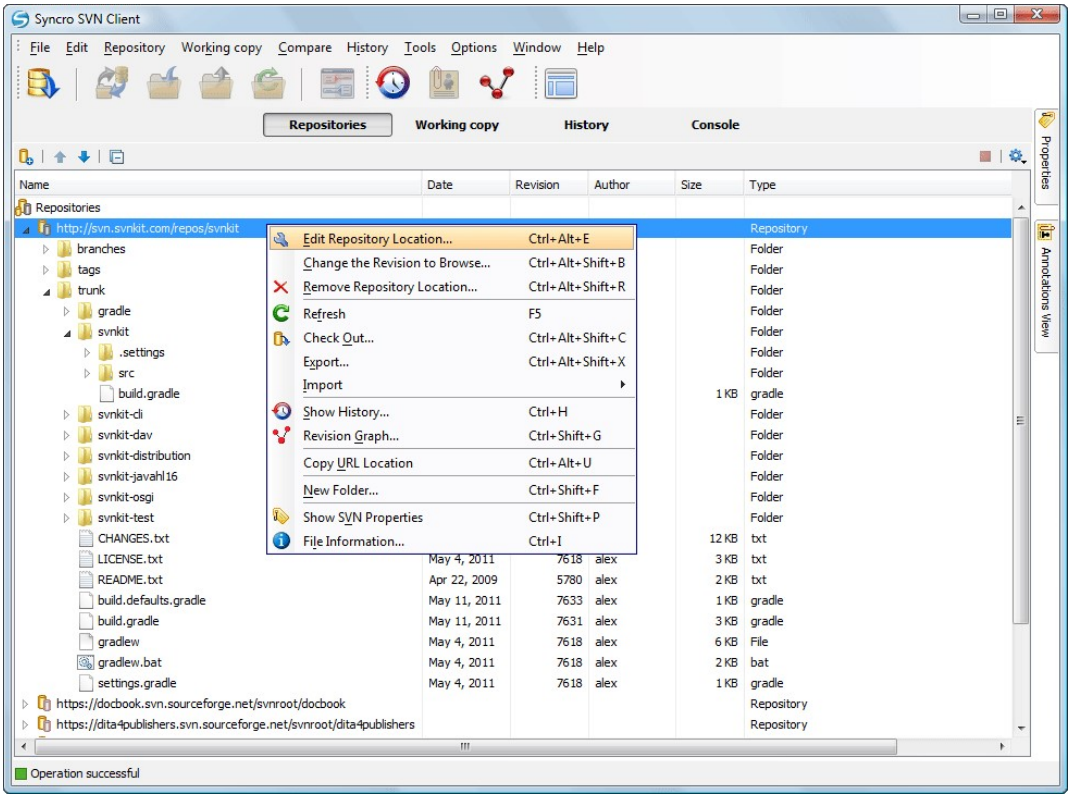


Рис. 1.1 - Subversion (SVN) Client

Subversion (SVN) — це централізована система керування версіями, розроблена для зберігання та контролю змін у проєктах. SVN дозволяє користувачам зберігати історію змін, здійснювати об'єднання та створювати гілки проектів.

SVN має інтуїтивно зрозумілий інтерфейс, що полегшує навчання нових користувачів. Система дозволяє налаштовувати права доступу для різних користувачів, що є важливим для командної роботи. SVN добре підходить для зберігання великих бінарних файлів, оскільки може обробляти їх без значних затримок.

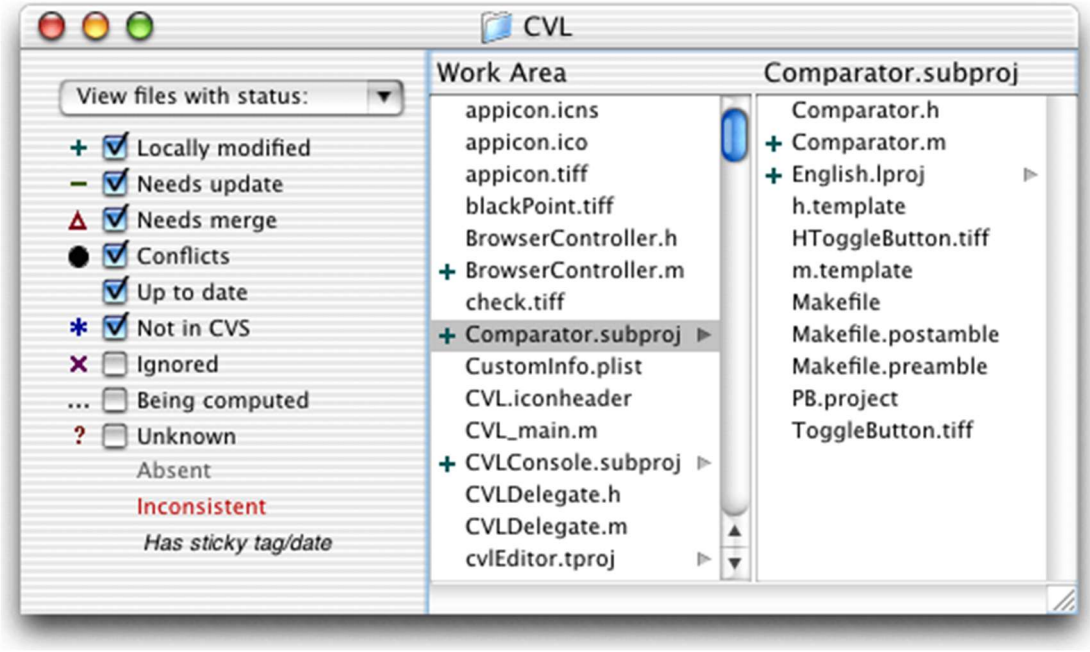


Рис. 1.2 - Concurrent Versions System

Concurrent Versions System (CVS) є однією з перших систем керування версіями, що забезпечує можливість спільної роботи над проектами. CVS є централізованою системою, яка дозволяє кільком користувачам працювати з одними й тими ж файлами одночасно, зберігаючи історію змін і дозволяючи злиття внесених змін.

CVS підтримує одночасний доступ кількох користувачів до одного проекту, що дозволяє легко об'єднувати зміни. Також система зберігає історію змін, що дозволяє відслідковувати зміни та відновлювати попередні версії, а

інтерфейс даної системи є відносно простим і зрозумілим, що полегшує навчання нових користувачів.

Хоч CVS і був одним із перших інструментів для керування версіями, має свої обмеження, особливо в порівнянні з сучасними системами, такими як Git і Mercurial. Однак він все ще використовується в деяких проектах завдяки своїй простоті та можливостям спільної роботи. Вибір CVS як системи керування версіями може бути виправданим для малих проектів або для команд, які вже мають досвід роботи з цією системою.

Однак він має також і недоліки. По перше всі користувачі повинні бути підключені до сервера для виконання операцій, що може обмежити роботу в автономному режимі. Хоча гілки можливі, процес їх створення та об'єднання може бути складнішим у порівнянні з розподіленими системами.

Основною перевагою цих систем є простота архітектури та відносна легкість у використанні. Однак централізовані СКВ мають певні недоліки, серед яких можна відзначити низьку стійкість до збоїв і необхідність постійного доступу до сервера для роботи з проєктами. Втрата доступу до центрального сервера може призвести до неможливості продовжувати роботу.

Розподілені системи керування версіями. На відміну від централізованих систем, розподілені СКВ зберігають повну копію історії змін проєкту на кожному клієнтському комп'ютері. Це означає, що розробники можуть працювати над проєктом навіть без підключення до центрального сховища, а синхронізація змін відбувається під час спільної роботи. Найпоширенішими розподіленими СКВ є Git та Mercurial.

Git, створений Лінусом Торвальдсом, є найпопулярнішою системою керування версіями, яка використовується як в розробці програмного забезпечення, так і в науковій діяльності. Основною перевагою Git є гнучкість у роботі з гілками та легкість у створенні альтернативних версій проєкту, які можуть бути об'єднані без втрати даних. Крім того, Git надає можливість паралельної роботи великої кількості розробників над різними аспектами проєкту, що робить його незамінним інструментом у командних проєктах.

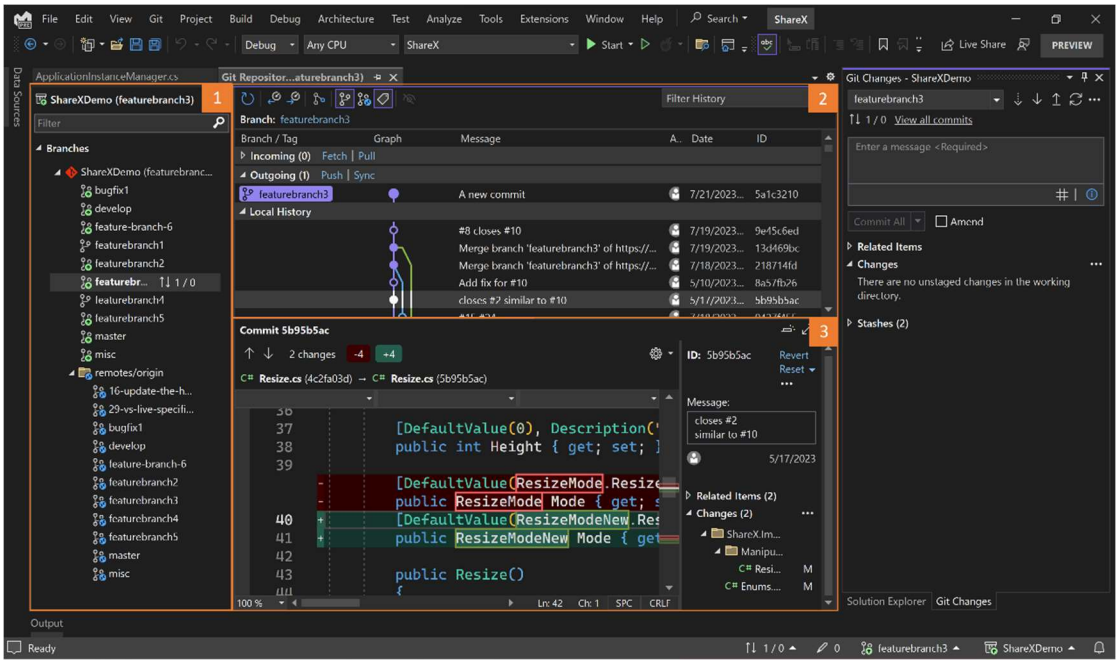


Рис. 1.3 - Git

Mercurial, у свою чергу, також є потужним інструментом для керування версіями. Основною перевагою цієї системи є простота у використанні та вища швидкість виконання деяких операцій порівняно з Git. Mercurial має інтуїтивний інтерфейс, що робить його придатним для невеликих проєктів, де простота та надійність мають першочергове значення.

1.2. Проблеми використання стандартних систем версій для наукових публікацій

Системи керування версіями, такі як Git, SVN та CVS, традиційно розроблялися для програмного забезпечення і мають обмеження у застосуванні для керування науковими публікаціями. Наукові статті мають свою специфіку, пов’язану з частими змінами тексту, рецензуванням, багаторівневим погодженням та публікацією. Стандартні системи керування версіями не завжди здатні ефективно задовольняти ці потреби, що створює ряд проблем при їхньому використанні у науковому середовищі.

Більшість систем керування версіями розроблені для роботи з вихідним кодом програм, який складається з простих текстових файлів. Однак, наукові публікації часто оформлюються у форматах, які містять складні структурні елементи (формули, таблиці, діаграми) і потребують специфічних інструментів для зручної роботи з ними. Це ускладнює порівняння різних версій документів, а також пошук відмінностей у великих текстових файлах.

Стандартні системи керування версіями не мають вбудованих механізмів для рецензування наукових робіт. Вони не підтримують функціонал надання рецензій, коментарів та відгуків у зручній для користувача формі. Внаслідок цього процес рецензування доводиться виконувати за допомогою сторонніх інструментів, що ускладнює керування відгуками та внесення змін на основі рецензій.

У наукових публікаціях важливу роль відіграє надання різного рівня доступу для авторів, рецензентів і редакторів. У стандартних системах керування версіями, таких як Git або SVN, права доступу часто обмежуються загальними ролями, що створює проблеми при необхідності делегування доступу до окремих частин документа або забезпечення конфіденційності процесу рецензування.

Наукові публікації потребують не тільки керування версіями, але й зберігання та обробки метаданих (автори, ключові слова, тематика, дата подання та затвердження, статус рецензування тощо). Стандартні системи керування версіями не передбачають таких функцій, що значно ускладнює ведення архіву публікацій та роботу з ними.

Наукові статті часто готуються в форматах PDF, LaTeX або DOCX. Стандартні системи керування версіями, такі як Git, не мають зручних інструментів для роботи з цими форматами, що ускладнює контроль версій таких документів. Відсутність візуального порівняння змін у форматах PDF чи DOCX створює додаткові труднощі при аналізі внесених правок.

Системи керування версіями не мають повної інтеграції з інструментами для обробки тексту, такими як MS Word, що є основним засобом для

підготовки наукових статей у багатьох установах. Це створює бар'єри для авторів, яким доводиться переходити між різними середовищами для створення та відстеження змін у публікаціях.

У процесі рецензування та публікації наукових статей необхідно мати інструменти для електронного підпису та затвердження статей. Стандартні системи керування версіями не передбачають таких механізмів, що знижує їхню ефективність у середовищі наукових публікацій.

Для ефективного керування науковими публікаціями стандартні системи керування версіями мають ряд обмежень, пов’язаних із їх первинним призначенням для керування вихідним кодом програмного забезпечення. Відсутність спеціалізованих функцій для рецензування, роботи з текстовими документами складної структури та ведення архіву публікацій з метаданими обумовлює необхідність розробки автоматизованої системи керування версіями, що буде враховувати специфіку наукових досліджень та забезпечувати ефективну взаємодію між авторами, рецензентами та редакторами.

Висновки до розділу 1

У першому розділі проведено аналіз існуючих систем керування версіями, таких як Git, Subversion (SVN), Concurrent Versions System (CVS) та інших, з точки зору їхньої придатності для використання у наукових дослідженнях. Було виявлено, що хоча дані системи є ефективними для керування версіями програмного коду, вони мають ряд суттєвих обмежень при застосуванні для наукових публікацій.

Проблеми використання стандартних систем керування версіями для наукових публікацій полягають у наступному:

— Обмежені можливості роботи з текстовими документами складної структури, що ускладнює процес порівняння та злиття змін у наукових статтях.

— Відсутність механізмів для багаторівневого рецензування, що не дозволяє зручно враховувати та аналізувати коментарі рецензентів та редакторів.

— Проблеми з централізованим керуванням доступом до різних частин документів, що не задовольняє вимоги конфіденційності при рецензуванні та редагуванні.

— Відсутність можливостей для ведення електронного архіву публікацій, зберігання метаданих, а також автоматизації процесу затвердження та публікації статей.

— Недостатня підтримка форматів документів, таких як PDF, DOCX та LaTeX, що ускладнює контроль версій для даних типів файлів.

Таким чином, для ефективного використання систем керування версіями у науковій діяльності необхідне створення спеціалізованих автоматизованих систем, які враховуватимуть особливості роботи з науковими публікаціями. Це передбачає розробку інструментів для інтеграції процесу рецензування, автоматизації затвердження документів, забезпечення зручного керування доступом та підтримку різних форматів файлів. Розробка таких систем дозволить підвищити ефективність наукових досліджень, зменшити кількість помилок у процесі редагування та оптимізувати роботу над колективними науковими проектами.

РОЗДІЛ 2. ПРОЕКТУВАННЯ СИСТЕМИ КЕРУВАННЯ ВЕРСІЯМИ ДЛЯ НАУКОВОГО ВИДАННЯ

2.1. Вимоги до системи

При розробці автоматизованої системи керування версіями наукового видання було сформульовано ключові функціональні та нефункціональні вимоги до програмного забезпечення. Ці вимоги враховують як функції, які система повинна виконувати, так і обмеження та параметри її продуктивності, надійності та безпеки.

З огляду на основну мету системи – забезпечення керування версіями публікацій та їх ефективної обробки, необхідно передбачити підтримку процесу автентифікації та авторизації користувачів. Для цього використовується система реєстрації та входу в акаунт за допомогою логіна й пароля. Додатково, для підвищення безпеки даних користувачів, було інтегровано компонент ASP.NET Core Identity, що надає можливість здійснювати не лише базову автентифікацію, але й керування ролями користувачів. Ця функція дозволяє налаштувати доступ до певних функцій системи в залежності від ролі користувача, наприклад, адміністратора, автора чи редактора.

Однією з основних функцій системи є керування публікаціями. В рамках цього процесу користувачі повинні мати можливість створювати, редагувати, переглядати та видаляти публікації. Для кожної публікації повинна зберігатися інформація про її заголовок, статус, автора та інші важливі метадані. Це дозволяє відстежувати етапи публікаційного процесу і сприяє прозорості роботи. Структуру даних публікацій реалізовано через модель Publication, що містить такі ключові поля, як Title, Status, AuthorId та інші.

Зручність управління публікаціями забезпечується через інтерфейс користувача, де передбачено можливість фільтрації публікацій за статусом або автором. Зокрема, розроблено окрему сторінку, яка дозволяє відображати всі

публікації з можливістю їх подальшого редагування безпосередньо через інтерфейс.

Система повинна використовувати реляційну базу даних для зберігання інформації про користувачів та публікації. Для цієї мети було обрано Entity Framework Core як інструмент для роботи з базою даних. Це рішення дозволяє ефективно управляти запитами до бази даних за допомогою моделей, що спрощує процеси створення та модифікації даних.

Крім функціональних вимог, визначено ряд нефункціональних аспектів, що впливають на стабільність та надійність системи. Програма має бути надійною, тобто гарантувати збереження даних навіть у випадку аварійних ситуацій. Для цього необхідно передбачити механізми резервного копіювання бази даних. Продуктивність системи також є важливим критерієм, і час відгуку на основні запити користувача не повинен перевищувати трьох секунд під звичайним навантаженням.

Щодо масштабованості, система повинна підтримувати роботу з великою кількістю користувачів і публікацій. Використання технологій SQL Server та Entity Framework Core дозволяє масштабувати систему у разі зростання обсягу даних або кількості користувачів без втрати продуктивності.

Безпека даних є ключовим аспектом розробки. Використання ASP.NET Core Identity гарантує захист паролів та іншої особистої інформації користувачів за допомогою шифрування й токенізації. Це мінімізує ризики несанкціонованого доступу до системи або втрати конфіденційних даних.

Для забезпечення зручності використання передбачено інтуїтивно зрозумілий інтерфейс користувача. Важливо, щоб інтерфейс був доступний як для досвідчених користувачів, так і для тих, хто не має спеціальних технічних навичок. Всі функції системи доступні через веб-інтерфейс, який також адаптовано для роботи на різних пристроях, що підвищує доступність системи.

Важливим критерієм є продуктивність та стійкість системи, тобто здатність функціонувати в умовах підвищеного навантаження або часткової

несправності компонентів. Це забезпечує безперебійну роботу системи та гарантує її стабільність навіть у критичних ситуаціях.

Крім цього, система повинна легко підтримуватись та супроводжуватись. Документованість коду дозволяє забезпечити простоту її підтримки та подальшого розвитку. Оскільки розробка базується на C#/.NET, система легко інтегрується з іншими продуктами Microsoft, що спрощує її розширення у майбутньому.

Таким чином, реалізація всіх зазначених вимог гарантує, що система керування версіями публікацій буде відповідати сучасним стандартам продуктивності, безпеки та зручності, а також ефективно виконуватиме свої основні функції.

2.2. Архітектура та проектування системи

Архітектура системи побудована на основі багаторівневого підходу, який забезпечує модульність, розподіл обов'язків між компонентами та полегшує підтримку і масштабування. Основною метою є створення ефективної системи для керування версіями наукових публікацій з чітким розподілом ролей користувачів та безпечною обробкою даних.

Система складається з трьох основних рівнів: рівня даних, логічного рівня та рівня представлення. Рівень даних відповідає за взаємодію з базою даних MS SQL, яка містить інформацію про користувачів, публікації та їхні версії. Для роботи з базою даних використовується ORM Entity Framework Core, що спрощує маніпуляції з даними і дозволяє уникнути прямого написання SQL-запитів, що підвищує безпеку і зручність роботи з даними.

Логічний рівень системи реалізує бізнес-правила та відповідає за обробку запитів користувачів. На цьому рівні відбувається перевірка прав доступу, управління публікаціями, контроль за версіями, рецензування та керування ролями користувачів. Логічний рівень взаємодіє з базою даних через Data Access Layer і передає результат на рівень представлення.

Рівень представлення реалізований за допомогою ASP.NET Core Razor Pages, що дозволяє створювати динамічні веб-сторінки для взаємодії з користувачами. Інтерфейс системи забезпечує доступ до таких функцій, як реєстрація, авторизація, керування публікаціями та їхніми версіями. Взаємодія користувача з системою відбувається через сторінки, які підготовлені для кожної конкретної функції, наприклад, для управління публікаціями або рецензування.

Взаємодія між усіма рівнями відбувається через чітко визначені інтерфейси. Кожен рівень виконує свою роль незалежно від інших, що дозволяє легко підтримувати та модифікувати систему. Рівень представлення працює лише з логічним рівнем і не взаємодіє безпосередньо з базою даних, що підвищує безпеку і стабільність системи. Логічний рівень обробляє запити користувачів, виконує необхідні операції з даними і передає результати на рівень представлення для відображення.

Архітектура системи має низку важливих переваг, серед яких модульність, гнучкість та підвищена безпека. Розділення функціональності між рівнями дозволяє легко змінювати або додавати нові компоненти без значних змін в інших частинах системи. Це забезпечує надійну основу для подальшого розвитку та розширення функціональних можливостей.

Архітектурний стиль системи базується на використанні принципів багаторівневої архітектури та архітектурного шаблону MVC (Model-View-Controller) із застосуванням ASP.NET Core. Цей підхід дозволяє чітко розмежовувати функціональні частини системи, забезпечуючи масштабованість, легкість у підтримці та можливість розширення.

Основний акцент зроблено на розподіл обов'язків між рівнями: модель відповідає за управління даними та взаємодію з базою даних, представлення — за відображення інформації та взаємодію з користувачем, а контролер — за обробку запитів, виконання бізнес-логіки та обмін інформацією між моделлю і представленням.

Архітектурний стиль системи також використовує шаблон Repository для організації взаємодії з базою даних. Це дає змогу логічному рівню системи не залежати від конкретних деталей реалізації зберігання даних. Замість прямого доступу до бази даних використовується набір інтерфейсів, що дозволяє легко змінювати або замінювати методи доступу до даних.

Окрім того, система використовує залежності через інверсію контролю (IoC) за допомогою вбудованого механізму Dependency Injection в ASP.NET Core. Це дозволяє спростити управління залежностями між класами та зменшити жорстке з'єднання між компонентами системи, що робить код більш гнучким і тестованим.

Застосування архітектурного стилю MVC у поєднанні з іншими шаблонами, такими як Repository і IoC, забезпечує не лише високий рівень організації системи, але й підтримує принципи розробки з урахуванням тестування (TDD) і розширюваності.

Архітектурний стиль системи побудований на принципах модульності та чіткого розподілу відповідальностей між компонентами. Основним вибором для реалізації архітектури є багаторівнева структура, яка забезпечує гнучкість, легкість у підтримці та масштабуванні.

В основі архітектурного стилю лежить концепція розподілу на три основні рівні: рівень даних, бізнес-логіки та представлення. Рівень даних відповідає за взаємодію з базою даних, в даному випадку — MS SQL. Для спрощення роботи з даними та уникнення прямого використання SQL-запитів застосовується ORM (Object-Relational Mapping) Entity Framework Core, що дозволяє розробникам працювати з об'єктами замість рядків запитів.

Бізнес-логіка реалізується на другому рівні, де обробляються запити користувачів, виконується перевірка прав доступу, керування публікаціями та контроль версій. Цей рівень працює як проміжна ланка між даними і представленням, забезпечуючи логіку обробки інформації.

Рівень представлення реалізується за допомогою ASP.NET Core Razor Pages. Це забезпечує динамічний і інтерактивний інтерфейс, який дозволяє

користувачам взаємодіяти з системою. На цьому рівні користувач отримує доступ до функціоналу системи, такого як реєстрація, авторизація, управління публікаціями та їх рецензування.

Для забезпечення чистоти архітектури і зменшення залежностей між компонентами використовуються шаблони проектування, такі як Repository для управління доступом до даних та Dependency Injection для інверсії контролю. Це дозволяє легко змінювати реалізації окремих компонентів без істотного впливу на всю систему.

Таким чином, архітектурний стиль системи підтримує принципи модульності, гнучкості та простоти у використанні. Це дозволяє реалізувати ефективну автоматизовану систему для керування версіями наукових публікацій, яка може легко адаптуватися до змін і розширення вимог користувачів.

База даних є ключовим елементом архітектури автоматизованої системи управління версіями наукового видання, оскільки вона забезпечує зберігання, обробку та організацію інформації, необхідної для функціонування системи. У нашій реалізації використовується реляційна база даних MS SQL, яка характеризується високою продуктивністю, надійністю та можливістю масштабування.

Структура бази даних спроектована з урахуванням принципів нормалізації. Це дозволяє зменшити надлишковість даних і покращити їх цілісність. Основні сутності бази даних включають користувачів, публікації, рецензії та ролі. Користувачі зберігають інформацію про всіх учасників системи, таких як адміністратори, автори та рецензенти. Кожен користувач має унікальний ідентифікатор, ім’я, електронну адресу, пароль та роль, що визначає його доступ до різних функцій системи.

Публікації містять дані про наукові роботи, такі як заголовок, опис, статус (наприклад, на розгляді або опубліковано) та дату створення. Важливою частиною є зв'язок кожної публікації з автором, що дозволяє вести облік внеску кожного автора у наукові дослідження. Рецензії

використовуються для зберігання інформації про оцінювання публікацій рецензентами, включаючи дані про оцінку, коментарі рецензента та статус рецензії. Цей механізм допомагає в управлінні процесом рецензування, що є критично важливим етапом у підготовці наукових статей.

Ролі визначають типи користувачів системи і їх права доступу, що дозволяє реалізувати механізм авторизації. Цей підхід забезпечує контроль над тим, хто і які дії може виконувати в системі, що особливо важливо для захисту чутливої інформації.

Взаємозв'язки між цими сутностями формуються через зовнішні ключі, що забезпечує цілісність даних і їх коректну інтеграцію. Наприклад, зв'язок між таблицями "Користувачі" та "Публікації" дозволяє визначити, які публікації належать конкретному автору.

Для спрощення роботи з базою даних у проекті використовується Entity Framework Core, що є ORM (Object-Relational Mapping) рішенням, яке дозволяє взаємодіяти з базою даних за допомогою об'єктно-орієнтованого підходу. Це значно спрощує реалізацію операцій CRUD (створення, читання, оновлення, видалення) і дозволяє розробникам зосередитися на бізнес-логіці програми, зменшуючи потребу у написанні складних SQL-запитів.

Завдяки використанню MS SQL та добре спроектованої структури бази даних, система забезпечує швидкий доступ до даних, їх надійне зберігання та легкість у підтримці. Це є критично важливими аспектами для функціонування автоматизованої системи управління версіями наукових публікацій, що сприяє підвищенню ефективності процесу наукової діяльності.

API (Application Programming Interface) є важливою складовою автоматизованої системи управління версіями наукового видання, оскільки він забезпечує інтерактивний зв’язок між різними компонентами системи, а також можливість інтеграції з зовнішніми сервісами. Використання API дозволяє створити гнучку архітектуру, що сприяє масштабуванню системи та спрощує її обслуговування.

У даній системі реалізовано функціонал, що дозволяє виконувати основні операції CRUD (створення, читання, оновлення, видалення) над ресурсами, такими як користувачі, публікації та рецензії. Систему спроектовано таким чином, щоб вона була інтуїтивно зрозумілою та зручною у використанні, що дозволяє розробникам легко інтегрувати її з іншими системами чи сервісами.

Для забезпечення безпеки даних, доступ до системи реалізується через механізм аутентифікації та авторизації. Використання токенів доступу, згенерованих під час аутентифікації, гарантує, що тільки авторизовані користувачі можуть виконувати дії з ресурсами системи. Це важливо для захисту чутливих даних, що зберігаються в системі, а також для підтримки вимог щодо конфіденційності.

Крім того, система може інтегруватися з зовнішніми сервісами для виконання специфічних завдань. Наприклад, інтеграція з системами електронної пошти дозволяє автоматично надсилати сповіщення про нові публікації або зміни статусу рецензій. Інтеграція з платіжними системами може бути використана для автоматизації процесів оплати за рецензійні послуги або доступ до платного контенту.

Ще однією важливою інтеграцією може бути використання сервісів аналітики для збору даних про взаємодію користувачів з системою. Це дозволяє розробникам отримувати цінну інформацію про поведінку користувачів, що може бути використано для покращення функціональності та загального користувацького досвіду.

Завдяки використанню API та інтеграцій, автоматизована система управління версіями наукового видання отримує можливість бути більш адаптивною та реагувати на зміни у вимогах користувачів, що сприяє підвищенню ефективності наукової діяльності та оптимізації процесів публікації.

Безпека системи є одним з найважливіших аспектів проектування автоматизованої системи управління версіями наукового видання. Для цього в

системі реалізовано ASP.NET Core Identity, який забезпечує комплексні механізми автентифікації та авторизації користувачів. Це дозволяє гарантувати, що лише авторизовані користувачі мають доступ до певних функцій та ресурсів системи.

Одним із ключових компонентів ASP.NET Core Identity є шифрування паролів. При реєстрації нового користувача його пароль не зберігається у відкритому вигляді. Замість цього система використовує сучасні алгоритми хешування, що забезпечують високу стійкість до атак, таких як Brute-force. Це значно підвищує безпеку користувацьких даних, оскільки навіть у разі витоку бази даних паролі залишаються недоступними для зловмисників.

Крім шифрування паролів, система реалізує механізм двофакторної автентифікації (2FA). Цей метод вимагає, щоб користувач при вході в систему надавав не лише свій пароль, але й додатковий код, який генерується на його мобільному пристрої або надісланий йому через SMS чи електронну пошту. Це забезпечує додатковий рівень захисту, ускладнюючи доступ до облікових записів навіть у випадку, якщо зловмисник отримав пароль.

Крім того, в системі передбачені різні рівні доступу для різних ролей користувачів, таких як адміністратори, автори та рецензенти. Це означає, що кожна роль має свої права та можливості, що дозволяє здійснювати детальне управління доступом до різних функцій системи. Наприклад, адміністратори мають доступ до всіх даних та функцій управління, тоді як автори можуть керувати лише своїми публікаціями, а рецензенти — лише публікаціями, призначеними для оцінки.

Цей розподіл прав доступу є критично важливим для підтримки безпеки та конфіденційності даних. Він запобігає можливості несанкціонованого доступу до чутливих даних, таких як особисті дані авторів, інформація про публікації та рецензії. Завдяки такому підходу, система може забезпечити надійний захист інформації та підтримувати довіру користувачів.

У процесі розробки системи також враховано можливість моніторингу та аудиту дій користувачів. Це може бути реалізовано через ведення журналів

(логів), які фіксують всі важливі події, пов’язані з доступом до системи. Такі журнали можуть бути використані для аналізу активності користувачів, виявлення підозрілих дій і вжиття заходів у разі необхідності.

Таким чином, реалізація механізмів безпеки у системі управління версіями наукового видання на базі ASP.NET Core Identity забезпечує високий рівень захисту даних і користувачів, знижує ризик несанкціонованого доступу та сприяє створенню надійного середовища для наукової діяльності.

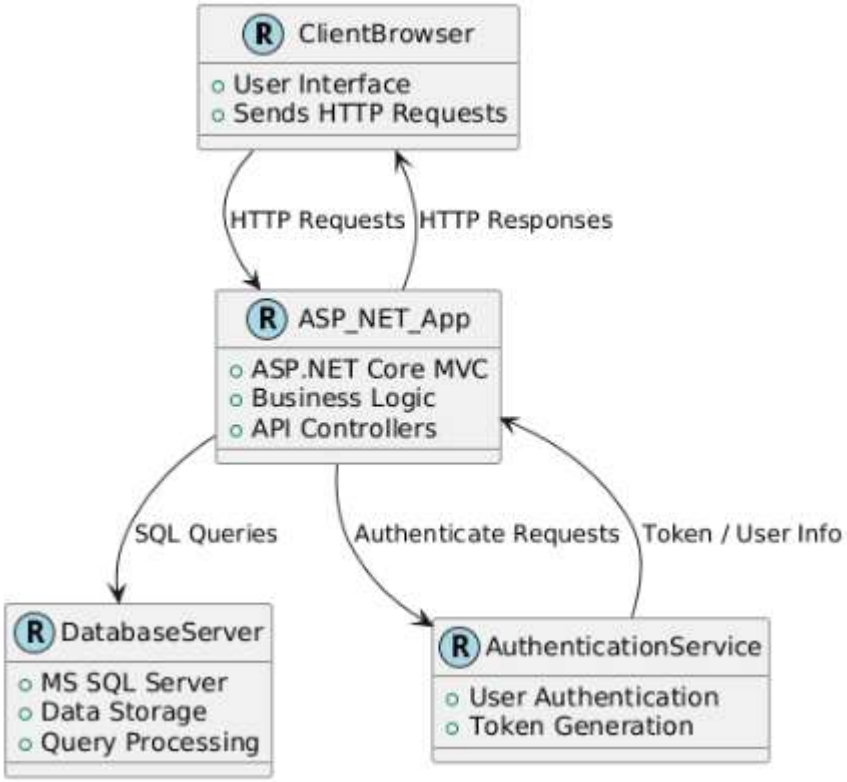


Рис. 2.1 – Загальна архітектура системи

У системі автоматизованої керування версіями наукового видання реалізовано багаторівневу архітектуру, що складається з трьох основних компонентів: веб-застосунку, сервера аутентифікації та бази даних. Ця архітектура забезпечує розподіл обов'язків між різними елементами, що, в свою чергу, сприяє підвищенню продуктивності, масштабованості та зручності обслуговування.

Веб-застосунок є центральною частиною архітектури системи. Він включає в себе три ключові компоненти: контролери, представлення та моделі.

Контролери виконують роль посередника між користувачем і системою, обробляючи запити та викликаючи відповідні дії. Основні функції контролерів охоплюють реєстрацію користувачів, аутентифікацію, управління статтями та оцінку рецензій. Вони забезпечують можливість створення нових облікових записів, а також входу до системи через механізми аутентифікації, які гарантують безпеку доступу.

Представлення відповідають за візуальне оформлення та взаємодію з користувачами, відображаючи інформацію у зручному для сприйняття вигляді. Цей компонент забезпечує інтуїтивно зрозумілий інтерфейс, що спрощує взаємодію користувачів з системою. Моделі визначають структуру даних, що використовуються у системі, та взаємодіють із базою даних для отримання і збереження інформації, що робить її доступною для користувачів та адміністративних ролей.

Сервер аутентифікації (Identity Server) є важливим елементом архітектури, оскільки забезпечує механізми для аутентифікації та авторизації користувачів. Він містить два основних компоненти: аутентифікацію та авторизацію. Аутентифікація включає методи для входу та виходу користувачів із системи, що дозволяє користувачам безпечно аутентифікуватися, використовуючи паролі, які зберігаються у зашифрованому вигляді. Авторизація, у свою чергу, управляє доступом користувачів до різних функцій системи на основі їхніх ролей, що гарантує, що лише уповноважені користувачі можуть виконувати певні дії, такі як публікація статей чи управління ролями.

База даних MS SQL є ключовою складовою архітектури, відповідальною за зберігання та управління всіма даними системи. Вона включає кілька таблиць, які виконують різні функції. Таблиця "Користувачі" зберігає дані про всіх користувачів, включаючи ідентифікатори, імена, адреси електронної пошти та хеші паролів, що дозволяє системі управляти доступом до функціоналу в залежності від ролей користувачів. Таблиця "Статті" містить інформацію про наукові статті, такі як заголовок, вміст, ідентифікатор автора

та статус публікації, що дає можливість відстежувати стан статей на різних етапах, від написання до публікації.

Таблиця "Рецензії" містить інформацію про рецензії, надані рецензентами, включаючи коментарі та рекомендації, що дозволяє зберігати історію рецензування та враховувати пропозиції рецензентів. Таблиця "Публікації" зберігає дані про офіційно опубліковані статті, включаючи дату публікації та ідентифікатор статті, що важливо для створення архіву публікацій. Також є таблиця "Історія", яка фіксує всі зміни, внесені до статей, включаючи версії, що дозволяє авторам повертатися до попередніх версій у разі потреби.

Взаємодія між компонентами системи відбувається через чітко визначені зв'язки. Веб-застосунок взаємодіє з контролерами для обробки запитів користувачів, а контролери, у свою чергу, здійснюють запити до бази даних для отримання або збереження даних. Сервер аутентифікації відповідає за перевірку користувачів та управління доступом до ресурсів системи. Цей підхід забезпечує узгодженість у роботі всіх компонентів і сприяє безпеці, ефективності та гнучкості системи.

Таким чином, архітектура системи автоматизованої керування версіями наукового видання відповідає сучасним вимогам до програмного забезпечення, забезпечуючи високу продуктивність, зручність використання та масштабованість.

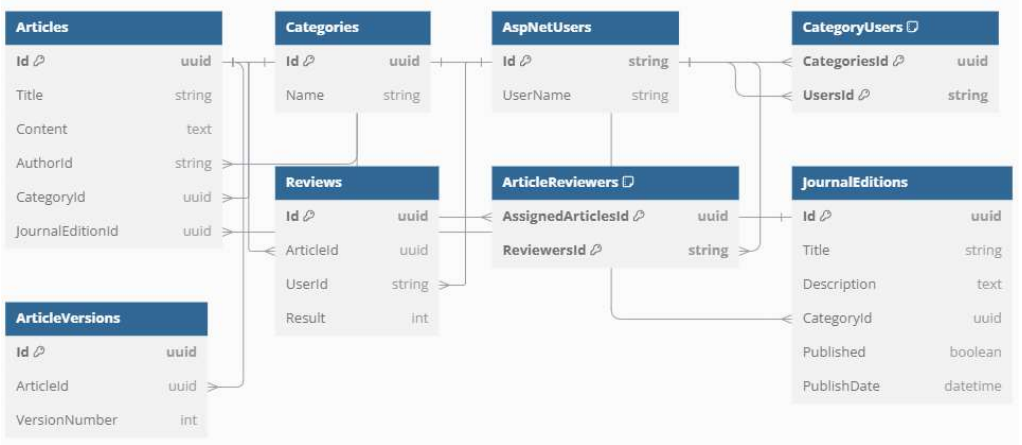


Рис. 2.2 – База даних

Таблиця Articles містить інформацію про статті, які створюють автори. У ній зберігаються такі дані, як заголовок статті, її зміст, ідентифікатор автора, що вказує на зв’язок із користувачем з таблиці AspNetUsers, а також категорія статті, яка зв’язана з таблицею Categories. Додатково стаття може бути прив’язана до випуску журналу, інформація про який зберігається у таблиці JournalEditions.

У таблиці Categories зберігаються всі категорії, до яких можуть належати статті. Кожна категорія має свій унікальний ідентифікатор і назву. Це дозволяє структурувати статті за тематикою та полегшує їх пошук і класифікацію.

Таблиця AspNetUsers використовується для зберігання інформації про користувачів системи. Кожен користувач має унікальний ідентифікатор, який слугує для ідентифікації у взаємодії з іншими таблицями, наприклад, коли потрібно вказати автора статті або рецензента.

CategoryUsers – це зв’язуюча таблиця, яка дозволяє створювати асоціації між користувачами і категоріями. Це означає, що користувач може бути пов’язаний з кількома категоріями, наприклад, для перегляду чи роботи над статтями певного тематичного спрямування.

Для кожної статті можна зберігати кілька версій у таблиці ArticleVersions. Тут записуються дані про номер версії статті та її належність до конкретної статті в таблиці Articles. Це дозволяє відслідковувати зміни та еволюцію статті з часом.

Рецензії на статті зберігаються у таблиці Reviews. Кожна рецензія прив’язана до певної статті, а також до рецензента, який є користувачем з таблиці AspNetUsers. Результат рецензії записується у відповідному полі, що дозволяє оцінити якість статті чи її відповідність вимогам.

Ця таблиця використовується для призначення рецензентів до конкретних статей. Вона створює зв’язки між статтями з таблиці Articles та рецензентами з таблиці AspNetUsers, забезпечуючи гнучке керування процесом рецензування.

У таблиці JournalEditions зберігаються дані про випуски журналу, до яких можуть входити статті. Кожен випуск має назву, опис, статус публікації та дату публікації. Також випуск може бути прив’язаний до певної категорії, що полегшує його тематичну класифікацію.

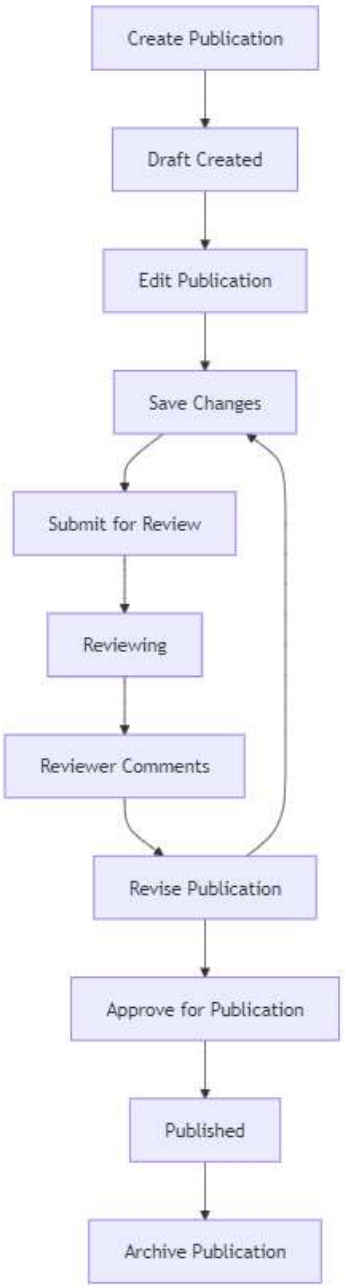


Рис. 2.3 – Процес керування публікаціями

На етапі рецензування статті, коли автор вважає, що публікація готова до оцінювання, він може подати її на рецензування. Статус статті змінюється на "На розгляді", і рецензенти отримують доступ до неї для оцінювання. Під

час рецензування рецензенти можуть залишати коментарі та рекомендації щодо внесення змін. Якщо автор враховує ці зауваження, він має можливість редагувати статтю, що призводить до створення нової версії документа.

Після завершення всіх змін та врахування зауважень рецензентів стаття може бути затверджена для публікації. На цьому етапі система автоматично змінює статус публікації на "Опубліковано", і стаття стає доступною для користувачів у журналі наукових видань.

Діаграма процесів управління публікаціями також може включати етапи, пов’язані з автоматичним повідомленням усіх учасників процесу про зміни статусу статті, а також ведення журналу змін, що забезпечує документування всіх дій, пов'язаних з публікацією.

Загалом, діаграма процесів роботи з публікаціями ілюструє логіку та послідовність дій, які виконуються в системі, що дозволяє зрозуміти, як ефективно організувати управління науковими статтями. Це сприяє більш чіткому розумінню взаємозв’язків між різними етапами процесу, а також підкреслює важливість версійності для забезпечення цілісності та якості наукових публікацій.

Висновки до розділу 2

У розділі 2 було детально описано архітектуру та проектування автоматизованої системи керування версіями наукових публікацій. Встановлено, що система має багаторівневу структуру, яка включає в себе веб-додаток, базу даних, а також механізми аутентифікації та авторизації користувачів.

Згідно з архітектурою, система реалізує модульну архітектурну модель, де різні компоненти, такі як контролери, моделі та представлення, чітко відокремлені один від одного. Це забезпечує зручність в обслуговуванні та розширюваності системи. Важливими аспектами архітектури є безпека, яка

досягається за рахунок використання ASP.NET Core Identity, що надає механізми захисту даних та управління ролями користувачів.

Проектування бази даних включає в себе ключові таблиці, такі як користувачі, статті, рецензії та публікації. Було визначено, що кожна таблиця містить необхідні поля для ефективного зберігання та обробки інформації, а також забезпечує зручний доступ до даних через структуровані запити.

Описані процеси управління публікаціями, включаючи їх створення, редагування та контроль версій, свідчать про інтеграцію автоматизації в систему. Система передбачає різноманітні функції для авторів, рецензентів та адміністраторів, що сприяє ефективному співробітництву між користувачами.

Отже, запропонована система є комплексним рішенням для керування науковими публікаціями, яке забезпечує необхідний рівень безпеки, функціональності та зручності використання. Наступні етапи роботи будуть зосереджені на реалізації описаних функціональних вимог та подальшій оптимізації системи.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ

3.1. Реалізація модулів системи

У сучасних веб-додатках управління користувачами є ключовим аспектом, що забезпечує безпеку та ефективність роботи системи. Для реалізації цієї функціональності в рамках проекту використовувалася технологія ASP.NET, яка включає в себе механізми реєстрації, аутентифікації, авторизації та управління ролями користувачів.

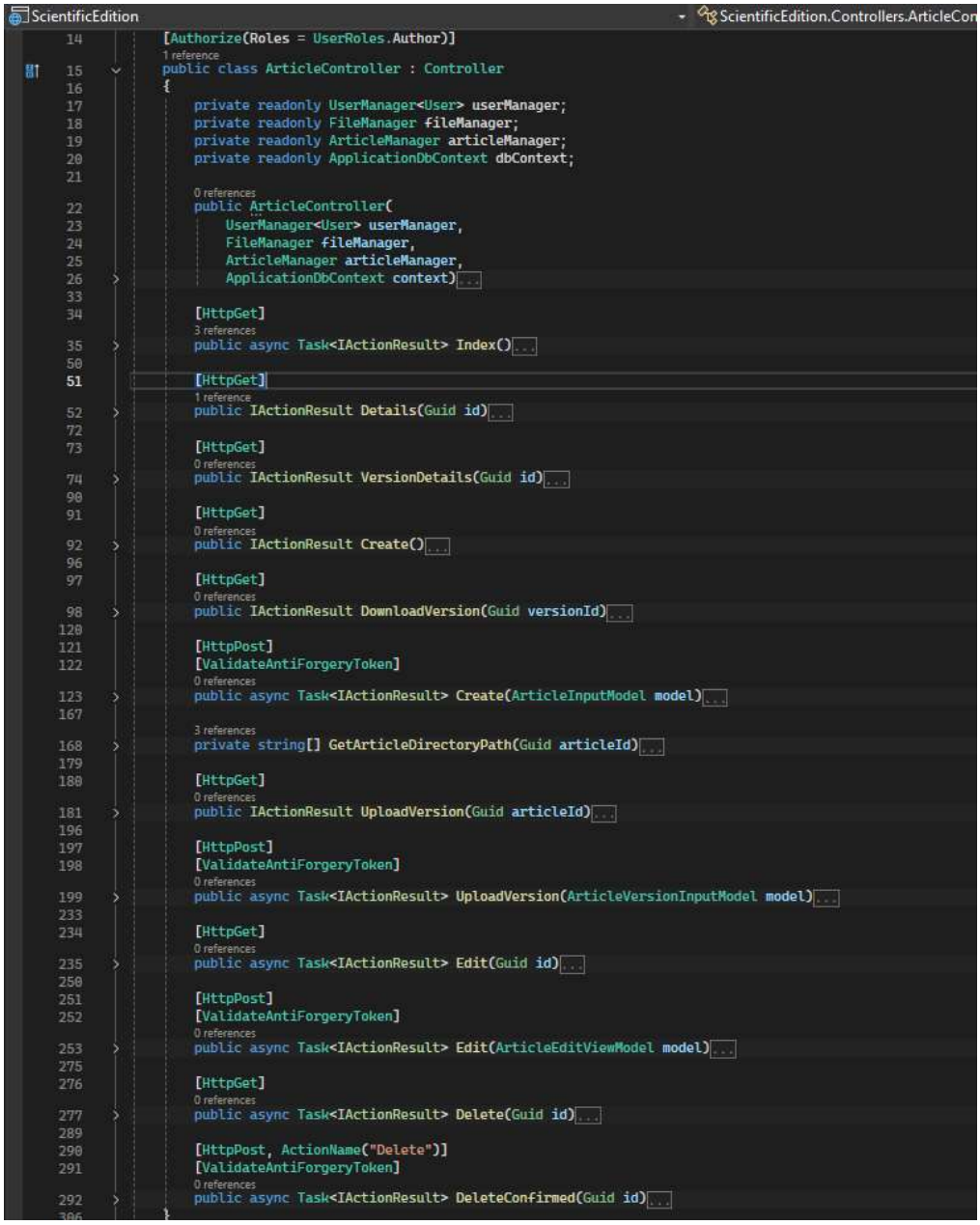


Рис. 3.1 – ArticleController.cs

У цьому коді реалізовано контролер для роботи з науковими статтями в веб-додатку для авторів. Контролер забезпечує функціональність створення, перегляду, редагування, завантаження, оновлення версій статей, а також їх видалення. Код написаний з використанням ASP.NET Core та принципів роботи з базою даних через Entity Framework.

Контролер, що реалізує взаємодію з науковими статтями, містить кілька основних методів, які забезпечують операції для авторів статей. Кожен метод виконує певну функцію, взаємодіє з базою даних і контролює доступ до ресурсу на основі ролей користувачів.

Контролер отримує сервіси через ін'єкцію залежностей (Dependency injection) у конструкторі. Це включає сервіси для управління користувачами, обробки файлів, а також для роботи з базою даних, яка містить дані про статті, їх версії та рецензії. Таке впровадження дозволяє централізовано керувати логікою додатку та забезпечує масштабованість при додаванні нових функцій або змін у бізнес-логіці.

Метод Index відповідає за отримання списку статей поточного автора, зокрема, статей, пов'язаних з користувачем, що аутентифікований та авторизований у системі. Цей метод здійснює запит до бази даних, отримуючи статті, які належать поточному автору, включаючи пов'язані з ними категорії, версії та рецензії. Результат передається до виду для подальшого відображення.

Перегляд детальної інформації про статтю – метод Details. Він дозволяє переглядати детальну інформацію про статтю, за умови, що стаття належить поточному автору. Якщо автор не співпадає з поточним користувачем, сторінка не буде відображена. У випадку наявності версій статті вони сортуються за датою завантаження та передаються в вигляд для подальшого перегляду.

Метод VersionDetails, надає можливість перегляду конкретної версії статті. Тут здійснюється перевірка авторства, щоб переконатися, що користувач має право переглядати дану версію. Окрім цього, версії статей

містять відомості про рецензії, що дозволяє користувачеві побачити всі коментарі та пропозиції, що стосуються конкретної версії.

Метод Create дозволяє користувачеві створити нову статтю, обравши категорію та завантаживши файл. Після цього система генерує нову версію статті з файлом, зберігає її в базі даних, і присвоює статті статус "Створено". Цей процес включає валідацію даних, перевірку правильності вибору категорії та наявності файлу для завантаження.

У методі DownloadVersion реалізовано можливість завантаження конкретної версії статті, що дозволяє авторам отримувати доступ до файлів своїх статей. Для цього перевіряється, чи існує файл на сервері, і чи має користувач права на доступ до нього. У разі успіху користувач отримує файл через механізм завантаження.

Метод UploadVersion дає можливість авторам завантажувати нові версії своїх статей, якщо їх статус перебуває на стадії "На доопрацюванні". Це дозволяє здійснювати редагування статей і подавати їх для рецензування. Після завантаження нової версії статус статті змінюється на "На рецензуванні".

Метод Edit дозволяє редагувати інформацію про статтю, зокрема, її назву та категорію. Після внесення змін, стаття оновлюється в базі даних. Це дозволяє авторам оновлювати свою роботу, коригуючи її відповідно до вимог або змінюючи її структуру.

Метод Delete дає можливість видаляти статті з системи. Після підтвердження видалення, стаття видаляється з бази даних, а також з файлової системи, де зберігаються файли, пов'язані зі статтею. Така операція здійснюється після перевірки існування статті та надання відповідних прав доступу.

Кожна операція взаємодії з даними виконується через Entity Framework, що дозволяє ефективно здійснювати запити та оновлення в базі даних. Для роботи з файлами використовується окремий сервіс FileManager, який відповідає за збереження та видалення файлів, забезпечуючи централізовану

обробку файлів статей. Завантаження та збереження файлів здійснюється за допомогою асинхронних методів, що підвищує ефективність роботи з великою кількістю даних.

Цей контролер є важливою частиною системи для авторів наукових статей, що дозволяє не лише створювати, редагувати та видаляти статті, а й керувати їх версіями. Інтеграція з файловою системою та базою даних забезпечує ефективне збереження даних і зручне управління статтями. Подальший розвиток системи може включати додаткові функції для рецензування, надання зворотного зв'язку або адміністрування статей.

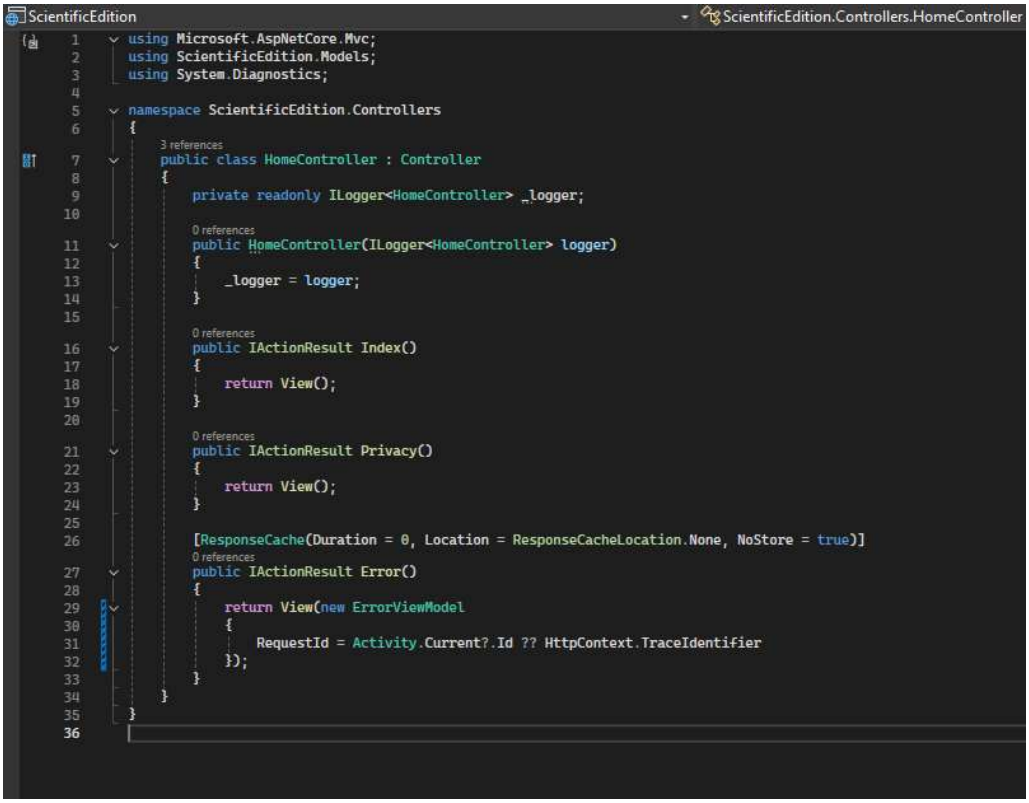


Рис. 3.2 – HomeController.cs

Контролер HomeController, представлений у цьому коді, є частиною стандартної архітектури MVC (Model-View-Controller) в ASP.NET Core. Він відповідає за обробку запитів, що надходять на головну сторінку та сторінку політики конфіденційності, а також за обробку помилок.

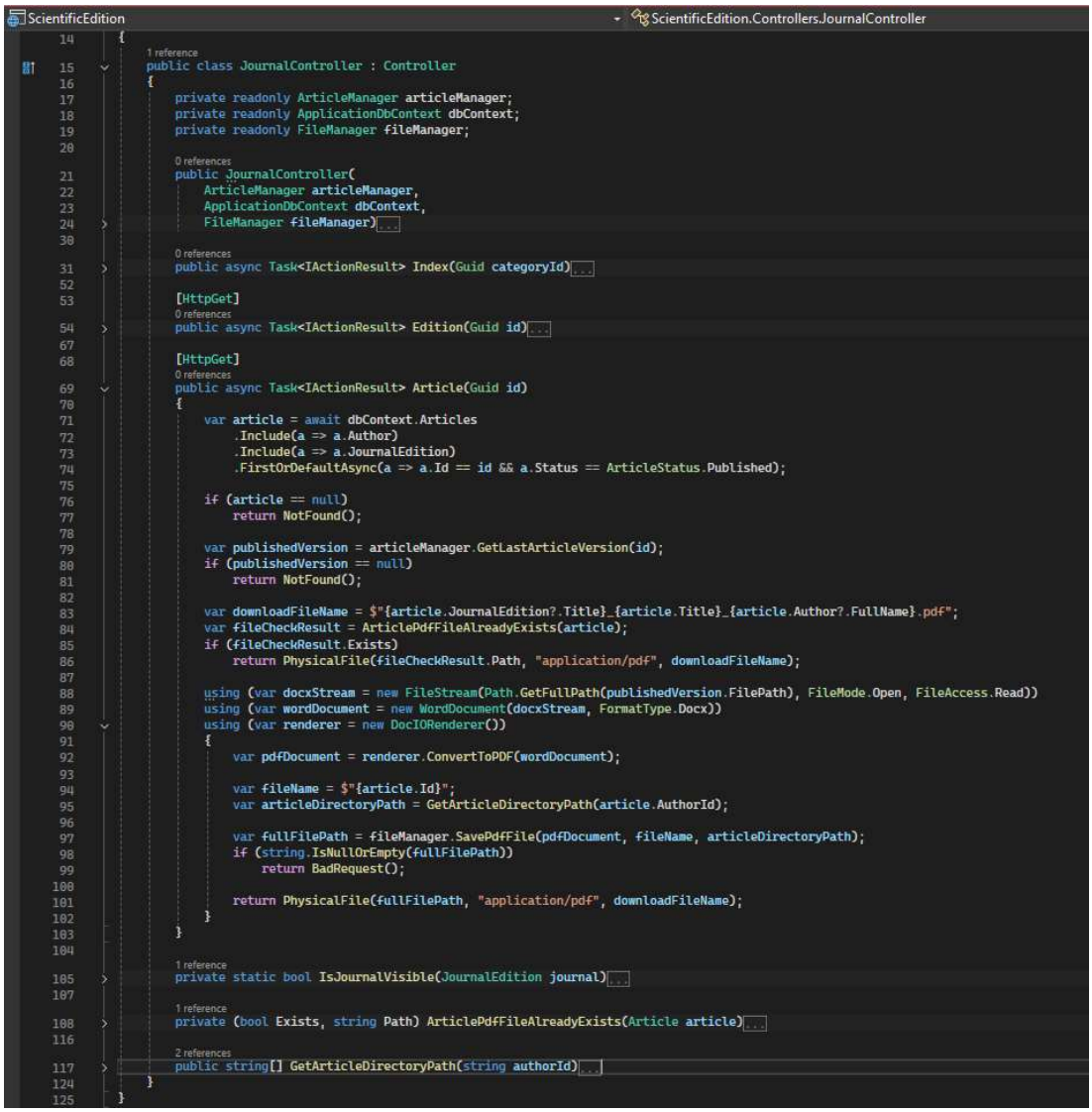
Конструктор класу приймає інтерфейс ILogger<HomeController>, що дозволяє логувати повідомлення та іншу інформацію для цього контролера. Логування є важливою частиною для відслідковування подій та помилок у програмі. У подальшому це дозволяє легко відслідковувати помилки та їх причини, а також зберігати статистику про роботу додатка.

Метод Index відповідає за рендеринг головної сторінки веб-додатку. Він повертає представлення, яке зазвичай містить HTML-код, що відповідає за відображення головної сторінки на клієнтському пристрої. Цей метод буде викликаний при переході на кореневу URL-адресу веб-додатку.

Метод Privacy викликається, коли користувач звертається до сторінки політики конфіденційності. Це також повертає представлення, яке містить текст політики конфіденційності та іншу необхідну інформацію. Зазвичай такі сторінки необхідні для забезпечення юридичних вимог щодо захисту персональних даних користувачів.

Метод Error обробляє помилки, які можуть виникнути в процесі роботи веб-додатку. Важливою частиною цього методу є атрибут [ResponseCache], який вимикає кешування для помилкових сторінок. Це гарантує, що помилки не будуть збережені в кеші браузера або сервера. Крім того, до моделі помилки передається ідентифікатор запиту, що дає змогу відслідковувати конкретні запити, під час яких виникла помилка.

Цей контролер є основою для більш складного додатку, що може мати додаткові маршрути та методи для роботи з іншими частинами додатку, такими як авторизація користувачів, управління контентом та інші функціональні можливості. Логування та обробка помилок є важливими складовими для підтримки стабільності додатку і полегшують його налагодження та адміністрування.




```
{
    1 reference
    public class JournalController : Controller
    {
        private readonly ArticleManager articleManager;
        private readonly ApplicationDbContext dbContext;
        private readonly FileManager fileManager;

        0 references
        public JournalController(
            ArticleManager articleManager,
            ApplicationDbContext dbContext,
            FileManager fileManager)...

        0 references
        public async Task<IActionResult> Index(Guid categoryId)...

        [HttpGet]
        0 references
        public async Task<IActionResult> Edition(Guid id)...

        [HttpGet]
        0 references
        public async Task<IActionResult> Article(Guid id)
        {
            var article = await dbContext.Articles
                .Include(a => a.Author)
                .Include(a => a.JournalEdition)
                .FirstOrDefaultAsync(a => a.Id == id && a.Status == ArticleStatus.Published);

            if (article == null)
                return NotFound();

            var publishedVersion = articleManager.GetLastArticleVersion(id);
            if (publishedVersion == null)
                return NotFound();

            var downloadFileName = $"{article.JournalEdition?.Title}_{article.Title}_{article.Author?.FullName}.pdf";
            var fileCheckResult = ArticlePdfFileAlreadyExists(article);
            if (fileCheckResult.Exists)
                return PhysicalFile(fileCheckResult.Path, "application/pdf", downloadFileName);

            using (var docxStream = new FileStream(Path.GetFullPath(publishedVersion.FilePath), FileMode.Open, FileAccess.Read))
            using (var wordDocument = new WordDocument(docxStream, FormatType.Docx))
            using (var renderer = new DocIORenderer())
            {
                var pdfDocument = renderer.ConvertToPDF(wordDocument);

                var fileName = $"{article.Id}";
                var articleDirectoryPath = GetArticleDirectoryPath(article.AuthorId);

                var fullFilePath = fileManager.SavePdfFile(pdfDocument, fileName, articleDirectoryPath);
                if (string.IsNullOrEmpty(fullFilePath))
                    return BadRequest();

                return PhysicalFile(fullFilePath, "application/pdf", downloadFileName);
            }
        }

        1 reference
        private static bool IsJournalVisible(JournalEdition journal)...

        1 reference
        private (bool Exists, string Path) ArticlePdfFileAlreadyExists(Article article)...

        2 references
        public string[] GetArticleDirectoryPath(string authorId)...
    }
}
```

Рис. 3.3 – JournalController.cs

Контролер JournalController в рамках веб-застосунку, розробленого на платформі ASP.NET Core, є загальнодоступним та виконує важливі функції для керування науковими публікаціями в системі журналів. Зокрема, він відповідає за відображення журналів, статей та їх публікацій, а також за завантаження статей у форматі PDF.

Конструктор цього контролера приймає три залежності, необхідні для його роботи: ArticleManager, який забезпечує управління статтями, ApplicationDbContext для доступу до бази даних, а також FileManager, що використовується для роботи з файлами. Ці залежності дозволяють

контролеру взаємодіяти з даними, зберігати файли та виконувати інші важливі операції.

Один із основних методів контролера — це Index, який відповідає за відображення журналів, що належать до певної категорії. Для цього метод отримує ідентифікатор категорії і виконує запит до бази даних для отримання журналів, які були опубліковані та мають дату публікації, що менша або дорівнює поточній. У разі відсутності відповідної категорії або журналів, метод повертає помилку "не знайдено". Якщо ж дані знайдено, вони передаються для відображення.

Метод Edition виконує подібну роль, але вже для конкретного журналу. Він отримує ідентифікатор журналу, шукає його в базі даних, перевіряє, чи є журнал опублікованим і доступним для перегляду, а також завантажує всі статті, що входять до складу цього видання. У разі невідповідності вимогам видимості журналу, повертається статус 404.

Метод Article відповідає за відображення конкретної статті. Він перевіряє, чи є стаття опублікованою і чи є у неї остання доступна версія. Якщо стаття існує і її версія доступна, контролер перевіряє, чи була вже створена PDF-версія статті. Якщо така версія існує, вона негайно передається на завантаження. Якщо ж PDF-версія відсутня, стаття конвертується з формату DOCX у PDF за допомогою бібліотеки Syncfusion DocIO. Після конвертації файл зберігається на сервері, і користувач отримує доступ до нього через завантаження.

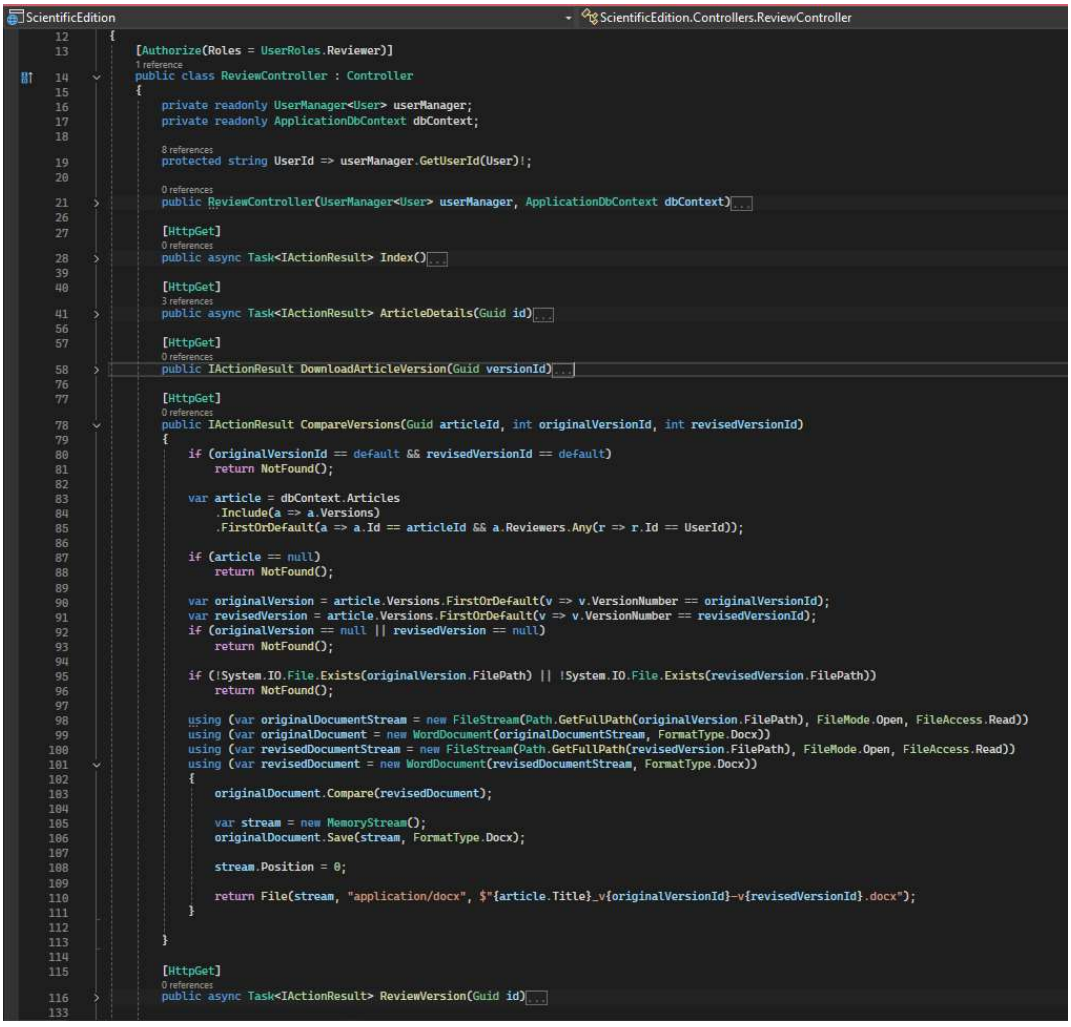
У свою чергу, методи IsJournalVisible і ArticlePdfFileAlreadyExists є допоміжними та використовуються для перевірки доступності журналів і наявності вже збережених PDF-файлів для статей. Перевірка на наявність файлу здійснюється за допомогою стандартних методів для роботи з файловою системою, що дозволяє уникнути повторного створення документів.

Метод GetArticleDirectoryPath генерує шлях до папки для збереження файлів статей, використовуючи ідентифікатор автора статті. Це дозволяє

зберігати файли у окремих папках для кожного автора, що сприяє кращій організації збережених документів. Якщо ідентифікатор автора не надано, генерується виключення, яке забезпечує захист від спроби доступу до неавторизованих файлів.

Функціонал, реалізований у даному контролері, використовує бібліотеку Syncfusion DocIO для конвертації документів формату DOCX в PDF, Entity Framework Core для роботи з базою даних, а також ASP.NET Core MVC для обробки запитів і рендерингу відповідей у вигляді HTML-сторінок.

Таким чином, цей контролер реалізує важливі функціональні можливості для роботи з науковими публікаціями на сайті, забезпечуючи користувачам доступ до актуальних журналів та статей, а також автоматично обробляючи документи для зручності завантаження та збереження.




```
{
    [Authorize(Roles = UserRoles.Reviewer)]
    1 reference
    public class ReviewController : Controller
    {
        private readonly UserManager<User> userManager;
        private readonly ApplicationDbContext dbContext;

        8 references
        protected string UserId => userManager.GetUserId(User)!;

        0 references
        public ReviewController(UserManager<User> userManager, ApplicationDbContext dbContext)...

        [HttpGet]
        0 references
        public async Task<IActionResult> Index()...

        [HttpGet]
        3 references
        public async Task<IActionResult> ArticleDetails(Guid id)...

        [HttpGet]
        0 references
        public IActionResult DownloadArticleVersion(Guid versionId)...

        [HttpGet]
        0 references
        public IActionResult CompareVersions(Guid articleId, int originalVersionId, int revisedVersionId)
        {
            if (originalVersionId == default && revisedVersionId == default)
                return NotFound();

            var article = dbContext.Articles
                .Include(a => a.Versions)
                .FirstOrDefault(a => a.Id == articleId && a.Reviewers.Any(r => r.Id == UserId));

            if (article == null)
                return NotFound();

            var originalVersion = article.Versions.FirstOrDefault(v => v.VersionNumber == originalVersionId);
            var revisedVersion = article.Versions.FirstOrDefault(v => v.VersionNumber == revisedVersionId);
            if (originalVersion == null || revisedVersion == null)
                return NotFound();

            if (!System.IO.File.Exists(originalVersion.FilePath) || !System.IO.File.Exists(revisedVersion.FilePath))
                return NotFound();

            using (var originalDocumentStream = new FileStream(Path.GetFullPath(originalVersion.FilePath), FileMode.Open, FileAccess.Read))
            using (var originalDocument = new WordDocument(originalDocumentStream, FormatType.Docx))
            using (var revisedDocumentStream = new FileStream(Path.GetFullPath(revisedVersion.FilePath), FileMode.Open, FileAccess.Read))
            using (var revisedDocument = new WordDocument(revisedDocumentStream, FormatType.Docx))
            {
                originalDocument.Compare(revisedDocument);

                var stream = new MemoryStream();
                originalDocument.Save(stream, FormatType.Docx);

                stream.Position = 0;

                return File(stream, "application/docx", $"{article.Title}_v{originalVersionId}-v{revisedVersionId}.docx");
            }

        }

        [HttpGet]
        0 references
        public async Task<IActionResult> ReviewVersion(Guid id)...
```

Рис. 3.4 – ReviewController.cs

Контролер ReviewController реалізує функціонал для рецензентів у веб-додатку для наукових публікацій. Його основною метою є забезпечення процесу рецензування наукових статей, наданих користувачам із відповідною роллю "Рецензент". Контролер інтегрує функції перегляду статей, завантаження версій статей, порівняння різних версій, а також надання та збереження рецензій на статті.

Однією з основних функцій контролера є отримання списку статей, призначених для рецензування конкретному користувачу, через метод Index. Цей метод виконує запит до бази даних для отримання статей, до яких рецензент має доступ, з інформацією про категорію статті та версії з відповідними рецензіями. Завдяки цьому рецензенти можуть швидко переглядати свої призначені статті, що полегшує процес рецензування.

Метод ArticleDetails відповідає за детальне відображення інформації про конкретну статтю, включаючи її версії та наявні рецензії. Він забезпечує рецензенту доступ до повної історії статті, включаючи дані про зміни у версіях. Система дозволяє рецензенту переглядати лише ті статті, до яких він був призначений, що забезпечує конфіденційність і відповідальність за рецензування.

Ще однією важливою функцією є завантаження файлів з конкретними версіями статей через метод DownloadArticleVersion. Це дозволяє рецензентам завантажувати відповідні файли версій статей для проведення детального аналізу. Метод враховує ситуації, коли файл статті не існує або коли користувач не має доступу до статті, що забезпечує коректність обробки запитів.

Для порівняння різних версій статей передбачений метод CompareVersions. Він дозволяє рецензентам порівнювати оригінальну та нову версію статті для виявлення змін, що були внесені автором між версіями. Це важливий інструмент для рецензування, оскільки дає змогу швидко та ефективно оцінити зміни, що були внесені, та впливають на якість роботи.

Методи ReviewVersion (HTTP GET та POST) забезпечують механізм для додавання рецензій до конкретної версії статті. Рецензент може заповнити форму для відгуку про статтю, зазначити результати рецензії (наприклад, схвалення або необхідність доопрацювання) і надати коментарі. Після надання рецензії система перевіряє, чи була стаття рецензована двічі, і, залежно від результатів, може змінити статус статті на «Прийнята» або «Потрібне доопрацювання». Це автоматизує процес управління статусом статей, полегшуючи роботу редакторів і авторів.

Методи контролера активно використовують Entity Framework для взаємодії з базою даних. Вони виконують запити для отримання інформації про статті, їх версії та рецензії, використовуючи методи асинхронного виконання для підвищення продуктивності та зручності роботи з великими обсягами даних. Більш того, контролер підтримує управління доступом для певної ролі, використовуючи атрибут [Authorize(Roles = UserRoles.Reviewer)], щоб гарантувати, що лише авторизовані рецензенти можуть здійснювати операції з рецензування статей.

Одним з важливих аспектів реалізації є використання бібліотеки Syncfusion.DocIO для порівняння версій статей у форматі DOCX. Ця бібліотека дозволяє рецензентам переглядати зміни, внесені між версіями, що є важливим інструментом для оцінки якості і змісту наукових публікацій. В кінцевому результаті, цей контролер автоматизує важливі етапи рецензування та дозволяє рецензентам ефективно взаємодіяти з публікаціями, знижуючи ручну працю та підвищуючи ефективність системи наукових публікацій.

Таким чином, контролер ReviewController забезпечує гнучкий і надійний механізм для рецензування наукових статей у веб-додатку, підвищуючи автоматизацію процесів та зручність для користувачів, забезпечуючи ефективний моніторинг і контроль за рецензуванням наукових публікацій.

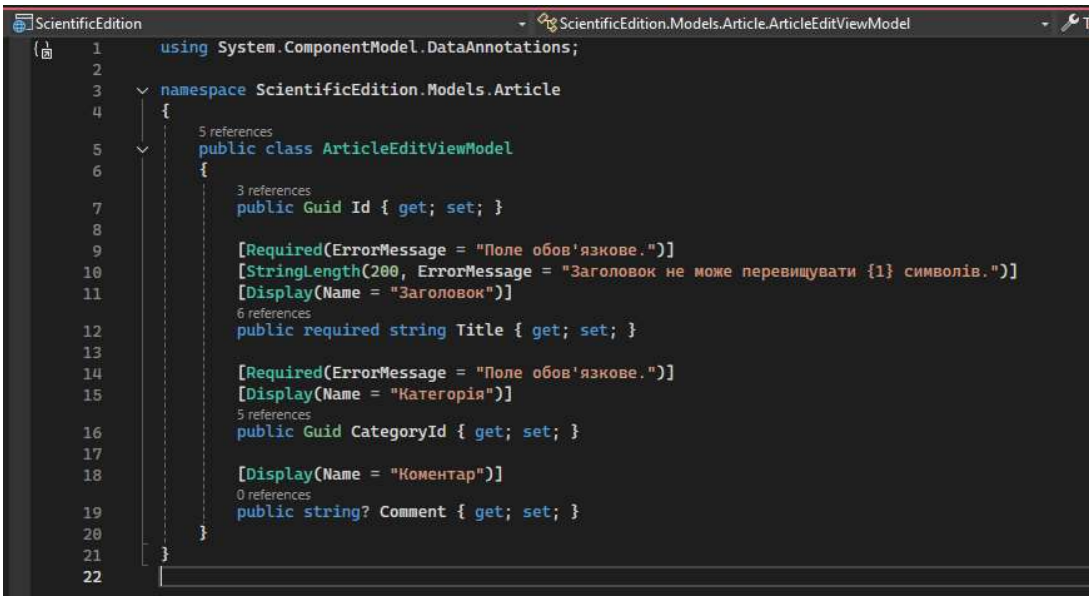


Рис. 3.5 – ArticleEditViewModel.cs

Клас ArticleEditViewModel є моделлю представлення для редагування статей у системі наукового видання. Він виконує роль посередника між користувачем та сервером, забезпечуючи правильне введення даних для редагування наукової статті. Модель включає в себе кілька властивостей, які визначають основні параметри статті, що підлягають редагуванню, зокрема заголовок, категорія та коментар. Усі ці поля мають атрибути валідації для забезпечення коректності введених даних.

Важливою властивістю є Id, що є унікальним ідентифікатором статті. Це поле типу Guid забезпечує однозначну ідентифікацію статті в базі даних, що дозволяє коректно здійснити її редагування або оновлення. Властивість Title визначає заголовок статті, який є обов'язковим для заповнення. Для цього застосовуються атрибути валідації: [Required], що гарантує обов'язковість заповнення цього поля, та [StringLength(200)], яке обмежує довжину заголовка до 200 символів. Ці атрибути дозволяють забезпечити коректність введених даних та зручність їх відображення в інтерфейсі користувача.

Властивість CategoryId визначає категорію статті і є обов'язковим для вибору, що підтверджується атрибутом [Required]. Категорії допомагають організувати наукові статті за темами, що має важливе значення для

класифікації та пошуку контенту на платформі. Відповідно, кожен запис про статтю пов'язується з певною категорією, що додає структури в організацію даних. Також є поле Comment, яке є необов'язковим і дозволяє автору або редактору додавати додаткову інформацію до статті. Це поле є текстовим і допускає порожнє значення, що забезпечує гнучкість у використанні.

Загалом, клас ArticleEditViewModel виконує важливу роль у системі управління статтями, дозволяючи забезпечити ефективний і зручний процес редагування контенту. Валідація введених даних гарантує, що усі поля будуть заповнені правильно та відповідатимуть встановленим вимогам, що сприяє збереженню якості статей та їх правильному відображенню в системі. Цей клас є невід'ємною частиною процесу редагування наукових статей, де важливо дотримуватися чітких вимог щодо структури і вмісту публікацій.

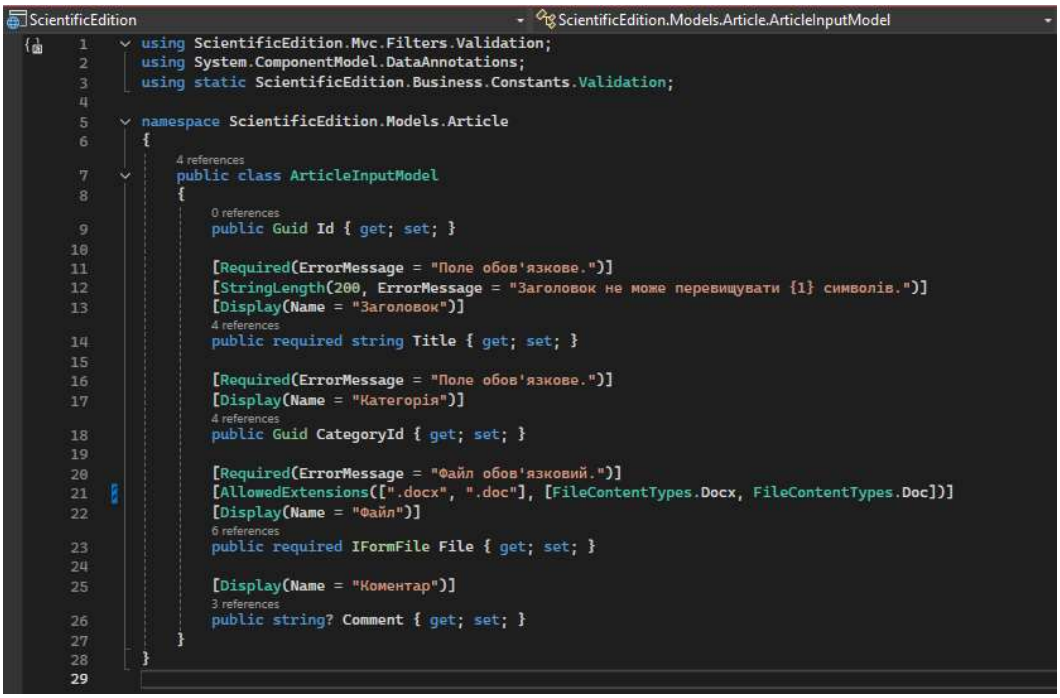


Рис. 3.6 – ArticleInputModel.cs

Клас ArticleInputModel є моделлю для створення та додавання нової статті до системи наукового видання. Ця модель містить важливі властивості, які визначають основні параметри статті, включаючи заголовок, категорію, файл і коментар. Вона також містить атрибути валідації, що допомагають забезпечити правильність введених даних.

Однією з важливих властивостей є Id, що є унікальним ідентифікатором статті. Це поле типу Guid дозволяє однозначно визначити статтю, однак у контексті моделі введення це поле може використовуватись для редагування вже існуючих статей.

Властивість Title представляє заголовок статті, яке є обов'язковим для заповнення. Для цього поля застосовуються два основних атрибути валідації: [Required], що вказує на обов'язковість введення значення, та [StringLength(200)], що обмежує довжину заголовка до 200 символів. Це дозволяє запобігти введенню надто довгих заголовків, що можуть створювати проблеми з відображенням або обробкою в інших частинах системи.

CategoryId визначає категорію, до якої належить стаття. Це поле також є обов'язковим і вказує на те, що кожна стаття має бути класифікована в певній категорії, що сприяє зручному сортуванню та пошуку наукових матеріалів.

Одним з ключових аспектів моделі є поле File, яке визначає файл статті, що має бути завантажений. Це поле є обов'язковим і забезпечується атрибутами валідації, які обмежують типи файлів, що можуть бути завантажені. Використовуються атрибути [Required] для позначення обов'язковості поля та [AllowedExtensions], який вказує на допустимі розширення файлів: .docx, .doc. Завдяки цьому, система гарантує, що тільки файли з правильним розширенням будуть завантажені. Така валідація допомагає уникнути помилок, пов'язаних з неправильними типами файлів.

Властивість Comment є необов'язковою і дозволяє користувачам додавати коментарі до статті. Це поле є текстовим і не має обмежень на довжину, що дозволяє користувачеві вільно вказувати додаткову інформацію про статтю.

Таким чином, клас ArticleInputModel є важливою частиною процесу додавання або редагування статей у системі. Його валідація та структурування дозволяють забезпечити правильність введення даних, відповідність встановленим вимогам і полегшують роботу як для користувачів, так і для

системи в цілому. Ця модель допомагає забезпечити ефективний процес створення наукових публікацій, які відповідатимуть стандартам платформи.

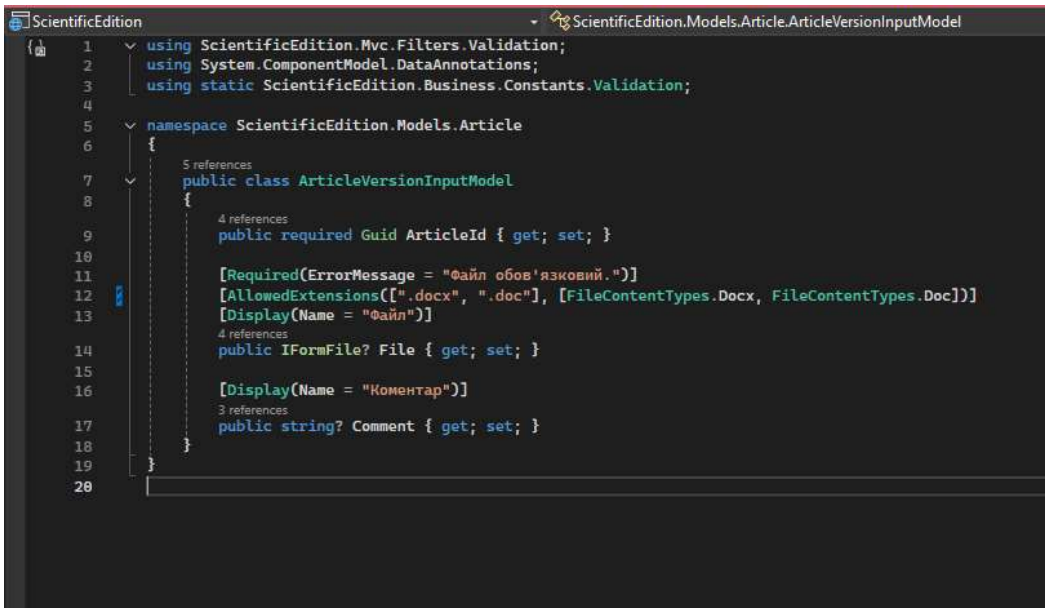


Рис. 3.7 – ArticleVersionInputModel.cs

Клас ArticleVersionInputModel є моделлю введення для додавання нової версії статті до системи наукового видання. Він визначає необхідні властивості, які повинні бути надані користувачем для створення або оновлення версії статті, зокрема, посилання на статтю, файл версії та додатковий коментар.

Перша властивість ArticleId є обов'язковою і представляє унікальний ідентифікатор статті, для якої додається нова версія. Це поле типу Guid використовується для того, щоб чітко асоціювати нову версію з конкретною статтею в системі.

Властивість File визначає файл, який містить нову версію статті. Це поле має атрибут [Required], що вказує на необхідність завантажити файл для створення версії статті. Додатково використовується атрибут [AllowedExtensions], який обмежує дозволені розширення файлів до .docx та .doc, щоб гарантувати, що завантажуються лише сумісні документи. Це

дозволяє запобігти завантаженню файлів з некоректними або непідтримуваними форматами.

Властивість Comment є необов'язковою і дозволяє користувачу додавати коментар до нової версії статті. Це поле є текстовим і не має обмежень на довжину, що дає користувачеві можливість описати особливості або зміни, внесені до цієї версії статті.

Таким чином, клас ArticleVersionInputModel визначає важливі параметри для створення та управління версіями статей у системі наукового видання. Валідація файлів і надання коментарів допомагають забезпечити правильність і коректність введених даних, що сприяє ефективному управлінню науковими публікаціями та їх версіями.

Моделі входять до складу системи і можуть бути використані в процесі створення або редагування статей і їх версій у системі. Валідація даних, зокрема обмеження на типи файлів і довжину текстів, допомагає забезпечити коректність роботи системи і мінімізує ризик помилок у введених даних. Вони також підвищують зручність для користувачів, забезпечуючи чітке управління науковими статтями та їх оновленнями, що є важливим елементом в процесах публікації наукових праць.

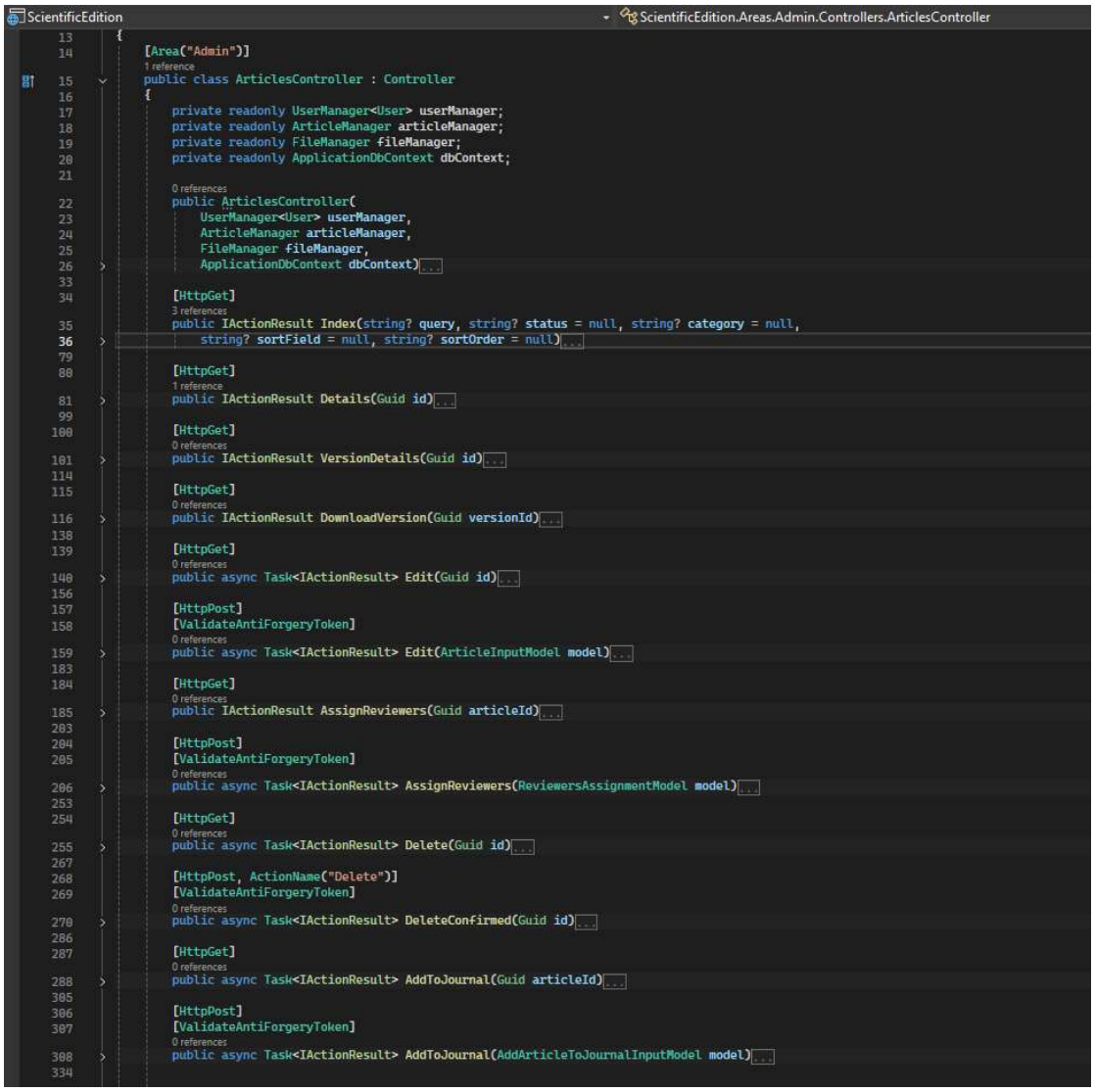


Рис. 3.8 – Areas/Admin/ArticlesController.cs

Контролер ArticlesController реалізує функціональність управління статтями в адміністративному розділі наукового видання. Його основна мета – забезпечити можливість адміністратору керувати статтями, включаючи перегляд, редагування, призначення рецензентів, завантаження версій, видалення та додавання статей до випусків видань.

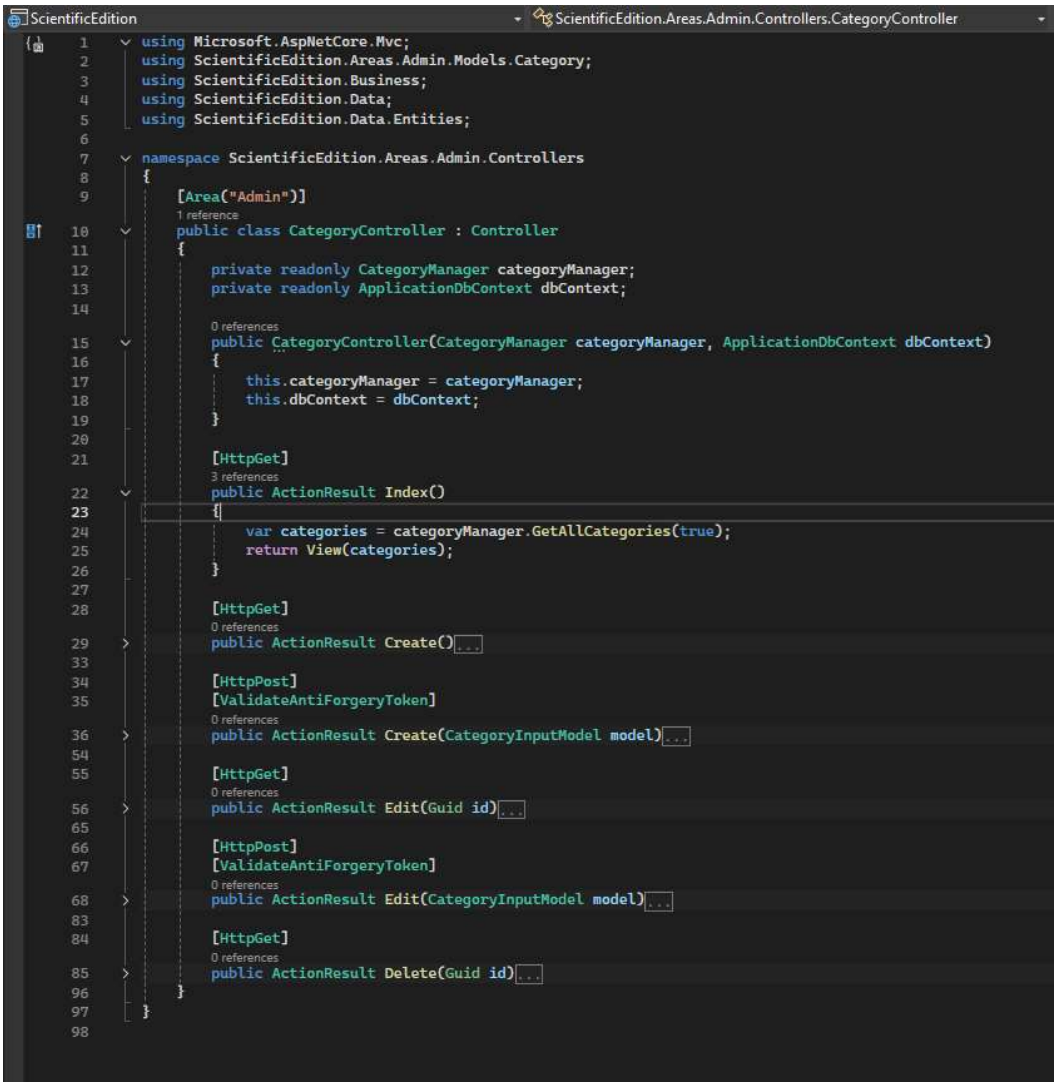


Рис. 3.9 – Areas/Admin/CategoryController.cs

Контролер CategoryController реалізує основну функціональність для управління категоріями у веб-застосунку, створеному з використанням платформи ASP.NET Core MVC. Цей контролер працює у межах адміністративного розділу програми, що позначено використанням атрибута [Area("Admin")], і надає інтерфейс для виконання CRUD-операцій (створення, читання, оновлення, видалення) з об’єктами категорій.

Основна функція контролера полягає у забезпеченні доступу до даних категорій та можливості їх обробки, використовуючи зв’язок із базою даних через ApplicationDbContext, що реалізує ORM Entity Framework. Окрім цього, у контролері використовується сервісний шар у вигляді класу

CategoryManager, який відповідає за виконання бізнес-логіки, пов’язаної з категоріями.

Для роботи з категоріями реалізовано декілька методів. Метод Index надає можливість отримати список усіх наявних категорій із бази даних і передати його у вигляді моделі представлення для відображення на веб-сторінці. Це забезпечує адміністраторам швидкий доступ до повного переліку категорій для їх подальшого управління.

Створення нових категорій здійснюється за допомогою методів Create. Метод із типом HTTP запиту GET відображає форму для введення даних нової категорії, тоді як відповідний метод із типом запиту POST обробляє ці дані. У процесі обробки даних перевіряється їх валідність, забезпечуючи відповідність вимогам моделі CategoryInputModel. Додатково реалізовано перевірку наявності категорії з аналогічною назвою у базі даних, що дозволяє уникнути створення дублікатів. У разі успішної обробки даних створюється новий запис категорії, який додається до бази даних.

Для редагування існуючих категорій використовується пара методів Edit. Метод GET отримує дані про категорію за її унікальним ідентифікатором і передає їх у форму редагування. Метод POST відповідає за внесення змін до бази даних після перевірки валідності отриманих даних. У разі успішного редагування категорія оновлюється, а користувач перенаправляється до загального списку категорій.

Видалення категорій реалізовано через метод Delete, який виконує пошук категорії за ідентифікатором і, якщо вона існує, видаляє відповідний запис із бази даних. Ця операція також забезпечує перенаправлення до сторінки зі списком категорій після успішного видалення.

Контролер дотримується принципів безпечної та надійної обробки даних. Використання атрибута [ValidateAntiForgeryToken] у методах POST забезпечує захист від CSRF-атак, що є ключовим аспектом забезпечення безпеки веб-застосунку. Додатково враховано обробку помилок, таких як спроби редагування чи видалення неіснуючих категорій. У таких випадках

повертається відповідь NotFound, яка повідомляє користувача про неможливість виконання запиту.

Таким чином, контролер CategoryController демонструє добре спроектовану архітектуру для управління категоріями, що поєднує логіку доступу до даних, обробки бізнес-правил і взаємодії з користувачем через інтерфейс представлення. Він забезпечує інтуїтивно зрозумілий, захищений та функціональний інструмент для адміністраторів веб-застосунку.

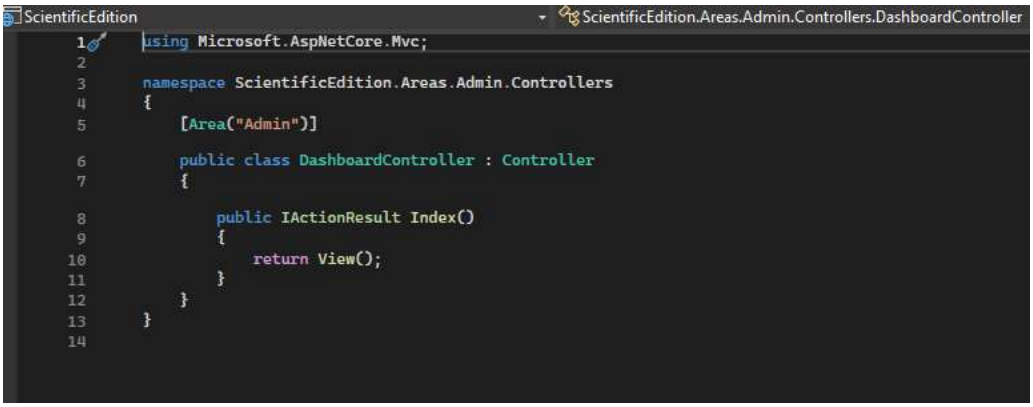


Рис. 3.10 – Areas/Admin/DashboardController.cs

Контролер DashboardController є простим прикладом реалізації адміністративного інтерфейсу у веб-застосунку, створеному на основі ASP.NET Core MVC. Його основна функція — відображення головної сторінки адміністративної панелі.

Контролер позначений атрибутом [Area("Admin")], що вказує на те, що він належить до адміністративної області програми. Це дозволяє логічно розділити функціональність застосунку, забезпечуючи чіткий поділ між адміністративним і користувацьким інтерфейсами. Адміністративна область може містити окремі представлення, маршрути та контролери, що спрощує організацію та масштабування програми.

Метод Index, реалізований у контролері, відповідає за обробку запиту типу GET до основного маршруту панелі адміністратора. Він повертає результат у вигляді представлення, використовуючи метод View(). За

замовчуванням система шукає відповідний файл представлення у директорії Views/Admin/Dashboard/Index.cshtml, що відповідає структурі MVC.

Цей контролер є початковою точкою входу для адміністратора. Основна сторінка, яка відображається методом Index, зазвичай використовується для надання загальної інформації або доступу до інших функціональних модулів адміністративної панелі, таких як управління користувачами, категоріями, контентом тощо.

Незважаючи на свою простоту, цей контролер має важливе значення для організації інтерфейсу адміністратора. Він може бути розширений додатковою логікою, наприклад, передачею даних про загальний стан системи (статистика, сповіщення, звіти) до представлення. Це дозволяє створити зручний і функціональний інтерфейс для адміністраторів веб-застосунку.

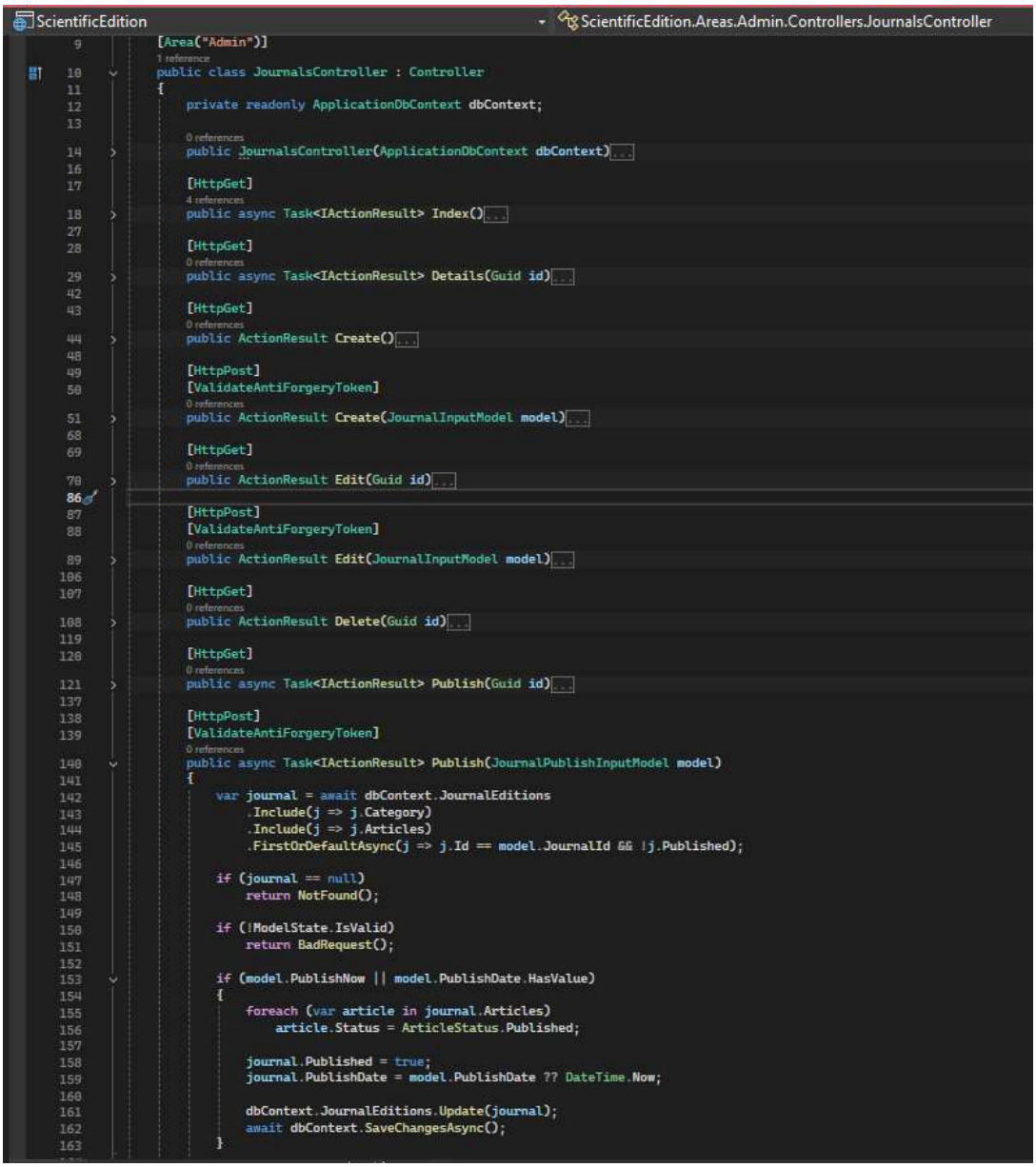


Рис. 3.11 – Areas/Admin/JournalsController.cs

Контролер JournalsController є важливим елементом адміністративної частини веб-додатку, що використовується для управління науковими журналами. Основним завданням цього контролера є забезпечення повного циклу CRUD-операцій (створення, читання, оновлення та видалення) для об’єктів, пов’язаних із журналами, а також реалізація функціоналу публікації випусків. Контролер побудований з використанням платформи ASP.NET Core MVC та інтегрується з базою даних через ORM Entity Framework Core.

Методи контролера реалізовані з урахуванням вимог до сучасних веб-додатків, таких як зручність для користувача, оптимізація роботи з базою

даних і забезпечення масштабованості. Метод Index відповідає за завантаження списку всіх наявних журналів із бази даних. Для цього використовується механізм завантаження пов'язаних даних через методи Include та ThenInclude, які дозволяють отримати інформацію про категорії журналів і їхні статті. Цей підхід забезпечує зменшення кількості запитів до бази даних, що позитивно впливає на продуктивність системи.

Метод Details надає можливість перегляду деталізованої інформації про конкретний журнал. Для цього із бази даних завантажуються всі відповідні дані, включаючи інформацію про статті та їхніх авторів. Важливим аспектом є перевірка наявності журналу в базі даних: якщо запис із вказаним ідентифікатором не знайдено, користувач отримує повідомлення про помилку.

Процеси створення та редагування журналів організовані таким чином, щоб забезпечити максимальну зручність для адміністратора. На етапі створення журналу адміністратор заповнює форму, де вказує назву, опис і категорію. Після цього система перевіряє коректність введених даних за допомогою механізму валідації поточного стану моделі ModelState, а також додає журнал до бази даних. Редагування передбачає попереднє завантаження даних про журнал для відображення їх у формі, що полегшує внесення змін.

Процес видалення журналу реалізовано з урахуванням безпечної роботи із даними. Перед видаленням перевіряється, чи існує запис у базі даних, і лише після цього він видаляється.

Окремої уваги заслуговує функціонал публікації журналу. Контролер надає можливість опублікувати журнал негайно або запланувати публікацію на конкретну дату. У разі публікації змінюється статус усіх статей журналу на Published, що дозволяє автоматизувати процеси управління контентом.

Асинхронність методів, таких як Index та Publish, є важливим аспектом цього контролера. Використання асинхронних запитів забезпечує зменшення затримок у роботі системи, особливо за умови високого навантаження. Це дозволяє підвищити масштабованість додатку та забезпечити комфортну роботу користувачів.

Таким чином, контролер JournalsController є центральним елементом управління науковими журналами в адміністративній частині системи. Його функціональність відповідає сучасним вимогам до веб-додатків і забезпечує гнучке та ефективне управління журналами, спрощуючи роботу адміністратора та покращуючи загальний досвід взаємодії з системою.

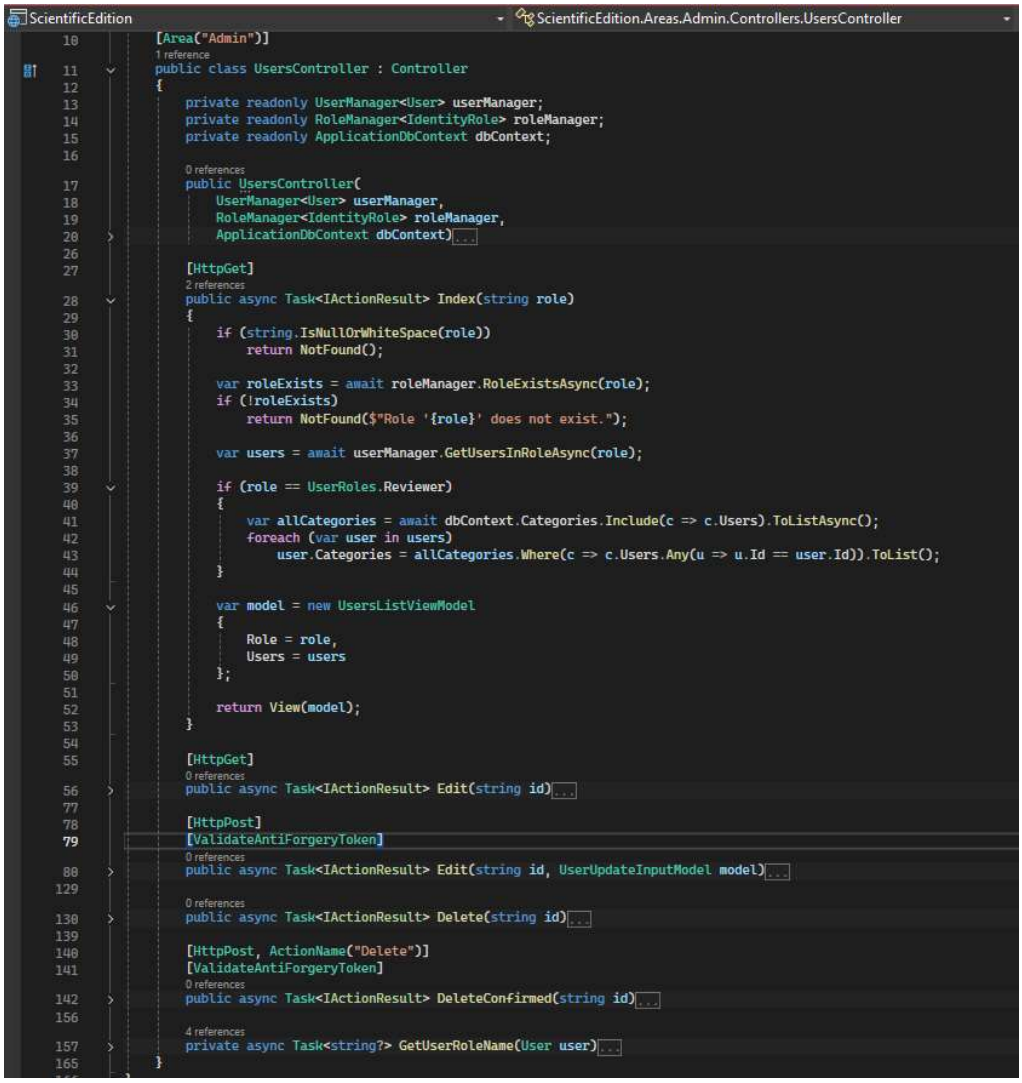


Рис. 3.12 – Areas/Admin/UsersController.cs

Контролер UsersController, що входить до складу адміністративної частини веб-додатку, реалізує функціонал для управління користувачами та їхніми ролями. Його основне призначення – забезпечення адміністративного доступу до функцій системи, зокрема редагування даних користувачів,

призначення ролей, видалення облікових записів, а також управління додатковими параметрами для специфічних ролей.

Ключовим елементом архітектури є використання сервісів ASP.NET Identity, таких як UserManager та RoleManager, що дозволяє централізовано управляти користувачами та ролями, дотримуючись високих стандартів безпеки. Дані користувачів і ролей інтегруються з базою даних через контекст ApplicationDbContext, що забезпечує доступ до всіх необхідних сутностей.

Однією з важливих функцій контролера є можливість відображення списку користувачів, які належать до конкретної ролі. Ця функція реалізується у методі Index, який перевіряє існування ролі за допомогою RoleManager і, у разі успішної перевірки, отримує всіх користувачів, пов’язаних із певною роллю. Особливістю є інтеграція категорій для ролі Reviewer, що дозволяє кожному користувачеві цієї ролі мати список пов’язаних категорій, які додатково завантажуються з бази даних. Це забезпечує гнучкість у налаштуванні специфічних параметрів для різних типів користувачів.

Функція редагування користувачів реалізована через методи Edit для обробки GET- та POST-запитів. У першому випадку завантажується повна інформація про користувача разом із пов’язаними категоріями для відображення у формі редагування. У другому випадку виконуються перевірки валідності введених даних, оновлення особистої інформації користувача, зміна ролей та управління категоріями для ролі Reviewer. Для цього використовуються методи UserManager, які забезпечують коректність змін і дотримання бізнес-логіки.

Процес видалення користувачів побудований із дотриманням принципів безпеки: перед фактичним видаленням адміністратора перенаправляють на сторінку підтвердження, де відображається основна інформація про користувача та його роль. Функціональність видалення підтримує асинхронність, що дозволяє обробляти запити без блокування основного потоку виконання.

Особливістю контролера є його здатність адаптувати функціонал до специфічних ролей. Наприклад, для користувачів із роллю Reviewer реалізовано розширені можливості управління категоріями, що дозволяє забезпечити їхню спеціалізацію в системі. Такий підхід значно підвищує гнучкість системи і її здатність адаптуватися до різних потреб користувачів.

Використання асинхронності у всіх методах контролера дозволяє досягти високої продуктивності системи, особливо за умов значного навантаження, оскільки обробка запитів не блокує основний потік виконання програми. Це забезпечує одночасну обробку великої кількості запитів, що є критично важливим для масштабованості системи.

Узагальнюючи, UsersController виконує роль центрального елемента для управління користувачами в адміністративному модулі. Його функціонал реалізований із дотриманням сучасних підходів до проектування веб-додатків, включаючи інтеграцію з ASP.NET Identity, використання асинхронності, дотримання принципів розділення обов’язків і забезпечення гнучкості. Це робить систему ефективною, масштабованою та легко адаптованою до потреб адміністратора.

3.2. Тестування та валідація системи

Тестування та валідація системи для управління версіями наукового видання є важливими етапами, які забезпечують її надійність і відповідність вимогам користувачів. У рамках цих процесів були застосовані різноманітні методи, що охоплюють функціональні, продуктивні, безпекові аспекти, а також зручність використання системи.

Одним із ключових етапів стало функціональне тестування, яке дозволило перевірити всі основні функції системи. Це тестування включало в себе перевірку механізму реєстрації та аутентифікації користувачів. В результаті тестування було підтверджено, що система коректно обробляє запити на реєстрацію, а також забезпечує аутентифікацію користувачів за

допомогою валідації введених даних. Зокрема, було виявлено, що система коректно реагує на спроби реєстрації з вже існуючими електронними адресами, що підтверджує її стабільність.

Продовжуючи тестування, увагу було приділено можливості управління статтями. Автори системи мали змогу додавати нові статті, редагувати їх, видаляти та переглядати. Кожна з цих функцій була протестована з різними наборами даних, щоб виявити можливі помилки. В результаті тестування функцій управління статтями вдалося виявити кілька недоліків, зокрема, пов'язаних із валідацією введених даних. Це дало змогу оперативно внести корективи та підвищити надійність системи.

Наступним важливим етапом стало тестування процесу рецензування статей. Рецензенти мали можливість переглядати статті та залишати свої коментарі. Під час тестування було виявлено, що автоматичне сповіщення авторів про зміни у статусі рецензії працює належним чином. Це підвищило оперативність зворотного зв'язку, що, в свою чергу, сприяло покращенню комунікації між авторами та рецензентами.

Тестування продуктивності системи дало змогу оцінити її реакцію на високі навантаження. Під час стрес-тестування, яке включало одночасне виконання запитів від великої кількості користувачів, система продемонструвала високу стабільність. Час відповіді залишався в межах прийнятних значень, що свідчить про її готовність до використання в умовах великої кількості одночасних запитів. Це стало особливо важливим фактором, оскільки система розроблена для використання у великій академічній спільноті.

У процесі тестування безпеки системи була проведена перевірка на наявність вразливостей, таких як SQL-ін'єкції та XSS-атаки. Цей етап виявився вкрай важливим, оскільки захист даних користувачів є пріоритетом для будь-якої системи. Результати аудиту безпеки показали, що система має надійні механізми захисту, проте виявлено кілька рекомендацій для подальшого

підвищення рівня безпеки, зокрема щодо складності паролів та впровадження двофакторної аутентифікації.

З метою оцінки зручності використання системи було проведено тестування з участю реальних користувачів. Спостереження за ними під час взаємодії з інтерфейсом системи виявило деякі проблеми в навігації, зокрема щодо розташування елементів управління. Отримані відгуки користувачів стали основою для вдосконалення інтерфейсу, що призвело до підвищення загального рівня задоволеності від використання системи.

Завдяки всебічному підходу до тестування та валідації системи було досягнуто високого рівня впевненості в її функціональності, продуктивності, безпеці та зручності використання. Виявлені недоліки були оперативно усунені, а рекомендації щодо покращення реалізовані, що дозволило значно підвищити якість продукту. У результаті тестування та валідації система готова до впровадження та використання в реальному середовищі, що відкриває нові можливості для авторів, рецензентів і адміністраторів у сфері наукових публікацій.

Висновки до розділу 3

Розділ присвячений розробці та впровадженню системи управління науковими статтями, демонструє комплексний підхід до створення програмного забезпечення, яке відповідає сучасним вимогам академічного середовища. В процесі розробки були реалізовані ключові функціональні модулі, включаючи управління користувачами, публікаціями, рецензуванням, а також адміністративну панель і механізми автоматизації. Кожен з цих модулів був ретельно спроектований та реалізований, що забезпечило функціональність, необхідну для ефективної роботи всіх учасників процесу.

Тестування та валідація системи виявилися критично важливими етапами, які забезпечили впевненість у її надійності та безпеці. Завдяки різноманітним тестуванням вдалося виявити та усунути потенційні недоліки,

що підвищило загальну якість продукту. Користувацьке тестування, проведене з реальними учасниками, дало змогу виявити проблеми в навігації та інтерфейсі, що стало основою для подальших удосконалень.

Таким чином, результати розробки та тестування підтверджують доцільність і ефективність системи управління науковими статтями, яка готова до впровадження в академічне середовище. Ця система створює нові можливості для авторів, рецензентів та адміністраторів, спрощуючи процес публікації та рецензування наукових матеріалів. Впровадження даного програмного забезпечення сприятиме підвищенню якості наукових публікацій, покращенню комунікації між учасниками процесу та оптимізації всіх етапів роботи з науковими статтями.

ВИСНОВКИ

У даній роботі було проведено комплексне дослідження систем керування версіями для наукових публікацій, що включає аналіз сучасних рішень, проектування власної системи та її реалізацію з подальшим тестуванням.

У першому розділі було здійснено детальний огляд існуючих систем керування версіями, що дозволило виявити їхні переваги та недоліки. Аналіз показав, що багато стандартних рішень не відповідають специфічним вимогам наукових видань, зокрема в аспектах, пов’язаних із управлінням публікаціями, рецензуванням та електронним підписом. Виявлені проблеми стали основою для розробки системи, яка б задовольняла потреби авторів, рецензентів і адміністраторів.

Другий розділ роботи був присвячений проектуванню системи, де були визначені основні вимоги, а також спроектовано її архітектуру. Це забезпечило структурованість системи та сприяло її ефективній реалізації. Розробка модулів системи здійснювалася з урахуванням найкращих практик у програмуванні, що дозволило досягти високої якості продукту.

У третьому розділі була представлена реалізація системи та проведене тестування її функціональності та безпеки. Завдяки всебічному тестуванню, що охоплювало різні аспекти роботи системи, вдалося виявити та усунути недоліки, що підвищило надійність та безпеку. Результати тестування показали, що система відповідає вимогам користувачів і готова до впровадження.

Отже, проведене дослідження та реалізація системи керування версіями для наукового видання доводять її доцільність та ефективність. Розроблена система забезпечує зручне управління науковими статтями, оптимізує процес рецензування та публікації, а також сприяє покращенню комунікації між учасниками наукового процесу. Це, у свою чергу, може суттєво підвищити якість наукових публікацій і спростити роботу дослідників у сфері наукової

діяльності. Впровадження даної системи матиме позитивний вплив на академічну спільноту, полегшуючи процеси публікації та рецензування наукових матеріалів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ ТА ЛІТЕРАТУРИ

1. Браун, Дж. (2020). Системи керування версіями: теорія та практика. Київ: Видавництво "Наука і практика".

2. Д'Елія, Д., та Алан, Р. (2019). Версійні системи в академічній сфері. Львів: Видавництво "Підручники і посібники".

3. Іваненко, С. (2018). Аналіз сучасних методів управління версіями в програмному забезпеченні. Журнал комп'ютерних наук, 45(3), 67-72.

4. Коваленко, М. (2021). Системи керування науковими публікаціями: переваги та недоліки. Науковий вісник, 32(2), 102-109.

5. Литвиненко, О., & Яковенко, Т. (2022). Інструменти для рецензування наукових статей. Наукові записки, 15(1), 15-23.

6. Мельник, І. (2019). Методології розробки програмного забезпечення: огляд та порівняння. Журнал програмних систем, 12(4), 45-56.

7. Назаренко, В. (2020). Академічна рецензія: принципи та практика. Харків: Видавництво "Фактор".

8. Олійник, С., & Кравченко, А. (2019). Керування версіями для наукових видань: потреби та рішення. Науковий журнал, 28(3), 90-96.

9. Петров, І. (2021). Сучасні тенденції в управлінні науковими статтями. Вісник університету, 39(6), 100-105.

10. Романов, О. (2019). Електронне рецензування: нові підходи та технології. Київ: Видавництво "Економіка".

11. Сидоренко, Т. (2020). Керування науковими статтями: виклики та перспективи. Науковий огляд, 22(1), 38-45.

12. Ткаченко, А. (2018). Автоматизація процесів у наукових публікаціях. Журнал інформаційних технологій, 17(4), 20-27.

13. Устименко, О. (2021). Системи електронного документообігу у науці. Київ: Видавництво "Академперіодика".

14. Федоренко, П. (2022). Методи валідації програмного забезпечення. Журнал програмних розробок, 6(3), 78-84.

15. Чумак, М. (2019). Оцінка якості наукових статей у сучасному світі. Науковий форум, 15(2), 92-99.

16. Шевченко, В. (2020). Інформаційні технології в науці: сучасні виклики. Львів: Видавництво "Технології".

17. Трофименко, А. (2021). Проблеми рецензування наукових статей в Україні. Київ: Видавництво "Освіта".

18. Джонсон, М. (2018). Version Control for Researchers. Journal of Academic Publishing, 10(1), 45-50.

19. Кемп, Л. (2019). Managing Scientific Publications: A Comprehensive Guide. New York: Academic Press.

20. Вагнер, Р. (2020). The Future of Scientific Review Processes. Science and Technology Review, 30(2), 201-210.

21. Картрайт, П. (2021). Innovations in Scientific Publishing and Review. Journal of Research Management, 18(3), 115-125.

22. Еванс, Л. (2018). Understanding Peer Review in Scientific Publishing. Global Journal of Science, 7(1), 67-75.

23. Хаус, Б. (2020). Effective Version Control Systems for Academic Authors. International Journal of Scholarly Publishing, 12(4), 80-85.

24. Лонг, Т. (2019). Automating the Peer Review Process: Opportunities and Challenges. Publishing Science, 22(3), 110-120.

25. Фернандес, А. (2021). Integrating Version Control with Academic Workflows. Journal of Documentation, 27(5), 22-29.

26. Гончар, О. (2020). Етика рецензування в наукових виданнях. Наукові записки, 13(2), 88-95.

27. Зінченко, Т. (2019). Проблеми автоматизації наукових видань. Вісник інформаційних технологій, 16(3), 57-63.

28. Грищенко, М. (2022). Наукові комунікації: тенденції та перспективи. Науковий огляд, 19(1), 34-40.

29. Левченко, Н. (2018). Системи управління версіями: підходи та практики. Науковий журнал, 25(4), 71-78.

30. Вороніна, К. (2021). Аналіз сучасних інструментів для рецензування статей. Харків: Видавництво "Фактор".

31. Кобець, С. (2019). Технології для автоматизації рецензування. Журнал програмного забезпечення, 9(2), 45-51.

32. Долинський, Ю. (2020). Академічні публікації: виклики сучасності. Київ: Видавництво "Університет".

33. Зінов'єва, Т. (2022). Управління версіями у науковій діяльності. Наукові записки, 30(1), 12-20.

34. Фролов, О. (2021). Дослідження нових методів рецензування. Вісник університету, 40(2), 68-75.

35. Астахова, Ю. (2020). Системи керування науковими даними: проблеми та рішення. Журнал інформатики, 11(4), 102-109.

36. Сердюк, Т. (2021). Наукові видання: сучасні технології та їх застосування. Київ: Видавництво "Наука".

37. Лупенко, Р. (2019). Рецензування наукових статей: теорія та практика. Науковий журнал, 18(3), 55-62.

38. Ткач, С. (2020). Проблеми рецензування в умовах цифровізації. Вісник інформаційних технологій, 10(2), 22-29.

39. Рябенко, В. (2022). Академічні рецензії: інновації та традиції. Львів: Видавництво "Підручники і посібники".

АНОТАЦІЯ

Розробка автоматизованої системи керування версіями наукового видання. Кваліфікаційна робота 2024 р. Здобувач денної форми навчання другого (магістерського) рівня вищої освіти спеціальності 122 Комп’ютерні науки освітньої програми «Комп’ютерні науки» 26Мд-Комп групи КОТЕНКО Олександр Дмитрович. Науковий керівник: професор кафедри комп'ютерних наук та інформаційних технологій, доктор технічних наук ІВАНОВ Дмитро Євгенійович.

Актуальність дослідження: Традиційні підходи до управління версіями наукових статей зазвичай базуються на ручному опрацюванні документів, що часто призводить до виникнення помилок та втрати важливої інформації. Відсутність автоматизованих засобів ускладнює контроль за змінами, унеможливлює швидке відстеження прогресу статті та сповільнює процес рецензування. Існуючі програмні рішення для керування науковими публікаціями зазвичай не враховують усіх аспектів взаємодії між авторами, рецензентами та редакторами, що обумовлює необхідність розробки нових, більш комплексних систем. Розробка автоматизованої системи керування версіями наукового видання дозволить зменшити часові та ресурсні витрати на управління статтями, підвищити ефективність рецензування та забезпечити зручність використання системи для всіх учасників наукового процесу.

Об'єкт дослідження: автоматизована система керування версіями наукового видання, що забезпечує збереження, відстеження та управління версіями наукових статей на різних етапах їх створення, редагування та публікації.

Предмет дослідження: методика розробки та впровадження автоматизованої системи керування версіями наукового видання, зокрема процеси збереження та відстеження змін, управління доступом до різних версій статей, інтеграція функцій рецензування та спільної роботи, а також забезпечення безпеки даних і зручності використання для всіх учасників редакційного процесу.

Мета дослідження: дослідження можливостей розробки автоматизованої системи керування версіями наукового видання, яка забезпечить ефективне відстеження, збереження та управління версіями наукових статей на всіх етапах їх підготовки та публікації, зокрема через інтеграцію функцій для спільної роботи авторів, редакторів і рецензентів, а також забезпечення зручності, безпеки та прозорості процесу керування контентом.

Завдання дослідження:

- Проаналізувати сучасні підходи та програмні засоби, що використовуються для керування науковими публікаціями.
- Розробити концептуальну модель автоматизованої системи, яка б враховувала специфіку взаємодії між авторами, рецензентами та редакторами.
- Реалізувати основні функціональні модулі системи, зокрема, керування версіями статей, рецензування та публікацію наукових матеріалів.
- Провести тестування розробленої системи на предмет функціональності, зручності використання та ефективності управління версіями статей.

Методологія дослідження базується на системному підході до розробки інформаційних систем, що включає етапи аналізу, проєктування, розробки, тестування та впровадження програмного продукту.

Результати роботи: автоматизована система керування версіями наукового видання відкриває нові можливості для ефективного управління публікацією наукових статей, покращуючи процеси редагування, рецензування та публікації матеріалів. Впровадження такої системи дозволяє автоматизувати відстеження змін у версіях статей, забезпечуючи зручний доступ до різних етапів підготовки публікацій. Це сприяє підвищенню ефективності роботи редакцій, зменшує ймовірність помилок та забезпечує збереження історії редагувань.

Розробка цієї системи сприяє значному вдосконаленню процесу співпраці між авторами, редакторами та рецензентами, що дозволяє кожному учаснику працювати з останньою актуальною версією документу. Інтерфейс системи забезпечує легкість у моніторингу змін, а також дозволяє зберігати та відновлювати попередні версії, що важливо для наукових публікацій, де точність і актуальність є критичними.

Завдяки автоматизації процесів публікації наукових статей, система підвищує продуктивність роботи та зменшує витрати часу, необхідного для керування контентом. Це також сприяє покращенню якості наукових видань, адже дозволяє зосередитись на аналізі та якості контенту, а не на технічних аспектах керування версіями.

Отже, автоматизована система керування версіями наукового видання відповідає вимогам сучасних наукових комунікацій і сприяє оптимізації робочих процесів, підвищенню ефективності та зниженню ризику помилок, що є важливими для забезпечення якості наукових публікацій.

Апробація результатів дослідження: участь у Всеукраїнській науково-практичній конференції Інтернет-конференції "Автоматизація та комп’ютерно-інтегровані технології у виробництві та освіті: стан, досягнення, перспективи розвитку" (м. Черкаси, 11-17 травня 2024 року).

Публікація: Результати дослідження висвітлено у матеріалах конференції.

Структура роботи: робота складається із вступу, трьох розділів, висновків, списку використаних джерел та літератури. Загальний обсяг роботи – 56 сторінок, список використаних джерел налічує 39 найменувань.

ANNOTATION

Development of an Automated Version Control System for Scientific Publications. Qualification Work, 2024. Full-time student of a second (Master's) level of higher education in the specialty 122 "Computer Science," educational program "Computer Science," group 26Md-Comp. KOTENKO Oleksandr. Scientific advisor: Professor of the Department of Computer Science and Information Technology, Doctor of Technical Sciences IVANOV Dmytro.

Relevance of the research: Traditional approaches to version control for scientific articles often rely on manual document processing, which frequently leads to errors and the loss of critical information. The absence of automated tools complicates tracking changes, impedes progress monitoring, and slows the review process. Existing software solutions for managing scientific publications typically fail to address all aspects of interaction among authors, reviewers, and editors, emphasizing the need for the development of more comprehensive systems. The creation of an automated version control system for scientific publications will reduce time and resource expenditures, enhance the efficiency of the review process, and improve usability for all participants in the scientific workflow.

Object of the research: An automated version control system for scientific publications that ensures the preservation, tracking, and management of versions of scientific articles at various stages of their creation, editing, and publication.

Subject of the research: The methodology for developing and implementing an automated version control system for scientific publications, including processes for saving and tracking changes, managing access to different article versions, integrating review and collaborative work functions, and ensuring data security and usability for all participants in the editorial process.

Purpose of the research: To explore the possibilities of developing an automated version control system for scientific publications that ensures effective tracking, storage, and management of article versions at all stages of preparation and publication. The system should integrate features for collaborative work among

authors, editors, and reviewers while ensuring usability, security, and transparency in content management processes.

Objectives of the study:

- Analyze current approaches and software tools used for managing scientific publications.
- Develop a conceptual model of an automated system considering the specifics of interaction among authors, reviewers, and editors.
- Implement the core functional modules of the system, including article version control, review, and publication management.
- Test the developed system for functionality, usability, and efficiency in managing article versions.

Methodology: The research methodology is based on a systematic approach to the development of information systems, including the stages of analysis, design, development, testing, and implementation of the software product.

Results: The automated version control system for scientific publications offers new opportunities for efficiently managing the publication of scientific articles, improving the editing, review, and publication processes. Implementing such a system automates version tracking, providing convenient access to various stages of publication preparation. This enhances the efficiency of editorial workflows, reduces the likelihood of errors, and ensures the preservation of editing history.

The development of this system significantly improves collaboration among authors, editors, and reviewers, enabling all participants to work with the most up-to-date document version. The system interface facilitates monitoring changes and allows for the storage and recovery of previous versions, which is crucial for scientific publications where accuracy and relevance are critical.

By automating the publication processes, the system boosts productivity and reduces the time required for content management. This also improves the quality of scientific publications by allowing participants to focus on analysis and content quality rather than technical aspects of version control.

Thus, the automated version control system for scientific publications meets the requirements of modern scientific communication and optimizes workflows, increases efficiency, and reduces error risks, which is essential for ensuring the quality of scientific publications.

Approbation of research results: Participation in the All-Ukrainian scientific-practical online conference "Automation and Computer-Integrated Technologies in Production and Education: State, Achievements, Development Prospects" (Cherkasy, May 11–17, 2024).

Publication: The research results are presented in the conference materials.

Structure of the work: The thesis consists of an introduction, three chapters, conclusions, a list of references, and a bibliography. The total length of the work is 56 pages, with 39 references listed.