Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

Навчально-науковий інститут комп'ютерних наук та інформаційних технологій
(повне найменування інституту, назва факультету (відділення))

Кафедра інформаційних систем та комп'ютерного моделювання
(повна назва кафедри (предметної, циклової комісії))

Пояснювальна записка

до дипломної роботи

перший (бакалаврський)
(рівень вищої освіти)

на тему: Створення клієнт-серверної системи автоматизованого формування документів

Виконав: студент 2 курсу групи ІСТс-21
спеціальності
126 "Інформаційні системи та технології"
(шифр і назва напряму підготовки, спеціальності)

Корнута Р.В.
(прізвище та ініціали)

Керівники Яркун В.І., Флуд Л.О.
(прізвище та ініціали)

Рецензент Мокрицька О.В.
(прізвище та ініціали)

Львів – 2024

Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

ННІ комп'ютерних наук та інформаційних технологій
Кафедра інформаційних систем та комп'ютерного моделювання
Рівень вищої освіти перший (бакалаврський)
Спеціальність 126 "Інформаційні системи та технології"
(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри ІСКМ
_____ Сторожук О.Л.
"7" 02 2024року

З А В Д А Н Н Я
НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Корнута Ростислав Васильович
(прізвище, ім'я, по батькові)

1. Тема роботи *Створення клієнт-серверної системи автоматизованого формування документів*
керівники роботи *Яркун В.І. ст..викл. каф.ІПЗ, Флуд Л.О. к.т.н. доц. каф. ІСКМ*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від 06 лютого 2024 р. № С-87
2. Термін подання студентом роботи 10.06.2024 р.
3. Вихідні дані до роботи *Аналіз шляхів вирішення поставлених задач, оцінка переваг і недоліків різних засобів для реалізації поставленого завдання та вибору найкращих з них. Аналіз сайтів аналогів. Вхідні дані для проведення оформлення записки. Література за тематикою роботи.*
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)
Вступ.
Розділ 1. Стан проблемної області.
Розділ 2. Інформаційне забезпечення.
Розділ 3. Програмне та технічне забезпечення.
Висновок. Список використаної літератури. Додатки
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Слайди для доповіді (підготовка матеріалу для доповіді загальним обсягом 10-12 слайдів)
6. Дата видачі завдання *07 лютого 2024 року*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1.	Огляд та системний аналіз проблемної області. Огляд літературних джерел на досліджувану тему. Постановка задачі проекту та формування функціональних вимог. Збір потрібних матеріалів.	*07.02.2024 р. 23.02.2024 р.*	*Виконано*
2.	Огляд сучасного стану проблемної області. Написання та оформлення першого розділу пояснювальної записки.	*06.02.2024 р. 01.03.2024 р.*	*Виконано*
3.	Написання та оформлення другого розділу. Аналіз інформаційного та математичного забезпечення.	*02.03.2024 р. 19.03.2024 р.*	*Виконано*
4.	Написання та оформлення третього розділу пояснювальної записки. Підготовка програмного забезпечення.	*20.03.2024 р. 20.05.2024 р.*	*Виконано*
5.	Оформлення пояснювальної записки та здача на рецензування.	*24.05.2024 р. 08.06.2024р.*	*Виконано*

Студент _____ *Корнута Р.В.*
(підпис) (прізвище та ініціали)

Керівники роботи _____ *Яркун В.І.*
(підпис) (прізвище та ініціали)

_____ *Флуд Л.О.*
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Дипломна робота містить 37 сторінок пояснювальної записки, 17 рисунків та 23 джерела.

У дипломній роботі розроблений програмний продукт, що представляє собою вебзастосунок і дозволяє користувачам більш зручніше та ефективніше працювати з документами, створювати та редагувати їх.

Програмний продукт реалізовано в середовищі програмування Intellij IDEA за допомогою мови програмування TypeScript та фреймворку Angular.js. База даних була створена у об’єктно-реляційній системі керування базами даних MongoDB.

Ключові слова: TypeScript, база даних, Angular.js, MongoDB, Kotlin.

ABSTRACT

The thesis contains 37 pages of explanatory note, 17 figures and 23 used literature sources.

The thesis developed a software product that is a web application and allows users to more conveniently and efficiently work with documents, create and edit them.

The software product is implemented in the Intellij IDEA programming environment using the TypeScript programming language and the Angular.js framework. The database was created in the object-relational database management system MongoDB.

Keywords: TypeScript, database, Angular.js, MongoDB, Kotlin.

Технічне завдання

Розробити клієнт-серверну систему автоматизованого формування документів, що надає можливість підвищити ефективність створення та редагування документів. Розроблена система повинна бути представленна як вебзастосунок.

Задачами, які мають виконуватись даним продуктом є:

- база даних, що задовольняє тему дипломної роботи;
- розробка вебзастосунку.

Система має забезпечувати наступні можливості:

- шукати та переглядати різні варіанти документів;
- строювати та редагувати документи;
- створювати та редагувати папки для зберігання документів;
- вхід в систему.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

ПК – персональний комп’ютер;

ПЗ – програмне забезпечення;

ІС – інформаційна система;

ОС – операційна система;

ПЗ – програмне забезпечення;

Фреймворк – інфраструктура програмних рішень;

API – набір визначень взаємодії різнотипного ПЗ ;

Interface Builder – середовище для розробки інтерфейса корситувача;

IDE – комп'ютерна програмне середовище;

UI – інтерфейс користувача;

ВСТУП

У сучасному світі цифровізації та автоматизації, ефективне управління документами є ключовим елементом успішної діяльності багатьох організацій та приватних осіб. Тема дипломної роботи "Створення клієнт-серверної системи автоматизованого формування документів" є надзвичайно актуальною, оскільки вона спрямована на вирішення важливих задач, пов'язаних зі створенням, зберіганням та редагуванням документів.

Розроблений вебсайт надає можливість користувачам створювати документи за заздалегідь підготовленими шаблонами. Це значно спрощує процес формування документів, дозволяючи зекономити час та мінімізувати ризик помилок, пов'язаних з ручним введенням інформації. Зручний інтерфейс сайту забезпечує швидкий та інтуїтивно зрозумілий доступ до всіх функцій, що робить його корисним для широкого кола користувачів.

Сайт буде особливо корисним для підприємств, що часто працюють з великою кількістю стандартних документів, таких як договори, звіти, заявки тощо. Він дозволяє централізовано зберігати всі документи у структурованих папках, що полегшує їх пошук та управління. Функція редагування документів безпосередньо на сайті забезпечує можливість швидких коригувань та оновлень, що особливо важливо у динамічному бізнес-середовищі.

Перспективність даного проекту полягає у його універсальності та масштабованості. Сайт можна легко адаптувати під потреби різних галузей та типів організацій, додавати нові шаблони та функціональні можливості відповідно до вимог користувачів. Крім того, інтеграція з іншими сервісами та системами (наприклад, системами управління завданнями чи обліку) відкриває нові горизонти для подальшого розвитку та вдосконалення.

Отже, розробка такого сайту сприятиме підвищенню ефективності та якості роботи з документами, забезпечуючи сучасні та зручні інструменти для їх створення, зберігання та редагування. Це важливий крок на шляху до цифрової трансформації, що дозволить зробити роботу з документами більш організованою та продуктивною.

Актуальність нашого дослідження являється ефективне управління документацією є важливою складовою успішного функціонування будь-якої організації. З розвитком цифрових технологій зростає потреба у створенні зручних та функціональних інструментів для роботи з документами. Сайти для створення, редагування та зберігання документів дозволяють значно скоротити час на виконання рутинних завдань, підвищуючи продуктивність працівників та забезпечуючи надійність збереження інформації. Враховуючи ці потреби, розробка вебсайту, що надає можливість роботи з документами за допомогою готових шаблонів, є надзвичайно актуальною.

Об’єкт дослідження – сайт для роботи з документами.

Предметом дослідження – є процеси створення, редагування та зберігання документів за допомогою вебсайту, що використовує готові шаблони для спрощення роботи користувачів.

Мета дипломної роботи. Метою дипломної роботи є створення сайту для роботи з документами, що надає можливість покращити ефективність створення та редагування документів за вже готовими шаблонами.

Практична значимість дипломної роботи полягає в розробці ефективного інструменту для роботи з документами, який дозволить користувачам швидко та легко створювати, редагувати та зберігати документи на основі готових шаблонів. Це сприятиме підвищенню продуктивності праці в організаціях, зменшенню витрат часу на рутинні завдання та забезпеченню надійного збереження документів. Використання даного вебсайту може бути корисним як для малих, так і для великих підприємств, які прагнуть оптимізувати свої процеси управління документацією.

РОЗДІЛ 1. СТАН ПРОБЛЕМНОЇ ОБЛАСТІ

1.1 Огляд аналогічних продуктів.

1.1.1 Застосунок Docusign

DocuSign – це провідна платформа для електронного підпису документів та управління цифровими транзакціями. Вона дозволяє користувачам підписувати, відправляти та керувати документами в електронному форматі, що спрощує та прискорює процес підписання і обміну документами (рисунок 1.1).

Основні можливості DocuSign:

1. **Електронний підпис**: Дозволяє користувачам підписувати документи онлайн з використанням електронного підпису, який має юридичну силу в багатьох країнах світу.
2. **Відправка документів**: Можливість відправляти документи на підпис іншим користувачам через електронну пошту або інші цифрові канали.
3. **Шаблони документів**: Створення та використання шаблонів для часто використовуваних документів, що економить час та зусилля.
4. **Моніторинг та звітність**: Відстеження статусу документів у реальному часі, включаючи інформацію про те, хто підписав, хто переглядав і хто затримує процес підписання.
5. **Інтеграції**: Інтеграція з різноманітними бізнес-додатками, такими як CRM-системи (наприклад, Salesforce), системи управління документами (наприклад, Google Drive) та інші.
6. **Безпека**: Високий рівень безпеки та відповідність міжнародним стандартам та регуляціям, таким як GDPR, eIDAS, та інших.

Переваги використання DocuSign:

- **Швидкість та ефективність**: Значно скорочується час на підписання та обмін документами.
- **Зручність**: Можливість підписання документів з будь-якого місця та на будь-якому пристрої.
- **Зниження витрат**: Зменшуються витрати на друк, доставку та зберігання паперових документів.

- **Юридична сила**: Електронні підписи в DocuSign відповідають вимогам багатьох законів і нормативних актів щодо електронного підпису.

DocuSign є корисним інструментом для підприємств будь-якого розміру, які прагнуть оптимізувати процеси підписання документів та перейти до безпаперових технологій.

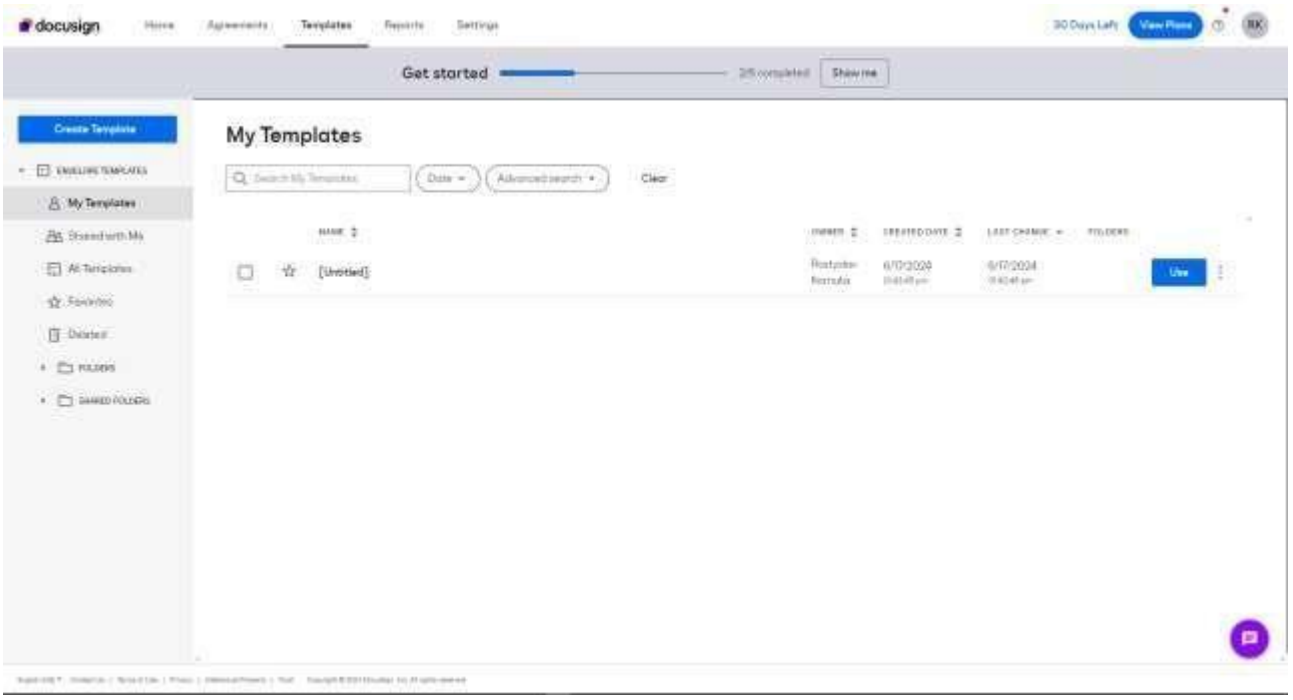

Рисунок 1.1 — Інтерфейс Docusign

1.1.2 Застосунок Document.online

Document.Online - це онлайн-сервіс для роботи з електронними документами, який пропонує широкий спектр функцій для співпраці, редагування та підписання документів. Він позиціонує себе як платформа для сучасного документообігу, що полегшує роботу з документами в будь-який час і з будь-якого місця (рисунок 1.2).

Основні можливості Document.Online:

- **Спільна робота:** Спільна робота над документами з колегами та партнерами в рамках одного сервісу.
- **Доступність:** Цілодобовий доступ до документів з будь-якого місця світу за допомогою смартфона, планшета або ноутбука.

- **Електронний підпис:** Забезпечення юридичної дійсності документів за допомогою електронного підпису протягом 10 секунд.
- **Підтримка платформ:** Доступність на будь-якій платформі та пристрої.
- **Мобільність:** Можливість роботи з документами на мобільних пристроях, планшетах та ноутбуках з будь-якою операційною системою.
- **Глобальний доступ:** Ведення бізнесу 24/7 практично з будь-якої точки світу.

Document.Online пропонує безкоштовний план з обмеженими можливостями та платні плани з розширеними функціями. Сервіс підходить для використання як приватними особами, так і small and medium-sized businesses (SMB).

Переваги Document.Online:

- **Зручність використання:** Інтуїтивно зрозумілий інтерфейс та простий у використанні функціонал.
- **Доступність:** Можливість використання з будь-якого пристрою та платформи.
- **Безпека:** Захист документів за допомогою шифрування та інших заходів безпеки.
- **Функціональність:** Широкий спектр функцій для роботи з документами.
- **Економія часу:** Зниження витрат часу на обробку документів.

Недоліки Document.Online:

- **Безкоштовний план має обмеження:** Обмежена кількість місця для зберігання документів та деякі функції недоступні.
- **Інтеграція з іншими сервісами:** Обмежена інтеграція з іншими онлайн-сервісами та програмами.

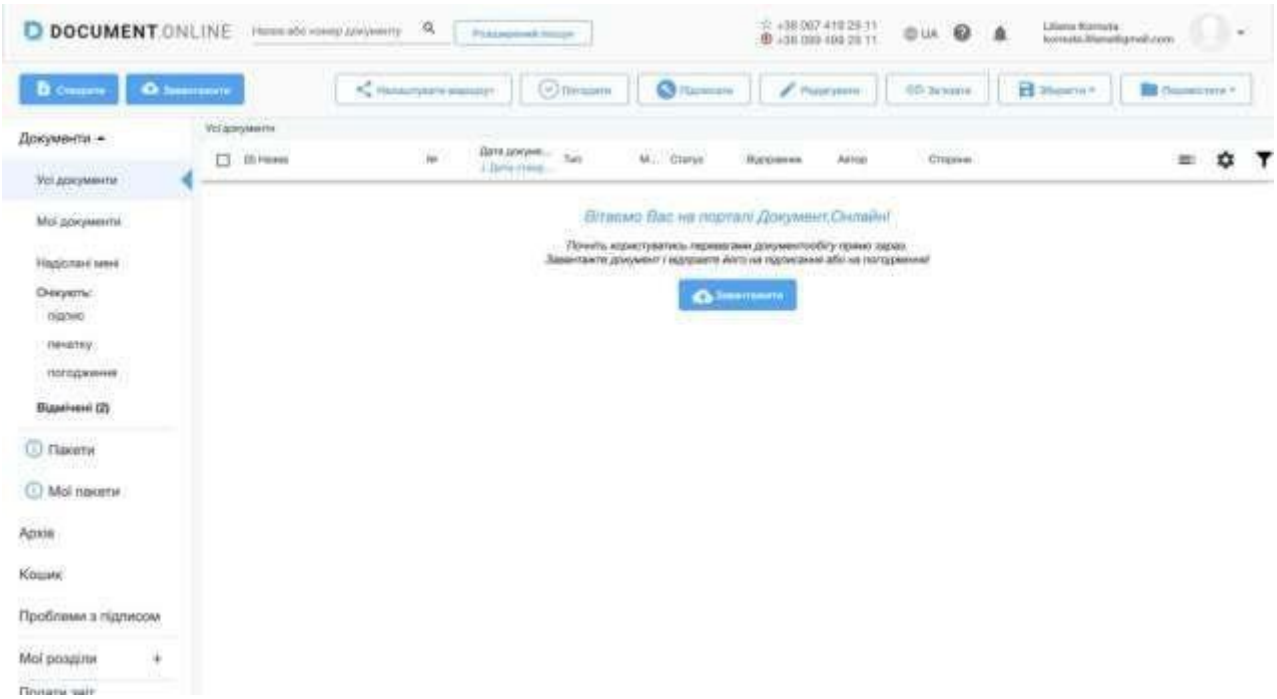

Рисунок 1.2 — Інтерфейс Document.online

1.2 Про системи автоматизованого формування документів

Системи автоматизованого формування документів (САФД) є програмними рішеннями, які значно спрощують процес створення, редагування та управління документами. Такі системи дозволяють автоматично генерувати документи на основі заздалегідь підготовлених шаблонів, що знижує кількість ручної роботи та мінімізує ризик помилок.

Основним компонентом САФД є шаблони документів, які можуть бути налаштовані під конкретні потреби організації. Ці шаблони зазвичай містять фіксовані текстові фрагменти та змінні поля, які заповнюються автоматично на основі введених даних або інтеграції з іншими системами (наприклад, базами даних чи CRM-системами).

Серед основних функцій САФД варто виділити: автоматичне заповнення документів, генерацію документів у різних форматах (наприклад, PDF, DOCX), зберігання та організацію документів у спеціальних папках, можливість колективної роботи над документами та їх редагування в режимі реального часу.

Впровадження САФД в організацію приносить ряд переваг, таких як зниження витрат часу на створення документів, підвищення точності та якості

документів, покращення контролю версій та забезпечення збереження інформації. Крім того, автоматизація процесів дозволяє співробітникам зосередитися на більш важливих завданнях, що сприяє підвищенню загальної ефективності роботи.

Отже, системи автоматизованого формування документів є невід'ємною частиною сучасного бізнесу, які забезпечують високий рівень автоматизації та оптимізації процесів документообігу.

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Пакетний менеджер NPM

Пакетний менеджер npm — менеджер пакетів для Node.js з сотнями тисяч пакетів [1]. Незважаючи на те, що він створює деякі з ваших структур (організацій), це не є головною метою. Головною метою, як ви торкнулися, є автоматизоване управління залежностями та пакетами. Це означає, що ви можете вказати всі залежності вашого проекту у файлі package.json, тоді будь-який час, коли ви (або хтось інший) має почати роботу з проектом, вони можуть просто запустити npm install і негайно мати всі встановлені залежності. Крім того, можна також вказати, на які версії залежить ваш проект, щоб запобігти розриву вашого проекту.

Це, безумовно, можна вручну завантажити бібліотеки, скопіювати їх у правильні каталоги, і використовувати їх таким чином. Проте, як ваш проект (і список залежностей) зростає, це швидко стає трудомістким і брудним. Це також ускладнює співпрацю та обмін вашим проектом.

Сподіваюся, це зробить більш зрозумілим, що є метою npm. Як розробник Javascript (як на стороні клієнта, так і на стороні сервера), npm є незамінним інструментом у моєму робочому процесі.

2.2 Середовище розробки Intellij IDEA

IntelliJ IDEA – це інтегроване середовище розробки (IDE), що пропонує широкий спектр функцій для розробників програмного забезпечення [2]. На рисунку 2.1 можна побачити інтерфейс Intellij IDEA.

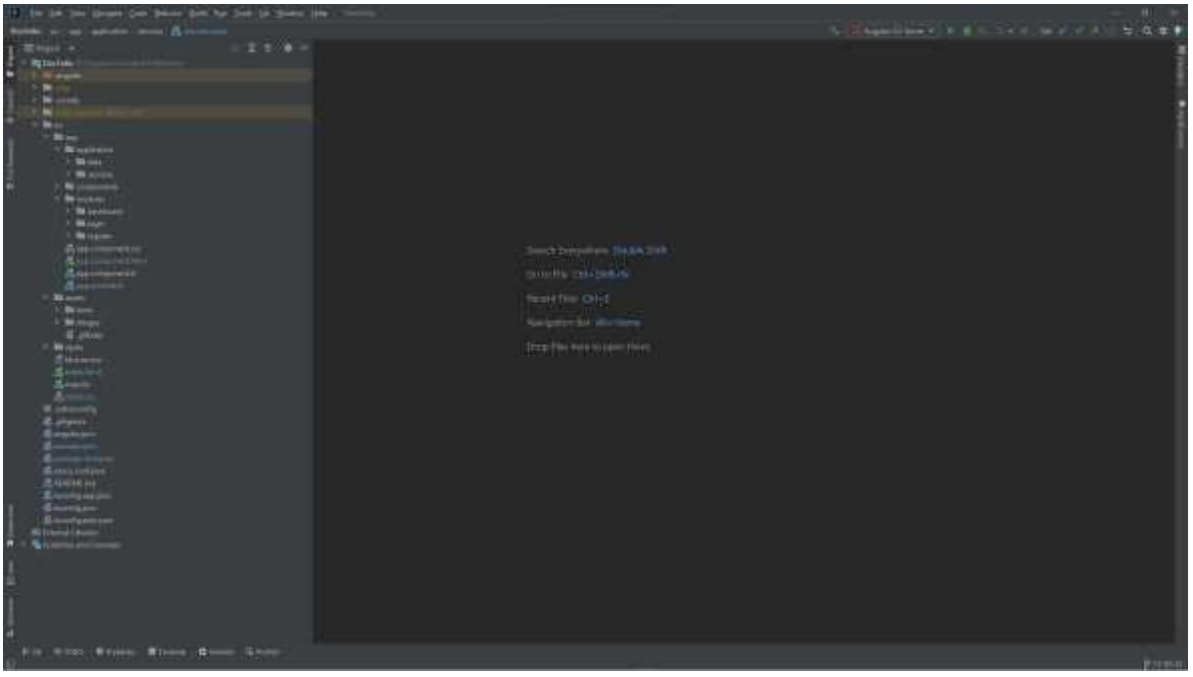

Рисунок 2.1 — Інтерфейс Intellij IDEA

Основні можливості IntelliJ IDEA:

- **Підтримка мов програмування:** Java, Kotlin, Scala, Groovy, C++, Python, PHP, JavaScript, HTML, CSS та багато інших.
- **Інтелектуальне автодоповнення коду:** Прогнозує наступні слова та фрази, економлячи ваш час і зусилля.
- **Рефакторинг коду:** Дозволяє безпечно та легко змінювати структуру коду без ризику помилок.
- **Аналіз коду на льоту:** Виявляє потенційні проблеми та помилки в коді під час його написання.
- **Навігація по коду:** Полегшує переміщення по проекту та пошук необхідних елементів.
- **Відлагодження:** Дозволяє запускати код покроково, встановлювати точки зупину та переглядати значення змінних.
- **Тестування:** Підтримує різні тестові фреймворки, такі як JUnit, TestNG та Cucumber.
- **Інтеграція з системами контролю версій:** Дозволяє легко відстежувати зміни в коді та співпрацювати з іншими розробниками.
- **Широкий спектр налаштувань:** Дозволяє налаштувати IDE під свої потреби та вподобання.

Переваги використання IntelliJ IDEA:

- **Підвищення продуктивності:** Завдяки своїм інтелектуальним функціям IntelliJ IDEA може значно прискорити процес розробки програмного забезпечення.
- **Покращення якості коду:** Інструменти аналізу та рефакторингу коду допомагають писати чистий, лаконічний та надійний код.
- **Зниження ймовірності помилок:** IntelliJ IDEA допомагає виявляти та виправляти помилки на ранніх стадіях розробки.
- **Спрощення співпраці:** Інтеграція з системами контролю версій полегшує роботу над спільними проектами.

IntelliJ IDEA доступний у двох основних виданнях:

- **Community Edition:** Безкоштовна версія з відкритим кодом, яка пропонує більшість основних функцій.
- **Ultimate Edition:** Платна версія, яка включає додаткові функції, такі як підтримку баз даних, розробку вебдодатків та плагіни для професійних інструментів.

IntelliJ IDEA – це чудовий вибір для розробників програмного забезпечення, які шукають потужне, гнучке та надійне середовище розробки.

2.3 Мова програмування TypeScript

TypeScript – це надмножина JavaScript, розроблена компанією Microsoft, яка додає до базової мови статичну типізацію та інші корисні функції. Ці доповнення роблять розробку великих та складних JavaScript-додатків більш надійною та масштабованою [6].

TypeScript використовується у широкому колі програмних проектів, включаючи:

- **Вебдодатки:** TypeScript став популярним вибором для розробки вебдодатків front-end та back-end.

- **Мобільні додатки:** TypeScript використовується для розробки мобільних додатків з використанням фреймворків, таких як React Native і NativeScript.
- **Десктопні додатки:** TypeScript можна використовувати для створення десктопних додатків з використанням фреймворків, таких як Electron.
- **Ігрові розробки:** TypeScript використовується для розробки ігор з використанням ігрових рушіїв, таких як Phaser і Babylon.js.

TypeScript – це потужна мова програмування, яка розширює можливості JavaScript, роблячи розробку великих та складних додатків більш надійною, масштабованою та приємною. Якщо ви розробляєте JavaScript-додатки, TypeScript – це цінний інструмент, який допоможе вам покращити якість вашого коду та продуктивність вашої роботи.

2.3.1 TypeScript на стороні клієнта

Приклади використання TypeScript на стороні клієнта:

- **Інтерактивні вебсторінки:** TypeScript може використовуватися для створення динамічних та інтерактивних вебсторінок, які реагують на дії користувача.
- **Односторінкові вебзастосунки (SPA):** TypeScript добре підходить для розробки SPA, оскільки він дозволяє створювати масштабовані та керовані кодові бази.
- **Вебігри:** TypeScript можна використовувати для розробки вебігор завдяки його можливостям щодо обробки графіки та звуку.

Поширені бібліотеки TypeScript на стороні клієнта:

- **React:** Бібліотека для створення декларативних інтерфейсів користувача.
- **Angular:** Фреймворк для розробки односторінкових вебзастосунків.
- **Vue.js:** Прогресивна JavaScript-фреймворк для створення інтерфейсів користувача.

TypeScript - це цінний інструмент для розробки JavaScript на стороні клієнта, який може покращити надійність, читабельність та масштабованість вашого коду.

Якщо ви розробляєте вебзастосунки, які потребують високої продуктивності та масштабованості, TypeScript - це чудовий вибір.

2.3.2 Переваги TypeScript

Переваги TypeScript:

- **Статична типізація:** TypeScript перевіряє типи змінних та виразів під час компіляції, що допомагає виявити потенційні помилки на ранніх стадіях розробки, економлячи час та зусилля.
- **Підвищена надійність:** Завдяки статичній типізації TypeScript код стає більш стійким до помилок, що призводить до зменшення кількості збоїв та непередбачуваної поведінки.
- **Покращена читабельність:** Код TypeScript стає більш чітким та зрозумілим завдяки чітким визначенням типів та описовим іменам змінних.
- **Інструментарій розробника:** TypeScript пропонує широкий спектр інструментів, таких як автозаповнення коду, перевірка коду та рефакторинг, які роблять процес розробки більш ефективним.
- **Сумісність з JavaScript:** TypeScript код компілюється в звичайний JavaScript, який може виконуватися в будь-якому браузері або серверній платформі, що підтримує JavaScript [7].

Деякі з ключових характеристик TypeScript:

- **Типи даних:** TypeScript підтримує різні типи даних, такі як числа, рядки, булеві значення, масиви, об'єкти та інтерфейси.
- **Інтерфейси:** Інтерфейси в TypeScript описують структуру об'єктів, забезпечуючи узгодженість та покращуючи читабельність коду.
- **Класи:** TypeScript підтримує об'єктно-орієнтоване програмування з класами, екземплярами, методами та спадщиною.
- **Модулі:** TypeScript використовує модулі для організації коду та запобігання конфліктам імен.
- **Дженерики:** Дженерики дозволяють створювати повторно використовуваний код, який може працювати з різними типами даних.

2.4 Фреймворк реалізації веб інтерфейсу Angular.js

Angular.js - це JavaScript-фреймворк з відкритим кодом, розроблений Google, що використовується для створення односторінкових вебдодатків (SPA) [12]. Він використовує архітектуру Model-View-Controller (MVC) та Model-View-ViewModel (MVVM) для спрощення розробки та тестування SPA. Angular.js пропонує ряд функцій, які роблять його популярним вибором для розробників, включаючи:

- **Декларативне програмування:** Angular.js використовує декларативний синтаксис для опису інтерфейсу користувача, що полегшує його читання та розуміння.
- **Компонентний дизайн:** Angular.js заохочує розбивку інтерфейсу користувача на повторно використовувані компоненти, що робить код більш модульним та організованим.
- **Data binding:** Angular.js автоматично синхронізує дані моделі з представленням, що економить час розробників та зменшує схильність до помилок.
- **Directives:** Angular.js пропонує широкий спектр директив, які розширюють можливості HTML, дозволяючи розробникам додавати поведінку до елементів DOM.
- **Dependency injection:** Angular.js використовує впровадження залежностей для спрощення тестування та покращення модульності коду.
- **Routing:** Angular.js має вбудовану систему маршрутизації для управління навігацією в SPA.

2.4.1 Чому Angular.js?

Angular є потужним фреймворком, який слід розглянути. Ось деякі з його ключових особливостей:

1. Структура, орієнтована на компоненти: Angular використовує модульну компонентну архітектуру, подібну до React. Це полегшує розробку та обслуговування складних інтерфейсів користувача.

2. Зв'язування даних: Angular має вбудовану систему зв'язування даних, яка автоматично синхронізує дані моделі з представленням. Це полегшує розробникам роботу з динамічними даними.

3. Директиви: Angular пропонує широкий спектр директив, які розширюють можливості HTML. Це полегшує додавання інтерактивності та поведінки до вебсторінок.

4. Інструментарій: Angular має багатий набір інструментів, які допомагають розробникам у створенні, тестуванні та розгортанні вебдодатків.

5. Спільнота: Angular має велику та активну спільноту розробників, які надають підтримку та ресурси.

6. Стабільність: Angular - це зрілий фреймворк, який підтримується Google. Це означає, що він надійний і має довгострокову підтримку. На рисунку 2.2 можна побачити історію версій Angular.JS

Рисунок 2.2 — Історія версій Angular.JS

2.4.2 Простота

Angular.js прагне до простоти за допомогою MVC структури, директив, шаблонів та системи залежностей. Ці функції можуть допомогти зробити код більш організованим, легшим для розуміння та тестування. Однак, порівняно з React, Angular може мати більш криву навчання через свою декларативну природу та більшу кількість функцій.

2.4.3 Продуктивність

Angular.js має вбудований контейнер залежностей, що значно спрощує процес управління залежностями в проектах. Цей контейнер, відомий як **сервіс-провайдер**, використовується для створення та ін'єкції екземплярів компонентів та сервісів.

Ось деякі ключові моменти про управління залежностями в Angular.js:

- **Сервіс-провайдер:** Цей компонент є серцем системи залежностей Angular.js. Він відповідає за створення та зберігання екземплярів компонентів та сервісів. Розробники можуть реєструвати екземпляри в сервіс-провайдері, надаючи їм унікальні ідентифікатори.
- **Ін'єкція залежностей:** Angular.js використовує механізм ін'єкції залежностей для надання компонетам та сервісам доступу до необхідних їм залежностей. Це робиться шляхом оголошення залежностей в конструкторах компонентів та сервісів, а потім Angular.js автоматично ін'єктує відповідні екземпляри.
- **Переваги:** Вбудований контейнер залежностей Angular.js пропонує ряд переваг [10][11], таких як:
    - **Зменшення коду:** Розробникам не потрібно писати код для ручного управління залежностями, що робить код більш лаконічним та читабельним.
    - **Покращена модульність:** Залежності чітко визначені та декларовані, що робить код більш модульним та легким у тестуванні.
    - **Гнучкість:** Сервіс-провайдер можна налаштувати для різних сценаріїв залежностей.

2.5 Kotlin

Kotlin - це сучасна, кросплатформова, статично типізована, універсальна мова програмування з виведенням типів, що працює поверх JVM (Java Virtual Machine) [22]. Її розробила компанія JetBrains, яка також створила популярні IDE IntelliJ IDEA та Android Studio.

Ось декілька ключових особливостей Kotlin:

- **Сумісність з Java:** Kotlin без проблем взаємодіє з Java-кодом, що робить її чудовим вибором для розробників Android, які хочуть писати більш лаконічний та безпечний код.
- **Безпека:** Kotlin має ряд функцій, які роблять її більш безпечною, ніж Java, наприклад, нульові посилання та функції розширення без ризику.
- **Лаконічність:** Kotlin має чіткий і лаконічний синтаксис, що робить код більш читабельним та легшим для написання та обслуговування.
- **Функціональність:** Kotlin підтримує функціональне програмування, що дозволяє писати більш чіткий та елегантний код.
- **Кросплатформність:** Kotlin можна використовувати для розробки різноманітних програм, включаючи мобільні додатки для Android, вебсайти, серверні програми та багато іншого.

РОЗДІЛ 3. ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1 База даних

База даних побудована на MongoDB. Архітектура бази повністю відповідає тематиці роботи. На рисунку 3.1 представлена схема бази даних:

Рисунок 3.1 — Схема бази даних

3.1.1 Що таке MongoDB?

MongoDB - це безкоштовна база даних NoSQL. NoSQL відрізняється від звичайних баз даних. Вона добре підходить для великих, розподілених даних. MongoDB організовує інформацію про документи. Вона може зберігати, надсилати та отримувати інформацію.

Це допомагає зберігати багато даних. Організації використовують її для багатьох речей, таких як спеціальні пошуки, балансування та кодування.

SQL - це стандартна мова для звичайних баз даних. Вона слідує певній структурі. MongoDB не використовує таблиці та рядки. Натомість вона використовує набори інформації, які називаються колекціями та документами. Документи - це набори даних, в той час як колекції містять ці набори. MongoDB працює з багатьма мовами програмування, такими як C, C++, C#, Go, Java, Python, Ruby та Swift.

3.1.2 Як працює MongoDB?

MongoDB надає сервер для створення баз даних. Дані зберігаються у вигляді записів, що складаються з колекцій та документів.

Документи містять дані, які користувач хоче зберегти. Вони мають пари полів і значень. Вони схожі на JSON, але використовують варіант під назвою

BSON. Він підходить для більшої кількості типів даних. Поля схожі на стовпці реляційної бази даних. Значення можуть бути різних типів, включаючи інші документи і масиви. Первинний ключ - це унікальний ідентифікатор. Структура змінюється додаванням або видаленням полів.

Набори документів - це колекції, як таблиці в базі даних. Вони можуть містити будь-який тип даних і не можуть перебувати в різних базах даних. Користувачі можуть створювати багато баз даних з багатьма колекціями.

До складу MongoDB входить оболонка mongo. Вона підключається до запущеної MongoDB. Користувачі можуть робити запити, оновлювати дані та виконувати адміністративні завдання.

BSON - це двійковий формат для JSON-подібних документів. Автоматичний шардинг розподіляє дані між системами для масштабованості. Він використовує єдиний майстер для узгодженості даних. Вторинні бази даних підтримують копії основної. Операції реплікуються для відмовостійкості.

3.1.3 Для чого використовується MongoDB?

Організація може використовувати MongoDB для наступних цілей:

- Зберігання. MongoDB може зберігати великі обсяги структурованих і неструктурованих даних і масштабується по вертикалі та горизонталі. Індекси використовуються для покращення ефективності пошуку. Пошук також здійснюється за полями, діапазонами та виразами.
- Інтеграція даних. Інтеграція даних для додатків, у тому числі для гібридних і мультихмарних додатків.
- Складні описи структур даних. Бази даних документів дозволяють вбудовувати документи для опису вкладених структур (структура в структурі) і можуть допускати варіації даних.
- Балансування навантаження. MongoDB можна використовувати для роботи на декількох серверах.

3.1.4 Особливості MongoDB

Можливості MongoDB включають в себе наступне:

- Реплікація. Набір реплік - це два або більше екземплярів MongoDB, які використовуються для забезпечення високої доступності. Набори реплік складаються з первинного та вторинного серверів. Первинний сервер MongoDB виконує всі операції читання і запису, в той час як вторинна репліка зберігає копію даних. Якщо первинна репліка виходить з ладу, використовується вторинна репліка.
- Масштабованість. MongoDB підтримує вертикальне та горизонтальне масштабування. Вертикальне масштабування працює шляхом додавання більшої потужності до існуючої машини, в той час як горизонтальне масштабування працює шляхом додавання більшої кількості машин до ресурсів користувача.
- Балансування навантаження. MongoDB виконує балансування навантаження без необхідності використання окремого, спеціального балансувальника навантаження, за допомогою вертикального або горизонтального масштабування.
- Без схем. MongoDB - це база даних без схем, що означає, що база даних може керувати даними без необхідності використання схеми.
- Документ. Дані в MongoDB зберігаються в документах з парами ключ-значення замість рядків і стовпців, що робить дані більш гнучкими в порівнянні з базами даних SQL.

3.1.5 Переваги MongoDB

MongoDB пропонує кілька потенційних переваг:

- Без схем. Як і інші NoSQL бази даних, MongoDB не вимагає попередньо визначених схем. Вона зберігає будь-який тип даних. Це дає користувачам можливість створювати будь-яку кількість полів у документі, що полегшує масштабування баз даних MongoDB у порівнянні з реляційними базами даних.
- Орієнтована на документи. Однією з переваг використання документів є те, що ці об'єкти відображаються у власні типи даних у декількох мовах

програмування. Наявність вбудованих документів також зменшує потребу у з'єднаннях з базами даних, що може знизити витрати.

- Масштабованість. Основною функцією MongoDB є її горизонтальна масштабованість, що робить її корисною базою даних для компаній, які працюють з великими обсягами даних. Крім того, шардинг дозволяє базі даних розподіляти дані між кластерами машин. MongoDB також підтримує створення зон даних на основі ключа шарда.
- Підтримка сторонніх розробників. MongoDB підтримує кілька механізмів зберігання даних і надає API, що підключаються, які дозволяють третім сторонам розробляти власні механізми зберігання даних для MongoDB.
- Агрегація. СУБД також має вбудовані можливості агрегації, що дозволяє користувачам запускати код MapReduce безпосередньо на базі даних, а не запускати MapReduce на Hadoop. MongoDB також включає власну файлову систему під назвою GridFS, подібну до розподіленої файлової системи Hadoop. Ця файлова система використовується в першу чергу для зберігання файлів, розмір яких перевищує обмеження BSON в 16 МБ на документ. Ця схожість дозволяє використовувати MongoDB замість Hadoop, хоча програмне забезпечення бази даних інтегрується з Hadoop, Spark та іншими фреймворками для обробки даних.

3.2 Створення вебзастосунку за допомогою Angular.js

Для забезпечення зручності роботи з даними можна створити вебзастосунок за допомогою Angular.js. Це розв’язує проблему кросплатформності та дозволяє запустити програму на будь-якому пристрої, що має вихід в інтернет [15].

Створення проекту

Для створення проекту потрібно встановити Angular CLI (Command Line Interface), який полегшує створення коду додатків і бібліотек, а також виконання різноманітних поточних завдань розробки, таких як тестування, комплектація та розгортання.

Для встановлення Angular CLI необхідно виконати команду:

npm install -g @angular/cli

Для створення нового проекту потрібно виконати команду:

ng new project-name

Ця команда створить новий проект в поточній директорії з усіма необхідними залежностями і сервером, так що ви можете легко створювати і обслуговувати вашу програму локально. Команда ng serve запускає сервер, стежить за файлами і перебудовує додаток, коли ви зробите зміни в цих файлах. За замовчуванням сервер працює за адресою http://localhost:4200/.

Основні принципи роботи

Основною фундаментальною одиницею Angular є компоненти, які використовуються в якості будівельних блоків. Компоненти можуть приймати різні параметри через декоратори. Для створення компонента необхідно виконати команду:

ng generate component component-name

Структура проекту

У типовому проекті Angular папки і файли організовані наступним чином:

app/: головна директорія для всіх модулів і компонентів додатку.

assets/: директорія для статичних ресурсів, таких як зображення.

environments/: файли конфігурації для різних середовищ (розробка, продакшн).

styles.css: глобальні стилі додатку.

Управління станом

Для управління станом системи можна використовувати бібліотеку NgRx, яка надає потужні інструменти для роботи зі станом в Angular додатках. NgRx реалізує шаблон проектування Redux.

Для встановлення NgRx потрібно виконати команду:

ng add @ngrx/store

Створення store виконується шляхом визначення редюсерів і дій. Редюсери вказують, як змінюється стан програми у відповідь на дії, надіслані до store. Дії не описують, як змінюється стан програми, а лише описують те, що

сталося. Редюсери є чистими функціями — вони завжди повертають однакове значення при певному вводі.

Маршрутизація

Для маршрутизації в Angular використовується Angular Router. Він дозволяє керувати переходами між різними сторінками або компонентами додатку. Для налаштування маршрутизації необхідно визначити маршрути в файлі app-routing.module.ts

3.3 Опис програмної реалізації

Даний програмний продукт був розробений та протестований у программному середовищі розробки Intellij IDEA.

Дана програма складається з таких частин:

— вебклієнт Angular.js;

— база даних MongoDB.

Для використання реалізованого вебзастосунку необхідно мати встановленним будь-який Інтернет браузер. Потрібно переконатися, що ваша система відповідає данним вимогам. Хоча це, як правило, не є проблемою, коли мова йде про вимоги до апаратних засобів, ви можете помітити, що це, наприклад, зовсім інша історія, коли мова йде про підтримувані операційні системи. Користувачі Firefox у Windows 2000, наприклад, помітять, що вони не зможуть оновитись з Firefox 12 на 13 у найближчому майбутньому, оскільки Mozilla знизила підтримку цієї операційної системи, починаючи з цієї версії браузера. Підтримка браузера Google Chrome операційними системами починається з Windows XP SP2, OS X 10.5.6, Ubuntu 10.04, Debian 6, OpenSuse 11.3 та Fedora Linux 14.

Основною фундаментальною одиницею Angular є компоненти, які використовуються в якості будівельних блоків. Компоненти можуть приймати різні параметри. Ці параметри називаються props.

На рисунку 3.2 представлена файлова структура вебклієнта:

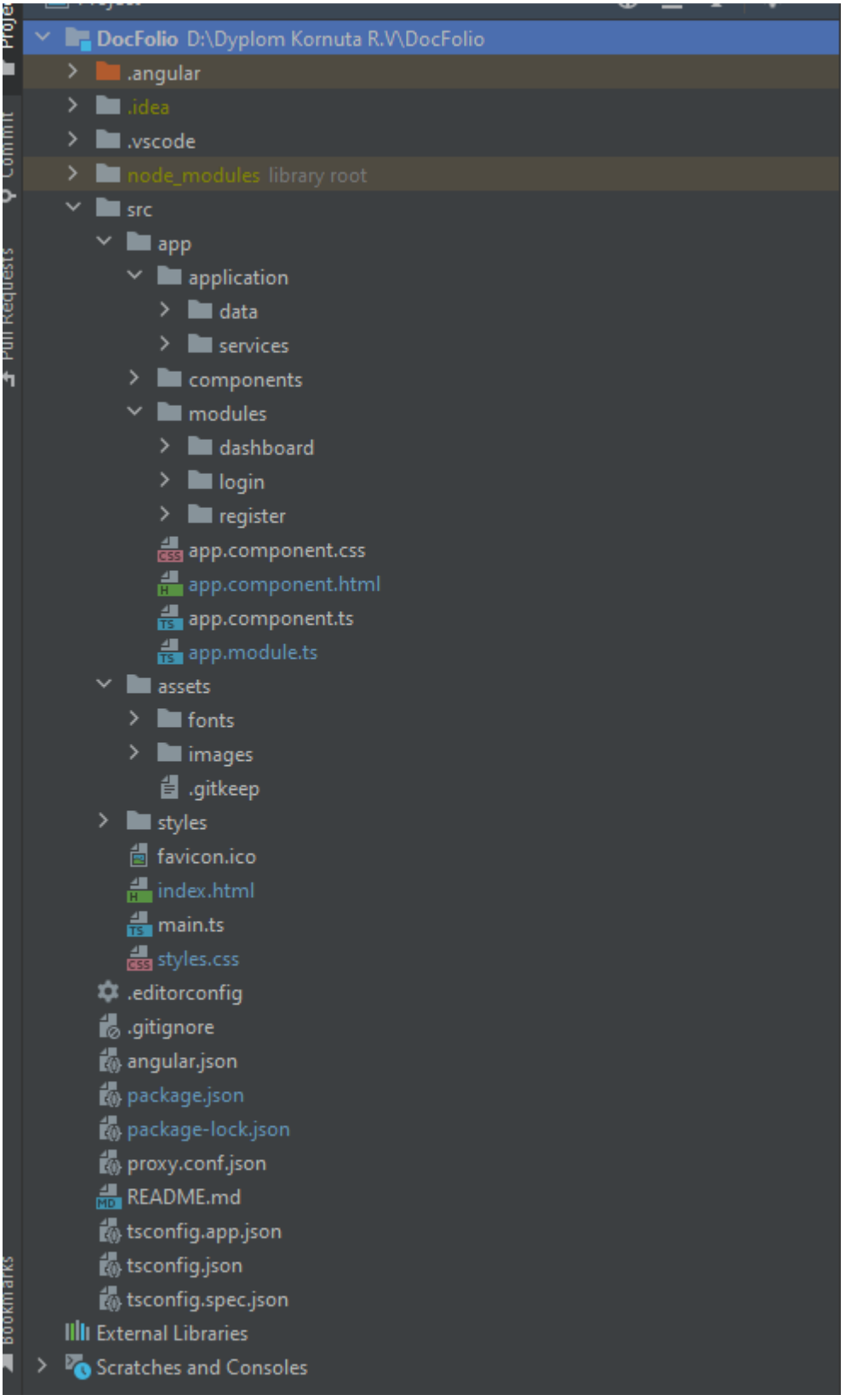

Рисунок 3.2 — Файлова стуктура вебклієнта

В папці components знаходяться основні компоненти застосунку, в папці modules знаходяться основні сторінки вебзастосунку, в папці application є ще дві папки, data, де знаходяться описані класи об’єктів таких як User, Document, Folder і тд. та services, де знаходяться сервіси які для роботи з API для

получення даних з бекенду, interceptor та guard які перевіряють чи користувач авторизований, якщо так, то пускають його на наступну сторінку, тобто document-gallery, якщо ні, повертають його на сторінку для авторизації. Вся маршрутизація знаходиться в файлі app.module.ts, додаткові стиля в папці styles, усі шрифти та зображення в папці assets/fonts, assets/images.

Для написання бекенду на мові програмування Kotlin я використав вебзастосунок spring initializr [21]. За допомогою нього можна вибрати на якій мові програмування буде писатись бекенд, вибрати версію Spring Boot'а та додати залежності для бекенду.

На рисунку 3.3 представлена файлова структура бекенду:

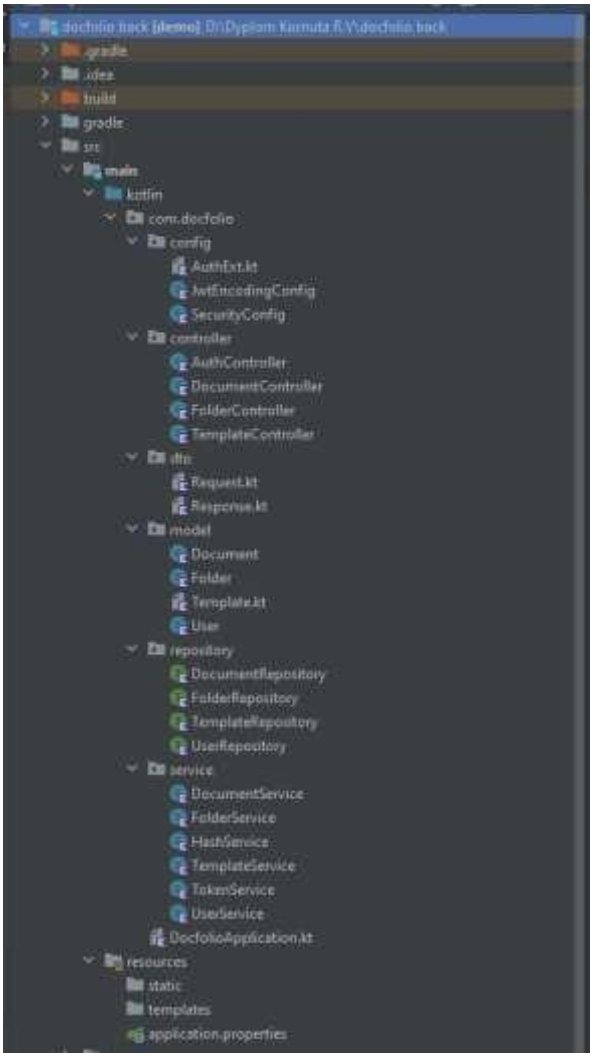

Рисунок 3.3 — Файлова структура бекенду

В папці config знаходяться кофігурації для авторизації та декодування Jwt Token'а, controller – контролери, в яких знаходяться прописані endpoint'и, по яким фронтенд може звертатись до бекенду та получати дані, dto – request'и та

response’и, а саме описані об’єкти які позначають що саме фронтенд повинен прислати бекенду на певний endpoint, та що саме бекенд повинен повернути фронтенду, model – описані класи для об’єктів Document, Folder, Template та User, в папці repository знаходиться описані інтерфейси до яких підключена база даних, ці інтерфейси використовуються в сервісах для роботи з базою даних, services – описані функціонали такі як створення, редагування, повернення різних об’єктів.

Щоб реалізувати доступ до системи лише авторизованим корисувачам, було розробленно авторизацію (рисунок 3.4) та реєстрацію (рисунок 3.5).

Рисунок 3.4 — Сторінка авторизації

Рисунок 3.5 — Сторінка реєстрації

Після успішної авторизації корисувач має змогу шукати, переглядати створені документи, сортувати їх, а також створювати документи (рисунок 3.6).

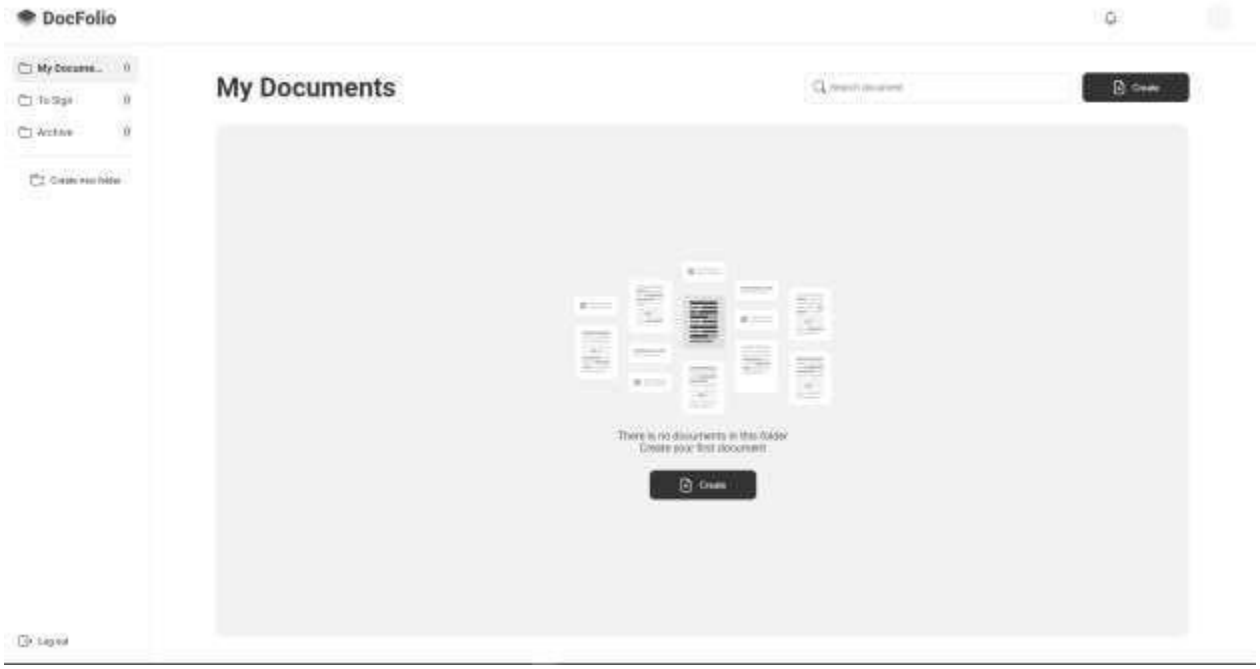

Рисунок 3.6 — Галерея документів

Користувач має змогу створювати папки та розподіляти створені документи по ним (рисунок 3.7).

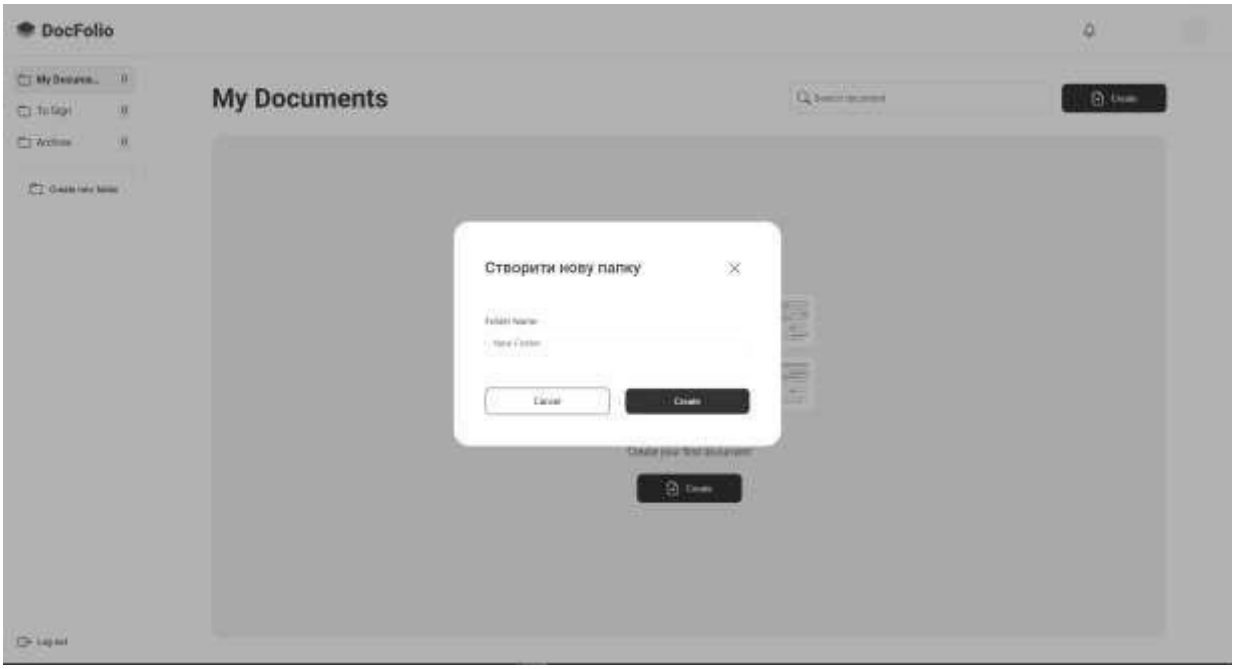

Рисунок 3.7 — Модальне вікно створення папки

Також користувач має змогу редагування назви папки (рисунок 3.8). та видалення папки, для видалення папки користувач має підтвердити свої дії (рисунок 3.9).

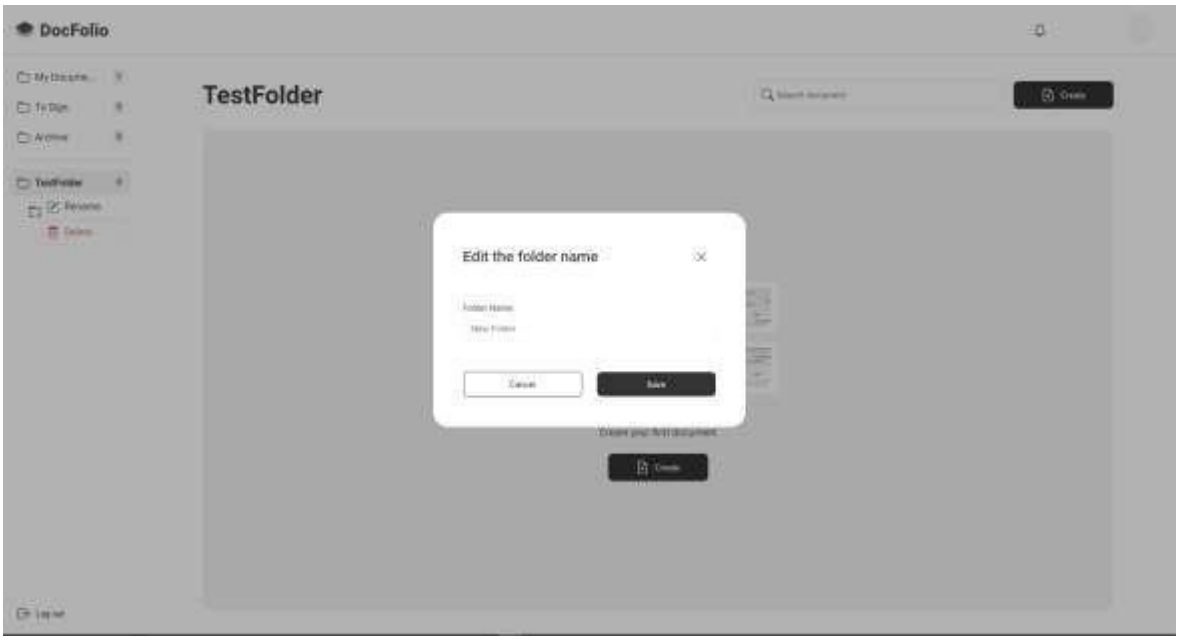

Рисунок 3.8 — Модальне віко редагування назви папки

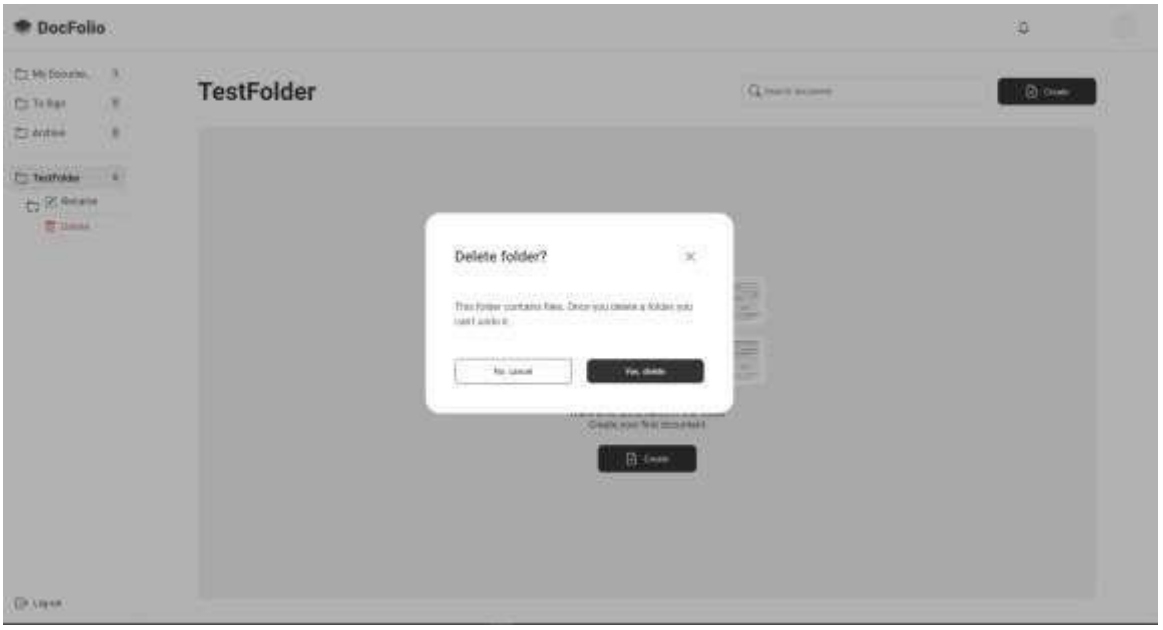

Рисунок 3.9 — Модальне вікно видалення папки

Якщо користувачу потрібно створити новий документ, він може відкрити модальне вікно та вибрати потрібний йому шаблон для документу (рисунок 3.10).

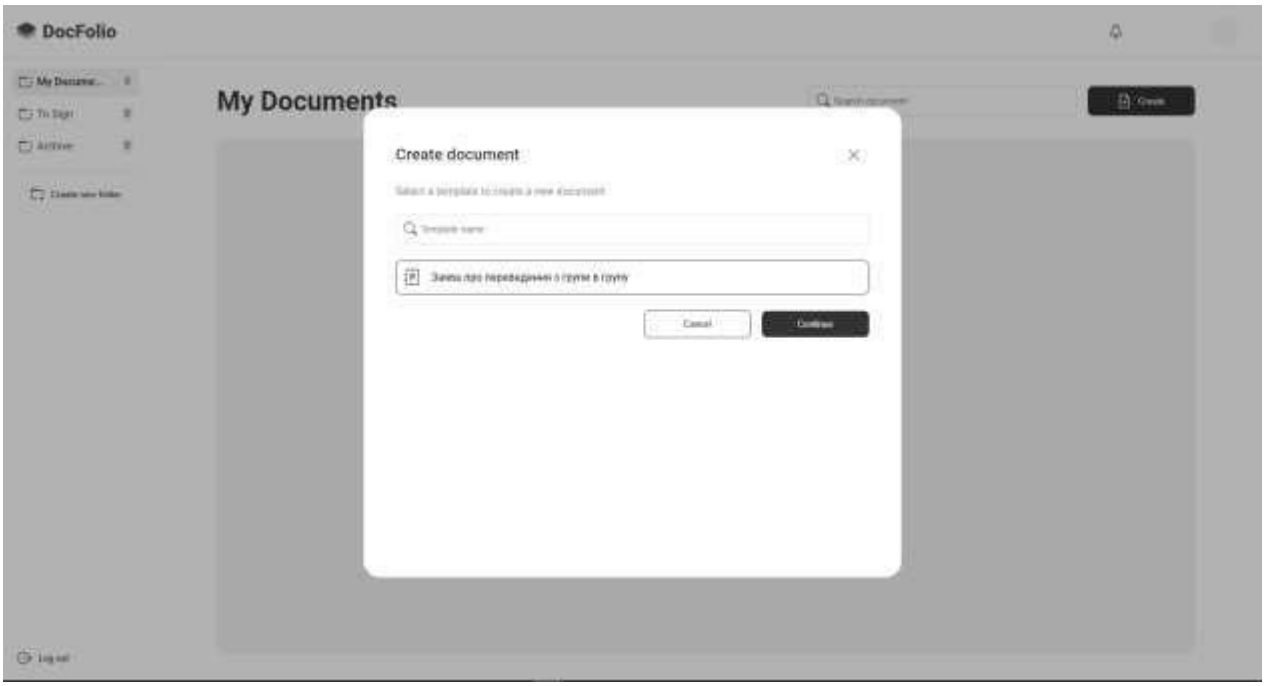

Рисунок 3.10 — Модальне вікно вибору шаблону документа

Після того як користувач вибрав шаблон документу, йому потрібно заповнити поля для даних (рисунок 3.11).

Рисунок 3.11 — Зразок інтерфейсу дипломної роботи студента

Галерея документів після створення документу (рисунок 3.12).

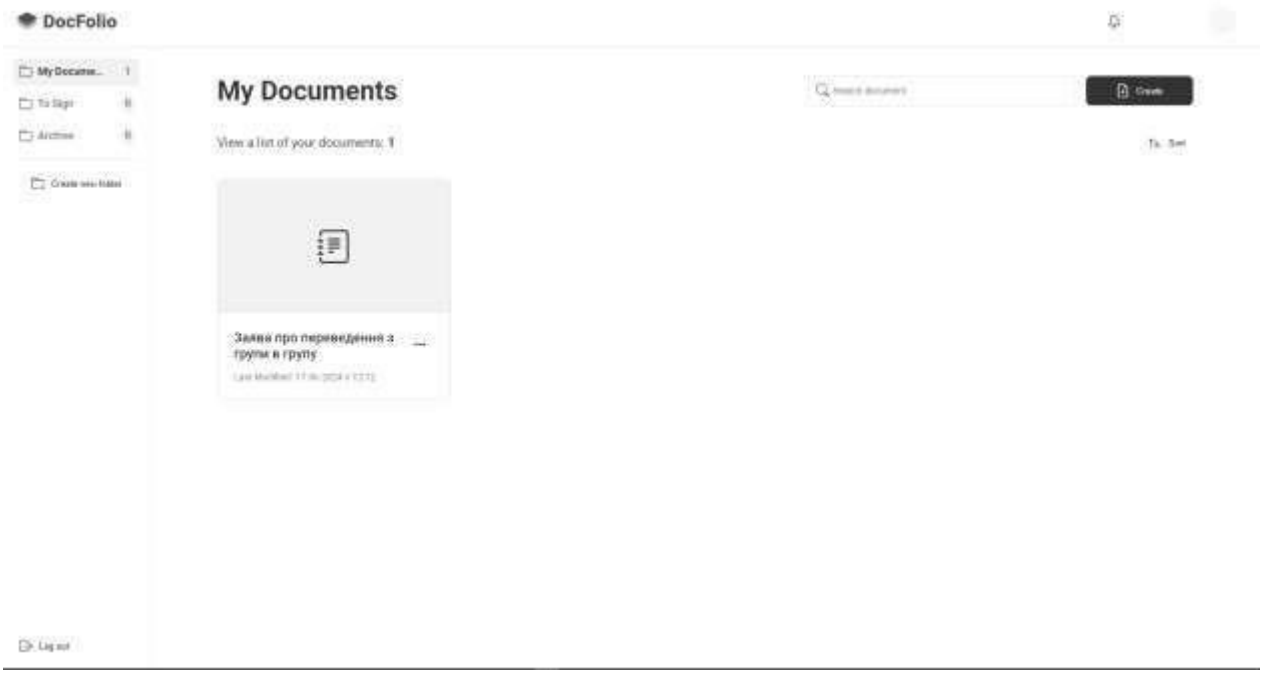

Рисунок 3.12 — Галерея документів

Користувач має можливість вибрати документ, та редагувати його в разі потреби (рисунок 3.13).

Рисунок 3.13 — Редагування документу

ВИСНОВКИ

В результаті виконаної роботи було розроблено клієнт-серверну систему автоматизованого формування документів, яка надає можливість створювати документи по заготовленим шаблонам, створювати та редагувати документи та папки.

Проаналізовано існуючі системи управління проектами, визначенно їх переваги та недоліки. Проаналізовано вимоги до нової системи управління дипломними проектами студента.

Базу даних розроблено за допомогою системи керування базами даних MongoDB. Її будова відповідає функціям та вимогам розроблюваної системи.

З використанням фреймворку Angular.js та мови програмування TypeScript реалізовано графічний вебзастосунок для роботи з документами. Завдяки тому, що програмна система написана за допомогою сучасної мови програмування та її провідних бібліотек, гарантується з часом підтримка всіх компонентів та їх подальша модернізація.

Дана система значно спрощує процес роботи з документами.

Реалізовані можливості програмного продукту повністю задовільняють поставленій задачі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. “NPM” [Електронний ресурс] Режим доступу: https://github.com/npm/cli (дата звернення 21.05.2024)
2. “Intellij IDEA overview” [Електронний ресурс] Режим доступу: https://www.jetbrains.com/help/idea/discover-intellij-idea.html (дата звернення 19.05.2024)
3. “Development in Intellij IDEA” [Електронний ресурс] Режим доступу: https://docs.nomagic.com/display/MD2021x/Development%2Bin%2BIntelliJ%2BIDEA (дата звернення 18.05.2024)
4. Hiren Dhaduk. “Angular vs React: Which to Choose for Your Front End in 2024?” [Електронний ресурс] Режим доступу: https://www.simform.com/blog/angular-vs-react/ (дата звернення 17.05.2024)
5. “TypeScript проти JavaScript – детальне порівняння” [Електронний ресурс] Режим доступу: https://youjs.org/typescript-proty-javascript-detalne-porivniannia/ (дата звернення 13.05.2024)
6. “TypeScript Documentation” [Електронний ресурс] Режим доступу: https://www.typescriptlang.org/docs/ (дата звернення 12.05.2024)
7. “TypeScript проти JavaScript – різниця між ними” [Електронний ресурс] Режим доступу: https://www.guru99.com/uk/typescript-vs-javascript.html (дата звернення 16.05.2024)
8. “Що таке TypeScript і навіщо він потрібен” https://foxminded.ua/typescript/ (дата звернення 15.05.2024)
9. Boris Cherny. “Programming TypeScript: Making Your JavaScript Applications Scale” [Електронний ресурс] Режим доступу: https://www.amazon.com/Programming-TypeScript-Making-JavaScript-Applications/dp/1492037656 (дата звернення 15.05.2024)
10. Anupriya Singh. “Top AngularJS Benefits and Features in 2024” [Електронний ресурс] Режим доступу: https://www.turing.com/resources/benefits-of-hiring-angular-developers (дата звернення 20.05.2024)

11. “AngularJs Advantages and Disadvantages – Why AngularJS is Popular” [Електронний ресурс] Режим доступу: https://data-flair.training/blogs/angularjs-advantages-and-disadvantages/ (дата звернення 10.05.2024)
12. “What is Angular?” [Електронний ресурс] Режим доступу: https://angular.dev/overview (дата звернення 14.05.2024)
13. “MongoDB Documentation” [Електронний ресурс] Режим доступу: https://www.mongodb.com/docs/ (дата звернення 12.05.2024)
14. “Doclime” [Електронний ресурс] Режим доступу: https://doclime.com/ (дата звернення 9.05.2024)
15. “Explain the Architecture Overview of Angular?” [Електронний ресурс] Режим доступу: https://www.geeksforgeeks.org/explain-the-architecture-overview-of-angular/ (дата звернення 21.05.2024)
16. David Flanagan. “JavaScript: The Definitive Guide: Master the World’s Most-Used Programming Language 7th Edition” [Електронний ресурс] Режим доступу: https://kupichitay.com.ua/product/999-grn-javascript-the-definitive-guide-master-the-worlds-most-used-programming-language-7th-edition-david-flanagan/ (дата звернення 19.05.2024)
17. Adam Freeman. “Pro Angular” [Електронний ресурс] Режим доступу: https://www.amazon.com/Pro-Angular-6-Adam-Freeman/dp/1484236483 (дата звернення 11.05.2024)
18. Eric Meyer. “CSS Pocket Reference: Visual Presentation for the Web 4th Edition” [Електронний ресурс] Режим доступу: https://www.amazon.com/CSS-Pocket-Reference-Visual-Presentation/dp/1449399037 (дата звернення 13.05.2024)
19. JonDuckett. “HTML and CSS: Design and Build Websites” [Електронний ресурс] Режим доступу: https://www.amazon.com/HTML-CSS-Design-Build-Websites/dp/B07JK2FKRS (дата звернення 12.05.2024)

20. “IntelliJ IDEA vs WebStorm: What are the differences?” [Електронний ресурс] Режим доступу: https://stackshare.io/stackups/intellij-idea-vs-webstorm (дата звернення 17.05.2024)
21. “Spring initializr” [Електронний ресурс] Режим доступу: https://start.spring.io (дата звернення 15.05.2024)
22. “Kotlin” [Електронний ресурс] Режим доступу: https://kotlinlang.org/docs/home.html (дата звернення 6.05.2024)
23. “MongoDB” [Електронний ресурс] Режим доступу: https://www.techtarget.com/searchdatamanagement/definition/MongoDB (дата звернення 2.05.2024)

ДОДАТКИ

Додаток А

Сторінка галереї документів, document-gallery.component.html

```
<div class="dashboard">

  <div class="dashboard__header">
    <h1 class="dashboard__header-title">{{this.folderService.selectedFolder.value.name}}</h1>
    <div class="dashboard__header-actions">
      <docfolio-search width="400px" placeholder="Search document"
                       [value]="documentSearchRequestService.searchRequest.value.searchQuery"
                       (valueChange)="handleSearchRequest($event)"></docfolio-search>
      <docfolio-button *ngIf="this.documentService.documents.length ||
folderService.selectedFolder.value.name !== 'Archive'" icon="file-add" title="Create"
(click)="openDocumentModal()"></docfolio-button>
    </div>
  </div>

  <div class="dashboard__subheader" *ngIf="this.documentService.documents.length">
    <h3 class="dashboard__subheader-title">View a list of your documents: <span
class="bold">{{this.documentService.documents.length}}</span></h3>
    <div class="dashboard__subheader-actions">
      <docfolio-button icon="sort" title="Sort" type="TEXT" size="SMALL"
(click)="onSort()"></docfolio-button>
    </div>
  </div>

  <div class="dashboard__section-grid" *ngIf="this.documentService.documents.length">
    <docfolio-document-card *ngFor="let document of this.documentService.sortedDocuments"
[document]="document"></docfolio-document-card>
  </div>

  <div class="dashboard__empty" *ngIf="!this.documentService.documents.length">
    <img  src="assets/images/documents-empty-state.svg">
    <h3>There is no documents in this folder
```

```
    <span *ngIf="folderService.selectedFolder.value.name !== 'Archive'">Create your first
document</span></h3>
   <docfolio-button title="Create" icon="file-add" *ngIf="folderService.selectedFolder.value.name
!== 'Archive'" (click)="openDocumentModal()"></docfolio-button>
  </div>

</div>

<docfolio-document-modal   *ngIf="isDocumentModalOpened"></docfolio-document-modal>
```

Функціональна частина галереї документів, document-gallery.component.ts

```
import {Component, OnInit} from "@angular/core";
import {TemplateModalService} from "../../../application/services/template-modal.service";
import {FolderService} from "../../../application/services/folder.service";
import {DocumentService} from "../../../application/services/document.service";
import {DocumentsSearchRequestService} from "../../../application/services/documents-search-
request.service";



enum SortDirection {
  DEFAULT = "DEFAULT",
  ASC = "ASC",
  DESC = "DESC"
}

@Component({
  templateUrl: 'document-gallery.component.html',
  styleUrls: [ 'document-gallery.component.scss' ]
})

export class DocumentGalleryComponent implements OnInit {
  isDocumentModalOpened: boolean = false;
  sortDirection: SortDirection = SortDirection.DEFAULT
  constructor(public documentModalService: TemplateModalService, public folderService:
FolderService, public documentService: DocumentService, public documentSearchRequestService:
```

```
DocumentsSearchRequestService) {}

  ngOnInit() {
    this.documentModalService.isOpened.subscribe((status: boolean) => {
      this.isDocumentModalOpened = status
    })
  }

  onSort() {
    switch (this.sortDirection) {
      case SortDirection.ASC:
        this.documentService.sortedDocuments.sort((a, b) => a.title.localeCompare(b.title))
        this.sortDirection = SortDirection.DESC
        break;
      case SortDirection.DESC:
        this.sortDirection = SortDirection.DEFAULT
        this.documentService.sortedDocuments.sort((a, b) => b.title.localeCompare(a.title))
        break;
      default:
        this.sortDirection = SortDirection.ASC
        this.documentService.sortedDocuments = this.documentService.documents
        break;
    }
  }

  handleSearchRequest(searchQuery: string) {
    this.documentSearchRequestService.searchRequest.value.searchQuery = searchQuery
    this.documentService.fetchData().subscribe()
  }

  openDocumentModal() {
    this.documentModalService.isOpened.next(true)
  }
}
```

Шаблон документу для переведення із групи в групу, transfer-from-group.component.html

```
<div class="box">
  <div class="header">
    <div class="header__container-info">
      <h3 class="header__title">{{getText('header-position')}}</h3>
      <h3 class="header__title">{{getText('header-university')}}</h3>
      <h3 class="header__title">{{getText('header-person')}}</h3>
    </div>
    <div class="header__container">
      <h3 class="header__subtitle">{{getText('subheader')}}<span class="header__subtitle-text">
{{course}}</span></h3>
      <div class="form__item">
        <span class="form__item-text">{{this.request.value.values['educationForm']}}</span>
        <span class="form__item-info">(форма здобуття освіти)</span>
      </div>
      <div class="form__item">
        <span class="form__item-text">{{this.request.value.values['faculty']}}</span>
        <span class="form__item-info">(факультет)</span>
      </div>
      <div class="form__item">
        <span class="form__item-text">{{this.request.value.values['fullName']}}</span>
        <span class="form__item-info">(прізвище, ім’я, по батькові студента)</span>
      </div>
    </div>
  </div>
  <div class="body">
    <div class="body__header">
      <h2 class="body__title">{{getText('title')}}</h2>
      <h3 class="body__subtitle">{{getText('subtitle')}}</h3>
    </div>
    <div class="body__text">
      {{getText('groupFrom')}}
      <div class="form__item">
        <span class="form__item-text">{{this.request.value.values['groupFrom']}}</span>
```

```
      <span class="form__item-info">(шифр групи)</span>
    </div>
    {{getText('groupTo')}}
    <div class="form__item">
      <span class="form__item-text">{{this.request.value.values['groupTo']}}</span>
      <span class="form__item-info">(шифр групи)</span>
    </div>
  </div>
  <div class="body__text">
    {{getText('reason')}}
    <div class="form__item">
      <span class="form__item-text">{{this.request.value.values['reason']}}</span>
      <span class="form__item-info">(тип даних)</span>
    </div></div>
  <div class="body__container">
    <div class="form__item">
      <span class="form__item-text">{{this.request.value.values['applicationDate']}}</span>
      <span class="form__item-info">(дата подання заяви)</span>
    </div>
    <div class="form__item">
      <span class="form__item-text">{{this.request.value.values['sign']}}</span>
      <span class="form__item-info">(підпис)</span>
    </div>
  </div>
</div>
<div class="footer">
  <h3 class="footer__header">{{getText('footer')}}</h3>
  <div class="footer__text">
    {{getText('footer-item')}}
    <div class="footer__text-container">
      <div class="form__item">
        <span class="form__item-text"></span>
        <span class="form__item-info">(назва факультету)</span>
      </div>
      <div class="form__item">
```

```
        <span class="form__item-text"></span>
        <span class="form__item-info">(дата)</span>
      </div>
      <div class="form__item">
        <span class="form__item-text"></span>
        <span class="form__item-info">(підпис, власне ім’я та прізвище)</span>
      </div>
    </div>
  </div>
 </div>
</div>
```

Функціональна частина шаблону transfer-from-group.component.ts

```
import {Component, Input} from "@angular/core";
import {TemplateService} from "../../../../../application/services/template.service";
import {DocumentRequest} from "../../../../../application/data/Document";
import {BehaviorSubject} from "rxjs";
import {Course} from "../../../../../application/data/UniversityData";

@Component({
 selector: 'template-transfer-from-group',
 templateUrl: 'transfer-from-group.component.html',
 styleUrls: [ 'transfer-from-group.component.scss' ]
})

export class TransferFromGroupComponent {
 courseValues: Array<Course> = Course.values
 @Input() request: BehaviorSubject<DocumentRequest> = new
BehaviorSubject<DocumentRequest>(new DocumentRequest())

 constructor(public templateService: TemplateService) {}

 getText(name: string): string {
  const field = this.templateService.selectedTemplate.value.preparedFields.find(f => f.name ===
name);
```

```
    return field ? field.text : '';
  }

  get course() {
    if(this.request.value.values['course']) {
      return this.courseValues.find(course => course.code ===
this.request.value.values['course'])!!.name
    } else return ''
  }
}
```

Сторінка створення документу, document-management.component.html

```
<div class="template" *ngIf="templateService.selectedTemplate.value">

  <div class="canvas__wrapper">
    <div class="template__header">{{templateService.selectedTemplate.value.name}}</div>
    <div class="canvas" #canvas style="padding: 60px 80px; background: white;">
      <template-transfer-from-group   [request]="request"></template-transfer-from-group>
    </div>
  </div>
  <div class="sidebar">
    <div class="sidebar__header">
      <h4>Основні дані</h4>
      <div class="empty"></div>
    </div>

    <div class="form">
      <ng-container *ngFor="let field of templateService.selectedTemplate.value.fields">
        <div [ngSwitch]="field.type">
          <docfolio-input *ngSwitchCase="'TEXT'" [label]="field.label"
[(value)]="request.value.values[field.name]"></docfolio-input>
          <docfolio-select *ngSwitchCase="'SELECT'" [label]="field.label" [options]="field.options!!"
[(value)]="request.value.values[field.name]"></docfolio-select>
          <docfolio-radio-group *ngSwitchCase="'OPTION'" [label]="field.label"
[options]="field.options!!" [(checked)]="request.value.values[field.name]"></docfolio-radio-
```

```
group>
        </div>
      </ng-container>
    </div>

    <div class="sidebar__actions">
      <docfolio-button title="Cancel" type="OUTLINED" (click)="onCancel()"></docfolio-button>
      <docfolio-button title="Save" (click)="onSave()"></docfolio-button>
    </div>

  </div>

</div>
```

Функціональна частина сторінки створення документу, document-management.component.ts

```
import {Component, OnInit, ViewChild} from "@angular/core";
import html2canvas from "html2canvas";
import { jsPDF } from "jspdf";
import {TemplateService} from "../../../application/services/template.service";
import {ActivatedRoute, Router} from "@angular/router";
import {DocumentRequest} from "../../../application/data/Document";
import {BehaviorSubject} from "rxjs";
import moment from "moment";
import {DocumentService} from "../../../application/services/document.service";
import {FolderService} from "../../../application/services/folder.service";

@Component({
  templateUrl: 'document-management.component.html',
  styleUrls: [ 'document-management.component.scss' ]
})

export class DocumentManagementComponent implements OnInit {
  @ViewChild('canvas') private canvas: any;
  documentId: string = ''
```

```
  request: BehaviorSubject<DocumentRequest> = new BehaviorSubject<DocumentRequest>(new DocumentRequest())

  constructor(public templateService: TemplateService, private router: Router, private activatedRoute: ActivatedRoute, public documentService: DocumentService, public folderService: FolderService) {}

  ngOnInit() {
    this.documentId = this.activatedRoute.snapshot.params['id']
    if(this.documentId) {
    this.documentService.fetchById(this.documentId).subscribe(response => {
       this.request.value.values = response.values
      })
     const document = this.documentService.documents.find(document => document.id === this.documentId)!!
      this.templateService.fetchById(document.templateId).subscribe(response => this.templateService.selectedTemplate.next(response))
    } else {
     this.initFormData()
    }
  }

  initFormData() {
    this.templateService.selectedTemplate.value.fields.map(field => {
    this.request.value.values[field.name] = ''
    })

    const date = new Date()

    this.request.value.values['applicationDate'] = moment(date).format('D MMMM YYYY [p.]')
  }

  onSave() {
    this.request.value.title = this.templateService.selectedTemplate.value.name
    this.request.value.templateId = this.templateService.selectedTemplate.value.id
```

```
  this.request.value.folderId = this.folderService.selectedFolder.value.id
  if(this.documentId) {
    const date = new Date()
    this.request.value.modifiedAt = date
    this.documentService.edit(this.documentId,  this.request.value)
      .subscribe((response => this.onCancel()))
  } else {
    this.documentService.create(this.request.value)
      .subscribe((response => this.onCancel()))
  }
 }

 onCancel() {
  this.router.navigate(['/dashboard'])
 }

 generatePDF() {
  if (this.canvas.nativeElement) {
    html2canvas(this.canvas.nativeElement, {scale: 2}).then(canvas => {
    const imgWidth = 208;
      const pageHeight = 295;
      const imgHeight = canvas.height * imgWidth / canvas.width;
      let heightLeft = imgHeight;

      const contentDataURL = canvas.toDataURL('image/png');
      const pdf = new jsPDF('p', 'mm', 'a4');
      let position = 0;

      pdf.addImage(contentDataURL, 'PNG', 0, position, imgWidth, imgHeight);
      pdf.save('download.pdf');
    });
  }
 }

}
```

Сторінка авторизації, login.component.html

```
<div class="login">
  <div class="login__container">
    <h1 class="login__title">Welcome Back!</h1>
    <h3 class="login__subtitle">Login to your account</h3>
    <docfolio-input label="Username" placeholder="Enter your username" width="600px"
[(value)]="username"></docfolio-input>
    <docfolio-input label="Password" placeholder="Enter your password" width="600px"
type="password" [(value)]="password"></docfolio-input>
    <div class="login__actions">
      <docfolio-button type="OUTLINED" title="Register"
(click)="navigateToRegister()"></docfolio-button>
      <docfolio-button title="Log In" (click)="login()"></docfolio-button>
    </div>
  </div>
</div>
```

Функціональна частина сторінки авториазації, login.component.ts

```
import {Component, HostListener} from "@angular/core";
import {Router} from "@angular/router";
import {AuthService} from "../../application/services/auth.service";
import {LoginData} from "../../application/data/LoginData";

@Component({
  templateUrl: 'login.component.html',
  styleUrls: [ 'login.component.scss' ]
})

export class LoginComponent {
  username: string = ''
  password: string = ''
  constructor(private router: Router, private authService: AuthService) {}
  login() {
    const request = new LoginData(this.username, this.password)
```

```
    this.authService.login(request)
      .subscribe(response => this.router.navigate(['/dashboard']))
  }

  navigateToRegister() {
    this.router.navigate(['/register'])
  }

  @HostListener('document:keydown', ['$event'])
  enterHandle(e: KeyboardEvent) {
    if(e.key === 'Enter') {
      this.login()
    }
  }
}
```

Модальне вікно вибору шаблону документа, template-modal.component.html

```
<div class="modal">
  <div class="modal__header">
    <div class="modal__title">Create document</div>
    <docfolio-button icon="cross" type="TEXT" size="SMALL" (click)="onClose()"></docfolio-
button>
  </div>
  <div class="modal__body">
    <div class="modal__subtitle">Select a template to create a new document</div>
    <docfolio-search placeholder="Template name"
(valueChange)="handleSearchRequest($event)"></docfolio-search>
    <div class="template">
      <div class="template_item" *ngFor="let template of templateService.templates"
                          (click)="onTemplateSelect(template)"
                          [class.active]="isTemplateSelected(template)">
        <i class="icon-journal"></i>
        <div class="template__info">
          <h3 class="template__title">{{template.name}}</h3>
        </div>
      </div>
    </div>
  </div>
  <div class="modal__footer">
    <docfolio-button title="Cancel" type="OUTLINED" (click)="onClose()"></docfolio-button>
    <docfolio-button title="Continue" (click)="onContinue()"></docfolio-button>
  </div>
</div>

<div class="backdrop" (click)="onClose()"></div>
```

Функціональна частина модального вікна вибору шаблону документа, template-modal.component.ts

```
import {Component, OnInit} from "@angular/core";
import {TemplateModalService} from "../../application/services/template-modal.service";
import {Template} from "../../application/data/Template";
```

```
import {TemplateService} from "../../application/services/template.service";
import {Router} from "@angular/router";
import {FolderService} from "../../application/services/folder.service";
import {TemplateSearchRequestService} from "../../application/services/template-search-
request.service";

@Component({
  selector: 'docfolio-document-modal',
  templateUrl: './template-modal.component.html',
  styleUrls: [ './template-modal.component.scss' ]
})

export class TemplateModalComponent implements OnInit {
  isModalOpened: boolean = false
  constructor(public templateModalService: TemplateModalService, public
templateSearchRequestService: TemplateSearchRequestService, public templateService:
TemplateService, private router: Router, private folderService: FolderService) {}
ngOnInit() {
    this.isModalOpened = this.templateModalService.status
    this.templateModalService.isOpened.subscribe((status: boolean) => {
    this.isModalOpened = status
    })
    this.templateService.fetchData().subscribe()
  }
  onTemplateSelect(template: Template) {
    this.templateService.selectedTemplate.next(template)
  }
  isTemplateSelected(template: Template): boolean {
    return this.templateService.selectedTemplate.value === template
  }
  onContinue() {

this.router.navigate([`/dashboard/${this.folderService.selectedFolder.value.id}/document/create`])
    this.onClose()
  }
```

```
  onClose() {
    this.templateModalService.isOpened.next(false)
  }

  handleSearchRequest(searchQuery: string) {
    this.templateSearchRequestService.searchRequest.value.searchQuery = searchQuery
    this.templateService.fetchData().subscribe()
  }
}
```

Сервіс документа з описаними функціями фетчання та редагування документа, document.service.ts

```
import {Injectable} from "@angular/core";
import {HttpClient} from "@angular/common/http";
import {BehaviorSubject, Observable, tap} from "rxjs";
import {FolderService} from "./folder.service";
import {Document, DocumentRequest} from "../data/Document";
import {DocumentsSearchRequestService} from "./documents-search-request.service";

@Injectable({
  providedIn: 'root'
})
export class DocumentService {
  documents: Array<Document> = []
  sortedDocuments: Array<Document> = []
  selectedDocument: BehaviorSubject<Document> = new
BehaviorSubject<Document>(this.documents[0])

  constructor(private http: HttpClient, private folderService: FolderService, private
documentSearchRequestService: DocumentsSearchRequestService) {}

  fetchData(): Observable<any> {
    return this.fetch()
      .pipe(tap(response => {
        this.selectedDocument.next(response[0])
```

```
        this.documents = response
        this.sortedDocuments = response
      }))
  }

  fetch(): Observable<any> {
    return
this.http.get(`api/documents/${this.folderService.selectedFolder.value.id}?order=${this.documentS
earchRequestService.searchRequest.value.order}&searchQuery=${this.documentSearchRequestSer
vice.searchRequest.value.searchQuery}`);
  }

  fetchById(id: string): Observable<any> {
    return this.http.get(`api/documents/byId/${id}`);
  }

  create(request: DocumentRequest): Observable<any> {
    return this.http.post('api/documents/create', request);
  }

  edit(id: string, request: DocumentRequest): Observable<any> {
    return this.http.post(`api/documents/edit/${id}`, request);
  }

  delete(id: string) {
    return this.http.delete(`api/documents/delete/${id}`, {})
  }
}
```

Додаток Б

Контролер авторизації, AuthController.kt

```
package com.docfolio.controller

import com.docfolio.dto.ApiException
import com.docfolio.dto.LoginDto
import com.docfolio.dto.LoginResponseDto
import com.docfolio.dto.RegisterDto
import com.docfolio.model.User
import com.docfolio.service.FolderService
import com.docfolio.service.HashService
import com.docfolio.service.TokenService
import com.docfolio.service.UserService
import org.springframework.web.bind.annotation.PostMapping
import org.springframework.web.bind.annotation.RequestBody
import org.springframework.web.bind.annotation.RequestMapping
import org.springframework.web.bind.annotation.RestController

@RestController
@RequestMapping("/api/user")
class AuthController(
    private val hashService: HashService,
    private val tokenService: TokenService,
    private val userService: UserService,
    private val folderService: FolderService
) {
    @PostMapping("/login")
    fun login(@RequestBody payload: LoginDto): LoginResponseDto {
        val user = userService.findByName(payload.name) ?: throw ApiException(400, "Login
failed")

        if (!hashService.checkBcrypt(payload.password, user.password)) {
            throw ApiException(400, "Login failed")
        }
```

```
        return LoginResponseDto(
            token = tokenService.createToken(user),
            userId = user.id
        )
    }

    @PostMapping("/register")
    fun register(@RequestBody payload: RegisterDto): LoginResponseDto {
        if (userService.existsByName(payload.name)) {
            throw ApiException(400, "Name already exists")
        }

        val user = User(
            name = payload.name,
            password = hashService.hashBcrypt(payload.password),
        )

        val savedUser = userService.save(user)

        folderService.createDefaultFoldersForUser(user)

        return LoginResponseDto(
            token = tokenService.createToken(savedUser),
            userId = user.id
        )
    }
}
```

Контролер документів, DocumentController.kt

```
package com.docfolio.controller

import com.docfolio.dto.DocumentRequestDto
import com.docfolio.dto.DocumentResponseDto
import com.docfolio.dto.DocumentSearchRequest
import com.docfolio.model.Document
```

```
import com.docfolio.model.Template
import com.docfolio.service.DocumentService
import org.springframework.data.domain.Pageable
import org.springframework.web.bind.annotation.*
import java.util.UUID

@RestController
@RequestMapping("/api/documents")
class DocumentController (
    private val documentService: DocumentService
) {
    @GetMapping("/{folderId}")
    fun fetchDocuments(@PathVariable folderId: UUID, searchRequest: DocumentSearchRequest):
List<DocumentResponseDto> {
        return documentService.fetch(folderId, searchRequest)
    }

    @GetMapping("/byId/{id}")
    fun fetchDocumentById(@PathVariable id: UUID): DocumentResponseDto {
        return documentService.fetchById(id)
    }

    @PostMapping("/create")
    fun createDocument(@RequestBody document: DocumentRequestDto): DocumentResponseDto
{
        return documentService.create(document)
    }

    @PostMapping("/edit/{id}")
    fun editDocument(@PathVariable id: UUID, @RequestBody request: DocumentRequestDto):
DocumentResponseDto {
        return documentService.edit(id, request)
    }

    @DeleteMapping("/delete/{id}")
```

```
    fun deleteDocument(@PathVariable id: UUID) {
        documentService.delete(id)
    }

    @PostMapping("/replace/{folderId}")
    fun replaceDocument(@PathVariable folderId: UUID, @RequestBody documentId: UUID) {
        documentService.replace(folderId, documentId)
    }
}
```

Контролер папок, FolderController.kt

```
package com.docfolio.controller

import com.docfolio.dto.FolderRequestDto
import com.docfolio.dto.FolderResponseDto
import com.docfolio.model.Folder
import com.docfolio.service.FolderService
import org.springframework.web.bind.annotation.*
import java.util.UUID

@RestController
@RequestMapping("/api/folders")
class FolderController(
    private val folderService: FolderService
) {
    @GetMapping("/{userId}")
    fun fetchFolders(@PathVariable userId: UUID): List<FolderResponseDto> {
        return folderService.fetchFolders(userId)
    }

    @PostMapping("/create")
    fun createFolder(@RequestBody request: FolderRequestDto): FolderResponseDto {
        return folderService.create(request)
    }
```

```
    @PostMapping("/delete/{id}")
    fun deleteFolder(@PathVariable id: UUID) {
        return folderService.delete(id)
    }

    @PostMapping("/edit/{id}")
    fun editFolder(@PathVariable id: UUID, @RequestBody request: String): FolderResponseDto {
        return folderService.edit(id, request)
    }
}
```

Контролер шаблонів, TemplateController.kt

```
package com.docfolio.controller

import com.docfolio.dto.TemplateResponseDto
import com.docfolio.dto.TemplateSearchRequest
import com.docfolio.model.Template
import com.docfolio.service.TemplateService
import org.springframework.web.bind.annotation.GetMapping
import org.springframework.web.bind.annotation.PathVariable
import org.springframework.web.bind.annotation.PostMapping
import org.springframework.web.bind.annotation.RequestBody
import org.springframework.web.bind.annotation.RequestMapping
import org.springframework.web.bind.annotation.RestController
import java.util.*

@RestController
@RequestMapping("/api/templates")
class TemplateController(
    private val templateService: TemplateService
) {
    @GetMapping
    fun fetchTemplates(searchRequest: TemplateSearchRequest): List<Template> {
        return templateService.fetchTemplate(searchRequest)
    }
```

```
    @GetMapping("/{id}")
    fun fetchTemplateById(@PathVariable id: UUID): Template {
        return templateService.fetchById(id)
    }

    @PostMapping("/create")
    fun createTemplate(@RequestBody template: Template): TemplateResponseDto {
        return templateService.create(template)
    }
}
```

Сервіс документів з описаними функціями роботи з документами, DocumentService

```
package com.docfolio.service

import com.docfolio.dto.DocumentRequestDto
import com.docfolio.dto.DocumentResponseDto
import com.docfolio.dto.DocumentSearchRequest
import com.docfolio.model.Document
import com.docfolio.repository.DocumentRepository
import com.docfolio.repository.FolderRepository
import org.springframework.stereotype.Service
import java.util.UUID

@Service
class DocumentService(
    private val documentRepository: DocumentRepository,
    private val folderRepository: FolderRepository
) {
    fun fetch(folderId: UUID, searchRequest: DocumentSearchRequest):
List<DocumentResponseDto> {
        val documents = documentRepository.findAllByFolderId(folderId)
        val filteredDocuments = documents.filter { it -> it.title.contains(searchRequest.searchQuery,
ignoreCase = true) }
        val sorterDocuments = when( searchRequest.order ) {
```

```
            "asc" -> filteredDocuments.sortedBy { it.title }
            "desc" -> filteredDocuments.sortedByDescending { it.title }
            else -> filteredDocuments
        }
        return sorterDocuments.map { DocumentResponseDto(it.id!!, it.folderId, it.templateId, it.title,
it.modifiedAt, it.values) }
    }

    fun fetchById(id: UUID): DocumentResponseDto {
        val document = documentRepository.findById(id).get()
        return DocumentResponseDto(document.id!!, document.folderId, document.templateId,
document.title, document.modifiedAt, document.values)
    }

    fun create(document: DocumentRequestDto): DocumentResponseDto {
        val folder = folderRepository.findById(document.folderId).get()
        val newDocument = Document(title = document.title, folderId = document.folderId,
templateId = document.templateId, type = document.templateType, values = document.values)
        val savedDocument = documentRepository.save(newDocument)
        folder.documents!!.add(newDocument)
        val updatedFolder = folder.copy(documents = folder.documents)
        folderRepository.save(updatedFolder)
        return DocumentResponseDto(savedDocument.id!!, savedDocument.folderId,
savedDocument.templateId, savedDocument.title, savedDocument.modifiedAt,
savedDocument.values)
    }

    fun edit(id: UUID, request: DocumentRequestDto): DocumentResponseDto {
        val document = documentRepository.findById(id).get()
        val folder = folderRepository.findById(request.folderId).get()
        val updatedDocument = document.copy(values = request.values, modifiedAt =
request.modifiedAt)
        documentRepository.save(updatedDocument)
        val allDocuments = documentRepository.findAllByFolderId(request.folderId)
        val updatedFolder = folder.copy(documents = allDocuments.toMutableList())
```

```
        folderRepository.save(updatedFolder)
        return DocumentResponseDto(updatedDocument.id!!, updatedDocument.folderId,
updatedDocument.templateId, updatedDocument.title, updatedDocument.modifiedAt,
updatedDocument.values)
    }

    fun delete(id: UUID) {
        val document = documentRepository.findById(id).get()
        val folder = folderRepository.findById(document.folderId).get()
        folder.documents!!.remove(document)
        val updatedFolder = folder.copy(documents = folder.documents)
        folderRepository.save(updatedFolder)
        documentRepository.delete(document)
    }
}
```

Сервіс папок з описаними функціями роботи з папками, FolderService

```
package com.docfolio.service

import com.docfolio.dto.FolderRequestDto
import com.docfolio.dto.FolderResponseDto
import com.docfolio.model.Folder
import com.docfolio.model.User
import com.docfolio.repository.FolderRepository
import com.docfolio.repository.UserRepository
import org.bson.types.ObjectId
import org.springframework.stereotype.Service
import java.util.UUID

@Service
class FolderService (
    private val folderRepository: FolderRepository,
    private val userRepository: UserRepository
) {
    fun fetchFolders(userId: UUID): List<FolderResponseDto> {
```

```
        val folders = folderRepository.findAllByUserId(userId)
        return folders.map { FolderResponseDto(it.id, it.name, it.documents, it.isDefaultFolder) }
    }

    fun create(folder: FolderRequestDto): FolderResponseDto {
        val user = userRepository.findById(folder.userId).get()
        val newFolder = Folder(name = folder.name, user = user)
        folderRepository.save(newFolder)
        return FolderResponseDto(newFolder.id, newFolder.name, newFolder.documents,
newFolder.isDefaultFolder)
    }

    fun delete(id: UUID) {
        return folderRepository.deleteById(id)
    }

    fun edit(id: UUID, request: String): FolderResponseDto {
        val folder = folderRepository.findById(id).get()
        val updatedFolder = folder.copy(name = request)
        folderRepository.save(updatedFolder)
        return FolderResponseDto(updatedFolder.id, updatedFolder.name, updatedFolder.documents,
updatedFolder.isDefaultFolder)
    }

    fun createDefaultFoldersForUser(user: User) {
        val folders = listOf("My Documents", "To Sign", "Archive")
        val request = folders.map { FolderRequestDto(it, user.id) }
        request.forEach{ folder ->
            run {
                val user = userRepository.findById(folder.userId).get()
                val newFolder = Folder(name = folder.name, user = user, isDefaultFolder = true)
                folderRepository.save(newFolder)
            }
        }    }}
```

Сервіс шаблонів з описаними функціями роботи з шаблонами, TemplateService

```
package com.docfolio.service

import com.docfolio.dto.TemplateResponseDto
import com.docfolio.dto.TemplateSearchRequest
import com.docfolio.model.Template
import com.docfolio.repository.TemplateRepository
import org.springframework.stereotype.Service
import java.util.UUID

@Service
class TemplateService (
    private val templateRepository: TemplateRepository
) {
    fun fetchTemplate(searchRequest: TemplateSearchRequest): List<Template> {
        val templates = templateRepository.findAll()
        val filteredTemplates = templates.filter { it -> it.name.contains(searchRequest.searchQuery,
ignoreCase = true) }
        val sorterTemplates = when( searchRequest.order ) {
            "asc" -> filteredTemplates.sortedBy { it.name }
            "desc" -> filteredTemplates.sortedByDescending { it.name }
            else -> filteredTemplates
        }        return sorterTemplates
    }

    fun fetchById(id: UUID): Template {
        return templateRepository.findById(id).get()
    }

    fun create(template: Template): TemplateResponseDto {
        val savedTemplate = templateRepository.save(template)
        return TemplateResponseDto(savedTemplate.id, savedTemplate.type, savedTemplate.name,
savedTemplate.fields)
    }}
```

Сервіс токена, використовується для авторизації користувача, TokenService

```
package com.docfolio.service

import com.docfolio.model.User
import org.springframework.security.oauth2.jwt.*
import org.springframework.stereotype.Service
import java.time.Instant
import java.time.temporal.ChronoUnit
import java.util.*

@Service
class TokenService(
    private val jwtDecoder: JwtDecoder,
    private val jwtEncoder: JwtEncoder,
    private val userService: UserService,
) {
    fun createToken(user: User): String {
        val jwsHeader = JwsHeader.with { "HS256" }.build()
        val claims = JwtClaimsSet.builder()
            .issuedAt(Instant.now())
            .expiresAt(Instant.now().plus(30L, ChronoUnit.DAYS))
            .subject(user.name)
            .claim("userId", user.id)
            .build()
        return jwtEncoder.encode(JwtEncoderParameters.from(jwsHeader, claims)).tokenValue
    }

    fun parseToken(token: String): User? {
        return try {
            val jwt = jwtDecoder.decode(token)
            val userId = UUID.fromString(jwt.claims["userId"].toString())
            userService.findById(userId)
        } catch (e: Exception) {
            null
        }    }}
```

Сервіс користувача з описаними функціями для роботи з користувачем, UserService

```
package com.docfolio.service

import com.docfolio.model.Folder
import com.docfolio.model.User
import com.docfolio.repository.UserRepository
import org.springframework.data.repository.findByIdOrNull
import org.springframework.stereotype.Service
import java.util.UUID

@Service
class UserService(
    private val userRepository: UserRepository,
) {
    fun findById(id: UUID): User? {
        return userRepository.findByIdOrNull(id)
    }
    fun findByName(name: String): User? {
        return userRepository.findByName(name)
    }

    fun existsByName(name: String): Boolean {
        return userRepository.existsByName(name)
    }

    fun findAll(): List<User> {
        return userRepository.findAll()
    }
    fun save(user: User): User {
        return userRepository.save(user)
    }}
```