

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»**

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«До захисту допущено»

Завідувач кафедри

_____ Євгенія СУЛЕМА

«___» _____ 2024 р.

Дипломний проєкт

на здобуття ступеня бакалавра

**за освітньо-професійною програмою «Інженерія програмного
забезпечення мультимедійних та інформаційно-пошукових систем»**

спеціальності 121 Інженерія програмного забезпечення

**на тему: «Вебзастосунок для випуску цифрових сертифікатів на основі
технології блокчейн»**

Виконав:

студент IV курсу, групи КП-02
Слободзян Максим Вікторович

Керівник:

Доцент кафедри ПЗКС, к.т.н.,
Шкурат Оксана Сергіївна

Консультант з нормоконтролю:

Доцент кафедри ПЗКС, к.т.н., доцент,
Онай Микола Володимирович

Рецензент:

Доцент кафедри СПіСКС, к.т.н., доцент,
Тарасенко-Клятченко Оксана Володимирівна

Засвідчую, що у цьому дипломному
проєкті немає запозичень з праць інших
авторів без відповідних посилань.

Студент _____

Київ – 2024 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет прикладної математики
Кафедра програмного забезпечення комп'ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення
мультимедійних та інформаційно-пошукових систем»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Євгенія СУЛЕМА

«__» _____ 2023 р.

ЗАВДАННЯ

на дипломний проєкт студенту

Слободзяну Максиму Вікторовичу

1. Тема проєкту «Вебзастосунок для випуску цифрових сертифікатів на основі технології блокчейн», керівник проєкту Шкурат Оксана Сергіївна, к.т.н., затверджені наказом по університету від «30» травня 2024 р. №2205-с.
2. Термін подання студентом проєкту «18» червня 2024 р.
3. Вихідні дані до проєкту: див. Технічне завдання.
4. Зміст пояснювальної записки:
 - аналіз існуючих рішень та обґрунтування теми дипломного проєкту;
 - обґрунтування вибору засобів реалізації застосунку;
 - розроблення вебзастосунку для випуску цифрових сертифікатів;
 - аналіз реалізації вебзастосунку.
5. Перелік обов'язкового графічного матеріалу:
 - структура бази даних (креслення);
 - алгоритм випуску цифрових сертифікатів (креслення);
 - схема авторизації користувача на основі РКС (плакат);
 - архітектура застосунку (плакат).

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Онай М.В., доцент		

7. Дата видачі завдання «31» жовтня 2023 р.

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	Вивчення літератури за тематикою проєкту	13.11.2023	
2.	Розроблення та узгодження технічного завдання	22.11.2023	
3.	Розроблення структури web-ресурсу	15.12.2023	
4.	Підготовка матеріалів першого розділу дипломного проєкту	28.12.2023	
5.	Розроблення дизайну сторінок та графічних елементів	01.02.2024	
6.	Підготовка матеріалів другого розділу дипломного проєкту	19.02.2024	
7.	Програмна реалізація web-ресурсу	11.03.2024	
8.	Тестування web-ресурсу	18.03.2024	
9.	Підготовка матеріалів третього розділу дипломного проєкту	27.03.2024	
10.	Підготовка матеріалів четвертого розділу дипломного проєкту	15.04.2024	
11.	Підготовка графічної частини дипломного проєкту	22.04.2024	
12.	Оформлення документації дипломного проєкту	27.05.2024	

Студент

Максим СЛОБОДЗЯН

Керівник проєкту

Оксана ШКУРАТ

АНОТАЦІЯ

Даний дипломний проєкт присвячений розробленню вебзастосунку для випуску цифрових сертифікатів на основі технології блокчейн.

Розроблене програмне забезпечення у вигляді вебзастосунку надає користувачам можливості обліку, створення та валідації цифрових сертифікатів про закінчення курсів, тренінгів, онлайн-занять тощо. Авторизовані користувачі мають змогу керувати створеними курсами, розробляти власні дизайни шаблонів, переглядати детальну інформацію про власні сертифікати. Неавторизовані користувачі можуть переглядати загальну інформацію про випущені за допомогою застосунку документи та здійснювати їх перевірку на цілісність та автентичність. Остаточне збереження даних про випущені сертифікати відбувається у публічному реєстрі блокчейну.

Актуальність проєкту полягає в поєднанні ергономічного користувацького інтерфейсу та високого рівня прозорості та безпеки застосунку. Користувач має змогу зручно, швидко та просто створити та опублікувати необхідні сертифікати, а використання технології блокчейну допоможе безпечно та відкрито зберігати публічні дані документів, що забезпечує надійний процес їх верифікації.

У даному дипломному проєкті розроблено архітектуру клієнтської та серверної частини вебзастосунку, алгоритми генерації та верифікації цифрових сертифікатів, авторизації користувача на основі PKC та JSON Web Token, структуру бази даних системи, графічний інтерфейс користувача та дизайн вебсторінок.

ABSTRACT

This diploma project is dedicated to the development of a web application for issuing digital certificates based on blockchain technology.

The software developed in the form of a website provides users with the ability to record, create, and validate digital certificates of completion of courses, trainings, online classes, etc. Authorized users can manage the created courses, develop their template designs, and view detailed information about their certificates. Unauthorized users can view general information about the documents issued through the application and check their integrity and authenticity. The final storage of issued certificate data takes place in the public blockchain registry.

The relevance of the project lies in the combination of an ergonomic user interface and a high level of transparency and security of the application. The user can conveniently, quickly, and easily create and publish the necessary certificates, and blockchain technology usage will help to securely and openly store public document data, ensuring a reliable verification process.

The following components are developed in this project: the architecture of the client and server parts of the web application, algorithms for generating and verifying digital certificates, user authorization based on PKC and JSON Web Token, the structure of the system database, graphical user interface, and web page design.

ДП.045440-01-90 Вебзастосунок для випуску цифрових сертифікатів на основі технології блокчейн. Відомість проєкту

Позначення	Найменування	Кіл-ть	Примітка
	Документація проєкту		
ДП.045440-02-91	Вебзастосунок для	5	
	випуску цифрових		
	сертифікатів на основі		
	технології блокчейн.		
	Технічне завдання		
ДП.045440-03-81	Вебзастосунок для	71	
	випуску цифрових		
	сертифікатів на основі		
	технології блокчейн.		
	Пояснювальна записка		
ДП.045440-04-51	Вебзастосунок для	4	
	випуску цифрових		
	сертифікатів на основі		
	технології блокчейн.		
	Програма та методика		
	тестування		
ДП.045440-05-34	Вебзастосунок для	17	
	випуску цифрових		
	сертифікатів на основі		
	технології блокчейн.		
	Керівництво користувача		
ДП.045440-06-99	Вебзастосунок для	1	
	випуску цифрових		
	сертифікатів на основі		
	технології блокчейн.		
	Структура бази даних.		
	ER-діаграма		

[illegible]

Факультет прикладної математики
Кафедра програмного забезпечення комп'ютерних систем

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Євгенія СУЛЕМА

“ ____ ” _____ 2023 р.

ВЕБЗАСТОСУНОК ДЛЯ ВИПУСКУ ЦИФРОВИХ СЕРТИФІКАТІВ
НА ОСНОВІ ТЕХНОЛОГІЇ БЛОКЧЕЙН

Технічне завдання

ДП.045440-02-91

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Оксана ШКУРАТ

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Максим СЛОБОДЗЯН

ЗМІСТ

1. Найменування та галузь застосування.....	3
2. Підстава для розроблення.....	3
3. Призначення розробки.....	3
4. Вимоги до програмного продукту.....	3
5. Вимоги до проєктної документації.....	5
6. Етапи проєктування.....	5
7. Порядок тестування розробки.....	5

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Вебзастосунок для випуску цифрових сертифікатів з використанням технології блокчейн.

Галузь застосування: Інформаційні технології.

2. ПІДСТАВА ДЛЯ РОЗРОБЛЕННЯ

Підставою для розроблення є завдання на дипломне проектування, затверджене кафедрою програмного забезпечення комп'ютерних систем Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» (КПІ ім. Ігоря Сікорського).

3. ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для створення, перегляду та верифікації цифрових сертифікатів. Основне завдання вебзастосунку полягає в забезпеченні зручного та водночас надійного процесу випуску цифрових сертифікатів. Використання технології блокчейн дозволяє забезпечити високий рівень захисту від фальсифікації та спрощує процес перевірки автентичності сертифікатів.

4. ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

Вебзастосунок повинен забезпечувати такі основні функції:

1. Авторизація користувачів на основі криптографії з відкритим ключем.

2. Створення, перегляд, пагінація та фільтрація курсів, у межах яких відбуватиметься процес випуску сертифікатів, управління списком осіб, що авторизовані до випуску.
3. Дизайн шаблонів цифрового файлу сертифікату з використанням власних динамічних атрибутів.
4. Можливість перегляду, пагінації та фільтрації публічної інформації про всі випущені сертифікати на платформі, отримання цифрового файлу документа для його власника.
5. Публікація даних про випущений сертифікат у блокчейн.
6. Верифікація цілісності та автентичності сертифікату на основі його цифрового файлу чи відповідного QR-коду.

Системні вимоги вебзастосунку:

1. Сумісність з найпопулярнішими веббраузерами (Google Chrome, Mozilla Firefox, Safari, Microsoft Edge, Opera).
2. Можливість розгортання застосунку на операційних системах Linux (Ubuntu, CentOS) або Windows.
3. Використання криптографічних методів захисту інформації для підвищення рівня безпеки застосунку.

Додаткові вимоги вебзастосунку:

1. Ергономічний користувацький інтерфейс.
2. Використання цифрового підпису на основі еліптичних кривих.
3. Впровадження механізмів безперервної інтеграції та розгортання застосунку.

5. ВИМОГИ ДО ПРОЄКТНОЇ ДОКУМЕНТАЦІЇ

У процесі виконання проекту повинна бути розроблена наступна документація:

1. Пояснювальна записка.

2. Програма та методика тестування.
3. Керівництво користувача.
4. Креслення:
 - а. “Структура бази даних. ER-діаграма”.
 - б. “Алгоритм випуску цифрових сертифікатів. Блок-схема алгоритму”.

6. ЕТАПИ ПРОЄКТУВАННЯ

Вивчення літератури за тематикою роботи.....	16.11.2023
Розроблення та узгодження технічного завдання.....	27.11.2023
Розроблення структури вебзастосунку.....	15.12.2023
Розроблення дизайну сторінок та графічних елементів.....	05.02.2024
Програмна реалізація вебзастосунку.....	15.03.2024
Тестування вебзастосунку.....	08.04.2024
Підготовка матеріалів текстової частини проєкту.....	29.04.2024
Підготовка матеріалів графічної частини проєкту.....	15.05.2024
Оформлення технічної документації проєкту.....	28.05.2024

7. ПОРЯДОК ТЕСТУВАННЯ РОЗРОБКИ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Факультет прикладної математики
Кафедра програмного забезпечення комп'ютерних систем

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Євгенія СУЛЕМА

“ ____ ” _____ 2024 р.

ВЕБЗАСТОСУНОК ДЛЯ ВИПУСКУ ЦИФРОВИХ СЕРТИФІКАТІВ
НА ОСНОВІ ТЕХНОЛОГІЇ БЛОКЧЕЙН

Пояснювальна записка

ДП.045440-03-81

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Оксана ШКУРАТ

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Максим СЛОБОДЗЯН

2024

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ.....	3
ВСТУП.....	6
1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА ОБҐРУНТУВАННЯ ТЕМИ ДИПЛОМНОГО ПРОЄКТУ.....	7
1.1. Аналіз особливостей випуску цифрових сертифікатів.....	7
1.2. Аналіз існуючих програмних рішень.....	10
1.3. Вимоги до вебзастосунку для випуску цифрових сертифікатів.....	16
1.4. Висновки до розділу 1.....	18
2. ОБҐРУНТУВАННЯ ВИБОРУ ЗАСОБІВ РЕАЛІЗАЦІЇ ВЕБЗАСТОСУНКУ.....	20
2.1. Вибір блокчейн-платформи.....	21
2.2. Вибір технологій для розроблення серверної частини.....	25
2.3. Вибір технологій для розроблення клієнтської частини.....	32
2.4. Вибір СКБД.....	35
2.5. Висновки до розділу 2.....	37
3. РОЗРОБЛЕННЯ ВЕБЗАСТОСУНКУ ДЛЯ ВИПУСКУ ЦИФРОВИХ СЕРТИФІКАТІВ.....	39
3.1. Загальний опис архітектури програмного забезпечення.....	39
3.2. Структура серверної частини.....	43
3.3. Структура клієнтської частини.....	51
3.4. Структура бази даних.....	54
3.5. Висновки до розділу 3.....	58
4. АНАЛІЗ РЕАЛІЗАЦІЇ ВЕБЗАСТОСУНКУ.....	59
4.1. Тестування вебзастосунку.....	59
4.2. Безперервнi інтеграція та розгортання застосунку.....	62
4.3. Рекомендації щодо подальшого вдосконалення.....	64
4.4. Висновки до розділу 4.....	66
ВИСНОВКИ.....	67
СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ.....	68
ДОДАТКИ.....	71

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

ACID – Atomicity, Consistency, Isolation, Durability, властивості роботи транзакцій у базі даних.

Amazon S3 – Amazon Simple Storage Service, сервіс для швидкого та простого зберігання різноманітних об'єктів від компанії Amazon Web Services, що має великий рівень масштабованості, доступності та безпеки системи.

API – Application Programming Interface, набір правил та інструкцій для комунікації між різними програмами.

CI/CD – Continuous Integration/Continuous Development, набір практик автоматизованих процесів збірки, тестування та розгортання програмного забезпечення для його швидкої та надійної розробки.

DOM – Document Object Model, програмний інтерфейс для роботи зі структурованими елементами.

ECDSA – Elliptic Curve Digital Signature Algorithm, алгоритм цифрового підпису з використанням відкритого ключа, що базується на використанні еліптичних кривих.

EVM – Ethereum Virtual Machine, обчислювальний Тюринг-повний механізм блокчейн-мережі Ethereum.

HTML – HyperText Markup Language, мова розмітки документів для перегляду вебсторінок у браузері.

HTTP – HyperText Transfer Protocol, протокол передачі даних.

JavaScript – динамічна мультипарадигмова мова програмування, що широко використовується для створення вебзастосунків.

JSON – JavaScript Object Notation, текстовий формат обміну даними, що використовує нотацію мови програмування JavaScript.

JSX – JavaScript XML, розширення мови програмування JavaScript з використанням формату даних XML.

JWT – JSON Web Token, стандарт токенів доступу до системи, що ґрунтується на форматі даних JSON.

KYC – Know Your Customer, процедура ідентифікації та встановлення особи користувача застосунку.

Nginx – незалежний проксі-сервер, що призначений для обслуговування статичних запитів та індексних файлів, балансування навантаження, проксування запитів тощо.

PDF – Portable Document Format, універсальний формат файлів для збереження будь-яких документів.

PKC – Public Key Cryptography, криптографічна система, у якій використовується два відповідних ключі: відкритий (публічний) та закритий (приватний).

Polygon – EVM-сумісна блокчейн-платформа, що підтримує виконання смарт-контрактів та націлена на створення багатомережевого блокчейну.

QR-код – Quick Response code, двовимірний штрих-код, що містить закодовану корисну інформацію та може бути легко розпізнаний різними сканувальними апаратами декодування.

RPC – Remote Procedure Call, протокол виклику віддалених процедур.

SQL – Structed Query Language, декларативна мова програмування, що використовується для формування різноманітних запитів створення, оновлення і керування схемами та записами реляційних баз даних.

URL – Uniform Resource Locator, стандартизована адреса певного ресурсу.

UTXO – Unspent Transaction Output, концепція у блокчейн-мережах, що означає загальну суму монет, яка доступна до використання власником.

XML – EXtensible Markup Language, мова розмітки для обміну даних в мережах.

ZKP – Zero Knowledge Proof, криптографічний метод доведення істинності твердження без розкриття будь-якої іншої інформації.

Біндинг – (від англ. Binding) зв'язування двох компонент, мов програмування, функцій, методів між собою. В контексті даної роботи – згенерований Go-модуль зв'язки для взаємодії зі смарт-контрактами.

Блокчейн – реєстр впорядкованого ланцюгу блоків послідовних транзакцій системи.

ООП – об'єктно-орієнтоване програмування, парадигма програмування, за якої програма розглядається як множина взаємозалежних об'єктів.

ПЗ – програмне забезпечення.

Проксі-сервер – мережевий сервер, що дозволяє користувачам виконувати непрямі запити до ресурсів інших серверів.

СКБД – система керування базою даних.

Смарт-контракт – комп'ютерний алгоритм, розроблений за допомогою спеціальних мов програмування, що містить набір правил, функцій, умов і призначений для їх автоматичного виконання у мережі.

Фреймворк – (від англ. Framework) набір бібліотек та інструментів, що використовуються для створення програмних застосунків.

ВСТУП

У сучасному світі дистанційне навчання та онлайн-курси, стрімка популяризація яких спричинена як глобальною пандемією COVID-19, так і веденням правового режиму воєнного стану в Україні, стали незамінними складовими елементами освітнього процесу. Вони відіграють ключову роль у забезпеченні доступності та гнучкості навчання. За статистикою, близько 70% студентів надають перевагу онлайн навчанню над традиційною системою освіти [1]. Разом з розвитком онлайн навчання зростає й важливість цифрової сертифікації учнів та студентів, як підтвердження рівня їх знань та компетентностей.

Традиційні методи видачі та перевірки паперових документів часто виявляються недостатньо швидкими, зручними та ефективними, а самі документи важко верифікувати. На заміну застарілих практик та стандартів приходить ера цифрових технологій. Серед переваг сучасних підходів можна виділити як зручність і гнучкість у процесі випуску сертифікатів, так і безпеку, цілісність та швидкість зі сторони верифікації даних. Проте, на сьогодні не існує єдиного рішення, що рівномірно об'єднує у собі всі вищезгадані характеристики. Деякі застосунки можуть бути менш ефективними у забезпеченні прозорості та безпеки даних, інші можуть виявитись недостатньо гнучкими у процесі налаштування та адаптації до користувацьких потреб.

Даний дипломний проєкт присвячений розробленню інноваційного цифрового рішення, що відповідає вимогам сучасності та вирішує проблеми, пов'язані з видачею та перевіркою сертифікатів. Мета дипломного проєкту – поєднати усі переваги цифрових технологій, забезпечуючи ефективний, зручний, прозорий і безпечний процес видачі та верифікації сертифікатів шляхом створення відповідного програмного забезпечення.

1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА ОБҐРУНТУВАННЯ ТЕМИ ДИПЛОМНОГО ПРОЄКТУ

1.1. Аналіз особливостей випуску цифрових сертифікатів

Сертифікування осіб є одним із ключових освітніх процесів як підтвердження рівня їх знань. За останні роки спостерігається значний ріст кількості різноманітних курсів, лекцій та тренінгів, які проводяться онлайн. Це створює попит на більш ефективні та сучасні методи видачі сертифікатів, які відповідають цифровим реаліям.

Діджиталізація процесу випуску сертифікатів значно полегшує цей процес – вона дозволяє зменшити час і зусилля, потрібні для оформлення та оброблення документації. Замість традиційних методів, таких як ручне заповнення та оброблення паперових документів, цифрові сертифікати можуть бути створені, підписані та відправлені автоматично з використанням спеціалізованих програмних засобів. Також важливою перевагою цифрових сертифікатів є їхня зручність для зберігання та подальшого використання, оскільки вони можуть бути легко доступні у електронному вигляді на різних пристроях і забезпечують більшу мобільність та доступність для користувачів.

Важливим аспектом сертифікатів є їхня верифікація, тобто перевірка автентичності та цілісності даних. Це гарантує, що документ був виданий відповідною стороною і конкретній особі, а також те, що дані у ньому не були змінені або пошкоджені. Додатковою перевагою цифрових сертифікатів є різні способи їх верифікації – використання QR-кодів з попередньо закодованою інформацією для ідентифікації документа в системі, гіперпосилань, що ведуть на сторінку верифікації, тощо. Такий підхід дозволяє не тільки швидко та зручно перевірити інформацію про той чи інший сертифікат, а й незалежно перевіряти сертифікати без необхідності звертатися до тієї самої організації чи особи, що їх видавала.

Це особливо корисно в ситуаціях, де потрібно перевірити дійсність сертифікатів без прямого контакту з їхнім власником або видавцем.

Грунтуючись на особливостях випуску цифрових сертифікатів, можна виділити дві головні характеристики, яким повинен задовільняти процес: зручність випуску та надійність і швидкість верифікації.

Зручність випуску цифрового сертифікату полягає у максимальній адаптації програмного застосунку під потреби користувачів для легкого та швидкого випуску документів без зайвих перешкод або складнощів, зокрема:

- Наявність інтуїтивного та простого інтерфейсу, який дозволяє заповнити необхідну інформацію та отримати сертифікат лише за кілька кроків.
- Вибір різних параметрів сертифіката та налаштування їх відповідно до потреб користувача.
- Швидкий та зручний випуск сертифікатів як для однієї, так і для багатьох осіб одночасно.
- Широкі можливості кастомізації шаблонів сертифікату.

Надійність і швидкість верифікації цифрових сертифікатів є критично важливими аспектами для забезпечення довіри до цих документів. Процес верифікації повинен бути надійним і забезпечувати точність та цілісність інформації у сертифікатах. Це означає, що система повинна гарантувати, що дані у сертифікаті не були змінені або підроблені. До того ж, верифікаційний процес повинен бути ефективним і швидким, щоб забезпечити миттєвий доступ до інформації про сертифікати. Використання різноманітних криптографічних методів дозволяє забезпечити досить високий рівень безпеки та надійності процесу випуску та верифікації цифрових сертифікатів. Цифрові підписи дозволяють підтверджувати автентичність та цілісність даних у сертифікатах. Вони гарантують, що інформація у сертифікаті не була змінена після його підписання випускаючим органом. Шифрування використовується для

захисту конфіденційності даних у сертифікатах під час їх передачі або зберігання. Криптографічні хеш-функції використовуються для створення унікального "відбитка" даних. Порівняння отриманого хешу з оригінальним значенням дозволяє перевірити цілісність даних.

Застосування технології блокчейну [2] у випуску цифрових сертифікатів відкриває нові можливості для забезпечення достовірності, безпеки та обліку інформації. Основною перевагою блокчейну як способу збереження даних є так званий ланцюжок блоків – кожна транзакція у системі записується в хронологічному порядку у вигляді блоків, кожен наступний з яких містить хеш-значення попереднього. Такий підхід до запису даних унеможливорює їх зміну чи часткове видалення, а у випадку децентралізованого рішення, також гарантує максимальну безпеку та незмінність інформації, оскільки кожен блок даних підтверджується всією мережею вузлів системи.

У випадку випуску цифрових сертифікатів, інформація про кожен сертифікат може бути занесена в блокчейн, де вона залишиться незмінною та доступною для перевірки усіма зацікавленими сторонами. Крім того, завдяки розподіленій природі блокчейну, інформація про сертифікати буде резервованою на кількох вузлах мережі, що забезпечить її стійкість до втрати та впливу зовнішніх факторів. Також, коли документ занесено до блокчейну, будь-яка зацікавлена сторона може перевірити його автентичність, просто перевіряючи відповідний запис у реєстрі. Це дозволяє швидко та ефективно перевіряти сертифікати без необхідності звертатися до централізованих органів або третіх сторін.

Отже, процес випуску цифрових сертифікатів повинен бути практичний та зручний для користувача, а сертифікат – захищеним від підробок. Верифікація документу повинна бути швидка для перевіряючої сторони без необхідності контакту з видавцем. Доволі важливим аспектом є гармонійне поєднання вищенаведених характеристик процесу, оскільки рішення що не гарантують безпеку і цілісність даних сертифікату, а також

мінімальний ризик його підробки, є недопустимими. Відповідно, навіть якщо сертифікат надійно захищений криптографічними методами захисту, але процес його випуску занадто громіздкий, не автоматизований, без інтуїтивно простого та зрозумілого інтерфейсу, користувачам буде складно користуватись таким застосунком.

1.2. Аналіз існуючих програмних рішень

Аналіз існуючих програмних рішень для випуску цифрових сертифікатів включає огляд різноманітних платформ та сервісів, які надають можливість створювати, підписувати та розповсюджувати ці документи. Це є важливий етап процесу розроблення програмного застосунку, оскільки аналіз дозволяє не тільки дослідити існуючі рішення та їх переваги на ринку, а й виявити недоліки та проблеми у системах. Проаналізувавши предметну область та наявні програмні рішення, можна зрозуміти які функції та характеристики будуть найбільш цінними для цільової аудиторії, а також запропонувати нові функції, які ще не було реалізовано, що покращуватимуть користувацький досвід і ефективність застосунку.

При аналізі існуючих програмних рішень випуску цифрових сертифікатів слід звернути увагу на такі ключові параметри:

- Інтерфейс користувача повинен бути інтуїтивно зрозумілим та зручним для використання, щоб спростити процес випуску сертифікатів та зменшити його часові та трудові витрати.
- Високий рівень безпеки, зокрема застосування криптографічних методів для захисту конфіденційності та цілісності даних, а також захист від несанкціонованого доступу та підробок.

Додатковою перевагою при розгляді рівня довіри та безпеки застосунку є використання технології блокчейну. Це підвищує надійність і прозорість системи, а сам підхід дозволяє будь-якій зацікавленій стороні перевірити автентичність сертифікату безпосередньо через блокчейн, без

необхідності звертатися до централізованих джерел або довіряти посередникам. Окрім цього, у випадку децентралізації додатку, особа, що хоче впевнитись в цілісності та достовірності сертифікату, не змушена довіряти застосунку і що дані не було змінено чи підроблено саме на рівні системи, оскільки уся історія даних міститься на публічній мережі, і будь-хто має змогу впевнитись, що особа є автором конкретного документу, і що документ належить саме тому чи іншому учневі.

На сьогодні існує чимало різноманітних сервісів для створення та верифікації цифрових сертифікатів, що мають як доволі схожі, так і відмінні функціональні можливості. До розгляду та аналізу оберемо найпопулярніші рішення. Також оберемо ті застосунки, що не вимагають додаткової процедури KYC [3], оскільки такі рішення вимагають підтвердження особи з використанням приватної інформації. Отже, розглянемо наступні сервіси: Certifier, DoxyChain, Accredible.

1.2.1. *Certifier*

Certifier [4] – популярний сервіс для створення та поширення цифрових сертифікатів, що надає широкі можливості для зручного керування процесом випуску сертифікатів. Це перший вибір для багатьох користувачів, які шукають ефективний, практичний і зручний спосіб створення та розповсюдження цифрових сертифікатів. На відміну від деяких інших сервісів, Certifier пропонує розширені можливості налаштування шаблонів сертифікатів, що дозволяє користувачам адаптувати їх до конкретних потреб та вимог.

Серед переваг сервісу можна виділити:

- Великий вибір готових до використання шаблонів сертифікатів з різноманітним дизайном під різні потреби.
- Автоматизований та зручний випуск великої кількості сертифікатів одночасно.

- Додавання власних полів та атрибутів до документу, що робить їх випуск максимально налаштованим під потреби конкретної галузі та користувачів.
- Інтеграція з іншими відомими застосунками, серед яких Google Sheets, LinkedIn, Zoom, Zapier, тощо.

На рис. 1.1 наведено приклад процесу дизайну сертифікату з використанням власних атрибутів й інструментарію платформи.

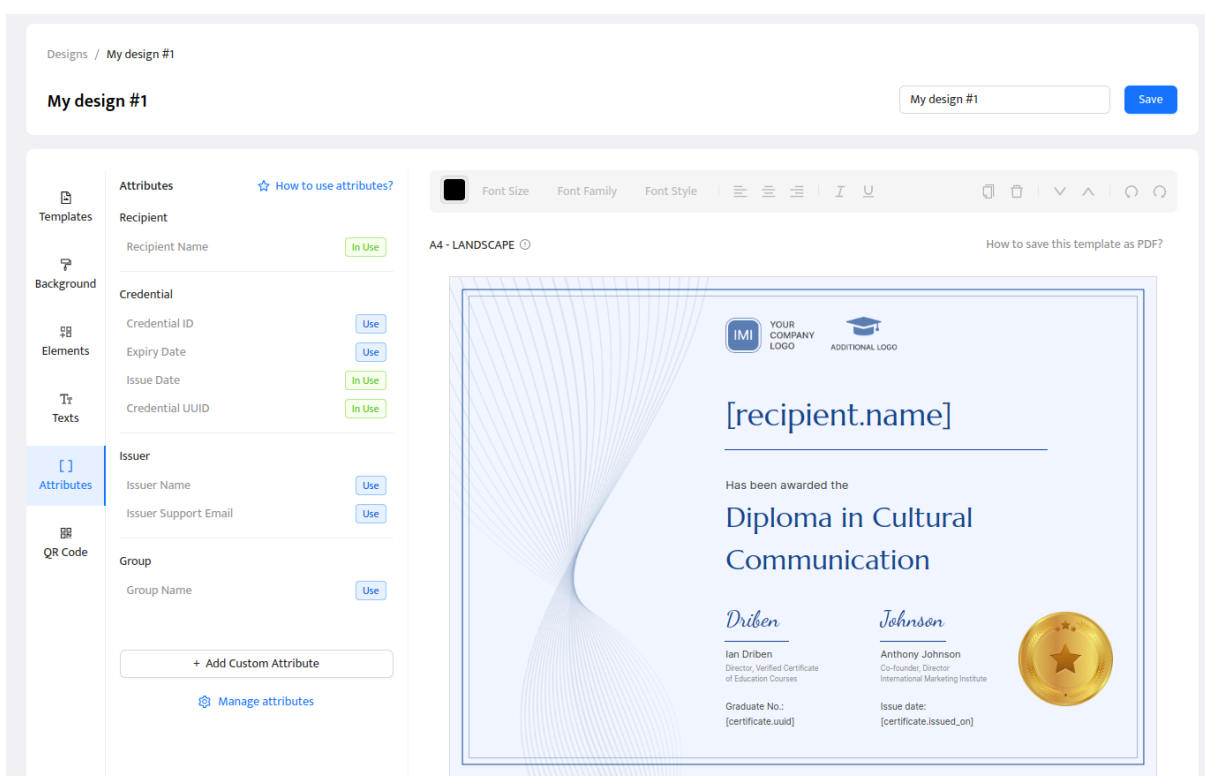


Рис. 1.1. Дизайн сертифікату за допомогою інструментарію Certifier

Серед недоліків системи в першу чергу слід звернути увагу на те, що сертифікати, випущені цією платформою, не можуть бути перевірені на автентичність та цілісність у разі втрати даних чи збою у системі. Це створює ризик втрати довіри до сертифікатів та надійності системи в цілому. Окрім цього, особа, яка хоче перевірити справжність сертифікату, повинна довіряти платформі оскільки невідомо, чи дані було змінено/видалено адміністратором чи модератором сервісу. Доволі важливою є зручність верифікації даних: платформа у безкоштовній версії

не дозволяє розмістити гіперпосилання/QR-код на документі, що буде містити дані про нього та посилання на сторінку верифікації. Це сповільнює процес, оскільки особі доводиться спочатку знаходити відповідний застосунок для верифікації, а потім вводити необхідні дані вручну, в процесі чого можуть статись випадкові помилки і тоді справжній сертифікат може бути відхилений системою перевірки. Також платформа не надає можливості створювати та випускати сертифікати різним учасникам, що можуть працювати в одній групі.

1.2.2. DoxyChain

DoxyChain [5] – платформа для випуску і захисту сертифікатів та облікових даних, що базується на технології блокчейну. Вона дозволяє випускати, підписувати, керувати документами в режимі реального часу, встановлювати термін їх дії чи відкликати їх, перевіряти статус і автентичність сертифікатів. Доступні системи моніторингу та аналітики випуску сертифікатів.

Використання цифрових підписів, перевірки цілісності файлу, а також застосування технології блокчейну для прозорості обліку даних роблять платформу надійною та безпечною, з великим рівнем довіри до неї.

Додатковою перевагою сервісу є можливість спільного випуску сертифікатів, що є корисним у випадках коли особою, що випускає сертифікат, можуть бути різні фахівці.

Алгоритм верифікації сертифікату є дуже простим: особа може на вибір або відсканувати QR-код, що присутній на сертифікаті, або ж завантажити файл для знаходження контрольної суми, ідентифікатора документу та перевірки його цілісності. Інтерфейс перевірки сертифікату на платформі DoxyChain зображено на рис. 1.2.

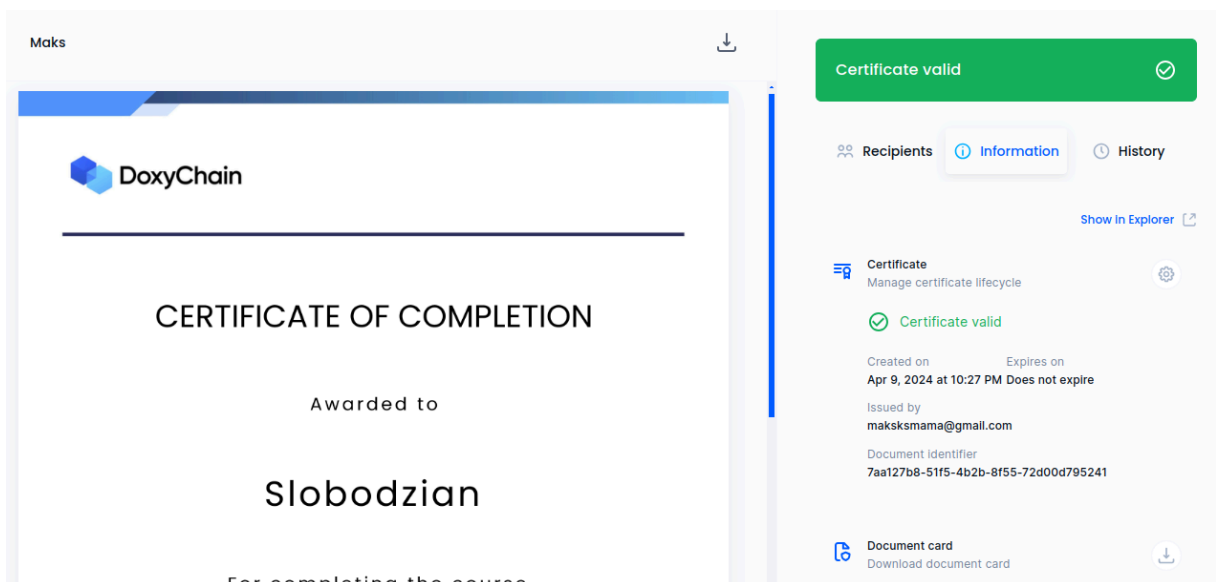


Рис. 1.2. Сторінка верифікації сертифікату за допомогою DoxyChain

Слід зазначити, що система, окрім своїх переваг, містить ряд помітних недоліків. Вони негативно впливають на загальну працездатність та сферу застосування системи, роблячи її менш привабливою для користувачів.

По перше, відсутня автоматизація процесу випуску сертифікатів та імпорту даних: користувачу доведеться кожен раз вручну вводити дані про кожен сертифікат окремо. Також відсутній випуск одразу декількох сертифікатів одночасно, що значно уповільнює та ускладнює процес.

По друге, система не підтримує додаткові динамічні атрибути сертифікату, що значно звужує сфери використання застосунку. Це може бути додаткова інформація про спеціалізацію або кваліфікації особи, додаткові відомості про навчальний курс чи тренінг або будь-яка інша різна інформація, яка може бути корисною для визначених ситуацій або застосувань. Таким чином, відсутність можливості додавати додаткові атрибути може обмежити гнучкість та універсальність системи.

Також немає змоги створити власний шаблон сертифікату, а їх дизайн містить рекламу платформи і значним чином обмежує індивідуалізацію та професійний вигляд документу.

1.2.3. *Accredible*

Accredible [6] – платформа для створення, видачі та керування безпечними цифровими сертифікатами та відзнаками. Система також базується на технології блокчейн, тому рівень безпеки та надійності застосунку доволі високий.

Як і попередні застосунки, Accredible надає широкі можливості та різноманітні інструменти у сфері випуску цифрових документів, серед яких:

- Дизайн сертифікатів з використанням різноманітних динамічних атрибутів, зарезервованих системою.
- Можливість імпорту даних та випуску декількох сертифікатів одночасно.
- Інтеграція з іншими застосунками для дизайну, створення, випуску та публікації сертифікатів.
- Кастомізація фірмових цільових сторінок, електронних листів, URL-адрес, якими поширюються сертифіковані облікові дані користувача.
- Статистичні дані про випуск, перегляд, поширення, верифікацію даних, що були випущені за певний період.

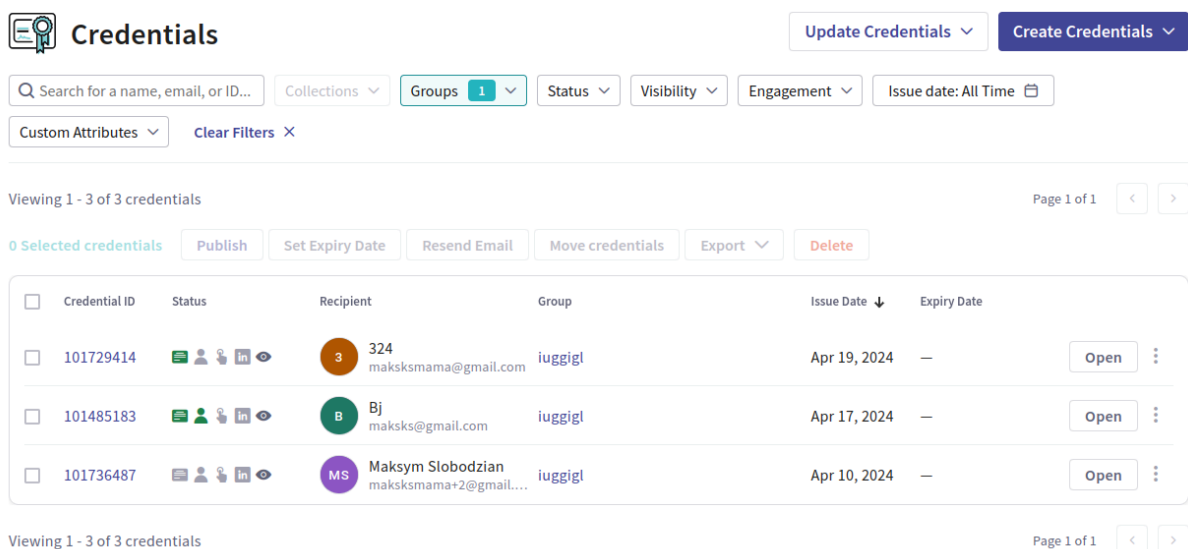


Рис. 1.3. Інтерфейс застосунку Accredible

Платформа надає використання можливостей розширеної функціональності в залежності від різних тарифних планів підписки. Сюди входять покращені рішення авторизації користувачів, доступ до API застосунку, служба технічної підтримки, тренувальні та ознайомчі програми платформи тощо.

Серед проблем застосунку можна виділити відсутність формування власних динамічних атрибутів для сертифікатів та незручний інтерфейс випуску та верифікації сертифікату.

Недоліком також є те, що для додавання даних в мережу блокчейну системі потрібно до 48-ми годин, що може вплинути на процес верифікації та викликати недовіру до платформи.

1.3. Вимоги до вебзастосунку для випуску цифрових сертифікатів

У результаті аналізу особливостей випуску цифрових сертифікатів було встановлено ключові вимоги та характеристики, яким повинен відповідати розроблюваний застосунок. Аналіз існуючих програмних рішень допоміг ознайомитися з найкращими практиками та проблемами на ринку, що дозволить забезпечити ефективніший розвиток нового застосунку.

Для більш глибокого аналізу необхідно сформулювати чіткі вимоги до програмного застосунку, що дозволять об'єктивно порівняти його з іншими рішеннями, визначити його сильні та слабкі сторони, а також оцінити відповідність вимогам та цілям проєкту. Наступними критеріями для розроблюваної системи є:

- Зручність користувацького інтерфейсу – система повинна мати інтуїтивно зрозумілий та практичний інтерфейс для полегшення і пришвидшення випуску цифрових сертифікатів та їх подальшої верифікації.
- Можливість індивідуального проєктування шаблону сертифікатів – користувачі системи повинні мати змогу самостійно розроблювати

дизайн сертифікату чи вибрати його з-поміж готових дизайнерських рішень.

- Можливість імпорту даних та випуск декількох сертифікатів одночасно – система дозволяє імпортувати існуючі масиви даних для заповнення шаблонів сертифікату і підтримує масовий їх випуск.
- Можливість додавання власних динамічних атрибутів документа – користувачі повинні бути в змозі додавати різні динамічні змінні до своїх сертифікатів, які розширюють сферу використання застосунку.
- Можливість доступу до випуску одного сертифікату різних учасників групи – система повинна підтримувати авторизований доступ та дозволи різних користувачів до спільних груп курсів, у межах яких випускаються ті чи інші сертифікати.
- Безпека та цілісність даних – система повинна забезпечувати безпеку даних та їх цілісність, а також можливість відновлення критично важливих даних для перевірки у разі збою.
- Швидкий та зручний процес верифікації даних – система та сам цифровий сертифікат повинні бути спроектовані таким чином, що процес верифікації даних може бути здійснено швидко та зручно для особи що перевіряє, а також без необхідності контакту з видавцем.

На основі сформованих вимог порівняємо характеристики існуючих програмних рішень. Це дозволить нам об'єктивно оцінити кожную систему та допоможе виявити спільні переваги та недоліки між різними застосунками, а також аспекти, що потребують покращень. Результати порівняння проаналізованих програмних рішень за вимогами до розроблюваного застосунку представлено у табл. 1.1.

Порівняльна характеристика існуючих рішень

Критерій	Сервіси		
	Certifier	DoxyChain	Accredible
Зручність користувацького інтерфейсу	+	+	+/-
Можливість дизайну сертифікатів	+	-	+
Можливість імпорту даних та випуск декількох сертифікатів одночасно	+	-	+
Можливість доступу до випуску одного сертифікату різних видавців з однієї групи	-	+	-
Можливість додавання власних динамічних атрибутів документа	+	-	-
Безпека та цілісність даних	+/-	+	+/-
Швидкий та зручний процес верифікації даних	-	+	+

Після порівняння існуючих програмних застосунків можна зробити висновок, що розглянуті системи не реалізують у повній мірі бажані функціональні можливості застосунку. Проаналізовані застосунки одночасно у повній мірі не задовольняють зручність випуску цифрових сертифікатів та високий рівень безпеки, надійність і швидкість верифікації, чого потребує дана предметна область.

Рішенням даних проблем є створення вебзастосунку, функціональність якого орієнтована саме на поєднанні практичності та зручності випуску непідроблюваних цифрових сертифікатів з можливістю їх надійної та швидкої верифікації. Проведений аналіз допоможе розробити оптимальне рішення, що відповідає потребам користувачів в даній області.

1.4. Висновки до розділу 1

У результаті проведеного аналізу особливостей випуску цифрових сертифікатів було встановлено важливі аспекти процесу, властивості, яким повинна задовольняти система та потребу в оптимальному поєднанні зручності для користувача та надійності системи. Також було розглянуто переваги використання технології блокчейну для забезпечення безпеки і цілісності даних.

Під час дослідження існуючих програмних рішень було розглянуто різноманітні застосунки, які використовуються для випуску цифрових сертифікатів. Було проаналізовано основні переваги та недоліки кожної системи.

На основі аналізу предметної області та конкурентних застосунків було сформовано перелік вимог, яким повинна відповідати система. Провівши порівняльну характеристику існуючих рішень, було встановлено, що жодна з розглянутих систем у повній мірі не задовольняє всім вимогам. Було прийнято рішення розробити програмний застосунок, що враховуватиме переваги та недоліки існуючих систем та відповідатиме поставленим вимогам.

2. ОБҐРУНТУВАННЯ ВИБОРУ ЗАСОБІВ РЕАЛІЗАЦІЇ ВЕБЗАСТОСУНКУ

Вибір засобів для реалізації вебзастосунку є ключовим етапом у процесі його розроблення. Від цього залежить якість, продуктивність, ефективність та безпека майбутнього застосунку. Неправильний вибір технологій може призвести до проблем у роботі системи, її низької продуктивності, незручностей для користувачів та потенційних загроз безпеці. Тому важливо здійснювати обґрунтований вибір технологій, що враховує вимоги, яким повинна задовольняти система, потреби користувачів, а також майбутні перспективи розвитку.

Застосунок для випуску цифрових сертифікатів доцільно розділити на дві основні частини: клієнтську та серверну. Цей підхід забезпечує оптимальну та чітку організацію і розподіл функціональних можливостей між компонентами системи. Також це забезпечить більшу гнучкість у її розробленні та підтримці.

Клієнтська частина застосунку відповідає за інтерфейс користувача та взаємодію з ним, тоді як серверна частина – за оброблення запитів користувачів та бізнес-логіку застосунку. Система, а саме її серверна частина, повинна забезпечувати надійне зберігання та оброблення даних за допомогою СКБД. Для забезпечення максимальної безпеки, надійності, цілісності даних та їх прозорості відповідає модуль застосунку, що базується на технології блокчейну.

Оскільки безпека, доступність та цілісність даних є однією з головних вимог застосунку, а вибір різноманітних застосунків для побудови модулю системи на основі блокчейну є досить широким, то для початку необхідно ретельно розглянути різні блокчейн-платформи. На основі вибору блокчейн-платформи можна обрати найкращі сумісні інструменти для розроблення клієнтської та серверної частин застосунку, а також спроектувати гнучку та ефективну архітектуру системи.

2.1. Вибір блокчейн-платформи

Критеріїв вибору конкретної платформи для розроблення блокчейн-модулю системи є доволі багато: масштабованість, тип системи, продуктивність, інструменти розроблення, підтримка різних протоколів, тощо. Оскільки потрібно забезпечити безпеку та цілісність даних водночас з вільною доступністю та можливістю верифікації, особливу увагу потрібно зосередити саме на типі блокчейну та протоколах, що підтримує блокчейн-платформа.

2.1.1. Тип блокчейну

Серед типів блокчейну можна виділити наступні три види: приватні, публічні та консорціумні. Кожен з них має свої властивості та застосування, які варто розглянути перед вибором конкретної платформи.

Публічний блокчейн

На сьогодні публічні блокчейни переважають у більшості реєстрів та мають таку назву через певні особливості – будь-хто має змогу переглядати, верифікувати та вносити транзакції в систему. Всі дані реєстру є загальнодоступними, а до її роботи та участі у механізмах валідації блоків та консенсусу системи може доєднатися кожен. Це не тільки забезпечує максимально можливий рівень прозорості та цілісності даних, а й створює децентралізовану мережу, де немає єдиної особи/групи осіб, що володіє більш ширшими правами та в змозі змінювати дані у будь-який спосіб. Також учасники такої мережі не повинні довіряти одне одному, оскільки вони можуть повністю покластись на правила протоколу системи.

Попри переваги публічних блокчейнів, їм притаманні незначні проблеми продуктивності, оскільки пропускна транзакційна здатність системи зменшується при збільшенні активних учасників у мережі. Також доволі складним може бути внесення змін та покращень до протоколу

застосунку, оскільки це потребує згоди та підтвердження більшої частини його учасників.

Приватний блокчейн

На відміну від публічного блокчейну, приватні блокчейни чітко визначають правила щодо перегляду та запису даних в систему. Такий тип системи дозволяє контролювати учасників та позбуватись потенційних зловмисників. Властивість децентралізованості при цьому втрачається, хоча система досі залишається розподіленою, оскільки її учасники досі зберігають локальні копії на своїх пристроях.

Приватні блокчейни найкраще підходять для систем, що потребують властивостей, які надає технологія блокчейну без необхідності робити саму систему, що доступна ззовні. До того ж, оскільки набір учасників системи обмежений, швидкість транзакцій у приватному блокчейні є дуже високою.

Консорціумний блокчейн

Даний тип блокчейну поєднує у собі властивості попередніх видів. Замість повністю відкритої чи навпаки закритої системи, консорціумні блокчейни можуть встановлювати гнучкі правила, де видимість даних може бути обмежена на різних рівнях чи взагалі не обмежуватись ніяким чином.

Такий тип блокчейнів найбільше підходить для спрощення взаємодії між кількома різними організаціями. Транзакційно-пропускна здатність системи, аналогічно до приватних блокчейнів, залишається доволі високою.

2.1.2. Протокол платформи

Різні блокчейн-технології охоплюють відмінний набір правил, процедур та механізмів, які регулюють функціонування самої платформи та взаємодію її складових частин. Протокол блокчейну визначає правила, за

якими відбувається створення, підтвердження та запис блоків у ланцюжок, а також роботу смарт-контрактів – програм для виконання певної бізнес логіки поверх системи блокчейну. Вони відкривають широкі можливості для розвитку різних додатків на блокчейні.

Дедалі популярнішими у використанні стають EVM-сумісні блокчейн-системи, де EVM [7] – середовище виконання для смарт-контрактів в Ethereum [8]. Воно робить виконання смарт-контрактів більш надійним та безпечним.

Отже, для аналізу протоколів систем необхідно звернути увагу на EVM-сумісні та несумісні платформи.

EVM-сумісні блокчейн-платформи

Системи, що підтримують середовище виконання смарт-контрактів в Ethereum, дозволяють не тільки розробляти різні застосунки поверх технології блокчейн, а й роблять їх сумісними з іншими системами цього виду. Це означає, що будь-який додаток, побудований на смарт-контрактах, може бути легко перенесений на ту чи іншу платформу без необхідності зміни будь-якого програмного коду. До того ж, віртуальна машина надає змогу розробникам писати високорівневий програмний код на різних мовах програмування, компілюючи його в машинний код. Це забезпечує гнучкість для розробників і дозволяє їм вибирати мову програмування, яка найкращим чином підходить для їхніх потреб і вмінь. Такий підхід робить розроблення смарт-контрактів більш доступною та придатною для широкого кола розробників.

EVM-несумісні блокчейн-платформи

Окрім систем, що використовують EVM, існують платформи, що можуть підтримувати віртуальні машини для розроблення смарт контрактів, або ж взагалі мати зовсім інші правила роботи з бізнес логікою додатків. Доволі часто такі системи використовуються для

вузькоспеціалізованих задач та розв'язання конкретних проблем. Наприклад, різні системи, що працюють на концепції UTXO (найпопулярніша серед них – усім відома мережа Bitcoin [9]) вирішують проблему безпечності та надійності обміну криптовалютою, висока пропускна здатність і масштабованість Solana [10] дозволяють мережі обробляти величезний обсяг транзакцій тощо. Недоліками ж цих систем є те, що через різні правила протоколу екосистеми додатків, в загальному випадку, вони не є сумісними з іншими.

2.1.3. Результати аналізу блокчейн-платформ

У ході аналізу блокчейн-платформ було розглянуто їх різні типи та протоколи систем. Відповідно до основних вимог системи за типом блокчейну доцільно обрати публічний блокчейн, оскільки повинна забезпечуватись максимальна безпека, цілісність даних та прозорість системи. Кожна особа, що бажає верифікувати сертифікат, не повинна довіряти розробленому додатку, що дані не було змінено чи видалено, а саме публічний блокчейн дозволить вільно та швидко перевірити це. Щодо протоколів платформи, EVM-сумісні системи є хорошим рішенням для швидкого та ефективного розроблення модуля системи. Сумісність зі схожими системами дозволить легко та швидко перемістити додаток на іншу платформу у разі можливих збоїв системи.

Серед наявних рішень публічного EVM-сумісного блокчейну було обрано мережу Polygon [11]. Перевагою у виборі між іншими схожими платформами стала порівняно висока пропускна здатність – один блок даних транзакцій формується за 3 секунди, у той час як термін формування блоку в інших системах – від 10 секунд та більше. Також, комісія за транзакції у цій мережі є в рази меншою, ніж в аналогічних EVM-сумісних мережах.

2.2. Вибір технологій для розроблення серверної частини

Для вибору засобів розроблення серверної частини системи необхідно врахувати особливості її архітектури та функціональних вимог. Оскільки планується використання платформи блокчейну, доцільно розділити сервер застосунку на централізовану та децентралізовану частини. Відповідно, для кожної з цих частин необхідно обрати ті технології, які допоможуть забезпечити високу продуктивність, масштабованість, безпеку та зручність розроблення.

2.2.1. Технології розроблення децентралізованої частини

Децентралізована частина додатку відповідає за модуль системи, що діє на основі блокчейну, і представлена набором смарт-контрактів. Відповідно для її розроблення необхідно вибрати мову програмування, яка надає найбільш зручні та ефективні засоби для роботи з смарт-контрактами. Також доволі важливо звернути увагу на наявні фреймворки, документацію, спільноту учасників, що допоможе для швидкого та зручного розроблення модулю.

Solidity

Solidity [12] – статично типізована мова програмування високого рівня, що спеціально призначена для розроблення смарт-контрактів на платформі Ethereum та інших системах, що є сумісними з даною системою. Вона є одним з найпопулярніших виборів розроблення блокчейн-додатків, оскільки є стабільною та більш перевіреною мовою, ніж інші аналоги.

Основними характеристиками Solidity є:

- Підтримка об'єктно-орієнтованого програмування: використання класів, об'єктів, наслідувань та інтерфейсів.
- Статична типізація, що дозволяє виявляти помилки у програмі під час її компіляції.

- **Безпека:** Solidity розроблена з максимальним урахуванням безпеки застосунків та надає інструментарій для захисту від різних типів атак на розроблені застосунки.
- **Легкість вивчення:** оскільки функціональність смарт-контрактів є обмеженою на відміну від інших різних централізованих застосунків, ця мова програмування має менший набір правил та простіший синтаксис, тому є доволі легкою для вивчення, особливо для розробників, що мають досвід у об'єктно-орієнтованому програмуванні.

Однією з головних переваг використання Solidity є її широка спільнота учасників, кожен з яких тим чи іншим способом вносить вклад в створення фреймворків для розроблення застосунків, документації, курсів вивчення мови тощо. Крім цього, існують різні інструменти для покращення якості та пришвидшення створення застосунків, їх тестування та запуску. Гнучкість та широкий набір функцій для розроблення застосунків роблять цю мову програмування найкращим вибором серед інших аналогів.

Серед недоліків можна виділити недостатньо високий рівень безпеки застосунків, оскільки розробники є більш схильні до помилок під час використання через складність мови та процесів, що відбуваються у блокчейні. Також ускладнюється аудит систем, оскільки архітектура додатків з використанням різних патернів та глибокого наслідування доволі часто стає складною та місцями заплутаною.

Vyper

Vyper [13] – також статично типізована мова програмування для розроблення децентралізованих застосунків на основі віртуальної машини Ethereum.

На відміну від Solidity, Vyper є більш новою та сучасною мовою, що має ряд відмінностей. По-перше, вона не є об'єктно-орієнтованою мовою

програмування, що може мати свої переваги чи недоліки у певних випадках. По-друге, вона має простіший синтаксис, що нагадує Python. Це значно розширює коло розробників, яким буде легше освоїти та ефективніше використовувати новий засіб розроблення смарт-контрактів. Простіший синтаксис також забезпечує надійніший рівень аудитування системи, що є дуже важливим процесом розроблення децентралізованих застосунків. Окрім цього, мова програмування Vyper була розроблена з акцентом на підвищення безпеки застосунків.

Головним недоліком цієї мови програмування є те, що вона є порівняно новою та досі перебуває в розробленні. Для впевненості у надійності, стабільності та безпеці застосунку найкращим вибором буде обрати старіші мови програмування, вектор атак на застосунки яких є давно відомим і детально вивченим. До того ж, екосистема інструментів та фреймворків для розроблення застосунків є не настільки зрілою як у інших аналогів, відповідно, створення додатків на її основі може бути більш ускладненим.

2.2.2. Технології розроблення централізованої частини

Задачами централізованої частини додатку є централізована обробка користувацьких запитів, бізнес логіки та даних системи. Важливими вимогами до технологій розроблення є зручність використання, швидкість та продуктивність застосунку. Відповідно до вимог системи випуску цифрових сертифікатів, додаток повинен мати змогу ефективно обробляти та агрегувати великі масиви даних, підтримуючи паралельне виконання задач. Враховуючи архітектурні особливості системи, важливим є вибір мови програмування, що найкраще підходить для інтеграції з децентралізованою частиною системи. Серед великого вибору засобів розроблення серверної частини застосунку, оберемо та розглянемо три найкращих технології для написання серверного програмного коду: Python, Go та Rust.

Python

Python [14] – динамічно типізована мова програмування високого рівня. Вона відома своєю простотою, читабельністю та універсальністю, через що й користується своєю великою популярністю. Простий, зрозумілий та лаконічний синтаксис робить її найкращою мовою програмування як для початківців, так і досвідчених розробників. Цей засіб розроблення використовується для широкого кола задач, включаючи веброботку, машинне навчання, наукові дослідження тощо.

Python має велику кількість бібліотек та фреймворків, через що розроблення застосунків стає зручнішим та продуктивнішим. Не виключенням є й інтеграція зі смарт-контрактами EVM-сумісних блокчейнів. Python стає все більш популярним вибором для розробників, які хочуть створювати децентралізовані програми. Існує багато бібліотек, які полегшують інтеграцію зі смарт-контрактами, наприклад Web3.py, py-evm, eth-utils тощо.

Проте наявними недоліками мови програмування Python є:

- Динамічна типізація: динамічна типізація Python може призвести до помилок під час виконання, якщо тип змінної не визначений чітко.
- Швидкість виконання програми: Python є повільнішим за інші мови програмування, що може значно відобразитись на швидкодії застосунку. Тому у застосунках, де важлива продуктивність та ефективність системи, розробники в останню чергу обирають Python як основну мову програмування серверної частини.
- Обмеження паралелізму: глобальне блокування інтерпретатора Python значно обмежує паралелізм системи – лише один потік може знаходитися у стані виконання в певний момент часу. Вплив цієї системи не є помітний при виконанні однопотокових програм, на відміну від багатопотокових, швидкість та продуктивність яких значно падає [15].

Go

Go [16] – компільована, строго типізована мова програмування від компанії Google, що зазвичай використовується для розроблення швидких та надійних програм. Як і Python, Go має доволі простий синтаксис, що робить її доволі легкою для вивчення та розуміння. Її широко використовують для створення вебсерверів, мікросервісів, інструментів розробника та інших програм, оскільки вона має вбудовану підтримку для мережових операцій та розроблення серверних програм.

Однією з переваг Go є ефективне використання ресурсів, що робить її популярним вибором для високонавантажених систем та розподілених застосунків. Крім того, Go має вбудовану підтримку паралельного програмування для створення ефективних та швидкодіючих програм, які використовують багатопотоковість. Легковагові потоки виконання, що зветься горутинами, дозволяють розробникам писати паралельні програми без складного керування синхронізацією.

Також для мови програмування Go існують великі та популярні бібліотеки для роботи зі смарт-контрактами та блокчейн-платформами різних типів та протоколів, серед яких навіть офіційна бібліотека Ethereum. Зростаюча спільнота розробників, що працюють з блокчейном, робить цю мову чудовим вибором для початківців та досвідчених користувачів, які хочуть створювати децентралізовані програми.

Серед наявних недоліків Go можна виділити обмежену функціональність ООП. Хоч Go і підтримує багато сучасних парадигм, проте поступається потужним набором інструментів іншим мовам програмування. Через це вона не підходить для масштабних проєктів з чіткою ієрархією класів. Попри це, Go залишається чудовим вибором для побудови мікросервісної архітектури. Окрім обмеженості ООП, ця мова є відносно молодого, тому відчувається недолік тієї кількості сторонніх бібліотек для розроблення застосунків та учасників спільноти, що притаманні тому ж Python.

Rust

Rust [17] – багатопарадигмова мова програмування загального призначення. Rust має строгу типізацію як і Go, поєднує читабельність Python та контроль пам'яті мов низького рівня, що допомагає у розробленні продуктивних і паралельних застосунків та безпечній роботі з пам'яттю. Оскільки мова компільована, швидкодія застосунків в рази перевищує інтерпретовані мови, до яких належить Python.

Особливістю Rust є так звана система володіння даними, що називається контролером позичань. Він захищає пам'ять застосунку від стану гонитви даних, їх суперечності між частинами програми, зміни ітераційного об'єкту тощо. Крім переваг, така властивість ускладнює розроблення програми з використанням деяких стандартних практик та патернів в програмуванні, що може бути незручно для початківців.

Екосистема бібліотек та фреймворків Rust знаходиться в активній фазі розвитку. Вона постійно розширюється, з'являються нові інструменти та рішення для різних задач розроблення програм. Це включає в себе бібліотеки для роботи з мережами, парсингу даних, роботи з базами даних, розроблення вебдодатків та багато іншого. Не винятком є і бібліотеки роботи з блокчейном та смарт-контрактами. Попри їх обмежену кількість, Rust може бути привабливим вибором для розроблення децентралізованих додатків. Проте з ростом популярності Rust і зацікавленості учасників спільноти в розвитку інфраструктури блокчейну, можна очікувати подальшого зростання кількості бібліотек та інструментів для роботи з цією технологією.

Окрім недоліків кривої навчання, що спричинена деякою складністю вивчення унікальних властивостей мови та малої підтримки бібліотек, можна також виділити доволі затяжну компіляцію. У порівнянні з компільованою мовою Go, збірка проєкту може тривати у декілька разів довше.

2.2.3. Результати аналізу технологій розроблення серверної частини

У ході аналізу технологій розроблення серверної частини було розглянуто різноманітні рішення для реалізації як централізованої, так і децентралізованої частини серверної системи. Встановлені вимоги до програмного застосунку допоможуть порівняти характеристики розглянутих засобів та обрати найкращі інструменти для швидкого та ефективного розроблення системи.

Для розроблення децентралізованої частини серверної системи було обрано мову програмування Solidity. Не зважаючи на простоту синтаксису та легкість розроблення смарт-контрактів, що надає Vyper, Solidity підтримує більше функціональних можливостей та ширший набір функцій, що є значною перевагою. Стабільність та зрілість цієї мови програмування підвищує надійність розроблених застосунків, а велика спільнота учасників готова допомогти вирішити всі проблеми, що можливо з'являться під час розроблення додатків. Також перевагою при виборі саме цієї мови стала велика кількість різноманітних фреймворків для тестування та запуску смарт-контрактів, що є дуже важливим аспектом безпеки та надійності додатку. Ці фреймворки надають зручний інтерфейс та широкий інструментарій для написання та покриття програмного коду юніт-тестами.

Щодо технологій розроблення централізованої частини серверу, було обрано мову програмування Go. По-перше, це популярна мова з великою кількістю переваг у швидкодії та продуктивності для розроблення мікросервісів та вебзастосунків. На відміну від Python, Go підтримує легку та ефективну модель управління потоками та процесами системи. Завдяки легким та простим конструкціям мови, а також швидкій компіляції програмного коду, на противагу Rust, розроблення застосунків є швидшим та простішим. По-друге, офіційна бібліотека Ethereum для Go надає більш ширші можливості для реалізації різноманітних архітектурних рішень

взаємодії з блокчейном та смарт-контрактами та має більш стабільні версії, ніж у розглянутих мовах програмування.

2.3. Вибір технологій для розроблення клієнтської частини

Клієнтська частина програмного застосунку відповідає за представлення даних та взаємодію з користувачем. Вона повинна бути зручною, інтуїтивно зрозумілою та естетично привабливою для легкої та ефективної взаємодії з застосунком. До того ж, обрана технологія повинна спрощувати та пришвидшувати процес створення застосунку для розробника. Для клієнтської частини вебзастосунку випуску цифрових сертифікатів розглянемо найпопулярніші рішення, що пропонують широку функціональність та різноманітні інструменти розроблення, а саме: React, Vue.js та AngularJS.

2.3.1. *React*

React [18] – JavaScript-бібліотека, що призначена для створення різноманітних інтерактивних інтерфейсів користувача. Вона дозволяє розробляти вебзастосунки на основі компонентного підходу, що робить елементи програми доступними до повторного використання, а сам код – модульним та легким для розуміння. Підтримка віртуального DOM допомагає ефективно оновлювати та рендерити лише ті елементи інтерфейсу, які дійсно були змінені, без необхідності оновлювати її повністю. Використання JSX, що дозволяє вставляти HTML-код у JavaScript, дозволяє пришвидшити створення компонент інтерфейсу та сприяє кращій читабельності програмного коду. Також у React наявний односторонній низхідний потік даних, що допомагає запобігати розповсюдженним помилкам під час проєктування застосунків.

Серед недоліків цього фреймворку є акцентування на користувацькому інтерфейсі. Попри те, що React має великий набір для створення ергономічного інтерфейсу користувача, для повноцінного

застосунку з використанням маршрутизації необхідно інтегрувати додаткові залежності та бібліотеки. На відміну від інших наявних бібліотек та фреймворків, React не спроектований, як універсальне рішення для розроблення вебзастосунків. Окрім цього, оскільки React доволі молода технологія, то документація по цьому інструменту є не завжди вичерпною. Це, як і ускладнений синтаксис з використанням JSX та інших особливостей фреймворку, робить React більш складним для вивчення серед початківців.

2.3.2. *Vue.js*

Vue.js – доступний, ефективний та універсальний JavaScript фреймворк для створення користувацьких вебінтерфейсів [19]. Завдяки своїй особливій реактивній системі керування станами є доволі гнучким та простим. Використання спеціальних директив, що розширюють можливості HTML дозволяє додавати динамічну поведінку та функціональність до елементів DOM. Це також дозволяє підтримувати програмний код у лаконічному та читабельному вигляді.

Перевагою Vue також є те, що фреймворк не фокусується лише на користувацькому інтерфейсі, а надає комплексний набір інструментів для розроблення повноцінних вебзастосунків. Простота у вивченні та легкість у використанні роблять цей фреймворк хорошим вибором як для новачків, так і для досвідчених розробників. Також за часту програми, що розробляються на Vue, містять менше коду, ніж при використанні інших фреймворків.

Хоч спільнота, набір бібліотек та інструментів розроблення Vue активно розширюються, конкурентні фреймворки для розроблення вебзастосунків мають більш зрілі екосистеми. Недоліком бібліотеки, незважаючи на компонентний підхід до архітектури, є недостатня гнучкість при роботі та взаємодією між компонентами. У такому випадку

використання аналогів фреймворку для розроблення більших та складніших додатків може бути більш доцільним.

2.3.3. *AngularJS*

AngularJS [20] – фреймворк для розроблення односторінкових вебзастосунків, що написаний на JavaScript. Подібно до React, AngularJS використовує компонентний підхід, що дозволяє модулізувати та повторно використовувати розроблений програмний код. Така компонентність разом з жорсткою структурою фреймворку робить його хорошим вибором для великих проєктів з багатьма розробниками. Як і Vue, наявність директив для створення HTML-тегів та атрибутів дозволяє розширювати динамічну поведінку DOM-елементів. Двостороннє зв'язування даних забезпечує синхронізацію даних моделі з даними представлення. Доволі важливою властивістю AngularJS є можливість його інтеграції та взаємодії з іншими інструментами розроблення вебзастосунків.

Недоліками фреймворку є складність його використання для відносно невеликих застосунків. Структура AngularJS та надмірність його набору функцій може ускладнити швидке створення прототипів та реалізацію простих інтерфейсів. До того ж, фреймворк потребує більше часу та зусиль для вивчення та освоєння інструментарію новачками. Також, через пряму взаємодію з DOM, швидкість та ефективність застосунків, що розроблені на базі AngularJS, знижується та поступається аналогічним рішенням з використанням інших фреймворків.

2.3.4. *Результати аналізу технологій розроблення клієнтської частини*

В ході аналізу технологій розроблення клієнтської частини вебзастосунку було розглянуто найпопулярніші рішення для створення користувацьких вебінтерфейсів, а саме їх переваги та недоліки. Для розроблення вебзастосунку для випуску цифрових сертифікатів було

прийнято рішення обрати фреймворк Vue.js. Попри його меншу популярність серед React та AngularJS, інструмент вирізняється своєю простотою вивчення та швидкістю розроблення застосунку. Vue.js надає зручні інструменти для ефективного розроблення як великих, так і малих проєктів, на відміну від Angular, з можливістю їх подальшого розширення та масштабування. Відсутність фокусу лише на компонентному підході, що властиве React, дозволяє мінімізувати кількість залежностей та використання сторонніх бібліотек у процесі розроблення застосунку.

2.4. Вибір СКБД

База даних є необхідною частиною будь-якого програмного застосунку, що забезпечує зберігання, керування та доступ до даних. СКБД розроблюваного вебзастосунку повинна відповідати за надійне та безпечне зберігання інформації про випущені сертифікати та пов'язані метадані. При виборі потрібної бази даних слід враховувати безпеку, масштабованість, вартість, спільноту тощо.

Окрему увагу слід також звернути на тип даних, що зберігатимуться в базі. Вибір необхідної бази даних прямозалежний від вимог та особливостей розроблюваного застосунку. Оскільки у системі можуть чітко прослідковуватись певні зв'язки (користувач має сертифікати, сертифікат має метадані, видавець випустив сертифікати), то доцільно надати перевагу реляційним базам, у котрих дані зберігаються у вигляді взаємопов'язаних таблиць.

Серед найпопулярніших реляційних СКБД можна виділити PostgreSQL та MySQL. Для вибору підходящої бази даних проведемо аналіз переваг та недоліків кожної з них.

2.4.1. PostgreSQL

PostgreSQL [21] – об'єктно-реляційна база даних з відкритим вихідним кодом, що використовує мову запитів SQL. Це одна з

найпопулярніших баз даних, що славиться надійністю, масштабованістю, надає широкі функціональні можливості та знаходить своє використання в практично будь-яких типах і розмірах застосунків.

Основними перевагами цієї СКБД є:

- Підтримка різноманітних типів даних, включаючи користувацько-визначені типи, масиви, JSON дані, геометричні примітиви тощо.
- Широкий спектр індексів, доступних для оптимізації ефективності запитів та пошуку даних.
- Повна підтримка ACID з можливістю рівня ізоляції транзакції, вибору точок збережень тощо.
- Підтримка збережених процедур на різних декларативних та недекларативних мовах програмування.
- Широка платформа для вибору наявних та розроблення власних розширень, що дозволяють не тільки ефективніше маніпулювати даними, а й розширити існуючі чи додати нові можливості бази даних в цілому.
- Підтримка різноманітних методів і можливостей реплікації даних.
- Велика спільнота учасників та обширна документація, що покриває усі аспекти та особливості застосунку.
- Масштабованість, ефективність та надійність системи.

Серед наявних недоліків PostgreSQL можна виділити складність управління та налаштування такої великої СКБД, що робить її більш підходящою для більш досвідчених розробників. Через свою широку функціональність, база даних може споживати більше ресурсів пам'яті та процесора, ніж наявні аналоги.

2.4.2. MySQL

MySQL [22] – реляційна СКБД з відкритим вихідним кодом. Це теж одна з найпоширеніших баз даних для малих та середніх проєктів.

Серед переваг цієї СКБД можна виділити:

- Високу продуктивність та здатність обробляти великі масиви даних.
- Відповідність стандартам ACID.
- Підтримку збережених процедур та розширень.
- Пізні моделі реплікації даних.
- Швидку систему індексування даних.
- Горизонтальну масштабованість шляхом додавання нових вузлів до кластеру бази.

Недоліками MySQL є недостатня підтримка різноманітних типів даних, проблеми блокування даних при паралельних операціях запису даних, мала кількість підтримуваних типів індексування даних. Це робить систему більш придатною для менших проєктів, у той час як для більших кращим вибором буде обрати іншу СКБД.

2.4.3. Результати аналізу СКБД

У результаті аналізу обраних СКБД було детально розглянуто їх переваги та недоліки. Хоча обидві бази даних мають широкий та потужний інструментарій для управління та зберігання даних, MySQL поступається у деяких властивостях PostgreSQL. Незважаючи на відносно просте та зручне управління кластерами в MySQL, PostgreSQL надає більш ширші можливості для оброблення різних типів даних, використання збережених процедур та розширень, індексації даних. Через велику спільноту учасників та її активне зростання PostgreSQL має не тільки вичерпну документацію, а й велику кількість бібліотек для взаємодії з СКБД для багатьох популярних мов програмування.

2.5. Висновки до розділу 2

В ході аналізу засобів для реалізації вебзастосунку було виділено та опрацьовано різноманітні найпопулярніші інструменти розроблення

вебзастосунків та його частин, зокрема різноманітні блокчейн-платформи, засоби розроблення клієнтських та серверних частин застосунків, СКБД. Було розглянуто переваги та недоліки відповідних інструментів, платформ, систем, відповідність вимогам та особливостям розроблюваного застосунку.

Серед розглянутих технологій та засобів розроблення вебзастосунку для випуску цифрових сертифікатів було обрано мову програмування Solidity та блокчейн-мережу Polygon для розроблення і запуску смарт-контрактів. Для розроблення централізованої серверної частини застосунку було обрано мову програмування Go. Клієнтська частина вебзастосунку розроблятиметься засобами фреймвору Vue.js. Обрана база даних системи – СКБД PostgreSQL.

3. РОЗРОБЛЕННЯ ВЕБЗАСТОСУНКУ ДЛЯ ВИПУСКУ ЦИФРОВИХ СЕРТИФІКАТІВ

3.1. Загальний опис архітектури програмного забезпечення

Архітектура програмного застосунку визначає основні структурні елементи та принципи їх організації та взаємодії, що спрямовані на досягнення бажаних функціональних можливостей системи. При виборі архітектури компонентів та взаємозв'язків системи важливо звернути увагу на її модульність, масштабованість, ефективність, гнучкість тощо. Врахування цих аспектів на етапі проектування допомагає розробити стійкий, функціональний та зручний у використанні програмний застосунок.

Вебзастосунок для випуску цифрових сертифікатів розроблено на базі клієнт-серверної архітектури, що є одним з найпоширеніших архітектурних шаблонів для створення програмних застосунків. Основними компонентами такої архітектури є клієнтська та серверна частини застосунку, що обмінюються даними та тісно взаємодіють між собою.

Клієнтська частина представлена вебінтерфейсом, який користувач використовує для взаємодії з системою через браузер. Цей інтерфейс дозволяє користувачеві переглядати, випускати та верифіковувати цифрові сертифікати. Серверна частина відповідає за оброблення запитів від клієнтів, виконання необхідних операцій (наприклад, генерація чи верифікація цифрових сертифікатів), зберігання та управління даними. Вона також включає систему автентифікації та авторизації користувачів для забезпечення захисту даних.

Взаємодія між клієнтською та серверною частинами відбувається через протокол передачі даних HTTP. Для виконання операції чи отримання даних клієнт формує відповідний HTTP-запит, керуючись правилами специфікації запиту даних JSON API. Серверна частина обробляє ці

HTTP-запити, виконує необхідні операції (наприклад, генерацію або верифікацію цифрових сертифікатів) та формує HTTP-відповіді аналогічно у форматі JSON. Ці відповіді містять необхідну інформацію або результати операцій і відправляються назад до клієнта. Такий підхід визначає однорідний формат обміну даних та стандартизовану взаємодію, що полегшує та пришвидшує розроблення застосунку.

Важливою рисою цієї архітектури є здатність до розділення функціональних можливостей між клієнтом і сервером, що полегшує розроблення, тестування та масштабування застосунку. Крім того, такий підхід дозволяє покращити безпеку, оскільки сервер може контролювати доступ до ресурсів та забезпечувати безпеку даних.

Загальна схема архітектури розробленого програмного застосунку зображена на рис. 3.1.

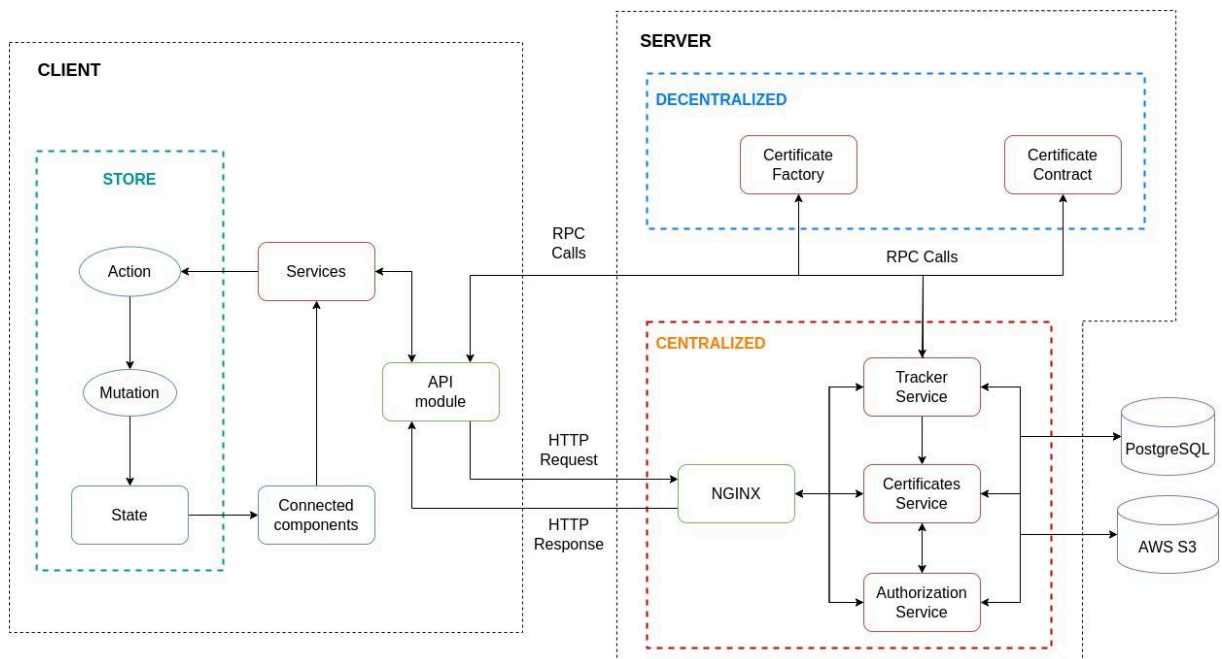


Рис. 3.1. Архітектура розробленого вебзастосунку

З урахуванням використання технології блокчейн, серверну частину вебзастосунку, можна розділити на дві частини – централізовану та децентралізовану.

Централізована компонента серверної частини відповідає за забезпечення основної логіки та функціональних можливостей вебзастосунку, зокрема оброблення користувачьких запитів, автентифікація та авторизація користувачів, зберігання основної бази даних, а також виконання важливих операцій, які вимагають централізованого керування та контролю, таких як управління доступом до даних або генерація цифрових сертифікатів. Вона спроектована з використанням мікросервісного підходу до розроблення застосунків. Такий архітектурний стиль полягає у розробленні кількох невеликих сервісів, що реалізують конкретні набори функцій та розробляються, розгортаються, масштабуються окремо від інших. Комунікація сервісів в розробленому програмному забезпеченні відбувається відповідно до специфікації JSON API HTTP-протоколу.

Серед інших переваг мікросервісної архітектури можна виділити:

- Гнучкість: мікросервіси розробляються як самостійні модулі, що робить їх простими у зміні та оновленні.
- Стійкість: відмова одного мікросервісу не впливає на всю систему. Це робить мікросервісні програми більш стійкими до помилок та збоїв.
- Легкість впровадження: невеликий розмір та самостійність мікросервісів роблять їх простішими у впровадженні та тестуванні, що може прискорити час виходу на ринок.
- Зниження складності застосунку: розбиття монолітної програми на менші, керовані мікросервіси може значно спростити кодову базу, що робить її більш зрозумілою та простою для обслуговування.

При взаємодії з мікросервісами програмного застосунку, клієнтська частина не звертається напряму до кожного з них. Вона використовує спеціальний API-шлюз, що представлено мікросервісом Nginx – незалежним HTTP проксі-сервером [23]. Такий підхід сприяє покращенню продуктивності та масштабованості ПЗ, забезпечуючи гнучкість

управління ресурсами та ефективне використання апаратного забезпечення. Цей сервіс допомагає налаштувати єдину точку входу в систему та перенаправлення певних запитів до відповідних мікросервісів, що значно полегшує інтеграцію з серверною частиною. При горизонтальному масштабуванні певного сервісу Nginx може також додатково виступати балансувальником навантаження. Це забезпечує високу доступність та надійність системи, дозволяючи розподіляти трафік між різними серверами для забезпечення оптимальної продуктивності та швидкодії. Сервер Nginx гнучко конфігурується та може обробляти велику кількість одночасних запитів, а також підтримувати різні протоколи передачі даних.

Децентралізована компонента серверної частини відповідає за збереження верифікаційних даних про випущені сертифікати та слугує джерелом їх правдивості та цілісності. Вона забезпечує надійну систему зберігання інформації, яка не піддається зміні або маніпуляції з боку окремих учасників. Представлена набором смарт-контрактів, що функціонують на EVM-сумісній блокчейн-мережі. Взаємодія як централізованої частини сервера, так і клієнта зі смарт-контрактами здійснюється за протоколом виклику віддалених процедур JSON RPC. Варто зазначити, що ця взаємодія є односторонньою – середовище виконання смарт-контрактів є ізольованим від навколишньої екосистеми.

Для надійного та масштабованого зберігання даних для вебзастосунку використовується база даних PostgreSQL. В залежності від обмежень системи та вимог проєкту, база даних для мікросервісів може бути як спільною, так і окремою для кожного. Тобто, кожен із мікросервісів, за потреби, розгортається з окремою базою даних, або ж мікросервіси взаємодіють з одним загальним екземпляром бази, використовуючи при цьому різні таблиці, функції, тригери тощо. Розроблене програмне забезпечення підтримує обидві опції розгортання кластеру бази даних, проте рекомендується використання окремих баз під

кожен окремий мікросервіс, оскільки вони стають більш незалежними один від одного, а також мінімізується ризик втрати одразу усіх сервісних даних.

Додатковим місцем для зберігання конкретних даних, а саме файлів випущених цифрових сертифікатів, було обрано сховище Amazon S3 [24]. Це об'єктний файловий хостинг, що дозволяє надійно зберігати та швидко отримувати збережені дані. Різноманітні конфігурації доступу та захисту сховища забезпечують безпеку, конфіденційність та контроль доступу до даних, що зберігаються в ньому. Крім того, Amazon S3 має велику масштабованість та доступність, що дозволяє ефективно зберігати та отримувати дані в будь-який момент часу. Завдяки різним рівням доступності та реплікації даних, Amazon S3 забезпечує надійність та відновлюваність даних у випадку виникнення проблем або збоїв.

3.2. Структура серверної частини

3.2.1. Децентралізована компонента

Як було зазначено у підрозділі 3.1, децентралізована компонента серверної частини вебзастосунку для випуску цифрових сертифікатів представлена у вигляді смарт-контрактів, розроблених з використанням високорівневої мови програмування Solidity.

Провівши аналогію з класичним об'єктно-орієнтованим програмуванням, їх можна сприймати за окремі класи, що реалізують бізнес-логіку програми. Об'єкти таких класів запускаються у децентралізованій розподіленій мережі блокчейну, тому стан і дані виконуваної програми ніхто не може підмінити.

Оскільки середовище їх виконання є ізольованим, смарт-контракти не можуть взаємодіяти з будь-якими зовнішніми API чи сервісами. Проте EVM побудована особливим чином та дозволяє взаємодіяти навколишній екосистемі застосунків зі смарт-контрактами шляхом використання протоколу виклику віддалених процедур JSON RPC.

В рамках вебзастосунку розроблено два класи контрактів, що є взаємозалежними між собою:

- Контракт фабрики курсів – дозволяє створювати нові курси шляхом розгортання нових контрактів та зберігати й агрегувати їх адреси. До створення нового курсу допускаються абсолютно усі користувачі, достатньо лише ініціалізувати смарт-контракт початковими даними, такими, як назва та опис курсу, і після його розгортання користувач має повний контроль над своїм об'єктом курсу.
- Контракт курсу – містить функції випуску, відкликання сертифікатів, додавання/видалення адміністраторів, а також агрегує дані про випущені та відкликані сертифікати. Завдяки реалізованим модифікаторам доступу, контракт може контролювати кому дозволено виконувати ті чи інші функції. Використання композиції структури даних словник дозволило реалізувати швидку та зручну перевірку різних даних про сертифікат, наприклад перегляд даних про випущений сертифікат з конкретною контрольною сумою певного користувача.

Окрім звичайного виклику функцій смарт-контрактів для взаємодії з децентралізованим застосунком, середовище виконання підтримує події, що спрацьовують при зміні стану контрактів або при їх виклику. Це дозволяє застосункам реагувати на зміни в блокчейні або в самому контракті без необхідності їх постійного опитування. Виклик події дозволяє передати корисні дані, що згодом можуть бути використані системою, що на них реагує. Таким чином, виклик подій та контроль зміни стану смарт-контрактів забезпечує ефективний спосіб оповіщення навколишньої екосистеми систем, що інтегруються з програмним застосунком.

Розроблення смарт-контрактів системи відбувалося на основі фреймворку Hardhat [25] – набору компонентів для розроблення,

компіляції, тестування та розгортання смарт-контрактів. Hardhat надає зручний CLI для виконання різноманітних команд, що полегшують процес створення ПЗ. Додатково, фреймворк підтримує генерацію модулів взаємодії зі смарт-контрактами, що у колі розробників децентралізованих систем називаються біндингами, на основі їх програмного коду для мови програмування Go. Це полегшує та пришвидшує інтеграцію системи з її децентралізованими компонентами. Приклад використання Hardhat для компіляції розроблених контрактів та генерації біндингів взаємодії зі смарт-контрактами зображено на рис. 3.2.

```
slbmax@slbmax-asus-v15:~/go/src/github.com/slbmax/diploma/contracts/certificates$
npm run generate-types

> hardhat-template@1.0.0 generate-types
> TYPECHAIN_FORCE=true npx hardhat typechain && npx hardhat gobind
```

Contract Name	Size (KiB)	Change (KiB)
StringSet	0.084	0.000
Paginator	0.084	0.000
EnumerableSet	0.084	0.000
CourseContract	4.183	0.000
CourseFactory	6.285	0.000

Nothing to compile

Contract Name	Size (KiB)	Change (KiB)
EnumerableSet	0.084	0.000
StringSet	0.084	0.000
Paginator	0.084	0.000
CourseContract	4.183	0.000
CourseFactory	6.285	0.000

Generated bindings for 7 contracts

Рис. 3.2. Генерація біндингів смарт-контрактів за допомогою Hardhat

3.2.2. Централізована компонента

Централізована компонента серверної частини вебзастосунку для випуску цифрових сертифікатів представлена у вигляді трьох наступних мікросервісів: авторизації, реагування на блокчейн-події та безпосередньо випуску сертифікатів. Кожен з них виконує свої функції відокремлено, що сприяє модульності та гнучкості архітектури системи. Розроблені мікросервіси комунікують та обмінюються даними між собою використовуючи HTTP API. Детальніше розглянемо призначення та функціональні можливості кожного з них.

Сервіс авторизації

Призначенням даного мікросервісу є автентифікація та авторизація користувачів і керування їх доступом до функціональних можливостей системи випуску цифрових сертифікатів.

Авторизація користувача відбувається з використанням JWT. Ця технологія дозволяє передавати дані про користувача у вигляді токенів, на основі яких здійснюється ідентифікація власника. JWT використовує цифрові підписи для безпечної авторизації та запобіганню підробки токенів. Розробники, відповідно до вимог та особливостей розроблюваного застосунку, можуть самостійно обрати та сформулювати корисні дані для автентифікації користувача в системі.

Розроблене API сервісу дозволяє виконувати перевірку введених користувацьких даних, генерацію JWT токенів для успішно авторизованих користувачів, їх заміну та оновлення. Також система має функцію перевірки даних JWT, яка, у разі коректного токена, повертає закодовану інформацію про користувача. На її основі можна розробляти проміжні методи в інших сервісах, що будуть контролювати доступ до конкретних ресурсів або функцій залежно від ролі користувача.

Характерною особливістю даного мікросервісу є те, що реалізований спосіб авторизації розроблений таким чином, що не використовує будь-які

приватні дані користувача – його паролі, номер телефону тощо. Оскільки блокчейн використовує криптографію з відкритим ключем (РКС), кожен користувач у системі може бути представлений своїм унікальним відкритим ключем у якості логіну. Для перевірки того, що користувач справді є володарем наданого відкритого ключа, необхідно впевнитись що користувач володіє відповідним приватним ключем з пари.

Якщо користувач кожен раз надсилатиме приватний ключ для перевірки, то це тягне за собою великі ризики його компрометації у разі передавання через мережу чи зберігання на сервері. В розробленому сервісі використовуються методи сучасної криптографії, що базуються на підписі певних повідомлень.

Користувач повинен підписати випадковий рядок, який генерується мікросервісом, за допомогою алгоритму ECDSA і передати підписане повідомлення на сервер. При цьому, користувач ніяким чином не розкриває свій приватний ключ та може використати різні програмні/апаратні утиліти та застосунки для цифрових підписів. Це збільшує надійність вебзастосунку з точки зору користувача, оскільки йому не потрібно довіряти тому, що його конфіденційні дані надійно зберігаються в системі та не будуть ніяким чином скомпрометовані.

Мікросервіс перевіряє отриманий підпис, використовуючи відкритий ключ, який користувач передає разом із підписом. Якщо підпис вірний і відповідає наданому відкритому ключу, то користувачу надається доступ до системи. Такий підхід забезпечує безпеку і конфіденційність, оскільки приватний ключ користувача залишається на його пристрої і не передається через мережу.

Схема відповідного алгоритму авторизації користувача на основі РКС представлена на рис. 3.3.

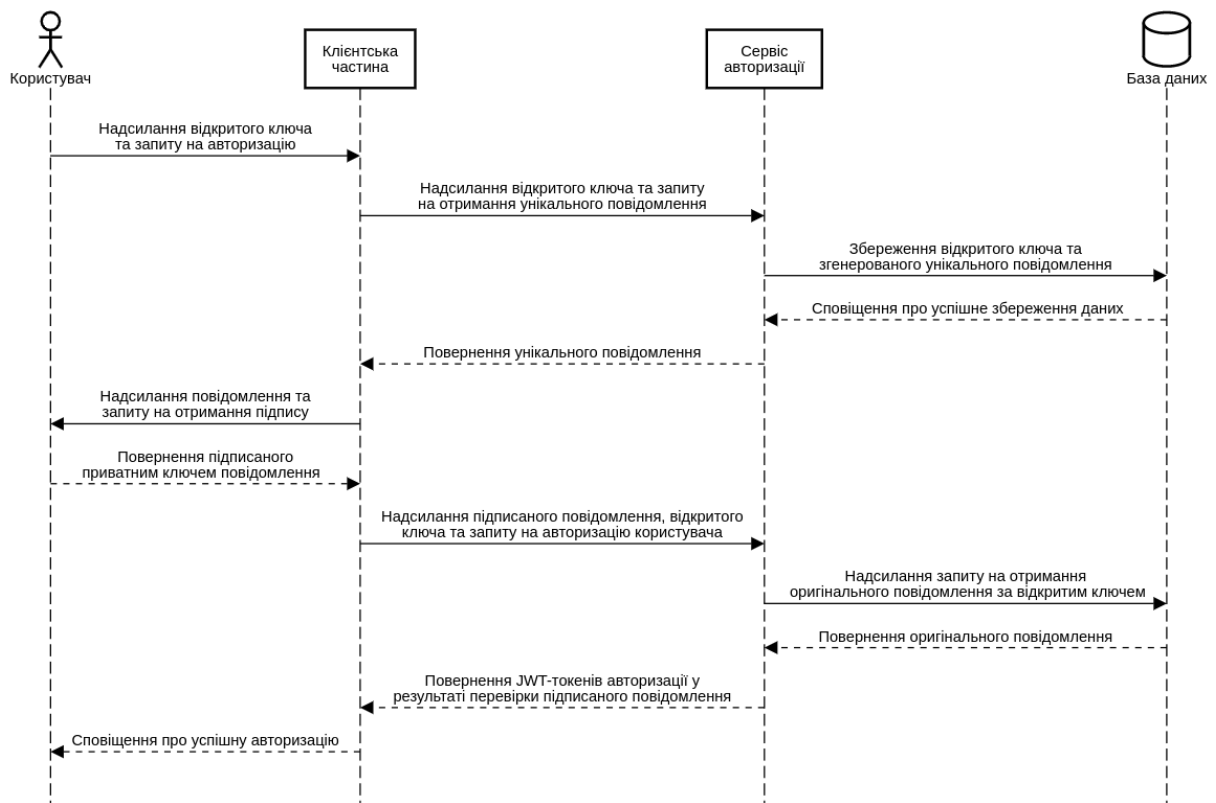


Рис. 3.3. Схема авторизації користувача на основі РКС

Сервіс реагування на блокчейн-події

Призначенням сервісу реагування на блокчейн-події є моніторинг та автоматизоване оброблення подій, що відбуваються на блокчейні у результаті взаємодії зі смарт-контрактами.

Кожна подія, наприклад, створення курсу чи випуску сертифікату, несе у собі корисну інформацію. Це може бути адреса нового смарт-контракту, адреса користувача, хеш-значення опублікованого сертифікату тощо. Задачею сервісу є відслідковування подій та декодування корисних даних, що знаходяться в їх тілі. Отримання інформації з блокчейну відбувається шляхом надсилання відповідного запиту-фільтру, в якому вказуються назви подій та адреси смарт-контрактів, що їх створюють. Приклад закодованої події смарт-контракту зображено на рис. 3.4.

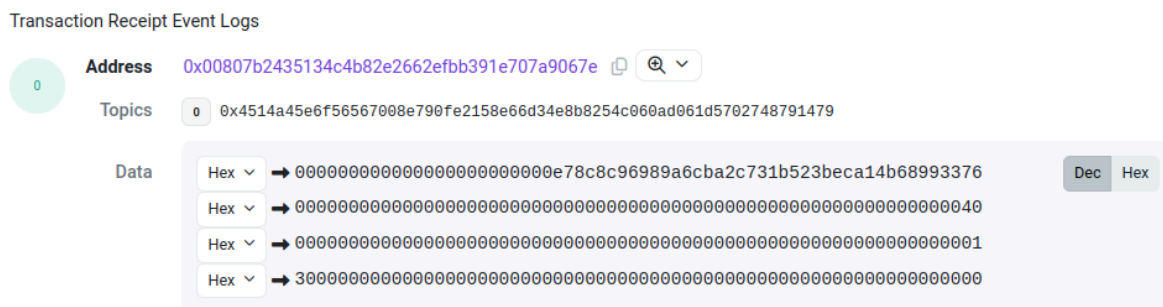


Рис. 3.4. Закодована подія смарт-контракту

Після того, як сервіс обробив опубліковану подію та отримав корисні дані про виконану функцію/транзакцію, його задачею є передача інформації іншим мікросервісам системи для подальшого оброблення даних або реагування. Додатково сервіс реагує на створення нових смарт-контрактів, щоб у подальшому відслідковувати їх події також.

Сервіс випуску сертифікатів

Головним призначенням даного мікросервісу є створення, збереження, облік, публікація та верифікація сертифікатів. Це є головний мікросервіс сервісної частини застосунку, оскільки він реалізує критично важливу функціональну можливість системи. Окрім цього, система надає можливості створення курсу, в рамках якого випускається сертифікат, управління списком авторизованих до випуску документів осіб, дизайну власних шаблонів сертифікатів тощо.

Для повноцінної роботи системи сервіс випуску сертифікатів приймає дані від сервісу реагування на блокчейн-події та оновлює актуальні дані у системі. Для валідації авторизованого доступу користувача мікросервіс використовує сервіс авторизації. Комунікація між сервісами здійснюються за допомогою HTTP API.

У мікросервісі реалізовано різноманітні алгоритми для швидкої та ефективної системи, а саме: паралельна генерація цифрових сертифікатів,

методи перевірки даних на основі цифрових підписів алгоритму ECDSA, потокове оброблення подій тощо.

Алгоритм випуску цифрових сертифікатів на основі технології блокчейну зображено на рис. 3.5.

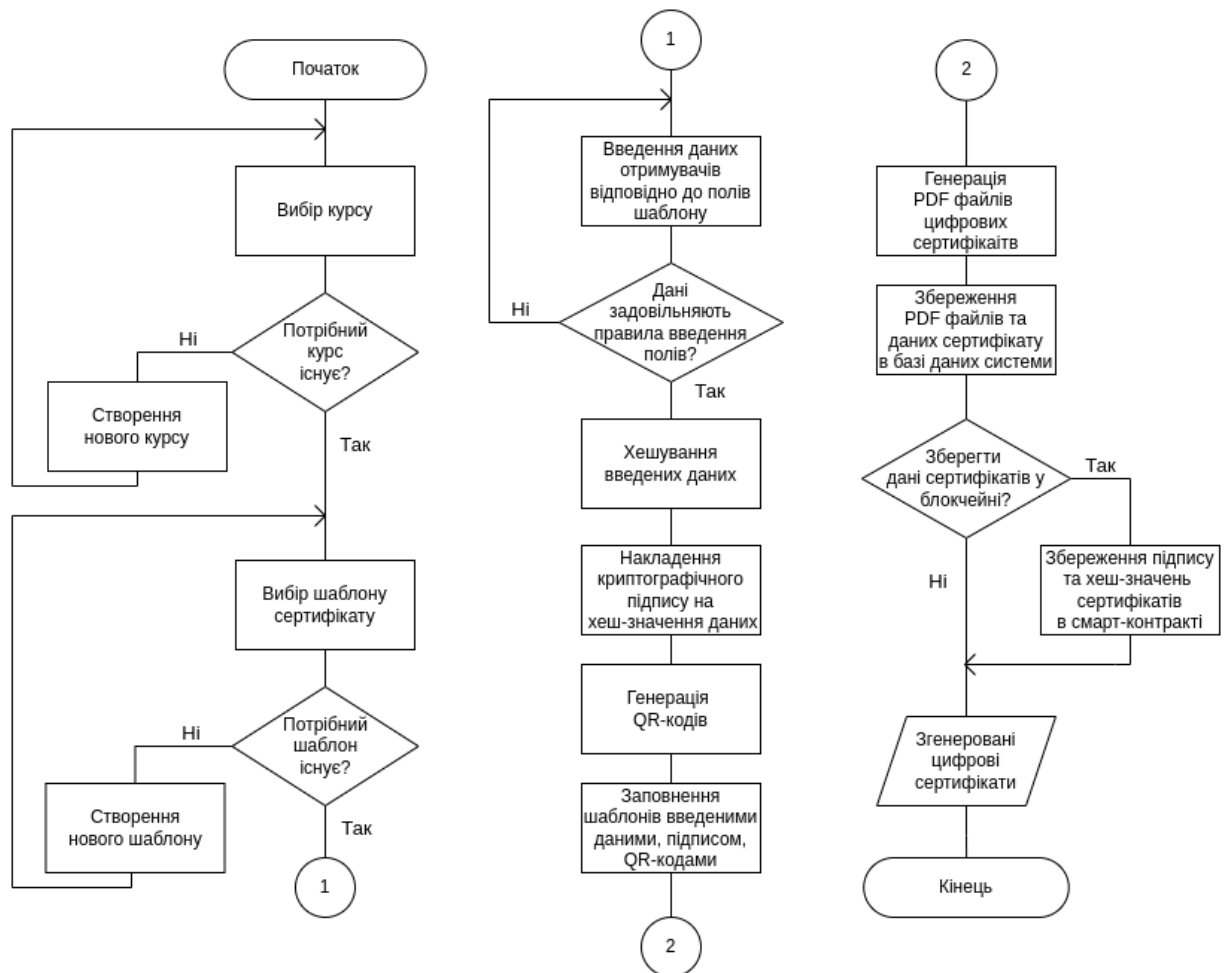


Рис. 3.5. Алгоритм випуску цифрових сертифікатів

Результатом випуску цифрового сертифікату є відповідний PDF-файл документу, дані про публікацію документу додатково публікуються в мережу блокчейн, якщо це було вказано особою, що випустила його. Додатково для забезпечення надійності та безпеки згенерованих PDF-файлів цифрових сертифікатів використовується сховище Amazon S3 без налаштувань публічного доступу. В такому разі мікросервіс повністю контролює, хто має право на завантаження певного файлу без можливості

обійти систему та заволодіти сертифікатом. Хоч і такі цифрові сертифікати, у більшості випадків, не повинні містити конфіденційної інформації, та користувач у праві самостійно обирати, з ким поділитись даним документом.

Верифікація сертифікату відбувається у відповідному обробнику HTTP-запиту маршруту верифікації на основі двох різних параметрів: ідентифікатора сертифікату та хеш-значення файлу цифрового документу.

Верифікація документа на основі його сертифіката відбувається при використанні QR-коду, який було попередньо проскановано. Він містить закодоване посилання на сторінку верифікації сертифіката разом з відповідним ідентифікатором запису. З його допомогою можна перевірити чи наявний такий документ у системі, інформацію про його дані та їх відповідність інформації зі смарт-контрактів, якщо запис про випуск сертифікату було збережено у блокчейні. Проте, такий підхід не дозволяє повністю впевнитись у тому, що на сам документ чи його паперову версію не було змінено, наприклад, накладено неправдиву інформацію поверх наявної чи підміну QR-коду та підпису особи, що його випустила.

В такому разі існує більш комплексний варіант верифікації, що базується на самому документі цифрового сертифікату. На вхід система приймає певне хеш-значення, яке обчислюється на основі масиву байтів електронного файлу. Цей спосіб дозволяє аналогічно ідентифікувати у системі та перевірити усі властивості цифрового документа, а також порівняти, що прикріплений файл оригінальний та відповідає тому, що було випущено засобами даного програмного застосунку.

3.3. Структура клієнтської частини

Реалізована клієнтська частина вебзастосунку для випуску цифрових сертифікатів виступає найвищим рівнем інтерфейсу взаємодії користувача з системою. Вона представлена графічним користувацьким інтерфейсом вебсторінок з використанням різних компонент та модулів, що включають

в себе авторизацію користувача, перегляд, випуск та верифікацію сертифікатів, взаємодію з API смарт-контрактів та серверної частини програмного застосунку. Надаючи простий та зручний інтерфейс роботи з додатком, клієнтська компонента покращує загальний досвід користувача та забезпечує швидке і ефективне використання програмного застосунку.

Детальніше розглянемо архітектуру клієнтської частини, що зображена на рис. 3.1:

- **Connected components** – набір сторінок і графічних компонентів, що слугують видимим інтерфейсом користувача.
- **Action** – ініційовані користувачем події чи його взаємодія з компонентами графічного інтерфейсу.
- **Services** – посередники між інтерфейсом користувача та серверною частиною програми, що реалізують виконання бізнес-логіки застосунку.
- **API module** – модуль інтеграції клієнтської частини з API серверної компоненти.
- **Mutation** – синхронні функції, що модифікують стан системи.
- **State** – об'єкт, що містить стан клієнтської частини застосунку, включаючи будь-які графічні дані, змінні тощо.

Керування станом системи було реалізовано за допомогою Pinia – бібліотеки управління станом у Vue, що дозволяє ділитися ним між різними компонентами/сторінками [26]. Вона надає сучасний підхід до керування та пропонує декларативний підхід до роботи зі станом, подіями та діями користувача. Додатковою перевагою Pinia є легка інтеграція з будь-яким іншим стеком бібліотек розроблення для Vue, що полегшує розроблення програмного застосунку.

Серед розроблених модулів виконання бізнес-логіки застосунку окремо можна виділити модуль для взаємодії з блокчейном та реалізації авторизації користувача на основі РКС засобами електронного гаманця MetaMask [27]. Це програмне забезпечення з відкритим вихідним кодом,

що агрегує приватний ключ користувача, реалізує основні криптографічні методи для генерації цифрових підписів та надає різні функціональні можливості для взаємодії з EVM-сумісними блокчейн-мережами. Варто зазначити, що приватний ключ ніколи не покидає межі пристрою користувача та не передається мережею – MetaMask реалізовано у вигляді розширення для браузерів, що зберігає секретні дані виключно локально та звертається до них лише у разі підтвердження дії користувача.

У вебзастосунку для випуску цифрових сертифікатів здійснено інтеграцію з MetaMask для виклику методів криптографічного підпису повідомлень та взаємодії зі смарт-контрактами з використанням приватного ключа користувача. Приклади відповідних методів підпису та взаємодії зображено на рис. 3.6, 3.7.

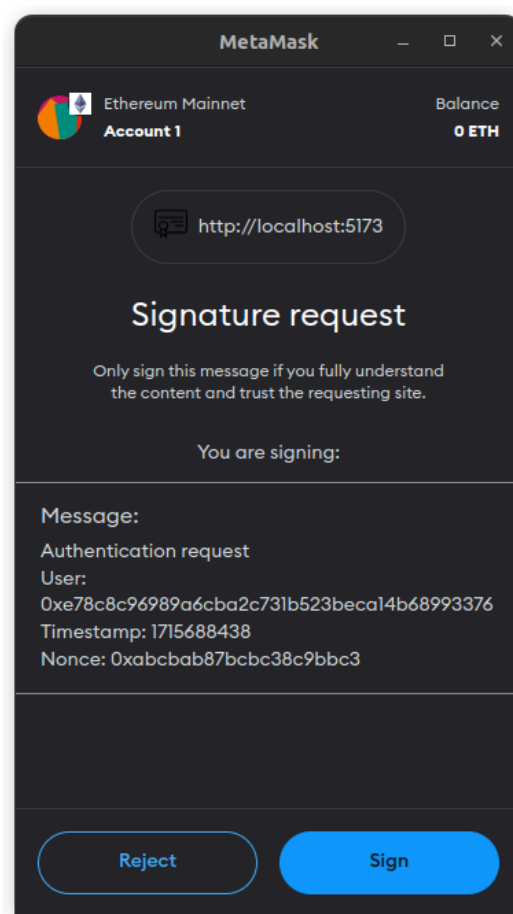


Рис. 3.6. Вікно MetaMask криптографічного підпису повідомлення

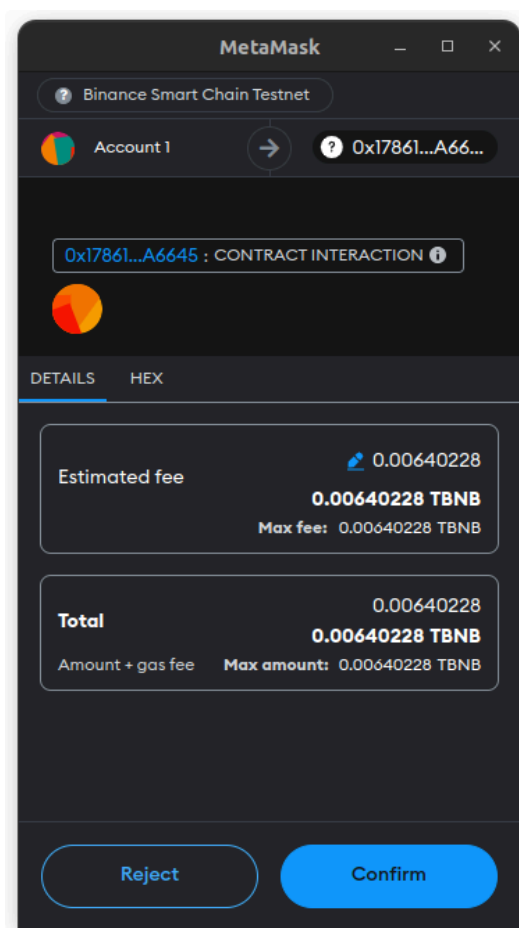


Рис. 3.7. Вікно MetaMask взаємодії зі смарт-контрактом

3.4. Структура бази даних

Оскільки серверна частина вебзастосунку для випуску цифрових сертифікатів складається з мікросервісів, кожен з них використовує власну базу даних. Це забезпечує незалежність компонент системи, тобто зміни чи збої в одній базі даних не впливають на інші. Такий підхід також дозволяє використовувати різні типи баз даних, які найкраще підходять для конкретних задач. У разі потреби чи недовліку системних ресурсів для розгортання інфраструктури компонентів застосунку мікросервіси можуть використовувати різні таблиці спільної бази даних.

Розглянемо структуру таблиць бази даних вебзастосунку для випуску цифрових сертифікатів, що зображена на рис. 3.8.

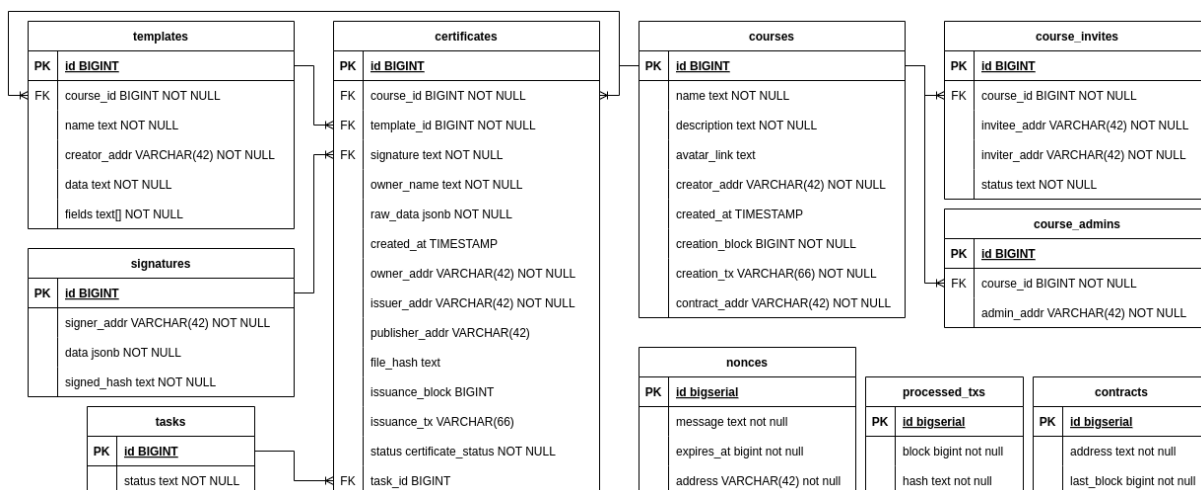


Рис. 3.8. ER-діаграма бази даних

Опишемо основні таблиці бази даних:

1. Курси (courses):

- id – унікальний ідентифікатор сутності курсу;
- name – назва курсу;
- description – опис курсу;
- avatar_link – посилання на аватар курсу;
- creator_addr – адреса користувача, що створив курс;
- created_at – дата створення курсу;
- creation_block – номер блоку створення курсу в блокчейн-мережі;
- creation_tx – хеш транзакції створення курсу в блокчейн-мережі;
- contract_addr – адреса смарт-контракту відповідного курсу.

2. Адміністратори курсів (course_admins):

- id – унікальний ідентифікатор сутності адміністратора;
- course_id – зовнішній ключ, ідентифікатор сутності курсу;
- admin_addr – адреса користувача – адміністратора курсу.

3. Запрошення до курсів (course_invites):

- id – унікальний ідентифікатор сутності запрошення;
- course_id – зовнішній ключ, ідентифікатор сутності курсу;

- `invitee_addr` – адреса користувача, що був запрошений стати адміністратором курсу;
- `inviter_addr` – адреса користувача, що запрошує стати адміністратором курсу;
- `status` – статус запрошення.

4. Шаблони (templates):

- `id` – унікальний ідентифікатор сутності шаблону;
- `course_id` – зовнішній ключ, ідентифікатор сутності курсу;
- `name` – назва шаблону
- `creator_addr` – адреса (відкритий ключ) користувача, що створив шаблон;
- `data` – дата створення шаблону;
- `fields` – список змінних полів шаблону для заповнення значеннями.

5. Підписи (signatures):

- `id` – унікальний ідентифікатор сутності підпису;
- `signer_addr` – адреса (відкритий ключ) користувача, що підписав повідомлення;
- `data` – дані, що підписуються користувачем;
- `signed_hash` – підпис хеш-значення даних, що підписуються користувачем.

6. Сертифікати (certificates):

- `id` – унікальний ідентифікатор сутності сертифікату;
- `course_id` – зовнішній ключ, ідентифікатор сутності курсу;
- `template_id` – зовнішній ключ, ідентифікатор сутності шаблону;
- `signature` – зовнішній ключ, ідентифікатор сутності підпису;
- `owner_name` – ім'я користувача, що отримав сертифікат;
- `raw_data` – корисні дані сертифікату в форматі JSON;
- `created_at` – дата створення сертифікату;

- owner_addr – адреса (відкритий ключ) користувача, якому було випущено сертифікат;
- issuer_addr – адреса (відкритий ключ) користувача, який випустив сертифікат;
- publisher_addr – адреса (відкритий ключ) користувача, який опублікував сертифікат;
- file_hash – хеш-значення згенерованого файлу сертифікату;
- issuance_block – номер блоку випуску сертифікату в мережі блокчейну;
- issuance_tx – хеш транзакції випуску сертифікату в мережі блокчейну;
- status – статус сертифікату;
- task_id – зовнішній ключ, ідентифікатор сутності задачі.

7. Задачі (tasks):

- id – унікальний ідентифікатор сутності задачі;
- status – статус виконуваної задачі.

8. Унікальні повідомлення (nonces):

- id – унікальний ідентифікатор сутності повідомлення;
- message – повідомлення для підпису;
- expires_at – дата, до якої повідомлення вважається дійсним;
- address – адреса (відкритий ключ) користувача.

9. Оброблені транзакції (processed_txs):

- id – унікальний ідентифікатор сутності обробленої транзакції;
- block – блок, коли транзакцію було прийнято;
- hash – унікальний хеш події з транзакції.

10. Контракти (contracts):

- id – унікальний ідентифікатор сутності контракту;
- address – адреса контракту;
- last_block – останній оброблений блок подій контракту.

3.5. Висновки до розділу 3

В ході розроблення вебзастосунку для випуску цифрових сертифікатів було спроектовано та описано загальну архітектуру системи, структуру та комунікацію її компонентів, алгоритми, схему бази даних застосунку.

В якості архітектурного шаблону було обрано клієнт-серверну архітектуру системи, що передбачає клієнтську та серверну частини програмного застосунку. В розробленні серверної частини вебзастосунку був використаний мікросервісний архітектурний стиль. Обрані протоколи комунікації – JSON HTTP та RPC. Детально були описані найважливіші алгоритми системи – авторизації користувача та безпосередньо випуску цифрового сертифікату з використанням технології блокчейну.

4. АНАЛІЗ РЕАЛІЗАЦІЇ ВЕБЗАСТОСУНКУ

4.1. Тестування вебзастосунку

Тестування вебзастосунку є важливим процесом розроблення програмного забезпечення. Воно дозволяє перевірити функціональні можливості, якість та продуктивність розробленого застосунку, виявити потенційні недоліки та критичні помилки у системі.

Тестування вебзастосунку для випуску цифрових сертифікатів здійснювалось на основі техніки Grey-box testing (тестування сірої скриньки) – метод тестування ПЗ при якому комбінуються White-box (біла скринька) та Black-box (чорна скринька) підходи тестування [28]. Детальніше розглянемо застосування обох технік для тестування розробленого вебзастосунку.

4.1.1. Тестування чорної скриньки

Підхід тестування чорної скриньки передбачає роботу виключно з зовнішніми інтерфейсами, такі як графічний інтерфейс користувача. Під час перевірки якості розробленої системи тестувальник не володіє даними щодо її внутрішньої архітектури та особливостей реалізації, тому сценарії тестування розробленого ПЗ проходять з користувацької сторони.

Серед основних протестованих процесів вебзастосунку для випуску цифрових сертифікатів методом чорної скриньки можна виділити:

- Авторизація користувача: перевірка успішної авторизації користувача; наявності токенів авторизації та оновлення, успішне оновлення авторизаційних токенів; перевірка доступу до захищених авторизацією ресурсів тощо.
- Перегляд випущених сертифікатів: наявність списку сертифікатів, що зареєстровані в системі; їх успішна фільтрація, пагінація, перегляд детальних даних щодо окремого документу.

- **Управління курсом:** успішне створення курсу; можливість додати/видалити осіб, що авторизовані до спільного випуску сертифікатів; перегляд сертифікатів, що були випущені в рамках цього курсу; можливість публікації, відкликання створених сертифікатів.
- **Дизайн шаблону сертифікату:** тестування коректної роботи елементів дизайну шаблону, можливість додавати статичні/динамічні атрибути сертифікату тощо.
- **Випуск сертифікату:** перевірка правильності заповнення динамічних атрибутів шаблону у цифровому документі, перевірка коректності даних, що збережені в системі та в блокчейн-мережі.
- **Верифікація сертифікату:** перевірка коректності процесу верифікації оригінальних та підроблених сертифікатів.

4.1.2. Тестування білої скриньки

Підхід тестування білої скриньки є протилежним до попередньої техніки, оскільки тестувальнику на етапі перевірки якості ПЗ відома вся внутрішня структура проєкту, кодова база, описані реалізації алгоритмів та потоки даних. Знання особливостей застосунку допомагає краще перевірити систему на відповідність вимогам, оскільки можна перевірити усі передбачувані та непередбачувані вхідні/вихідні дані, шляхи виконання процесів та розроблених алгоритмів. Доволі легко знаходити прогалини та недоліки у системі за допомогою тестів, коли виникає невідповідність вихідних даних процесу системи з наперед визначеними коректними значеннями.

Часто техніка тестування білої скриньки передбачає написання автоматизованих тест-кейсів для модульного, інтеграційного чи системного тестувань. Для тестування вебзастосунку для випуску цифрових сертифікатів на основі білої скриньки було розроблено набір модульних тестів, що перевіряють правильність реалізації набору смарт-контрактів

децентралізованої частини системи. Зокрема, перевірено коректність реалізації процесів створення курсу, випуску/відклику сертифікату, додавання/відкликання осіб, що авторизовані до спільного випуску сертифікатів тощо.

Розроблений набір тестів дозволив впевнитися у надійності та правильності реалізації смарт-контрактів ще перед їх безпосереднім розгортанням на блокчейн-мережі, що допомогло уникнути не тільки наявності помилок у системі, а й зберегти кошти, що потребуються для запуску ПЗ. Створення набору модульних тестів було реалізовано засобами фреймворку Hardhat. Також за допомогою його методів було визначено відсоток коду кожного зі смарт-контрактів, що є покритий тест-кейсами. Було досягнуто 100%-ового покриття ПЗ смарт-контрактів. Приклад виконання команди перевірки відсоткового покриття коду зображено на рис. 4.1.

```
slbmax@slbmax-asus-v15:~/go/src/github.com/slbmax/diploma/contracts/certificates$
npm run coverage

> hardhat-template@1.0.0 coverage
> npx hardhat coverage

Version
=====
> solidity-coverage: v0.8.2

Instrumenting for coverage...
=====

> CourseContract.sol
> CourseFactory.sol
> interfaces/ICourseContract.sol
> interfaces/ICourseFactory.sol

16 passing (4s)
```

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	100	100	100	100	
CourseContract.sol	100	100	100	100	
CourseFactory.sol	100	100	100	100	
contracts/interfaces/	100	100	100	100	
ICourseContract.sol	100	100	100	100	
ICourseFactory.sol	100	100	100	100	
All files	100	100	100	100	

```
> Istanbul reports written to ./coverage/ and ./coverage.json
slbmax@slbmax-asus-v15:~/go/src/github.com/slbmax/diploma/contracts/certificates$
```

Рис. 4.1. Виконання Hardhat-команди для перевірки покриття коду

4.2. Безперервні інтеграція та розгортання застосунку

У ході розробки ПЗ, його розгортання, тестування та подальшого покращення важливо забезпечити пришвидшення та автоматизацію цих процесів. Відповідні методології, що використовуються для впровадження вищенаведених характеристик процесів, називаються CI/CD – безперервні інтеграція та розгортання застосунку [29].

Процеси безперервної інтеграції (CI) застосунку спрямовані на інтеграцію змін в кодову базу ПЗ від декількох розробників одночасно. Основним переліком дій, що включають в себе процеси CI, є:

- Версіювання розробленого застосунку.
- Відстеження та збереження історії змін.
- Вирішення конфліктів.
- Збірка коду та автоматизоване відтворення тест-кейсів системи.

Процеси безперервного розгортання (CD) застосунку передбачають зберігання, перевірку та розгортання зібраного коду. Процеси CI та CD переплітаються між собою та є нечітко визначеними, тому часто до методології CD відносять також процес збірки та тестування програмного коду.

Методології CI/CD у вебзастосунку для випуску цифрових сертифікатів реалізовано за допомогою використання технології Git та платформи GitHub.

Git – потужний інструмент контролю та керування версіями файлів, який завдяки своїм функціональним можливостям став стандартом у світі розроблення програмного забезпечення [30]. Він дозволяє зручно версіонувати розроблення застосунку, прискорити випуск нових версій та покращити якість кінцевого продукту.

Разом з системою Git тісно використовується платформа GitHub – сервіс для зберігання Git-репозиторіїв [31]. Окрім збереження кодової бази та історії версій, GitHub надає розширний інструментарій для збірки та тестування власних проєктів в ізольованих середовищах.

З використанням технологій Git та GitHub для вебзастосунку для випуску цифрових сертифікатів було розроблено спеціальний репозиторій програмного коду, що підтримує версіонування, а також збірку і тестування програмного коду.

Збірка готового до запуску застосунку здійснюється з використанням інструментарію Docker [32]. Це технологія контейнеризації, що дозволяє створювати стандартизоване ізольоване середовище, що називається контейнером, для запуску ПЗ. Окрім цього, Docker агрегує у собі метод збору програмного коду, що повністю залежний від використаної мови програмування, надаючи користувачам просту функціональність запуску ПЗ без використання чи встановлення додаткових залежностей.

Збірка та тестування вебзастосунку для випуску цифрових сертифікатів здійснюється з використанням інструментарію Github Actions Workflows [33], що дозволяє створювати визначені користувачем автоматизовані процеси виконання задач. Загалом, було розроблено процеси для збірки ізольованих контейнерів виконання програмного коду мікросервісів та клієнтської частини застосунку, а також запуск та виконання модульних тестів смарт-контрактів.

Додатково було реалізовано процес доставки та зберігання зібраних контейнерів на платформу DockerHub [34]. Окрім зберігання Docker-контейнерів, платформа надає засоби для їх поширення серед інших розробників, тобто публічно опублікований контейнер може бути встановлений та запущений будь-ким, що дозволяє поширювати ПЗ та ділитися ним з іншими розробниками та користувачами.

Приклад збірки та доставки контейнера за допомогою інструментарію Github Workflows на платформу DockerHub зображено на рис. 4.2.

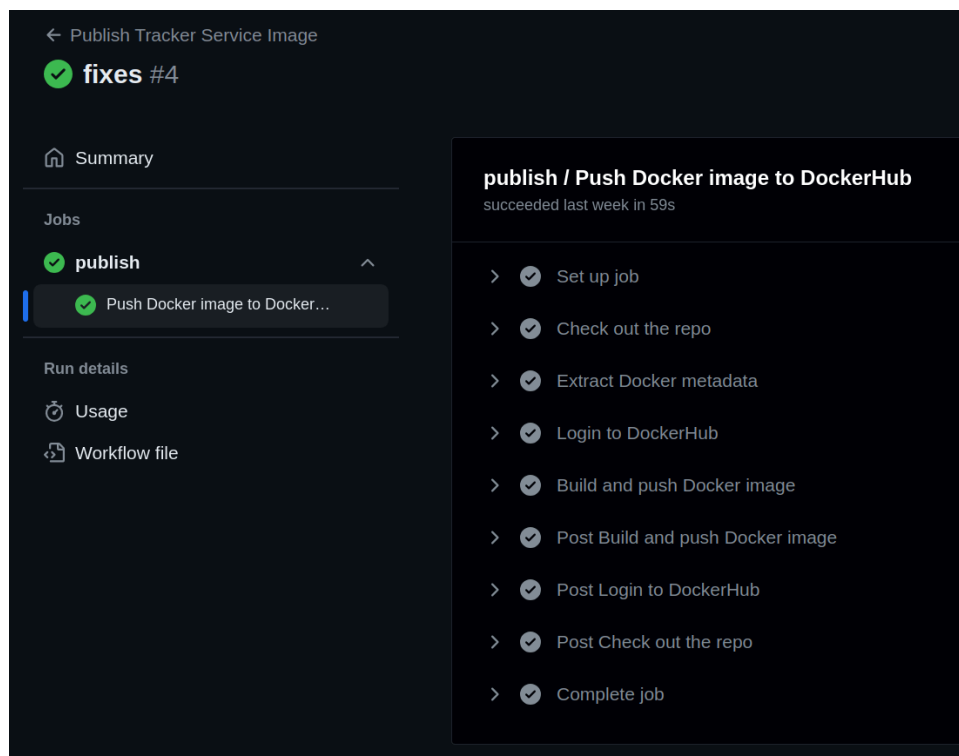


Рис. 4.2. Приклад виконання автоматизованого процесу Github Actions

4.3. Рекомендації щодо подальшого вдосконалення

Підвищення продуктивності, покращення існуючих та впровадження нових функціональних можливостей системи є важливим етапом життєвого циклу ПЗ. Важливо аналізувати результати роботи розробленого застосунку, дані користувацького досвіду та поточні технічні тренди для забезпечення зручної, якісної та надійної системи.

Можна виділити наступні основні напрямки покращення розробленого вебзастосунку для випуску цифрових сертифікатів:

- Розширення функціональних можливостей дизайну шаблонів сертифікатів: розроблена версія застосунку надає лише необхідні функціональні можливості для дизайну особистих шаблонів. Можливими напрямками її покращення та розширення є:
 - Додавання різноманітних елементів, такі як наліпки, стікери, фігури тощо.
 - Розроблення буферу подій для їх скасування чи повторення.

- Впровадження готових дизайнерських рішень шаблонів сертифікатів для різних сфер використання цифрових сертифікатів.
- Додавання різних стандартизованих форматів розміру сертифікату.
- Автоматизована система розсилання сертифікатів: використання поштової системи розсилки згенерованих цифрових сертифікатів дозволить моментально сповіщати користувачів про успішний випуск їх документу, що одразу можна завантажити.
- Інтеграція з освітніми платформами: впровадження інтеграції з різноманітними онлайн сервісами проходження курсів, тренінгів, навчальних програм тощо.
- Дослідження та впровадження новітніх технічних трендів: інтеграція сучасних технологій покращить безпеку, продуктивність та якість системи. Серед таких інновацій можна виділити технологію ZKP – доведення з нульовим розголошенням. Це спеціальні криптографічні методи, що дозволяють одній стороні доводити, що певне твердження являється істинним, при цьому не розкриваючи жодної інформації про твердження, окрім його достовірності [35]. Ця технологія стрімко розвивається та набуває поширення в межах децентралізованих технологій та блокчейну, оскільки допомагає підвищити приватність та безпеку конфіденційних даних у публічних мережах. За допомогою ZKP у вебзастосунку для випуску цифрових сертифікатів можна розробити функціональність, при якій власники сертифікатів зможуть вільно ділитися з верифікуючою стороною тими чи іншими частинами даних, не розкриваючи при цьому повну інформацію чи файл документа.
- Удосконалення графічного інтерфейсу користувача: розроблений інтерфейс користувача є простим та інтуїтивно-зрозумілим, та його

можна вдосконалити шляхом використання різноманітних підказок та керівництво щодо використання застосунку, спрощення систем фільтрування, пагінування сутностей та дизайну шаблонів сертифікату. Це допоможе користувачеві зручніше та ефективніше використовувати застосунок.

- Підтримка для мобільних версій вебзастосунку: наразі система для випуску цифрових сертифікатів орієнтована на десктопні браузері та розширення екрану, тому деякі компоненти системи можуть відображатись неправильно. Впровадження адаптивного дизайну та масштабування елементів графічного інтерфейсу користувача забезпечує зручне та ефективне використання застосунку на різних платформах.

4.4. Висновки до розділу 4

В ході аналізу реалізації вебзастосунку для випуску цифрових сертифікатів було розглянуто процес тестування якості розробленого ПЗ техніками чорної та білої скриньок з використанням інструментарію фреймворку Hardhat. Крім того, було розглянуто концепції методологій CI/CD та способи їх реалізації в системі. Також було виявлено та досліджено основні напрямки вдосконалення розробленого застосунку.

ВИСНОВКИ

Метою дипломного проєкту було забезпечити зручний та водночас надійний процес випуску та верифікації цифрових сертифікатів. Підставою для розроблення застосунку стало завдання на дипломне проєктування, затверджене кафедрою ПЗКС НТУУ «КПІ ім. Ігоря Сікорського».

На етапі аналізу предметної області було розглянуто її особливості та проаналізовано існуючі рішення. На основі їх переваг та недоліків було встановлено актуальність розроблення вебзастосунку, оскільки жоден з наявних аналогів не задовольняє всі вимоги зручного та водночас надійного процесу випуску цифрових сертифікатів.

В ході аналізу засобів розробки застосунку було обрано EVM-сумісну блокчейн-мережу Polygon, мови програмування Go і Solidity для реалізації бізнес-логіки системи, JavaScript і фреймворк Vue.js – для графічного інтерфейсу користувача, СКБД PostgreSQL.

Під час розроблення вебзастосунку для випуску цифрових сертифікатів було описано структури та взаємозв'язки у системі, деталі реалізації окремих компонент та алгоритмів, спроектовано структуру бази даних застосунку. В основу розробленого ПЗ було покладено клієнт-серверну архітектуру з використанням мікросервісного підходу.

Розроблений вебзастосунок дозволяє користувачам системи зручно та легко переглядати, створювати, публікувати та верифікувати цифрові сертифікати. Використання технології блокчейну та криптографічних методів підвищує рівень прозорості та безпеки системи. Додатково було впроваджено методи безперервної інтеграції та розгортання, досліджено основні напрямки подальшого вдосконалення застосунку.

Розробка виконана у повному обсязі, реалізовані вимоги до програмного продукту, що наведені у технічному завданні. Розроблено вичерпну документацію проєкту, тестування ПЗ проведено відповідно до програми та методики тестування.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

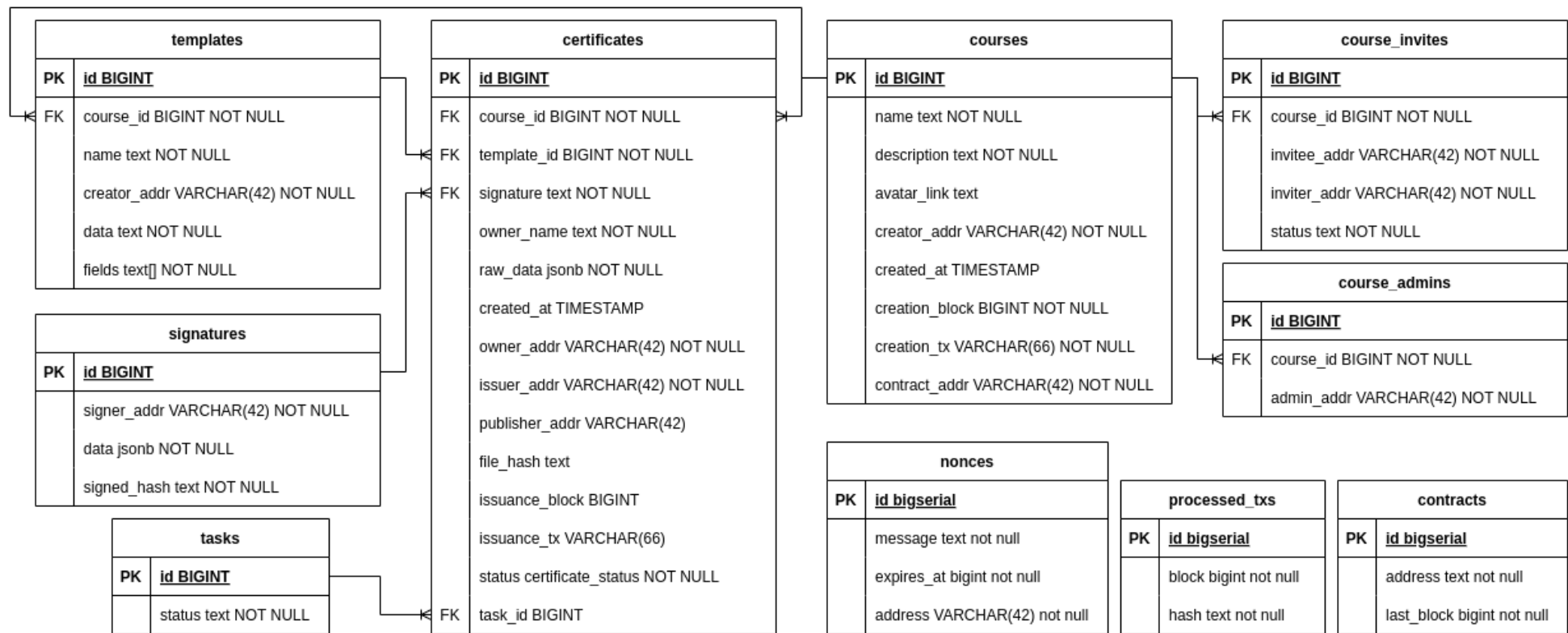
1. Peck D. Online Learning Statistics: The Ultimate List in 2024 [Електронний ресурс]. / Devlin Peck – Режим доступу: <https://www.devlinpeck.com/content/online-learning-statistics> – (03.02.2024).
2. What is Blockchain Technology [Електронний ресурс]. – Режим доступу: <https://aws.amazon.com/what-is/blockchain> – (06.02.2024).
3. KYC (Know Your Client): що це і навіщо? [Електронний ресурс]. – Режим доступу: <https://youcontrol.com.ua/blog/kyc-know-your-customer-shcho-tse-i-navishcho/> – (06.02.2024).
4. Certifier [Електронний ресурс]. – Режим доступу: <https://certifier.io/> – (10.02.2024).
5. DoxyChain [Електронний ресурс]. – Режим доступу: <https://www.doxychain.com/> – (12.02.2024).
6. Accredible [Електронний ресурс]. – Режим доступу: <https://www.accreditable.com/> – (15.02.2024).
7. Ethereum Virtual Machine (EVM) [Електронний ресурс]. – Режим доступу: <https://ethereum.org/en/developers/docs/evm/> – (15.02.2024).
8. What is Ethereum? [Електронний ресурс]. – Режим доступу: <https://ethereum.org/en/what-is-ethereum/> – (28.02.2024).
9. Початок роботи з Біткойн [Електронний ресурс]. – Режим доступу: <https://bitcoin.org/uk/getting-started> – (02.03.2024).
10. Solana [Електронний ресурс]. – Режим доступу: <https://solana.com/> – (02.03.2024).
11. What is Polygon (MATIC)? [Електронний ресурс]. – Режим доступу: <https://www.coinbase.com/learn/crypto-basics/what-is-polygon> – (05.03.2024).
12. Solidity [Електронний ресурс]. – Режим доступу: <https://soliditylang.org/> – (05.03.2024).

13. Vyper [Электронный ресурс]. – Режим доступа: <https://docs.vyperlang.org/en/stable/> – (06.03.2024).
14. Python: The most popular language [Электронный ресурс]. – Режим доступа: <https://datascientest.com/en/python-the-most-popular-language> – (10.03.2024).
15. Ajitsaria A. What Is the Python Global Interpreter Lock (GIL)? / Abhinav Ajitsaria [Электронный ресурс]. – Режим доступа: <https://realpython.com/python-gil/> – (10.03.2024).
16. Go [Электронный ресурс]. – Режим доступа: <https://go.dev/> – (12.03.2024).
17. Rust [Электронный ресурс]. – Режим доступа: <https://www.rust-lang.org/> – (12.03.2024).
18. React [Электронный ресурс]. – Режим доступа: <https://react.dev/> – (18.03.2024).
19. Vue.js [Электронный ресурс]. – Режим доступа: <https://vuejs.org/> – (19.03.2024).
20. What Is AngularJS? [Электронный ресурс]. – Режим доступа: <https://builtin.com/software-engineering-perspectives/angularjs> – (21.03.2024).
21. PostgreSQL [Электронный ресурс]. – Режим доступа: <https://www.postgresql.org/> – (23.03.2024).
22. MySQL [Электронный ресурс]. – Режим доступа: <https://www.mysql.com/> – (23.03.2024).
23. NGINX [Электронный ресурс]. – Режим доступа: <https://nginx.org/en/> – (01.04.2024).
24. Amazon S3 [Электронный ресурс]. – Режим доступа: <https://aws.amazon.com/s3/> – (03.04.2024).
25. Getting started with Hardhat [Электронный ресурс]. – Режим доступа: <https://hardhat.org/hardhat-runner/docs/getting-started> – (05.04.2024).

26. Introduction | Pinia [Електронний ресурс]. – Режим доступу: <https://pinia.vuejs.org/introduction> – (16.04.2024).
27. Metamask [Електронний ресурс]. – Режим доступу: <https://metamask.io/> – (16.04.2024).
28. White/black/grey box-тестування [Електронний ресурс]. – Режим доступу: <https://qalight.ua/baza-znaniy/white-black-grey-box-testuvannya/> – (13.05.2024).
29. Філімонов С. Що таке CI/CD, як він працює та коли знадобиться на проєкті. Лайфхаки та bad practices / Сергій Філімонов, Микита Андреев [Електронний ресурс]. – Режим доступу: <https://highload.today/uk/blogs/shho-take-ci-cd-yak-vin-pratsyuie-ta-koli-znadbitysia-na-proiekti-lajfhaki-ta-bad-practices/> – (14.05.2024).
30. Git [Електронний ресурс]. – Режим доступу: <https://git-scm.com/> – (14.05.2024).
31. What Is GitHub? A Beginner's Introduction to GitHub [Електронний ресурс]. – Режим доступу: <https://kinsta.com/knowledgebase/what-is-github/> – (14.05.2024).
32. Docker Overview [Електронний ресурс]. – Режим доступу: <https://docs.docker.com/get-started/overview/> – (14.05.2024).
33. Using workflows - GitHub Docs [Електронний ресурс]. – Режим доступу: <https://docs.github.com/en/actions/using-workflows> – (14.05.2024).
34. Docker Hub Container Image Library | App Containerization [Електронний ресурс]. – Режим доступу: <https://hub.docker.com/> – (15.05.2024).
35. What are zero-knowledge proofs? [Електронний ресурс]. – Режим доступу: <https://www.coinbase.com/learn/crypto-glossary/what-are-zero-knowledge-proofs> – (17.05.2024).

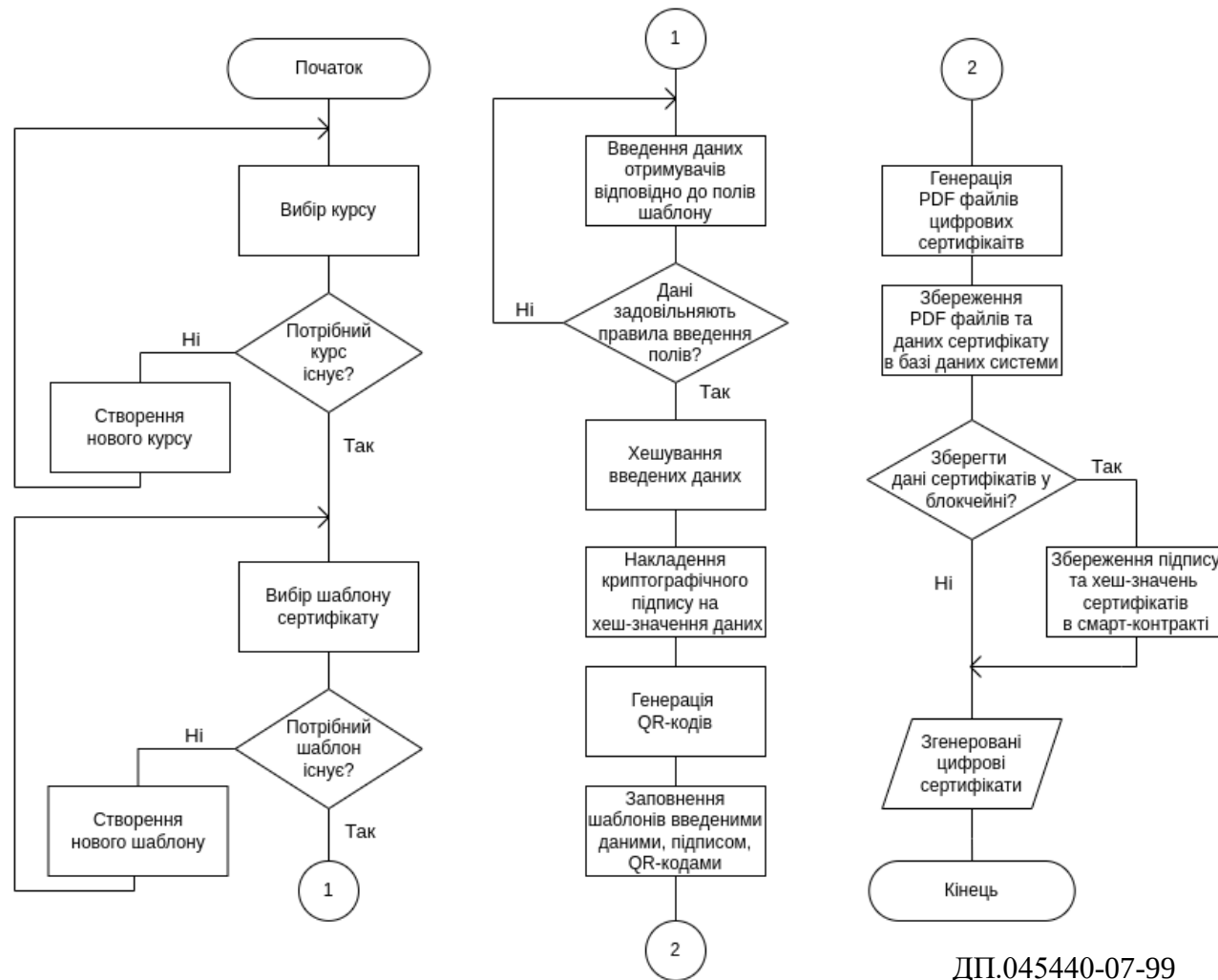
ДОДАТКИ

Додаток 1
Копії графічних елементів



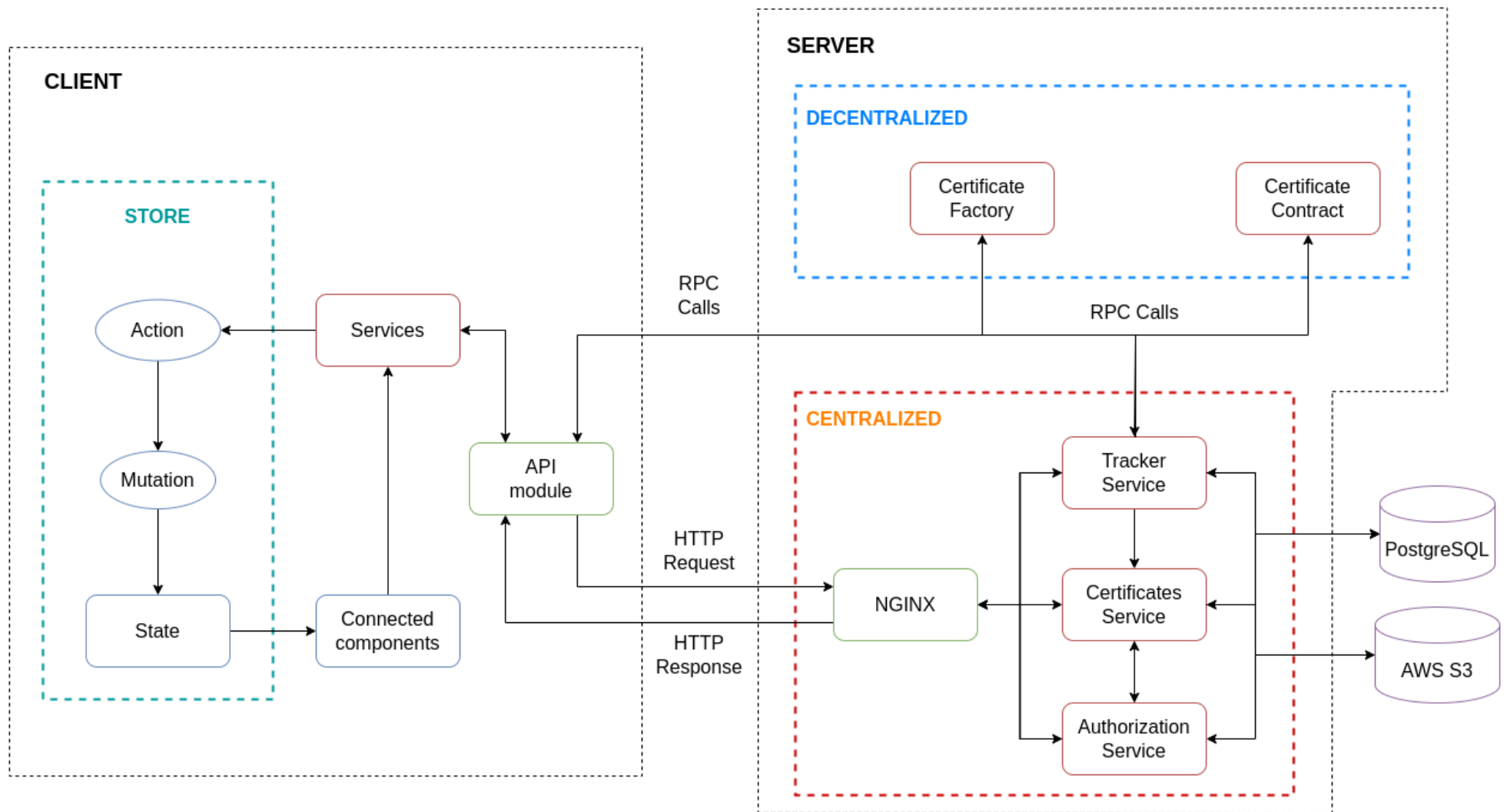
ДП.045440-06-99

Вебзастосунок для випуску цифрових
сертифікатів на основі технології блокчейн.
Структура бази даних. ER-діаграма



ДП.045440-07-99

Вебзастосунок для випуску цифрових сертифікатів на основі технології блокчейн.
Алгоритм випуску цифрових сертифікатів.
Блок-схема алгоритму



Архітектура застосунку
Слободзян М.В., група КП-02

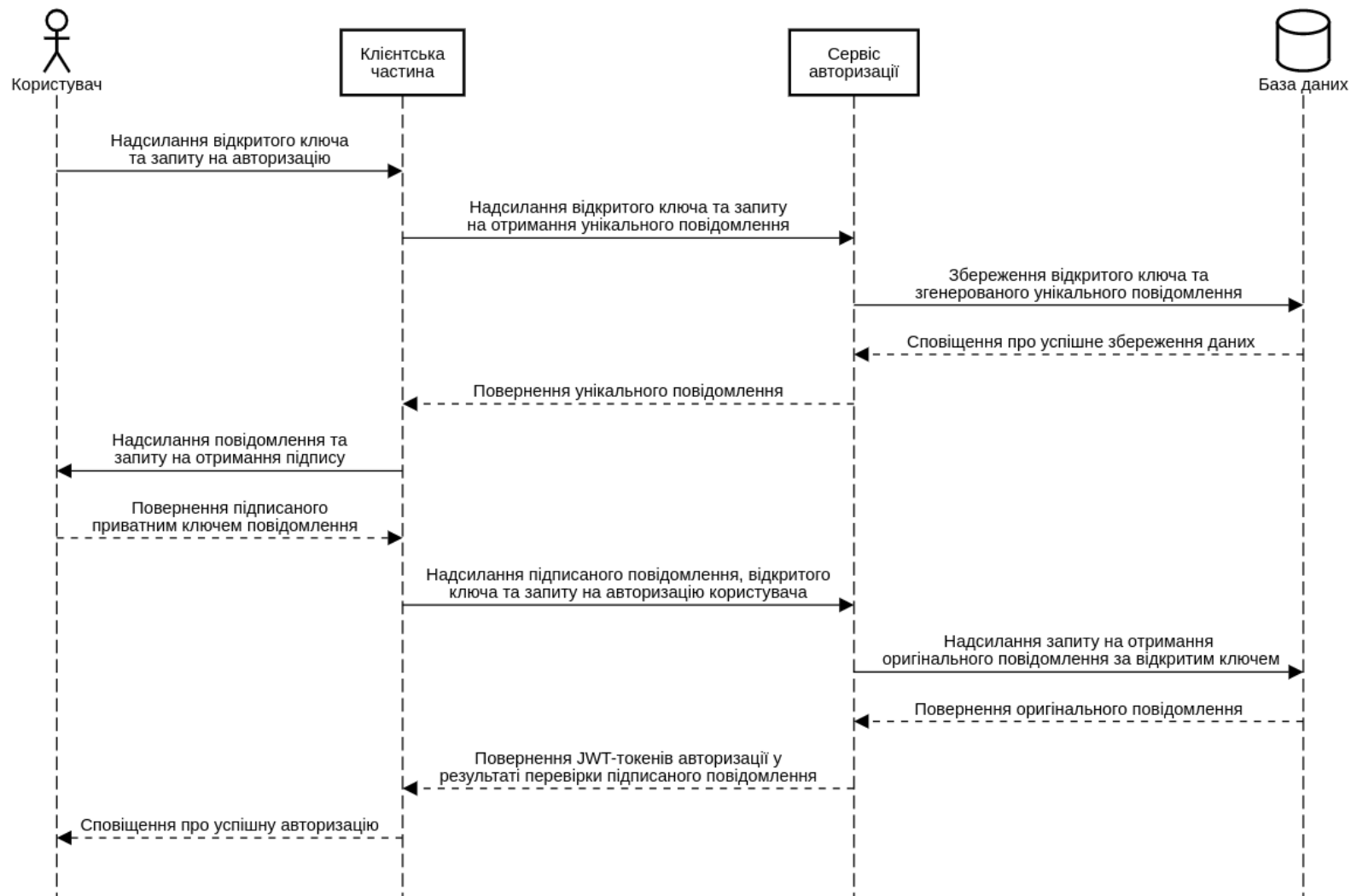


Схема авторизації користувача на основі РКС
Слободзян М.В., група КП-02

Додаток 2
Лістинг програми

Фрагменти коду смарт-контрактів

```
pragma solidity ^0.8.19;

import "./interfaces/ICourseFactory.sol";
import "./interfaces/ICourseContract.sol";
import "./CourseContract.sol";
import "@openzeppelin/contracts/utils/structs/EnumerableSet.sol";
import "@solarity/solidity-lib/libs/arrays/Paginator.sol";

contract CourseFactory is ICourseFactory {

    using EnumerableSet for EnumerableSet.AddressSet;
    using Paginator for EnumerableSet.AddressSet;

    EnumerableSet.AddressSet internal _courseContracts;

    function deployCourseContract(
        DeployCourseContractParams calldata params_
    ) external {
        address newCourseContract_ = address(new CourseContract(
            ICourseContract.CourseContractInitParams(
                params_.courseName,
                params_.description,
                address(this),
                msg.sender
            )))

        _courseContracts.add(newCourseContract_);

        emit CourseContractDeployed(msg.sender, newCourseContract_,
params_);
    }

    function getCourseContractsCount() external view returns (uint256) {
        return _courseContracts.length();
    }
}

pragma solidity ^0.8.19;

import "./interfaces/ICourseContract.sol";
import "./interfaces/ICourseFactory.sol";

contract CourseContract is ICourseContract {
    ICourseFactory public override courseFactory;

    string public name;
    string public description;

    // issuer address => is issuer or not
    mapping(address => bool) public issuers;
    // sha256 hash => certificate owner
    mapping(string => address) public certificateOwners;
    // holder address => sha256 hash => certificate data
    mapping(address => mapping(string => CertificateData)) public
userCertificates;

    // modifier to check whether the caller is an admin or not
    modifier onlyIssuer() {
        require(issuers[msg.sender], "Caller is not the issuer");
        _;
    }
}
```

```

    constructor( CourseContractInitParams memory params_ ) {
        courseFactory = ICourseFactory(params_.courseFactoryAddress);
        description = params_.courseDescription;
        name = params_.courseName;

        issuers[params_.issuer] = true;
    }

    function issueCertificates(address issuer, IssueCertificateData[]
calldata data) external onlyIssuer {
        for (uint i = 0; i < data.length; i++ ) {
            IssueCertificateData calldata certificateData = data[i];

            require(certificateOwners[certificateData.sha256Hash] ==
address(0), "Certificate already issued");

            CertificateData memory certificate;
            certificate.sha256Hash = certificateData.sha256Hash;
            certificate.holder = certificateData.holder;
            certificate.signature = certificateData.signature;

            certificate.issuer = issuer;
            certificate.publisher = msg.sender;
            certificate.issuedBlockNumber = block.number;

            userCertificates[certificate.holder][certificate.sha256Hash] =
certificate;
            certificateOwners[certificate.sha256Hash] = certificate.holder;

            emit CertificateIssued(certificate.publisher,
certificate.sha256Hash);
        }
    }

    function revokeCertificate(string calldata _sha256Hash) external
onlyIssuer {
        address holder = certificateOwners[_sha256Hash];
        require(holder != address(0), "Certificate not found");

        userCertificates[holder][_sha256Hash].isRevoked = true;

        emit CertificateRevoked(_sha256Hash);
    }

    function addIssuer(address newIssuer) external onlyIssuer {
        require(!issuers[newIssuer], "Issuer already exists");

        issuers[newIssuer] = true;

        emit IssuerAdded(newIssuer);
    }

    function deleteIssuer(address oldIssuer) external onlyIssuer {
        require(oldIssuer != msg.sender, "Cannot remove yourself");
        require(issuers[oldIssuer], "Issuer does not exist");

        issuers[oldIssuer] = false;

        emit IssuerDeleted(oldIssuer);
    }
}

```

Фрагменти коду сервісу авторизації

```
package handlers

import (
    "net/http"

    "github.com/slbmax/diploma/backend/auth-svc/internal/nonce"
    "github.com/slbmax/diploma/backend/auth-svc/internal/service/api/helpers"
    "github.com/slbmax/diploma/backend/auth-svc/internal/service/api/ctx"
    "github.com/slbmax/diploma/backend/auth-svc/internal/service/api/requests"
    "gitlab.com/distributed_lab/ape"
    "gitlab.com/distributed_lab/ape/problems"
)

func Login(w http.ResponseWriter, r *http.Request) {
    // validating request
    request, err := requests.NewLoginRequest(r)
    if err != nil {
        ape.RenderErr(w, problems.BadRequest(err)...)
        return
    }

    var (
        logger = ctx.Log(r)
        db      = ctx.DB(r)
        conf    = ctx.ServiceConfig(r)
    )

    userNonce, err := db.Nonce().FilterByAddress(request.Address).Get()
    if err != nil {
        logger.WithError(err).Error("failed to query db")
        ape.RenderErr(w, problems.InternalError())
        return
    }
    if userNonce == nil {
        ape.RenderErr(w, problems.NotFound())
        return
    }

    if err = db.Nonce().FilterByAddress(request.Address).Delete(); err != nil {
        logger.WithError(err).Error("failed to query db")
        ape.RenderErr(w, problems.InternalError())
        return
    }

    if err = nonce.VerifySignature(userNonce.ToHash(),
        request.SignedMessage, request.Address); err != nil {
        logger.WithError(err).Error("failed to verify signature")
        ape.RenderErr(w, problems.Unauthorized())
        return
    }

    // generating new JWT pair
    pair, err := helpers.GenerateJwtPair(request.Address, conf)
    if err != nil {
        logger.WithError(err).Error("failed to generate jwt")
        ape.RenderErr(w, problems.InternalError())
        return
    }
}
```

```

    }

    ape.Render(w, pair)
}

package handlers

import (
    "net/http"

    "github.com/slbmax/diploma/backend/auth-svc/internal/nonce"
    "github.com/slbmax/diploma/backend/auth-
svc/internal/service/api/helpers"

    "github.com/slbmax/diploma/backend/auth-svc/internal/service/api/ctx"
    "github.com/slbmax/diploma/backend/auth-
svc/internal/service/api/requests"
    "gitlab.com/distributed_lab/ape"
    "gitlab.com/distributed_lab/ape/problems"
)

func Login(w http.ResponseWriter, r *http.Request) {
    // validating request
    request, err := requests.NewLoginRequest(r)
    if err != nil {
        ape.RenderErr(w, problems.BadRequest(err)...)
        return
    }

    var (
        logger = ctx.Log(r)
        db      = ctx.DB(r)
        conf    = ctx.ServiceConfig(r)
    )

    userNonce, err := db.Nonce().FilterByAddress(request.Address).Get()
    if err != nil {
        logger.WithError(err).Error("failed to query db")
        ape.RenderErr(w, problems.InternalError())
        return
    }

    if userNonce == nil {
        ape.RenderErr(w, problems.NotFound())
        return
    }

    if err = db.Nonce().FilterByAddress(request.Address).Delete(); err !=
nil {
        logger.WithError(err).Error("failed to query db")
        ape.RenderErr(w, problems.InternalError())
        return
    }

    if err = nonce.VerifySignature(userNonce.ToHash(),
request.SignedMessage, request.Address); err != nil {
        logger.WithError(err).Error("failed to verify signature")
        ape.RenderErr(w, problems.Unauthorized())
        return
    }

    // generating new JWT pair
    pair, err := helpers.GenerateJwtPair(request.Address, conf)
    if err != nil {
        logger.WithError(err).Error("failed to generate jwt")
    }
}

```

```

        ape.RenderErr(w, problems.InternalError())
        return
    }

    ape.Render(w, pair)
}

package nonce

import (
    "github.com/ethereum/go-ethereum/common/hexutil"
    "github.com/ethereum/go-ethereum/crypto"
    "gitlab.com/distributed_lab/logan/v3/errors"
    "strings"
)

func DecodeSignature(signature string) ([]byte, error) {
    signatureBytes, err := hexutil.Decode(signature)
    if err != nil {
        return nil, err
    }
    if len(signatureBytes) != 65 {
        return nil, errors.New("bad signature length")
    }
    if signatureBytes[64] == 0 || signatureBytes[64] == 1 {
        signatureBytes[64] = signatureBytes[64] + 27
    }
    if signatureBytes[64] != 27 && signatureBytes[64] != 28 {
        return nil, errors.New("bad recovery byte")
    }
    signatureBytes[64] -= 27

    return signatureBytes, nil
}

func VerifySignature(hash []byte, signature string, addresses ...string)
error {
    if len(addresses) < 1 {
        return errors.New("no addresses provided for signature
verification")
    }

    signatureBytes, err := DecodeSignature(signature)
    if err != nil {
        return errors.Wrap(err, "failed to decode signature")
    }

    recoveredPubkey, err := crypto.SigToPub(hash, signatureBytes)
    if err != nil {
        return errors.Wrap(err, "failed to recover pubkey from signed
message")
    }

    recoveredAddr :=
strings.ToLower(crypto.PubkeyToAddress(*recoveredPubkey).Hex())

    for _, address := range addresses {
        if address == recoveredAddr {
            return nil
        }
    }
}

```

```

        return errors.New("recovered address didn't match any of the given
ones")
    }

package nonce

import (
    "crypto/md5"
    "crypto/rand"
    "fmt"
    "io"
    "math"
    "math/big"
    "strconv"
    "time"
)

func New() (nonce string, timestamp time.Time, err error) {
    timestamp = time.Now().UTC()
    hash := md5.New()
    randInt, err := rand.Int(rand.Reader, big.NewInt(math.MaxInt64))
    if err != nil {
        return
    }
    if _, err = io.WriteString(hash, strconv.FormatInt(timestamp.Unix(),
10)); err != nil {
        return
    }
    if _, err = io.WriteString(hash, strconv.FormatInt(randInt.Int64(),
10)); err != nil {
        return
    }
    nonce = fmt.Sprintf("%x", hash.Sum(nil))
    return
}

func GetSigningMessage(nonce string, timestamp int64, userAddr string)
string {
    return fmt.Sprintf(
        "Authentication request\n"+
        "User: %s\n"+
        "Timestamp: %v\n"+
        "Nonce: %s",
        userAddr, timestamp, nonce,
    )
}

func EthPrefixedMessage(message string) string {
    prefixedMessage := fmt.Sprintf("\x19Ethereum Signed Message:\n%d%s",
len(message), message)
    return prefixedMessage
}

```

Фрагменти коду відслідковування блокчейн-подій

```

package listeners

import (
    "context"

    "github.com/ethereum/go-ethereum/common"
    "github.com/ethereum/go-ethereum/ethclient"
    "github.com/pkg/errors"

```

```

        "github.com/slbmax/diploma/backend/tracker-svc/internal/config"
        "github.com/slbmax/diploma/backend/tracker-svc/internal/data"
        postgres "github.com/slbmax/diploma/backend/tracker-
svc/internal/data/pg"
        "github.com/slbmax/diploma/backend/tracker-
svc/internal/service/tracking/executors"
        "gitlab.com/distributed_lab/logan/v3"
        "gitlab.com/distributed_lab/running"
    )

type CertificatesListener struct {
    db          data.DB
    logger      *logan.Entry
    settings    config.CertificatesListenerSettings
    node        *ethclient.Client
    executor    *executors.CertificatesExecutor
}

func NewCertificatesListener(cfg config.Config) *CertificatesListener {
    return &CertificatesListener{
        db:          postgres.New(cfg.DB()),
        settings:    cfg.CertificatesListenerSettings(),
        logger:      cfg.Log().WithField("service",
"certificates_listener"),
        executor:    executors.NewCertificatesExecutor(cfg),
        node:        cfg.Node(),
    }
}

func (l *CertificatesListener) InsertFactory() error {
    return l.db.Contracts().Insert(data.Contract{
        Address:    l.settings.Factory,
        LastBlock:  int64(l.settings.StartBlock),
    })
}

func (l *CertificatesListener) Listen(ctx context.Context) {
    running.WithBackOff(
        ctx,
        l.logger,
        "certificates_runner",
        l.processContracts,
        l.settings.Backoff.NormalPeriod,
        l.settings.Backoff.MinAbnormalPeriod,
        l.settings.Backoff.MaxAbnormalPeriod,
    )
}

func (l *CertificatesListener) processContracts(ctx context.Context) error {
    {
        contracts, err := l.db.Contracts().Select()
        if err != nil {
            return errors.Wrap(err, "failed to select contracts")
        }
        if len(contracts) == 0 {
            l.logger.Info("no contracts to process")
            return nil
        }
    }

    from, to, addresses, err := l.resolveExecutorParams(contracts)
    if err != nil {
        return errors.Wrap(err, "failed to resolve executor params")
    }
}

```

```

        l.logger.Debug("Iterating from %d to %d blocks...", from, to)

        if err =
            l.executor.FromBlock(from).ToBlock(to).WithAddresses(addresses).Execute(ctx); err != nil {
                return errors.Wrap(err, "failed to execute certificates executor")
            }

        if err =
            l.db.Contracts().UpdateLastBlock(stringifyAddresses(addresses), int64(to)); err != nil {
                return errors.Wrap(err, "failed to update last block")
            }

        return nil
    }

    func (l *CertificatesListener) resolveExecutorParams(
        contracts []data.Contract,
    ) (from uint64, to uint64, addresses []common.Address, err error) {
        from = ^uint64(0)

        for _, contract := range contracts {
            if uint64(contract.LastBlock) < from {
                from = uint64(contract.LastBlock)
            }
        }

        addresses = append(addresses,
            common.HexToAddress(contract.Address))
    }

    lastBlock, err := l.node.BlockNumber(context.Background())
    if err != nil {
        return 0, 0, nil, errors.Wrap(err, "failed to get last block number")
    }

    if from+l.settings.IterationSize > lastBlock {
        return from, lastBlock, addresses, nil
    } else {
        return from, from + l.settings.IterationSize, addresses, nil
    }
}

func stringifyAddresses(addresses []common.Address) (arr []string) {
    for _, address := range addresses {
        arr = append(arr, address.Hex())
    }

    return
}

package executors

import (
    "context"
    "fmt"
    "math/big"

    "github.com/ethereum/go-ethereum"
    "github.com/ethereum/go-ethereum/common"
    "github.com/ethereum/go-ethereum/core/types"
    "github.com/ethereum/go-ethereum/ethclient"

```

```

        "github.com/pkg/errors"
        "github.com/slbmax/diploma/backend/tracker-svc/internal/config"
        "github.com/slbmax/diploma/backend/tracker-svc/internal/data"
        postgres "github.com/slbmax/diploma/backend/tracker-
svc/internal/data/pg"
        "github.com/slbmax/diploma/backend/tracker-
svc/internal/service/tracking/handlers"
        "gitlab.com/distributed_lab/logan/v3"
    )

type CertificatesExecutor struct {
    processedTxns data.ProcessedTxQ
    log            *logan.Entry

    certificatesHandler *handlers.CertificatesHandler
    factoryHandler      *handlers.CertificatesFactoryHandler

    from, to *uint64
    node     *ethclient.Client
    addresses []common.Address
}

func NewCertificatesExecutor(cfg config.Config) *CertificatesExecutor {
    return &CertificatesExecutor{
        log:            cfg.Log().WithFields(logan.F{"component":
"certificates_executor"}),
        node:           cfg.Node(),
        certificatesHandler: handlers.NewCertificatesHandler(cfg),
        factoryHandler:  handlers.NewCertificatesFactoryHandler(cfg),
        processedTxns:   postgres.NewProcessedTxnsQ(cfg.DB()),
    }
}

func (e *CertificatesExecutor) WithAddresses(address []common.Address)
*CertificatesExecutor {
    e.addresses = address
    return e
}

func (e *CertificatesExecutor) FromBlock(from uint64) *CertificatesExecutor
{
    e.from = &from
    return e
}

func (e *CertificatesExecutor) ToBlock(to uint64) *CertificatesExecutor {
    e.to = &to
    return e
}

func (e *CertificatesExecutor) validateParameters() error {
    if e.addresses == nil || e.from == nil || e.to == nil {
        return errors.New("addresses, from and to parameters are
required")
    }
    return nil
}

func (e *CertificatesExecutor) Execute(ctx context.Context) error {
    if err := e.validateParameters(); err != nil {
        return err
    }
}

```

```

        topics := append(e.certificatesHandler.SupportedTopics(),
e.factoryHandler.SupportedTopics()...)

    logs, err := e.node.FilterLogs(ctx, ethereum.FilterQuery{
        FromBlock: new(big.Int).SetUint64(*e.from),
        ToBlock:    new(big.Int).SetUint64(*e.to),
        Addresses: e.addresses,
        Topics:    [][]common.Hash{topics},
    })
    if err != nil {
        return errors.Wrap(err, "failed to get logs based on filter")
    }
    if len(logs) == 0 {
        return nil
    }

    e.log.Debug(fmt.Sprintf("Found %d logs", len(logs)))

    for _, log := range logs {
        uid := e.getLogUniqueID(log)

        tx, err := e.processedTxs.FilterByHash(uid).Get()
        if err != nil {
            return errors.Wrap(err, "failed to get processed tx by
hash")
        }
        if tx != nil {
            e.log.Infof("Transaction with hash %s is already
processed, omitting...", log.TxHash.Hex())
            continue
        }

        if e.factoryHandler.SupportsLog(log) {
            if err = e.factoryHandler.Handle(log); err != nil {
                return errors.Wrap(err, "failed to handle factory
log")
            }
        } else if e.certificatesHandler.SupportsLog(log) {
            if err = e.certificatesHandler.Handle(log); err != nil {
                return errors.Wrap(err, "failed to handle
certificates log")
            }
        }

        if err = e.processedTxs.Insert(data.ProcessedTx{
            Hash: uid,
            Block: log.BlockNumber,
        }); err != nil {
            return errors.Wrap(err, "failed to insert processed tx")
        }
    }

    if err = e.certificatesHandler.SubmitIssuedCertificates(); err != nil
{
        return errors.Wrap(err, "failed to submit issued certificates")
    }

    return nil
}

func (e *CertificatesExecutor) getLogUniqueID(log types.Log) string {
    return fmt.Sprintf("%s-%v", log.TxHash.String(), log.Index)
}

```

Фрагменти коду генерації цифрового сертифікату

```
package issuer

import (
    "context"
    "crypto/sha256"
    "fmt"
    "sync"
    "time"

    "github.com/pkg/errors"
    "github.com/slbmax/diploma/backend/certificates-
svc/internal/certificates/generator"
    "github.com/slbmax/diploma/backend/certificates-svc/internal/config"
    "github.com/slbmax/diploma/backend/certificates-svc/internal/data"
    "github.com/slbmax/diploma/backend/certificates-svc/internal/data/pg"
    "github.com/slbmax/diploma/backend/certificates-svc/internal/s3"
    "gitlab.com/distributed_lab/logan/v3"
    "gitlab.com/distributed_lab/running"
)

const (
    maxConcurrency = 50
)

type CertificateIssuer interface {
    Run(ctx context.Context)
}

type certificateIssuer struct {
    db      data.DB
    s3      s3.CertificatesClient
    logger  *logan.Entry
    certgen generator.Generator
}

func NewCertificateIssuer(config config.Config) CertificateIssuer {
    return &certificateIssuer{
        db:      pg.New(config.DB()),
        s3:      config.CertificatesClient(),
        certgen: config.CertificateGenerator(),
        logger:  config.Log().WithField("service", "certificate-
issuer"),
    }
}

func (i *certificateIssuer) Run(ctx context.Context) {
    running.WithBackOff(
        ctx,
        i.logger,
        "certificate-issuer",
        i.issueCertificates,
        time.Second*2,
        time.Second*2,
        time.Second*2,
    )
}

func (i *certificateIssuer) issueCertificates(parentCtx context.Context)
error {
    pendingCertificates, err :=
i.db.Certificates().SelectPendingIssuance()
```

```

        if err != nil {
            return errors.Wrap(err, "failed to select pending
certificates")
        }
        if len(pendingCertificates) == 0 {
            return nil
        }

        i.logger.WithFields(logan.F{"count":
len(pendingCertificates)}).Debug("found pending certificates")

        var (
            // limit the number of concurrent goroutines
            limiter = make(chan struct{}, maxConcurrency)
            certs   = make(chan data.Certificate, maxConcurrency)
            errs    = make(chan error, maxConcurrency)

            processedCertificates []data.Certificate

            ctx, cancel = context.WithCancel(parentCtx)
        )

        defer cancel()

        go func() {
            wg := sync.WaitGroup{}

            for _, cert := range pendingCertificates {
                select {
                case <-ctx.Done():
                    i.logger.Debug("context is done, stopping
processing certificates")
                    errs <- ctx.Err()
                    return
                case limiter <- struct{}{}:
                    wg.Add(1)
                }

                go func(cert data.CertificateIssuanceData) {
                    defer wg.Done()

                    i.logger.WithFields(logan.F{"id":
cert.Id}).Debug("processing certificate")

                    created, er := i.processCertificate(cert)
                    if er != nil {
                        errs <- errors.Wrap(er, "failed to process
certificate")
                    }

                    return

                    certs <- created
                    <-limiter

                    i.logger.WithFields(logan.F{"id":
cert.Id}).Debug("certificate processed")
                }(cert)
            }

            wg.Wait()
            close(certs)
        }()

        for {

```

```

        select {
        case <-parentCtx.Done():
            return parentCtx.Err()
        case cert, ok := <-certs:
            if !ok {
                if err =
i.db.Certificates().UpdateCreated(processedCertificates); err != nil {
                    return errors.Wrap(err, "failed to update
created certificates")
                }

                i.logger.WithFields(logan.F{"count":
len(pendingCertificates)}).Debug("certificates processed")

                return nil
            }

            processedCertificates = append(processedCertificates,
cert)

            case err = <-errs:
                return errors.Wrap(err, "failed to process certificate")
            }
        }

func (i *certificateIssuer) processCertificate(certData
data.CertificateIssuanceData) (data.Certificate, error) {
    // generating pdf
    certificateRaw, err := i.certgen.Generate(certData)
    if err != nil {
        return certData.Certificate, errors.Wrap(err, "failed to
generate certificate")
    }

    // uploading to s3
    if err = i.s3.UploadCertificate(certData.Certificate.Id,
certificateRaw); err != nil {
        return certData.Certificate, errors.Wrap(err, "failed to upload
certificate")
    }

    // calculating hash
    hash := fmt.Sprintf("%x", sha256.Sum256(certificateRaw))

    cert := certData.Certificate
    cert.FileHash = &hash
    cert.Status = data.CertificateStatusCreated

    return cert, nil
}

```

package generator

```

import (
    "bytes"
    "encoding/base64"
    "encoding/json"
    "fmt"
    "html/template"
    "io"

    "github.com/skip2/go-qrcode"

    "github.com/SebastiaanKlippert/go-wkhtmltopdf"

```

```

        "github.com/slbmax/diploma/backend/certificates-svc/internal/data"
        "gitlab.com/distributed_lab/logan/v3/errors"
    )

    type Generator interface {
        Generate(certificate data.CertificateIssuanceData) ([]byte, error)
    }

    const (
        DefaultLibPath = "/usr/bin/wkhtmltopdf"

        DefaultParamIssuer      = "auto_issuer_address"
        DefaultParamReceiver    = "auto_receiver_address"
        DefaultParamSignature    = "auto_issuer_signature"
        DefaultParamReceiverName = "auto_receiver_name"

        DefaultParamQR = "qr"
    )

    type CertificateParams struct {
        Template string
        Data      json.RawMessage

        Signature string
        Issuer     string
        Receiver   string
    }

    type generator struct {
        qrLink string
    }

    func New(libpath string, path string) Generator {
        wkhtmltopdf.SetPath(libpath)

        return &generator{
            qrLink: path,
        }
    }

    func (g *generator) Generate(certificate data.CertificateIssuanceData)
    ([]byte, error) {
        content, err := g.prepareContent(certificate)
        if err != nil {
            return nil, errors.Wrap(err, "failed to prepare certificate
content")
        }

        pdfg, err := wkhtmltopdf.NewPDFGenerator()
        if err != nil {
            return nil, errors.Wrap(err, "failed to create PDF generator")
        }

        pdfg.AddPage(wkhtmltopdf.NewPageReader(content))

        pdfg.PageHeight.Set(99)
        pdfg.PageWidth.Set(132)
        pdfg.MarginBottom.Set(0)
        pdfg.MarginTop.Set(0)
        pdfg.MarginLeft.Set(0)
        pdfg.MarginRight.Set(0)

        if err = pdfg.Create(); err != nil {
            return nil, errors.Wrap(err, "failed to create PDF")
        }
    }

```

```

    }

    return pdfg.Bytes(), nil
}

func (g *generator) prepareContent(content data.CertificateIssuanceData)
(io.Reader, error) {
    var params map[string]interface{}
    if err := json.Unmarshal(content.RawData, &params); err != nil {
        return nil, errors.Wrap(err, "failed to unmarshal data")
    }

    qr, err := qrcode.Encode(fmt.Sprintf("%s/%v", g.qrLink, content.Id),
qrcode.Medium, 250)
    if err != nil {
        return nil, errors.Wrap(err, "failed to generate QR-code")
    }

    params[DefaultParamQR] = base64.StdEncoding.EncodeToString(qr)
    params[DefaultParamIssuer] = content.IssuerAddr
    params[DefaultParamReceiver] = content.OwnerAddr
    params[DefaultParamSignature] = content.Signature
    params[DefaultParamReceiverName] = content.OwnerName

    tmpl, err := template.New("certificate").Parse(content.TemplateData)
    if err != nil {
        return nil, errors.Wrap(err, "failed to parse template")
    }

    var result bytes.Buffer
    if err = tmpl.Execute(&result, params); err != nil {
        return nil, errors.Wrap(err, "failed to execute template")
    }

    return &result, nil
}

```

Фрагменти коду пайплайнів

name: Push Docker image to DockerHub

on:

```

  workflow_call:
    inputs:
      service_name:
        type: string
        required: true
      service_path:
        type: string
        required: true
    secrets:
      docker_hub_username:
        required: true
      docker_hub_password:
        required: true

```

jobs:

```

  publish:
    name: Push Docker image to DockerHub
    runs-on: ubuntu-latest
    steps:
      -
        name: Check out the repo

```

```

    uses: actions/checkout@v3
  -
    name: Extract Docker metadata
    id: metadata
    uses: docker/metadata-action@v5
    with:
      images: slbmax/diploma-${{ inputs.service_name }}
  -
    name: Login to DockerHub
    uses: docker/login-action@v3
    with:
      username: ${ secrets.docker_hub_username }
      password: ${ secrets.docker_hub_password }
  -
    name: Build and push Docker image
    uses: docker/build-push-action@v5
    with:
      context: ../${{ inputs.service_path }}
      push: true
      tags: ${ steps.metadata.outputs.tags }
      labels: ${ steps.metadata.outputs.labels }

name: Publish Tracker Service Image

on:
  push:
    branches:
      - '**'
    paths:
      - 'backend/tracker-svc/**'

jobs:
  publish:
    uses: ../.github/workflows/publish-docker-image.yml
    with:
      service_name: tracker-svc
      service_path: backend/tracker-svc
    secrets:
      docker_hub_username: ${ secrets.DOCKER_HUB_USERNAME }
      docker_hub_password: ${ secrets.DOCKER_HUB_PASSWORD }

name: Run tests for contracts

on:
  push:
    branches:
      - '**'
    paths:
      - 'contracts/**'

jobs:
  test:
    runs-on: ubuntu-latest
    defaults:
      run:
        working-directory: ./contracts/certificates
    steps:
      -
        name: Checkout the repository
        uses: actions/checkout@v3
      -
        name: Setup node
        uses: actions/setup-node@v3

```

```

    with:
      node-version: "18.18.x"
      cache: npm
      cache-dependency-path: './contracts/certificates/package-
lock.json'
  -
    name: Install packages
    run: npm install
    shell: bash
  -
    name: Run tests
    run: npm run test
    shell: bash

```

Фрагменти коду модульних тестів для смарт-контрактів

```

const truffleAssert = require("truffle-assertions");
const Reverter = require("../helpers/reverter");
const Web3Utils = require("web3-utils");
require('@nomiclabs/hardhat-truffle5');
const { accounts } = require("../scripts/utils/utils");
const { assert } = require("chai");

const CourseFactory = artifacts.require("CourseFactory");
const CourseContract = artifacts.require("CourseContract");

let OWNER;
let USER1;
let ADMIN1;
let ADMIN2;
let courseFactory;
let courseContract;

const nonIssuerReason = "Caller is not the issuer"
const sha256HashVal = "123412341234123412341234123412341234";

before("setup", async() => {
  OWNER = await accounts(0);
  USER1 = await accounts(1);
  ADMIN1 = await accounts(2);
  ADMIN2 = await accounts(3);

  courseFactory = await CourseFactory.new();
  const tx = await courseFactory.deployCourseContract(["test course",
"test desc"], {
    from: OWNER,
  });

  courseContract = await
CourseContract.at(tx.receipt.logs[0].args.newCourseContractAddress);
})

describe("course creation", () => {
  it("should successfully create a course", async () => {
    assert.equal(await courseFactory.getCourseContractsCount(), 1);

    assert.equal("test course", await courseContract.name())
    assert.equal("test desc", await courseContract.description())
  })
})

describe("add issuers", () => {

```

```

    it("should revert on invoking by non-issuing account", async () => {
      await truffleAssert.reverts(courseContract.addIssuer(USER1, {from:
USER1}), nonIssuerReason);
    })

    it ("should revert for already existing issuer", async () => {
      await truffleAssert.reverts(courseContract.addIssuer(OWNER, {from:
OWNER}), "Issuer already exists");
    })

    it ("should successfully add issuer", async () => {
      await courseContract.addIssuer(USER1, {from: OWNER});
      assert.equal(true, await courseContract.issuers(USER1));
    })
  })

  describe("remove issuers", () => {
    it("should revert on invoking by non-issuing account", async () => {
      await truffleAssert.reverts(courseContract.deleteIssuer(USER1,
{from: ADMIN1}), nonIssuerReason);
    })

    it ("should revert for non-existing issuer", async () => {
      await truffleAssert.reverts(courseContract.deleteIssuer(ADMIN1,
{from: OWNER}), "Issuer does not exist");
    })

    it ("should revert for removing yourself", async () => {
      await truffleAssert.reverts(courseContract.deleteIssuer(OWNER,
{from: OWNER}), "Cannot remove yourself");
    })

    it ("should successfully remove issuer", async () => {
      await courseContract.deleteIssuer(USER1, {from: OWNER});
      assert.equal(false, await courseContract.issuers(USER1));
    })
  })

  describe("issue certificate", () => {
    it ("should revert for non-issuer", async () => {
      await
truffleAssert.reverts(courseContract.issueCertificates(ADMIN1, [
        {
          sha256Hash: sha256HashVal,
          holder: USER1,
          signature: "signature1"
        }
      ]), {from: ADMIN1}), nonIssuerReason);
    })

    it ("should revert for same checksum", async () => {
      await
truffleAssert.reverts(courseContract.issueCertificates(OWNER, [
        {
          sha256Hash: sha256HashVal,
          holder: USER1,
          signature: "signature1"
        },
        {
          sha256Hash: sha256HashVal,
          holder: USER1,
          signature: "signature1"
        }
      ]), {from: OWNER}), "Certificate already issued");
    })
  })

```

```

    it ("should successfully issue certificate", async () => {
      const tx = await courseContract.issueCertificates(USER1, [
        {
          sha256Hash: sha256HashVal,
          holder: USER1,
          signature: "signature1"
        }
      ], {from: OWNER});

      assert.equal(tx.receipt.logs[0].event, "CertificateIssued");
      assert.equal(tx.receipt.logs[0].args.issuer, OWNER);
      assert.equal(tx.receipt.logs[0].args.sha256Hash, sha256HashVal);

      assert.equal(await courseContract.certificateOwners(sha256HashVal,
USER1);
      const c = await courseContract.userCertificates(USER1,
sha256HashVal);

      assert.equal(c.signature, "signature1");
      assert.equal(c.issuer, USER1);
      assert.equal(c.publisher, OWNER);
      assert.equal(c.sha256Hash, sha256HashVal);
      assert.equal(c.holder, USER1);
      assert.equal(c.isRevoked, false);
      assert.equal(c.issuedBlockNumber.toString(), "13");
    })
  })

  describe("revoke certificate", () => {
    it ("should revert for non-issuer", async () => {
      await
truffleAssert.reverts(courseContract.revokeCertificate(sha256HashVal,
{from: ADMIN1}), nonIssuerReason);
    })

    it ("should revert for non-existing certificate", async () => {
      await truffleAssert.reverts(courseContract.revokeCertificate("non-
existing-checksum", {from: OWNER}), "Certificate not found");
    })

    it ("should successfully revoke certificate", async () => {
      const tx = await courseContract.revokeCertificate(sha256HashVal,
{from: OWNER});
      assert.equal(tx.receipt.logs[0].event, "CertificateRevoked");
      assert.equal(tx.receipt.logs[0].args.sha256Hash, sha256HashVal);

      const c = await courseContract.userCertificates(USER1,
sha256HashVal);
      assert.equal(c.isRevoked, true);
    })
  })
})

```

Фрагменти коду клієнтської частини

```

<script setup>
import {deleteLoading, notifyError, notifyLoading, notifySuccess} from
'../notifications.js';
import {ref} from 'vue';
import {useRoute} from 'vue-router';
import CheckMark from "@/components/CheckMark.vue";
import CrossMark from "@/components/CrossMark.vue";
import NoticeMark from "@/components/NoticeMark.vue";

```

```

import Arrow from "@/components/Arrow.vue";
import {useApi} from "@/api/main.js";

const VerifyStatusValid = "valid";
const VerifyStatusInvalid = "invalid";
const VerifyStatusUnknown = "unknown";

const VerificationTypeQr = "qr";
const VerificationTypeFile = "file";

const response = ref();

const updateVerificationData = (verificationType, value) => {
  const id = notifyLoading('Fetching data...');
  useApi().getVerificationData(verificationType, value).then((data) => {
    response.value = data;
    deleteLoading(id);
    notifySuccess('Data fetched');
  }).catch((error) => {
    deleteLoading(id);
    notifyError(error);
  })
}

const route = useRoute();
if (route.query.certificateId) {
  const certificateId = route.query.certificateId;
  updateVerificationData(VerificationTypeQr, certificateId);
}

function processFile(event) {
  const fileHash = getSha256HashBlobString(event.target.files[0]);
  fileHash.then((hash) => {
    updateVerificationData(VerificationTypeFile, hash)
  }).catch((error) => {
    notifyError(error)
  });
}

const getSha256HashBlobString = (fileBlob) => {
  return new Promise((resolve, reject) => {
    const reader = new FileReader();
    const b = new Blob([fileBlob]);
    reader.readAsArrayBuffer(b);

    reader.onloadend = function () {
      const arrayBuffer = reader.result;

      crypto.subtle.digest('SHA-256', arrayBuffer).then(hashBuffer => {
        const hashArray = Array.from(new Uint8Array(hashBuffer));
        const hashHex = hashArray.map(byte => byte.toString(16).padStart(2, '0')).join('');
        resolve(hashHex); // Resolve the promise with the hashHex string
      }).catch(error => {
        reject('Hashing failed: ' + error); // Reject the promise with the error
      });
    };

    reader.onerror = function () {
      reject('File reading failed: ' + reader.error); // Reject the promise if file reading fails
    };
  });
};

```

```

    });
  };

</script>

export const getHTMLFromTemplate = (template) => {
  const { inputs, background } = { ...template };
  const HTML = document.createElement('html');
  const body = document.createElement('body');
  const fontData = document.createElement('link');
  fontData.href =
'http://fonts.googleapis.com/css2?family=Montserrat:ital,wght@0,100..900;1,
100..900&display=swap';
  fontData.rel = 'stylesheet';
  const div = document.createElement('div');
  const img = document.createElement('img');
  img.src = 'data:image/png;base64,{{ .qr }}';
  const generatedParagraphs = inputs.map((input) => {
    const p = document.createElement('p');

    p.style.left = input.left + 'px';
    p.style.top = input.top + 'px';
    p.style.textAlign = input.align;
    p.style.fontSize = input.fontSize + 'px';
    p.style.color = input.color;
    if (input.class === 'input') {
      p.innerText = `{{ .${input.value.replace("dynamic_", '')} }}`;
    } else {
      p.innerText = input.value;
    }

    return p;
  });
  const styleElem = document.createElement('style');
  styleElem.textContent = `
    html {
      print-color-adjust: exact;
      -webkit-print-color-adjust: exact;
    }
    p {
      position: absolute;
      color: white;
      width: 1000px;
      text-shadow:
        black -1px -1px 2px,
        black 1px 1px 2px;
      margin: 0;
      white-space: nowrap;
    }
    body {
      margin: 0;
      font-family: "Montserrat", sans-serif;
    }
    div {
      height: 750px;
      min-width: 1000px;
      max-width: 1000px;
      margin: 0;
      background: ${background} ;
    }
    img {
      position: absolute;
      left: 25px;

```

```

        bottom: 25px;
        margin: 0;
    }
    `;
    body.append(...generatedParagraphs);
    body.append(img);
    body.append(styleElem);
    body.append(div);
    HTML.append(fontData);
    HTML.append(body);

    return HTML.outerHTML;
};

<template>
  <main class="certificate container">
    <Popup :open="addFieldPopupOpen">
      <div class="popup_header">
        <h3 class="title title--xs">Add text</h3>
        <button class="modal__close btn_close" @click="closeAddFieldPopup">
          <ModalClose/>
        </button>
      </div>
      <div class="popup_container">
        <input class="input_wrapper input" placeholder="Enter text" v-
model="fieldPopupValue"/>
        <Button :on-click="SaveFieldClick">Save field</Button>
      </div>
    </Popup>
    <Popup :open="addInputPopupOpen">
      <div class="popup_header">
        <h3 class="title title--xs">Add variable</h3>
        <button class="modal__close btn_close" @click="closeAddInputPopup">
          <ModalClose/>
        </button>
      </div>
      <div class="popup_container">
        <input class="input_wrapper input" placeholder="Variable name" v-
model="inputPopupValue"/>
        <Button :on-click="SaveInputClick">Save input</Button>
      </div>
    </Popup>
    <Popup :open="saveTemplatePopupOpen">
      <div class="popup_header">
        <h3 class="title title--xs">Choose template name</h3>
        <button class="modal__close btn_close"
@click="closeTemplatePopupOpen">
          <ModalClose/>
        </button>
      </div>
      <div class="popup_container">
        <input class="input_wrapper input" placeholder="Template name" v-
model="templateNamePopupValue">
        <Button :on-click="saveTemplateClick">Save template</Button>
      </div>
    </Popup>
    <div
      :key="certificateComponentKey"
      class="design-container"
      :style="{ background: `${background} center/1000px 750px no-repeat`
}"
    >
      <CertificateInputs

```

```

        :inputs="inputs"
        :rerender-certificate="rerenderCertificate"
    />
    <QRcode/>
</div>
<div class="side-bar">
    <Button :onclick="openAddFieldPopup">Add Text</Button>
    <Button :onclick="openAddInputPopup">Add Variable</Button>

    <p></p>
    <p>Set text alignment:</p>
    <div class="comp-alignment">
        <Button id="btn-left-alignment" class="active"
:onclick="setLeftAlignment">Left</Button>
        <Button id="btn-center-alignment"
:onclick="setCenterAlignment">Center</Button>
        <Button id="btn-right-alignment"
:onclick="setRightAlignment">Right</Button>
    </div>
    <p></p>
    <p></p>
    <div>
        <label class="btn btn--hover" for="color-pick-text">Change text
color
        <input
            type="color"
            id="color-pick-text"
            name="color-pick-text"
            @input="changeNextColor"
        />
    </label>
    </div>
    <p></p>
    <p :style="{ color: nextColor }">Current text color</p>
    <p></p>
    <BackgroundPicker
        :background-picker-active="backgroundPickerActive"
        :change-background-color="changeBackgroundColor"
        :change-background-image="changeBackgroundImage"
        :switch-background-picker-active="switchBackgroundPickerActive"
    />
    <p></p>
    <Button :onclick="openSaveTemplatePopup">Save template</Button>
</div>
</main>
</template>

```

Додаток 3
Копія презентації

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ
СІКОРСЬКОГО”



ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМП'ЮТЕРНИХ СИСТЕМ

Вебзастосунок для випуску цифрових сертифікатів на основі технології блокчейн

Виконав: Студент групи КП-02 Слободзян Максим Вікторович

Керівник: к.т.н., доцент кафедри ПЗКС Шкурат Оксана Сергіївна

Київ – 2024

АКТУАЛЬНІСТЬ



За статистикою, близько 70% студентів надають перевагу онлайн навчанню над традиційною системою освіти



Застарілі практики традиційних методів видачі та перевірки паперових документів

Відсутність гармонійного поєднання зручності застосунку та рівня безпеки і прозорості системи



ПОСТАНОВКА ЗАДАЧІ



Мета проєкту: забезпечення зручного та водночас надійного процесу випуску цифрових сертифікатів та їх верифікації.

Завдання:

Аналіз:

1. Дослідження особливостей процесу та існуючих програмних рішень
2. Визначення основних вимог до системи

Розроблення:

1. Вибір та обґрунтування оптимальних засобів реалізації застосунку
2. Розроблення модулів та компонент системи

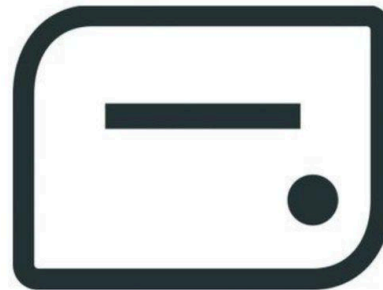
Результати:

1. Тестування розробленого застосунку
2. Порівняння з аналогами
3. Опис можливих напрямів подальшого вдосконалення застосунку

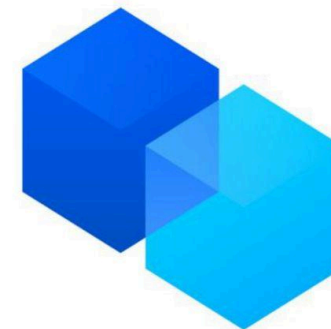
ІСНУЮЧІ РІШЕННЯ



Certifier



Accredible



DoxyChain

ВИМОГИ ДО ВЕБЗАСТОСУНКУ



1. Зручний користувацький інтерфейс.
2. Безпека та цілісність даних.
3. Можливість індивідуального проектування шаблону сертифікатів.
4. Можливість імпорту даних та випуск декількох сертифікатів одночасно.
5. Можливість додавання власних динамічних атрибутів документа.
6. Можливість доступу до випуску сертифікатів групи учасників.
7. Швидкий та зручний процес верифікації даних.

ТЕХНОЛОГІЇ РОЗРОБЛЕННЯ



Блокчейн-розробка



MetaMask

Мови програмування



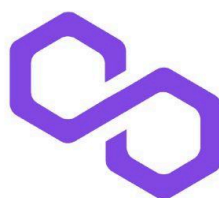
Solidity



JavaScript



Go



Polygon

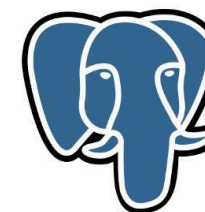
Фреймворки



HardHat

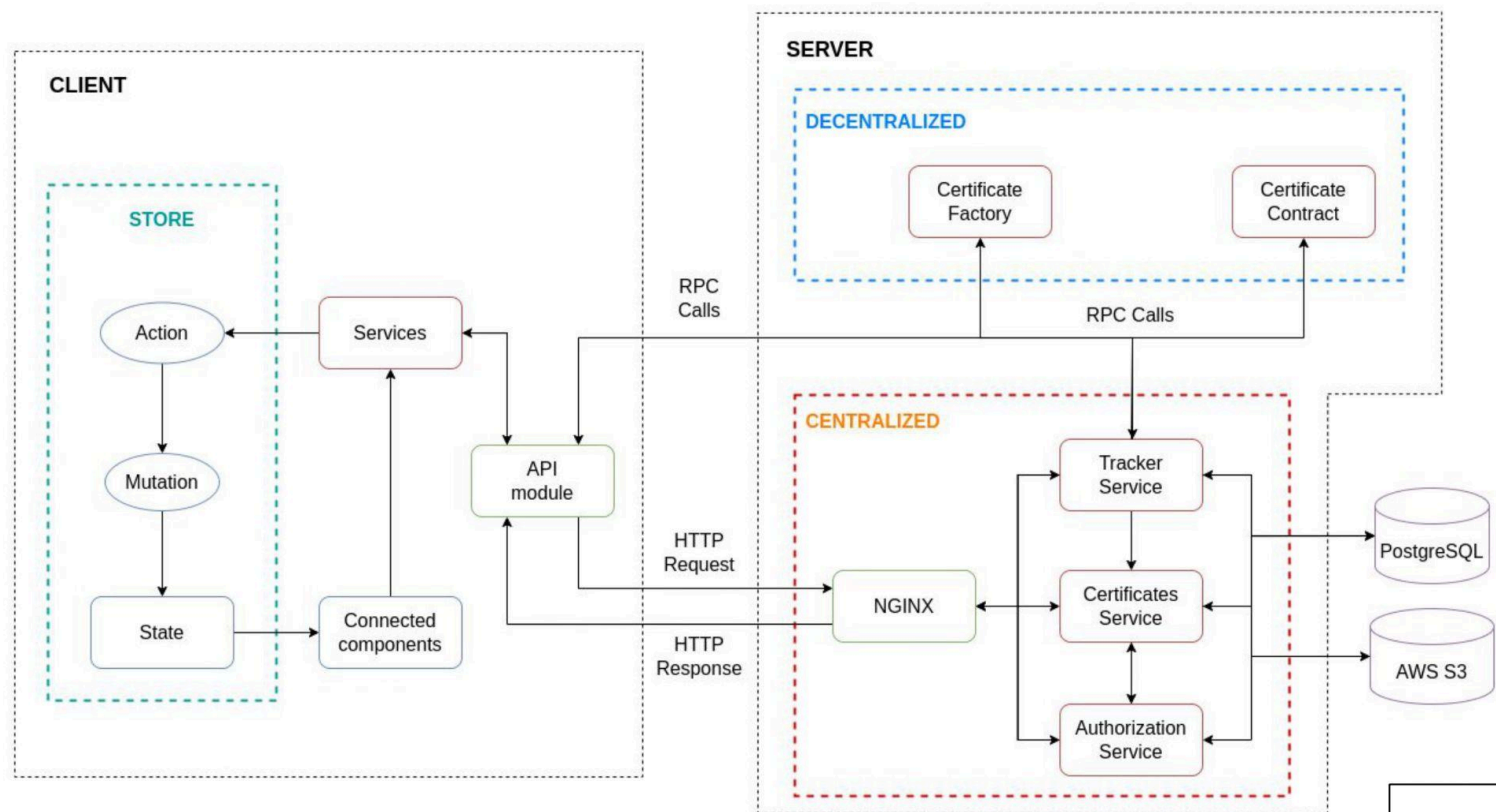


Vue.js

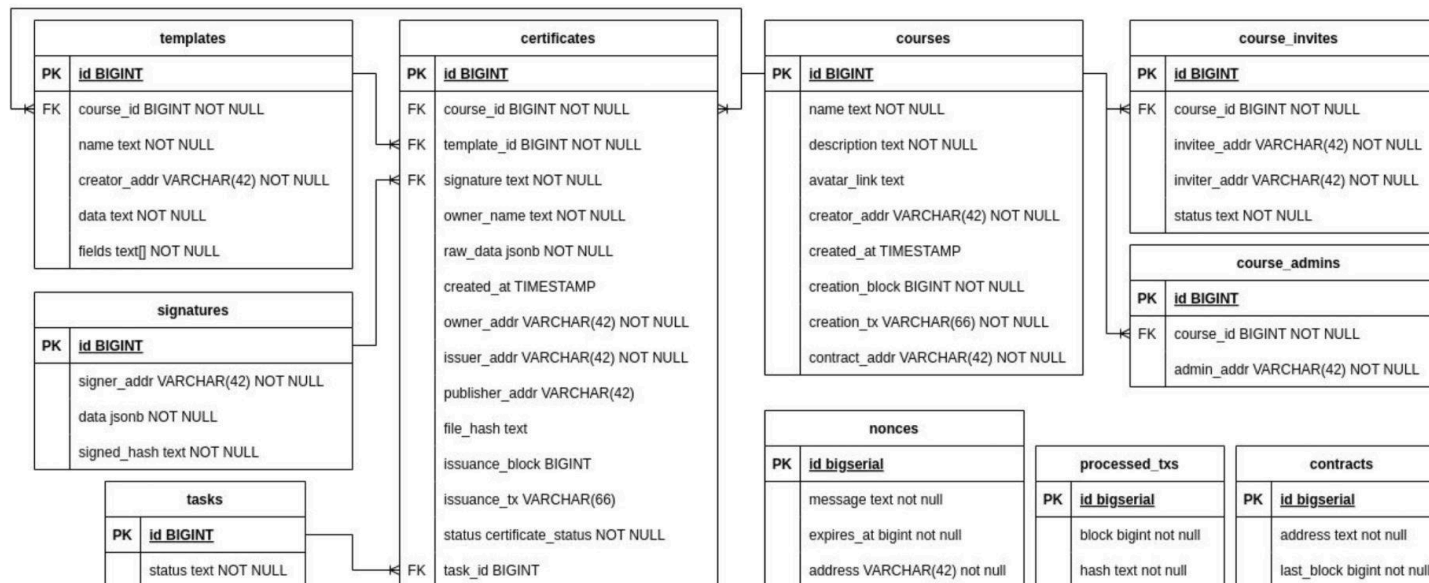


PostgreSQL

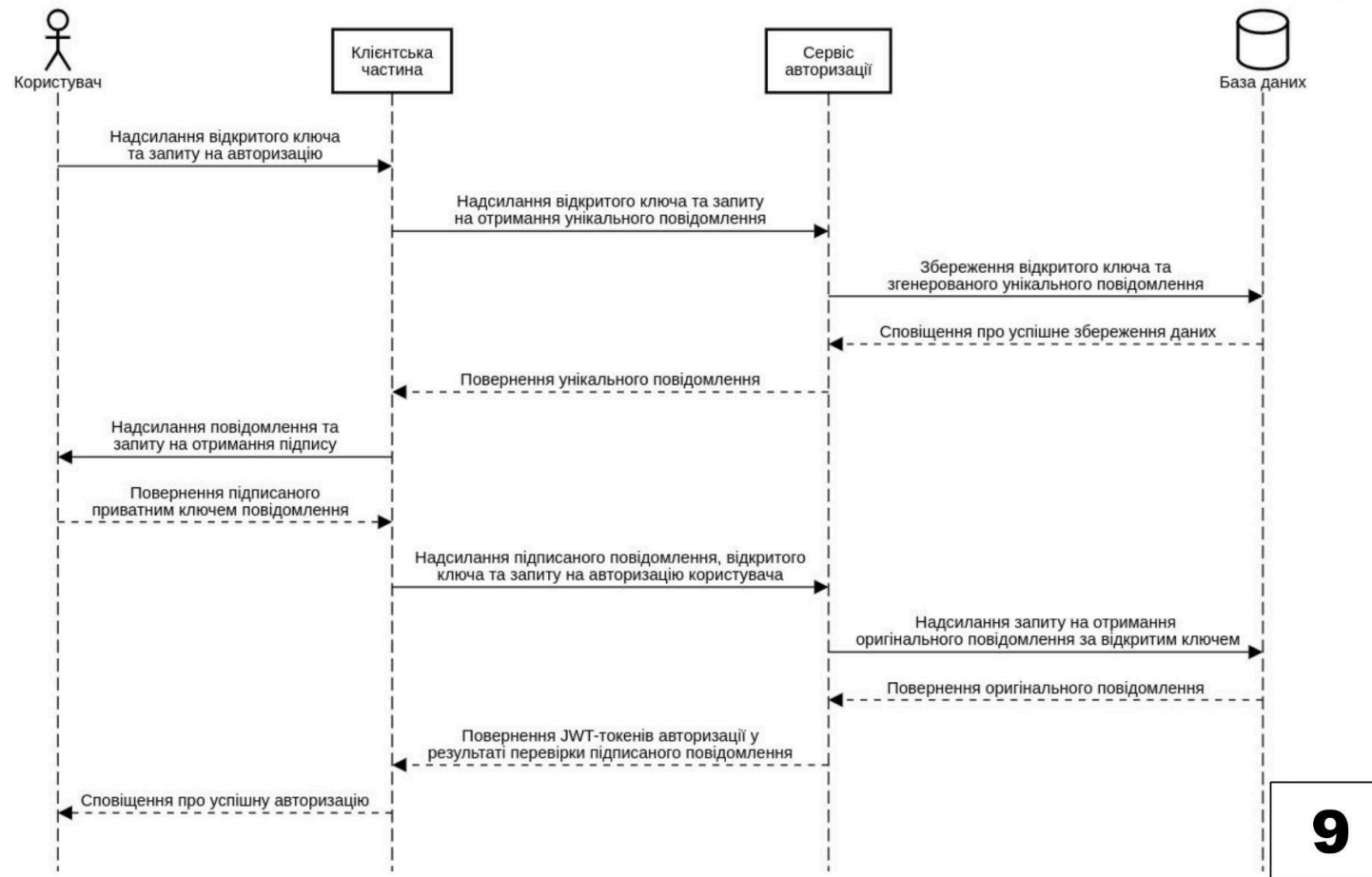
АРХІТЕКТУРА СИСТЕМИ



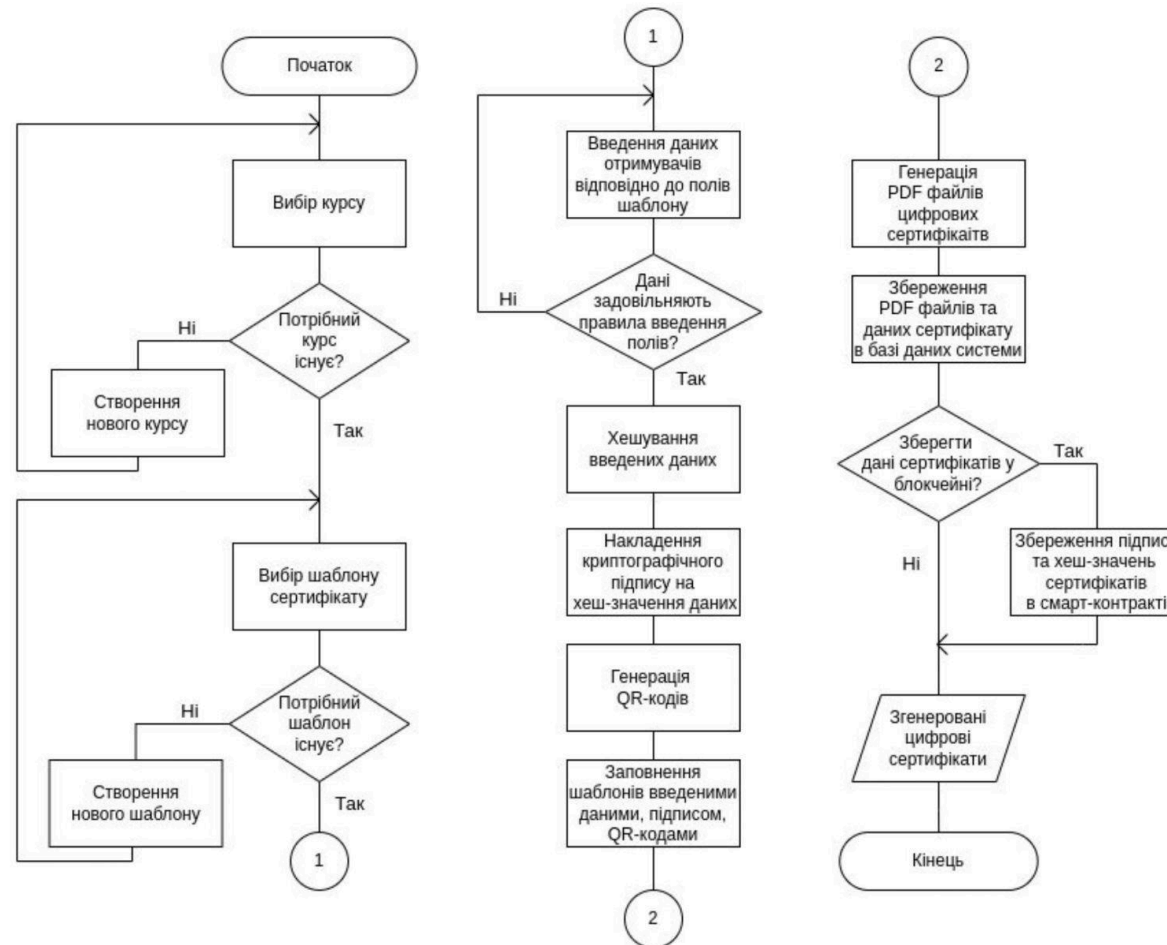
СТРУКТУРА БАЗИ ДАНИХ



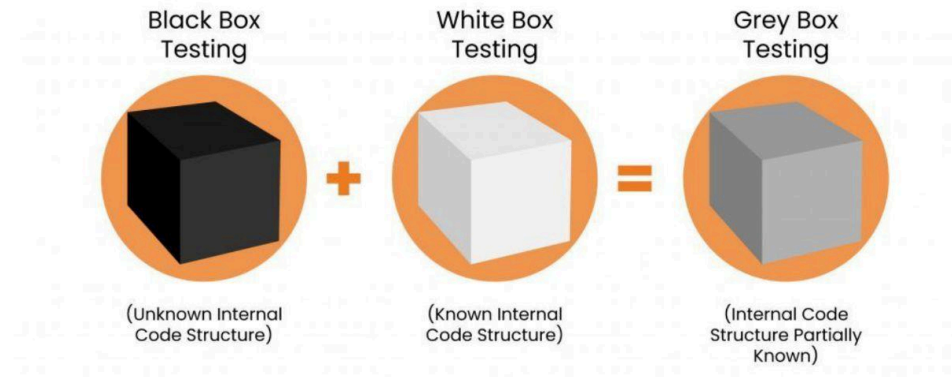
АЛГОРИТМ АВТОРИЗАЦІЇ НА ОСНОВІ РКС



АЛГОРИТМ ВИПУСКУ ЦИФРОВИХ СЕРТИФІКАТІВ

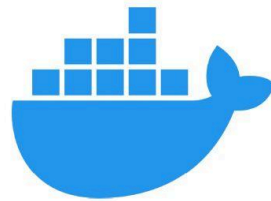


ТЕСТУВАННЯ ВЕБЗАСТОСУНКУ

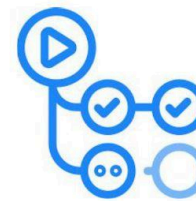


File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	100	100	100	100	
CourseContract.sol	100	100	100	100	
CourseFactory.sol	100	100	100	100	
contracts/interfaces/	100	100	100	100	
ICourseContract.sol	100	100	100	100	
ICourseFactory.sol	100	100	100	100	
All files	100	100	100	100	

БЕЗПЕРЕРВНА РОЗРОБКА



Docker



GitHub Actions

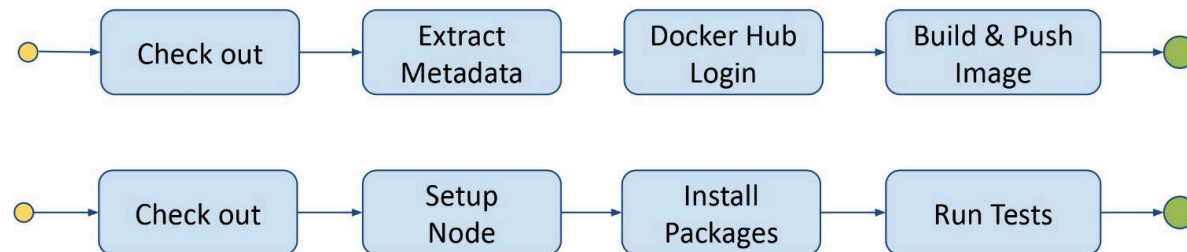


Схема завдань CI/CD застосунку

ГОЛОВНА СТОРІНКА



DiCertifier Certificates Verify Certificate My Profile [Log out](#)

Revolutionize learning validation: effortlessly issue and verify Digital Certificates with DiCertifier

Leverage advanced security for **authenticity**, ensure **instant accessibility** for learners, and maintain **immutable records** for **easy and reliable** verification. Empower your institution with **seamless, credible, and secure** credentialing solutions.

All issued certificates

Search

All

ГОЛОВНА СТОРІНКА (ПРОДОВЖЕННЯ)



All issued certificates

... Search

All
Created
Published
Revoked



Algorithms and Data Structures - Part 1

Receiver: Max Slobodzian

View certificate info



PERCEPTRON - the simplest neural network

Receiver: Viktoria Korchyk

View certificate info



Amazon Web Services for Developers

Receiver: Max Slobodzian

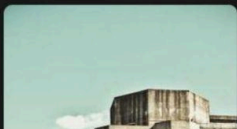
View certificate info



Software Development Processes and...

Receiver: Vasyl Koval

View certificate info









ДИЗАЙН ШАБЛОНУ СЕРТИФІКАТУ



DiCertifier Certificates Verify Certificate My Profile Log out

dynamic_date

Certificate of Completion

This is to certify that

auto_receiver_name

For succesfully completing 20 hours of

Vue basics for beginners

QR-code

Issuer address:
auto_issuer_address

Receiver address:
auto_receiver_address

Issuer signature hash:
auto_issuer_signature



Add Text

Add Variable

Set text alignment:

LeftCenterRight

Change text color

Current text color

Background

Save template

ЗАПОВНЕННЯ ФОРМИ ДЛЯ ВИПУСКУ СЕРТИФІКАТІВ



Certificates Form

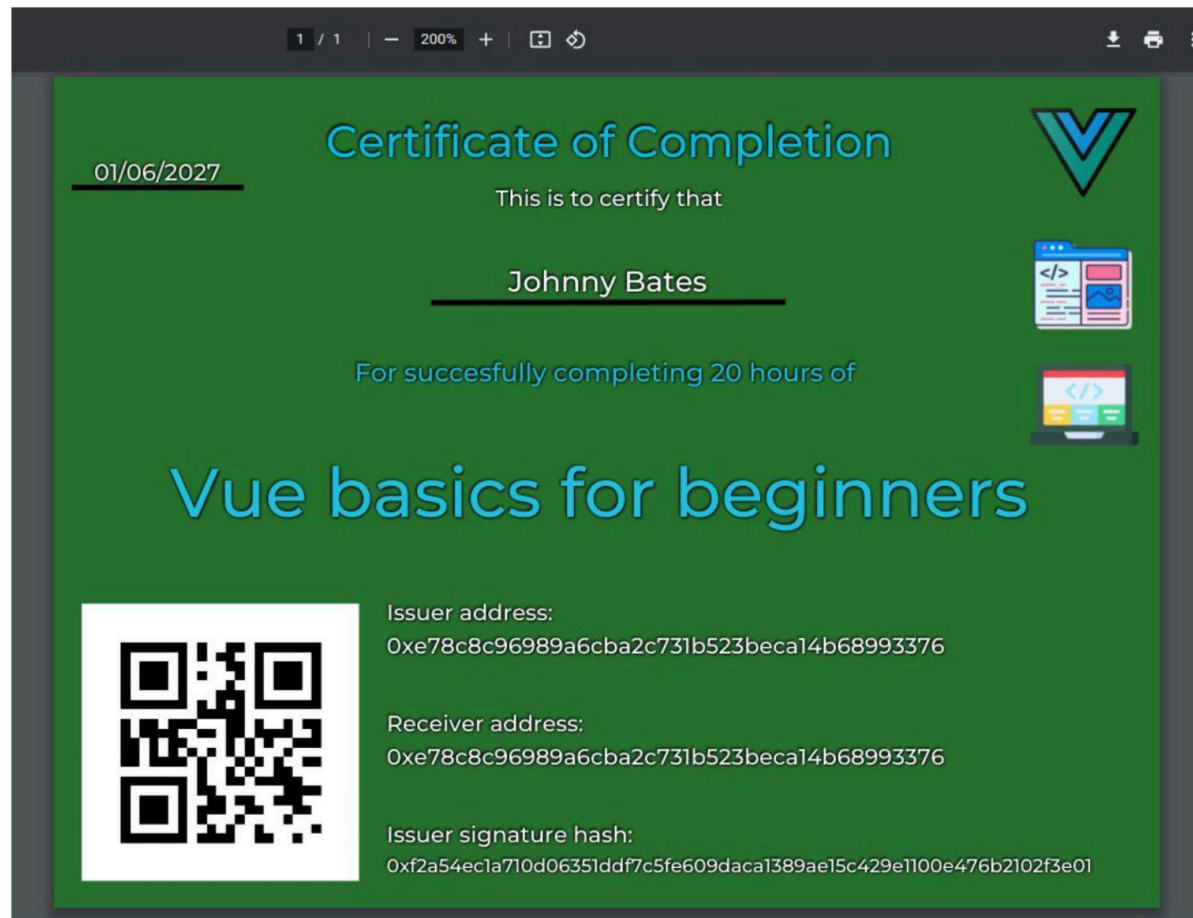
Create credentials via file

#	Receiver address	Receiver name	date	my_custom_atr
☒ 1	0xe78c8c96989a6cba2c731	Max Slobodzian	27/02/2024	Some custom data
☒ 2	0xe78c8c96989a6cba2c731	Maximo Slobodziano	26/05/2024	Some custom data
☒ 3	0xe78c8c96989a6cba2c731	Mogger de rizzlo	11/09/2001	Some custom data

Add row

Issue certificates

ПРИКЛАД ЗГЕНЕРОВАНОГО ЦИФРОВОГО СЕРТИФІКАТУ



ВЕРИФІКАЦІЯ СЕРТИФІКАТУ



Certificate Verification

Upload Certificate 

RECEIVER

0xe78c8c96989a6cba2c731b523beca14b68993376

— Certificate receiver matches the blockchain data

↪ ! Was not published on blockchain

ISSUER

0xe78c8c96989a6cba2c731b523beca14b68993376

— Certificate issuer matches the blockchain data

↪ ! Was not published on blockchain

SIGNATURE

0x75cafc989cc05de98440b5497cbe488182845910f50701e1cb05d2868c57fad1

✓ Signature was created by certificate issuer

✓ Signature contains data about this certificate

— Signature matches the blockchain data

↪ ! Was not published on blockchain

CERTIFICATE STATUS

Created

— Certificate status matches the blockchain data

↪ ! Was not published on blockchain

PUBLISHER

— Certificate publisher matches the blockchain data

↪ ! Was not published on blockchain

FILE HASH

26d645680970759dffa155724f0a9780545a3e5e1fdcf03c857f713b5916d448

— File checksum matches the blockchain data

↪ ! Was not published on blockchain

✓ File checksum matches the issued certificate

ПОРІВНЯННЯ З АНАЛОГАМИ



Критерій	Certifier	DoxyChain	Accredible	Розроблене ПЗ
Зручність користувацького інтерфейсу	+	+	+/-	+
Можливість проєктування дизайну сертифікатів	+	-	+	+
Можливість імпорту даних та випуску декількох сертифікатів одночасно	+	-	+	+
Можливість доступу до випуску сертифікатів групи учасників	-	+	-	+
Можливість додавання власних динамічних атрибутів документа	+	-	-	+
Безпека та цілісність даних	+/-	+	+/-	+
Швидкий та зручний процес верифікації даних	-	+	+	+

НАПРЯМКИ РОЗВИТКУ ПРОЄКТУ



Автоматизована система розсилання сертифікатів



Розширення функціональних можливостей для дизайну шаблонів сертифіката



Інтеграція з освітніми платформами



Дослідження та впровадження технологій для підвищення рівня безпеки конфіденційних даних

ВИСНОВКИ



1. Виконано аналіз існуючих програмних рішень.
2. Визначено та обґрунтовано основні вимоги до системи.
3. Спроектовано архітектуру системи та структуру бази даних.
4. Реалізовано функціональні можливості вебзастосунку для випуску цифрових сертифікатів.
5. Проведено тестування вебзастосунку та його порівняння з аналогами за встановленими вимогами.
6. Впроваджено технології та засоби безперервного розроблення.
7. Досліджено шляхи подальшого вдосконалення застосунку.



Дякую за увагу!

Факультет прикладної математики
Кафедра програмного забезпечення комп'ютерних систем

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Євгенія СУЛЕМА

“ ____ ” _____ 2023 р.

ВЕБЗАСТОСУНОК ДЛЯ ВИПУСКУ ЦИФРОВИХ СЕРТИФІКАТІВ
НА ОСНОВІ ТЕХНОЛОГІЇ БЛОКЧЕЙН

Програма та методика тестування

ДП.045440-04-51

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Оксана ШКУРАТ

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Максим СЛОБОДЗЯН

ЗМІСТ

1. Об'єкт випробувань.....	3
2. Мета тестування.....	3
3. Методи тестування.....	3
4. Засоби та порядок тестування.....	4

1. ОБ'ЄКТ ВИПРОБУВАНЬ

Вебзастосунок для випуску цифрових сертифікатів на основі технології блокчейн, розроблений з використанням мов програмування Go, Solidity та JavaScript, що дозволяє зручно та швидко створювати надійні цифрові документи з можливістю їх прозорої верифікації.

2. МЕТА ТЕСТУВАННЯ

Метою тестування вебзастосунку для випуску цифрових сертифікатів на основі технології блокчейн є перевірка:

1. Функціональної працездатності вебзастосунку.
2. Заборони неавторизованого доступу.
3. Відповідності функціональності вебзастосунку вимогам технічного завдання.
4. Функціональної працездатності серверної частини.
5. Надійності та захищеності даних платформи та випущених цифрових сертифікатів.
6. Функціональної працездатності набору смарт-контрактів.
7. Зручності роботи з розробленим застосунком.
8. Продуктивності та швидкодії системи.

3. МЕТОДИ ТЕСТУВАННЯ

Тестування вебзастосунку для випуску цифрових сертифікатів на основі технології блокчейн виконується методом Gray Box Testing. Окрім безпосередньої перевірки програмного продукту на відповідність функціональним вимогам, також виконується тестування коду різних частин застосунку. Тестування вебзастосунку відбувається на рівні модулів системи.

Використовуються наступні методи:

1. Модульне тестування.
2. Тестування продуктивності вебзастосунку.
3. Ручне тестування інтерфейсу.
4. Аналіз граничних значень.

4. ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Тестування виконується засобами інструментарію HardNat.

Працездатність вебзастосунку для випуску цифрових сертифікатів на основі технології блокчейн перевіряється шляхом:

1. Динамічного ручного тестування – введенням граничних та недопустимих значень в поля, які можна редагувати.
2. Тестування зручності використання.
3. Тестування роботи вебзастосунку у різних браузерах: Google Chrome, Mozilla Firefox, Safari.
4. Ручного тестування на відповідність функціональним вимогам.
5. Тестування продуктивності вебзастосунку.
6. Тестування коректної взаємодії різних компонент системи.

Факультет прикладної математики
Кафедра програмного забезпечення комп'ютерних систем

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Євгенія СУЛЕМА

“ ____ ” _____ 2024 р.

ВЕБЗАСТОСУНОК ДЛЯ ВИПУСКУ ЦИФРОВИХ СЕРТИФІКАТІВ
НА ОСНОВІ ТЕХНОЛОГІЇ БЛОКЧЕЙН

Керівництво користувача

ДП.045440-05-34

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Оксана ШКУРАТ

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Максим СЛОБОДЗЯН

ЗМІСТ

1. ОПИС СТРУКТУРИ ВЕБЗАСТОСУНКУ.....	3
2. ОПИС ВМІСТУ ВЕБСТОРИНОК.....	4
2.1. Головна сторінка.....	4
2.2. Профіль користувача.....	6
2.3. Сторінка курсу.....	10
2.4. Сторінка дизайну шаблону сертифікату.....	11
2.5. Сторінка випуску сертифікатів.....	12
2.6. Сторінка верифікації сертифікатів.....	15

1. ОПИС СТРУКТУРИ ВЕБЗАСТОСУНКУ

Вебзастосунок для випуску цифрових сертифікатів на основі технології блокчейн надає простий та зручний у використанні інтерфейс взаємодії користувача. Він дозволяє легко та швидко переглядати, випускати та верифікувати цифрові сертифікати. Ергономічність інтерфейсу забезпечує інтуїтивне розміщення компонент та елементів управління, що спрощує навігацію та зменшує час ознайомлення користувачів з системою.

Застосунок розроблено в темній темі, що зменшує навантаження на очі при тривалому використанні та забезпечує комфортну роботу в умовах низького освітлення. Головна мова інтерфейсу – англійська, що робить його доступним для широкого кола користувачів по всьому світу.

Графічний інтерфейс користувача вебзастосунку складається з набору вебсторінок. Кожна вебсторінка розроблена для виконання певних процесів, таких як перегляд сертифікатів, дизайн шаблонів, заповнення даних, випуск документу тощо. Навігація між сторінками може здійснюватися як з використанням навігаційних можливостей браузера, так і за допомогою верхньої панелі меню застосунку.

Основними розробленими сторінками вебзастосунку для випуску цифрових сертифікатів є:

- головна сторінка;
- профіль користувача;
- сторінка курсу;
- сторінка дизайну шаблону сертифікату;
- сторінка випуску сертифікатів;
- сторінка верифікації сертифікатів.

У наступній частині наведено детальний огляд розроблених компонентів інтерфейсу користувача з прикладами їх використання.

2. ОПИС ВМІСТУ ВЕБСТОРИНОК

2.1. Головна сторінка

Першою сторінкою, на яку потрапляє гість/користувач вебзастосунку для випуску цифрових сертифікатів, є головна сторінка платформи. На головній сторінці знаходиться загальна інформація про можливості системи. Вона дозволяє новим користувачам швидко та просто ознайомитися з основними функціями та перевагами платформи.

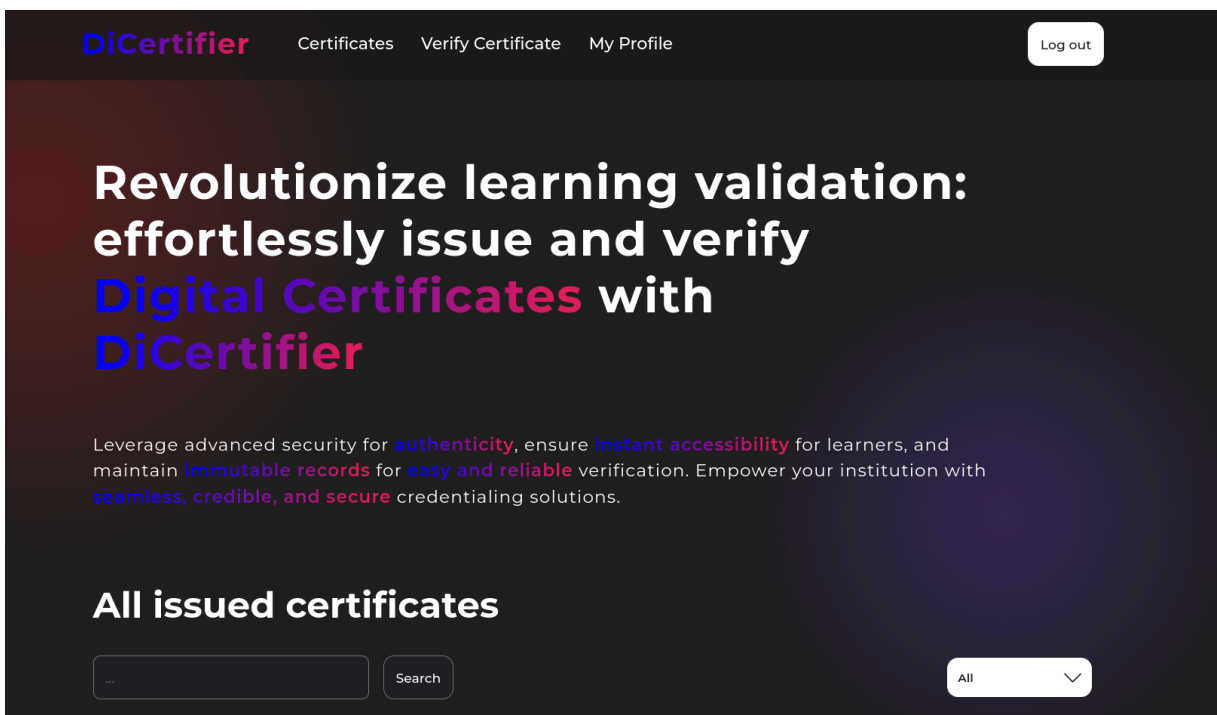


Рис. 1. Головна сторінка вебзастосунку

Зверху над основною інформацією про вебзастосунок знаходиться навігаційна панель керування користувача. Вона містить інтуїтивно зрозумілі посилання на інші сторінки вебзастосунку, що будуть детально розглянуті в процесі їх опису. Також присутня кнопка для авторизації користувача у системі.

Знизу під основною інформацією знаходиться блок даних про усі цифрові сертифікати, що були створені за допомогою розробленого

застосунку. Блок даних випущених сертифікатів являє собою набір карток сертифікатів, а також графічні елементи їх пошуку, фільтрації та пагінації. Вигляд блоку зображено на рис. 2.

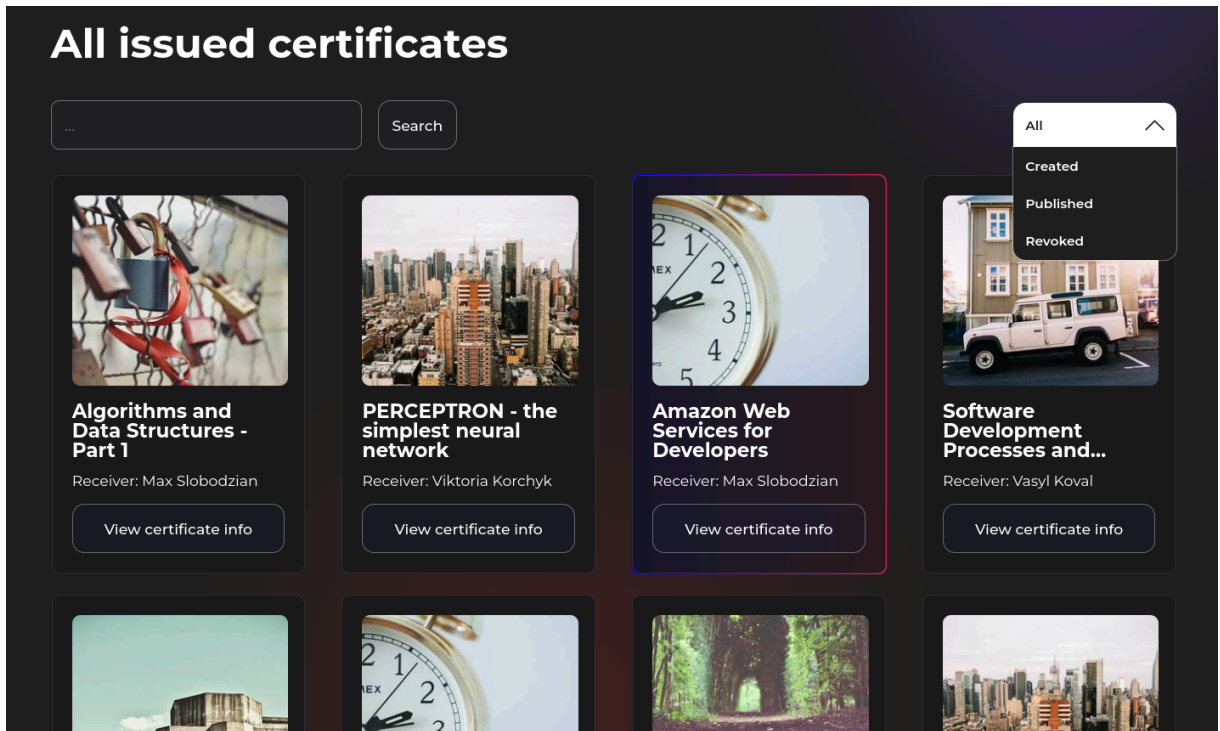


Рис. 2. Блок даних про створені сертифікати

Картка сертифікату складається з короткої інформації про курс та отримувача. За потреби можна натиснути кнопку перегляду даних про сертифікат, яка відкриває модальне вікно з детальною інформацією про документ. Воно містить інформаційні блоки даних про документ та курс, у якому було випущено сертифікат.

Якщо відповідний документ було опубліковано у мережі блокчейну, буде присутній відповідний блок інформації про дані транзакції, її автора та отримувача.

Для авторизованих користувачів та авторів сертифікату є доступною кнопка завантаження файлу згенерованого цифрового сертифікату.

Модальне вікно з інформацією про сертифікат зображено на рис. 3.

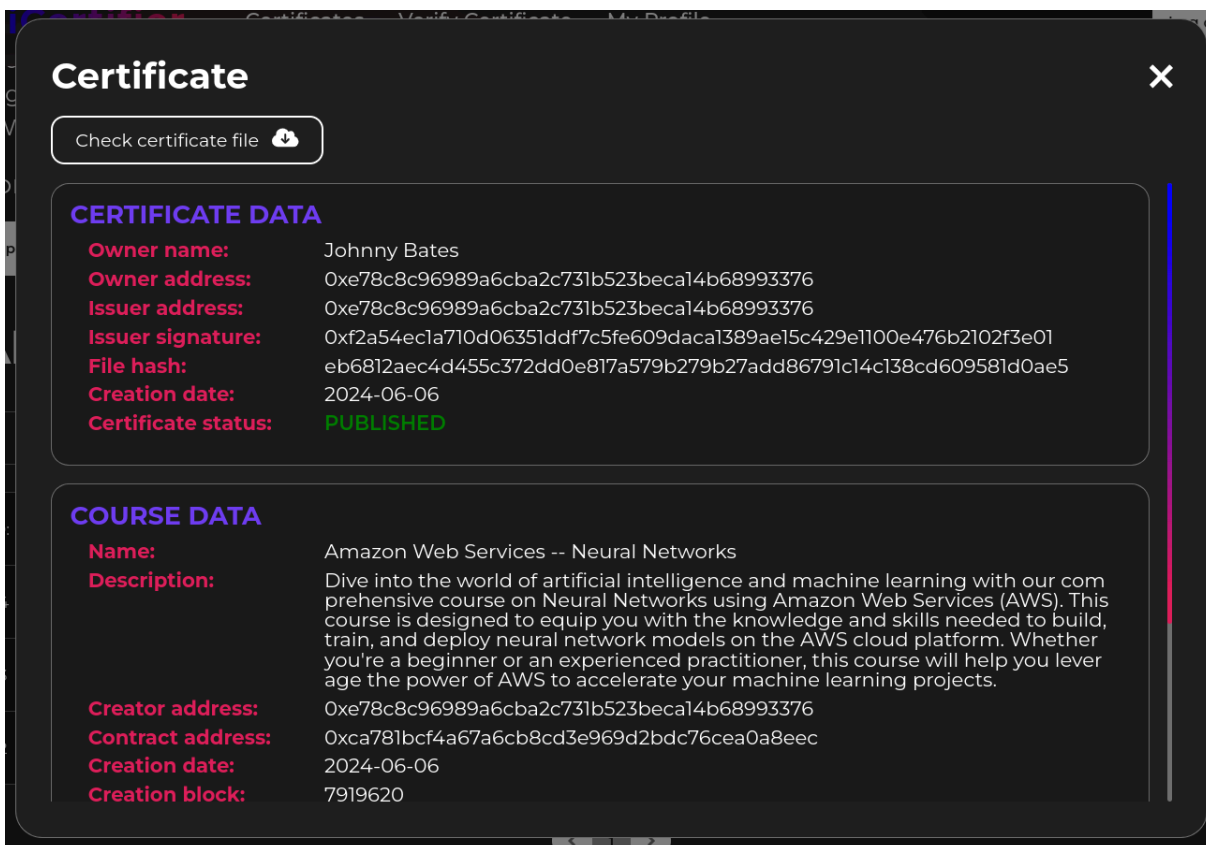


Рис. 3. Модальне вікно сертифікату

2.2. Профіль користувача

До сторінки профілю користувача можна перейти, скориставшись верхньою навігаційною панеллю. Обов'язковою умовою переходу є авторизація користувача у системі.

Особливістю розробленого вебзастосунку є нестандартна процедура авторизації користувачів. Їм не потрібно реєструватись у системі, щоб мати змогу повноцінно користуватись усіма можливостями. Натомість у застосунку присутня інтеграція з програмним гаманцем MetaMask, в якому зберігається унікальний закритий ключ користувача. Застосунок не взаємодіє з ним напряму і використовує методи цифрового підпису повідомлень з відкритим ключем, щоб авторизувати користувача. Гаманець MetaMask надає користувачеві простий інтерфейс для швидкого підписання авторизаційного повідомлення, який зображено на рис. 4, після

чого система може перевірити цифровий підпис та надати токени авторизації користувачу.

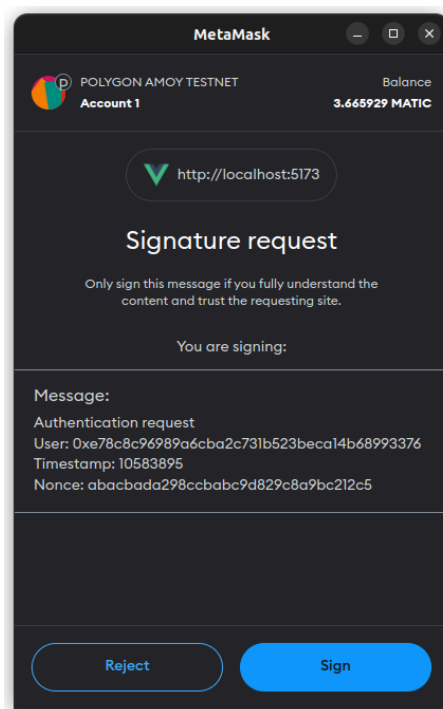


Рис. 4. Вікно підпису повідомлення MetaMask

У системі присутні оповіщення про відповідні дії користувача та можливі помилки. Наприклад, коли користувач підписує авторизаційне повідомлення, система авторизує користувача та оповіщує, що процес авторизації був успішний. Інакше з'являється сповіщення про невдачу та можливі причини.

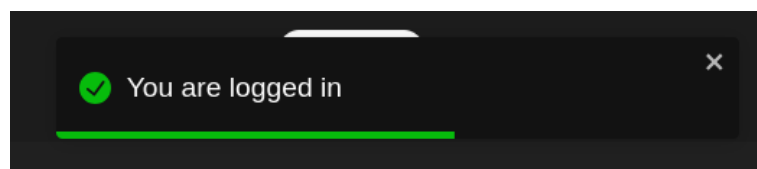


Рис. 5. Сповіщення про успішну дію

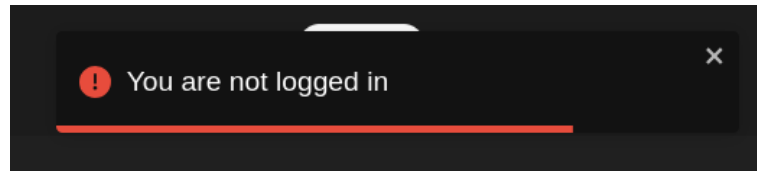


Рис. 6. Сповідання про помилку

Після успішної авторизації користувач може перейти на сторінку профілю, що зображена на рис. 7.

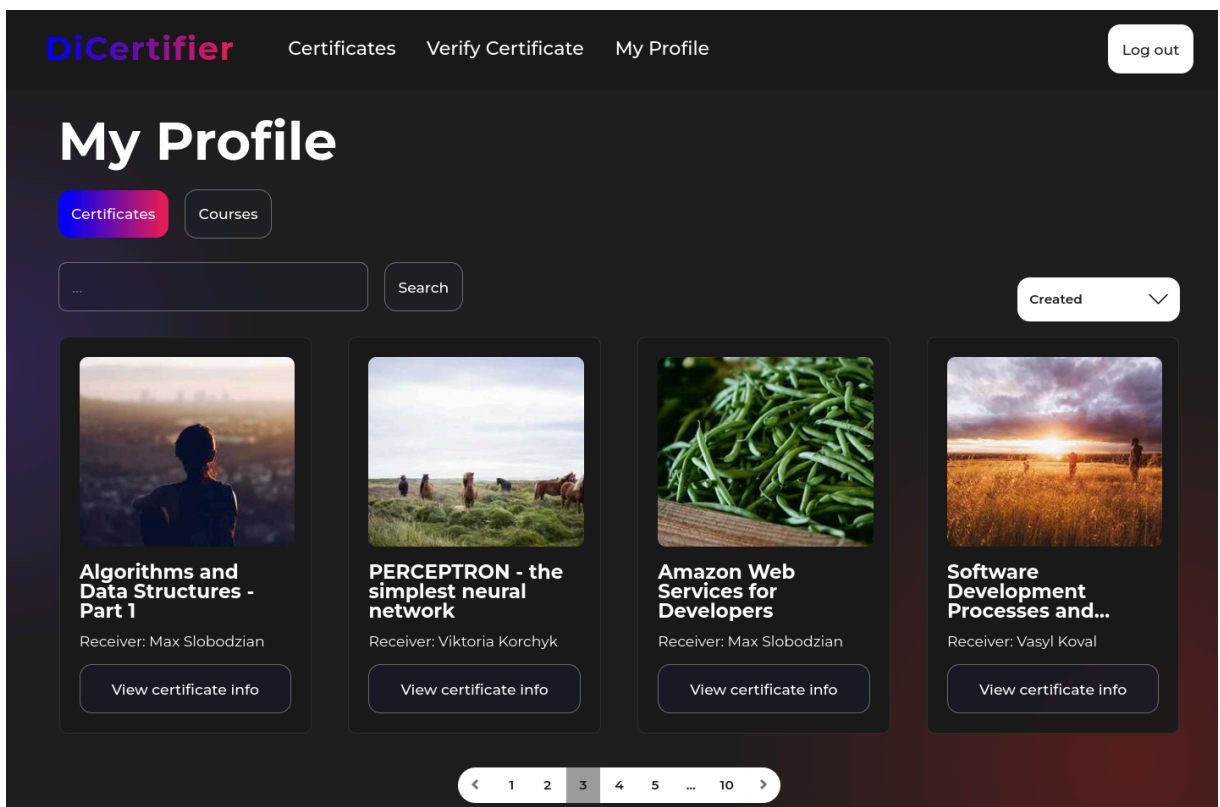


Рис. 7. Сторінка профілю користувача

Сторінка профілю користувача складається з двох основних компонент, між якими можна переключатись – блок даних сертифікатів та курсів.

Блок даних сертифікатів є схожим до блоку, що розташований на головній сторінці, за виключенням того, що в профілі відображаються лише документи, власником яких є користувач.

При переключенні компоненти сторінки користувачу відображається список курсів платформи. Вигляд сторінки курсів користувача зображено на рис. 8.

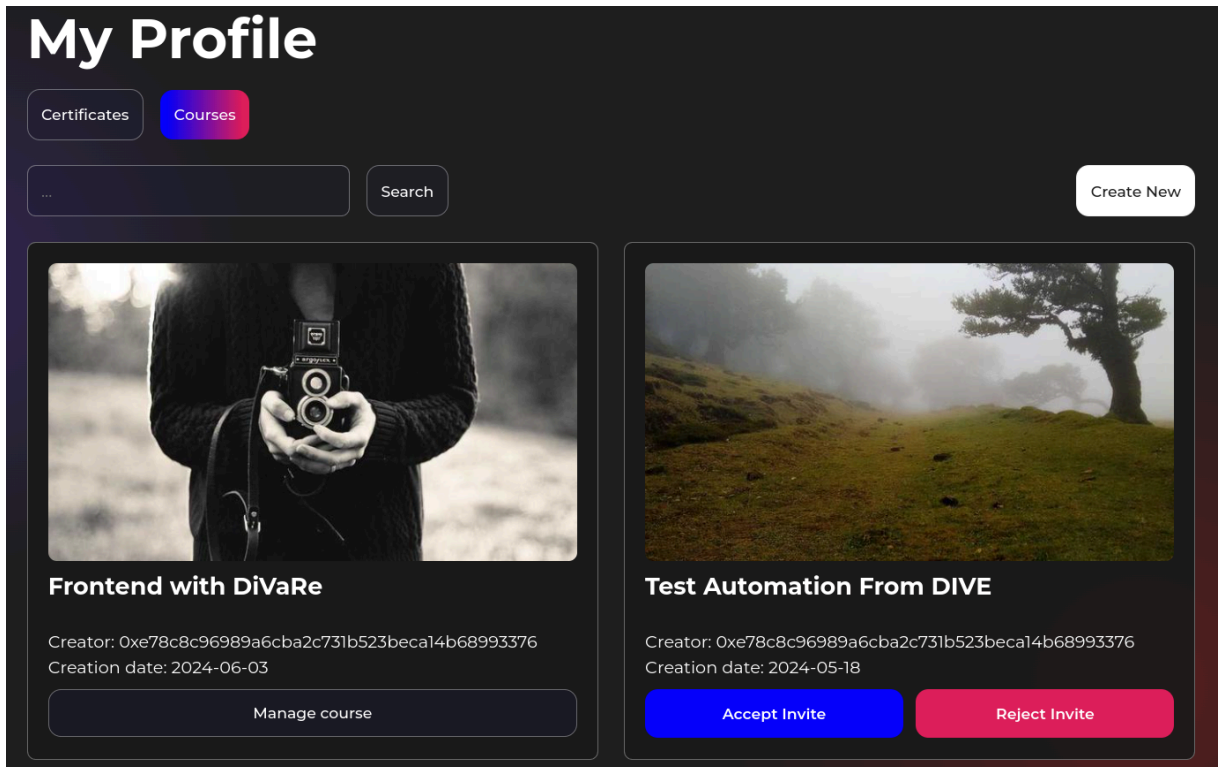


Рис. 8. Курси користувача

Курси, як і сертифікати, представлені відповідними картками. Серед курсів, що відображаються користувачу, є лише ті, в яких користувач є адміністратором або запрошений ним стати. Натискаючи на відповідні кнопки, користувач може прийняти чи відхилити запрошення.

Також на сторінці курсів є кнопка створення нового курсу, яка відкриває вікно введення даних про новий курс. Після успішного введення засобами гаманця MetaMask формується відповідна транзакція для взаємодії з блокчейном для створення смарт-контракту нового курсу. Користувач має змогу переглянути деталі транзакції та дати згоду на її підписання, після чого вона надсилається у мережу та створюється новий курс.

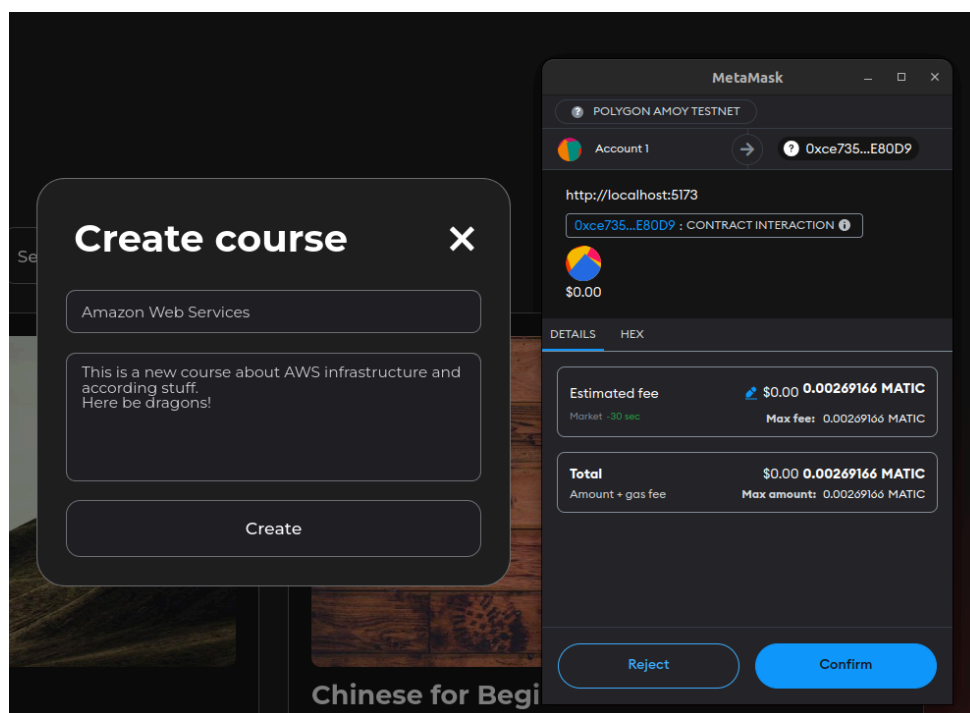


Рис. 9. Приклад запиту на підпис транзакції

Для карток курсів, де користувач є дійсним адміністратором, відображається кнопка управління курсом, яка перенаправляє користувача на сторінку курсу.

2.3. Сторінка курсу

На сторінці курсу користувач може переглянути загальну інформацію, яка включає в себе назву, опис, відповідну адресу смарт-контракту. Наявна панель управління для завантаження обкладинки курсу, керуванням списком адміністраторів, що можуть випускати сертифікати у рамках цього курсу. Також присутні кнопки розроблення дизайну шаблону та випуску сертифікатів.

Присутня таблиця сертифікатів, що були випущені у рамках цього курсу, аналогічно підтримується пошук та пагінація сутностей. В залежності від статусу сертифікату, доступні різноманітні дії, такі як перегляд детальної інформації, публікація сертифікату на блокчейні чи його відкриття.

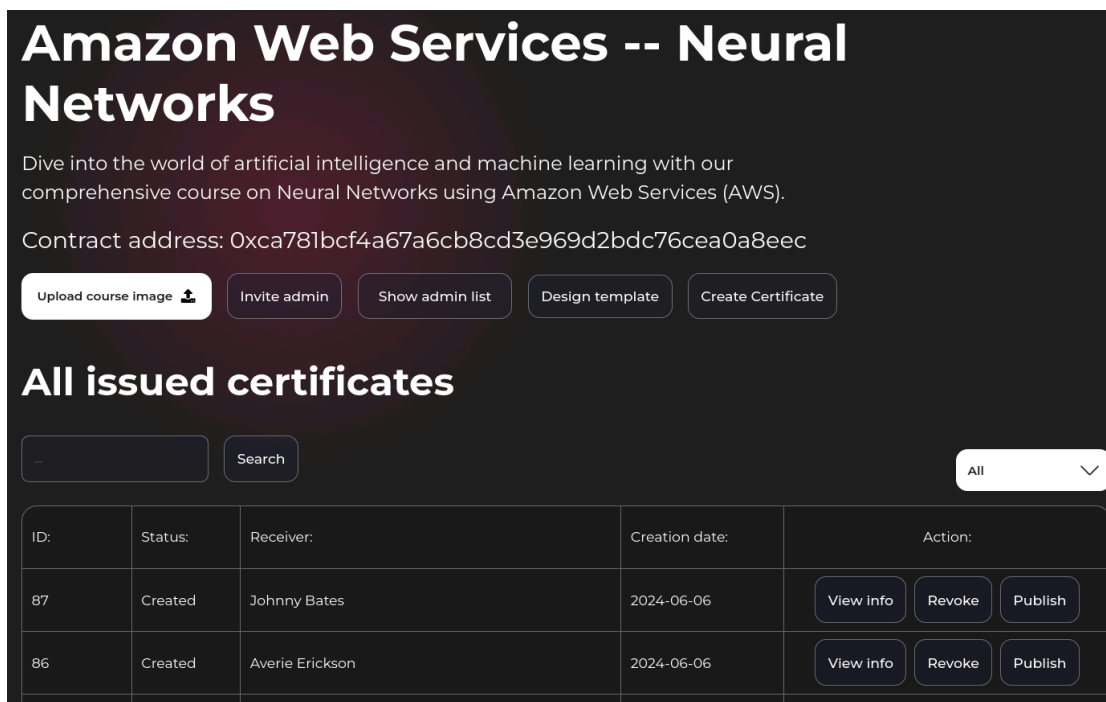


Рис. 10. Сторінка курсу

2.4. Сторінка дизайну шаблону сертифікату

При взаємодії з кнопкою дизайну шаблону сертифікату користувача направляє на відповідну сторінку, що зображена на рис. 11.

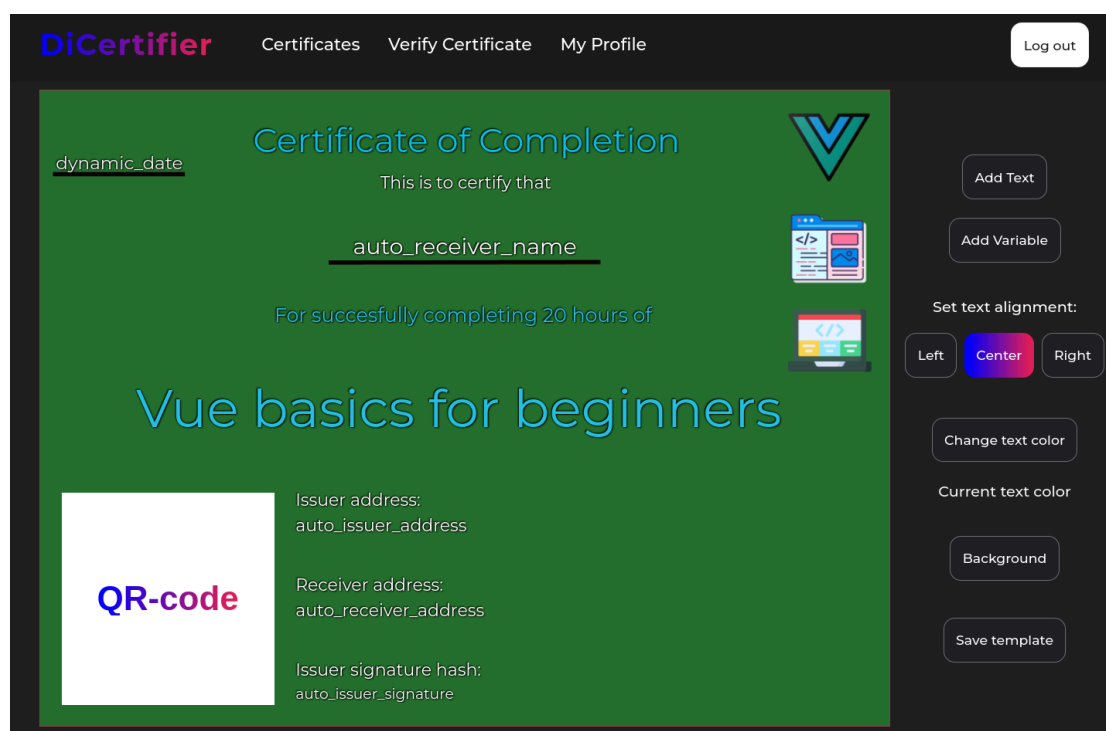


Рис. 11. Сторінка дизайну шаблону сертифікату

На сторінці дизайну шаблону адміністраторам курсу дозволяється самостійно розробляти шаблон документу, який в подальшому може заповнюватися динамічними полями. Для цього на боковій панелі для них доступний ряд наступних інструментів:

- додавання тексту;
- додавання динамічних атрибутів;
- встановлення вирівнювання тексту;
- зміна кольору тексту;
- вибір кольору заднього фону;
- встановлення картинки заднього фону;
- збереження шаблону сертифікату.

Користувач може вільно рухати текстові та динамічні поля, збільшувати/зменшувати розмір шрифту. Для видалення полів з шаблону достатньо пересунути їх за межі рамки, що є дуже зручним у використанні.

Додатково за замовчуванням створюється квадратна зона, яка помічає місце майбутнього QR-коду сертифікату та автоматичні поля, що можуть бути заповнені системою та не потребують зайвого введення повторюваних даних.

Після закінчення розроблення шаблону, користувач може зберегти розроблений шаблон, попередньо назвавши його для подальшої ідентифікації. Відповідний шаблон трансформується під вимоги генератора та зберігається на сервері застосунку.

2.5. Сторінка випуску сертифікатів

Після того, як відповідний шаблон цифрового сертифікату було створено, адміністратор курсу може розпочати процес випуску сертифікатів. Аналогічно до сторінки профілю користувача, сторінка випуску цифрових сертифікатів має два стани. Перший стан, що зображено на рис. 12, відповідає за автоматизований випуск цифрових сертифікатів з використанням імпорту даних.

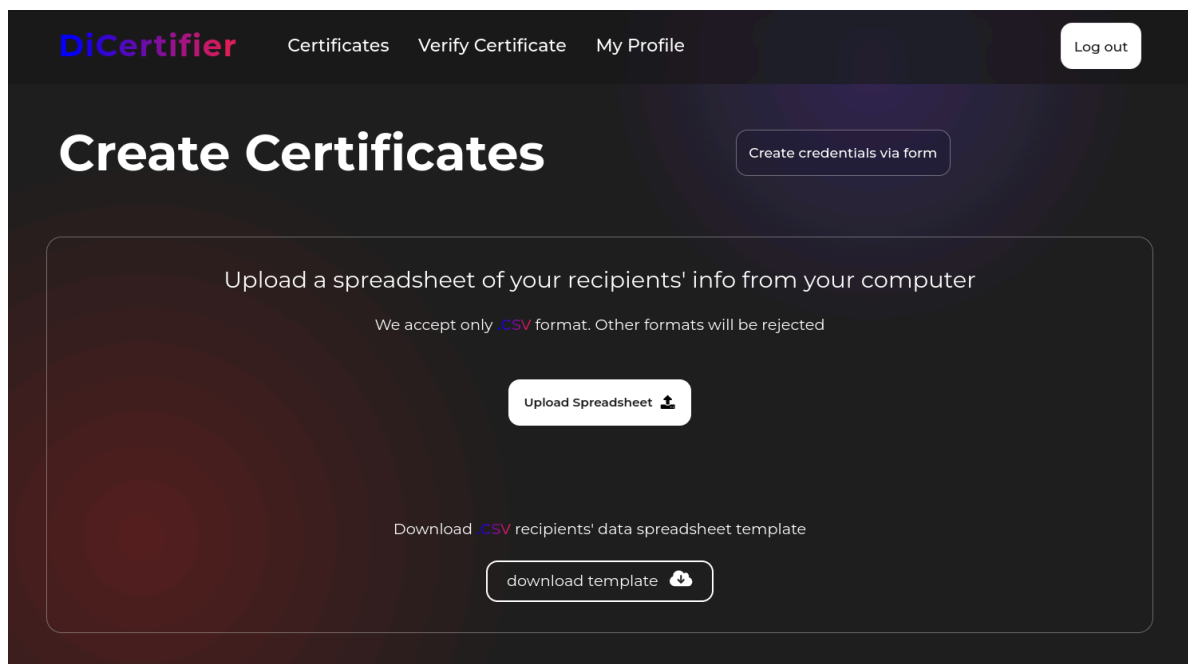


Рис. 12. Сторінка автоматизованого випуску сертифікатів

Цей варіант випуску сертифікатів дозволяє імпортувати дані з існуючих систем обліку/збереження даних, таблиць, документів, баз даних тощо. Для цього необхідно завантажити файл у форматі CSV, що представляє собою таблицю, де у якості колонок є поля обов'язкових даних користувача та динамічні параметри обраного шаблону. Для заповнення такого файлу система надає можливість завантаження прикладу шаблону з використанням усіх обов'язкових колонок.

За допомогою кнопки переключення альтернативного режиму випуску цифрових сертифікатів користувач може перемкнутись на сторінку випуску документів шляхом заповнення відповідної форми, що зображена на рис. 13. Вона представляє собою аналогічну таблицю даних, де користувач може самостійно вводити потрібні дані, з можливістю їх редагування, додавання та видалення рядків таблиці.

Ця форма також має автоматичну валідацію даних, що визначає помилки та повідомляє адміністратора про місце їх виникнення та причину. Приклад роботи валідації форми зображено на рис. 14.

Certificates Form Create credentials via file

#	Receiver address	Receiver name	date	my_custom_atr
X 1	0xe78c8c96989a6cba2c731	Max Slobodzian	27/02/2024	Some custom data
X 2	0xe78c8c96989a6cba2c731	Maximo Slobodzianio	26/05/2024	Some custom data
X 3	0xe78c8c96989a6cba2c731	Mogger de rizzlo	11/09/2001	Some custom data

Add row

Issue certificates

Рис. 13. Форма ручного введення даних про сертифікати

#	Receiver address	Receiver name	date	my_custom_atr
X 1	Invalid	Max Slobodzian	27/02/2024	Some custom data
X 2	0xe78c8c96989a6cba2c731		26/05/2024	Some custom data
X 3	0xe78c8c96989a6cba2c731	Mogger de rizzlo		

Add row

ERRORS OCCURED:

- Credential id [2]; field [Receiver name]:
- Empty field detected
- Credential id [3]; field [date]:
- Empty field detected
- Credential id [1]; field [Receiver address]:
- Incorrect receiver address detected
- Credential id [3]; field [my_custom_atr]:
- Empty field detected

Рис. 14. Приклад валідації даних форми

Після того, як було імпортовано дані чи заповнено форму, дані хешуються для підпису адміністратором з використанням можливостей гаманця MetaMask. Після цього дані відправляються на сервер, де розпочинається власне генерація файлів цифрових сертифікатів та запис

даних про випуск у систему. Приклад згенерованого файлу цифрового сертифікату зображено на рис. 15.

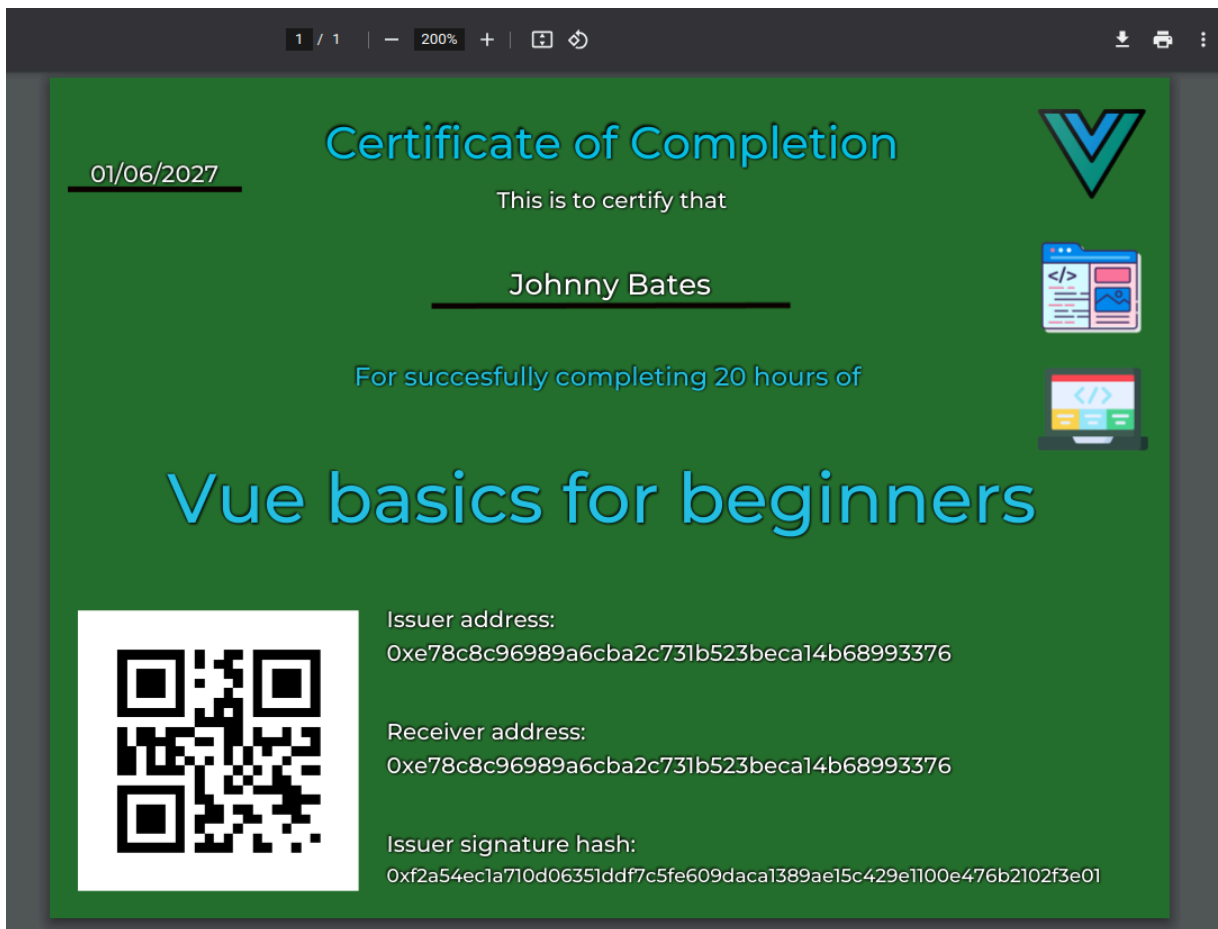


Рис. 15. Згенерований PDF-файл цифрового сертифікату

Після того, як сертифікати було випущено, вони стануть доступні до перегляду на головній сторінці, сторінці курсу та у особистих кабінетах користувачів-отримувачів. Подальші дії зі створеними сертифікатами адміністратор може проводити на сторінці курсу, що була описана у попередніх розділах.

2.6. Сторінка верифікації сертифікатів

Сторінка верифікації сертифікатів призначена для перевірки отриманого документа на автентичність та цілісність. Будь-який гість та

користувач системи має змогу перевірити сертифікат без необхідності попередньої авторизації. Інтерфейс сторінки зображено на рис. 16.

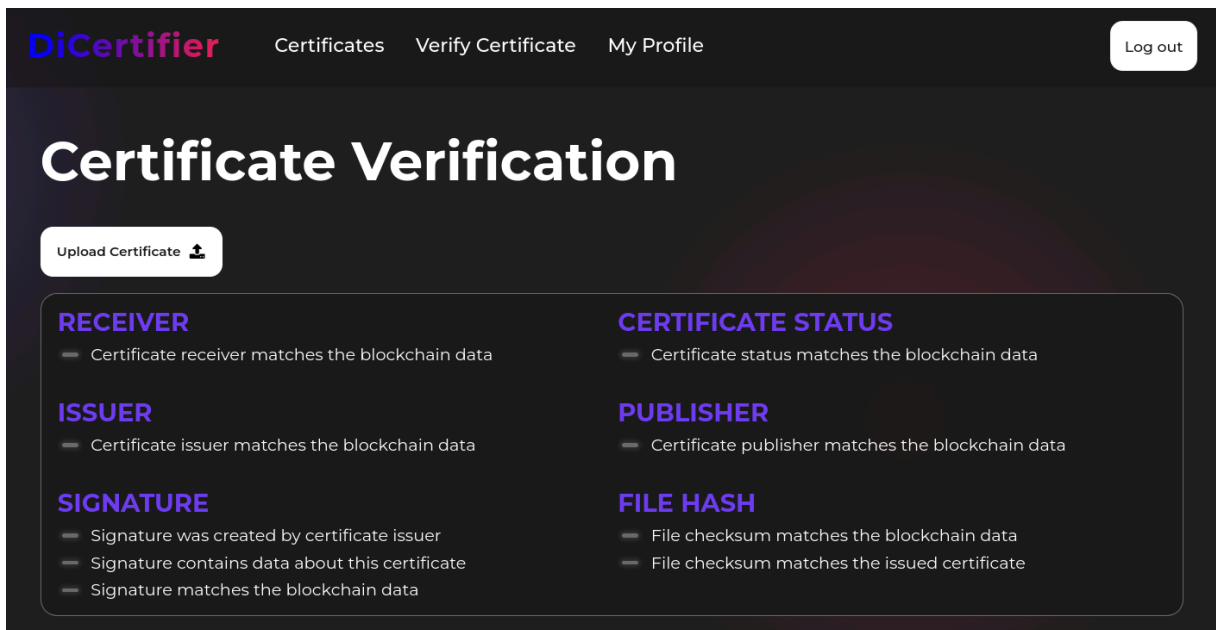


Рис. 16. Сторінка верифікації сертифікату

Перед завантаженням файлу чи сканування QR-коду сертифікату поля верифікації знаходяться в нейтральному стані. Після того, як користувач обрав потрібний файл, система знаходить відповідні дані та звіряє їх з даними блокчейну, якщо сертифікат було попередньо опубліковано.

В залежності від результату верифікації цифрового сертифікату користувачеві відображаються дані про документ та їх достовірність і відповідність.

Нижче наведено приклади та скріншоти можливих випадків верифікації сертифікату:

- сертифікат не знайдено — з оригінальним файлом відбулись незначні зміни, що змінило його хеш-значення;
- сертифікат знайдено, але дані системи було атаковано та змінено;
- сертифікат цілісний.

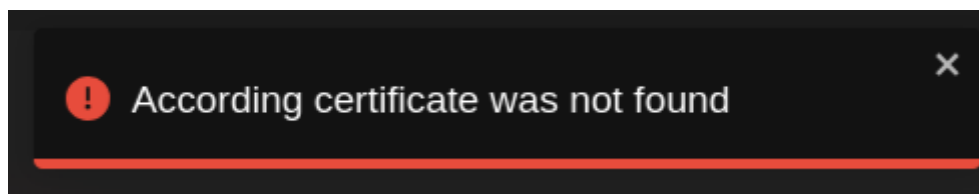


Рис. 17. Верифікація сертифікату. Документ не було знайдено

RECEIVER 0xe78c8c96989a6cba2c731b523beca14b68993376 ✓ Certificate receiver matches the blockchain data	CERTIFICATE STATUS Published ✓ Certificate status matches the blockchain data
ISSUER 0xe78c8c96989a6cba2c731b523beca14b68993375 ✗ Certificate issuer matches the blockchain data ↶ ! Was altered, original -- 0xE78C8C96989a6cba2c731B523beCa14B68993376	PUBLISHER 0xe78c8c96989a6cba2c731b523beca14b68993376 ✓ Certificate publisher matches the blockchain data
SIGNATURE 0xf2a54ec1a710d06351ddf7c5fe609daca1389ae15c429e1100e476b2102f3e01 ✗ Signature was created by certificate issuer ↶ ! Was altered, original -- 0xe78c8c96989a6cba2c731b523beca14b68993376 ✓ Signature contains data about this certificate ✓ Signature matches the blockchain data	FILE HASH fa84f0edc8628b23b9e0fe482640ada307919f97af12681149168b50c2411d95 ✓ File checksum matches the blockchain data ✓ File checksum matches the issued certificate

Рис. 18. Верифікація сертифікату. Дані змінено

RECEIVER 0xe78c8c96989a6cba2c731b523beca14b68993376 ✓ Certificate receiver matches the blockchain data	CERTIFICATE STATUS Published ✓ Certificate status matches the blockchain data
ISSUER 0xe78c8c96989a6cba2c731b523beca14b68993376 ✓ Certificate issuer matches the blockchain data	PUBLISHER 0xe78c8c96989a6cba2c731b523beca14b68993376 ✓ Certificate publisher matches the blockchain data
SIGNATURE 0xf2a54ec1a710d06351ddf7c5fe609daca1389ae15c429e1100e476b2102f3e01 ✓ Signature was created by certificate issuer ✓ Signature contains data about this certificate ✓ Signature matches the blockchain data	FILE HASH fa84f0edc8628b23b9e0fe482640ada307919f97af12681149168b50c2411d95 ✓ File checksum matches the blockchain data ✓ File checksum matches the issued certificate

Рис. 19. Верифікація сертифікату. Сертифікат цілісний