МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

КРИВОРІЗЬКИЙ ДЕРЖАВНИЙ ПЕДАГОГІЧНИЙ УНІВЕРСИТЕТ

Факультет фізико-математичний

Кафедра інформатики та прикладної математики

«Допущено до захисту»
Завідувач кафедри
_____ Соловйов В. М.
(підпис)

«___» _____ 20__ р.

Реєстраційний № _____

«___» _____ 20__ р.

ВЕБ-ФРЕЙМВОРК ДЛЯ СТВОРЕННЯ СИСТЕМ ГЕНЕРАЦІЇ ТИПОВОЇ ДОКУМЕНТАЦІЇ

Кваліфікаційна робота студента групи
Ім-16
ступінь вищої освіти «магістр»
спеціальності
014.09 Середня освіта (інформатика)
Загородька Павла Володимировича

Керівник к. ф.-м. н., доц.
Мерзликін Павло Володимирович

Оцінка:
Національна шкала _____
Шкала ECTS ____ Кількість балів ____
Голова ЕК _____ _____
(підпис) (прізвище, ініціали)

Члени ЕК _____ _____
(підпис) (прізвище, ініціали)

_____ _____
(підпис) (прізвище, ініціали)

_____ _____
(підпис) (прізвище, ініціали)

_____ _____
(підпис) (прізвище, ініціали)

Кривий Ріг – 2021

ЗАПЕВНЕННЯ

Я, Загородько Павло Володимирович, розумію і підтримую політику Криворізького державного педагогічного університету з академічної доброчесності. Запевняю, що ця кваліфікаційна робота виконана самостійно, не містить академічного плагіату, фабрикації, фальсифікації. Я не надавав і не одержував недозволену допомогу під час підготовки цієї роботи. Використання ідей, результатів і текстів інших авторів мають покликання на відповідне джерело. Із чинним Положенням про запобігання та виявлення академічного плагіату в роботах здобувачів вищої освіти Криворізького державного педагогічного університету ознайомлений. Чітко усвідомлюю, що в разі виявлення у кваліфікаційній роботі порушення академічної доброчесності робота не допускається до захисту або оцінюється незадовільно.

Зміст

ВСТУП

Розвиток сучасних технологій впливає на всі аспекти людського життя. Однією з таких сфер є сфера бізнесу. Організації по всьому світу використовують нові цифрові досягнення в області технологій для впровадження рішень автоматизації, здатних відтворювати дії людини, усуваючи рутинні завдання і, таким чином, перетворюючи роботу працівників в більш цінну [37].

Досить багато організацій, підприємств та навчальних закладів мають справу з великою кількістю процесів. Розвиток у такому випадку буде сприяти розширенню, що в свою чергу підтверджує потребу у автоматизації. Генерація документації є одним з першим векторів автоматизації [20]. Зазвичай, створення інструментарію для обробки та автоматизації робочого процесу документообігу є складним та комплексним процесом, але, він виправдовує себе.

Згідно дослідницького відділення McKinsey Global Institute міжнародної консалтингової компанії McKinsey & Company [24], що від 9 до 26 відсотків робочого часу може бути заощаджено за допомогою автоматизації. А до 2030 року до 30 відсотків поточних робочих місць буде витіснене, з медіаною в п'ятнадцять відсотків, що еквівалентно близько 400 мільйонам повних робочих днів.

Проблема автоматизації документації також має місце в лікарнях. Так, на сайті про медичні технології Electronic Health Reporter в статті [36], опублікованій Стівеном Вілсоном, описано, як саме автоматизація впливає на медичну сферу. Лікарні працюють з досить великою кількістю різних документів – договорів з лікарем та лікарських посвідчень до табелів обліку робочого часу та інших організаторських форм. Велику кількість таких документів досить складно зберігати та знаходити за потреби, якщо це паперові версії документів. Ще одним, не менш вагомим фактором на користь автоматизації документації є робота з пацієнтами. Використання електронних форм для заповнення дозволяє лікарям сконцентруватися саме на роботі, а не заповнені паперових документів.

Також пацієнти з легкістю зможуть отримувати електронні версії рецептів з будь-якого девайсу без проблем.

З вищесказаного випливає **актуальність** роботи.

Об’єкт дослідження: цифровий документообіг.

Предмет дослідження: автоматизація генерації типової документації.

Мета: створити веб-фреймворк для генерації типової документації. Для досягнення мети дослідження були поставлені такі **задачі**:

- проаналізувати сучасний стан і проблеми автоматизації цифрового документообігу;

- порівняти наявні на ринку програмні рішення, визначити їх переваги й недоліки;

- висунути функціональні вимоги до веб-фреймворку;

- вибрати інструменти розробки;

- спроектувати серверну (back-end) складову;

- спроектувати клієнтську (front-end) складову;

- розробити структуру бази даних;

- реалізувати опрацювання шаблонів документів;

- дослідити швидкодію розробленого фреймворку;

- продемонструвати його роботу на конкретних задачах документообігу;

- створити інструкцію користувача.

Новизна роботи полягає в тому, що в результаті її виконання створено новий веб-фреймворк, здатний працювати з найпоширенішими форматами електронних документів і, на відміну від аналогів, здатний генерувати форми з документів та працювати із різними видами ключових слів в залежності від

потреб користувача. Також пацієнти з легкістю зможуть отримувати електронні версії рецептів з будь-якого пристрою.

Практичне значення роботи полягає в тому, що створений веб-фреймворк може використовуватися для налагодження генерації типової документації в рамках організації.

Апробація:

- за результатами роботи подано тези на 4th Workshop for Young Scientists in Computer Science & Software Engineering (CS&SE@SW 2021).

- на основі веб-фреймворку створено систему генерації додатків до дипломів, яка використовується Криворізьким державним педагогічним університетом.

Структура роботи. Робота складається з вступу, трьох розділів, висновків та списку використаних джерел (49 найменуваннь).

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Проблеми автоматизації документообігу

Автоматизація документообігу являє собою комплексну проблему, тому спершу розглянемо основні поняття та те, чому автоматизації – невіддільний атрибут будь-якого сучасного підприємства чи закладу.

Документ, згідно словника Merian-Webster[2], можна визначити, як комп’ютерний файл, який містить у собі написаний текст. Більш формальним є визначення Міжнародної організації із стандартизації (ISO). Згідно до ISO 12651-2 [44], документ - записана інформація або об'єкт, який можна розглядати як одне ціле.

Більш широким терміном, який і визначає поняття автоматизації документообігу, є система управління документами (англ. Document Management System) – використання комп’ютерної системи та програмного забезпечення для зберігання, управління та контролю електронної документації та електронних зображень паперової інформації за допомогою сканеру документів. Виходячи з цього поняття, система автоматизації документообігу (англ. Document Automation System) – це система для управління електронною документацією. Тобто, система автоматизації документообігу є частиною системи управління документами.

Ручна обробка документів викликає велику кількість проблем[39]:

1. Низька ефективність. Створення документа є трудомісткою задачею. Окрім внесення окремих даних, людині потрібно брати до уваги форматування тексту та даних, вибірка пов’язаних даних, обробка яких потребує час. Такі проблеми потребують витрату великої кількості часу на створення одного документа;

2. Помилки. Людина може помилятися з різних причин: неуважність, людський фактор, неможливість врахувати всі аргументи;

3. Захист. Без системи управління документами важно відслідковувати, що саме відбувається з документом у певний момент часу, хто має до нього доступ, та хто його редагував. Усі ці фактори збільшують вірогідність викрадення документів або конфіденційних даних.

Тепер розглянемо основні проблеми автоматизації документообігу для індустрії та бізнесу [29].

1. Ризик втрати роботи. Велика кількість людей сьогодні мають страх втрати своєї роботи через автоматизацію, та їх страх не є безпричинним. Дослідженням цієї проблеми займалась міжнародна мережа компаній з надання професійних послуг у сфері консалтингу та аудиту PricewaterhouseCoopers[48]. Так, в їхньому дослідженні під назвою «Will robots really steal our jobs?»[18] показані сфери, у яких співробітники можуть втратити свою роботу. У порядку спадання виділимо найбільш важливі:

- транспортування та зберігання;
- виробництво;
- будівництво;
- сервіси адміністрування та підтримки;
- оптова та роздрібна торгівля;

2. Ціна автоматизації. Повноцінна автоматизація документообігу потребує багато часу та коштів, але далекій перспективі дозволяє економити час та кошти.

3. Гнучкість системи. В залежності від системи автоматизації, кожна має свої налаштування та степінь налаштування. Проблема в додаванні нового функціоналу може потребувати додаткового часу.

Розглянемо основні проблеми в автоматизації документації з погляду програмної реалізації:

1. Підбір інструментів. Наразі існує невелика кількість Open-Source бібліотек для повноцінної обробки документів з урахуванням усього функціоналу;

2. Існування декількох стандартів текстових документів. Беручи до уваги популярні текстові редактори Microsoft Word та LibreOffice Writer, можна зазначити, що кожен з них має окремий стандарт для обробки документів. Кожен зі стандартів є обширним та великим за обсягом, що викликає проблему створення універсальних обробників;

3. Проблеми з обробкою XML-файлів документів.

Розглянемо останню проблему детальніше. Надалі, буде використано назву Document XML Keywords Split Issue (Проблема розриву ключових слів в XML документі)

У своїй основі, кожен документ формату Microsoft Word та LibreOffice Writer має структуру архіву. В такому архіві описана основна структура документу у вигляді XML-коду. Розглянемо проблему на прикладі документів різних форматів з однаковим текстом.

Текст буде таким: {Test data}.

Структуру документу docx можна бачити на Рис. 1.1.

Рис. 1.1. XML-код файлу Microsoft Word.

Червоним кольором виділено частину з нашим текстом. Можна помітити, що хоча й в оригінальному файлі текст записаний в один рядок, у XML-коді перший символ знаходиться в окремому XML-тегі, а вся інша частина в іншому. Тепер розглянемо файл LibreOffice Writer (Рис 1.2).

Рис. 1.2. XML-код файлу LibreOffice Writer.

Тут ми бачимо, що дані збереглися як одне ціле. Тепер спробуємо відредагувати обидва документи, додавши туди такий текст: Data: {Test_ 2}. Результати можна побачити на Рис. 1.3 та Рис. 1.4.

```
        <w:r>
            <w:rPr>
                <w:lang w:val="en-US" />
            </w:rPr>
            <w:t>{Test data}</w:t>
        </w:r>
        <w:r w:rsidR="00FF610A">
            <w:t xml:space="preserve"></w:t>
        </w:r>
        <w:r w:rsidR="0046710B">
            <w:rPr>
                <w:lang w:val="en-US" />
            </w:rPr>
            <w:t xml:space="preserve">Data: </w:t>
        </w:r>
        <w:r w:rsidR="00C16C1D">
            <w:rPr>
                <w:lang w:val="en-US" />
            </w:rPr>
            <w:t>{Test</w:t>
        </w:r>
        <w:r w:rsidR="0046710B">
            <w:rPr>
                <w:lang w:val="en-US" />
            </w:rPr>
            <w:t>_</w:t>
        </w:r>
        <w:bookmarkStart w:id="0" w:name="_GoBack" />
        <w:bookmarkEnd w:id="0" />
        <w:r w:rsidR="00C16C1D">
            <w:rPr>
                <w:lang w:val="en-US" />
            </w:rPr>
            <w:t>2}</w:t>
        </w:r>
    </w:p>
```

Рис. 1.3. Файл Microsoft Word після зміни

```
<office:body>
    <office:text>
        <text:sequence-decls>
            <text:sequence-decl text:display-outline-level="0" text:name="Illustration" />
            <text:sequence-decl text:display-outline-level="0" text:name="Table" />
            <text:sequence-decl text:display-outline-level="0" text:name="Text" />
            <text:sequence-decl text:display-outline-level="0" text:name="Drawing" />
            <text:sequence-decl text:display-outline-level="0" text:name="Figure" />
        </text:sequence-decls>
        <text:p text:style-name="P1">
            {Test data}
            <text:span text:style-name="T1">Data: </text:span>
            {Test_
            <text:span text:style-name="T2">2</text:span>
            }
        </text:p>
    </office:text>
</office:body>
```

Рис. 1.4. Файл LibreOffice Writer після зміни

На обох рисунках можна бачити те, що з’явилися розриви слів, які насправді не розірвані в текстовому редакторі.

У випадку з Microsoft Word розриви з’являються випадково без чіткого шаблону. Але, з LibreOffice Writer ситуація трохи інша: після зміни вже збереженого тексту з’являються такі розриви слів.

На перший погляд, така проблема здається незначною, але при зростанні обсягу документу це буде викликати значні проблеми. У цьому випадку, шаблонами можна вважати текст, який розташований між фігурними дужками. Якщо уявити, що таких шаблонів буде не 2, а 200 або 2000, то перегляд коректності та виправлення кожного шаблону буде займати велику кількість часу.

Підсумовуючи, можна бачити, що обробка шаблонів таким чином може викликати багато проблем, саме тому перш ніж рухатись до реалізації перейдемо до огляду наявних реалізацій для обробки документів.

1.2 Огляд наявного програмного забезпечення для автоматизації документообігу

Дослідивши наявні реалізації, зараз можна виділити декілька видів програмного забезпечення для автоматизації документообігу:

1. Пропрієтарне API для обробки документів;

2. Системи автоматизації генерації документів;

3. Системи управління документами;

4. Desktop-орієнтоване програмне забезпечення для автоматизації документообігу (переважно застаріле);

5. Cloud-орієнтоване програмне забезпечення для автоматизації документообігу;

Найбільш цікавими для нашого дослідження будуть системи автоматизації генерації документів та хмарне програмне забезпечення для автоматизації документообігу.

Спочатку оглянемо пропрієтарні системи автоматизації генерації документів, сюди входять і API.

Hypatos[15][38] – система автоматизації документообігу з використанням штучного інтелекту за допомогою технології Cognitive process automation (CPA). Є досить високоякісним та професійним інструментом. Підтримка AWS та роботи в хмарних сховищах. Доступне API та безкоштовна версія.

PandaDoc[27] – система для автоматизації документів, заявок та контрактів. Особливість сервісу полягає в можливості електронних підписів, які називаються eSign. Є підтримка спеціальних форм, інтеграції з сервісами, single sign-on (SSO), Salesforce інтеграція, можливість створення «робочих потоків» (workflow) для управління процесом обробки документів. Приклад документу можна бачити на Рис. 1.5.

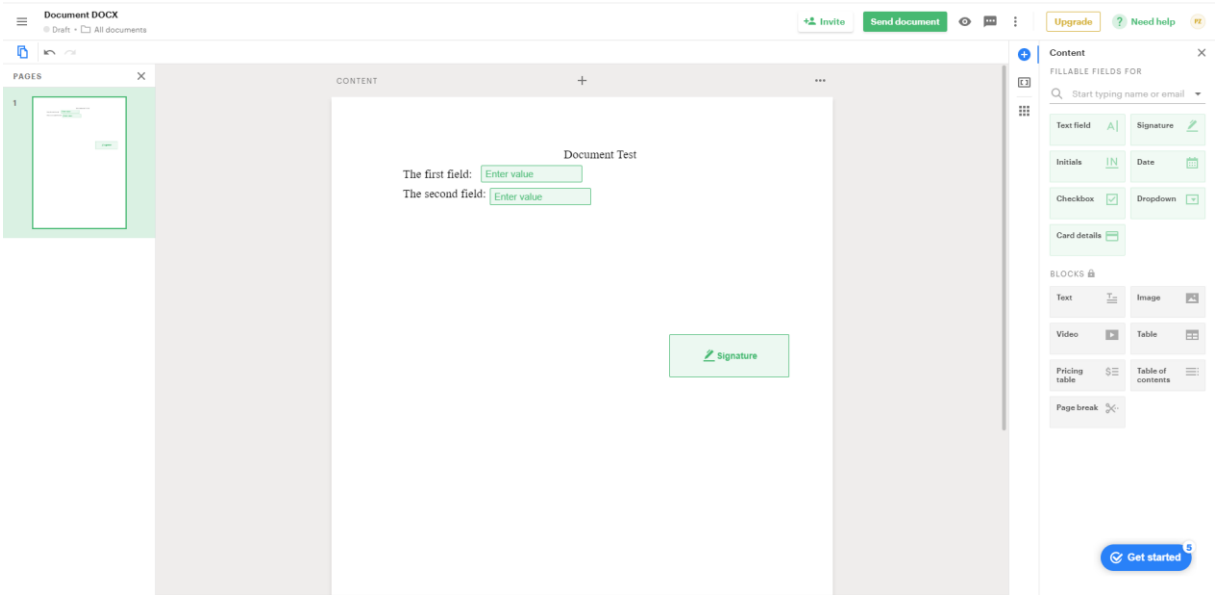

Рис. 1.5. Приклад документу в системі PandaDoc.

DocuPhase[12] – система для автоматизації бізнес процесів. Є підтримка веб-форм, за допомогою яких можна генерувати готові PDF-файли. Також система управління документами зі зручним інтерфейсом для обміну та управлінням файлами різними відділеннями.

Docupilot[11] – система автоматизації та генерації документації. Підтримка роботи с такими сервісами як Zapier, DropBox, Docusign. Хороший шаблонізатор з підтримкою умовних операторів, таблиць та циклів. Робота с docx, pptx, pdf або з використанням власного шаблону за допомогою WYSIWYG редактора. Підтримується відправка повідомлення на пошту (email). Є документація та приклади використання внутрішнього API. Приклад простого Invoice-документу можна бачити на Рис. 1.6.

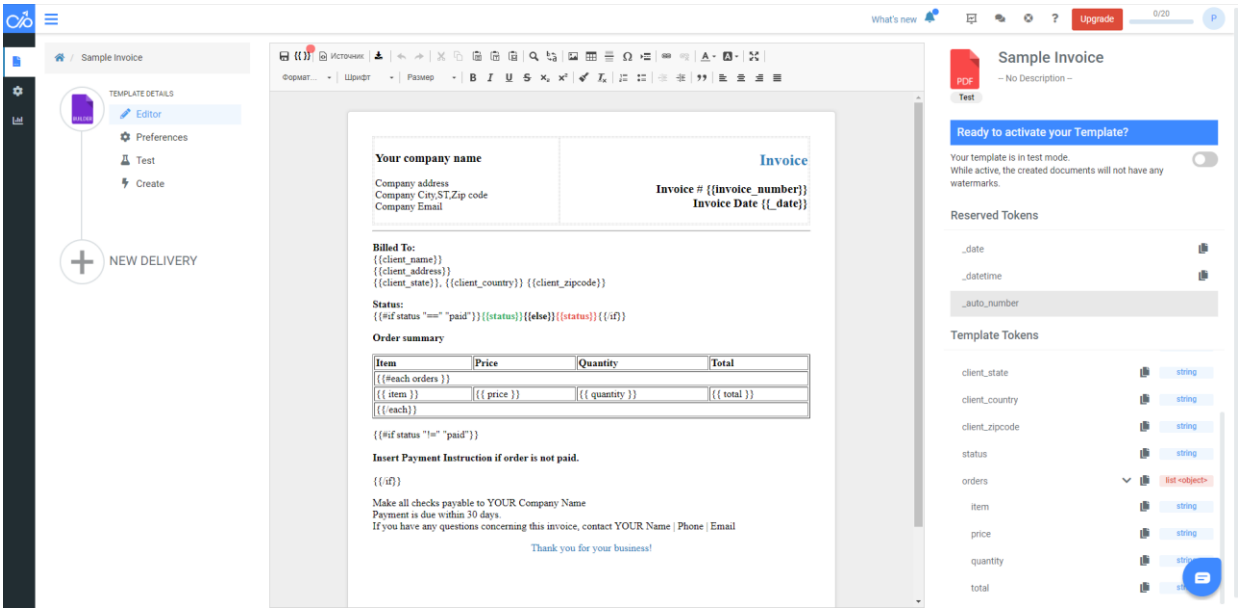

Рис. 1.6. Invoice документ в сервісі Docupilot.

Contactbook[9] – платформа для організації, збереження та роботи з документами. Сервіс включає в себе роботу з файлами типу docx та pdf. Реалізована інтеграція з 3000+ програмами. Є доступ до публічного API.

Juro[19] – програмне забезпечення для автоматизації роботи з різними контрактами. Існує можливість створення власного процесу ухвалення документів, історія змін документів та аналітика, а також можливості OCR. Приклад документу в системі Jura показано на Рис 1.7.

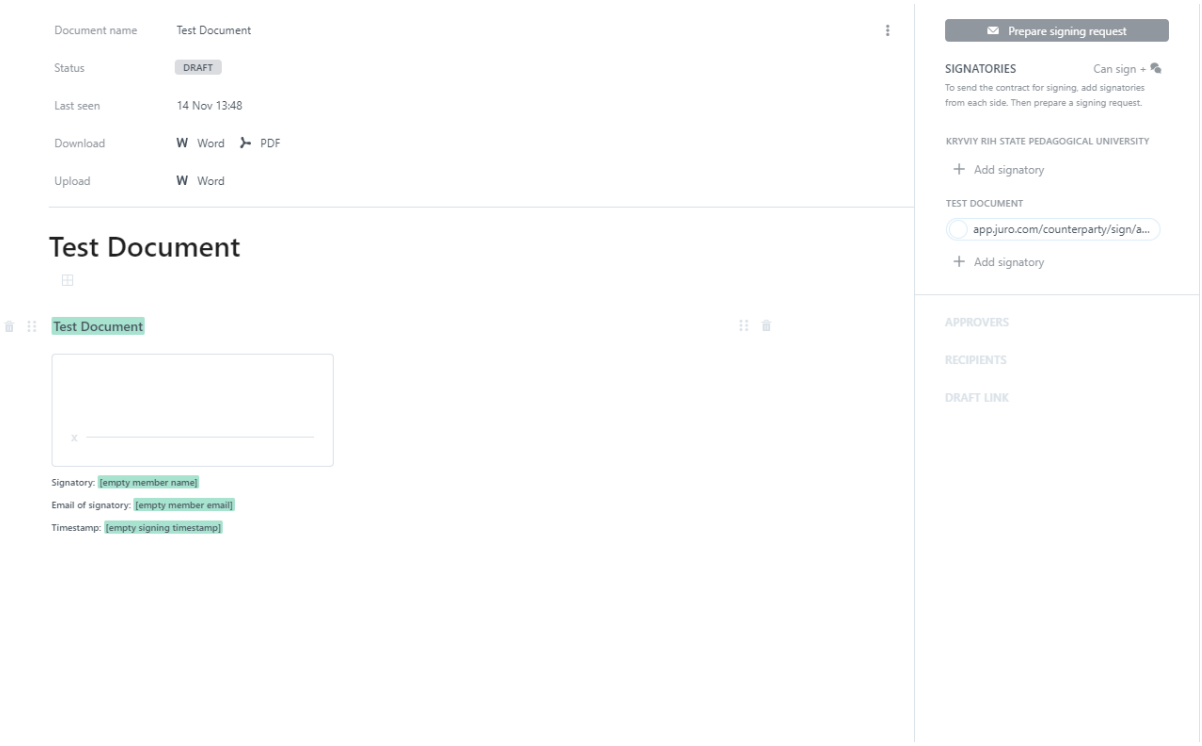

Рис. 1.7. Документ в системі Juro.

OnTask[26] – програмне забезпечення для автоматизації роботи з документами, можливістю створювати форми, та Workflow схеми. Існують можливості організації електронних підписів, захист документів, шаблони документів, шаблон для перевірки вакцинації робітників щепленням проти Covid-19. Приклад форми та Workflow схеми для перевірки вакцинації показано на Рис. 1.8 та Рис. 1.9.

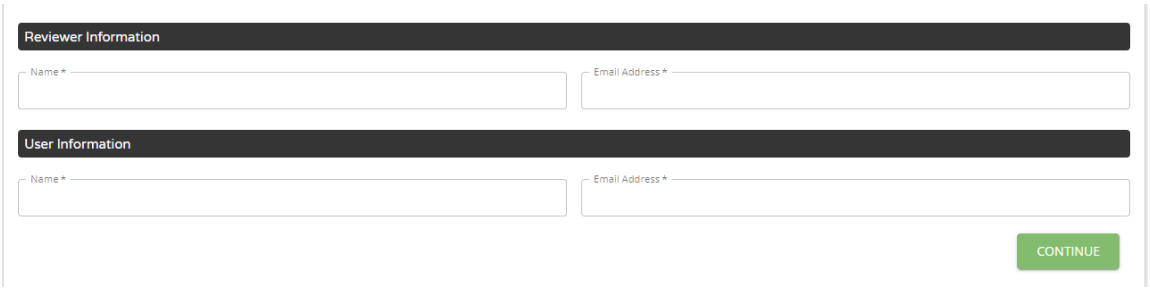

Рис. 1.8. Приклад форми сервісу OnTask.

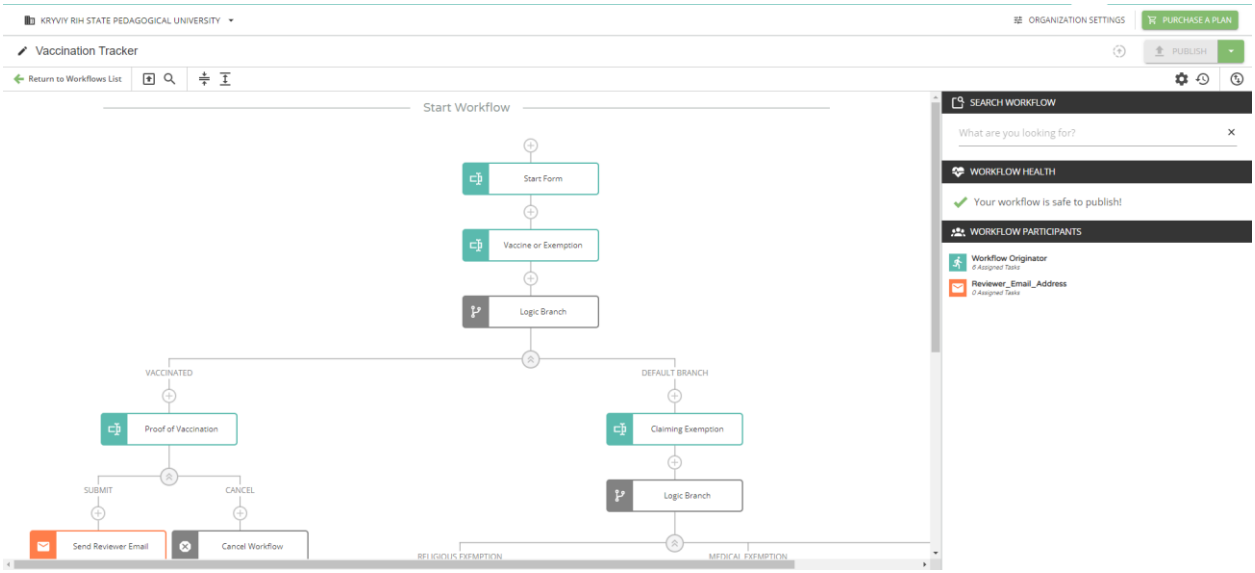

Рис. 1.9. Workflow-схема процесу для перевірки вакцинованих робітників

Тепер розглянемо програмне забезпечення, яке є відкритим.

Одним з таких є Docassemble[10] – відкрита система для роботи з веб-формами та документами. Система організована на Python, YAML та Markdown. Система орієнтована на «Інтерв'ю» питання. Тобто, одна веб-форма розділена на декілька питань, і в кінці можна отримати результат. Є підтримка написання чистого коду в YAML конфігураційних файлах. За допомогою Markdown можна динамічно створювати файли форматів PDF, RTF та DOCX.

M2Doc[23] – відкритий настільний застосунок для автоматизації роботи з файлами типу MS Word. Має додаток до MS Word та Eclipse IDE. Генератор працює по принципу обробки даних з генераційної конфігурації .genconf. Дозволяє працювати з оригінальним API на Java.

1.3 Постановка задачі

Виходячи із аналізу наявних підходів до обробки документів, нами було обрано використання LibreOffice Uno API для обробки документів.

Ми будемо розробляти фреймворк, за допомогою якого можна буде автоматизувати генерацію документів для користувачів. За допомогою клієнтської частини, користувач буде завантажувати документи, після чого

документ буде відправлятися до серверної частини. Після завантаження користувач може вказати початок та кінець ключових слів для знаходження, або скористатися стандартними. Після обробки файлу генерується форма, за допомогою якої користувач зможе згенерувати та отримати свій файл у відповідь.

Для більш ефективної обробки ми будемо реалізовувати два варіанти: перший –буде обробка файлу без сторонніх бібліотек, а інший із використанням бібліотеки LibreOffice Uno API.

Висновки до розділу 1

1. Автоматизація документообігу є проблематичним та складним процесом, але й є значно ефективнішим у порівнянні із класичною документацією.

2. Існує велика кількість різних програм та бібліотек для автоматизації документообігу. Більшість із них орієнтовані на якусь конкретну ціль, як-от автоматизація електронних підписів, створення форм опитування, додаткова логіка обробки документів та інші. Такі застосунки корисні, якщо потрібно автоматизувати якусь конкретну складову документообігу, або створити послідовний процес обробки документу різними людьми.

3. Аналіз наявних реалізацій показав, що їх використання у чистому виглядів не є можливим для повноцінної обробки багатьох типів документів. Таким чином, найбільш функціональним та придатним до використання є LibreOffice Uno API. Але, позаяк документи є не завжди добре структурованими, використання спеціальних обробників XML-коду є важливим етапом в обробці документів.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ ВЕБ-СИСТЕМИ ДЛЯ СТВОРЕННЯ СИСТЕМ ГЕНЕРАЦІЇ ТИПОВОЇ ДОКУМЕНТАЦІЇ

2.1. Обґрунтування вибору інструментів розробки

Оглянута в розділі 1.2. проблема демонструє, що обробка документів у чистому вигляді є проблематичною. Для вирішення цієї проблеми проаналізуємо деякі бібліотеки, призначені для обробки документів.

Таблиця 2.1.

Бібліотеки та фреймворки для роботи із документами

Бібліотека	Підтримка форматів	Опис
XDocReport[49]	docx, odt	Бібліотека базується на XML обробці коду документів.
Apache Poi[7]	docx, pptx, xlsx	Бібліотека для обробки різних файлів форматів Office Open XML формату.
ODF Toolkit[25]	odt, odf, xslx	Бібліотека для обробки документів OpenDocument Format. Включає в себе декілька інших бібліотек для перевірки правильності XML-коду документа, перетворення схем та перетворення XSLX в XML.
Aspose[13]	docx	Закритий фреймворк для обробки різних видів документів. Має API для різних мов програмування.
Syncfusion Java Word Library[17]	docx, dotx, docm,	Закрита бібліотека для роботи із WordML документами. Має

	dotm, html, txt	обширний функціонал для різних версій Microsoft Office.
LibreOffice SDK[21]	odt, docx та інші[14]	Бібліотека для обробки великої кількості документів. Являє собою майже весь функціонал пакету програм LibreOffice.
NPOI[5]	xls, xlsx, docx	Бібліотека для обробки Microsoft Office Word та Excel файлів. Є версією Apache POI для мови C#
Spire.Doc for .Net[1]	doc, docx	Бібліотека для мови C#, яка охоплює версії Word 97-03, Word 2007, Word 2010, Word 2013, Word 2016, Word 2019. Має обширний конвертор.

Аналіз цих програмних засобів показав, що найбільш функціональним та зручним для використання є LibreOffice SDK, який і буде використовуватися для подальшої розробки. Оскільки LibreOffice SDK підтримує декілька мов програмування для розробки нами було вибрано Java, яка підтримується.

Виходячи із вибору мови та інструменту, потрібно вибрати відповідний засіб для розробки front-end частини та back-end.

Для розробки front-end складової можна зупинитися на одні із 3 популярних бібліотек[4] – Angular, React, Vue.js.

Загалом, всі три бібліотеки підтримують на створення односторінкових застосунків (SPA) та звичайних веб-сторінок. Тому порівняємо їхню популярність згідно Stack Overflow на рис 2.1., 2.2 та 2.3.

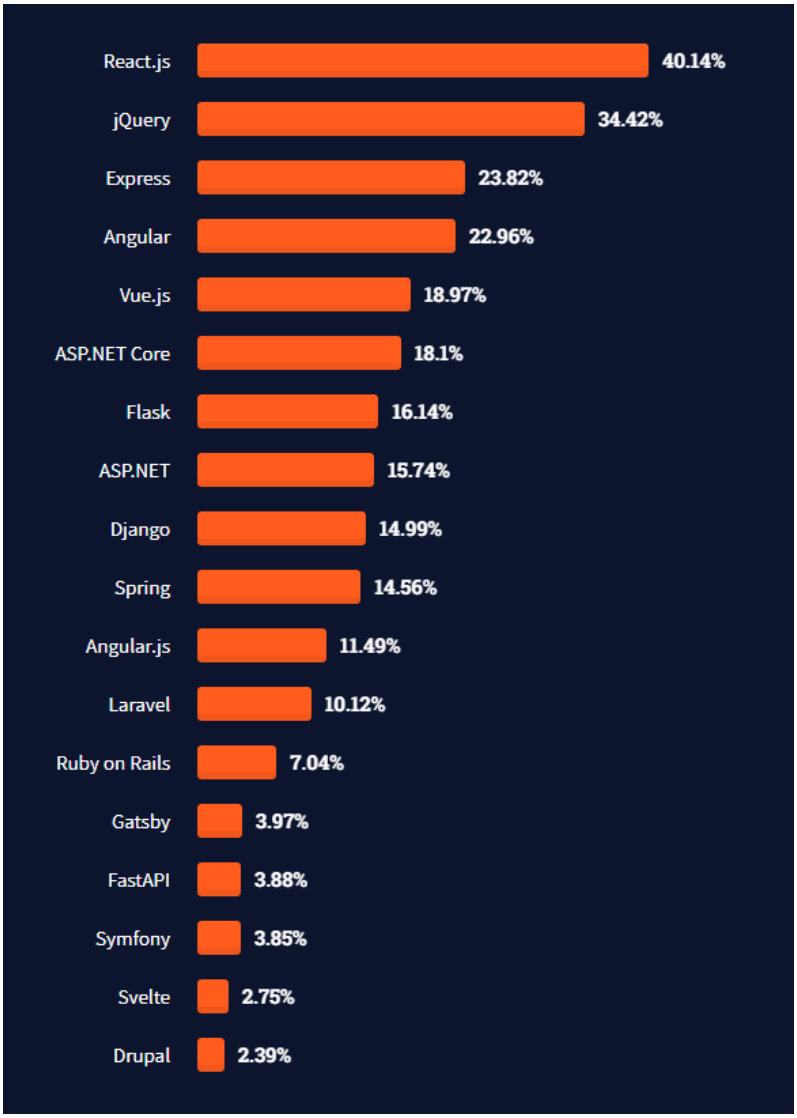

Рис. 2.1. Популярність веб-фреймворків із урахуванням всіх респондентів[33]

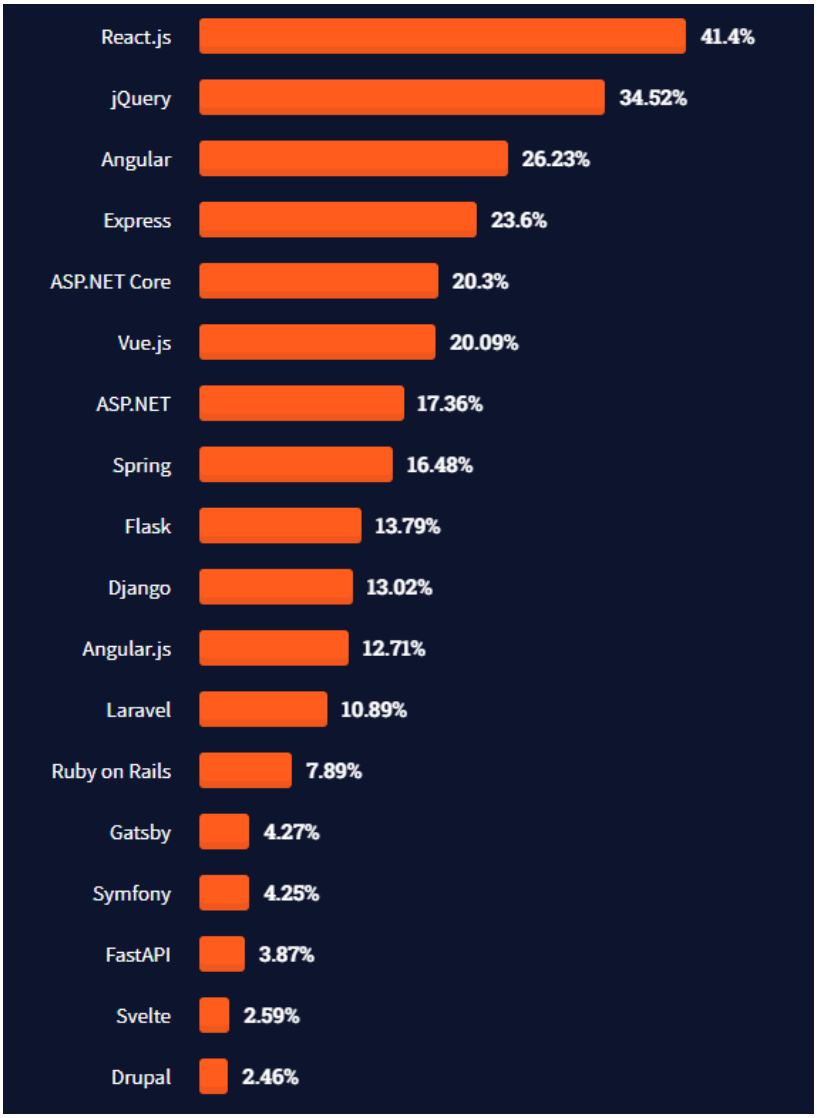

Рис. 2.2. Популярність веб-фреймворків із урахуванням тільки професіональних розробників[33]

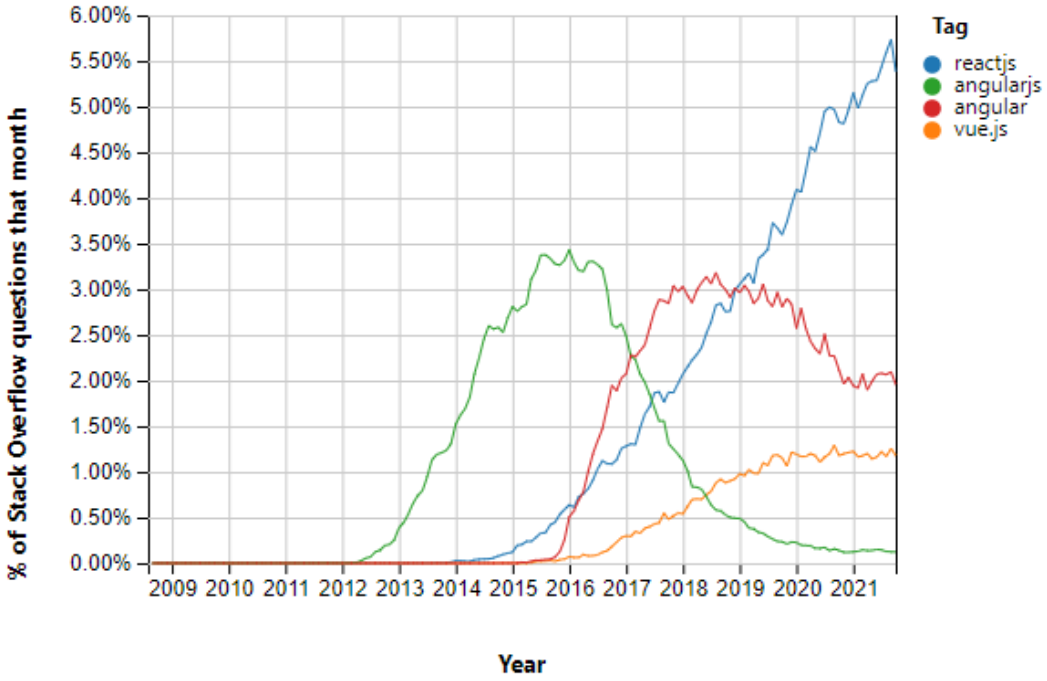

Рис. 2.3. Графік росту популярності фреймворків [34]

Загалом, можна бачити тенденцію, що React іде першим, а Angular та Vue.js далі. Але, позаяк їх функціонал приблизно однаковий, тому нами було обрано Vue.js через те, що він є простим та популярним останнім часом. А також, підходить під потреби клієнтської частини.

Для back-end частини також було проаналізовано популярні фреймворки. Згідно списку фреймворків[3] найбільш популярним та функціональним є фреймворк Spring.

2.2. Проектування back-end складової системи

Для проектування back-end складової системи вирішено було використовувати мікросервісну архітектуру.

Загалом, було вирішено розділити на такі мікросервіси:

1. Мікросервіс для входу та генерації токенів;

2. Мікросервіс для обробки документів (збереження та управління документами);

3. Мікросервіс для генерації документів відповідно до заповнених даних.

Розглянемо кожен мікросервіс окремо.

Мікросервіс входу та генерації токенів буде відповідати за журналювання користувачів, та роботу із токенами. Як основний механізм для входу на сайт було обрано JWT (JSON Web Token). Порівнявши плюси та мінуси використання JWT[47], можна дійти висновку, що його використання є ефективним та надійним для мікросервісної архітектури.

Мікросервіс має окрему кінцеву точку (end-point) для перевірки коректності переданого токену.

Для повноцінної реалізації перевірки кожного компоненту потрібно виконати такі кроки:

1. Створити точку входу, яка буде реалізувати інтерфейс org.springframework.security.web.AuthenticationEntryPoint;

2. Створити фільтр, який наслідує клас OncePerRequestFilter:

- отримаємо заголовок Authorization запиту;

- перевірка чи не порожній заголовок Authorization;

- використовуючи Spring WebFlux, та його реалізацію WebClient відправляємо запит до кінцевої точки для перевірки правильності токену;

- перевіряємо чи є повернений запит коректним, і якщо так, то за допомогою UsernamePasswordAuthenticationToken класу задаємо SecurityContext аутентифікацію в SecurityContextHolder.

3. Створити конфігураційний клас, який наслідує WebSecurityConfigurerAdapter та реалізує WebMvcConfigurer;

4. Додати відповідну точку входу та фільтр.

Мікросервіс для обробки документів створений для виконання основних операцій над документами – створення, збереження, отримання записів, а також роботи із шаблонами документів.

Мікросервіс для генерації документів буде використовувати розроблену бібліотеку для обробки документів за допомогою LibreOffice SDK.

Для реалізації шару роботи із базою даних було використано Spring JPA[32]. Приклад реалізації моделі показано на Рис. 2.4.

```
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String username;
    private String email;
    private String password;
    private String role;


    public User(String username, String email, String password) {
        this.username = username;
        this.email = email;
        this.password = password;
    }

    public User() {

    }

    public void setId(Long id) {
        this.id = id;
    }

    public Long getId() {
        return id;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getEmail() {
        return email;
    }
```

Рис. 2.4. Реалізація моделі користувача за допомогою Spring JPA.

2.3. Проектування front-end складової системи

Основою для розробки front-end системи було обрано Webpack [43] у ролі комплектатор модулів, та Babel [8] у ролі компілятора мов програмування JavaScript та TypeScript.

За основу було взято фреймворк для створення веб-сторінок Vue.js 3[42].

Для реалізації маршрутизації було взято один із проектів Vue.js Vue Router[40]. Він є обов’язковою складовою для авторизації користувачів.

Для реалізації авторизації потрібно використовувати управління станами програми[35] (State Management). Виникає така потреба через те, що ми будемо використовувати SFC[31] (Single File Component) у Vue.js. Single File Component – спеціальні файли із розширенням .vue, які дозволяють розділяти стилі, логіку та шаблон документу. Приклад найпростішого компоненту показано на Рис. 2.5.

```vue
<template>
    <Link class="header-button" :to="to"><slot></slot></Link>
</template>

<script lang="ts">
import Link from '@/components/common/Link.vue'
import { defineComponent } from '@vue/runtime-core'

export default defineComponent({
    components: { Link },
    props: {
        to: String
    }
})
</script>

<style scoped>
    .header-button {
        height: inherit;
        color: black;
        line-height: 55px;
        margin-right: 1.5vw;
        text-decoration: none;
        font-family: Verdana, Geneva, Tahoma, sans-serif;
    }
</style>
```

Рис. 2.5. Приклад SFC.

Як можна бачити, файл поділений на три частини. Перша частина це template, це html-код компоненту. Наступна частина – це саме скрипт компоненту розташований в script. І останній це стиль компоненту розташований у style.

Використання компоненту є простим процесом, як приклад можна розглянути компонент Link. Для його використання його потрібно імпортувати, а потом використовувати як звичайний тег, якщо його ім’я ціле, то його можна писати з великої букви, якщо ж ніж, то його ім’я пишеться через тире, як це показано на Рис 2.6.

```vue
<template>
    <main>
        <welcome-page-header/>
        <hr class="line"/>
        <what-we-provide-section/>
        <hr class="line"/>
        <welcome-page-footer/>
    </main>
</template>
<script>
import WelcomePageHeader from '../welcome-page/WelcomePageHeader.vue';
import WhatWeProvideSection from '../welcome-page/WhatWeProvideSection.vue';
import WelcomePageFooter from '../welcome-page/WelcomePageFooter.vue';

export default {
    components: {
        WelcomePageHeader,
        WhatWeProvideSection,
        WelcomePageFooter
    }

}
</script>
```

Рис. 2.6. Приклад використання kebab-case синтаксису

У останньому випадку використовується kebab-case[45] синтаксис компонентів – використання тире як роздільника слів. У випадку назв компонентів використовується PascalCase[28] – кожне слово пишеться із великої букви без пробілів.

Оскільки потрібно об’єднувати SFC між собою для передачі даних, тому було створено бібліотеку Vuex[46]. Ця бібліотека дозволяє повноцінно зберігати дані між компонентами без проблем. Приклад її використання можна бачити на Рис 2.7.

```
const allowedPaths: string[] = ['/', '/signin', '/signup'];
router.beforeEach((to, from, next) => {
    if (allowedPaths.includes(to.fullPath) || store.getters['isAutheticated']()) {
        next();
    } else {
        next({ path: '/signin' });
    }
});
```

Рис. 2.7. Приклад перевірки авторизації користувача із використанням Vue Router та Vuex

За допомогою хуку beforeEach, ми перехоплюємо кожен виклик та перевіряємо, чи є ця сторінка доступною для всіх, чи потрібно авторизація. Як Саме в останнього випадку використовується конструкція store.getters[‘isAuthenticated’]. Ця функція перевіряє чи є у користувача відповідний токен та чи можна його використовувати.

Для використання іконок для сайту було вирішено використовувати безкоштовну версію бібліотеки Vue FontAwesome[41].

Позаяк частина компонентів є складними та важкостворюваними, було вирішено використовувати частину функціоналу фреймворку для графічного інтерфейсу Element Plus[6].

2.3. Структура бази даних

Як основу для бази даних було обрано MySQL.

Схему бази даних можна бачити на Рис 2.8.

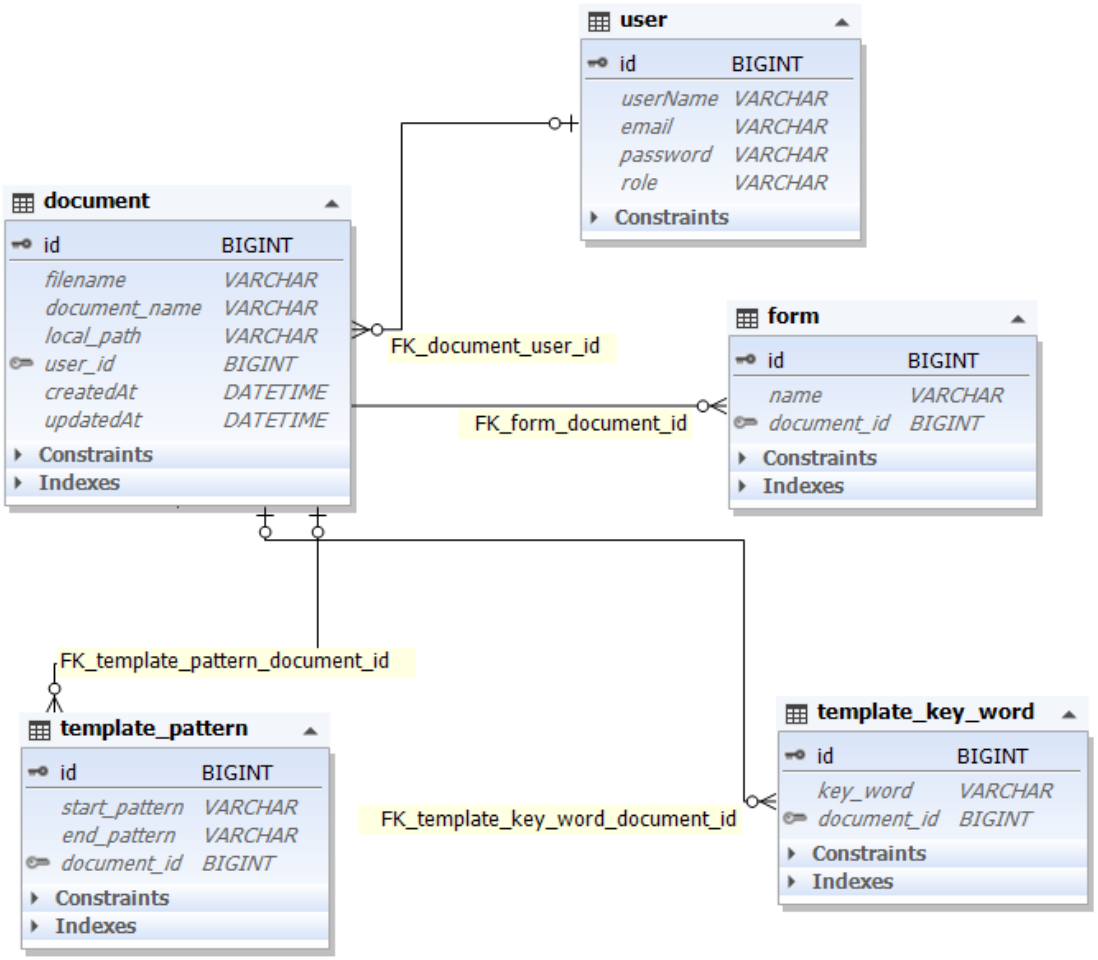

Рис. 2.8. Схема бази даних проєкту

Розглянемо кожну таблицю окремо.

- user – таблиця для збереження даних про користувачів;

- document – таблиця для збереження даних про документи. Сам документ зберігається у файловій системі, а його шлях вказано в полі local_path;

- template_pattern – таблиця для збереження шаблону для документів. Зберігається початок та кінець ключового слова для конкретного документа;

- template_key_word – таблиця для збереження самого ключового слова шаблону документу для зменшення часу на повторний пошук його у документі;

- form – таблиця для збереження даних про форму, згенеровану на основі полів шаблону документу.

2.4. Обробка шаблонів документів

Для обробки документів було обрано два підходи:

1. XML-обробка документів;

2. обробка документів з використання LibreOffice SDK.

Було розроблено спеціальну бібліотеку, яка об’єднує в собі обидва підходи. Бібліотека називається Lycorse Document Processing Library (далі Lycorse DPL).

Розглянемо більш докладно реалізацію та використання першого підходу.

Перше, що потрібно вирішити перед обробкою документів – це Document XML Keywords Split Issue (DXKS проблема), яка була описана у першому розділі. Для того, щоб вирішити поставлено проблему, перше, що потрібно зрозуміти – внутрішню структуру документів.

Документи типу Microsoft Word та LibreOffice Writer мають просту структуру, по своїй суті, вони є архівами, які зберігають всередині себе структуру документу у вигляді XML-коду (рис. 2.9, 2.10).

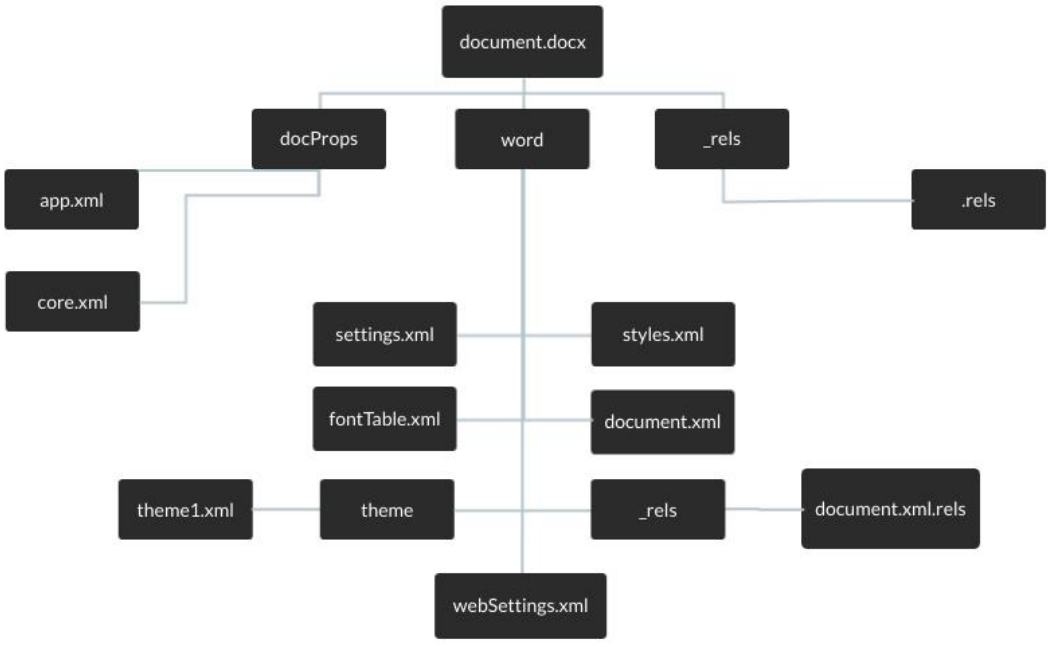

Рис 2.9. Схема документу Microsoft Word

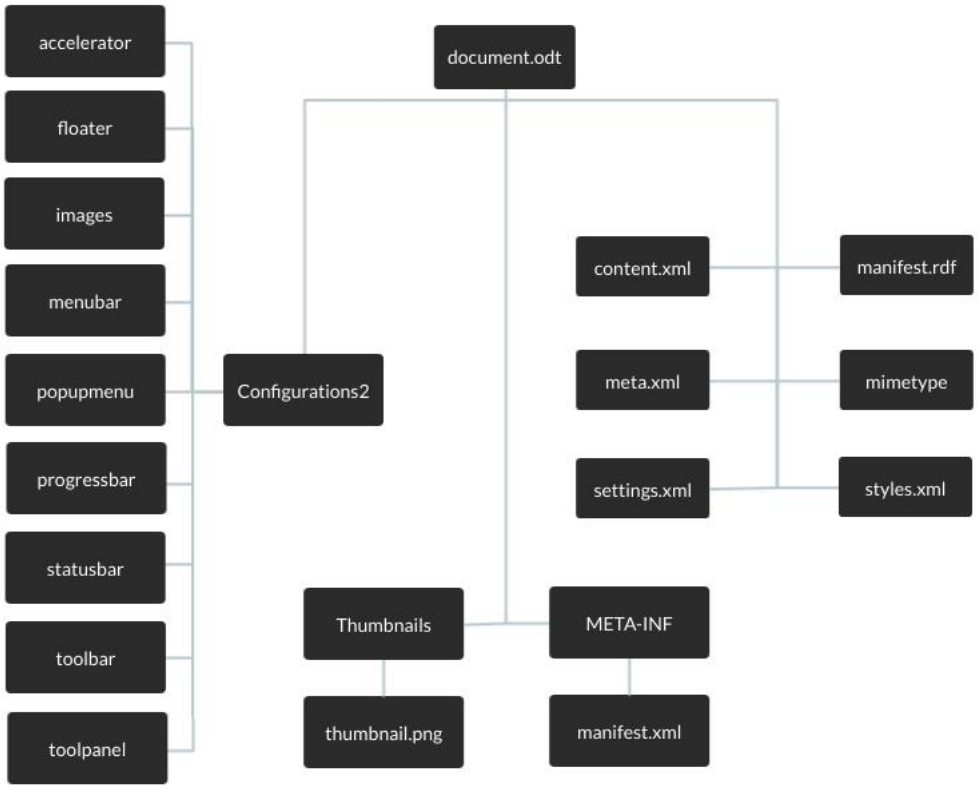

Рис. 2.10. Схема документу LibreOffice Writer

Як можна бачити, структура та файли є схожими та не мають суттєвих розбіжностей. Найбільш цікавими для нас є document.xml у випадку Microsoft Word, та content.xml у випадку LibreOffice Writer. Обидва файли зберігають в собі весь текст документу у вигляді XML-коду.

Виходячи із такої структури, зрозуміло, що найпростіший спосіб відредагувати документ, – це виділити файл змісту документа, замінити його та заново завантажити в той же архів.

Рухаючись до реалізації, нам потрібно розглянути алгоритм знаходження ключових слів:

1. Знаходимо початок ключового слова та кінець;

2. Створюємо зміну для контролю зсувів в тексті;

3. Вибираємо частину тексту, яка доступна до початку XML-тегу;

4. Знаходимо початок іншого XML-тегу;

5. Перевіряємо, чи є текст після XML-тегу;

6. Якщо текст є, то видаляємо його із вмісту документу та додаємо до початкового; якщо ні, то рухаємося далі;

7. Якщо досягнуто кінця ключового слова, то перевіряємо, чи є ще ключові слова; якщо ні, то завершуємо, якщо так, то повертаємося до кроку 1.

Загалом, алгоритм є простим та не складним, а також дозволяє швидко обробляти документ. Нижче показано приклад реалізації такого алгоритму не заглиблюючись у деталі.

```
        private String formPatternText() {
            startOfPattern = findStartOfPattern();
            endOfPattern = findEndOfPattern();
            offset = startOfPattern;
            StringBuilder result = new
StringBuilder(getTextBeforeNextTag(startOfPattern));
            startOfPatternOffset = startOfPattern + result.length();
            while (endOfPattern >= offset) {
                result.append(findAndRemoveNextText());
            }
            return result.toString();
        }
```

Загальний приклад використання розробленого класу:

```
    OdtDocumentPatternsAdjust odtDocumentPatternsAdjust = new
OdtDocumentPatternsAdjust(new Pattern("${", "}"));
    odtDocumentPatternsAdjust.adjustPatterns(ResourcesManager.getResourceFile(
"Test_Document.odt"));
```

Тепер розглянемо, як можна обробляти документи за допомогою LibreOffice SDK.

Загалом, LibreOffice SDK працює за принципом запуску графічного інтерфейсу та відтворення відповідних дій. Тобто, фактично, всі дії, які можна відтворити вручну за допомогою графічного інтерфейсу, можна виконати і за допомогою коду. Загальна схема обробки документа показана на Рис. 2.11.

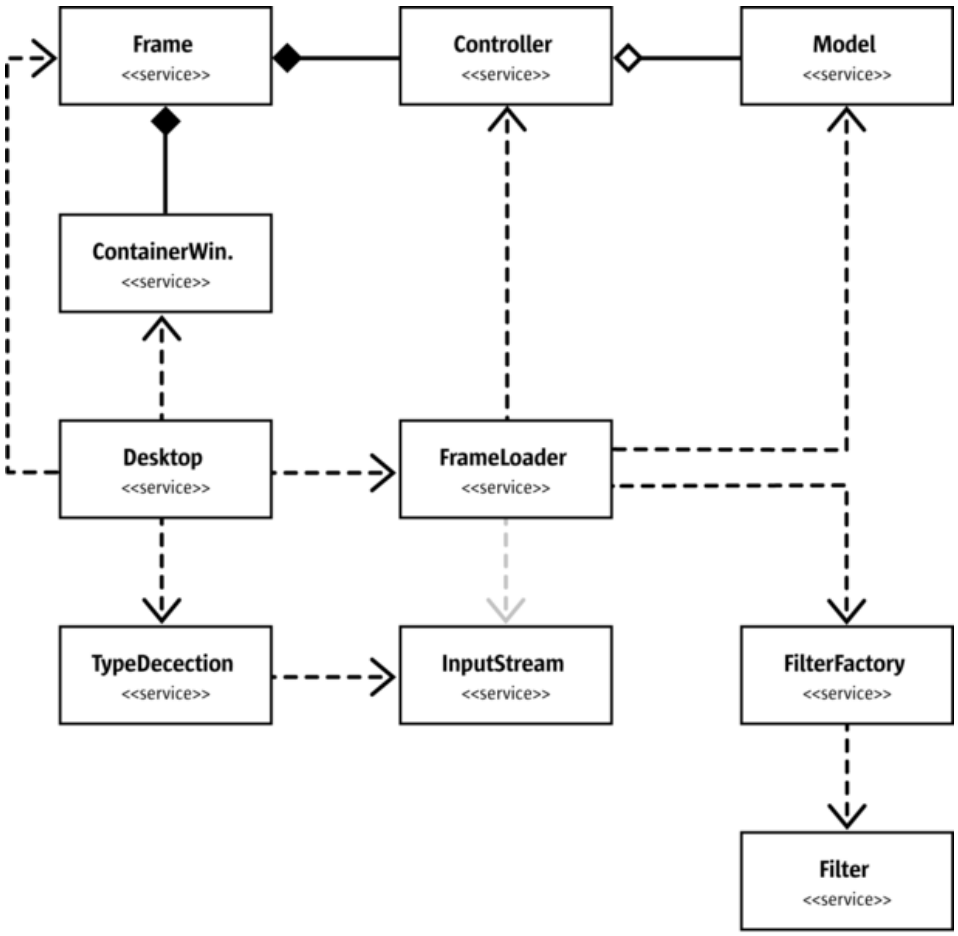

Рис. 2.11. Сервіси, задіяні в обробці документа за допомогою LibreOffice SDK[16]

Власне завантаження відбувається за допомогою класу com.sun.star.frame.XComponentLoader, в якому є один метод, сигнатура якого наведена нижче.

```
com::sun::star::lang::XComponent loadComponentFromURL ( [in] string
aURL,
                    [in] string aTargetFrameName,
                    [in] long nSearchFlags,
```

```
                    [in] sequence < com::sun::star::beans::PropertyValue
aArgs > )
```

Перший аргумент aURL це саме шлях до файлу. Другий аргумент aTargetFrameName використовується для вказівки, яке вікно потрібно завантажувати. Оскільки процес є складним, подальші деталі буде пропущені. Третій аргумент nSearchFlags має схожий до другого функціонал. Останній аргумент aArgs є досить цікавим. Він вказує, яким чином повинен оброблятися документ. Тобто, сюди входить багато різних вхідних та вихідних вказівок, наприклад, в якому режимі відкривати документ. Всю структуру доступних вказівок реалізовано за допомогою сервісу com.sun.star.document.MediaDescriptor [30].

Загалом, описане завантаження документу можна реалізувати таким чином:

```
        public XComponentLoader loadComponentLoader() throws Exception {
            if (instanceWithContext == null) {
                return (componentLoader =
UnoRuntime.queryInterface(XComponentLoader.class,
createInstanceWithContext(defaultLoadInstanceWithContext)));
            }
            return (componentLoader =
UnoRuntime.queryInterface(XComponentLoader.class, instanceWithContext));
        }
        public XComponent loadComponent(String filepath, TargetFrame
targetFrame, FrameSearchFlag frameSearchFlag, DocumentProperties
documentProperties) throws Exception {
            if (componentLoader == null) {
                loadComponentLoader();
            }
```

```
            component = componentLoader.loadComponentFromURL(filepath,
targetFrame.getTargetFrameName(), frameSearchFlag.getFrameSearchFlag(),
documentProperties.getPropertyValuesAsArray());

            textDocument = UnoRuntime.queryInterface(XTextDocument.class,
component);

            return component;

        }
```

Висновки до розділу 2.

1. Огляд інструментів розробки показав, що найбільш доцільним будуть Spring, Vue.js 3, LibreOffice SDK, які дозволяють максимально ефективно та швидко розробляти потрібну систему;

2. Позаяк система має в собі декілька модулів, найбільш ефективним рішенням було використання мікросервісної архітектури серверної частини. Для реалізації захисту було використано Spring Security, а у ролі токену JWT;

3. Для спрощення створення графічного інтерфейсу було використано SFC, які дозволили розділити компоненти за файлами та більш зручно організовувати створення інтерфейсу. Для реалізації захисту було використано State Management та його реалізацію Vuex. Для створення маршрутизації бібліотеку Vue Router;

4. Створення документів є складним процесом. Тому було вирішено використати XML-підход та LibreOffice SDK та об’єднати їх в одну бібліотеку. Перший підхід вирішує DXKS проблему, для того, щоб реалізувати більш просту та ефективну обробку документів. Другий же підхід дозволяє працювати із усіма можливими аспектами документів за допомогою управління документом за допомогою графічного інтерфейс засобами бібліотеки LibreOffice SDK.

РОЗДІЛ 3. ТЕСТУВАННЯ

3.1. Дослідження швидкодії створеної системи

Для дослідження швидкодії будуть використовуватися дві частини бібліотеки Lycorse DPL – XML-обробка коду та обробка документу із застосуванням LibreOffice SDK.

Таблиця 3.1.

Тестування швидкодії коду із використанням XML-обробки коду

Кількість ключових слів	Тип документу	Швидкість обробки
100	odt	0.031
100	docx	0.028
1000	odt	0.054
1000	docx	0.041
10000	odt	1.091
10000	docx	4.384

Таблиця 3.2.

Тестування швидкодії коду із використання LibreOffice SDK

Кількість ключових слів	Тип документу	Швидкість обробки
100	odt	1.86
100	docx	1.83
1000	odt	5.18
1000	docx	5.02
10000	odt	3 хвилини 21.52 секунд
10000	docx	3 хвилини 30.28 секунд

Розглядуючи результати XML-обробки коду можна бачити, що кількість ключових слів майже не впливає на швидкість обробки і залежить лише на кількість «розірваних» ключових слів. Так, вийшло що при обробці 10000 ключових слів документ типу odt був оброблений за 1 секунду, а документ типу docx було оброблено за 4 секунду. Відбулося це через те, що DXKS проблема виникала набагато частіше у документу типу docx, а ніж у документа типу odt.

Обробки із використанням LibreOffice SDK однозначно є значно повільнішою, але це компенсується функціоналом. Не менш цікава ситуація виникла при обробці 10000 ключових слів для обох видів документів. Як можна бачити, що 100 ключових слів займало 1.8 секунд, а обробки 1000 ключових слів займала 5 секунд в середньому. Але, обробки 10000 ключових слів зайняла 3 хвилини та 20-30 секунд. Якщо брати до уваги результати обробки 1000 ключових слів, то можна було б припустити, що обробки буде займати в 10 разів більше, тобто десь близько хвилини, але отриманий результат 3 хвилини. Проаналізувавши проблему можна припустити, що вона виникає тому що обробки документів, саме LibreOffice Writer потребує деякий час для того, щоб відобразити відповідні елементи, так як у випадку odt файл мав 218 сторінок, а docx файл мав 313 сторінок. Така кількість сторінок впливає на швидкість завантаження документу та його обробку.

3.2. Приклади застосування системи

Для того, щоб почати працювати із системою, потрібно зайти в систему, або зареєструватися. Для входу в систему потрібно заповнити свої дані – ім’я користувача та пароль (Рис. 3.1.).

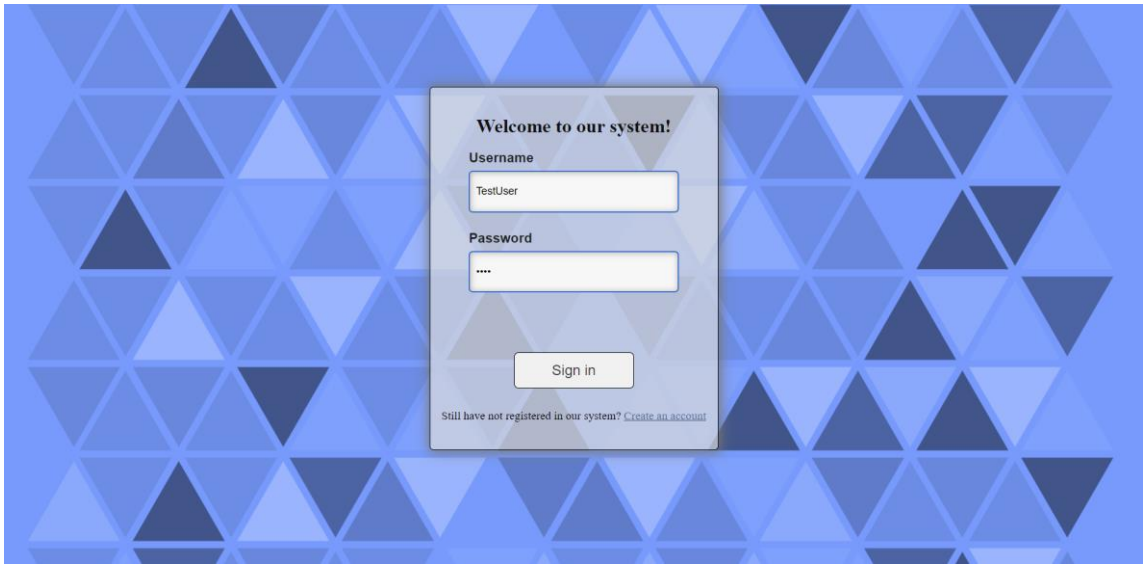

Рис. 3.1. Сторінка входу до системи

Для реєстрації можна перейти по посиланню «Create an account» внизу сторінки. Сторінка реєстрації показана на Рис. 3.2.

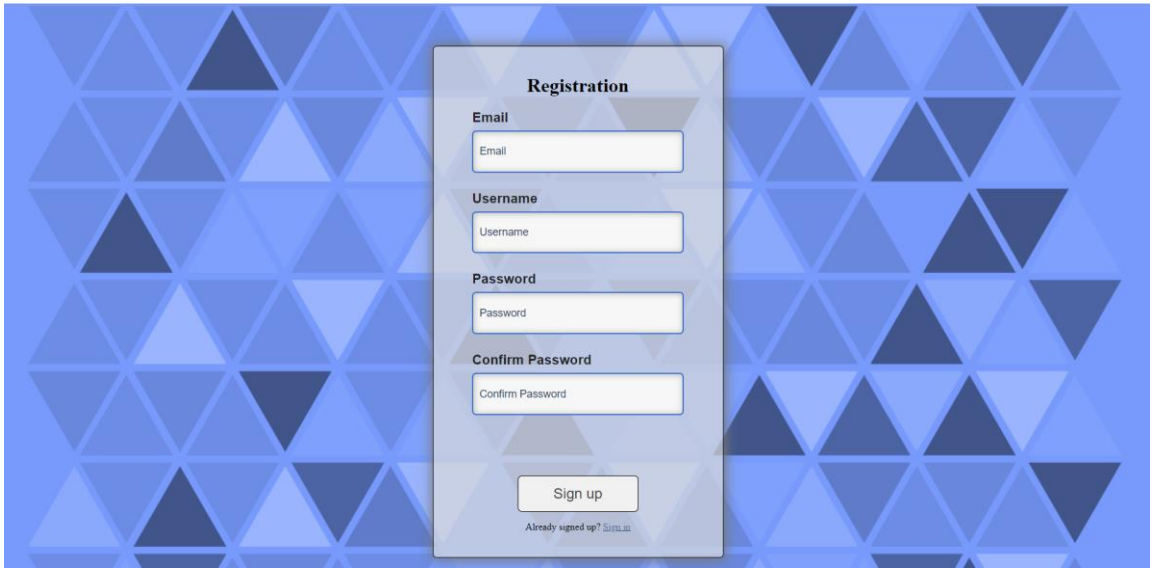

Рис. 3.2. Сторінка реєстрації

Для реєстрації потрібно вписати дійсну пошту, ім’я користувача, пароль та повторити пароль. Всі поля перевіряються на правильність та в разі помилки користувач отримає повідомлення про помилку.

Після входу до системи користувач попадає на сторінку управління документами (Рис. 3.3.).

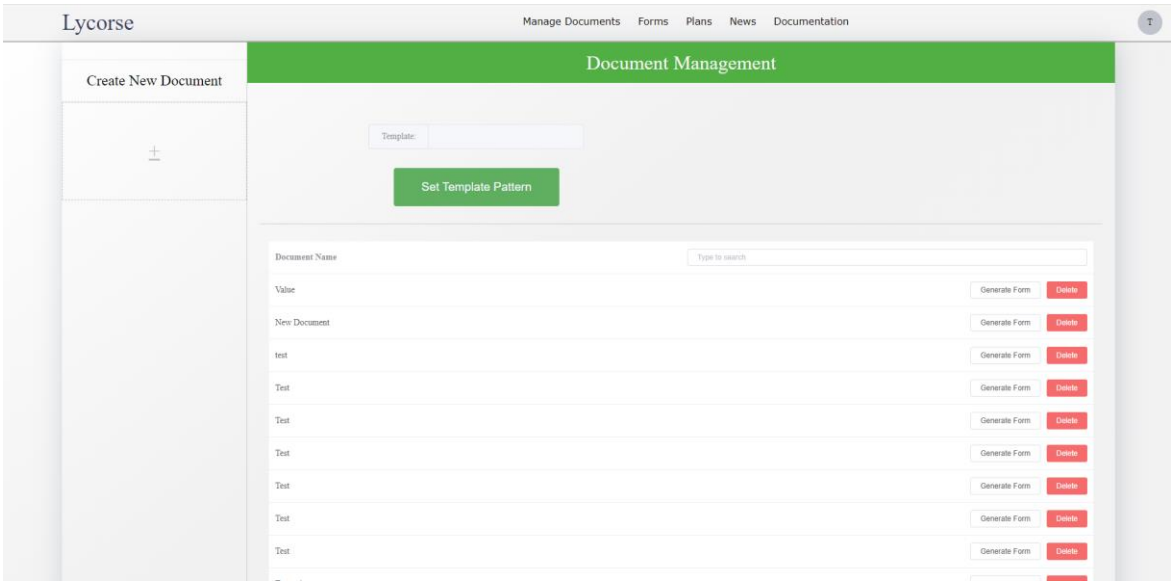

Рис. 3.3. Сторінка управління документів

Для додавання нового документу потрібно натиснути на поле, яке знаходиться збоку сторінки під назвою «Create New Document» (Рис. 3.4.).

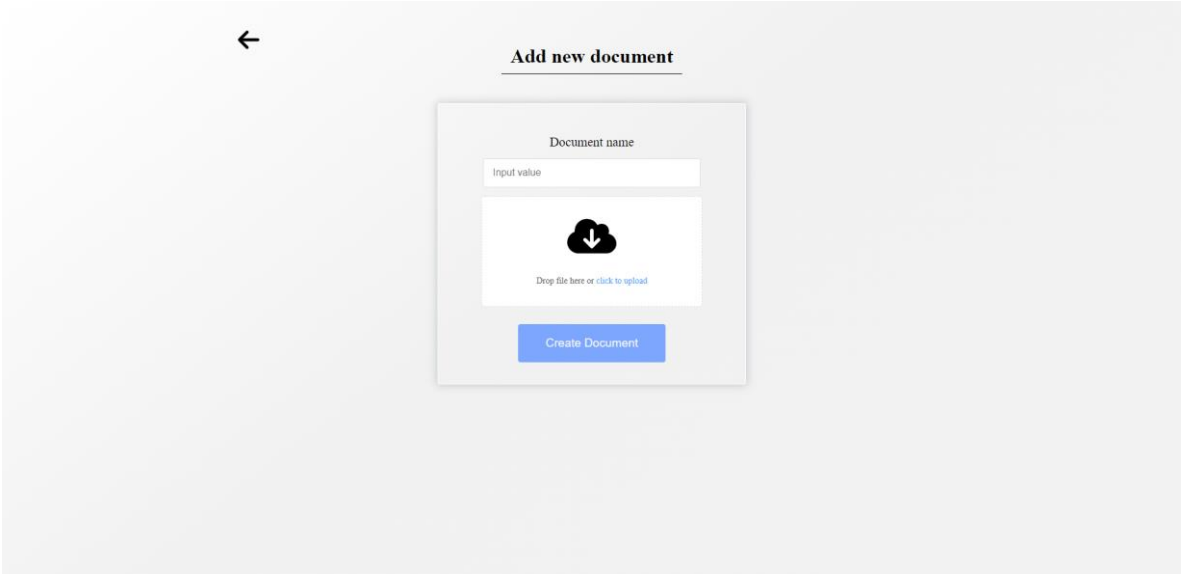

Рис. 3.4. Сторінка додавання нового документу

На цій сторінці вказується ім’я нового документа, а також завантажується документ або за допомогою системного діалогу, або за допомогою перенесення файлу прямо на компонент завантаження файлу. Після чого документ потрібно завантажити за допомогою натискання кнопки Create Document.

За допомогою відповідної кнопки Set Template Pattern можна зазначити стандартний для всіх документів шаблон для ключових слів, а тобто початок ключового слова та його кінець.

Таблиця документів поділена на сторінки по 10 документів на кожну сторінку (рис. 3.5.).

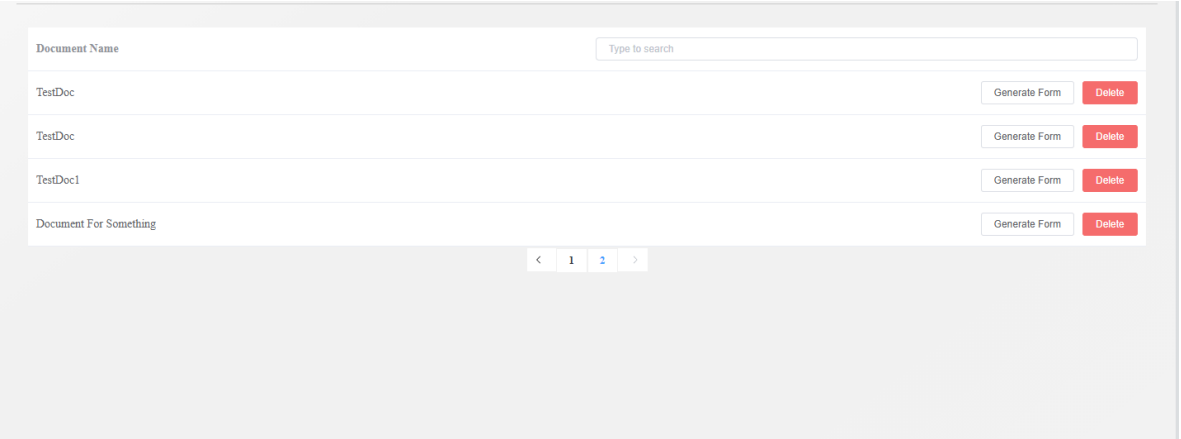

Рис. 3.5. Таблиця документів

Використовуючи кнопки можна згенерувати форму для документу (Generate Form) та видалити відповідний документ (Delete).

Кнопка генерації форми згенерує відповідну форму згідно ключових полів документа. Всі форми можна знайти на сторінці Forms (Рис. 3.6.).

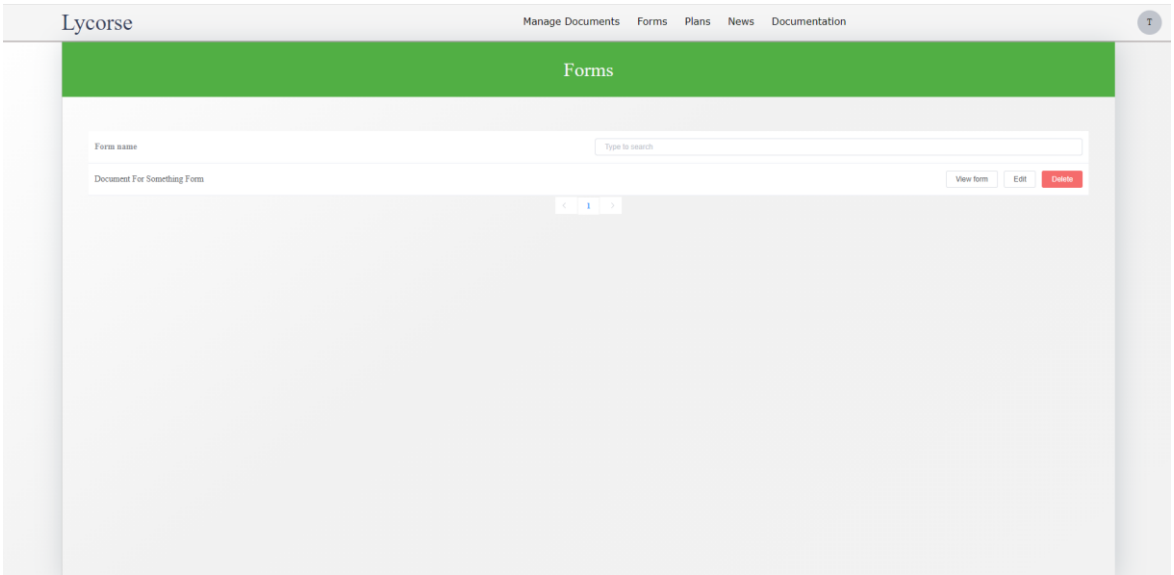

Рис. 3.6. Сторінка форм

Тут доступні 3 кнопки для перегляду форми (View Form), редагування полів (Edit) та видалення (Delete).

При натисканні на перегляд форми можна переглянути відповідну форму у вже згенерованому вигляді (Рис. 3.7.).

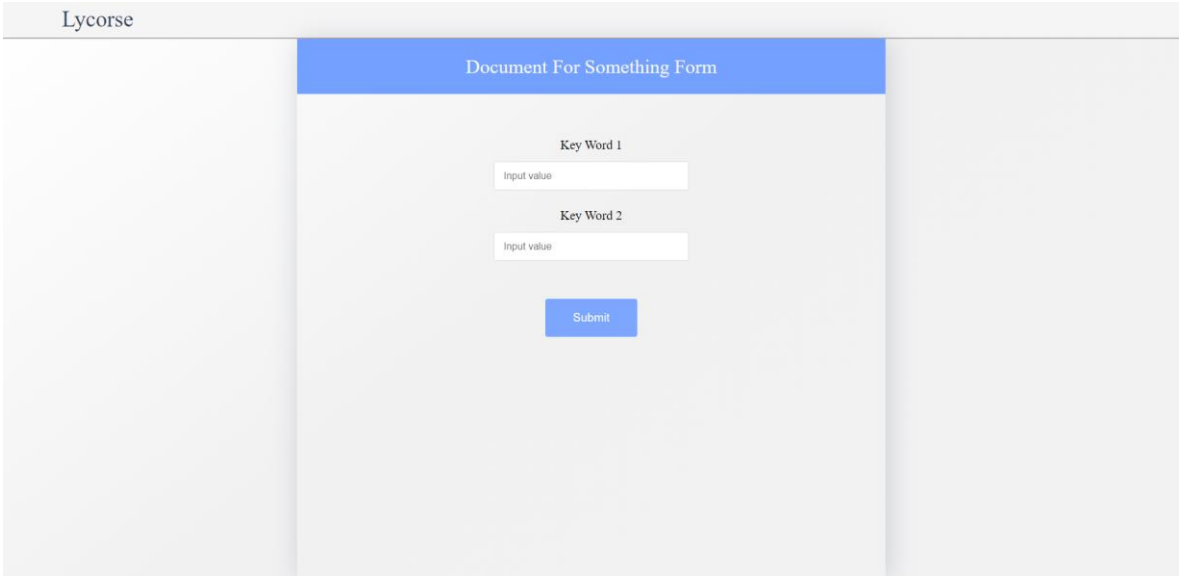

Рис. 3.7. Сторінка згенерованої форми

Заповнивши поля після натискання на кнопку «Submit» користувачу автоматично завантажується згенерований документ відповідно до шаблону. Приклад документу можна бачити на Рис. 3.8 та Рис. 3.9.

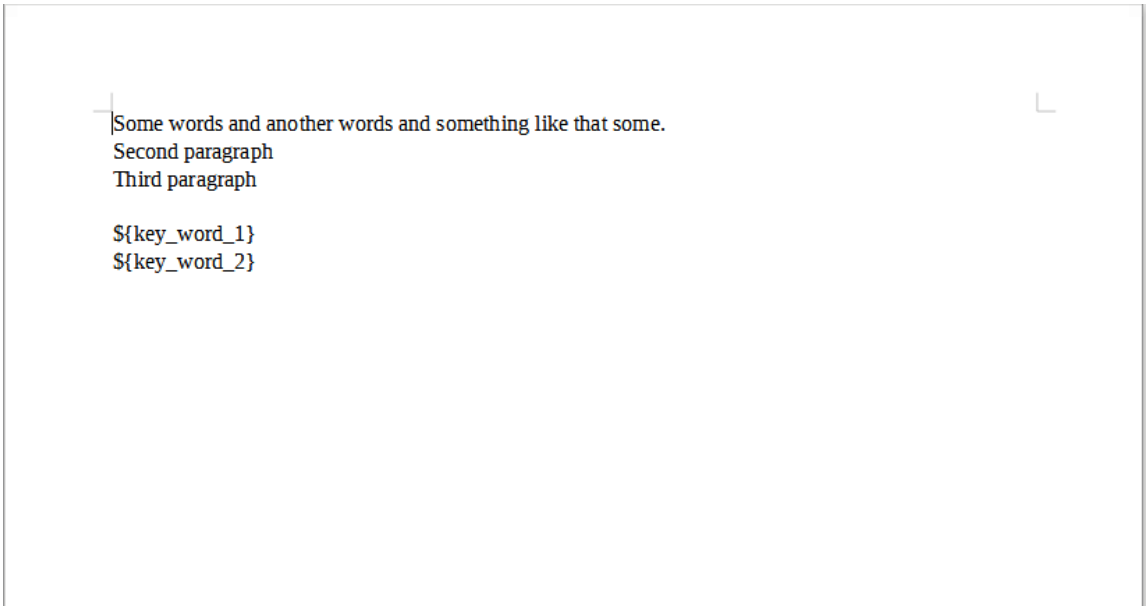

Рис. 3.8. Документ шаблон

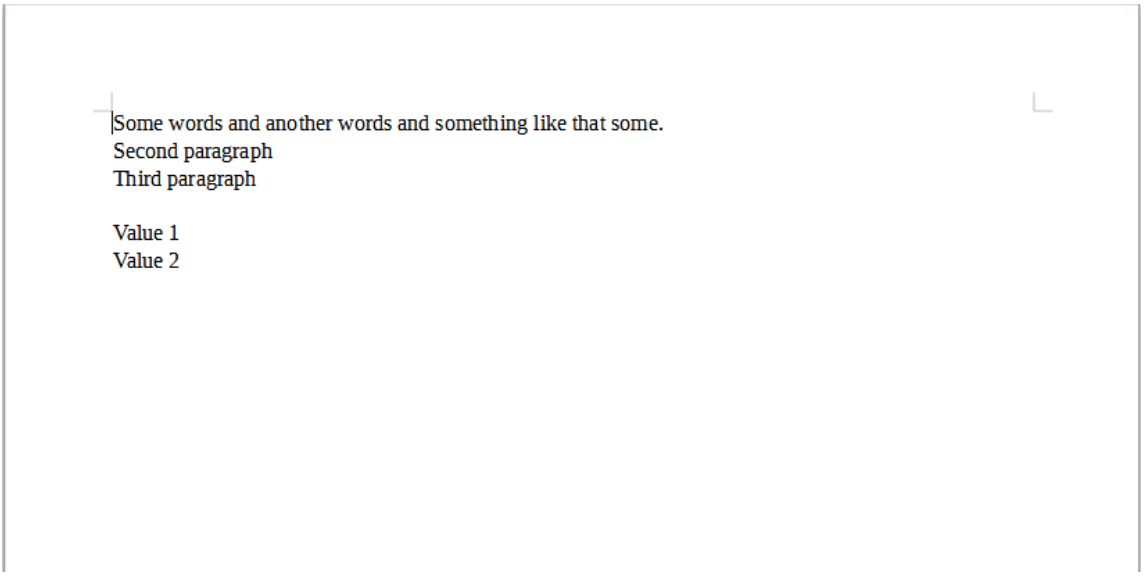

Рис. 3.9. Згенерований документ

3.3. Інструкція користувача

В даному підрозділі буде описано базові операції із використанням бібліотеки Lycorse DPL. Більш розширена документація буде розроблена на сторінці бібліотеки в Github[22].

Для початку роботи із бібліотекою можна використовувати дві різні частини.

Перша частина – XML-обробка документів.

Для виправлення ключових слів в документу та їх отримання можна використовувати такий код:

```
	DocumentPatternsAdjust documentPatternsAdjust =
DocumentPatternsAdjustController.initializePatternAdjuster(
                    ResourcesManager.getResourceFile("TestDocument.odt"),
                    new Pattern("${", "}"));
	documentPatternsAdjust.adjustPatterns();
```

DocumentPatternAdjust головний клас, який займається обробкою документівю. Як основа для нього вказується файл, який потрібно обробити та вид шаблону, тобто, як відрізнити початок та кінець ключового слова. Метод

adjustPatterns виправляє помилки в ключових словах та відповідно до типу файлу замінює файл змісту файлу.

Для створення своєї реалізації можна використовувати інтерфейс DocumentPatternAdjust (Рис. 3.10.)

```
public interface DocumentPatternsAdjust {
    String adjustPatterns() throws IOException;
    String adjustPatterns(File archive) throws IOException;
    String adjustPatterns(File archive, Pattern pattern) throws IOException;
    List<String> getFoundPatterns();
}
```

Рис. 3.10. Інтерфейс DocumentPatternAdjust

Для обробки самого змісту документу використовується інтерфейс XmlPatternAdjustProcessor (Рис. 3.11.)

```
public interface XmlPatternsAdjustProcessor {
    List<String> processXml(Pattern pattern, String xmlContent);
    List<String> processXml();
}
```

Рис. 3.11. Інтерфейс XmlPatternAdjustProcessor

Як базову реалізацію цього інтерфейсу можна використовувати абстрактний клас BaseXmlPatternAdjustProcessor (Рис. 3.12.).

```java
public abstract class BaseXmlPatternAdjustProcessor implements XmlPatternsAdjustProcessor {
    protected List<String> foundPatterns;

    protected String xmlContent;
    protected Pattern pattern;
    protected int offset = 0;
    protected int startOfPattern;
    protected int endOfPattern;
    protected int startOfPatternOffset;

    public BaseXmlPatternAdjustProcessor(String xmlContent, Pattern pattern) {
        this.xmlContent = xmlContent;
        this.pattern = pattern;
        foundPatterns = new ArrayList<>();
    }

    protected int findStartOfPattern() {
        startOfPattern = xmlContent.indexOf(pattern.getStartPattern(), offset);
        return startOfPattern;
    }

    protected int findEndOfPattern() {
        endOfPattern = xmlContent.indexOf(pattern.getEndPattern(), offset);
        return endOfPattern;
    }

    public String getXmlContent() { return xmlContent; }

    public void setXmlContent(String xmlContent) { this.xmlContent = xmlContent; }

    public Pattern getPattern() { return pattern; }

    public void setPattern(Pattern pattern) { this.pattern = pattern; }

    public List<String> getFoundPatterns() { return foundPatterns; }
}
```

Рис. 3.12. Абстрактний клас BaseXmlPatternAdjustProcessor

Сама ж реалізація класу для обробки змісту документу міститься у класі DocumentXmlPatternAdjustProcessor.

Для роботи із використанням LibreOffice SDK можна використовувати два різних підходи.

Перший підхід є спрощенням для обробки документів. Включає в себе лише базову обробку документу – заміна відповідних ключових слів.

```
File resourceFile = ResourcesManager.getResourceFile("TestDocument.odt");

DocumentManager documentManager =
DocumentManagerProvider.createDocumentManager(resourceFile);
```

```
Document document = documentManager.openDocument(resourceFile);

document.replace("${Search}", "Value");

document.saveDocument();
```

В даному прикладі за допомогою DocumentManagerProvider визначається тип документ, а за допомогою методу openDocument відкривається відповідний документ. Метод replace замінює відповідне ключове слово на значення. Метод saveDocument зберігає документ, також існує можливість збереження документа в іншому форматі із використанням методу saveDocumentAs та типу документу із перелічуваного типу даних enum DocumentConvertTypes.

Для створення своєї реалізації можна використовувати інтерфейс Document (Рис. 3.13.)

```
public interface Document {
    void saveDocument(File file);
    void saveDocument(String filepath);
    void saveDocument();
    void saveDocumentAs(File file, DocumentConvertTypes convertTo);
    void saveDocumentAs(String filepath, DocumentConvertTypes convertTo);
    void saveDocumentAs(DocumentConvertTypes convertTo);
    void replace(String search, String replace);
    void close();
}
```

Рис. 3.13. Інтерфейс Document

Для реалізації класу управління документами можна використати інтерфейс DocumentManager (рис 3.14.).

```
public interface DocumentManager {
    Document openDocument(File file);
}
```

Рис. 3.14. Інтерфейс DocumentManager

Всі внутрішні реалізації класу Document використовують клас LibreOfficeUnoManager. Саме цей клас реалізує другий підхід для роботи із

документами із використанням LibreOffice SDK. Клас є складнішим у використання, а ніж його проста реалізація.

Розглянемо декілька базових прикладів використанням. Перший приклад – знайти всі входження слова у документі.

```
File document = ResourcesManager.getResourceFile("TestTemplate.odt");
LibreOfficeUnoManager libreOfficeUnoManager = new LibreOfficeUnoManager();
libreOfficeUnoManager.openDocument(document);
System.out.println(libreOfficeUnoManager.findAll("${project}"));
```

Загалом, методи та їх сигнатури досить схожі. Для заміни значення використовується метод replaceAll.

```
libreOfficeUnoManager.replaceAll("${Search}", "Value");
```

Для роботи з текстом використовується спеціальний клас Text. Розглянемо приклад використання.

```
LibreOfficeUnoManager libreOfficeUnoManager = new LibreOfficeUnoManager();
libreOfficeUnoManager.openDocument(ResourcesManager.getResourceFile("TestDocument.odt"));
Text allDocumentText = libreOfficeUnoManager.findAllAsText("and").get(0);
allDocumentText.createCursor().gotoStartOfTheSentence(true);
allDocumentText.setCenteredAdjustment();
```

Метод findAllAsText повертає всі входження слова в документі у вигляди списку класів Text. Для того, щоб рухатися по тексту використовуються так звані курсори. Курсор – інструмент, який дозволяє імітувати реальний курсор в тексті. В даному випадку, використовується метод для створення нового курсору createCursor. Для того, щоб його перемістити до точки початку речення використовується метод gotoStartOfTheSentence із параметром true. У всіх методах курсору існує метод без параметру та метод із параметром типу Boolean. Цей параметр потрібен для того, щоб позначити, чи виділяти нам текст на даному етапі чи рухатися далі. Якщо параметр вказано як true, то фрагмент тексту

виділяється. Методи класу курсор реалізовані із використанням шаблону проектування Builder. Приклад можна бачити нижче.

```
allDocumentText.createCursor()
        .gotoStartOfTheSentence()
        .gotoNextSentence()
        .gotoNextWord()
        .gotoPreviousWord();
```

Також, є підтримка роботи із розташування параграфу. Наприклад, метод setCenteredAdjustment центрує виділений курсором текст. Всього існує 4 таких методи:

- setCenteredAdjustment;
- setLeftAdjustment;
- setRightAdjustment;
- setJustifiedAdjustment.

Для створення більш гнучких рішень можна використовувати універсальний метод setParagraphAdjustment із параметром типу ParagraphAdjust, в якому перелічено всі можливі варіанти розташувань.

Всі класи реалізовані на основі відповідних UNO компонентів, тому при бажанні їх можна отримати. Наприклад, клас Text містить в собі метод nativeTextRange, який повертає об’єкт типу XTextRange. В класі Cursor клас типу XTextCursor можна отримати за допомогою методу getTextCursor.

Також, можна використовувати більш складну процедуру ініціалізації документу. Для цього можна використовувати код, який наведено нижче.

```
libreOfficeUnoManager.initializeContext();
libreOfficeUnoManager.loadComponent(OdtFilePathHandler.normalizeFilepath(file), documentProperties);
```

Метод loadComponent повертає клас типу XComponent, який використовується для основних операцій над документом.

Ще один приклад із використанням методів LibreOfficeUnoManager та використанням UNO компонентів.

```
	XComponentLoader componentLoader =
libreOfficeUnoManager.loadComponentLoader();

	XComponent component = componentLoader.loadComponentFromURL(

		filepath,

		targetFrame.getTargetFrameName(),

		frameSearchFlag.getFrameSearchFlag(),

		documentProperties.getPropertyValuesAsArray());

	XTextDocument textDocument =
UnoRuntime.queryInterface(XTextDocument.class, component);
```

Таким способом можна завантажити та отримати оригінальний клас XTextDocument, та працювати з ним із використанням LibreOffice SDK.

Частина методів та сигнатур перелічено на Рис. 3.15.

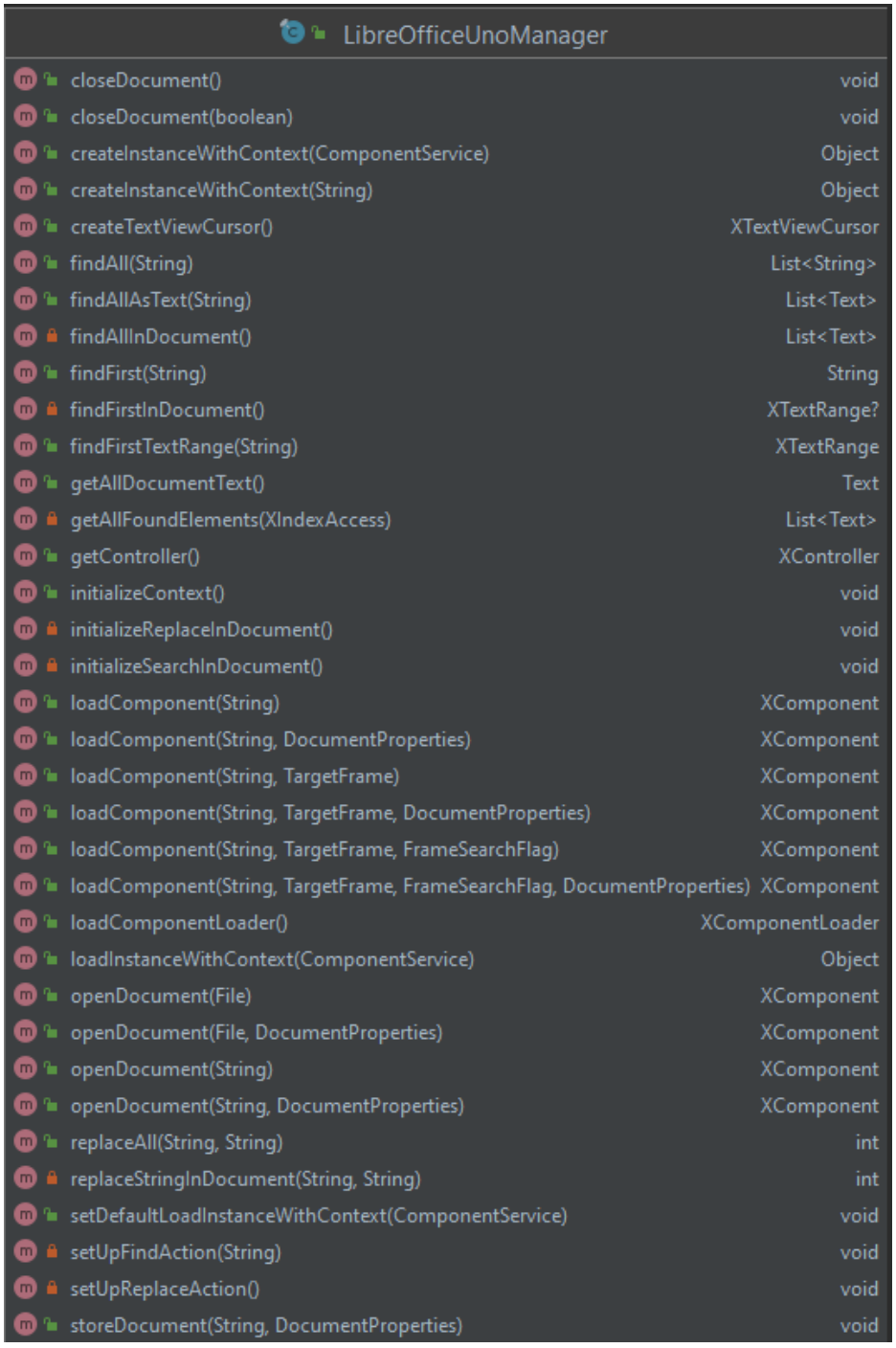

Рис. 3.15. Методи класу LibreOfficeUnoManager

ВИСНОВКИ

Аналіз сучасного стану автоматизації цифрового документообігу дає підстави стверджувати, що актуальність цього напрямку зростає, особливо в умовах збільшення частки дистанційної роботи внаслідок пандемії.

Порівняння наявних на ринку програмних інструментів показав, що сучасні системи генерації цифрової документації є досить розвиненими й функціональними, однак більшості з них притаманні певні недоліки: висока вартість, недостатня підтримка популярних форматів цифрових документів, закритість вихідного коду.

Урахування зазначених недоліків дозволило висунути функціональні вимоги до веб-фреймворку, серед яких зрозумілий інтерфейс, можливість працювати із форматами Microsoft Word та LibreOffice, можливість генерації форм відповідно до полів документів.

Беручи до уваги вимоги до програмного продукту, було обрано такі інструменти розробки:

- серверні інструменти розробки – Spring (Spring boot, Spring Security, Spring WebFlux, Spring JPA), jjwt (Java JWT: JSON Web Token for Java and Android), Connector/J (Mysql Java Connector).

- клієнтські інструменти розробки – Vue.js 3 (Vue Cli, Vue Router, Vuex, Vue i18n, Vue Class Component, Vue FontAwesome, SFC, Element Plus), Typescript, Javascript, Babel, Webpack.

Серверна (back-end) складова має мікросервісну архітектуру, реалізує основні можливості аутентифікації за допомогою JWT. Реалізує основний функціонал для повноцінної обробки документів.

Клієнтська (front-end) складова дає можливість легкої обробки документів за допомогою зручного та обширного графічний інтерфейс; дає

можливість завантажувати та обробляти документа, а також генерувати відповідні форми для генерації документів.

База даних створена за допомогою MySQL. Має зрозумілу та розподілену структуру. Збереження файлів реалізовано шляхом збереження файлів у файловій системі та заповнення відповідного поля в базі даних.

Було реалізовано опрацювання шаблонів документів, що лежить в основі фреймворку, який є основним результатом роботи. Серед особливостей слід відзначити універсальність рішення (можливість роботи з різними форматами документів, підтримка різних способів запису ключових слів тощо), доступність (система відкрита, тому кожен може на її основі реалізувати потрібні складові документообігу), простота використання (реалізовано єдину надбудову над двома різними API).

Аналіз швидкодії фреймворку показав, що використання обробника XML-коду є ефективним та швидким для обробки. Використання ж бібліотеки Lycorse DPL рекомендується використовувати менше ніж десять тисяч ключових слів.

Для демонстрації можливостей фреймворку було створено систему генерації додатків до дипломів здобувачів освіти, яка нині використовується в Криворізькому державному педагогічному університеті.

Завдяки розробленій документації користувача проєкт може в майбутньому використовуватися для розробки інших рішень у галузі автоматизації документообігу.

Список використаних джерел

1. .NET Word API – Processing Word in C#, VB.NET, ASP.NET. – Access mode: https://www.e-iceblue.com/Introduce/word-for-net-introduce.html#.YaT-xeomyUk (дата звернення: 01.12.2021)
2. “Document.” Merriam-Webster.com Dictionary, Merriam-Webster. – Access mode: https://www.merriam-webster.com/dictionary/document (дата звернення: 28.11.2021)
3. 10 Best Java Frameworks to Use in 2021. – Access mode: https://hackr.io/blog/java-frameworks (дата звернення: 29.1\1.2021)
4. 10 Best JavaScript Frameworks to Use in 2021. 09 Nov, 2021, – Access mode: https://hackr.io/blog/best-javascript-frameworks (дата звернення: 29.11.2021)
5. a .NET library that can read/write Office formats without Microsoft Office installed. No COM+, no interop. Access mode: https://github.com/nissl-lab/npoi (дата звернення: 01.12.2021)
6. A Vue 3 UI Framework | Element Plus. – Access mode: https://element-plus.org/en-US/ (дата звернення: 29.12.2021)
7. Apache POI - the Java API for Microsoft Documents. Access mode: https://poi.apache.org/ (дата звернення: 29.11.2021)
8. Babel The compiler for next generation JavaScript. – Access mode: https://babeljs.io/ (дата звернення: 02.12.2021)
9. ContactBook – Better contacts. – Access mode: https://contractbook.com/ (дата звернення: 01.10.2021)
10. Docassemble. – Access mode: https://docassemble.org/ (дата звернення: 01.10.2021)
11. Docupilot | Document Automation Software, Document Generation Software. – Access mode: https://docupilot.app/ (дата звернення: 01.10.2021)
12. Enterprise Automation Software | DocuPhase. – Access mode: https://www.docuphase.com/ (дата звернення: 01.10.2021)

13. File Format APIs for Word Excel PDF Email PowerPoint Barcode Images OCR Note & 3D. Access mode: https://www.aspose.com/ (дата звернення: 01.10.2021)
14. Framework/Article/Filter/FilterList OOo 3 0. Apache OpenOffice Wiki. – Access mode: https://wiki.openoffice.org/wiki/Framework/Article/Filter/FilterList_OOo_3_0 (дата звернення: 29.09.2021)
15. Get beyond OCR with automatic data extraction. – Access mode: https://hypatos.ai/en (дата звернення: 01.10.2021)
16. Handling Documents – Apache OpenOffice Wiki. – Access mode: https://wiki.openoffice.org/wiki/Documentation/DevGuide/OfficeDev/Handling_Documents (дата звернення: 03.12.2021)
17. Introducing the New Java Word Processing Library. Access mode: https://www.syncfusion.com/blogs/post/introducing-the-new-java-word-processing-library.aspx (дата звернення: 04.12.2021)
18. John Hawksworth, Richard Berriman, Saloni Goel, Will robots really steal our jobs? – Access mode: https://www.pwc.com/hu/hu/kiadvanyok/assets/pdf/impact_of_automation_on_jobs.pdf (дата звернення: 25.11.2021)
19. Juro | The all-in-one contract automation platform. – Access mode: https://juro.com/ (дата звернення: 30.1\1.2021)
20. Kenneth Frank Guide to Workflow Automation [Electronic resource]. – Oct 27, 2020. – Access mode: https://www.jotform.com/workflow-automation-guide/ (дата звернення: 01.10.2021)
21. LibreOffice 7.2 SDK – Overview. Access mode: https://api.libreoffice.org/ (дата звернення: 03.12.2021)
22. Lycorse Document Processing Library. – Access mode: https://github.com/CodePsi/Lycorse-DPL (дата звернення: 08.12.2021)
23. M2Doc. – Access mode: https://www.m2doc.org/ (дата звернення: 01.12.2021)

24. McKinsey & Company, Jobs lost, jobs gained: Workforce transitions in a time of automation. – December 2017. – Access mode: https://www.mckinsey.com/~/media/BAB489A30B724BECB5DEDC41E9BB9FAC.ashx (дата звернення: 29.09.2021)
25. ODT Toolkit. Access mode: https://tdf.github.io/odftoolkit/odfdom/index.html (дата звернення: 04.12.2021) (дата звернення: 25.11.2021)
26. OnTask | Workflow Automation, Digital Forms & Documents. – Access mode: https://www.ontask.io/ (дата звернення: 29.11.2021)
27. PandaDoc – Create, Approve, Track & eSign Docs 40% Faster. – Access mode: https://www.pandadoc.com/ (дата звернення: 29.11.2021)
28. PascalCase Definition. – Access mode: https://techterms.com/definition/pascalcase (дата звернення: 04.12.2021)
29. Process Automation: Advantages and Disadvantages for IT Companies. – November 28, 2017. – Access mode: https://www.gb-advisors.com/process-automation/ (дата звернення: 29.11.2021)
30. Service MediaDescriptor. – Access mode: https://www.openoffice.org/api/docs/common/ref/com/sun/star/document/MediaDescriptor.html (дата звернення: 13.11.2021)
31. Single File Component | Vue.js. – Access mode: https://v3.vuejs.org/guide/single-file-component.html (дата звернення: 29.11.2021)
32. Spring Data JPA. – Access mode: https://spring.io/projects/spring-data-jpa (дата звернення: 20.11.2021)
33. Stack Overflow Developer Survey 2021. – Access mode: https://insights.stackoverflow.com/survey/2021#most-popular-technologies-webframe (дата звернення: 03.12.2021)
34. Stack Overflow Trends. – Access mode: https://insights.stackoverflow.com/trends?tags=reactjs%2Cvue.js%2Cangularjs%2Cangular (дата звернення: 03.12.2021)

35. State Management | Vue.js. – Access mode: https://v3.vuejs.org/guide/state-management.html
36. Steve Wilson, How Document Automation is Changing the Healthcare Industry. – Nov 27, 2017. – Access mode: https://electronichealthreporter.com/document-automation-changing-healthcare-industry/ (дата звернення: 24.07.2021)
37. The evolution of process automation [Electronic resource]. – IBM Institute for Business Value. – 2018. – Access mode: https://www.ibm.com/downloads/cas/QAQMRGVN (дата звернення: 25.07.2021)
38. The Ultimate Guide to Document Automation in 2021. – Access mode: https://research.aimultiple.com/document-automation/ (дата звернення: 25.07.2021)
39. Understanding the Value Proposition of Document Assembly. – Access mode: https://www.accusoft.com/resources/blog/understanding-the-value-proposition-of-document-assembly/ (дата звернення: 26.07.2021)
40. Vue Router. – Access mode: https://next.router.vuejs.org/ (дата звернення: 29.11.2021)
41. Vue.js | Font Awesome. – Access mode: https://fontawesome.com/v5.15/how-to-use/on-the-web/using-with/vuejs (дата звернення: 29.11.2021)
42. Vue.js. – Access mode: https://v3.vuejs.org/ (дата звернення: 29.11.2021)
43. Webpack. – Access mode: https://webpack.js.org/ (дата звернення: 29.11.2021)
44. What is Document Management (DMS)? – Access mode: https://www.aiim.org/what-is-document-imaging (дата звернення: 26.11.2021)
45. What is kebab case? – Access mode: https://www.theserverside.com/definition/Kebab-case (дата звернення: 04.12.2021)

46. What is Vuex? | Vuex. – Access mode: https://vuex.vuejs.org/ (дата звернення: 01.12.2021)
47. What to Consider Before Using JWT. 14.10.2020, – Access mode: https://serengetitech.com/tech/what-to-consider-before-using-jwt/ (дата звернення: 11.11.2021)
48. Wikipedia contributors. PricewaterhouseCoopers. – Wikipedia, The Free Encyclopedia, – 2021 Nov 5, 16:07 UTC. – Access mode: https://en.wikipedia.org/w/index.php?title=PricewaterhouseCoopers&oldid=1053716560 (дата звернення: 05.11.2021)
49. XDocReport means XML Document reporting. It's Java API to merge XML document created with MS Office (docx) or OpenOffice (odt), LibreOffice (odt) with a Java model to generate report and convert it if you need to another format (PDF, XHTML...). Access mode: https://github.com/opensagres/xdocreport (дата звернення: 05.11.2021)