Міністерство освіти і науки України
Одеський національний технологічний університет
Кафедра комп'ютерної інженерії

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО КВАЛІФІКАЦІЙНОЇ РОБОТИ

на тему *Розробка інтернет-застосунку обробки документів*
(назва кваліфікаційної роботи згідно наказу ОНТУ)

Здобувача *Сєчіна Ю. Д.*
(прізвище, ініціали)
2 курсу *543б* групи
Керівники: *ст. викл. Сіренко О. І.*
(посада, прізвище та ініціали)
д.т.н., проф. Артеменко С. В.
(посада, прізвище та ініціали)
Консультанти:
д.е.н., проф. Басюркіна Н. Й.
(посада, прізвище та ініціали)
к.т.н., ст. викл. Нєнов О. Л.
(посада, прізвище та ініціали)

Кваліфікаційна робота допускається до захисту

Рішення кафедри від *10.06* 2023 р., протокол № 8

Завідувач кафедри комп. інженерії _____ Сергій АРТЕМЕНКО
(назва кафедри) (підпис) (Ім'я ПРІЗВИЩЕ)

Одеса – 2023 рік

ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНОЛОГІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерної інженерії, програмування та кіберзахисту*
Кафедра *комп'ютерної інженерії*
Ступінь вищої освіти *бакалавр*
Спеціальність *123 «Комп'ютерна інженерія»*
Освітня програма *Розробка ігор та інтерактивних медіа у віртуальній реальності*

ЗАТВЕРДЖУЮ
Зав. кафедри *комп'ютерної інженерії*
Сергій АРТЕМЕНКО
«*10*» *серпня* 2022 року

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА

Сєчіна Юрія Дмитровича

1. Тема роботи *Розробка інтернет-застосунку обробки документів*

Затверджена наказом університету від «10» *серпня* 2022 р., наказ № *440-03*

2. Термін здачі здобувачем закінченої роботи *5 червня 2023 р.*

3. Вихідні дані роботи

1. Текстовий процесор Microsoft Word 2. Java-компілятор JDK версії 17 або вище 3. Інтегроване середовище розробки IntelliJ IDEA 4. Система керування базами даних PostgreSQL 5. Редактор MS Power Point

4. Перелік питань, які потрібно розробити

1. Вступ. 2. Дослідження предметної області. 3. Проектування алгоритмів роботи. 4. Впровадження алгоритмів. 5. Тестування. 6. Економічні розрахунки. 7. Охорона праці.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Слайд 1. Вступ. Слайд 2. Аналіз аналогів. Слайд 3. Вимоги до функціональності проекту. Слайд 4. Загальна схема роботи застосунку. Слайд 5. Детальна схема роботи застосунку. Слайд 6. Схема бази даних. Слайд 7. Обрані технології. Слайд 8. Діаграма потоків даних. Слайд 9. Архітектура застосунку. Слайд 10. Структура сайту. Слайд 11. Методи тестування. Слайд 12. Економічні показники. Слайд 13. Висновки.

6. Консультанти по роботі, із зазначенням розділів роботи, що стосуються їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Економіка	*Басюркіна Н. Й., д.е.н., проф.*		
Охорона праці	*Нєнов О. Л., к.т.н., ст. викл.*		
Нормоконтроль	*Сіренко О. І., ст. викл.*		

7. Дата видачі завдання *25.02.2023*

Керівники _____ *Олександр СІРЕНКО*

_____ *Сергій АРТЕМЕНКО*

Завдання прийняв до виконання _____ *Юрій СЄЧІН*

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1.	Написання вступу та анотації	30.02.2023	
2.	Написання першого розділу	10.03.2023	
3.	Написання другого розділу	25.03.2023	
4.	Написання третього розділу	05.04.2023	
5.	Написання четвертого розділу	10.04.2023	
6.	Написання п’ятого розділу	20.04.2023	
7.	Написання висновків	03.05.2023	
8.	Підготовка демонстраційних матеріалів	05.05.2023	
9.	Оформлення пояснювальної записки	15.05.2023	
10.	Подання дипломної роботи на затвердження	01.06.2023	

Керівники роботи _____ *Олександр СІРЕНКО*

_____ *Сергій АРТЕМЕНКО*

Несу відповідальність за ідентичність електронного та друкованого варіантів кваліфікаційної роботи, даю згоду на обробку персональних даних та не заперечую проти розміщення кваліфікаційної роботи на офіційних web-ресурсах ОНТУ.

Підтверджую, що в кваліфікаційній роботі відсутні порушення норм академічної доброчесності.

Здобувач – дипломник _____ *Юрій СЄЧІН*

АНОТАЦІЯ

Дипломна робота викладена на 78 сторінках, вона містить 5 розділів, 21 ілюстрацій, 8 таблиць, 16 джерел в переліку посилань.

Об'єктом дослідження є процес розробки та впровадження веб-шаблонізатора.

Предмет дипломного проектування – розробка веб-шаблонізатора на основі використання формату *DOCX*.

Основна мета проекту – створити програмне забезпечення, що дозволить підставляти значення з бази даних у відповідні місця шаблону *DOCX*, тим самим автоматизуючи процес створення документів на основі шаблонів.

Результатом дослідження є створення функціональності, яка дозволить зручно та швидко створювати документи на основі шаблонів.

Для досягнення поставленої мети необхідно провести аналіз існуючих рішень, на основі аналізу сформувати функціональні вимоги до проекту, провести вибір та обгрунтування технологій, розробити архітектуру програмного забезпечення та впровадити розроблені алгоритми. Також важливим етапом є перевірка працездатності проведенням тестування отриманого рішення.

У рамках роботи була розроблена програма, що дозволяє користувачам швидко та зручно створювати нові документи на основі шаблонів, використовуючи сайт в якості інтерфейсу.

Ключові слова: документ, шаблон, *DOCX*, програмне забезпечення, сайт, веб-застосунок.

ABSTRACT

The thesis consists of 78 pages and includes 5 chapters, 21 illustrations, 8 tables, and 16 references in the bibliography.

The object of research is the development and implementation process of a web templating engine.

The subject of the diploma project is the development of a web templating engine based on the use of the DOCX format.

The main goal of the project is to create software that allows inserting values from a database into the corresponding places of a DOCX template, thus automating the process of document creation based on templates.

The result of the research is the creation of functionality that enables convenient and fast document generation based on templates.

To achieve the set goal, it is necessary to conduct an analysis of existing solutions, based on the analysis, formulate functional requirements for the project, select and justify technologies, develop the architecture of the software, and implement the designed algorithms. Testing the obtained solution for functionality is also an important stage.

As part of the work, a program was developed that allows users to quickly and conveniently create new documents based on templates, using a website as an interface.

Keywords: *document, template, DOCX, software, website, web application.*

ЗМІСТ

стор.

ВСТУП

Веб-застосунок – це програмне забезпечення, яке розробляється для виконання функцій через Інтернет. Вони використовуються для різних цілей, наприклад, для обробки та зберігання даних, для комунікації з користувачами, для організації та ведення бізнесу, для надання онлайн-сервісів, для покращення ефективності роботи та багато іншого. В залежності від цілей, можна виділити такі основні види веб-застосунків:

1. Сайти-візитки: малі сайти, які використовуються для представлення компанії, продукту чи послуги.
2. Електронні магазини: веб-застосунки, що надають можливість покупцям здійснювати покупки онлайн.
3. Соціальні мережі: веб-застосунки, що дозволяють користувачам спілкуватися та обмінюватися інформацією.
4. Веб-прикладання: прикладне програмне забезпечення, яке використовується для виконання конкретних завдань, наприклад, для обробки даних, і використовує сайт як інтерфейс для взаємодії з користувачем.
5. Веб-сервіси: веб-застосунки, що надають можливість взаємодії між різними системами та програмами через Інтернет.

До основних завдань у галузі обробки документів можна віднести наступне:

1. Обробка тексту: завдання полягає у пошуку, обробці та зміні текстового вмісту документа, наприклад, видалення певних слів, заміна фраз тощо.
2. Імпорт даних: завдання імпорту даних з інших документів або форматів, таких як *CSV* або *Excel*-файли.
3. Експорт документів: завдання полягає у перетворенні документів на інші формати, такі як *PDF*, *HTML*, *XML* і т.д.
4. Розпізнавання тексту: ця задача полягає в автоматичному розпізнаванні тексту на зображенні і його перетворенні в текстовий формат.
5. Конвертація мов: ця задача включає в себе переклад тексту документа на іншу мову або зміну мови в документі.

Однією з ключових переваг веб-застосунку для обробки документів є можливість його використання в різних галузях, де необхідно створювати багато документів на основі шаблонів. Зокрема, це може бути корисно для фінансових установ, юридичних фірм, медичних установ, учбових закладів та інших організацій.

Однак, розробка такого веб-застосунку також має свої виклики. Необхідно враховувати особливості роботи з документами у форматі *DOCX*, забезпечувати стабільну роботу з великою кількістю даних, а також забезпечувати безпеку інформації. Усі ці виклики можна успішно вирішити за допомогою правильної розробки та тестування програмного забезпечення. Після завершення розробки та тестування веб-застосунку можна очікувати покращення продуктивності роботи з документами, а також зниження часу на їх створення.

Наочна область проектування полягає у вивченні потреб користувачів, розробці ергономічного інтерфейсу користувача, створенні функціональності для роботи з шаблонами документів, розробці системи безпеки та забезпечення швидкості та стабільності роботи програми. Актуальність вирішуваного завдання полягає у необхідності створення ефективного та швидкого інструменту, що дозволить значно скоротити час на створення документів за допомогою шаблонів. Використання даного засобу може бути особливо корисним для організацій, що часто працюють зі стандартними документами (наприклад, звіти, договори тощо).

Об'єктом дослідження є процес розробки та впровадження веб-шаблонізатора. Предмет дипломного проектування – розробка веб-шаблонізатора на основі використання формату *DOCX*. Основна мета проекту – створити програмне забезпечення, що дозволить підставляти значення з бази даних у відповідні місця шаблону *DOCX*, тим самим автоматизуючи процес створення документів на основі шаблонів. Результатом дослідження є створення функціональності, яка дозволить зручно та швидко створювати документи на основі шаблонів. Для досягнення поставленої мети необхідно розробити функціональні вимоги, провести аналіз та вибір технологій, розробити архітектуру програмного забезпечення, реалізувати необхідний функціонал та перевірити працездатність програми. Також важливим етапом є проведення тестування отриманого рішення, а також документування розробленого програмного забезпечення.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Переваги використання шаблонізаторів документів в сучасному світі

Шаблонізатор документів – це програмне забезпечення, яке дозволяє створювати документи на основі попередньо заданих шаблонів. Шаблонізатори документів можуть бути корисними для багатьох видів документів, таких як звіти, рахунки, контракти, електронні листи та інші, і дозволяють ефективно створювати великі обсяги документів, зменшуючи час та зусилля, які потрібно витратити на їх створення.

Основним компонентом шаблонізатора документів є шаблон, який містить загальну структуру та формат документа. Шаблон може містити певні рядки тексту, поля, таблиці, зображення та інші елементи, які можуть бути заповнені під час створення документа. Крім того, шаблон може містити вбудовані скрипти або код, який дозволяє змінювати контент документа в залежності від різних умов та параметрів.

Для того, щоб створити документ за допомогою шаблонізатора, потрібно ввести відповідні дані у форму або файл. Дані можуть бути збережені в різних форматах, таких як *CSV, XML* та *JSON*, або у базі даних. Після того, як дані будуть введені, шаблонізатор документів використовує вбудовані правила та скрипти для генерації документа на основі шаблону та введених даних.

Одним з найбільш важливих переваг шаблонізаторів документів є те, що вони дозволяють зберігати шаблони та введені дані в різних форматах, що дозволяє легко створювати документи великої кількості форматів. Наприклад, якщо підприємство має різні формати рахунків для різних клієнтів, шаблонізатор може дозволити легко створювати всі ці формати на основі одного загального шаблону, що зменшує час та зусилля, необхідні для створення документів. Крім того, шаблонізатори документів дозволяють забезпечити однакову структуру та форматування документів для всіх клієнтів, що забезпечує однаковість та професійний вигляд. Також шаблонізатори документів забезпечують можливість зберігати шаблони та дані в базі даних або в хмарних сервісах, що дозволяє легко переглядати та оновлювати шаблони та дані з будь-якого місця з доступом до Інтернету.

Іншою важливою перевагою шаблонізаторів документів є зменшення ризику помилок під час створення документів. Шаблонізатори дозволяють використовувати вже налаштовані та перевірені шаблони, що зменшує шанси на помилки в форматуванні та орфографії документу. Крім того, шаблонізатори документів можуть дозволяти автоматично заповнювати деякі поля, що дозволяє додатково зменшити час та зусилля, які потрібно витратити на створення документа.

Шаблонізатори документів також можуть бути дуже корисними для учбових закладів, оскільки вони дозволяють зберігати та використовувати шаблони для створення звітів, що значно полегшує роботу, як працівників та викладачів, так і студентів та абітурієнтів. Крім того, шаблонізатори документів можуть допомогти відповідати вимогам регулюючих органів щодо збереження та форматування документів.

Узагальнюючи, шаблонізатори документів є корисними інструментами для створення документів, які дозволяють зберігати та використовувати шаблони для швидкого та ефективного створення великої кількості однотипних документів.

1.2 Веб-застосунки як сучасний спосіб надання сервісів

Веб-застосунки або інтернет-додатки – це програми, призначені для роботи у веб-браузерах. Вони доступні через Інтернет і надають користувачам можливість взаємодіяти з програмою через веб-інтерфейс. Веб-програми можна використовувати для різноманітних цілей, включаючи соціальні мережі, електронну комерцію, онлайн-банкінг, електронну пошту тощо.

Принцип роботи веб-додатків можна розбити на кілька ключових компонентів. До них належать клієнт, сервер і сама веб-програма.

Клієнтом є веб-браузер користувача, наприклад *Google Chrome*, *Firefox* або *Safari*. Коли користувач вводить адресу веб-застосунку, браузер надсилає запит на сервер. Цей запит містить інформацію про веб-програму, до якої користувач хоче отримати доступ.

Сервер – це комп’ютер, на якому розміщено веб-додаток. Коли сервер отримує запит користувача, він отримує необхідні файли та дані для веб-додатку та надсилає їх назад у браузер користувача. Цей процес відомий як цикл запит-

відповідь. Кожен сервер має свою веб-адресу, або *URL*. *URL* – це рядок тексту, який ідентифікує унікальний ресурс, який може бути доступним в Інтернеті.

Сам веб-додаток складається з комбінації коду на стороні клієнта та на стороні сервера. Клієнтський код написаний на мові розмітки *HTML*, мові стилю *CSS* та мові програмування *JavaScript*, і виконується у браузері користувача. Код на стороні сервера може бути написаний на будь-якій сучасній мові програмування, включаючи *Python, Java* або також *JavaScript*, і виконується на сервері.

Як зазначено у статті [1], у комп'ютерних мережах зв'язок між клієнтом і сервером зазвичай здійснюється за допомогою протоколу, який називається клієнт-серверною моделлю. У цій моделі клієнтом є користувач або пристрій, який запитує послугу або інформацію, а сервером є комп'ютер або програма, яка надає запитувану послугу або інформацію.

Щоб встановити зв'язок між клієнтом і сервером, клієнт надсилає серверу повідомлення із запитом. Потім сервер отримує повідомлення запиту, обробляє його та надсилає клієнту відповідь. Клієнт отримує повідомлення-відповідь і обробляє його відповідним чином.

Існує кілька протоколів, які використовуються для зв'язку клієнт-сервер, зокрема *HTTP*, *FTP*, *SMTP*, *POP3* та *IMAP*. Ці протоколи визначають правила та формати для повідомлень, якими обмінюються клієнт і сервер, наприклад, структуру повідомлень запиту та відповіді, використовуване кодування та методи автентифікації та безпеки.

Зв'язок між клієнтом і сервером може бути як синхронним, так і асинхронним. У синхронному зв'язку клієнт чекає відповіді сервера, перш ніж продовжити, тоді як в асинхронному зв'язку клієнт може продовжувати виконання інших завдань, чекаючи відповіді сервера. Синхронний зв'язок часто використовується в програмах реального часу, таких як онлайн-ігри, тоді як асинхронний зв'язок використовується в таких програмах, як електронна пошта або передача файлів.

Загалом зв'язок клієнт-сервер є ключовим аспектом комп'ютерних мереж і дозволяє користувачам отримувати доступ і використовувати широкий спектр послуг та інформації, доступних в Інтернеті (рис. 1.1).

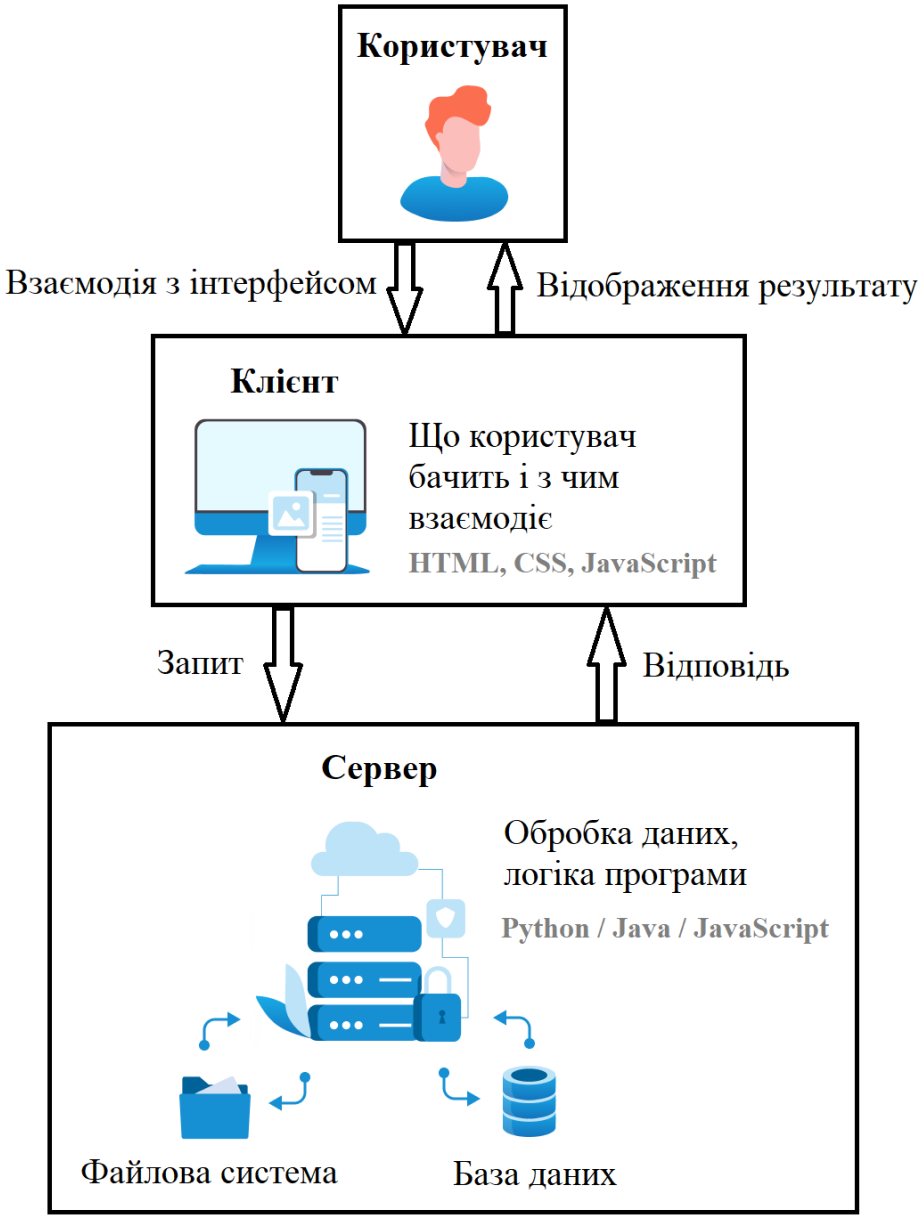


Рис. 1.1 – Клієнт-серверна архітектура

Коли користувач взаємодіє з веб-додатком, наприклад заповнює форму або натискає кнопку, код на стороні клієнта надсилає запит коду на стороні сервера. Код на стороні сервера обробляє запит і надсилає відповідь коду на стороні клієнта. Ця відповідь може містити нову веб-сторінку, дані для оновлення поточної веб-сторінки або іншу інформацію.

Веб-додатки можна розділити на два типи: статичні та динамічні. Статичні веб-програми складаються з фіксованих веб-сторінок, які доставляють у браузер користувача такими, якими вони є. Динамічні веб-програми, з іншого боку, складаються з веб-сторінок, які генеруються на льоту сервером на основі введення користувача або інших даних.

Веб-програми стають все більш популярними в сучасну цифрову епоху, оскільки вони надають користувачам зручний спосіб доступу та взаємодії з програмним забезпеченням з будь-якої точки світу. Вони пропонують підприємствам низку переваг, таких як економія коштів, масштабованість і покращений досвід роботи з користувачем.

Однією з ключових переваг веб-додатків є те, що вони не залежать від платформи. Це означає, що вони можуть працювати на будь-якому пристрої з веб-браузером і підключенням до Інтернету, включаючи настільні комп'ютери, ноутбуки, планшети та смартфони. Вони також не вимагають встановлення чи завантаження, що робить їх легкими у доступі та використанні, та позбавляє користувачів від встановлення оновлень.

Економія коштів також є значною перевагою веб-додатків. Вони позбавляють компаній від необхідності купувати та обслуговувати дороге обладнання та програмне забезпечення, оскільки веб-програма розміщена на сервері. Це також означає, що компанії можуть заощадити на персоналі та витратах на інфраструктуру.

Веб-програми також є масштабованими, що означає, що вони можуть працювати з великими обсягами користувачів і даних без значних проблем із продуктивністю. Це особливо важливо для підприємств, які відчувають швидке зростання або сезонні стрибки трафіку, оскільки вони можуть легко масштабувати свій веб-додаток відповідно до підвищеного попиту.

Ще однією важливою перевагою веб-додатків є покращення взаємодії з користувачем. Вони забезпечують зручний та інтерактивний інтерфейс, що дозволяє користувачам легко орієнтуватися в програмі та взаємодіяти з нею. Веб-програми також можна оптимізувати для різних пристроїв, забезпечуючи стабільну роботу користувачів на настільних ПК, ноутбуках, планшетах і смартфонах.

Безпека є критично важливим фактором для веб-додатків, оскільки вони часто стають мішенями хакерів і кіберзлочинців. Безпека веб-додатків включає такі заходи, як брандмауери, шифрування *SSL*, контроль доступу та сканування вразливостей. Регулярні перевірки безпеки та тестування можуть допомогти виявити та усунути будь-які вразливі місця у веб-програмі.

Підсумовуючи, веб-додатки є потужним інструментом для підприємств, які прагнуть покращити свою ефективність, масштабованість і взаємодію з

користувачем. Вони працюють, використовуючи комбінацію коду на стороні клієнта та на стороні сервера, і можуть бути розроблені за допомогою різноманітних мов програмування та фреймворків. Завдяки численним перевагам веб-програми вже стали невід'ємною частиною нашого повсякденного життя і стають все більш популярними в багатьох галузях.

1.3 Аналіз існуючих рішень у галузі генерації документів на основі шаблонів

Процес генерації документів на основі шаблонів протягом тривалого часу є важливою темою в галузі автоматизації документообігу. Існує багато існуючих рішень, спрямованих на автоматизацію процесу створення документів за допомогою шаблонів. Ці рішення варіюються від простих онлайн-інструментів до складних програмних рішень корпоративного рівня.

Одним з найпоширеніших рішень для створення документів на основі шаблонів є влаштовані функції в *Microsoft Word*. *Word* пропонує різноманітні шаблони, які користувачі можуть використовувати для швидкого та легкого створення документів. Також можна створювати власні шаблони та завантажувати їх у загальний простір, щоб інші користувачі могли їх знаходити та використовувати для створення подібних документів у майбутньому. В *Word* існує режим проектування, під час якого можна встановити різні елементи введення, такі як випадаючий список, радіокнопка, прапорець та вибір дати. Також існує тип "*field*", який встановлює плейсхолдер, і коли користувачу потрібно буде заповнити ці поля, *Word* буде запитувати по одному полю, яке значення потрібно вставити замість нього. Однак це рішення обмежене у своїх можливостях і може не підійти для більш складних завдань створення документів.

Ще одним популярним рішенням для створення документів є *Google Docs*. *Google Docs* – це хмарне рішення, яке дозволяє створювати документи та працювати над одним документом декільком користувачам одночасно. Він також пропонує різноманітні шаблони, які користувачі можуть використовувати для створення різних типів документів. Однак сервіс має обмежені можливості, коли справа стосується автоматизації. *Google Docs* надає можливість написати свої власні скрипти. Таким чином, наприклад, можна взяти вибірку даних у табличному

форматі, і створювати документи на базі шаблонів, підставляючи у потрібні місця дані конкретного запису таблиці. Для цього потрібно розробити певний алгоритм, та володіти певними навичками програмування, так як скрипти пишуться на мові програмування *JavaScript*.

Існують також спеціалізовані програмні рішення для автоматизації документообігу, які розроблені спеціально для створення документів на основі шаблонів. Деякі приклади такого програмного забезпечення включають *DocuSign* та *PandaDoc*. Ці рішення зазвичай дорожчі та орієнтовані на користувачів корпоративного рівня. Окрім загальних елементів введення, таких як випадаючі списки та прапорці, вони надають дуже широкий спектр функцій, що може викликати складнощі та заплутати нових користувачів. Прикладом таких функцій є можливості цифрового підпису та тригери, які можуть приховувати або відображати елементи, в залежності від певних умов.

Для створення документів на основі шаблонів можна використовувати мови програмування та спеціалізовані бібліотеки. Прикладами таких бібліотек є *Docx4j* та *Apache POI*. Вони підтримують формат документів *DOCX* і дозволяють читати, створювати та редагувати файли, починаючи від пошуку та заміни тексту, закінчуючи редагуванням стилів та колонтитулів. Звичайно, що для їх використання потрібно володіти навичками програмування на високому рівні, та вміти спроектувати алгоритм роботи та зручний інтерфейс.

В останні роки також відбулася поява рішень для створення документів на основі штучного інтелекту. Ці рішення використовують алгоритми машинного навчання для аналізу існуючих документів і створення нових документів на основі тієї самої структури та вмісту. Ці рішення все ще знаходяться на ранніх стадіях розробки, але обіцяють майбутнє у сфері автоматизації документообігу.

В таблиці 1.1 наведено порівняння найпоширеніших сервісів генерації документів.

Таблиця 1.1

Порівняння найпоширеніших сервісів генерації документів

Критерій	*Microsoft Word*	*Google Docs*	*DocuSign, PandaDoc*	Бібліотеки
Простота використання	Зручний інтерфейс зі знайомими інструментами та функціями	Зручний інтерфейс з інтуїтивно зрозумілими інструментами та функціями	Складний інтерфейс	Потребують навичок програмування
Швидкість	Залежить від швидкості комп’ютера та складності документа	Швидкий	Може працювати повільно через важку обробку документів	Залежить від швидкості комп’ютера та алгоритму
Ціна	Одноразова оплата або за моделлю підписки	Безкоштовно з обліковим записом Google	На основі підписки	Існують як безкоштовні, так і платні рішення
Співпраця (робота декількох людей над одним документом)	Обмежені функції співпраці з відстеженням змін і коментарів	Співпраця в реальному часі з кількома користувачами	Функції співпраці з обміном документами та підписанням	Тільки з додатковим алгоритмом поверх бібліотеки

Продовження таблиці 1.1

Критерій	*Microsoft Word*	*Google Docs*	*DocuSign, PandaDoc*	*Бібліотеки*
Інтеграція	Інтегрується з іншими продуктами *Microsoft*, такими як *Excel* і *PowerPoint*	Інтегрується з іншими продуктами *Google*, такими як Таблиці та Презентації	Інтегрується з іншими програмами, такими як *Salesforce* і *Dropbox*	Можна інтегрувати за допомогою додаткових бібліотек
Електронний підпис	Немає вбудованої функції електронного підпису	Немає вбудованої функції електронного підпису	Вбудована функція електронного підпису	Присутні методи для реалізації електронного підпису
Безпека	Зашифроване зберігання документів і захист паролем	Зашифроване зберігання документів і двофакторна аутентифіка-ція	Зашифроване зберігання документів і багатофак-торна автентифіка-ція	Захист паролем. Можливість додаткового захисту з додатковими бібліотеками

Загалом, існує багато рішень для створення документів на основі шаблонів. Вибір рішення залежить від конкретних вимог користувача та складності завдання формування документа. У той час як прості рішення, такі як *Microsoft Word* і *Google Docs*, підходять для базових завдань створення документів, для більш складних завдань можуть знадобитися спеціалізовані програмні рішення для автоматизації роботи з документами.

Також необхідно враховувати певні умови та вимоги кінцевих користувачів, саме тому і з’являється потреба у реалізації власної системи генерації документів для кожного випадку, що буде враховувати ці умови та вимоги.

1.4 Завдання автоматизації формування документів

Необхідність оперативного створення документів з можливістю налаштування параметрів заповнення шаблона, а також необхідність забезпечення роботи користувачів, які не мають спеціальної підготовки, потребує вирішення наступних основних завдань:

1. Організація інтерфейсу та середовища розробки документів, що полегшують підготовку та формування друкованих документів.
2. Забезпечення можливості створення нових форм документів та виконання налаштування параметрів користувачами, які не мають спеціальних навичок програмування та знань фізичної структури бази даних чи мови *SQL*.
3. Організація зберігання створених шаблонів та побудованих запитів для їх повторного застосування.
4. Забезпечення можливості точного позиціонування елементів документа при розробці шаблона, застосування засобів оформлення та форматування документа.
5. Підтримка роботи користувача з даними відповідно до термінології, прийнятої в конкретній предметній галузі.

У Вікіпедії [2] вказано, що процес автоматизованого формування документів, зазвичай, включає два основні етапи: перший – створення візуального макета документа (шаблона), другий – зіставлення елементів шаблона з таблицями бази даних та його заповнення (рис. 1.2).

Способи реалізації механізмів розробки шаблона та налаштування параметрів заповнення створеного шаблона визначають універсальність, гнучкість засобів та можливість оперативно створювати та модифікувати документи.

Ринок програмного забезпечення сьогодні пропонує велику кількість автоматизованих засобів формування документів. Проте аналіз програмних засобів та існуючих підходів дозволив виявити низку проблем та обмежень їх застосування.

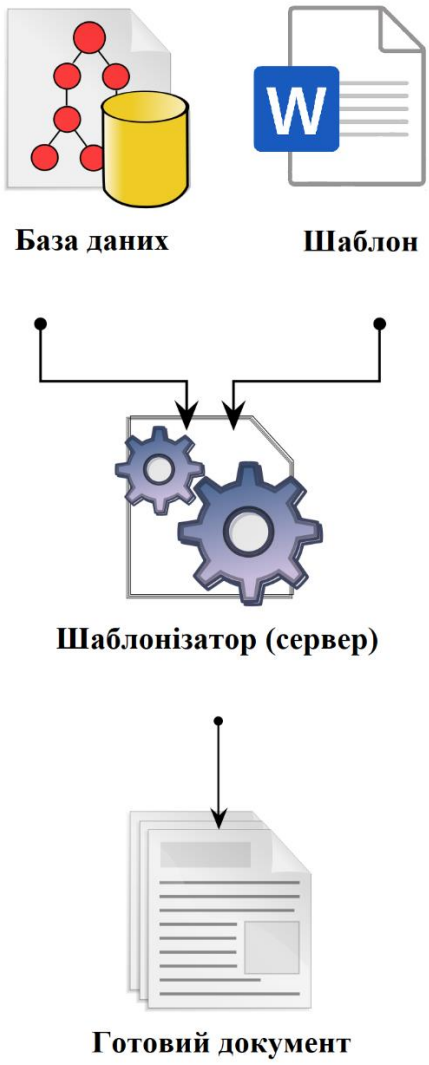


Рис. 1.2 – Загальна схема роботи шаблонізаторів

1.5 Функціональна модель оперативної генерації документів

В основі запропонованого рішення лежить оригінальний технологічний підхід, що поєднує засоби управління документами з інтерфейсом у вигляді сайту в мережі Інтернет та базою даних. Відповідно до запропонованого підходу розроблено функціональну модель процесу оперативної генерації документів.

Процес формування документів включає такі основні етапи:

1. Реєстрація (авторизація) користувача на сайті.
2. Заповнення користувачем інформації в особистому кабінеті, для її подальшого використання при підстановці даних.

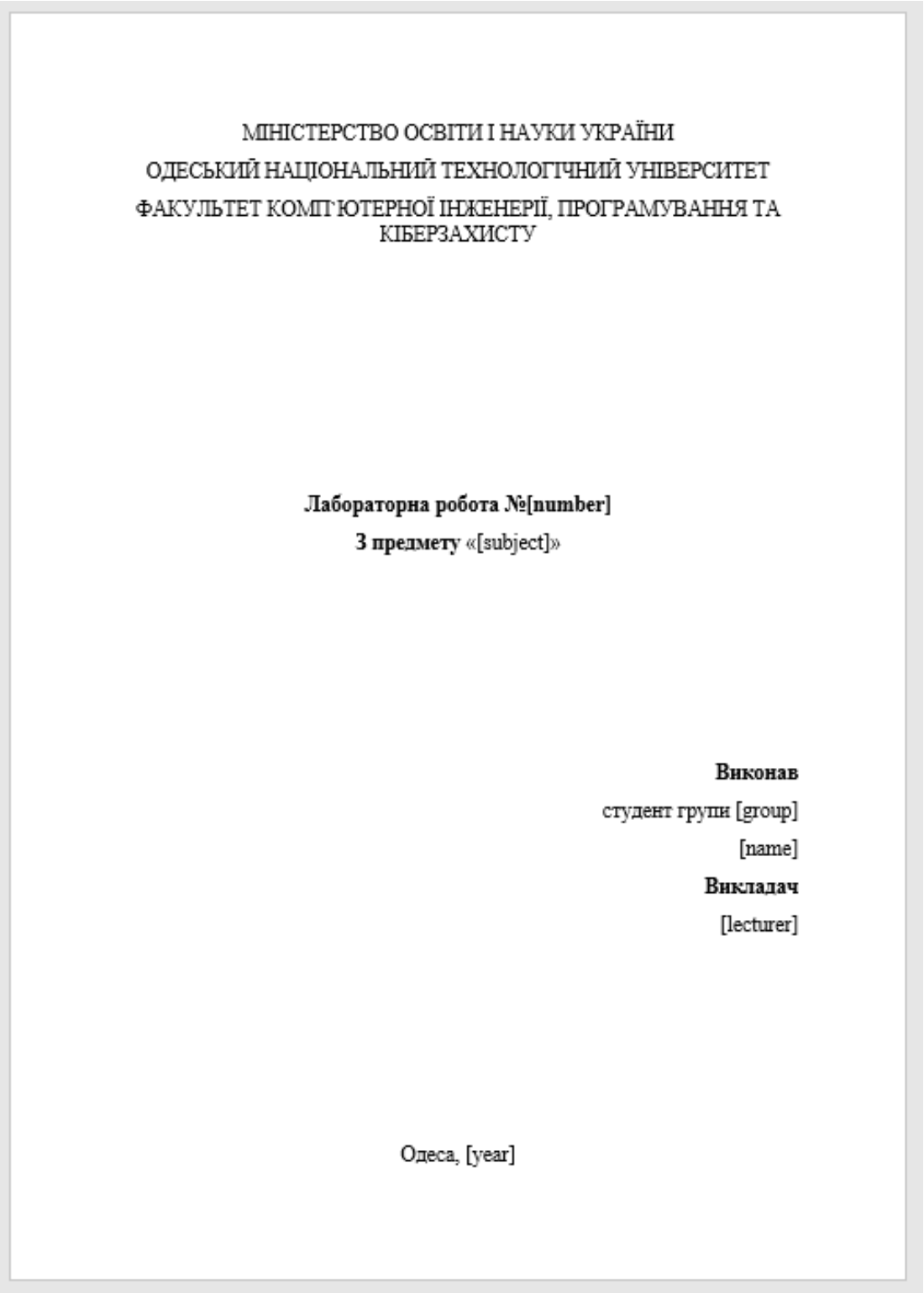
3. Створення шаблона документа формату *DOCX* – розробка візуального макета документа відповідно до вимог та методичних вказівок, а також аналіз вихідного шаблона із побудовою візуального уявлення його структури.
4. Пошук та вибір потрібного шаблона із списку усіх шаблонів на відповідній сторінці сайту.
5. Побудова запитів до бази даних – підготовка даних для налаштування та заповнення шаблона відповідно до його інформаційної структури.
6. Налаштування параметрів заповнення шаблона – зіставлення елементів шаблона з даними вихідних таблиць або даними, отриманими в результаті виконання запитів.
7. Генерація документа на основі розробленого шаблона – автоматичне заповнення шаблона даними з бази даних відповідно до заданих параметрів та умов заповнення.
8. Перетворення шаблона формат *DOCX* у потрібний формат та передача згенерованого документу на сторону клієнта.

1.6 Створення шаблона документа

На етапі створення шаблона користувач створює візуальний макет. Як оболонка розробки шаблона використовується гіпертекстовий редактор, наприклад, *Microsoft Word*, та шаблон є файлом з розширенням *DOCX*. Гіпертекстовий редактор *Microsoft Word* надає великі можливості щодо оформлення та позиціонування елементів документа.

Слабка структурованість шаблона, створеного як електронний документ, вимагає застосування спеціальних методів, дозволяють описати структуру документа. Для моделювання інформаційної структури пропонується спеціалізована мова розмітки. Текст шаблона поділяється на дві основні частини: статична, яка залишається незмінною в процесі роботи з шаблоном, та змінна, яка задається за допомогою символьної розмітки та динамічно формується за результатами запитів до бази даних. За допомогою розмітки визначаються ролі фрагментів документа в його загальній структурі, їх зміст та візуальне відображення. Найменування

елементів у розмітці відображає сенс інформації, що вноситься. Розмітка дає можливість описувати документи різної структури та змісту (рис. 1.3).


МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНОЛОГІЧНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ, ПРОГРАМУВАННЯ ТА КІБЕРЗАХИСТУ

Лабораторна робота №[number]
З предмету «[subject]»

Виконав
студент групи [group]
[name]
Викладач
[lecturer]

Одеса, [year]

Рис. 1.3 – Приклад готового шаблона

На наступному етапі виконується синтаксичний аналіз тексту шаблона. У процесі аналізу спочатку здійснюється пошук плейсхолдерів, тобто ключових фраз,

які потрібно буде згодом замінити вхідними даними. Після знаходження усіх плейсхолдерів, йде процес їх замінювання на значення, що зберігаються у базі даних, або інші значення, що не зберігаються у базі даних. Прикладом даних, що не зберігаються у базі даних, але використовуються як значення для заміни плейсхолдерів, можна назвати поточну дату, рік, та їх видозмінені значення, яке генеруються у коді, в момент, коли вони необхідні. Далі йде стадія перевірки користувачем процесу замінювання, під час якої можна буде змінити дані, що надійшли із бази даних, або вписати своє значення, якщо відповідний плейсхолдер не було знайдено ні в базі даних, ні серед значень, що генеруються безпосередньо у коді.

1.7 Вимоги до функціональності системи

Враховуючи недоліки та обмеження існуючих рішень, а також з урахуванням вимог, що висуваються до засобів генерації документів, стає актуальною реалізація свого веб-шаблонізатора, простого у використанні, що не потребує в процесі експлуатації навичок програмування. Для досягнення цієї мети потрібно реалізувати наступні кроки:

1. Розробити модуль, що буде відповідати за підстановку значень у потрібні місця шаблону.
2. Проаналізувати проектну область і спроектувати базу даних відповідно до вимог.
3. Реалізувати інтерфейс користувача, відповідні функції якого забезпечують:
    a) реєстрацію та авторизацію користувачів в системі;
    b) заповнення і оновлення форми з індивідуальними даними в особистому кабінеті;
    c) завантаження, перегляд та управління шаблонами;
    d) підстановку індивідуальних даних у обраний шаблон;
    e) передача згенерованого кінцевого документа користувачеві для подальшого завантаження на файлову систему останнього.

Висновки до першого розділу

Цей розділ присвячено аналітичним аспектам шаблонізаторів. Було розглянуто шаблонізатори в цілому та клієнт-серверна архітектура як зручний сучасний спосіб надання послуг та сервісів. Виявлено основні задачі шаблонізаторів та актуальність їх використання в сучасному світі.

Шаблонізатори є корисними інструментами для створення документів, які дозволяють зберігати та використовувати шаблони для швидкого та ефективного створення великої кількості однотипних документів, що дуже актуально у наш час.

Були проаналізовані та порівняні сучасні існуючі рішення генерації документів на базі шаблонів, виявлено їх переваги та недоліки. В результаті було прийнято рішення розробки власного веб-застосунку для забезпечення усіх умов та на основі аналізу сформовано вимоги до проекту.

РОЗДІЛ 2
ПРОЕКТУВАННЯ СИСТЕМИ

2.1 Проектування алгоритму роботи веб-застосунку

Процес проектування застосунку починається з розробки алгоритму, який визначає послідовність операцій і дій, необхідних для досягнення певних цілей. Алгоритм є основою для подальшої реалізації програмного забезпечення та визначає його логіку. Якісно розроблений алгоритм забезпечує ефективну та надійну роботу застосунку, допомагає вирішувати поставлені завдання та забезпечує задоволення потреб користувачів. Алгоритм взаємодії користувача з системою - це послідовність кроків або інструкцій, які визначають, як користувач повинен взаємодіяти з системою або програмним додатком.

Алгоритм взаємодії користувача з системою і подальшої роботи застосунку включає наступні кроки: користувач створює обліковий запис, або входить до існуючого. Якщо це новий акаунт, користувачу потрібно обрати в особистому кабінеті тип акаунту: студент або працівник – це впливає на ключові слова, які будуть пов’язані з акаунтом. Наприклад, окрім спільних полів, таких як ім’я та фамілія, з типом акаунту «працівник» пов’язані поля «посада» та «науковий ступінь», а з типом акаунту «студент» пов’язані такі поля, як «номер групи» та «спеціальність». Після цього слід заповнити форму з індивідуальними даними, щоб автоматизувати процес підстановки значень у шаблони. Ці дані можна заповнити або відредагувати пізніше.

Коли налаштування облікового запису закінчено, можна переходити до роботи з шаблонами. Робота починається зі сторінки «Шаблони», де відображено всі шаблони, що зберігаються на сервері. Біля кожного шаблона є відповідна кнопка для вибору відповідного шаблона та переходу на сторінку конкретного шаблона для подальшої роботи з ним. Також на сторінці «Шаблони» є кнопка для додавання нового шаблона на сервер.

Шаблон створюють користувачі на своєму комп’ютері у форматі *DOCX*, як звичайний документ, вписуючі у відповідні місця ключові слова у квадратних дужках. Для допомоги на відповідній сторінці на сайті буде приклад та повний список ключових слів. Після створення шаблону потрібно завантажити його на

сервер за допомогою відповідної кнопки, і цей шаблон зможуть побачити та використати усі користувачі.

На сторінці кожного конкретного шаблона є текст з описом шаблона, який можна змінити; кнопка для завантаження шаблона у початковому вигляді, на випадок, якщо потрібно буде змінити сам шаблон; кнопка видалення шаблона з сервера; та кнопка «Продовжити».

Кнопка «Продовжити» починає процес формування остаточного документу із обраного шаблона. Сервер зчитує усі плейсхолдери з обраного шаблону і знаходить потрібне відповідне значення для усіх розпізнаних ключових слів, будь то індивідуальні значення користувача із бази даних, або сьогоднішня дата. Після цього сервер генерує сторінку підтвердження, де присутні усі знайдені плейсхолдери, як розпізнані, так і не розпізнані, у порядку їх появи у шаблоні, а також поля для введення значень. Для розпізнаних плейсхолдерів ці поля будуть вже заповнені знайденими значеннями. Таким чином можна вписати дані для заміни не розпізнаних, унікальних саме для цього шаблона плейсхолерів, або перевірити, чи усі значення були знайдені та підставлені правильно, та відредагувати їх. В кінці сторінки присутні відповідні кнопки для генерації остаточного документа та його завантаження у форматі *DOCX* або *PDF*.

Описаний процес можна зручно ілюструвати у вигляді діаграми послідовності (рис. 2.1). Діаграма послідовності показує послідовність повідомлень, які передаються між об’єктами в рамках якогось конкретного виконання сценарію. Об'єкти зображаються у вигляді прямокутників з назвами, а повідомлення між ними у вигляді стрілок з написаними на них назвами. В даному випадку об’єктами є користувач (клієнт) та сервер (сайт).

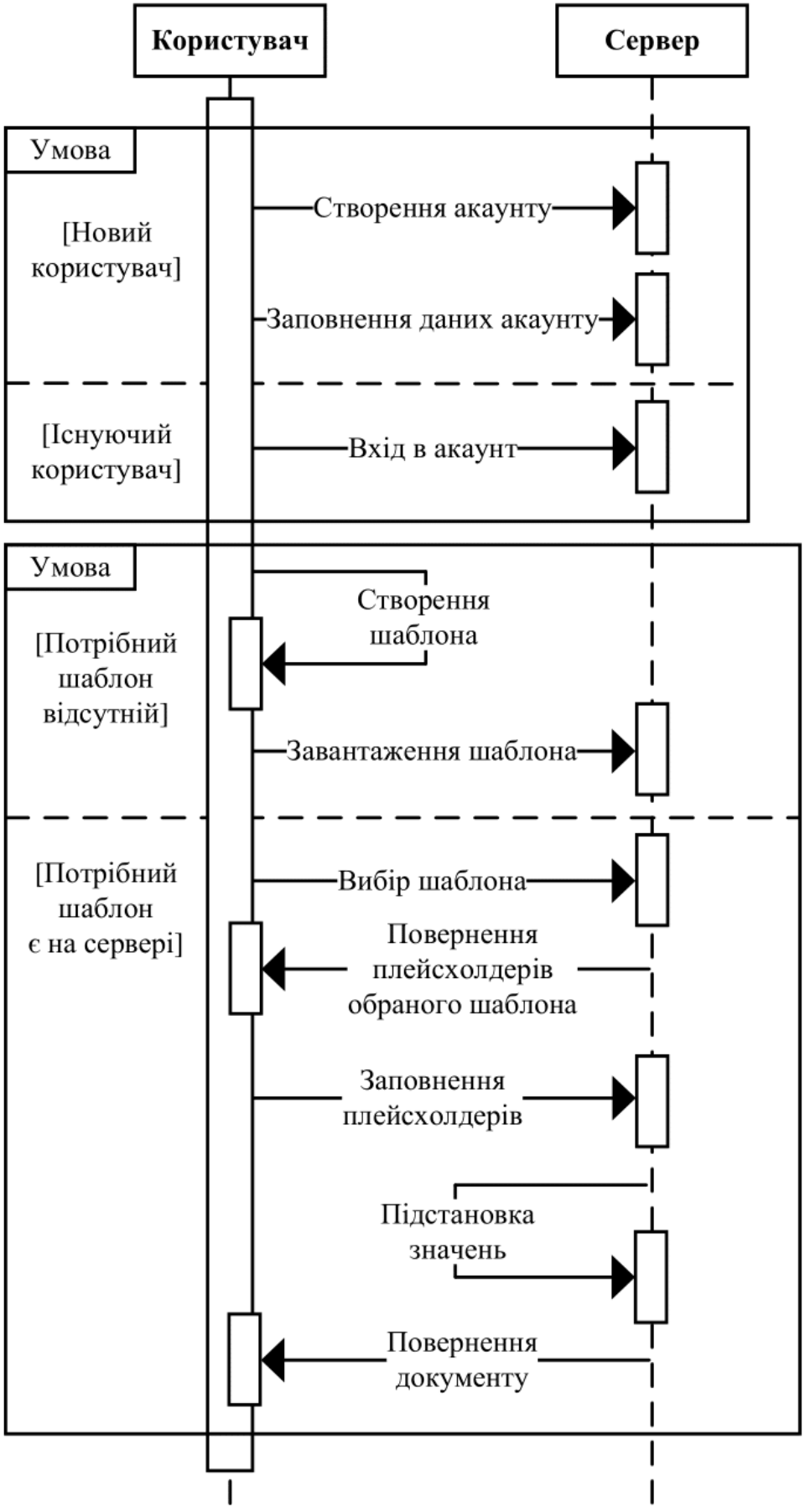


Рис. 2.1 – Діаграма послідовності взаємодії користувача з системою

2.2 Проектування архітектури веб-застосунку

Веб-застосунки зазвичай будуються за архітектурною моделлю *MVC*. *MVC* або *Model-View-Controller* (Модель-Представлення-Контролер) – це архітектурний підхід для розробки програмного забезпечення, який дозволяє розділити додаток на три основні компоненти: модель, представлення (вид) і контролер. Кожен з цих компонентів виконує свої функції і забезпечує зручний та ефективний процес розробки (рис. 2.2).

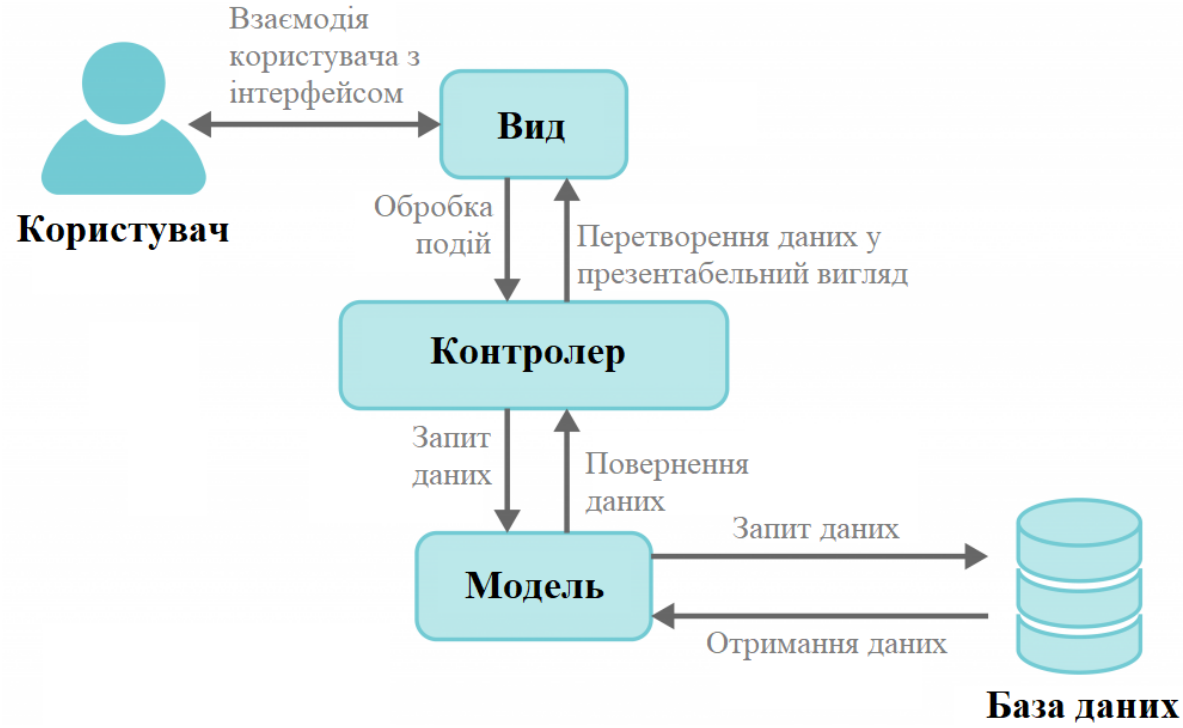


Рис. 2.2 – Схема роботи застосунку з архітектурою *MVC*

Модель представляє собою представлення даних в додатку. Це можуть бути дані з бази даних або файлової системи. Модель забезпечує доступ до даних та їх обробку. Вона повинна бути незалежною від інших компонентів додатку та забезпечувати можливість її повторного використання. В запланованому проекті в якості моделі використовується база даних.

Представлення, або вид, відповідає за відображення даних на екрані. Він містить у собі всі необхідні елементи інтерфейсу користувача, такі як кнопки, тексти, форми тощо. Вид може взаємодіяти з контролером та моделлю, але не повинен залежати від них. В даному випадку до компоненту представлення

належать *HTML* файли, файли стилів *CSS*, картинки для сторінок, а також шаблони *DOCX*.

Контролер служить посередником між моделлю та видом. Він приймає запити від користувача та відправляє їх до моделі для обробки. Після цього контролер забезпечує відображення результатів на відповідному виді. У Вікіпедії [3] зазначено, що *MVC* не має суворої реалізації, і відповідно немає загальноприйнятого визначення, де має розташовуватися бізнес-логіка, але в цьому випадку бізнес логіка буде знаходитись у контролері (рис. 2.3).

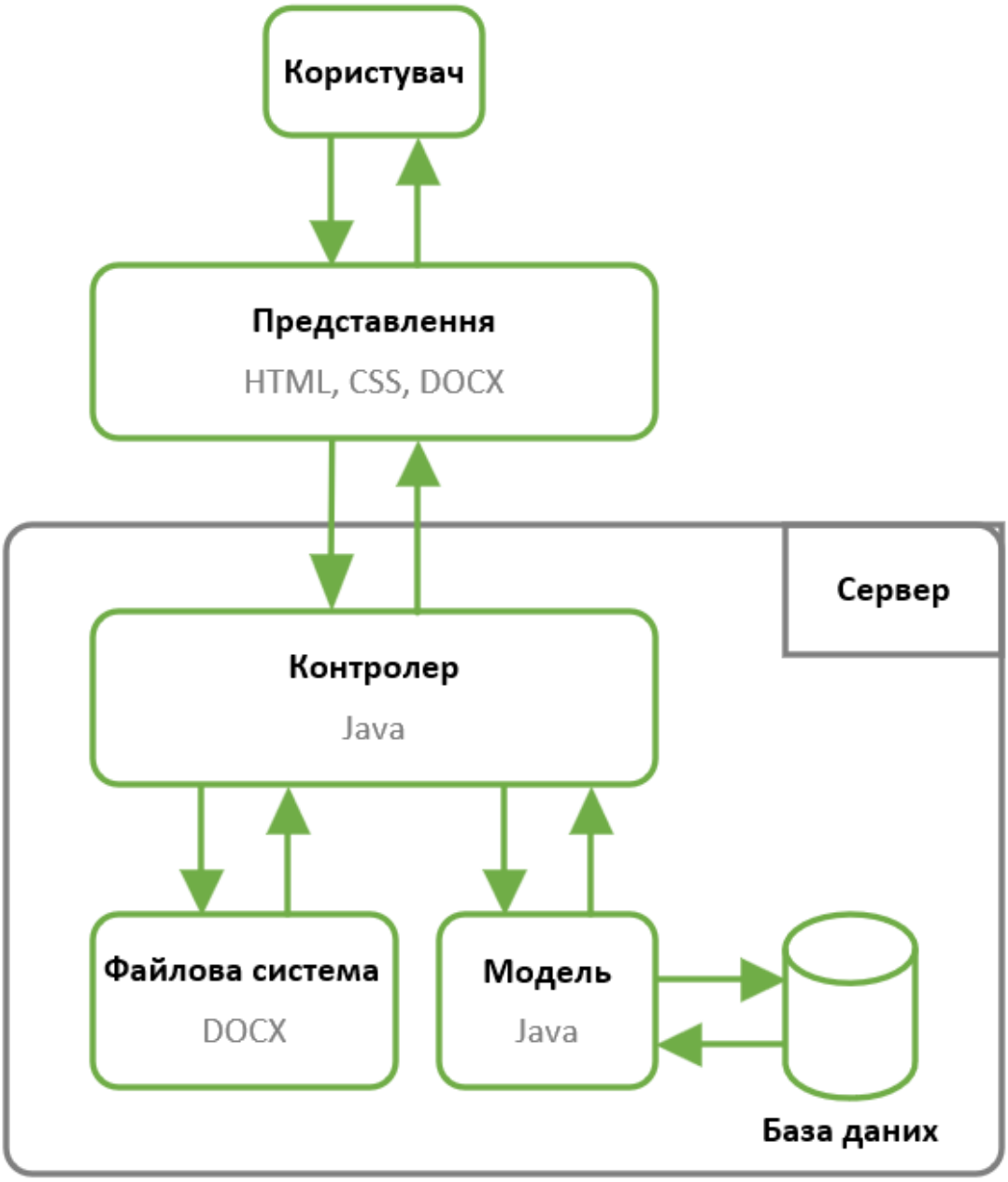


Рис. 2.3 – Схема *MVC* у контексті розроблюваного застосунку

Основною перевагою підходу *MVC* є можливість розбити додаток на компоненти, що дозволяє зробити його більш структурованим та зручним у розробці та підтримці. Крім того, зміни в одному з компонентів не впливають на роботу інших, що сприяє зменшенню кількості помилок та забезпечує більшу гнучкість додатка.

Широке застосування підходу *MVC* пояснюється його універсальністю та можливістю застосування для різних типів програмного забезпечення. Зокрема, цей підхід застосовують у веб-розробці для створення веб-додатків, а також у мобільній розробці для створення додатків для мобільних пристроїв.

Крім того, підхід *MVC* дозволяє розробникам зосередитися на окремих компонентах додатку, що сприяє збільшенню продуктивності та швидкості розробки. Завдяки розділенню додатка на компоненти, розробники можуть працювати над різними частинами додатка одночасно, що дозволяє прискорити процес розробки.

Нарешті, застосування підходу *MVC* підвищує зручність підтримки додатка. Якщо виникає потреба у зміні функціональності додатка, розробник може змінити лише один компонент, не змінюючи при цьому інших компонентів додатка. Це значно спрощує процес підтримки додатка та зменшує ризик появи помилок під час внесення змін.

2.3 Модуль *Spring MVC*

Spring MVC – це частина *Spring* фреймворку для розробки веб-застосунків за архітектурним шаблоном *MVC* на мові *Java*. *Spring MVC* використовує *Java*-сервлети для обробки *HTTP*-запитів і генерації *HTTP*-відповідей. Коли користувач надсилає запит на сервер, *Spring MVC* перехоплює його і передає в контролер. Контролер обробляє запит, отримуючи необхідні дані з моделі та передає їх у подання для відображення користувачеві.

Spring MVC також надає механізми для валідації даних та обробки винятків. Наприклад, можна використовувати анотації *@Valid* та *@ExceptionHandler* для валідації даних та обробки виняткових ситуацій відповідно.

Ще однією можливістю *Spring MVC* є використання *RESTful* веб-сервісів. *Spring MVC* дозволяє створювати *RESTful* веб-сервіси, які можуть обслуговувати

запити на читання та запис даних. Він надає механізми для маршалізації та демаршалізації даних у різних форматах, таких як *XML* та *JSON*, а також підтримує стандарти *RESTful* веб-сервісів, такі як *HTTP*-методи та *URI*.

Переваги *Spring MVC* можна наочно продемострувати наступним чином. Припустимо, що проект має домен «*scribe.ua*», і в проекті є певний *HTML* файл з якимись прикладами. Цей файл має назву «*example.html*» і зберігається на сервері, у папці проекту, у підпапці «*general*». Серверу потрібно пояснити, що цю сторінку потрібно відображати, коли користувач переходить на сторінку «*example*» цього сайту. Для цього потрібно створити метод з анотацією *@GetMapping*, і в параметрах анотації вказати «/*example*». Це значить, що цей метод буде викликатися, коли користувач буде переходити на сторінку «*example*», повний адрес якої буде виглядати так: «*scribe.ua/scribe*». При цьому, сам метод не містить ніякої інформації про домен.

Метод повинен повернути рядок, що означає путь до *HTML* файлу, який необхідно відобразити. Так як файл знаходиться у папці проекту, можна вказати відносний шлях. Також при певних налаштуваннях можна не вказувати розширення файлу. Цей метод може містити певну логіку, перед тим як повернути сторінку користувачеві. Метод необхідно розташувати у класі-контролері, що має відповідну анотацію:

```
@Controller
public class GeneralController {

    @GetMapping("/example")
    public String examplePage() {
        return "general/example";
    }

}
```

Spring MVC дозволяє проаналізувати адресу, яку запрошує клієнт, та повертати відповідь для певного шаблону адрес. Наприклад, користувач намагається переглянути шаблон з ідентифікаційним номером 5. Для цього він переходить за адресою «*scribe.ua/template/5*». Сервер бачить, що ця адреса відповідає шаблону, який написаний у параметрі анотації *@GetMapping*, і викликає метод *viewPage*. П’ятірку, яка означає ідентифікаційний номер, метод отримує в якості першого параметру, який має відповідну анотацію *@PathVariable*. В самому методі сервер

надсилає запит до бази даних, щоб знайти відповідний запис з ідентифікаційним номером 5 і додає цей об'єкт до моделі, яка також є частиною *Spring MVC*. Усі об'єкти, що додаються в модель, можна зчитати при відображенні сторінки, і підставляти значення полів об'єкта у відповідні місця сторінки:

```
@GetMapping("template/{id}")
public String viewPage(@PathVariable("id") Long id, Model model) {
    Template template = templateService.findById(id);
    if (template == null) {
        return "redirect:/templates?templateNotFound";
    }
    model.addAttribute("template", template);
    return "template/view";
}
```

В цілому, *Spring MVC* є потужним фреймворком для розробки веб-додатків на мові *Java*. Він надає механізми для поділу програми на компоненти, управління взаємодією між ними та забезпечення безпеки програми. Крім того, він інтегрується з іншими технологіями *Java*, дозволяючи розробникам створювати складніші та ефективніші веб-додатки.

2.4 Управління обліковими записами

Сервіс повинен керувати обліковими записами і в *Spring* є окремий модуль для цього типу задач. *Spring Security* – це високорівневий фреймворк безпеки, який надає механізми автентифікації та авторизації для програм, написаних на базі *Spring*. Він ґрунтується на обробці запитів фільтрами та використовує безліч конфігураційних можливостей для забезпечення гнучкого налаштування прав доступу.

Одним із ключових принципів *Spring Security* є модульність. У статті [4] написано, що фреймворк надає безліч модулів, які можна використовувати за необхідності. Наприклад, є модуль для базової автентифікації та авторизації, модуль для роботи з *OAuth* 2.0, модуль захисту від атак *CSRF* і т.д.

Spring Security також має гнучку систему конфігурації, яка дозволяє налаштовувати поведінку фреймворку залежно від вимог конкретної програми. Наприклад, можна налаштувати правила доступу для конкретних *URL*-адрес, вказати, які ролі мають доступ до певних сторінок або методів, визначити, які користувачі можуть виконувати певні дії тощо.

Spring Security також забезпечує безпеку на рівні методів та класів програми. Це досягається шляхом використання анотацій, які можуть бути застосовані до методів чи класів. Наприклад, анотація *@Secured* вказує, які ролі повинні мати користувачі, щоб мати доступ до методу, а анотація *@PreAuthorize* дозволяє задавати більш складні правила доступу на основі *Spring Expression Language*.

Нарешті, *Spring Security* надає безліч вбудованих механізмів безпеки, які допомагають захистити програму від різних уразливостей. Наприклад, фреймворк забезпечує захист від атак *CSRF, XSS, SQL*-ін'єкцій і т.д. Крім того, *Spring Security* надає інструменти для аудиту безпеки, які дозволяють відстежувати дії користувачів у додатку, щоб виявляти можливі загрози та вразливості.

Однією з ключових переваг *Spring Security* є його гнучкість та розширюваність. Фреймворк дозволяє легко налаштовувати правила доступу, інтегруватися з іншими фреймворками та інструментами та створювати власні модулі безпеки.

Крім того, *Spring Security* поставляється на базі відкритого вихідного коду і має активну спільноту розробників, які підтримують та розширюють фреймворк. Це дозволяє швидко реагувати на нові загрози безпеки та надавати нові функціональні можливості.

Spring Security є потужним і гнучким фреймворком безпеки, який надає механізм автентифікації та авторизації, а також безліч інструментів для захисту програм на базі *Spring* від різних загроз. Завдяки своїй модульності та гнучкості фреймворк може бути використаний для різних типів додатків, від простих веб-сайтів до складних корпоративних систем.

2.5 Шаблонізатор веб-сторінок *Thymeleaf*

Thymeleaf – це шаблонізатор, який використовується для створення веб-сторінок на *Java*. Він дозволяє розробникам створювати динамічні сторінки, які можуть легко адаптуватися під різні вимоги. Він надає зручний спосіб зв'язування даних з *HTML*-шаблонами, що робить процес створення веб-сторінок більш ефективнішим.

Однією з головних особливостей *Thymeleaf* є його здатність працювати як у серверній, так і клієнтській стороні. Це означає, що його можна використовувати для

створення статичних і динамічних сторінок, а також для створення *AJAX*-запитів. Це дозволяє розробникам створювати більш інтерактивні веб-сторінки за допомогою простих та зрозумілих засобів.

Згідно Вікіпедії [5], *Thymeleaf* заснований на *XML* / *HTML*-шаблонах і використовує різні атрибути для обробки та відображення даних. Він може використовуватися у поєднанні з багатьма фреймворками, такими як *Spring*, який робить його ще більш гнучким.

На рисунку 2.4 зображено загальну схему роботи шаблонізатора *Thymeleaf*:

– контролер. Отримує дані з моделі, обробляє їх та передає до двигуна *Thymeleaf*;
– двигун *Thymeleaf*. Отримавши дані від контролера, *Thymeleaf* використовує спеціальний парсер та генерує «чистий» *HTML*-код з отриманими даними;
– парсер. Він аналізує шаблон та створює DOM-дерево з HTML-коду та тегів Thymeleaf;
– HTML-сторінка. Кінцевий HTML-код, який відображається у браузері.

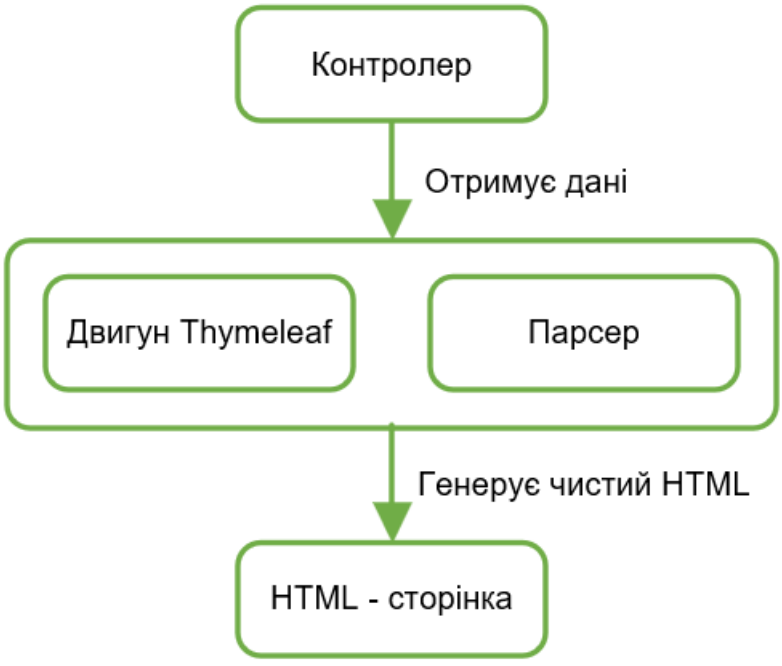


Рис. 2.4 – Схема роботи шаблонізатора *Thymeleaf*

Thymeleaf також підтримує безліч функцій, які полегшують роботу розробників. Він надає зручний спосіб включення фрагментів у *HTML*-шаблони, що

робить їх більш гнучкими та модульними. Крім того, він надає безліч вбудованих функцій, які можуть бути використані для обробки та форматування даних.

Наприклад, за допомогою синтаксису *Thymeleaf* можна виводити значення полів об’єкта, звертаючись до них по назві. Можна виводити або не виводити певний блок, в залежності від умови. Або можна вивести декілька однакових блоків з різними даними, які будуть передані як колекція об’єктів:

```
<div th:each="template : ${templates}" style="display: flex; margin:
20px 0;">
    <br>

    <div style="width: 30%; float: left;">
        <div style="display: block; text-align: center; margin-top:
10px;">
            <strong>
                <span th:text="${template.filename}"></span>
            </strong>
        </div>
        <div style="float: none; padding: 5px; margin-top: 10px;">
            <a
th:href="@{/templates/{id}(id=${template.templateId})}">
                <div class="scribe-element" style="width: 100%">
                    Переглянути шаблон
                </div>
            </a>
        </div>
    </div>

    <div style="width: 78%; margin-left: 2%; float: left;">
        <textarea                                    class="form-control"
th:text="${template.description}"
                  style="height:     100px;     min-height:     50px;"
readonly></textarea>
    </div>
</div>
```

2.6 Проектування бази даних

Бази даних є невід’ємною частиною більшості сучасних додатків. В даному застосунку доцільно використовувати саме реляційну базу даних. Реляційна база даних (РБД) – це тип бази даних, який використовує табличне подання даних, де кожна таблиця складається з рядків (кортежів) та стовпців (атрибутів). У реляційної БД дані зберігаються у вигляді таблиць, які пов'язані між собою за допомогою ключів.

У РБД дані можуть бути додані, змінені, видалені або вилучені за допомогою стандартної мови запитів – *SQL* (*Structured Query Language*). *SQL* використовується

для визначення структури бази даних (створення таблиць, визначення зв'язків з-поміж них тощо), і навіть виконання запитів на вибірку, оновлення чи видалення даних із таблиць.

JDBC (*Java Database Connectivity*) – це стандартний *API* для роботи з базами даних *Java*. Він дозволяє програмістам підключатися до бази даних, надсилати запити та отримувати результати. *JDBC* може працювати з будь-яким типом бази даних, для якої існує драйвер *JDBC*. Використання *JDBC* вимагає знання мови *SQL* та самостійне перетворення об’єктів в табличний запис та навпаки.

З часом для спрощення розробки додатків, зменшення кількості коду, який повторюється кожний новий проект, та підвищення продуктивності, з’явилася технологія що має назву *Object-Relational Mapping*. *ORM* – це технологія, яка полегшує роботу з базами даних, приховуючи від програміста деталі взаємодії з БД. Я написано у статті [6], *ORM* автоматично перетворює об’єкти в табличний запис та навпаки, і надає готові методи базових операцій *CRUD* (створення, читання, оновлення, видалення).

Кожна мова програмування має декілька своїх реалізацій *ORM*. Найбільш популярною реалізацією для *Java* є *Hibernate*. Він надає набір інструментів для зручної та ефективної взаємодії з базами даних. *Hibernate* дозволяє розробникам працювати з даними як з об'єктами *Java*, а не з *SQL*-запитами. Це полегшує процес розробки та зменшує кількість коду. Однією з ключових переваг *Hibernate* є його підтримка безлічі СУБД, включаючи *MySQL*, *Oracle* та *PostgreSQL*. *Hibernate* також надає механізм для автоматичної генерації схеми БД на основі класів *Java*.

Hibernate працює за принципом відображення класів *Java* на таблиці у базі даних. Для кожного класу *Java* у програмі створюється відповідна таблиця у базі даних (рис. 2.5).

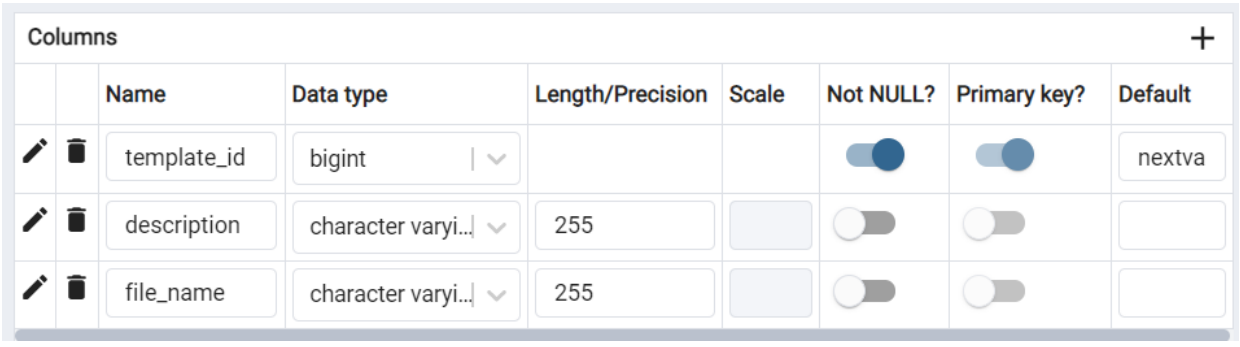


Рис. 2.5 – Автоматично створена таблиця на основі класу-сутності

Поля класу *Java* відповідають стовпцям таблиці, а об'єкти класу *Java* відповідають записам таблиці. Для зв’язку таблиць та встановлення обмежень використовують анотації. Такий клас, що відповідає певній таблиці в БД, називають сутністю.

```
@Entity
@Table(name = "template")
public class Template {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "template_id")
    private Long templateId;
    @Column(name = "filename", unique = true)
    private String filename;
    @Column(name = "description")
    private String description;
    public Template() {
    }
    public Long getTemplateId() {
        return templateId;
    }
    public void setTemplateId(Long templateId) {
        this.templateId = templateId;
    }
    public String getFilename() {
        return filename;
    }
    public void setFilename(String filename) {
        this.filename = filename;
    }
    public String getDescription() {
        return description;
    }
    public void setDescription(String description) {
        this.description = description;
    }
}
```

Так як технологія *ORM Hibernate* автоматично створює таблиці бази даних на основі *Java*-коду, варто зосередитися на проектуванні саме *Java*-класів. Спочатку необхідно визначити ключові об’єкти застосунку. Ключовими об’єктами є користувач, а точніше його акаунт, та шаблони. Тоді класи та їх поля можуть виглядати так:

1. *Account*. Містить інформацію тільки про акаунт користувача.
    a) *accountId: Long*. Первинний ключ;
    b) *email: String*. Електронна пошта;
    c) *password: String*. Пароль;

d) *person: Person*. Клас з даними користувача;

e) *roles: Set<Role>*. Набір ролей акаунту;

2. *Person* (Абстрактний клас). Містить інформацію про особисті дані користувача.
   a) *accountId: Long*. Первинний і зовнішній ключ для зв'язку з класом *Account*;
   b) *name: String*. Ім'я;
   c) *surname: String*. Прізвище;
   d) *patronymic: String*. Ім'я по батькові (патронім);
   e) *account: Account*. Посилання на об'єкт акаунту;
3. *Employee* (Наслідує *Person*).
   a) *position: String*. Посада;
   b) *degree: String*. Науковий ступінь;
4. *Student* (Наслідує *Person*).
   a) *faculty: String*. Факультет;
   b) *degree: String*. Освітній ступінь;
   c) *specialty: String*. Спеціальність;
   d) *year: String*. Курс навчання;
5. *Role*.
   a) *roleId: Long*. Первинний ключ;
   b) *name: RoleName*. Назва ролі;
6. *Template*.
   a) *templateId: Long*. Первинний ключ;
   b) *filename: String*. Назва файлу *DOCX* документа;
   c) *description: String*. Опис шаблона.

Схематично ці зв'язки можна зобразити за допомогою моделі «сутність-зв'язок» (рис. 2.6).

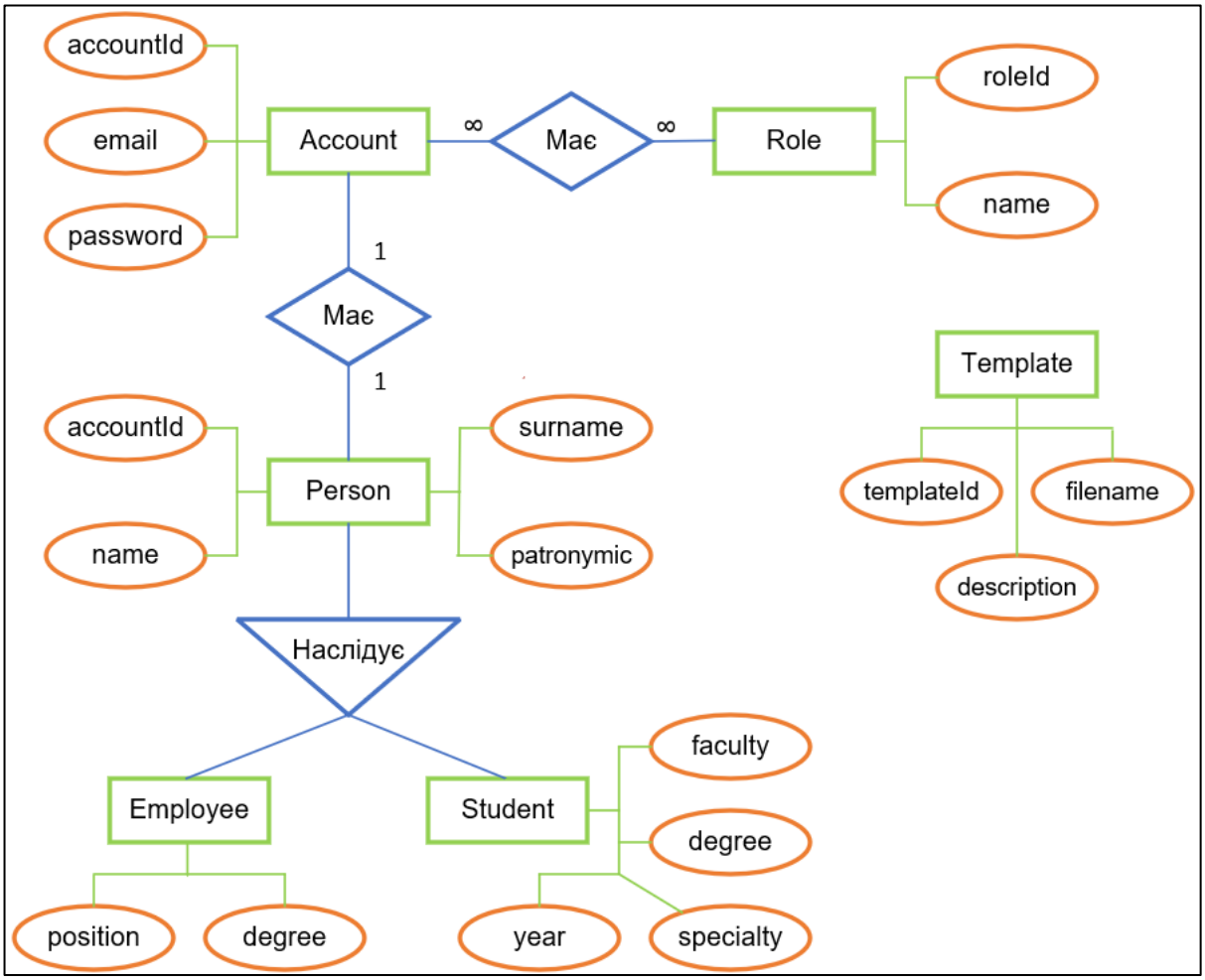


Рис. 2.6 – Модель «сутність-зв’язок»

При першому запуску застосунку *Hibernate* автоматично створить таблиці на основі цих класів. Класи *Person*, *Employee* та *Student* злилися в одну таблицю. Таблиця *account* та *role* мають відношення багато до багатьох, що неприпустимо, тому між ними була створена проміжна таблиця *account_role*. Результат можна побачити на рисунку 2.7.

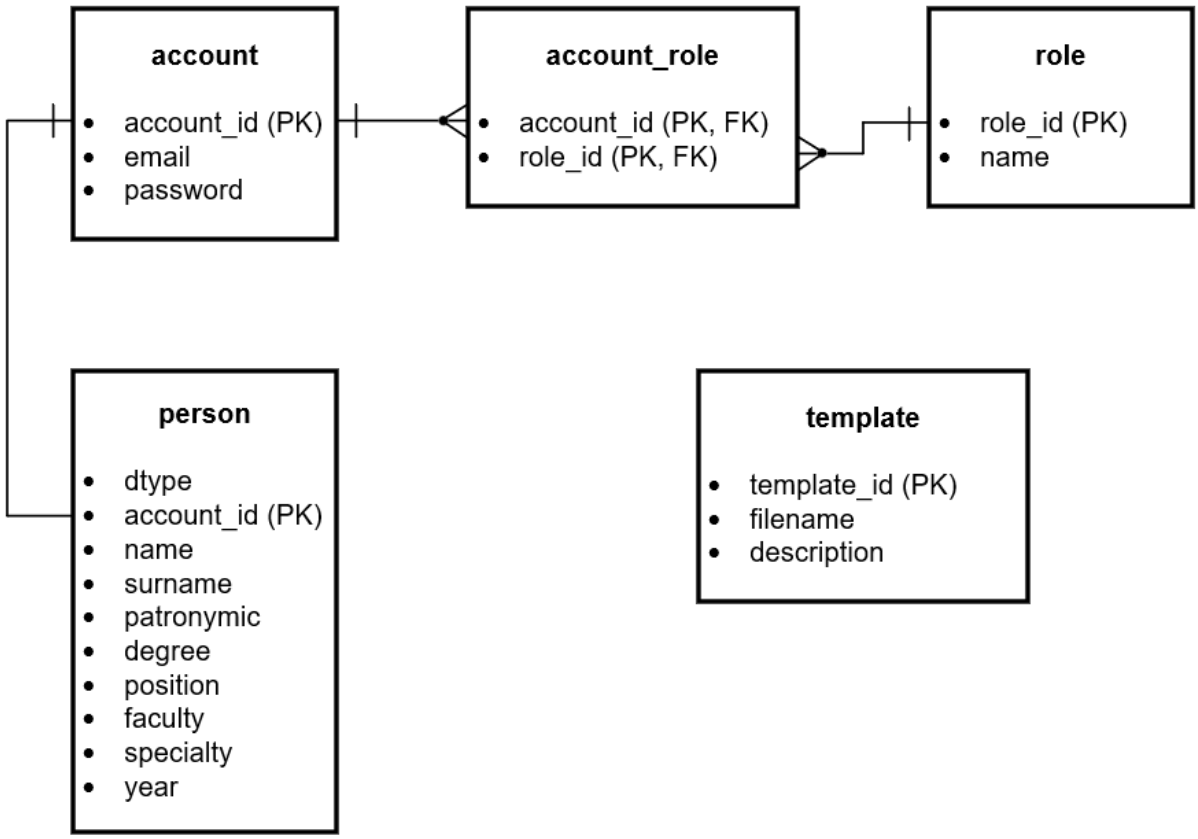


Рис. 2.7 – *ER*-діаграма

ORM автоматизують базові операції взаємодії з БД, але у проектах зазвичай потрібні більш складні запроси, такі як пошук запису за певними параметрами. *Spring Data* – це проект у рамках *Spring Framework*, який надає абстракції для роботи з різними типами сховищ даних, включаючи реляційні бази даних та *NoSQL*-сховища. *Spring Data* використовує різні технології *ORM*, включаючи *Hibernate*, для роботи з базами даних. Він надає набір анотацій та інтерфейсів для спрощення написання коду. Таким чином можна створювати складні запроси до баз даних лише на основі назв методів.

Наприклад, необхідно отримати шаблон за певною назвою. Використовуючи переваги *Spring Data*, потрібно створити інтерфейс-репозиторій, надавши інтерфейсу відповідну анотацію, і в цьому інтерфейсі написати метод без реалізації, але надавши назву відповідно синтаксису *Spring Data*:

```
@Repository
public interface TemplateRepository extends CrudRepository<Template,
Long> {
    Template findByFilename(String filename);
}
```

У класах, де необхідно звернутися до цього методу, потрібно звертатися через інтерфейс, використовуючи поліморфізм. На етапі компіляції, *Spring* автоматично проаналізує назву методів інтерфейсу-репозиторію, створить клас, що буде імплементувати цей інтерфейс, та підставить цей клас у потрібні місця, за допомогою впровадження залежностей:

```
@Service
public class TemplateService {
    @Autowired
    private TemplateRepository templateRepository;
    public Template findByFilename(String filename) {
        return templateRepository.findByFilename(filename);
    }
}
```

Бази даних є важливою складовою *MVC*-архітектури, і *Java* надає зручні та ефективні інструменти для роботи з ними. *JDBC* дозволяє виконувати *SQL*-запити та взаємодіяти з різними СУБД. Однак написання *SQL*-запитів може бути складним та трудомістким процесом. *ORM*-фреймворки, такі як *Hibernate*, надають більш високорівневий спосіб взаємодії з базами даних, дозволяючи працювати з даними як з об'єктами *Java*. *Spring Data* надає зручний спосіб написання запросів до БД.

Висновки до другого розділу

Цей розділ присвячено проектуванню архітектури застосунку та демонстрації технологій розробки. Спочатку було розроблено алгоритм взаємодії користувача з системою для уявлення кінцевого вигляду продукту. Далі було розглянуто фреймворк *Spring* як найпопулярніший фреймворк для *Java* для проектування веб-систем.

Для розробки архітектури додатку була обрана архітектурна модель *MVC*. Цей архітектурний підхід дозволяє розділити додаток на три основні компоненти: модель, представлення і контролер. Кожен з цих компонентів виконує свої функції і забезпечує зручний та ефективний процес розробки. Для втілення цієї моделі на практиці обрано технологію *Spring MVC*.

Керування обліковими записами відбувається за допомогою фреймворку *Spring Security*. Для більш зручного та швидкого створення веб-сторінок обрано шаблонізатор *Thymeleaf*. Для полегшення роботи з базою даних обрано технології *ORM Hibernate* та *Spring Data*.

РОЗДІЛ 3
РОЗРОБКА ТА ТЕСТУВАННЯ ЗАСТОСУНКУ

3.1 Обґрунтування вибору технологій

Існує безліч мов програмування (МП), кожна я яких має свої переваги та недоліки. В залежності від особливостей, кожна МП більше підходить для певного напрямку програмування. При виборі мови програмування для створення серверної частини сайту необхідно враховувати безліч факторів, таких як продуктивність, масштабованість, надійність та наявність відповідних інструментів та бібліотек (екосистема) (табл. 3.1).

Таблиця 3.1

Порівняння мов програмування для розробки серверної частини сайту

Мова	Продуктивність	Масштабованість	Надійність	Екосистема
Java	Висока	Дуже висока	Висока	Дуже велика
JavaScript	Середня	Середня	Низька	Велика
Python	Середня	Дуже висока	Висока	Велика
PHP	Низька	Середня	Середня	Велика
Ruby	Середня	Дуже висока	Висока	Середня
C#	Висока	Дуже висока	Висока	Велика

Java – популярна мова програмування для створення серверних додатків. Як зазначено у статті [7], *Java* має багато переваг. Вона забезпечує високу надійність та стійкість до помилок завдяки строгій типізації та перевірці помилок на етапі компіляції. Крім того, *Java* надає багатий набір інструментів для створення масштабованих додатків, які можуть підтримувати велику кількість користувачів. Велике співтовариство розробників та користувачів забезпечує обмін знаннями та досвідом, що може допомогти вирішити виникаючі проблеми. *Java* також є платформонезалежною мовою, що означає, що на ній можна розробляти додатки для будь-якої операційної системи, і вони будуть працювати на будь-якій іншій системі, яка підтримує віртуальну машину *Java* (*JVM*). Багата екосистема бібліотек та інструментів робить розробку додатків на *Java* більш швидкою та ефективною. Все

це робить *Java* чудовим вибором для створення надійних та масштабованих серверних додатків, в тому числі й для розробки серверної частини сайту.

Загалом, *Java* надає безліч інструментів для створення надійних і масштабованих додатків, що робить її відмінним вибором для розробки серверної частини сайту. Одним з таких інструментів є фреймворк *Spring*.

Spring – це фреймворк для розробки додатків на *Java*, який складається з безлічі модулів, які можуть бути використані для створення різних компонентів додатків, таких як веб-сайти, *REST API*, бекенд-сервіси та багато іншого. Кожний модуль можна підключати тільки якщо він необхідний, що позитивно впливає на продуктивність. Крім того, модульна структура фреймворку робить масштабованість застосуноку більш швидкою та простою. На рисунку 3.1 зображена діаграма з модулями *Spring*, відповідно до діаграми у онлайн-бібліотеці [8].

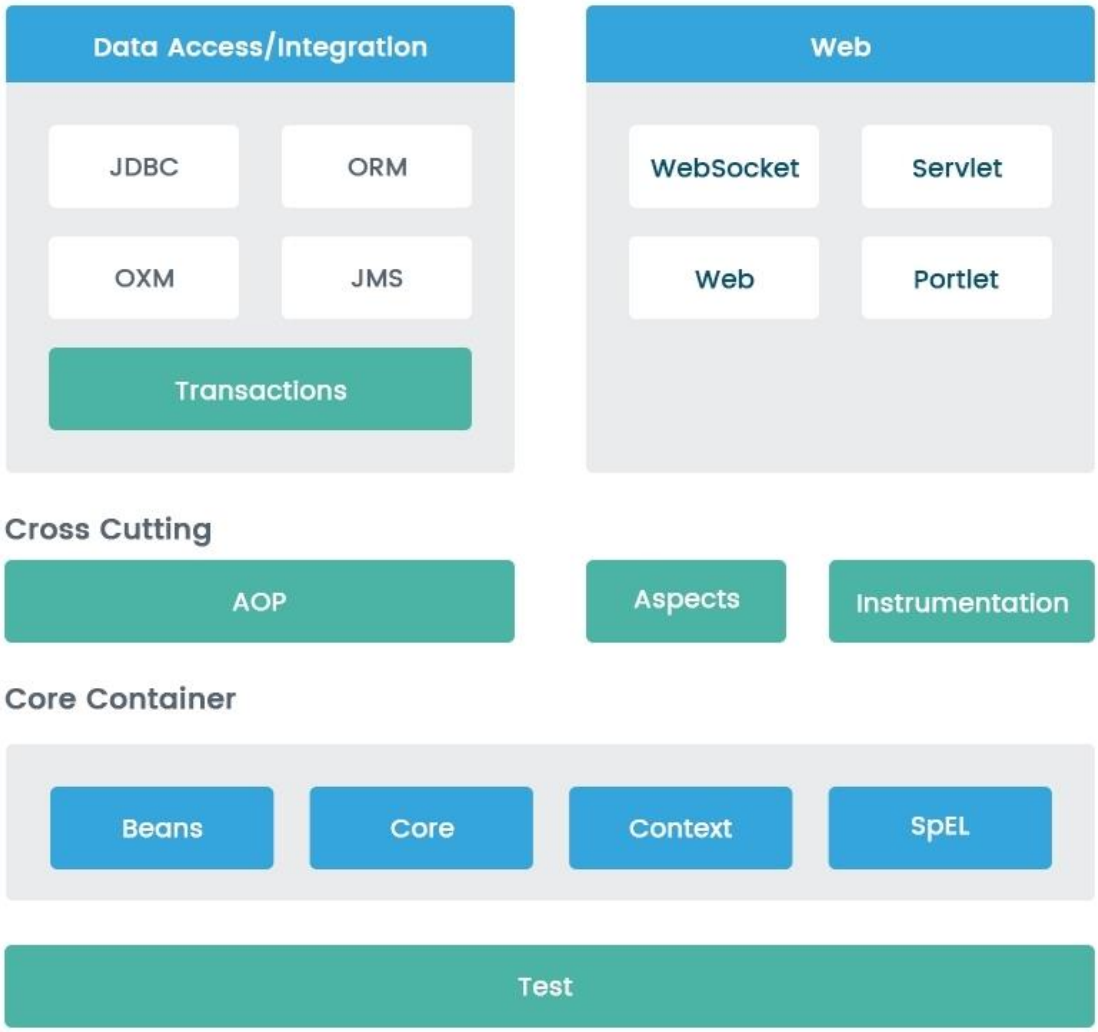


Рис. 3.1 – Структура фреймворку *Spring*

Spring – один з найпопулярніших та найефективніших фреймворків для розробки додатків на мові *Java*. Інверсія управління (*Inversion of Control, IoC*) та впровадження залежностей (*Dependency Injection, DI*) є ключовими концепціями у фреймворку *Spring*, які дозволяють створювати слабозв'язані об'єкти та керувати залежностями між ними, що спрощує тестування та полегшує супровід коду.

Інверсія управління – це підхід, при якому управління об'єктами та їх залежностями переноситься на контейнер, який створює, конфігурує та керує життєвим циклом об'єктів. Це означає, що класи не створюють екземпляри інших класів, а отримують їх з контейнера, що зменшує зв'язність між класами та спрощує їх тестування та зміну.

Впровадження залежностей – процес надання зовнішньої залежності програмному компоненту. Є специфічною формою інверсії управління, коли вона застосовується до управління залежностями. У повній відповідності з принципом єдиного обов'язка об'єкт віддає піклування про побудову необхідних йому залежностей зовнішньому, спеціально призначеному для цього спільного механізму. Це робиться автоматично контейнером при створенні об'єкта.

Застосування *IoC* та *DI* у *Spring* дозволяє створювати програми з високою гнучкістю та масштабованістю. Класи стають більш незалежними та перевикористовуваними, що зменшує зв'язність та спрощує тестування та зміну коду. Крім того, фреймворк *Spring* надає багато інших функцій, таких як підтримка транзакцій, кешування, безпеки та інші, що допомагають розробляти якісні програми швидко та ефективно.

Для реалізації певного функціоналу також було підключено модулі *Spring Boot*, *Spring MVC*, *Spring Security* та *Spring Data*. Крім того, що вони надають розробникам потужний набір інструментів, вони всі є частиною однієї екосистеми *Spring*, завдяки чому ці модулі добре інтегруються та взаємодіють один з одним. Все це дозволяє прискорити та спростити розробку програм на *Java*, забезпечуючи при цьому високу продуктивність та безпеку.

3.2 Впровадження алгоритмів

Найчастіше фреймворки потребують певної структури проекту, і *Spring* в тому числі, тому часто зручно використати автоматизований процес генерації

нового проекту з необхідною структурою. Такі функції можуть бути влаштовані в *IDE*, але для генерації *Spring*-проектів існує більш зручний веб-сервіс *Spring Initializr*. В результаті завантажується заархівована папка проекту з потрібною структурою.

Під час створення проекту за допомогою сервісу *Spring Initializr*, можна одразу підключити необхідні залежності із числа популярних. Залежності в *Spring* додаються не у вигляді *jar*-файлів, а за домогою систем збірок проекту. В даному випадку використовується *Maven*, і кожна залежність має вигляд блоку коду з деякими параметрами (наприклад версія бібліотеки). Ці залежності, і не тільки, прописуються у файлі "*pom.xml*", що має певну структуру. В цей файл необхідно вручну додати бібліотеку для редагування *DOCX* файлів, адже її не було на *Spring Initializr*:

```
<dependency>
    <groupId>org.apache.poi</groupId>
    <artifactId>poi-ooxml</artifactId>
    <version>5.2.3</version>
</dependency>
```

У папці з вихідним кодом існує дві папки: "*java*" – тут зберігаються усі *Java*-класи; та "*resources*" – тут зберігаються файли *HTML, CSS*, картинки для сайту, а також файл «*application.properties*» з конфігурацією проекту. Папка "*java*" містить наступні підпапки (пакети):

- *config*. Містить класи з конфігурацією проекту;
- *controller*. Містить контролери *MVC*;
- *dto*. Містить класи *DTO*;
- *model*. Містить класи-сутності та пов'язані з ними сервіси;
- *scribe*. Містить класи для роботи з *DOCX* файлами та пов'язану з цим логіку;
- *service*. Містить усю іншу бізнес логіку, що викликається у контролерах.

Розпочати розробку необхідно з моделі, тобто з частини яка працює з базою даних, тому що усі інші компоненти залежать від цієї частини. Модель, в свою чергу, не залежить від інших компонентів, тому ропочинати розробку з моделі буде ще й зручно.

Як вже було сказано, в пакеті “*model*” будуть зберігатися не тільки класи-сутності, а й сервісні класи пов’язані з ними. Для цього в пакеті було створено два підпакети: “*entity*” – для сутностей, та “*service*” – для сервісів. Сутності та зв’язки між ними були спроетовані та описані у попередньому розділі, залишилося тільки обрати потрібні анотації технології *ORM Hibernate* та правильно їх налаштувати для досягнення необхідних зв’язків та обмежень.

Для кожного поля вказується анотація “*Column*”, яка вказує фреймворку, що ці поля повинні мати колонку в таблиці в БД. В дужках можна вказати налаштування для кожної колонки: “*name*” задає назву колонки, “*unique = true*” забороняє повтори однакових значень, а “*updatable = false*” забороняє змінювати значення в подальшому. Для поля “*accountId*” задано анотації “*Id*”, яка означає що поле є первинним ключем; та “*GeneratedValue*”, яка вказує на те, що це поле має бути згенеровано автоматично. Зв’язок «один до одного» з класом *Person* дозволяє втілити анотація “*OneToOne*”: атрибут “*cascade*” визначає каскадність зв’язку, тобто як змінення запису в таблиці account будуть впливати на зв’язаний запис в таблиці *person*; “*fetch*” визначає, в який момент часу з БД буде завантажена сутність *Person* (впливає на продуктивність); “*mappedBy*” використовується для вказівки назви поля типу *Account* в сутності *Person* (потрібно для двостороннього зв’язку «один до одного»). Для зв’язку «багато до багатьох» з класом *Role* недостатньо відповідної анотації “*ManyToMany*”, тому що цей зв’язок передбачає створення проміжної таблиці, яку необхідно налаштувати за допомогою анотації “*JoinTable*”:

```
@Entity
@Table(name = "account")
public class Account {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "account_id")
    private Long accountId;

    @Column(name = "email", unique = true, updatable = false)
    private String email;

    @Column(name = "password")
    private String password;

    @OneToOne(cascade = CascadeType.ALL, fetch = FetchType.EAGER,
mappedBy = "account")
    private Person person;
```

```
    @ManyToMany(cascade = CascadeType.PERSIST, fetch =
FetchType.EAGER)
    @JoinTable(
            name = "account_role",
            joinColumns = @JoinColumn(name = "account_id"),
            inverseJoinColumns = @JoinColumn(name = "role_id"),
            uniqueConstraints = @UniqueConstraint(columnNames =
{"account_id", "role_id"})
    )
    private Set<Role> roles = new HashSet<>();
```

Spring Data автоматично генерує запити до бази даних, в залежності від назв методів. Його використання передбачає створення інтерфейсів-репозиторіїв. Для цих репозиторіїв створено підпакет “*repository*” в пакеті “*model.service*”. Ці репозиторії повинні бути помічені анотацією “*Repository*”, а також розширяти один із інтерфейсів фреймворку *Spring Data*. В проекті було створено три репозиторії (для сутностей *Account*, *Role* та *Template*) та написано методи, що можуть знадобитися надалі, використовуючи спеціальний діалект *Spring Data*:

```
@Repository
public interface AccountRepository extends CrudRepository<Account,
Long> {
    boolean existsByEmail(String email);
    Account findByEmail(String email);
    @Override
    Account save(Account account);
    @Override
    void delete(Account account);
    @Transactional
    void deleteByEmail(String email);
}
```

Для встановлення зв’язку з базою даних необхідно вказати певні налаштування: яку саме БД застосунок повинен використовувати, ім’я користувача, пароль, назва бази даних та інші. Використання фреймворку *Spring Boot* спрощує цей процес за умови дотримання правильної структури проекту та іменування, тому для зв’язку застосунку з базою даних необхідно вказати ці параметри у файлі «*application.properties*» у вигляді «ключ=значення» з правильно названими ключами:

```
spring.datasource.driverClassName=org.postgresql.Driver
spring.datasource.url=jdbc:postgresql://localhost:5432/scribe
spring.datasource.username=postgres
spring.datasource.password=1234
spring.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=create
```

Наступним етапом є створення веб-сторінок та контролерів за допомогою фреймворку *Spring MVC*. Для початку потрібно наочно продемострувати як дані будуть переходити від користувача в базу даних. Для цього була розроблена діаграма потоків даних (*Data Flow Diagram, DFD*). Вона допомагає візуалізувати, як дані проходять через систему, вказує на те, які процеси обробляють ці дані та як вони зберігаються (рис. 3.2).

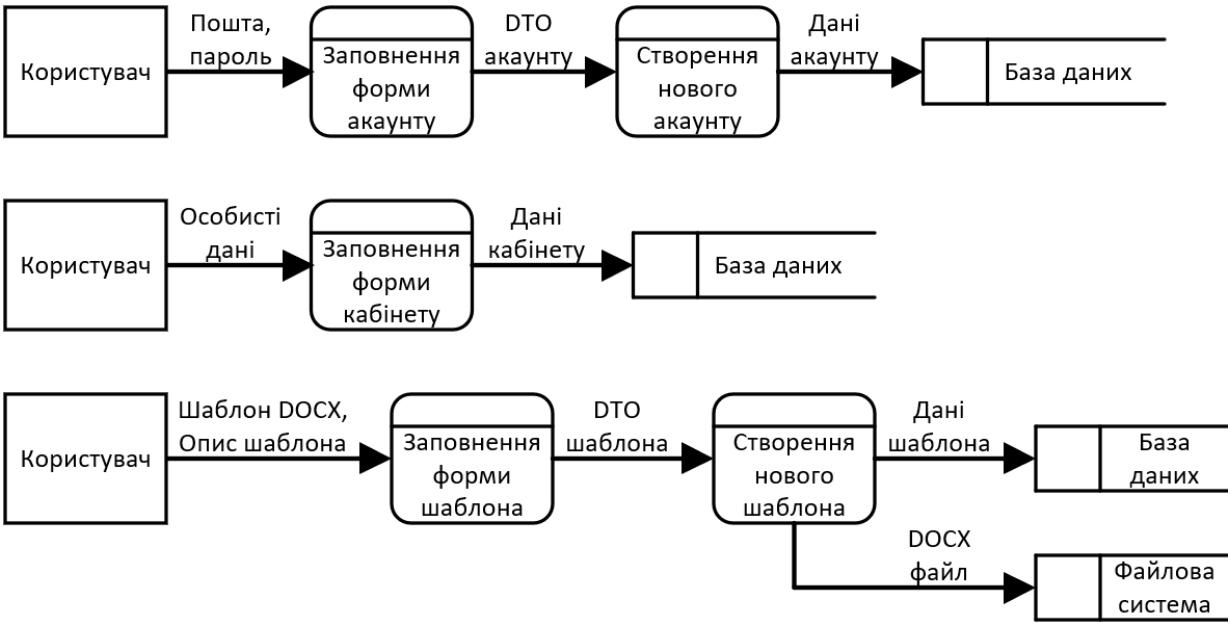


Рис. 3.2 – Діаграма потоків даних

Для створення контролера *Spring MVC* необхідно створити клас, який буде обробником запитів. Цей клас повинен бути анотований анотацією "*Controller*", яка вказує на те, що цей клас є контролером. Контролер може містити один або кілька методів, кожен з яких опрацьовує певний запит. Для кожного такого методу необхідно вказати два параметри: *REST*-метод запиту та відносний *URL*.

REST (*Representational State Transfer*) – це стиль архітектури програмного забезпечення для побудови сервісів в інтернеті. *REST* спирається на протокол *HTTP* та має низку методів. Запит на один *URL* але з різними методами обробляється по-різному. Наприклад, метод *GET* використовується для перегляду даних, метод *POST* – для створення нового запису в БД, метод *PUT* – для оновлення даних, а метод *DELETE* – для видалення.

Отже, в залежності від *REST*-методу, обирається відповідна анотація для методу контролера, а в її параметрах вказується відносний *URL*. Кожний метод

контролеру повинен повертати рядок, що означає назву *HTML*-сторінки, яку потрібно повернути та відобразити, або відносний *URL*, на який потрібно перенаправити користувача:

```
@PostMapping("/registration")
public     String     registerAccount(@ModelAttribute("registration")
RegistrationDto registrationDto,
                                RedirectAttributes redirectAttributes) {
    if (accountService.isExist(registrationDto.getEmail())) {
        redirectAttributes.addFlashAttribute("registration",
registrationDto);
        return "redirect:/authentication/registration?emailTaken";
    }
    if
(!registrationDto.getPassword().equals(registrationDto.getPasswordCon
firmation())) {
        redirectAttributes.addFlashAttribute("registration",
registrationDto);
        return
"redirect:/authentication/registration?confirmationError";
    }
    accountService.saveDto(registrationDto);
    return "redirect:/authentication/login?success";
}
```

Для передачі даних між клієнтом та сервером, зазвичай використовують *DTO*-класи. Клас *DTO* (*Data Transfer Object*) – це клас в *Java*, який використовується для передачі даних між різними рівнями додатку або між додатками через мережу. *DTO* містить дані, які потрібно передати, а також методи для отримання і встановлення цих даних. Він не містить логіки, пов'язаної з обробкою даних, а служить лише для передачі їх між різними компонентами. Використання *DTO* замість класів-сутностей має кілька переваг, серед яких:

- розділення відповідальності;
- підвищення безпеки;
- спрощення версіонування *API*;
- спрощення серіалізації;
- покращення продуктивності.

Веб-сторінки мають розширення “*.html*” та зберігаються у папці “*resources/templates*”. Для їх створення використовується мова розмітки *HTML* та шаблонізатор *Thymeleaf*, який полегшує зв’язок між *java*-кодом та мовою *HTML*. Принцип їх спільної роботи наступний: При переході на сторінку додавання нового

шаблона, у відповідному методі контролеру в параметрах необхідно прийняти об’єкт типу *Model*, який являє собою контейнер для передачі даних з контролера у представлення. Цей об’єкт не потребує анотацій та його автоматично надає *Spring MVC*. Використовуючи відповідний метод, у модель можна додавати так звані атрибути для передачі їх до шару представлення для відображення даних на веб-сторінці, або для зручного подальшого прийняття вже заповненого атрибута з форми:

```
@GetMapping("/new")
public String newPage(Model model) {
    model.addAttribute("template", new NewTemplateDto());
    return "template/new";
}
```

Без використання шаблонізатора *Thymeleaf*, необхідно було б трансформувати об’єкти у формат *JSON*, потім відтворювати із *JSON* об’єкти *JavaScript* і вже потім відтворювати інформацію на веб-сторінці. З використанням *Thymeleaf*, можна звертатись до полів об’єкта за назвою атрибуту та відповідною назвою полів об’єкта, відтворюючи інформацію на веб-сторінці без використання *JavaScript*:

```
<h4>Додати шаблон</h4>
<form            th:method="post"            th:action="@{/templates}"
th:object="${template}" enctype="multipart/form-data">
    <div>
        <label for="file" class="control-label">Оберіть файл:</label>
        <input   id="file"   class="form-control"   th:field="*{file}"
type="file" accept=".docx"
               style="height:  auto;  padding:  0;  background-color:
#CDB99C; color: White;">
    </div>
    <div style="margin: 20px 0 20px 0;">
        <label  for="description"  class="control-label">Опис  шаблона
(максимум 300 символів):</label>
        <textarea        id="description"        class="form-control"
th:field="*{description}"
                 maxlength="100"   style="height:300px;   min-height:
50px;"></textarea>
    </div>
    <div class="form-group">
        <input  type="submit"  class="form-control  btn  btn-primary"
value="Додати"/>
    </div>
</form>
```

Thymeleaf також полегшує передачу інформації в зворотний бік. Заповнивши форму на веб-сторінці, на метод контролеру надійде повноцінний *java*-об’єкт, який був доданий до моделі, замість кожного поля форми окремо. Для його отримання у

контролері необхідно до сигнатури відповідного методу додати параметр того ж тип, що був доданий до моделі, і додати до цього параметру анотацію “*ModelAttribute*”, вказавши в параметрах анотації назву доданого атрибуту:

```
@PostMapping
public    String    add(@ModelAttribute("template")    NewTemplateDto
templateDto) {
    if (templateDto.getFile().isEmpty()) {
        return "redirect:/templates/new?emptyFile";
    }
    if
(templateService.isExist(templateDto.getFile().getOriginalFilename())
) {
        return "redirect:/templates/new?fileExist";
    }
    try {
        fileService.save(templateDto.getFile());
        Template template = templateService.saveDto(templateDto);
        return  "redirect:/templates/"  +  template.getTemplateId()  +
"?success";
    } catch (IOException e) {
        e.printStackTrace();
        return "redirect:/templates/new?error";
    }
}
```

Бізнес-логіку прийнято виносити з контролерів в окремі класи, кожний з яких відповідає за певний функціонал. В результаті було створено багато класів і інтерфейсів, що переплітаються між собою. Для цієї частини застосунку було створено діаграму класів, що відображає назву класів та інтерфейсів, та зв’язки між ними (рис. 3.3).

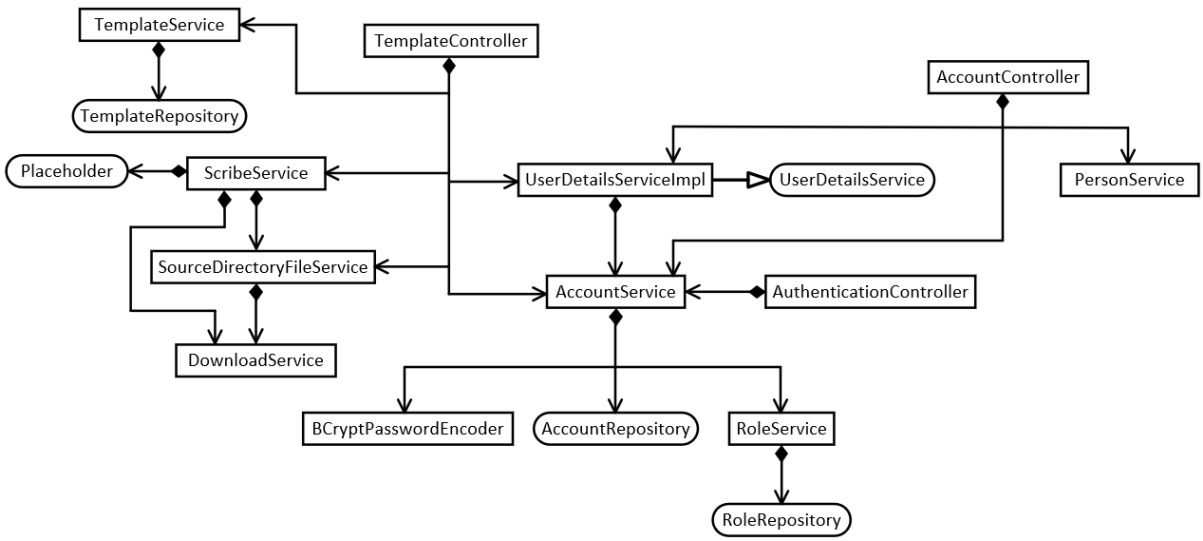


Рис. 3.3 – Діаграма класів компоненту контролера

Веб-сторінки містять різні елементи інтерфейсу з посиланнями на інші сторінки, в результаті чого склалося певне “дерево” з гіперпосилань, яке можна представити за допомогою діаграми (рис. 3.4).

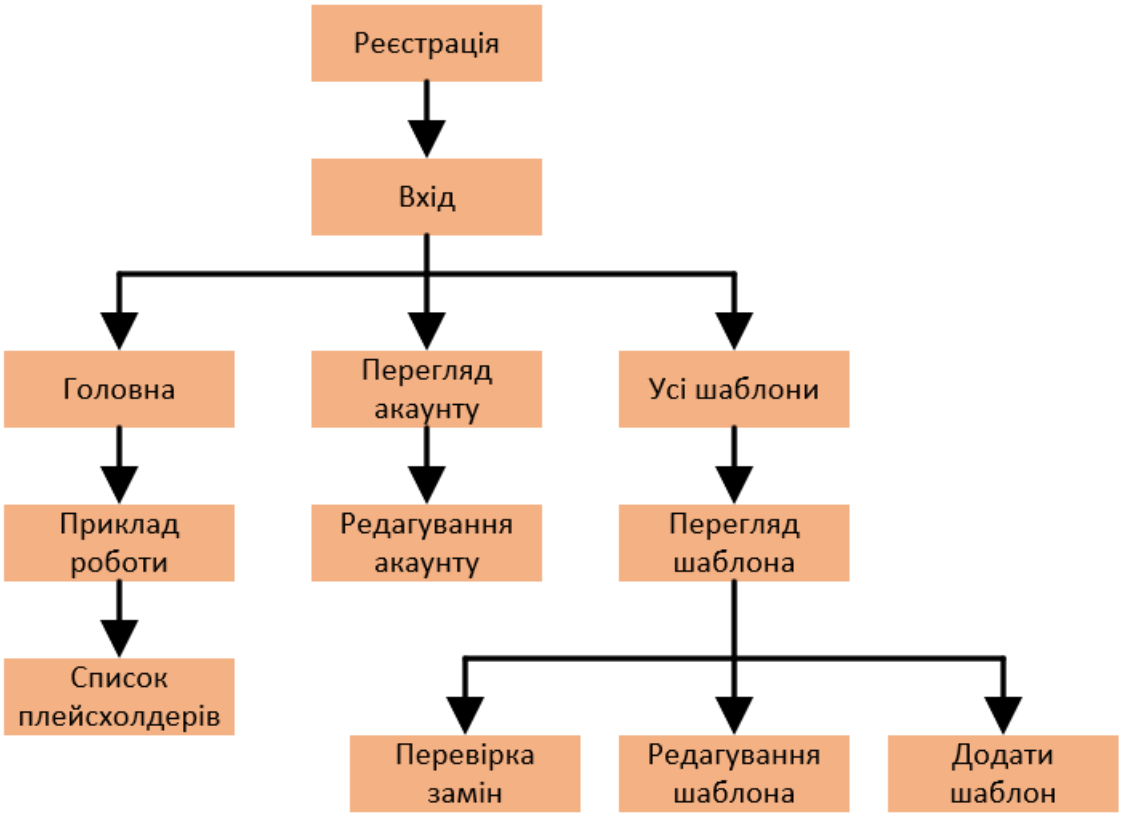


Рис. 3.4 – Структура сайту

Для редагування *DOCX* файлів було обрано бібліотеку *Apache POI*. Вона надає великий набір класів для обробки різних файлів пакету *MS Office*. Для зручності використання, скорочення кількості коду, та автоматизації деяких операцій, було створено клас *Scribe*, в пакеті “*scribe*”. Цей класс надає тільки необхідні функції в контексті даного застосунку, такі як відкриття файлу, закриття файлу, пошук плейсхолдерів та заміна тексту:

```
public class Scribe {
    private final String filename;
    private final FileInputStream fileInputStream;
    private final XWPFDocument document;
    private final WordReplacer replacer;
    public Scribe(File file) {
        try {
            filename = file.getName();
            fileInputStream = new FileInputStream(file);
            document = new XWPFDocument(fileInputStream);
            replacer = new WordReplacer(document);
```

```
            } catch (IOException e) {
                throw new RuntimeException(e);
            }
        }
        public String getFilename() {
            return filename;
        }
        public XWPFDocument getDocument() {
            return document;
        }
        public Set<String> getPlaceholders(String regex) {
            Set<String> result = new LinkedHashSet<>();
            String text = new XWPFWordExtractor(document).getText();
            Matcher matcher = Pattern.compile(regex).matcher(text);
            while (matcher.find()) {
                result.add(matcher.group());
            }
            return result;
        }
        public void replace(String placeholder, String replacement) {
            replacer.replaceWordsInText(placeholder, replacement);
            replacer.replaceWordsInTables(placeholder, replacement);
        }
        public void replaceAll(Map<String, String> map) {
            for (String placeholder : map.keySet()) {
                replace(placeholder, map.get(placeholder));
            }
        }
        public void close() {
            try {
                fileInputStream.close();
                document.close();
            } catch (IOException e) {
                throw new RuntimeException(e);
            }
        }
    }
```

Також було створено сервісні класи: Клас *SourceDirectoryFileService* надає функції відкриття, збереження та видалення файлу за його назвою, у файловій системі, у директорії що винесена у конфігураційний файл «*application.properties*»; а також клас *ScribeService*, що автоматизує відкриття та закриття потоків, а також надає функції для завантаження файлів на стороні клієнта.

Останнім етапом є підключення облікових записів за допомогою *Spring Security*. Головна частина налаштування безпеки знаходиться в одному методі, який повинен відповідати певним вимогам. Цей метод повинен бути помічений анотацією "*Bean*", повертати *SecurityFilterChain*, та приймати *HttpSecurity*. Отриманий *HttpSecurity* налаштовується у тілі метода за допомогою патерну програмування «будівельник». Викликаючи відповідні методи, можна налаштувати, які адреси

будуть доступні для усіх, які – тільки для користувачів з відповідними правами доступу, на яку адресу потрібно перенаправити користувача при спробі перейти на недоступну йому адресу тощо:

```
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws
Exception {
    http
            .csrf(csrf -> csrf.disable())
            .authorizeHttpRequests(auth ->
                    auth
                            .requestMatchers("/authentication/**",
"/css/**", "/pictures/**").permitAll()
                            .anyRequest().authenticated()
            )
            .formLogin()
            .loginPage("/authentication/login")
            .defaultSuccessUrl("/account")
            .permitAll()
            .and()
            .logout()
            .invalidateHttpSession(true).clearAuthentication(true)
            .logoutRequestMatcher(new
AntPathRequestMatcher("/authentication/logout"))
            .logoutSuccessUrl("/authentication/login?logout")
            .permitAll()
            .and()
            .authenticationProvider(authenticationProvider());
    return http.build();
}
```

Така конфігурація дозволяє неавторизованим користувачам відвідувати тільки сторінки реєстрації та авторизації. Усі інші сторінки доступні авторизованим користувачам з будь-якою роллю.

Для розгортання застосунку на сервері необхідно переконатись, що на сервері встановлена *JDK* версії 17 або вище (*Java* підтримує зворотню сумістність). Також на сервері потрібна бути встановлена реляційна система керування база даних (СКБД). В даному випадку проект підключений до СКБД *PostgreSQL*, але *Java*-додатки легко переносити на інші СКБД у разі такої необхідності. Для цього потрібно поміняти відповідні значення у конфігураційному файлі «*application.properties*». Також необхідно вручну створити базу даних з назвою, що вказана у «*application.properties*», адже *Hibernate* автоматично створює тільки таблиці, але не саму базу даних. Усі залежності завантажуються автоматично через інтернет завдяки системі автоматичної збірки проектів *Maven*.

3.3 Вибір методу тестування

Тестування є важливою частиною розробки програмного забезпечення, яке забезпечує якість, функціональність і продуктивність програмних додатків. Існують різні методи тестування, кожен з яких має свої переваги та недоліки. Часто для тестування великих застосунків використовують одразу декілька методів тестування. Як пише спілка програмістів [9], найчастіше використовуваними є наступні різновиди тестування:

1. Юніт-тестування. Це метод тестування програмного забезпечення, який передбачає перевірку окремих частин коду (юнітів) на відповідність заданим специфікаціям;

2. Модульне тестування. Це різновид тестування, яка передбачає тестування окремих компонентів програмного забезпечення. Зазвичай його виконують розробники за допомогою автоматизованих інфраструктур тестування, і це може допомогти виявити помилки на ранніх стадіях процесу розробки. Модульне тестування є відносно недорогим і ефективним, оскільки його можна автоматизувати та виконувати швидко. Однак його може бути складно налаштувати та підтримувати, і він може не виявляти всі типи помилок;

3. Інтеграційне тестування. Це тип тестування, який передбачає перевірку того, як різні компоненти програмного забезпечення працюють один з одним. Інтеграційне тестування може бути дорогим і трудомістким, оскільки вимагає багато координації між різними командами та може включати тестування вручну;

4. Системне тестування. Це різновид тестування, що передбачає тестування всієї програмної системи в цілому. Тестування системи може бути дорогим і трудомістким, оскільки вимагає комплексного середовища тестування та може включати ручне тестування;

5. Ручне тестування. Це тип тестування, який передбачає перевірку працездатності застосунку вручну. Його можна виконувати на будь-якому етапі процесу тестування та охоплювати широкий спектр сценаріїв, які може бути важко автоматизувати. Ручне тестування може бути відносно недорогим і ефективним для невеликих проектів або певних типів тестування, але може стати дорогим і трудомістким для великих або складних проектів.

Для даного застосунку найбільше підійде саме ручне тестування, оскільки автоматизоване тестування інтерфейсної частини складно налаштовується та не є вигідним для малих проектів, а бізнес логіка пов’язана зі зміною файлів *DOCX*. Тексти в таких файлах містять безліч варіантів форматування тексту, таблиць, зображень та інших елементів, які можуть з'явитися в документі. Людина може швидше та точніше помітити неправильне відображення елементів, які може пропустити автоматичне тестування. Крім того, детальне автоматизоване тестування таких файлів передбачає використання спеціалізованих бібліотек, які можуть не мати відповідного функціоналу для перевірки кожного параметру налаштування стилів.

Також ручне тестування може бути корисним для дослідницького тестування, коли людині надається свобода досліджувати програмне забезпечення та виявляти будь-які проблеми, які, можливо, не були виявлені автоматизованим тестуванням.

3.4 Тестування застосунку

Локальний сервер – це серверне програмне забезпечення, яке встановлюється та запускається на локальному комп'ютері або мережі для розробки та тестування веб-сайтів. Воно дозволяє розробникам створювати та налагоджувати сайти локально перед тим, як розмістити їх на публічних серверах. Локальний сервер імітує роботу реального сервера, дозволяючи перевіряти функціональність та зовнішній вигляд сайту перед його публікацією в Інтернеті.

При переході на сайт, система автоматично перенаправляє неавторизованого користувача на сторінку входу в обліковий запис. Для створення нового облікового запису, можна перейти на сторінку реєстрації, нажавши відповідну кнопку в меню зверху сторінки (рис. 3.5).

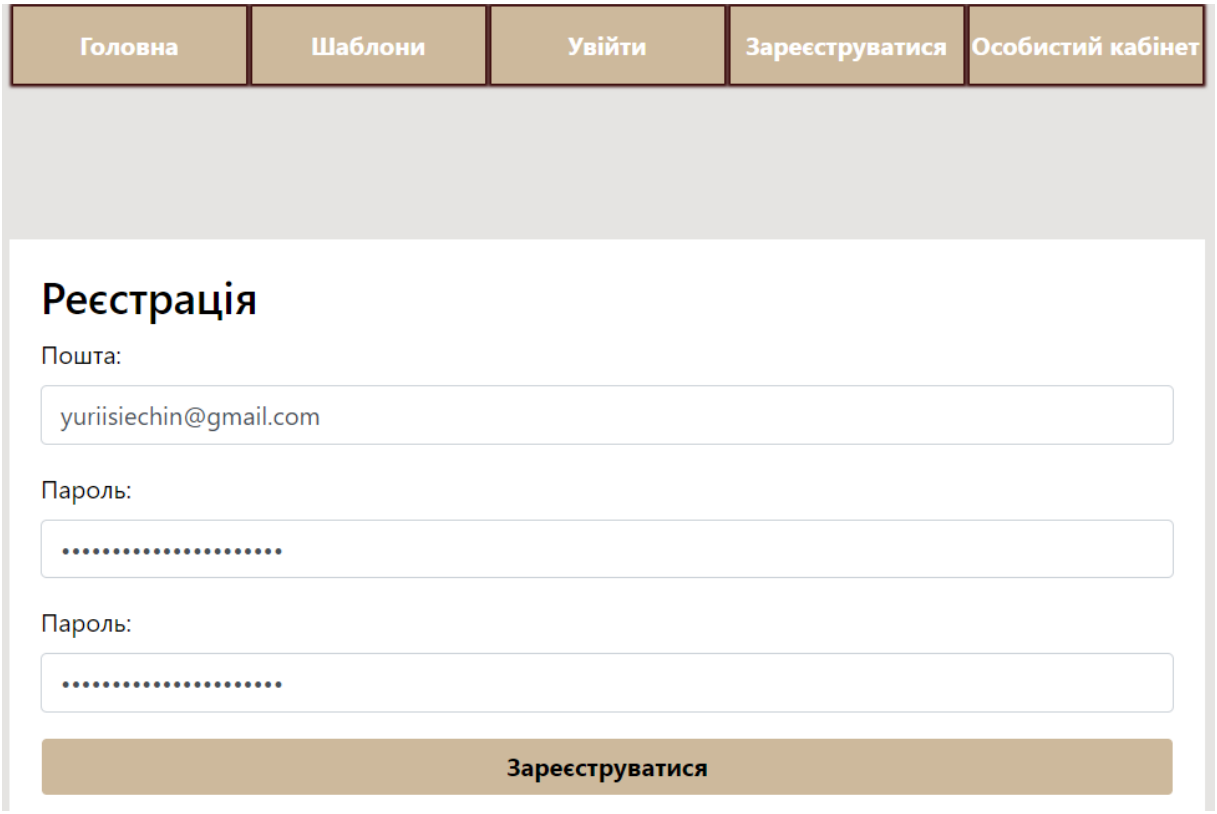


Рис. 3.5 – Сторінка реєстрації

Після успішної реєстрації, користувача буде знову перенаправлено до сторінки входу, з відповідним повідомленням про успішну реєстрацію. Після входу до облікового запису, користувачеві стають доступні інші сторінки сайту. При переході авторизованого користувача на сторінку входу або реєстрації, система перенаправить його на головну сторінку з відповідним повідомленням (рис. 3.6).

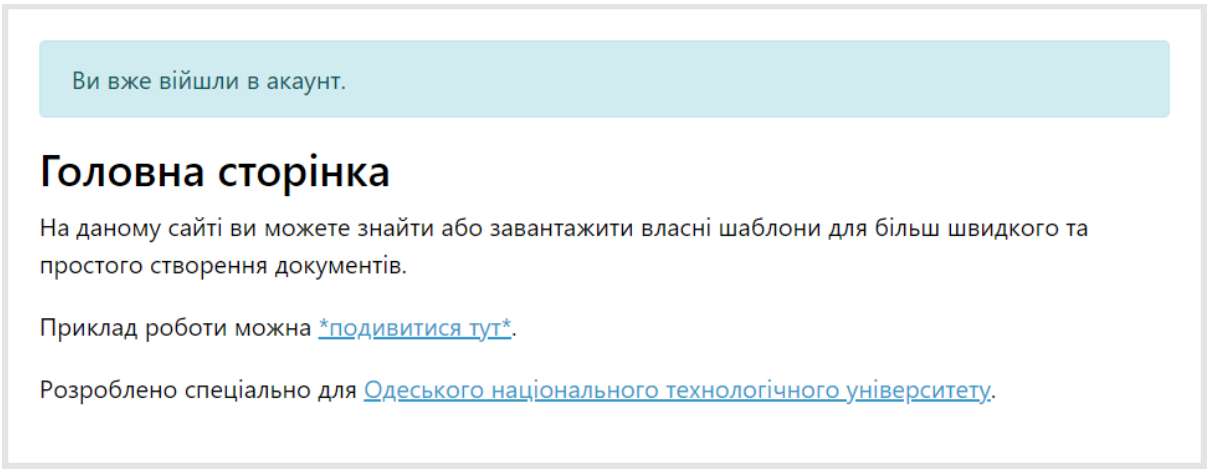


Рис. 3.6 – Головна сторінка

Наступним етапом є тестування особистого кабінету. При першому переході користувач обирає тип акаунту і заповнює притамані цьому типу акаунту поля (рис. 3.7). Під час тестування було протестовано обидва типи акаунту; збереження значень після виходу з облікового запису та входу знову; редагування полів вже існуючого акаунту.

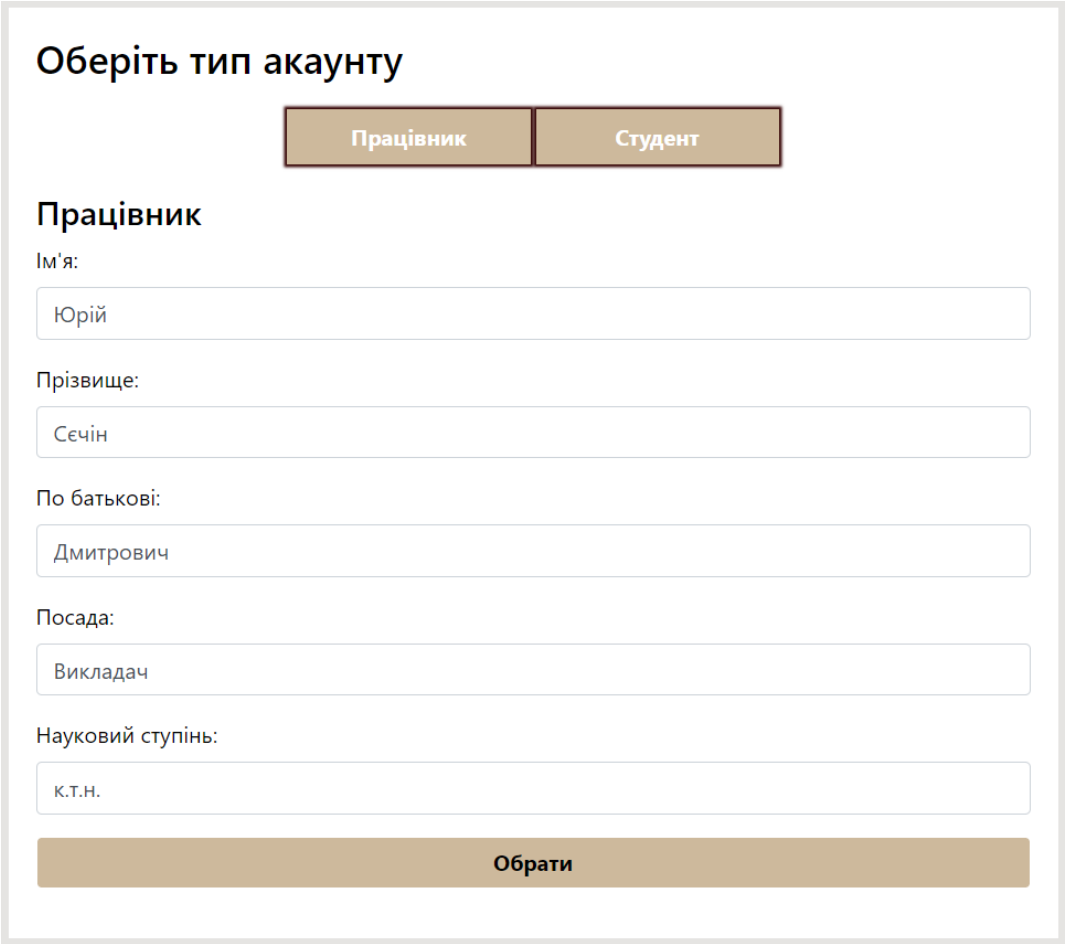


Рис. 3.7 – Вибір типу акаунту

Коли налаштування особового кабінету завершено, можна переходити до тестування розділу шаблонів. Для тестування було створено і завантажено на сервер декілька *DOCX* файлів. Було протестовано їх перегляд, редагування опису, видалення, завантаження оригіналу на комп’ютер користувача, а також пошук шаблонів за ключовими словами на сторінці усіх шаблонів (рис. 3.8 – 3.10).

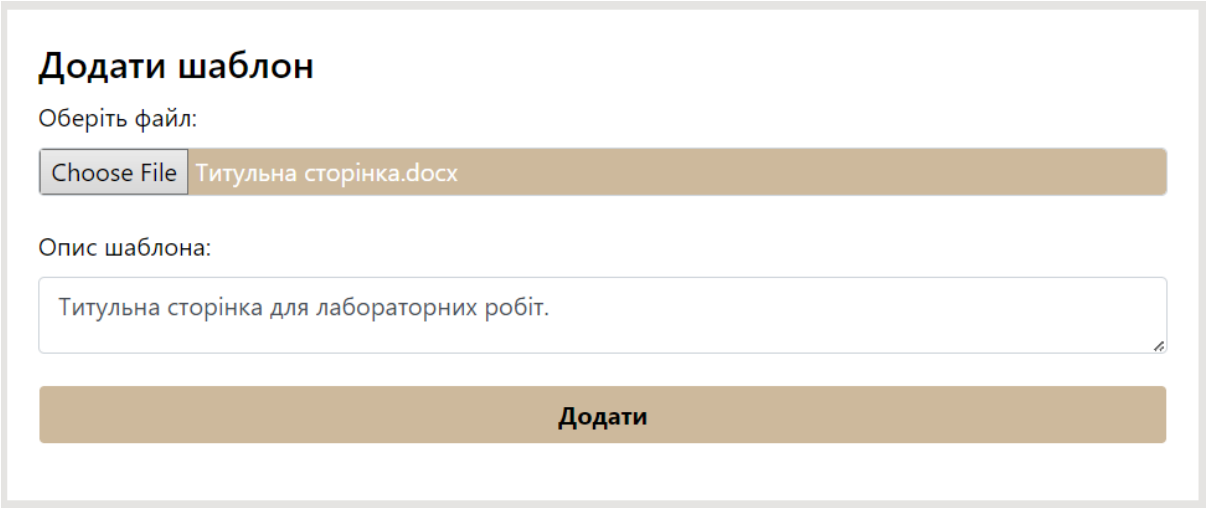


Рис. 3.8 – Завантаження шаблона на сервер

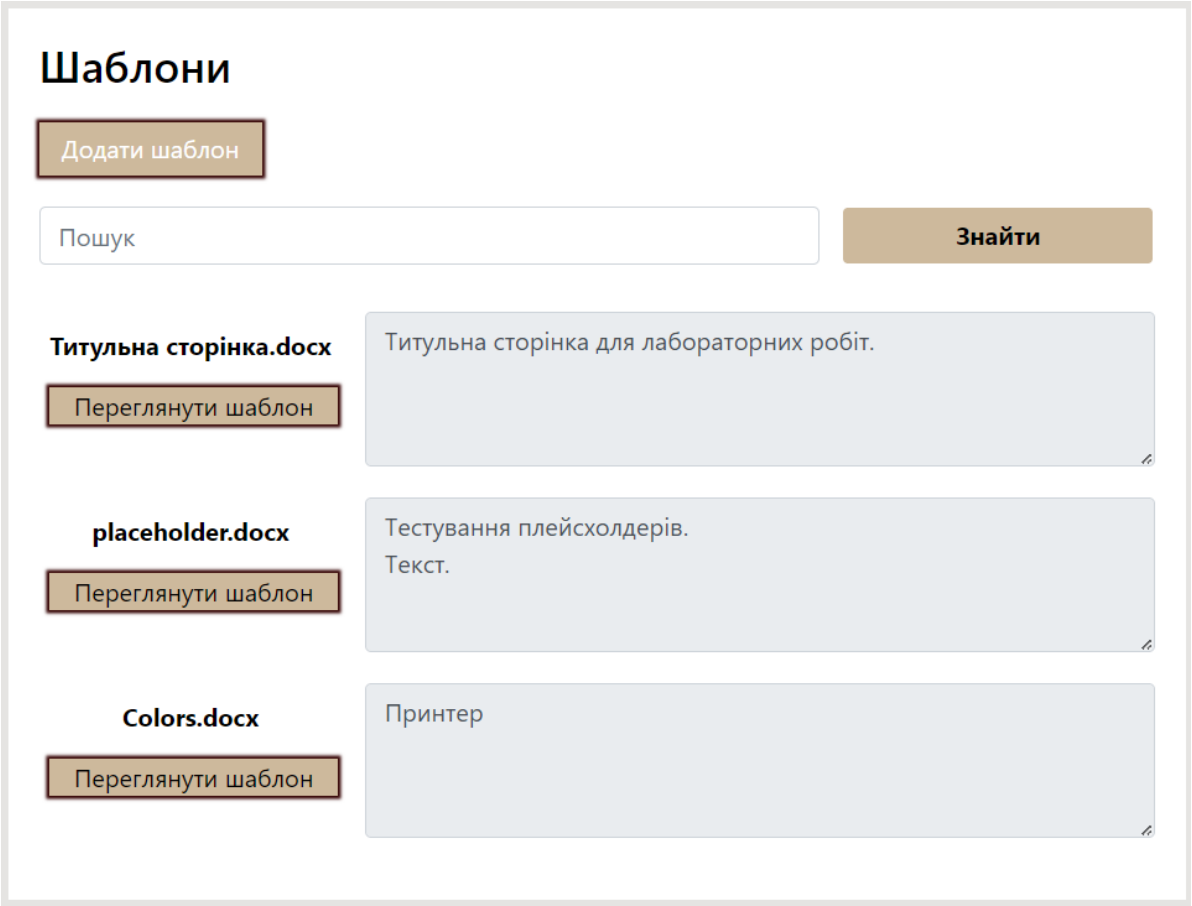


Рис. 3.9 – Сторінка з усіма шаблонами

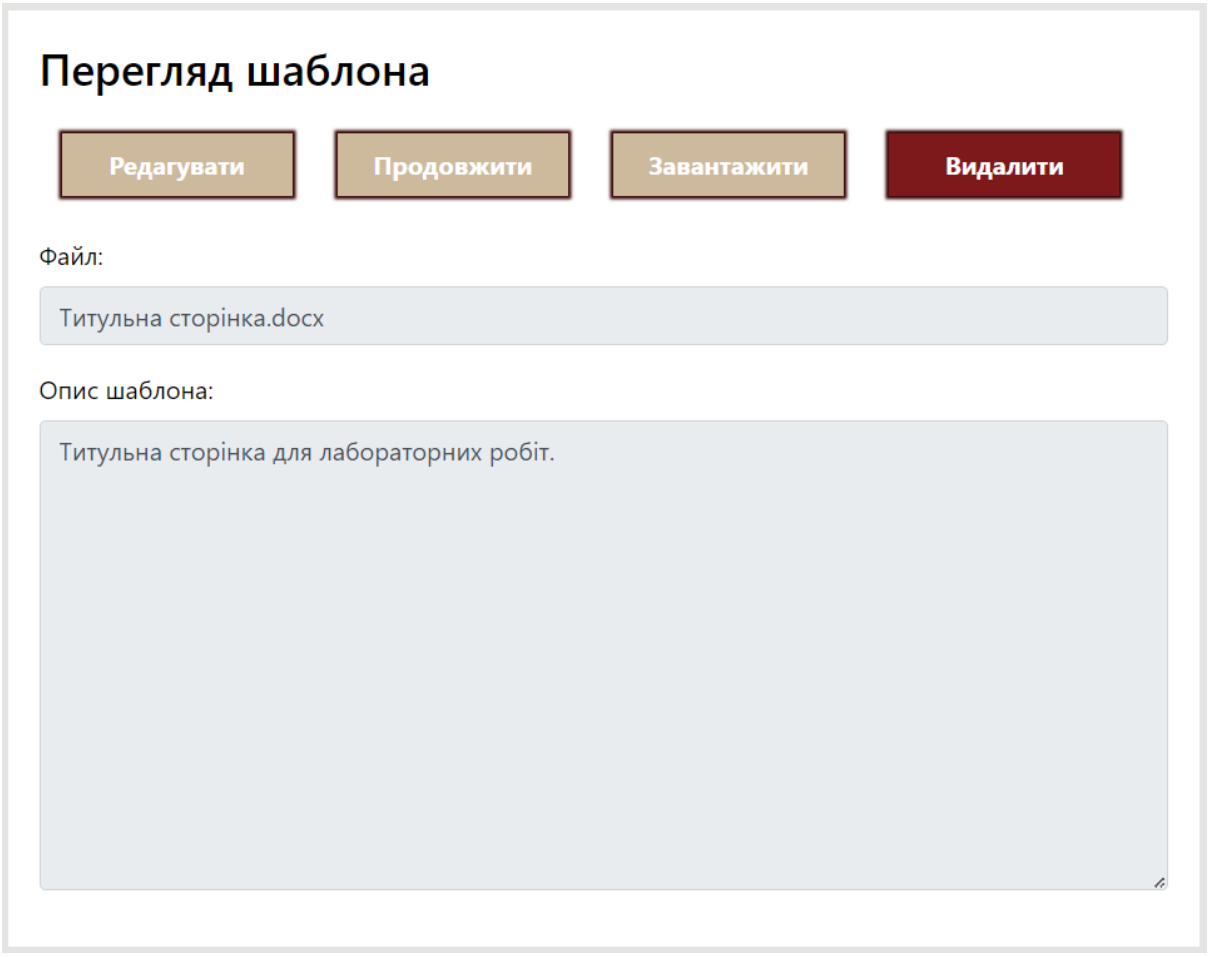


Рис. 3.10 – Сторінка перегляду певного шаблона

Останнім етапом тестування є тестування підставлення у шаблон потрібних значень. Тестовий сценарій буде проводитися на прикладі шаблона зі змістом титульної сторінки для лабораторних робіт. Натиснувши кнопку «Продовжити» на сторінці перегляду певного шаблона, користувач потрапляє на сторінку з усіма плейсхолдерами обраного шаблона. Деякі значення для підстановки, такі як ініціали та рік, вже були заповнені, як і планувалося в розділі проектування. Користувачу залишається лише заповнити решту полів та обрати потрібний формат для завантаження готового документу (рис. 3.11).

Рис. 3.11 – Заповнення значень для підстановки у шаблон

Обидва формати файлу були завантажені без проблем. Усі плейсхолдери були замінені на відповідні значення. Форматування стилів в згенерованих документах збереглося відповідно стилям шаблону.

Висновки до третього розділу

Цей розділ присвячено безпосередньо розробці застосунку. Спочатку було обгрунтовано вибір технологій, обраних на стадії проектування. Далі було описано процес розробки застосунку з демонстрацією коду його деяких окремих частин.

Тестування є важливою частиною розробки програмного забезпечення, яке забезпечує якість, функціональність і продуктивність програмних додатків. В цьому розділі було перераховано та описано основні методи тестування застосунку.

В результаті було прийнято рішення тестувати розроблений застосунок методом ручного тестування. Після цього було описано процес тестування розробленого застосунку. Тестування показало, що застосунок працює як було заплановано.

РОЗДІЛ 4
ОЦІНКА ЕФЕКТИВНОСТІ РОЗРОБКИ

В даній дипломній роботі передбачено створення веб-застосунку, який дозволить швидко та зручно створювати багато однотипних документів, підставляючи значення з бази даних у відповідні місця шаблону *DOCX*, тим самим автоматизуючи процес створення документів на основі шаблонів. Використання даного засобу може бути особливо корисним для організацій, що часто працюють зі стандартними документами (наприклад, звіти, договори тощо). Прикладами таких організацій можуть бути фінансові установи, юридичні фірми, медичні установи та учбові заклади.

В даному розділі розглядається вплив розробленого застосунку на різні сфери життя людей. Розділ також містить розрахунок витрат на розробку та оцінку науково-технічної ефективності розробки.

У методичних вказівках [10] зазначено: «В умовах відкритої ринкової економіки розширюється діапазон оцінки ефективності науково-технічних розробок, а отже, збільшується кількість основних видів ефективності науково-дослідних та дослідно-конструкторських робіт (НДДКР), які необхідно визначити з метою цієї оцінки». До них належать:

- науково-технічний ефект;
- економічний ефект;
- соціальний ефект;
- маркетинговий ефект;
- екологічний ефект.

«Науково-технічний ефект проявляється у підвищенні науково-технічного рівня, поліпшенні параметрів техніки і технологій, що випливає з відкриття нових законів та закономірностей у природі, а отже, і нових технологічних засобів виробництва речовин, матеріалів та видів продукції.» [10]. Розроблювана система не відкриває новий напрямок в області розробки програмного забезпечення, але надає новий підхід у вже існуючій області генерації документів на основі шаблонів, з точки зору користування кінцевими користувачами, надаючи більш зручний та простий спосіб створення документів із шаблонів.

«Економічний ефект полягає в отриманні економічних результатів від науково-технічних розробок як в цілому для народного господарства, так і для кожного виробничого суб'єкта. Економічна ефективність науково-технічних розробок за відповідною системою показників має відображати вплив їхньої результативності на розвиток економіки країни в цілому, а також регіонів, галузей, організацій і підприємств, що беруть участь у реалізації технологічних нововведень.» [10]. Даний застосунок може бути корисним для великої кількості виробничих суб'єктів, працюючих з документами, оскільки дозволяє швидко та ефективно створювати документи на основі шаблонів та бази даних. Це може зменшити витрати на робочу силу, а також забезпечити високу якість документів, що зменшує ризик неприємних наслідків в майбутньому.

За проведеним орієнтовним розрахунком витрати на розробку складатимуть 37280 грн (або 1000 доларів США). Детальніші статті витрат наведені в таблиці 4.1.

Таблиця 4.1

Статті витрат на розробку

Статті витрат	**Кількість**	**Одиниця виміру**	**Ціна (грн)**
Праця одного розробника	2 місяці	Місячна заробітна плата	33100
Електроенергія	300 кВт	кВт*год.	500
Опалення	91,6 $м^2$	грн./$м^2$	1500
Холодна вода	26 $м^3$	грн./$м^2$	900
Інтернет	60 днів	грн./день	480
Оргтехніка	1 багато-функціональний пристрій	Папір, картридж	800
Загалом			37280

В методичних вказівках [10] зазначено: «Соціальний ефект відображає зміни умов діяльності людини в суспільстві. Його прояв спостерігається в змінах характеру та умов праці, підвищенні життєвого рівня населення, поліпшенні побутових його умов, розширенні можливостей духовного розвитку особистості, у

змінах стану довкілля.» Даний додаток може полегшити та прискорити роботу людей, які безпосередньо чи опосередковано працюють з документами, знижуючи їх навантаження та сприяючи збільшенню продуктивності. Крім того, це може мати позитивний вплив на якість роботи та життєвий рівень працівників та студентів.

Відповідно до роботи [10], «Маркетинговий ефект відображає потреби ринку в наукових дослідженнях і розробках та можливість їх реалізації.»

Як було зазначено в першому розділі, серед конкурентів можна виділити такі проекти як *Google Docs, DocuSign* та *PandaDoc*, але є також і інші рішення. Велика кількість таких проектів свідчить про те, що автоматизація документообігу є актуальним та перспективним напрямком у галузі розробки програмного забезпечення та має попит.

Розроблювана система може стати цікавою для підприємств та організацій, які потребують створення багатьох документів на основі шаблонів. Це може забезпечити успіх проекту та підвищити його популярність на ринку. В контексті учбових закладів, додаток може привернути увагу абітурієнтів, підвищити рівень задоволеності поточних студентів та викладачів. Крім того, застосунок може підвищити рівень репутації вищого навчального закладу та залучити нових спонсорів та партнерів.

Більшість сучасних онлайн-сервісів надають свої послуги на основі підписки. Наприклад, проект *DocuSign* надає три варіанти підписки: 15 доларів на місяць, 45 доларів на місяць, або 65 доларів на місяць. В залежності від варіанту підписки, користувач отримує різний набір функцій. У разі комерційного впровадження розробленого застосунку на ринку, рекомендовано надавати один варіант пакету щомісячної підписки ціною в 350 грн./міс. (або 10 дол./міс.) для юридичних осіб, та безкоштовне для студентів. У разі продажу самого проекту з усіма правами та вихідним кодом рекомендована ціна в 50000 гривень (або 1350 доларів).

Як зазначено у книжці [12], в Україні відсутнє комплексне галузеве регулювання ринку розробки ПЗ з боку держави: «...відсутні спеціальні акти, які передбачають на ринку загальні вимоги до компаній, їх послуг чи продуктів. При цьому такі вимоги не встановлені і в неспеціальних нормативно-правових актах.»

Екологічний ефект застосуноку не має прямого впливу на екологію, але опосередковано екологічний ефект може проявлятися в зменшенні використання

паперу, в наслідок зменшення помилок при створенні документів. В свою чергу це зменшує потребу в рубці дерев, а отже, допомагає зберегти лісові масиви.

В роботі [10] описано, як визначати науково-технічну ефективність (НТЕ): «НТЕ результатів прикладних робіт визначають на основі показників науково-технічного рівня. Оцінка науково-технічної ефективності НДДКР відбувається на основі показника ($О_{НТЕ}$), який представляє собою ступінь досягнення максимально можливого рівня, значення якого дорівнює 1 (одиниці)». Тобто:

$$О_{НТЕ} = \frac{К^{Ф}_{НТЕ}}{К^{П}_{НТЕ}}, \quad (4.1)$$

де $К^{Ф}_{НТЕ}$ – коефіцієнт фактичного рівня науково-технічної ефективності;

$К^{П}_{НТЕ}$ – коефіцієнт потенціально можливого рівня науково-технічної ефективності (дорівнює одиниці).

Значення показника $К^{Ф}_{НТЕ}$ визначають на основі шкали експертних оцінок, згідно таблиці з методичних вказівок [10] (табл. 4.2).

Таблиця 4.2

Шкала експертних оцінок для виміру рівня науково-технічної ефективності проектів

№	Групи показників	Характиристика показників	Інтервал рейтингового числа	Коефіцієнт значущості показників
1	Науково-технічний рівень	Перевищує кращі світові аналоги	10	0,35
		Відповідає світовому рівню	7 – 9	
		Нижче кращих світових аналогів	5 – 6	
		Перевищує кращі вітчизняні аналоги	3 – 4	
		Відповідає вітчизняному рівню	1 – 2	
		Нижче вітчизняного рівня	0	
2	Перспективність	Першочергова значущість	8 – 10	0,35
		Значущий	5 – 7	
		Корисний	1 – 4	

Продовження таблиці 4.2

№	Групи показників	Характиристика показників	Інтервал рейтингового числа	Коефіцієнт значущості показників
3	Потенційний масштаб практичного використання	Світовий ринок	10	0,20
		Галузі національної економіки	7 – 9	
		Галузь (регіон)	3 – 6	
		Окремі підприємства (об’єднання)	1 – 2	
4	Ступінь вірогідності досягнення позитивних результатів	Великий	10	0,10
		Середній	5 – 9	
		Малий	1 – 4	

Примітка: об'єкт оцінки і аналог(и), які порівнюють за однаковими показниками, наведеними у співставленому вигляді відхилення в значеннях кожного з показників, мають бути однаковими для варіантів, що порівнюються.

Згідно роботі [10], визначають $К^{Ф}_{НТЕ}$ на основі експертної оцінки науково-технічного рівня розробки. З цією метою:

- розроблюють перелік специфічних показників, необхідних для виміру науково-технічного рівня розробки;
- формують групу аналогів, які реалізовані на світовому і вітчизняному ринках;
- здійснюють відповідні розрахунки для співставлення показників і визначення балів по таблиці 4.2.

До числа специфічних показників відносять [10]:

- для нової техніки: продуктивність, споживання інженерних ресурсів на виробітку одиниці продукції, потреба в робочих, які обслуговують обладнання, експлуатаційні витрати на одиницю продукції;
- для нових матеріалів і речовин: вміст корисних речовин для виробітки готової продукції, питома вага відходів у загальному обсязі переробленої сировини, вартість одиниці … нового матеріалу;
- для нових технологій: якість виробленої продукції, енергоємність і трудомісткість продукції, собівартість одиниці продукції.

З метою спрощення визначення $К^{Ф}_{НТЕ}$ у таблиці 4.3 не введено показника витрат на одиницю продукції. Таблиця взята з методичних вказівок [10].

Таблиця 4.3

Порівняльні показники для виконання оцінки НТЕ

Показники	Варіанти технології	
	Розробленої	Співвідносної (аналога)
Рівень новізни	Галузь	Окремі підприємства
Якість продукції	Висока	Вища
Витрати на створення: – енергоспоживання (кВт/год) – трудомісткість (днів) – собівартість (доларів)	 0,93 60 1000	 1,17 100 1500

На основі співставлення даних таблиці встановлюють бали по характеристиках чотирьох груп і на цій основі розраховують значення інтегрального показника НТЕ:

$$НТЕ = \sum Б_i * К_i^3, \quad (4.2)$$

де i = 1 ÷ 4;

Б – бали (рейтингове число);

К – коефіцієнт значущості показників.

Рівень науково-технічної ефективності НДДКР розраховано на основі наведених даних прикладу (табл. 4.4) [10].

Таблиця 4.4

Експертна оцінка і розрахунок величини інтегрального показника НТЕ

№	Групи показників	Рейтинг експертів			Середня за експертними оцінками	НТЕ
		№1	№2	№3		
1	Науково-технічний рівень	8	8	7	7,67	2,68 (7,67 * 0,35)
2	Перспективність	7	5	6	6,0	2,1 (6,0 * 0,35)

Продовження таблиці 4.4

№	Групи показників	Рейтинг експертів			Середня за експертними оцінками	НТЕ
		№1	№2	№3		
3	Потенційний масштаб практичного використання	5	6	6	5,67	1,13 (5,67 * 0,20)
4	Ступінь вірогідності досягнення позитивних результатів	7	6	9	7,33	0,73 (7,33 * 0,10)
					Разом	6,64

$$НТЕ = 7,67 * 0,35 + 6,0 * 0,35 + 5,67 * 0,2 + 7,33 * 0,1 =$$
$$= 2,68 + 2,1 + 1,13 + 0,73 = 6,64$$

Отриманий результат необхідно порівняти з максимально можливим значенням, яке дорівнює 10 балам (10 * 0,35 + 10 * 0,35 + 10 * 0,2 + 10 * 0,1).

Отже, оцінка рівня НТЕ може бути зроблена за допомогою інтегрального коефіцієнта оцінки НТЕ ($К_{НТЕ}$):

$$К_{НТЕ} = \frac{НТЕ}{10} * 100\%, \quad (4.3)$$

На основі даних табл. 4.3 можно дійти до висновку, що $К_{НТЕ}$ відповідає 66,4 %, тобто:

$$\frac{6,64}{10} * 100 = 66,4\ \%$$

Отриманий показник необхідно порівняти із середнім значенням, яке дорівнює 5,0. Якщо $К_{НТЕ}$ менше середнього значення, розробляти продукт недоцільно, так як витрати на розробку не окупляться. В тому випадку, коли значення $К_{НТЕ}$ перевищує середнє значення, має бути зроблено висновок про достатній рівень НТЕ (табл. 4.5). Таблиця розроблена за методичними вказівками [10].

Таблиця 4.5

Шкала оцінок рівня НТЕ технології

Числове значення показника НТЕ	Рівень НТЕ технології
0 – 5	недостатній
5,1 – 6,0	цілком достатній
6,1 – 8,0	достатній
8,1 – 9,0	достатньо високий
9,1 – 10	високий

Таким чином, рівень НТЕ технології можна визнати достатнім. Отже, розроблену технологію пропонується впроваджувати у виробництво.

РОЗДІЛ 5
ОХОРОНА ПРАЦІ

Людина одержує близько 80% інформації саме через зоровий канал, написано у книзі [13]. Освітлення безпосередньо впливає на цей процес. Неякісне освітлення не лише втомлює і псує зір і нервову систему, а й викликає втому організму, зменшує продуктивність людини в цілому. Воно впливає на настрій та здоров'я людей, а також на функціональність та безпеку приміщення. Саме тому дуже важливо підтримувати сприятливі умови освітлення у робочих приміщеннях.

Виробниче освітлення буває трьох видів, в залежності від джерела світла:

- природне. Освітлення приміщень прямим або відбитим світлом неба;
- штучне. Здійснюється штучними джерелами світла, такими як лампи різного походження;
- суміщене. Поєднує одночасне використання природного та штучного освітлення.

Освітлення характеризується кількісними та якісними показниками. До кількісних показників відносяться:

1. Світловий потік (*F*). Променева енергія, яку оцінює людське око за світловим відчуттям. Одиниця світлового потоку – люмен (лм).
2. Сила світла (*I*). Являє собою відношення світлового потоку, що розповсюджується від джерела всередині елементарного тілесного кута, який має в собі цей напрямок, до цього тілесного кута. За одиницю сили світла прийнята кандела (кд).
3. Освітленість (*E*). Величина світлового потоку, який припадає на одиницю освітлюваної поверхні. Вимірюється в люксах (лк).
4. Яскравість (*L*). Відношення сили світла елемента поверхні до проекції, перпендикулярної напряму, що розглядається. Виходячі з формули, яскравість вимірюється в кд/м^2.

В свою чергу до якісних показників належать:

1. Показник осліпленості (*P*). Є критерієм осліплювальної дії освітлювальної установки.

2. Показник дискомфорту (*M*). Критерій оцінювання неприємних віддучів через нерівномірний розподіл яскравості.
3. Коефіцієнт пульсації освітленості (K_n). Характеризує відносну глибину коливань освітленості в результаті змін у часі світлового потоку газорозрядних ламп, які живляться змінним струмом. Вимірюється у відсотках як і усі коефіцієнти.
4. Контраст об'єкта з фоном. Відношення абсолютної величини різниці між яскравістю об'єкта і фона до яскравості фона.

Штучне освітлення включає в себе різноманітні типи освітлювальних пристроїв, які використовуються для надання світла у різних ситуаціях при недостатньому або відсутньому природному освітленні. Як зазначено у книзі [14], це може бути освітлення на робочому місці, аварійне освітлення, а також освітлення для спеціальних потреб. Кожен з цих типів має свої унікальні характеристики і функції.

Робоче освітлення використовується для надання належної видимості на робочих місцях, де необхідно точне бачення деталей або виконання робіт, які вимагають високої концентрації. До прикладів робочого освітлення належать стільницеві лампи, світильники для настільних ламп, світильники для освітлення робочих поверхонь тощо. Вони забезпечують яскраве, насичене світло, спрямоване безпосередньо на робочу область, забезпечуючи оптимальні умови для виконання завдань.

Аварійне освітлення встановлюється з метою забезпечення безпеки і орієнтації людей у випадку аварії або відключення основного джерела освітлення. Це може бути необхідним в будівлях, таких як офіси, готелі, торгові центри, лікарні, аеропорти тощо. Аварійне освітлення зазвичай включає в себе автономні світильники з резервним джерелом живлення, таким як акумулятори. Вони активуються автоматично, коли основне освітлення відключається, і надають достатню кількість світла для безпечного евакуювання з будівлі.

Спеціальне освітлення використовується для конкретних потреб і вимог. Це може бути освітлення в музеях для підсвічування цінних експонатів, освітлення на сценах для театральних вистав, освітлення в студіях для фотографії або відеозйомки, освітлення в лабораторіях для спеціальних досліджень тощо. Спеціальне освітлення

зазвичай має специфічні характеристики, такі як можливість регулювання яскравості, кольору світла або напрямку освітлення, щоб забезпечити оптимальні умови для конкретних завдань.

Кожен з цих типів штучного освітлення має свою важливість і використовується залежно від конкретних потреб. Робоче освітлення допомагає покращити продуктивність і забезпечити комфортні умови праці. Аварійне освітлення є життєво важливим у випадку надзвичайних ситуацій, забезпечуючи безпеку та орієнтацію людей. Спеціальне освітлення дозволяє досягнути оптимальних результатів у виконанні специфічних завдань та створити необхідну атмосферу.

Лампи розжарювання і газорозрядні лампи довгий час були двома типами штучного освітлення, які використовувалися для освітлення приміщень. Найбільш поширеними газорозрядними лампами є люмінесцентні. Але останнім часом все більше ці два типи замінюють світлодіодними лампами, адже останні мають багато переваг, зазначено у статті [15].

Одна з головних переваг світлодіодних ламп, порівняно з газорозрядними, полягає у їх екологічності. Ще одним важливим позитивним аспектом нового типу ламп є відсутність помітного мерехтіння, що характерне для люмінесцентних джерел світла. Це дає можливість використовувати світлодіодні лампи для освітлення рухомих механізмів та інших місць, де використання люмінесцентних ламп може бути небезпечним або спричиняти швидку втомлюваність очей. Світлодіодні лампи живляться постійним струмом, тому вони не мерехтять.

Крім цього, сучасні світлодіодні лампи мають багато інших переваг. Єдиним недоліком світлодіодних ламп є їхня висока ціна порівняно з іншими типами ламп. Загальне порівняння цих типів ламп наведено в таблиці 5.1.

Таблиця 5.1

Порівняння основних типів ламп для штучного освітлення

Характеристика	**Лампи розжарювання**	**Газорозрядні лампи**	**Світлодіодні лампи**
Принцип роботи	Нагрівання нитки або спіралі	Газовий розряд	Електролюмінесценція
Ефективність	Низька	Середня	Висока
Термін служби	Середній	Короткий	Тривалий
Яскравість і колірна температура	Тепле приємне світло	Висока яскравість, висока колірна температура	Спектр від теплого білого до холодного білого
Вартість	Низька	Середня	Висока
Використання	Декоративне освітлення	Вуличне освітлення, рекламні табло, автомобільні фари	Побутове освітлення, офісне освітлення, промислове освітлення
Споживання енергії	Високе	Високе	Низьке
Стабільність світла	Поступове зменшення	Відносно стабільне	Висока стабільність

Для забезпечення сприятливих умов зорової роботи нормують мінімальну освітленість на найтемнішій ділянці робочої поверхні. При штучному освітленні нормуються абсолютне значення освітленості, а також якісні показники. Для України ці норми прописані у Державних будівельних нормах (ДБН В.2.5-28:2018).

Для визначення оптимального освітлення існують формули розрахунку. Для розрахунку штучного освітлення існують наступні методи:

– метод коефіцієнта використання світлового потоку. Використовується для розрахунку загального рівномірного освітлення при горизонтальній робочій поверхні;

– точковий метод. Використовується для розрахунку локалізованого і комбінованого освітлення, освітлення похилих та вертикальних площин і для перевірки розрахунку рівномірного загального освітлення, коли відбитим світловим потоком можна знехтувати. Суть методу полягає в визначенні освітленності точки світловим потоком, який падає від випромінювача світла;
– метод питомої потужності. Розрахунок за потужністю є найпростішим, але і найменш точним, тому його використовують лише для орієнтовних розрахунків. Питома потужність – це відношення сумарної потужності джерел світла до площі поверхні, що освітлюється.

Наприклад, необхідно розрахувати освітлення для комп'ютерного класу. В такому випадку використовується метод коефіцієнта використання світлового потоку. Його формула має наступний вигляд:

$$F = \frac{E_{\text{н}} S K z}{\mu}, \tag{5.1}$$

де *Ен* – нормована мінімальна освітленість;

S – площа приміщення;

K – коефіцієнт запасу;

z – коефіцієнт нерівномірності освітлення;

μ – коефіцієнт використання світлового потоку ламп.

Згідно ДБН [16], нормована мінімальна освітленість при загальному освітленні для кабінетів та офісів складає 300 лк. Розміри класу складають 8х6 м, отже площа класу (*S*) складає 48 м2. Коефіцієнт запасу (*K*) враховує зниження освітленості в процесі експлуатації в результаті зменшення світлового потоку джерела світла в процесі горіння, зниження ККД світильників у результаті забруднення стін і стелі приміщення. В даному прикладі значення коефіцієнту запасу дорівнює 1,5. Коефіцієнт нерівномірності освітлення (*z*) визначається відношенням середньої освітленості до мінімальної, в даному випадку взято значення 1,2. Коефіцієнт використання світлового потоку ламп (μ) визначає ефективність перетворення електричної енергії, споживаної лампою, в світлову енергію, яка випромінюється. В даному випадку цей коефіцієнт буде складати 56%.

Підставивши значення у формулу, можна визначити, що загальний світловий потік для обраного комп’ютерного класу складає 46286 лм:

$$\frac{300 * 48 * 1{,}5 * 1{,}2}{0{,}56} = \frac{25920}{0{,}56} = 46286$$

Для освітлення цього приміщення планується використовувати люмінесцентні лампи типу ЛБ-40. Світловий потік однієї такої лампи ($F_л$) складає 2850 лм. Для того, щоб визначити кількість таких ламп, необхідних для досягнення загального світлового потоку в 46286 лм, необхідно поділити загальний світловий потік на світловий потік лампи:

$$n = \frac{F}{F_л}, \tag{5.2}$$

$$\frac{46286}{2850} = 16.24$$

Так як не можна встановити лампу частково, отримане значення необхідно округлити до найближчого верхнього цілого числа. Отже, для досягнення загального світлового потоку в 46286 лм, потрібно встановити 17 ламп типу ЛБ-40.

В наш час для розрахунку освітленості широко використовуються спеціалізовані комп’ютерні програми, що симулють розподіл світла і відтворюють результат в об’ємному графічному представленні. Найпопулярнішим прикладом такого типу програм є безкоштовна програма *DIALux*.

Для фактичного вимірювання освітленості будь-якого походження використовують прилад люксметр. Існують різні типи люксметрів, від простих ручних до більш складних, які можуть бути підключені до комп'ютера та зберігати дані про рівень освітлення протягом тривалого періоду часу. Деякі моделі можуть мати інші функції, наприклад, вимірювання температури або вологості повітря.

ЗАГАЛЬНІ ВИСНОВКИ

У даній дипломній роботі було розглянуто проблему створення багатьох однотипних документів та напрямок розробки програмного забезпечення, що займається вирішенням даної проблеми – автоматизація документообігу, а також проаналізовано актуальність цього напрямку.

Шаблонізатори являють собою зручний інструмент, який дозволяє значно скоротити час та полегшити процес створення великої кількості однотипних документів за допомогою шаблонів, що є актуальною проблемою у наш час. Використання шаблонізаторів може бути особливо корисним для багатьох організацій, що часто працюють зі стандартними документами (наприклад, звіти, договори тощо). Зокрема, це може бути корисно для фінансових установ, юридичних фірм, медичних установ, учбових закладів та інших організацій.

Було розглянуто шаблонізатори в цілому та проаналізовано існуючі рішення, на основі чого було виявлено основні задачі шаблонізаторів та певні проблеми існуючих рішень. В результаті було сформовано вимоги до проекту.

На основі проведеного аналізу і сформованих вимог, було розроблено простий та зручний алгоритм взаємодії користувача з системою за допомогою веб-інтерфейсу (сайту). Далі було розроблено архітектуру проекту та обрано технології для реалізації розроблених алгоритмів, після чого було описано загальний процес реалізації.

В результаті дипломної роботи було отримано веб-застосунок, який дозволяє підставляти значення з бази даних у відповідні місця шаблону *DOCX*, тим самим автоматизуючи процес створення документів на основі шаблонів.

Розроблений продукт було протестовано методом ручного тестування. Тестування показало, що застосунок виконує задані задачі і відповідає поставленим вимогам.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. *Mehul Rajput. Web Application Architecture: Everything You Need to Know About.* [Електронний ресурс] // *MindInventory*, 2021. – Режим доступу: *mindinventory.com/blog/web-application-architecture/*
2. *Document automation – Wikipedia*. [Електронний ресурс] // *The Free Encyclopedia*. – Режим доступу: *en.wikipedia.org/wiki/Document_automation*
3. *Model-View-Controller – Wikipedia*. [Електронний ресурс] // *The Free Encyclopedia*. – Режим доступу: *ru.wikipedia.org/wiki/Model-View-Controller*
4. *Manish Sharma. What is Spring security*. [Електронний ресурс] // *Java Development Journal*, 2021. – Режим доступу: *javadevjournal.com/spring/what-is-spring-security/*
5. *Thymeleaf – Wikipedia*. [Електронний ресурс] // *The Free Encyclopedia*. – Режим доступу: *en.wikipedia.org/wiki/Thymeleaf*
6. *JPA Object Relational Mapping – Javatpoint*. [Електронний ресурс] // *Javatpoint*. – Режим доступу: *javatpoint.com/jpa-object-relational-mapping*
7. Переваги мови програмування *Java* – Новини Полтавщини. [Електронний ресурс] // Новини Полтавщини, 2021. – Режим доступу: *np.pl.ua/2021/03/perevahy-movy-prohramuvannia-java/*
8. *Understanding Spring modules – Packt*. [Електронний ресурс] // Онлайн бібліотека *Pacht*. – Режим доступу: *subscription.packtpub.com/book/programming/9781788838382/1/ch01lvl1sec03/understanding-spring-modules*
9. *Types of Software Testing – GeeksforGeek*. [Електронний ресурс] // Спілка програмістів *GeeksforGeek*. – Режим доступу: *geeksforgeeks.org/types-software-testing/*
10. Методичні вказівки до оцінки науково-технічної ефективності розробки нової технології, нового обладнання та інших інновацій. Для студентів всіх спеціальностей СВО «бакалавр» і «магістр» денної і заочної форм навчання. Укладачі Басюркіна Н.Й., Свистун Т.В. Одеса: ОНТУ, 2023 р. 18 с.
11. Зарплати українських розробників – *DOU* [Електронний ресурс] // *DOU* – спільнота програмістів. – Режим доступу: *dou.ua/lenta/articles/salary-report-devs-winter-2023/*

12. І. Самоходський, О. Шелест. Зелена книга «Регулювання ринку розробки програмного забезпечення», 2017. [Електронний ресурс]. – Режим доступу: *cdn.regulation.gov.ua/af/2a/db/8b/regulation.gov.ua_Green%20Book_Software%20Development%20Market_BRDO.pdf*
13. Катренко Л. А., Катренко А. В. «Охорона праці в галузі комп'ютингу» – Львів, 2012. – 544 с.
14. Я. І. Бедрій. «Безпека життєдіяльності» – Київ: Кондор, 2009. – 286 с.
15. «Переваги світодіодних ламп в порівнянні з іншими джерелами світла» – Івано-Франківськміськсвітло. [Електронний ресурс]. – Режим доступу: *misksvitlo.if.ua*/переваги-світодіодних-ламп-в-порівня/
16. Мінрегіон України. «Державні будівельні норми України: природне і штучне освітлення» (ДБН В.2.5-28:2018). Офіційне видання. – Київ, 2018. – 133 с.