Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мережних технологій факультету інформатики

Кваліфікаційна робота

освітній ступінь – бакалавр

на тему: «**Розробка системи управління нормативно-правовою базою документів університету**»

Виконав: студент 4-го року навчання,

освітньо-наукової програми

«Інженерія програмного забезпечення», 121

Шевчик Ілля Миколайович

Керівник: Глибовець А. М.

доктор технічних наук , доцент

Рецензент

_____
(прізвище та ініціали)

Кваліфікаційна робота захищена з оцінкою _____
Секретар ЕК _____
“_____” _____ 2024 р.

Київ – 2024

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мережних технологій факультету інформатики

ЗАТВЕРДЖУЮ

Зав. кафедри мережних технологій,

_____

“___” _____ 2024 р.

ЗАВДАННЯ

ДЛЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ СТУДЕНТУ

Шевчику Іллі Миколайовичу

1. Тема роботи**:** Розробка системи управління нормативно-правовою базою документів університету
   керівник роботи Глибовець Андрій Миколайович
   затверджені наказом вищого навчального закладу від «__»_____20__року №____
2. Строк подання студентом роботи _____
3. План роботи _____

4. **Календарний план виконання роботи**:

№	Назва етапу	Термін виконання	Примітка
1.	Отримання теми кваліфікаційної роботи	27.10.2023	
2.	Створення плану роботи	21.12.2023	
3.	Ознайомлення з проєктом	30.01.2024	
4.	Розробка практичної частини проєкту	23.04.2024	
5.	Написання першого розділу роботи	30.04.2024	
6.	Описання практичної частини роботи	07.05.2024	
7.	Перегляд змісту роботи з керівником	08.05.2024	
8.	Внесення змін відповідно до зауважень наукового керівника	15.05.2024	
9.	Створення презентації	17.05.2024	
10.	Захист кваліфікаційної роботи	27.05.2024	

Студент Шевчик І.М.

Керівник Глибовець А.М.

“____” _____

Зміст

Анотація

Дана робота присвячена розробці та впровадженню сервісу для управління нормативно-правовою базою університету. В роботі детально розглянуто та проаналізовано вимоги до поставленої задачі.

В другому розділі роботи описується стек використаних технологій та архітектура проєкту. Визначені основні переваги використання саме такого підходу до розробки сервісу управління документами.

В заключному розділі описується процес самої розробки. Результатом роботи є готове рішення для збереження, перегляду та редагування бази документів університету згідно з окресленими вимогами.

Вступ

У сучасному цифровому світі управління інформаційними ресурсами стає все більш важливим завданням для вищих навчальних закладів. Університети впроваджують власні сучасні розробки для оптимізації цих процесів та забезпечення ефективного управління даними. В цьому контексті розробка сервісу для керування нормативно-правовою базою університету має велике значення.

Актуальність роботи

Розвиток інформаційних технологій і зростання обсягу документообігу в університетському середовищі вимагає сучасних підходів до управління. Ця робота є актуальною через потребу університету в зручному, ефективному та безпечному інструменті для організації та збереження документів.

Мета дослідження

Метою даної роботи є розробка та впровадження сервісу для управління нормативно-правовою базою університету. Головною метою є створення ефективної системи, яка забезпечить зручний доступ до документів для всіх користувачів університетського простору, сприятиме покращенню управління документами та спростить процеси взаємодії з ними.

Розділ 1. Аналіз поставленої задачі та предметної області

1.1 Аналіз завдання

Головне завдання кваліфікаційної роботи – побудова сервісу управління нормативно-правовою базою університету. Одним з основних критеріїв є створення інтерфейсу, який дозволить додавати існуючі документи до системи, а також прикріплювати до них додатки. Цей інтерфейс має бути зручним та інтуїтивно зрозуміли, спрощуючи процеси керування базою документів. Він повинен відповідати описаним в цьому розділі вимогам.

1.2 Типи документів та опис вимог для них

Університетська база документів є досить масштабною і об’ємною, тому для зручного доступу до них відбувається сегрегація на такі три основні логічні категорії [1]:

- Внутрішні
- Вхідні
- Вихідні

В свою чергу документи з внутрішнього типу поділяються на наступні підтипи:

- Нормативно-розпорядчі
  - Розпорядження
  - Доручення
  - Протокол
  - Наказ по контролю та діловодству
  - Наказ про відрядження
  - Наказ студентського відділу кадрів
  - Наказ відділу кадрів та роботи з персоналом
  - Наказ по аспірантурі
  - Наказ по докторантурі
- Службові
- Інші

Таким чином ми маємо 13 основних типів документів, які потрібно вміти оброблювати.

1.2.1 Детальний опис мета-інформації документів типу «Внутрішні» → «Нормативно-розпорядчі»

Тип документу - обов’язкове поле, набуває значень одного з підсписку Нормативно-Розпорядчих типів документу

Ідентифікатор - обов’язкове поле, що видається системою

Назва - обов’язкове поле, що має обмеження на довжину в максимум 150 символів, та мінімально в 5 символів

Номер - необов’язкове поле і має наступний формат:

- Порядковий номер для наступних типів документів: Протокол, Доручення, Розпорядження, Наказ по контролю та діловодству
- Формат «порядковий номер + прочерк + буква». Приклади
  Наказ про відрядження – 1-в, 12-в, 35-в
  Наказ студентського відділу кадрів – 3-с, 94-с
  Наказ відділу кадрів та роботи з персоналом – 1-к, 2-к, 17-к
  Наказ по аспірантурі – 1-а, 22-а, 31-а
  Наказ по докторантурі – 3-д, 12-д, 30-д

Дата підписання - обов’язкове поле у форматі дд.мм.рррр

Дата початку чинності - обов’язкове поле у форматі дд.мм.рррр. За замовчуванням дорівнює даті підписання

Дата кінцевого терміну виконання – необов’язкове поле у форматі дд.мм.рррр

Автор – необов’язкове поле, відповідає за людину що створила даний документ

Виконавці – зазначаються люди, що будуть виконувати цей наказ. І один з них позначається як головний виконавець.

Узгодження – люди, що мають узгодити документ, до того як він потрапить кореспонденту на підписання. Має бути підтримка черги узгоджень

Кореспондент – одна або більше людей, що підписали документ.

Доступність – конкретні люди, або групи людей що мають доступ до документа

Чинність – обов’язкове поле, документ автоматично набуває чинності з вказаної дати підписання. Також документ стає нечинним, якщо видалась нова версія або чинність було скасовано іншим документом

Категорія – необов’язкове поле, що позначає приналежність документа до певної категорії

Опис – необов’язкове поле, з обмеженням на максимальну довжину в 500 символів.

Додатки – частина поточного документа, що немає власного номера. І містить наступні обов’язкові поля:

- Тип додатку (один з наступних):
  1. Статут
  2. Положення
  3. Правила
  4. Додатки до правил
  5. Інструкція
  6. Рекомендації (методичні)
  7. Програма
  8. Договір

9. Ухвала
10. Інші документи

- Назва – обов’язкове поле додатку з обмеженням на мінімальну довжину в 5 символів
- Чинність – обов’язкове поле у форматі «Чинний» чи «Нечинний». Поле є незалежним від чинності документу, оскільки може окремо втрачати чи набувати чинності

Зауваження стосовно типів додатків: перераховані загальні типи додатків. Ніякі користувачі системи не мають змоги додавати нові типи, вони задаються лише в коді, щоб уникнути хаосу в класифікації і дублювання типів. Завчасно пропонується виокремити документи з відсутніми типами в тип «Інші», щоб не зупиняти процес завантаження даних у базу. Після додавання відсутніх зараз типів, вже додані додатки документів можна буде оновити змінивши тип «Інші» на відповідний для додатка тип.

1.2.2 Детальний опис мета-інформації документів типу «Внутрішні» → «Службові» та «Внутрішні» → «Інші»:

Тип документу - обов’язкове поле, набуває значень одного з наступних: Службові документи, Інші внутрішні документи.

Номер - необов’язкове поле, у форматі «порядковий номер + прочерк + в», приклад: 122-в

Наступні поля мають аналогічний опис, як в попередньому типі документів:

Ідентифікатор, Назва, Дата підписання, Дата початку чинності , Дата кінцевого терміну виконання, Автор, Виконавці, Узгодження, Кореспондент, Доступність, Чинність, Категорія, Опис.

Додатки – аналогічно до додатків з Нормативно-Розпорядчих документів, тільки тип додатків обмежений, а саме: Акт, Інші документи

1.2.3 Детальний опис мета-інформації документів типу «Вхідні» та «Вихідні»:

Дата підписання – обов’язкове поле у випадку вхідних документів є фактично «Датою отримання», у випадку вихідних документів є фактично «Датою відправки»

Номер – необов’язкове поле. На даному етапі в системі зберігатимуться документи глобальних типів, тому формат номеру наперед не визначений. Коли будуть додані підтипи для вхідних та вихідних документів буде також змінений формат відповідно до підтипу.

Додатки - аналогічно до додатків з Нормативно-Розпорядчих документів, тільки тип додатків обмежений до типу «Інші документи»

1.3 Опис бізнес процесу додавання вже існуючих документів

Система має надавати змогу заповнити всі поля документів у відповідності до встановлений критеріїв. В інтерфейсі передбачено зберігання вже підписаних та узгоджених документів, тому необхідно забезпечити зберігання інформації про всіх людей, що підписали або узгодили його. Документи зберігаються в загальній базі документів без необхідності проходити процедури узгодження та підписання.

Основний функціонал при заповненні документів на клієнтській стороні системи [2]:

- Заповнення та перевірка на валідність у відповідності до формату мета-інформації про документи
- Можливість пошуку вже наявних документів для встановлення зв’язку між ними
- Пошук та додавання категорій до документа
- Управління доступом до документа та обов’язками користувачів
- Можливість додавання додатків

1.3.1 Процес додавання додатків

При додавання додатка до системи вибирається тип додатку з доступного переліку для даного типу документів. Якщо ж відсутній потрібний тип, користувач може вибрати опцію «Інші». Чинність додатку встановлюється за замовченням такою ж як і чинність документа, проте може бути змінена за потреби[2].

Для додатків типу «Ухвала» має бути можливість додавання погоджень. Погодження – це сутність, що має в собі наступну мета-інформацію:

- Номер погодження (обов’язкове поле)
- Назва погодження (обов’язкове поле)
- Дата (обов’язкове поле)
- Файл (може бути відсутнім)

1.4 Додаткові сутності пов’язані з документами

1.4.1 Опис груп користувачів

Для забезпечення зручного доступу до документів для великої кількості користувачів пропонується створювати групи користувачів. Кожна така група може мати змогу переглядати документи певних типів.

Основні характеристики групи:

- Наявність поля «Назва групи»
- Можливість додавати користувачів до групи або видаляти їх звідти
- Створення, редагування та видалення групи може виконуватись лише адміністратором
- При видаленні групи зв’язок між користувачем і групою повинен видалятись. Так само видаляється і зв’язок між групою і типом документ до якого відкрити доступ

В системі мають бути закладені дві початкові групи: «Публічний доступ» та «Всі працівники».

1.4.2 Опис категорій документів

Одним з завдань даної роботи буде створення категорій документів, що допоможе краще структурувати нормативно-правову базу університету. Так кожен документ може належати кільком категоріям або не належати жодній. В свою чергу створені категорії можуть мати кілька документів або не мати жодного. Категорія має лише одне мета поле – назва категорії.

Всі дії з категоріями, а саме створення, редагування та видалення можуть виконуватись лише адміністратором, а перегляд категорій буде доступний всім користувачам, що мають право на пошук, створення, редагування та видалення документів.

1.5 Опис процесу видалення документів

Для вирішення поставленої даної задачі буде застосовуватись метод «м’якого» видалення*(soft deletion)*, також відомий як «видалення з позначкою». Основна концепція полягає в тому, що дані не видаляються повністю з бази даних, а замість цього вони позначаються як видалені за допомогою булевого поля. Зважаючи на те, що ми маємо справу з досить чутливою для університету інформацією, нам необхідно вміти зберігати дані таким чином, щоб уникнути випадкового видалення. Крім того, завдяки «м’якому» видаленню ми можемо легко відновлювати дані і забезпечувати їх цілісність, зокрема зв’язки між документами і додатками.

Під час видалення документа важливо виконати наступні кроки:

- Позначити всі додатки цього документа як видалені.
- Позначити всі погодження додатків як видалені.

- Видалити зв’язок між документами, якщо він існує. Оскільки документи мають зв’язок «один до багатьох», то може виникнути блокування додавання нових зв’язків після видалення документа.

Повне видалення документів та додатків на даному етапі не буде розглядатись, хіба що може бути реалізоване згодом за необхідності.

1.6 Підсумок поставленого завдання

Отже, описані вище вимог визначають необхідність розробки системи, котра міститиме в собі такі основні моделі, як документи, додатки до документів, погодження додатків, групи користувачів(а також виокремити виконавців, кореспондентів та тих, хто узгоджуватиме документи), категорії. Визначити зв'язки між сутностями та сформувати процес безпечного видалення. Потрібно забезпечити збереження файлів документів у відповідних сервісах, а також можливість їх розпізнавання та індексації.

Розділ 2 Структура проєкту та опис використаних технологій

2.1 Архітектура проєкту

Завдання даної роботи полягає в розробці та впровадженні сервісу в рамках екосистеми SmartUkma. Цей проєкт організований у вигляді мікросервісної архітектури та складається з кількох модулів. Основні сервіси з якими доведеться взаємодіяти під час розробки:

- ui-server: модуль, що відповідає за клієнтську сторону;
- document-management: сервіс, відповідальний за взаємодію з документами
- user-info: компонент, що зберігає інформацію про користувачів системи та дані, що стосуються навчання в університеті
- account-server: модуль для авторизації користувачів та інших сервісів в системі

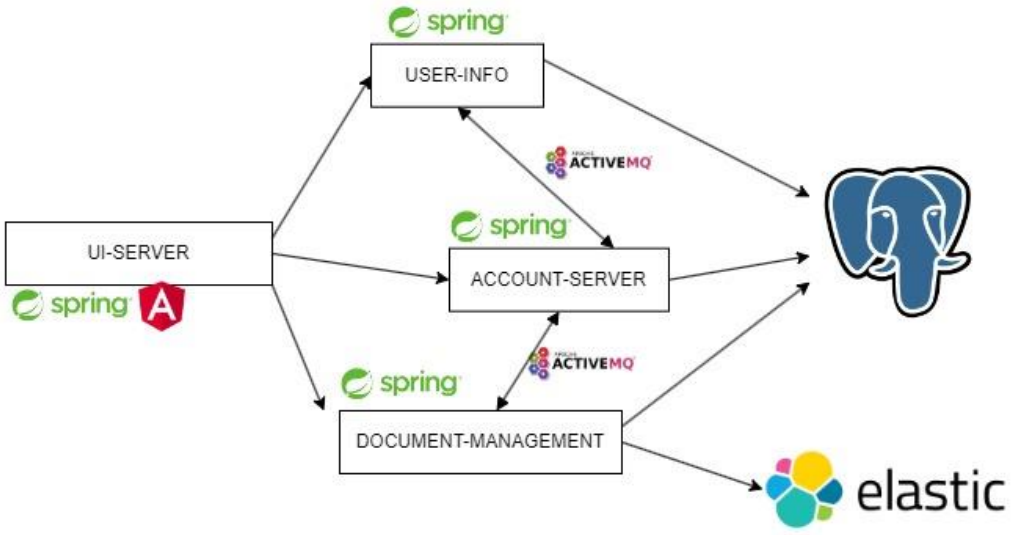


Рисунок 2.1 Діаграма архітектури сервісів

Таких підхід до розробки, розбиття логіки на декілька незалежних компонент, є цілком доцільним, адже допомагає досягти наступних переваг[3]:

- Незалежна розробка. Кожен програміст може працювати без прив’язки до інших, що значно пришвидшує процес розробки.
- Організація кодової бази та структуризація проєкту. За конкретні операції відповідає окремі частини додатку.
- Легкість в масштабуванні.
- Швидкість впровадження нового функціоналу. Мікросервісний підхід дозволяє зменшити час, потрібний для випуску оновлення.

Незважаючи на ці переваги, мікросервісна архітектура має і певні недоліки, одним з яких буде ускладнений процес розгортання застосунку. Чим більше маємо компонентів в програмі, тим важче ними керувати. Проте існують рішення здатні полегшити цей процес.

Docker – інструментарій для управління ізольованими контейнерами. Дозволяє легко «упакувати» програму з усіма її залежностями в контейнер, який можна легко запускати в будь-якому оточенні. Самі розробники цієї технології кажуть, що вони беруть на себе все налаштування, дозволяючи нам зосередитись на процесі написання коду.

Проте що робити, коли таки контейнерів треба запускати багато, можливо 5, 10 а то і більше. На допомогу приходить docker-compose – інструмент для запуску багатоконтейнерних застосунків. За допомогою конфігураційного файлу docker-compose.yaml, в якому вказано, як контейнери мають запускатись, спілкуватись між собою та використовувати ресурси. Docker compose дозволяє з автоматизованим процесом запускати цілий комплекс служб однією командою. Це робить процес розгортання складних застосунків простішим і більш керованим.

Звичайно лишень цими інструментами не обійтись і для зручного масштабування сервісів бажано також використовувати оркестратори контейнерів. Проте для локальної розробки де нам не так важливий

горизонтальний скейл проєкту, docker compose значно облегшує роботу. Приклад розгортання необхідних сервісів наведено в додатку А.

Збереження кодової бази відбувається за рахунок використання GitLab, платформа, що позиціонує себе не тільки, як інструмент для спільної розробки програмного забезпечення, а як повноцінний DevSecOps (розробка, безпека, операції), що є розширенням поняття DevOps. Для кожного мікросервіса налаштований свій пайплайн CI/CD, котрий дозволяє автоматично збирати проєкту та проводити його тестування з подальшим розгортанням. У випадку таких сервісів як document-management-service він дозволяє завантажувати нову його версію в maven репозиторій університету.

2.2 Збереження даних

2.2.1 Діаграма сутностей та відношень

Для проєктування бази даних сервісу було створено ER діаграму(додаток Б). На ній відображено всі сутності та відношення між ними. Так, основною сутністю в діаграмі виступають документи. І вони мають циклічний зв’язок один до багатьох, оскільки документ може оновлювати інші документи, проте сам документ може бути оновлений лише одним документом. Зв’язок є необов’язковим з обох сторін.

Додатки залежать від документів відношенням один до багатьох, при цьому зв’язок є необов’язковим зі сторони документа, адже документ може не мати додатків, проте кожен додаток повинен бути прив’язаний до певного документа. Погодження в свою чергу пов’язані з додатками таким самим зв’язком, оскільки лише додатки типу «Ухвала» можуть містити погодження і при цьому не обов’язково, але погодження обов’язково має належати до додатку.

Під час проєктування користувачів, що будуть взаємодіяти з документами було прийнято рішення розділити їх на окремі сутності, що відповідають виконавцям, кореспондентам, тим хто повинен ознайомитись чи узгодити

документ. Даний підхід дозволяє позбутись зайвих полів для окремих користувачів. Так, наприклад, користувач, що має ознайомитись з документами не потребує інформації про те чи є він головним виконавцем цього документа. Зв'язок між користувачами і документами виступає один до одного і зв'язок необов'язковий зі сторони документа. Таке рішення зумовлено тим, що документ може не мати певну групу користувачів.

Категорії пов'язані з документами зв'язком багато до багатьох (M до N) і він є необов'язковим з обох сторін, оскільки документ може не мати категорій, а категорія може бути порожньою.

2.2.2 Використання Postgresql

Для збереження даних була обрана система керування баз даних Postgresql. В цілому вибір реляційної бази даних зумовлений тим, що ми маємо чітко визначену структуру сутностей та відношень між ними. Окрім того дане рішення є найбільш вживаним і дає нам наступні переваги[5]:

- Простота: можливості sql і те, як дані зберігаються, дозволяються нам досить легко отримувати потрібну інформацію, виконувати складні запити, ефективно здійснювати пошук.
- Цілісність даних: реляційні бази даних надають механізми для забезпечення цілісності даних, зв'язків та інше. В будь-який момент часу збережена інформація відповідатиме заданим обмеженням та формі.
- Транзакцій: дозволяють групувати декілька дій в один блок, забезпечуючи консистетність даних в разі виникнення помилок та атомарність операції

З-поміж багатьох різних реляційних баз даних був обраний саме postgres через свою перевагу в швидкості операцій типу read-write в порівнянні з конкурентами, а отже він є більш вживаним, коли мова іде про роботу з великими об'ємами даних і складними запитами. На додачу, він був з самого початку створювався для сумісності з ACID і є оптимальним рішенням при конкурентних

транзакціях. Загалом postgres має більше підтримуваних функцій «з коробки», кращу вибірку типів даних, кращу підтримку роботи з процедурами та інше.

2.2.3 Використання Elasticsearch

Однією з поставлених задач даної роботи є можливість пошуку документів за ключовими словами в них. Оскільки документи можуть бути двох типів: у вигляді тексту і у вигляді зображень, нам потрібно вміти оброблювати розпізнавані документи для подальшого їх пошуку. З цією задачею чудово впорається elasticsearch – пошуковий двигун, розроблений на базі Apache Lucene і написаний на Java[4]. Основні можливості цього інструменту:

- Повнотекстовий пошук
- Збереження даних в безструктурному форматі(JSON-форматі)
- Сортування та фільтрація результатів
- Масштабованість та відмово стійкість

Одним з конкурентів Elasticsearch може виступати PostgreSQL, який також надає можливості повнотекстового пошуку. Дане рішення виглядає досить спокусливим, адже полегшує процес налаштування та конфігурації інфраструктури. Проте краще зосередитись на використанні Elasticsearch, оскільки він оптимізований під пошук даних(використання розподіленої архітектури та інвертованих індексів), краще масштабується горизонтально, в той час як postgres традиційно орієнтований на вертикальне масштабування.

Основною одиницею збережених даних в elasticsearch є документи. В нашому випадку кожен документ зберігатиме інформацію про те, якого типу є текст, чи це додаток чи це документ. Зберігатиметься унікальний ідентифікатор для подальшого пошуку сутності в базі даних і, звичайно, зберігається текст який буде індексований для пошуку.

2.3 Серверна частина застосунку

Серверна частина застосунку SmartUkma розробляється на Java з використанням фреймворку Spring Boot. Це одне з найпопулярніших ринкових рішень для того, щоб швидко створити свій веб-сервіс. Зручність фреймворку зумовлена тим, що позбавляє програміста зайвої конфігурації проєкту. З використанням простого Spring потрібно було проводити багато налаштовувань перед тим, як приступити до розробки самого продукту. За допомогою Spring boot стало легше керувати залежностями: тепер, якщо потрібно використати певну технологію, достатньо імпортувати лише один модуль, який автоматично підтягне необхідні для роботи залежності. Наприклад, з імпортом модуля spring-boot-elasticsearch імпортуються 9 інших модулів і при цьому відбувається автоматична їх конфігурація.

2.3.1 API специфікація

Взаємодія з сервером відбувається за допомогою Rest API. Опис всіх необхідних ресурсів відбувається в декларативний спосіб(додаток В), а саме структура запиту: параметри, тіло запиту, формат відповіді тощо. Це дозволяє отримати наступні переваги:

1. Автоматизація кодування та однорідний код: за допомогою openapi генератора зникає необхідність в ручному прописуванні ресурсів. Також всі частини згенерованого коду будуть відповідати єдиному стандарту.
2. Зменшення ризику помилок: генерація коду на основі специфікації дозволяє уникнути помилок при ручному написанні коду. На власному досвіді не один раз стикаєшся з тим, що десь неправильно зазначається метод чи описується ендпоінт, що тягне за собою подальші неприємності.
3. Швидка інтеграція: дозволяє швидко і легко інтегрувати API з вже існуючими системами, що забезпечує більшу надійність і стабільність застосунку.

Якщо виникає необхідність розширити функціонал згенерованих контроллерів, то це легко можна зробити за допомогою наслідування такого контроллера і перевизначені потрібного метода. Всі переваги contract-first підходу зберігаються, адже ви не змінюєте сигнатуру методу, натомість взаємодієте з наданою.

В рамках розробки ми будемо використовувати наступний список контрактів:

- Document: визначені CRUD операції для взаємодії з документами, також визначена фільтрації за параметрами
- Addition: визначені CRUD операції для взаємодії з додатками, присутнє читання додатків за ID документа
- Coordination: визначені CRUD операції для взаємодії з погодженнями ухвал
- Category: контракт для роботи з сутністю категорій
- UserGroupDocumentType: допомагає встановити зв’язки між типами документів та групами користувачів

Список всієї специфікації API, що використовується при написанні практичної частини додана в додатку Ґ.

Також зазначимо, що аналогічні контракти прописуються зі сторони клієнта. Використовуючи один і той самий контракт ми маємо згенеровані їх реалізації і під Java і на Typescript, що значно спрощує і пришвидшує процес розробки і інтеграції нового API. Для генерації Java коду використовується maven плагін openapi-generator-maven-plugin, а для генерації на стороні клієнта використовується npm пакет openapi-generator-cli

2.3.2 Використання Hibernate

Hibernate – інструмент, що використовується для об’єктно-реляційного відображення в Java. Даний фреймворк дозволяє відображати

класи Java на відповідні таблиці бази даних. За допомогою нього можна легко визначити сутності з якими доведеться взаємодіяти та налаштувати зв'язки між ними. Зменшення бойлерплейт коду, оптимізація написання запитів до бази даних та взаємодії із звичайним JDBC, управління транзакціями, кешування - це все надає змогу зосередитись програмісту на написання бізнес-логіки і підвищує його продуктивність.

```
@OneToMany(mappedBy = "document")
@Where(clause = "deleted = false")
private Set<AdditionEntity> additions = new HashSet<>();

@OneToMany(mappedBy = "document")
@Where(clause = "deleted = true")
private Set<AdditionEntity> deletedAdditions = new HashSet<>();
```

Рисунок 2.2. Зв'язок один до багатьох з фільтрацією

Розглянемо на прикладі взаємодії між документами та додатками, де Hibernate реалізувати відносини з фільтрацією даних. Діаграма визначених сутностей за допомогою Hibernate зображена в додатку Г.

Однак, для ефективного використання Hibernate необхідно мати глибоке розуміння процесів, що відбуваються «під капотом». Інакше можна потрапляти в ситуації з циклічним отриманням даних при спробі побудові двостороннього зв'язку, або отримати непередбачувані результати виконання запитів тощо.

2.4 Опис клієнтської частини

На даний момент існує багато популярних фреймворків/бібліотек для написання клієнтської частини застосунку. В проєкті використовується Angular 16 версії, який є сумісним з наступними версіями Node.JS ^16.14.0 || ^18.10.0. Це потужний фреймворк, який дозволяє створювати динамічні, інтерактивні користувацькі інтерфейсами.

До основних переваг можна віднести[6]:

- Компонентно-орієнтований підхід, що сприяє кращій структуризації коду та його перевикористання.
- Використання Typescript, підвищує якість коду, допомагає уникати типових помилок, має краще автодоповнення та навігацію по коду. Для багатьох цей пункт може бути сумнівний, адже іноді простіше було б використати звичайний Javascript, проте коли мова йде про enterprise рішення, то безперечну перевагу має саме Typescript.
- RxJs, інструмент для написання асинхронного та ефективного коду. Підвищує швидкість роботи застосунку.
- Інструменти розробки, Angular надає такий інструментом, як Angular CLI, який дозволяє автоматизувати створення нових компонентів, збирання і розгортання застосунку.
- Підтримка. Оскільки Angular був розроблений компанією Google, він має велику спільноту розробників, що забезпечує регулярне оновлення продукту.

Angular використовується разом з бекенд-проксі у вигляді Spring, який забезпечує зв’язок між клієнтською частиною і серверними даними. Сполучення Angular і Spring в одному проєкті є популярним рішенням для створення повноцінних веб-додатків, які вимагають надійного, масштабованого фронтенду та бекенду.

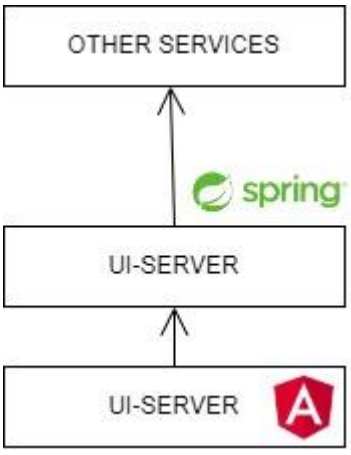


Рисунок 2.3 Архітектура ui-server

2.5 Підсумок розділу

У цьому розділі ми розглянули архітектури застосунку SmartUkma, визначили основні переваги даного рішення. Також познайомились з технологічним стеком проєкту, описавши, яку проблему вирішує кожен інструмент.

Особливу увагу хочеться приділити актуальності використаних технологій, що забезпечує не тільки високий рівень функціональності, але й додає впевненості в тому, що дані інструменти будуть підтримуватись і в досить далекому майбутньому.

Розділ 3. Розробка клієнтської частини

3.1 Оновлення зовнішнього вигляду

Процес розробки можна розділити на дві частини. На першому етапі основним завданням було переписати та уніфікувати існуючі сторінки застосунку, забезпечивши їх відповідність єдиному зовнішньому стилю. Для виконання цього завдання був застосований ui-kit від velzon. Він ідеально підходить для швидкої розробки панелі адміністратора, адже містить в собі набір базових структур, необхідних компонентів для побудови форм, таблиць, діаграм, містить набір іконок та заготовки складних ui елементів. Під собою він містить відомий всім фреймворк bootstrap, полегшуючи взаємодію з ним.

Процес оновлення кодової бази можна поділити на наступні послідовні кроки:

1. Копіювання сторінки: створення нової версії на основі існуючої компоненти.
2. Оновлення дизайну: застосування ui-kit до нової компоненти
3. Рефакторинг інтерфейсу редагування та створення. Для сутностей, що містять невелику кількість мета-інформації до заповнення, якщо це доцільно, використовували модальні вікна замість повноцінних сторінок.
4. Оновлення роутингу: інтеграція нових сторінок та додавання хлібник крихт для кращої навігації.

Наприклад, сторінка «Стипендії» після редагування виглядає наступним чином:

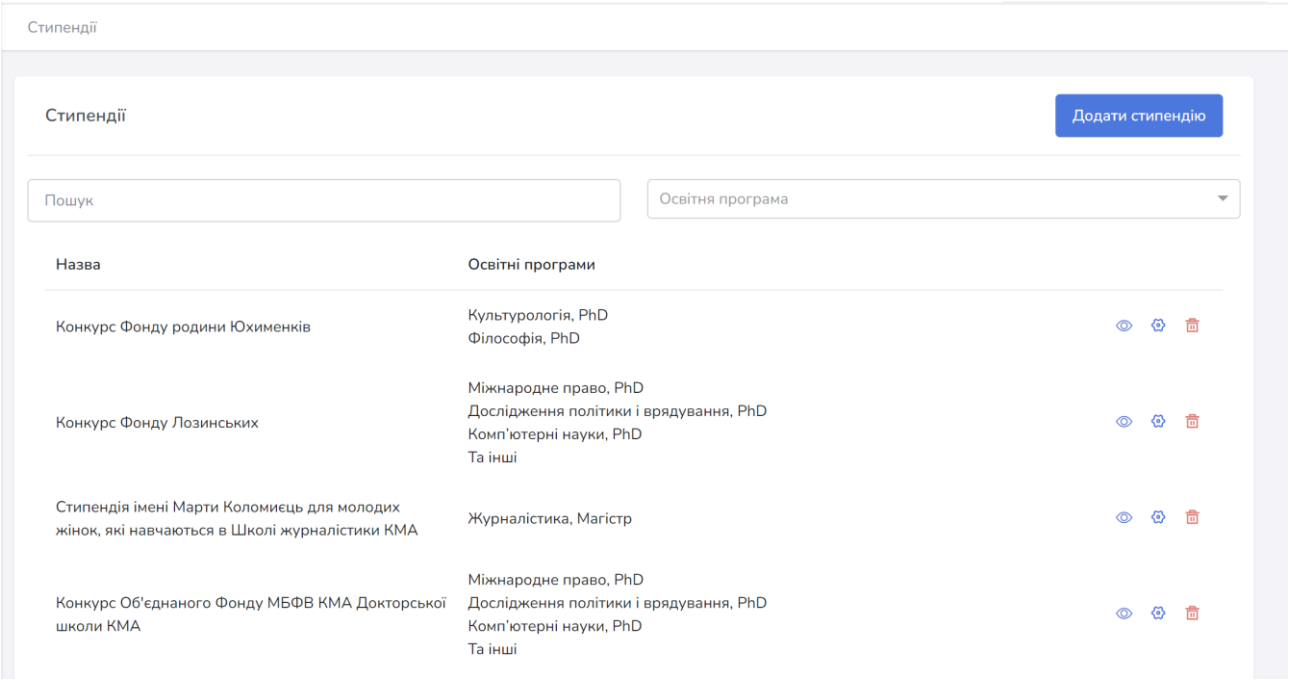


Рисунок 3.1 Приклад оновленого дизайну

Використання ui-kit дозволило покращити гнучкість сторінки, яка тепер адаптується відповідно до розмірів екран, уніфікувати стиль всіх сторінок. Загалом був прооблений перехід на новий дизайн для наступних сторінок: Стипендії, Працівники, Студенти, Персонал факультету та Сертифікатні програми.

Відбулось значне скорочення кодової бази, оскільки зменшилась кількість коду потрібного для стилізації компонентів. В свою чергу код став більш читабельним та легким для підтримки. Дані оптимізації дозволили підвищити швидкість роботи програми за рахунок зменшення вихідного коду.

3.2 Реалізація створення документів

Для створення документів було прийнято рішення розбити компонент на три менші складові:

- Основна інформація: відбувається заповнення основних полів документа, таких як назва, номер, опис, список категорій тощо

- Доступність: розділ, що відповідає за керування доступом до документа та розподілом обов’язків між користувачами, а саме визначення виконавців, кореспондентів тощо
- Додатки: відповідає за додавання додатків до документ разом з погодженнями ухвал. Відображає список вже доданих документів з можливістю їх редагування

Для керування і перевірки коректності введених даних використовуються реактивні форми з відповідного пакету Angular. Ось декілька основних їх переваг[7]:

- Модельне керування станом: реактивні форми використовують модель реактивного програмування, що дозволяє керувати станом форми через потоки даних, що забезпечує синхронізацію між станом форми і її моделлю.
- Валідація: в реактивних формах її можна легко інтегрувати з моделлю. Валідатори можна комбінувати між собою і перевикористовувати, що стає в нагоді нам, враховуючи кількість обмежень на вхідні дані для документів.
- Продуктивність: за рахунок того, що зміна у формі це фактично потік даних, відбувається певна оптимізація на рівні перевірок та побудові DOM, що дозволяє забезпечити кращу продуктивність для складних форм.

Після заповнення і валідації форми відбувається завантаження даних на сервер. Оскільки ми взаємодіємо з декількома сутностями одночасно, процес загрузки є досить нетривіальним.

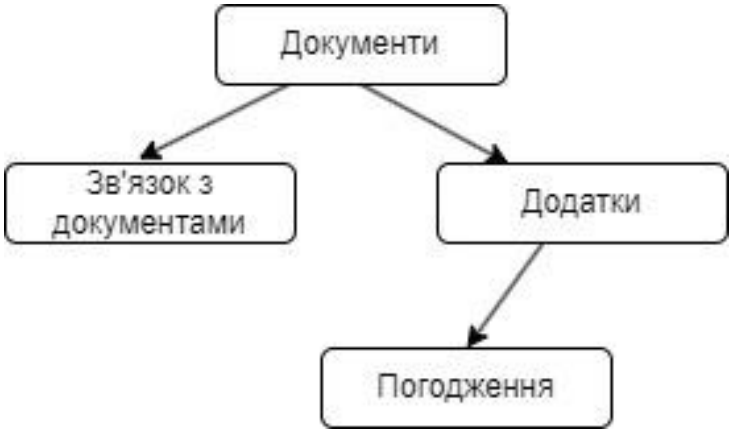


Рисунок 3.2 Процес завантаження документів

Спочатку відбувається завантаження документа з його метаданими та разом з файлами (відбувається перевірка типу файлів документа, якщо присутня нерозпізнавана версія, використовується окремий ендпоінт на сервері). Результатом цієї дії буде створений на сервері унікальний ідентифікатор документа, використовуючи який ми можемо створювати зв’язок з попередніми версіями документа (використовується окремий ресурс, що додає зв’язок, але не скасовує чинність попереднього документа). Завантаження додатків відбувається також окремим викликом, а саме йде підв’язування його до ID документа. Завантаження погоджень до ухвали має аналогічний характер до додатків. Кожне погодження підв’язується до унікального ідентифікатора попередньо створеного додатка.

Слід також зауважити, що кожен додаток, кожен зв’язок і кожне погодження це окремий виклик на сервер і щоб оптимізувати процес використовується forkJoin оператор[8], котрий на вхід отримає перелік observables(в даному випадку наші виклики на сервер) і чекає на виконання їх всіх, продукуючи в результаті масив з повернених кожним викликом значень.

3.3 Оновлення документів

Процес оновлення документів дуже схожий на вищеописаний процес їх додавання. Існує лише декілька концептуальних відмінностей, а саме під час оновлення початкова модель документа заповнена вже існуючими значеннями, також змінюються ендпоінти виклику на сервері. Для уникнення дуплікації коду прийнято рішення застосувати взірець «Стратегія»

Стратегія – це поведінковий шаблон проєктування, котрий дозволяє виділити набір алгоритмів, функцій, тощо в окремі класи. В залежності від ситуації ми можемо використовуватимемо потрібну нам реалізацію. Таким чином ми виділяємо два набори поведінки, один для створення документів, інший для їх оновлення. Далі ми створюємо два шляхи, котрі використовують однакові компоненти, проте передаємо їм різні реалізації взаємодії з моделлю.

Також під час оновлення документів нам треба розуміти, що саме відбувається з додатками та їх погодженнями. Для цього ми вводимо для них чотири стани:

- Створити: означає що це додаток/погодження, які треба створити.
- Оновити: додаток чи погодження були оновлені.
- Видалити: додаток був видалений, але якщо статус додатку перед цим був «Створити», додаток просто видаляється з пам’яті, адже він ще не був доданий в систему, в інших випадках він залишається в пам’яті, проте не відображається.
- Без змін: додаток ніяк не був змінений, проте нам все одно потрібно пройтись по його погодженнях, якщо такі наявні і виконати відповідні дії по оновленню, створенню чи видаленню їх.

Даний приклад наглядно демонструє як використання шаблонів програмування допомагають уникнути значної дуплікації коду. Відповідний підхід з використанням стратегії наявний і при оновленні

дизайну попередніх сторінок застосунку, однак формат використання стратегії може варіюватись в залежності від типу сторінок, якщо це модальні вікна, тоді в компоненті при натисканні на кнопку явно вказується яку стратегію використовувати. У випадку з окремими компонентами на створення та редагування ін’єкція стратегії визначається разом шляхами.

3.4 Створення категорій та відображення типів документів

Для взаємодії з категоріями були додані відповідні сторінки. Процес створення нових категорій має значно простіший вигляд, адже потребує лише її назви. На додачу під час створення чи редагування категорії існує можливість змінити список документів приналежних до даної категорії. Для цього додається список вже існуючих документів в системі і біля кожного з них є позначка чи додати документ до категорії. Спеціальний ресурс відповідає за додавання чи видалення цього відношення.

Важливою функцією також є те, що після видалення категорії автоматично припиняється її зв’язок з документами, що забезпечує коректне управління даними.

Ще одним покращенням роботи є сторінка перегляду типів документів. Хоча ми і не можемо створювати нові(з урахуванням поставленої задачі список типів документів є фіксований, в той час як список типів додатків може бути не до кінця визначений), нам потрібно керувати групами користувачів і їх доступом до певного типу. Зворотній функціонал також підтримується в програмі, ми можемо керувати групами користувачів, створювати нові, визначати типи документів до яких відкритий доступ.

3.5 Підсумковий результат

У третьому розділі ми охарактеризували ключові аспекти практичної частини даної кваліфікаційної роботи. Ми визначили, що основні зусилля

були спрямовані на оновлення інтерфейсу SmartUkma та додавання нового функціоналу по взаємодії з документами. Детально було описано процес розробки сторінок створення та редагування документів. Робота над сторінками перегляду списку документів та перегляду окремого документа винесена поза межі даної роботи, адже відбувалось лише їх покращення та рефакторинг. Інтерфейс розробленого функціоналу розміщений в додатку Д.

Висновок

В ході написання кваліфікаційної роботи було проведено глибокий аналіз предметної області, що дозволило краще зрозуміти вимоги до зберігання нормативно-правової бази університету. Опис бізнес-процесів додавання, редагування та видалення документів допоміг визначити ключові аспекти взаємодії між собою документів, додатків, користувачів та інших сутностей в системі.

Було розглянути архітектурні рішення, що застосовуються в рамках проєкту Smart. Описана мета використання таких технологій як Elasticsearch, Spring Boot, Hibernate, Angular та їх переваги. Розроблена ER-діаграма для розуміння того, як спроєктована база даних.

В третьому розділі продемонстрували процес роботи над практичною частиною. Описали перехід на новий інтерфейс з використання Velzon ui-kit. Зазначено процес реалізації створення, редагування документів, також управління групами користувачів, категоріями та типами документів.

Загалом результати кваліфікаційної роботи демонструють успішне впровадження комплексної системи з управління нормативно-правовою базою університету. Виконана робота стала важливим вкладом у розвиток інформаційної системи університетського середовища та може слугувати основою для подальших покращень та доробок в цій галузі.

Список використаних джерел

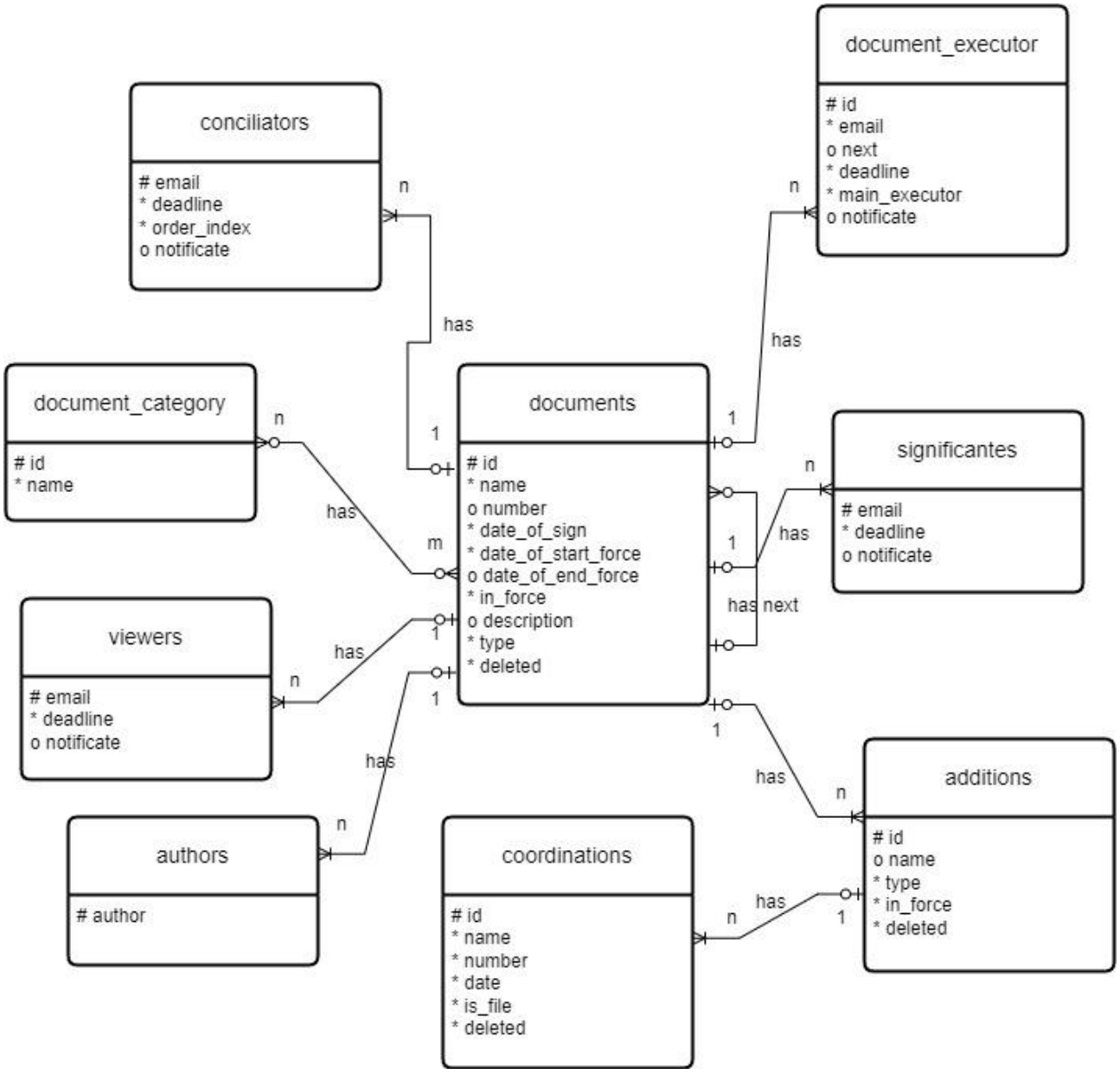
1. Типи документів НаУКМА [Електронний ресурс]. /Andrii Hlybovets, Oleksandra Mykhailenko// – 2024 Режим доступу https://app.clickup.com/20593116/v/dc/kmeew-587/kmeew-427
2. Бізнес процес ручного додавання документів [Електронний ресурс]. /Oleksandra Mykhailenko// – 2024 Режим доступу https://app.clickup.com/20593116/v/dc/kmeew-607/kmeew-447
3. Advantages and Disadvantages of Microservices Architecture [Електронний ресурс]. – 2024 Режим доступу https://cloudacademy.com/blog/microservices-architecture-challenge-advantage-drawback/
4. Elasticsearch documentation [Електронний ресурс]. – 2024 Режим доступу https://www.elastic.co/elasticsearch
5. PostgreSQL vs MySQL: The critical difference [Електронний ресурс]. – 2023 Режим доступу https://www.integrate.io/blog/postgresql-vs-mysql-which-one-is-better-for-your-use-case/
6. The Good and the Bad of Angular Development [Електронний ресурс]. – 2022 Режим доступу https://www.altexsoft.com/blog/the-good-and-the-bad-of-angular-development/
7. Reactive Angular Forms [Електронний ресурс]. – 2024 Режим доступу https://www.gmihub.com/blog/reactive-forms-in-angular/
8. RxJs api reference [Електронний ресурс]. – 2024 Режим доступу https://rxjs.dev/api/index/function/forkJoin

Додатки

Додаток А – Docker compose файл для запуску необхідних сервісів

```
version: "3.7"
services:
  postgres:
    image: postgres:13.4
    ports:
      - "5432:5432"
    volumes:
      - smart_kma_volume:/var/lib/postgresql/data
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: postgres
  activemq:
    image: rmohr/activemq:5.14.4-alpine
    ports:
      - "8161:8161"
      - "61616:61616"
  elasticsearch:
    image: docker.elastic.co/elasticsearch/elasticsearch:8.11.0
    container_name: elasticsearch
    ulimits:
      memlock:
        soft: -1
        hard: -1
      nofile:
        soft: 65536
        hard: 65536
    volumes:
      - ./elastic-data:/var/lib/elasticsearch/data
    ports:
      - 9200:9200
      - 9600:9600
  account-server:
    image: account-server:latest
    build: ./../account-server
    ports:
      - "8080:8080"
  info-server:
    image: info-server:latest
    build: ./../user-info
    ports:
      - "8081:8081"
volumes:
  smart_kma_volume:
    external: true
```

Додаток Б – ER діаграма

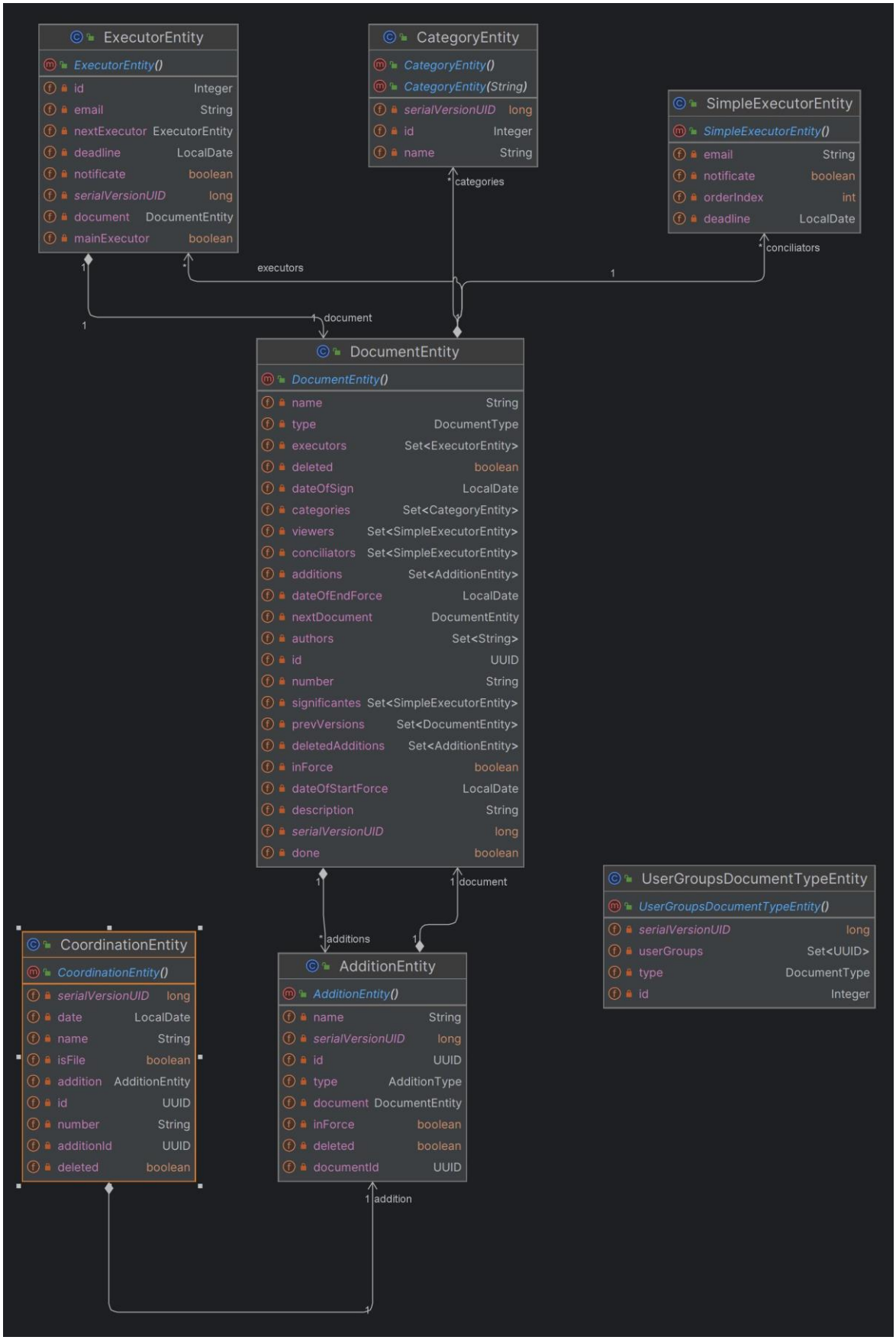


Додаток В – Декларативне визначення Rest API

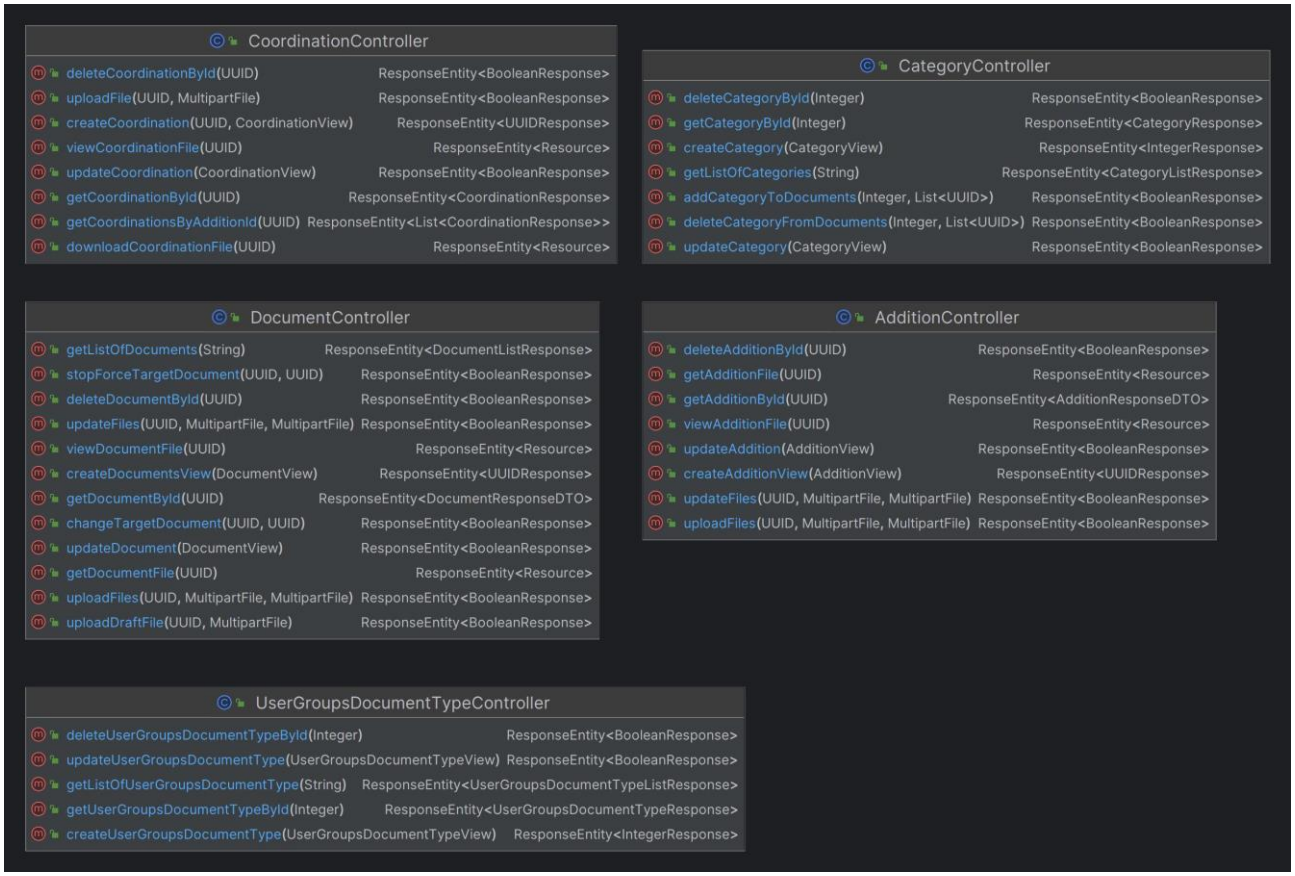
Приклад визначення ресурсу отримання списку документів:

```
paths:
  /api/document:
    get:
      tags:
        - document-controller
      summary: Get list of documents
      operationId: getListOfDocuments
      parameters:
        - in: query
          required: false
          name: restrict
          schema:
            type: string
      responses:
        200:
          description: List of documents
          content:
            application/json:
              schema:
                  type: DocumentListResponse
```

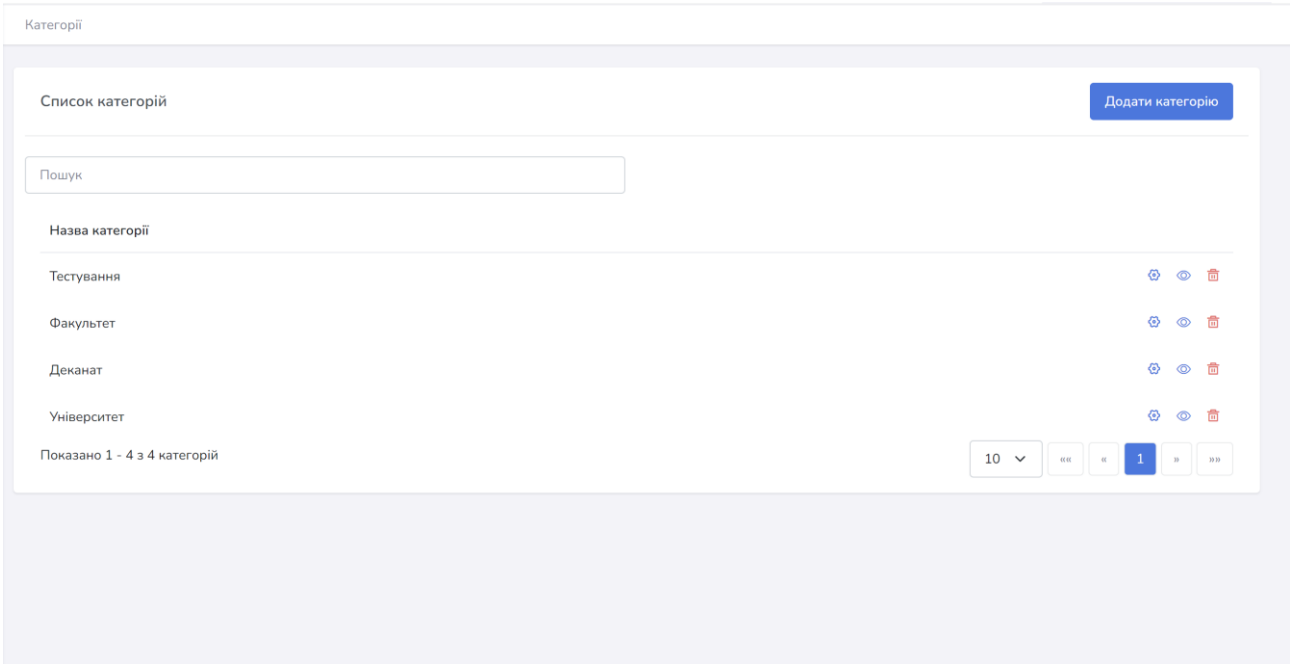
Додаток Г – Діаграма сутностей визначених за допомогою Hibernate

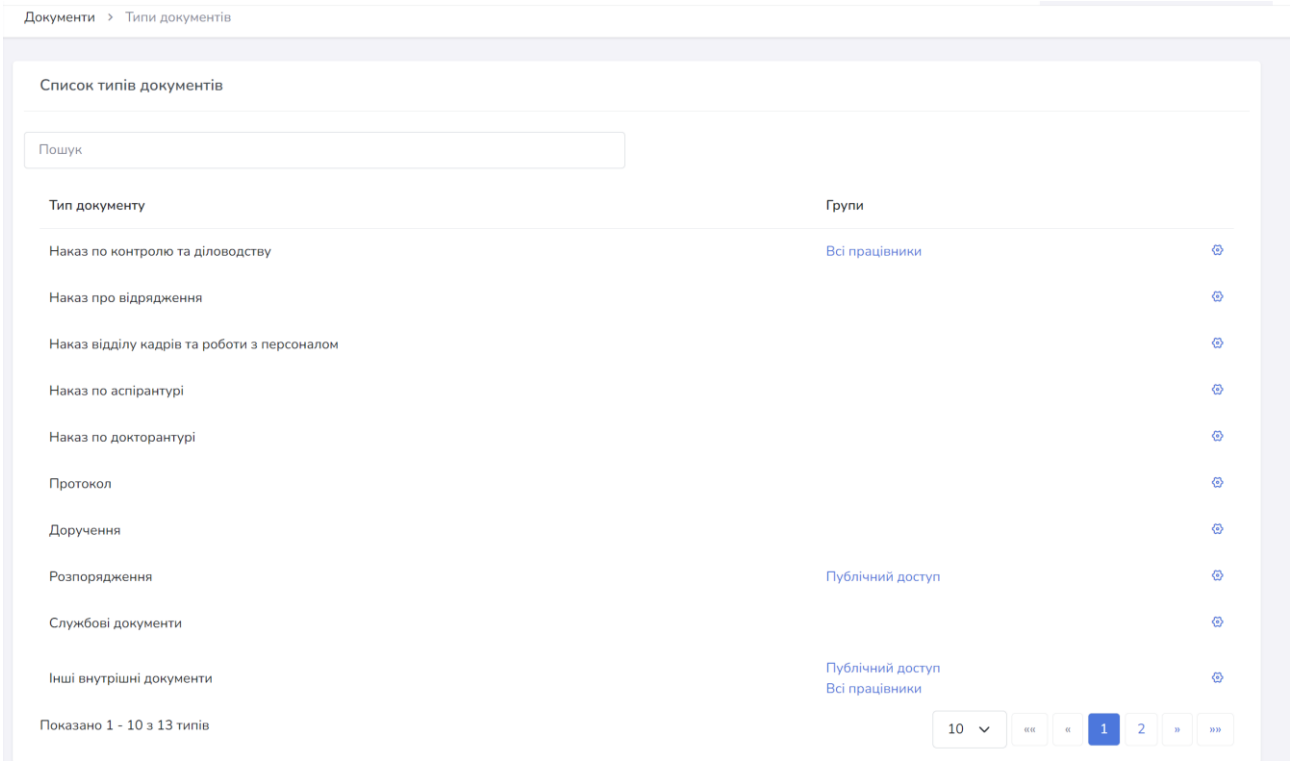


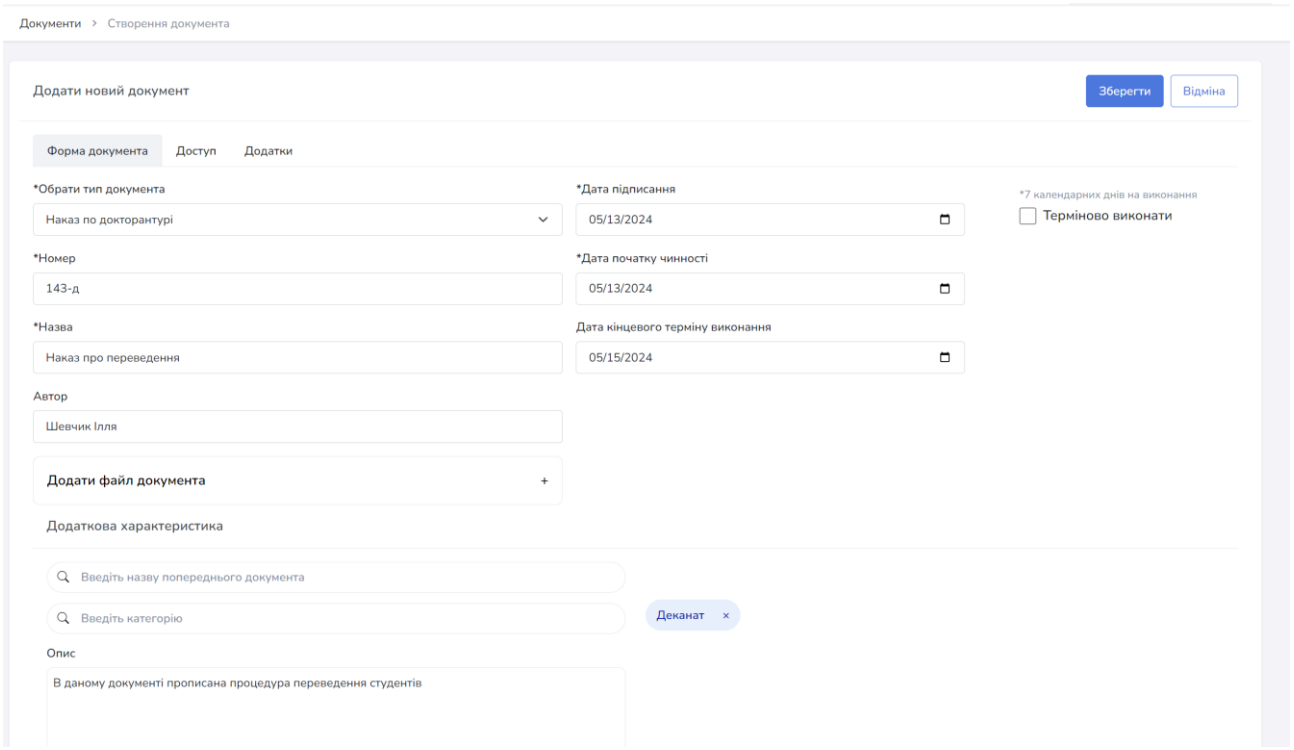
Додаток Ґ – Список контролерів та відповідних ресурсів

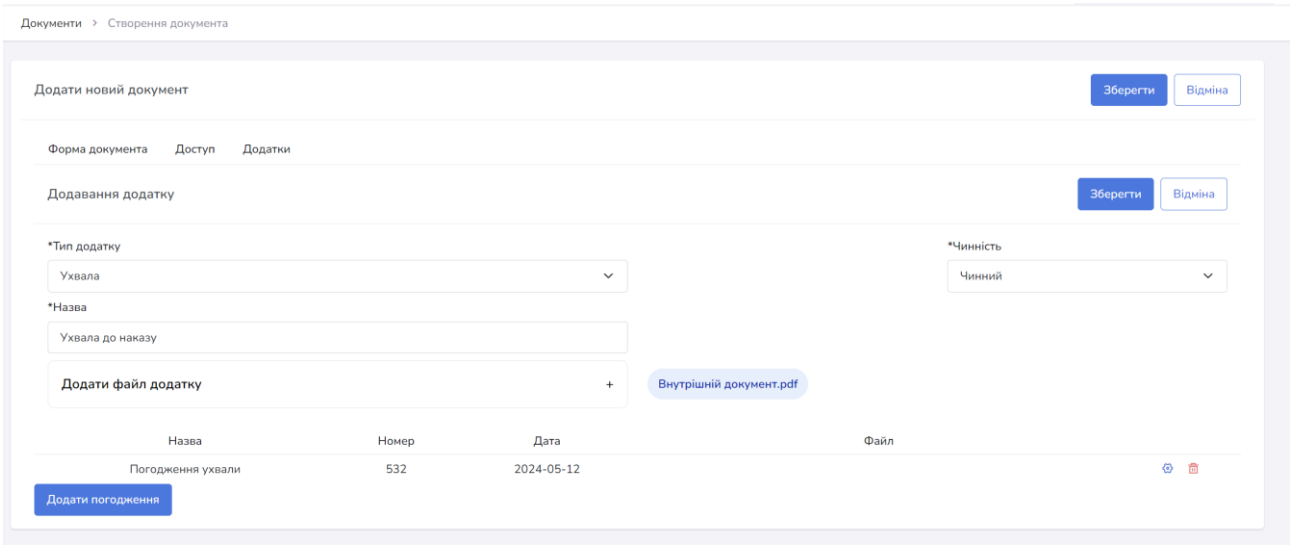


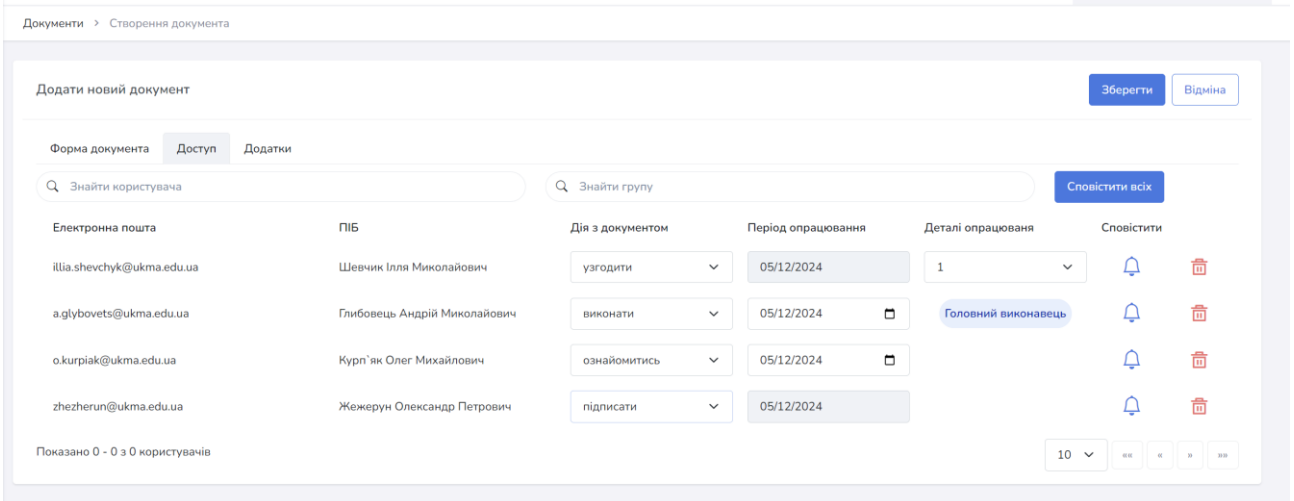
Додаток Д – Інтерфейс застосунку






Документи › Створення документа
Додати новий документ
Зберегти
Відміна
Форма документа
Доступ
Додатки
*Обрати тип документа
Наказ по докторантурі
*Дата підписання
05/13/2024
*7 календарних днів на виконання
Терміново виконати
*Номер
143-д
*Дата початку чинності
05/13/2024
*Назва
Наказ про переведення
Дата кінцевого терміну виконання
05/15/2024
Автор
Шевчик Ілля
Додати файл документа
Додаткова характеристика
Введіть назву попереднього документа
Введіть категорію
Деканат
Опис
В даному документі прописана процедура переведення студентів


Документи › Створення документа
Додати новий документ
Зберегти
Відміна
Форма документа
Доступ
Додатки
Додавання додатку
Зберегти
Відміна
*Тип додатку
Ухвала
*Чинність
Чинний
*Назва
Ухвала до наказу
Додати файл додатку
Внутрішній документ.pdf
Назва
Номер
Дата
Файл
Погодження ухвали
532
2024-05-12
Додати погодження


Документи › Створення документа
Додати новий документ
Зберегти
Відміна
Форма документа
Доступ
Додатки
Знайти користувача
Знайти групу
Сповістити всіх
Електронна пошта
ПІБ
Дія з документом
Період опрацювання
Деталі опрацювання
Сповістити
illia.shevchyk@ukma.edu.ua
Шевчик Ілля Миколайович
узгодити
05/12/2024
1
a.glybovets@ukma.edu.ua
Глибовець Андрій Миколайович
виконати
05/12/2024
Головний виконавець
o.kurpiak@ukma.edu.ua
Курп`як Олег Михайлович
ознайомитись
05/12/2024
zhezherun@ukma.edu.ua
Жежерун Олександр Петрович
підписати
05/12/2024
Показано 0 - 0 з 0 користувачів
10