Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

РОЗРОБКА СИСТЕМИ УПРАВЛІННЯ ЕЛЕКТРОННИМ АРХІВОМ СТУДЕНТСЬКИХ РОБІТ З ВИКОРИСТАННЯМ PYTHON

DEVELOPMENT OF A MANAGEMENT SYSTEM FOR THE ELECTRONIC ARCHIVE OF STUDENT WORKS USING PYTHON

спеціальність 121 «Інженерія програмного забезпечення»
освітня програма «Інженерія програмного забезпечення»

Виконав: здобувач вищої освіти
групи ІПЗ-41
Жук Ілля Сергійович

Керівник:
Сичук Віктор Анатолійович

Кваліфікаційну роботу
допущено до захисту
«___» _____ 2026 р.
Гарант освітньої програми:
к.т.н., доцент
Ліщина Наталія Миколаївна
_____

Луцьк – 2026 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *інженерії програмного забезпечення*
Ступінь вищої освіти *бакалавр*
Галузь знань: *12 «Інформаційні технології»*
Спеціальність: *121 «Інженерія програмного забезпечення»*
Освітня програма: «*Інженерія програмного забезпечення»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____
«___» _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

ЖУКУ Іллі Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи *«Розробка системи управління електронним архівом студентських робіт з використанням Python»*
Керівник роботи: *к.т.н., доцент Сичук В. А.*
затверджені наказом закладу вищої освіти від *«31» грудня 2025 р.* *№ 541/01-02*

2. Строк подання здобувачем вищої освіти кваліфікаційної роботи бакалавра
«29» травня 2026 р.

3. Вихідні дані до роботи: *Python, Flask, SQLite, HTML, CSS, JavaScript, шаблони Jinja2, Visual Studio Code, вимоги до електронного архіву студентських робіт, методичні вказівки до виконання кваліфікаційної роботи бакалавра.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, що потрібно розробити): *аналіз сучасного стану проблем автоматизації обліку персоналу та управління задачами, постановка завдання, формування вимог до розроблюваної інформаційної системи, обґрунтування вибору технологій та засобів розробки, практична реалізація об'єкта проєктування, розробка структури бази даних, обґрунтування принципів захисту інформаційно-комп'ютерної системи, тестування та налагодження програмного забезпечення*.

5. Перелік графічного матеріалу: *20 рисунків, 4 лістинги, 2 таблиці.*

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз предметної області	*Сичук В. А.*		
Специфікація вимог до розробленої системи	*Сичук В. А.*		
Розробка об’єкта проектування	*Сичук В. А.*		
Нормоконтроль	*Суринович О. М.*		
Гарант ОП	*Ліщина Н. М.*		
Показник запозичень тексту	***___%***		
Академічна доброчесність	*Сичук В. А.*		

7. Дата видачі завдання *«3» лютого 2026 р.*

№	Назва етапів кваліфікаційної роботи бакалавра	Строк виконання етапів роботи	Примітка
1	Огляд літературних джерел по темі кваліфікаційної роботи бакалавра	до 14.02.2026 р.	
2	Аналіз проблеми розробки та впровадження об’єкту проектування	до 03.03.2026 р.	
3	Обґрунтування вибору шляхів, технологій і засобів вирішення поставленого завдання	до 14.03.2026 р.	
4	Розробка функціонально-структурної схеми роботи об’єкта проектування та проектування бази даних	до 14.04.2026 р.	
5	Практична реалізація об’єкта проектування та розробка бази даних	до 20.04.2026 р.	
6	Тестування та налагодження об’єкта проектування	до 04.05.2026 р.	
7	Здача чистового варіанту кваліфікаційної роботи бакалавра на кафедру	до 29.05.2026 р.	

Здобувач вищої освіти _____ Ілля ЖУК

Керівник кваліфікаційної роботи _____ Віктор СИЧУК

АНОТАЦІЯ

Жук І. С. Розробка системи управління електронним архівом студентських робіт з використанням Python. Рукопис. Кваліфікаційна робота бакалавра ОП «Інженерія програмного забезпечення» спеціальності 121 «Інженерія програмного забезпечення». Луцький національний технічний університет. Луцьк, 2026.

Кваліфікаційна робота бакалавра складається зі вступу, трьох розділів, висновків, списку використаних джерел і додатків.

У роботі розроблено систему управління електронним архівом студентських робіт з використанням Python. У першому розділі досліджено проблеми зберігання, систематизації та пошуку студентських робіт, проаналізовано існуючі підходи до організації електронних архівів і сформульовано завдання роботи. У другому розділі визначено вимоги до програмного продукту, спроєктовано структуру системи, моделі даних, ролі користувачів і механізми роботи з файлами. У третьому розділі описано реалізацію основних функцій архіву, зокрема додавання, редагування, перегляд, пошук і класифікацію робіт, а також розглянуто питання захисту даних і тестування. Розроблена система забезпечує впорядковане збереження студентських матеріалів і підвищує зручність доступу до них.

Ключові слова: електронний архів, студентські роботи, Python, вебзастосунок, база даних, пошук, керування файлами.

ABSTRACT

Zhuk I. S. Development of a Management System for the Electronic Archive of Student Works Using Python. Manuscript. Qualification work for bachelor's degree in Software Engineering, specialty 121 Software Engineering. Lutsk National Technical University. Lutsk, 2026.

The bachelor's qualification work consists of an introduction, three chapters, conclusions, a list of references, and appendices.

The work presents the development of a management system for an electronic archive of student works using Python. The first chapter examines the problems of storing, organizing, and searching student papers, analyzes existing approaches to electronic archive organization, and defines the tasks of the work. The second chapter specifies the software requirements and presents the system structure, data models, user roles, and file handling mechanisms. The third chapter describes the implementation of the main archive functions, including adding, editing, viewing, searching, and classifying works, as well as data protection and testing. The developed system provides structured storage of student materials and improves access to them.

Keywords: electronic archive, student works, Python, web application, database, search, file management.

ЗМІСТ

ВСТУП

Актуальність теми. У закладах освіти накопичується значна кількість студентських робіт різних типів: лабораторних, практичних, курсових, бакалаврських, магістерських і матеріалів практики. Якщо такі файли зберігаються в розрізнених папках або передаються між викладачами без єдиного каталогу, ускладнюється пошук потрібного документа, контроль доступу, аналіз наповнення архіву та повторне використання матеріалів у навчальному процесі. Вебзастосунок на основі Flask дає змогу організувати маршрути, шаблони, обробку запитів і взаємодію з користувачами в межах компактної Python-системи [1]. Для локального зберігання структурованих даних доцільно застосовувати SQLite через стандартний модуль Python sqlite3 [2]. Для попереднього перегляду PDF-файлів може використовуватися PyMuPDF, який підтримує рендеринг сторінок у растрові зображення [3].

Мета роботи полягає у розробці системи управління електронним архівом студентських робіт з використанням Python, яка забезпечує завантаження, збереження, класифікацію, пошук, фільтрацію, перегляд і контроль доступу до навчальних матеріалів.

Для досягнення поставленої мети необхідно вирішити такі завдання:

– здійснити аналіз предметної області електронного архівування студентських робіт і визначити проблеми зберігання, пошуку, класифікації та доступу до навчальних матеріалів;

– сформувати вимоги до системи з урахуванням ролей користувачів, типів робіт, атрибутів архівного запису, обмежень доступу, завантаження файлів і попереднього перегляду документів;

– спроєктувати структуру вебзастосунку, базу даних, каталог довідників і логіку взаємодії між користувачами, архівними роботами, файлами та навчальними сутностями;

– обґрунтувати вибір Python, Flask, SQLite, Werkzeug і PyMuPDF для реалізації системи електронного архіву;

– реалізувати основні модулі системи, зокрема авторизацію, керування користувачами, каталогами, архівними роботами, завантаженням файлів, пошуком, фільтрацією, статистикою та попереднім переглядом;

– перевірити працездатність системи, коректність рольового доступу, збереження файлів, пошуку, фільтрації, перегляду документів і керування довідниками.

Об’єктом дослідження є процеси зберігання, класифікації, пошуку та використання студентських робіт у навчальному середовищі.

Предметом дослідження є методи, моделі та програмні засоби розробки вебсистеми електронного архіву студентських робіт на основі Python, Flask і SQLite.

Практична цінність розробки полягає у створенні вебзастосунку, який дає змогу централізовано зберігати студентські роботи, швидко знаходити їх за навчальними ознаками, обмежувати доступ залежно від ролі користувача, переглядати PDF-документи в браузері та керувати довідниками кафедр, спеціальностей, груп, предметів і викладачів.

РОЗДІЛ 1
АНАЛІЗ МЕТОДІВ РОЗРОБКИ СИСТЕМИ УПРАВЛІННЯ ЕЛЕКТРОННИМ АРХІВОМ СТУДЕНТСЬКИХ РОБІТ ТА ПОСТАНОВКА ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

1.1 Аналіз сучасного стану проблеми

Електронне архівування студентських робіт передбачає не лише збереження файлів, а й опис кожного матеріалу навчальними атрибутами, підтримку пошуку, фільтрації, контроль доступу та довготривале використання архіву. Для закладу освіти важливо мати можливість швидко знайти роботу за автором, групою, спеціальністю, кафедрою, дисципліною, керівником або типом роботи. Якщо матеріали зберігаються тільки у папках файлової системи, ускладнюється класифікація, виникають дублікати, складно обмежити доступ і неможливо централізовано переглядати статистику архіву.

Одним із найпоширеніших рішень для створення інституційних репозитаріїв є DSpace. На рисунку 1.1 показано приклад сторінки опису елемента в DSpace.

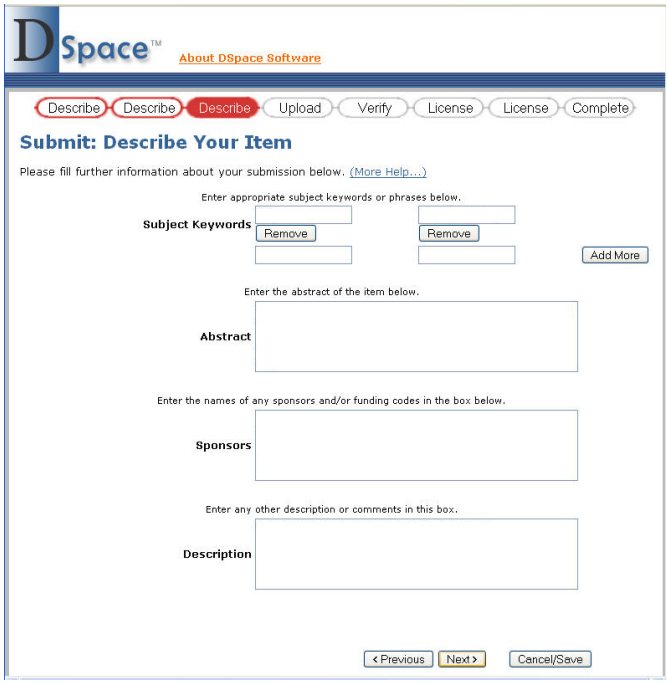

Рисунок 1.1 – Сторінка опису елемента в DSpace [4]

Система використовується академічними, неприбутковими та комерційними організаціями для побудови відкритих цифрових репозиторіїв, підтримує різні типи цифрових файлів, гнучке налаштування, груповий контроль доступу й інтеграції із зовнішніми сервісами [5].

Перевагою DSpace є орієнтація на довготривале збереження цифрового контенту, підтримка різних форматів файлів, структуровані метадані, колекції, права доступу та пошук. Такий підхід добре підходить для університетських репозиторіїв, де потрібно зберігати наукові публікації, дисертації, набори даних і навчальні матеріали. Недоліком у контексті розроблюваної системи є складність розгортання й адміністрування, а також надмірність частини функцій для локального архіву студентських робіт, який має бути простим для кафедри або навчальної групи.

EPrints є ще однією відкритою платформою для створення цифрових репозиторіїв. Офіційний сайт позиціонує її як систему для репозиторіїв дослідницьких спільнот, а документація описує простий і розширений пошук за назвою, авторами, документами, датою та іншими полями [6, 7]. На рисунку 1.2 наведено фрагмент сторінки EPrints із доступом до актуального релізу платформи.

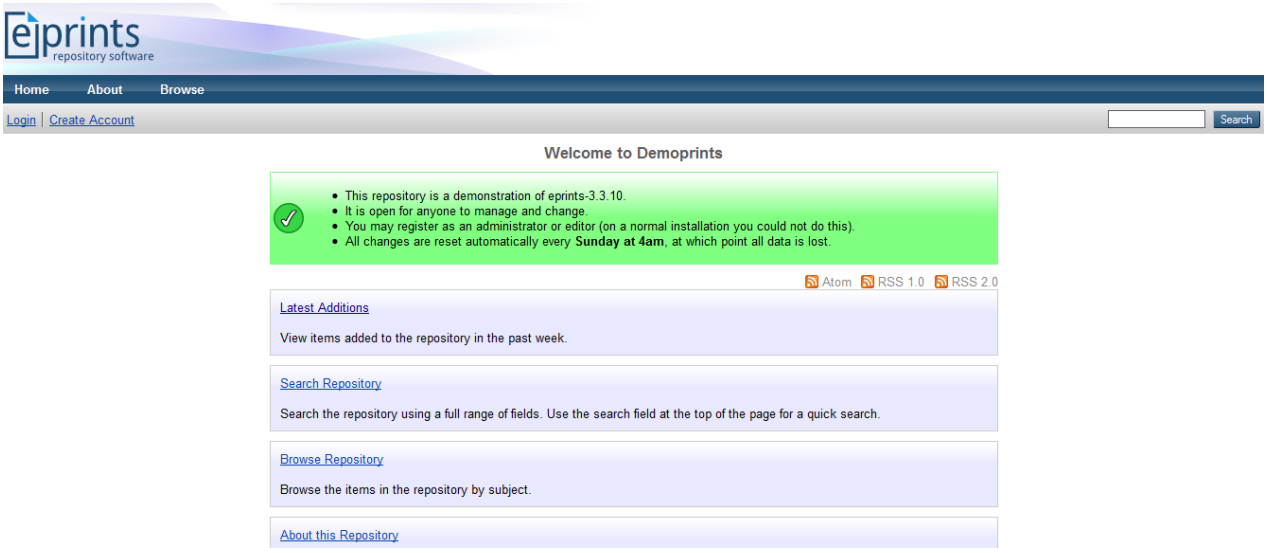

Рисунок 1.2 – Сторінка платформи EPrints [6]

Перевагою EPrints є розвинена модель пошуку, можливість індексації повнотекстових документів, підтримка метаданих і використання в академічних репозитаріях. Водночас система є універсальним репозитарним рішенням, яке потребує окремого серверного налаштування й адміністрування. Для архіву студентських робіт доцільніше реалізувати компактнішу структуру, де довідники кафедр, спеціальностей, груп, дисциплін і викладачів прямо відповідають навчальному процесу.

Figshare є веборієнтованим репозитарієм для збереження, поширення та цитування дослідницьких результатів. У документації зазначено, що платформа приймає різні типи файлів, підтримує метадані, попередній перегляд у браузері, пошук, організаційні репозитарії та довготривале збереження матеріалів [8]. Перевагою Figshare є зручність публікації відкритих матеріалів і підтримка великої кількості форматів. Недоліком для розроблюваної системи є орієнтація на відкриту наукову комунікацію, а не на внутрішній контроль доступу до навчальних робіт різних ролей користувачів.

Zenodo також використовується для розміщення наукових матеріалів, наборів даних, програмного забезпечення та інших цифрових об’єктів. Довідка Zenodo описує процес створення нового запису: завантаження файлів, заповнення метаданих, вибір типу ресурсу, налаштування видимості, попередній перегляд і публікацію (рис. 1.3).

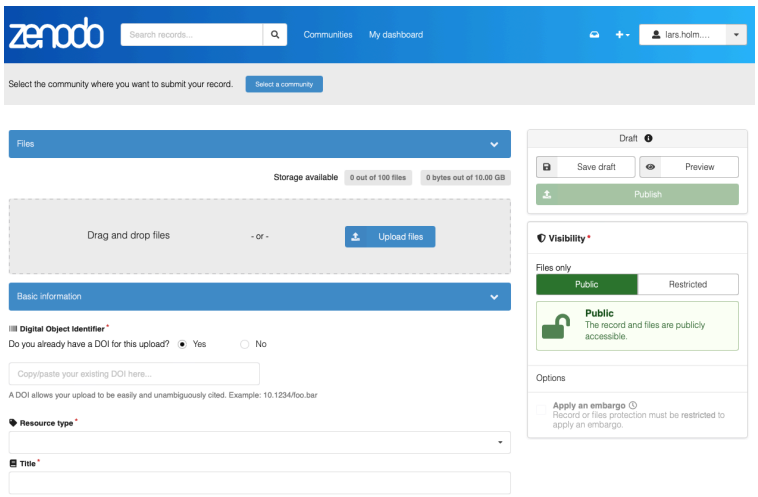

Рисунок 1.3 – Форма створення нового запису в Zenodo [9]

Перевагою Zenodo є простий процес публікації, підтримка метаданих, різних типів ресурсів, налаштування видимості й попереднього перегляду перед оприлюдненням. Однак Zenodo призначений насамперед для публікації матеріалів у відкритому науковому середовищі, тоді як електронний архів студентських робіт потребує локального керування користувачами, навчальними довідниками, рівнями доступу, внутрішніми файлами та попереднім переглядом документів без обов'язкової публікації назовні.

У таблиці 1.1 наведено порівняння розглянутих аналогів із розроблюваною системою електронного архіву студентських робіт.

Таблиця 1.1 – Порівняння аналогів електронних репозитаріїв

Критерій	DSpace	EPrints	Figshare	Zenodo	Розроблювана система
Основне призначення	Інституційний цифровий репозитарій	Академічний репозитарій із пошуком і метаданими	Вебрепозитарій дослідницьких результатів	Публікація наукових матеріалів і наборів даних	Локальний архів студентських робіт
Переваги	Довготривале збереження, різні формати, контроль доступу	Пошук, метадані, повнотекстова індексація	Попередній перегляд, цитування, багато форматів	Зручне завантаження, метадані, видимість, DOI	Навчальні довідники, ролі, локальні файли, PDF прев'ю
Недоліки	Складне розгортання для невеликого архіву	Потребує окремого адміністрування	Орієнтація на відкриту публікацію	Не призначений для внутрішнього навчального доступу	Менше глобальних репозитарних функцій, але точніша відповідність задачі
Корисні ідеї	Метадані, колекції, права доступу	Пошук за атрибутами й документами	Перегляд файлів у браузері	Форма завантаження й налаштування видимості	Фільтри, ролі, довідники, прев'ю, керування файлами

Аналіз аналогів показав, що наявні репозитарні платформи добре розв'язують задачі довготривалого збереження, публікації, опису метаданих і відкритого пошуку. Проте для студентських робіт потрібна система з вужчою

навчальною орієнтацією: ролі гостя, зареєстрованого користувача, викладача й адміністратора; довідники кафедр, спеціальностей, груп, дисциплін і викладачів; різні рівні доступу до файлів; підтримка PDF, DOC, DOCX, TXT і RTF; формування прев'ю документів; локальне зберігання файлів і швидка фільтрація за навчальними ознаками. Саме ці вимоги визначають доцільність розроблення власної вебсистеми на Python, Flask і SQLite.

1.2 Постановка завдання на кваліфікаційну роботу бакалавра

В результаті аналізу процесів зберігання студентських робіт було встановлено, що електронний архів повинен поєднувати функції файлового сховища, каталогу навчальних атрибутів і системи контролю доступу. Для користувача важливо швидко знайти роботу за назвою, автором, кафедрою, групою, спеціальністю, роком, курсом, типом або оцінкою, а для викладача й адміністратора – мати можливість додавати матеріали, керувати довідниками та контролювати права доступу. На основі цього було сформовано наступні завдання:

– здійснити аналіз предметної області електронного архівування студентських робіт, визначити типи матеріалів, які зберігаються в системі, проблеми розрізненого файлового збереження, потребу в метаданих і вимоги до пошуку навчальних документів;

– сформувати вимоги до системи з урахуванням ролей гостя, зареєстрованого користувача, викладача й адміністратора, типів робіт, атрибутів архівного запису, обмежень доступу, допустимих форматів файлів, текстового попереднього перегляду та графічних прев'ю PDF;

– спроєктувати структуру вебзастосунку, базу даних, каталог довідників і логіку взаємодії між користувачами, архівними роботами, файлами та навчальними сутностями, передбачивши зв'язки між кафедрами, спеціальностями, групами, предметами, викладачами й завантаженими роботами;

– обґрунтувати вибір Python, Flask, SQLite, Werkzeug і PyMuPDF для реалізації системи електронного архіву, врахувавши потребу у вебінтерфейсі, локальній базі даних, захищеній роботі з паролями, завантаженні файлів і створенні прев’ю документів;

– реалізувати основні модулі системи, зокрема авторизацію, реєстрацію, керування користувачами, одноразові посилання активації та скидання пароля, довідники кафедр, спеціальностей, груп, предметів і викладачів, додавання та редагування архівних робіт, пошук, фільтрацію, сортування, статистику й попередній перегляд;

– перевірити працездатність системи, коректність рольового доступу, збереження й заміни файлів, видалення архівних записів, формування PDF-прев’ю, текстового перегляду Word/TXT/RTF-документів, роботи фільтрів і дотримання обмежень керування довідниками.

РОЗДІЛ 2
СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ

2.1 Аналіз, визначення вимог до розроблюваного програмного забезпечення та проєктування програмного забезпечення

Під час аналізу предметної області встановлено, що система управління електронним архівом студентських робіт повинна зберігати не лише файли робіт, а й навчальні метадані, без яких архів швидко перетворюється на набір розрізнених документів. Кожна робота має бути описана назвою, ПІБ студента, кафедрою, спеціальністю, групою, навчальним роком, курсом, типом роботи, дисципліною, викладачем, оцінкою, форматом файлу, рівнем доступу та службовими даними про завантаження. Такий набір атрибутів потрібний для пошуку, фільтрації, сортування, статистичного аналізу та обмеження доступу до матеріалів.

Основні варіанти використання системи наведено на рисунку 2.1.

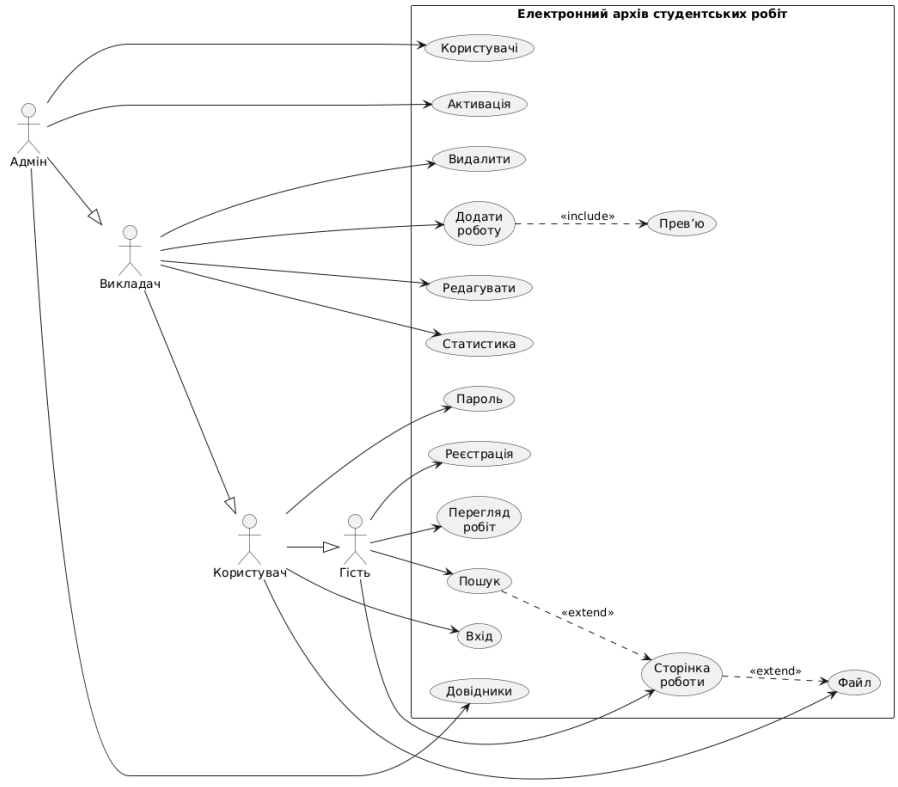

Рисунок 2.1 – UML-діаграма варіантів використання розроблюваної системи

Основними групами користувачів визначено гостя, зареєстрованого користувача, викладача й адміністратора. Гість може переглядати лише відкриті матеріали архіву, зареєстрований користувач отримує доступ до ширшого переліку робіт, викладач додає та редагує архівні записи, а адміністратор керує користувачами, ролями й довідниками. Такий розподіл прав відповідає призначенню системи, оскільки студентські роботи можуть містити навчальні матеріали з різними рівнями відкритості, а їх завантаження й редагування повинні виконуватися відповідальними користувачами.

Функціональні вимоги до системи охоплюють роботу з обліковими записами, архівними записами, файлами, довідниками, пошуком і статистикою. Система повинна підтримувати реєстрацію користувача, вхід до облікового запису, завершення сеансу, активацію користувача та скидання пароля за одноразовим токеном. Для архівних робіт необхідно реалізувати додавання нового запису, редагування метаданих, заміну або збереження файлу, видалення запису користувачем з достатніми правами, перегляд детальної сторінки роботи та завантаження файлу відповідно до рівня доступу. Окремою вимогою є підтримка попереднього перегляду, щоб користувач міг оцінити зміст роботи без обов’язкового завантаження документа.

Для зручної роботи з великим архівом система повинна підтримувати пошук за назвою, студентом, кафедрою, спеціальністю, групою, навчальним роком, курсом, типом роботи, форматом файлу та діапазоном оцінки. Користувач має мати змогу комбінувати кілька фільтрів і сортувати результати. Адміністратор повинен керувати довідниками кафедр, спеціальностей, навчальних груп, дисциплін і викладачів, оскільки ці дані багаторазово використовуються під час створення архівних записів. Додатково передбачено сторінку статистики, яка узагальнює кількість робіт за навчальними атрибутами та допомагає оцінювати наповнення архіву.

Нефункціональні вимоги визначають якість і надійність роботи системи. Паролі користувачів повинні зберігатися у вигляді хешів, а не відкритого тексту. Доступ до дій має перевірятися відповідно до ролі користувача, причому

адміністративні функції не повинні бути доступні звичайним користувачам або гостям. Система має приймати лише допустимі формати файлів: PDF, DOC, DOCX, TXT і RTF, а також контролювати максимальний розмір завантаження. Для збереження цілісності даних довідники не повинні видалятися, якщо вони використовуються в архівних записах. Інтерфейс повинен забезпечувати зрозумілу навігацію між каталогом робіт, формою додавання, сторінкою перегляду, довідниками, користувачами та статистикою.

Проєктування системи виконано як вебзастосунок на Python із використанням Flask. Flask забезпечує маршрутизацію запитів, оброблення форм, роботу із сесіями та передавання даних у HTML шаблони [1]. Для зберігання структурованих даних використано SQLite, що є доцільним для локального вебзастосунку з помірним обсягом даних і простою схемою розгортання [2]. Файли студентських робіт зберігаються у файловій системі в каталозі завантажень, а база даних містить службові імена файлів, метадані та зв’язки з користувачами й довідниками. Узагальнену компонентну структуру системи наведено на рисунку 2.2.

Рисунок 2.2 – UML-діаграма компонентів розроблюваної системи

Основними компонентами системи є вебінтерфейс, маршрути Flask, модуль автентифікації та ролей, модуль керування архівними роботами, модуль

довідників, модуль статистики, файлова підсистема, база даних SQLite і модуль формування попереднього перегляду. Вебінтерфейс складається з HTML шаблонів, які відображають каталог робіт, сторінку роботи, форми входу й реєстрації, форму додавання або редагування роботи, сторінки довідників, користувачів і статистики. Маршрути Flask приймають запити користувачів, перевіряють права доступу, виконують SQL запити та передають результати у шаблони.

Узагальнену UML-діаграму класів системи наведено на рисунку 2.3.

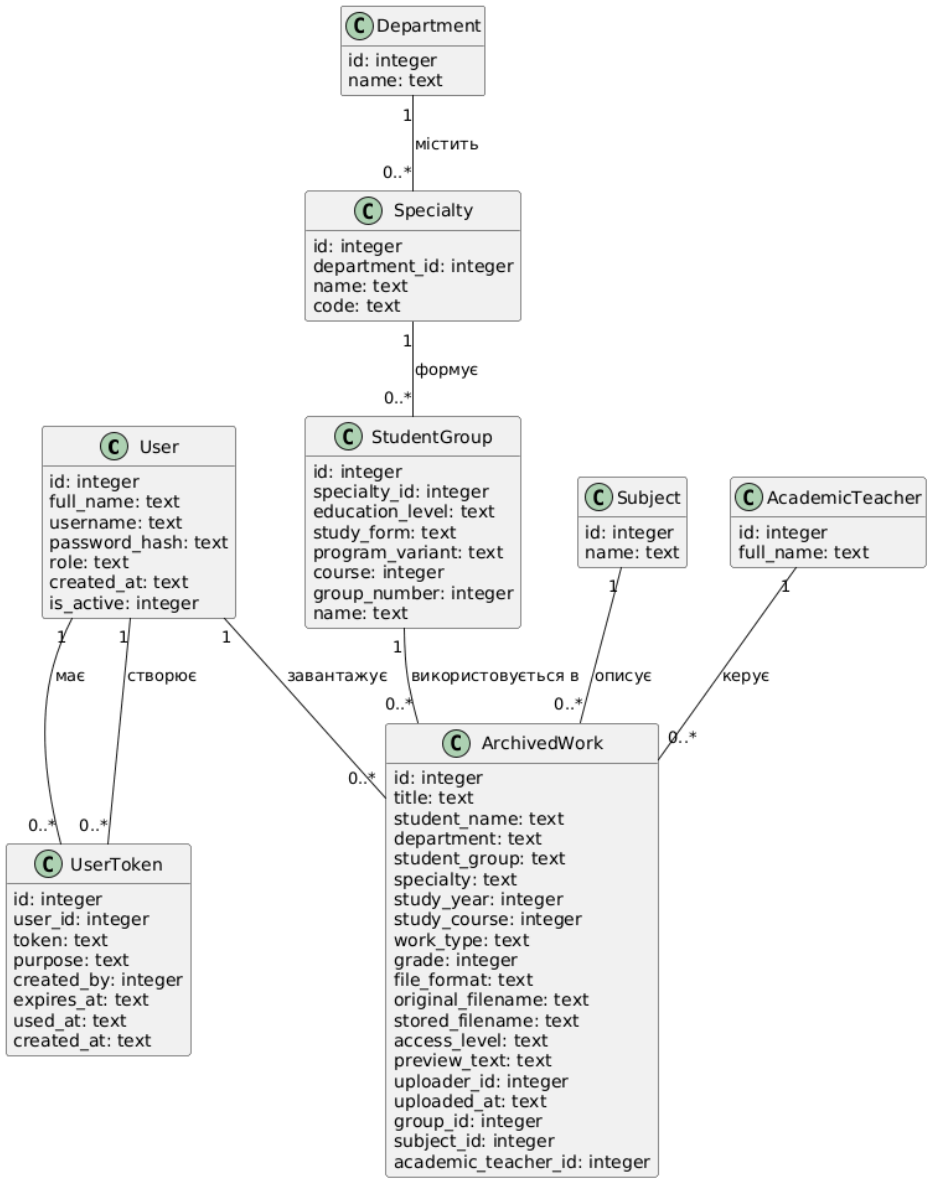

Рисунок 2.3 – UML-діаграма класів системи управління електронним архівом студентських робіт

Проєктна модель даних системи побудована навколо сутності архівної роботи. Архівний запис зберігає навчальні атрибути, назву, дані студента, файл, формат, оцінку, рівень доступу, текст попереднього перегляду, дату завантаження та посилання на користувача, який додав роботу. Користувач має роль і може бути пов’язаний із завантаженими роботами. Одноразові токени використовуються для активації облікового запису або скидання пароля. Довідники кафедр, спеціальностей, груп, дисциплін і викладачів допомагають стандартизувати дані, щоб у фільтрах не виникали різні варіанти одного й того самого значення.

Послідовність додавання роботи до архіву наведено на рисунку 2.4.

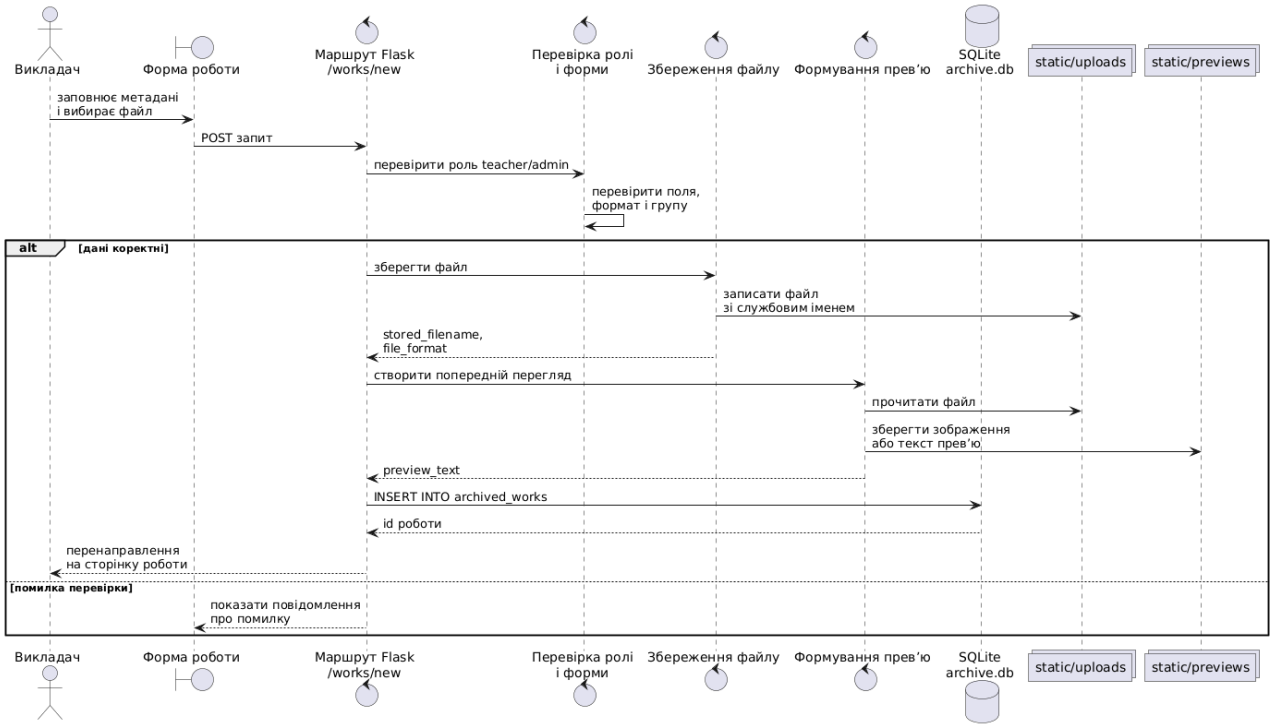

Рисунок 2.4 – UML-діаграма послідовності додавання студентської роботи до архіву

Сценарій додавання студентської роботи починається з перевірки ролі користувача. Якщо користувач має роль викладача або адміністратора, система відкриває форму створення архівного запису. Після надсилання форми перевіряються обов’язкові поля, коректність навчальної групи, допустимість

формату файлу та наявність прикріпленого документа. Далі файл отримує безпечне службове ім’я, зберігається в каталозі завантажень, а система визначає його формат і формує текстове або графічне прев’ю. Для PDF документів використовується PyMuPDF, що дає змогу отримувати текст і будувати зображення першої сторінки [3].

Сценарій пошуку та перегляду роботи орієнтований на користувача, який відкриває каталог архіву. Послідовність пошуку та перегляду архівної роботи наведено на рисунку 2.5.

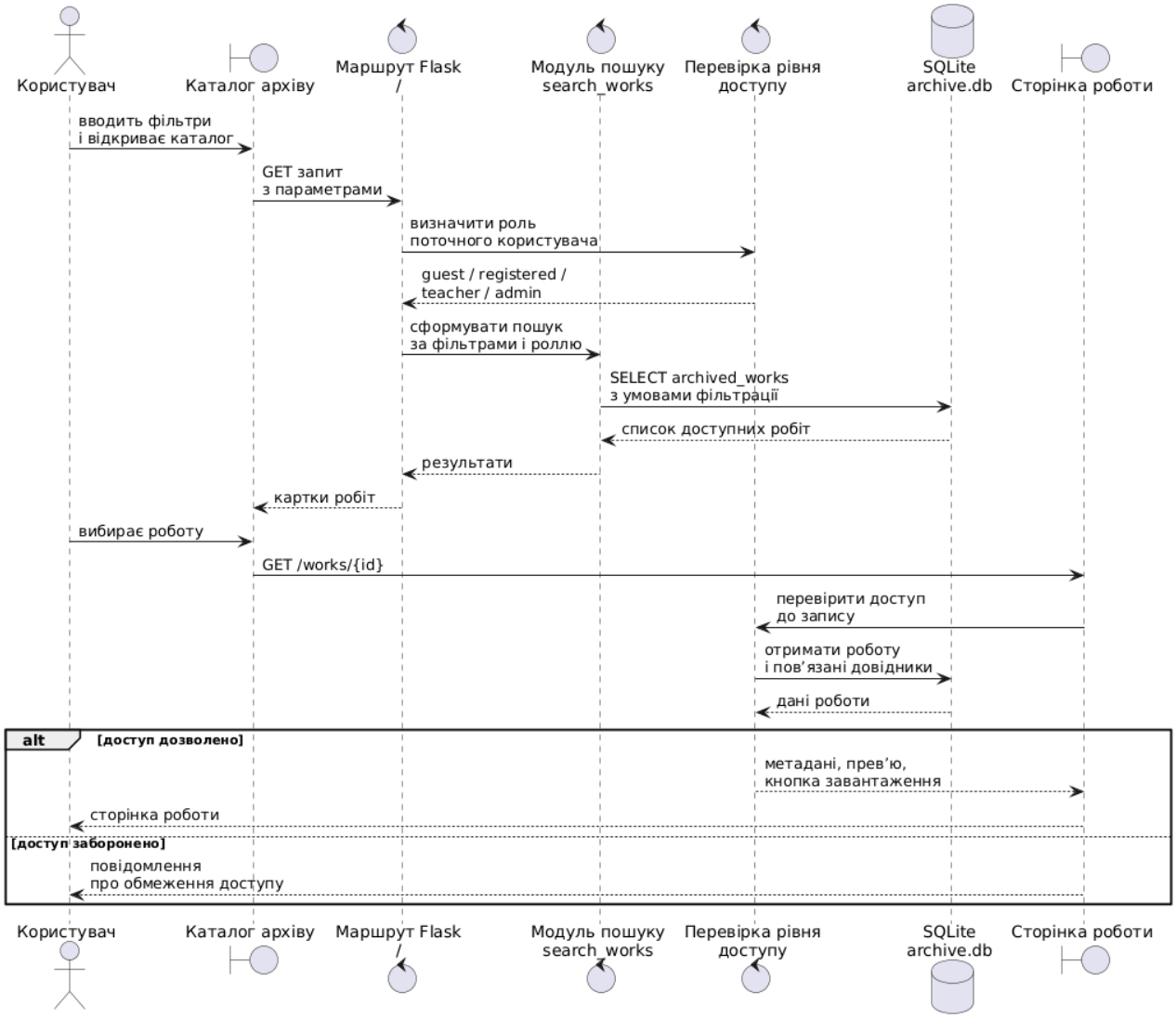

Рисунок 2.5 – UML-діаграма послідовності пошуку та перегляду архівної роботи

Система приймає параметри фільтрації, формує SQL запит, враховує поточну роль користувача та повертає лише ті записи, які доступні для перегляду. Наприклад, гість бачить тільки відкриті матеріали, зареєстрований користувач – матеріали з рівнем доступу для авторизованих користувачів, викладач – ширший набір навчальних матеріалів, а адміністратор має повний доступ. Після вибору конкретної роботи система відкриває сторінку з метаданими, прев'ю та кнопкою завантаження, якщо роль користувача дозволяє отримати файл.

Запропонована структура забезпечує зберігання студентських робіт, керування навчальними довідниками, рольове обмеження доступу, пошук за метаданими, попередній перегляд файлів і формування статистики наповнення архіву.

2.2 Вибір засобів, методів і алгоритмів вирішення поставленого завдання

Вибір засобів і алгоритмів для системи електронного архіву студентських робіт здійснювався з урахуванням потреби у вебінтерфейсі, локальному збереженні метаданих, завантаженні навчальних документів, рольовому доступі, швидкому пошуку та попередньому перегляді файлів. Для такої системи важливо поєднати просту архітектуру з достатньою гнучкістю, оскільки архів містить різні типи робіт, різні рівні доступу і декілька навчальних довідників.

Для серверної частини обрано Python і Flask. Flask є легким вебфреймворком, у якому маршрути описуються без надмірної структури, а логіка сторінок може бути безпосередньо пов'язана з шаблонами, формами й запитами до бази даних [1]. Для кваліфікаційної роботи це зручно, оскільки система має чіткий набір сторінок: головний каталог, статистику, реєстрацію, вхід, перегляд роботи, додавання й редагування запису, керування користувачами та довідниками.

Під час вибору вебтехнології розглядалися Flask, Django і FastAPI. Flask має невеликий обсяг службового коду, зручну маршрутизацію, просте підключення HTML шаблонів і можливість безпосередньої роботи з SQLite. Його особливістю є те, що частину механізмів, зокрема рольовий доступ, довідники й адміністративні сторінки, потрібно реалізовувати самостійно. Для цієї роботи це не є недоліком, оскільки дає змогу побудувати систему саме під структуру електронного архіву студентських робіт. Django має вбудовану адміністративну панель, ORM, систему користувачів і розвинену структуру проєкту, однак для компактного навчального архіву може бути надмірним і вимагати складнішої організації коду. FastAPI зручний для створення API, має автоматичну документацію та високу продуктивність, проте він більше орієнтований на API-сервіси, ніж на класичний вебзастосунок із HTML формами, шаблонами й серверним відображенням сторінок. Тому для реалізації системи обрано Flask як найбільш відповідний варіант.

Для збереження структурованих даних обрано SQLite. Вона доступна через стандартний модуль sqlite3 Python, що дає змогу створювати таблиці, виконувати параметризовані запити та працювати з локальним файлом бази без окремого сервера [2]. Для архіву студентських робіт це достатньо, оскільки система орієнтована на локальне або невелике інституційне використання, а обсяг метаданих не потребує складної серверної СКБД.

Під час вибору засобу зберігання даних порівнювалися SQLite, PostgreSQL і MySQL. SQLite має перевагу у простому розгортанні, збереженні всієї бази в одному локальному файлі, відсутності потреби в окремому сервері та стандартній підтримці в Python. Її обмеженням є те, що вона не призначена для великої кількості одночасних операцій запису в масштабній мережевій системі, однак для навчального електронного архіву це обмеження не є критичним. PostgreSQL забезпечує високу надійність, масштабованість, розвинені засоби адміністрування і розмежування доступу, але потребує окремого сервера та складнішого розгортання. MySQL також є поширеною серверною СКБД і добре підтримується вебхостингами, проте для локальної версії архіву її використання

ускладнило б встановлення й супровід системи. Тому SQLite обрано як найдоцільніший варіант для поточної реалізації, а PostgreSQL або MySQL можуть розглядатися у разі подальшого масштабування.

Для захисту облікових записів використано засоби Werkzeug. Функція generate_password_hash створює хеш пароля для збереження, а check_password_hash перевіряє введений пароль без зберігання відкритого тексту [10]. У системі це застосовано для звичайної реєстрації, створення користувачів адміністратором, активації облікового запису і скидання пароля через одноразовий токен.

Окрему увагу приділено роботі з файлами. Flask описує типовий сценарій завантаження через форму multipart, перевірку розширення, збереження файла і використання secure_filename для безпечного імені файла [11]. У розроблюваній системі допустимими форматами визначено PDF, DOC, DOCX, TXT і RTF. Для кожного завантаження формується службове ім’я, зберігається оригінальна назва, визначається формат і створюється текстове або графічне прев’ю.

Для попереднього перегляду PDF документів обрано PyMuPDF. Бібліотека дає змогу відкривати PDF, отримувати текстові фрагменти та створювати зображення сторінки, що використовується для мініатюри роботи [3]. Для DOCX документів застосовано підхід читання XML вмісту через zipfile, оскільки DOCX є ZIP пакетом із внутрішніми XML файлами. Модуль zipfile Python призначений для роботи з ZIP архівами, зокрема читання їхнього вмісту [12].

Алгоритми, використані у системі, можна поділити на алгоритми доступу, пошуку, завантаження і попереднього перегляду. Алгоритм рольового доступу базується на числових рівнях ролей guest, registered, teacher і admin. Для кожної захищеної дії система порівнює рівень поточного користувача з мінімально потрібним рівнем. Такий підхід спрощує перевірку доступу до додавання робіт, керування користувачами і редагування довідників.

Алгоритм пошуку формує SQL запит поступово. Спочатку враховується мінімальний рівень доступу до роботи, потім додаються умови за текстовим запитом, кафедрою, групою, спеціальністю, роком, курсом, типом роботи,

форматом файла та діапазоном оцінки. Після цього результати сортуються за вибраним критерієм і передаються у шаблон головної сторінки. Такий підхід дає змогу комбінувати декілька фільтрів без дублювання окремих функцій пошуку.

Алгоритм завантаження роботи починається з перевірки форми та файла. Система контролює наявність обов'язкових полів, допустиме розширення, навчальну групу, рівень доступу та числові значення курсу, року й оцінки. Після цього файл зберігається у каталозі uploads, а для запису створюються метадані в таблиці archived_works. Якщо робота редагується і користувач завантажує новий файл, старий файл і пов'язане прев'ю замінюються.

Алгоритм роботи з одноразовими посиланнями використовується для активації облікового запису і скидання пароля. Для користувача генерується випадковий токен, задається строк дії, попередні невикористані токени такого типу позначаються недійсними, а після успішного встановлення пароля токен отримує позначку використання. Це зменшує ризик повторного застосування старих посилань.

Узагальнення обраних засобів реалізації наведено в таблиці 2.1.

Таблиця 2.1 – Обрані засоби реалізації електронного архіву

Засіб	Призначення в системі
Python	основна мова реалізації серверної логіки та оброблення файлів
Flask	маршрутизація, оброблення форм, сеансів, шаблонів і сторінок вебінтерфейсу
SQLite	локальне збереження користувачів, токенів, довідників і архівних робіт
Werkzeug	хешування паролів і безпечна нормалізація імен завантажених файлів
PyMuPDF	отримання тексту з PDF і побудова графічних мініатюр
zipfile	читання внутрішньої структури DOCX документів для текстового прев'ю
HTML шаблони	формування сторінок каталогу, перегляду роботи, статистики, входу та адміністрування

Отже, для реалізації системи управління електронним архівом студентських робіт обрано Python, Flask, SQLite, Werkzeug, PyMuPDF і стандартні засоби роботи з файлами. Такий набір технологій відповідає вимогам до локального вебархіву, забезпечує контроль доступу, безпечне завантаження документів, пошук за навчальними атрибутами, попередній перегляд і можливість подальшого розширення системи.

РОЗДІЛ 3
РОЗРОБКА СИСТЕМИ УПРАВЛІННЯ ЕЛЕКТРОННИМ АРХІВОМ СТУДЕНТСЬКИХ РОБІТ

3.1 Практична реалізація об’єкта проєктування

Практична реалізація системи виконана як Flask застосунок із набором маршрутів, HTML шаблонів, локальною базою SQLite та файловими каталогами для завантажених документів і попередніх переглядів. У головному файлі app.py зосереджено створення застосунку, налаштування шляхів, опис ролей, маршрути сторінок, функції роботи з базою даних, валідацію форм і допоміжну логіку оброблення файлів.

Головну сторінку електронного архіву з фільтрами й картками робіт показано на рисунку 3.1.

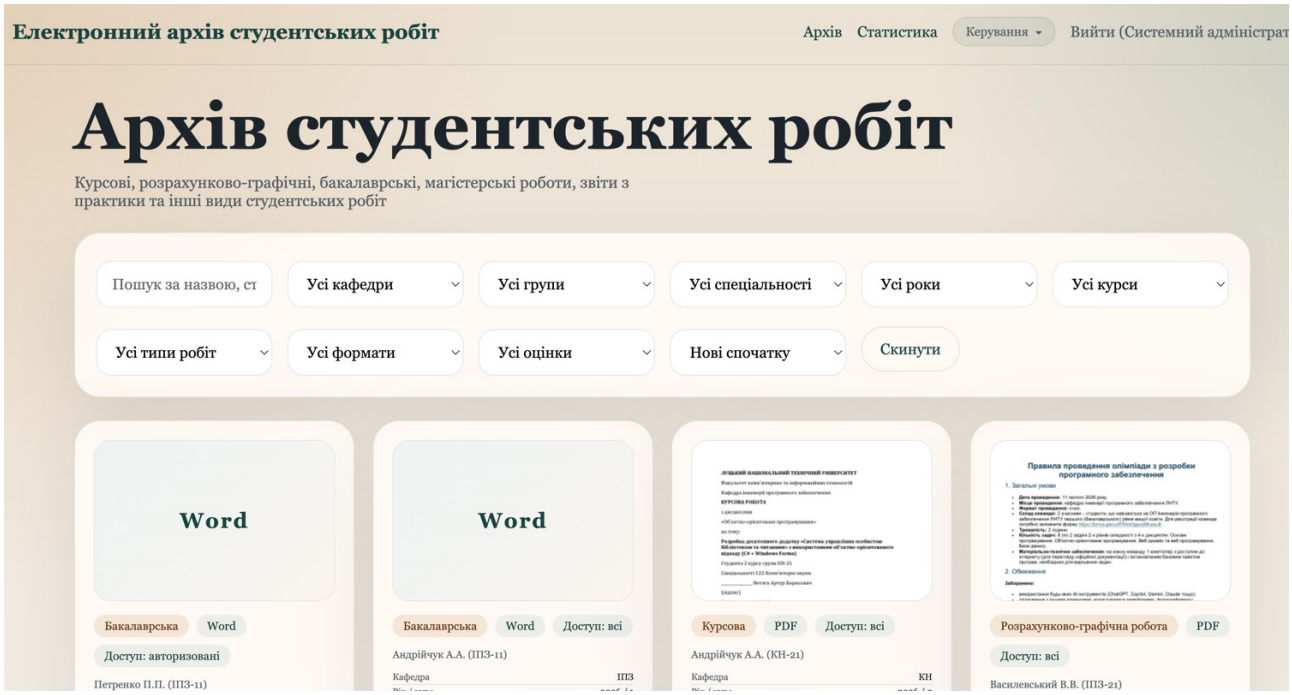

Рисунок 3.1 – Головна сторінка електронного архіву студентських робіт

Основний сценарій перегляду архіву реалізовано в маршруті index. Він зчитує параметри фільтрів із рядка запиту, передає їх у функцію пошуку, формує

доступні варіанти фільтрації та повертає HTML шаблон каталогу. Фрагмент маршруту наведено у лістингу 3.1.

Лістинг 3.1 – Маршрут головного каталогу архівних робіт

```
@app.route('/')
def index():
    filters = {
        'q': request.args.get('q', '').strip(),
        'department': request.args.get('department', '').strip(),
        'student_group': request.args.get('student_group',
'').strip(),
        'specialty': request.args.get('specialty', '').strip(),
        'study_year': request.args.get('study_year', '').strip(),
        'work_type': request.args.get('work_type', '').strip(),
        'file_format': request.args.get('file_format',
'').strip(),
        'grade_range': request.args.get('grade_range',
'').strip(),
        'sort': request.args.get('sort', 'uploaded_desc').strip(),
    }
    works = search_works(filters, current_role())
    return render_template('index.html',
works=prepare_work_cards(works),
                           filters=filters,
filter_options=get_filter_options())
```

Кінець лістингу 3.1

Інтерфейс додавання роботи складається з полів назви, ПІБ студента, групи, року, типу роботи, предмета, викладача, оцінки, рівня доступу та файла. Кафедра, спеціальність і курс не вводяться вручну, а визначаються за вибраною групою. Форму додавання роботи показано на рисунку 3.2.

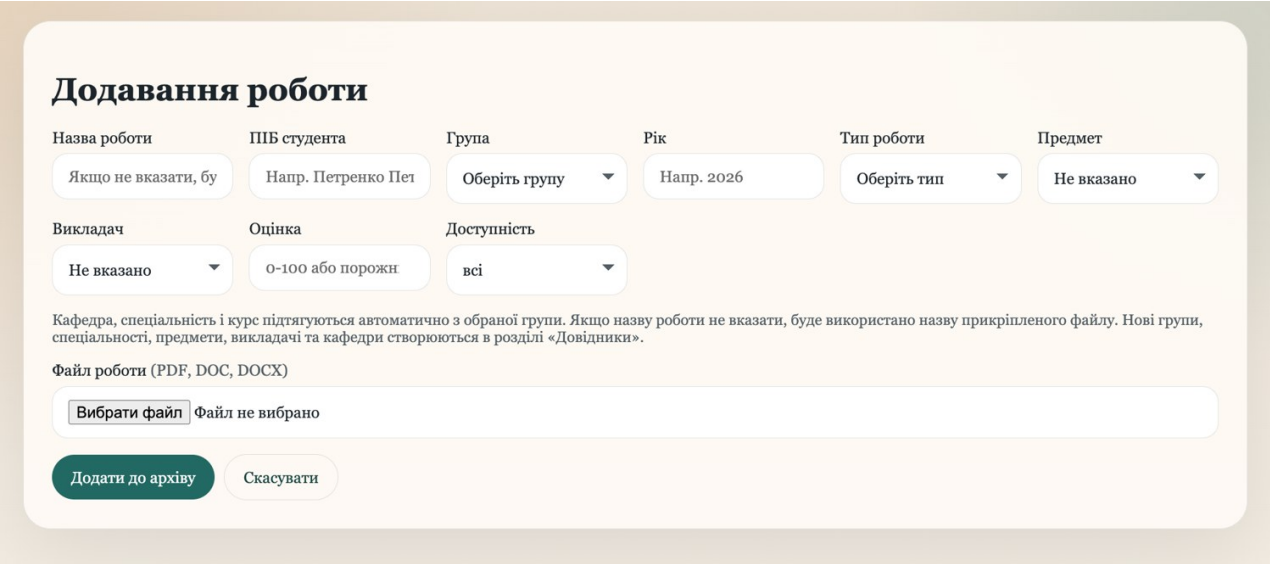

Рисунок 3.2 – Форма додавання роботи до архіву

Після надсилання форми маршрут create_work виконує валідацію, зберігає файл під безпечною унікальною назвою, отримує текстове прев'ю і додає запис у таблицю archived_works. Ключовий фрагмент реалізації додавання роботи наведено у лістингу 3.2, а розширений фрагмент оброблення файлів подано в додатку А.

Лістинг 3.2 – Додавання архівної роботи та збереження метаданих

```
@app.route('/works/new', methods=['GET', 'POST'])
@require_role('teacher')
def create_work():
    form = request.form.to_dict()
    file = request.files.get('archive_file')
    errors, group = validate_work_form(form, file, is_create=True)
    if errors:
        for error in errors:
            flash(error, 'error')
        return render_template('work_form.html', form_data=form,
                               catalogs=get_catalog_options())

    storage_name, file_format, preview_text, original_name =
save_upload(file)
```

```
    title = form.get('title', '').strip() or original_name
    execute('''INSERT INTO archived_works
               (title, student_name, department, student_group,
specialty,
                study_year, study_course, work_type, grade,
file_format,
                original_filename, stored_filename, access_level,
preview_text,
                uploader_id, uploaded_at, group_id, subject_id,
academic_teacher_id)
              VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?,
?, ?, ?, ?, ?)''',
            (title, form['student_name'].strip(),
group['department_name'],
             group['name'], group['specialty_name'],
int(form['study_year']),
             group['course'], form['work_type'],
optional_int(form.get('grade')),
             file_format, original_name, storage_name,
form['access_level'],
             preview_text, g.user['id'], utc_now(), group['id'],
             optional_int(form.get('subject_id')),
             optional_int(form.get('academic_teacher_id'))))
```

Кінець лістингу 3.2

Сторінка перегляду конкретної роботи поєднує метадані, дії завантаження, редагування або видалення та блок попереднього перегляду. Для PDF використовується вбудований об’єкт браузера, а для DOCX, TXT і RTF відображається текстовий фрагмент. Сторінку перегляду архівної роботи з PDF-прев’ю показано на рисунку 3.3.

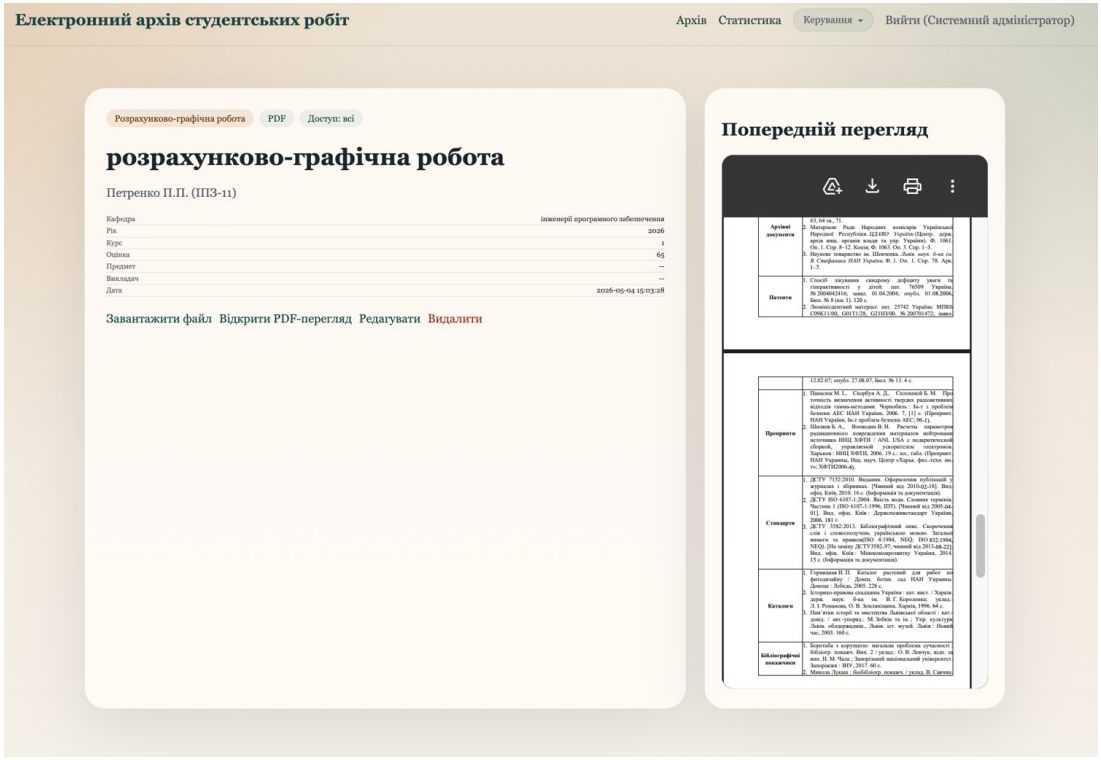

Рисунок 3.3 – Сторінка перегляду роботи з попереднім переглядом PDF

Пошук і фільтрація виконуються на рівні SQL запиту. Функція search_works формує список умов за поточними фільтрами, додає обмеження за роллю користувача та вибирає порядок сортування. Розширений фрагмент реалізації пошуку архівних робіт із фільтрами наведено в додатку Б. Приклад фільтрації робіт за типом наведено на рисунку 3.4.

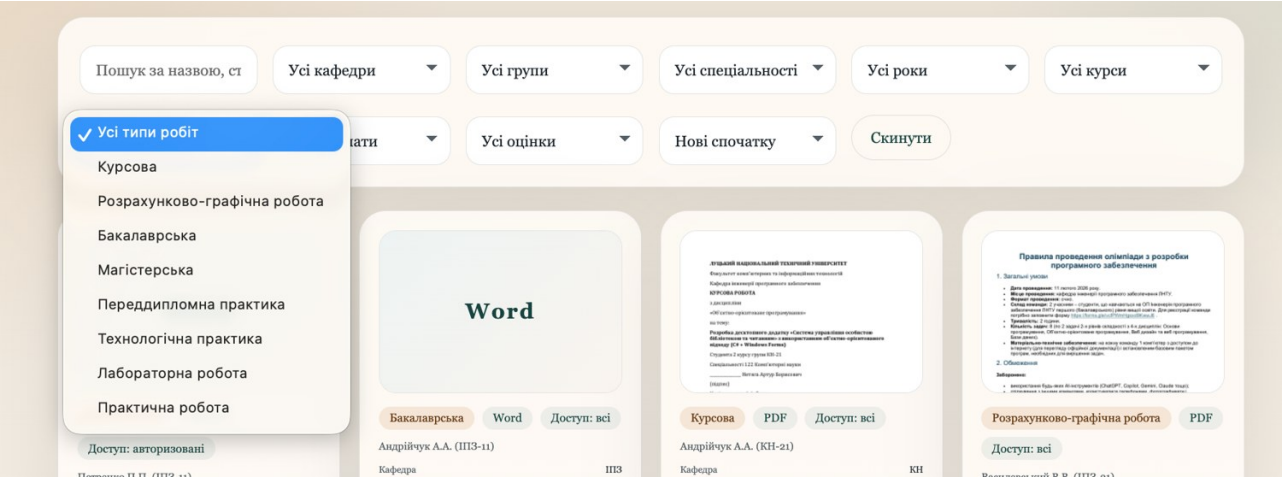

Рисунок 3.4 – Фільтрація робіт за типом у каталозі

3.2 Розробка бази даних

База даних системи реалізована у SQLite і зберігається у файлі instance/archive.db. Її структура поєднує таблиці користувачів, одноразових токенів, навчальних довідників і архівних робіт. Файли документів не записуються у базу як BLOB дані: у таблиці зберігається службова назва файла, а сам файл розміщується в каталозі static/uploads. Для PDF додатково формується зображення першої сторінки в каталозі static/previews.

Схему бази даних електронного архіву наведено на рисунку 3.5.

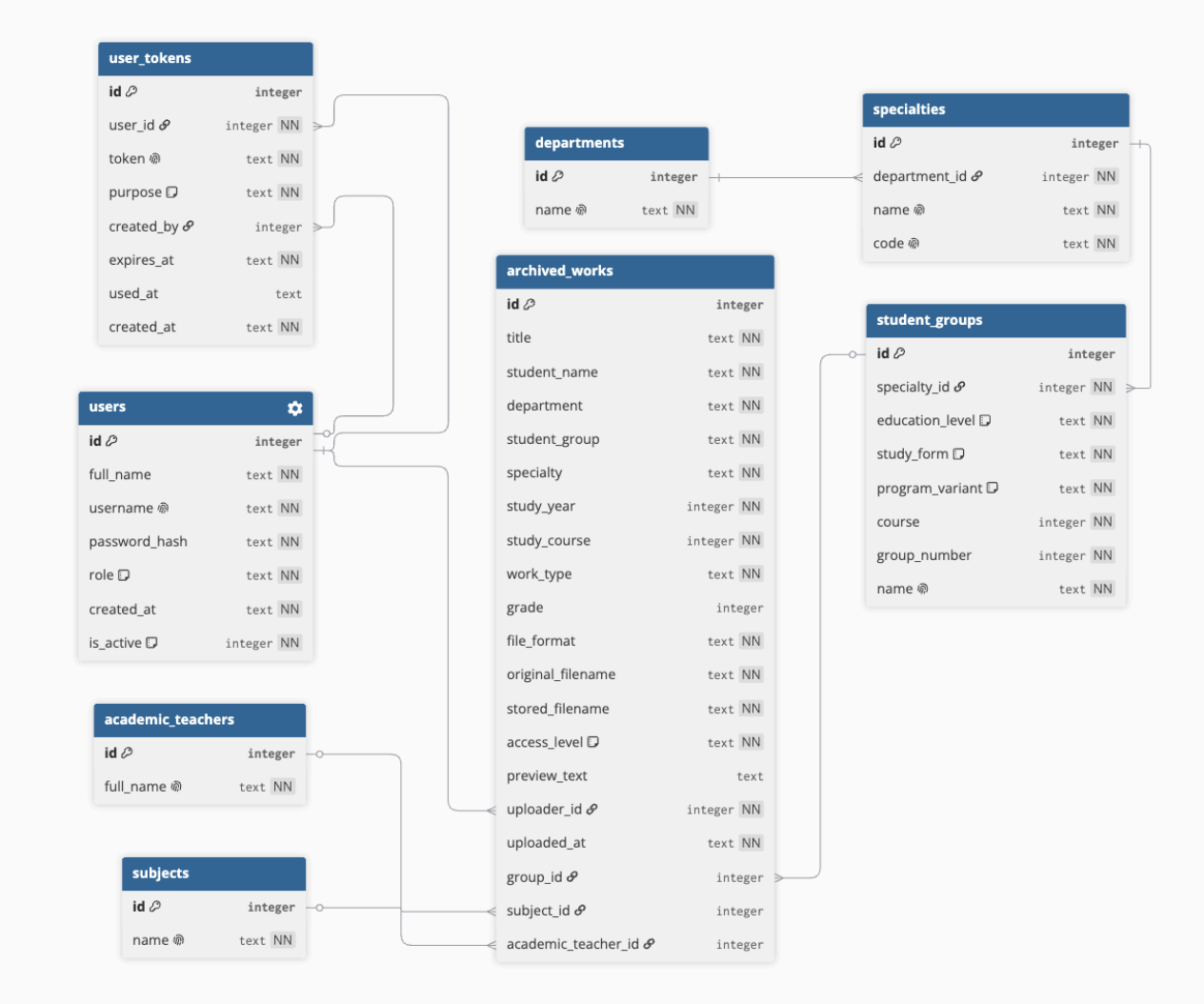

Рисунок 3.5 – Схема бази даних електронного архіву студентських робіт

Сутність users містить облікові записи користувачів, їхні ролі та ознаку активності. Таблиця user_tokens забезпечує одноразові посилання для активації або скидання пароля. Навчальні довідники представлені таблицями departments, specialties, student_groups, subjects і academic_teachers. Вони використовуються під час додавання роботи, а також у фільтрах головної сторінки.

Адміністративний інтерфейс керування довідниками показано на рисунку 3.6.

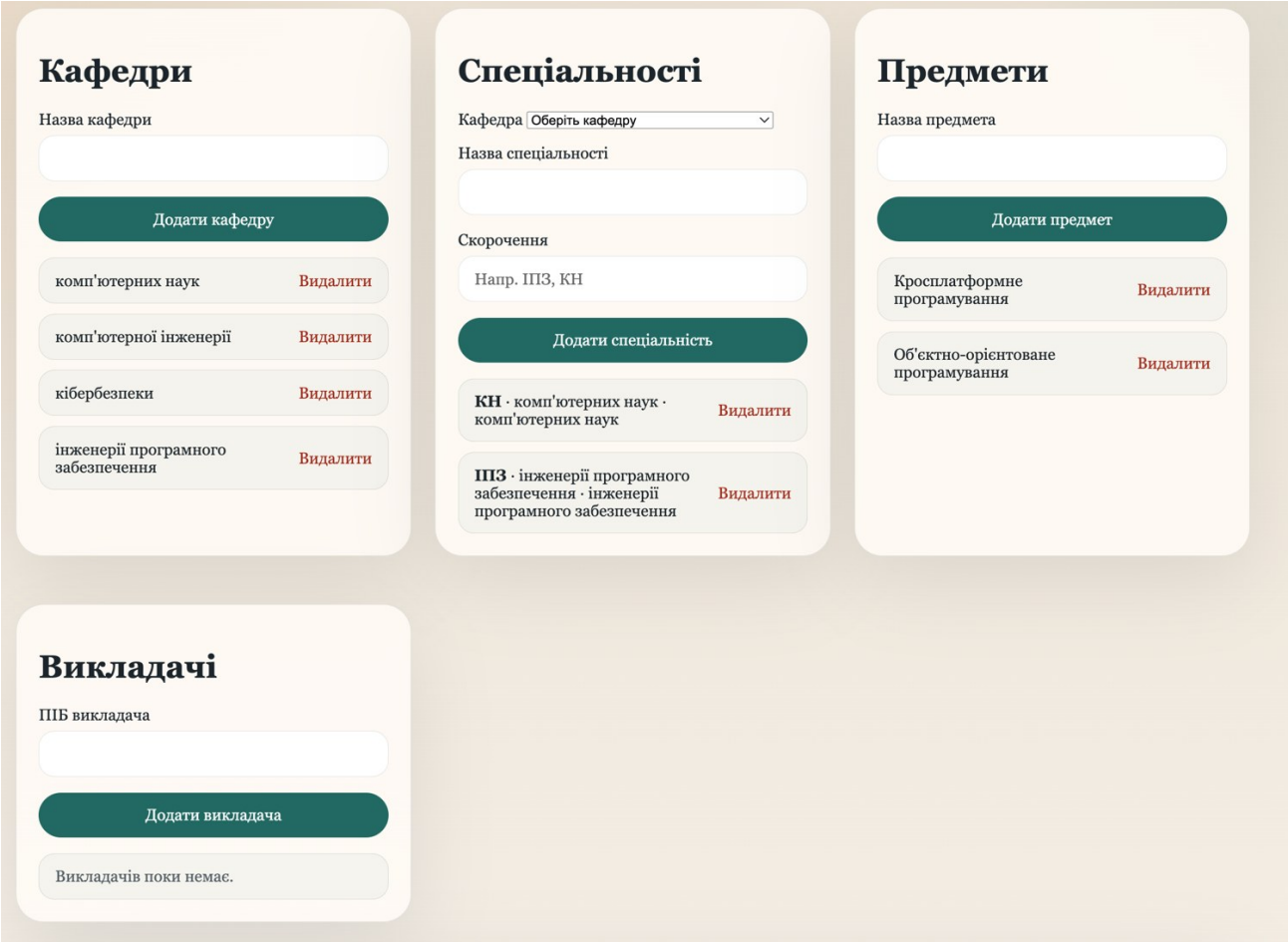

Рисунок 3.6 – Сторінка керування довідниками

Створення таблиць виконується під час ініціалізації застосунку. Фрагмент SQL структури наведено у лістингу 3.3.

Лістинг 3.3 – Ініціалізація основних таблиць SQLite

```
def init_db() -> None:
    db = sqlite3.connect(DATABASE_PATH)
    cursor = db.cursor()
    cursor.executescript('''
        CREATE TABLE IF NOT EXISTS users (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            full_name TEXT NOT NULL,
            username TEXT NOT NULL UNIQUE,
            password_hash TEXT NOT NULL,
            role TEXT NOT NULL CHECK (role IN ('registered',
'teacher', 'admin')),
            created_at TEXT NOT NULL
        );

        CREATE TABLE IF NOT EXISTS archived_works (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            title TEXT NOT NULL,
            student_name TEXT NOT NULL,
            grade INTEGER CHECK (grade IS NULL OR grade BETWEEN 0
AND 100),
            stored_filename TEXT NOT NULL,
            access_level TEXT NOT NULL CHECK (access_level IN
('guest', 'registered', 'teacher', 'admin')),
            uploader_id INTEGER NOT NULL,
            FOREIGN KEY (uploader_id) REFERENCES users(id)
        );
    ''')
    db.commit()
    db.close()
```

Кінець лістингу 3.3

Сутність archived_works містить як метадані роботи, так і службові поля для зв’язку з файлами: original_filename, stored_filename, file_format і

preview_text. Інтерфейс редагування груп і пов’язаних навчальних атрибутів показано на рисунку 3.7.

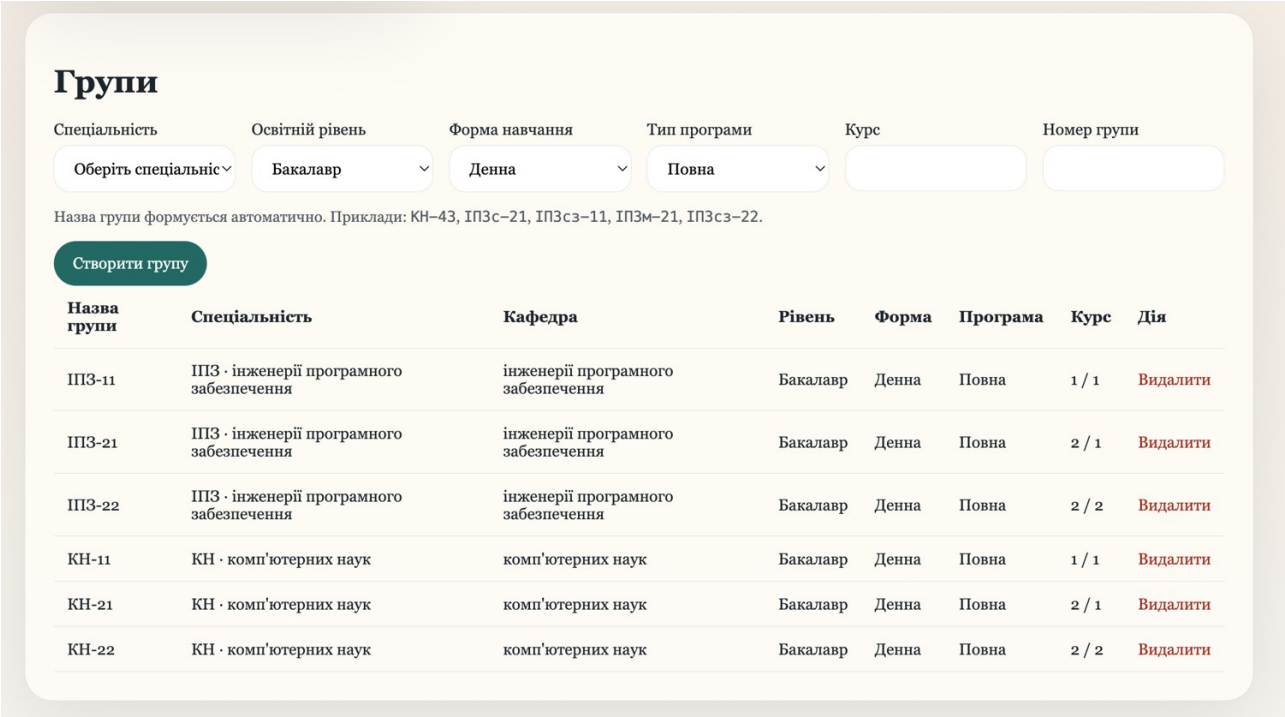

Рисунок 3.7 – Керування навчальними групами

3.3 Захист інформаційно-комп’ютерної системи

Захист інформаційно-комп’ютерної системи побудовано навколо рольового доступу, безпечного збереження паролів, одноразових токенів, обмеження типів файлів і контролю прав на зміну архівних записів.

У системі визначено чотири рівні доступу: guest, registered, teacher та admin. Для кожної роботи задається мінімальний рівень доступу, а функція ensure_access перевіряє, чи поточний користувач має право переглядати або завантажувати файл.

Керування користувачами показано на рисунку 3.8.

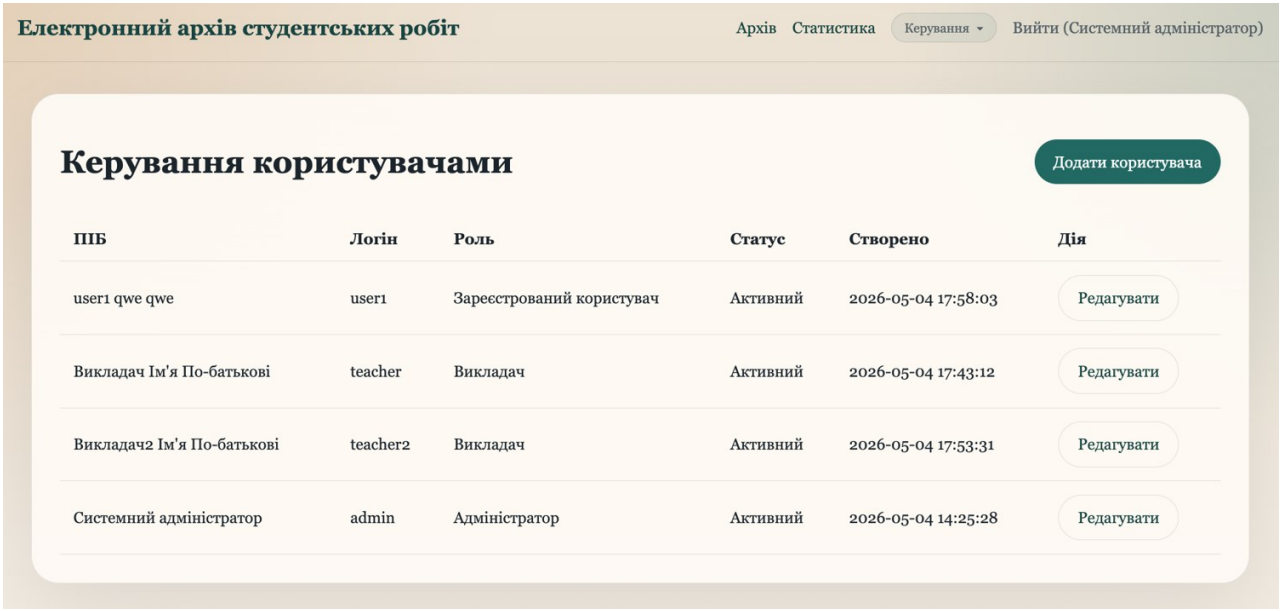

Рисунок 3.8 – Сторінка керування користувачами

Паролі не зберігаються у відкритому вигляді. Для їх створення використовується generate_password_hash, а під час входу пароль перевіряється функцією check_password_hash. Нові користувачі можуть створюватися викладачем або адміністратором як неактивні, після чого для них генерується одноразове посилання активації.

Модальне вікно редагування користувача й генерації посилання показано на рисунку 3.9.

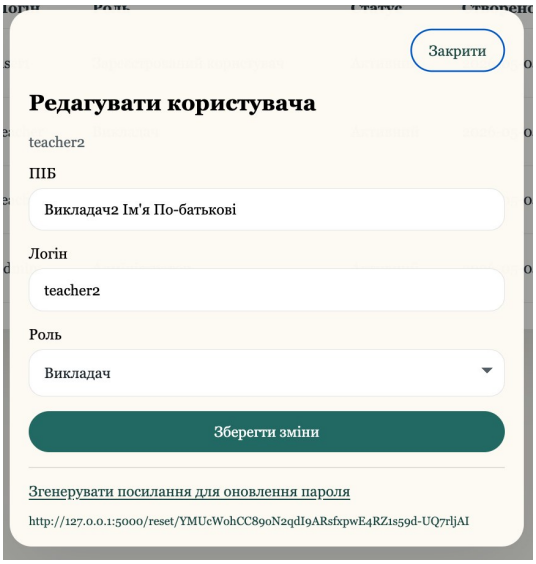

Рисунок 3.9 – Редагування користувача та службові дії з паролем

Перевірка ролі реалізована через декоратор require_role. Якщо користувач не має достатнього рівня доступу, він перенаправляється на сторінку входу. Фрагмент механізму рольового доступу наведено у лістингу 3.4.

Лістинг 3.4 – Реалізація рольового доступу в системі

```
ROLE_LEVELS = {'guest': 0, 'registered': 1, 'teacher': 2, 'admin': 3}

def current_role() -> str:
    return g.user['role'] if getattr(g, 'user', None) else 'guest'

def require_role(min_role: str):
    def decorator(view):
        @wraps(view)
        def wrapped(*args, **kwargs):
            if ROLE_LEVELS[current_role()] < ROLE_LEVELS[min_role]:
                flash('Для цієї дії потрібна авторизація з вищим рівнем доступу.', 'error')
                return redirect(url_for('login'))
            return view(*args, **kwargs)
        return wrapped
    return decorator

def ensure_access(required_role: str) -> None:
    if ROLE_LEVELS[current_role()] < ROLE_LEVELS[required_role]:
        flash('У вас немає доступу до цього файлу.', 'error')
        raise PermissionError()
```

Кінець лістингу 3.4

Завантаження файлів також має захисні обмеження. Система приймає лише PDF, DOC, DOCX, TXT і RTF, контролює максимальний розмір запиту,

нормалізує початкову назву через secure_filename і зберігає файл під UUID-іменем. Розширений фрагмент перевірки та збереження файлів наведено в додатку А.

Додатковий рівень захисту реалізований у видаленні довідників. Кафедру, спеціальність, групу, предмет або викладача не можна видалити, якщо вони вже використовуються в архівних роботах. Це захищає цілісність даних і запобігає появі архівних записів без коректних навчальних зв’язків.

3.4 Тестування та налагодження інформаційно-комп’ютерної системи

Тестування та налагодження системи виконувалося за сценаріями, які відповідають основним ролям користувачів. Для гостя перевірявся перегляд відкритих робіт, робота пошуку й фільтрів. Для зареєстрованого користувача перевірявся доступ до матеріалів із рівнем registered. Для викладача тестувалися додавання, редагування, видалення робіт, керування користувачами та генерація одноразових посилань. Для адміністратора перевірялося повне керування довідниками.

Окремо перевірялося формування статистики архіву. Сторінка статистики агрегує доступні користувачу роботи за категоріями: типи робіт, спеціальності, курси, форми навчання, роки та освітні рівні. Сторінку статистики електронного архіву показано на рисунку 3.10.

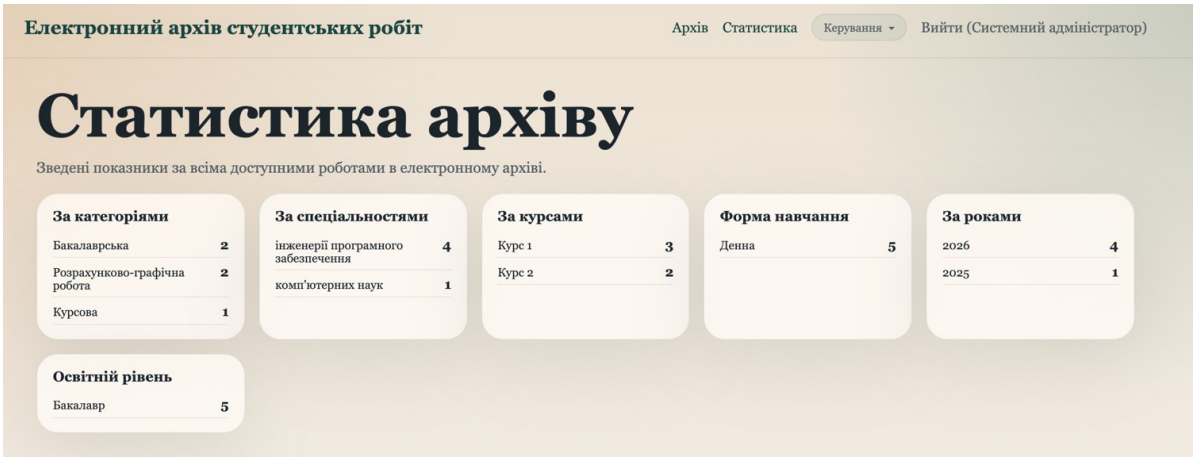

Рисунок 3.10 – Сторінка статистики електронного архіву

Під час тестування додавання роботи перевірялися обов’язкові поля, неправильний рік, некоректна оцінка, відсутність файла, спроба завантаження недопустимого формату, вибір групи, автоматичне підставлення кафедри та спеціальності, а також створення текстового або графічного попереднього перегляду.

Список груп у формі додавання роботи показано на рисунку 3.11.

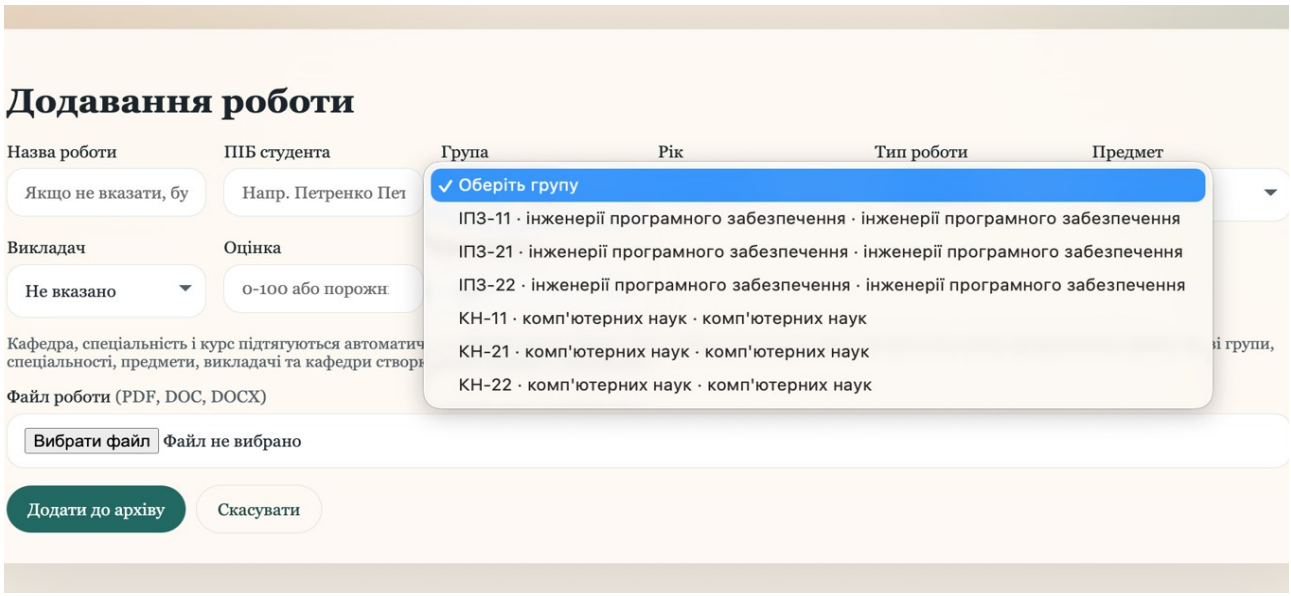

Рисунок 3.11 – Вибір навчальної групи під час додавання роботи

Для перевірки функціональних можливостей пошуку було виконано серію тестових запитів із використанням різних комбінацій параметрів. Зокрема, здійснювався пошук за текстовими фрагментами, назвою кафедри, академічною групою, спеціальністю, роком виконання роботи, курсом навчання, типом роботи, форматом файлу, а також за діапазонами отриманих оцінок. Такий підхід дозволив перевірити коректність роботи механізму фільтрації та точність відображення результатів.

Також перевірялося сортування за датою завантаження, оцінкою, роком і назвою.

Каталог після додавання декількох робіт і картки з прев’ю показано на рисунку 3.12.

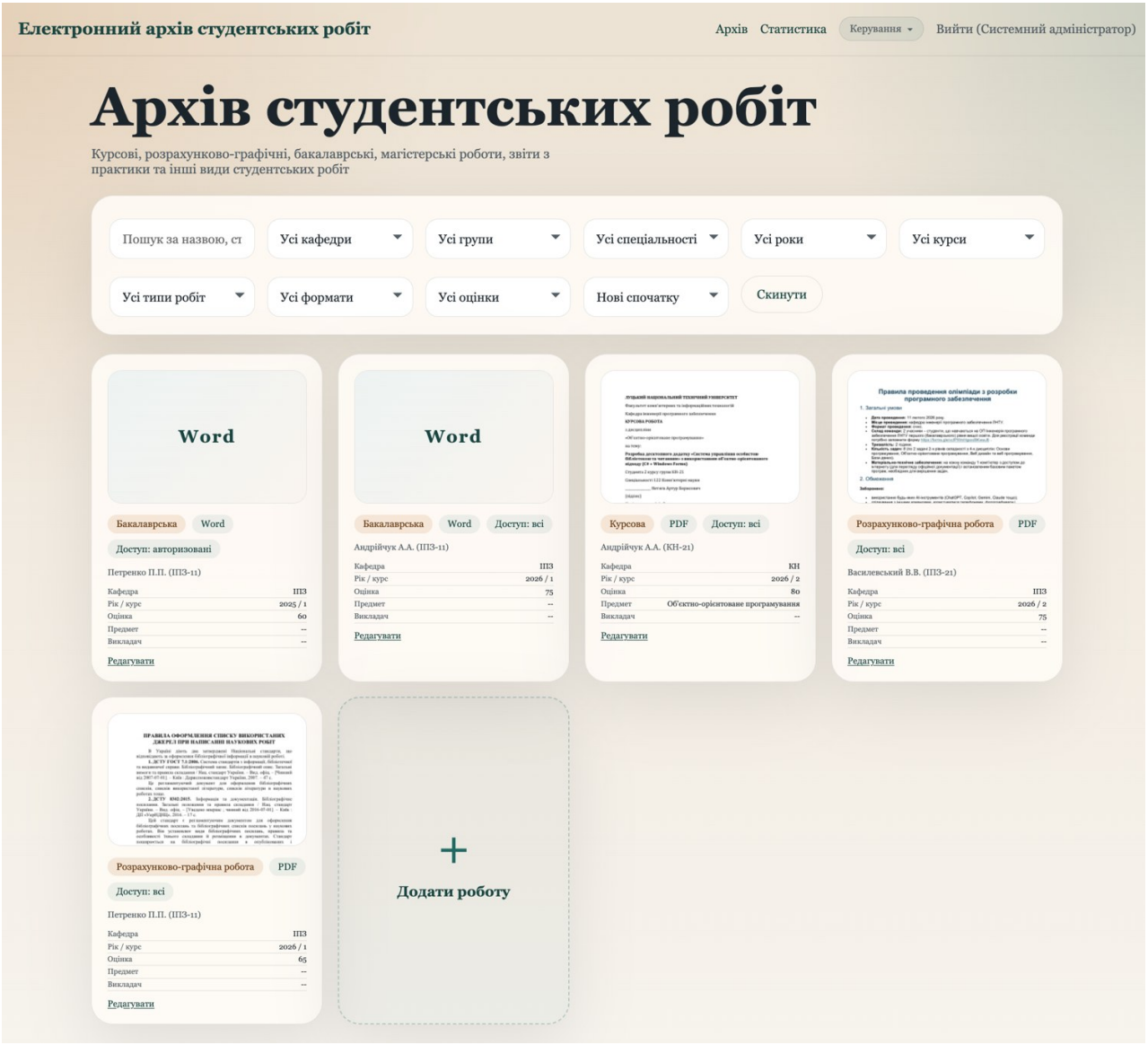

Рисунок 3.12 – Каталог після додавання робіт до архіву

Налагодження попереднього перегляду виконувалося окремо для PDF і текстових документів. Для PDF перевірялося відкриття файла у браузері та створення мініатюри першої сторінки через PyMuPDF. Для DOCX перевірялося читання word/document.xml із ZIP структури документа, для TXT – зчитування тексту з файла, для RTF – спрощене очищення службових керувальних послідовностей.

За результатами тестування встановлено, що система коректно виконує основні сценарії: реєстрацію та вхід, керування користувачами, додавання архівних робіт, пошук, фільтрацію, статистику, попередній перегляд і захист

доступу. Виявлені під час перевірок помилки форми обробляються повідомленнями, а некоректні дії не призводять до пошкодження бази даних або втрати файлів.

ВИСНОВКИ

У кваліфікаційній роботі розроблено систему управління електронним архівом студентських робіт з використанням Python, Flask, SQLite та PyMuPDF. Створений програмний продукт поєднує зберігання файлів, каталог навчальних атрибутів, рольовий доступ, пошук, фільтрацію, статистику та попередній перегляд документів.

Здійснено аналіз предметної області електронного архівування студентських робіт. Визначено основні проблеми розрізненого зберігання файлів, складності пошуку, відсутності стандартизованих метаданих і потреби в контрольованому доступі до навчальних матеріалів.

Сформовано вимоги до системи. Передбачено ролі гостя, зареєстрованого користувача, викладача й адміністратора, облік типів робіт, кафедр, спеціальностей, груп, предметів, викладачів, років навчання, курсів, оцінок, форматів файлів і рівнів доступу.

Спроєктовано структуру вебзастосунку та бази даних. Визначено таблиці користувачів, токенів, кафедр, спеціальностей, груп, предметів, викладачів і архівних робіт, а також логіку зв’язку між довідниковими даними та завантаженими файлами.

Обґрунтовано вибір технологій реалізації. Flask використано для побудови вебінтерфейсу й маршрутизації, SQLite – для локального зберігання структурованих даних, Werkzeug – для роботи з паролями та безпечними назвами файлів, PyMuPDF – для формування прев’ю PDF-документів.

Реалізовано основні модулі системи: реєстрацію та вхід користувачів, керування архівними роботами, завантаження й заміну файлів, пошук, фільтрацію, сортування, перегляд деталей роботи, завантаження файлу, PDF-прев’ю, статистику, керування користувачами та адміністрування довідників.

Перевірено працездатність ключових сценаріїв системи. Результати роботи підтверджують досягнення поставленої мети, а розроблений електронний

архів може використовуватися для впорядкованого зберігання студентських робіт і швидкого доступу до них відповідно до ролі користувача.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Quickstart. Flask Documentation. URL: https://flask.palletsprojects.com/en/stable/quickstart/ (дата звернення: 02.03.2026).

2. sqlite3 – DB API 2.0 interface for SQLite databases. Python Documentation. URL: https://docs.python.org/3/library/sqlite3.html (дата звернення: 08.03.2026).

3. Tutorial. PyMuPDF Documentation. URL: https://pymupdf.readthedocs.io/en/latest/tutorial.html (дата звернення: 15.03.2026).

4. File:Dspace.jpg. Wikimedia Commons. URL: https://commons.wikimedia.org/wiki/File:Dspace.jpg (дата звернення: 25.03.2026).

5. DSpace Home. DSpace. URL: https://dspace.org/ (дата звернення: 01.04.2026).

6. EPrints. URL: https://www.eprints.org/ (дата звернення: 07.04.2026).

7. Searches. EPrints Documentation. URL: https://wiki.eprints.org/w/Searches (дата звернення: 12.04.2026).

8. What is Figshare? Figshare Documentation. URL: https://info.figshare.com/docs/what-is-figshare/ (дата звернення: 14.04.2026).

9. Create new upload. Zenodo Help. URL: https://help.zenodo.org/docs/deposit/create-new-upload/ (дата звернення: 19.04.2026).

10. Utilities. Werkzeug Documentation. URL: https://werkzeug.palletsprojects.com/en/stable/utils/ (дата звернення: 27.04.2026).

11. Uploading Files. Flask Documentation. URL: https://flask.palletsprojects.com/en/stable/patterns/fileuploads/ (дата звернення: 12.05.2026).

12. zipfile – Work with ZIP archives. Python Documentation. URL: https://docs.python.org/3/library/zipfile.html (дата звернення: 14.05.2026).

ДОДАТКИ

Додаток А

Фрагмент оброблення завантажених файлів

```
def allowed_file(filename: str) -> bool:
    return '.' in filename and filename.rsplit('.',
1)[1].lower() in ALLOWED_EXTENSIONS

def save_upload(file) -> tuple[str, str, str, str]:
    original_name = secure_filename(file.filename)
    extension = original_name.rsplit('.', 1)[1].lower()
    storage_name = f'{uuid.uuid4().hex}.{extension}'
    destination = UPLOAD_DIR / storage_name
    file.save(destination)
    preview_text = extract_preview_text(destination,
extension)
    return storage_name, FILE_FORMATS[extension],
preview_text, original_name
```

Додаток Б

Фрагмент пошуку архівних робіт із фільтрами

```
def search_works(filters: dict[str, str], role: str) ->
list[sqlite3.Row]:
    conditions = ["? >= CASE access_level WHEN 'guest'
THEN 0 WHEN 'registered' THEN 1 WHEN 'teacher' THEN 2
ELSE 3 END"]
    params: list[Any] = [ROLE_LEVELS[role]]

    if filters['q']:
        conditions.append('(title LIKE ? OR student_name
LIKE ? OR department LIKE ? OR specialty LIKE ?)')
        pattern = f"%{filters['q']}%"
        params.extend([pattern, pattern, pattern,
pattern])

    query = f'''SELECT archived_works.*, users.full_name
AS uploader_name
                FROM archived_works
                JOIN users ON users.id =
archived_works.uploader_id
                WHERE {' AND '.join(conditions)}
                ORDER BY uploaded_at DESC'''
    return query_all(query, tuple(params))
```