Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мережних технологій факультету інформатики

Кваліфікаційна робота

освітній ступінь – магістр

на тему: «**Розробка системи документообороту для університету / Development of the system of document flow**»

Виконав: студент 2-го року навчання,

освітньо-наукової програми

«Інженерія програмного забезпечення», 121

Курп’як Олег Михайлович

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 інформатики факультету інформатики

ЗАТВЕРДЖУЮ

Зав. кафедри інформатики,

к.ф.-м.н. Гороховський С.С.

_____

“___” _____ 2024 р.

ЗАВДАННЯ

ДЛЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ СТУДЕНТУ

Курп’яку Олегу Михайловичу

1. Тема роботи**:** Розробка системи документообороту для університету керівник роботи Глибовець Андрій Миколайович
   затверджені наказом вищого навчального закладу від «__»_____2024року №____
2. Строк подання студентом роботи _____
3. План роботи _____

4. **Календарний план виконання роботи**:

№	Назва етапу	Термін виконання	Примітка
1.	Отримання теми кваліфікаційної роботи	20.10.2023	
2.	Створення плану роботи	10.12.2023	
3.	Ознайомлення з проєктом	14.01.2024	
4.	Розробка практичної частини проєкту	23.04.2024	
5.	Написання першого розділу роботи	05.04.2024	
6.	Описання практичної частини роботи	01.05.2024	
7.	Перегляд змісту роботи з керівником	12.05.2024	
8.	Внесення змін відповідно до зауважень наукового керівника	20.05.2024	
9.	Створення презентації	30.05.2024	
10.	Захист кваліфікаційної роботи	10.06.2024	

Студент Курп’як О. М.

Керівник Глибовець А.М.

“____” _____

ЗМІСТ

ВСТУП

В сучасному світі, де все навколо стало цифровим, складно залишити таку звичну річ як документи в паперовому форматі. Документів стає все більше, а значить знайти необхідну інформацію все складніше. Світ вже давно зрозумів, що всю інформацію потрібно зберігати в цифровому вигляді. Документи потрібно не просто зберігати, а ще й зв’язувати між собою – документ А доповнив документ Б, документ В скасував документ Г. Для цього всього необхідна система документообігу, за допомогою якої буде зручно організувати роботу з документами, де можна легко знайти необхідну інформацію та зрозуміти, який документ є актуальним в конкретний момент часу. Також така система має допомагати в процесі внесення нових документів в систему. Процес створення, узгодження і фінального затвердження документу має бути прозорим, передбачуваним, а всі дії користувачів системи мають бути залоговані. Також так як це цифровий документообіг, то в такій системи не обійтись без цифрового підпису. В системі має бути можливість підписати документ цифровим ключем, а також перевірити, що деякий документ дійсно був підписаний.

РОЗДІЛ 1. ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Поняття документообігу

Документообіг — це рух документів в установі від моменту створення або від одержання зі сторони до моменту передачі на зберігання до архіву. Українське законодавство надає наступне визначення терміну: документообіг в установі — рух службових документів з моменту їх створення або одержання до завершення виконання або відправлення [1].

Електронний документообіг має ж на меті автоматизувати багато етапів звичайного документообігу. Зокрема процес передачі документу між різними підрозділами надає можливість робити підтвердження документу чи передачу до наступного виконавця онлайн, також має автоматизовану систему сповіщень. В електронному документообігу автоматизовані такі процеси як автоматичне вступання в силу певного документу за умови відомої дати, з якої починає діяти цей документ. Також до переваг електронного документообігу слід віднести:

- Зручний пошук документів – загальна інформація про документ, а також його вміст проіндексовані системою документообігу. Це все дає можливість використовувати вільно текстовий пошук серед актуальних документів, а також в архіві
- Зручний перегляд документів і всіх пов’язаних з ним документів. Якщо в документу є зв’язок з іншим документом, то в системі електронного документообігу передбачено можливість перегляду цих пов’язаних документів. Такий функціонал прибирає потребу в зверненні до фізичного реєстру документі чи до архіву
- Доступ до документів з будь-якого куточку світу. Так як всі документи зберігаються на єдиному сервері, який доступний онлайн, то відповідно і користувачі системи мають доступ до цих документів онлайн

1.2. Основні елементи документообігу

Перед реалізацією документообігу слід розібратись з ключовими елементами такої системи. Нижче наведено список ключових елементів системи електронного документообігу

- Документ – це сам документ, який необхідно зберігати. Документ також може містити різного роду додатки. Документ складається з різного роду атрибутів як то назва, дата початку дії, виконавець та інші, а також файл документу з розпізнаним текстом та цифрового підпису накладеного на цей документ.
- Зв’язок між документами – це зв’язок між старим документом та новим, який скасовує дію попереднього. Нові документи можуть скасовувати один чи декілька попередніх документів.
- Виконавці – це люди, які в реальному житі відповідають за певну дію, яка необхідна для реалізації документу. Для кожного типу документу визначено певну послідовність виконавців, які по завершенню своєї дії мають підтвердити її виконання або навпаки скасувати подальшу обробку документу, якщо з документом є певні проблеми.
- Сховище даних – сховище, яке буде відповідальним за збереження зв’язків документів, а також за збереження поточного стану документів.
- Система сповіщень – система, яке надсилає користувачам повідомлення про зміну статусу документа або нагадування про те, що користувач має зробити певні дії з документом
- Цифровий підпис – елемент системи, який дозволить накладати цифровий підпис з використанням відповідних криптографічних алгоритмів. Також система має передбачати можливість верифікації цифрового підпису в документі
- Система прав доступу, яка визначає які користувачі чи групи користувачів мають доступ до певного документу. Також для система має

регулювати права роботи з самим документом: хто може переглядати, а хто може редагувати.

- Шаблони документів – більшість документів маю стандартний вигляд і відрізняються між собою лише дрібними деталями, тому для таких «стандартних» документів цілком доцільно мати бібліотеку шаблонів, яка пришвидшить створення нового документу.

1.3. Цифровий підпис

Електронний цифровий підпис (ЕЦП) - це дані в електронній формі, отримані за результатами криптографічного перетворення, які додаються до інших даних або документів і забезпечують їх цілісність та ідентифікацію автора[2].

Для цифрового підпису потрібно декілька компонентів: сам файл, який потрібно підписати, приватний ключ, алгоритм на основі якого буде згенеровано цифровий підпис та алгоритм для верифікації підпису.

В якості ключа для підпису використовують асиметричні ключі згенеровані з використанням затверджених алгоритмів. NIST(National Institute of Standards and Technology)[3] станом на 2023 рік затвердив 5 версію специфікації, яка і декларує алгоритми для цифрового підпису, а також алгоритми для створення ключів. Сам ключ генерується акредитованим центром сертифікації ключів[4], який також є відповідальним за підтвердження цифрового підпису, так як саме ця установа зберігає в себе відкриту частину ключа. Список таких установ регулюється державним органом.

Алгоритм створення ключа та самого цифрового підпису складається з наступних етапів:

1. Вибір параметрів. На цьому етапі задаються параметри, які визначать якість та складність підпису. NIST має рекомендації, яким варто слідувати під час підбору цих параметрів

1.1. Вибір хеш функції (*H(x)*)

1.2. Вибрати довжину ключа(*L*) та довжину вихідного хеша (*N*). На сьогодні NIST рекомендує обирати *L = 3072 і N = 256*

1.3. Також необхідно вибрати просте число *q*, яке складається з *N* біт та просте число *p*, яке складається з *L* біт. Число *p* має бути підібране таким чином, щоб *(p-1) * q = 1*. Також необхідно вибрати число *h* з проміжку *(2, p-2)*

1.4. Після цього необхідно вирахувати число g використовуючи наступну формулу

$$g = \frac{h^{p-1}}{\mathrm{q} * \mathrm{mod(p)}}$$

1.5. Параметри *p*, *q* і *g* це є параметри для шифрування і вони можуть буди відомі іншим користувачам

1.6. Потрібно вибрати приватний ключ *x* такий що *1 < x < q*

1.7. Та вирахувати публічний ключ y за формулою

$$y = g^x \ mod \ p$$

2. Генерація підпису для повідомлення *m*

2.1. Потрібно вибрати число *k* таке що *0 < k < q*

2.2. Вирахувати *r* за формулою

$$r = \ (g^k \ mod \ p) \ mod \ q$$

2.3. Вирахувати *s* за формулою

$$s = (k^{-1} * (H(m) + xr)) \ mod \ q$$

2.4. Параметри *r* і *s* є підписом для повідомлення *m* підписане ключем *x*

3. Верифікація підпису

3.1. Потрібно знайти *w* таке що

$$s * \ w \ mod \ q \ = \ 1$$

3.2. Знайти *u1* за формулою

$$u1 = hw \ mod \ q$$

3.3. Знайти *u2* за формулою

$$u2 = rw \ mod \ q$$

3.4. Знайти *v* за формулою

$$v = \big((gu1 * yu2) \ mod \ p\big) \ mod \ q$$

4. Якщо *v* та *r* рівні, то підпис верифіковано.

Цифровий підпис є унікальним для кожного повідомлення. Тобто якщо взяти документ і підписати його, а пізніше трохи змінити його наповнення, то старий підпис більше не буде працювати для нового стану документу.

1.4. Особливості документообігу університету

В університеті існує багато різних типів документів, які мають свої певні правила створення, обробки, свої правила доступу та інколи унікальні атрибути. Прикладами таких документів будуть: розпорядження, доручення, наказ про відрядження, наказ відділу кадрів та роботи з персоналом та інші. Також кожен з цих основних типів документів має в собі додатки, які також є різних типів: статут, положення, інструкції та інші. Для кожного з типів документів є власний набір правил як саме документ має бути затверджений та підписаний. Для кожного типу документа є конкретні люди чи відділи, які мають погодити цей документ, а також є конкретні люди чи відділи, які в майбутньому будуть відповідати за виконання цього документу.

У випадку університету також варто говорити про документи, які напряму пов’язані з процесом навчання – відомості студентів про оцінки. Система електронного документообігу університету має передбачати можливість автоматичної генерації документів такого типу. Система має знати про поняття «дисципліна», «група студентів-слухачів», «оцінка» та «викладач». На основі цієї

інформації система має давати змогу автоматично генерувати такий документ та дозволяти викладачу накласти ЕЦП на такий документ.

РОЗДІЛ 2. ОГЛЯД АРХІТЕКТУРИ РІШЕННЯ

2.1. Функціональні вимоги

Система електронного документообігу має відповідати університетським вимогам [8]. Зокрема система має давати можливість зберігати документи з визначеним набором атрибутів та валідацій, які відрізняються для кожного з типів документів. Система документообігу НаУКМА визначає наступні типи документів:

- Протокол
- Доручення
- Розпорядження
- Накази по контролю діловодства
- Накази про відрядження
- Накази студентського відділу кадрів
- Накази відділу кадрів та роботи з персоналом
- Накази по аспірантурі
- Накази по докторантурі

Кожен з типів документів має наступні характеристики:

- ID – автоматично згенерований згенерований системою унікальний номер документу
- Назва – назва документу кирилицею з обмеженням в 150 символів
- Порядковий номер. В різних типів документів формат цього номера є різним. Також варто зазначити, що кожного року відлік має починатись з 1
    - Протокол, Доручення, Розпорядження, Накази по контролю діловодства – просто порядковий номер цього документу від початку року. В кожного з типів документів своя власна нумерація
    - Накази про відрядження(в), Накази студентського відділу кадрів(с), Накази відділу кадрів та роботи з персоналом(к), Накази по

аспірантурі(а), Накази по докторантурі(д) – номер «прочерк» «суфікс». Приклад «10-а» - Наказ по аспірантурі з номером 10

- Дата підписання – дата коли документ був підписаний відповідальною особою
- Дата початку чинності – дата з якої всі вимоги документу вступають в силу
- Дата кінцевого терміну виконання – дата кінця дії документу
- Виконавці – одна або декілька осіб, які відповідають за виконання документу. Виконавці мають бути впорядкованими
- Автор – Особа, що створила документ
- Узгодження – особа чи особи, які узгоджують документ до того, як він буде підписаний. Якщо це декілька осіб, то список має бути впорядкованим
- Кореспонденти – особа чи особи, які підписують документ
- Опис – короткий опис документу
- Додатки – в документу є додатки, які фактично є окремими документами, але вона існують лише як частина основного. Додатки мають наступні характеристики:
    - Назва
    - Тип – Статут, Положення, Правила, Додатки до правил, Інструкції, Рекомендації(методичні), Програма, Договір, Ухвала, Інші документи
    - Чинність – додаток

2.2. Архітектура SMART

Система SMART побудована з використанням мікросервісної архітектури(рисунок 1). Вибрано підхід саме з мікросервісною архітектурою через зручність розробки, легкість самих мікросервісів, а також такий підхід дозволяє масштабувати кожен окремий сервіс незалежно від інших.

Для між сервісної комунікації використовуються REST API як основний засіб комунікації. Такий підхід дозволяє реалізовувати складну логіку в різних елементах система і при цьому дає можливість спілкуватись різним компонентам системи між собою. Також в окремих випадках для між сервісної комунікації використовуються черги повідомлень. Використання черг повідомлень дозволяє реалізувати event-driven підхід в окремих місцях. REST API в системі використовується тоді, коли важливо дочекатись результату дії іншого сервісу, натомість черга повідомлень використовується для того, щоб сповістити інші частини системи про те, що відбулась деяка події, але результат обробки цього повідомлення вже не так важливий.

В SMART варто виділити наступні міскросервіси:

- Account-server – сервер, який відповідає за видачу прав користувачу. Всі сервіси спілкуються з account-server, щоб отримати актуальні права користувача і тим самим можуть або дозволити, або заборонити певну дію.
- Info-server – сервер, який зберігає основну і загальну інформацію, яка стосується університету і всіх внутрішніх сутностей. Цей сервер виділений окремо, щоб зберігати інформацію в єдиному форматі і єдиному місці. Всі інші сервіси можуть робити запити до info-server, щоб, наприклад, отримати детальну інформацію про користувача.
- Course-work, document-management, document-generator, storage-server, notifications-server, ui-constructor – це набір спеціалізованих сервісів, кожен з яких відповідає за певний функціонал:
    - Course-work – процес запису на курсові роботи, оцінки за курсові роботи та все, що пов’язано з цим
    - Document-management – електронний документообіг
    - Document-generator – сервіс, який дозволяє генерувати документи на основі шаблонів з використанням авто-генерованих форм

    - Storage-server – сервер для збереження файлів: картинки для профілів, файли для сайтів та інше.
    - Notifications-server – сервер, який займається розсилкою сповіщень користувачам
    - Ui-constructor – сервер, який відповідає за налаштування університетських сайтів та їхнє відображення
- Ui-server – сервер, який відповідає за обробку запитів з SMART та проксює запити до всіх інших сервісів системи.

В системі використовується 3 сховище даних:

- реляційна база даних Postgres для збереження інформації про всі сутності та їхні зв’язки, які існують в системі. Кожен мікросервісів має окрему схему.
- документ орієнтована база даних ElasticSearch для індексування тексту документів.
- файлова система для збереження файлів документів, картинок та інших.

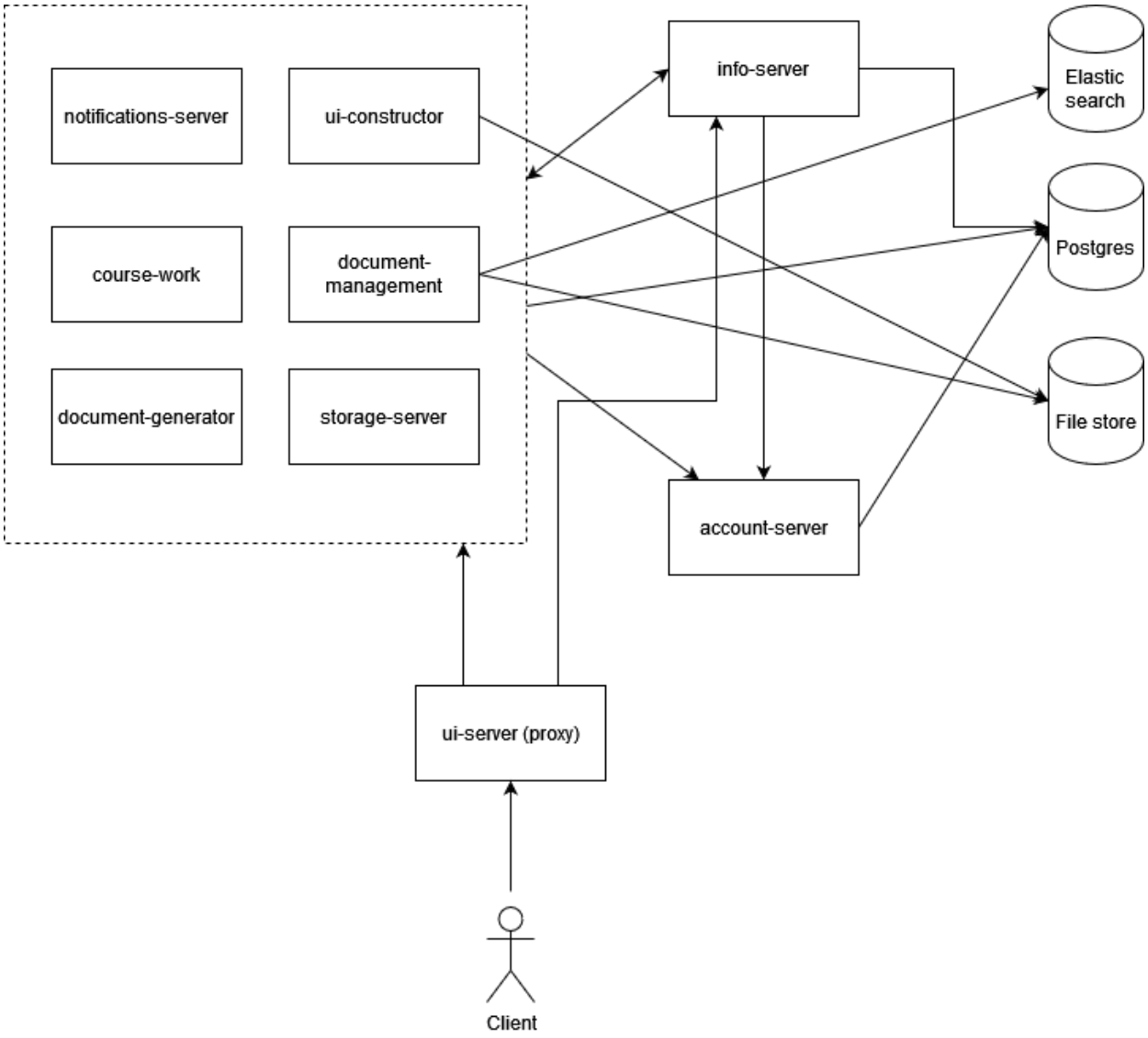

Рисунок 1

2.3. Технології та SDLC

Всі сервіси побудовані з використанням Java 11 та Spring Boot 2. Для багатьох сучасних ентерпрайз рішень це є стандартний вибір. Java одна з найбільш поширених мов програмування з великим open source community. Spring Boot – це приклад хорошого open source рішення, яке є де-факто стандарт в Java світі. Загалом такий стек розробки дозволяє швидко нарощувати функціонал, легко шукати рішення для «стандартних» проблем та дозволяє швидко залучати менш досвідчених розробників.

В SMART кожен сервіс зберігається в окремому GIT репозиторії. Це дозволяє окремо керувати життєвим циклом мікросервісу. Кожен GIT репозиторій містить в собі .gitlab-ci.yml, який описує кроки, які необхідно виконати, щоб код був зібраний, протестований, пройшов реліз бібліотеки сервісу, а також сервіс був автоматично задеплоїний на сервер. Для автоматичної збірки коду репозиторії використовується Gitlab CI [6]. Нище наведено приклад конфігурації для сервісу document-management(Лістинг 1)

```
services:
  - docker:dind

variables:
  DOCKER_HOST: "tcp://docker:2375"
  DOCKER_TLS_CERTDIR: ""
  DOCKER_DRIVER: overlay2
  MAVEN_OPTS: "-Dmaven.repo.local=.m2/repository"
  SERVICE_FOLDER: "/var/ukma-servers/documents/"
  INPUT_JAR: "./document-management-service/target/document-management-
service.jar"
  OUTPUT_JAR: "document-management-service-new.jar"
  SERVICE_DIR: 'document-management-service'

cache:
  paths:
    - .m2/repository/

stages:
  - build
  - smart-docker-build
  - publish-snapshot
  - maven-release
  - deploy

include:
  - project: 'ukma/devops/ci-cd'
    file: 'build/build.yaml'
  - project: 'ukma/devops/ci-cd'
    file: 'build/publish.yaml'
  - project: 'ukma/devops/ci-cd'
    file: 'deploy/deploy.yaml'
```

Лістинг 1

В мікросервісній архітектурі частою проблемою є дублювання коду. Особливо це дублювання можна помітити, коли кожен окремий сервіс реалізує бібліотеку для REST запитів з іншим сервісами. Тобто якщо є сервіси «A» і «B», які роблять запити, до сервісу «C», то в сервісу «A» і «B» будуть свої власні бібліотеки для зв’язку з сервісом «C». В SMART вибрано підхід, коли кожен сервіс сам створює

бібліотеку, яка буде використовуватись іншим сервісами для інтеграції. Для цього використовується OpenAPI для декларації всіх REST API ендпоінтів, які існують в межах кожного сервісу. Нище наведено приклад yaml конфігурації для створення нового документу (Лістинг 2).

```
openapi: 3.0.0
info:
  description: Document Service
  version: "1.0"
  title: Document Service

tags:
  - name: document-controller
    description: Document Controller
servers:
  - url: http://localhost:8080

paths:
  /api/document:
    put:
      tags:
        - document-controller
      summary: Create document view
      operationId: createDocumentsView
      requestBody:
        required: true
        content:
          application/json:
            schema:
              type: DocumentView
      responses:
        200:
          description: Id of new document
          content:
            application/json:
              schema:
                type: UUIDResponse
```

Лістинг 2

Таким чином в SMART також реалізовано contract-first підхід, коли спершу пропонується опис контракту та його обговорення і тільки після цього його реалізація. В кожному сервісі використовується openapi-maven-generator [5], який з Open API yaml файлів генерує код для самого сервісу, а також генерує клієнтський код, який буде використовуватись іншими сервісами для інтеграції. Для того, щоб код з клієнтською бібліотекою був доступний для інших сервісів після кожного коміту відбувається автоматичний реліз проєкту за допомогою maven, а сам зібраний артефакт автоматично завантажується в Gitlab Package Registry [7].

2.4. Архітектура документообороту

Документообіг імплементовано в окремому сервісі document-management(Рисунок 2). Цей сервіс зберігає всю інформацію про документи та зв’язки між ними, а також використовує сервіс info-server для отримання інформації про користувачів системи.

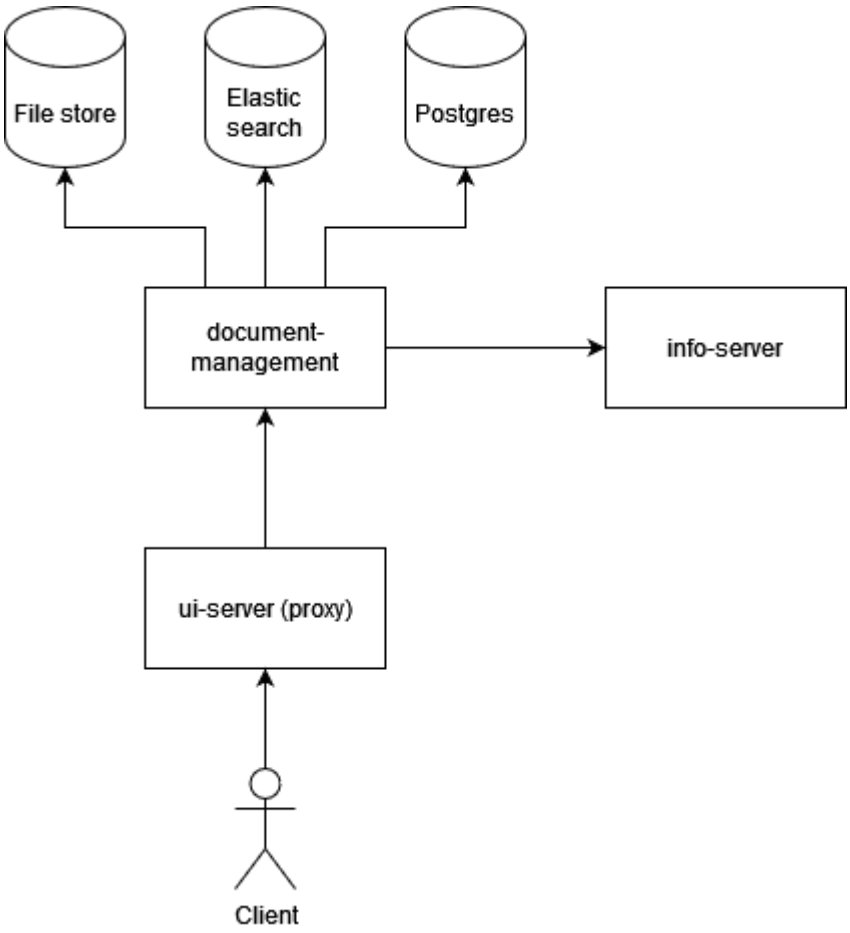

Рисунок 2

Сервіс документообігу використовує три сховища для збереження інформації. Перший це звичайна реляційна база даних, де зберігається основна інформація про документ: назва, тип, номер, виконавці, дата створення, дата погодження та інше. В якості реляційної бази даних було обрано Postgres. Для збереження основної інформації обрана реляційна база даних саме через можливість використовувати ACID. В системі документообігу важливе узгодження даних, тобто якщо з’явився

новий документ, який скасовує дію попереднього, то нам важливо зберегти цей зв’язок атомарно і тут нам якраз потрібні транзакції.

Однією з вимог до системи документообігу університету є можливість вільнотекстового пошуку по змісту самого документу чи його додатків. Для вирішення цієї задачі було обрано ElasticSearch, який є документ-орієнтованою базою даних, яка і була розроблена для того, щоб швидко і оптимально шукати інформацію у великих текстах. При завантажені нового документу в систему, текст цього документу разом з іншими додатковими полями потрапляє в індекс створений в середині ElasticSearch. Всі документу зберігаються в одному індексі, що дає можливість шукати будь-які документи за ключовими словами незважаючи на тип документа.

В якості третього сховища виступає файлова система, куди сервіс документообігу зберігає «чисті» PDF файли та підписані файли.

2.5. Структура бази даних

В реляційні базі даних зберігається основна інформація про документ, а також зв’язки з всіма іншими сутностями. Структура бази даних сервісу документообігу має такий вигляд (Рисунок 3):

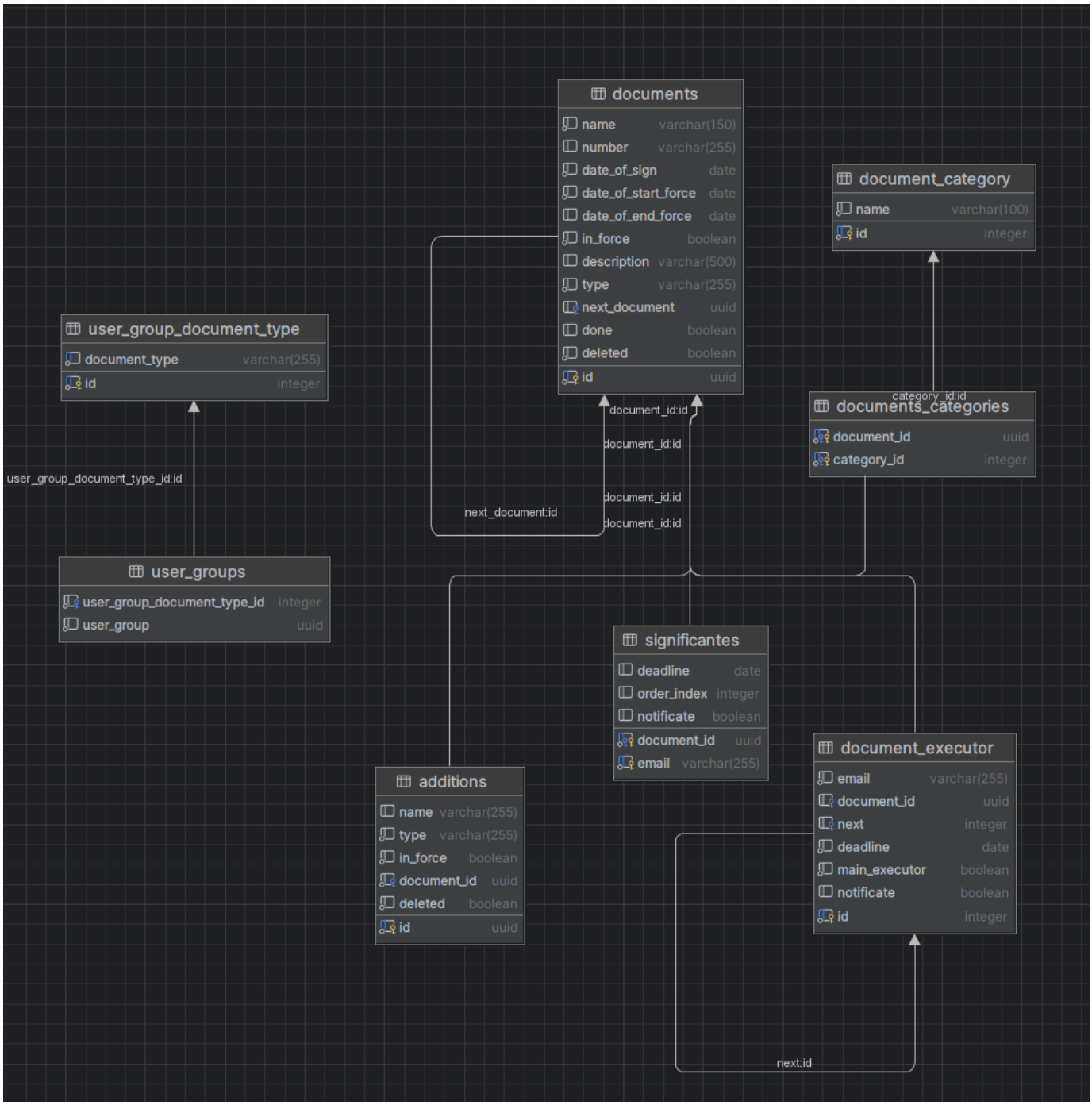

Рисунок 3

В документа є декілька виконавців, які впорядковані між собою зв’язним списком, де кожен виконавець знає про наступного виконавця. Також у виконавців є дедлайн, протягом якого необхідно виконати дію з їхнього боку.

В документу є поле next_document, яке вказує на наступний документ, який скасовує дію поточного. Підхід з «наступний документ», а не «попередній

документ» було обрано через те, що система має передбачати можливість одним новим документом скасувати дію декількох попередніх. Саме така реалізація дозволяє це легко зробити. Також її зручно використовувати і в зворотному вигляді: отримати список всіх попередніх документів, які були скасовані певним документом.

Доступ до документів відбувається на основі того чи група(-и), до якої належить певний користувач, має доступ до певного типу документа, якщо група(-и) користувачів мають такий доступ, то відповідно вони можуть переглянути цей документ, а також зробити з ним певні дії за умови наявності прав на такі дії. Самі групи користувачів зберігаються на стороні info-server.

РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ

3.1. Процес авторизації

На даний момент НаУКМА використовує систему Microsoft з обліковими записами створені в Microsoft. Було прийнято рішення для авторизації використати Microsoft Azure Active Directory[8]. Active Directory – це система розроблена Microsoft, яка дозволяє об’єднювати користувачів в групи і надавати певні доступи цим групам. Обрано саме такий підхід через ряд факторів:

- Так як всі облікові записи всіх студентів, викладачів і працівників вже існують в цій системі, то зникає потреба заново їх реєструвати.
- Можна використовувати авторизацію через існуючий обліковий Microsoft і тим самим уникнути всіх проблем пов’язаних з звичайною авторизацією через логін і пароль.
- Microsoft AD виступає в якості довіреного джерела інформації. Якщо користувач зміг авторизуватись в Microsoft AD з правильним доменним іменем, то ми вважаємо такого користувача «валідним».
- Microsoft AD для авторизації використовує відкритий протокол OAuth2, який є простий в інтеграції

Система SMART побудована таким чином, що в кожного користувача є певний набір ролей для кожного сервера: тобто окрема роль для info-server, окрема роль для document-management, окрема роль для course-work та інші. Це зроблено для того, щоб видавати користувачам право на роботу з певним сервером лише за потреби. Процес авторизації зображено на Рисунку 4.

Користувач на ініціює авторизацію на стороні ui-server після чого одразу ж відбувається перенаправлення сторінку Microsoft, де потрібно ввести логін та пароль до обліково запису. Якщо данні введені правильно, то користувач потрапляє назад на сторінку SMART, а ui-server отримує інформацію про пошту користувача, аксес токен та інше. UI-server використовує отриману пошту для того, щоб

отримати список ролей та прав цього користувача для кожного з серверів. В кінці користувач потрапляє на сторінку SMART, яка відображається у відповідності до того, до яких сервері користувач має доступ і які саме права він має в межах тих серверів.

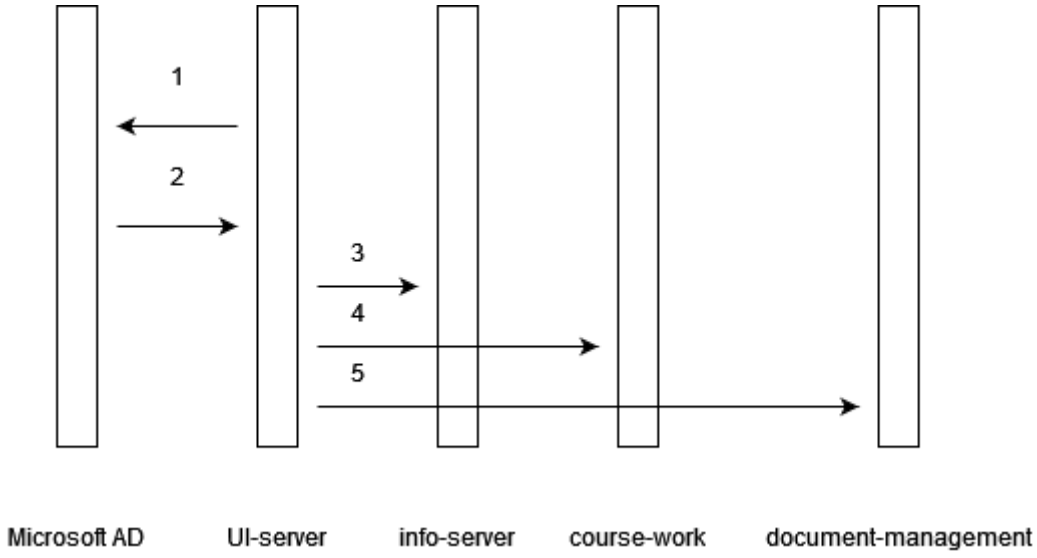

Рисунок 4

Для авторизації користувача користувача та побудови сесійного користувача на стороні імлементовано наступні методи (Лістинг 3). Такий підхід дозволяє на стороні ui-server робити первинну валідацію прав. Додатково кожен сервер робить свою додаткову валідацію, бо на стороні конкретного сервера є більше контексту. Наприклад: для запиту «редагувати курсову роботу» на стороні ui-server буде перевірка на наявність відповідного дозволу, а вже на стороні course-work сервера будуть додаткові перевірки на те чи може користувач редагувати саме цю курсову роботу і чи вона є в стані, коли редагування дозволено.

```
@Override
public OidcUser loadUser(final OidcUserRequest userRequest) throws OAuth2AuthenticationException {
    final OidcUser user = super.loadUser(userRequest);
    final String username = user.getAttribute(NAME_ATTRIBUTE_KEY);

    if (!StringUtils.hasText(username)) {
        log.error("Can't found email in attributes: {}", user.getAttributes());
        final OAuth2Error oAuth2Error = new OAuth2Error(INVALID_REQUEST_ERROR, "Can't find attribute
" + NAME_ATTRIBUTE_KEY + " in attributes", null);
        throw new OAuth2AuthenticationException(oAuth2Error);
```

```
    }

    Log.info("Sign-in user [{}]", username);
    final Map<ServerNameEnum, List<String>> permissionsPerServer = loadPermissionsForUser(username);
    Log.debug("Permissions per server for user {} is\n{}", username, permissionsPerServer);
    return new AuthenticatedUser(permissionsPerServer, user.getIdToken(), getUserInfo(username,
permissionsPerServer.get(ServerNameEnum.INFO_SERVER)), username);
}

private Map<ServerNameEnum, List<String>> loadPermissionsForUser(final String email) {
    final Map<ServerNameEnum, List<String>> permissionsPerServer = new HashMap<>();

    Optional.ofNullable(getRolePermissionsOnServer(email,
userInfoRoleController::getCurrentUserRole, ServerNameEnum.INFO_SERVER))
        .ifPresent(permissions -> permissionsPerServer.put(ServerNameEnum.INFO_SERVER,
permissions));
    Optional.ofNullable(getRolePermissionsOnServer(email,
courseWorkRoleController::getCurrentUserRole, ServerNameEnum.COURSE_WORK))
        .ifPresent(permissions -> permissionsPerServer.put(ServerNameEnum.COURSE_WORK,
permissions));
    Optional.ofNullable(getRolePermissionsOnServer(email,
documentRoleController::getCurrentUserRole, ServerNameEnum.DOCUMENTS))
        .ifPresent(permissions -> permissionsPerServer.put(ServerNameEnum.DOCUMENTS, permissions));
    Optional.ofNullable(getRolePermissionsOnServer(email, storageRoleController::getCurrentUserRole,
ServerNameEnum.STORAGE))
        .ifPresent(permissions -> permissionsPerServer.put(ServerNameEnum.STORAGE, permissions));
    Optional.ofNullable(getRolePermissionsOnServer(email,
uiConstructorRoleController::getCurrentUserRole, ServerNameEnum.UI_CONSTRUCTOR))
        .ifPresent(permissions -> permissionsPerServer.put(ServerNameEnum.UI_CONSTRUCTOR,
permissions));
    Optional.ofNullable(getRolePermissionsOnServer(email,
documentGeneratorRoleController::getCurrentUserRole, ServerNameEnum.DOCUMENT_GENERATOR))
        .ifPresent(permissions -> permissionsPerServer.put(ServerNameEnum.DOCUMENT_GENERATOR,
permissions));
    getUiServerRole(email)
        .ifPresent(role -> permissionsPerServer.put(ServerNameEnum.UI_SERVER, List.of(role)));

    return permissionsPerServer;
}

@SuppressWarnings("unchecked")
private static List<String> getRolePermissionsOnServer(String email, Function<String, MapResponse>
loader, ServerNameEnum serverName) {
    try {
        final MapResponse response = loader.apply(email);
        if (response.getError() != null) {
            Log.warn("Can't find role on {} server for email {}. error {}", serverName, email,
response.getError());
            return null;
        } else {
            Log.info("On server {} user {} has role-permissions {}", serverName, email,
response.getResult());
            Object permissions = Preconditions.checkNotNull(
                response.getResult().get("permissions"), "Role response from server " + serverName +
" doesn't contains permissions key"
            );
            return (List<String>) permissions;
        }
    } catch (Exception e) {
        Log.error("Exception while loading role-permissions on server {} for email {}", serverName,
email, e);
        return null;
    }
}
```

Лістинг 3

3.2. Реалізація серверної частини

Створення нового документу є складним процесом, до якого залучено декілька компонентів системи та декілька сховищ даних. Коли новий документ додається в систему, то система перш за все перевіряє чи користувач, який пробує це зробити має відповідні права. За умови, що в користувача є відповідні права система зберігає основну інформацію про документ в базу даних, зберігає файл на диск, а також відправляє файл на індексацію в ElasticSearch. Після того, як всі ці елементи системи відпрацювали автоматично відбувається відправка сповіщень на пошту всім зацікавленим особам: кореспондентам, виконавцям та тим, хто буде узгоджувати документ.

Для збереження даних на стороні сервера використовується ORM бібліотека JPA з провайдером Hibernate ORM. Ця бібліотека дозволяє зручно описувати відповідність полів з java об’єкту в колонки таблиць в базі даних. За допомогою цієї бібліотеки дуже зручно описувати не тільки відповідність об’ктів до таблиць, а також ж і відповідність зв’язків, які існують в базі даних (foreign keys). Нище наведено приклад опису зв’язків між документом та кореспондентами (Лістинг 4). Цей зв’язок сконфігурований в сутності DocumentEntity

```
@ElementCollection
@CollectionTable(name = "significantes", joinColumns = @JoinColumn(name = "document_id"))
private Set<SimpleExecutorEntity> significantes = new HashSet<>();;
```

Лістинг 4

Після збереження документу в базу даних для нього автоматично генерується UUID. Цей UUID використовується в системі для ідентифікації документа. Зокрема саме з цим UUID файл буде збережено на диск. Також з цим UUID файл буде проіндексовано.

Для роботи з ElasticSearch на стороні сервера використовується бібліотека з екосистеми Spring (Лістинг 5):

```
<dependency>
    <groupId>org.springframework.data</groupId>
    <artifactId>spring-data-elasticsearch</artifactId>
</dependency>
```

Лістинг 5

Ця бібліотека є обгорткою для Java HTTP клієнта для роботи з ElasticSearch. Робота з ElasticSearch також вимагає створення об’єкта, який буде відповідати документу з самого ElasticSearch. Для збереження документів в індекс використовується наступний java об’єкт (Лістинг 6):

```
@Getter
@Setter
@FieldNameConstants
@NoArgsConstructor
@Document(indexName = "document")
public class ElasticDocument {
    @Field(name = "id",type = FieldType.Text)
    private String id;

    @Field(type = FieldType.Keyword)
    private UUID documentId;

    @Field(type = FieldType.Keyword)
    private ElasticDocumentType type;

    @Field(type = FieldType.Text)
    private String text;

    @Id
    @AccessType(AccessType.Type.PROPERTY)
    public String getId() {
        return documentId.toString() + '-' + type;
    }

}
```

Лістинг 6

Цей об’єкт містить ідентифікатор документу, тип (документ або додаток) та текст документу.

Після того, як документ був збережений інші учасники системи можуть почати взаємодіяти з цим документом: переглядати, узгоджувати та інше.

3.3. Реалізація електронного підпису

Цифровий підпис відбувається на стороні клієнта в браузері. Такий підхід обрано для того, щоб не передавати по мережі приватний ключ та пароль користувача. Для цифрового підпису було обрано бібліотеку розроблену Інститутом Інформаційних Технологій (далі ІІТ) [9]. Процес накладання цифрового підпису виглядає наступним чином (Рисунок 5):

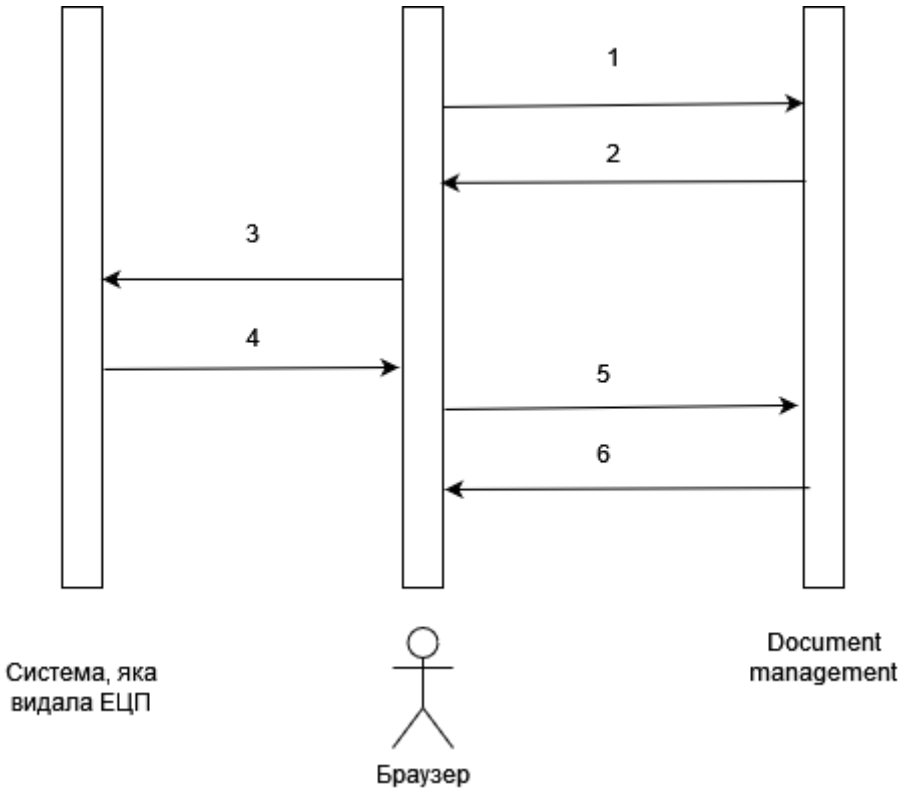

Рисунок 5

Процес накладання цифрового підпису складається з наступних кроків:

1. Користувач завантажив свій приватний ключ, ввів пароль та зробив запит на завантаження файлу
2. Сервер відправив користувачу файл, який потрібно підписати
3. Якщо приватний ключ і пароль введені правильно, то бібліотека ІІТ робить запит до системи, яка видала цей ключ користувачу.
4. Система відправляє цифровий підпис у вигляді base64 рядка. Це є оригінальний файл з підписом
5. Браузер відправляє запит на збереження підписаного файлу
6. Система відповідає, що все зберегла

Після цих кроків файл підписаний, а сам підписаний файл можна завантажити і перевірити.

Нище наведено код на стороні клієнта, який відповідає за обробку процесу накладання цифрового підпису (Лістинг 7):

```
submit() {
  forkJoin({
    keyFile: this.readPrivateKey(),
    signFile: this.store.documentService.getDocumentFile(this.documentId)
      .pipe(
        mergeMap(response => this.readFile(response)),
        catchError(e => {
          this.toast.handleError({
            code: 0,
            message: 'Помилка підчас завантаження файлу документу'
          });
          this.stateMessage = '';
          return of({error: e})
        })
      )
  })
    .pipe(
      mergeMap(signData => {
        this.stateMessage = 'Підпис документу'

        return from(this.euSignFile.SignDataInternal(true, signData.signFile.data, true));
      }),
      catchError(e => {
        this.toast.handleError({
          code: 0,
          message: (e?.message || e) || 'Помилка підчас підпису документу'
        })
        return of(null)
      })
    )
    .subscribe(sign => {
      if (sign) {
        this.stateMessage = 'Збереження підпису'
        this.store.documentService.signDocument(this.documentId, {sign: sign})
          .subscribe(
            response => {
              this.stateMessage = '';
              this.toast.success('Документ підписано');
              this.activeModal.close(true);
            },
            error => {
              this.toast.handleError(error.error as ErrorDto);
              this.activeModal.close(false);
            }
          )
      }
    })
}

private initialize() {
  let self = this;
  return new Observable(subscriber => {
    self.euSignFile.IsInitialized()
      .then(function (result) {
        if (result) {
          subscriber.next();
          subscriber.complete();
          return;
        }
```

```
        return self.euSignFile.Initialize(self.euSettings)
      }).then(function () {
      subscriber.next();
      subscriber.complete();
    })
      .catch(function (e) {
        subscriber.error(e);
        subscriber.complete();
      });
  });
}

private readPrivateKey(): Observable<any> {
  this.stateMessage = 'Зчитування ключа';

  return new Observable(subscriber => {
    let password = this.formGroup.controls.password.value;
    let euSignFile = this.euSignFile;
    if (!this.formGroup.controls.keyFile.value || !password) {
      subscriber.error({
        message: 'Виберіть файл приватного ключа та вкажіть пароль'
      });
      subscriber.complete();
      this.stateMessage = '';
      return;
    }

    this.readFile(this.formGroup.controls.keyFile.value)
      .pipe(
        map(
          keyFileData => {
            if (keyFileData.file.name.endsWith(".jks")) {
              return euSignFile.GetJKSPrivateKeys(keyFileData.data)
                .then(function (jksKeys) {

                  return euSignFile.ReadPrivateKeyBinary(jksKeys[0].privateKey, password);
                });
            }

            return euSignFile.ReadPrivateKeyBinary(keyFileData.data, password);
          }
        ),
        catchError(e => {
          subscriber.error(e);
          subscriber.complete();
          return of(null);
        })
      )
      .subscribe(result => {
        subscriber.next(result);
        subscriber.complete();
      })
  });
}

private readFile(file): Observable<any> {
  return new Observable(subscriber => {
    let reader = new FileReader();
    reader.onloadend = function (evt) {
      if (evt.target.readyState != FileReader.DONE)
        return;

      subscriber.next({
        file: file,
        data: new Uint8Array(evt.target.result)
```

```
      });
      subscriber.complete()
    };
    reader.readAsArrayBuffer(file);
  });
}
```

Лістинг 7

3.4. Реліз та проміжні версії

Як зазначалось в пункті 2.3 в SMART реалізований процес розробки таким чином, що він дозволяє після кожного коміту в гілку master отримувати резіл версію. Це зроблено для того, щоб у випадку, коли в коміті є зміни, які впливаються на REST API даного сервісу, то це означає, що потрібно заново згенерувати код клієнтської бібліотеки і використовувати нову версію в сервісах, які будуть використовувати нову версію. Для цього використовується наступна конфігурація в Gitlab CI (Лістинг 8):

```
maven-release:
  stage: maven-release
  image: maven:3.6.3-openjdk-11
  only:
    - master
  except:
    variables:
      - $CI_COMMIT_MESSAGE =~ /maven-release-plugin/ || $SKIP_RELEASE == "true"
  before_script:
    - mkdir -p ~/.ssh/
    - cp $DEPLOY_PRIVATE_KEY ~/.ssh/id_rsa && chmod 600 ~/.ssh/id_rsa
    - cp $KNOWN_HOSTS ~/.ssh/known_hosts
    - apt-get update && apt-get install -y git
    - git config --global user.email "no_reply_smart@ukma.edu.ua"
    - git config --global user.name "Smart UKMA"
  script:
    - git checkout master
    - git remote set-url origin git@gitlab.com:${CI_PROJECT_PATH}.git
    - git pull origin master
    - mvn ${MAVEN_CLI_OPTS} -B --no-transfer-progress release:prepare
release:perform -s $SETTINGS_XML -DbranchName=${CI_COMMIT_REF_NAME} -
Dfrontend.skip=true -Darguments="-Dmaven.javadoc.skip=true -Dmaven.test.skip=true
-DskipTests=true -Dfrontend.skip=true ${MAVEN_CLI_OPTS}"
    - git push origin master:master --follow-tags
```

Лістинг 8

При розробці рішення з використанням мікросервісної архітектури часто виникає потреба зробити змінити в двох і більше сервісах. Якщо в сервісі А є зміни REST API, а сервіс В має використовувати ці нові зміни, то виникає потреба на

етапі розробки використовувати проміжні зміни сервісу А без межу змін в основну гілку. Для цього проміжні версії, які створюються для кожної гілки в git. Наприклад, якщо поточно версія в сервісі є 2.1.10-SNAPSHOT і розробник завантажив гілку my-new-feature, то проміжною версією буде 2.1.10-my-new-feature-SNAPSHOT. Для створення проміжних версій в системі Gitlab CI є наступна конфігурація (Лістинг 9)

```
publish-snapshot:
  stage: publish-snapshot
  image: maven:3.6.3-openjdk-11
  when: manual
  script:
    - current_version=$(mvn help:evaluate -Dexpression=project.version -q -
DforceStdout)
    - echo $current_version
    - new_version="${current_version/SNAPSHOT/$CI_COMMIT_REF_NAME}-SNAPSHOT"
    - echo $new_version
    - mvn versions:set -DnewVersion=$new_version -Dfrontend.skip=true
    - mvn ${MAVEN_CLI_OPTS} -B --no-transfer-progress clean package deploy -
DskipTests=true -Dfrontend.skip=true -s $SETTINGS_XML
  except:
    - master
```

Лістинг 9

ВИСНОВОК

В процесі написання кваліфікаційної роботи було проведено аналіз предметної області документообігу і зокрема ознайомлення з особливостями документообігу університету. На основі вимог було запропоноване рішення для електронного документообігу, яке було інтегровано в існуючу систему SMART. В другому розділі було розглянути особливості розробки та інтеграції в системі SMART. Також було надано детальний опис накладання цифрового підпису на документи.

Загалом результати кваліфікаційної роботи надають імплементацію рішення, яке відповідає заданим вимогам та є інтегрованим з існуючими системами. Система електронного документообігу допоможе прибрати людський фактор на багатьох етапах запровадження нових документів в університеті. Дана роботу є важливим внеском в розвиток внутрішніх систем НаУКМА.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Документообіг[Електронний ресурс] – 2023 Режим доступу https://uk.wikipedia.org/wiki/%D0%94%D0%BE%D0%BA%D1%83%D0%BC%D0%B5%D0%BD%D1%82%D0%BE%D0%BE%D0%B1%D1%96%D0%B3
2. Що таке електронний цифровий підпис (ЕЦП)? [Електронний ресурс] – 2024 Режим доступу https://www.kmu.gov.ua/usi-pitannya-po-e-poslugam/sho-tak-elektronnij-cifrovij-pidpis-ecp
3. National Institute of Standards and Technology [Електронний ресурс] – 2024 Режим доступу https://en.wikipedia.org/wiki/National_Institute_of_Standards_and_Technology
4. Акредитований центр сертифікації ключів [Електронний ресурс] – 2023 Режим доступу https://uk.wikipedia.org/wiki/%D0%90%D0%BA%D1%80%D0%B5%D0%B4%D0%B8%D1%82%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B9_%D1%86%D0%B5%D0%BD%D1%82%D1%80_%D1%81%D0%B5%D1%80%D1%82%D0%B8%D1%84%D1%96%D0%BA%D0%B0%D1%86%D1%96%D1%97_%D0%BA%D0%BB%D1%8E%D1%87%D1%96%D0%B2
5. OpenAPI Generator [Електронний ресурс] – 2024 Режим доступу https://openapi-generator.tech/
6. Get started with Gitlab CI/CD [Електронний ресурс] – 2024 Режим доступу https://docs.gitlab.com/ee/ci/
7. Maven packages in the package registry [Електронний ресурс] – 2024 Режим доступу https://docs.gitlab.com/ee/user/packages/maven_repository/
8. Active Directory[Електронний ресурс] – 2024 Режим доступу https://en.wikipedia.org/wiki/Active_Directory

9. Типи документів НаУКМА [Електронний ресурс]. /Andrii Hlybovets, Oleksandra Mykhailenko// – 2024 Режим доступу https://app.clickup.com/20593116/v/dc/kmeew-587/kmeew-427
10. АТ «Інстритут Інформаційних Технологій» [Електронний ресурс] – 2024 Режим доступу https://iit.com.ua
11. ЕЛЕКТРОННИЙ ДОКУМЕНТООБІГ В СИСТЕМІ УПРАВЛІННЯ ПІДПРИЄМСТВОМ [Електронний ресурс]. / Кравченко О.В., Ткаченко А.А./ - 2018 Режим доступу https://er.chdtu.edu.ua/bitstream/ChSTU/1254/1/31-2018.pdf
12. ЕЛЕКТРОННИЙ ДОКУМЕНТООБІГ, ТЕНДЕНЦІЇ ТА ПЕРСПЕКТИВИ [Електронний ресурс]. / М.Б. Величкевич, Н.В. Мітрофан, Н.Е. Кунанець/ - 2010 Режим доступу https://science.lpnu.ua/sites/default/files/journal-paper/2019/apr/16036/vis689ism-44-53.pdf