МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«РОЗРОБКА СЕРВІСУ ГЕНЕРАЦІЇ СЕРТИФІКАТІВ»**

Виконав: студент 2 курсу, групи 8.1212-іпз-1
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми інженерія програмного забезпечення
(назва освітньої програми)
В.Ю. Єфіменко
(ініціали та прізвище)

Керівник декан математичного факультету, професор, д.т.н. Гоменюк С.І.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент завідувач кафедри фундаментальної та прикладної математики, професор, д.т.н. Гребенюк С.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти магістр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Єфіменку Владиславу Юрійовичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка сервісу генерації сертифікатів

керівник роботи Гоменюк Сергій Іванович, д.т.н., професор
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 01 » травня 2023 року № 642-с

2. Строк подання студентом роботи 27.11.2023 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі, аналіз предметної області.
2. Проєктування.
3. Реалізація та тестування.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою докладу

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 03.05.2023 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	17.05.2023	
2.	Збір вихідних даних.	07.06.2023	
3.	Обробка методичних та теоретичних	28.06.2023	
	джерел.		
4.	Розробка першого та другого розділу.	30.08.2023	
5.	Розробка третього розділу.	08.11.2023	
6.	Оформлення та нормоконтроль	20.11.2023	
	кваліфікаційної роботи магістра.		
7.	Захист кваліфікаційної роботи.	14.12.2023	

Студент _____ В.Ю. Єфіменко
(підпис) (ініціали та прізвище)

Керівник роботи _____ С.І. Гоменюк
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Розробка сервісу генерації сертифікатів»: 56 с., 26 рис., 11 джерел, 5 додатків.

ANGULAR, API, FRAMEWORK, MVC, NODEJS, RXJS, TYPESCRIPT, UML.

Об’єкт дослідження – система, інструменти для взаємодії Angular та Node.js, засоби взаємодії системи з користувачем.

Мета роботи – розробити сервіс генерації сертифікатів.

Методи дослідження – моделювання, проєктування, програмний, аналітичний.

Сучасний світ, в якому ми живемо, вимагає від нас постійного вдосконалення та розвитку. В освіті та бізнесі сертифікати стали невід’ємною частиною документації та визнання компетенції. Сертифікати доводять нашу кваліфікацію та можуть відкривати двері до нових можливостей. У зв’язку з цим актуальність розробки сервісу генерації сертифікатів надзвичайно висока.

Як правило, подібні системи мають різний функціонал, який охоплює різні аспекти взаємодії з сертифікатами. Але відсутність або обмеженість потрібного функціоналу не є виключенням. Одним з прикладів такої проблеми є дороговизна подібних сервісів.

Таким чином, за результатами роботи створено зручний та ефективний сервіс генерації сертифікатів з використанням Angular та Node.js.

SUMMARY

Master’s qualifying paper «Development Service for Certificate Generation»: 56 p., 26 figures, 11 references, 5 supplements.

ANGULAR, API, FRAMEWORK, MVC, NODEJS, RXJS, TYPESCRIPT, UML.

The object of the study is a system, tools for interaction between Angular and Node.js, means of interaction between the system and the user.

The aim of the study is to develop a certificate generation service.

The methods of research are modeling, design, software, analytical.

The modern world we live in requires us to constantly improve and develop. In education and business, certificates have become an integral part of documentation and recognition of competence. Certificates prove our qualifications and can open doors to new opportunities. In this regard, the relevance of developing a certificate generation service is extremely high.

As a rule, such systems have different functionalities that cover different aspects of interaction with certificates. However, the absence or limited functionality is not an exception. One example of this problem is the high cost of such services.

Thus, based on the results of our work, we have created a convenient and efficient certificate generation service using Angular and Node.js.

ЗМІСТ

ВСТУП

Сучасні користувачі очікують зручності та доступності в усьому. Онлайн-сервіси для генерації сертифікатів надають змогу отримувати або перевіряти свої сертифікати в будь-який час та в будь-якому місці. Це зробить процес отримання та користування сертифікатами більш зручним і доступним.

За допомогою спеціалізованого сервісу можна забезпечити надійність та автентичність виданих сертифікатів. Це важливо, оскільки багато сертифікатів може використовуватися для доказування кваліфікації перед роботодавцями або іншими зацікавленими сторонами. Застосування криптографічних методів та систем безпеки дозволить уникнути підробки сертифікатів.

Цей сервіс спростить процес видачі та користування сертифікатами, забезпечить їх надійність та автентичність, покращить зручність для користувачів та сприятиме збереженню природних ресурсів. Розробка такого сервісу може мати значущий вплив на освіту, бізнес та суспільство в цілому, роблячи процес отримання та використання сертифікатів більш сучасним та зручним.

Виходячи з цього, було вирішено створити сервіс, котрий би мав зрозумілий інтерфейс та надавала гнучкий функціонал користувачам.

Актуальність дослідження: актуальність теми зумовлена необхідністю спрощення процесу видавання сертифікатів, а також зменшення обсягу друкованої документації.

З огляду на це, можна виділити наступні цілі і задачі нашого дослідження:

Мета: розробити сервіс генерації сертифікатів.

Задачі:

– сформувати вимоги до сервісу;
– спроєктувати та побудувати архітектуру сервісу;
– реалізувати сервіс генерації сертифікатів;
– протестувати роботу сервісу.

Об’єкт дослідження: процес генерації сертифікату користувачем, інструменти для реалізації сервісу генерації сертифікатів, функціонал системи, необхідний користувачам.

Предмет дослідження: створення сервісу, що дозволяє користувачам генерувати сертифікати.

Методи дослідження: моделювання, проєктування, програмний, аналітичний.

Перший розділ присвячено збору та аналізуванню вимог до сервісу, огляду інструментів розробки та опису сервісу.

У другому розділі розглянуто етапи проєктування сервісу, наведено детальний опис прецедентів.

Третій розділ присвячено реалізації та тестуванню роботи сервісу, наведено детальне керівництво користувача, яке описує процес роботи з сервісом генерації сертифікатів.

1 ТЕХНІЧНЕ ЗАВДАННЯ

1.1 Терміни та визначення

1.1.1 Загальні терміни

Система – сервіс генерації сертифікатів.

Angular – це фреймворк для створення вебдодатків, який базується на TypeScript. Angular дозволяє створювати потужні та масштабовані додатки з використанням компонентної архітектури та забезпечує розробникам ряд інструментів для зручної роботи.

Firebase – це облачна платформа, розроблена компанією Google, яка надає набір інструментів і сервісів для розробки вебдодатків для мобільних пристроїв і вебплатформ.

Node.js – це відкрита серверна платформа для виконання JavaScript коду на стороні сервера.

ДВІ – Діаграма Варіантів Використання чи Use Case Diagram.

ДД – Діаграма Діяльності.

ДП – Діаграма Послідовності.

Користувач – людина, котра зареєстрована у системі та може взаємодіяти з функціоналом.

1.1.2 Технічні терміни

БД – база даних, місце збереження інформації системи.

MVC – архітектурний шаблон, який використовується під час проєктування та розробки програмного забезпечення.

1.2 Функціональні вимоги

1.2.1 Призначення і цілі створення системи

Функціональне призначення системи – реалізувати можливість генерації сертифікатів.

Експлуатаційне призначення системи: система може експлуатуватися користувачем системи.

Мета створення системи – розробка сервісу генерації сертифікатів.

1.2.2 Загальні функціональні можливості системи

Система має надавати користувачам такі можливості:

– створення / редагування / перегляд / видалення / вибір шаблону сертифікату;
– завантаження даних для генерації за шаблоном;
– створення/редагування/перегляд/видалення/завантаження згенерованих сертифікатів;
– вхід/вихід до/з системи.

1.3 Нефункціональні вимоги

Інтерфейс користувача: система повинна відображати коректно інтерфейс користувача на будь-якому пристрої.

Підтримка браузерів: система повинна працювати для наступних браузерів останніх версій: Mozilla Firefox, Google Chrome, Safari, Microsoft Edge, Opera.

Вимоги до продуктивності: система повинна відображати будь-яку сторінку не довше, ніж за 2 секунди. Система повинна генерувати кожен

сертифікат не довше, ніж 10 секунд.

Вимоги до безпеки: система не повинна надавати доступ неавторизованим користувачам доступ до функціоналу системи.

1.4 Опис предметної області

Предметною областю є розробка сервісу генерації сертифікатів.

Дана система повинна надавати користувачам можливість згенерувати сертифікати, обравши шаблон та дані.

Процес взаємодії з системою проходить наступним чином.

Користувач, повинен ввести адресу сайту в браузері та ввести дані в форму входу до системи. Якщо користувач вперше на сайті, він повинен спочатку зареєструватися.

Після входу до системи користувач має можливість взаємодіяти з такими розділами як: шаблони, сертифікати.

Перед початком генерації сертифікату/ів, користувач повинен створити перший шаблон та завантажити дані.

Процес генерації сертифікату/ів проходить наступним чином. Користувач обирає розділ «Шаблони», система відображає інтерфейс взаємодії з даним розділом.

Для створення нового шаблону користувач натискає кнопку «Створити», система відображає форму побудови шаблону.

Після створення, користувач натискає на кнопку «Зберегти», система зберігає шаблон.

Далі користувач обирає розділ «Сертифікати», система відображає інтерфейс взаємодії з даним розділом.

Користувач обирає шаблон, завантажує дані та натискає на кнопку «Генерація», система завантажує сертифікати на пристрій користувача.

1.5 Опис системи

Сучасний світ, в якому ми живемо, вимагає від нас постійного вдосконалення та розвитку. В освіті та бізнесі сертифікати стали невід'ємною частиною документації та визнання компетенції. Сертифікати доводять нашу кваліфікацію та можуть відкривати двері до нових можливостей.

Ручна генерація сертифікатів може бути часовою та ресурсозатратною задачею. Особливо це стосується організацій, які регулярно видають сертифікати, такі як навчальні заклади або компанії зі стажуванням. Розробка сервісу генерації сертифікатів дозволить автоматизувати цей процес і зекономити час та ресурси.

У зв'язку з цим актуальність розробки сервісу генерації сертифікатів надзвичайно висока.

Можливості системи:

– керування шаблонами;
– використання набору даних користувача;
– генерація сертифікатів.

1.6 Огляд інструментів розробки

1.6.1 Angular

Angular – це потужний та високопродуктивний фреймворк для розробки вебдодатків з використанням HTML, CSS та TypeScript. Angular заснований на принципах компонентної архітектури, де кожен елемент вебдодатку представлений компонентом, який містить свою логіку та представлення. Компоненти Angular є самодостатніми, що дозволяє їх використовувати в різних частинах додатку, що підвищує його масштабованість та підтримку [6].

Розглянемо нижче основні переваги фреймворку Angular [2].

Декларативність. Angular використовує декларативні шаблони для відображення даних, що дозволяє розробникам зосередитися на функціональному програмуванні та забезпечити легку та читабельну кодову базу.

Широкі можливості. Angular надає вбудовані директиви, такі як *ngIf та *ngFor, які дозволяють керувати відображенням даних на сторінці. Крім того, Angular надає розробникам можливість використовувати підключені бібліотеки, такі як RxJS, для роботи з асинхронними операціями та роботи з потоками даних.

Ін'єкція залежностей. Angular має потужну систему ін'єкції залежностей, яка дозволяє забезпечити легку тестовість додатку та збільшити його модульність. Розробникам не потрібно прямо включати залежності в клас, вони можуть бути включені за допомогою ін'єкції.

Маршрутизація. Angular має потужну систему маршрутизації, яка дозволяє розробникам змінювати сторінки без перезавантаження сторінки. Це дозволяє забезпечити користувачам більш зручний та швидкий досвід взаємодії з додатком.

Підтримка мобільних пристроїв. Angular дозволяє розробникам створювати мобільні додатки з використанням іонів (Ionic), який є фреймворком для розробки гібридних мобільних додатків на базі Angular.

Узагальненість. Angular дозволяє розробникам створювати універсальні додатки, які можуть працювати як на клієнтському боці, так і на серверному. Це дозволяє розробникам забезпечувати більшу кількість можливостей для оптимізації додатку та його підтримки.

Angular є популярним вибором для розробки вебдодатків, особливо для більш складних та масштабних проєктів. За допомогою Angular розробники можуть створювати високопродуктивні додатки, які мають легку та читабельну кодову базу, а також підтримку мобільних пристроїв та інші переваги, які забезпечують більш зручний та ефективний досвід взаємодії з додатком.

1.6.2 Node.js

Node.js – це вільне та відкрите серверне середовище виконання JavaScript, яке дозволяє розробникам виконувати код JavaScript на стороні сервера. Ця технологія стала популярною завдяки своїм унікальним особливостям та перевагам. Наведемо деякі з них [10].

Асинхронний та подійно-орієнтований підхід. Node.js побудований на асинхронному програмуванні, що дозволяє виконувати багатозадачні операції без блокування потоку виконання. Це забезпечує високу продуктивність та швидкодію додатків, особливо в умовах великої навантаженості.

Широкий вибір модулів та пакетів. Node.js має велику кількість готових модулів та бібліотек, які роблять розробку швидше та простіше. Наявність Node Package Manager (NPM) дозволяє розробникам легко інтегрувати сторонні бібліотеки в свій проєкт.

Швидкість виконання. Node.js використовує рушій Chrome V8, який є дуже швидким та ефективним. Це робить Node.js ідеальним вибором для створення високопродуктивних вебдодатків, особливо під час роботи з багатьма одночасними запитами.

Спільна мова на сервері та клієнті. Використання JavaScript як мови програмування як на стороні клієнта, так і на стороні сервера, спрощує розробку та обмін кодом між фронтендом і бекендом.

Можливість створення в реальному часі. Node.js ідеально підходить для розробки вебдодатків, які вимагають реального часу, таких як чати, ігри, стрімінгові платформи та інші додатки, які взаємодіють з користувачами без затримок.

Активна спільнота та підтримка. Node.js має велику та активну спільноту розробників, що сприяє постійному розвитку та оновленням. Це гарантує актуальність та підтримку технології.

Підтримка мікросервісної архітектури. Node.js часто використовується для розробки мікросервісів, що дає змогу розбити додаток на менші, незалежні компоненти, що спрощує розробку та масштабування.

1.6.3 Firebase

Firebase – це інтегрована платформа від Google для розробки мобільних та вебдодатків. Вона надає різноманітні інструменти та сервіси, які полегшують розробку, тестування, розгортання та управління додатками.

Наведемо, що включають основні можливості Firebase [11].

База даних реального часу. Firebase має базу даних реального часу, яка дозволяє синхронізувати дані в реальному часі між різними клієнтами. Це особливо корисно для мобільних та вебдодатків, які потребують миттєвого оновлення інформації.

Аутентифікація користувачів. Firebase надає можливості для автентифікації користувачів з використанням різних провайдерів, таких як Google, Facebook, Twitter, та інші.

Хостинг та розгортання. Ви можете легко розгортати свій вебсайт або мобільний додаток на Firebase. Firebase Hosting надає швидке та безкоштовне хостингове рішення.

Cloud Functions. За допомогою Cloud Functions ви можете написати серверний код, який викликається подіями Firebase, такими як зміни в базі даних або завантаження нового зображення.

Cloud Firestore. Це документ-орієнтована база даних, яка дозволяє зберігати та запитувати дані вашого додатку.

Cloud Messaging. Для надсилання повідомлень на мобільні пристрої.

Analytics: Firebase надає інструменти для аналізу використання додатків, включаючи відстеження подій, залучення користувачів та інше.

Інструменти тестування та випробування. Firebase надає інструменти для тестування додатків, включаючи A/B-тестування та тестування в реальному часі.

Firebase є популярним вибором для розробників завдяки своїй простоті використання та різноманіттю сервісів, які вона надає для розробки та управління додатками.

2 ПРОЄКТУВАННЯ

2.1 Використання UML під час розробки системи

UML є стандартною мовою для моделювання систем, яка надає зручний і загальний спосіб представлення системних архітектур, процесів та структурних відносин. Використання UML дозволяє команді розробників спілкуватися із зацікавленими сторонами, визначати вимоги до системи та створювати однорідні моделі, які можуть бути легко зрозумілі як розробникам, так і замовникам.

Однією з основних переваг використання UML є здатність створювати графічні моделі, які ілюструють систему. Завдяки цьому, команда розробників може краще розуміти структуру системи, її компоненти, взаємодію та процеси. Це допомагає зменшити непорозуміння та помилки під час розробки, оскільки всі учасники проєкту мають спільне бачення системи.

Крім того, UML надає можливість створення різних видів діаграм, таких як діаграми класів, діаграми послідовності, діаграми активності та інші. Кожен тип діаграми призначений для конкретної задачі під час розробки. Наприклад, діаграми класів дозволяють моделювати структуру системи та класи, діаграми послідовності допомагають узгоджувати порядок виконання операцій, а діаграми активності – моделювати процеси та потоки даних. Такий підхід дозволяє команді розробників використовувати належний тип діаграми для кожного конкретного завдання, підвищуючи ефективність та точність розробки.

З використанням UML також можна легко створити документацію системи. Багато інструментів, призначених для моделювання UML, автоматично генерують текстову документацію на основі моделей. Це значно спрощує процес документування системи та підтримує її актуальність протягом усього циклу розробки.

Ще однією важливою перевагою використання UML є підтримка інтеграції з іншими інструментами розробки, такими як CASE-системи (Computer-Aided

Software Engineering), які допомагають автоматизувати та полегшити процеси розробки системи. Інтеграція UML з CASE-системами дозволяє автоматично генерувати код, проводити аналіз якості, виконувати тестування та інші завдання.

UML є важливим кроком для успішної розробки системи. Ця методологія надає зручні інструменти для моделювання системи, комунікації між учасниками проєкту та підтримки документації. Вона також сприяє підвищенню точності та ефективності розробки. Тому використання UML є необхідною складовою для будь-якого проєкту розробки програмного забезпечення, який має на меті досягти успіху.

2.2 Діаграма варіантів використання

Діаграма варіантів використання (Use Case Diagram) – це один з основних видів діаграм в рамках Unified Modeling Language (UML), який використовується для моделювання функціональності системи з точки зору її користувачів та їхніх можливих дій.

Діаграма варіантів використання допомагає ідентифікувати основні функціональні вимоги до системи та визначити, як користувачі взаємодіють із системою. Вона відображає варіанти використання або сценарії, які описують, як користувачі взаємодіють з системою та як система реагує на їхні дії.

Основні елементи діаграми варіантів використання включають в себе: акторів, варіанти використання та відносини.

Актори – зовнішні сутності або ролі, які взаємодіють із системою. Це можуть бути реальні користувачі, інші системи, або навіть інші частини системи. Актори представлені графічними об’єктами, які зазвичай зображуються у верхній частині діаграми.

Варіанти використання (Use Cases) – функціональність системи, яка відображає те, що система може робити для акторів. Кожен варіант використання

описує конкретний сценарій взаємодії між акторами та системою. Вони представлені графічними об’єктами, які розташовані в середній частині діаграми.

Відносини (Associations) – відносини між акторами та варіантами використання показують, які актори беруть участь у конкретних сценаріях взаємодії. Вони представлені лініями, які з’єднують акторів та варіанти використання.

Діаграма варіантів використання використовується для досягнення кількох важливих цілей:

– уточнення вимог до системи: вона допомагає команді розробників краще розуміти, як користувачі бажають взаємодіяти з системою та які функції їм потрібні;
– комунікація зі зацікавленими сторонами: діаграми варіантів використання є зрозумілими та легкими в інтерпретації для не-технічних учасників проєкту, таких як замовники або менеджери (вони допомагають уникнути непорозумінь та сприяють комунікації між всіма сторонами проєкту);
– основа для подальших етапів розробки: діаграми варіантів використання можуть слугувати основою для подальших етапів розробки, таких як проєктування архітектури системи та розробка коду (вони допомагають розробникам легше розуміти потреби користувачів та визначити, як система має працювати).

Однак, важливо зазначити, що діаграма варіантів використання не є вичерпною та деталізованою моделлю системи. Вона слугує вихідним пунктом для подальшого аналізу та проєктування. Для отримання повної картини системи можуть бути потрібні інші діаграми UML, такі як діаграма класів, діаграма послідовностей та діаграма станів.

На рисунку 2.1 представлена діаграма варіантів використання.

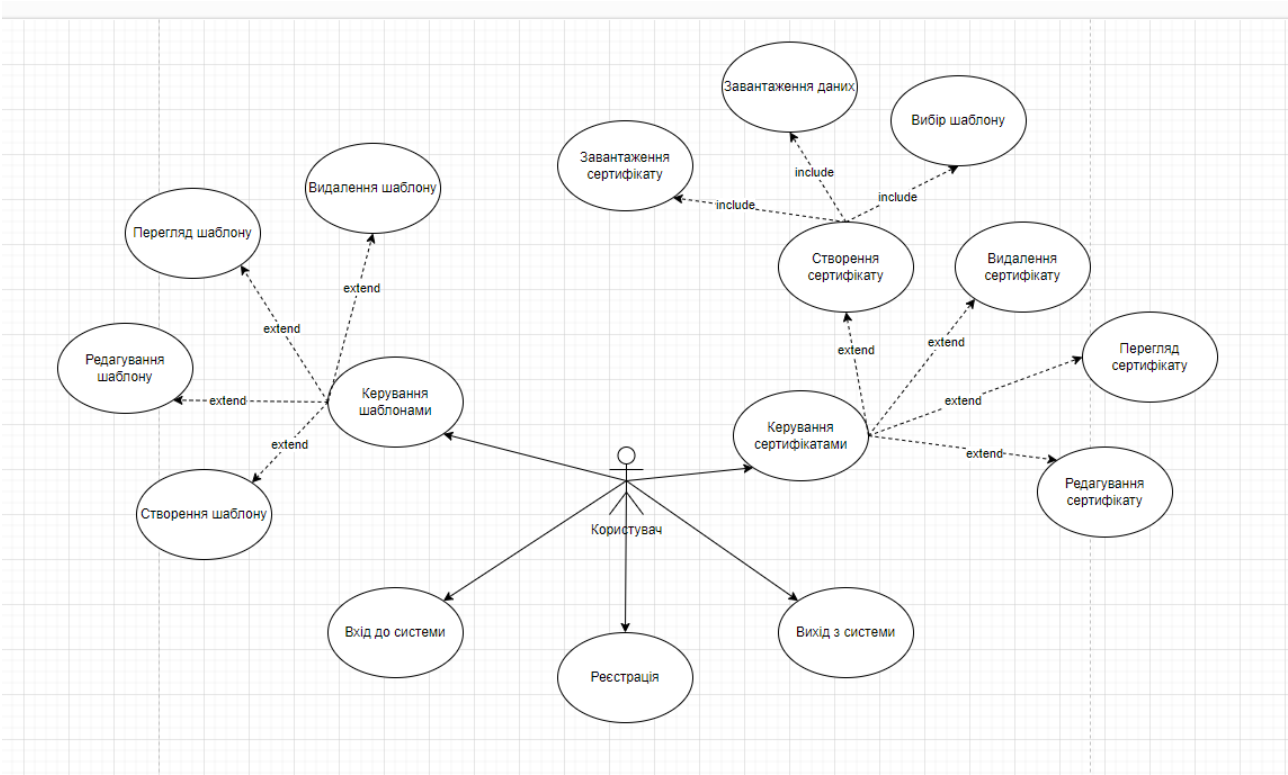

Рисунок 2.1 – Діаграма варіантів використання

На діаграмі представлено актора «Користувач», який безпосередньо є користувачем системи.

Виділено 1 основний варіант використання – «Створення сертифікату». Після того, як користувач авторизується у системі та перейде у розділ «Шаблони», система відображає інтерфейс взаємодії з даним розділом. Для створення нового шаблону користувач натискає кнопку «Створити», система відображає форму побудови шаблону. Після створення, користувач натискає на кнопку «Зберегти», система зберігає шаблон. Далі користувач обирає розділ «Сертифікати», система відображає інтерфейс взаємодії з даним розділом. Користувач обирає шаблон, завантажує дані та натискає на кнопку «Генерація», система завантажує сертифікати на пристрій користувача. На кожну з цих дій система посилає запит до серверу і повідомляє користувача про результат дії.

Варіанти використання визначають функціональні можливості. Кожен з них представляє певний спосіб використання. Таким чином, кожен варіант використання відповідає послідовності дій для того, щоб користувач міг отримати певний результат (див. рис. 2.2, 2.3).

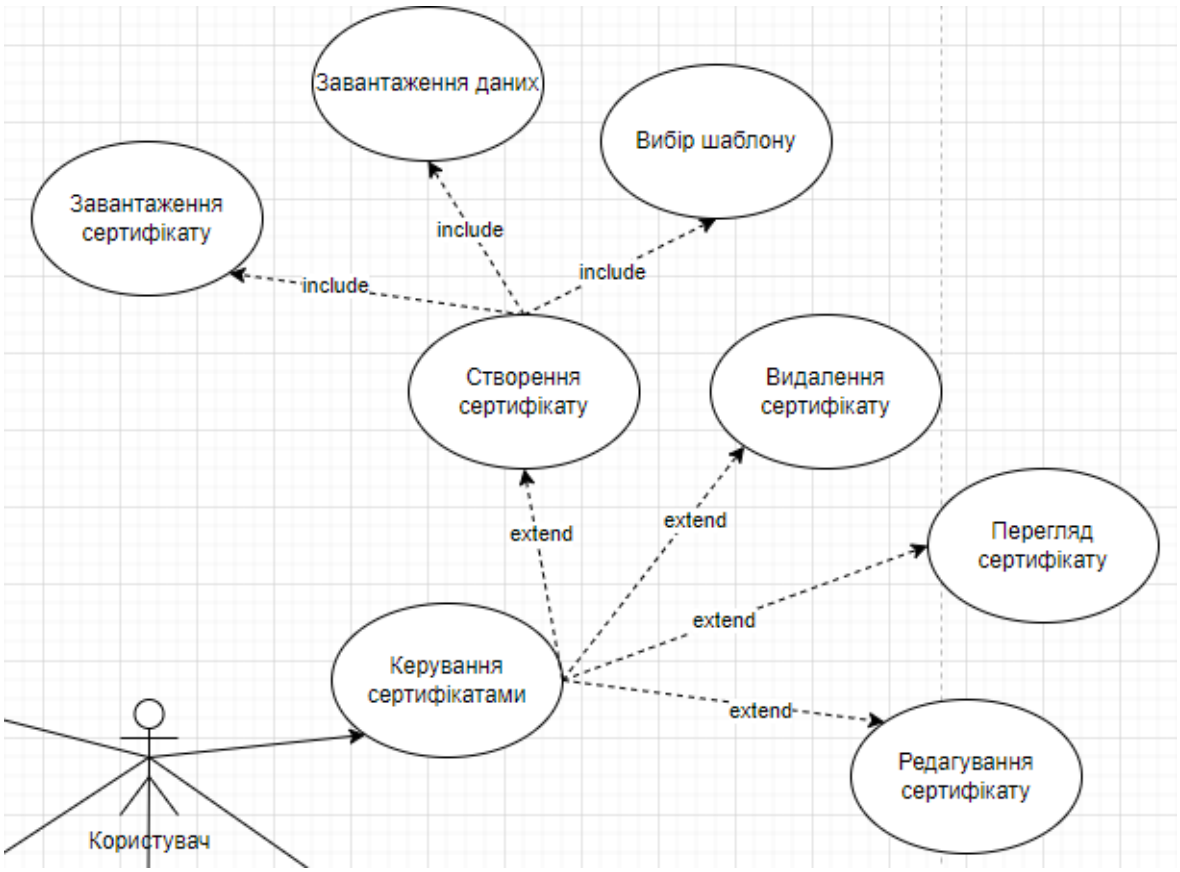

Рисунок 2.2 – Діаграма варіантів використання «Керування сертифікатами»

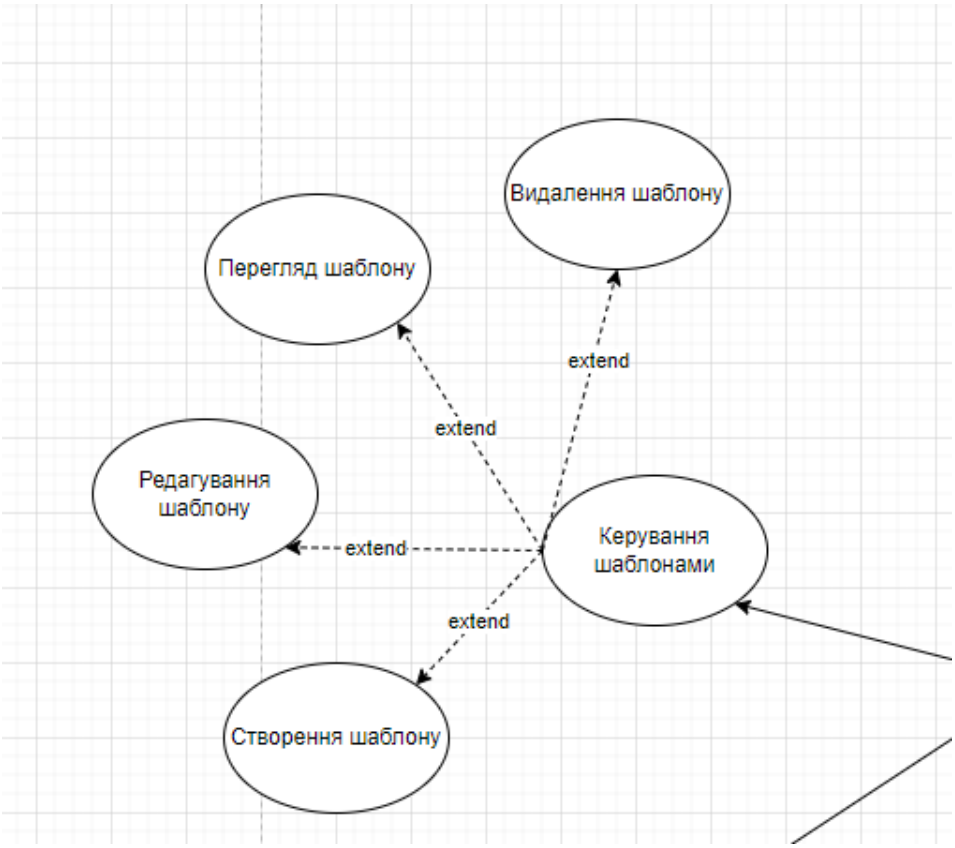

Рисунок 2.3 – Діаграма варіантів використання «Керування шаблонами»

2.2.1 Опис варіантів використання

Прецедент «Реєстрація»

Призначення: даний варіант використання надає можливість користувачу зареєструватися в системі.

Основний потік подій: даний варіант використання починає виконуватися, коли клієнт натискає на кнопку «Зареєструватися». Система відображає форму реєстрації, користувач вводить дані. Якщо введені дані валідні, то система створює новий обліковий запис та переадресовує на сторінку налаштування профілю.

Виняткова ситуація 1: користувач ввів невалідні дані – система відображає відповідне повідомлення. Користувач може повторно ввести дані.

Виняткова ситуація 2: користувач зареєстрований у системі – система відображає відповідне повідомлення та переадресовує користувача на сторінку входу до системи.

Виняткова ситуація 3: користувач натиснув кнопку «Скасувати» – переадресовує користувача на сторінку входу до системи.

Прецедент «Вхід до системи»

Призначення: даний варіант використання надає можливість зареєстрованому користувачу увійти у систему для використання.

Основний потік подій: даний варіант використання починає виконуватися, коли зареєстрованому користувачу потрібно авторизуватися. Система пропонує ввести логін та пароль. Після того, як користувач ввів їх, система перевіряє правильність введених даних, і у випадку вдачі надає йому функціонал.

Альтернативний потік: якщо логін чи пароль невірні – система сповіщає про це користувача. Користувач може спробувати ще раз пройти авторизацію.

Передумова: перед початком виконання даного варіанта використання користувач повинен бути зареєстрований у системі.

Прецедент «Вихід з системи»

Призначення: даний варіант використання надає можливість користувачу

вийти з системи.

Основний потік подій: даний варіант використання починає виконуватися, коли користувачу потрібно вийти з системи. Користувач натискає на значок профіля та у списку дій обирає «Вихід». Після того, як користувач вийшов, система відображає головну сторінку.

Передумова: перед початком виконання даного варіанта використання користувач повинен авторизуватися в системі.

Прецедент «Керування шаблонами»

Призначення: даний варіант використання надає можливість користувачу взаємодіяти з шаблонами.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач системи натискає на пункт меню «Шаблони» в особистому кабінеті. Система відображає список шаблонів та інструменти взаємодії з ними.

Альтернативний потік подій: якщо жодного шаблону ще не було додано, система повідомить про це у вигляді напису «Немає жодного шаблону».

Передумова: перед початком виконання даного варіанта використання користувач повинен виконати вхід до системи.

Прецедент «Створення шаблону»

Призначення: даний варіант використання надає можливість користувачу додавати нові шаблони.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач системи натискає кнопку «Створити», система відображає конструктор створення нового шаблону. Після створення конструкції шаблону, користувач натискає кнопку «Зберегти», система зберігає новий шаблон та відображає сторінку «Шаблони».

Передумова: перед початком виконання даного варіанта використання користувач повинен знаходитися в розділі «Шаблони».

Виняткова ситуація 1: користувач натиснув кнопку «Скасувати» – система відображає сторінку «Шаблони».

Прецедент «Видалення шаблону»

Призначення: даний варіант використання надає можливість користувачу видалити шаблон.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач системи натискає піктограму кошику в рядку шаблонів, система видаляє шаблон.

Передумова: перед початком виконання даного варіанта використання користувач повинен знаходитися в розділі «Шаблони».

Прецедент «Редагування шаблону»

Призначення: даний варіант використання надає можливість користувачу редагувати шаблон.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач системи натискає на піктограму олівця в рядку шаблону, система відображає інтерфейс редагування шаблону. Після внесення змін, користувач натискає кнопку «Зберегти», система зберігає оновлені дані та відображає сторінку «Шаблони».

Передумова: перед початком виконання даного варіанта використання користувач повинен знаходитися в розділі «Шаблони».

Виняткова ситуація 1: користувач натиснув кнопку «Скасувати» – система відображає сторінку «Шаблони».

Прецедент «Перегляд шаблону»

Призначення: даний варіант використання надає можливість користувачу переглядати шаблон.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач системи натискає на піктограму ока в в рядку шаблону, система відображає попередній перегляд шаблону.

Передумова: перед початком виконання даного варіанта використання користувач повинен знаходитися в розділі «Шаблони».

Прецедент «Керування сертифікатами»

Призначення: даний варіант використання надає можливість користувачу взаємодіяти з сертифікатами.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач системи натискає на пункт меню «Сертифікати» в особистому кабінеті. Система відображає інструментарій взаємодії зі сертифікатами.

Передумова: перед початком виконання даного варіанта використання користувач повинен виконати вхід до системи.

Прецедент «Створення сертифікату»

Призначення: даний варіант використання надає можливість користувачу створювати сертифікати.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач системи натискає кнопку «Створити», система відображає інтерфейс створення сертифікату. Після вибору даних для генерації та шаблону, користувач натискає кнопку «Згенерувати», система генерує сертифікат та завантажує його на пристрій користувача.

Передумова: перед початком виконання даного варіанта використання користувач повинен знаходитися в розділі «Сертифікати», а також повинен бути створеним мінімум один шаблон.

Виняткова ситуація 1: завантажено невалідні дані – система відображає відповідне повідомлення. Користувач може змінити дані.

Виняткова ситуація 2: не створено жодного шаблону – система відображає відповідне повідомлення та пропонує створити шаблон.

Виняткова ситуація 3: користувач натиснув кнопку «Скасувати» – система відображає сторінку «Сертифікати».

Виняткова ситуація 4: відсутній зв’язок з сервером – система відображає відповідне повідомлення.

Прецедент «Завантаження даних»

Призначення: даний варіант використання надає можливість користувачу завантажити дані для генерування сертифікату.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач системи натискає на кнопку «Завантажити дані». Система зчитує та зберігає дані.

Передумова: перед початком виконання даного варіанта використання користувач повинен знаходитися в інтерфейсі створення сертифікату.

Прецедент «Вибір шаблону»

Призначення: даний варіант використання надає можливість користувачу вибрати шаблон.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач системи натискає на селектор «Шаблони». Система пропонує обрати з переліку створених шаблонів. Після того, як користувач обрав шаблон, система запам’ятовує вибір.

Передумова: перед початком виконання даного варіанта використання користувач повинен знаходитися в інтерфейсі створення сертифікату.

Прецедент «Завантаження сертифікату»

Призначення: даний варіант використання надає можливість користувачу завантажити згенерований сертифікат.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач системи натискає на кнопку «Згенерувати», система зберігає сертифікат на пристрій користувача.

Передумова: перед початком виконання даного варіанта використання користувач повинен знаходитися в інтерфейсі створення сертифікату.

Прецедент «Видалення сертифікату»

Призначення: даний варіант використання надає можливість користувачу видалити сертифікат.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач системи натискає піктограму кошику в рядку сертифікатів, система видаляє сертифікат.

Передумова: перед початком виконання даного варіанта використання користувач повинен знаходитися в розділі «Сертифікати».

Прецедент «Редагування сертифікату»

Призначення: даний варіант використання надає можливість користувачу редагувати сертифікат.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач системи натискає на піктограму оновлення в рядку сертифікату, система оновлює дані згідно змінам у шаблоні.

Передумова: перед початком виконання даного варіанта використання користувач повинен знаходитися в розділі «Сертифікати».

Прецедент «Перегляд сертифікату»

Призначення: даний варіант використання надає можливість користувачу переглядати згенеровані сертифікати.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач системи натискає на кнопку піктограму ока в в рядку сертифікату, система відображає попередній перегляд сертифікату.

Передумова: перед початком виконання даного варіанта використання користувач повинен знаходитися в розділі «Сертифікати».

2.3 Діаграма діяльності

Діаграма діяльності (Activity Diagram) є однією із типів діаграм в рамках Unified Modeling Language (UML), і вона використовується для моделювання послідовності дій, процесів та взаємодії між об’єктами або компонентами в системі. Діаграма діяльності дозволяє графічно подавати процеси та послідовність кроків, які відбуваються в системі, та використовується для аналізу, проєктування та документування системи.

Основні елементи діаграми діяльності включають в себе: дії, рішення, паралельну діяльність, поцільовість дій, та пакети.

Дії (Actions). Дії відображають окремі кроки, які виконуються в процесі. Вони представлені графічними прямокутниками зі скругленими кутами та містять короткий опис того, що саме відбувається на кожному кроці.

Рішення (Decisions). Рішення вказують на можливість розгалуження процесу, де вибір варіанту залежить від умови або результату попереднього

кроку. Ці рішення зображаються вигнутими стрілками, де різні гілки вказують на різні шляхи виконання.

Паралельна діяльність (Parallel Activities). Діаграма може відображати паралельну діяльність, коли кілька процесів відбуваються одночасно. Це показується за допомогою ліній, які розгалужуються і об'єднуються, щоб вказати паралельні потоки.

Поцільовість дій (Control Flow). Лінії у діаграмі діяльності, які з'єднують дії та рішення, вказують на послідовність дій та потік контролю в процесі. Вони показують, як виконуються дії від однієї до іншої.

Пакети (Swimlanes). Пакети відділяють різних виконавців чи об'єкти, які беруть участь в діяльності, і вони поділяють діаграму на різні лінії чи області, кожна з яких відповідає за виконання конкретних дій чи кроків.

Наведемо діаграму діяльності, що описує модель поведінки варіанта використання «Створити сертифікат». Діаграма представлена на рисунках 2.4 – 2.5.

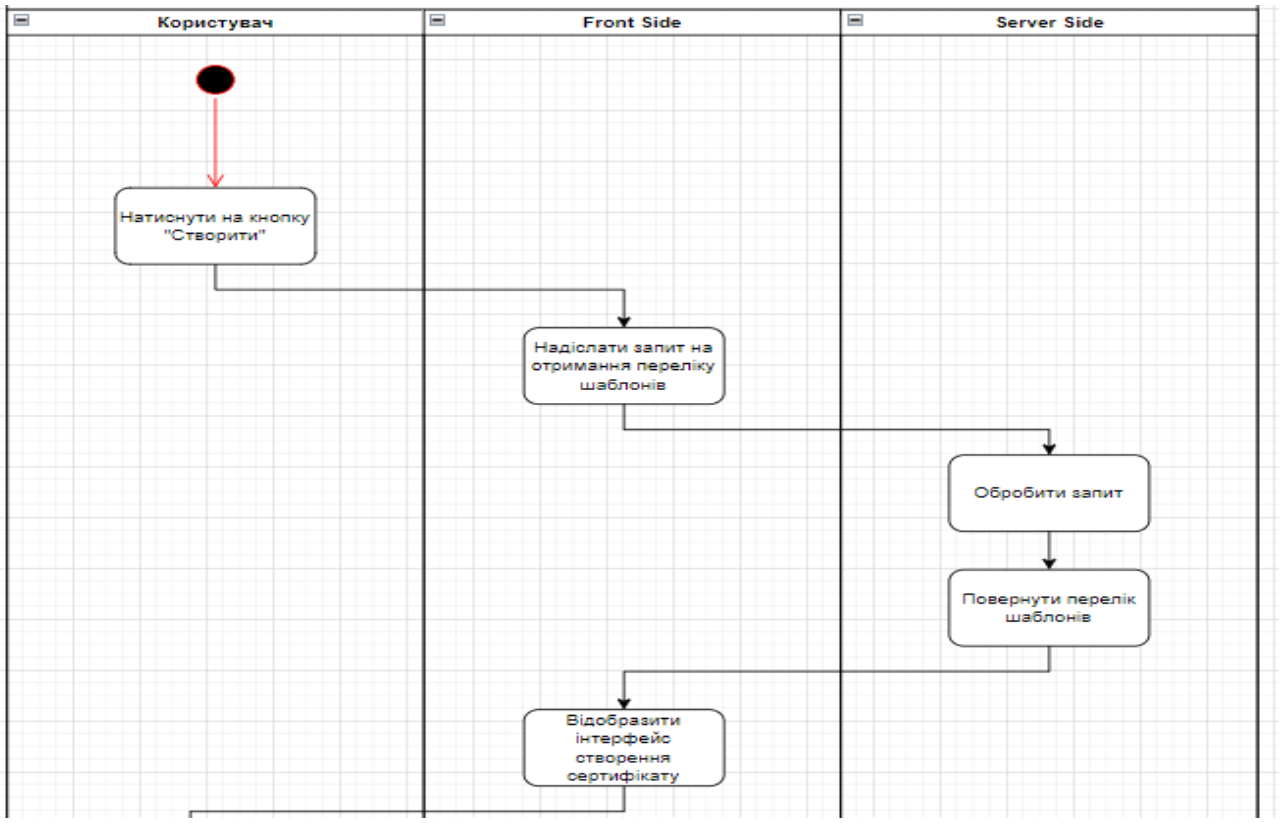

Рисунок 2.4 – Діаграма діяльності

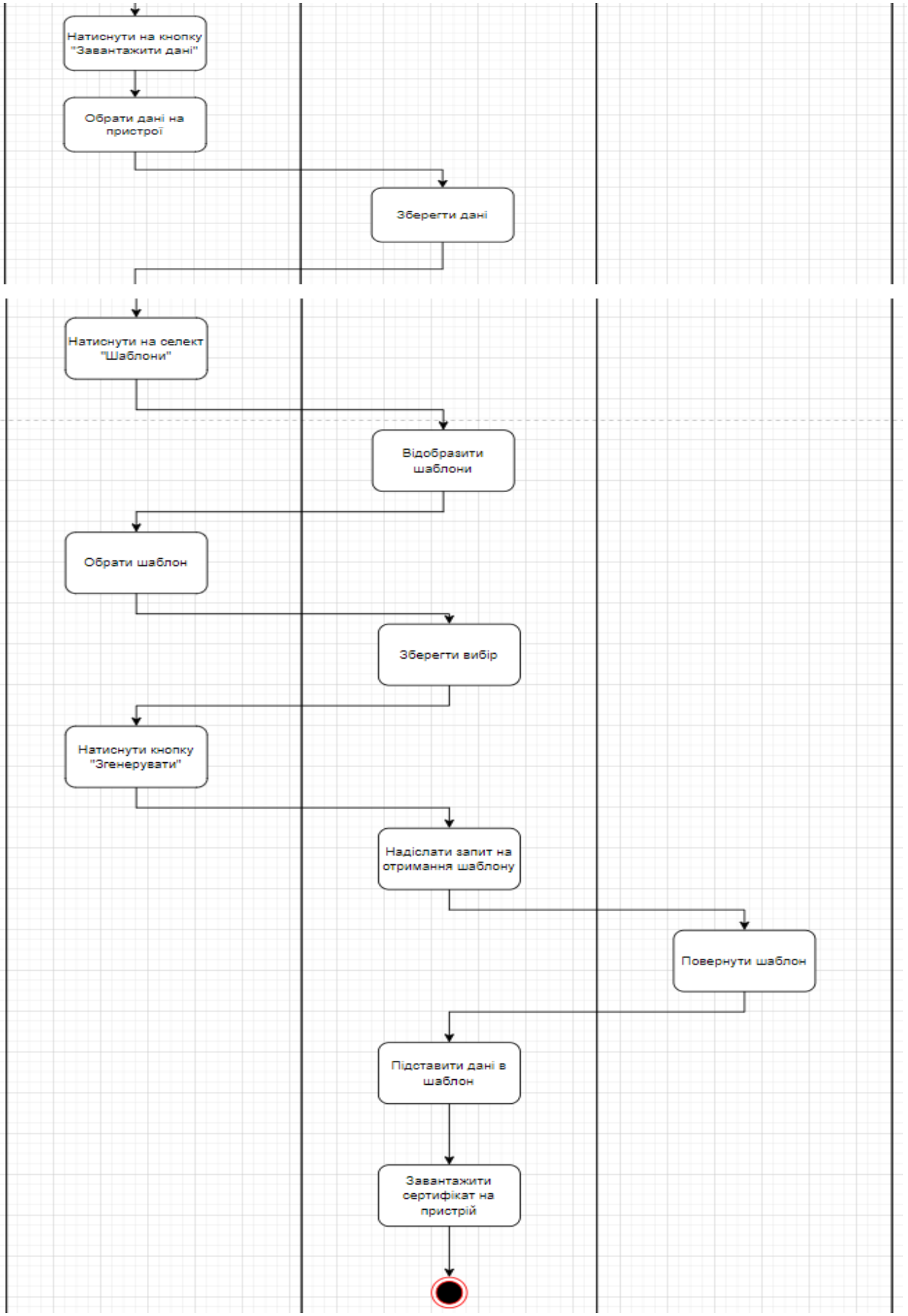

Рисунок 2.5 – Діаграма діяльності

2.4 Діаграма послідовності

Діаграма послідовності (Sequence Diagram) є однією з ключових діаграм в рамках Unified Modeling Language (UML), і вона використовується для моделювання взаємодій між різними об’єктами або компонентами в системі у вигляді послідовності подій та повідомлень. Діаграма послідовності дозволяє графічно відобразити, як об’єкти спілкуються один з одним та в якому порядку обмінюються повідомленнями, що допомагає розробникам легше розуміти внутрішню логіку системи та взаємодію між її компонентами.

На рисунку 2.6 описана діаграма послідовності прецедента «Створити сертифікат».

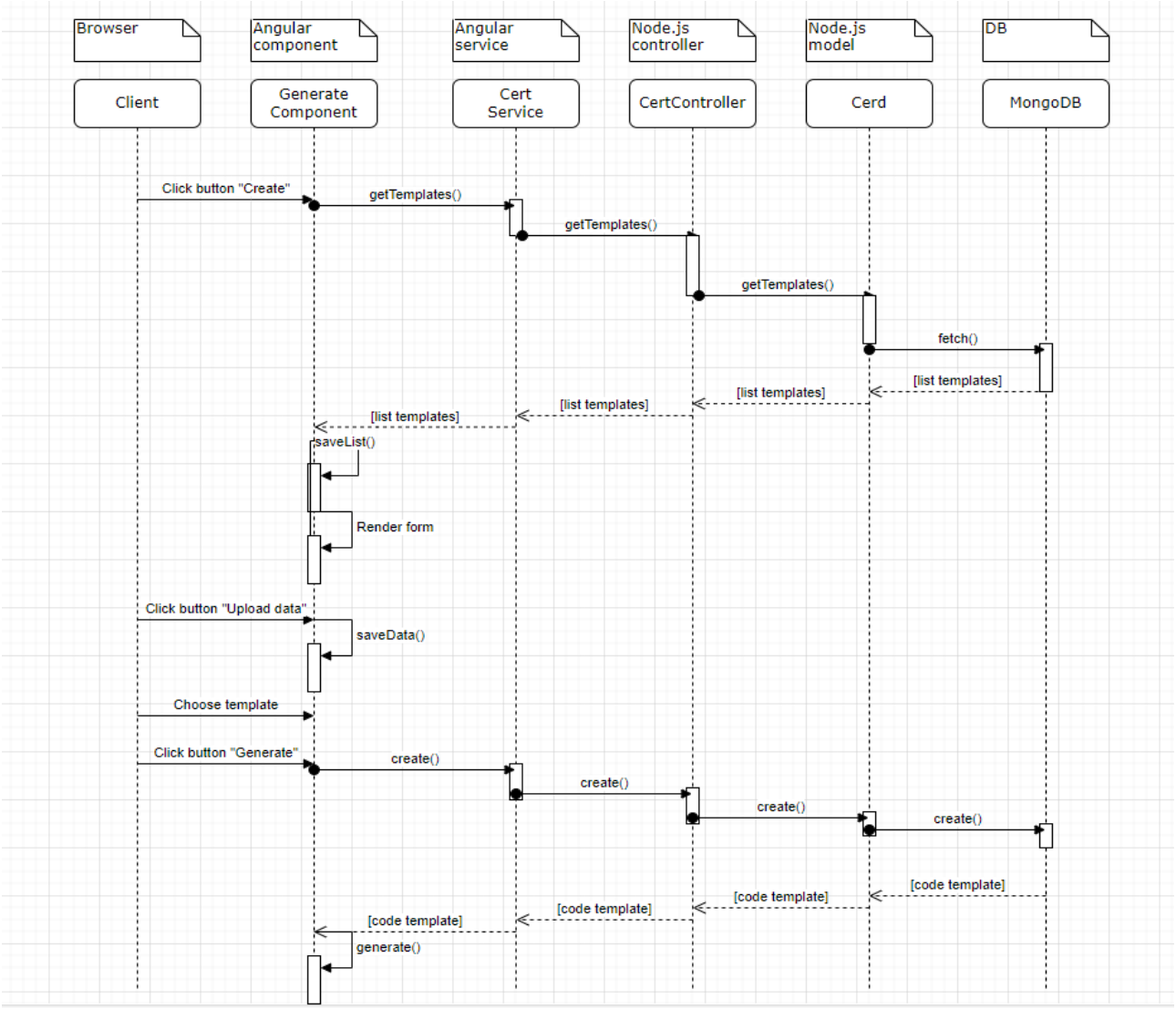

Рисунок 2.6 – Діаграма послідовності

Основні елементи діаграми послідовності включають в себе: об’єкти, послідовність, повідомлення та операції.

Об’єкти (Objects). Об'єкти, які беруть участь у взаємодії, представлені у верхній частині діаграми. Кожен об’єкт зазвичай має ім’я та область життєдіяльності, яка вказує на тривалість існування об’єкта під час взаємодії.

Послідовність (Lifelines). Послідовності визначають тривалість існування кожного об’єкта в межах діаграми. Вони представлені у вигляді вертикальних ліній, які іноді мають імена об’єктів поруч.

Повідомлення (Messages). Повідомлення показують обмін інформацією між об’єктами. Вони представлені стрілками, які з’єднують об’єкти та показують напрямок обміну даними чи керуванням

Операції (Operations). Вибіркові операції можуть вказувати, які дії виконуються в межах кожного повідомлення, або специфіку обробки даних.

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

3.1 Опис інструментів розробки

Для реалізації були використані Node.js та фреймворк Angular.

RxJS – це бібліотека, що реалізує принципи реактивного програмування JavaScript. Заснована на об’єктах типу Observable, вона спрощує написання та контроль асинхронного та подвійного коду [4].

3.2 Angular-компонент

Angular-компонент – це основна будівельна одиниця Angular-додатків, яка відповідає за відображення та поведінку частини користувацького інтерфейсу (UI). Компоненти Angular допомагають організувати код додатка шляхом розділення його функціональності на невеликі та самостійні частини.

Angular використовує концепцію компонентної архітектури, яка полегшує розробку та обслуговування коду шляхом розділення додатка на невеликі та перевикористовувальні компоненти [1, 4].

Приклад коду компонента наведено у Додатку А.

3.3 Angular-сервіс

Angular-сервіс – це клас, який надає певний функціонал та може бути використаний для обробки логіки, яка не пов’язана напряму з компонентами. Сервіси в Angular використовуються для розділення та організації логіки додатка, яка не є частиною компонентів, такої як обробка даних, взаємодія з сервером, обробка подій та інше.

Загалом, використання Angular-сервісів дозволяє розробникам створювати модульні та підтримувані додатки, обираючи правильний рівень абстракції для конкретної функціональності [3].

Приклад коду сервісу наведено у Додатку Б.

3.4 Node.js-сервер

Node.js-сервер – це сервер, який використовує середовище виконання Node.js для обробки запитів і відповідей на основі JavaScript або інших мов, що компілюються в JavaScript. Node.js є відкритим вихідним кодом, платформонезалежним середовищем для виконання серверного JavaScript-коду.

Node.js використовується для створення різних типів серверних додатків, таких як вебсервери, API-сервери, чат-сервери та інші. Він дозволяє розробникам використовувати одну мову (JavaScript) для реалізації як клієнтської, так і серверної частин додатка [7, 8].

Приклад коду серверу наведено у Додатку В.

3.5 Firebase Realtime Database

Firebase Realtime Database – це хмарна база даних від Google, яка надає можливість реального часу для зберігання та синхронізації даних між різними клієнтами. Це один із сервісів Firebase, який дозволяє легко розробляти додатки, які потребують обміну даними в реальному часі.

Firebase Realtime Database є частиною платформи Firebase, яка також включає інші сервіси, такі як аутентифікація користувачів, хостинг, аналітика та інші. Це дозволяє розробникам швидко створювати та масштабувати додатки з використанням різноманітних хмарних сервісів [9, 11].

Приклад колекції БД наведено у Додатку Г.

3.6 Основні класи системи

Так як основним прецедентом системи є «Генерація сертифікату», то основними класами системи будуть ті, що надають можливість взаємодіяти з сертифікатами.

Головними класами є:

– CertService: відповідає за дії пов’язані з Node.js API-сервером, а саме містить запити на відправку даних сертифікату, а також завантаження у вигляді PDF;
– TempService: відповідає за дії, що пов’язані з Firebase Realtime Database засобами API (містить логіку щодо створення, отримання, редагування та видалення шаблону сертифікату);
– CertComponent: відповідає за інтерфейс взаємодії з вищевказаними сервісами.

Детально ознайомитися з кодом проєкту можна у додатку Д.

3.7 Тестування проєкту

3.7.1 Unit-тест

Unit-тест – це вид тестування програмного забезпечення, який використовується для перевірки індивідуальних одиниць програмного коду (зазвичай функцій або методів) на коректність їх роботи [5].

Основна ідея unit-тестування полягає в тому, щоб перевірити, чи працює конкретна частина коду так, як очікується, без необхідності запускати всю програму чи систему.

Приклади такого тестування наведено на рисунках 3.1–3.2.

```
it('should create', () => {
  expect(component).toBeTruthy();
});

it('should initialize the form with controls', () => {
  expect(component.form.get('email')).toBeDefined();
  expect(component.form.get('password')).toBeDefined();
});

it('should set error message based on query parameters', () => {
  const queryParams: Params = { loginAgain: true };
  spyOn(component.route.queryParams, 'subscribe').and.callFake(callback => callback(queryParams));

  component.ngOnInit();

  expect(component.message).toBe('Insert data');
});
```

Рисунок 3.1 – Тестування SigninComponent

```
  component = fixture.componentInstance;
  templateServiceSpy = TestBed.inject(TemplateService) as jasmine.SpyObj<TemplateService>;
  routerSpy = TestBed.inject(Router) as jasmine.SpyObj<Router>;
});

it('should create', () => {
  expect(component).toBeTruthy();
});

it('should call getAll method on ngOnInit', () => {
  const mockTemplates = [{ id: '1', title: 'Template 1', text: 'Test text 1', date: new Date() }, { i
  templateServiceSpy.getAll.and.returnValue(of(mockTemplates));

  component.ngOnInit();

  expect(templateServiceSpy.getAll).toHaveBeenCalled();
  expect(component.templates).toEqual(mockTemplates);
});
```

Рисунок 3.2 – Тестування TemplateComponent

3.7.2 Integration-тест

Integration-тест (тест інтеграції) – це вид тестування програмного забезпечення, який перевіряє взаємодію між різними компонентами чи модулями системи [5].

На відміну від unit-тестів, які перевіряють коректність роботи окремих частин коду (наприклад, функцій чи методів), integration-тести спрямовані на перевірку взаємодії між цими частинами та визначення, чи працюють вони разом правильно. На рисунку 3.3 зображено тестування сервісу.

```
describe('AuthService', () => {
  let authService: AuthService;
  let httpTestingController: HttpTestingController;

  beforeEach(() => {
    TestBed.configureTestingModule({
      imports: [HttpClientTestingModule],
      providers: [AuthService],
    });

    authService = TestBed.inject(AuthService);
    httpTestingController = TestBed.inject(HttpTestingController);
  });

  afterEach(() => {
    httpTestingController.verify();
  });

  it('should be created', inject([AuthService], (service: AuthService) => {
    expect(service).toBeTruthy();
  }));

  it('should login user and set token', () => {
    const mockUser: User = { email: 'test@example.com', password: 'password' };
    const mockResponse: FbAuthResponse = { idToken: 'mockToken', expiresIn: '3600' };

    authService.login(mockUser).subscribe(response => {
      expect(response).toEqual(mockResponse);
      expect(authService.token).toBe('mockToken');
    });
```

Рисунок 3.3 – Тестування сервісу

3.8 Керівництво користувача

Користувач сайту повинен володіти певною кваліфікацією. Навички користувача для роботи з ПК, та навички роботи з web-браузером. Знайомство з Керівництвом користувача.

3.8.1 Підготовка до роботи

Запуск системи. Доступ до сайту здійснюється через мережу Інтернет за допомогою звичайного web-браузера. Адреса сайту в мережі Інтернет: https://cert.loc. Для коректної роботи клієнтської частини повинен

використовуватися браузер Google Chrome, Mozilla Firefox, Opera, Safari.

При вході на Сайт користувач потрапляє на сторінку входу до системи.

На Сайті розрізняються наступні групи користувачів:

– анонімний користувач (не увійшли або не зареєстровані, мають доступ до сторінок входу);
– користувач – приватна особа, авторизований на сайті (далі Користувач), що має доступ до функцій користувача у особистому кабінеті.

3.8.2 Вхід до системи

При вході на сайт, система відображає сторінку входу до системи (рис. 3.4). Користувачу потрібно ввести логін та пароль у відповідні поля та натиснути кнопку «Вхід».

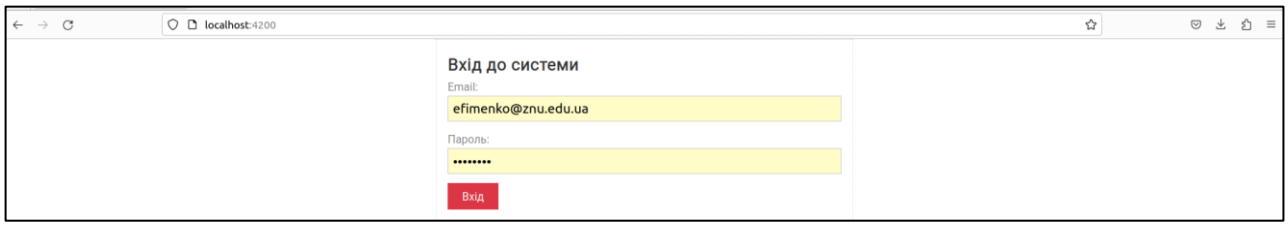

Рисунок 3.4 – Форма входу до системи

Під час заповнення форми, система автоматично перевіряє валідність, якщо дані валідні, то кнопка «Вхід» буде активною. Якщо користувач ввів невалідні дані, то система сповістить його про це у вигляді відповідного повідомлення під невалідним полем (рис. 3.5).

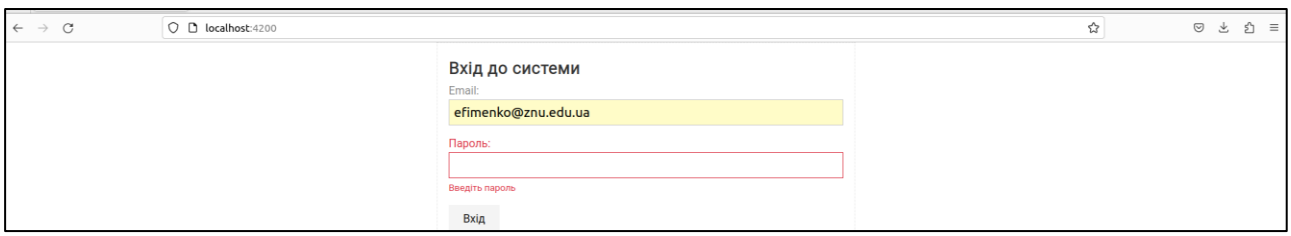

Рисунок 3.5 – Невалідні дані

Після авторизації, система відобразить сторінку з списком створених користувачем шаблонів. Якщо жодного шаблону не було створено, то система відображає повідомлення «Шаблони відсутні» (рис. 3.6).

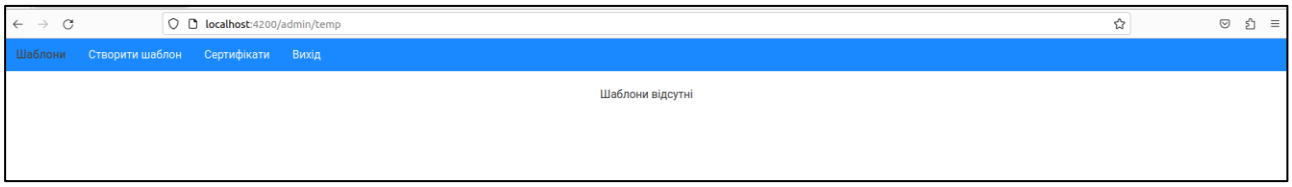

Рисунок 3.6 – Список шаблонів

3.8.3 Взаємодія зі шаблонами

Для того щоб створити новий шаблон, користувач повинен натиснути на пункт меню «Створити шаблон» на панелі керування. Система відображає інтерфейс, який складається з назви шаблону, конструктора шаблону та кнопки збереження результату (рис. 3.7).

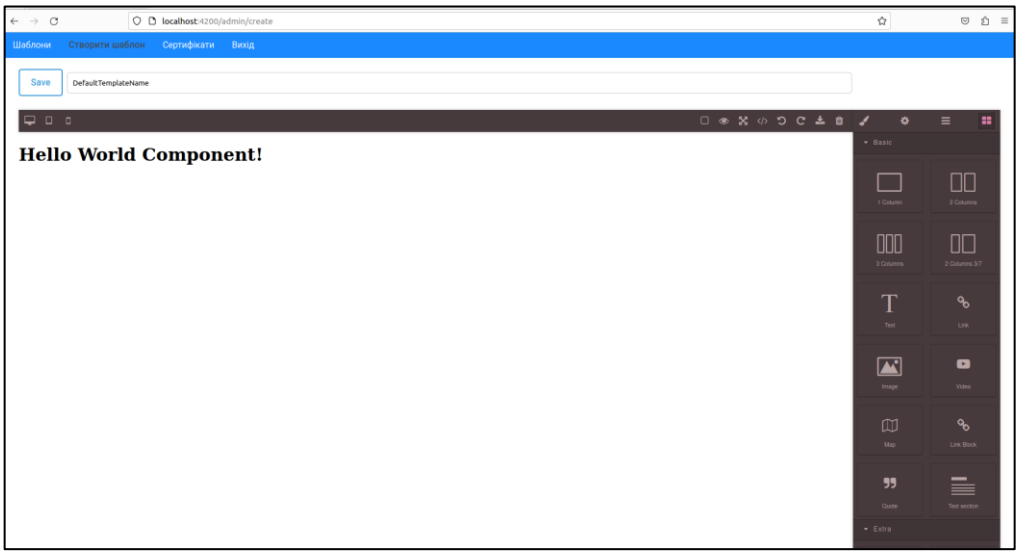

Рисунок 3.7 – Інтерфейс створення шаблону

За замовчуванням в шаблоні вже є деякі дані, такі як: назва за замовчуванням, компонент конструктора. Вже на цьому етапі шаблон можна

зберігати та взаємодіяти з ним в подальших діях. Конструктор шаблону пропонує різні компоненти з яких його можна побудувати (блоки, посилання, зображення).

Для прикладу продемонструємо додавання зображення, яке буде слугувати тлом нашого майбутнього сертифікату. Для цього в правобічній панелі потрібно обрати елемент «Image» та перетягнути його на тло, попередньо видалив блок з текстом. Система відображає інтерфейс завантаження зображення (див. рис. 3.8, 3.9).

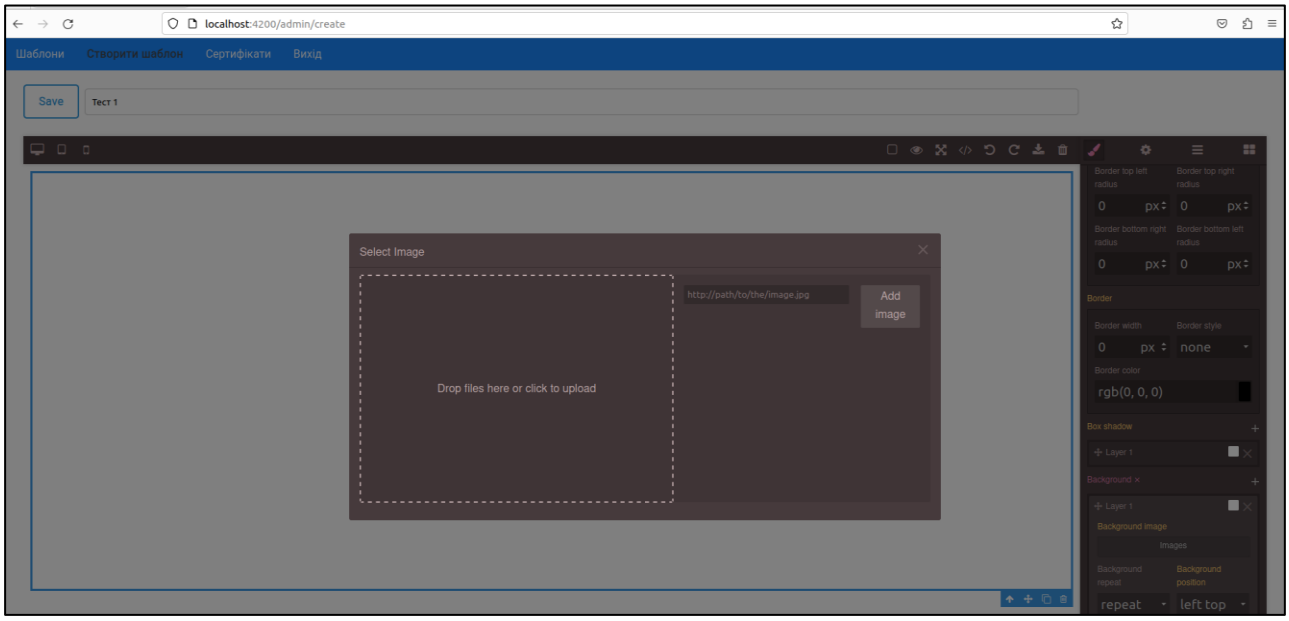

Рисунок 3.8 – Інтерфейс завантаження зображення

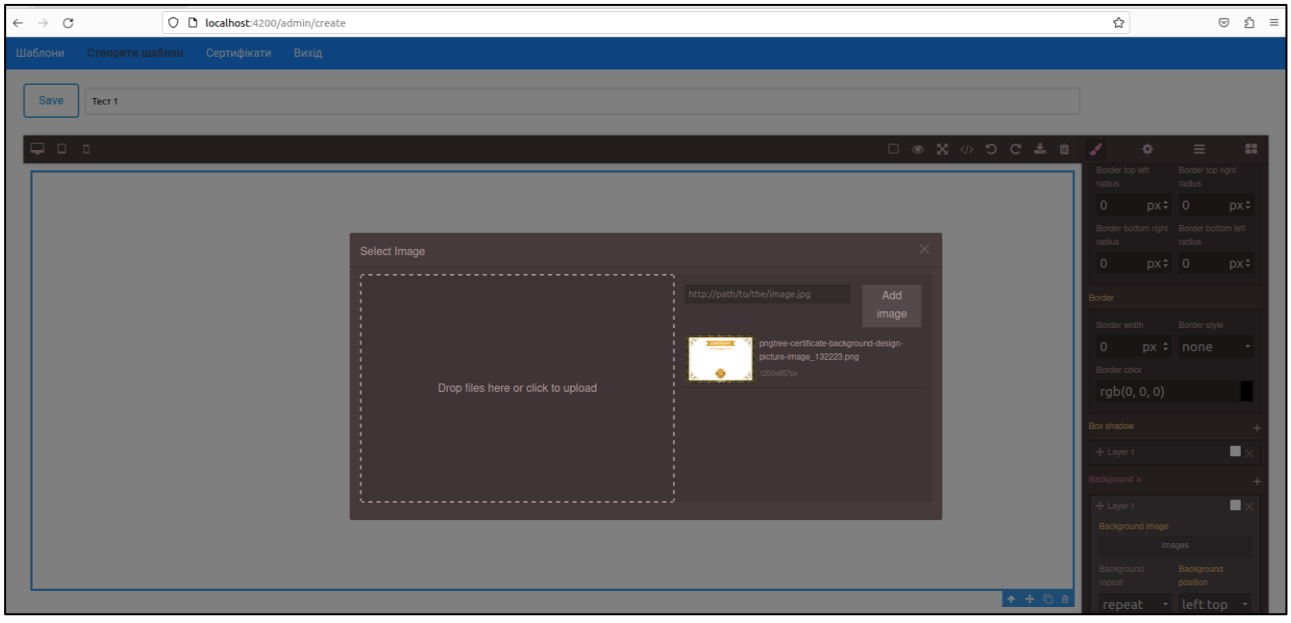

Рисунок 3.9 – Завантажене зображення

Після вибору зображення, користувач повинен подвійним кліком натиснути на його мініатюру, система вставляє зображення на тло (можна змінити розмір та задати інші стилі) (рис. 3.10).

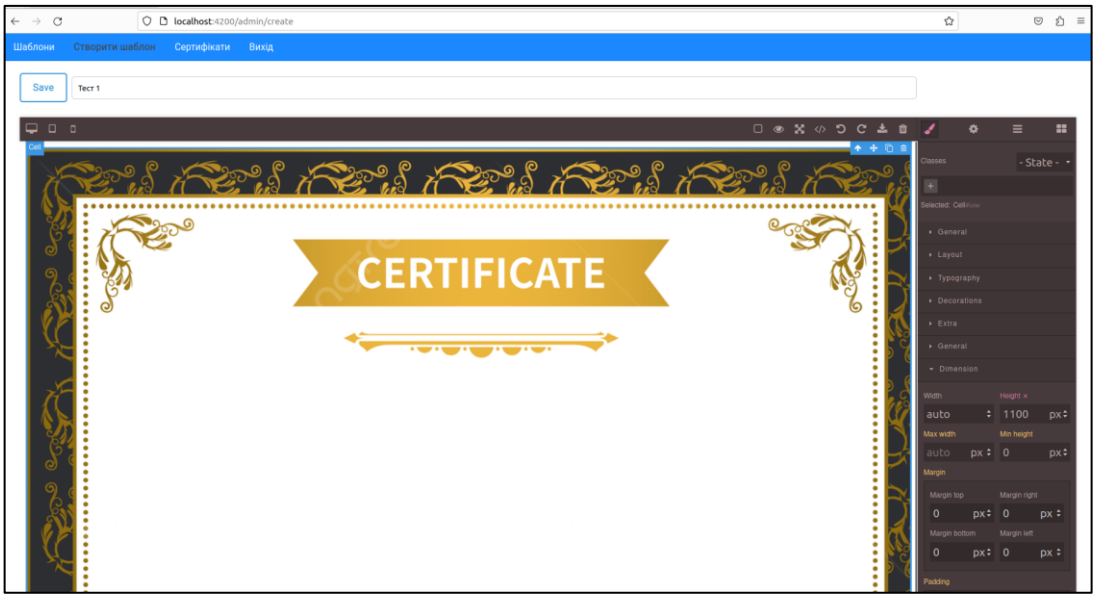

Рисунок 3.10 – Обране зображення та полотні

Далі користувач наповнює полотно потрібними елементами. Якщо потрібно при генерації сертифікату підставляти користувацькі дані, то потрібно вставити елемент «Text» у форматі «{field}», де field це назва стовпця з файлу з користувацькими даними. Також кожен елемент можна окремо стилізувати, наприклад змінити колір (рис. 3.11).

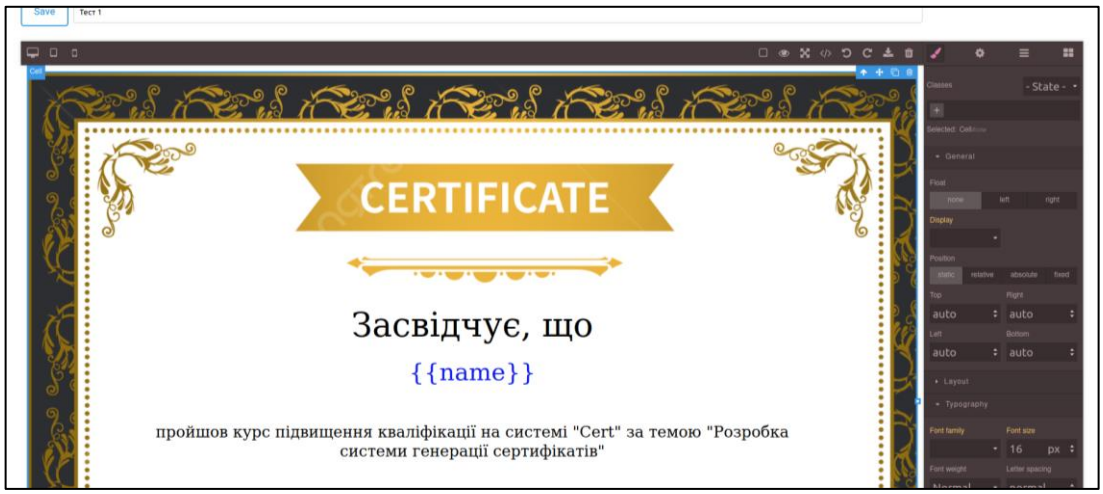

Рисунок 3.11 – Приклад сертифікату

Після конструювання сертифікату, користувач вказує назву та натискає на кнопку «Save», система відображає сторінку зі списком шаблонів (рис. 3.12).

Рисунок 3.12 – Створений шаблон

Якщо користувачу потрібно змінити назву або конструкцію шаблону, то він повинен натиснути на кнопку «Редагувати» в рядку відповідного шаблону. Система відображає інтерфейс редагування шаблону, де користувач має можливість внести зміни (рис. 3.13).

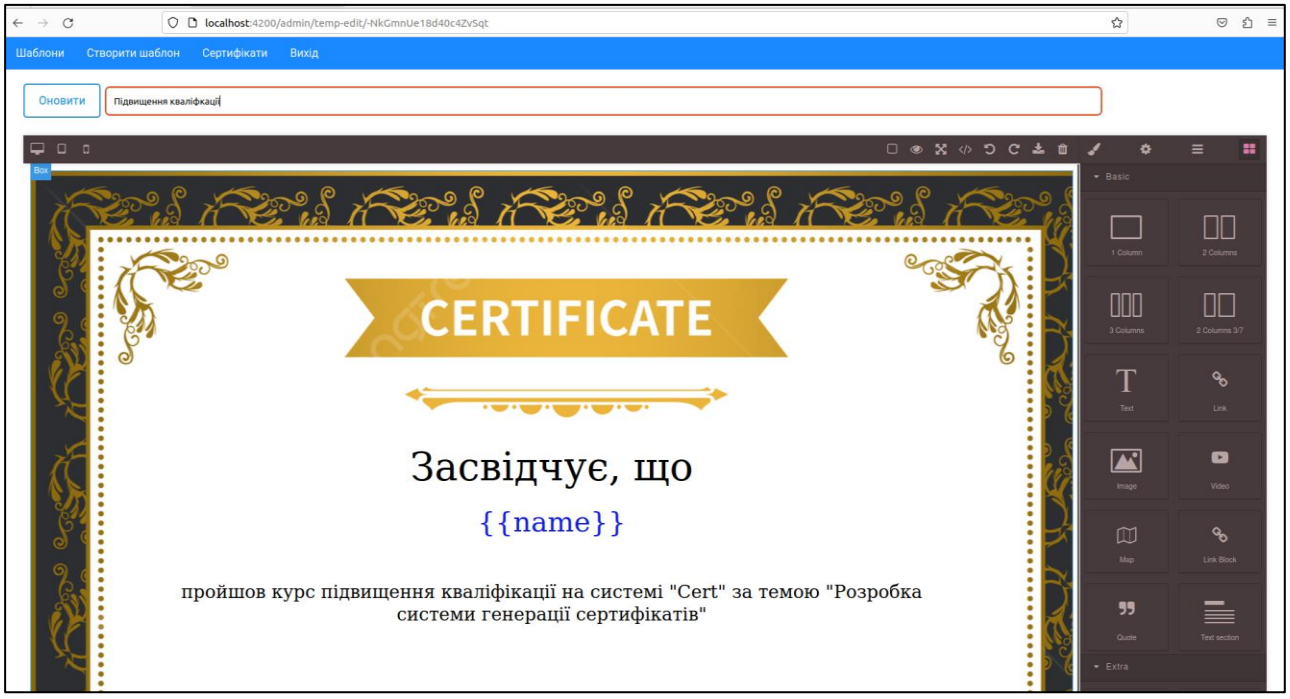

Рисунок 3.13 – Редагування сертифікату

Після внесення змін, користувач натискає на кнопку «Оновити», система оновлює дані та відображає сторінку з списком шаблонів (рис. 3.14).

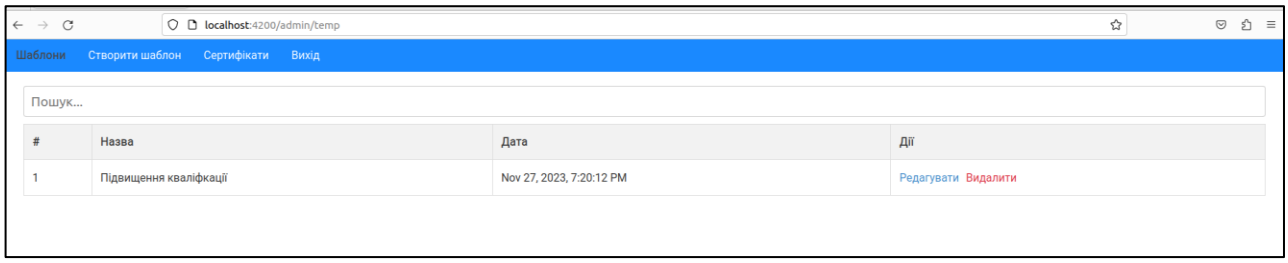

Рисунок 3.14 – Демонстрація оновлення даних

3.8.4 Взаємодія зі сертифікатами

Після створення шаблону, користувач має можливість створити сертифікат на його основі. Для цього потрібно натиснути на пункт меню «Сертифікати», система відображає інтерфейс взаємодії (рис. 3.15).

Рисунок 3.15 – Інтерфейс створення сертифікатів

Якщо користувачу потрібно підставити в шаблони користувацькі дані, то він повинен завантажити CSV-файл в форматі UTF-8 з роздільником «;» зі свого пристрою. Для цього він повинен натиснути на кнопку «Upload CSV file» та обрати файл з пристрою (див. рис. 3.16, 3.17).

Далі користувач обирає шаблон зі списку створених (див. рис. 3.18) та натискає на кнопку «Generate and download PDF», система генерує сертифікати та відображає їх в окремих вкладках браузера, а також зберігає на пристрій користувача (див. рис. 3.19).

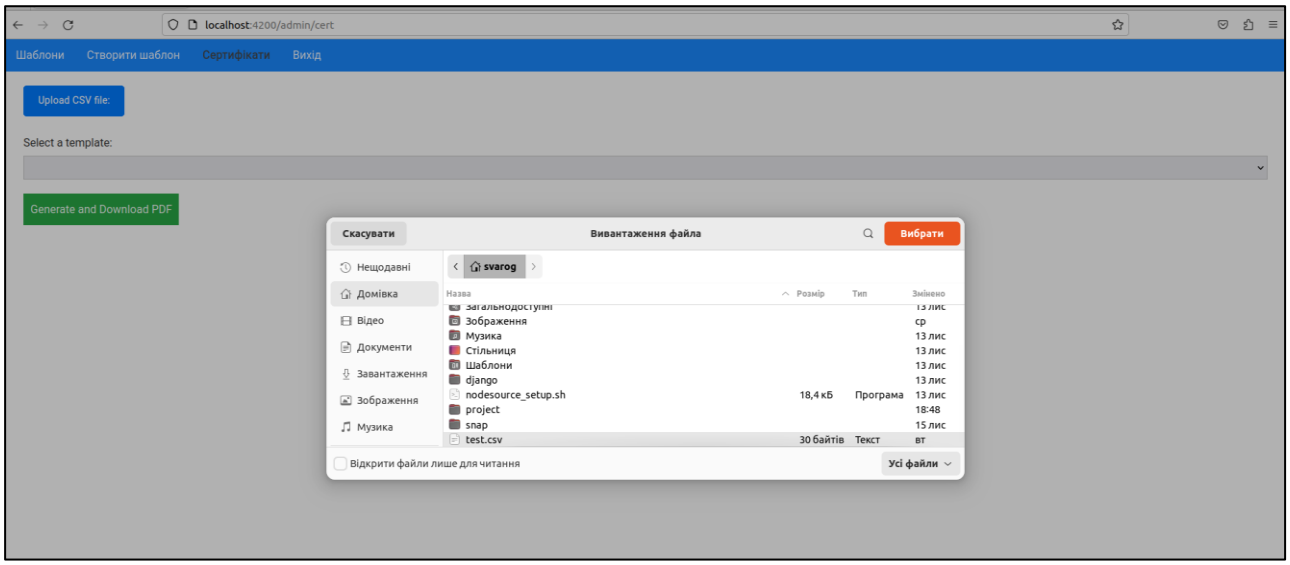

Рисунок 3.16 – Вибір файлу

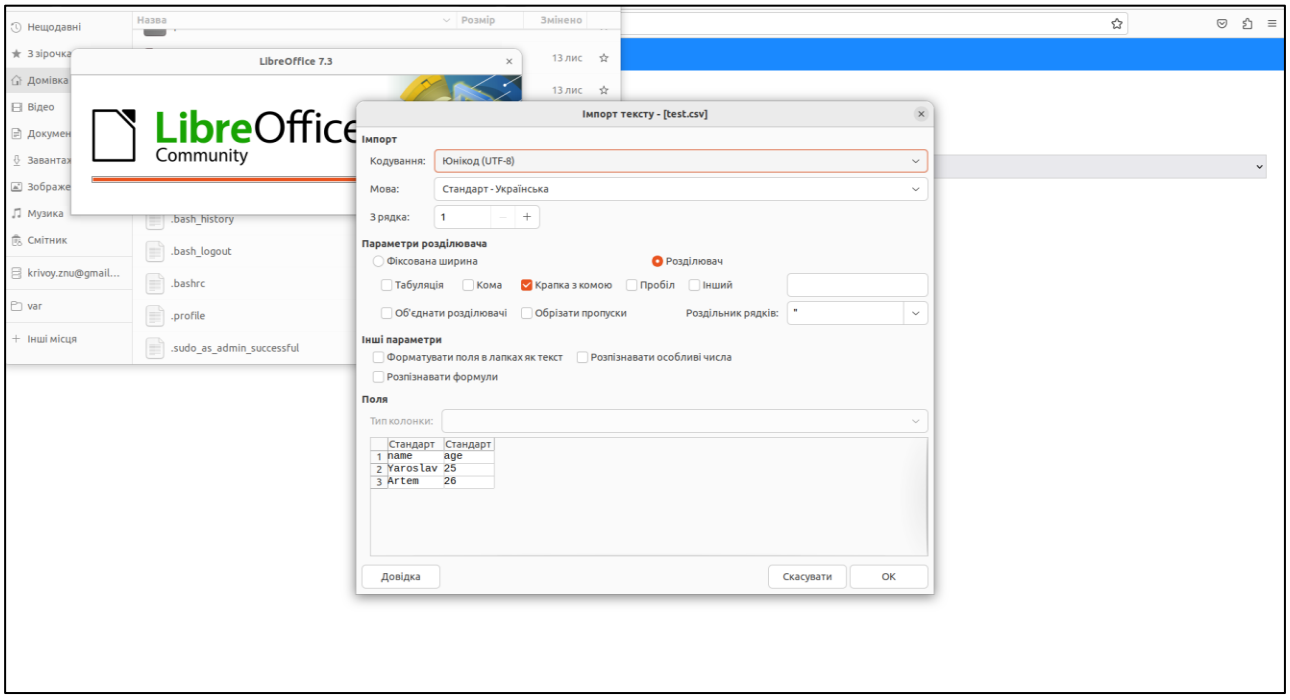

Рисунок 3.17 – Вміст файлу

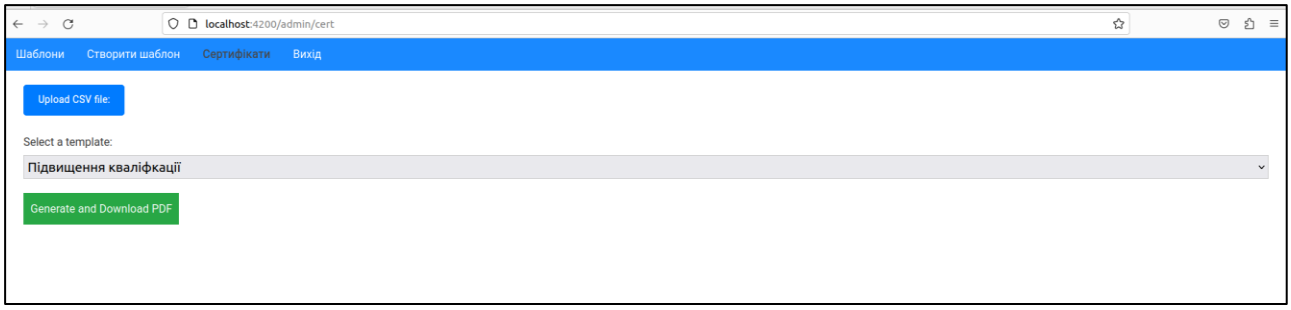

Рисунок 3.18 – Вибір шаблону

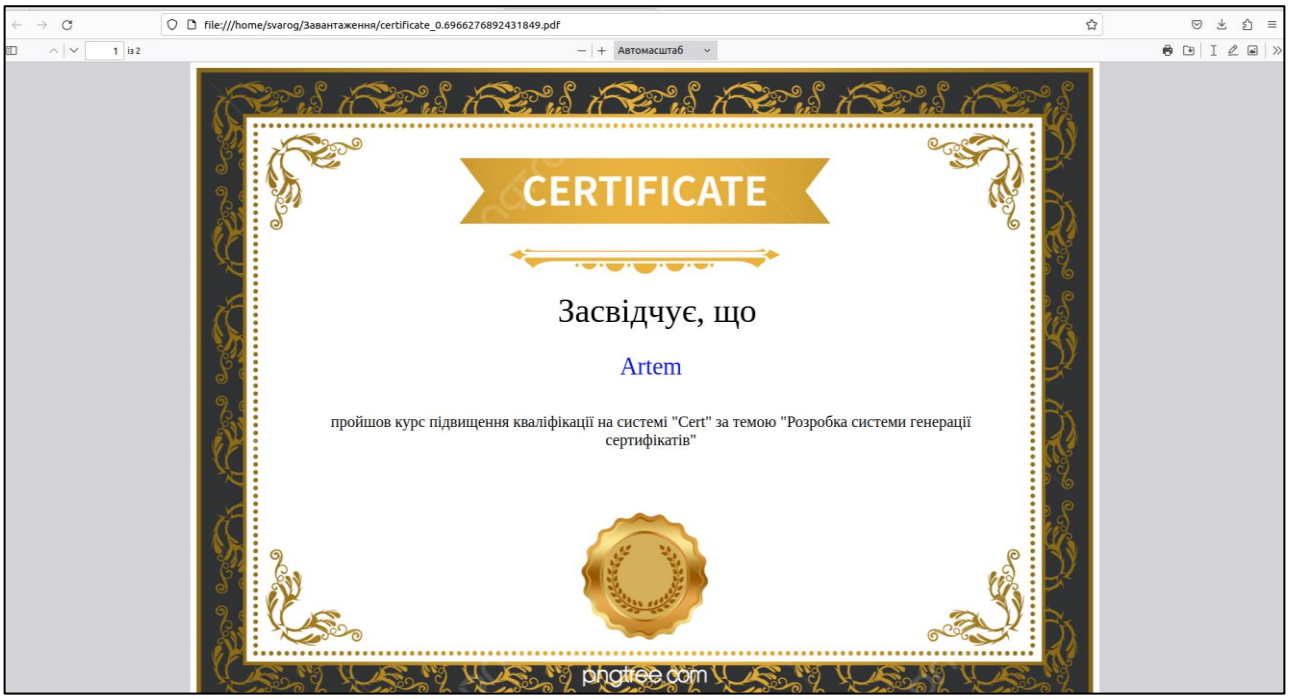

Рисунок 3.19 – Відображення сертифікату

ВИСНОВКИ

В результаті роботи було написано технічне завдання на розробку системи генерації сертифікатів. Для створення цієї системи було обрано фреймворк Angular зі сторони клієнта та Node.js зі сторони сервера, за їх широкі можливості у сфері створення web-систем.

У відповідності з метою кваліфікаційної роботи була розроблена система обліку навчального навантаження із застосуванням наступних технологій:

– Node.js та Firebase для реалізації back end API;
– Angular для реалізації front end частини, а також відправки запитів до серверу.

У відповідності з поставленими задачами були виконані наступні етапи створення системи:

– сформовані вимоги до системи (функціональні та нефункціональні (інтерфейс, кросбраузерність, безпека, продуктивність)), проведено огляд предметної області та інструментів розробки;
– спроєктована та побудована структура системи (побудовані діаграми прецедентів, діяльності та послідовності; надано детальний опис прецедентів);
– реалізовано систему генерації сертифікатів (наведена інструкція по створенню компонентів системи, надано керівництво користувача та структура проєкту);
– протестована робота системи.

ПЕРЕЛІК ПОСИЛАНЬ

1. Angular Developer Documentation. URL: https://angular.io/docs/ (дата звернення: 03.08.2023).
2. Bampakos A. Angular Projects: Build modern web apps by exploring Angular 12 with 10 different projects and cutting-edge technologies. Birmingham : Packt Publishing, 2021. 344 p.
3. Bampakos A., Deeleman P. Learning Angular: A no-nonsense guide to building web applications with Angular 15. Birmingham : Packt Publishing, 2023. 446 p.
4. Chebbi L. Reactive Patterns with RxJS for Angular: A practical guide to managing your Angular application’s data reactively and efficiently using RxJS 7. Birmingham : Packt Publishing, 2022. 224 p.
5. Fain Y., Moiseev A. Angular Development with TypeScript. New York : Manning Publications, 2019. 560 p.
6. Freeman A. Pro Angular: Build Powerful and Dynamic Web Apps. London : Apress, 2022. 905 p.
7. Li D. Introduction to Backend Development with Node.js and Express: Building Robust Web Applications with Node.js and Express Framework. Vancouver : Kindle Edition, 2023. 94 p.
8. Node.js Developer Documentation. URL: https://nodejs.org/en/docs (дата звернення: 01.10.2023).
9. Machado K. Angular 11 e Firebase: Construindo uma aplicação integrada com a plataforma do Google. Casa do Código, 2021. 193 p.
10. Rappl F. Modern Frontend Development with Node.js: A compendium for modern JavaScript web development within the Node.js ecosystem. Birmingham : Packt Publishing, 2022. 208 p.
11. Wieruch R. The Road to Firebase: Your journey to master Firebase in JavaScript. Independently published, 2021. 199 p.

ДОДАТОК А

Angular-компонент

```
import { Component, OnInit } from '@angular/core';
import grapesjs from 'grapesjs';
import 'grapesjs-preset-webpage';
import {Template} from '../../../auth/interfaces';
import {TemplateService} from '../../services/template.service';

@Component({
  selector: 'app-web-builder',
  templateUrl: './web-builder.component.html',
  styleUrls: ['./web-builder.component.scss']
})
export class WebBuilderComponent implements OnInit {

  public editor: any = null;
  templateName = 'DefaultTemplateName';

  constructor(private templateService$: TemplateService) { }

  ngOnInit(): void {
    this.editor = grapesjs.init({
      // Indicate where to init the editor. You can also pass an HTMLElement
      container: '#gjs',
      // Get the content for the canvas directly from the element
      // As an alternative we could use: `components: '<h1>Hello World
Component!</h1>'`,
      fromElement: true,
```

```
    // Size of the editor
    height: '100vh',
    width: '100wh',
    // Disable the storage manager for the moment
    storageManager: false,
    // Avoid any default panel
    panels: { defaults: [] },
    plugins: ['gjs-preset-webpage'],
    pluginsOpts: {
      'gjs-preset-webpage': {
        // options
      }
    },
  });
}

save(): void {
  const htmlCode = this.editor.getHtml();
  const cssStyles = this.editor.getCss();
  // Створити об'єкт для відправки до бази даних
  const templateToSave: Template = {
    title: this.templateName,
    text: htmlCode,
    css: cssStyles,
    date: new Date(),
  };

  // Викликати метод сервісу для відправки до бази даних
  this.templateService$.create(templateToSave).subscribe(
    (response) => {
```

```
        // Обробіть успішну відповідь, якщо потрібно
        console.log('Template saved successfully:', response);
      },
      (error) => {
        // Обробіть помилку, якщо є
        console.error('Error saving template:', error);
      }
    );
  }
}
```

ДОДАТОК Б

Angular-сервіс

```
import { Injectable } from '@angular/core';
import {FbCreateResponse, Template} from '../../auth/interfaces';
import {HttpClient} from '@angular/common/http';
import {Observable} from 'rxjs';
import {environment} from '../../../environments/environment';
import {map} from 'rxjs/operators';

@Injectable({
  providedIn: 'root'
})
export class TemplateService {

  constructor(private http: HttpClient) {}

  create(post: Template): Observable<Template> {
    return this.http.post(`${environment.fbDbUrl}/templates.json`, post)
      .pipe(map((response: FbCreateResponse) => {
        return {
          ...post,
          id: response.name,
          date: new Date(post.date)
        };
      }));
  }

  getAll(): Observable<Template[]> {
```

```
  return this.http.get(`${environment.fbDbUrl}/templates.json`)
    .pipe(map((response: {[key: string]: any}) => {
      return Object
        .keys(response)
        .map(key => ({
          ...response[key],
          id: key,
          date: new Date(response[key].date)
        }));
    }));
 }

 getById(id: string): Observable<Template> {
  return this.http.get<Template>(`${environment.fbDbUrl}/templates/${id}.json`)
    .pipe(map((post: Template) => {
      return {
        ...post, id,
        date: new Date(post.date)
      };
    }));
 }

 remove(id: string): Observable<void> {
  return this.http.delete<void>(`${environment.fbDbUrl}/templates/${id}.json`);
 }

 update(post: Template): Observable<Template> {
  return
this.http.patch<Template>(`${environment.fbDbUrl}/templates/${post.id}.json`,
post);
```

```
  }
}
```

ДОДАТОК В

Node.js-сервер

```
const express = require('express');
const bodyParser = require('body-parser');
const puppeteer = require('puppeteer');
const cors = require('cors');

const app = express();
const port = 3000;

app.use(bodyParser.json({ limit: '50mb' }));
app.use(cors());

app.post('/generate-pdf', async (req, res) => {
  const { htmlCode } = req.body;

  const browser = await puppeteer.launch();
  const page = await browser.newPage();

  await page.setContent(htmlCode);

  const pdfBuffer = await page.pdf({ width: '1498px', height: '1069px' })

  await browser.close();

  res.setHeader('Content-Type', 'application/pdf');
  res.send(pdfBuffer);
});
```

```
app.listen(port, () => {
  console.log(`Server is running on http://localhost:${port}`);
});
```

ДОДАТОК Г

Firebase Realtime Database

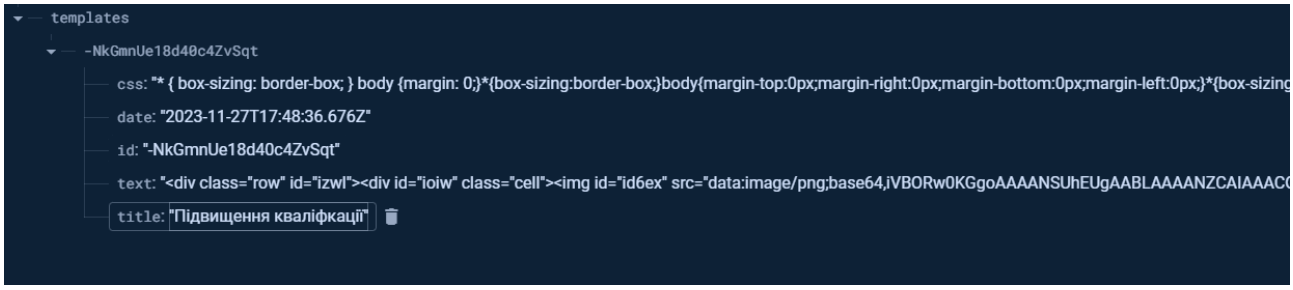

Рисунок Г.1 – Приклад елементу колекції

ДОДАТОК Д

Посилання на Git

https://bitbucket.org/s_var_og/cert/src/master/