

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Інститут інформатики та радіоелектроніки,
Факультет комп'ютерних наук і технологій

(повне найменування інституту, назва факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавр

(ступінь вищої освіти)

на тему ПРОГРАМНА РЕАЛІЗАЦІЯ ПІДСИСТЕМИ УПРАВЛІННЯ
НАВЧАННЯМ З ВИКОРИСТАННЯМ ТЕХНОЛОГІЇ БЛОКЧЕЙН
SOFTWARE IMPLEMENTATION OF LEARNING MANAGEMENT
SUBSYSTEM USING BLOCKCHAIN TECHNOLOGY

Виконав: студент 4 курсу, групи КНТ-137

Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

Чемерис К.М.

(прізвище та ініціали)

Керівник Табунщик Г.В.

(прізвище та ініціали)

Рецензент Шитікова О.В.

(прізвище та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
 (повне найменування закладу вищої освіти)

Інститут, факультет інформатики та радіоелектроніки, КНТ
 Кафедра програмних засобів
 Ступінь вищої освіти бакалавр
 Спеціальність 121 Інженерія програмного забезпечення
 (код і найменування)
 Освітня програма (спеціалізація) Інженерія програмного забезпечення
 (назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
С.О. Субботін
 “ ” 20__ року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

Чемериса Кирила Михайловича

(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Програмна реалізація підсистеми управління навчанням з використанням технології блокчейн. Software implementation of learning management subsystem using blockchain technology

керівник проєкту (роботи) Табунщик Галина Володимирівна, к.т.н., професор,
 (прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від “30” березня 2021 року № 103

2. Строк подання студентом проєкту (роботи) 28 травня 2021 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Розробка архітектури програми.

3. Основні рішення щодо реалізації компонентів системи. 4. Експлуатація, тестування та експериментальне дослідження програми.

5. Додатки

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

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	Табунщик Г.В., професор		
Нормоконтролер	Камінська Ж.К., асистент		

7. Дата видачі завдання “01” лютого 2021 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	2–3 тижні	Розділ 1
3	Розробка архітектури програми.	4 тиждень	Розділ 2
4	Основні рішення щодо реалізації компонентів системи.	5–6 тижні	Розділ 3
5	Експлуатація, тестування та експериментальне дослідження програмного забезпечення.	7 тиждень	Розділ 4
6	Оформлення пояснювальної записки та документів до неї.	8 тиждень	Додатки
7	Нормоконтроль та рецензування.	9 тиждень	
8	Захист роботи.	10 тиждень	

Студент(ка)

(підпис)

Чемерис К.М.
(прізвище та ініціали)

Керівник проєкту (роботи)

(підпис)

Табунщик Г.В.
(прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра:
171 с., 8 табл., 36 рис., 3 дод., 59 джерел.

ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ, БЛОКЧЕЙН, СИСТЕМА УПРАВЛІННЯ НАВЧАННЯМ, СМАРТ-КОНТРАКТ, ETHEREUM, DAPP, IPFS

Об'єкт дослідження – технологія блокчейн в сфері освіти.

Предмет дослідження – теоретичні аспекти та практичне застосування технології блокчейн в системі управління навчанням.

Мета роботи – виявлення підходів до застосування технології блокчейн в системі управління навчанням.

Матеріали, методи та технічні засоби: структурне та об'єктно-орієнтоване програмування, платформа блокчейн Ethereum, програмне забезпечення Ganache і інтерпретатор командного рядка, мови програмування Solidity, JavaScript, CSS, та HTML, програмні платформи і бібліотеки Node.js, React, Web3.js та IPFS, інтегроване середовище розробки програм Visual Studio Code, персональний комп'ютер з процесором AMD 6 ядер, оперативною пам'яттю 16 Гб, 23" монітором, веб-камерою, клавіатурою та мишею, що функціонує під управлінням операційної системи Microsoft Windows 10 Pro 64-bit.

Результати. Розроблений децентралізований застосунок для управління навчанням.

Висновки. Розроблена програмна підсистема управління навчанням із застосуванням технології блокчейн для підвищення організації навчального процесу, надійного зберігання результатів виконаних студентами робіт, оцінених викладачами.

Галузь використання – організація навчального процесу, оцінювання знань студентів, зберігання результатів робіт та перегляд оцінок в освітніх установах викладачами та студентами.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 171 pages, 8 tables, 36 figures, 3 appendixes, 59 sources.

INFORMATION TECHNOLOGIES, BLOCKCHAIN, LEARNING MANAGEMENT SYSTEM, EDUCATION, SMART CONTRACT, ETHEREUM, DAPP, IPFS

Object of study is blockchain technology in education.

Subject of study are theoretical aspects and practical application of blockchain technology in a learning management system.

Materials, methods, and tools: structural and object-oriented programming, Ethereum blockchain platform, Ganache software and command line interpreter, programming languages Solidity, JavaScript, CSS, and HTML, software platforms and libraries Node.js, React, Web3.js and IPFS, integrated development environment Visual Studio Code, IBM-compatible personal computer with an AMD 6 cores processor, a 16GB RAM, 23" monitor, a webcam, a keyboard and a mouse, that runs under the control of the Microsoft Windows 10 Pro 64-bit operating system.

Results. Developed a decentralized application for learning management.

Conclusions. A software subsystem for learning management with the use of blockchain technology has been developed to improve the organization of the educational process, reliable storage of the results of work performed by students, evaluated by teachers.

Field of use is a organization of the educational process, assessment of students' knowledge, storage of work results and review of grades in educational institutions by teachers, students.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ І СКОРОЧЕНЬ	9
ВСТУП	10
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ	12
1.1 Огляд технології блокчейн	12
1.1.1 Структура блокчейн-ланцюга	14
1.1.2 Типи блокчейнів	15
1.1.2.1 Публічний блокчейн	15
1.1.2.2 Приватний блокчейн	16
1.1.2.3 Гібридний блокчейн (консорціум)	16
1.1.3 Алгоритми консенсусу в технології блокчейн.....	17
1.1.3.1 Коротко про консенсус	17
1.1.3.2 Proof-of-Work (PoW)	18
1.1.3.4 Delegated-Proof-of-Stake (DPoS).....	19
1.1.3.5 Proof-of-Elapsed-Time (PoET).....	20
1.1.3.6 Інші алгоритми консенсусу	21
1.1.4 Види атак на блокчейн.....	21
1.1.4.1 Атака Фінні	21
1.1.4.2 Атака Сивілли	22
1.1.4.3 Атака 51%.....	22
1.1.4.4 Інші атаки	23
1.2 Покоління блокчейнов та їх застосування в різних сферах.....	23
1.3 Смарт-контракт.....	27
1.4 Технологія блокчейн в освіті	29
1.4.1 Історія застосування технології блокчейн в освіті	30
2 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ.....	32
2.1 Схема програми і опис модулів розроблюваної системи.....	32
2.1.1 Загальна схема роботи системи	32
2.1.2 Діаграма прецедентів з описами потоків подій.....	33
2.1.3 Безпека. Діаграма послідовності	40
2.1.4 Діаграма об'єктів смарт-контракту	46
2.2 Дизайн децентралізованого застосунку	48
2.3 Вибір типу блокчейн та платформи блокчейн	55
2.3.1 Обґрунтування вибору типу блокчейн.....	55
2.3.2 Огляд існуючих платформ блокчейн та вибір.....	57
2.3.2.1 Hyperledger Fabric.....	57
2.3.2.2 Ethereum.....	58

2.3.2.3 EOS.....	59
2.3.2.4 Обґрунтування вибору платформи блокчейн. Децентралізована файлова система IPFS.....	60
2.4 Вибір мови програмування.....	61
2.5 Вибір інструментарію розробки програмного забезпечення.....	62
2.5.2 Огляд існуючих засобів розробки	62
2.5.1.1 Atom.....	62
2.5.1.2 Visual Studio Code.....	63
2.5.1.3 WebStorm.....	64
2.5.2 Порівняння середовищ розробки та вибір	64
3 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ	66
3.1 Розробка децентралізованого застосунку	66
3.2 Ініціалізація проєкту	67
3.2.1 Необхідні компоненти для розробки DApp.....	67
3.2.2 Структура проєкту DApp.....	69
3.3 Структура смарт-контракту.....	70
3.4 Розробка клієнтської частини	75
3.5 Установка і застосування децентралізованого сховища IPFS	81
3.6 Безпека передачі та зберігання даних, вибір криптосистеми	83
3.6.1 Коротко про безпеку і обґрунтування вибору криптосистеми	83
3.6.2 Підключення та використання криптосистеми в проєкт	85
4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ	89
4.1 Опис застосування програми	89
4.1.1 Призначення програми	89
4.1.2 Можливості програми.....	89
4.1.3 Основні характеристики програми.....	90
4.1.4 Обмеження, які накладаються на область застосування програми.....	90
4.1.5 Мінімальний склад технічних та програмних засобів.....	90
4.2 Керівництво системного програміста	91
4.2.1 Загальні відомості про програму	91
4.2.1.1 Призначення програми	91
4.2.1.2 Функції програми	91
4.2.1.3 Мінімальний склад апаратних засобів	92
4.2.1.4 Мінімальний склад програмних засобів	92
4.2.1.5 Вимоги до системного програмісту	93
4.2.2 Структура програми.....	93
4.2.3 Налагодження програми.....	94

4.2.4	Перевірка програми.....	94
4.2.5	Повідомлення системному програмісту	95
4.3	Керівництво програміста	96
4.3.1	Призначення й умови застосування програми.....	96
4.3.2	Характеристики програми.....	97
4.3.3	Звернення до програми.....	98
4.3.4	Вхідні та вихідні дані.....	99
4.3.5	Повідомлення.....	99
4.4	Керівництво оператора	100
4.4.1	Призначення програми	100
4.4.2	Умови виконання програми	101
4.4.2.1	Мінімальний склад апаратних засобів	101
4.4.2.2	Мінімальний склад програмних засобів	101
4.4.3	Виконання програми.....	101
4.4.3.1	Реєстрація та вхід в систему.....	101
4.4.3.2	Перегляд списку курсів та управління.....	103
4.4.3.3	Завантаження файлів, здача робіт і оцінювання	104
4.4.3.4	Перегляд журналу дій і оцінок.....	106
4.4.4	Повідомлення оператору	110
4.5	Методика і результати тестування	112
4.5.1	Опис методики проведення тестування	112
4.5.1.1	Тестування смарт-контрактів. Тестування безпеки.....	112
4.5.1.2	Тестування функціональності програми, розробка чек-листа.....	114
4.5.2	Результати тестування	115
	ВИСНОВКИ	119
	ПЕРЕЛІК ПОСИЛАНЬ	121
	ДОДАТОК А Текст смарт-контракту.....	127
	ДОДАТОК Б Текст програми.....	139
	ДОДАТОК В Слайди презентації	162

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ І СКОРОЧЕНЬ

API – Application Programming Interface;
DApp – Decentralized Application;
DPoS – Delegated Proof-of-Stake;
EVM – Ethereum Virtual Machine;
IMS – Information Management System;
IPFS – InterPlanetary File System;
IT – Information Technology (Інформаційні технології);
LMS – Learning Management System;
MOOC – Massive Open Online Course;
PoET – Proof-of-Elapsed-Time;
PoS – Proof-of-Stake;
PoW – Proof-of-Work;
P2P – Peer-to-Peer;
SHA – Secure Hash Algorithm;
БД – база даних;
ДН – дистанційне навчання;
ЄАУ – європейська асоціація університетів;
ЖКГ – житлово-комунальне господарство;
ЖЦ – життєвий цикл;
ПЗ – програмне забезпечення;
СК – смарт-контракт.

ВСТУП

В даний час в інформаційної технології (ІТ) блокчейн є найбільш популярною технологією і може розвиватися як відокремлена технологія, яка може застосовуватися і за рамками систем криптовалют. Технологія блокчейн дає можливість всьому людству оптимізувати найбільш різні сфери життя. Головною перевагою даної технології є те, що її вкрай важко або практично неможливо зламати та немає необхідності в залученні третіх осіб. Технологія блокчейн вперше була запропонована Сатоши Накамото у 2008 році і спочатку була розроблена для використання цифрової валюти Біткоїн (Bitcoin) у 2009 році, але з часом використовується в усі сфери діяльності, в тому числі і в освіту [1].

Використання даної технології має ряд переваг: прозорість, децентралізованість, рівноправність, а також безпеку та багато інших.

Технологія блокчейн в сфері освіти також може бути важливою і потрібною, оскільки потрібен надійний і більш безпечний спосіб фіксації, зберігання та використання отриманих результатів. У блокчейн технології немає єдиного сервера з даними, тому зловмисники не зможуть зламати зашифровану систему, вкрати або підробити документи. Так що блокчейн може демонструвати реальні результати, які зберігаються, і виконані роботи, в тому числі дипломи, а також сертифікати всім, кому це буде необхідно, наприклад, захищати авторство, отримувати визнання, подати заявку на винаходи, перевірити справжність диплома на співбесіду, визначити рівень знання та т.д. Система з використанням технології блокчейн також може надавати максимально повну картину навчального процесу кожного студента, що потенційно підвищує точність оцінювання. Роботи в галузі використання блокчейн в освіті поки знаходиться в початкової стадії [2].

Проблема дослідження полягає в дефіциті теоретичних описів підходів до застосування технології блокчейн в області навчання. Ця проблема пов'язана з тим, що технологія блокчейн є ще новою, недостатньо вивченою в сучасних реаліях і все ще знаходиться в стані невизначеності з правої точки зору.

Актуальність теми:

- необхідність переходу до цифрових технологій;
- можливість підвищення організації навчання, надійності зберігання документів, ефективності та якості життя громадян;
- дефіцит теоретичних описів схем застосування технології блокчейн в області навчання.

Об'єкт дослідження – технологія блокчейн в сфері освіти.

Предмет дослідження – теоретичні аспекти та практичне застосування технології блокчейн в системі управління навчання.

Мета роботи – виявлення підходів до застосування технології блокчейн в системі управління навчанням.

Для реалізації поставленої мети необхідно виконати ряд завдань:

- провести аналіз джерел за темою роботи технології блокчейн, вивчити її принципи роботи;
- провести аналіз джерел в області застосування технології блокчейн в різних сферах діяльності;
- виділити основні напрямки технології блокчейн в області навчання;
- описати підходи до застосування технології блокчейн в системі управління навчанням;
- розробити підсистему управління навчання і застосувати технологію блокчейн;
- оцінити спроби практичного впровадження технології блокчейн в системі управління навчанням, виявити головні проблеми та можливі шляхи розвитку технології блокчейна.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд технології блокчейн

Блокчейн – це новітня технологія, яка базується на принципі децентралізації та організовує базу даних (БД), що складається з ланцюжка послідовних блоків, в кожному з яких зберігається інформація різного виду (транзакції, документи про власність, контракти, твори мистецтва і т.д.), а також інформація про попередньому блоці.

Концепція технології блокчейн була запропонована Сатоши Накамото (Satoshi Nakamoto) у 2008 році і вперше була застосована у вигляді цифрової криптовалюти – біткоїн (Bitcoin) у 2009 році. Після появи біткоїнів сфера застосування технології блокчейн починається розповсюджувати ширше [3].

Основною ідеєю блокчейну є децентралізація. У звичних нам системах є один сервер, який завжди зберігає єдину інформацію у БД та її може бути легко зламати. А з блокчейном все це не так, оскільки дані зберігаються у кожного учасника мережі та синхронізуються. Таку систему називають Peer to Peer (P2P) системою, тобто децентралізованою. У такому випадку інформація зберігається надійніше та вічно, зламувати їх вкрай трудно або практично неможливо. На рис. 1.1 представлена різниця між звичайною централізованою та децентралізованою системою.

У децентралізованій системі замість довіри використовується система, яка заснована на доказі за допомогою криптографії. Це дозволяє будь-яким двом сторонам безпосередньо створювати транзакцію між собою без необхідності довіреної третьої сторони [4].

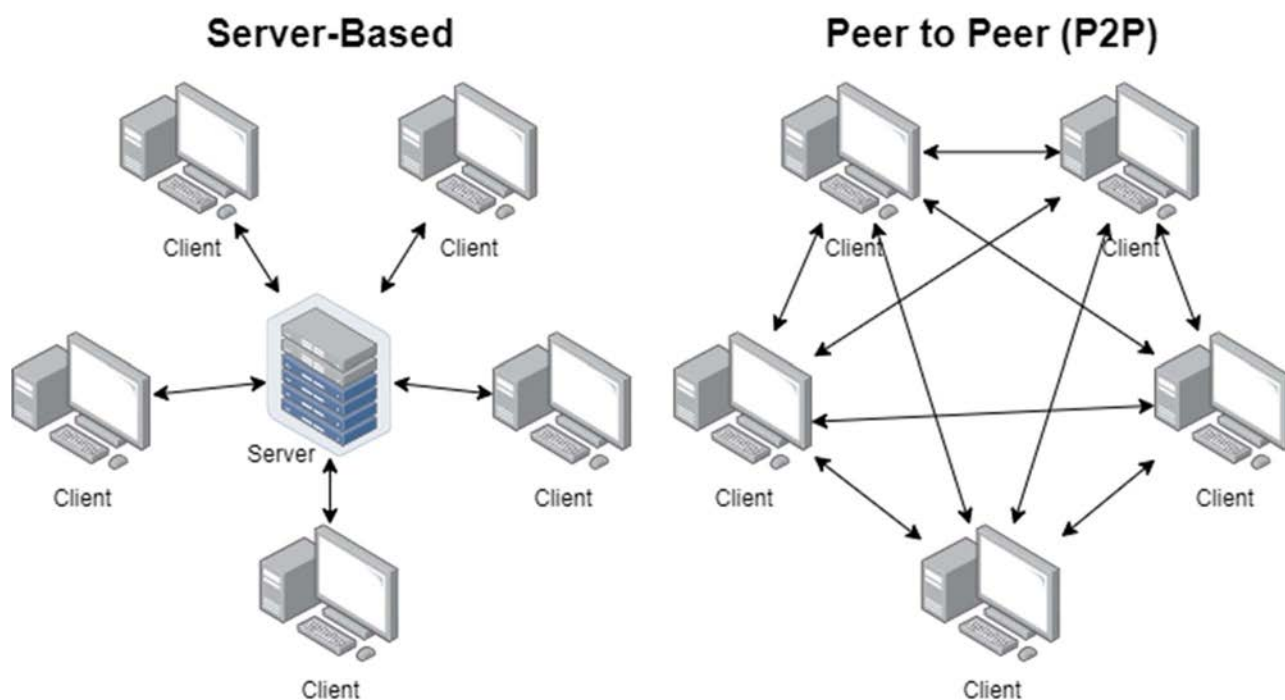


Рисунок 1.1 – Централізована система та децентралізована (Peer to Peer) система

Технологія блокчейн активно розвивається і в майбутньому буде використана в більшість різних напрямках завдяки наступним перевагам [5, 6]:

- децентралізація, де використовується вся мережа, а не один комп'ютер (організація і т.д.). У такому випадку, навіть якщо один або кілька комп'ютерів не може виконувати ніяких функцій, інші можуть зберігати інформацію, що ускладнює атаки і підробку інформації;

- доказовість кожної транзакції. Є криптографічне підтвердження кожної транзакції записи тощо. Зокрема, є закриті (ключі, які належать конкретної особі) і публічні (ключі, які може використовувати всі користувачі мережі) ключі, необхідні для захисту даних, підтвердження дійсності і т.д;

- прозорість (загальний доступ). Хто завгодно та коли завгодно може побачити, які саме операції проводились;

- безпека. Дані про транзакції зберігаються із застосуванням криптографії. Без допомоги ключа ніхто не може читати дані;

- неможливість зміни або видалення доданого блоку. Блок проходить перевірку і додається в блокчейн, а потім дані синхронізуються між усіма учасниками. Після цього змінити або видалити інформацію в блокчейн неможливо, це значно збільшить безпеку;
- висока швидкість транзакцій. Оскільки блокчейн-мережі є одноранговими, транзакції відбуваються безпосередньо між вузлами та ніяк не залежить від їх місцезнаходження;
- постійна робота. Блокчейн-мережа завжди доступна користувачам, ця мережа не має ніяких обмежень по часу роботи та не йде в оффлайн на свята;
- універсальність. Блокчейн може застосовуватися в різних сферах, таких як фінансовий сектор, нерухомість, охорони здоров'я, медицина, а також освіта.

1.1.1 Структура блокчейн-ланцюга

Блокчейн-ланцюг складається з трьох основних частин [7]:

- блок. Ця частина містить список транзакцій, які занесені до реєстру за деякий період. У кожній конкретній реалізації блокчейна розмір блоку, період накопичення транзакцій відрізняються. Транзакцію можна розуміти просто як запис деяких даних;
- цепочка. Хеш – це таке число, яке виходить як результат математичної операції хешування – зв'язування нового блоку даних з попереднім блоком, математично об'єднуючи їх в єдиний ланцюг в порядку зі створення в часі. Це найбільш складна для розуміння концепція технології блокчейн. Функція хешування є математичний алгоритм, який дозволяє відобразити дані будь-якого розміру у вигляді бітової рядка заданої довжини (зазвичай 32 символи). Як криптографічної хеш-функції зазвичай використовується алгоритм Secure Hash Algorithm з розміром хешу – 256 біт (SHA-256);
- мережа. Така мережа блокчейну складається з «повних вузлів», де на кожному вузлі зберігається повна копія всіх транзакцій, які були записані в

блокчейн. Ці вузли розташовані в різних місцях по всьому світу і можуть контролюватися ким завгодно.

На рис. 1.2 представлена структура блокчейну. Оскільки база зберігається у кожного учасника мережі, редагувати або видалити інформацію, яка додана у блокчейні, неможливо. Незмінюваність транзакцій у блокчейні досягається шляхом об'єднання транзакцій в блоки, які впорядковані в часі, і розрахунку криптографічної хеш-функції для кожного з блоків.

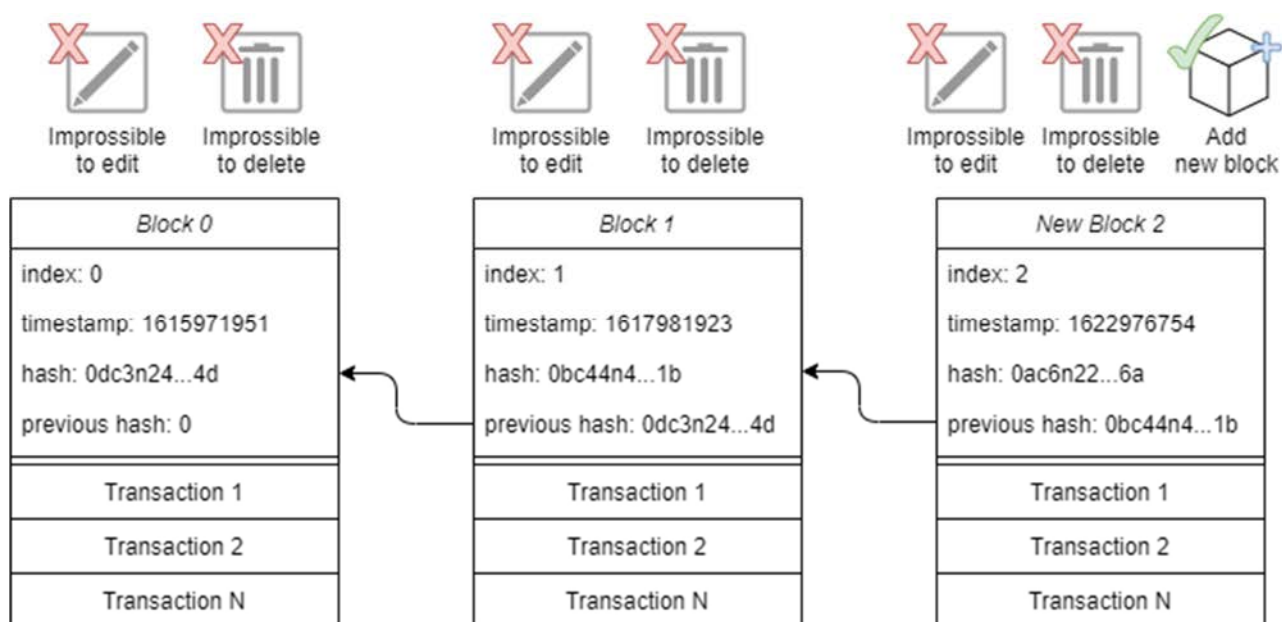


Рисунок 1.2 – Структура блокчейну

1.1.2 Типи блокчейнів

Існують різні типи блокчейнів, такі як публічний, приватний та гібридний (консорціум). Вони відрізняються в залежності від ступеня приватності.

1.1.2.1 Публічний блокчейн

Ця розподілена БД відкрита для всіх. Даний блокчейн характеризується повною відкритістю, тобто кожен може отримати до нього доступ і виконувати

різноманітні операції, і кожен може брати участь в процесі досягнення консенсусу для перевірки транзакції та для створення наступного блоку. У даної типі блокчейну немає ніякого центрального реєстру або достовірної третьої особи, тому публічний блокчейн повністю децентралізований [8].

1.1.2.2 Приватний блокчейн

Це мережа, яка управляється однією організацією, та в її роботі можна брати участь тільки на запрошення такої організації і відповідного від неї дозволу. Тому у учасників можуть бути різні дозволи на читання, запис та перевірку блокчейну. Інформація повинна бути зашифрованою в цілях захисту конфіденційності. У приватних блокчейнів відсутня важлива особливість – децентралізація. Приватні блокчейни швидше, ефективніше і економніше, ніж публічні, які вимагають багато часу і електроенергії для перевірки транзакцій [9].

1.1.2.3 Гібридний блокчейн (консорціум)

Даний блокчейн можна розглядати як частково приватний, а також частково публічний блокчейн, де керує не одна організація, а набір заздалегідь визначених вузлів, який відповідає за консенсус і перевірку блоку. Ці вузли вирішують хто може бути частиною мережі, а хто майнінгом. Цей блокчейн частково централізований через контроль з боку деяких обраних вузлів валідатора, на відміну від приватного блокчейну, який повністю централізований, і публічного блокчейну, який повністю децентралізований. Консорціум вирішує, які будуть дозволи у учасників мережі на читання та запис. Крім того, обмеження консенсусу набором вузлів не гарантує незмінність і незворотність, оскільки контроль над консорціумом з боку більшості вузлів може призвести до втручання в блокчейн [10].

1.1.3 Алгоритми консенсусу в технології блокчейн

1.1.3.1 Коротко про консенсус

Консенсус – це такий процес досягнення угоди, зазвичай позитивного і одноголосного, в якій дозволяється рішення або дію спільно без попереднього голосування або обговорення конкретного питання. Це ідеально вписується в структуру розподіленого реєстру блокчейну. [8].

Оскільки блокчейн не контролюється жодним центральним органом, який би перевіряв і оновлював реєстр з транзакціями та свіжими даними, розподілені вузли все одно повинні узгоджувати валідацію транзакції, тому важливо використовувати алгоритм консенсусу. Зацікавлені сторони в мережі повинні вирішити на рівноправній основі те, які транзакції повинні бути додані в блокчейн.

Роль консенсусних алгоритмів полягає в досягненні рівня надійності мережі, побудованої на серії вузлів (таких пристроїв, які з'єднані з іншими пристроями, як частина комп'ютерної мережі). Припустимо, що транзакція здійснена, алгоритм почне працювати – обмінюватися інформацією в мережі, щоб перевірити, чи може дана дія мати місце [11].

Такий процес також застосовується для створення нових вузлів даних у блокчейні або при синхронізації мережевого обладнання так, щоб забезпечити узгодженість всього консенсусу.

Кожна технологія блокчейн відрізняється, тому що всі вони використовують різні консенсусні алгоритми, щоб зробити свою платформу краще за інших. Існують різні типи консенсус-алгоритмів. У кожного є свої переваги та недоліки, при спробі збалансування безпеки, з функціональністю і масштабованістю. Найбільш використовуваними з них є Proof-of-Work (PoW) та Proof-of-Stake (PoS), а також Delegated-Proof-of-Stake (DPoS) [12].

1.1.3.2 Proof-of-Work (PoW)

Proof-of-Work – найбільш популярний алгоритм, який націлений на докази виконання роботи і використовується для підтвердження транзакцій та створення нових блоків. Основний сенс полягає в тому, щоб вузли мережі блокчейн, які підтверджують транзакції, проробляли досить складну тривалу обчислювальну роботу (знаходження рішення задачі), результат якої легко і швидко перевіряється іншими вузлами мережі.

Винагорода від блокчейн мережі отримує той вузол, який повністю провів всі необхідні обчислення. Тому всі вузли борються між собою, збільшуючи ємність обчислювальних ресурсів, заради того, щоб опинитися першим вузлом, який отримає винагороду [13].

Такий алгоритм широко використовується у майнінгу криптовалюту, для перевірки транзакцій та майнінг нових токенів. Проекти Bitcoin, Ethereum та Litecoin використовують даний алгоритм консенсусу [14].

Переваги:

- безпека. Даний алгоритм прагне так, щоб зловживання послугами (DoS-атак або організації розсилок спаму) були занадто дорогими і вимагають занадто високих енерговитрат [15].

Недоліки:

- обмежена масштабованість;
- низька пропускна здатність;
- дана модель потенційно вразлива перед атакою 51% [16];
- великі енергетичні витрати. Велика кількість вузлів виробляють обчислення, але в підсумку винагороду отримує лише один той вузол, який перший проводив успішну роботу [13].

1.1.3.3 Proof-of-Stake (PoS)

Даний алгоритм є найбільш поширеною альтернативою PoW. Основний сенс полягає в тому, що алгоритм формування блоку не залежить від потужності обладнання, а від балансу вузлів. Тобто в цій алгоритмі для створення наступного блоку з великою ймовірністю вибирається той вузол, який має великий баланс. Можливість вибору вузла для створення блоку пропорційна балансу (кількості монет).

Цей алгоритм використовують проєкти Cardano, EOS, Stellar, Tezos і NEO [17].

В даний час мережа Ethereum знаходиться в процесі поступової відмови від майнінгу на основі протоколу консенсусу PoW на користь принципу докази володіння (PoS) [18].

Переваги:

- підвищена масштабованість на базовому рівні;
- потенційна економія енергії в порівнянні з PoW алгоритмів [15];
- невразливий для атаки 51% [17].

Недоліки:

- мотивація в концентрації коштів, що може призводити до централізації мережі. Це веде до зниження безпеки [13].

1.1.3.4 Delegated-Proof-of-Stake (DPoS)

Даний алгоритм розроблений у 2013 році Деніелем Ларімером та є розвитком PoS, являє собою алгоритм консенсусу іншого типу, який заснований на системах голосування. У DPoS-системах учасники використовують свої маркери, щоб вибирати валідаторів, які перевіряють і додають блоки за винагороду. Даний алгоритм використовують блокчейн-проєкти BitShares, потім в Steem і EOS [19].

Переваги:

- висока масштабованість;
- висока пропускна здатність;
- низькі витрати енергії на обслуговування мережі;
- низькі комісії.

Недоліки:

- недостатня безпека.

1.1.3.5 Proof-of-Elapsed-Time (PoET)

Даний алгоритм є одним з найбільш справедливих узгоджених алгоритмів, який вибирає наступний блок, при цьому використовуючи тільки чесні кошти. У цьому алгоритмі кожен валідатор в мережі отримує реальний шанс створити свій власний блок. Всі вузли, щоб робити це, очікують випадкова кількість часу і додають підтвердження свого очікування в блоці. Створені блоки передаються в мережу для розгляду іншими учасниками. Переможцем вважається той валідатор, який має найменше значення таймеру в частині докази. Блок переможного вузла валідатору додається в блокчейн. В алгоритмі є додаткові перевірки, щоб не дати вузлів завжди перемагати на виборах, для цього генерується найнижче значення таймеру. Такий алгоритм широко використовується в багатьох приватних та корпоративних блокчейнах, в основному Hyperledger [15].

Переваги:

- висока ефективність;
- мала ресурсомісткість.

Недоліки:

- не підходить для публічних блокчейнів;
- низька безпека;
- залежність від третьої сторони, яка розробляє набір інструкцій для процесору.

1.1.3.6 Інші алгоритми консенсусу

Крім перерахованих вище популярних консенсусних-алгоритмів, які використовує більшість платформ, на ринку існує велика кількість інші специфічні, такі як Leased Proof-of-Stake (LPoS), Simplified-Byzantine-Fault-Tolerance (SBFT), Delegated-Byzantine-Fault-Tolerance (DBFT), Proof-of-Activity (POA), Proof-of-Importance (PoI), Proof-of-Capacity (PoC), Proof-of-Burn (PoB), а також Proof-of-Weight (PoWeight). Такі алгоритми хоч і не були прийняті багатьма проєктами, але вони можуть зіграти велику роль у майбутньому [11].

1.1.4 Види атак на блокчейн

Блокчейни з деякими алгоритмами можуть бути зламані, наприклад, блокчейн з використанням алгоритму PoW може бути підданий атакам, заснованим на атаці «подвійного витрачання» – повторної витрати коштів (це якщо зломисник намагається одночасно витратити свої криптовалюти у двох одержувачів [8]). Дана атака може бути успішною, якщо в мережі блокчейну використовується мале число підтверджень блоків або, якщо у зломисника знаходиться більша кількість обчислювальної потужності, ніж у всіх інших майнерів [20].

1.1.4.1 Атака Фінні

Атака Фінні – різновид атаки подвійного витрачання. Атакуючий готує транзакцію з оплатою товару і разом з нею готує блок, що містить транзакцію на переклад цих коштів на інший свій рахунок, але не публікує цей блок в мережі. Як тільки транзакція з оплатою підтверджується одним з майнерів і зломисник отримує товар, він негайно публікує заздалегідь підготовлений блок в мережу. В цьому випадку в мережі виявляється два ланцюжки блоків однакової довжини. І якщо інші майнери будуть розвиватимуть другу ланцюжку, що містить

транзакцію на переказ грошей на рахунок атакуючого, то транзакція переказу грошей продавцю буде відхилена, і, отже, продавець втратить гроші, так як товар вже був відправлений.

Захистом в даному випадку є очікування продавцем деякого достатнього числа підтверджень транзакцій, що зменшує імовірність даної атаки, але не усуває її повністю [20].

1.1.4.2 Атака Сивілли

Дана атака є видом атаки в тимчасовій мережі, в результаті якої жертва підключається тільки до вузлів, контрольованим зловмисником.

В ході атаки Сивілли створюється величезна кількість підроблених учасників, в результаті чого алгоритм консенсусу схиляється в їхній бік. Спостерігачеві ці особистості здаються окремими користувачами, але насправді всі вони контролюються одним об'єктом. В результаті зловмисник має значний вплив на кількість голосів і може зруйнувати або пошкодити мережу. Оскільки більшість блокчейнів стягують комісію за участь в консенсусі, атака Сивілли стане дуже дорогою операцією. До того ж, чим більше справжніх учасників блокчейну, тим блокчейн стабільніше [21, 22].

1.1.4.3 Атака 51%

Така атака вважається, якщо у зловмисника зосереджено понад 50%, тобто не менше половини обчислювальної потужності мережі блокчейну, відданої під майнінг. В даному випадку атакуючий, володіючи більшістю голосів, може відхиляти або підтверджувати необхідні йому транзакції швидше створюючи нові блоки, ніж інші майнері. Захиститися від такої атаки неможливо. Її можна тільки запобігти шляхом моніторингу мережі. У закритих блокчейнах, необхідно упевнитися, що вузли мережі перебувають в довірених місцях [20].

Таку атаку може провести держава або представник крупного бізнесу (банк або ворожий хедж-фонд). Витрати на необхідні обчислювальні потужності та енергію для подібної атаки повинні бути величезними. Це дуже дорога атака [8].

1.1.4.4 Інші атаки

Існують і інші типи атак крім подвійного витрачання.

Вони включають атаки повторного відтворення, атаки на мережу, атаки на пули майнерів, а також атаки на користувачів.

Атаки повторного відтворення – це така атака, яка вдає із себе кібератаку, при якій зловмисник перехоплює дані, і повторно здійснює їх передачу через мережу [23].

Атаки на мережу у блокчейні не мають такої сили як в випадку централізованих мереж, так як в розподіленій мережі можна атакувати лише окремі вузли або пули.

Атаки на пули майнерів спрямовані на підвищення обчислювальної потужності зловмисника, що підвищить його шанси на успішне проведення атаки «подвійного витрачання».

Атаки на користувачів спрямовані на спробу вкрати їх закриті ключі для отримання контролю над їх гаманцями [20].

1.2 Покоління блокчейнов та їх застосування в різних сферах

З моменту створення першого блокчейну ця технологія пройшла кілька етапів розвитку, демонструючи у кожному новому поколінні більш високу швидкість транзакцій та рівень безпеки. Область застосування технології блокчейн також розширилася і вийшла далеко за межі криптовалюти. Сьогодні блокчейн доводять свою ефективність в освіті, медицині, торгівлі, виробництві

та багатьох інших сферах, а попит та популярність технології стрімко зростає в міру її розвитку [24].

Станом на березень 2021 використовувалися п'ять працездатних покоління блокчейну:

- Blockchain 1.0 (2009). Це валюта. Криптовалюта застосовуються в різних застосунках, що мають відношення до грошей, наприклад системи переказів і цифрових платежів [25]. Найбільш популярна криптовалюта – біткоіни. У мережі біткоіни використовувався алгоритм докази роботи (PoW), однак основною проблемою досі залишається масштабованість [24]. У біткоіни швидкість досягається до 10 транзакцій в секунду, а ноди заповнюються в середньому за 10 хвилин [27];

- Blockchain 2.0 (2013). Це контракти. Цілі класи економічних, ринкових і фінансових застосунків, в основі яких лежить блокчейн, працюють з різними типами фінансових інструментів – з акціями, облігаціями, ф'ючерсами, заставними, правовими титулами, розумними активами і розумними контрактами [25]. До такого покоління належить перша відома платформа Ethereum, якій вдалося досягти швидкості до 20 транзакцій в секунду. А ноди заповнюються в середньому за 10-15 секунд [24, 27]. В даній мережі використовувався алгоритм докази роботи (PoW), але потім пізніше використовувався алгоритм докази частки (PoS);

- Blockchain 3.0 (2017). Це застосунки, область застосування яких виходить за рамки грошових розрахунків, фінансів і ринків. Вони поширюються на сфери державного управління, охорони здоров'я, науки, освіти, культури і мистецтва [25]. Багато платформи даного покоління, такі як Cardano і EOS, працюють на механізмі консенсусу делеговане доказ частки володіння (DPoS), що дозволяє істотно знизити витрати на електроенергію і генерувати блок за 0,5 – 5 секунд [24];

- Blockchain 4.0 (2018). У цьому поколінні використовуються повністю нові алгоритми консенсусу. Такі проєкти, як російський стартап MetaHash, стверджують, що використовують блокчейн-технологію четвертого

покоління, засновану на протоколі TraceChain. Децентралізована мережу обміну цифровими активами і платформа для розробки DPoS, що працюють в режимі реального часу, побудована на автоматичному самонавчального протоколі маршрутизації сигналів. Початкова швидкість в 50 000 транзакцій в секунду, порівнянна з платіжними системами Visa і MasterCard, поступово збільшується в міру додавання вузлів з більшою пропускну здатністю [24]. До такого покоління також відноситься проєкт Seele, в якому величезну роль грає протокол нейронного консенсусу [28]. Алгоритм заснований на «натуральних числах», що дозволяє досягти консенсусу за допомогою асинхронних обчислень даних для великої кількості вузлів мережі. Чим вище кількість користувачів в мережі, тим швидше досягається консенсус. А швидкість досягається до 1 000 000 транзакцій в секунду [29];

– Blockchain 5.0 (2019 – майбутнє). Творцем блокчейну останнього покоління вважається Relictum Pro – нескінченно масштабована децентралізована платформа на базі технології HYPERNET з розвиненою системою інтелектуальних контрактів, здатна формалізувати більше 80% подій людського життя, від логістики та купівлі/продажу товарів до авторських прав. У Relictum Pro розмір блоку в 8 000 разів менше, ніж в біткоіни, а пропускну здатність досягає одного мільйона транзакцій в секунду [24]. Ноди заповнюються в більшості випадків за 0.5 - 1 секунд. У цьому проєкті використовується алгоритм консенсусу PoT, який полягає в його максимальному захисті від колізій, захист від атаки 51% і навіть в стійкості до створення квантового комп'ютера [27].

Блокчейн відомий переважно як фінансова технологія, на якій працює криптовалюта. Але застосування розподілених реєстрів може поширитися далеко за рамки сфери фінансів. Найбільші компанії з різних областей також звернули увагу на цю технологію і готові використовувати всі корисні можливості блокчейну, які він може дати [30]. У таблиці 1.1 представлені різні приклади перших напрацювань використання блокчейн-технологій з угрупованням по класах застосунків [31].

Таблиця 1.1 – Застосування технології розподіленого реєстру в застосунках Blockchain

Клас застосунків	Області застосування застосунків
Blockchain 1.0	
Інформація про конкретну транзакцію та її цінності, призначеної в системі	Криптовалюта в різних застосунках, що мають відношення до фінансових транзакціях, наприклад системи переказів і цифрових платежів
Blockchain 2.0	
Гарантійні зобов'язання	Оформлення гарантійних зобов'язань, тристоронній арбітраж, багатостороння підпис, угоди з використанням рахунків Escrow
Фінансові транзакції	Цінні папери, акції компаній, краудфайдинг, облігації, взаємні фонди, похідні фінансові інструменти, ануїтети, пенсії
Приватні документи	Боргові розписки, договори, парі, підписи, заповіти, доручення
Документи, що вимагають засвідчення	Страхові свідоцтва, свідоцтва про власність, нотаріальне завірення документів
Реєстрація нематеріальних активів	Патенти, торгові марки, авторські права, бронювання і т.д.
Blockchain 3.0	
Свідоцтва та ліцензії, завірені державою	Свідоцтва про право власності на земельні ділянки та нерухомість, свідоцтва про реєстрацію транспортних засобів, ліцензії на право заняття певними видами діяльності

Продовження табл. 1.1

Клас застосунків	Області застосування застосунків
Посвідчення, завірені державою	Посвідчення особи, паспорту, свідоцтво про реєстрацію виборця, водійські посвідчення, свідоцтво про народження, шлюб і смерть
Інформація та документація, що відноситься до медицини	Дані історії хвороби пацієнтів медичних установ, інформація про результати обстежень, реєстрація прав доступу медичного персоналу до певних даних і конкретним пацієнтам
Інформація та документація в сфері освіти, науки, культури	Дані та інформація про студентів і викладачів, науковців, працівників культури і мистецтва, різних транзакціях в сфері освіти, науки, культури (в т.ч. показниках роботи установ і окремих осіб)
Інформація та документація в сфері житлово-комунального господарства (ЖКГ)	Дані та інформація про різні транзакції в сфері ЖКГ: показники споживання електроенергії, води, телекомунікаційних послуг, функціонування систем «розумного будинку» і т.п.

1.3 Смарт-контракт

Смарт-контракт (СК) – це така комп’ютерна програма, яка зберігається в блокчейн і використовується для укладання та виконання різних завдань (угоди, проведення транзакцій і т.д.) при дотриманні певних умов без участі третьої особи. СК можуть автоматично виконувати всі необхідні умови договору [32].

Ідея СК вперше була запропонована американським вченим в галузі інформатики та криптографії Нік Сабо в 1994 році. СК використовують для проведення різних операцій у блокчейні. Вони як комп’ютерні алгоритми, які автоматично можуть контролювати і перевіряти запис даних. При цьому всі умови проведення операцій повинні бути формалізовані і переведені у

відповідний програмний код. В алгоритмі блокчейну біткоіни були закладені принципи виконання СК, але не були розроблені в клієнтському програмному забезпеченні (ПЗ), тому СК широко застосовуються після запуску блокчейну Ethereum у 2015 році [33].

Основною платформою для роботи зі СК є Ethereum, але існують і інші спеціалізовані платформи, такі як EOS, NEO, Aeternity, Tron, Hyperledger Fabric, Stellar і інші. Ethereum дає можливість розробникам програмувати СК різної складності з допомогою мови програмування Solidity, який схожий на JavaScript [32, 34].

Технологія СК сама по собі також є перспективною технологією для багатьох галузей, в тому числі і для освіти. СК виключають можливість підробки дипломів, а також зробить освіту дешевше і якісніше, покращує процес навчання. Спілкування між викладачами і студентами відбувається безпосередньо, студенти можуть завантажувати виконані роботи, а викладачі можуть залишати оцінки і при цьому результати студентів назавжди залишаться у блокчейні [35].

До головних переваг СК відноситься [36, 37]:

- автономність. Немає необхідності користуватися послугами юриста для проведення транзакцій використовуючи СК. Виконання відбувається в автоматичному режимі;
- швидкість. Оскільки СК використовують програмний код для автоматизації завдань замість ручного виконання, вони збільшують швидкість виконання процесів і зменшує ризик появи помилок;
- зниження затрат. Витрати знижуються при використанні СК, оскільки вони не залежать від третьої особи;
- безпека. Використовуючи СК задіюються криптографічні інструменти та шифрування, що забезпечують документам більш високий захист;
- точність. СК дозволяють уникнути неточностей при введенні даних, які могли б допускатися при заповненні форм вручну.

1.4 Технологія блокчейн в освіті

Основною частиною освітнього процесу є підсумкове і проміжне оцінювання, такі як іспит, лабораторні/практичні роботи, контрольні роботи та інші навчальні заходи, в ході яких учні демонструють свої навчальні знання, вміння, навички і т.д. Тут потрібен надійний і безпечний спосіб фіксації, зберігання і використання отриманих результатів. Навчальні заклади та атестаційні центри, які проводять такі заходи, використовують спеціальні процедури і мають працівників, які оформляють екзаменаційні документи і видають паперові сертифікати. У цифровому освітньому середовищі можна відмовитися від паперових документів і скористатися технологією блокчейн [38, 39].

Сьогодні багато освітні організації використовують інформаційно-керуючі системи (IMS) для обліку учнів, відвідуваності занять, планування занять і роботи викладачів. Одночасно вони використовують системи управління навчанням (LMS), де знаходяться навчальні матеріали та завдання, а також збираються дані про хід і результати навчальної роботи [40]. Більшість LMS мають центральний реєстр і достовірне третя особа, тобто дані зберігаються на одному місці, тому є проблеми з безпекою і надійності – на даний момент існують кілька видів атак на таку мережу для зміни даних і т.д.

Блокчейн може з успіхом застосовуватися в сфері освіти для формування цифрового портфоліо зберігання атестатів і дипломів, екзаменаційних і творчих робіт, результатів іспитів і освітніх досягнень (тексти виконаних контрольних робіт, відеозапису з виступами екзаменованих і інші) у вигляді унікальних цифрових записів в розподіленій БД. Блокчейн дозволяє демонструвати результати і творчі роботи всім, кому це необхідно, захищати авторство, подавати заявки на винаходи і отримувати визнання [41].

Головна і вічна проблема з сертифікатами та дипломами про успішну закінчення навчання полягає в тому, що ніхто не знає, як їх перевірити. Використання паперу для підтвердження своїх повноважень здається застарілою

системою, тим більше що підроблені дипломи реальних університетів можна легко купити в інтернеті. Така проблема часто з'являється на співбесіді при прийомі на роботу [42]. Тому цінність цієї технології для освіти полягає в тому, що вона гарантує надійність і безпеку, а самі записи можуть містити різні типи даних. Наприклад, за допомогою технології блокчейн можна зберігати інформацію про іспити, виданих дипломах і сертифікатах разом з інформацією про те, хто і коли їх проводив або видавав. Таким чином, паперовий документ втрачає свою унікальність - тут всі бажаючі можуть негайно, не звертаючись до архівів, переконатися в його автентичності та отримати його завірену копію [41].

Деякі освітні програми і організації вже почали використовувати замість паперових посвідчень про успішне закінчення навчання цифрові посвідчення. Цифрові посвідчення можна включити в блокчейн, що підвищить їх доступність і захист від підробок. Блокчейн можна використовувати і в якості портфеля особистих досягнень. Кожен може додати до свого запису виконану ним роботу, літературний твір, наукову статтю, опис винаходу і т.д [41].

Розвиток технології блокчейн в освіті – спільне завдання, яку намагаються вирішувати і державні органи, і бізнес. Існує три принципових рішення: публічний, приватний і змішаний блокчейн [43].

1.4.1 Історія застосування технології блокчейн в освіті

Впровадження технології блокчейн в освіту вже почалося і такі технології широко поширені в зарубіжних країнах і відкривають величезні можливості для його ефективного використання в подальшому. Так, наприклад, в Японії, Сінгапурі, США, Гонконгу, Естонії, Великобританії технології блокчейн вже активно впроваджуються, а сінгапурська система освіти активно використовує онлайн-навчання і вважається однією з кращих [44].

Вперше, технологію блокчейн в освіті офіційно застосували у 2017 році. Це був University of Nicosia (Cyprus), який вирішив подібним способом модернізувати, спростити процес пошуку і зберігання будь-яких документів про

вміння студента (диплом, сертифікат, наукова робота). Це був перший університет, який приймає для оплати за навчання криптовалюту біткоїн. Університет відкрив Massive Open Online Course (MOOC), які доступні для дистанційного навчання студентам більш ніж з 83 країн. Отримуючи диплом, можна не сумніватися в його значенні. University of Nicosia є членом ЄАУ (Європейської Асоціації Університетів). Як дипломи, так і сертифікати навчального закладу, справжні і визнаються в будь-якої країні [1, 45].

В Японії посилено використовують платформи блокчейну в освітній сфері. В кінці лютого 2019 року Sony та Fujitsu повідомили про створення блокчейн-системи, призначеної для зберігання і перегляду достовірної інформації про освіту громадян. Технологія виведе на чисту воду хто шукає роботу, які надають підроблені дипломи, а також іноземців, які намагаються отримати статус японського резидента обманним шляхом. Японські мовні школи в Японії також можуть порівняти видані сертифікати з зареєстрованими даними в децентралізованому реєстрі, щоб переконатися в їх автентичності [46].

Державний Мельбурнський університет планує впровадити блокчейн в систему атестації вузу, щоб уникнути підробок документів про освіту і нагороди. Розробка університету заснована на стандарті Blockcerts, розробленому в минулому році компанією Learning Machine Technologies і кафедрою Media Lab Массачусетського технологічного інституту [47].

Найбільший корейський розробник онлайн-навчання на азіатському ринку Ubion Co. заявив про початок проєкту по впровадженню технології блокчейн в сучасні LMS для створення стабільного і прозорого доступу до персональної інформації і отриманим студентами сертифікатам. Даний проєкт буде реалізований спільно з малими стартапами з Індонезії і протестований на базі LMS Moodle [48].

2 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

2.1 Схема програми і опис модулів розроблюваної системи

2.1.1 Загальна схема роботи системи

У даній роботі буде реалізована підсистема управління навчанням з використанням технології блокчейн. У підсистемі будуть реалізовані як мінімум наступні основні функції:

Для адміністратора:

- реєстрація користувачів. Тільки адміністратор може зареєструвати користувачів і зберегти їх в блокчейн за запитами користувачів. Подробиці про реєстрацію користувачів будуть описані в наступних підрозділах;

- повний контроль і зміна навчального процесу;
- моніторинг мережі блокчейна.

Для викладачів:

- реєстрація і авторизація. Для реєстрації користувач заповнює форму заявки з деякими особистими даними і відправляє, потім адміністратор перевіряє і виконує реєстрацію для збереження даних про викладача в блокчейн;

- додавання і редагування нового курсу;
- додавання завдань і матеріалів до курсу;
- додавання учасників (викладачів і студентів) в обраний курс;
- перевірка робіт студентів та оцінювання.

Для студентів:

- реєстрація і авторизація. Реєстрація для студентів проводиться за тим же принципом, що і для викладачів;

- перегляд списку курсів. З'явиться список показує лише ті курси, в яких студенти брали участь;

- вибір потрібного курсу і отримання матеріалів, завдань;
- здача своїх робіт в блокчейн;
- отримання повідомлень і перегляд своїх оцінок;
- перегляд журналу дій.

На рис. 2.1 представлена діаграма прецедентів, яка описує основні функціональні вимоги нашої системи з точки зору прецедентів. А в наступному підрозділі описані деякі прецеденти.

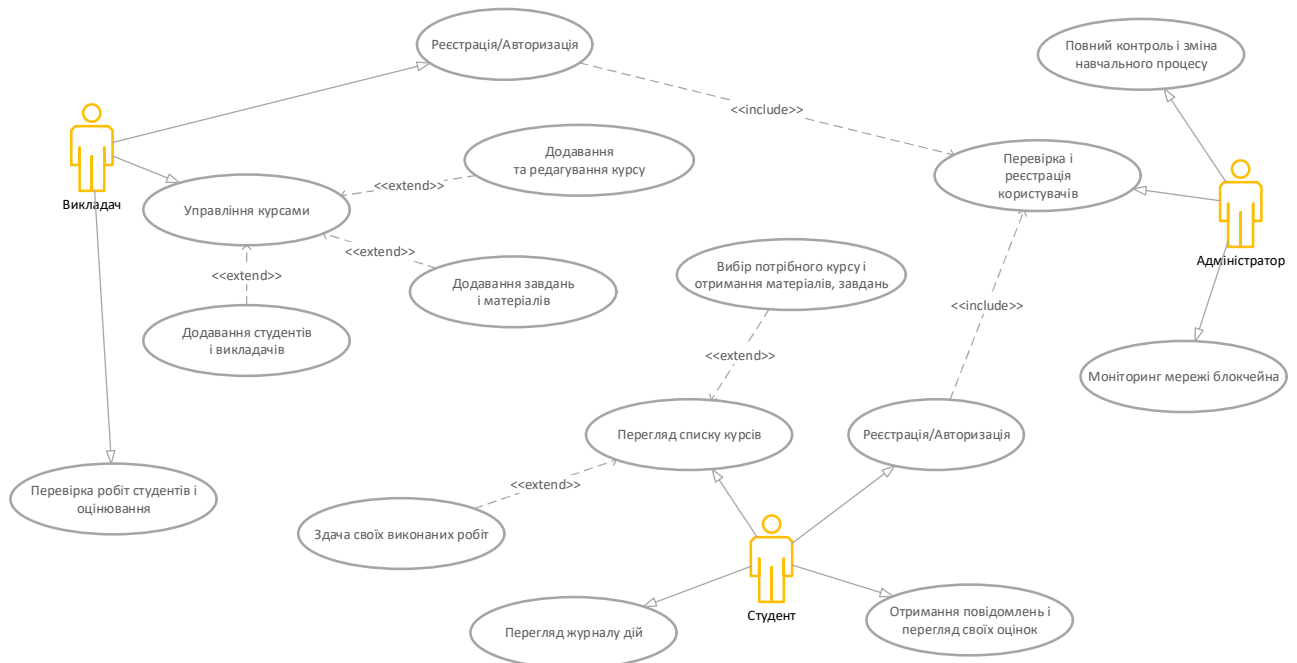


Рисунок 2.1 – Діаграма прецедентів

2.1.2 Діаграма прецедентів з описами потоків подій

Прецедент «Реєстрація та авторизація користувача» показаний на рис. 2.2 та описаний потік подій. У даної системі управління навчанням реєстрація користувачів проводиться по іншому. Як правило, будь-які незареєстровані користувачі (викладачі та студенти) не можуть самотійно безпосередньо зареєструватися в систему через такі наступні причини:

- підроблені акаунти;
- кілька акаунтів на одну людину;
- аккаунт людини, яка ще не вступила в освіту/академію;
- аккаунт з неоригінальними або некоректними даними, наприклад ПІБ,

група.

Якби користувачі самостійно зареєстрували з некоректними даними, то такі акаунти могли б зберігатися марними в блокчейн і при цьому обсяг розподіленої бази даних збільшився б у всіх учасників мережі. Тому, нові користувачі можуть тільки заповнити форму заявки з особистими даними і подати його до навчального закладу, після чого адміністратор перевірить всі дані про студента або викладача, а потім виконає реєстрацію користувача, далі дані для входу в систему автоматично відправляються користувачеві по e-mail. В кінцевому підсумку користувач може увійти в систему за допомогою отриманих даних для входу і встановити пароль для вилучення приватного ключа за бажанням.

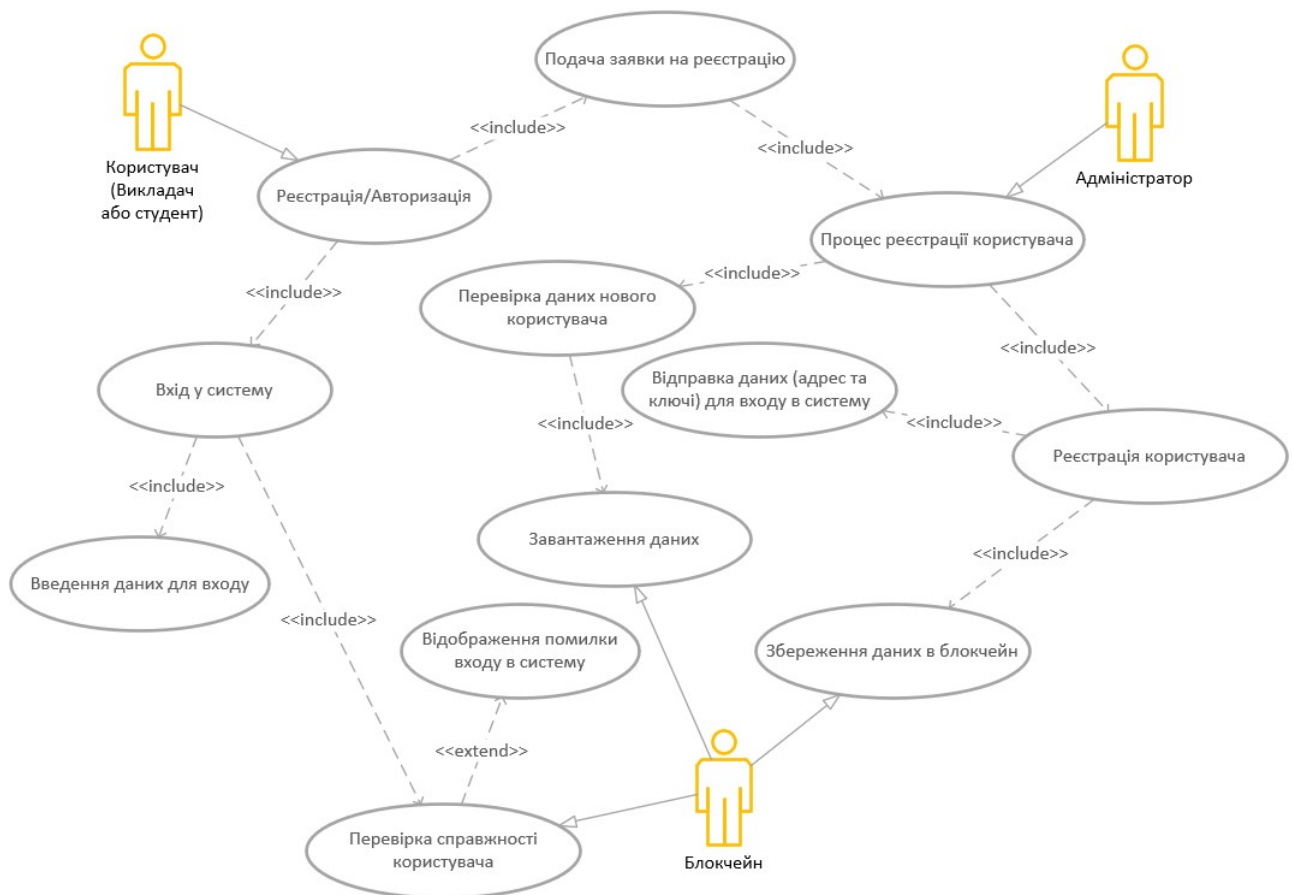


Рисунок 2.2 – Прецедент «Реєстрація та авторизація користувача»

Прецедент «Реєстрація та авторизація користувача»:

Основні актори: користувач, адміністратор.

Основний виконавець – користувач.

Зацікавлена особа – користувач (викладач і студент).

Передумови – наявність форми введення даних.

Постумова – користувач зареєстрований.

Потік подій:

Основний потік:

Крок 1. Незареєстрований користувач вводить особисті дані у формі подачі заявки і відправляє;

Крок 2. Адміністратор виконує процес реєстрації нового користувача:

- адміністратор отримує дані, введені користувачем, і перевіряє їх, в разі конкретності даних (новий користувач є студентом або викладачем освіти і відсутній в блокчейн) – перехід до наступного кроку, а в разі некоректності даних – перехід до кроку 4;

- адміністратор виконує реєстрацію нового користувача, в разі успішної реєстрації – перехід до наступного кроку, інакше – перехід до кроку 4;

- дані про нового користувача зберігаються в блокчейн;

- відправляються дані (адреса та ключі) користувачеві для входу в систему і подальшого використання.

Крок 3. Користувач отримує дані для аутентифікації і виконує вхід в систему:

- користувач вводить дані для входу в систему;

- блокчейн перевіряє справжність користувача, в разі успішного входу в систему – перехід до наступного кроку, інакше виконується альтернативний потік A1;

- користувач успішно увійшов в систему.

Крок 4. Кінець потоку.

Альтернативний потік:

A1. Система відображає помилку входу в систему, перехід до кроку 4 основного потоку.

Прецедент «Управління курсами» показаний на рис. 2.3 та описаний потік подій. У даній діаграмі описані базові операції користувачів для організації навчальних курсів.

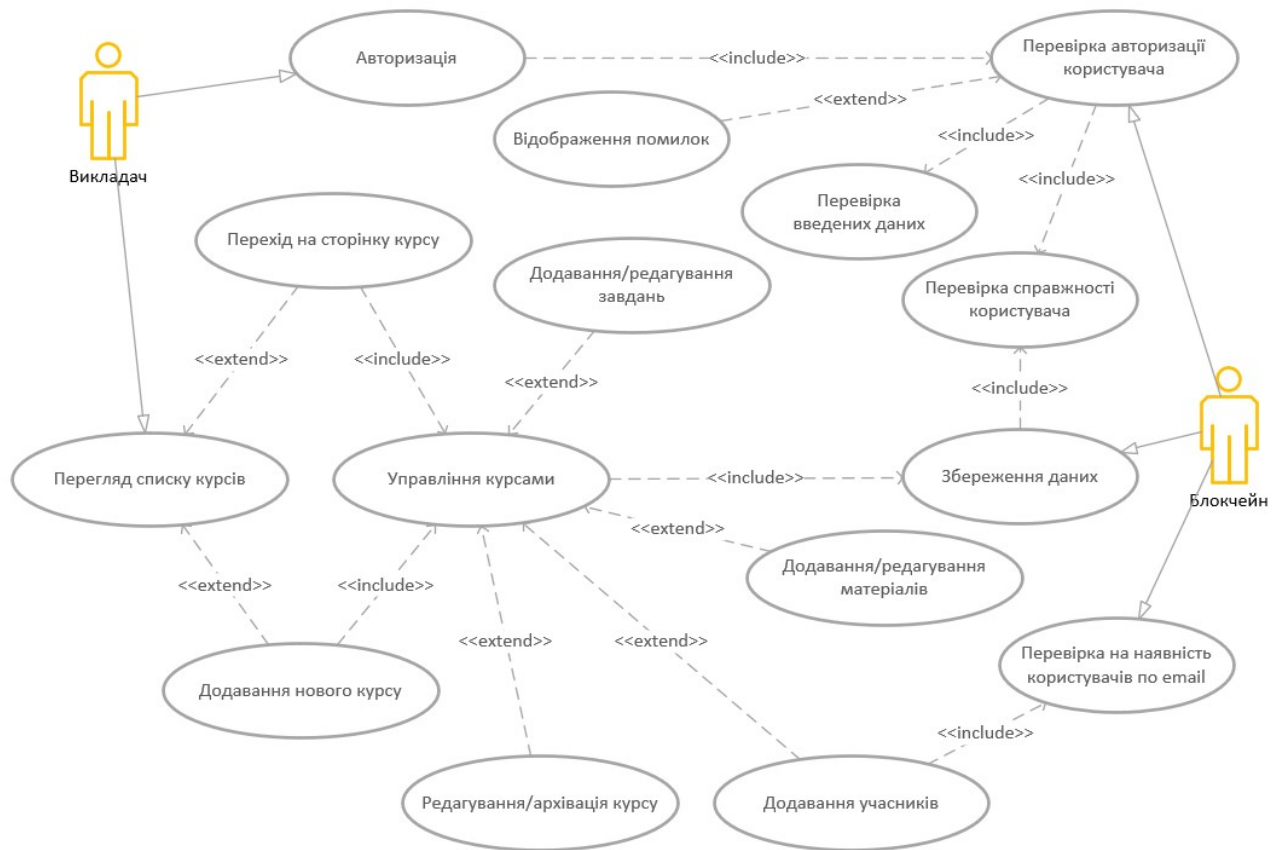


Рисунок 2.3 – Прецедент «Управління курсами»

Прецедент «Управління курсами»:

Основний виконавець – зареєстрований користувач.

Зацікавлена особа – користувач (викладач).

Передумови – наявність форми введення даних.

Постумова – доданий курс, додані завдання, матеріали та учасники до курсу.

Потік подій:

Основний потік:

Крок 1. Користувач виконує авторизацію;

Крок 2. Система перевіряє авторизацію:

- виконується перевірка введених даних;
- виконується перевірка справжності користувача;
- у разі невдалої авторизації виконується альтернативний потік A1.

Крок 3. Користувач входить на сторінці списку курсів;

Крок 4. Виконується альтернативний потік A2 або A3;

Крок 5. Користувач управляє курсами:

а) виконується альтернативний потік A4 або A5 або A6 або A7;

б) збереження даних:

- 1) система перевіряє справжність користувача;
- 2) змінені дані зберігаються в блокчейн.

Крок 6. Кінець потоку.

Альтернативні потоки:

A1. Система відображає помилку входу в систему, перехід до кроку 1 основного потоку;

A2. Користувач вибирає курс і переходить на його сторінку, перехід до кроку 5 основного потоку;

A3. Користувач додає новий курс, вводив деякі дані про курс, перехід до кроку 5 основного потоку;

A4. Користувач редагує деякі дані про курс або архівує курс, перехід до кроку 5 основного потоку;

A5. Користувач додає учасників в курс:

A5.1. Додавання викладачів по e-mail або студентів за номером групи;

A5.2. Система перевіряє наявність обраних користувачів в блокчейн, якщо є і не є учасниками обраного курсу, то виконується додавання нових учасників в курс, інакше скасовується додавання;

A5.3. Перехід до кроку 5 основного потоку.

A6. Користувач додає/редагує нові завдання, перехід до кроку 5 основного потоку;

A7. Користувач додає/редагує матеріали до курсу, перехід до кроку 5 основного потоку.

– у разі невдалої авторизації виконується альтернативний потік А1.

Крок 3. Студент обирає курс і завдання та переходить на його сторінку;

Крок 4. Відправка виконаних робіт:

а) студент відправляє свої роботи файлами;

б) шифрування документи публічним ключем курсу;

в) збереження даних:

1) система перевіряє справжність студента, в позитивному випадку – перехід до наступного кроку, в іншому випадку – перехід до кроку 1;

2) система зберігає дані про документи (хеш і найменування) і подія про здачу робіт в блокчейн.

Крок 5. Викладач переглядає журнал зданих робіт студентів;

Крок 6. Перегляд і перевірка роботи обраного студента:

– викладач завантажує роботу з блокчейна;

– розшифрування роботи студента приватним ключем курсу, який доступний тільки викладачам курсу, потім перевіряє роботу;

– виконується альтернативний потік А2 або А3.

Крок 7. Студент завантажує з блокчейна журнал своїх оцінок і переглядає їх;

Крок 8. Кінець потоку.

Альтернативні потоки:

А1. Система відображає помилку вході в систему, перехід до кроку 1 основного потоку;

А2. Залишення коментар:

А2.1. Користувач залишає коментар до виконаної роботи студента;

А2.2. Збереження даних:

А.2.2.1. Система перевіряє справжність викладача, в позитивному випадку – перехід до наступного кроку, в іншому випадку – перехід до кроку 1;

А.2.2.2. Система зберігає дані в блокчейн.

А2.3. Перехід до кроку 6.3 основного потоку.

А3. Виставлення оцінки:

А3.1. Користувач виставляє оцінку до виконаної роботи студента;

А3.2. Збереження даних:

А3.2.1. Система перевіряє справжність викладача, в позитивному випадку – перехід до наступного кроку, в іншому випадку – перехід до кроку 1;

А3.2.2. Система зберігає дані в блокчейн.

А3.3. Перехід до кроку 6.3 основного потоку.

2.1.3 Безпека. Діаграма послідовності

Оскільки застосунок буде децентралізованим і блокчейн в даній системі буде публічним, а збережені дані в блокчейн можуть бути доступними всім учасникам мережі. Тому для досягнення цілей безпеки використовуються криптографічні алгоритми в деяких випадках, таких як авторизація та аутентифікація користувача, шифрування особистих даних перед зберіганням в блокчейн і т.д.

У звичайних традиційних вебзастосунках для реєстрації і авторизації використовуються лише мінімум логін і пароль, які зберігаються на одному сервері в базі даних, логін є ідентифікатором користувача, а пароль є ключем для входу в систему. У децентралізованих застосунках реєстрація і авторизація користувачів виконуються по іншому. Паролі та інші особисті ключі не зберігаються в публічному блокчейні, тому для входу в систему користувач використовує свій приватний ключ, який зберігається в своєму надійному місці, наприклад на своєму пристрої. До доповнення можна використовувати свій пароль для шифрування і розшифрування свого приватного ключа, оскільки деякі користувачі можуть користуватися громадськими комп'ютерами, а в цих комп'ютерах можуть зберігатися приватні ключі кожного користувача. Крім того, пароль було б легше запам'ятати і простіше авторизуватися.

Ключі користувачів використовуються не тільки для авторизації, але і для шифрування і розшифрування деяких даних.

Щоб користувач мав свої параметри для доступу до системи, спочатку адміністратор отримує особисті дані, введені користувачем через форму подачі заявки на реєстрацію, і перевіряє, потім виконує процес реєстрації зі збереженням аккаунта в блокчейн. У процесі реєстрації генеруються три важливі параметри аккаунта – унікальний адрес, публічний та приватний ключі, і відправляються користувачеві для їх зберігання на своєму пристрої. Адрес аккаунту є ідентифікатором користувача в мережі блокчейн. А приватний і публічний ключі використовуються для аутентифікації та шифрування/розшифрування деяких даних перед зберіганням в блокчейн.

Аутентифікація буде відбуватися кожного разу при будь-якої дії користувача, яке вносить зміни в блокчейн, як показано на рис. 2.5. Аутентифікація виконується шляхом порівняння двох приватних ключів, зашифрованих хеш-алгоритмом SHA-256, де перший ключ є поточним приватним ключем користувача, який був витягнутий зі свого пристрою, а другий зашифрований приватний ключ, витягнутий з блокчейну по електронній пошті.

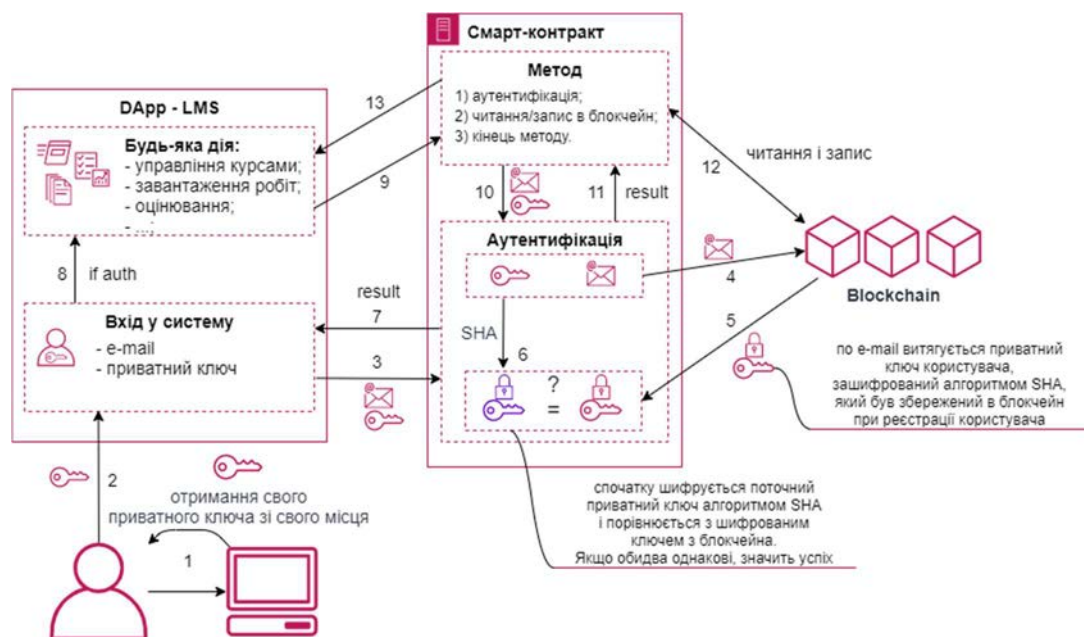


Рисунок 2.5 – Схема аутентифікації

На рис. 2.6 представлена діаграма послідовності додавання курсу і викладачів до курсу з генерацією публічних та приватних ключів курсу. Деяким користувачам потрібно захистити якісь свої дані в блокчейне, наприклад, студенти хочуть захистити свої роботи від читання іншими студентами, а надати доступ до своїх робіт студенти хочуть лише тільки викладачам обраного курсу. Тому для цього запропонована ідея – використання криптосистеми з відкритим ключем, де при створенні нового курсу генеруються приватний та публічний ключ для створеного курсу. Публічний ключ курсу буде зберігається в блокчейн і доступний всім учасникам мережі та використовується студентами для шифрування своїх робіт перед зберіганням в блокчейн. А приватний ключ курсу доступний тільки викладачам конкретного курсу та використовується для розшифрування робіт студентів конкретного курсу. При цьому роботи конкретного студента не можуть читати інші студенти або викладачі, які не беруть участі в конкретному курсі. Приватний ключ курсу також буде зберігатися в блокчейн, але зашифрованим приватним ключем викладача. В одному курсі можуть бути кілька викладачів, а у кожного викладача свій унікальний приватний ключ. Тому в блокчейн для кожного викладача одного курсу зберігається окремий приватний ключ курсу, зашифрований своїм приватним ключем. Так що викладач кожен раз, коли додає іншого викладача до свого курсу, розшифрує приватний ключ курсу і відправляє іншого викладача, щоб інший міг мати приватний ключ курсу і зашифрує своїм же приватним ключем, а потім зберігається в блокчейн.

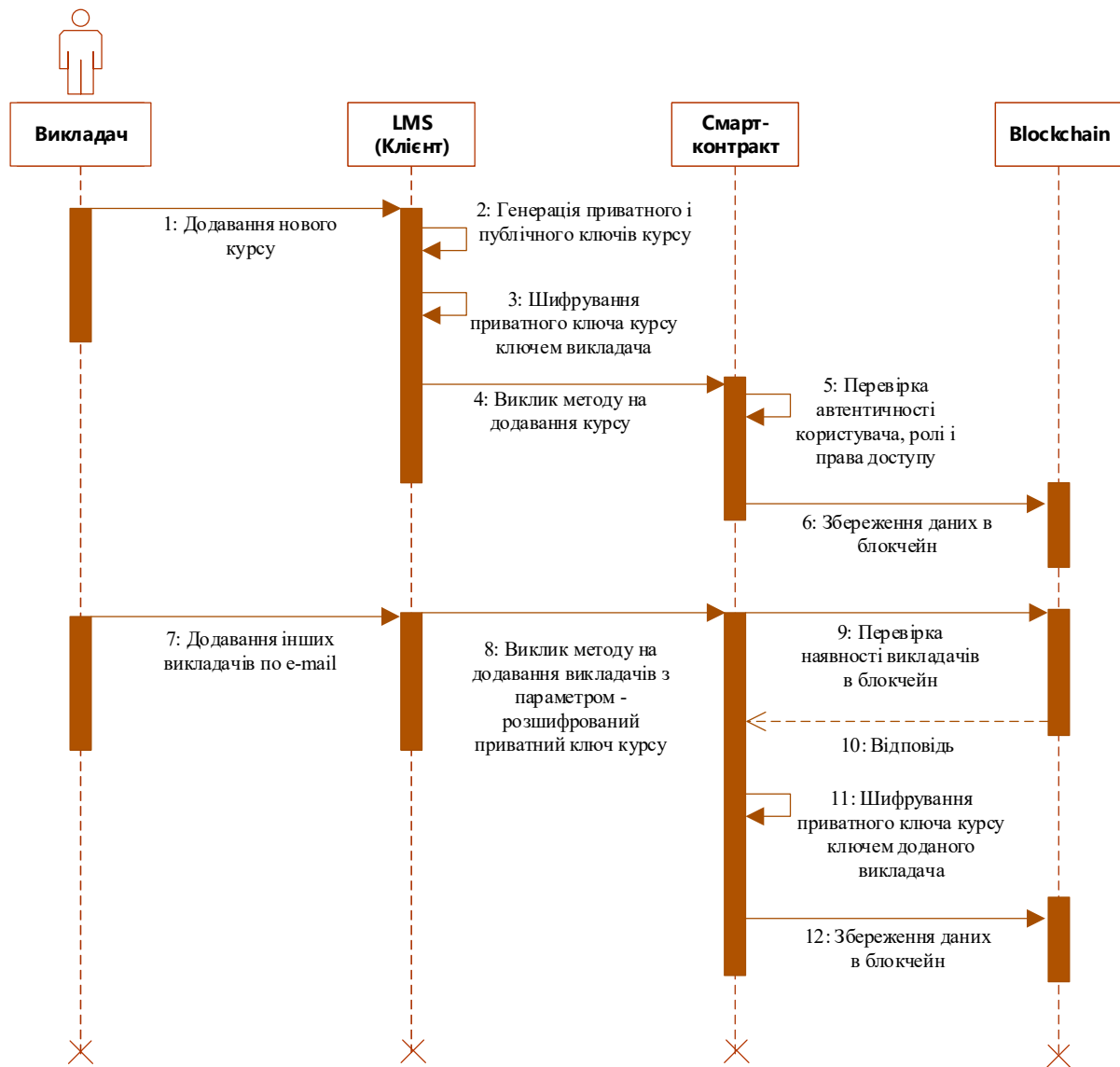


Рисунок 2.6 – Діаграма послідовності додавання курсу і викладачів до курсу з генерацією публічних та приватних ключів курсу

На рис. 2.7 представлена діаграма послідовності відправки робіт студентів і їх оцінювання викладачем з використанням ключів. У цій діаграмі описано, за яким принципом виконуються відправка робіт в блокчейн, скачування, перегляд і оцінювання робіт з метою збереження безпеки. Спочатку студент на обране завдання через форму введення даних завантажує свої роботи і система шифрує їх публічним ключем обраного курсу, потім перевіряється справжність і права доступу студента перш ніж зберегти роботи в блокчейн. Для ефективності та стабільної роботи блокчейна, в платформі блокчейн зберігається

лише хеш файлу малої довжини, який по суті є посиланням на повний файл в іншому сховищі даних. Всі повноцінні роботи зберігаються зашифрованими в окремому децентралізованому сховищі даних, яке за принципом є блокчейном, але призначене тільки для зберігання даних будь-якого розміру на кожному вузлі мережі без третьої особи. У нашому випадку в якості децентралізованого сховища даних буде використаний InterPlanetary File System (IPFS).

Викладачі через якийсь час отримують роботи студентів для перевірки і оцінювання. Спочатку викликається метод на отримання хешу файлів, що зберігаються в блокчейн, потім по хешам викачуються повні зашифровані файли з децентралізованого сховища даних (IPFS в нашому випадку). Після чого система розшифрує завантажувані файли приватним ключем курсу та викладачі мають можливість переглядати і оцінити роботи. При оцінюванні викладачем спочатку СК перевіряє справжність, роль і права доступу викладача, потім зберігаються результати і події дій в блокчейн.

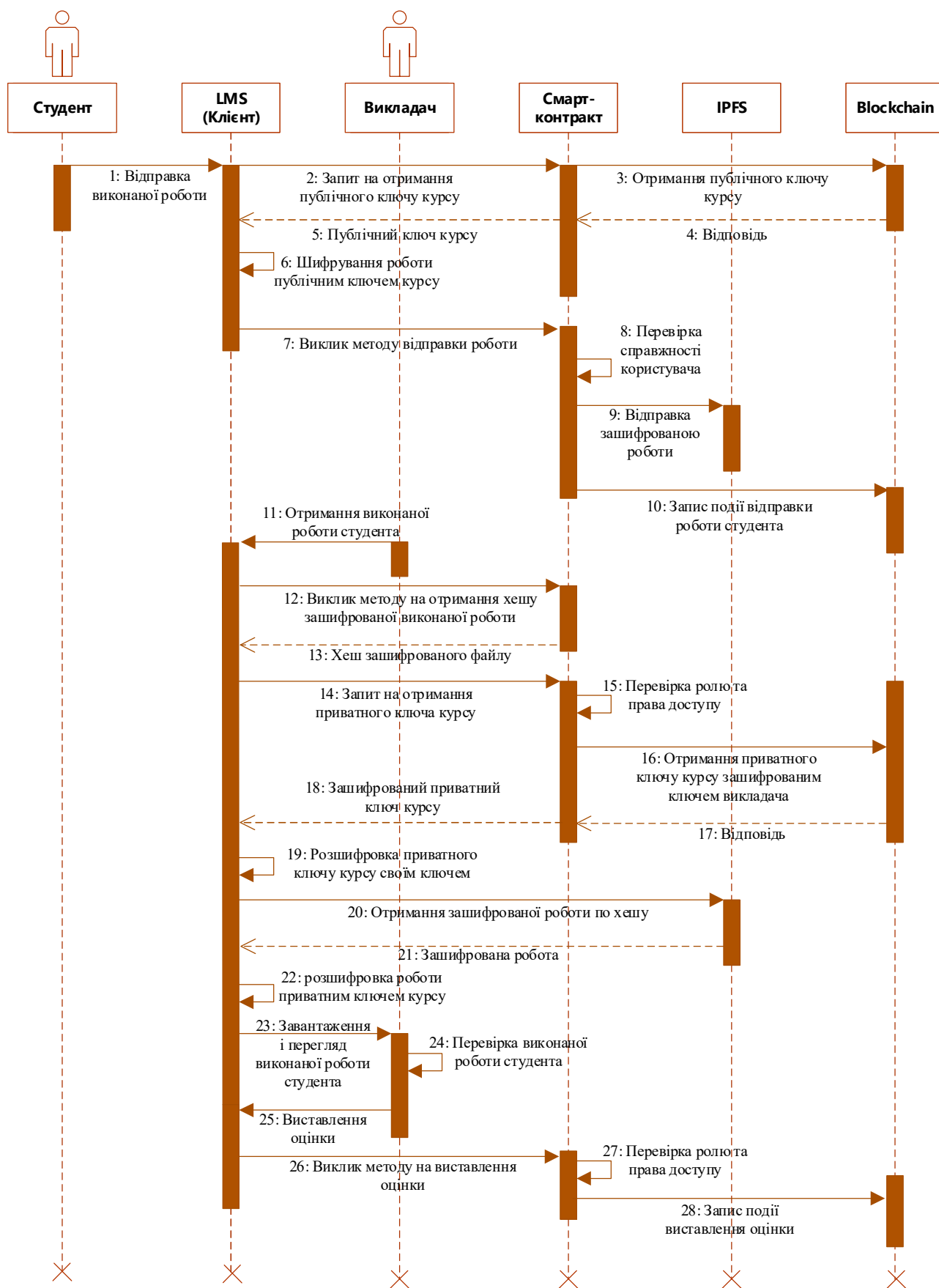


Рисунок 2.7 – Діаграма послідовності основного процесу

2.1.4 Діаграма об'єктів смарт-контракту

На рис. 2.8 зображена діаграма об'єктів, яка демонструє загальну структуру ієрархії об'єктів СК, їх атрибутів, методів, подій і взаємодій між ними. СК по структурі схожий на базу даних, але СК є важливою частиною мережі блокчейна, яка перевіряє і виконує всі дії, бажані користувачем, і організовує зберігання даних в блокчейн. Далі докладно описані деталі СК.

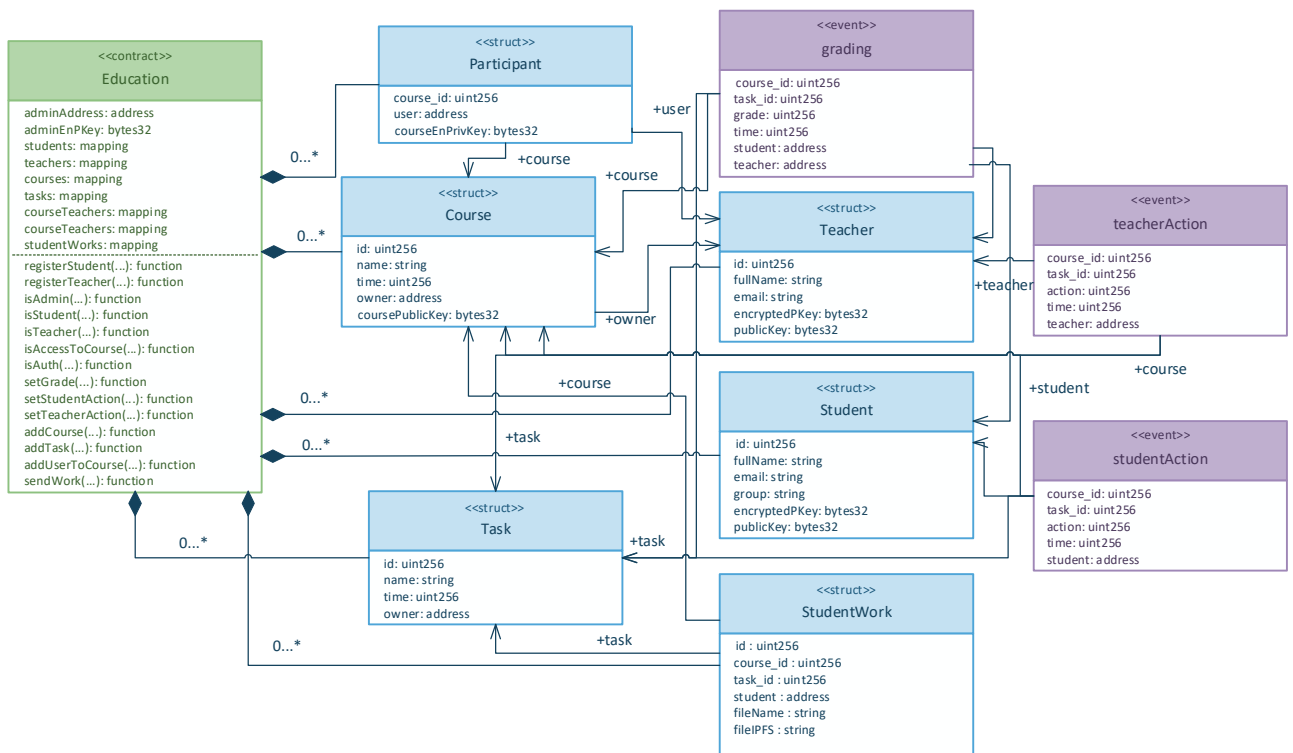


Рисунок 2.8 - Діаграма об'єктів

Для даної роботи достатній один СК Education і містить наступні об'єкти:

– Course. Основний клас «Курс», який може бути ініціалізований при створенні викладачем у формі створення нового курсу. У цьому курсі зберігаються деякі дані, такі як ім'я, час створення, власник курсу, а також публічний ключ курсу. Зазвичай при створенні курсу генеруються два ключі – закритий і відкритий ключі курсу. Публічний ключ курсу доступний всім і використовується студентами для шифрування своїх виконаних робіт. А приватний ключ курсу доступний тільки всім викладачам даного курсу, щоб ті

з змогли розшифрувати і перевірити роботи студентів. Тому приватний ключ буде зашифрований ключем викладача і зберігається в блокчейн;

- Task. Основний клас «Завдання», який є частиною курсу і може бути ініціалізований при створенні викладачем у формі створення. У цій задачі зберігаються дані, схожі в класі «Курс»;

- Student. Основний клас «Студент», який зберігає всі особисті дані про студента, який зареєстрований адміністратором у системі управління навчанням. При реєстрації студента генеруються приватний і закритий ключ, публічний ключ зберігається в блокчейн і доступний всім, а приватний ключ в блокчейн зберігається тільки зашифрованим без можливості розшифрування. Приватний ключ використовується для аутентифікації користувача;

- Teacher. Основний клас «Викладач», який зберігає всі особисті дані про викладача, який зареєстрований адміністратором у системі управління навчанням. При реєстрації викладача також генеруються і зберігаються обидва ключа за тим же принципом, що і при реєстрації студента;

- Participant. Клас «Учасник», який містить три елементи – ідентифікатор курсу і адреса користувача (викладача або студента), а також приватний ключ даного курсу, зашифрований ключем викладача. Для студентів приватний ключ курсу не використовується, тому залишається порожнім;

- StudentWork. Клас «Робота студента», який ініціалізований при здачі виконаної роботи студентом. Даний клас містить ідентифікатор курсу і завдання, адреса студента, який здав, і хеш файлу (виконаної роботи), який є посиланням на повний файл в децентралізованому файловому сховищі;

- Grading. Клас «Оцінка», який для СК вважається подією. Дана подія додається в блокчейн при виставленні оцінки викладачем в формі. Він містить в собі ідентифікатор курсу, завдання, оцінку, адреса викладача, який виставив оцінку, і студента, якому поставили оцінку;

- teacherAction. Подія викладача, яке автоматично додається при різних діях викладача, таких як створення курсу, завдання, додавання користувачів до курсу, а також додавання матеріалів і т.д.;

– teacherStudent. Подія студента, схоже на подію викладача і додається тільки при дій студента.

Події в СК використовуються для зручного і швидкого перегляду журналу дій користувачів і оцінок, яких неможливо змінити.

СК також містить різні методи, які викликаються користувачем і системою. У нашому СК будуть реалізовані наступні методи:

- реєстрація користувачів (викладача, студента). Даний метод має доступ лише адміністратору системи;
 - перевірка достовірності користувача. Метод перевіряє, чи є користувач авторизованим;
 - перевірка ролі користувача. Даний метод важливий, тому що інтерфейс клієнтської програми буде залежати від ролі авторизованого користувача. Ролі в цій системі можуть бути різними, такі як адміністратор, викладач і студент;
 - перевірка доступу до курсу. Перевіряє, чи є користувач учасником курсу;
 - виставлення оцінки викладачем;
 - додавання нового курсу і завдання;
 - додавання користувачів до курсу;
 - здача виконаної роботи студентом.
- І найважливіше, СК містить адреси адміністратора і його зашифровані ключі.

2.2 Дизайн децентралізованого застосунку

Для роботи підсистеми управління навчанням з використанням технології блокчейн потрібен для користувача простий інтерфейс, який повинен представляти всі необхідні дані і управління зацікавленим користувачам. Інтерфейс повинен бути максимально простий у використанні. Інтерфейс ще потрібен для швидкого управління і тестування роботи блокчейна замість використання командного рядку.

На рис. 2.9 представлений ескіз сторінки входу в систему. При запуску програми спочатку перевіряється увійшов чи користувач в систему, якщо ні, то запускається форма входу в систему. У користувача є можливість увійти в систему лише заповнивши 2 поля – e-mail та пароль, за допомогою якого розшифрують приватний ключ користувача для входу. Після чого децентралізований застосунок (DApp) тимчасово зберігає адрес та приватний ключ користувача, який успішно увійшов в систему, для подальшого використання. Крім того, для зручної і швидкої розробки даної підсистеми додатково буде розміщений список зареєстрованих користувачів з кнопками для швидкого входу в систему.

The sketch shows a web browser window titled "A Web Page". The address bar contains "https://". The page content includes a user profile box at the top with a person icon, "Full name", "Role: Unauthorized user", and "Address: 0x0...". Below this is a "Sign in" section with two input fields labeled "e-mail" and "password (or private key)", and a "Sign in" button. At the bottom, there is a "Quick sing in (for testing)" section with two user entries, each showing a person icon, "Full name", "Role", "Address: 0x0...", and a "Sign in" button.

Рисунок 2.9 – Ескіз сторінки входу в систему

На рис. 2.10 представлений ескіз сторінки реєстрації користувачів (студента або викладача). Дана сторінка доступна тільки адміністратору системи. Його користувачі не можуть безпосередньо зареєструватися з деяких причин, вказаних в попередньому підрозділі. Тому користувачі, які хочуть бути зареєстрованим в системі, заповнюють форму подачі заявки адміністратору через інший веб-сервер або через e-mail. Після чого адміністратор перевірить

дані введені користувачем і виконає реєстрацію. При реєстрації будуть згенеровані унікальні приватний і публічний ключі користувача і його адресу. Як тільки реєстрація пройде успішно, дані автоматично відправляться користувачеві по e-mail.

A Web Page

https://

Full name
Admin
Address: 0x0345346gt5t3gerg345

Exit

Register

Teacher register

full name

e-mail

Sign in

address:
public key:
private key:

Student register

full name

e-mail

group

Sign in

address:
public key:
private key:

Рисунок 2.10 – Ескіз сторінки реєстрації користувачів

На рис. 2.11 представлений ескіз сторінки списку курсів. У списку включаються тільки ті курси, в яких бере участь поточний користувач. Для викладача доступна кнопка «Додати новий курс». Також є можливість відфільтрувати список.

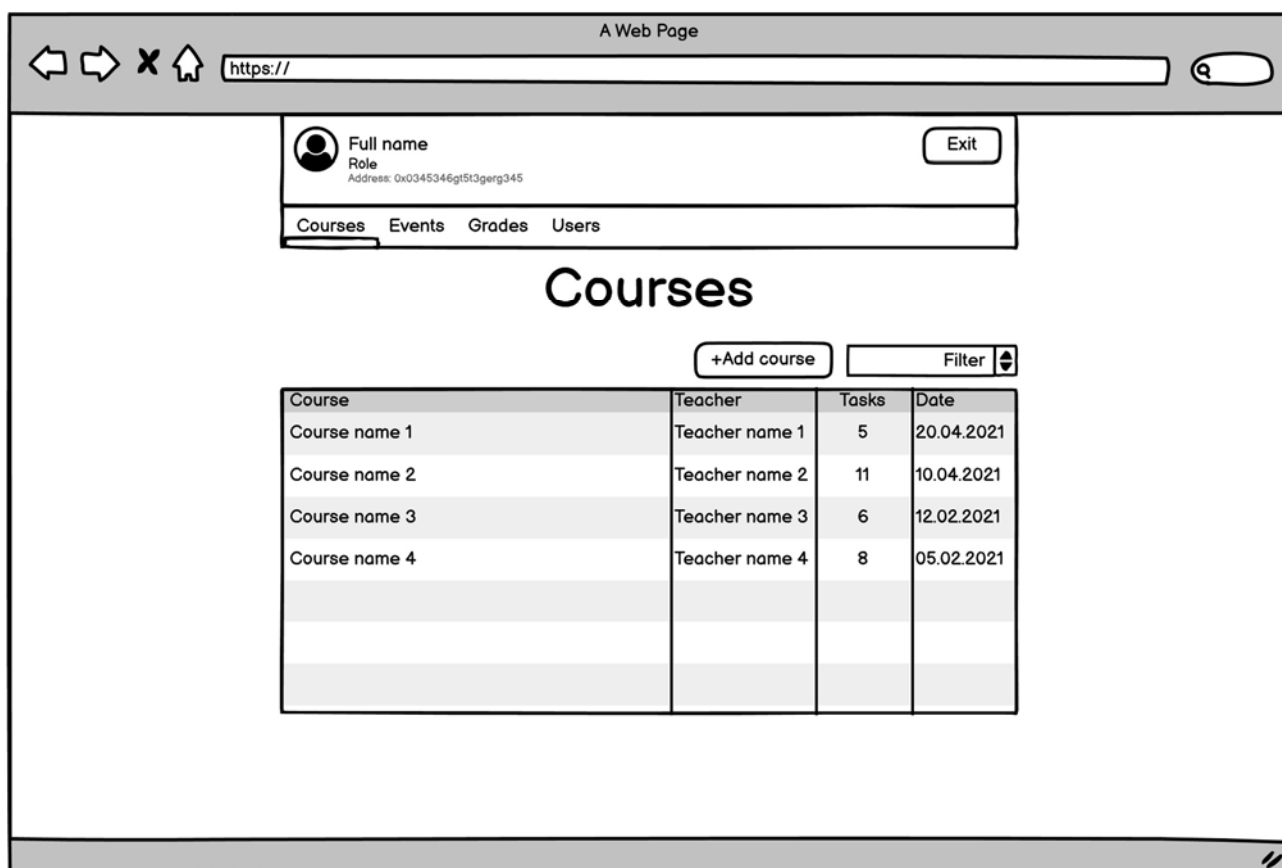


Рисунок 2.11 – Ескіз сторінки списку курсів

На рис. 2.12 представлений ескіз сторінки обраного курсу. На цій сторінці зліва буде відображений список завдань з можливістю для викладачів додати нове завдання, а праворуч будуть відображені два списки – список викладачів і список студентів, які беруть участь в обраному курсі. Тільки викладачам доступні кнопки додавання користувачів до курсу. Крім того, на сторінці є 2 вкладки – список завдань і журнал оцінок студентів. Журнал оцінок студентів доступний тільки викладачам.

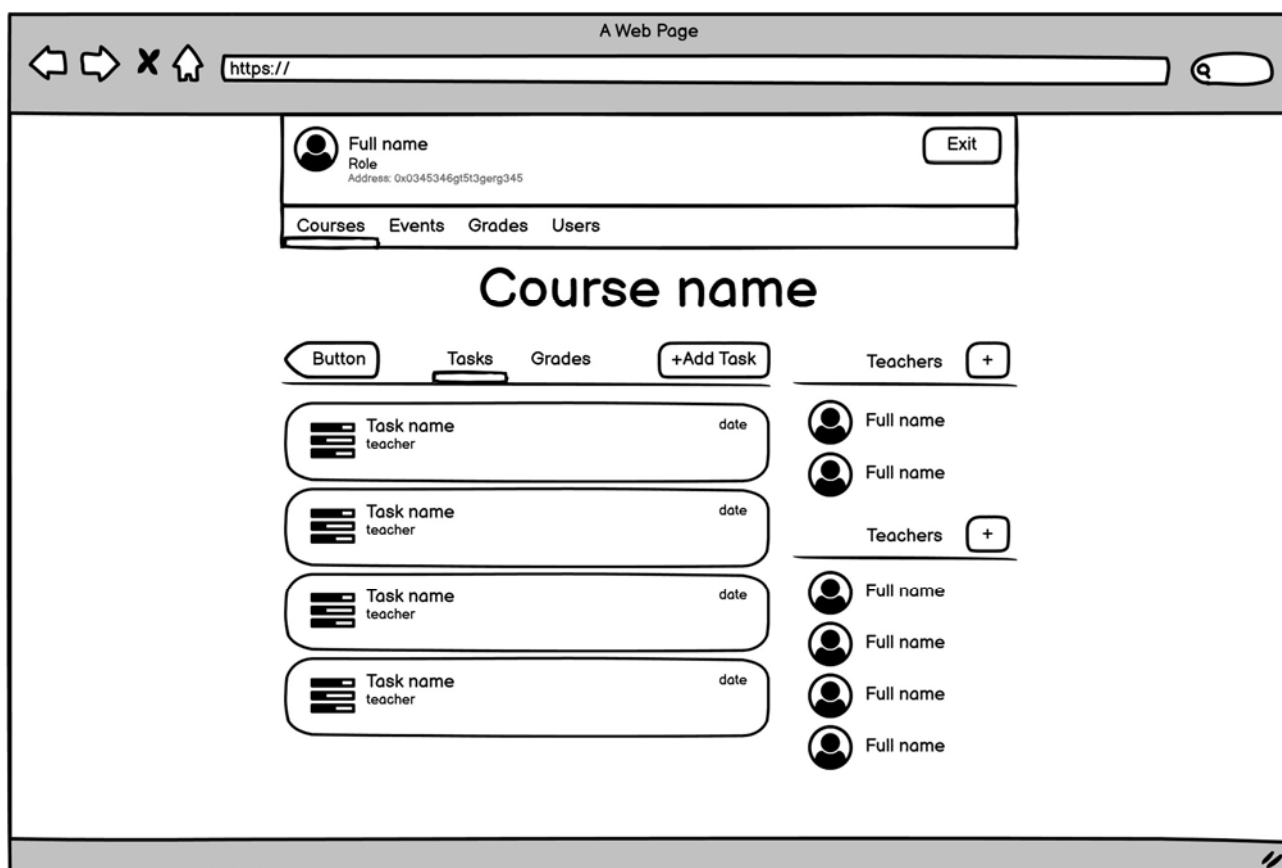


Рисунок 2.12 – Ескіз сторінки курсу

На рис. 2.13 представлений ескіз сторінки обраного завдання для студента. На цій сторінці студент може лише завантажити виконані роботи і здати, а також подивитися оцінку і коментарі.

На рис. 2.14 представлений ескіз сторінки обраного завдання для викладача. Дана сторінка схожа на сторінку для студента, але відрізняється видом в формі перегляду робіт і виставлення оцінки. Також викладачам доступна вкладка «Список робіт студентів», на цій вкладці викладач може швидко переглядати і перевірити виконані роботи студентів на конкретне завдання.

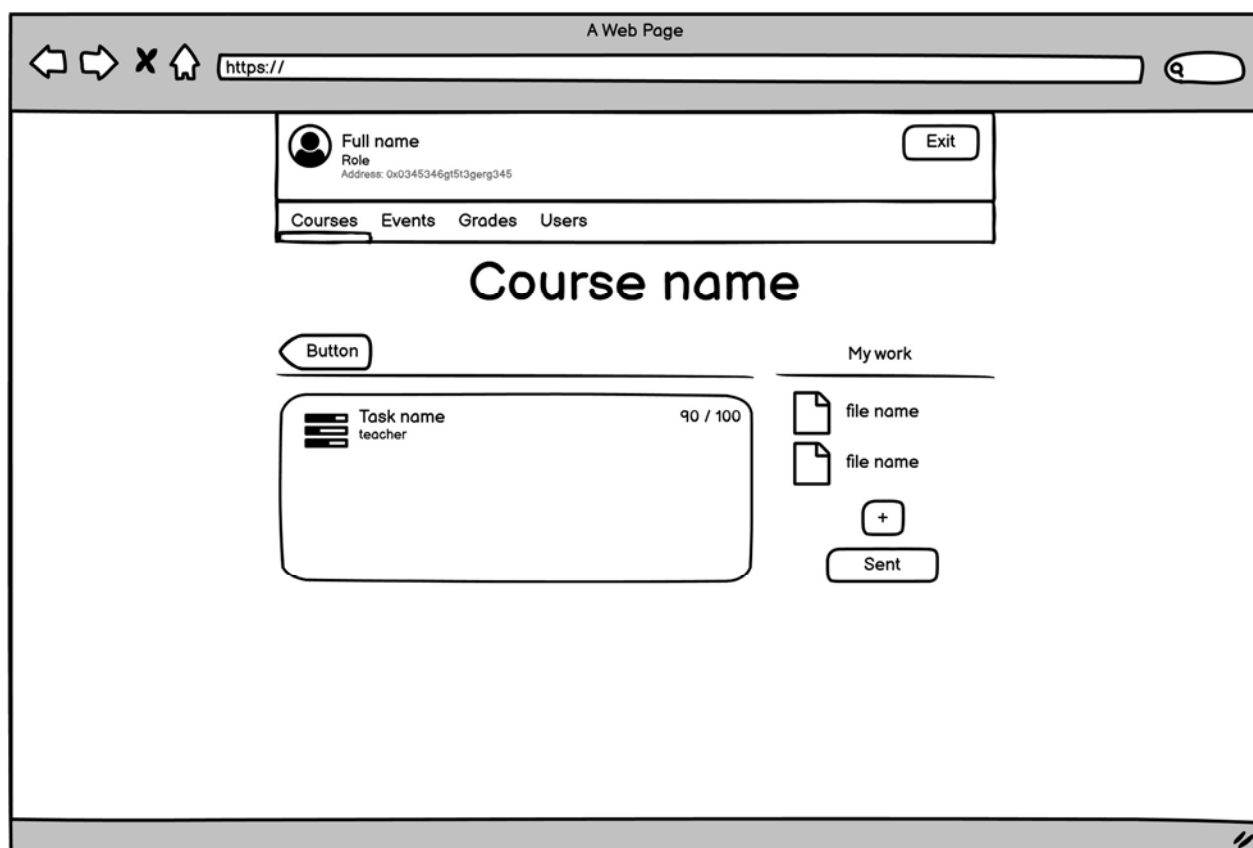


Рисунок 2.13 – Ескіз сторінки завдання для студентів

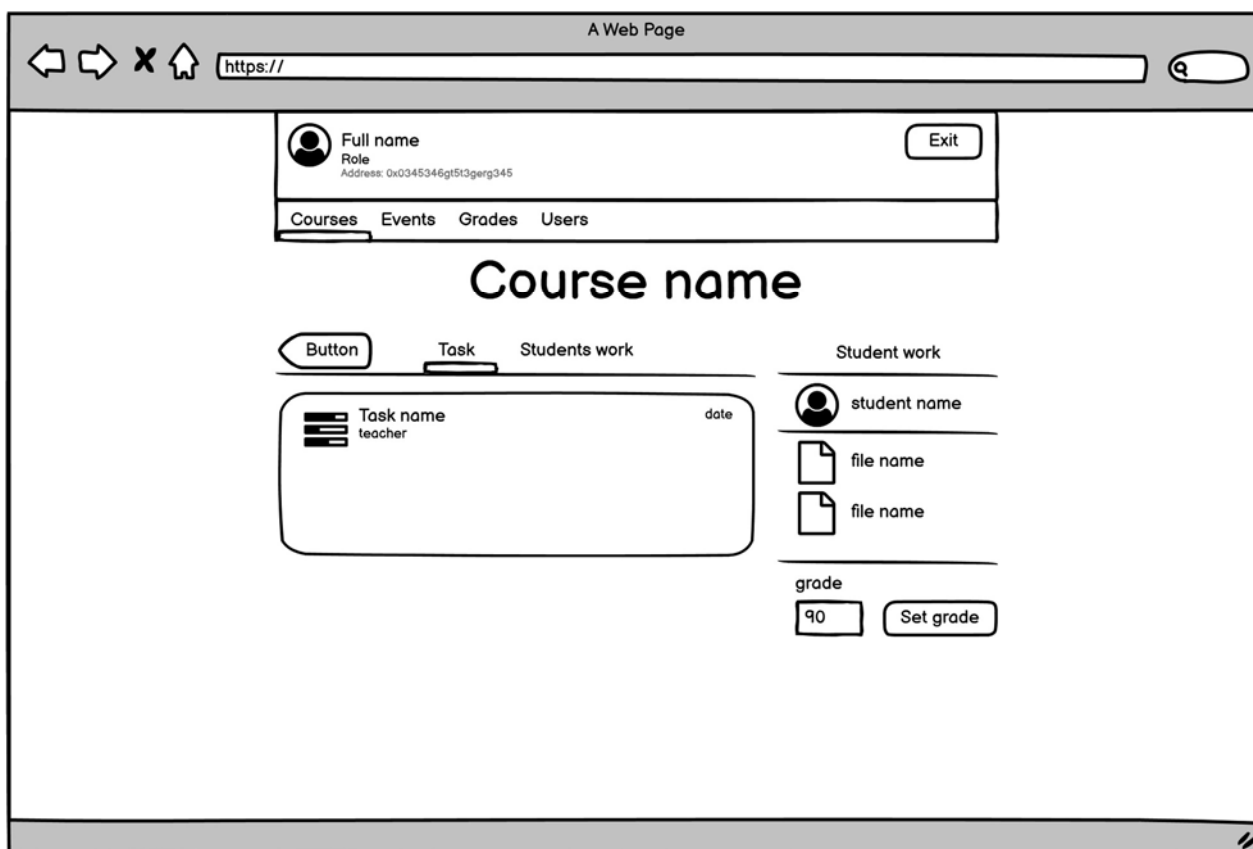


Рисунок 2.14 – Ескіз сторінки завдання для викладачів

На рис. 2.15 представлений ескіз сторінки журналу оцінок студентів конкретного курсу. Ця сторінка доступна тільки викладачам.

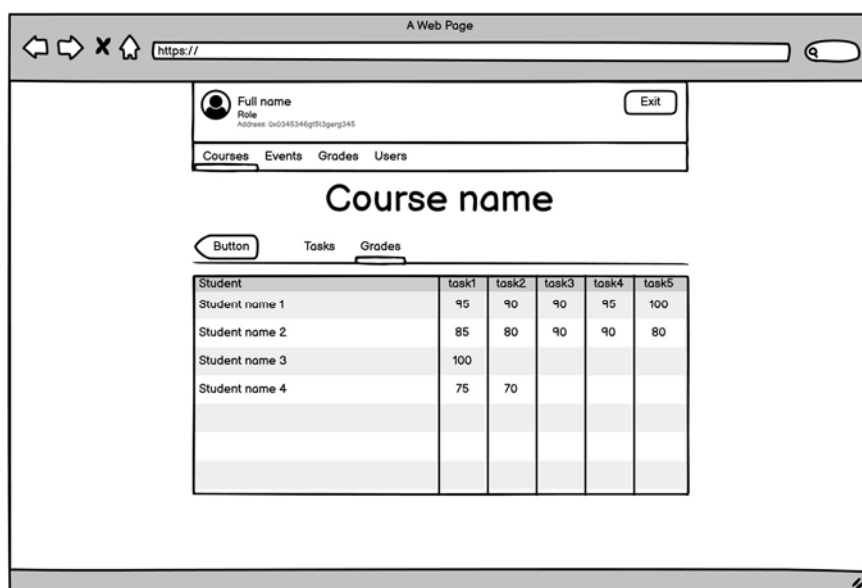


Рисунок 2.15 – Ескіз сторінки журналу оцінок студентів конкретного курсу (для викладачів)

На рис. 2.16 представлений ескіз сторінки журналу дій. На цій сторінці будь-який зареєстрований користувач може побачити список своїх дій з датами, а також дій інших викладачів своїх курсів.

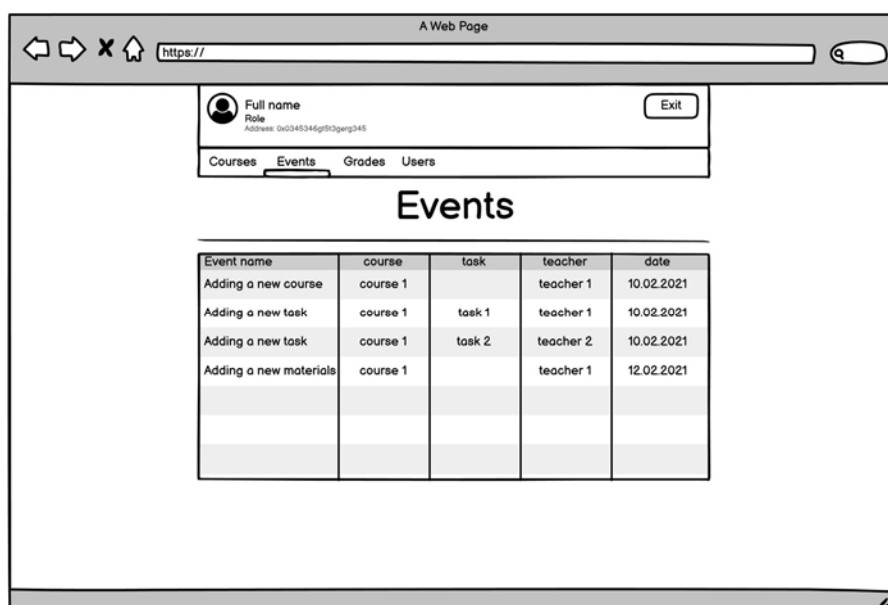


Рисунок 2.16 – Ескіз сторінки журналу дій

На рис. 2.17 представлений ескіз сторінки журналу своїх оцінок. Дана сторінка доступна тільки студентам для перегляду своїх оцінок з усіх курсів, в яких брав участь.

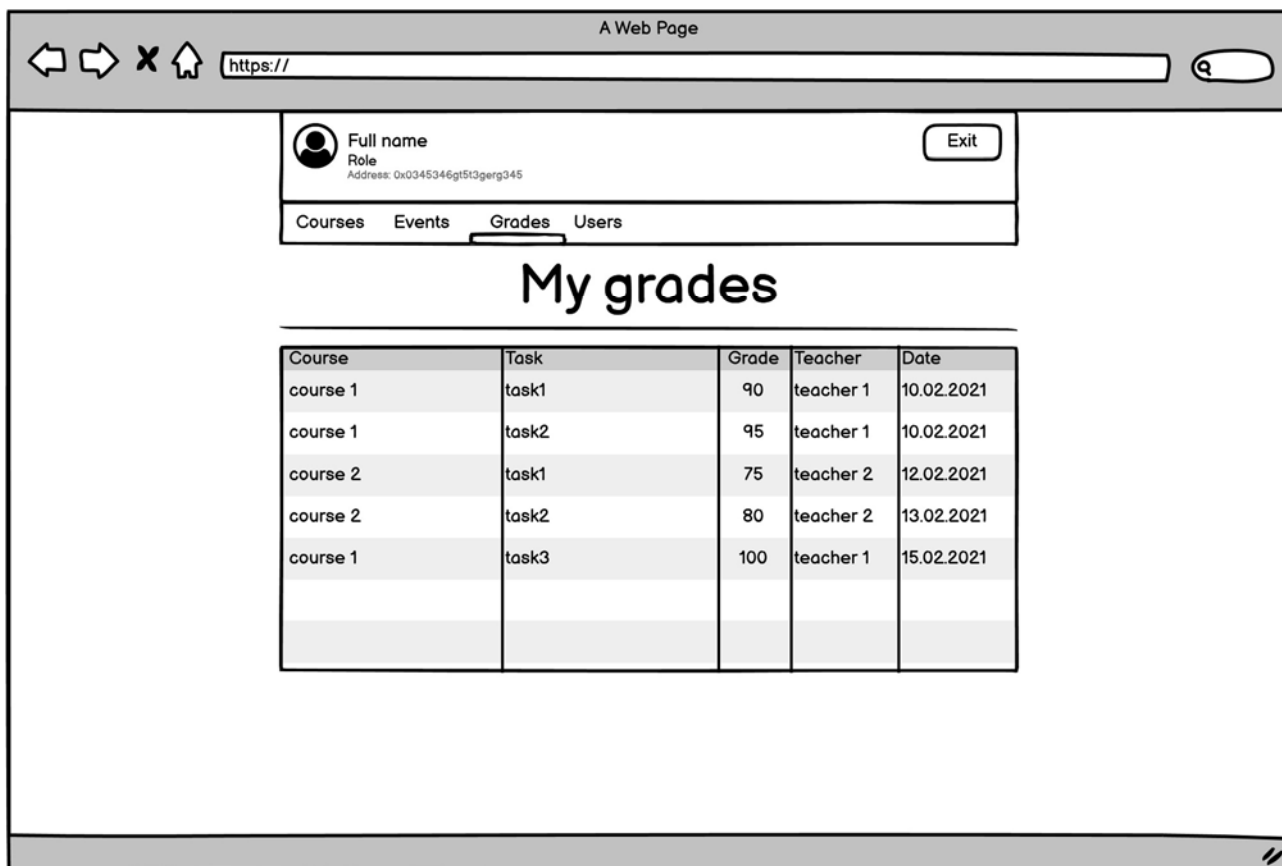


Рисунок 2.17 – Ескіз сторінки журналу своїх оцінок (для студентів)

2.3 Вибір типу блокчейн та платформи блокчейн

2.3.1 Обґрунтування вибору типу блокчейн

На даний момент існують три основні типи блокчейнів: публічні, гібридні та приватні мережі. Кожен тип вибирається залежно від поставлених завдань. У таблиці 2.1 перераховані основні завдання, які можуть бути вирішені за рахунок застосування блокчейн-технології [7].

Нагадаємо, що публічні мережі більш децентралізовані і будь-хто може взяти участь в їх роботі на будь-якому рівні – підтримувати роботу повного вузла, створити блоки, а також записувати нові дані. Такі мережі краще

забезпечують свій захист і незмінюваність, ніж приватні та гібридні мережі. Приватні мережі спільно використовуються тільки довіреними сторонами і можуть бути повністю невидимими для широкої публіки. А гібридні мережі відкриті для широкої публіки, але участь в їх роботі контролюється.

Для нашого випадку найбільше підходить публічний блокчейн, в якому інформації можуть зберігати на всіх вузлах учасників мережі. Деякі збережені інформації можуть бути незашифрованими і доступні всім для читання, та інші інформації – зашифрованими і доступні тільки тим, у кого є ключі для розшифровки. У нашому публічному блокчейну для повного автоматичного контролювання інформацій буде використаний СК.

Таблиця 2.1 – Завдання, рішення яких забезпечує блокчейн-технологія

Основна ціль	Тип блокчейн
Переміщення цінностей між сторонами, які не є довіреними	Публічний
Переміщення цінностей між довіреними сторонами	Приватний
Створення децентралізованих організацій	Публічний або гібридний
Створення децентралізованих контрактів	Публічний або гібридний
Забезпечення ідентифікації для людей або речей	Публічний
Запис інформації для відкритого обліку	Публічний
Запис інформації для закритого обліку	Публічний або гібридний
Забезпечення аудиту записів або систем	Публічний або гібридний
Запис інформації про справжність сертифікатів і дипломів	Публічний
Організація захисту систем	Публічний

2.3.2 Огляд існуючих платформ блокчейн та вибір

До приватного типу блокчейна відноситься найбільш популярна платформа Hyperledger Fabric. А до публічного типу блокчейна відносяться більшість платформ, такі як Bitcoin, Ethereum, EOS і т.д.

Bitcoin є платформою блокчейн, але сфера застосування й функціонал більш обмежені. Дана платформа використовується виключно в області криптовалют і краще справляється з функціями платіжного засобу. У Bitcoin СК гарантує тільки проведення платіжних операцій. Тому Bitcoin не підходить для нашої мети і не розглядається.

2.3.2.1 Hyperledger Fabric

Hyperledger Fabric є фреймворк для розробки розподілених реєстрів на основі блокчейн. Він пропонує розширені засоби забезпечення конфіденційності – доступом до загальних даних мають тільки дозволені (заздалегідь відомі) учасники мережі. Основною частиною платформи, за допомогою якої може самостійно виконувати угоди між сторонами, є СК, які описують бізнес-процеси і представляються у вигляді рядків коду. Відслідковують і незворотні транзакції зміцнюють довіру між організаціями [7, 49].

Переваги:

- контрольована мережа. Формування децентралізованої довіри в мережі з відомими учасниками, а не у відкритій мережі з анонімними користувачами;
- конфіденційні транзакції. Надання доступу тільки до тих даних, якими є бажання поділитися з конкретними учасниками;
- простий початок роботи. Для програмування СК можна використовувати заздалегідь відомі мови програмування без необхідності вивчення.

Недоліки:

- відносно невелике ком'юніті;

– високий поріг входження.

2.3.2.2 Ethereum

Платформа Ethereum розроблена командою Віталіка Бутеріна в 2015 році для реалізації DApps на базі блокчейн з використанням СК. Ethereum – це одночасно і криптовалюта, і функціональне децентралізоване середовище, яке по-справжньому революціонізувало всю ІТ-сферу. Ethereum дав абсолютно нові можливості по створенню децентралізованих проєктів і стартапів на основі блокчейн і розумних контрактів, тому можна застосовувати в самих різних сферах [50].

У платформі використовується алгоритм PoW – доказ роботи, що вимагає обчислювальні потужності учасників для безпечного функціонування мережі. Але в найближчий майбутньому буде використаний алгоритм PoS – доказ частки володіння, в якому не потрібні ніяких обчислювальних потужностей [51].

Згідно статистику на сайті State of the DApps [52] в даний час на блокчейн Ethereum більше 2700 dApp-застосунків і щодня активними користувачами цих застосунків є близько 90 000 чоловік. Блокчейн Ethereum містить популярні застосунки, такі як Storj, IDEX і Augur. Серед популярних категорій – ігри, обмінники, гаманці, зберігання, фінанси, управління, енергетика, соціальні мережі, медіа, ідентифікатори, безпека і страхування вантажу.

Переваги:

- універсальність. Підтримка різних мов програмування і різних за складністю алгоритмів;
- широке застосування. СК мають розширені можливості і застосовуються в різних сферах;
- гнучкість. Платформа відкрита до поліпшень, модернізацій, оновлень;
- доступність. Ethereum є відкритою платформою для розробки DApp в різних областях;

- поліпшення. Ethereum переходить на модель PoS, яка дозволяє підвищити ефективність мережі в багато разів;
- швидкість. Формування блоків мають високу швидкість.

Недоліки:

- недостатність документаций. На даний момент відсутні чіткі документації, що ускладнює взаємодію з платформою. Але з часом з'являється більше потрібної інформації.

2.3.2.3 EOS

Платформа EOS була розроблена компанією Block.one під керівництвом Данієла Ларімер і Брандао Данієла, і була запущена в 2018 році. Платформа EOS також призначена для розробки DApp з використанням СК. Особливістю EOS є надання обчислювальних ресурсів для розробників і можливість роботи на основі Web Assembly: Binaryen і WAVM. СК в EOS можуть розподіляти квоти обчислювальних ресурсів мережі шляхом ставок на токени, ринковими комісіями або токен голосуванням [53, 54].

Згідно статистику на сайті State of the DApps [52] в даний час на блокчейн EOS близько 330 dApp-застосунків і щодня активними користувачами цих застосунків є близько 36 000 чоловік. Найбільш популярні категорії DApp на EOS включають звичайні і азартні ігри.

Переваги:

- широке застосування. Платформа EOS надає рішення для планування, аутентифікації і створення інтернет-застосунків;
- високий рівень масштабованості. В EOS використовується алгоритм PoS, що дозволяє обробляти декілька мільйонів транзакцій в секунду;
- гнучкість. Завдяки PoS виключається ймовірність хардфорка мережі через хакерську атаку.

Недоліки:

– слабка децентралізація платформи. Цей недолік відштовхує багатьох розробників від її використання.

2.3.2.4 Обґрунтування вибору платформи блокчейн. Децентралізована файлова система IPFS

Наше DApp буде реалізовано на платформі Ethereum на базі технології блокчейн з використанням СК. Оскільки Ethereum є найбільш популярною платформою на сьогоднішній день і за рамками криптовалют. У Ethereum обмежена швидкість транзакцій – 20 транзакцій в секунду, а блоки створюються за 10-15 сек. Але це цілком достатньо для сфери освіти, оскільки учасників мережі буде небагато. СК для Ethereum – програма, яка виконується в мережі Ethereum і працює виключно так, як запрограмовано, без ризику простою, шахрайства та втручання третьої сторони. Так що завдяки СК є можливість розробити DApp для більшості цілей в різних сферах, в тому числі і в освіті.

У платформі Ethereum блокчейн є публічним, тому кожен учасник може мати доступ до даних, що зберігаються в блокчейн, для читання. Але деякі збережені дані можуть бути зашифровані користувачем для надання доступу лише деяким учасникам, які мають ключі для розшифровки. Запис можуть виконувати тільки довірені люди в мережі, наприклад зареєстровані користувачі адміністратором. Користувачі в DApp можуть виконувати різні функції для отримання і зберігання даних в блокчейн, все це може контролювати сам СК.

У DApp в блокчейн можна зберігати тільки дані малого розміру, як і в традиційних базах даних, а для зберігання великих файлів (зображення, документи, архіви і т.д.), наприклад, в нашому випадку викладачі будуть вивантажувати якісь матеріали, а студенти – свої роботи (наукові статті, лабораторні, курсові та дипломи), буде використаний IPFS.

IPFS – це децентралізована файлова система, яка використовує такий же протокол, як BitTorrent, щоб розподіляти дані по мережі. В основу IPFS закладені принципи DHT (distributed hash table, розподілена хеш-таблиця) і Merkle DAG

(directed acyclic graph, спрямований ациклічний граф) [55]. У IPFS окремий вузол не зберігає всі файли. Він містить тільки ті файли, які йому потрібні. Таким чином, якщо файл не популярний, його зберігає невелику кількість вузлів. Наприклад, при завантаженні файлу студентом через IPFS, даний файл зберігається на своєму вузлі і на вузлах тих викладачів певного курсу, яким студент здає свої роботи. А також при завантаженні матеріалів викладачем, ці матеріали будуть зберігатися ширше – на вузлах всіх учасників певного курсу, кому це буде необхідно.

2.4 Вибір мови програмування

У нашому випадку взаємодія користувача з децентралізованим застосунком буде відбуватися у браузері, тому клієнтська частина цього застосунку буде розроблена на простих мовах програмування JavaScript, HTML і CSS. Їх досить для відображення потрібних сторінок, виконання всіх необхідних функцій даної підсистеми управління навчанням, а також взаємодії з блокчейн мережею.

JavaScript – це мова програмування, який дає можливість реалізувати складну поведінку веб-сторінки, наприклад динамічно оновлювати контент сторінки, виконувати функції, що викликаються користувачем через інтерфейс, а також взаємодіяти з СК для зміни і зберігання даних в блокчейн Ethereum.

HTML – це мова розмітки, який використовується для структурування і відображення веб-сторінки і її контенту.

CSS – це мова стилів, за допомогою якого можна встановити стиль відображення веб-сторінок.

СК Ethereum розробляються на одному з декількох мов, спроектованих для трансляції в байт-код віртуальної машини Ethereum – Solidity, Vyper, Serpent і LLL, а також Mutan. Кожен з цих мов схожий на більш відомих мовах програмування, наприклад Solidity схожий на C/C++ і JavaScript, мова Serpent

схожий на високорівневу мову Python, а Mutan заснований на Go (golang), розроблений всередині компанії Google [56].

У нашому випадку для розробки смарт-контракту Ethereum буде вибрана мова Solidity, оскільки Solidity на сьогоднішній день є найбільш популярною об'єктно-орієнтованою і предметно-орієнтованою мовою програмування для написання смарт-контрактів для платформи Ethereum. Крім того, завдяки схожості з мовами C/C++ і JavaScript вибрана мова Solidity більш зручна в написанні без необхідності тривалого навчання.

2.5 Вибір інструментарію розробки програмного забезпечення

2.5.2 Огляд існуючих засобів розробки

2.5.1.1 Atom

Atom – безкоштовний редактор коду, розроблений GitHub для Windows, Linux і macOS з підтримкою плагінів, і вбудованих під керуванням Git. Atom заснований на Electron – фреймворку кроссплатформенної розробки, який включає Node.js і движок рендерингу Chrome. У Atom також можна писати власні плагіни на JavaScript, які розширюють можливість.

Крім того, в Atom є багаті можливості по налаштуванню, в яких можна налаштувати майже всі на свій смак. У нього вбудовані файл-менеджер, просунуті функції пошуку і заміни, різноманітні курсори, опції згортання коду, ясний інтерфейс, можливість імпорту правил і тем [57].

Переваги:

- безкоштовність. Atom IDE поширюється безкоштовно;
- кроссплатформеність. Працює на macOS, Windows і Linux;
- пакети. Atom має вбудований менеджер пакетів;
- оформлення інтерфейсу;
- мультифункціональність. Atom IDE підтримує майже всі мови, які використовуються для створення застосунків;
- простота і гнучкість.

Недоліки:

- продуктивність. При роботі з великими проєктами потрібно багато ОЗУ;
- запуск. Atom IDE дуже повільно запускається;
- збій. Як правило, відбувається збій або зависання текстовими файлами понад 10 МБ, що робить його менш корисним в якості звичайного текстового редактора.

2.5.1.2 Visual Studio Code

Visual Studio Code – редактор вихідного коду для кросплатформної розробки веб та хмарних застосунків. Дане середовище розроблено Microsoft для Windows, Linux і macOS. VS Code дозволяє розробляти як консольні застосунки, так і застосунки з графічним інтерфейсом, вебзастосунки і т.д. Продукт підтримує розробку для платформ ASP.NET і Node.js, а також React. Visual Studio Code заснований на Electron і реалізується через веб-редактор Monaco, розроблений для Visual Studio Online [58].

Переваги:

- безкоштовність. VS Code поширюється безкоштовно;
- кросплатформеність. Працює на macOS, Windows і Linux;
- пакети. Для VS Code є безлічі плагінів для розширення можливостей;
- оформлення інтерфейсу;
- мультифункціональність. VS Code підтримує майже всі мови, які використовуються для створення застосунків;
- простота і гнучкість.

Недоліки:

- продуктивність. При роботі з великими проєктами потрібно багато ОЗУ.

2.5.1.3 WebStorm

WebStorm – повнофункціональне інтегроване середовище розробки для JavaScript і є одним з продуктів розробника JetBrains. Інтерфейс схожий на інших IDE того ж розробника, таких як IntelliJ IDEA і т.д., а також на Android Studio, яке побудоване на базі вихідного коду продукту IntelliJ IDEA Community.

Це середовище відмінно підходить для різних платформ, таких як React, Angular, Vue.js і т.д. З її допомогою можна займатися налагодженням node-скриптів і запускати тести на вбудованому сервері. Також можна запускати і налагоджувати пртм-скрипти (за допомогою деревоподібного інтерфейсу). І для всього цього не потрібні ніякі плагіни [59].

Переваги:

- системна вимога. Можна комфортно працювати в середовищі навіть на слабкому комп'ютері;
- кросплатформерність. У будь-якої програми, написаного на Java, за замовчуванням є ця перевага. WebStorm можна запустити на будь-якій платформі, на якій встановлена Java;
- пакети. Для WebStorm є плагіни для розширення можливостей;
- оформлення інтерфейсу;
- простота і гнучкість.

Недоліки:

- платність. 30 днів можна безкоштовно користуватися, після чого доведеться платити.

2.5.2 Порівняння середовищ розробки та вибір

У таблиці 2.2 наведено порівняння середовищ розробки вебзастосунків згідно загальним статистикам, відгукам користувачів та своїм оцінкам при використанні.

Для розробки нашого застосунку було обране середовище розробки Visual Studio Code, оскільки дане середовище безкоштовне і зручне у використанні, має багатий вбудований функціонал, а також підтримує всі необхідні нами мови програмування. Для підтримки мови програмування Solidity можна встановити додатковий модуль.

Таблиця 2.2 – Порівняння середовищ розробки

Хар-ки \ IDE	Atom	Visual Studio Code	WebStorm
Дизайн	5/5	5/5	5/5
Зручність	5/5	5/5	5/5
Зрозумілість	5/5	5/5	5/5
Продуктивність	4/5	4/5	5/5
Служба підтримки	+	+	+
Укр./рос. мова	-	+	-
Підсвічування синтаксису	+	+	+
Рефакторинг	+	+	+
Автодоповнення коду	+	+	+
Пошук по проєкту	+	+	+
Налагодження	+	+	+
Тестування застосунків	+	+	+
Вартість	Безкоштовно	Безкоштовно	Платно, 30 днів безкоштовно
Мови, які підтримуються	Практично всі мови	Практично всі мови	JavaScript, CSS, HTML

3 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ

3.1 Розробка децентралізованого застосунку

Процес розробки ПЗ – набір правил, згідно з якою побудована розробка ПЗ. На даний момент майже всі традиційні вебзастосунки є централізованими, тобто сервери такого застосунку належать певної компанії або особі. У традиційних вебзастосунків архітектура «клієнт-сервер» характеризується наявністю не менше двох взаємодіючих, самостійних процесів – клієнт і сервер. Процеси, які здійснюють певну роботу, наприклад, службу бази даних або файлової системи, називаються серверами, а процеси, що запитують служби у серверів за допомогою посилки запиту і подальшого очікування відповіді від сервера, називаються клієнтами. Фактично ці процеси – ПЗ, яке встановлено на різних обчислювальних машинах і взаємодіє між собою через обчислювальну мережу за допомогою мережевих протоколів, але вони можуть бути розташовані також і на одній машині.

Децентралізованими називаються застосунки, які функціонують на основі технології блокчейн. Їх головними особливостями є відкритий код і стійкість до атак. Основний код і дані (бекенд) DApp зберігаються і виконуються в блокчейн тимчасової мережі, на відміну від застосунків клієнт-сервер, дані і код яких обробляються на серверах. DApp може мати для користувача інтерфейс, написаний будь-якою мовою (у нашому випадку HTML, CSS, JavaScript), який підтримує протокол зв'язку з децентралізованою частиною (у нашому випадку платформа Ethereum з використанням СК). Інтерфейс може бути повністю розміщений у децентралізованому сховищі, в IPFS, як у нашому випадку. У IPFS може зберігатися не тільки клієнтська частина, але і будь-які файли, завантажені користувачами DApp, наприклад матеріали до курсів, наукові статті, лабораторні, курсові, дипломні роботи.

На рис. 3.1 можна побачити різницю архітектури традиційного вебзастосунку та децентралізованого застосунку. Є три основні компоненти

нашого DApp, які потрібно розробити – клієнтська частина (Інтерфейс на мовах HTML, CSS і JS для взаємодії користувачем з DApp, а також зв'язування з IPFS і Ethereum), децентралізоване сховище IPFS і децентралізована частина (платформа Ethereum зі СК на мовою Solidity).

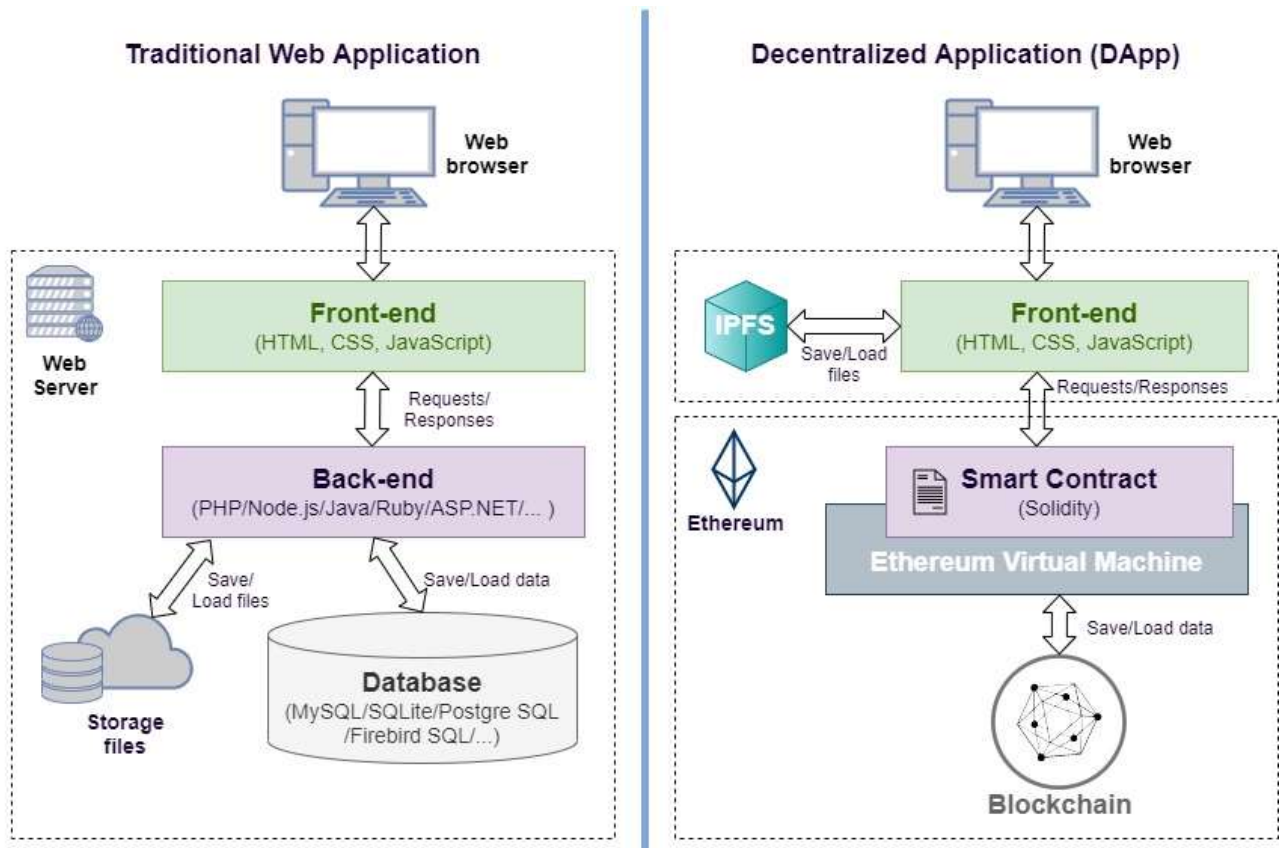


Рисунок 3.1 – Архітектури традиційного вебзастосунку і DApp

3.2 Ініціалізація проєкту

3.2.1 Необхідні компоненти для розробки DApp

Для початку розробки необхідні наступні компоненти, які потрібно встановити для подальшої розробки:

- Node.js;
- Web3.js;
- Truffle;
- Ganache;

Node.js – це програмна платформа, яка заснована на движку V8 (що транслює в машинний код), що перетворює JavaScript з вузькоспеціалізованої мови в мову загального призначення. Node.js ще необхідний, щоб зв'язати вебзастосунок з системою контрактів.

Web3.js це JS бібліотека, яка дозволяє використовувати Application Programming Interface (API) Ethereum за допомогою звичайного JS.

Truffle – це середовище розробки світового класу, середа тестування і конвеєр активів для блокчейнів з використанням віртуальної машини Ethereum (EVM), покликані полегшити життя розробникам. З Truffle можна отримати:

- вбудована компіляція СК і їх розгортання;
- автоматизоване тестування СК для швидкого розвитку;
- управління мережею для розгортання в будь-якій кількості державних і приватних мереж;
- управління пакетами за допомогою NPM;
- інтерактивна консоль для прямого контрактного спілкування;
- налаштований конвеєр складання з підтримкою жорсткої інтеграції.
- Ganache використовується для налаштування персонального блокчейну Ethereum для тестування наших СК мовою Solidity. Це забезпечує більше можливостей у порівнянні з Remix. У Ganache є основні функції:
 - генерація кількох учасників мережі, за замовчуванням 10 учасників, з унікальними адресами і з балансом 100 ЕТН за замовчуванням. При цьому можна перемкнути акаунт на одному комп'ютері і виконувати різні дії. Це вважається величезним плюсом для зручного і швидкого тестування DApp;
 - перегляд списку блоків, які додаються, з детальними описами;
 - перегляд списку транзакцій, які додаються, з детальними описами;
 - перегляд даних, що зберігаються в блокчейн через СК;
 - перегляд журналу подій, які додаються, з детальними описами;
 - перегляд журналу логів.

3.2.2 Структура проєкту DApp

Після установки необхідних компонентів створюється проєкт в IDE Visual Studio Code (рис. 3.2), який призначений для розробки вебзастосунків на стандартних мовах HTML, CSS, JavaScript. Додатково ще встановлюється плагін для підтримки мови Solidity, на якому буде написаний СК Ethereum. В даному проєкті під ім'ям LMS є папки:

- папка `migrations` містить в собі `js` файли, які допомагають розгортати контракти в мережі Ethereum. При введенні команди `truffle migrate` будуть використані ці файли, щоб розгортати контракти, написані на Solidity в `json` файли;
- папка `node_modules` зберігає всі необхідні модулі, встановлені за допомогою менеджера пакетів `npm`, що входить до складу Node.js. Такими модулями може бути що завгодно для розширення функціональності, наприклад модуль асиметричного шифрування, модуль `web3` і т.д;
- папка `public` містить всі елементи призначеного для користувача інтерфейсу, такі як HTML файли, зображення і CSS файли;
- папка `src` зберігає вихідні коди програми і містить кілька підпапок – підпапка `abis`, яка зберігає контракти типу JSON після міграції, підпапка `components`, яка зберігає головні компоненти програми, в основному зберігає головний клас застосунку `App.js`, підпапка `contracts`, яка зберігає СК на мові Solidity, підпапка `js`, яка зберігає JavaScript файли;
- папка `test` містить скрипти, які призначені для функціонального тестування програми.

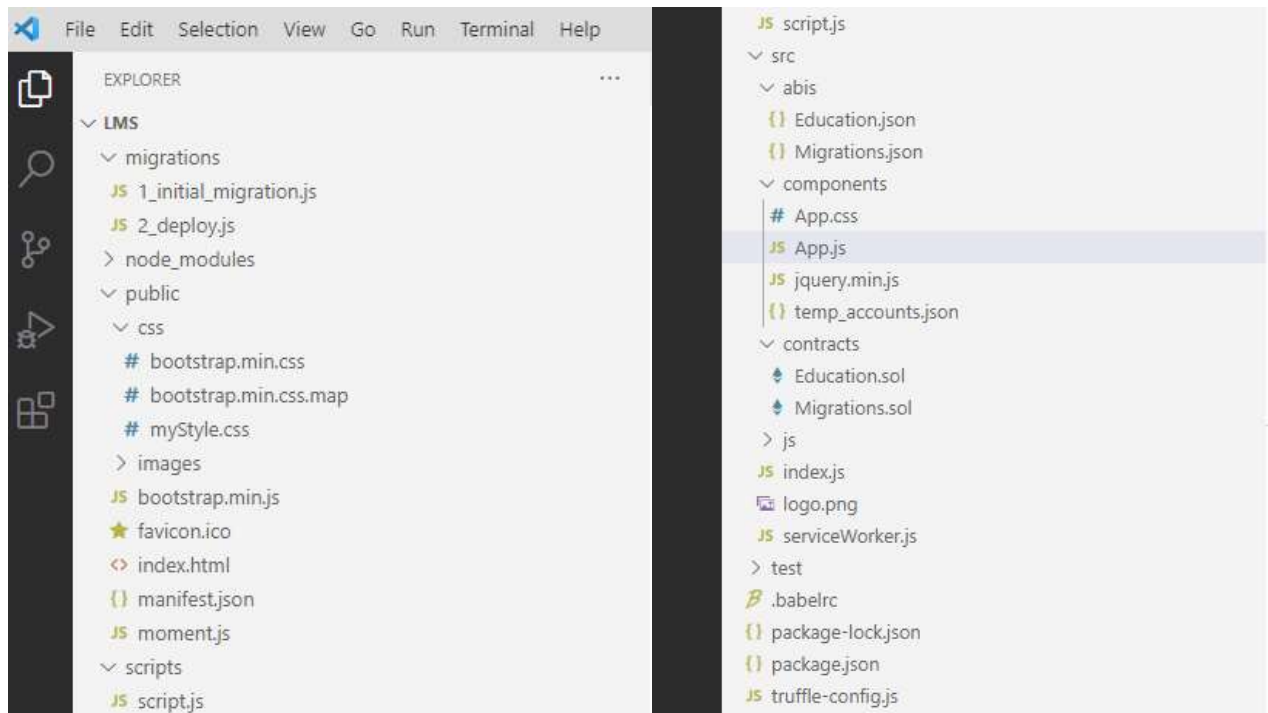


Рисунок 3.2 – Структура проекту DApp

3.3 Структура смарт-контракту

У нашому проєкті СК написаний на мові Solidity останньої версії. СК у нашому проєкті на Solidity представляється як один клас – contract Education, в якій описує всі змінні, структури, події, модифікатори і функції.

При ініціалізації СК викликається конструктор з параметрами – приватний ключ і адреса адміністратора, які використовуються для реєстрації адміністратора (лістинг 3.1). Для всіх користувачів в блокчейн зберігається хеш свого приватного ключа, який використовується виключно для аутентифікації. У нашому випадку для хешування використовується алгоритм SHA-256.

Лістинг 3.1 – Заголовок та конструктор СК

```
pragma solidity >=0.4.22 <0.9.0;

// Смарт-контракт "Навчання"
contract Education {
    address private adminAddress; // адрес адміністратора
    bytes32 private adminEnPKey; // хеш приватного ключа адміністратора
```

```

// конструктор смарт-контракту
constructor(string memory pKey) public {
    adminAddress = msg.sender;
    adminEnPKey = sha256(abi.encodePacked(pKey));
}

struct ObjectName {
    ...
}
...
}

```

Для Solidity все сутності (студент, викладач, курс, завдання, робота студента, учасник) подаються у вигляді структури, наприклад у лістингу 3.2 наведений код структури студента:

Лістинг 3.2 – Код структури студента

```

//об'єкт - Студент
struct Student {
    uint256 id;
    string profileIPFS;
    string fullName;
    string email;
    string group;
    bytes32 encryptedPrivateKey; //хеш приватного ключа (для аутентифікації)
    string publicKey; //публічний ключ
}

```

Всі структури зберігаються як словники типу `mapping`, в яких кожному ідентифікатору або адресі відповідає структура, що описує якась сутність. Наприклад, для прив'язки цифрової сутності студента і викладача до конкретної адреси введені такі словники:

```

mapping(address => Student) public students; // карта (адреса студента - об'єкт студента)
mapping(address => Teacher) public teachers; // карта (адреса препода - об'єкт препода)

```

А інші структури прив'язані до ідентифікатору, як в лістингу 3.3.

Лістинг 3.3 – Код словників типу mapping

```
uint256 public coursesCount; //кількість курсів у блокчейн (ця змінна
//потрібна, тому що не працює length() або size() карти, масива)
mapping(uint256 => Course) public courses; //курси
uint256 public tasksCount;
mapping(uint256 => Task) public tasks; //завдання
...

uint256 public studentWorksCount;
mapping(uint256 => StudentWork) public studentWorks; //роботи студентів
```

Для перевірки стану користувача в системі використовуються модифікатори доступу, які перевіряють чи є адміном, викладачем або студентом, авторизований користувач, чи є учасником певного курсу і т.д. (лістинг 3.4).

Лістинг 3.4 – Код модифікаторів доступу

```
// модифікатор - чи є адміном
modifier isAdmin {
    require(isAdminF(msg.sender));
    _;
}

// модифікатор - чи є студентом
modifier isStudent(address _address) {
    require(isStudentF(_address));
    _;
}

// модифікатор - чи є викладачем
modifier isTeacher(address _address) {
    require(isTeacherF(_address));
    _;
}

// модифікатор - не авторизований чи користувач
modifier isUnauthUser(address _address) {
    require(!isTeacherF(_address) && !isStudentF(_address));
    _;
}

// модифікатор - чи є учасником певного курсу (чи є доступ до курсу)
modifier isAccessToCourse(address _address, uint256 course_id) {
    require(isAccessToCourseF(_address, course_id));
    _;
}
```

```
// модифікатор - аутентифікація (авторизований чи користувач)
modifier isAuth(string memory _enPKey) {
    require(isAuthF(msg.sender, _enPKey));
    _;
}
```

Ці модифікатори ставляться перед блоком функції, тому функція може виконається чи ні в залежності від умов модифікаторів, як наприклад в лістингу 3.5.

Лістинг 3.5 – Код функції з модифікаторами

```
// функція виставлення оцінки, дана функція доступна тільки викладачам
function setGrade(
    uint256 course_id,
    uint256 task_id,
    uint256 grade,
    address student,
    string memory enPKey //хеш приватного ключу
) public
    isTeacher(msg.sender) //модифікатор - чи є викладачем
    isAccessToCourse(msg.sender, course_id) //модифікатор - чи є учасником п
    evного курсу (чи є доступ до курсу)
    isAuth(enPKey) //модифікатор - аутентифікація (авторизований чи корист.)
{
    //запис події виставлення оцінки
    emit grading(
        course_id,
        task_id,
        grade,
        block.timestamp,
        student,
        msg.sender
    );
}
```

Як показано на схемі аутентифікації з рис. 2.5., що процес аутентифікації відбувається кожного разу при виклику деяких функцій для внесення змін і збереження даних в блокчейн. Для цього використовується модифікатор isAuth (enPKey), який ставиться перед блоку функції, цей модифікатор приймає приватний ключ користувача для отримання хеша, а потім порівнює його з

хешом приватного ключа, який заздалегідь був збережений в блокчейн при реєстрації. У лістингу 3.6 наведений код модифікатора аутентифікації.

Лістинг 3.6 – Код модифікатора аутентифікації

```
// функція - аутентифікація (авторизований чи користувач)
function isAuthF(address _address, string memory _enPKey)
    public
    view
    returns (bool result) //результат, true - okay, false - not okay
{
    //for admin
    if (adminAddress == _address) {
        if (sha256(abi.encodePacked(_enPKey)) == adminEnPKey) {
            result = true;
        }
    }
    //for student
    if (!result) {
        for (uint256 i = 0; i < studentsCount; i++) {
            if (studentsAddr[i] == _address) {
                if (
                    sha256(abi.encodePacked(_enPKey)) ==
                    students[_address].encryptedPrivateKey
                ) {
                    result = true;
                }
            }
        }
    }
    //for teacher
    if (!result) {
        for (uint256 i = 0; i < teachersCount; i++) {
            if (teachersAddr[i] == _address) {
                if (
                    sha256(abi.encodePacked(_enPKey)) ==
                    teachers[_address].encryptedPrivateKey
                ) {
                    result = true;
                }
            }
        }
    }
}
```

При завантаженні файлів користувачем, наприклад, відправка своїх робіт студентами, в блокчейн зберігається хеш, тип і ім'я файлу, а повний файл зберігається в IPFS, але зашифрованим публічним ключем обраного курсу.

Хеш, тип і ім'я файлу використовуються для визначення і скачування повного файлу з IPFS (лістинг 3.7).

Лістинг 3.7 – Код структури роботи студента з хешем файлу IPFS

```
//об'єкт - Робота студента
struct StudentWork {
    uint256 id;
    uint256 course_id;
    uint256 task_id;
    address student;
    string fileName; //ім'я файлу
    string fileType; //тип файлу
    string fileIPFS; //хеш файлу (шлях до повного файлу в IPFS)
}
```

3.4 Розробка клієнтської частини

Як тільки проєкт створений, можна приступити до розробки. У нас є важливі елементи для динамічного відображення інтерфейсу і обробки функцій - App.js і index.html. Для повного DApp розробляються кілька .html файлів, але у нас є один файл, оскільки нашою метою є розробка підсистеми управління навчанням, а не цілої системи, з мінімальним інтерфейсом, який може бути достатній для виконання і тестування всіх необхідних функцій підсистеми на базі блокчейн з використанням СК. Одним файлом .html за допомогою App.js можна динамічно переключити сторінку, змінювати інтерфейс в залежності від ролі користувача, для цього в App.js використовується різні змінні – поточний стан програми (назва поточної сторінки, номер поточної навігації і т.д.) і функція renderApp(), яка динамічно оновлює інтерфейс за завданням параметрам та станами застосунку. У лістингу 3.8 приведений HTML код структури окремих контентів, за замовчуванням всі вони приховані. Код управління контентом і динамічного оновлення інтерфейсу приведений в лістингу 3.9.

Лістинг 3.8 – HTML код структури окремих контентів

```

<!--основний контент - Реєстрація-->
<div id="contentRegister" style="display: none;">
    ...
</div>
<!--основний контент - Логін-->
<div id="contentLogin" style="display: none;">
    ...
</div>
<!--основний контент - Список курсів-->
<div id="contentCourses" style="display: none;">
    ...
</div>
<!--основной контент - Новий курс-->
<div id="contentNewCourse" style="display: none;">
    ...
</div>
<!--основний контент - Сторінка завдання-->
<div id="contentTask" style="display: none;">
    ...
</div>
<!--основний контент - Додавання учасників в курс-->
<div id="contentAddUsersToCourse" style="display: none;">
    ...
</div>
<!--основний контент - Журнал подій-->
<div id="contentEvents" style="display: none;">
    ...
</div>
<!--основний контент - Журнал своїх оцінок-->
<div id="contentGrades" style="display: none;">
    ...
</div>

```

Лістинг 3.9 – Код управління контентом і динамічного оновлення інтерфейсу в App.js

```

//функція динамічного відображення інтерфейсу застосунку
async renderApp() {
    var contentLogin = $("#contentLogin");
    var contentCourses = $("#contentCourses");
    var contentCourse = $("#contentCourse");
    ...

    //Якщо користувач не зареєстрований
    if (App.thisApp1.currentRole == '') {
        //loader.hide();
        contentLogin.show();
        ...
    }
}

```

```

        $('#titlePage').text("Login");
        ...
    }
    else {
        //Якщо поточний користувач - адміністратор
        if (App.thisApp1.currentRole == "Admin") {
            navigatorRegister.show();
            ...
            if (App.thisApp1.currentPage == "register") {
                $('#titlePage').text("Registration");
                contentRegister.show();
            }
            ...
        }
    }
    else {
        //Якщо поточний користувач - викладач
        if (App.thisApp1.currentRole == "Teacher") {
            ...
            $('#currentAccountRole').html("Teacher");
            ...
            //Якщо поточний користувач - студент
        } else if (App.thisApp1.currentRole == "Student") {
            ...
            $('#currentAccountRole').html("Student");
            ...
        }
        //Якщо поточна сторінка - список курсів
        if (App.thisApp1.currentPage == "courses") {
            $('#titlePage').text("Courses");
            ...
            contentCourses.show();
        }
        else if (App.thisApp1.currentPage == "createCourse") {
            $('#titlePage').text("New course");
            ...
            contentNewCourse.show();
        }
        else if (...) {
            ...
        }
    }
}

```

Для використання API Ethereum в клієнтської частини на JavaScript був встановлений пакет бібліотек web3.js і імпортовано в проєкт через файл App.js за допомогою коду `import Web3 from 'web3'`. Далі для взаємодії web3.js з вузлами по протоколу HTTP потрібно створити екземпляр web3, який слід буде почати

при запуску програми. У лістингу 3.10 представлений базовий код для підключення до вузла. Спочатку перевіряємо web3 – чи є він невизначеним. Якщо web3 визначено, ми скористаємося вже існуючими екземпляром. В іншому випадку ми створюємо екземпляр шляхом підключення до нашого призначеного для користувача вузлу. Тут ми ще маємо на увазі, що вузол доступний локально через порт 7545.

Лістинг 3.10 – Базовий код для підключення до вузла по протоколу HTTP за допомогою web3

```
//якщо заздалегідь ініціалізовано
if (App.thisApp1.web3 != null) {

//вказуємо на поточний провайдер, який був ініціалізований заздалегідь
App.thisApp1.web3Provider = App.thisApp1.web3.currentProvider;
App.thisApp1.web3 = new Web3(App.thisApp1.web3.currentProvider);
} else {
    //ініціалізація в перший раз
    App.thisApp1.web3Provider = new Web3.providers.HttpProvider('http://localhost:7545');
    App.thisApp1.web3 = new Web3(App.thisApp1.web3Provider);
}
```

Примірник web3 містить об'єкт eth (web3.eth), призначений спеціально для взаємодії з блокчейном Ethereum. Велика частина API пакету web3.js представлена всередині цього об'єкта. Все API можна зробити синхронними або асинхронними, за замовчуванням всі синхронні. Для взаємодії з блокчейном доступні багато функцій і класів, такі як `getBlock(...)`, `getAccounts()`, `Contract(...)` і т.д.

Метод `getBlock()` повертає інформацію про блок за його порядковому номеру або хешу. Також можна замість номера використовувати ключові слова `earliest` (генезісний блок), `latest` (останній доданий блок) або `pending` (блок в майнінг). За замовчуванням запитується `web3.eth.defaultBlock`, якому відповідає псевдонім `latest`.

Метод `getAccounts()` повертає список адрес учасників мережі, які були згенеровані в Ganache для тестування, можна також додати нового учасника мережі в список, для цього код приведений в лістингу 3.11.

Метод `create()` в нашому застосунку використовується при реєстрації користувача (викладача або студента) адміністратором з генерацією приватного і публічного ключа.

Лістинг 3.11 – Код додавання нового облікового запису та повернення списку адрес учасників мережі

```
//створення нового учасника мережі
var newAcc = App.thisApp1.web3.eth.accounts.create();
//outpit: {address: "0x9936e190eB228783067214e29bdad2c590A6Dde4", privateKey: "0xd92069d2d96979ef112f643a89900617a2a90fe9370c3a6511a5ae1cd02d661e", signature: f, sign: f, encrypt: f}
//додавання нового учасника в список
App.thisApp1.accounts.push(newAcc.address);
//Всі учасники мережі блокчейна
App.thisApp1.accounts = await App.thisApp1.web3.eth.getAccounts();
```

Крім того, для виклику методів СК, відстеження подій, необхідні для виконання всіх завдань користувача спочатку повинен бути розгорнутий і ініціалізований СК, розроблений на мові Solidity, в проєкт. Коли розробка СК мовою Solidity закінчена, потрібно спочатку розгортати його за допомогою створеного файлу `2_deploy.js`, який знаходиться в папці `migrations`, і команди `truffle migrate`. У лістингу 3.12 наведений код файлу `2_deploy.js`, він створений вручну і необхідний для розгортання СК в json файл.

Лістинг 3.12 – Код файлу `2_deploy.js`

```
//посилання на СК .sol файл
const Education = artifacts.require("Education");
module.exports = async function(deployer) {
    //розгортання з параметром - приватний ключ адміністратора
    await deployer.deploy(Education, "2e376a83ea7b315cfd5c6423441d438f4042ec4613a2d7acb6c6e07f6cab8606");
    const education = await Education.deployed();
};
```

У лістингу 3.13 наведений код отримання посилання на розгорнутий СК для подальшої роботи з методами і даними СК.

Лістинг 3.13 – Код додавання нового облікового запису та повернення списку адрес учасників мережі

```
import Education from '../abis/Education.json' //розгорнутий смарт-
контракт
...
//Ідентифікатор блокчейна
App.thisApp1.networkId = await App.thisApp1.web3.eth.net.getId();
//Дані мережі блокчейна
App.thisApp1.networkData = Education.networks[App.thisApp1.networkId];
if (App.thisApp1.networkData) {
    //Отримання посилання на розгорнутий смарт-
контракт, написаний на Solidity
    App.thisApp1.educationContract = new App.thisApp1.web3.eth.Contract(Edu
cation.abi, App.thisApp1.networkData.address);
}
```

У лістингу 3.14 наведений код прикладу виклику методів СК, а також отримання змінних СК. Функції та змінні в СК повинні бути публічними, щоб мати API доступ до них. А приватні функції і змінні в СК доступні для виклику тільки всередині самого СК.

Лістинг 3.14 – Код виклику методів і змінних СК

```
//отримання кількості курсів в мережі
App.thisApp1.educationContract.methods.coursesCount.call().call({ from: A
pp.thisApp1.account }, function (error, coursesCount) {
    for (var i = 0; i < coursesCount; i++) {
        App.thisApp1.educationContract.methods.courses(i).call({ from: App.th
isApp1.account }, function (error, course) {
            ...
            //отримання об'єкта викладач за адр.(course[3] - адр. власн. курсу)
            App.thisApp1.educationContract.methods.teachers(course[3]).call({ f
rom: App.thisApp1.account }, function (error, teacher) {
                ...
            });
        });
    }
    contentCourses.show();
});
```

Крім виконання функцій і змінних СК також можна отримати подій з СК для перегляду. У лістингу 3.15 наведений код прикладу отримання подій викладача.

Лістинг 3.15 – Код отримання подій викладача через СК

```
//повернення подій дій викладача (створення курсу, завдань, учасників до курсу,
матеріалів до курсу і т.д)
App.thisApp1.educationContract.getPastEvents(
  "teacherAction",
  {
    fromBlock: 0, //від першого блоку
    toBlock: "latest" //до останнього блоку отримати події
  }, (errors, events) => {
    if (!errors) { //якщо без помилок
      ...
      //перехід по подіям
      for (var i = 0; i < events.length; i++) {
        var resultsEvent = events[i].returnValues;
        ...
      }
    }
  }
}
```

Повний код даної програми наведено в додатку Б «Текст програми».

3.5 Установка і застосування децентралізованого сховища IPFS

Для зберігання файлів в IPFS спочатку потрібно встановити бібліотеку `ipfs-http-client`, яка реалізує API IPFS Core, що дозволяє застосункам змінюватися між вбудованим вузлом `js-ipfs` і будь-яким віддаленим вузлом IPFS без необхідності змінювати код. Крім того, ця клієнтська бібліотека реалізує набір корисних функцій.

Для установки потрібно ввести команду через менеджер пакетів `npm`:

```
npm install --save ipfs-http-client
```

Потім в App.js потрібно імпортувати і створити екземпляр клієнту HTTP API IPFS:

```
//Імпорт IPFS бібліотеки
const { create } = require('ipfs-http-client')
const ipfs = create({ host: 'ipfs.infura.io', port: 5001, protocol: 'http'
  })
```

Можна змінити будь-які параметри – URL, порт, протокол при необхідності. У нашому випадку для тестування все залишаються так.

Далі для додавання файлів досить використовувати одну функцію add(), як наведено у лістингу 3.16. Можна завантажити практично що завгодно, будь то текстові документи, архіви, відео та зображення в незашифрованому, а також зашифрованому вигляді. Дана функція використовується для додавання матеріалів до курсу викладачем, а також для додавання робіт студентами.

Листинг 3.16 – Код додавання даних в IPFS

```
//додавання даних в IPFS, в такому випадку додається зашифрований файл пу
бличним ключем
ipfs.add(encrypted)
  .then(result => {
    //при успішному додаванні отримуємо хеш файлу (шлях до повного ф
айлу в IPFS)
    console.log("hash file ipfs = " + result.path)
    //приклад:https://ipfs.infura.io/ipfs/QmdcK5ZnynfS6w6TSdY5KrwTYw
DiVWqkMKV1P8AQ6353Nm
  })
  .catch(err => {
    //помилка
    console.log(err)
  });
```

А для отримання файлу з IPFS достатній всього лише URL. Далі є різні способи завантаження файлу по URL, в нашому випадку для цього використовується XMLHttpRequest, як наведено у лістингу 3.17.

Листинг 3.17 – Код завантаження файлу з IPFS

```
//URL на повний файл в IPFS
var url = "https://ipfs.infura.io/ipfs/" + work[6]; //хеш файлу IPFS

console.log("crypted data: " + url);
//запит
var request = new XMLHttpRequest();

//тип запиту по отримання по URL
request.open('GET', url);
request.responseType = 'text';
//функція на завантаження по URL
request.onload = function () {
//Вміст файлу з URL
console.log(request.response);

//розшифровка вміст файлу
var decrypted = keyCourse.decrypt(request.response);

//код для автоматичного скачування файлу
var blob = new Blob([decrypted], { type: work[5] }); //Данні та тип файлу
var link = document.createElement("a");
link.href = window.URL.createObjectURL(blob);
link.download = work[4]; //Ім'я файлу
link.click();

};
// виклик функції на завантаження файлу по URL. Без виклику нічого не ста
неться.
request.send();
```

3.6 Безпека передачі та зберігання даних, вибір криптосистеми

3.6.1 Коротко про безпеку і обґрунтування вибору криптосистеми

У нашому проєкті дані, що зберігаються в блокчейн і в IPFS, доступні всім учасникам мережі, тому децентралізована система повинна мати максимальний рівень безпеки даних, що зберігаються в блокчейн і переданих між точками.

Під безпекою інформаційної системи розуміється захищеність системи від випадкового або навмисного втручання в нормальний процес її функціонування, від спроб розкрадання (несанкціонованого отримання) інформації та її модифікації. При передачі інформації по каналах зв'язку і

зберіганні в публічний блокчейн використовуються криптографічні алгоритми шифрування. Існує кілька видів криптосистем, одна з них ділить за кількістю ключів, що застосовуються в конкретному алгоритмі:

- безключові криптосистеми, які не використовують в обчисленнях ніяких ключів;
- одноключові криптосистеми, які працюють з одним ключовим параметром (секретним ключем);
- двохключові криптосистеми, які на різних стадіях роботи в них застосовуються два ключові параметри: приватний і публічний ключі.

У більшості випадках в нашому застосунку найбільше підходить двухключова криптосистема RSA і ми використовуємо її. Завдяки асиметричному криптосистему користувачам не доводиться передавати іншим користувачам один приватний ключ симетричної криптосистеми через інший захищений зв'язок для шифрування і дешифрування даних. За допомогою асиметричної криптосистеми досить мати обидва ключа, один з них – публічний і доступний всім для шифрування даних, а приватний ключ доступний тільки однієї особі для розшифрування даних.

Асиметричне шифрування на основі відкритого ключа RSA на даний момент використовує більшість продуктів на ринку інформаційної безпеки. Бувають різні розміри ключа RSA, для асиметричного шифрування RSA розмір ключа 512 біт – є технічно мінімальним, але він небезпечний. Тому в даний час абсолютним мінімумом є 1024 біта, а рекомендується 2048 біт. За рівнем безпеки розмір ключа 1024 біта в RSA відповідають 128 бітам симетричного алгоритму (AES), а 2048 біт RSA – 256 бітам AES (таб. 3.1).

Таблиця 3.1 – Відповідність розмірів ключа за рівнем безпеки

Асиметричне шифрування (RSA)	Симетричне шифрування (AES)
1024 біт	128 біт
2048 біт	256 біт

Для нашого DApp рекомендовано обраний розмір – 2048 біт, але на етапі розробки обраний 1024 біт для спрощення тестування.

Алгоритм RSA у нашому проєкті буде застосований в деяких випадках, таких як зберігання зашифрованих робіт студентів, розшифрування робіт студентів викладачами для їх читання, передача особистих даних між вузлами, реєстрація та аутентифікація.

3.6.2 Підключення та використання криптосистеми в проєкт

Для використання криптосистеми можна завантажити бібліотеку 'node-rsa' і підключити його в клієнт для подальшого використання. Для підключення бібліотеку спочатку використовується команда в консолі 'npm install node-rsa' в тому ж каталозі, де зберігається проєкт. Потім у файлі App.js на початку ставиться import NodeRSA from 'node-rsa'. Далі потрібно форматувати його з параметром – розмір ключа, в нашому випадку 1024 біт:

```
const key = new NodeRSA({ b: 1024 });
```

При ініціалізації автоматично генеруються обидва ключа – приватний і публічний ключ. Обидва ключі можна отримати наступним чином:

```
const coursePublicKey = key.exportKey('pkcs8-public-pem');
const coursePrivateKey = key.exportKey('pkcs1-pem');
```

Можна також імпортувати вже існуючі ключі, наприклад публічний ключ якогось користувача, що зберігається в блокчейн, свій приватний ключ зі свого пристрою и т.д.:

```
key.importKey(publicKeyUser, 'pkcs8-public-pem');
key.importKey(App.thisApp.accountTempPrivateKey, 'pkcs1-pem');
```

Шифрувати і дешифрувати дані тими ж ключами можна так:

```
var encrypted = key.encrypt(data, 'base64');
var decrypted = key.decrypt(encrypted);
```

Як і описано в схемах у другому розділі, ключі вищевказаних чином генеруються при реєстрації користувачів адміністратором, а також при створенні нового курсу викладачем.

Після реєстрації користувачів адміністратором згенерований публічний ключ нового користувача зберігається в блокчейн через СК у структурі struct Student або struct Teacher, а згенерований приватний ключ відправляється новому користувачеві через e-mail і зберігається тільки на своєму місці. У лістингу 3.18 представлений код реєстрації викладача з генерацією ключів.

Листинг 3.18 – Код функції реєстрації викладача з генерацією ключів

```
async registerTeacher() {
    // ініціалізація і генерація ключів
    const key = new NodeRSA({ b: 1024 });
    // отримання ключів
    publicKey = key.exportKey('pkcs8-public-pem');
    const privateKey = key.exportKey('pkcs1-pem');
    // виклик методу смарт-контракту
    App.thisApp1.educationContract.methods.registerTeacher(profile, fullName,
    email, address, sha(privateKey), publicKey,
    App.thisApp1.accountTempPrivateKey).send({ from: App.thisApp1.account, gas: 3000
    000 }, function (receipt) {
        //якщо немає помилок
        if (receipt == null) {
            console.log("successfully registered teacher + ");
        }
        else
            console.log("Error: " + receipt);
    });
}
```

Згідно зі схемою з рисунку 2.6 «Діаграма послідовності додавання курсу і викладачів до курсу з генерацією публічних та приватних ключів курсу» реалізовані дві основні функції у лістингу 3.19 і 3.20. При виконанні функції

`addCourse()` генеруються обидва ключа для курсу, потім викладач шифрує приватний ключ для курсу своїм публічним ключем і зберігає його в блокчейн. А при виконанні функції `addTeacherToCourse()` додається інший викладач до курсу при цьому створивши для іншого викладача свій зашифрований приватний ключ курсу.

Листинг 3.19 – Код функції додавання нового курсу викладачем з генерацією ключів курсу

```

async addCourse() {
    //генерація ключів для курсу
    const key = new NodeRSA({ b: 1024 });
    const coursePublicKey = key.exportKey('pkcs8-public-pem');
    const coursePrivateKey = key.exportKey('pkcs1-pem');
    const userKey = new NodeRSA({ b: 1024 }); //імпорт ключів викладача
    App.thisApp1.educationContract.methods.teachers(App.thisApp1.account).call({
    from: App.thisApp1.account }, function (error, teacher) {
        userKey.importKey(teacher[5], 'pkcs8-public-pem');
        userKey.importKey(App.thisApp1.accountTempPrivateKey, 'pkcs1-pem');
        //шифрування приватного ключа публічним ключем викладача
        const encryptedCoursePrivateKey = userKey.encrypt(coursePrivateKey, 'base6
4');
        //виклик методу смарт-контракту
        App.thisApp1.educationContract.methods.addCourse(name, App.thisApp1.accountTempPrivateKey, coursePublicKey, encryptedCoursePrivateKey).send({ from: App.thisApp1.account, gas: 3000000 }, function (receipt) {
            if (receipt == null) {
                App.thisApp1.currentPage = "courses";
                App.thisApp1.renderApp();
            } else {
                console.log("Error: " + receipt);
            }
        });
    });
}

```

Листинг 3.20 – Код функції додавання іншого викладача зі своїм приватним ключем курсу

```

async addTeacherToCourse() {
    var email = $('#emailAddTeacher').val();
    //імпорт свого приватного ключа, який потім потрібен для розшифрування приватного ключа курсу з блокчейна
    const key = new NodeRSA({ b: 1024 });
    key.importKey(App.thisApp1.accountTempPrivateKey, 'pkcs1-pem');
    //виклик методу смарт-контракту для отримання шифрованого приватного ключа курсу
    App.thisApp1.educationContract.methods.getEnPrivateKeyCourse(App.thisApp1.account, App.thisApp1.currentCourseId).call({ from: App.thisApp1.account }, function (error, enPKeyCourse) {

```

```

    const coursePrivateKey = key.decrypt(enPKeyCourse);
    //виклик методу смарт-контракту для отримання препода по е-
mail, необхідне для отримання його публічного ключа
    App.thisApp1.educationContract.methods.getTeacherFromEmail(email).call({ f
rom: App.thisApp1.account }, function (error, teacher) {
        const addedTeacherKey = new NodeRSA({ b: 1024 });
        //отримання публічного ключа препода по e-mail
        addedTeacherKey.importKey(teacher[5], 'pkcs8-public-pem');
        //шифрування приватного ключа курсу ключем доданого викладача і збережен
ня в блокчейн
        const newEnCoursePrivateKey = addedTeacherKey.encrypt(coursePrivateKey);
        App.thisApp1.educationContract.methods.addTeacherEmail(email, App.thisAp
pl.currentCourseId, App.thisApp1.accountTempPrivateKey, newEnCoursePrivateKey.to
String()).send({ from: App.thisApp1.account, gas: 3000000 }, function (receint)
{
            if (receint == null) {
                App.thisApp1.currentPage = "course";
                App.thisApp1.renderApp();
            }
            else
                console.log("Error: " + receint);
        });
    });
});
}

```

За допомогою публічного ключа обраного курсу студент може шифрувати свої роботи до зберігання в мережі. Викладачі отримують і розшифрують дані роботи приватним ключем курсу. Коди таких реалізацій представлені в додатку Б «Текст програми».

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

4.1 Опис застосування програми

4.1.1 Призначення програми

Дана програма призначена для організації і контролю навчального процесу, оцінювання ефективності навчання та знань студентів, відстеження дій користувачів, а також надійного і безпечного зберігання наукових робіт в рамках ДН.

4.1.2 Можливості програми

Програма дозволяє:

- зареєструватися в системі / увійти в систему. Для реєстрації користувач повинен подати заявку адміністратору, після чого адміністратор проводить реєстрацію і відправляє дані користувачеві для входу;
- мати адміністратору можливість повного контролю і зміни навчального процесу;
- мати адміністратору можливість моніторингу мережі блокчейну;
- викладачам керувати курсами (додавання курсу і завдань, учасників до курсу);
- завантажувати незашифровані і зашифровані файли в IPFS;
- студентам здати свої роботи;
- викладачам отримувати і перевіряти роботу студента, а також оцінити її;
- викладачам переглядати журнал оцінок студентів;
- переглядати журнал своїх дій.

4.1.3 Основні характеристики програми

Дана програма є децентралізованим застосунком на основі блокчейн Ethereum, в якому немає центрального серверу для управління і зберігання даних. Тому ця програма може працювати завжди і має стійкість до атак, а також надійне і безпечне зберігання даних про навчальний процес.

4.1.4 Обмеження, які накладаються на область застосування програми

Дана програма може використовуватися для організації навчального процесу, управління навчанням і призначена тільки зареєстрованим користувачам, які навчаються/працюють в певних закладах освіти/академіях. Програма створена для роботи під управлінням будь-якої операційної системи, в якій є браузер з підтримкою JavaScript і cookie.

4.1.5 Мінімальний склад технічних та програмних засобів

Для роботи системи необхідний персональний комп'ютер з характеристиками не нижче:

- процесор з тактовою частотою 1 ГГц або вище;
- оперативна пам'ять об'ємом 2 Гб або вище;
- жорсткий диск об'ємом 128 Гб або вище;
- будь-який графічний адаптер;
- монітор з роздільною здатністю екрану не менше 1024x768 пікселів;
- мережевий інтерфейс Ethernet 10/100/100 Base-TX.

Для запуску і виконання системи необхідний браузер Internet Explorer версії 6.0 і вище, або Opera версії 9.0 і вище, або Firefox версії 2.0 і вище, або Google Chrome версії 7.0 і вище, або будь-який браузер з підтримкою JavaScript

і cookie. Застосунком можна користуватися у будь-яких операційних системах, які підтримують будь-який з перерахованих браузерів.

4.2 Керівництво системного програміста

4.2.1 Загальні відомості про програму

4.2.1.1 Призначення програми

Дана програма призначена для організації і контролю навчального процесу, оцінювання ефективності навчання та знань студентів, відстеження дій користувачів, а також надійного і безпечного зберігання наукових робіт в рамках ДН.

4.2.1.2 Функції програми

Для адміністратора:

- реєстрація користувачів. Тільки адміністратор може зареєструвати користувачів і зберегти їх в блокчейн за запитами користувачів;
- повний контроль і зміна навчального процесу;
- моніторинг мережі блокчейна.

Для викладачів:

- реєстрація і авторизація. Для реєстрації користувач заповнює форму заявки з деякими особистими даними і відправляє адміністратору для перевірки;
- додавання і редагування нового курсу;
- додавання завдань і матеріалів до курсу;
- додавання учасників (викладачів і студентів) в обраний курс;
- перевірка робіт студентів та оцінювання.
- перегляд журналу оцінок студентів.
- перегляд журналу дій.

Для студентів:

- реєстрація і авторизація. Реєстрація для студентів проводиться за тим же принципом, що і для викладачів;
- перегляд списку курсів;
- вибір потрібного курсу і отримання матеріалів, завдань;
- здача своїх робіт;
- отримання повідомлень і перегляд своїх оцінок;
- перегляд журналу дій.

4.2.1.3 Мінімальний склад апаратних засобів

Для роботи системи необхідний персональний комп'ютер з характеристиками не нижче:

- процесор з тактовою частотою 1 ГГц або вище;
- оперативна пам'ять об'ємом 2 Гб або вище;
- жорсткий диск об'ємом 128 Гб або вище;
- будь-який графічний адаптер;
- монітор з роздільною здатністю екрану не менше 1024x768 пікселів;
- мережевий інтерфейс Ethernet 10/100/100 Base-TX.

4.2.1.4 Мінімальний склад програмних засобів

Для запуску і виконання системи необхідний браузер Internet Explorer версії 6.0 і вище, або Opera версії 9.0 і вище, або Firefox версії 2.0 і вище, або Google Chrome версії 7.0 і вище, або будь-який браузер з підтримкою JavaScript і cookie. Застосунком можна користуватися у будь-яких операційних системах, які підтримують будь-який з перерахованих браузерів.

4.2.1.5 Вимоги до системного програмісту

Системний програміст повинен мати мінімум середню технічну освіту і досвід програмування на мовах високого рівня.

У перелік завдань, які виконуються системним програмістом, повинен входити:

- завдання підтримки працездатності технічних засобів;
- завдання установки і підтримки працездатності системних програмних засобів – операційної системи;
- завдання установки, настройки і підтримки працездатності середовища розробки і компонентів (програм) для настройки та тестування блокчейну Ethereum.

4.2.2 Структура програми

Дана система складається з наступних частин:

- клієнтська частина;
- децентралізоване сховище IPFS;
- платформа Ethereum зі СК.

У клієнтської частини включаються інтерфейс, обробка дій користувача і зв'язування з IPFS і Ethereum для збереження і зміни даних. Ця частина розробляється на мовах HTML, CSS і JavaScript.

Децентралізоване сховище IPFS зберігає в собі інтерфейс децентралізованого застосунку (клієнтська частина), оскільки у цієї системи немає третьої особи (центрального сервера), і інші файли, завантажені користувачами, наприклад матеріали до курсів, наукові статті, лабораторні, курсові та дипломні роботи. IPFS пов'язаний з клієнтської частини для завантаження і збереження даних невеликого розміру, а з платформою Ethereum пов'язаний лише тим, що СК Ethereum зберігає в блокчейн лише хеш (посилання) завантаженого файлу в IPFS.

Децентралізована частина EVM являє собою середовище виконання СК в системі Ethereum. Її основною особливістю є ізолюваність від зовнішнього світу, тобто код, який працює всередині EVM, не має доступу до мережі, файлової системи або іншим подібним елементам. Коли є код (або дані) у блокчейні, необхідний консенсус, щоб домовитися про те, що цей код буде робити. Всі вузли згодні з тим, як повинен себе вести EVM (тобто згодні з його правилами), і всі мають однакові дані в рамках даного блокчейну, тому кожен вузол буде обробляти одні й ті ж дані. З цієї точки зору, віртуальна машина EVM схожа на один великий всесвітній комп'ютер.

Крім тих, при активної розробці і тестування є ще зв'язок з Ganache, яка використовується для настройки персонального блокчейну Ethereum для тестування СК.

4.2.3 Налагодження програми

Дана програма не вимагає яких-небудь налаштувань на склад технічних та програмних засобів.

4.2.4 Перевірка програми

Для перевірки працездатності програми можна скористатися кількома способами:

- при режимі розробки та тестування запустити Ganache для настройки блокчейну і вебзастосунок командою `npm run start` в терміналі. Кожна веб-сторінка повинна бути відображена правильно і дані повинні бути успішно завантажені для відображення;
- у коді програми ставити консольні висновки повідомлень про успішний запуск і виконання програми, наприклад, `Console.log("The application has been successfully launched and is ready to use!")`, а також повідомлення про помилки при виклику винятків. Якщо під час запуску програми не викликаються

ніякі виключення і помилки типу error/warning в браузерній консолі, а також відображаються всі повідомлення про успішний запуск і виконання, то програма справна і готова до роботи.

4.2.5 Повідомлення системному програмісту

У ході виконання і тестування програми програмісту видаються різні повідомлення різних типів (error, log, info) через Console в браузері. Такі повідомлення виникають при виклику будь-яких функцій або винятків, в яких ставляться `Console.log()`, `Console.err()` або `Console.info()`, а також `window.alert()`.

На даний момент існують декілька повідомлень, які можуть видаватися при заході на веб-сторінку, а також при виконанні різних функцій:

- «Failed to load file to IPFS» – не вдалося завантажити файл в IPFS;
- «Data encryption error» – помилка шифрування даних;
- «Data decryption error» – помилка дешифрування даних;
- «Method" ... "of a smart contract does not exist» – метод «...» смарт-контракту не існує;
- «Failed to execute method" ... "of smart contract» – не вдалося виконати метод «...» смарт-контракту.

При запуску також може виводиться модальне вікно, яке показано на рис. 4.1.

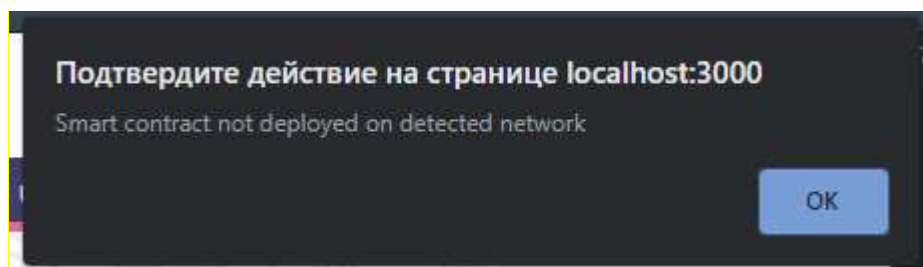


Рисунок 4.1 – Модальне вікно з повідомленням про помилку

Причина: СК ще не розгорнутий.

Рішення: розгорнути СК за допомогою команди `truffle migrate`, після чого перезапустити систему.

4.3 Керівництво програміста

4.3.1 Призначення й умови застосування програми

Основним призначенням децентралізованою системою є організація і контроль навчального процесу, оцінювання ефективності навчання та знань студентів, відстеження дій користувачів, а також надійне і безпечне зберігання наукових робіт в рамках ДН. Дана система дозволяє користувачам виконувати такі функції:

Для адміністратора:

- реєстрація користувачів. Тільки адміністратор може зареєструвати користувачів і зберегти їх в блокчейн за запитами користувачів;
- повний контроль і зміна навчального процесу;
- моніторинг мережі блокчейна.

Для викладачів:

- реєстрація і авторизація. Для реєстрації користувач заповнює форму заявки з деякими особистими даними і відправляє адміністратору для перевірки;
- додавання і редагування нового курсу;
- додавання завдань і матеріалів до курсу;
- додавання учасників (викладачів і студентів) в обраний курс;
- перевірка робіт студентів та оцінювання.
- перегляд журналу оцінок студентів.
- перегляд журналу дій.

Для студентів:

- реєстрація і авторизація. Реєстрація для студентів проводиться за тим же принципом, що і для викладачів;
- перегляд списку курсів;
- вибір потрібного курсу і отримання матеріалів, завдань;

- здача своїх робіт;
- отримання повідомлень і перегляд своїх оцінок;
- перегляд журналу дій.

Для роботи системи необхідний персональний комп'ютер з характеристиками не нижче:

- процесор з тактовою частотою 1 ГГц або вище;
- оперативна пам'ять об'ємом 2 Гб або вище;
- жорсткий диск об'ємом 128 Гб або вище;
- будь-який графічний адаптер;
- монітор з роздільною здатністю екрану не менше 1024x768 пікселів;
- мережевий інтерфейс Ethernet 10/100/100 Base-TX.

Для запуску і виконання системи необхідний браузер Internet Explorer версії 6.0 і вище, або Opera версії 9.0 і вище, або Firefox версії 2.0 і вище, або Google Chrome версії 7.0 і вище, або будь-який браузер з підтримкою JavaScript і cookie. Застосунком можна користуватися у будь-яких операційних системах, які підтримують будь-який з перерахованих браузерів.

4.3.2 Характеристики програми

Система розрахована на цілодобовий режим роботи протягом 365 днів на рік без можливості збою, оскільки система є децентралізованою, де немає центральний сервер, і буде працювати завжди.

Засобами контролю і моніторингу є автоматизовані вбудовані в платформу програмні компоненти, які самостійно виконують такі дії в разі помилки:

- система автоматично перезапускається в разі некритичної помилки з повідомленням користувачеві про помилку і прийняття необхідного перезапуску;
- посилає адміністратору через e-mail звіт про помилку з конкретними даними в тому, де, чому і що сталося.

4.3.3 Звернення до програми

Зовнішнє управління вмістом системи здійснюється через веб-інтерфейс, тому для запуску системи потрібен будь-який Web-браузер з підтримкою JavaScript і cookie. Щоб відкрити вебзастосунок потрібно у адресному рядку Web-браузера ввести адресу сайту.

Для того, щоб передати повне управління всіма необхідними функціями конкретної ролі, необхідно ввести дані для входу в систему або зареєструватися.

При активної розробці використовується програма Ganache, яка для розробників контролює мережу блокчейн Ethereum, в Ganache включається генерація декількох акаунтів при першій запуску платформи блокчейн, а також в системі вбудована тестова функція швидкої реєстрації декількох користувачів адміністратором і входу в систему для обраного згенерованого аккаунта.

Щоб запустити систему і виконувати всі функції, розроблені в системі, програмісту необхідно виконувати ряд кроків:

Крок 1. Відкрити Ganache;

Крок 2. У Ganache вибрати заздалегідь створений workspace. У разі першого запуску, створити новий workspace і вибрати шлях до файлу truffle-config.js, який зберігається у проєкті, потім вибрати деякі параметри (кількість акаунтів та їх баланс, які будуть згенеровані для подальшого тестування);

Крок 3. У коді вказати доступний для програміста порт локальної мережі, за замовчуванням 7545. Потім відкривши термінал в обраному папці проєкту, вводити команду `npm run start` для запуску системи. Терміналом можна користуватися в самому IDE;

Крок 4. У запущеної системі в тестовому списку зареєстрованих користувачів вибрати і виконувати швидкий вхід в систему. Якщо список порожній, то потрібно увійти в систему в ролі адміністратора, потім виконувати тестову функцію для швидкої реєстрації всіх користувачів різних ролей за доступними адресами, згенерованих Ganache.

4.3.4 Вхідні та вихідні дані

Внесення змін до працюючу версію буде прийнято після перезапуску сервера. Перезапуск сервера може бути здійснений як з зупинкою HTTP-сервера і перериванням з'єднань, так і без них.

При активної розробки в IDE Visual Studio Code після внесення змін в кодах при збереженні клавішею Ctrl+S сервер автоматично перезапускається, якщо сервер був запущений локально.

У разі внесення змін СК мовою Solidity необхідно оновити/перестворити workspace в Ganache, потім виконувати команду `truffle migrate`, після чого запустити/перезапустити систему командою `npm run start`.

4.3.5 Повідомлення

В ході виконання і тестування програми програмісту видаються різні повідомлення різних типів (`error`, `log`, `info`) через Console в браузері. Такі повідомлення виникають при виклику будь-яких функцій або винятків, в яких ставляться `Console.log()`, `Console.err()` або `Console.info()`, а також `window.alert()`.

На даний момент існують декілька повідомлень, які можуть видаватися при заході на веб-сторінку, а також при виконанні різних функцій:

- «Failed to load file to IPFS» – не вдалося завантажити файл в IPFS;
- «Data encryption error» – помилка шифрування даних;
- «Data decryption error» – помилка дешифрування даних;
- «Method" ... "of a smart contract does not exist» – метод «...» смарт-контракту не існує;
- «Failed to execute method" ... "of smart contract» – не вдалося виконати метод «...» смарт-контракту.

При запуску також може виводиться модальне вікно, яке показано на рис. 4.2.

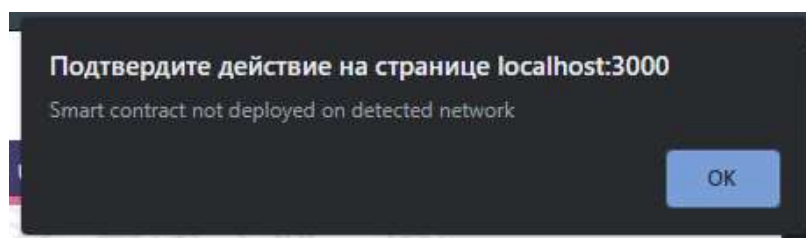


Рисунок 4.2 – Модальне вікно з повідомленням про помилку

Причина: СК ще не розгорнутий.

Рішення: розгорнути СК за допомогою команди `truffle migrate`, після чого перезапустити систему.

4.4 Керівництво оператора

4.4.1 Призначення програми

Дана програма призначена для організації і контролю навчального процесу, оцінювання ефективності навчання та знань студентів, відстеження дій користувачів, а також надійного і безпечного зберігання наукових робіт в рамках ДН.

У програмі є такі функції:

- реєстрація і авторизація;
- повний контроль і зміна навчального процесу;
- моніторинг мережі блокчейну;
- додавання і редагування нового курсу;
- додавання завдань і матеріалів до курсу;
- додавання учасників (викладачів і студентів) в обраний курс;
- вибір потрібного курсу і отримання матеріалів, завдань;
- здача своїх робіт;
- отримання повідомлень і перегляд своїх оцінок;
- перевірка робіт студентів та оцінювання;
- перегляд журналу оцінок студентів;
- перегляд журналу дій.

4.4.2 Умови виконання програми

4.4.2.1 Мінімальний склад апаратних засобів

Для роботи системи необхідний персональний комп'ютер з характеристиками не нижче:

- процесор з тактовою частотою 1 ГГц або вище;
- оперативна пам'ять об'ємом 2 Гб або вище;
- жорсткий диск об'ємом 128 Гб або вище;
- будь-який графічний адаптер;
- монітор з роздільною здатністю екрану не менше 1024x768 пікселів;
- мережевий інтерфейс Ethernet 10/100/100 Base-TX.

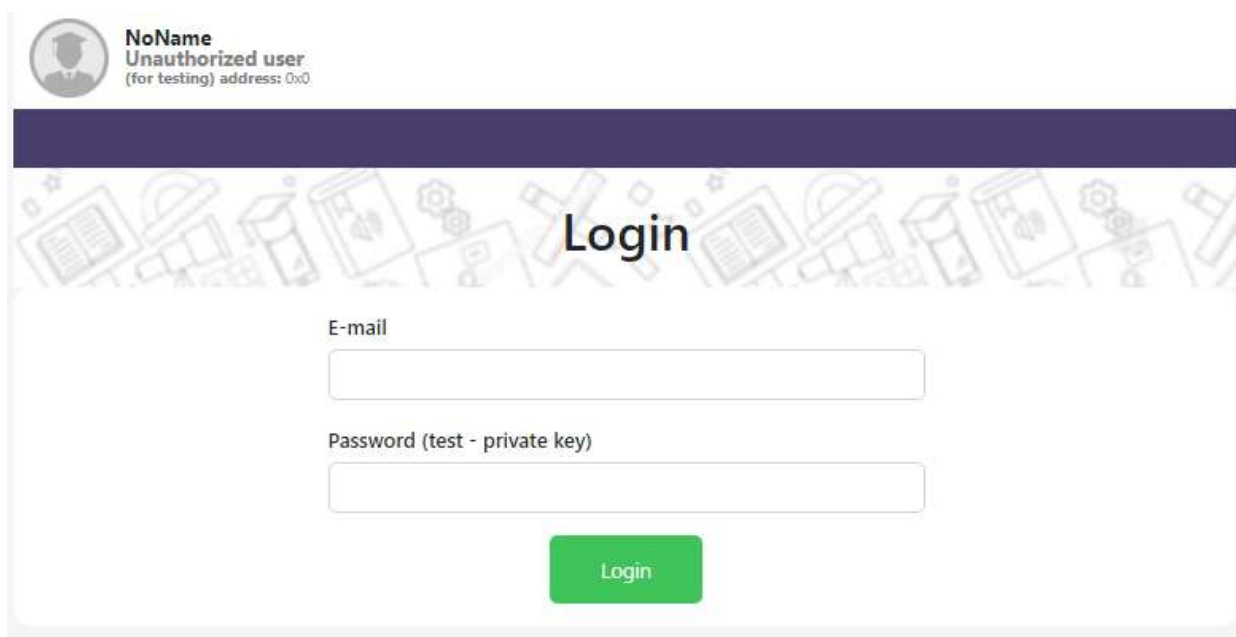
4.4.2.2 Мінімальний склад програмних засобів

Для запуску і виконання системи необхідний браузер Internet Explorer версії 6.0 і вище, або Opera версії 9.0 і вище, або Firefox версії 2.0 і вище, або Google Chrome версії 7.0 і вище, або будь-який браузер з підтримкою JavaScript і cookie. Застосунком можна користуватися у будь-яких операційних системах, які підтримують будь-який з перерахованих браузерів.

4.4.3 Виконання програми

4.4.3.1 Реєстрація та вхід в систему

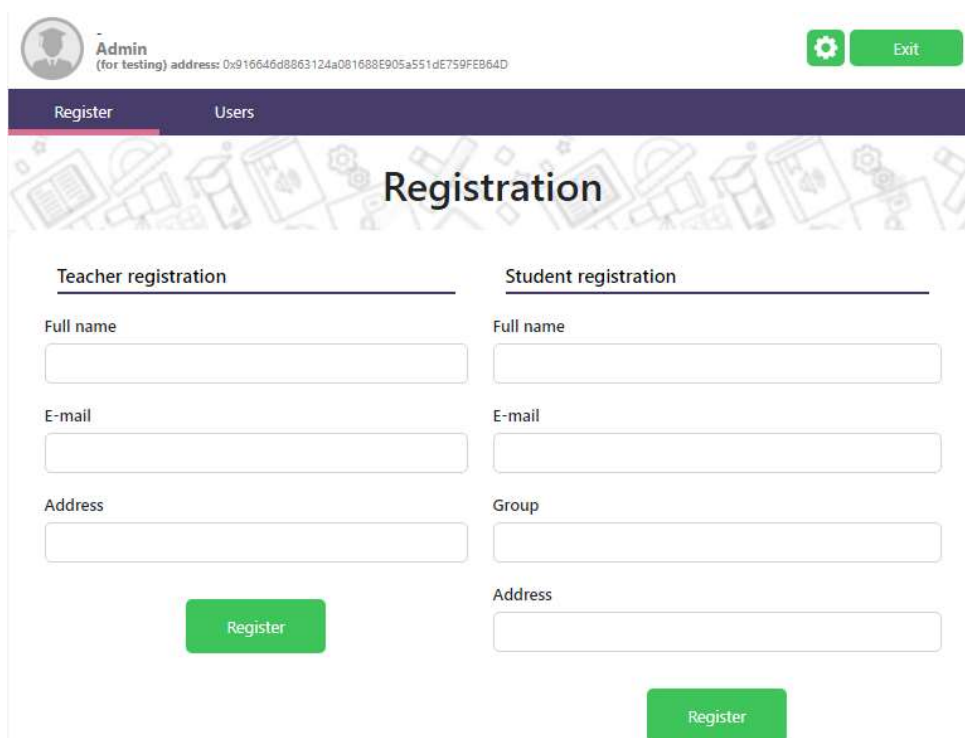
При запуску вебзастосунки відкриється сторінка входу в систему, в якій необхідно ввести свій e-mail і password, якщо був зареєстрований в системі заздалегідь (рис. 4.3).



The screenshot shows the 'Login' interface. At the top left, there is a user profile icon and the text 'NoName Unauthorized user (for testing) address: 0x0'. Below this is a dark purple header bar. The main area has a background pattern of educational icons. The title 'Login' is centered in a large, bold font. Below the title are two input fields: 'E-mail' and 'Password (test - private key)'. A green 'Login' button is positioned below the password field.

Рисунок 4.3 – Форма входу в систему

Коли користувач хоче зареєструватися, він подає заявку адміністратору через іншу підсистему, після чого адміністратор проводить реєстрацію користувача, є 2 форми реєстрації в залежності від ролі користувача (рис. 4.4).



The screenshot shows the 'Registration' interface. At the top left, there is a user profile icon and the text 'Admin (for testing) address: 0x916646d8863124a081688E905a551dE759FEB64D'. To the right are a gear icon and an 'Exit' button. Below this is a dark purple header bar with 'Register' and 'Users' tabs. The main area has a background pattern of educational icons. The title 'Registration' is centered in a large, bold font. Below the title are two columns: 'Teacher registration' and 'Student registration'. Each column has input fields for 'Full name', 'E-mail', and 'Address'. The 'Student registration' column also has a 'Group' field. A green 'Register' button is at the bottom of each column.

Рисунок 4.4 – Форма реєстрації користувачів адміністратором

4.4.3.2 Перегляд списку курсів та управління

На рис. 4.5 представлена головна сторінка (курси за замовчанням) користувача з можливістю переходу по сторінках через навігаційне меню (рис. 4.5, п. 1), на даній сторінці відображається список своїх курсів з можливістю фільтрації (рис. 4.5, п. 3) – всі курси (у тому числі курси, викладач яких не є власником) і тільки свої курси. Є можливість налаштувати свої дані і вийти з системи натиснувши на кнопки з правого верхнього кута (рис. 4.5, п. 4).

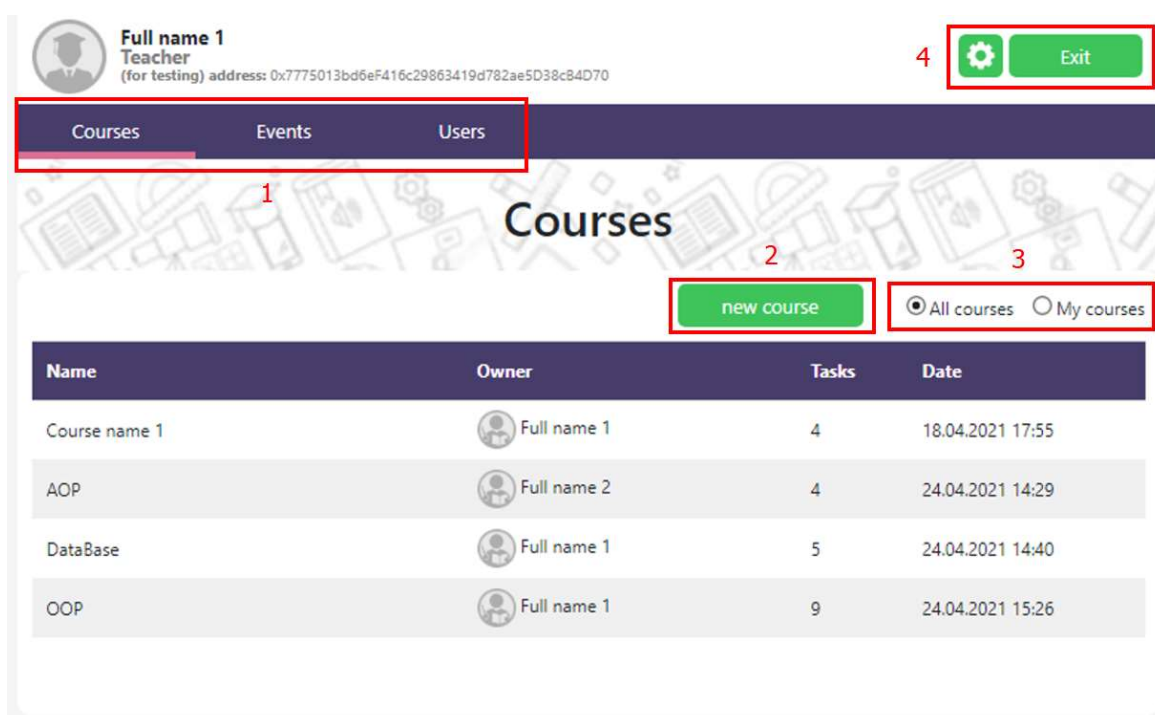


Рисунок 4.5 – Головна сторінка зі списками курсів

На рис. 4.6 відображена сторінка обраного курсу, щоб перейти на таку сторінку необхідно натиснути на назву потрібного курсу зі списку курсів.

На цій сторінці буде відображений список завдань і таблиця оцінок студентів (доступна тільки викладачам), вкладка за замовчуванням буде список завдань (рис. 4.6, п. 2). Управління курсами (додавання завдань, учасників) доступне тільки викладачам. Для того, щоб додавати нове завдання потрібно натиснути на кнопку «new task» (рис. 4.6, п. 3) і в формі ввести назву завдання,

після чого натиснути кнопку для додавання. Для того, щоб додати учасників до курсу потрібно натиснути на кнопку «Add» (рис. 4.6, п. 4) і в формі ввести деякі дані про користувачів, яких потрібно додати.

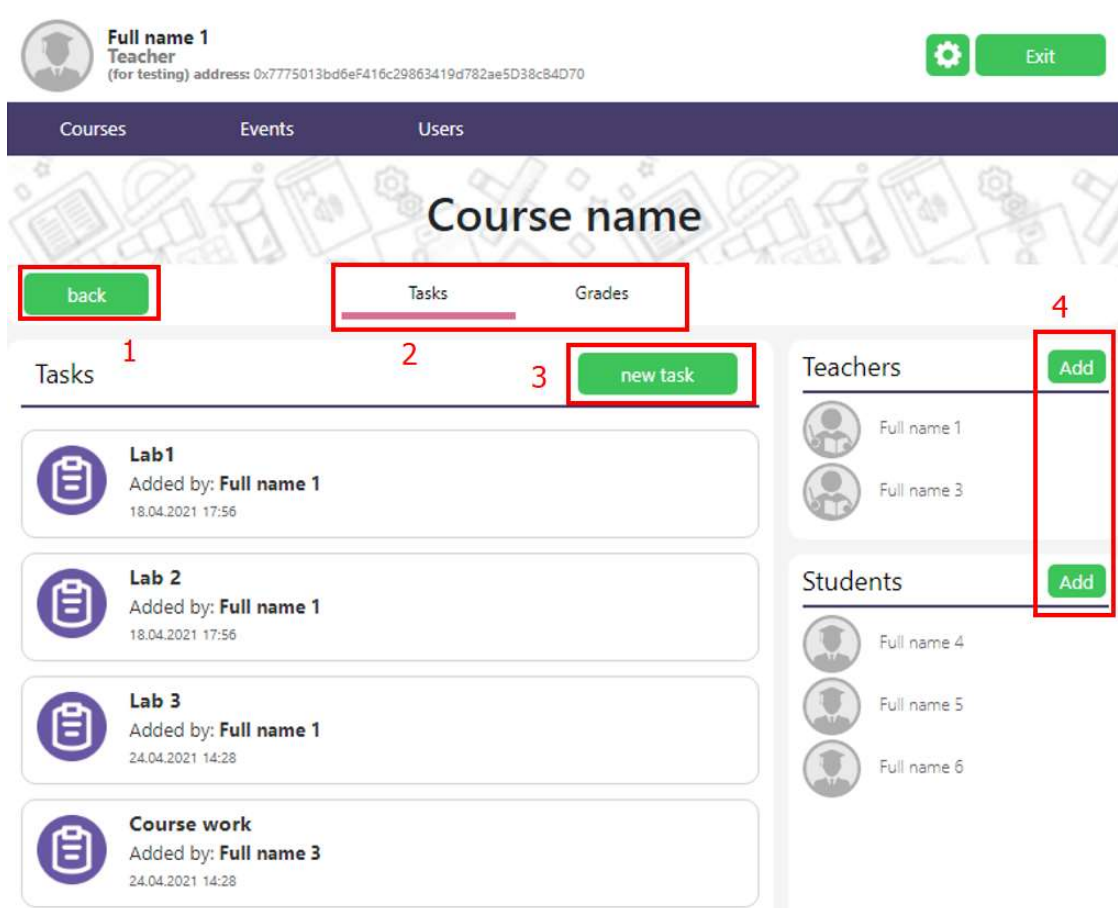


Рисунок 4.6 – Сторінка обраного курсу з управліннями

4.4.3.3 Завантаження файлів, здача робіт і оцінювання

Щоб здати роботи потрібно натиснути на кнопку «вибрати файли» (рис. 4.7, п. 1), після чого відкриється діалогове вікно вибору файлів (рис. 4.8), потім натиснути на кнопку «Hand over» (рис. 4.7, п. 2). Якщо розмір файлу перевищує 1 МБ, то з'явиться повідомлення «Big file!». Якщо файл успішно завантажений, то виконається процес шифрування, при цьому з'явиться напис «Encrypting documents and loading, please wait ...» під кнопки «Hand over».

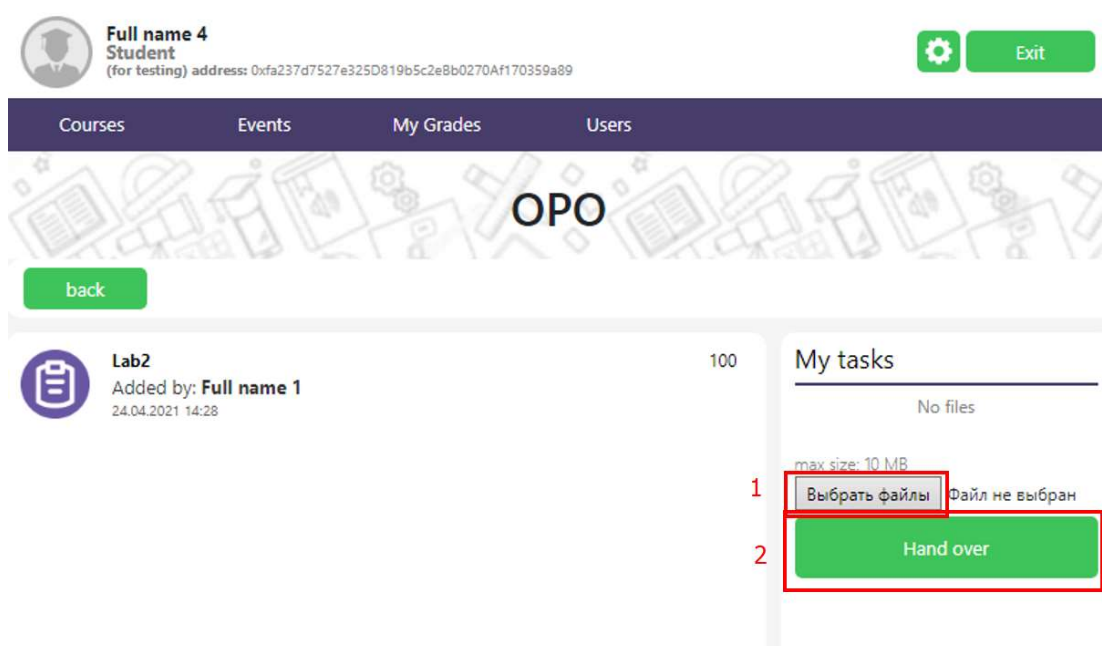


Рисунок 4.7 – Форма завантаження файлу і здача

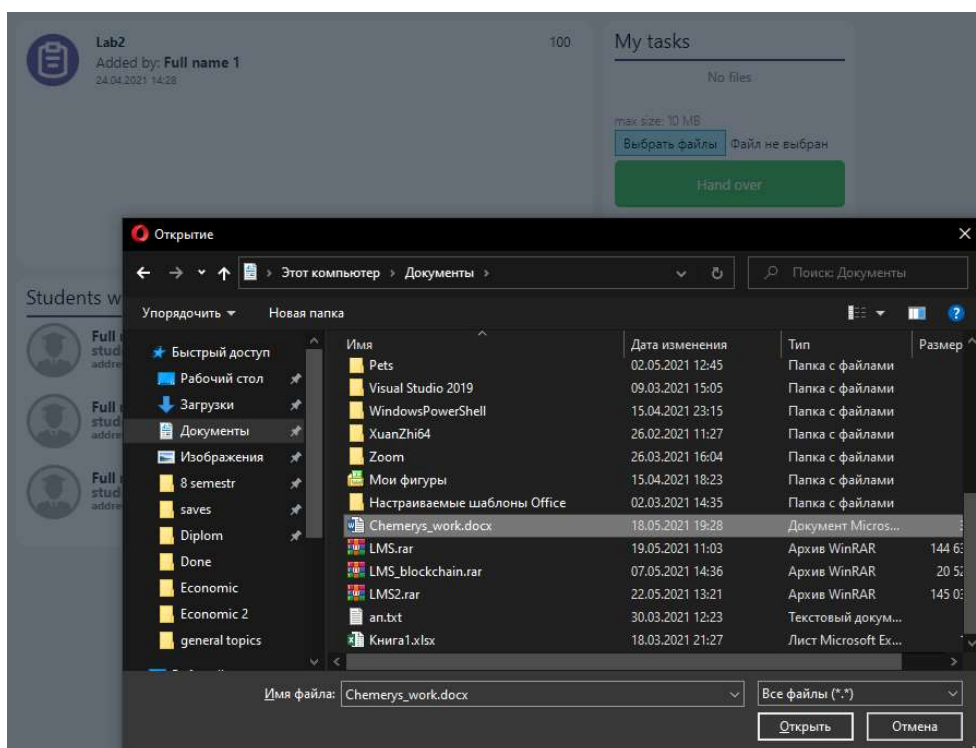


Рисунок 4.8 – Діалогове вікно вибору файлів

Щоб перевірити роботу студента і оцінити, потрібно спочатку вибрати студент зі списку студентів, які здали роботи (рис. 4.9, п. 3), після вибору можна спочатку натиснути на прикріплений файл (рис. 4.9, п. 1) для скачування, при

цьому виконується процес дешифрування. Для оцінювання потрібно вказати число в поле, після чого натиснути на кнопку «Grade» (рис. 4.9, п. 2).

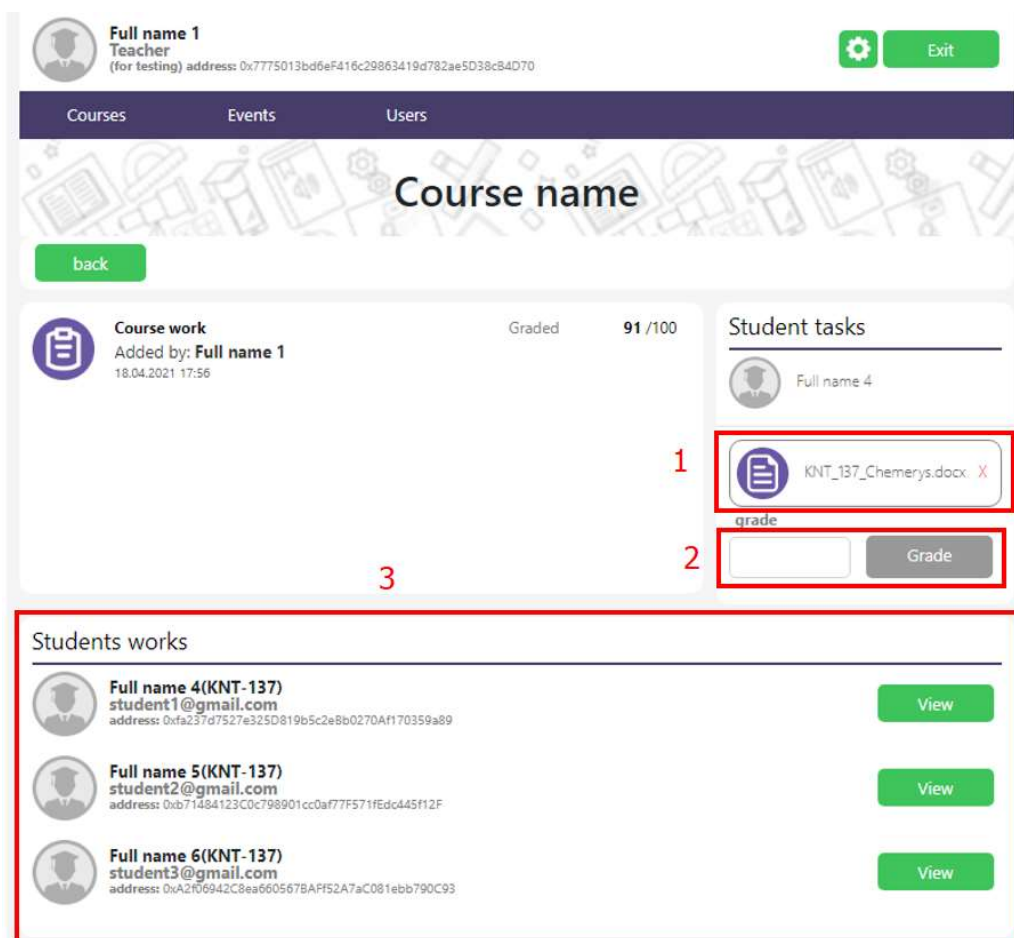


Рисунок 4.9 – Перегляд роботи студентів та оцінювання

4.4.3.4 Перегляд журналу дій і оцінок

Для того щоб переглядати журнал своїх дій необхідно перейти у вкладці «Events» (рис. 4.10, п. 1). За замовчуванням буде відображено список всіх своїх дій за весь час, але є можливість виконати фільтр (рис. 4.10, п. 2). Далі наведено опис кожного параметра фільтрації.

Full name 1
Teacher
(for testing) address: 0x7775013bd6eF416c29863419d782ae5D38c84D70

Courses **Events** Users

1 **Events** 2

event: All events course: All courses from: dd.MM.yy to: dd.MM.yy

Event	Course	Task	Date
create a course	OPO	-	18.04.2021 17:55
add another teacher to a course	OPO	-	18.04.2021 17:55
add students to a course	OPO	-	18.04.2021 17:56
create a new task	OPO	Lab1	18.04.2021 17:56
create a new task	OPO	Course work	18.04.2021 17:56
create a new task	OPO	-	24.04.2021 14:28
create a new task	OPO	Lab2	24.04.2021 14:28
create a course	AOP	-	24.04.2021 14:29
create a course	DataBase	-	24.04.2021 14:40
add another teacher to a course	DataBase	-	24.04.2021 14:41
add students to a course	DataBase	-	24.04.2021 14:41
create a new task	DataBase	Lab1	24.04.2021 14:41
create a course	OOP	-	24.04.2021 15:26
add another teacher to a course	OOP	-	24.04.2021 15:27
add students to a course	OOP	-	24.04.2021 15:27
create a new task	OOP	Course work	24.04.2021 15:27

Рисунок 4.10 – Журнал дій

На рис. 4.11 відображений параметр фільтрації – тип події. За умовчанням вибраний тип – всі події.

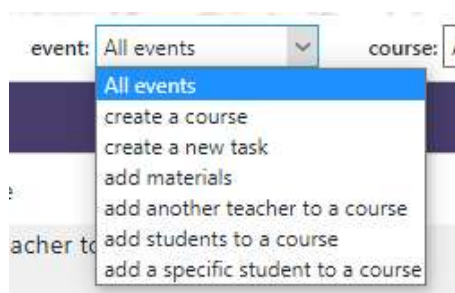


Рисунок 4.11 – Вибір типу подій

На рис. 4.12 відображений параметр фільтрації – курс, тут відображається список своїх доданих курсів. За умовчанням вибраний параметр – всі курси.

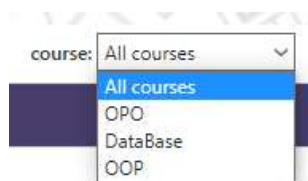


Рисунок 4.12 – Вибір курсу

На рис. 4.13 відображений параметр фільтрації – діапазон дати (від ... до ...). За замовчуванням не обрані значення, тобто будуть відображені всі події за будь-який час.

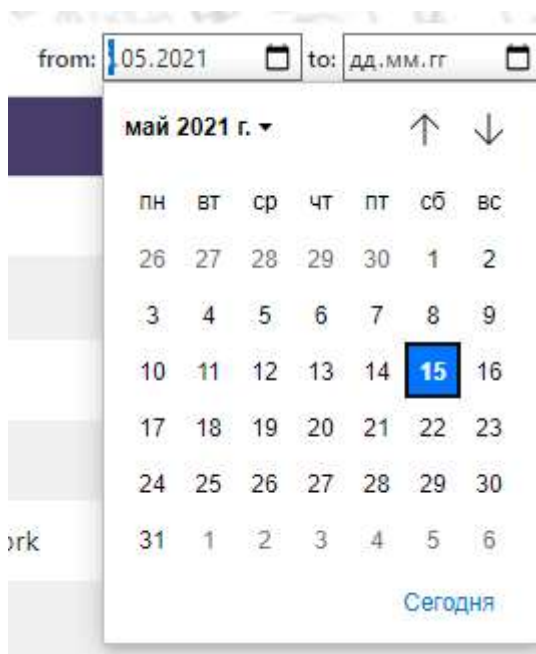
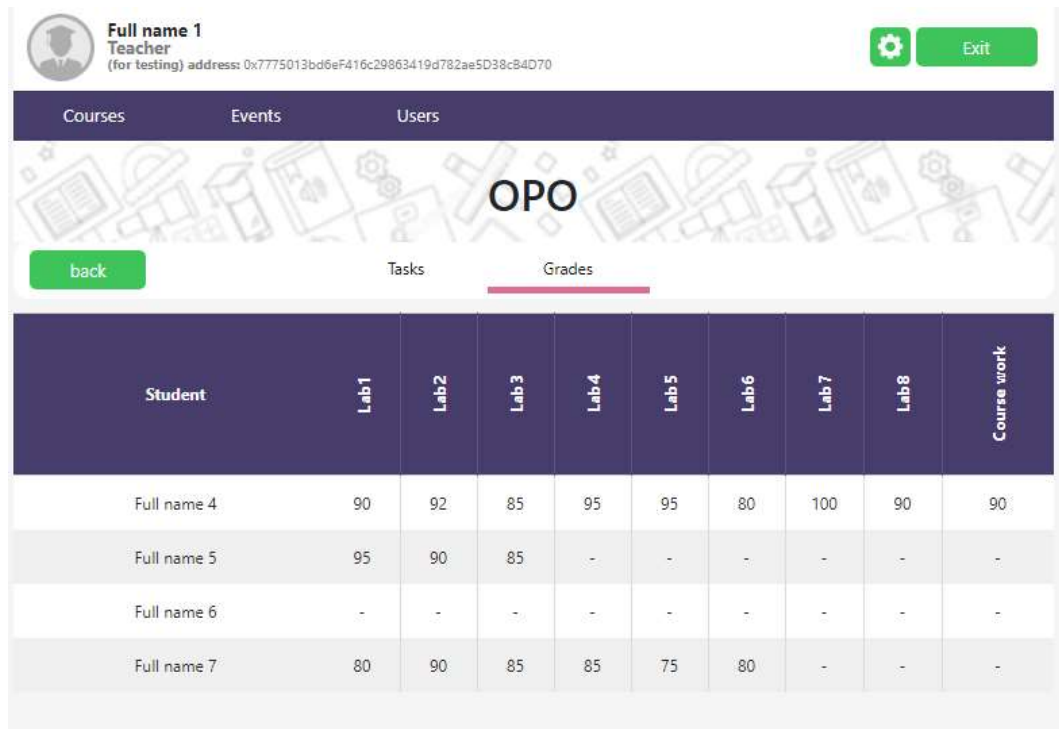


Рисунок 4.13 – Вибір початкової і кінцевої дати

Для того, щоб переглядати журнал оцінок студентів за виконані роботи, необхідно в обраному курсі перейти у вкладці «Grades» (рис. 4.14).

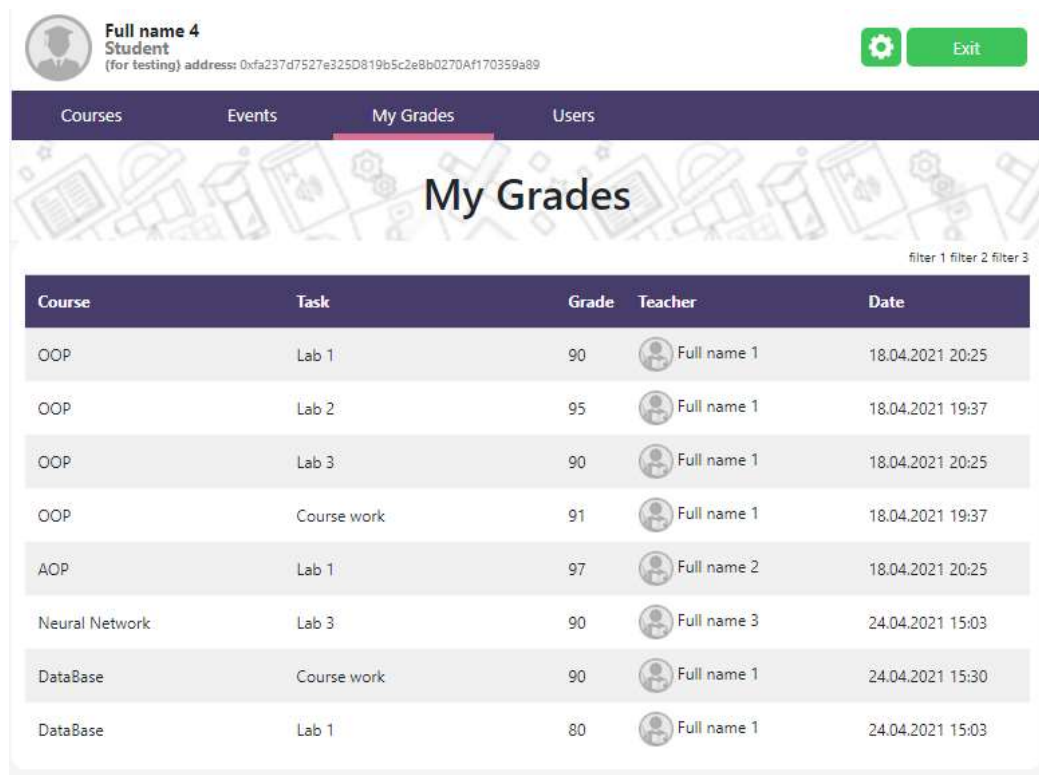


The screenshot shows the OPO Teacher interface. At the top, there is a header with a user profile for 'Full name 1 Teacher' and a green 'Exit' button. Below the header is a navigation bar with 'Courses', 'Events', and 'Users'. The main area is titled 'OPO' and has a 'back' button. Below this is a tabbed interface with 'Tasks' and 'Grades' (selected). The 'Grades' tab displays a table of student grades.

Student	Lab 1	Lab 2	Lab 3	Lab 4	Lab 5	Lab 6	Lab 7	Lab 8	Course work
Full name 4	90	92	85	95	95	80	100	90	90
Full name 5	95	90	85	-	-	-	-	-	-
Full name 6	-	-	-	-	-	-	-	-	-
Full name 7	80	90	85	85	75	80	-	-	-

Рисунок 4.14 – Журнал оцінок студентів

Для того, щоб переглядати журнал своїх оцінок, студент повинен перейти у вкладці «My grades» (рис. 4.15).



The screenshot shows the OPO Student interface. At the top, there is a header with a user profile for 'Full name 4 Student' and a green 'Exit' button. Below the header is a navigation bar with 'Courses', 'Events', 'My Grades' (selected), and 'Users'. The main area is titled 'My Grades' and has filter buttons. Below this is a table of the student's grades.

Course	Task	Grade	Teacher	Date
OOP	Lab 1	90	Full name 1	18.04.2021 20:25
OOP	Lab 2	95	Full name 1	18.04.2021 19:37
OOP	Lab 3	90	Full name 1	18.04.2021 20:25
OOP	Course work	91	Full name 1	18.04.2021 19:37
AOP	Lab 1	97	Full name 2	18.04.2021 20:25
Neural Network	Lab 3	90	Full name 3	24.04.2021 15:03
DataBase	Course work	90	Full name 1	24.04.2021 15:30
DataBase	Lab 1	80	Full name 1	24.04.2021 15:03

Рисунок 4.15 – Журнал своїх оцінок

4.4.4 Повідомлення оператору

У таблиці 4.1 перераховані всі необхідні повідомлення, які будуть відображені в ході виконання роботи програми.

Таблиця 4.1 – Повідомлення, які будуть відображені в ході виконання роботи програми

Повідомлення	Тип	Коли з'явилося
The field cannot be empty	елемент HTML	З'являється в пусте текстове поле при спробі збереження змін даних, при спробі входу в систему і т.д
Invalid email	елемент HTML	з'являється в текстове поле email при спробі збереження даних, входу в систему
No internet connection	Модальне вікно alert	з'являється в оффлайн режимі при виконанні якоїсь дії, в осн. при спробі входу в систему, або при запуску програми
Invalid login or password/key	елемент HTML	з'являється при спробі входу в систему
Failed to login	елемент HTML	з'являється при спробі входу в систему по інших причинах
Generating keys for the user, please wait...	елемент HTML	з'являється під формою при реєстрації користувача адміністратором
Failed to register, please repeat later	елемент HTML	з'являється під формою при реєстрації користувача адміністратором

Продовження табл. 4.1

Повідомлення	Тип	Коли з'явилося
Data changed successfully!	елемент HTML	з'являється при підтвердженні внесення змін
Operation failed!	елемент HTML	з'являється при спробі виконання якоїсь операції
Keys are being generated and a course is being added, please wait...	елемент HTML	з'являється під формою при додаванні курсу
Keys are being generated for the participant, please wait...	елемент HTML	з'являється під формою при додаванні учасника
User with this e-mail does not exist in the system	елемент HTML	з'являється при спробі додавання користувача в список учасників курсу
User added successfully	елемент HTML	з'являється при додавання користувача в список учасників курсу
The file is big!	елемент HTML	з'являється при спробі завантаження файлу розміром більше, ніж вказано
Failed to load file, please try again	елемент HTML	з'являється при спробі завантаження файлу по якійсь причині
File upload in progress...	елемент HTML	з'являється при завантаженні файлу
The file is encrypted, please wait...	елемент HTML	з'являється після завантаження файлу
The file is decrypted, please wait...	елемент HTML	з'являється після скачування файлу

4.5 Методика і результати тестування

4.5.1 Опис методики проведення тестування

Забезпечення якості є невід’ємною частиною життєвого циклу (ЖЦ) розробки будь-яких застосунків. Щоб забезпечити успішну розробку будь-якої програми, фахівець-тестувальник повинен брати участь у всіх етапах розробки – від створення концепції та аналізу вимог, до створення специфікацій тестування та випуску готового продукту. Після розробки необхідно пройти різні тестування застосунка.

Як завжди, найчастіше для більшості застосунків використовуються такі тестування, як тестування верстки, зручності використання, функціональності, безпеки, конфігурації і т.д. Головною метою нашої роботи є застосування блокчейн платформи Ethereum і розробка СК тільки для реалізації підсистеми управління навчанням, а не цілої системи. Інтерфейс даної програми не повноцінний і розроблений тільки для швидкого тестування і взаємодії з платформою Ethereum для управління і зберігання даних про навчальних процесах в блокчейн і IPFS, тому тестування верстки та перевірки зручності не будуть проведені. Вони можуть бути проведені після доопрацювання повноцінної системи перед релізом.

4.5.1.1 Тестування смарт-контрактів. Тестування безпеки

Тестування СК також є важливою метою, оскільки в разі виявлення помилок і вразливостей не можна змінити код розгорнутого СК, опублікованого в блокчейн, – в цьому випадку створюється новий контракт. Навіть невелика помилка або вразливість в логіці програми може приводити до серйозних наслідків.

В даному тестуванні необхідно перевірити працездатність і правильність виконання всіх методів і модифікаторів СК, таких як перевірка аутентифікації і ролі користувача, реєстрація користувачів і інші методи для управління

навчанням. Перед будь-яким методом для управління навчанням повинен бути робочий модифікатор для перевірки автентичності і ролі користувача перш, ніж виконувати завдання.

Перевірити СК можна різними способами, основним з них є написання unit-тестів з використанням JavaScript. Такий спосіб тестування присутня в Truffle.

Для запуску тестів знадобляться Javascript-бібліотеки:

- Mocha - це бібліотека, яка містить загальні функції для тестування;
- Chai - це бібліотека, яка підтримує різноманітні функції для перевірок.

За замовчуванням Mocha є в Truffle і ми можемо без проблем використовувати функції бібліотеки. А Chai необхідно встановити вручну, для цього в терміналі в каталозі проєкту потрібно ввести команду `npm install chai`.

Після розробки СК можна приступити до тестування, для початку потрібно створити файл `.js` в папці `test` і писати в ньому код, який показаний в лістингу 4.1. У лістингу приведений приклад тестування двох методів, для тестування всіх інших методів СК написаний код за аналогічним зразком і протестовано.

Тестування СК виконується командою `truffle test`, при цьому система і Ganache повинні бути запуснені.

Листинг 4.1 – Код для тестування СК

```
require('chai')
  .use(require('chai-as-promised'))
  .should()
//завантаження смарт-контракту
const DecentralizedLMS = artifacts.require('./Education')
contract("Education", function ([account]) {
  let educationContract = null;
  let pKeyAdmin = "2e376a83ea7b315cfd5c6423441d438f4042ec4613a2d7acb6c6e07f6cab8
606";
  var web3Provider = new Web3.providers.HttpProvider('http://localhost:7545');
  var web3 = new Web3(web3Provider);
  beforeEach(async () => {
    //ініціалізація СК
```

```

    educationContract = await DecentralizedLMS.new("2e376a83ea7b315cfd5c6423441d
438f4042ec4613a2d7acb6c6e07f6cab8606")
  })
  describe('testing smart contract...', () => {
    //Вважається успіхом, якщо...
    describe('success', () => {
      // перевірка модератора - чи є адміністратором
      it('is admin', async () => {
        expect(await educationContract.isAdminF(account)).to.eq(true)
      })
      // перевірка функції - реєстрація викладача
      it('teacher registration process', async () => {
        const key = new NodeRSA({ b: 1024 });
        // отримання ключів
        const publicKey = key.exportKey('pkcs8-public-pem');
        const privateKey = key.exportKey('pkcs1-pem');
        // реєстрація викладача адміністратором
        await educationContract.registerTeacher("profile", "Full name 1", "email
1@gmail.com", web3.eth.getAccounts()[1], privateKey, publicKey, pKeyAdmin);
        // результати
        const isAuthF = await educationContract.isAuthF(web3.eth.getAccounts()[1], privateKey);
        const isTeacherF = await educationContract.isTeacherF(web3.eth.getAccounts()[1]);
        expect(isAuthF && isTeacherF).to.eq(true);
      })
    })
  })
  ...
});

```

4.5.1.2 Тестування функціональності програми, розробка чек-листа

Функціональність програми повинна бути повністю протестована. Особливу увагу слід приділити запуску, оновленню, реєстрації та вході в систему, забезпечення, роботі зі специфічними функціями і повідомленнями про помилки.

В рамках функціонального тестування, необхідно розробити чек-лист і виконати всі тести.

При проходженні чек-листів необхідно відзначити статус навпроти кожного протестованого пункту. У таблиці 4.2 наведено всі можливі варіанти статусів.

Таблиця 4.2 – Всі можливі варіанти статусів

Passed	Перевірка пройдена успішно, багів не знайдені.
Failed	Знайдений один або більше багів.
Blocked	Неможливо перевірити, тому що один з багів блокує поточну перевірку.
Not run	Ще не перевірено.
Skipped	Перевірка не буде з будь-якої причини. Наприклад, поточний функціонал ще не реалізований.
In Progress	Поточний пункт, над яким працює тестувальник.

Після завершення тестування не повинно бути осередків, позначених «Not run».

4.5.2 Результати тестування

Після проведення тестування системи були отримані всі результати, які пояснюють про те, що підсистема працює нормально без проблем і не викликається якийсь збій в системі. На рис. 4.15 наведені результати unit-тестування СК, все методи СК працюють правильно і безпечно, в кожному методі присутні необхідні модифікатори.

```

Администратор: C:\Windows\system32\cmd.exe

Contract: Education
testing smart contract...
success
  ✓ is admin (111ms)
  ✓ teacher registration process (587ms)
  ✓ teacher registration process (581ms)
  ✓ student registration process (683ms)
  ✓ course adding process (475ms)
  ✓ task adding process (379ms)
  ✓ participant adding process (1587ms)
  ✓ files adding process (1081ms)
  ✓ delivery of work by a student (791ms)
  ✓ assessment of work by the teacher (188ms)
  ✓ event recording (91ms)

11 passing (11s)

C:\Users\kiril\Documents\LMS>

```

Рисунок 4.15 – Результати тестування СК

У таблиці 4.3 представлений заповнений чек-лист після проходження тестування функціональності.

Під час даного тестування було перевірено:

- роботу всіх обов'язкових функцій застосунку;
- працездатність різних форм користувача у застосунку;
- додавання, редагування та видалення інформації тощо;
- процес шифрування і дешифрування даних;
- аутентифікація кожної ролі користувача.

Таблиця 4.3 – Чек-лист після проходження тестування функціональності

Чек-лист функціонального тестування				
№	DApp-застосунок управління навчанням	Windows		
		Google Chrome 90.0.4430	Opera 76.0.4017.154	Microsoft Edge 90.0.818
1	Реєстрація та особистий профіль			
2	Перевірити реєстрацію користувача (для адміністратора)	Passed	Passed	Passed
3	Перевірити вхід в систему	Passed	Passed	Passed
4	Перевірити редагування профілю	Passed	Passed	Passed
5	Перевірити валідацію полів	Passed	Passed	Passed
6	Незарєєстрований користувач (спроба перегляду і управління курсами і т.д.)	Passed	Passed	Passed
7	Вихід із системи	Passed	Passed	Passed
8	Перегляд курсу і завдань			
9	Перевірити конкретність відображення списку курсів	Passed	Passed	Passed
10	Перевірити фільтр списку курсів	Skipped	Skipped	Skipped
11	Перевірити перехід на сторінку обраного курсу	Passed	Passed	Passed
12	Перевірити конкретність відображення списку завдань	Passed	Passed	Passed
13	Перевірити перехід на сторінку обраного завдання	Passed	Passed	Passed
14	Перевірити конкретність відображення списку викладачів	Passed	Passed	Passed
15	Перевірити конкретність відображення списку студентів	Passed	Passed	Passed
16	Журнал дій і оцінок			
17	Перевірити конкретність відображення списку дій	Passed	Passed	Passed

Продовження табл. 4.3

Чек-лист функціонального тестування				
№	DApp-застосунок управління навчанням	Windows		
		Google Chrome 90.0.4430	Opera 76.0.4017.154	Microsoft Edge 90.0.818
18	Перевірити конкретність додавання всіх різних видів дій	Passed	Passed	Passed
19	Перевірити фільтр списку дій	Passed	Passed	Passed
20	Перевірити конкретність відображення журналу оцінок	Passed	Passed	Passed
21	Управління курсами			
22	Перевірити обмеження студента на управління курсами	Passed	Passed	Passed
23	Створення нового курсу з генерацією власних ключів курсу	Passed	Passed	Passed
24	Перевірити, додається чи новий курс в список	Passed	Passed	Passed
25	Створення нового завдання	Passed	Passed	Passed
26	Перевірити, додається чи нове завдання в список	Passed	Passed	Passed
27	Перевірити додавання учасників до обраному курсу (студента по e-mail, студентів по групі, іншого викладача по e-mail з генерацією ключів курсу)	Passed	Passed	Passed
28	Перевірити додавання матеріалів до курсу	Skipped	Skipped	Skipped
29	Здача робіт і оцінювання			
30	Перевірити додавання файлів (процес шифрування ключем курсу, додавання в IPFS і в блокчейн)	Passed	Passed	Passed
31	Перевірити валідацію доданих файлів	Failed (баг 0XXX1)	Failed (баг 0XXX1)	Failed (баг 0XXX1)
32	Перевірити процес шифрування доданих файлів ключем курсу і додавання даних в IPFS і блокчейн	Passed	Passed	Passed
33	Здача робіт в формі	Passed	Passed	Passed
34	Перевірити, чи можна скасувати задачу роботи	Skipped	Skipped	Skipped
35	Перевірити конкретність відображення списку зданих робіт студентів	Passed	Passed	Passed
36	Перевірити скачування файлу (процес дешифрування і скачування)	Passed	Passed	Passed
37	Перевірити виставлення оцінки	Passed	Passed	Passed

Всі знайдені баги можуть бути занесені в системі відслідковування помилок у вигляді баг-репортів. У таблиці 4.4 відображено список багів, які отримані в ході тестування. Всі знайдені функціональні баги не критичні і не призводять до серйозних наслідків. Підсистема управління навчанням працює нормально і виконує всі необхідні функції без проблем, залишилися невеликі проблеми з обмеженням завантаження файлів, які необхідно виправити.

Таблиця 4.4 – Перелік багів

1	Id	0XXX1
	Опис	Не працює перевірка розміру завантаженого файлу у формі задачі робіт
	Фактичний результат	При додаванні файлів в формі задачі робіт виконується процес перевірки типу і розмір кожного файлу. Файл розміром, який перевищує максимальний допустимий розмір, завантажуються без проблем і не виводиться повідомлення про помилку.
	Очікуваний результат	При додаванні файлів виконується процес перевірки типу і розмір кожного файлу. Якщо тип файлу не підтримується або якщо розмір файлу перевищує максимальний допустимий розмір, то скасовується додавання файлу і виводиться повідомлення про помилку.

ВИСНОВКИ

В процесі виконання роботи була проаналізована і вирішена задача про підходи до застосування технології блокчейн в області навчання з метою підвищення організації навчального процесу, безпеки і надійності зберігання даних про навчаннях, а також розроблена програмна підсистема управління навчанням з використанням технології блокчейн.

З метою отримання підходів до застосування технології блокчейн в системі управління навчанням був проведений теоретичний аналіз предметної області, вивчені принципи роботи, типи і особливості технології блокчейн, а також розглянуті історії розвитку і переваги використання технології блокчейн в сфері освіти.

Далі була розроблена архітектура програми. На цьому етапі були визначені межі та структура програми, розроблені та описані абстрактні моделі системи на мові UML на основі структурних діаграм, діаграм поведінки, а також діаграм взаємодії. Визначені можливі вразливі місця і представлені запропоновані функціональні схеми для підвищення безпеки системи. Розроблений дизайн системи використовуючи ескізи з описами. Був проведений аналіз існуючих блокчейн платформ, мов програмування і середовищ розробки програми, виявлені їх переваги та недоліки, а також обґрунтований їх вибір.

На етапі розробки програми була вивчена архітектура традиційного вебзастосунку та децентралізованого застосунку, визначена різниця між ними і розроблена децентралізована система на блокчейн платформі Ethereum з використанням розроблених смарт-контрактів і децентралізованого сховища IPFS. Спочатку були визначені окремі компоненти програми для їх розробки, докладно описана структура проєкту, описані структура смарт-контракту і процес розробки децентралізованого застосунку.

Після розробки програми були проведені різні тестування, такі як тестування працездатності, безпеки і точності виконання всіх методів СК за допомогою написаних unit-тестів на JavaScript, а також тестування

функціональності системи. Для тестування всіх існуючих функцій підсистеми був розроблений чек-лист і заповнений результатами тестування. На стадії розробки, в ході тестування були виявлені і описані кілька помилок і були виправлені. Дана програма працює нормально, виконує всі функції без проблем і не викликає якийсь збій в системі.

Надалі дана підсистема може бути удосконалено та розширено для повного управління навчанням.

ПЕРЕЛІК ПОСИЛАНЬ

1. Технология Блокчейн в Образовании [Электрон. ресурс] / А. Бойко – Режим доступа: <https://merehead.com/ru/blog/how-use-blockchain-education-industry/>.
2. Возможности использования технологий распределенного реестра (цепочек блоков/blockchain) в сфере образования [текст] / С.Д. Каракозов, А.Ю. Уваров, 2018 17 с.
3. Блокчейн: что это такое простыми словами, как работает технология blockchain и преимущества использования [Электрон. ресурс] / Режим доступа: <https://alpari.com/ru/beginner/glossary/blockchain/>.
4. AN INTRODUCTION TO BLOCKCHAIN [Electron. resource]/ Nick Revych – Access mode: <https://www.mindtory.com/an-introduction-to-blockchain/>.
5. Преимущества и недостатки технологии блокчейн [Электрон. ресурс] / Yulia Rieth – Режим доступа: <https://magazine.decenter.org/ru/1-blokchein-i-kriptovalyuty/2-preimushestva-i-nedostatki-tekhnologii-blokchein>.
6. Технологія блокчейн: перспективи розвитку в Україні [текст] / І.В. Давидова, 2017 р. – 4 с.
7. Блокчейн для чайников [текст] / Тиана Лоуренс, – Москва, Санкт-Петербург, Киев, 2018 г. – 274 с.
8. Блокчейн от А до Я [текст] / Лоран Лелу, 2017 г. – 288 с.
9. В чём разница между частным и публичным блокчейном [Электрон. ресурс] / Антипов Г., 2018 г. – Режим доступа: <https://coinspot.io/beginners/v-chyom-raznica-mezhdu-chastnym-i-publichnym-blokchejnom/>.
10. Everything you Wanted to Know about the Blockchain [text] / Depak Puthal, Nisha Malik, Saraju P. Mohanty, 2018 г. – 9 р.
11. Алгоритмы Консенсуса в Блокчейне: POW, POS и другие [Электрон. ресурс] / 2020 г. – Режим доступа: <https://magazine.decenter.org/ru/1-blokchein-i-kriptovalyuty/2-preimushestva-i-nedostatki-tekhnologii-blokchein>.

12. Что такое Алгоритм Консенсуса в Blockchain? [Электрон. ресурс] – Режим доступа: <https://academy.binance.com/ru/articles/what-is-a-blockchain-consensus-algorithm>.

13. Обзор 9 алгоритмов блокчейн консенсуса [Электрон. ресурс] / Григорчук К., 2018 г. – Режим доступа: <https://digiforest.io/blog/blockchain-consensus-algorithms>.

14. Proof of Work (PoW) [Electron. resource] / Jake Frankenfield, 2021, – Access mode: <https://www.investopedia.com/terms/p/proof-work.asp>.

15. Алгоритмы консенсуса в блокчейне [Электрон. ресурс] / Лецюк М., 2020 г. – Режим доступа: <https://3commas.io/ru/blog/algoritmy-consensusa-v-blockchaine>.

16. Что такое атака 51%? [Электрон. ресурс] / 2020 г. – Режим доступа: <https://forklog.com/chto-takoe-ataka-51/>.

17. Proof-of-Work и Proof-of-Stake: Преимущества и недостатки [Электрон. ресурс] / 2018 г. – Режим доступа: <https://cryptocurrency.tech/proof-of-work-i-proof-of-stake-preimushhestva-i-nedostatki/>.

18. Блокчейн – принципы и основы [текст] / Цихилов А., 2019 г. – 352 с.

19. Delegated-Proof-of-Stake как альтернатива PoW и PoS: сравнение и перспективы [Электрон. ресурс] / Шпак К. – Режим доступа: <https://forklog.com/sp/DPoS/>.

20. Виды атак на блокчейн и умные контракты [текст] / Г.Г. Трубач. – 4 с.

21. Атака Сибиллы (Sybil attack) - многочисленные поддельные идентичности [Электрон. ресурс] – Режим доступа: <https://botcoin.online/kriptovalyuty/ataka-sibilly-sybil-attack-mnogochislennye-poddel-nye-identichnosti>.

22. Блокчейн для бабушки за 60 минут [текст] / Е. Хата, 2018 г. – 107 с.

23. Что такое Атака Повторного Воспроизведения? [Электрон. ресурс] – Режим доступа: <https://academy.binance.com/ru/articles/what-is-a-replay-attack>.

24. Блокчейн 5.0: как эволюционировала технология [Электрон. ресурс] – Режим доступа: <https://letknow.news/publications/blokcheyn-50-kak-evolyucionirovala-tehnologiya-33003.html>.
25. Блокчейн. Схема новой экономики [текст] / Мелани Свон, 2015 г. – 52 с.
26. Blockchain – Блокчейн [Электрон. ресурс] – Режим доступа: <https://www.it.ua/ru/knowledge-base/technology-innovation/blockchain>.
27. Relictum PRO - Blockchain 5.0 последнего поколения [Электрон. ресурс] – Режим доступа: <https://tirus.info/ru/about/Relictum-PRO>.
28. Seele - Blockchain 4.0 [Электрон. ресурс] / 2018 г. – Режим доступа: <https://steemit.com/seele/@volcha/seele-blockchain-4-0>.
29. Seele — Blockchain 4.0. Обзор. [Электрон. ресурс] / 2018 г. Некрасов Н. – Режим доступа: https://medium.com/@BlockchainRG_RU/seele-blockchain-4-0-obzor-3dd09067c635.
30. Сферы применения блокчейна [Электрон. ресурс] – Режим доступа: <https://polygant.net/ru/blog/sfery-primeneniya-blokcheyna/>.
31. Блокчейн как коммуникационная основа формирования цифровой экономики: преимущества и проблемы [текст] / М.М. Пряников, А.В. Чугунов, 2017 г. – 7 с.
32. Смарт-контракт [Электрон. ресурс] – Режим доступа: <https://bits.media/tag/smart-contract/>.
33. Смарт-контракты и платформы для их реализации [Электрон. ресурс] / 2018 г. – Режим доступа: <https://decenter.org/ru/smart-kontrakty-i-platformy-dlya-ikh-realizatsii>.
34. Что такое смарт-контракты, где можно их применять и при чём тут блокчейн [Электрон. ресурс] / Шишко Д., 2019 г. – Режим доступа: <https://tproger.ru/blogs/smart-contracts/>.
35. Смарт-контракты: создание, возможности, перспективы [Электрон. ресурс] / 2018 г. – Режим доступа: <https://coinpost.ru/p/263-vsya-pravda-pro-smart-kontrakty-sozdanie-vozmozhnosti-perspektivy>.

36. Автономность и безопасность. Как работают смарт-контракты [Электрон. ресурс] / 2021 г. – Режим доступа: <https://www.rbc.ru/crypto/news/600bd6409a79473b23a6d3c4>.

37. Преимущества и недостатки смарт-контрактов [Электрон. ресурс] – Режим доступа: <https://polygant.net/ru/blog/preimushhestva-i-nedostatki-smart-kontraktov/>.

38. The Blockchain for Education: An Introduction [Electron. resource] / Audrey Watters, 2016. – Access mode: <http://hackeducation.com/2016/04/07/blockchain-education-guide>.

39. Sony develops system for authentication, sharing, and rights management using blockchain technology [Electron. resource] / Sony Global Education team, 2017. – Access mode: <https://www.sonyged.com/2017/08/10/news/press-blockchain/>.

40. Что такое Learning Management System (LMS) и как с ее помощью управлять обучением [Электрон. ресурс] / 2018 г. – Режим доступа: <https://www.ispring.ru/elearning-insights/что-такое-lms>.

41. Возможности использования технологий распределенного реестра (цепочек блоков / blockchain) в сфере образования [текст] / С.Д. Каракозов, А.Ю. Уваров, 2018 г. – 17 с.

42. Объединяя блокчейн и образование [Электрон. ресурс] / 2018 г. – Режим доступа: <https://vc.ru/education/53595-obedinyaya-blokcheyn-i-obrazovanie>.

43. The Blockchain in Education study has been designed and supported by the European Commissions Joint Research Centre's (LRC) [text] / Grech A. Anthony F. Camilleri, 2017, – 136 p.

44. Блокчейн как инструмент цифровой экономики в образовании [текст] / В.П. Кузнецова, И.А. Бондаренко, 2018 г. – 8 с.

45. Блокчейн в образовании – применение и перспективы технологии [Электрон. ресурс] – Режим доступа: <https://prostocoin.io/blog/blockchain-education>.

46. Fujitsu и Sony запустили блокчейн-систему, которая расскажет работодателям правду об образовании соискателей [Электрон. ресурс] / 2019 г. –

Режим доступа: <https://www.iksmedia.ru/news/5567678-Fujitsu-i-Sony-zapustili-blokchejns.html>.

47. University of Melbourne to issue recipient-owned blockchain records [Electron. resource] / 2017. – Access mode: <https://about.unimelb.edu.au/newsroom/news/2017/october/university-of-melbourne-to-issue-recipient-owned-blockchain-records>.

48. Корейский EduTech-стартап планирует использовать блокчейн в современных LMS [Электрон. ресурс] / 2020 г. – Режим доступа: <https://sberbank-university.ru/edutech-club/pulse/novosti/1495/>.

49. Что такое Hyperledger Fabric? [Электрон. ресурс] – Режим доступа: <https://www.ibm.com/ru-ru/topics/hyperledger>.

50. Эфириум (Ethereum): что это, как работает, преимущества и недостатки [Электрон. ресурс] / 2019 г. – Режим доступа: <https://informjournal.ru/efirium-ethereum-chto-eto-kak-rabotaet-preimushhestva-i-nedostatki/>.

51. Ethereum переходит на модель Proof of Stake [Электрон. ресурс] / 2021 г. – Режим доступа: <https://itc.ua/news/ethereum-perehodit-na-model-proof-of-stake-majning-na-videokartah-stanet-bessmyslennym/>.

52. Explore Decentralized Applications [Electron. resource] – Access mode: <https://www.stateofthedapps.com>

53. Обзор криптовалюты EOS [Электрон. ресурс] / Скворцов Ю., 2018 г. – Режим доступа: <https://coinlife.com/useful/obzor-kriptovaljuty-eos-ubijca-ethereum-nanosit-pervyj-udar/>.

54. Топ-10 блокчейн-платформ [Электрон. ресурс] / 2018 г. – Режим доступа: <https://investfuture.ru/news/id/top-10-blokcheyn-platform>.

55. Блокчейн. Разработка приложений [текст] / Нараян Прасти, Санкт-Петербург, «БХВ-Петербург» 2018 г. – 252 с.

56. Смарт-контракт [Электрон. ресурс] – Режим доступа: ru.wikipedia.org/wiki/Смарт-контракт.

57. Обзор Atom [Электрон. ресурс] / 2019 г. – Режим доступа: <https://bizzapps.ru/p/atom-io/>.

58. Обзор Visual Studio Code [Электрон. ресурс] / 2019 г. – Режим доступа: <https://bizzapps.ru/p/vs-code/>.

59. Лучшие редакторы кода и IDE для JavaScript-разработчиков [Электрон. ресурс] / 2019 г. – Режим доступа: <https://techrocks.ru/2019/12/05/best-code-editors-for-javascript-developers/>.

ДОДАТОК А
Текст смарт-контракту

У лістингу А.1 наведений код смарт-контракту з коментарями.

Лістинг А.1 – Код смарт-контракту з коментарями

```
pragma solidity >=0.4.22 <0.9.0;

// Смарт-контракт "Навчання"
contract Education {
    address private adminAddress; // адрес адміністратора
    bytes32 private adminEnPKey; // хеш приватного ключа адміністратора

    // конструктор смарт-контракту
    constructor(string memory pKey) public {
        adminAddress = msg.sender;
        adminEnPKey = sha256(abi.encodePacked(pKey));
    }

    //об'єкт - Студент
    struct Student {
        uint256 id;
        string profileIPFS;
        string fullName;
        string email;
        string group;
        bytes32 encryptedPrivateKey; //хеш приватного ключа (для аутентифікації)
        string publicKey; //публічний ключ
    }

    //об'єкт - Викладач
    struct Teacher {
        uint256 id;
        string profileIPFS;
        string fullName;
        string email;
        bytes32 encryptedPrivateKey; //хеш приватного ключа (для аутентифікації)
        string publicKey; //публічний ключ
    }

    //об'єкт - Завдання
    struct Task {
        uint256 id;
        uint256 course_id;
        string name;
        uint256 timeOfCreation;
        uint256 maxGrade;
        address owner; //власник (хто додав це завдання)
    }

    //об'єкт - Курс
    struct Course {
        uint256 id;
        string name;
        uint256 timeOfCreation;
        address owner; //власник (хто додав цей курс)
        string coursePublicKey; //публічний ключ курсу, використовується студент
ами для шифрування своїх робіт
    }

    //об'єкт - Учасник курсу
    struct Participant {
        uint256 course_id;
```

```

        address user;
        string courseEnPrivKey; //зашифрований приватний ключ курсу ключем викладача, використовується для розшифрування робіт студентів
    }

    //об'єкт - Робота студента
    struct StudentWork {
        uint256 id;
        uint256 course_id;
        uint256 task_id;
        address student;
        string fileName; //ім'я файлу
        string fileType; //тип файлу
        string fileIPFS; //хеш файлу (шлях до повного файлу в IPFS)
    }

    uint256 public studentsCount; //кількість студентів в блокчейне (ця змінна потрібна, тому що не працює length() або size() карти, масива)
    uint256 public teachersCount;
    mapping(uint256 => address) public studentsAddr; // карта (id - адреса студента)
    mapping(address => Student) public students; // карта (адреса студента - об'єкт студента)
    mapping(uint256 => address) public teachersAddr; // карта (id - адреса препода)
    mapping(address => Teacher) public teachers; // карта (адреса препода - об'єкт препода)

    uint256 public courseTeachersCount;
    uint256 public courseStudentsCount;

    mapping(uint256 => Participant) public courseTeachers; //учасники (викладачі) курсу
    mapping(uint256 => Participant) public courseStudents; //учасники (студенти) курсу

    uint256 public coursesCount; //кількість курсів у блокчейн (ця змінна потрібна, тому що не працює length() або size() карти, масива)
    mapping(uint256 => Course) public courses; //курси
    uint256 public tasksCount;
    mapping(uint256 => Task) public tasks; //завдання

    uint256 public studentWorksCount;
    mapping(uint256 => StudentWork) public studentWorks; //роботи студентів

    //функція реєстрації студента
    function registerStudent(
        string memory _profileIPFS,
        string memory _fullName,
        string memory _email,
        string memory _group,
        address _address,
        string memory _privateKey, //хеш приватного ключа. Він потрібен для аутентифікації аккаунта
        string memory _publicKey,
        string memory _enPKey
    ) public isAdmin isAuth(_enPKey) {
        studentsAddr[studentsCount] = _address;

        students[_address] = Student({
            id: studentsCount,
            profileIPFS: _profileIPFS,
            email: _email,

```

```

        fullName: _fullName,
        group: _group,
        encryptedPrivateKey: sha256(abi.encodePacked(_privateKey)), //хешува
ння приватного ключа
        publicKey: _publicKey
    });
    studentsCount++;
}

//функція реєстрації викладача
function registerTeacher(
    string memory _profileIPFS,
    string memory _fullName,
    string memory _email,
    address _address,
    string memory _privateKey, //хеш приватного ключа. Він потрібен для ауте
нтифікації аккаунта
    string memory _publicKey,
    string memory _enPKey
) public isAdmin isAuth(_enPKey) {
    teachersAddr[teachersCount] = _address;

    teachers[_address] = Teacher({
        id: teachersCount,
        profileIPFS: _profileIPFS,
        fullName: _fullName,
        email: _email,
        encryptedPrivateKey: sha256(abi.encodePacked(_privateKey)), //хешува
ння приватного ключа
        publicKey: _publicKey
    });
    teachersCount++;
}

// модифікатор - чи є адміном
modifier isAdmin {
    require(isAdminF(msg.sender));
    _;
}

// модифікатор - чи є студентом
modifier isStudent(address _address) {
    require(isStudentF(_address));
    _;
}

// модифікатор - чи є викладачем
modifier isTeacher(address _address) {
    require(isTeacherF(_address));
    _;
}

// модифікатор - не авторизований чи користувач
modifier isUnauthUser(address _address) {
    require(!isTeacherF(_address) && !isStudentF(_address));
    _;
}

// модифікатор - чи є учасником певного курсу (чи є доступ до курсу)
modifier isAccessToCourse(address _address, uint256 course_id) {
    require(isAccessToCourseF(_address, course_id));
    _;
}

```

```

// модифікатор - аутентифікація (авторизований чи користувач)
modifier isAuth(string memory _enPKey) {
    require(isAuthF(msg.sender, _enPKey));
    _;
}

//подія виставлення оцінки
event grading(
    uint256 course_id,
    uint256 task_id,
    uint256 grade,
    uint256 time,
    address student,
    address teacher
);

// дію студента
// action_id:
// 0 - отримання матеріалів, завдань;
// 1 - завантаження (здача) виконаної роботи;
event studentAction(
    uint256 course_id,
    uint256 task_id,
    uint256 action_id,
    uint256 time,
    address student
);

// дію препода
// action_id:
// 0 - створення курсу;
// 1 - створення нового завдання;
// 2 - додавання матеріалів;
// 3 - додавання іншого викладача до курсу
// 4 - додавання студентів до курсу
// 5 - додавання конкретного студента до курсу
event teacherAction(
    uint256 course_id,
    uint256 task_id,
    uint256 action_id,
    uint256 time,
    address teacher
);

// функція - аутентифікація (авторизований чи користувач)
function isAuthF(address _address, string memory _enPKey)
    public
    view
    returns (bool result)
{
    if (adminAddress == _address) {
        if (sha256(abi.encodePacked(_enPKey)) == adminEnPKey) {
            result = true;
        }
    }
    if (!result) {
        for (uint256 i = 0; i < studentsCount; i++) {
            if (studentsAddr[i] == _address) {
                if (
                    sha256(abi.encodePacked(_enPKey)) ==
                    students[_address].encryptedPrivateKey
                ) {

```

```

        result = true;
    }
}
}
if (!result) {
    for (uint256 i = 0; i < teachersCount; i++) {
        if (teachersAddr[i] == _address) {
            if (
                sha256(abi.encodePacked(_enPKey)) ==
                teachers[_address].encryptedPrivateKey
            ) {
                result = true;
            }
        }
    }
}

// функція - чи є викладачем
function isTeacherF(address _address) public view returns (bool result) {
    for (uint256 i = 0; i < teachersCount; i++) {
        if (teachersAddr[i] == _address) {
            result = true;
            break;
        }
    }
}

// функція - чи є студентом
function isStudentF(address _address) public view returns (bool result) {
    for (uint256 i = 0; i < studentsCount; i++) {
        if (studentsAddr[i] == _address) {
            result = true;
            break;
        }
    }
}

// функція - чи є адміном
function isAdminF(address _address) public view returns (bool result) {
    result = _address == adminAddress;
}

// функція - чи є учасником певного курсу (чи є доступ до курсу)
function isAccessToCourseF(address _address, uint256 course_id)
    public
    view
    returns (bool result)
{
    for (uint256 i = 0; i < courseStudentsCount; i++) {
        if (
            courseStudents[i].user == _address &&
            courseStudents[i].course_id == course_id
        ) {
            result = true;
            break;
        }
    }
    if (!result) {
        for (uint256 i = 0; i < courseTeachersCount; i++) {
            if (
                courseTeachers[i].user == _address &&

```

```

        courseTeachers[i].course_id == course_id
    ) {
        result = true;
        break;
    }
}

}

}

// функція виставлення оцінки, дана функція доступна тільки викладачам
function setGrade(
    uint256 course_id,
    uint256 task_id,
    uint256 grade,
    address student,
    string memory enPKey //хеш приватного ключу
)
    public
    isTeacher(msg.sender) //модифікатор - чи є викладачем
    isAccessToCourse(msg.sender, course_id) //модифікатор - чи є учасником п
евного курсу (чи є доступ до курсу)
    isAuth(enPKey) //модифікатор - аутентифікація (авторизований чи користув
ач)
{
    //запис події виставлення оцінки
    emit grading(
        course_id,
        task_id,
        grade,
        block.timestamp,
        student,
        msg.sender
    );
}

// функція дії студента, дана функція доступна тільки студентам
function setStudentAction(
    uint256 course_id,
    uint256 task_id,
    uint256 action_id
) private isStudent(msg.sender) {
    emit studentAction(
        course_id,
        task_id,
        action_id,
        block.timestamp,
        msg.sender
    );
}

// функція дії викладача, дана функція доступна тільки викладачам
function setTeacherAction(
    uint256 course_id,
    uint256 task_id,
    uint256 action_id
) private isTeacher(msg.sender) {
    emit teacherAction(
        course_id,
        task_id,
        action_id,
        block.timestamp,
        msg.sender
    );
}

```

```

}

// отримання кількості завдань певного курсу
function getTasksCourseCount(uint256 course_id)
    public
    view
    returns (uint256 result)
{
    result = 0;
    for (uint256 i = 0; i < tasksCount; i++) {
        if (tasks[i].course_id == course_id) result++;
    }
}

//додавання курсу викладачем
function addCourse(
    string memory _name,
    string memory enPKey,
    string memory _coursePublicKey,
    string memory _courseEnPKey
) public isTeacher(msg.sender) isAuth(enPKey) {
    courses[coursesCount] = Course({
        id: coursesCount,
        name: _name,
        timeOfCreation: block.timestamp,
        owner: msg.sender,
        coursePublicKey: _coursePublicKey
    });

    courseTeachers[courseTeachersCount] = Participant({
        course_id: coursesCount,
        user: msg.sender,
        courseEnPrivKey: _courseEnPKey
    });
    courseTeachersCount++;
    setTeacherAction(coursesCount, 0, 0);
    coursesCount++;
}

//додавання нового завдання до курсу викладачем
function addTask(
    string memory _name,
    uint256 course_id,
    uint256 _maxGrade,
    string memory enPKey
)
    public
    isTeacher(msg.sender)
    isAccessToCourse(msg.sender, course_id)
    isAuth(enPKey)
{
    tasks[tasksCount] = Task(
        tasksCount,
        course_id,
        _name,
        block.timestamp,
        _maxGrade,
        msg.sender
    );
    setTeacherAction(course_id, tasksCount, 1);
    tasksCount++;
}

```

```

// додавання матеріалів викладачем (!)
function addMaterials(uint256 course_id, string memory enPKey)
    public
    isTeacher(msg.sender)
    isAuth(enPKey)
{
    setTeacherAction(course_id, 0, 2);
}

// додавання препода до курсу по address
function addTeacher(
    address teacher,
    uint256 course_id,
    string memory enPKey,
    string memory _courseEnPKey
)
    public
    isTeacher(msg.sender)
    isTeacher(teacher)
    isAccessToCourse(msg.sender, course_id)
    isAuth(enPKey)
{
    courseTeachers[courseTeachersCount] = Participant({
        course_id: course_id,
        user: teacher,
        courseEnPrivKey: _courseEnPKey
    });
    courseTeachersCount++;

    setTeacherAction(course_id, 0, 3);
}

// отримання зашифрованого приватного ключа для курсу
function getEnPrivateKeyCourse(address _address, uint256 course_id)
    public
    view
    returns (string memory enPrivateKeyCourse)
{
    for (uint256 i = 0; i < courseTeachersCount; i++) {
        if (
            courseTeachers[i].user == _address &&
            courseTeachers[i].course_id == course_id
        ) {
            enPrivateKeyCourse = courseTeachers[i].courseEnPrivKey;
            break;
        }
    }
}

//отримання препода по e-mail
function getTeacherFromEmail(string memory email)
    public
    view
    returns (Teacher memory teacher)
{
    for (uint256 i = 0; i < teachersCount; i++) {
        if (
            keccak256(bytes(teachers[teachersAddr[i]].email)) ==
            keccak256(bytes(email))
        ) {
            teacher = teachers[teachersAddr[i]];
            break;
        }
    }
}

```

```

    }
}

//додавання препода до курсу по e-mail
function addTeacherEmail(
    string memory email,
    uint256 course_id,
    string memory enPKey,
    string memory _courseEnPrivKey
)
    public
    isTeacher(msg.sender)
    isAccessToCourse(msg.sender, course_id)
    isAuth(enPKey)
{
    address teacherAddress;
    bool ok = false;
    for (uint256 i = 0; i < teachersCount; i++) {
        if (
            keccak256(bytes(teachers[teachersAddr[i]].email)) ==
            keccak256(bytes(email))
        ) {
            teacherAddress = teachersAddr[i];
            ok = true;
            break;
        }
    }
    if (ok && isTeacherF(teacherAddress)) {
        courseTeachers[courseTeachersCount] = Participant({
            course_id: course_id,
            user: teacherAddress,
            courseEnPrivKey: _courseEnPrivKey
        });
        courseTeachersCount++;

        setTeacherAction(course_id, 0, 3);
    }
}

//додавання студентів в курс по групі
function addStudents(
    uint256 course_id,
    string memory _group,
    string memory enPKey
)
    public
    isTeacher(msg.sender)
    isAccessToCourse(msg.sender, course_id)
    isAuth(enPKey)
{
    for (uint256 i = 0; i < studentsCount; i++) {
        bool result = false;
        for (uint256 j = 0; j < courseStudentsCount; j++) {
            if (
                courseStudents[j].course_id == course_id &&
                courseStudents[j].user == studentsAddr[i]
            ) {
                result = true;
                break;
            }
        }
        if (
            keccak256(bytes(students[studentsAddr[i]].group)) ==

```

```

        keccak256(bytes(_group)) &&
        !result
    ) {
        courseStudents[courseStudentsCount] = Participant({
            course_id: course_id,
            user: studentsAddr[i],
            courseEnPrivKey: ""
        });
        courseStudentsCount++;
    }
}
setTeacherAction(course_id, 0, 4);
}

//додавання студента в курс по address
function addStudent(
    address studentAddr,
    uint256 course_id,
    string memory enPKey
)
    public
    isTeacher(msg.sender)
    isAccessToCourse(msg.sender, course_id)
    isAccessToCourse(studentAddr, course_id)
    isAuth(enPKey)
{
    courseStudents[courseStudentsCount] = Participant({
        course_id: course_id,
        user: studentAddr,
        courseEnPrivKey: ""
    });
    courseStudentsCount++;
    setTeacherAction(course_id, 0, 5);
}

//додавання студента в курс по e-mail
function addStudentEmail(
    string memory email,
    uint256 course_id,
    string memory enPKey
)
    public
    isTeacher(msg.sender)
    isAccessToCourse(msg.sender, course_id)
    isAuth(enPKey)
{
    address studentAddress;
    bool ok = false;
    for (uint256 i = 0; i < studentsCount; i++) {
        if (
            keccak256(bytes(students[studentsAddr[i]].email)) ==
            keccak256(bytes(email))
        ) {
            studentAddress = studentsAddr[i];
            ok = true;
            break;
        }
    }
    if (ok && isAccessToCourseF(studentAddress, course_id)) {
        courseStudents[courseStudentsCount] = Participant({
            course_id: course_id,
            user: studentAddress,
            courseEnPrivKey: ""
        });
    }
}

```

```

    });
    courseStudentsCount++;

    setTeacherAction(course_id, 0, 3);
}
}

//отримання матеріалів курсу
function getMaterialTask(uint256 _course_id, string memory enPKey)
    public
    isStudent(msg.sender)
    isAccessToCourse(msg.sender, _course_id)
    isAuth(enPKey)
{
    setStudentAction(_course_id, 0, 0);
}

//здати роботу
function handOverWork(
    uint256 _course_id,
    uint256 _task_id,
    string memory enPKey
)
    public
    isStudent(msg.sender)
    isAccessToCourse(msg.sender, _course_id)
    isAuth(enPKey)
{
    setStudentAction(_course_id, _task_id, 1);
}

//Відправка своєї роботи студентом
function sendWork(
    uint256 _course_id,
    uint256 _task_id,
    string memory hashFiles,
    string memory nameFiles,
    string memory typeFiles,
    string memory enPKey
)
    public
    isStudent(msg.sender)
    isAccessToCourse(msg.sender, _course_id)
    isAuth(enPKey)
{
    studentWorks[studentWorksCount] = StudentWork({
        id: studentWorksCount,
        course_id: _course_id,
        task_id: _task_id,
        student: msg.sender,
        fileName: nameFiles,
        fileType: typeFiles,
        fileIPFS: hashFiles
    });
    studentWorksCount++;
}
}

```

ДОДАТОК Б
Текст програми

У лістингу Б.1 наведений текст програми.

Лістинг Б.1 – Текст програми

```
import Education from '../abis/Education.json' //розгорнутий смарт-контракт
import TempAccounts from './temp_accounts.json' //акаунти для швидкого входу в
систему (режим тестування)
import React, { Component } from 'react';
import $ from 'jquery';
import Web3 from 'web3';
import NodeRSA from 'node-rsa';

//Імпорт IPFS бібліотеки
const { create } = require('ipfs-http-client')
const ipfs = create({ host: 'ipfs.infura.io', port: 5001, protocol: 'http' })

//Головний клас застосунку
class App extends Component {

  thisApp1 = this;
  constructor(props) {
    super(props)
  }

  //ініціалізація
  async componentWillMount() {
    await this.loadBlockchainData(this.props.dispatch)
  }

  //завантаження даних блокчейна
  async loadBlockchainData(dispatch) {

    App.thisApp1 = this;
    App.thisApp1.web3 = null;
    App.thisApp1.web3Provider = null;
    App.thisApp1.account = '0x0';
    App.thisApp1.accountTempPrivateKey = '';
    App.thisApp1.contracts = {};
    App.thisApp1.accounts = null;
    App.thisApp1.currentRole = '';
    App.thisApp1.tempAccounts = [];

    App.thisApp1.educationContract = null;

    App.thisApp1.allCourses = true;
    App.thisApp1.currentPage = "courses";
    App.thisApp1.currentCourseId = 0;
    App.thisApp1.currentTaskId = 0;
    App.thisApp1.currentStudent = '0x0';
    App.thisApp1.currentNavigationCourse = 0;

    App.thisApp1.networkId = null;
    App.thisApp1.networkData = null;

    App.thisApp1.nameStudentEvents = ["receiving materials", "submitting work"];
    App.thisApp1.nameTeacherEvents = ["create a course", "create a new task", "a
dd materials", "add another teacher to a course", "add students to a course", "a
dd a specific student to a course"];

    //якщо заздалегідь ініціалізовано
    if (App.thisApp1.web3 !== null) {
```

```

        //вказуємо на поточний провайдер, який був ініціалізований заздалегідь
        App.thisApp1.web3Provider = App.thisApp1.web3.currentProvider;
        App.thisApp1.web3 = new Web3(App.thisApp1.web3.currentProvider);
    } else {
        //ініціалізація в перший раз
        App.thisApp1.web3Provider = new Web3.providers.HttpProvider('http://localhost:7545');
        App.thisApp1.web3 = new Web3(App.thisApp1.web3Provider);
    }
    //Всі учасники мережі блокчейна
    App.thisApp1.accounts = await App.thisApp1.web3.eth.getAccounts();
    //Ідентифікатор блокчейна
    App.thisApp1.networkId = await App.thisApp1.web3.eth.net.getId();
    //Дані мережі блокчейна
    App.thisApp1.networkData = Education.networks[App.thisApp1.networkId];
    if (App.thisApp1.networkData) {
        //Отримання посилання на розгорнутий смарт-контракт, написаний на Solidity
        App.thisApp1.educationContract = new App.thisApp1.web3.eth.Contract(Education.abi, App.thisApp1.networkData.address)
    } else {
        window.alert('Smart contract not deployed to detected network.')
    }
    $.each(TempAccounts, function (index, value) {
        App.thisApp1.tempAccounts.push(value);
    });
    console.log(App.thisApp1.tempAccounts);
    //швидкий вхід (вхід як 1-ий викладач або адміністратор)
    if (App.thisApp1.tempAccounts.length > 1) {
        App.thisApp1.account = App.thisApp1.accounts[1];
        App.thisApp1.accountTempPrivateKey = App.thisApp1.tempAccounts[1].privateKey;
    }
    else {
        App.thisApp1.account = App.thisApp1.accounts[0];
        App.thisApp1.accountTempPrivateKey = "2e376a83ea7b315cfd5c6423441d438f4042ec4613a2d7acb6c6e07f6cab8606";
    }
    $('#fileLoadBigFile').hide();
    $('#addCourseForm').submit(function (e) {
        App.thisApp1.addCourse();
        e.preventDefault();
    });
    $('#addTaskForm').submit(function (e) {
        App.thisApp1.addTask();
        e.preventDefault();
    });
    $('#addTeacherToCourseForm').submit(function (e) {
        App.thisApp1.addTeacherToCourse();
        e.preventDefault();
    });
    $('#addStudentsToCourseForm').submit(function (e) {
        App.thisApp1.addStudentsToCourse();
        e.preventDefault();
    });
    $('#loadFileForm').submit(function (e) {
        App.thisApp1.loadFile();
        e.preventDefault();
    });
    var accountExitBtn = $('#accountExitBtn');
    accountExitBtn.click(function () {
        toExit();
    });

```

```

        if (App.thisApp1.account == '0x0' || App.thisApp1.account == '' || App.thisApp1.account == null) {
            App.thisApp1.currentRole = '';
            App.thisApp1.renderApp();
        }
        else {
            App.thisApp1.educationContract.methods.isTeacherF(App.thisApp1.account).call({ from: App.thisApp1.account }, function (error, result) {
                if (result) {
                    App.thisApp1.currentRole = 'Teacher';
                    App.thisApp1.renderApp();
                }
                else {
                    App.thisApp1.educationContract.methods.isStudentF(App.thisApp1.account).call({ from: App.thisApp1.account }, function (error, result) {
                        if (result) {
                            App.thisApp1.currentRole = 'Student';
                            App.thisApp1.renderApp();
                        }
                        else {
                            App.thisApp1.educationContract.methods.isAdminF(App.thisApp1.account).call({ from: App.thisApp1.account }, function (error, result) {
                                if (result) {
                                    App.thisApp1.currentRole = 'Admin';
                                    App.thisApp1.renderApp();
                                }
                            });
                        }
                    });
                }
            });
        }
    });
}

//функція динамічного відображення інтерфейсу застосунку
async renderApp() {
    console.log("tempPrivKey = " + App.thisApp1.accountTempPrivateKey);
    var accountExitBtn = $('#accountExitBtn');
    var contentCourses = $('#contentCourses');
    ...
    var contentLogin = $('#contentLogin');
    var contentRegister = $('#contentRegister');

    contentCourses.hide();
    contentNewCourse.hide();
    ...
    navigatorRegister.hide();
    navigatorUsers.hide();
    accountExitBtn.hide();
    //Якщо користувач не зареєстрований
    if (App.thisApp1.currentRole == '') {
        //loader.hide();
        contentLogin.show();
        $('#currentAccountName').html("NoName");
        $('#currentAccountRole').html("Unauthorized user");
        $('#currentAccountAddress').html(App.thisApp1.account);
        $('#titlePage').text("Login");

        $('#loginAdminAddress').text(App.thisApp1.tempAccounts[0].address);
        $('#loginAdminPKey').text(App.thisApp1.tempAccounts[0].privateKey);
        $('#loginAdminBtn').click(function () {
            toLogin(App.thisApp1.tempAccounts[0].address, App.thisApp1.tempAccounts[0].privateKey);
        });
    }
}

```

```

var listLoginTeacherUser = $('#listLoginTeacherUser');
listLoginTeacherUser.empty();
var listLoginStudentUser = $('#listLoginStudentUser');
listLoginStudentUser.empty();
for (var i = 0; i < App.thisApp1.tempAccounts.length; i++) {
    //получение данных о преподавателей
    App.thisApp1.educationContract.methods.teachers(App.thisApp1.tempAccounts[i].address).call({ from: App.thisApp1.tempAccounts[0].address }, function (error, teacher) {
        if (!error) {
            if (teacher[2] != null && teacher[2] != '') {
                var teacherUser = $('#loginTeacherUser');
                teacherUser.find('.loginTeacherBtn').attr("id", 'loginTeacherBtn' + parseInt(teacher[0]));
                teacherUser.find('.loginTeacherName').text(teacher[2]);
                teacherUser.find('.loginTeacherEmail').text(teacher[3]);
                teacherUser.find('.loginTeacherAddress').text(App.thisApp1.tempAccounts[(parseInt(teacher[0]) + 1)].address);
                teacherUser.find('.loginTeacherPKey').text(App.thisApp1.tempAccounts[(parseInt(teacher[0]) + 1)].privateKey);
                console.log("loaded teacher address = " + App.thisApp1.tempAccounts[(parseInt(teacher[0]) + 1)].address);
                listLoginTeacherUser.append(teacherUser.html());
                $('#loginTeacherBtn' + parseInt(teacher[0])).click(function () {
                    toLogin(App.thisApp1.tempAccounts[(parseInt(teacher[0]) + 1)].address, App.thisApp1.tempAccounts[(parseInt(teacher[0]) + 1)].privateKey);
                });
            }
        }
        else {
            console.log("error");
        }
    });
    //получение данных о студентах
    App.thisApp1.educationContract.methods.students(App.thisApp1.tempAccounts[i].address).call({ from: App.thisApp1.tempAccounts[0].address }, function (error, student) {
        if (!error) {
            if (student[2] != null && student[2] != '') {
                var studentUser = $('#loginStudentUser');
                studentUser.find('.loginStudentBtn').attr("id", 'loginStudentBtn' + parseInt(student[0]));
                studentUser.find('.loginStudentName').text(student[2]);
                studentUser.find('.loginStudentEmail').text(student[3]);
                studentUser.find('.loginStudentGroup').text(" (" + student[4] + ")");
                console.log("loaded student address = " + App.thisApp1.tempAccounts[(parseInt(student[0]) + 4)].address);
                studentUser.find('.loginStudentAddress').text(App.thisApp1.tempAccounts[(parseInt(student[0]) + 4)].address);
                studentUser.find('.loginStudentPKey').text(App.thisApp1.tempAccounts[(parseInt(student[0]) + 4)].privateKey);
                listLoginStudentUser.append(studentUser.html());
                $('#loginStudentBtn' + parseInt(student[0])).click(function () {
                    toLogin(App.thisApp1.tempAccounts[(parseInt(student[0]) + 4)].address, App.thisApp1.tempAccounts[(parseInt(student[0]) + 4)].privateKey);
                });
            }
        }
        else {
            console.log("error");
        }
    });
}

```

```

    }
}
else {
    navigator.show();
    accountExitBtn.show();
    navigatorUsers.show();
    $("#selectedNavigatorRegister").hide();
    ...
    $("#selectedNavigatorUsers").hide();
    if (App.thisApp1.currentPage == "users") {
        $('#titlePage').text("Users");
        contentUsers.show();
        $("#selectedNavigatorUsers").show();
        $('#adminAddress').text(App.thisApp1.tempAccounts[0].address);
        $('#adminPKey').text(App.thisApp1.tempAccounts[0].privateKey);
        var listTeacherUsers = $('#listTeacherUsers');
        listTeacherUsers.empty();
        var listStudentUsers = $('#listStudentUsers');
        listStudentUsers.empty();
        for (var i = 0; i < App.thisApp1.tempAccounts.length; i++) {
            App.thisApp1.educationContract.methods.teachers(App.thisApp1.tempAccounts[i].address).call({ from: App.thisApp1.account }, function (error, teacher) {
                if (!error) {
                    if (teacher[2] != null && teacher[2] != '') {
                        var teacherUser = $('#teacherUser');
                        teacherUser.find('.teacherName').text(teacher[2]);
                        teacherUser.find('.teacherEmail').text(teacher[3]);
                        teacherUser.find('.teacherAddress').text(App.thisApp1.tempAccounts[(parseInt(teacher[0]) + 1)].address);
                        listTeacherUsers.append(teacherUser.html());
                    }
                }
                else {
                    console.log("error");
                }
            });
            App.thisApp1.educationContract.methods.students(App.thisApp1.tempAccounts[i].address).call({ from: App.thisApp1.account }, function (error, student) {
                if (!error) {
                    if (student[2] != null && student[2] != '') {
                        var studentUser = $('#studentUser');
                        studentUser.find('.studentName').text(student[2]);
                        studentUser.find('.studentEmail').text(student[3]);
                        studentUser.find('.studentGroup').text("(" + student[4] + ")");
                        studentUser.find('.studentAddress').text(App.thisApp1.tempAccounts[(parseInt(student[0]) + 4)].address);
                        listStudentUsers.append(studentUser.html());
                    }
                }
                else {
                    console.log("error");
                }
            });
        }
    }
}
//Якщо поточний користувач - адміністратор
if (App.thisApp1.currentRole == "Admin") {
    navigatorRegister.show();
    $("#currentAccountRole").html("Admin");
    $("#currentAccountName").html("-");
    $("#currentAccountAddress").html(App.thisApp1.account);
    if (App.thisApp1.currentPage == "register") {
        $('#titlePage').text("Registration");
    }
}

```

```

        $("#selectedNavigatorRegister").show();
        contentRegister.show();
    }
    else if (App.thisApp1.currentPage == "users") {
        $("#selectedNavigatorUsers").show();
    }
}
else {
    $("#currentAccountAddress").html(App.thisApp1.account);
    //Якщо поточний користувач - викладач
    if (App.thisApp1.currentRole == "Teacher") {
        navigatorCourses.show();
        navigatorEvents.show();
        $("#currentAccountRole").html("Teacher");
        App.thisApp1.educationContract.methods.teachers(App.thisApp1.account).
        call({ from: App.thisApp1.account }, function (error, teacher) {
            $("#currentAccountName").html(teacher[2]);
        });
        //Якщо поточний користувач - студент
    } else if (App.thisApp1.currentRole == "Student") {
        navigatorCourses.show();
        navigatorEvents.show();
        navigatorGrades.show();
        $("#currentAccountRole").html("Student");
        App.thisApp1.educationContract.methods.students(App.thisApp1.account).
        call({ from: App.thisApp1.account }, function (error, student) {
            $("#currentAccountName").html(student[2]);
        });
    }
    if (App.thisApp1.currentPage == "courses") {
        $("#selectedNavigatorCourses").show();
    }
    else if (App.thisApp1.currentPage == "events") {
        $("#selectedNavigatorEvents").show();
    }
    else if (App.thisApp1.currentPage == "grades") {
        $("#selectedNavigatorGrades").show();
    }
    //Якщо поточна сторінка - список курсів
    if (App.thisApp1.currentPage == "courses") {
        $('#titlePage').text("Courses");
        //отримання кількості курсів в мережі
        App.thisApp1.educationContract.methods.coursesCount.call().call({ from
: App.thisApp1.account }, function (error, coursesCount) {
            console.log("coursesCount = " + coursesCount);
            for (var i = 0; i < coursesCount; i++) {
                //отримання об'єкта курс по id
                App.thisApp1.educationContract.methods.courses(i).call({ from: App
.thisApp1.account }, function (error, course) {
                    var listCoursesRow = $('#listCoursesRow');
                    listCoursesRow.empty();
                    //отримання об'єкта викладач за адресою (course[3] - адреса влас
ника курсу)
                    App.thisApp1.educationContract.methods.teachers(course[3]).call(
{ from: App.thisApp1.account }, function (error, teacher) {

                        App.thisApp1.educationContract.methods.tasksCount.call().call(
{ from: App.thisApp1.account }, function (error, tasksCount) {

                            var dateAddedCourse = getDateDateFormat(course[2]);

```

```

        App.thisApp1.educationContract.methods.getTasksCourseCount(course[0]).call({ from: App.thisApp1.account }, function (error, tasksCourseCount) {
            {
                ...
            });
        });
    });
}
contentCourses.show();
})
}
else if (App.thisApp1.currentPage == "createCourse") {
    $('#titlePage').text("New course");
    contentNewCourse.show();
}
else if (App.thisApp1.currentPage == "createTask") {
    App.thisApp1.educationContract.methods.courses(App.thisApp1.currentCourseId).call({ from: App.thisApp1.account }, function (error, course) {
        $('#titlePage').text(course[1]);
    });
    var listTeachersCourse = $('#listTeachersCourse2');
    var listStudentsCourse = $('#listStudentsCourse2');
    listTeachersCourse.empty();
    listStudentsCourse.empty();
    App.thisApp1.educationContract.methods.courseTeachersCount.call().call({ from: App.thisApp1.account }, function (error, courseTeachersCount) {
        for (var i = 0; i < courseTeachersCount; i++) {
            App.thisApp1.educationContract.methods.courseTeachers(i).call({ from: App.thisApp1.account }, function (error, participant) {
                console.log("courseID = " + App.thisApp1.currentCourseId);
                if (participant[0] == App.thisApp1.currentCourseId) {
                    var teacherCourseTemplate = $('#teacherCourseTemplate2');
                    App.thisApp1.educationContract.methods.teachers(participant[1]).call({ from: App.thisApp1.account }, function (error, teacher) {
                        teacherCourseTemplate.find('#nameTeacherCourse').text(teacher[2]);

                        listTeachersCourse.append(teacherCourseTemplate.html());
                    });
                }
            });
        }
    });
    App.thisApp1.educationContract.methods.courseStudentsCount.call().call({ from: App.thisApp1.account }, function (error, courseStudentsCount) {
        for (var i = 0; i < courseStudentsCount; i++) {
            App.thisApp1.educationContract.methods.courseStudents(i).call({ from: App.thisApp1.account }, function (error, participant) {
                if (participant[0] == App.thisApp1.currentCourseId) {
                    var studentCourseTemplate = $('#studentCourseTemplate2');
                    App.thisApp1.educationContract.methods.students(participant[1]).call({ from: App.thisApp1.account }, function (error, student) {
                        studentCourseTemplate.find('#nameStudentCourse').text(student[2]);

                        listStudentsCourse.append(studentCourseTemplate.html());
                    });
                }
            });
        }
    });
    contentNewTask.show();
}
else if (App.thisApp1.currentPage == "addUsersToCourse") {

```

```

        App.thisApp1.educationContract.methods.courses(App.thisApp1.currentCourseId).call({ from: App.thisApp1.account }, function (error, course) {
            $('#titlePage').text(course[1]);
        });
        var listTeachersCourse = $('#listTeachersCourse3');
        var listStudentsCourse = $('#listStudentsCourse3');
        listTeachersCourse.empty();
        listStudentsCourse.empty();
        App.thisApp1.educationContract.methods.courseTeachersCount.call().call(
            ({ from: App.thisApp1.account }, function (error, courseTeachersCount) {
                for (var i = 0; i < courseTeachersCount; i++) {
                    App.thisApp1.educationContract.methods.courseTeachers(i).call({ from: App.thisApp1.account }, function (error, participant) {
                        if (participant[0] == App.thisApp1.currentCourseId) {
                            var teacherCourseTemplate = $('#teacherCourseTemplate3');
                            App.thisApp1.educationContract.methods.teachers(participant[1]).call({ from: App.thisApp1.account }, function (error, teacher) {
                                teacherCourseTemplate.find('#nameTeacherCourse').text(teacher[2]);

                                listTeachersCourse.append(teacherCourseTemplate.html());
                            });
                        }
                    });
                }
            });
        App.thisApp1.educationContract.methods.courseStudentsCount.call().call(
            ({ from: App.thisApp1.account }, function (error, courseStudentsCount) {
                for (var i = 0; i < courseStudentsCount; i++) {
                    App.thisApp1.educationContract.methods.courseStudents(i).call({ from: App.thisApp1.account }, function (error, participant) {
                        if (participant[0] == App.thisApp1.currentCourseId) {
                            var studentCourseTemplate = $('#studentCourseTemplate3');
                            App.thisApp1.educationContract.methods.students(participant[1]).call({ from: App.thisApp1.account }, function (error, student) {
                                studentCourseTemplate.find('#nameStudentCourse').text(student[2]);

                                listStudentsCourse.append(studentCourseTemplate.html());
                            });
                        }
                    });
                }
            });
        contentAddUsersToCourse.show();
    }
    else if (App.thisApp1.currentPage == "course") {
        App.thisApp1.educationContract.methods.courses(App.thisApp1.currentCourseId).call({ from: App.thisApp1.account }, function (error, course) {
            $('#titlePage').text(course[1]);
        });
        if (App.thisApp1.currentNavigationCourse == 0) {
            var listTasks = $('#listTasks');
            listTasks.empty();
            App.thisApp1.educationContract.methods.tasksCount.call().call({ from: App.thisApp1.account }, function (error, tasksCount) {
                for (var i = 0; i < tasksCount; i++) {
                    App.thisApp1.educationContract.methods.tasks(i).call({ from: App.thisApp1.account }, function (error, task) {
                        console.log("task[2] = " + task[2]);
                        if (task[1] == App.thisApp1.currentCourseId) {
                            App.thisApp1.educationContract.methods.teachers(task[5]).call({ from: App.thisApp1.account }, function (error, teacher) {
                                App.thisApp1.educationContract.methods.isStudentF(App.thisApp1.account).call({ from: App.thisApp1.account }, function (error, result) {

```



```

        var listTeachersCourse = $('#listTeachersCourse');
        var listStudentsCourse = $('#listStudentsCourse');
        listTeachersCourse.empty();
        listStudentsCourse.empty();
        App.thisApp1.educationContract.methods.courseTeachersCount.call().call({
            from: App.thisApp1.account }, function (error, courseTeachersCount) {
                for (var i = 0; i < courseTeachersCount; i++) {
                    App.thisApp1.educationContract.methods.courseTeachers(i).call({
                        from: App.thisApp1.account }, function (error, participant) {
                            if (participant[0] == App.thisApp1.currentCourseId) {
                                var teacherCourseTemplate = $('#teacherCourseTemplate');
                                App.thisApp1.educationContract.methods.teachers(participant[
                                    1]).call({ from: App.thisApp1.account }, function (error, teacher) {
                                        teacherCourseTemplate.find('#nameTeacherCourse').text(teacher[2]);
                                        listTeachersCourse.append(teacherCourseTemplate.html());
                                    });
                                }
                            });
                        }
                    });
                App.thisApp1.educationContract.methods.courseStudentsCount.call().call({
                    from: App.thisApp1.account }, function (error, courseStudentsCount) {
                        for (var i = 0; i < courseStudentsCount; i++) {
                            App.thisApp1.educationContract.methods.courseStudents(i).call({
                                from: App.thisApp1.account }, function (error, participant) {
                                    if (participant[0] == App.thisApp1.currentCourseId) {
                                        var studentCourseTemplate = $('#studentCourseTemplate');
                                        App.thisApp1.educationContract.methods.students(participant[
                                            1]).call({ from: App.thisApp1.account }, function (error, student) {
                                                studentCourseTemplate.find('#nameStudentCourse').text(student[2]);
                                                console.log("FFFFF = " + student[2]);
                                                listStudentsCourse.append(studentCourseTemplate.html());
                                            });
                                        }
                                    });
                                }
                            });
                        contentCourse.show();
                    }
                }
            }
        }
        else {
            contentCourseGrades.show();
        }
    }
    else if (App.thisApp1.currentPage == "task") {
        App.thisApp1.educationContract.methods.tasks(App.thisApp1.currentTaskId).call({
            from: App.thisApp1.account }, function (error, task) {
                $('#nameTask').text(task[2]);
                App.thisApp1.educationContract.methods.teachers(task[5]).call({
                    from: App.thisApp1.account }, function (error, teacher) {
                        $('#ownerTask').text(teacher[2]);
                    });
                $('#dateTask').text(getDateTimeFormat(task[3]));
                if (App.thisApp1.currentRole == "Student") {
                    $('#listStudentUserTask').hide();
                    var listStudentTasks = $('#listStudentTasks');
                    listStudentTasks.empty();
                    $('#formStudentTasksForStudent').show();
                    $('#formStudentTasksForTeacher').hide();
                    App.thisApp1.educationContract.methods.studentWorksCount.call().call({
                        from: App.thisApp1.account }, function (error, studentWorksCount) {
                            for (var i = 0; i < studentWorksCount; i++) {

```

```

        App.thisApp1.educationContract.methods.studentWorks(i).call({
from: App.thisApp1.account }, function (error, work) {
    if (parseInt(work[1]) == App.thisApp1.currentCourseId &&
        parseInt(work[2]) == App.thisApp1.currentTaskId &&
        work[3] == App.thisApp1.account
    ) {
        var studentTaskTemplate = $('#studentTaskTemplate');
        studentTaskTemplate.find('.studentTaskName').text(work[4])
    }
    listStudentTasks.append(studentTaskTemplate.html());
});
});

App.thisApp1.educationContract.getPastEvents(
    "studentAction",
    {
        fromBlock: 0,
        toBlock: "latest"
    }, (errors, events) => {
        if (!errors) {
            var finded = false;
            for (var i = 0; i < events.length; i++) {
                var resultsEvent = events[i].returnValues;
                if (resultsEvent.student == Number(App.thisApp1.account) &
&
                    resultsEvent.course_id == App.thisApp1.currentCourseId &
&
                    resultsEvent.task_id == App.thisApp1.currentTaskId &&
                    resultsEvent.action_id == 1
                ) {
                    finded = true;
                    $('#statusTask').text("Handed over");
                    ...
                    $('#bigFileError').hide();
                }
            }
            if (!finded) {
                $('#statusTask').hide();
                ...
                $('#bigFileError').show();
            }
        }
    });

App.thisApp1.educationContract.getPastEvents(
    "grading",
    {
        fromBlock: 0,
        toBlock: "latest"
    }, (errors, events) => {
        if (!errors) {
            var finded = false;
            for (var i = 0; i < events.length; i++) {
                var resultsEvent = events[i].returnValues;
                if (resultsEvent.student == Number(App.thisApp1.account) &
&
                    resultsEvent.course_id == App.thisApp1.currentCourseId &
&
                    resultsEvent.task_id == App.thisApp1.currentTaskId

```

```

    ) {
        finded = true;
        $('#statusTask').text("Graded");
        $('#currentGradeTask').text(resultsEvent.grade);
        $('#currentGradeTask').show();
        $('#maxGradeTask').show();
        $('#maxGradeTask').text("/") + task[4]);
    }
}
if (!finded) {
    $('#statusTask').text("Handed over");
    $('#currentGradeTask').hide();
    $('#maxGradeTask').text(task[4]);
}
}
);
}
else {
    $('#formStudentTasksForStudent').hide();
    ...
    $('#nameStudentTask').text("-");

    if (App.thisApp1.currentStudent == '0x0') {
    }
    else {
        $('#gradeBtn').attr("class", "myBtn2Gray");
        App.thisApp1.educationContract.methods.students(App.thisApp1.currentStudent).call({ from: App.thisApp1.account }, function (error, student) {
            $('#nameStudentTask').text(student[2]);
        });
        var listStudentTasks = $('#listStudentTasks0');
        listStudentTasks.empty();
        App.thisApp1.educationContract.getPastEvents(
            "grading",
            {
                fromBlock: 0,
                toBlock: "latest"
            }, (errors, events) => {
                if (!errors) {
                    var finded = false;
                    $('#statusTask').text("No graded");
                    App.thisApp1.educationContract.methods.studentWorksCount.call().call({ from: App.thisApp1.account }, function (error, studentWorksCount) {
                        for (var i = 0; i < studentWorksCount; i++) {
                            App.thisApp1.educationContract.methods.studentWorks(i).call({ from: App.thisApp1.account }, function (error, work) {
                                if (parseInt(work[1]) == App.thisApp1.currentCourseId &&
                                    parseInt(work[2]) == App.thisApp1.currentTaskId &&
                                    work[3] == App.thisApp1.currentStudent
                                ) {
                                    $('#studentNonTaskTemplate0').hide();
                                    $('#gradeBtn').attr("class", "myBtn2");
                                    $('#statusTask').show();
                                    var studentTaskTemplate = $('#studentTaskTemplate0');
                                    studentTaskTemplate.find('.studentTaskName').text(work[4]);
                                    studentTaskTemplate.find('.studentTaskName').attr("id", "studentTaskName" + work[0]);
                                }
                            });
                        }
                    });
                }
            });
        }
    }
}

```

```

listStudentTasks.append(studentTaskTemplate.html()
);
$('#studentTaskName' + work[0]).click(function ()
{
    //URL на повний файл в IPFS
    var url = "https://ipfs.infura.io/ipfs/" + work[
6]; //хеш файлу IPFS

    console.log("crypted data: " + url);
    //запит
    var request = new XMLHttpRequest();
    //тип запиту по отримання по URL
    request.open('GET', url);
    request.responseType = 'text';
    //функція на завантаження по URL
    request.onload = function () {
        //Вміст файлу з URL
        console.log(request.response);
        const key = new NodeRSA({ b: 1024 });
        key.importKey(App.thisApp1.accountTempPrivateK
ey, 'pkcs1-pem');

        App.thisApp1.educationContract.methods.getEnPr
ivateKeyCourse(App.thisApp1.account, App.thisApp1.currentCourseId).call({ from:
App.thisApp1.account }, function (error, enPKeyCourse) {
            const coursePrivateKey = key.decrypt(enPKeyC
ourse);

            const keyCourse = new NodeRSA({ b: 1024 });
            keyCourse.importKey(coursePrivateKey, 'pkcs1
-pem');

            console.log("decrypt data...");
            //розшифровка вміст файлу
            var decrypted = keyCourse.decrypt(request.re
sponse);

            //код для автоматичного скачування файлу
            var blob = new Blob([decrypted], { type: wor
k[5] });

            var link = document.createElement("a");
            link.href = window.URL.createObjectURL(blob);
            link.download = work[4];
            link.click();
        });
    };
    // виклик функції на завантаження файлу по URL.
    Без виклику нічого не станеться.
    request.send();
});
}
});
});
for (var i = 0; i < events.length; i++) {
    var resultsEvent = events[i].returnValues;
    if (resultsEvent.student == Number(App.thisApp1.currentS
tudent) &&
        resultsEvent.course_id == App.thisApp1.currentCourseId
        &&
        resultsEvent.task_id == App.thisApp1.currentTaskId
    ) {
        finded = true;
        $('#statusTask').text("Graded");
        $('#statusTask').show();
        $('#gradeBtn').attr("class", "myBtn2Gray");
        $('#currentGradeTask').text(resultsEvent.grade);
        $('#currentGradeTask').show();
    }
}

```

```

        $('#maxGradeTask').show();
        $('#maxGradeTask').text("/" + task[4]);
    }
}
if (!finded) {
    $('#currentGradeTask').hide();
    $('#maxGradeTask').text(task[4]);
}
}
}
);
}
}
$('#listStudentUserTask').show();
var listStudentUsers = $('#listStudentUserTask');
listStudentUsers.empty();
App.thisApp1.educationContract.methods.courseStudentsCount.call().call({
    from: App.thisApp1.account }, function (error, courseStudentsCount) {
    for (var i = 0; i < courseStudentsCount; i++) {
        App.thisApp1.educationContract.methods.courseStudents(i).call({
            from: App.thisApp1.account }, function (error, participant) {
                if (participant[0] == App.thisApp1.currentCourseId) {
                    var studentUser = $('#studentUserTask');
                    App.thisApp1.educationContract.methods.students(participant[
1])).call({ from: App.thisApp1.account }, function (error, student) {
                        studentUser.find('.studentBtn').attr("id", 'studentBtn' +
parseInt(student[0]));
                        studentUser.find('.studentName').text(student[2]);
                        studentUser.find('.studentEmail').text(student[3]);
                        studentUser.find('.studentGroup').text("(" + student[4] +
"");
                        studentUser.find('.studentAddress').text(App.thisApp1.temp
Accounts[(parseInt(student[0]) + 4)].address);
                        listStudentUsers.append(studentUser.html());
                        $('#studentBtn' + parseInt(student[0])).click(function ()
{
                            App.thisApp1.currentStudent = App.thisApp1.tempAccounts[
(parseInt(student[0]) + 4)].address;
                            App.thisApp1.renderApp();
                        });
                    });
                }
            });
        }
    });
}
});
contentTask.show();
});
}
else if (App.thisApp1.currentPage == "events") {
    $('#titlePage').text("Events");
    //повернення подій дій викладача (створення курсу, завдань, учасники
в до курсу, матеріалів до курсу і т.д)
    App.thisApp1.educationContract.getPastEvents(
        "teacherAction",
        {
            fromBlock: 0, //від першого блоку
            toBlock: "latest" //до останнього блоку отримати події
        }, (errors, events) => {
            if (!errors) { //якщо без помилок
                var listEventsRow = $('#listEventsRow');
                listEventsRow.empty();
                var event_id = 0;
                var as_event_id = 0;
            }
        }
    );
}
}

```

```

var as_event_id1 = 0;
//перехід по подіям
for (var i = 0; i < events.length; i++) {
    var resultsEvent = events[i].returnValues;
    if (resultsEvent.teacher == Number(App.thisApp1.account)) {
        var colorBack = "background-color: rgb(240, 240, 240);";
        if (event_id % 2 == 0)
            colorBack = "background-color: rgb(255, 255, 255);";
        var dateAddedEvent = getDateTimeFormat(JSON.parse(resultsEve
nt.time));

        ...
        App.thisApp1.educationContract.methods.courses(JSON.parse(re
sultsEvent.course_id)).call({ from: App.thisApp1.account }, function (error, cou
rse) {

            $('#eventTeacherCourse' + as_event_id).text(course[1]);
            $('#eventTeacherCourse' + as_event_id).click(function () {
                toCourse(JSON.parse(resultsEvent.course_id));
            });
            $('#eventMyGradesCourseToTask' + as_event_id).click(funci
on () {

                toTask(JSON.parse(resultsEvent.task_id));
            });
            as_event_id++;
        });
        var task_id = 0;
        if (JSON.parse(resultsEvent.action_id) == 1) {
            task_id = JSON.parse(resultsEvent.task_id);
        }
        else {
            task_id = -1;
        }
        App.thisApp1.educationContract.methods.tasks(task_id).call({
from: App.thisApp1.account }, function (error, task) {
            if (task[2] == "") {
                $('#eventTeacherTask' + as_event_id1).text("-");
            }
            else {
                $('#eventTeacherTask' + as_event_id1).text(task[2]);
            }
            as_event_id1++;
        });
        event_id++;
    }
}
console.log(events);
if (events.length > 0)
    console.log("events " + JSON.parse(events[0].returnValues.time
));
    }
}
);
contentEvents.show();
}
if (App.thisApp1.currentPage == "grades") {
    $('#titlePage').text("My Grades");
    App.thisApp1.educationContract.getPastEvents(
        "grading",
        {
            //filter: { value: [0] },
            fromBlock: 0,
            toBlock: "latest"
        }, (errors, events) => {
        if (!errors) {

```

```

var listMyGradesRow = $('#listMyGradesRow');
listMyGradesRow.empty();
var event_id = 0;
var event_id0 = 0;
var as_event_id = 0;
var as_event_id1 = 0;
for (var i = 0; i < events.length; i++) {
    var resultsEvent = events[i].returnValues;
    if (resultsEvent.student == Number(App.thisApp1.account)) {
        var colorBack = "background-color: rgb(240, 240, 240);";
        if (event_id % 2 == 0)
            colorBack = "background-color: rgb(255, 255, 255);";
        var resultsEvent = events[i].returnValues;
        var dateAddedEvent = getDateTimeFormat(resultsEvent.time);
        ...
        App.thisApp1.educationContract.methods.teachers(resultsEvent
.teacher).call({ from: App.thisApp1.account }, function (error, teacher) {
            App.thisApp1.educationContract.methods.courses(events[event_id0].returnValues.course_id).call({ from: App.thisApp1.account }, function (error, course) {
                console.log("course " + course[1]);
                $('#eventMyGradesTeacher' + as_event_id).text(teacher[2]);
            });

            $('#eventMyGradesCourse' + as_event_id).text(course[1]);
            $('#eventMyGradesCourse' + as_event_id).click(function (
        ) {
            toCourse(events[as_event_id].returnValues.course_id);
        });
            $('#eventMyGradesCourseToTask' + as_event_id).click(function (
tion () {
            toTask(events[as_event_id].returnValues.task_id);
        });
            console.log("as_event_id = " + as_event_id);
            console.log(events[as_event_id].returnValues);
            console.log(events[as_event_id].returnValues.task_id);
            App.thisApp1.educationContract.methods.tasks(events[as_event_id].returnValues.task_id).call({ from: App.thisApp1.account }, function (error, task) {
                console.log("task " + task);
                if (task[2] == "") {
                    $('#eventMyGradesTask' + as_event_id1).text("-");
                }
                else {
                    $('#eventMyGradesTask' + as_event_id1).text(task[2]);
                }
                as_event_id1++;
            });
            as_event_id++;
        });
        event_id0++;
    });
    event_id++;
}
}
console.log(events);
}
}
);
contentCourses.hide();
...
navigatorCourses.show();
}
}

```

```

    }
}

async regTeacher() {
    //створення нового учасника мережі
    var newAcc = App.thisApp1.web3.eth.accounts.create();
    //console.log(newAcc);
    //outpit: {address: "0x9936e190eB228783067214e29bdad2c590A6Dde4", privateKey
: "0xd92069d2d96979ef112f643a89900617a2a90fe9370c3a6511a5ae1cd02d661e", signTran
saction: f, sign: f, encrypt: f}

    //додавання нового учасника в список
    App.thisApp1.accounts.push(newAcc.address);
    console.log(App.thisApp1.web3.eth.getAccounts());

    var account = newAcc.address;
    var pkey = "" + newAcc.privateKey;

    App.thisApp1.educationContract.methods.registerTeacher(profile, fullName, em
ail, account, pkey, "sdfsdf", App.thisApp1.accountTempPrivateKey).send({ from: A
pp.thisApp1.account, gas: 3000000 }, function (receint) {
        if (receint == null) {
            console.log("successfully registered teacher + " + account);
        }
        else
            console.log("Error: " + receint);
    });
}

async sendWork() {
    var fileLoad = $('#fileLoad1');
    console.log(fileLoad.val());
    $('#statusSendWork').show();
    const key = new NodeRSA({ b: 1024 });
    var ok = false;
    for (var i = 0; i < fileLoad[0].files.length; i++) {
        var selFile = fileLoad[0].files[i];
        console.log("file = " + selFile);
        console.log("name = " + selFile.name);
        console.log("type = " + selFile.type);

        //ограничен размер файла
        if (selFile.size > 10485760) {
            console.log("Big file!");
            //$('#maxSizeFile').hide();
            $('#bigFileError').show();
            //formInputFile.reset();
            fileLoad.val('');
            //fileLoadBigFile.show();
            $('#statusSendWork').hide();
        }
        else {
            ok = true;
            const reader = new window.FileReader();
            reader.readAsArrayBuffer(selFile);
            reader.onloadend = () => {
                const buffer = Buffer(reader.result);
                console.log('buffer', buffer);
                $('#maxSizeFile').hide();
                $('#bigFileError').hide();
                App.thisApp1.educationContract.methods.courses(App.thisApp1.currentCou
rseId).call({ from: App.thisApp1.account }, function (error, course) {
                    key.importKey(course[4], 'pkcs8-public-pem');
                });
            };
        }
    }
}

```

```

        const encrypted = key.encrypt(buffer, 'base64');
        //додавання даних в IPFS, в такому випадку додається зашифрований фа
        йл публічним ключем
        ipfs.add(encrypted)
            .then(result => {
                //console.log(result[0].hash)
                //при успішному додаванні отримуємо хеш файлу (шлях до повного ф
айлу в IPFS)

                console.log("hash file ipfs = " + result.path)
                //https://ipfs.infura.io/ipfs/QmdcK5ZnynfS6w9TSdY5KrwTYwDiVWqkMK
V1P8AQ6353Nm

                App.thisApp1.educationContract.methods.handOverWork(App.thisApp1
.currentCourseId, App.thisApp1.currentTaskId, App.thisApp1.accountTempPrivateKey
).send({ from: App.thisApp1.account, gas: 3000000 }, function (receipt) {
                    if (receipt == null) {
                        console.log("event Created!");
                        //console.log(result);
                        // $('#statusSendWork').hide();
                        App.thisApp1.renderApp();
                    }
                    else
                        console.log("Error: " + receipt);
                });
                App.thisApp1.educationContract.methods.sendWork(App.thisApp1.cur
rentCourseId, App.thisApp1.currentTaskId, result.path, selFile.name, selFile.typ
e, App.thisApp1.accountTempPrivateKey).send({ from: App.thisApp1.account, gas: 3
000000 }, function (receipt) {
                    if (receipt == null) {
                        console.log("work was successfully sent!");
                        //console.log(result);
                        $('#statusSendWork').hide();
                        App.thisApp1.renderApp();
                    }
                    else {
                        console.log(error);
                        $('#statusSendWork').hide();
                    }
                });
            })
            .catch(err => {
                //помилка
                console.log(err)
                $('#statusSendWork').hide();
            });
        });
    }
}

async setGrade() {
    var grade = $('#grade0').val();
    App.thisApp1.educationContract.methods.setGrade(App.thisApp1.currentCourseId
, App.thisApp1.currentTaskId, parseInt(grade), App.thisApp1.currentStudent, App.
thisApp1.accountTempPrivateKey).send({ from: App.thisApp1.account, gas: 3000000
}, function (receipt) {
        if (receipt == null) {
            App.thisApp1.renderApp();
        }
        else
            console.log("Error: " + receipt);
    });
}

```

```

async addCourse() {
  var name = $('#newCourseName').val();
  $('#addCourseStatus').text("Generating keys and course is being added, please wait...");

  //генерація ключів для курсу
  const key = new NodeRSA({ b: 1024 });
  const coursePublicKey = key.exportKey('pkcs8-public-pem');
  const coursePrivateKey = key.exportKey('pkcs1-pem');

  //імпорт ключів викладача
  const userKey = new NodeRSA({ b: 1024 });
  App.thisApp1.educationContract.methods.teachers(App.thisApp1.account).call({
    from: App.thisApp1.account }, function (error, teacher) {
    userKey.importKey(teacher[5], 'pkcs8-public-pem');
    userKey.importKey(App.thisApp1.accountTempPrivateKey, 'pkcs1-pem');

    //шифрування приватного ключа публічним ключем викладача
    const encryptedCoursePrivateKey = userKey.encrypt(coursePrivateKey, 'base64');

    //виклик методу смарт-контракту
    App.thisApp1.educationContract.methods.addCourse(name, App.thisApp1.accountTempPrivateKey, coursePublicKey, encryptedCoursePrivateKey).send({ from: App.thisApp1.account, gas: 3000000 }, function (receipt) {
      if (receipt == null) {
        App.thisApp1.currentPage = "courses";
        App.thisApp1.renderApp();
      }
      else
        console.log("Error: " + receipt);
    });
  });
}

async addTask() {
  var name = $('#nameNewTask').val();
  var maxGrade = $('#maxGradeNewTask').val();
  App.thisApp1.educationContract.methods.addTask(name, App.thisApp1.currentCourseId, maxGrade, App.thisApp1.accountTempPrivateKey).send({ from: App.thisApp1.account, gas: 3000000 }, function (receipt) {
    if (receipt == null) {
      App.thisApp1.currentPage = "course";
      App.thisApp1.renderApp();
    }
    else
      console.log("Error: " + receipt);
  });
}

async addStudentToCourse() {
  var email = $('#emailAddStudent').val();
  // за тим же принципом, як в addTeacherToCourse(), але на практиці використовується registerAllUsers()
}

async addTeacherToCourse() {
  var email = $('#emailAddTeacher').val();

  //імпорт свого приватного ключа, який потім потрібен для розшифрування приватного ключа курсу з блокчейна
  const key = new NodeRSA({ b: 1024 });
  key.importKey(App.thisApp1.accountTempPrivateKey, 'pkcs1-pem');

```

```

    //виклик методу смарт-
    контракту для отримання шифрованого приватного ключа курсу
    App.thisApp1.educationContract.methods.getEnPrivateKeyCourse(App.thisApp1.ac
count, App.thisApp1.currentCourseId).call({ from: App.thisApp1.account }, functi
on (error, enPKeyCourse) {
    const coursePrivateKey = key.decrypt(enPKeyCourse);
    console.log("decrypted course PKey = " + coursePrivateKey);
    //виклик методу смарт-контракту для отримання препода по е-
    mail, необхідне для отримання його публічного ключа
    App.thisApp1.educationContract.methods.getTeacherFromEmail(email).call({ f
rom: App.thisApp1.account }, function (error, teacher) {
    const addedTeacherKey = new NodeRSA({ b: 1024 });
    //отримання публічного ключа препода по e-mail
    addedTeacherKey.importKey(teacher[5], 'pkcs8-public-pem');
    //шифрування приватного ключа курсу ключем доданого викладача і збережен
    ня в блокчейн
    const newEnCoursePrivateKey = addedTeacherKey.encrypt(coursePrivateKey);

    console.log(newEnCoursePrivateKey.toString());
    App.thisApp1.educationContract.methods.addTeacherEmail(email, App.thisAp
p1.currentCourseId, App.thisApp1.accountTempPrivateKey, newEnCoursePrivateKey.to
String()).send({ from: App.thisApp1.account, gas: 3000000 }, function (receint)
{
    if (receint == null) {
        App.thisApp1.currentPage = "course";
        App.thisApp1.renderApp();
    }
    else
        console.log("Error: " + receint);
    });
});
});
}

async addStudentsToCourse() {
    var group = $('#groupAddStudents').val();
    App.thisApp1.educationContract.methods.addStudents(App.thisApp1.currentCours
eId, group, App.thisApp1.accountTempPrivateKey).send({ from: App.thisApp1.account
t, gas: 3000000 }, function (receint) {
    if (receint == null) {
        App.thisApp1.currentPage = "course";
        App.thisApp1.renderApp();
    }
    else
        console.log("Error: " + receint);
    });
});

async registerAllUsers() {
    App.thisApp1.tempAccounts.push({ address: App.thisApp1.account, privateKey:
App.thisApp1.accountTempPrivateKey });
    console.log(App.thisApp1.accounts);
    console.log(App.thisApp1.accounts.length);
    const publicKey = '';
    for (var i = 1; i < App.thisApp1.accounts.length; i++) {
        // ініціалізація і генерація ключів
        const key = new NodeRSA({ b: 1024 });
        // отримання ключів
        publicKey = key.exportKey('pkcs8-public-pem');
        const privateKey = key.exportKey('pkcs1-pem');
        console.log("key = " + key);
        console.log("publicKey = " + publicKey);
    }
}

```

```

        console.log("privateKey = " + privateKey);
        App.thisApp1.tempAccounts.push({ address: App.thisApp1.accounts[i], privateKey: privateKey }); //newAcc.privateKey
    }
    for (var i = 1; i <= 3; i++) {
        //вызов метода смарт-контракта
        App.thisApp1.educationContract.methods.registerTeacher("profile", "Full name " + i, "teacher" + i + "@gmail.com", App.thisApp1.tempAccounts[i].address, App.thisApp1.tempAccounts[i].privateKey, publicKey, App.thisApp1.accountTempPrivateKey).send({ from: App.thisApp1.account, gas: 3000000 }, function (receipt) {
            //якщо немає помилок
            if (receipt == null) {
                console.log("successfully registered teacher + ");
            }
            else
                console.log("Error: " + receipt);
        });
    }
    for (var i = 4; i < App.thisApp1.tempAccounts.length; i++) {
        var group = "KNT-137";
        if (i >= 4 + (App.thisApp1.tempAccounts.length - 4) / 2)
            group = "KNT-117";
        App.thisApp1.educationContract.methods.registerStudent("profile", "Full name " + i, "student" + (i - 3) + "@gmail.com", group, App.thisApp1.tempAccounts[i].address, App.thisApp1.tempAccounts[i].privateKey, publicKey, App.thisApp1.accountTempPrivateKey).send({ from: App.thisApp1.account, gas: 3000000 }, function (receipt) {
            if (receipt == null) {
                console.log("successfully registered student + ");
            }
            else
                console.log("Error: " + receipt);
        });
    }
    console.log(JSON.stringify(App.thisApp1.tempAccounts, null, 4));
    App.thisApp1.renderApp();
}
render() {
    return (
        <div></div>
    );
}
}
function toLogin(_address, _tempPKey) {
    console.log("le1");
    App.thisApp1.account = _address;
    App.thisApp1.accountTempPrivateKey = _tempPKey;
    if (App.thisApp1.account == '0x0' || App.thisApp1.account == '' || App.thisApp1.account == null) {
        App.thisApp1.currentRole = '';
        App.thisApp1.renderApp();
    }
    else {
        App.thisApp1.educationContract.methods.isTeacherF(App.thisApp1.account).call({ from: App.thisApp1.tempAccounts[0].address }, function (error, result) {
            if (result) {
                App.thisApp1.currentRole = 'Teacher';
                App.thisApp1.currentPage = "courses";
                App.thisApp1.renderApp();
            }
            else {

```

```

    App.thisApp1.educationContract.methods.isStudentF(App.thisApp1.account).
    call({ from: App.thisApp1.tempAccounts[0].address }, function (error, result) {
        if (result) {
            App.thisApp1.currentRole = 'Student';
            App.thisApp1.currentPage = "courses";
            App.thisApp1.renderApp();
        }
        else {
            App.thisApp1.educationContract.methods.isAdminF(App.thisApp1.account
            ).call({ from: App.thisApp1.tempAccounts[0].address }, function (error, result)
            {
                if (result) {
                    App.thisApp1.currentRole = 'Admin';
                    App.thisApp1.currentPage = "register";
                    App.thisApp1.renderApp();
                }
            });
        }
    });
}
});
}
}
function toExit() {
    App.thisApp1.account = '0x0';
    App.thisApp1.currentRole = '';
    App.thisApp1.accountTempPrivateKey = '';
    App.thisApp1.renderApp();
}
function toCourse(id) {
    App.thisApp1.currentPage = "course";
    App.thisApp1.currentNavigationCourse = 0;
    App.thisApp1.currentCourseId = id;
    App.thisApp1.renderApp();
}
function toTask(id) {
    App.thisApp1.currentPage = "task";
    App.thisApp1.currentTaskId = id;
    App.thisApp1.renderApp();
}
export default App;

```

ДОДАТОК В
Слайди презентації

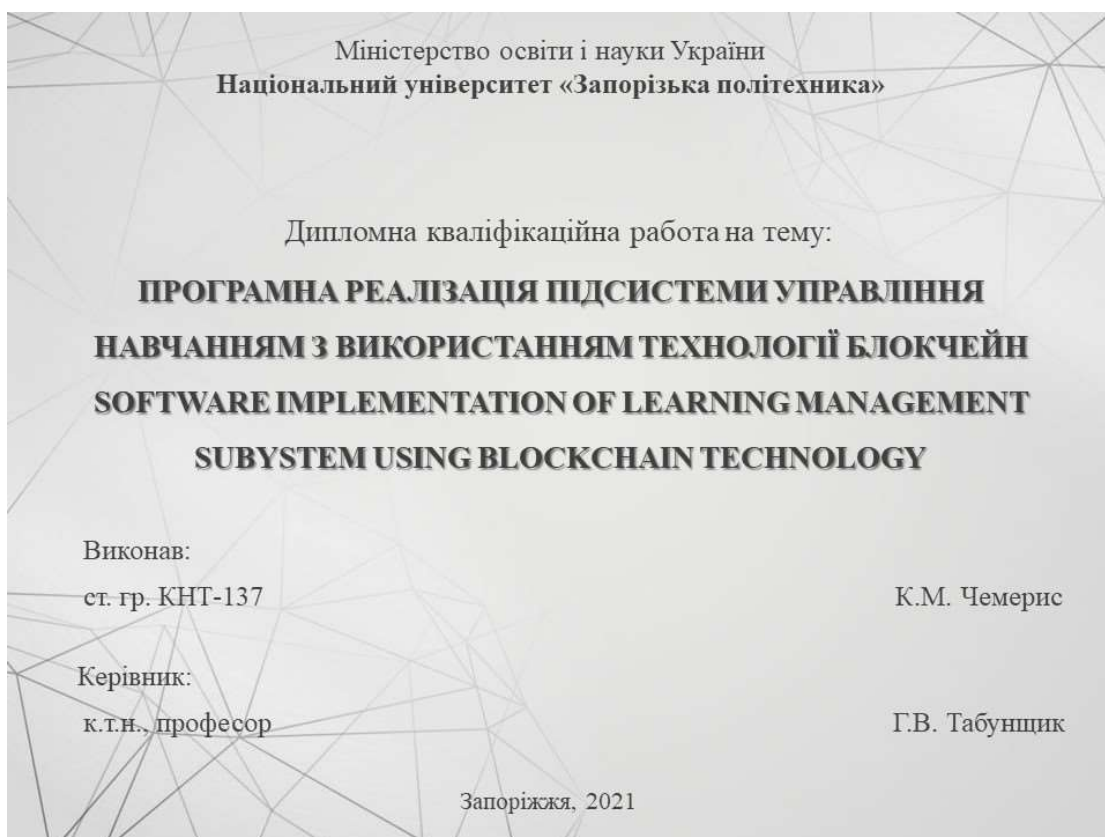


Рисунок В.1 – Слайд 1

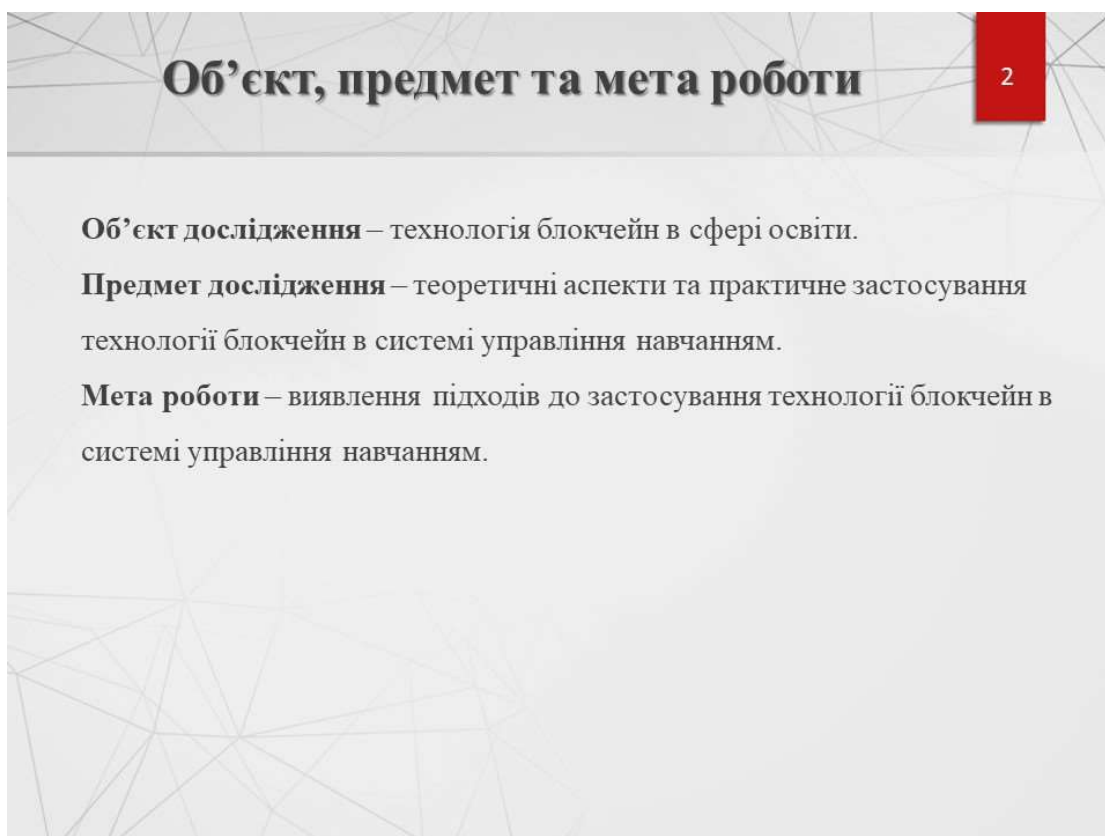


Рисунок В.2 – Слайд 2

Завдання роботи

3

Для досягнення поставленої мети необхідно виконати ряд завдань:

- провести аналіз джерел за темою роботи технології блокчейн, вивчити її принципи роботи;
- провести аналіз джерел в області застосування технології блокчейн в різних сферах діяльності, в сфері освіти;
- виділити основні напрямки технології блокчейн в області навчання;
- описати підходи до застосування технології блокчейн в системі управління навчанням;
- розробити підсистему управління навчання і застосувати технологію блокчейн;
- оцінити спроби практичного впровадження технології блокчейн в системі управління навчанням, виявити головні проблеми та можливі шляхи розвитку технології блокчейна.

Рисунок В.3 – Слайд 3

Огляд технології блокчейн

4

Блокчейн – це новітня технологія, яка базується на принципі децентралізації та організовує базу даних, що складається з ланцюжка послідовних блоків, в кожному з яких зберігається інформація різного виду (транзакції, документи про власність, контракти, твори мистецтва і т.д.).

Переваги технології блокчейн:

- децентралізація;
- доказ. кожної транзакції;
- загальний доступ;
- безпека;
- неможливість зміни/видалення;
- висока швидкість;
- постійна робота;
- універсальність.

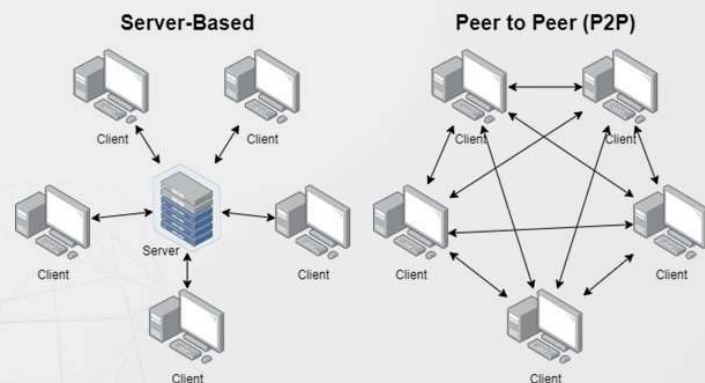


Рисунок В.4 – Слайд 4

Огляд технології блокчейн

5

Блокчейн-ланцюг складається з трьох основних частин:

- блок (містить список транзакцій);
- цепочка (хешування для зв'язування нового блоку даних з попереднім);
- мережа (на кожному вузлі зберігається повна копія всіх транзакцій).

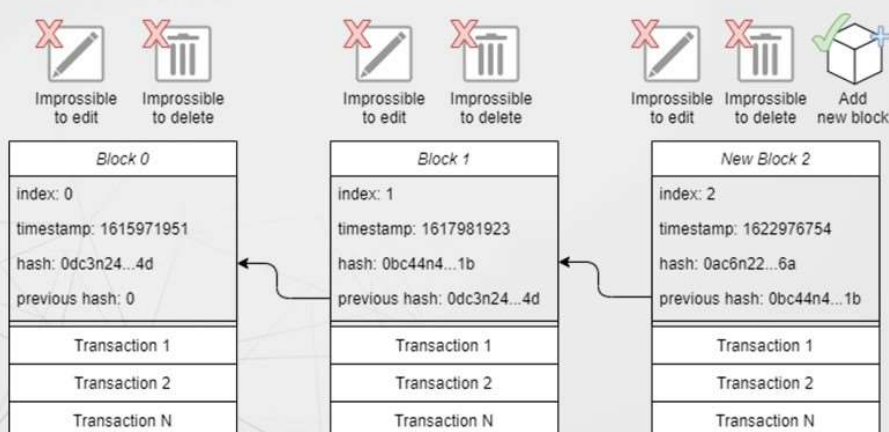


Рисунок В.5 – Слайд 5

Основні функції підсистеми. Діаграма прецедентів

6

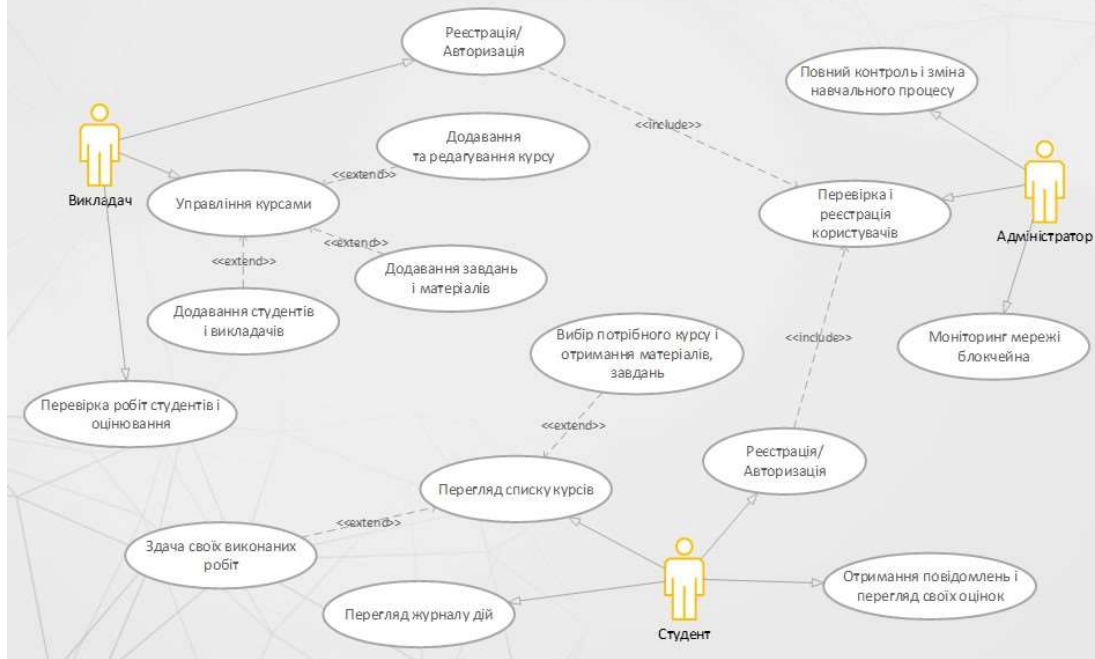


Рисунок В.6 – Слайд 6

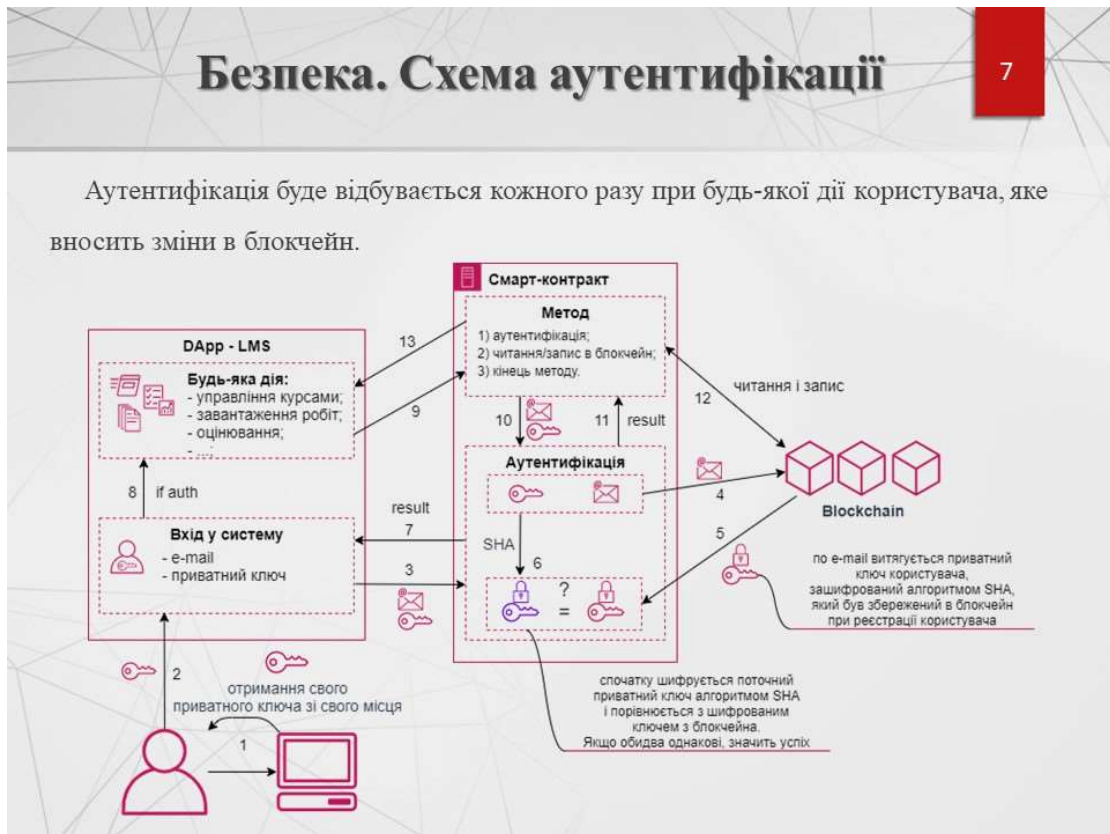


Рисунок В.7 – Слайд 7

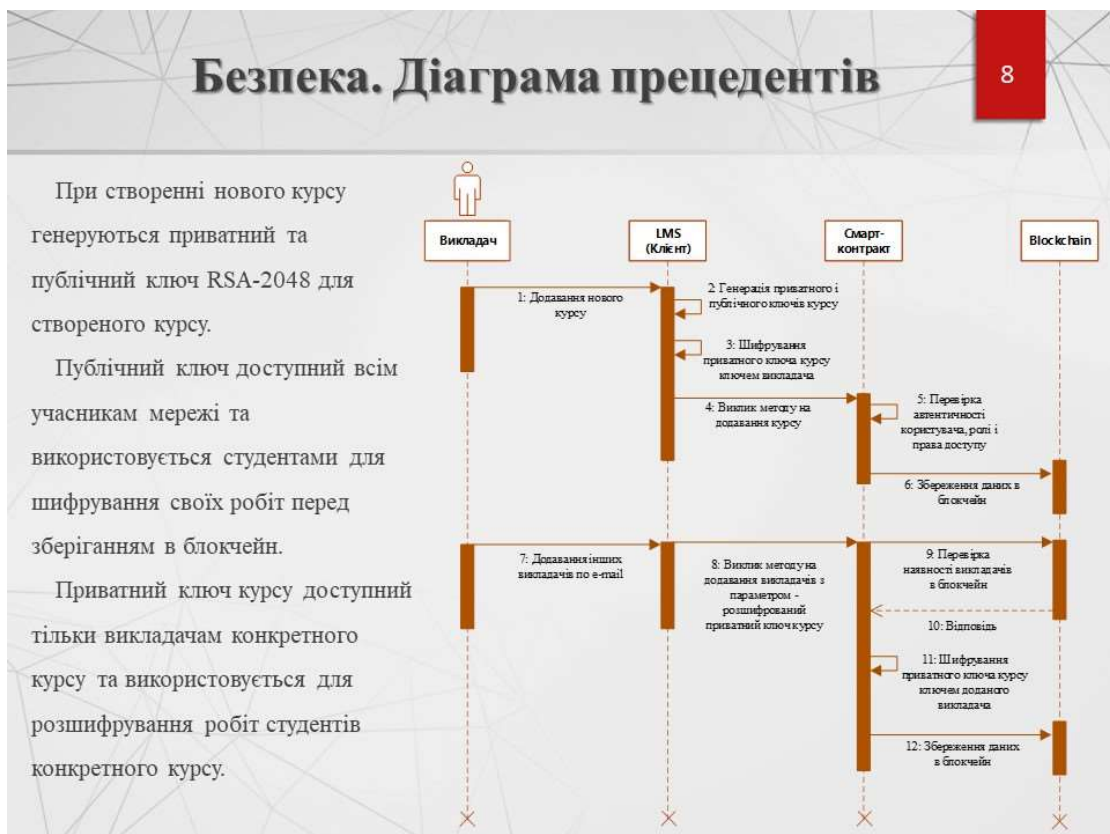


Рисунок В.8 – Слайд 8

Порівняння блокчейн платформ

9

Платформа Хар-ки	Hyperledger Fabric	Ethereum	EOS
Тип блокчейн	Приватний	Публічний	Публічний
Алгоритм консенсусу	Practical Byzantine fault tolerance, Kafka-based Ordering Service, SOLO	Proof-of-Work (в найближчий майбутньому буде використаний алгоритм Proof-of-Stake)	Proof-of-Stake
Час створення блоку	до 10 хв.	10-15 с.	0.5-2 с.
Кількість транзакцій в секунду	близько 1 000	до 20	від 1 000 до 900 000
Смарт-контракти	+	+	+
Дата релізу	2018	Липень 2014	Липень 2017
Мови програмування, які підтримуються	Golang	Solidity, Vyper, Serpent і LLL, а також Mutan	C/C++
Популярні категорії Dapp	бізнес, спеціальні DApp для підприємств	обмінники, гаманці, зберігання, фінанси, управління, енергетика, соціальні мережі, медіа, ідентифікатори, безпека і страхування вантажу, ігри	звичайні і азартні ігри

Рисунок В.9 – Слайд 9

Вибір мови програмування. Компоненти для розробки

10

Взаємодія користувача з децентралізованим застосунком буде відбуватися у браузері, тому клієнтська частина цього застосунку розроблена на мовах програмування JavaScript, HTML і CSS.

Смарт-контракти Ethereum розробляються на одному з декількох мов – Solidity, Vyper, Serpent і LLL, а також Mutan. У нашому випадку для розробки смарт-контракту Ethereum вибрана найбільш популярна об'єктно-орієнтована і предметно-орієнтована мова Solidity, яка схожа на мови C/C++ і JavaScript.

Необхідні наступні доп. компоненти для подальшої розробки:

- децентралізоване сховище IPFS;
- Node.js;
- Web3.js;
- Truffle;
- Ganache.

Рисунок В.10 – Слайд 10

Порівняння середовищ розробки

11

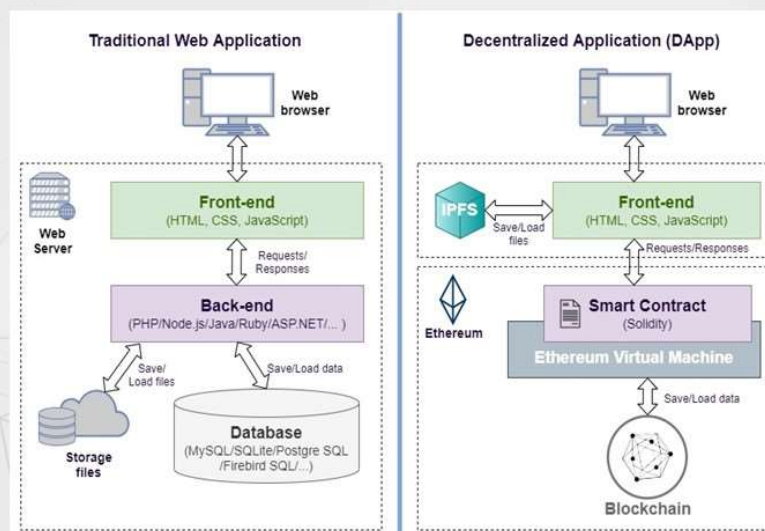
Хар-ки	Atom	Visual Studio Code	WebStorm
Дизайн	5/5	5/5	5/5
Зручність	5/5	5/5	5/5
Зрозумілість	5/5	5/5	5/5
Продуктивність	4/5	4/5	5/5
Служба підтримки	+	+	+
Укр./рос. мова	-	+	-
Підсвічування синтаксису	+	+	+
Рефакторинг	+	+	+
Автодоповнення коду	+	+	+
Пошук по проєкту	+	+	+
Налагодження	+	+	+
Тестування застосунків	+	+	+
Вартість	Безкоштовно	Безкоштовно	Платно, 30 днів безкоштовно
Мови, які підтримуються	Практично всі мови	Практично всі мови	JavaScript, CSS, HTML

Рисунок В.11 – Слайд 11

Архітектура децентралізованого застосунку

12

На рис. можна побачити різницю архітектури традиційного вебзастосунку та децентралізованого застосунку.



Три основні компоненти нашого застосунку:

- клієнтська частина (HTML, CSS і JS);
- децентралізоване сховище IPFS;
- децентралізована частина (платформа Ethereum зі СК на мовою Solidity).

Рисунок В.12 – Слайд 12



Рисунок В.13 – Слайд 13

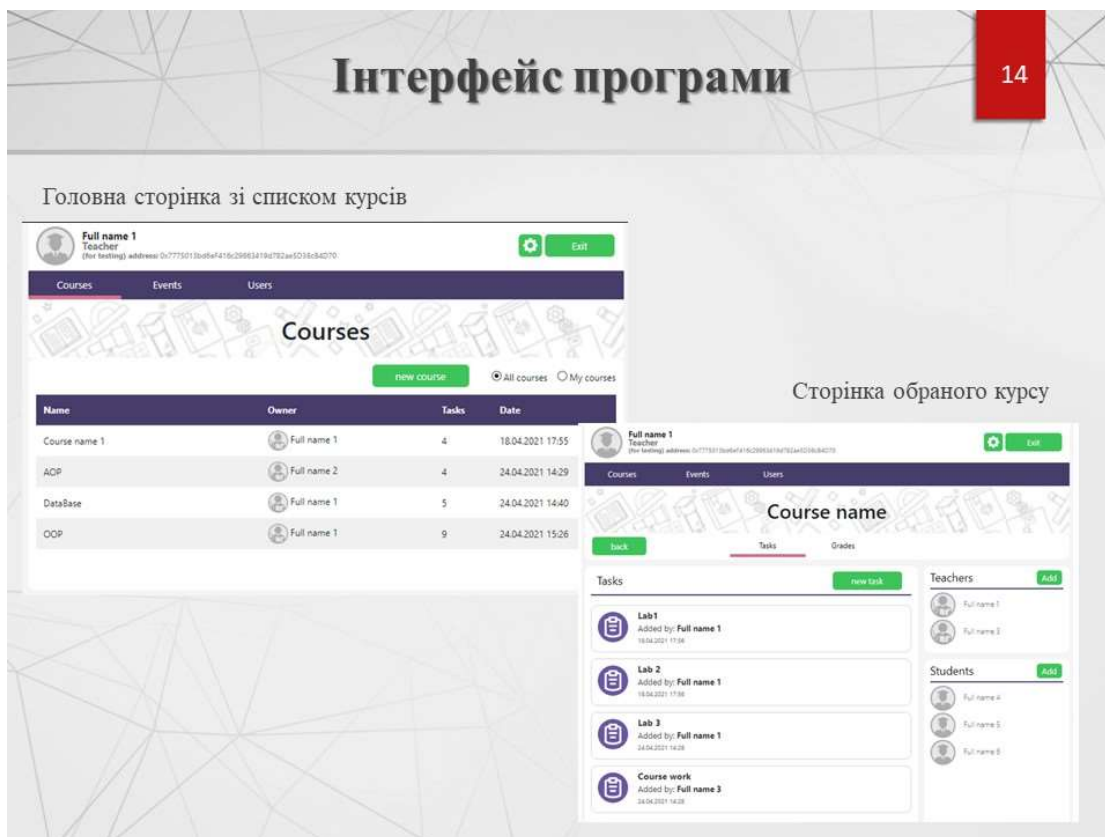


Рисунок В.14 – Слайд 14

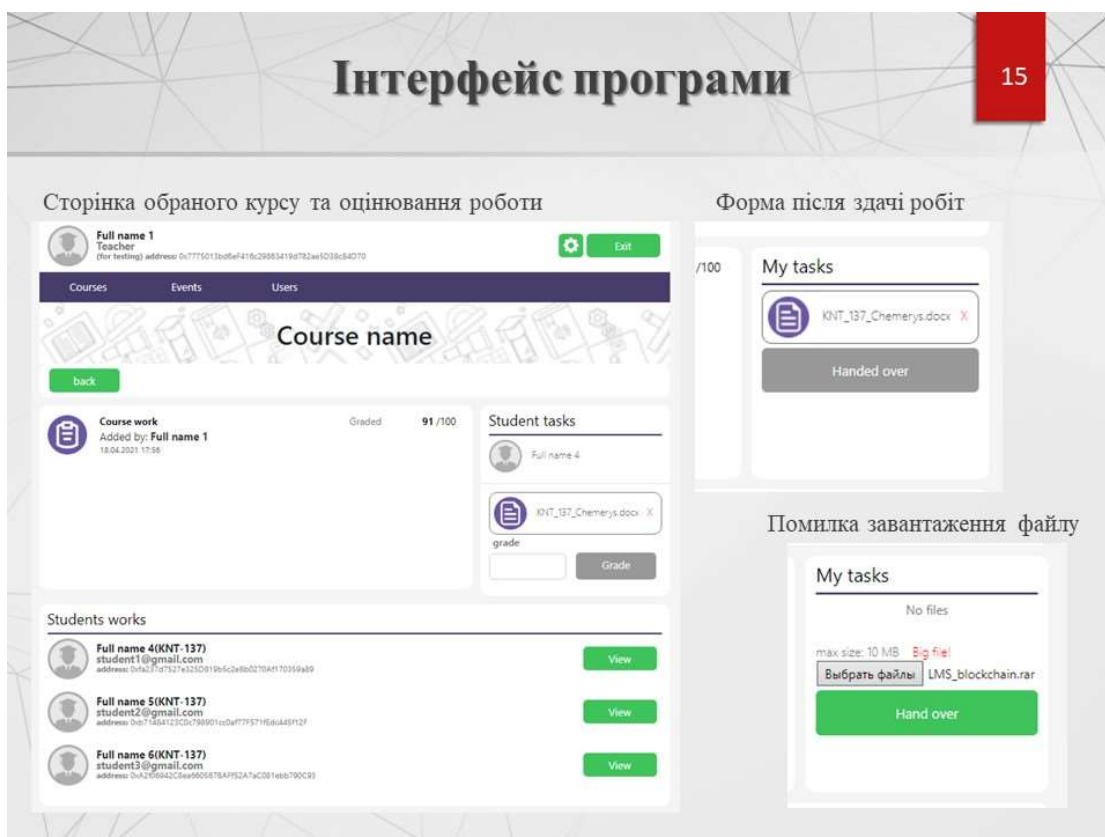


Рисунок В.15 – Слайд 15

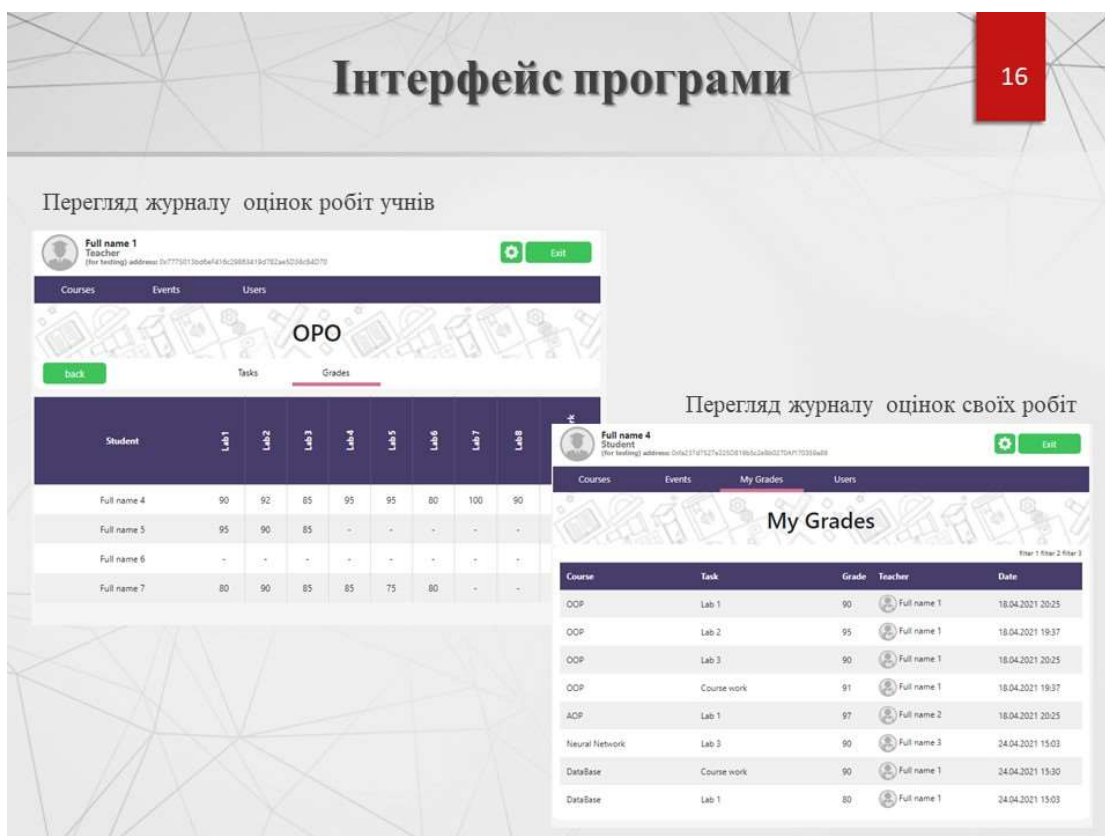


Рисунок В.16 – Слайд 16

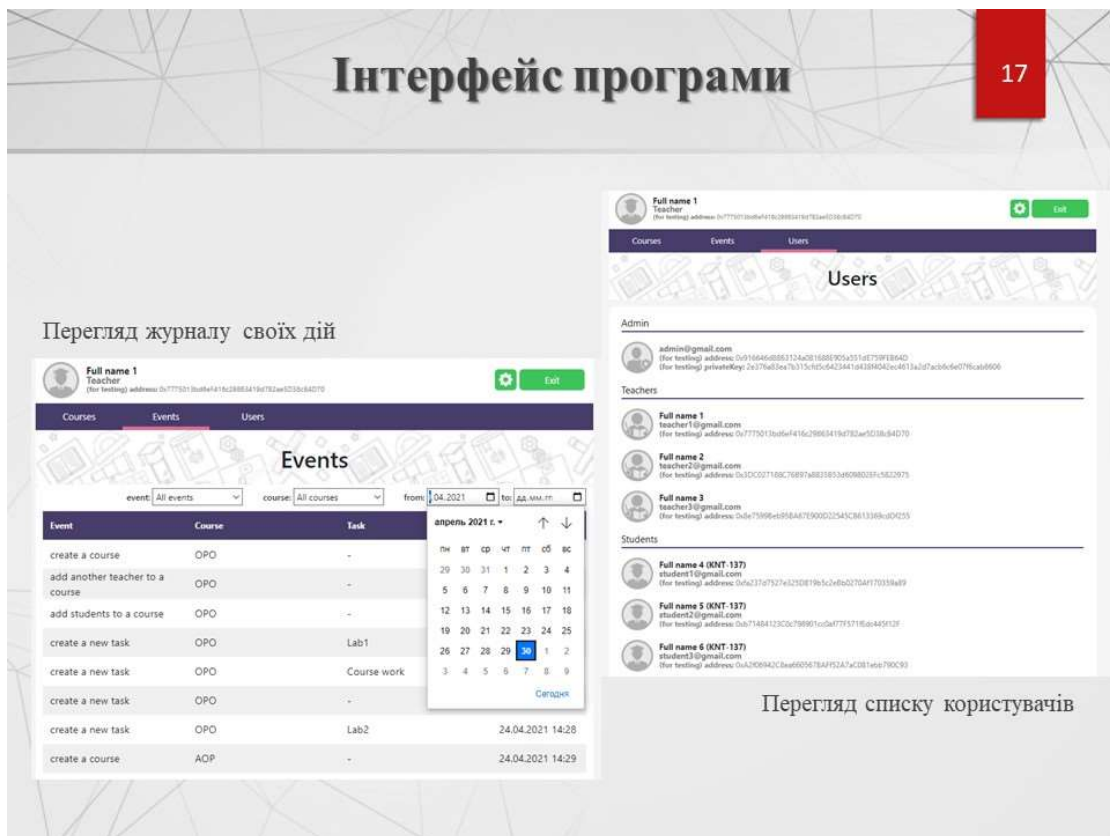


Рисунок В.17 – Слайд 17

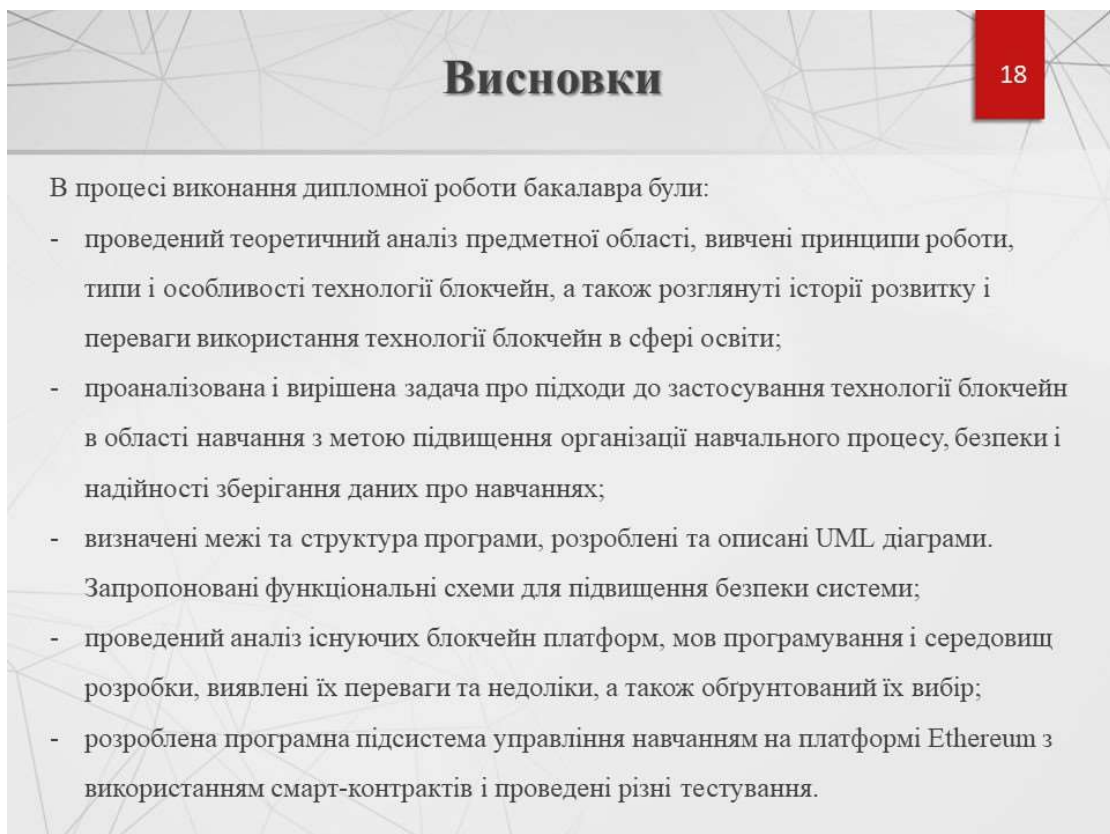


Рисунок В.18 – Слайд 18