НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Навчально-науковий інститут атомної та теплової енергетики

Кафедра цифрових технологій в енергетиці

«До захисту допущено»
Завідувач кафедри
_____ Наталія АУШЕВА
“___”_____2025 р.

Дипломна робота
на здобуття ступеня бакалавр

За освітньою програмою “Цифрові технології в енергетиці”

Спеціальності 122 “Комп’ютерні науки”

на тему: “Протоколювання на основі блокчейн в системах менеджменту АЕС”

Виконав: студент 4 курсу, групи ТР-15
Шевченко Кирило Сергійович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник к.т.н, Олександр КАЛЕНЮК _____ _____
(посада, науковий ступінь, вчене звання, ім’я, ПРІЗВИЩЕ) (підпис)

Рецензент _____ _____
(посада, науковий ступінь, вчене звання, ім’я, ПРІЗВИЩЕ) (підпис)

Н.контроль асистент Антон ПАСІЧНЮК _____ _____
(посада, ім’я, ПРІЗВИЩЕ) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ - 2025

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

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ

Кафедра ЦИФРОВИХ ТЕХНОЛОГІЙ В ЕНЕРГЕТИЦІ

Рівень вищої освіти – перший (бакалаврський)
спеціальність 122 “Комп’ютерні науки”
Освітньо-професійна програма “Цифрові технології в енергетиці”

ЗАТВЕРДЖУЮ
Завідувач кафедри ЦТЕ
_____ Наталія АУШЕВА
(підпис)
“___” _____2025 р.

ЗАВДАННЯ
на дипломну роботу студенту

_____Шевченку Кирилу Сергійовичу_____
(прізвище, ім’я, по батькові)

1. Тема роботи “Протоколювання на основі блокчейн в системах менеджменту АЕС”
Науковий керівник Каленюк Олександр Сергійович, к.т.н
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від “____” _____ 2025 року № _____
2. Термін подання студентом роботи 09.06.2025
3. Вихідні дані до роботи мова програмування Python, веб-фреймворк Flask, шаблонізатор Jinja, СУБД SQLite, середовище розробки Visual Studio Code
4. Перелік питань, які потрібно розробити _____
1) провести аналіз серед наявних аналогів
2) визначити головні функції веб-застосунку, що керує відображенням та публікацією повідомлень, захищених від несанкціонованого доступу
3) аргументувати вибрані інструменти, алгоритми та технології для реалізації веб-системи

4) побудувати архітектуру програмного забезпечення та бази даних

5) створити прикладний програмний веб-інтерфейс

6) представити процес застосування веб-інтерфейсу

5. Орієнтований перелік ілюстративного матеріалу: схематична аналітична основа, що складається з 3 рівнів організаційної культури та 3 рівнів інтеграції управління, схема функції Keccak, функціональна схема «губки» алгоритму SHA-3,журнал подій розробленого програмного забезпечення, скріншоти роботи користувачів з веб-застосунком, скріншоти середовища розробки.

6. Дата видачі завдання 20.03.2025

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1.	Вибір теми роботи	20.03.2025	
2.	Аналіз методів та засобів розв’язання задачі	17-20.04.25	
3.	Розробка архітектури та загальної структури системи	21-25.04.25	
4.	Розробка окремих підсистем	25.04-02.05.25	
5.	Програмна реалізація системи	03-07.05.25	
6.	Оформлення пояснювальної записки	08-12.05.25	
7.	Захист програмного забезпечення	14.05.2025	
8.	Передзахист	27.05.2025	
9.	Захист	16-20.06.2025	

Студент _____ Кирило ШЕВЧЕНКО
(підпис) (прізвище та ініціали)

Керівник _____ Олександр КАЛЕНЮК
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Кваліфікаційна робота бакалавра: 72 с., 18 рисунків, 1 додаток, 22 джерела.

Об’єкт дослідження: процеси журналювання повідомлень у системах управління з використанням технології блокчейн, реалізовані у програмному забезпеченні.

Мета роботи: розробка програмного забезпечення для створення та валідації незмінного ланцюга повідомлень з використанням хеш-функцій та принципів блокчейн.

Методи: аналіз існуючих підходів до забезпечення незмінності даних, веб-програмування з використанням Python, Flask, HTML, CSS, застосування криптографічних хеш-функцій (SHA-3), організація логіки зберігання та перевірки даних.

Результат: програмний інструментарій для створення та валідації незмінного ланцюга повідомлень з використанням хеш-функцій та принципів блокчейн.

Ключові слова: БЛОКЧЕЙН, PYTHON, FLASK, HASH, ЖУРНАЛЮВАННЯ, ЛОГУВАННЯ.

ANNOTATION

Qualification work of bachelor's degree: 72 p., 18 figures, application, 22 sources.

Object of study: processes of logging messages in control systems using blockchain technology implemented in the software.

The goal of the work: to develop software for creating and validating an unchanging chain of messages using hash functions and blockchain principles.

Methods: analysis of existing approaches to ensuring data integrity, web programming using Python, Flask, HTML, CSS, application of cryptographic hash functions (SHA-3), organization of data storage and verification logic.

Result: software toolkit for creating and validating an immutable chain of messages using hash functions and blockchain principles.

Keywords: BLOCKCHAIN, PYTHON, FLASK, HASH, MESSAGE LOGGING.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ, СИМВОЛІВ І ТЕРМІНІВ

АЕС – атомна електростанція

КІ – критична інфраструктура

ЯРО – ядерний регулюючий орган

ІТ – інформаційні технології

ІКТ – інформаційно-комунікаційні технології

КІК – кіберінформаційна компонента

ERP – планування ресурсів підприємства

MES – система управління виробництвом

SCADA – система диспетчерського контролю та збору даних

ПЛК – програмований логічний контролер

РСУ – розподілена система управління

SHA-3 – алгоритм криптографічного хешування SHA-3

ETH – платформа блокчейну Ethereum

HTML – мова розмітки гіпертексту

CSS – таблиці каскадних стилів

SQL – мова структурованих запитів

Блокчейн – розподілений ланцюг блоків, що містять дані, захищені криптографічно.

Протоколювання – процес фіксації подій у системах управління.

Журналювання – технічна реалізація протоколювання зберіганням записів у лог-файлах.

Цілісність даних – збереження даних у незмінному стані.

Автентифікація – перевірка справжності користувача або пристрою.

Авторизація – надання прав доступу до ресурсів після автентифікації.

Криптографія – наука про захист інформації шляхом її шифрування.

Хешування – перетворення даних у фіксований за довжиною код (хеш).

Децентралізація – відсутність єдиного центру управління або контролю.

ВСТУП

Інформаційні системи відіграють ключову роль у забезпеченні стабільного та безпечного функціонування критично важливої інфраструктури, зокрема атомних електростанцій (АЕС). У таких системах надзвичайно важливою є надійна фіксація подій, що відбуваються, адже своєчасне виявлення збоїв, порушень або підозрілих дій може мати вирішальне значення для безпеки. Традиційні методи журналювання подій часто не гарантують цілісність даних, оскільки можуть бути вразливими до змін або видалення з боку внутрішніх чи зовнішніх загроз.

Метою цієї дипломної роботи є розробка програмного забезпечення для захищеного журналювання подій у системі менеджменту АЕС з використанням технології блокчейн. Система базується на веб-інтерфейсі, реалізованому за допомогою фреймворку Flask, із використанням криптографічного хешування алгоритмом SHA-3 для перевірки цілісності даних. Рішення також передбачає створення інструментів для перевірки ланцюга подій та виявлення фальсифікацій.

Актуальність теми обумовлена зростаючими вимогами до надійності інформаційної безпеки на об'єктах критичної інфраструктури, необхідністю мінімізувати ризики людського чинника та підвищити довіру до систем автоматизованого управління.

Об’єктом дослідження є процеси збереження, захисту та перевірки достовірності інформації про події в інформаційних системах АЕС. Предметом дослідження є використання блокчейн-технології як засобу забезпечення незмінності та цілісності журналу подій.

У межах цієї роботи буде проведено аналіз існуючих підходів до протоколювання в системах управління, досліджено можливості інтеграції блокчейн-технологій, реалізовано прототип системи та оцінено її ефективність. Отримані результати можуть бути використані для впровадження на підприємствах енергетичного сектору, а також для подальших досліджень у галузі захищених інформаційних систем.

1 ТЕОРЕТИКО-МЕТОДОЛОГІЧНІ ЗАСАДИ ПРОТОКОЛЮВАННЯ НА БАЗІ БЛОКЧЕЙН У СИСТЕМАХ УПРАВЛІННЯ АЕС

У цьому розділі розглянуто ключові поняття, пов’язані з функціонуванням атомних електростанцій (АЕС) як об’єктів критичної інфраструктури, а також визначено актуальні виклики у сфері протоколювання подій. Особлива увага приділяється теоретичним засадам застосування технології блокчейн для забезпечення цілісності, достовірності та незмінності даних у системах управління АЕС. У межах розділу обґрунтовується доцільність використання децентралізованих технологій у контексті підвищення кібербезпеки та функціональної надійності інформаційних систем атомних електростанцій.

1.1 Особливості функціонування АЕС як об'єкта критичної інфраструктури

Для стабільного та безпечного існування сучасне суспільство та його члени повинні постійно отримувати низку різноманітних продуктів та послуг, мати доступ до низки критично важливих ресурсів тощо. Для цього необхідно створювати та експлуатувати низку активів, мереж та систем, як фізичних, так і віртуальних. Стрімкий розвиток технологій, зокрема в ІТ-секторі, що спостерігається в останні десятиліття, спричинив кардинальні, іноді навіть революційні зміни, що призвели до посилення взаємозв'язку, взаємопроникнення та взаємозалежності різноманітних мереж і систем, виробничих, фінансових, комерційних та інших процесів у всіх сферах життєдіяльності більшості країн світу. Це суттєво підвищує вразливість таких систем та об'єктів і значно ускладнює забезпечення їх надійного захисту та безпеки. Ці процеси розгортаються на тлі різкої ескалації терористичних загроз, особливо на міжнародному рівні,

збільшення кількості техногенних катастроф, у тому числі спричинених людським фактором, зростання кількості природних катаклізмів, викликаних, зокрема, глобальною зміною клімату. Всі ці фактори пояснюють рівень уваги, що приділяється провідними країнами світу захисту об'єктів, систем та ресурсів, які є найбільш критичними для безпеки їх громадян, суспільств та держав [1].

Враховуючи велику кількість факторів, які так чи інакше впливають на життя сучасної людини, суспільства чи держави, необхідно чітко визначити коло тих систем, мереж та об'єктів, функціонування яких забезпечує надання послуг та виконання функцій, критично важливих для існування людини, суспільства та держави. Саме на це питання має дати відповідь визначення «критичної інфраструктури». Зауважимо, що, незважаючи на схожість визначень, наведених у законодавствах провідних країн та міжнародних організацій, існують відмінності, які, вочевидь, відображають національне або інституційне (у випадку ЄС чи НАТО) застосування цього терміну в їхніх регуляторних системах. Законодавство США, які є лідером у розвитку цієї сфери безпеки, трактує КІ як «системи та активи, фізичні або віртуальні, настільки життєво важливі для Сполучених Штатів, що непрацездатність або знищення таких систем та активів матиме виснажливий вплив на безпеку, національну економічну безпеку, національне здоров'я та безпеку населення або будь-яку комбінацію цих питань» [1].

У Стратегії національної безпеки «Україна у світі, що змінюється» (2012) цей термін згадується в контексті визначення шляхів посилення енергетичної безпеки та забезпечення інформаційної безпеки.

Вперше серед «актуальних загроз національній безпеці» виокремлено загрози для КІ, а в розділі про загрози кібербезпеці та інформаційним ресурсам згадано вразливість критичних об'єктів безпеки до кібератак. Крім того, безпека критичної інфраструктури вперше згадується як один із «ключових напрямів державної політики у сфері національної безпеки» та визначаються її пріоритети. Відсутність визначення КІ в українському законодавстві та, відповідно, переліку об'єктів, які можуть бути віднесені до такої інфраструктури, неодноразово блокувала виконання першочергових безпекових завдань, як, наприклад, у п. 6 Рішення Ради національної безпеки і оборони від 1 березня 2014 року «Про

невідкладні заходи щодо забезпечення національної безпеки, суверенітету і територіальної цілісності України» (введено в дію Указом Президента України № 189/2014 від 02 березня 2014 року). 189/2014 від 02 березня 2014 року), яким Міністерству внутрішніх справ України доручено забезпечити «посилення безпеки об'єктів енергетичного сектору та критичної інфраструктури» [2].

Слід також зазначити, що поняття «безпека та захищеність», яке використовується у визначенні захисту критичної інфраструктури, охоплює як безпеку як таку (включаючи фізичний захист), так і операційну безпеку та охорону Провідні країни, які оголосили захист КІ та підвищення її стійкості пріоритетними завданнями безпеки після терористичних атак 11 вересня, вважають, що вона має бути захищеною від усіх загроз (підхід «від усіх небезпек»). Як правило, національне законодавство провідних країн світу розрізняє три основні категорії загроз критичній інфраструктурі, виходячи з їх походження. Але навіть тут існують відмінності. Скажімо, у США та Канаді до загроз КІ відносять зловмисні дії (зловмисні дії груп або окремих осіб, наприклад, терористів чи злочинців), природні небезпеки (урагани, торнадо, землетруси, цунамі, повені, екстремальні погодні умови тощо) та надзвичайні ситуації техногенного характеру (авіакатастрофи, ядерні аварії, пожежі, аварії в системах енергопостачання, викиди небезпечних речовин тощо) [1].

У цьому випадку ми говоримо про так званий ефект доміно та/або каскадний ефект. Що стосується спектру загроз КІ, які існують в Україні, то їх характер визначається безпековим середовищем, в якому наразі перебуває країна. Бойові дії в рамках Антитерористичної операції на Донбасі, що характеризуються високим рівнем зношеності основних фондів і серйозними проблемами з екологічною та техногенною безпекою, стрімко підвищують рівень загрози аварій на об'єктах підвищеної небезпеки, таких як вугільні шахти, об'єкти енергетики, хімічні та металургійні заводи, а також в інженерних мережах, як внаслідок випадкового пошкодження або втрати контролю над технологічними процесами, так і внаслідок терористичних актів або диверсій [3].

Функції ЯРО (ядерного регулюючого органу) включають регуляторні функції, встановлення вимог і норм безпеки, розробку кодексів і стандартів,

незалежну оцінку безпеки і розгляд звітів з аналізу безпеки, видачу дозволів, інспекцію, правозастосування, координацію з іншими національними та міжнародними органами, а також інформування громадськості. Можлива структура ЯРО показана на рисунку 1.1. У деяких країнах ЯРО може складатися з декількох органів, і в такому випадку необхідно вжити ефективних заходів для забезпечення належного виконання та координації регуляторних функцій та обов'язків [3].

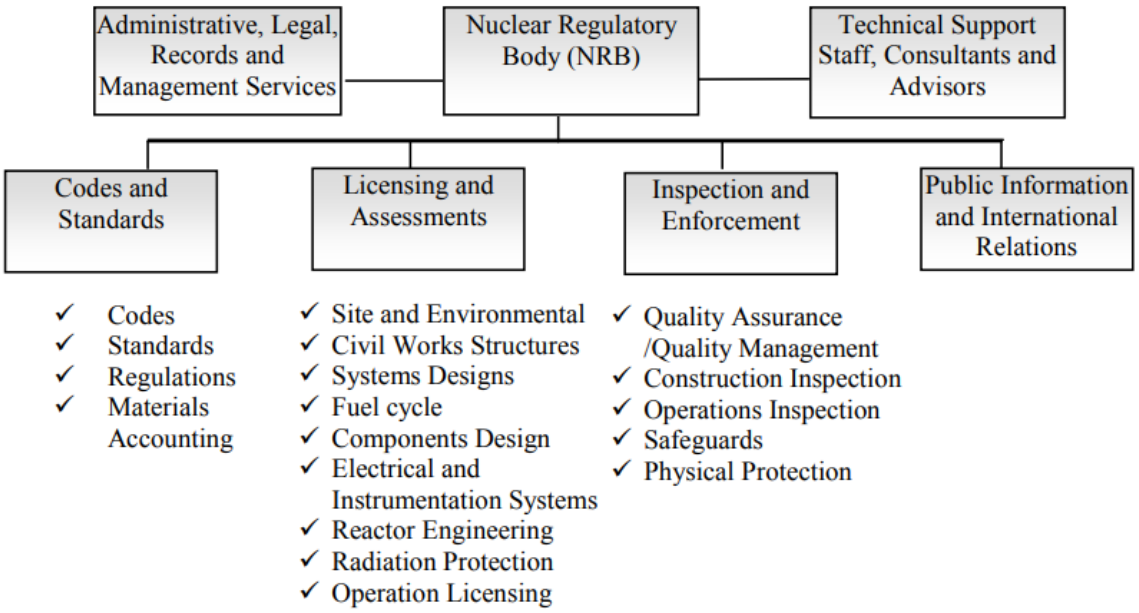

Рисунок 1.1 – Можлива структура ядерного органу регулювання

Можливості ЯРО також можуть бути розширені за рахунок допомоги та порад незалежних консультантів, допоміжних організацій, інших держав або міжнародних організацій, чий досвід є визнаним і добре відпрацьованим. ЯРО повинен створити і підтримувати базову спроможність здійснювати регуляторний аналіз і оцінку, або оцінювати оцінки, виконані для нього консультантами або іншими організаціями. Очікується, що регулюючий орган, який розвивається, буде ліцензувати реактор, який або будується в іншій країні, або вже має ліцензію на будівництво, видану надійним регулюючим органом, наприклад, в країні походження. У таких випадках, при ліцензуванні, ЯРО, що розробляє, дотримуючись своїх власних правил, в значній мірі покладатиметься на роботу,

виконану експертами в країні походження обраного реактора. Для здійснення діяльності, необхідної для ліцензування об'єктів ядерної енергетики, може бути створена низка підрозділів, які виконуватимуть описані вище функції [3].

Ефективна реалізація ядерного проєкту потребує впровадження сучасної системи управління інформаційними потоками. Ще на початкових етапах реалізації проєкту доцільно встановити комп’ютеризовану систему документообігу, яка забезпечуватиме обмін, контроль, збереження та архівацію всієї кореспонденції та технічної документації. Завдяки швидкісному інтернет-з’єднанню різні учасники проєкту, навіть у віддалених локаціях, повинні мати постійний зв’язок. У разі обмеженого доступу до національної мережі, необхідно передбачити розгортання спеціалізованих засобів супутникового зв’язку. Це забезпечить безперервну комунікацію між проєктним майданчиком, офісами постачальників, органами регулювання та іншими зацікавленими сторонами [5].

1.2 Менеджмент систем на АЕС

Цифрові технології все частіше впроваджуються в системи диспетчеризації та управління на ядерних об'єктах. Нові ядерні установки та сучасні проекти ядерних установок використовують цифрові системи контролю та управління, а під час модернізації існуючих установок цифрові технології впроваджуються в системи контролю та управління. Цифрові системи КВПіА дозволяють ефективно обробляти великі обсяги технологічних даних, вимагаючи при цьому меншої участі людини, ніж попередні системи КВПіА. Однак цифрові системи контролю та управління також є більш вразливими до загроз кібербезпеки. Кібератаки можуть поставити під загрозу безпеку та захищеність ядерних установок, що може призвести, наприклад, до втрати контролю над процесом або радіологічних наслідків [7].

На ядерних установках системи ІТ та КІК складаються з наступних рівнів та компонентів (WINS 2019b):

- рівень 4 (IT): Планування ресурсів підприємства (ERP);
- рівень 3 (IT): Система управління виробництвом (MES);
- рівень 2: система диспетчерського контролю та збору даних (SCADA) (I&C): Надає інформацію, зібрану з ПЛК і РСУ, операторам диспетчерської, дозволяючи їм керувати всім процесом або заводом;
- рівень 1: Програмовані логічні контролери (ПЛК)/розподілені системи управління (РСУ) (I&C): ПЛК і РСУ використовуються в локальних комп'ютерах для збору інформації, наприклад, даних датчиків з різних частин об'єкта. Вони також керують різними процесами, наприклад, відкриттям і закриттям клапанів;
- рівень 0 (I&C): Датчики та виконавчі механізми.

Рівні 4 і 3 представляють IT-частину, а рівні 2-0 – частину управління та контролю. Для надання необхідної інформації для заводських та бізнес-додатків може існувати певний рівень зв'язку або комунікації між IT та КВП (WINS 2019b) [6].

Системи IT та КВП схильні до кіберзагроз з різних джерел, таких як зовнішні загрози, внутрішні загрози та загрози технічного розвитку (внутрішні зміни в інфраструктурі, яка контролює та управляє ядерними процесами). Зовнішні загрози включають такі загрози, як комп'ютерний черв'як Stuxnet (Collins and McCombie, 2012). Внутрішні загрози можуть бути навмисними або ненавмисними, наприклад, відкриття вкладення електронної пошти, зараженого шкідливою програмою. Системи ІКТ можуть використовувати стандартні IT-компоненти, що робить їх вразливими до того ж шкідливого програмного забезпечення, яке загрожує сфері автоматизації офісу [7].

Для промислових систем управління розроблено багато стандартів кібербезпеки та керівних принципів. Багато з цих стандартів були розглянуті в (Linnosmaa et al. 2021) з точки зору фінського ядерного контролю та управління. В огляді розглядаються найбільш важливі стандарти серії IEC 62443 (Промислові комунікаційні мережі – IT-безпека мереж і систем), серії ISO 27000 (Інформаційні технології – Методи забезпечення безпеки – Системи управління інформаційною безпекою), IEC 62645 (Атомні електростанції – Прилади, системи управління та

електроенергетичні системи – Вимоги до кібербезпеки), IEC 62859 (Атомні електростанції – Прилади і системи управління – Вимоги до координації безпеки та кібербезпеки) та NSS 17 МАГАТЕ (Комп'ютерна безпека на ядерних установках).

Теоретична основа складається з моделі організаційної культури Едгара Шайна (1992, 2010) і концепції та рівнів інтегрованого управління (Jørgensen et al. 2006), як зображено на рисунку 1.2 [8].

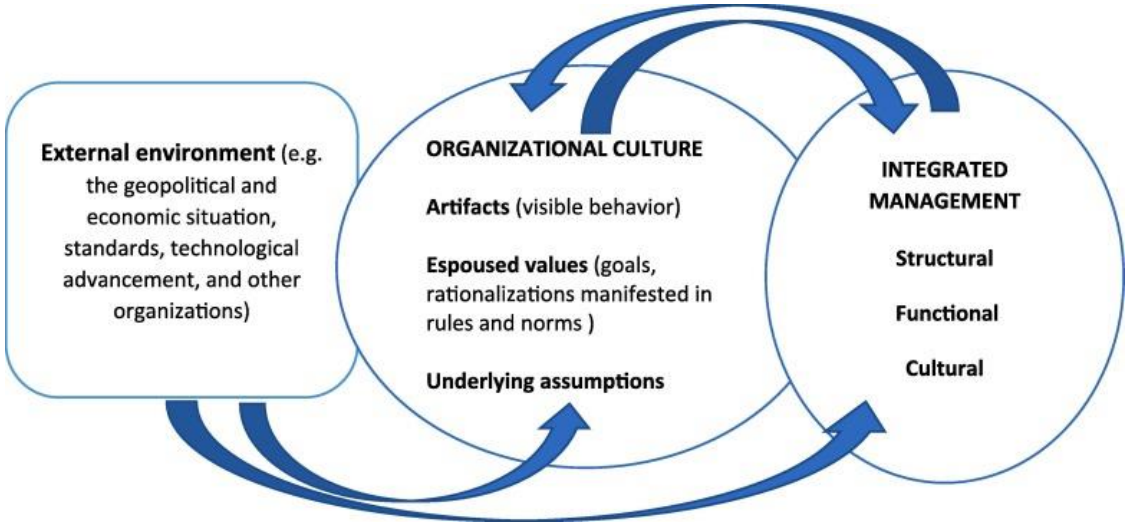

Рисунок 1.2 – Аналітична основа, що складається з 3 рівнів організаційної культури та 3 рівнів інтеграції управління

У сучасних дослідженнях менеджменту атомних електростанцій дедалі більше уваги приділяється взаємозв’язку між організаційною культурою, безпекою та захищеністю. Концепції організаційної культури дають змогу зрозуміти, як у межах підприємства формуються спільні цінності, переконання, правила поведінки та структурні особливості, які прямо впливають на якість управління, у тому числі й на управління безпекою. Організаційна культура – це колективне явище, в межах якого окремі особи є носіями загальноприйнятих підходів, проте саме сукупність взаємодій визначає загальну атмосферу та управлінську модель підприємства.

Існують два основних підходи до розуміння організаційної культури: антропологічний та інструментальний. Антропологічний погляд вважає культуру

органічним продуктом міжособистісної взаємодії, який неможливо напряму керувати, але на який можна впливати. Натомість інструментальний підхід передбачає, що культуру можна цілеспрямовано формувати, наприклад, за допомогою управління ресурсами, реорганізації структур чи навчання персоналу. Обидва підходи є надзвичайно цінними при аналізі управлінських практик на АЕС, особливо коли йдеться про інтеграцію функцій безпеки та захищеності [9].

Аналіз документів здійснювався методом якісного контент-аналізу, тоді як інтерв'ю аналізувалися за допомогою тематичного аналізу. Обидва методи передбачають кодування, інтерпретацію та узагальнення даних для створення нових смислів і глибшого розуміння досліджуваного явища. У результаті були виділені ключові теми, які репрезентують досвід респондентів, а також узагальнені у формі типових цитат і виявлених ідей, що представлені в аналітичних таблицях у наступних розділах.

Інтерпретація отриманих даних здійснювалась із застосуванням моделі трирівневої організаційної культури Едгара Шайна та концепції трьох рівнів інтеграції управління: структурного, функціонального й культурного. Структурна інтеграція включає політики та нормативну базу, функціональна – процедури й системи, а культурна – внутрішнє прийняття спільних цінностей і постійне прагнення до поліпшення [10].

1.3 Поняття та принципи протоколювання

Журналювання, або протоколювання або логування, – це запис подій, що відбуваються в системах і мережах організації. Журнали складаються із записів; кожен запис містить інформацію, пов'язану з конкретною подією, що сталася в системі або мережі. Багато журналів в організації містять записи, пов'язані з комп'ютерною безпекою. Ці журнали комп'ютерної безпеки генеруються багатьма джерелами, зокрема програмним забезпеченням, таким як антивірусне програмне забезпечення, брандмауери, системи виявлення та запобігання вторгненням;

операційними системами на серверах, робочих станціях та мережевому обладнанні; а також додатками. Кількість, обсяг і різноманітність журналів комп'ютерної безпеки значно зросли, що створило потребу в управлінні журналами комп'ютерної безпеки – процесі створення, передачі, зберігання, аналізу та утилізації даних журналів комп'ютерної безпеки. Управління журналами має важливе значення для забезпечення того, щоб записи про комп'ютерну безпеку зберігалися в достатній мірі детально протягом відповідного періоду часу.

Регулярний аналіз журналів корисний для виявлення інцидентів безпеки, порушень політики, шахрайських дій та операційних проблем. Журнали також корисні при проведенні аудиту та криміналістичного аналізу, підтримці внутрішніх розслідувань, встановленні базових показників, визначенні операційних тенденцій та довгострокових проблем.

Організації також можуть зберігати та аналізувати певні журнали для дотримання вимог федерального законодавства та нормативних актів, зокрема Федерального закону про управління інформаційною безпекою 2002 року (FISMA), Закону про переносимість і відповідальність за медичне страхування 1996 року (HIPAA), Закону Сарбейнса-Окслі 2002 року (SOX), Закону Грамма-Ліча-Блайлі (GLBA) та Стандарту безпеки даних індустрії платіжних карток (PCI DSS) [11].

Фундаментальною проблемою в управлінні журналами, яка виникає в багатьох організаціях, є ефективне балансування між обмеженою кількістю ресурсів для управління журналами та безперервним надходженням даних журналів. Створення та зберігання журналів може бути ускладнене кількома факторами, зокрема великою кількістю джерел журналів, неузгодженістю вмісту, форматів і міток часу між джерелами, а також зростаючими обсягами даних журналів. Управління журналами також передбачає захист конфіденційності, цілісності та доступності журналів. Ще однією проблемою в управлінні журналами є забезпечення того, щоб адміністратори безпеки, системні та мережеві адміністратори регулярно виконували ефективний аналіз даних журналів. Виконання наведених нижче рекомендацій має сприяти більш ефективному та результативному управлінню журналами для федеральних департаментів та агентств.

Щоб запровадити та підтримувати успішну діяльність з управління реєстраційними записами, організація повинна розробити стандартні процеси для здійснення управління реєстраційними записами. В рамках процесу планування організація повинна визначити свої вимоги та цілі щодо ведення журналів. На їх основі організація повинна розробити політику, яка чітко визначає обов'язкові вимоги та запропоновані рекомендації щодо діяльності з управління журналами, включаючи створення, передачу, зберігання, аналіз та утилізацію журналів. Організація також повинна забезпечити, щоб відповідні політики та процедури включали та підтримували вимоги та рекомендації щодо управління журналами [11].

Основні операційні процеси управління журналами зазвичай включають конфігурацію джерел журналів, виконання аналізу журналів, ініціювання реагування на виявлені події та управління довгостроковим зберіганням. Адміністратори мають й інші обов'язки, зокрема такі:

1. Моніторинг стану реєстрації всіх джерел журналів.

2. Моніторинг процесів ротації та архівування журналів.

3. Перевірка наявності оновлень і виправлень для програмного забезпечення для ведення журналів, а також їх придбання, тестування і розгортання.

4. Забезпечення синхронізації годинника кожного хоста для ведення журналів із загальним джерелом часу.

5. Переналаштування реєстрації за необхідності на основі змін політики, технологічних змін та інших факторів Документування та звітування про аномалії в налаштуваннях, конфігураціях та процесах ведення журналів [13].

Наприклад, пристрій мережевої безпеки може виявити атаку на певний комп'ютер; журнали ОС цього комп'ютера можуть вказати, чи був користувач у системі під час атаки і чи була атака успішною. Багато журналів ОС створюються у форматі syslog;. Інші журнали ОС, наприклад, у системах Windows, зберігаються у пропрієтарних форматах. На рисунку 1.3 показано приклад даних журналу, експортованих із журналу безпеки Windows [14].

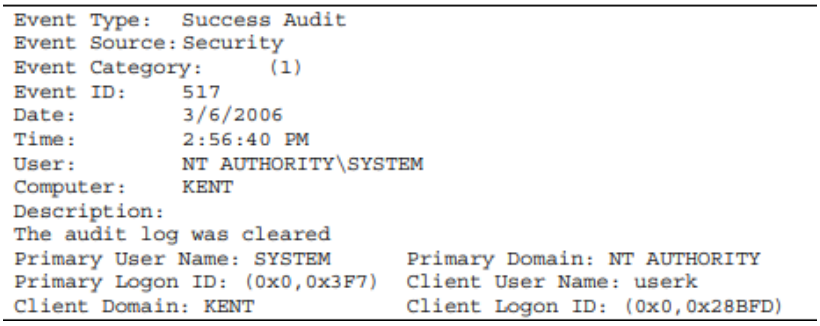

Рисунок 1.3 – Приклад даних журналу, експортованих із журналу безпеки Windows

Операційні системи та програмне забезпечення для забезпечення безпеки забезпечують основу та захист для додатків, які використовуються для зберігання, доступу та маніпулювання даними, що використовуються для бізнес-процесів організації. Більшість організацій покладаються на різноманітні комерційні готові програми, такі як сервери та клієнти електронної пошти, веб-сервери та браузери, файлові сервери та клієнти для обміну файлами, а також сервери та клієнти баз даних. Багато організацій також використовують різні комерційні або урядові готові бізнес-додатки, такі як управління ланцюжками поставок, фінансовий менеджмент, системи закупівель, планування ресурсів підприємства та управління відносинами з клієнтами [14].

На додаток до програмного забезпечення COTS і GOTS, більшість організацій також використовують додатки, розроблені на замовлення, пристосовані до їхніх конкретних вимог.8 Деякі додатки генерують власні файли журналів, тоді як інші використовують можливості ведення журналів операційної системи, на якій вони встановлені. Програми суттєво відрізняються за типами інформації, яку вони реєструють. Нижче перераховані деякі з найбільш поширених типів інформації та потенційні переваги кожного з них:

- запити клієнта і відповіді сервера, які можуть бути дуже корисними для реконструкції послідовності подій і визначення їхнього очевидного результату. Якщо програма реєструє успішні автентифікації користувачів, зазвичай можна визначити, який саме користувач зробив кожен запит;

•інформація про облікові записи, наприклад, успішні та невдалі спроби автентифікації, зміни облікових записів (наприклад, створення та видалення облікових записів, призначення привілеїв облікових записів) та використання привілеїв [14].

Однак, журнали часто мають пропрієтарні формати, що ускладнює їх використання, а дані, які вони містять, часто сильно залежать від контексту, що вимагає більше ресурсів для перегляду їх вмісту. На Рисунку 1.4 наведено приклад запису журналу веб-сервера разом із поясненням інформації, записаної в ньому [15].

Рисунок 1.4 – Приклад даних журналу веб-сервера

Управління журналами може бути корисним для організації у багатьох відношеннях. Воно допомагає гарантувати, що записи про комп'ютерну безпеку зберігатимуться достатньо детально протягом відповідного періоду часу. Регулярний перегляд і аналіз журналів корисний для виявлення інцидентів безпеки, порушень політики, шахрайських дій і операційних проблем незабаром після того, як вони сталися, а також для надання інформації, корисної для вирішення таких

проблем. Журнали також можуть бути корисними для проведення аудиту та криміналістичного аналізу, підтримки внутрішніх розслідувань організації, встановлення базових показників та визначення операційних тенденцій і довгострокових проблем.

1.4 Блокчейн-технологія: архітектура, типи та механізми

Інформаційні та інтернет-технології стрімко розвиваються, тому в комунікації часто використовуються інтерактивні медіа, наприклад, аудіо, зображення та відео. Зображення складають значну частину мультимедіа.

Блокчейн – це захищена база даних, доступ до якої надається через мережу учасників, де актуальна інформація доступна всім учасникам одночасно. Блокчейн – одна з головних технологічних новин останнього десятиліття. Але за поверхневими розмовами не завжди є глибоке, чітке розуміння того, що таке блокчейн, як він працює і для чого він потрібен. Незважаючи на свою репутацію непроникності, основна ідея блокчейну досить проста. І вона має великий потенціал для зміни індустрій знизу догори.

Простіше кажучи, блокчейн – це технологія, яка дозволяє безпечно обмінюватися інформацією. Дані, очевидно, зберігаються в базі даних. Транзакції записуються в обліковій книзі, яка називається реєстр. Блокчейн – це тип розподіленої бази даних або реєстру, що означає, що повноваження щодо оновлення блокчейну розподілені між вузлами, або учасниками, публічної або приватної комп'ютерної мережі. Це відома як технологія розподіленого реєстру (DLT). За внесення змін до блокчейну вузли винагороджуються цифровими токенами або валютою [16].

Блокчейн забезпечує постійний, незмінний і прозорий запис даних і транзакцій. Це, в свою чергу, дає можливість обмінюватися будь-чим, що має цінність, чи то фізичним предметом, чи чимось більш нематеріальним.

Блокчейн має три основні атрибути:

1. По-перше, база даних блокчейну повинна бути криптографічно захищеною. Це означає, що для доступу до бази даних або додавання даних до неї потрібні два криптографічні ключі: відкритий ключ, який, по суті, є адресою в базі даних, і закритий ключ, який є індивідуальним ключем, що має бути аутентифікований мережею.

2. Далі, блокчейн – це цифровий журнал або база даних транзакцій, що означає, що вони відбуваються повністю онлайн.

3. І, нарешті, блокчейн – це база даних, до якої надається спільний доступ у публічній або приватній мережі. Однією з найвідоміших публічних мереж блокчейн є блокчейн Bitcoin. Будь-хто може відкрити гаманець Bitcoin або стати вузлом мережі. Інші блокчейни є приватними мережами. Вони більше застосовуються в банківській сфері та фінтеху, де люди повинні точно знати, хто бере участь, хто має доступ до даних і хто має приватний ключ до бази даних. Інші типи блокчейнів включають консорціумні блокчейни та гібридні блокчейни, які поєднують в собі різні аспекти публічних і приватних блокчейнів.

Незважаючи на весь свій потенціал, блокчейн ще не став тим, хто змінить правила гри, як дехто очікує.

Коли до даних у блокчейні отримують доступ або змінюють їх, запис зберігається у «блоці» разом із записами інших транзакцій. Збережені транзакції шифруються за допомогою унікальних, незмінних хешів. Нові блоки даних не перезаписують старі; вони «зв'язані» між собою, тому будь-які зміни можна відстежити.

Ці блоки зашифрованих даних постійно «прив'язані» один до одного, а транзакції записуються послідовно і нескінченно довго, створюючи ідеальну історію аудиту, яка дозволяє бачити минулі версії блокчейну [17].

Коли в мережу додаються нові дані, більшість вузлів повинні перевірити і підтвердити легітимність нових даних на основі дозволів або економічних стимулів, також відомих як механізми консенсусу. Коли консенсус досягнуто, створюється новий блок, який приєднується до ланцюжка. Після цього всі вузли оновлюються відповідно до реєстру блокчейну.

У публічній мережі блокчейн перший вузол, який достовірно доведе легітимність транзакції, отримує економічний стимул. Цей процес називається «майнінг» [17].

Існує два способи досягнення консенсусу між вузлами блокчейну: через приватні блокчейни, де корпорації, яким довіряють, є сторожами змін чи доповнень до блокчейну, або через публічні, масові блокчейни.

Більшість публічних блокчейнів досягають консенсусу за допомогою системи «доказу роботи» або «доказу частки». У системі підтвердження роботи перший вузол або учасник, який підтверджує додавання нових даних або транзакцію в цифровому реєстрі, отримує певну кількість токенів як винагороду. Щоб завершити процес верифікації, учасник, або «майнер», повинен вирішити криптографічне питання. Перший майнер, який розгадає головоломку, отримує токени.

Спочатку люди на різних блокчейнах майнили як хобі. Але оскільки цей процес є потенційно прибутковим, майнінг блокчейнів був індустріалізований. Ці пули майнінгу блокчейнів привертають увагу великою кількістю енергії, яку вони споживають.

У вересні 2022 року Ethereum, криптовалютна мережа з відкритим вихідним кодом, вирішила проблему енергоспоживання, модернізувавши свою програмну архітектуру до блокчейну з підтвердженням частки. Ця подія, відома як «злиття», розглядається криптофілами як знаменний момент в історії блокчейну. За допомогою системи підтвердження частки інвестори вносять свої криптовалюти в загальний пул в обмін на можливість заробити токени в якості винагороди. У системах з підтвердженням частки майнери оцінюються на основі кількості монет нативного протоколу, які вони мають у своїх цифрових гаманцях, і часу, протягом якого вони їх мають. Майнер з найбільшою кількістю монет на кону має більше шансів бути обраним для підтвердження транзакції і отримати винагороду [17].

Блокчейн і DLT можуть створити нові можливості для бізнесу, зменшуючи ризики і витрати на дотримання нормативних вимог, створюючи більш економічно ефективні транзакції, сприяючи автоматизованому і безпечному виконанню контрактів і підвищуючи прозорість мережі. Розглянемо це докладніше:

• зниження ризиків та витрат на комплаєнс. Банки покладаються на процеси «знай свого клієнта» (KYC) для залучення та утримання клієнтів. Але багато існуючих процесів KYC є застарілими і призводять до витрат у розмірі до 500 мільйонів доларів США на рік для кожного банку. Нова система DLT може вимагати лише одну перевірку KYC для кожного клієнта, що сприятиме підвищенню ефективності, зниженню витрат, підвищенню прозорості та покращенню якості обслуговування клієнтів;

•скоротився до менш ніж чотирьох годин – від випуску до затвердження акредитива;

• автоматизоване та безпечне виконання контрактів. Смарт-контракти – це набори інструкцій, закодованих у токенах, випущених на блокчейні, які можуть самостійно виконуватися за певних умов. Вони можуть уможливити автоматизоване виконання контрактів. Наприклад, одна роздрібна компанія хотіла впорядкувати свої зусилля з управління ланцюгами поставок, тому почала записувати всі процеси і дії, від постачальника до клієнта, і кодувати їх у смарт-контракти на блокчейні. Це не лише полегшило відстеження походження продуктів харчування для їх безпечного споживання, але й вимагало менше людських зусиль і покращило можливість відстежувати втрачені товари.

Всі цифрові активи, включаючи криптовалюти, базуються на технології блокчейн. Децентралізовані фінанси (DeFi) – це група додатків у криптовалюті або блокчейні, покликаних замінити нинішніх фінансових посередників послугами на основі смарт-контрактів. Як і блокчейн, додатки DeFi децентралізовані, а це означає, що кожен, хто має доступ до додатку, має контроль над будь-якими змінами або доповненнями, внесеними до нього. Це означає, що користувачі потенційно мають більш прямий контроль над своїми грошима [18].

Криптовалюта – це лише верхівка айсберга. Сфери використання блокчейну швидко розширюються і виходять за рамки обміну між людьми, особливо коли блокчейн поєднується з іншими новими технологіями. Приклади інших варіантів використання блокчейну включають наступне:

• за допомогою блокчейну компанії можуть створювати незмивний аудиторський слід шляхом послідовного і безстрокового запису транзакцій. Це дозволяє створювати системи, які зберігають статичні записи (наприклад, про право власності на землю) або динамічні записи (наприклад, про обмін активами);

• блокчейн дозволяє компаніям відстежувати транзакцію аж до її поточного стану. Це дозволяє компаніям точно визначити, звідки виникли дані і куди вони були доставлені, що допомагає запобігти витоку даних;

• блокчейн підтримує смарт-контракти.

Хоча блокчейн може потенційно змінити правила гри, виникають сумніви щодо його справжньої цінності для бізнесу. Одне з головних занепокоєнь полягає в тому, що, незважаючи на всі ідеї, гіперболічні заголовки і мільярдні інвестиції, залишається дуже мало практичних, масштабованих варіантів використання блокчейну.

Однією з причин цього є поява конкуруючих технологій. Наприклад, у сфері платежів блокчейн – не єдиний фінтех, який порушує ланцюжок створення вартості: 60% з майже 12 мільярдів доларів, інвестованих у фінтех-компанії США у 2021 році, були спрямовані на платежі та кредитування. З огляду на те, наскільки складними можуть бути блокчейн-рішення, і той факт, що прості рішення часто є найкращими, блокчейн не завжди може бути відповіддю на платіжні виклики [18].

Забігаючи наперед, дехто вважає, що цінність блокчейну полягає в додатках, які демократизують дані, уможливлюють співпрацю та вирішують конкретні больові точки. Дослідження McKinsey показує, що саме в цих конкретних випадках використання блокчейн має найбільший потенціал, а не у сфері фінансових послуг.

За оцінками McKinsey, протягом наступного десятиліття блокчейн розвиватиметься за двома основними напрямками:

1. Зростання блокчейну як сервісу (BaaS). BaaS – це хмарний сервіс, який створює цифрові продукти для середовищ DLT і блокчейну без будь-яких вимог до інфраструктури. Наразі цей напрямок очолюють великі технологічні компанії.

2. Інтероперабельність між блокчейн-мережами та зовнішніми системами. Підвищена інтероперабельність означатиме, що розрізнені блокчейн-мережі та

зовнішні системи зможуть переглядати, отримувати доступ та обмінюватися даними одна одної, зберігаючи при цьому цілісність. Апаратна стандартизація та масштабовані алгоритми консенсусу дозволять використовувати міжмережеві варіанти використання, такі як Інтернет речей на блокчейн-інфраструктурі.

Ці тенденції стануть можливими частково через посилення тиску з боку регуляторів і споживачів, які вимагають більшої прозорості ланцюгів поставок, а частково через економічну невизначеність, оскільки споживачі шукають незалежні, централізовано регульовані системи. А великі корпорації, які запускають успішні пілотні проекти, зміцнюють довіру споживачів та інших організацій [19].

Невзаємозамінні токени (NFT) карбуються на блокчейнах зі смарт-контрактами, таких як Ethereum або Solana. NFT представляють унікальні активи, які неможливо відтворити – в цьому і полягає їхня особливість – і якими не можна обмінятися один на один. До таких активів можна віднести що завгодно – від картини Пікассо до цифрового собачого мема «This is fine». Оскільки NFT створюються на основі блокчейнів, їхні унікальні ідентифікатори та право власності можна перевірити за допомогою реєстру. У деяких NFT власник отримує роялті щоразу, коли NFT торгується [19].

Ринок NFT надзвичайно волатильний: у 2021 році один NFT, створений цифровим художником Майком Вінкельманом, також відомим як Beeple, був проданий на аукціоні Christie's за $69,3 млн. Але з літа 2022 року продажі NFT різко скоротилися. Станом на 2023 рік, згідно зі звітом криптоаналітичної компанії dappGambl, 95% NFT практично нічого не коштують.

Блокчейн називають «машиною правди». Хоча він усуває багато проблем, що виникли в веб 2.0, таких як піратство і шахрайство, він не є панацеєю для цифрової безпеки.

У 2022 році хакери вчинили саме так, викравши понад 600 мільйонів доларів з ігрової блокчейн-платформи Ronin Network, орієнтованої на ігрову індустрію. Цей виклик, на додаток до перешкод, пов'язаних з масштабуванням і стандартизацією, необхідно буде вирішити. Але блокчейн все ще має значний потенціал як для бізнесу, так і для суспільства [20].

2 ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розділ присвячений процесу розробки програмного забезпечення для протоколювання подій у системах управління АЕС із використанням блокчейн-технологій. Розглядаються ключові аспекти проектування, зокрема визначення функціональних та нефункціональних вимог, моделювання архітектури веб-застосунку, а також побудова структури бази даних. Запропоноване рішення має відповідати вимогам до безпеки, достовірності даних та надійності роботи у критичних умовах.

2.1 Функціональні вимоги

Програмне забезпечення повинно забезпечувати створення та збереження повідомлень у захищеному вигляді з використанням алгоритму хешування SHA-3, що дозволяє гарантувати їхню цілісність у межах побудованого ланцюга [20]. Кожне нове повідомлення повинно автоматично містити хеш попереднього запису, формуючи таким чином логічну послідовність, що відповідає принципу блокчейну. Система повинна автоматично зберігати дату та час створення повідомлення, його заголовок, основний текст та відповідне хеш-значення у базі даних.

Продукт також повинен надавати веб-інтерфейс, за допомогою якого користувачі можуть авторизуватися, переглядати наявні повідомлення та додавати нові. Після створення повідомлення система повинна розраховувати його хеш, зберігати всі дані в базу та формувати оновлений ланцюг. Початкове повідомлення та обліковий запис повинні створюватися автоматично під час ініціалізації бази даних.

Окремий програмний модуль повинен дозволяти редагування записів. Також він повинен виконувати перевірку достовірності постів, порівнюючи фактичні хеші з тими, що були задекларовані під час створення. У випадку збігу хешів система

повинна підтвердити валідність ланцюга, в іншому випадку – повідомити про виявлену невідповідність.

Таким чином, програмне забезпечення повинно реалізовувати функції збереження, захисту, верифікації та перегляду повідомлень у формі безпечного ланцюга подій, що забезпечує цілісність і довіру до внесених даних.

2.2 Нефункціональні вимоги

Розроблюване програмне забезпечення для протоколювання подій на основі блокчейн ланцюга повинно бути стабільним, надійним і забезпечувати безперебійну роботу упродовж тривалого часу без втрати даних. Воно повинно працювати у середовищі операційної системи Windows та підтримувати виконання на комп'ютерах з базовими апаратними характеристиками. Для роботи програмного забезпечення повинні використовуватись лише відкриті бібліотеки та фреймворки, зокрема Python, Flask, SQLite, hashlib та HTML/CSS [20].

Система повинна бути легкою у встановленні та супроводі. Для цього розробка повинна передбачати використання віртуального середовища Python (venv) та чітко задокументовану послідовність команд для розгортання застосунку. Також необхідно забезпечити можливість ініціалізації бази даних з нуля та запуску локального веб-сервера у режимі розробки.

Користувацький інтерфейс повинен бути мінімалістичним, інтуїтивно зрозумілим та забезпечувати швидкий доступ до основних функцій системи: перегляду повідомлень, створення нових та виходу з облікового запису. Програмне забезпечення повинно мати адаптивний інтерфейс, придатний для використання на різних екранах.

Крім того, важливо забезпечити безпеку збережених даних. Прямий доступ до бази повинен бути обмежений, а всі операції з повідомленнями здійснюватися через інтерфейс веб-застосунку. Доступ до адміністративних функцій, зокрема створення повідомлень, повинен мати лише автентифікований користувач.

Зазначені нефункціональні вимоги зосереджуються на продуктивності, переносимості, безпеці, простоті обслуговування та зручності користування, що забезпечує ефективне функціонування програмного забезпечення в умовах реального середовища.

2.3 Архітектура програмного забезпечення

Розроблюване програмне забезпечення для протоколювання подій в системах управління АЕС побудоване за принципами багаторівневої архітектури (Three-tier architecture), що забезпечує модульність, гнучкість та простоту масштабування. Архітектура включає три основні рівні:

1. Рівень представлення (Presentation Layer).

На цьому рівні реалізовано користувацький інтерфейс, через який оператор або інший користувач взаємодіє із системою. Інтерфейс створено з використанням шаблонізатора Jinja, який динамічно формує HTML-сторінки на основі даних, отриманих від сервера. Користувач може переглядати журнал подій, створювати нові записи, фільтрувати дані, переглядати деталі транзакцій та блоків.

2. Рівень логіки (Application / Business Logic Layer).

Цей рівень реалізований за допомогою мікрофреймворку Flask, який обробляє запити від користувача, виконує маршрутизацію, перевірку прав доступу, валідацію введених даних і взаємодію з блокчейн-модулем. Основні функції включають:

- додавання події у систему з подальшим записом у блокчейн;
- формування нового блоку;
- хешування та перевірка цілісності;
- взаємодія з базою даних SQLite;
- логування дій користувачів.

3. Рівень зберігання даних (Data Layer).

Для зберігання інформації використовується база даних SQLite, яка містить:

- таблицю користувачів;
- таблицю подій;
- таблицю блоків (з хешами, мітками часу, зв’язками між блоками).

База даних є вбудованою і зберігається локально, що спрощує розгортання та тестування, але при цьому забезпечує достатню надійність для пілотного проєкту.

Додаткові компоненти:

1. Модуль блокчейну: реалізований на Python, відповідає за створення нових блоків, хешування записів, перевірку цілісності ланцюга.

2. Система сповіщень: генерує повідомлення у разі критичних змін або подій, які порушують політики безпеки.

3. Система аудиту: фіксує всі важливі дії в системі для подальшого аналізу.

2.4 Вибір мови програмування

При розробці програмного забезпечення для протоколювання на основі блокчейн в системах менеджменту АЕС було обрано високорівневу мову програмування Python.

Як написано на офіційному сайті, Python – це інтерпретована, об'єктно-орієнтована мова програмування високого рівня з динамічною семантикою. Високорівневі вбудовані структури даних у поєднанні з динамічною типізацією та динамічним зв'язуванням роблять її дуже привабливою для швидкої розробки додатків, а також для використання в якості мови сценаріїв або мови-клею для з'єднання існуючих компонентів між собою.

Простий, легкий для вивчення синтаксис Python робить акцент на читабельності, а отже, знижує витрати на підтримку програм. Python підтримує модулі та пакети, що заохочує модульність програм та повторне використання коду. Інтерпретатор Python та велика стандартна бібліотека доступні у вигляді вихідних кодів або двійкових файлів безкоштовно для всіх основних платформ і можуть вільно розповсюджуватися.

Часто програмісти закохуються в Python через підвищену продуктивність, яку вона забезпечує. Оскільки немає етапу компіляції, цикл редагування-тестування-налагодження відбувається неймовірно швидко. Налагоджувати програми на Python легко: помилка або неправильний ввід ніколи не призведе до помилки сегментації. Натомість, коли інтерпретатор виявляє помилку, він згенерує виключення. Якщо програма не перехоплює виняток, інтерпретатор виводить трасування стеку. Налагоджувач на рівні коду дозволяє перевіряти локальні та глобальні змінні, обчислювати довільні вирази, встановлювати точки зупинки, переглядати код по одному рядку за раз тощо. Налагоджувач написаний самою мовою Python, що свідчить про інтроспективність Python. З іншого боку, часто найшвидший спосіб налагодити програму – це додати кілька операторів виводу у вихідний код: швидкий цикл редагування-тестування-налагодження робить цей простий підхід дуже ефективним [19].

Останніми роками Python переживає сплеск популярності, ставши однією з найпоширеніших мов програмування в усьому світі. Її застосування поширюється на нові та захоплюючі сфери, такі як штучний інтелект, машинне навчання та наука про дані.

Фактично, Python займає першу позицію в індексі TIOBE завдяки своєму постійному зростанню та використанню. Враховуючи її широке розповсюдження та універсальність, розуміння Python є більш важливим, ніж будь-коли.

Філософія проектування Python наголошує на читабельності та простоті коду, що дозволяє розробникам писати чіткий, логічний код для малих та великих проектів. Як мова високого рівня, Python абстрагується від значної частини складності, пов'язаної з програмуванням, що дозволяє розробникам зосередитися на вирішенні проблем, а не турбуватися про технічні деталі, що лежать в основі.

Python лежить в основі багатьох технологій і додатків, якими ми користуємося щодня. Наприклад, YouTube використовує її для обробки відео, а пошукові системи – для обробки величезних обсягів даних.

Python постійно вважається однією з найпопулярніших мов програмування у світі. Фактично, Python неодноразово займав перше місце в індексі TIOBE

Programming Community, включаючи 2023 рік, що зміцнило його позицію як мови, якій віддають перевагу розробники.

В опитуванні розробників Stack Overflow Developer Survey 2024 року Python було визнано найпоширенішою та найбажанішою мовою програмування. Цей стабільний рейтинг підкреслює зростаючий вплив Python та його широке застосування у різних галузях [19].

Python – мова загального призначення, а це означає, що її можна використовувати для створення широкого спектру додатків. Від веб-розробки до аналізу даних, від штучного інтелекту до наукових обчислень – універсальність Python не має собі рівних.

Наприклад, аналітики даних використовують Python для створення візуалізацій та маніпулювання даними, а веб-розробники – для створення динамічних веб-сайтів.

Простий і зрозумілий синтаксис Python робить її ідеальною мовою для початківців. Його команди базуються на англійській мові, а просте розташування допомагає програмістам-початківцям легко розуміти код. Ця простота також робить Python придатною для швидкої розробки та створення прототипів, скорочуючи час, необхідний для переходу від концепції до реалізації.

Відкритий характер Python призвів до розвитку величезної екосистеми бібліотек та фреймворків. Якщо вам потрібні інструменти для веб-розробки (Django, Flask), аналізу даних (pandas, NumPy), машинного навчання (TensorFlow, scikit-learn) або будь-якої іншої задачі, у Python є бібліотека для цього [19].

Python може похвалитися великою та активною спільнотою розробників, які роблять свій внесок у її постійне вдосконалення. Ця підтримка спільноти означає, що існує незліченна кількість навчальних посібників, форумів та документації, які допоможуть як новачкам, так і досвідченим розробникам.

Процвітаюча спільнота також сприяє створенню нових інструментів, бібліотек та фреймворків, що ще більше розширює можливості Python.

Широке використання Python у різних галузях робить її цінною навичкою для розробників. Компанії по всьому світу, від технологічних гігантів, таких як Google

та Facebook, до фінансових установ, таких як JP Morgan Chase, покладаються на Python у своїх технологічних рішеннях.

Така повсюдність гарантує високий попит на розробників Python, що робить її розумним вибором для кар'єри.

Python постійно розвивається, щоб відповідати потребам сучасних розробників. В останніх версіях, таких як Python 3.10 та 3.11, було впроваджено значні покращення продуктивності та нові функції, що робить мову актуальною та ефективною.

Результатом є те, що все більше людей знають Python і з більшою ймовірністю використовують її для власних проектів або пропонують її іншим [20].

Порівняння Python з іншими мовами програмування наведено у таблиці 2.1.

Таблиця 2.1 – Порівняння Python з іншими мовами програмування

Характеристика	Python	Java	JavaScript	C++
Простота синтаксису	Висока	Середня	Середня	Низька
Крива навчання	Плавна	Помірна	Плавна	Різка
Продуктивність	Середня	Висока	Середня	Дуже висока
Сфери застосування	Універсальна	Корпоративна	Веб-розробка	Системне ПЗ, ігри
Бібліотеки та фреймворки	Розгалужені	Розгалужені	Розгалужені	Розгалужені

2.5 Веб-фреймворк Flask

Для розробки було обрано Flask – популярний фреймворк для створення веб-додатків на мові Python. Завдяки своїй простоті та мінімалістичному дизайну, його обирають команди розробників, які працюють над малими та середніми проектами. BairesDev допомагає компаніям досягати успіху в розробці програмного

забезпечення на замовлення та проектах веб-розробки, використовуючи можливості сервісів Flask.

Завдяки своїй легкій та гнучкій природі Flask допомагає команді розробників BairesDev створювати веб-додатки та завершувати проекти веб-розробки на Python для різних бізнесів та типів проектів. Від веб-додатків та API до мікросервісів та візуалізації даних, Flask є чудовим інструментом для підтримки малих та середніх проектів.

Flask дозволяє створювати динамічні та інтерактивні веб-сайти, що робить його популярним вибором для розробки веб-додатків. Фреймворк поєднує в собі маршрутизацію, шаблонування та інтеграцію з базами даних для вирішення різноманітних завдань, включаючи визначення маршрутів, обробку запитів користувачів, рендеринг HTML-шаблонів та взаємодію з базами даних. Це ще більше полегшує створення багатофункціональних веб-додатків і робить Flask найкращим фреймворком для веб-розробки [20].

Фреймворк Flask є чудовим вибором для розробки RESTful API, оскільки він пропонує надійні можливості для створення API. Використовуючи функції обробки та маршрутизації запитів Flask, розробники BairesDev мають можливість визначати кінцеві точки API, одночасно легко керуючи такими методами HTTP, як POST, GET, PUT та DELETE. Розширення Flask-RESTful ще більше спрощує роботу з розробки API завдяки зручним функціям, таким як перевірка вхідних даних, серіалізація відповідей та синтаксичний аналіз запитів.

Легкий та модульний дизайн Flask робить його ідеальним для розробки мікросервісів. Flask дозволяє розбивати великі додатки на менші, незалежні сервіси, які безперешкодно взаємодіють один з одним. Розробники BairesDev, які використовують Flask, отримують можливість створювати індивідуальні мікросервіси з унікальними API та функціональністю. Це значно спрощує управління та масштабування складних додатків.

Поряд з бібліотеками візуалізації, такими як Plotly, Bokeh або Matplotlib, Flask є відмінним вибором для створення інтерактивних додатків для візуалізації даних та дашбордів. З її допомогою розробники BairesDev створюють візуальні представлення даних, генерують діаграми та надають інтерактивні користувацькі

інтерфейси для аналізу та дослідження даних. Flask також пропонує можливості шаблонування, які дозволяють розробникам створювати динамічні візуалізації та оновлювати їх на основі даних, введених користувачем [19].

2.6 Шаблонізатор Jinja

Jinja є найпопулярнішим шаблонізатором для проектів на Python і використовується в таких проектах, як Flask, Django та Ansible; останнім часом він також набув популярності для взаємодії з базами даних у поєднанні з SQL, частково завдяки використанню в таких популярних інструментах, як dbt.

Шаблонізатор – це програмне забезпечення, яке генерує динамічний контент шляхом заміни спеціальних заповнювачів у шаблоні на реальні дані під час запуску.

Використовуючи шаблон, ви можете відокремити структуру документа від його різних частин (вхідних даних): це означає, що ви можете повторно використовувати ту саму структуру, не починаючи з нуля.

Наприклад, маркетингова команда може створити шаблон рекламного листа, який можна налаштувати для різних продуктів або повідомлень залежно від аудиторії [19].

2.7 Система управління базами даних SQLite

SQLite – це система управління базами даних. Це програмне забезпечення, яке дозволяє користувачам взаємодіяти з реляційною базою даних. У SQLite база даних зберігається в одному файлі – риса, яка відрізняє його від інших систем управління базами даних. Цей факт забезпечує велику доступність: копіювання бази даних не складніше, ніж копіювання файлу, в якому зберігаються дані, а

надання доступу до бази даних може означати надсилання вкладення в електронному листі.

На жаль, переносимість підписів SQLite робить його поганим вибором, коли багато різних користувачів оновлюють таблицю одночасно (щоб зберегти цілісність даних, тільки один користувач може писати до файлу за один раз). Також може знадобитися додаткова робота для забезпечення безпеки приватних даних через ті ж самі особливості, які роблять SQLite доступним. Крім того, SQLite не пропонує таку саму функціональність, як багато інших систем баз даних, що обмежує деякі розширені можливості, які пропонують інші реляційні системи баз даних. Нарешті, SQLite не перевіряє типи даних. Там, де багато інших програм для роботи з базами даних відкидають дані, які не відповідають схемі таблиці, SQLite дозволяє користувачам зберігати дані будь-якого типу в будь-якому стовпчику [18].

SQLite створює схеми, які обмежують тип даних у кожному стовпчику, але не робить їх обов'язковими.

Навіть враховуючи недоліки, переваги доступу до бази даних та маніпулювання нею без залучення серверних додатків є величезними. SQLite використовується в усьому світі для тестування, розробки і в будь-яких інших випадках, коли є сенс розміщувати базу даних на тому ж диску, що і код програми. Розробники SQLite вважають його одним з найбільш тиражованих програмних продуктів у світі [18].

2.8 Вибір середовища розробки

Для розробки програмного забезпечення для протоколювання на основі блокчейн в системах менеджменту AEC було обрано програмне забезпечення Visual Studio Code від компанії Microsoft [18].

Visual Studio Code поєднує в собі простоту редактора вихідного коду з потужними інструментами розробника, такими як завершення та налагодження коду IntelliSense.

Visual Studio Code підтримує macOS, Linux та Windows – тож ви можете розпочати роботу з нуля, незалежно від платформи.

В основі Visual Studio Code лежить блискавичний редактор вихідного коду, який ідеально підходить для щоденного використання. Підтримуючи сотні мов, VS Code допомагає миттєво підвищити продуктивність завдяки підсвічуванню синтаксису, узгодженню дужок, автоматичним відступам, виділенню блоків, фрагментам і багато чому іншому. Інтуїтивно зрозумілі комбінації клавіш, легке налаштування та створені спільнотою відображення клавіатурних скорочень дозволять вам легко орієнтуватися у вашому коді.

Для серйозного кодування знадобляться інструменти, які розуміють код ширше, ніж просто блоки тексту. Visual Studio Code включає вбудовану підтримку завершення коду IntelliSense, розширене семантичне розуміння коду та навігацію, а також рефакторинг коду [18].

Visual Studio Code включає загальнодоступну модель розширюваності, яка дозволяє розробникам створювати і використовувати розширення, а також налаштовувати свій досвід редагування-збірки-налагодження [18].

Порівняння характеристик обраного програмного забезпечення з іншими продуктами подано у вигляді таблиці 2.2.

Таблиця 2.2 – Порівняння характеристик середовищ розробки

Характеристика	VS Code	PyCharm	Sublime Text	Atom	Eclipse
Тип ПЗ	Редактор коду (легкий IDE)	Повноцінне IDE	Легкий текстовий редактор	Текстовий редактор	Повноцінне IDE
Підтримка Python	Висока (через розширення)	Вбудована	Через плагіни	Через плагіни	Через плагіни
Інтерфейс	Сучасний, адаптивний	Функціональний, складний	Мінімалістичний	Простий, схожий на Notepad++	Класичний
Швидкодія	Висока	Від середньої до високої	Дуже висока	Середня	Середня
Розширюваність	Дуже висока (Marketplace)	Висока (плагіни JetBrains)	Висока	Висока	Висока
Вбудований термінал	Є	Є	Ні	Ні	Є
Інтеграція з Git	Вбудована	Вбудована	Через плагіни	Через плагіни	Через плагіни
Підтримка кількох мов	Так	Переважно Python	Так	Так	Так
Платформа	Windows, Linux, macOS	Windows, Linux, macOS	Windows, Linux, macOS	Windows, Linux, macOS	Windows, Linux, macOS
Ціна	Безкоштовно	Є безкоштовна та платна версії	Безкоштовно (частково платна)	Безкоштовно	Безкоштовно

3 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

У розділі подано практичну реалізацію розробленого програмного забезпечення, що забезпечує протоколювання подій у системах управління АЕС з використанням блокчейн-структури. Детально розглядаються ключові етапи програмної реалізації, включаючи використані алгоритми, інструменти та фреймворки, а також структуру основних програмних модулів. Особливу увагу приділено механізму формування та збереження хеш-ланцюга, який є основою для забезпечення цілісності та незмінності записів.

3.1 Алгоритм хешування SHA-3

SHA-3 – це скорочення від Secure Hash Algorithm 3. Він унікальним серед інших функцій SHA, оскільки він використовує підхід, який називається побудовою «губки» (від англ. «sponge»), як зображено на рисунку 3.1 [17].

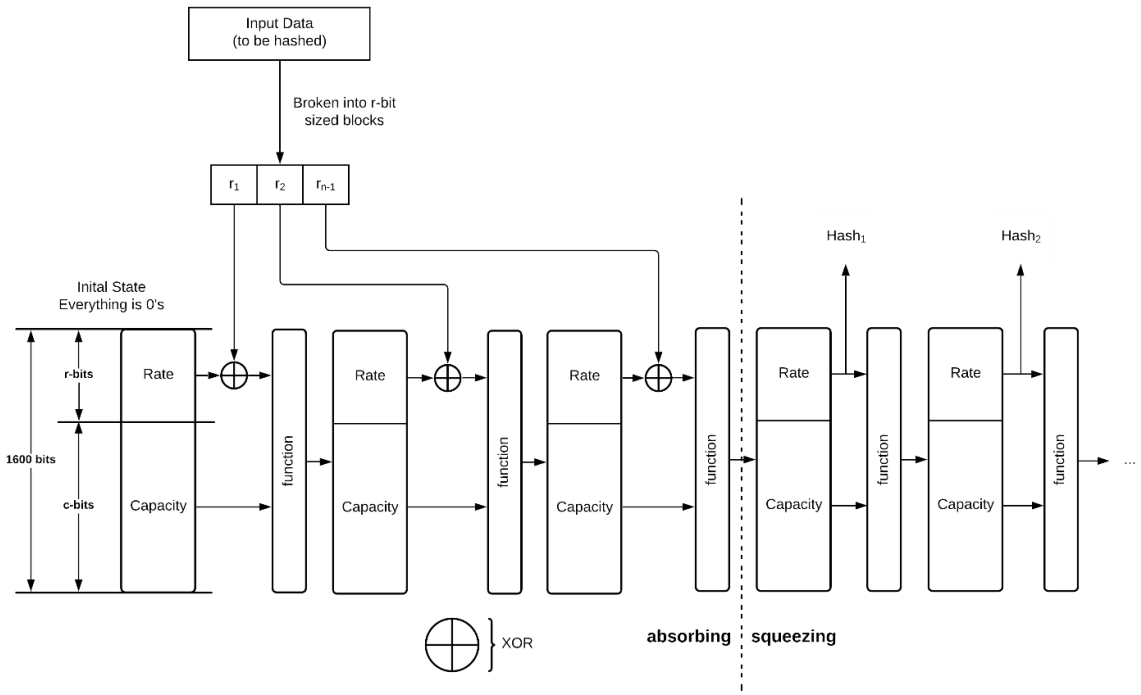

Рисунок 3.1 – Функціональна схема «губки» алгоритму SHA-3

Побудова губки SHA-3 працює за принципом:

1. Розбиття вхідних даних для хешування на частини розміром r біт, у випадку SHA-3 біти швидкості + ємності складають 1600 біт.

2. Перші вхідні значення швидкості та пропускної здатності – це нулі, які об'єднуються з першим блоком швидкості r1

3. Об'єднаний блок швидкості та пропускної здатності пропускається через функцію, як правило, з декількох раундів.

4. Перші r-біт функції – це швидкість, а решта c-біт – пропускна здатність.

5. Вищевказані r-біти об'єднуються з r2 і передаються в іншу функцію

6. Це робиться до тих пір, поки не залишиться жодного блоку даних.

7. На останньому блоці даних хеш береться з r-біт на виході функції.

8. Якщо потрібно більше бітів, то функція пропускає їх з вказаною вище швидкістю і пропускною здатністю без введення додаткових даних.

Оскільки дані рідко ідеально діляться на рівну кількість блоків, SHA-3 повинен розбивати вхідні дані, щоб вони ідеально ділилися на r біт. Це доповнення виконується шляхом додавання 1 біта, потім заповнення нулями або більшою кількістю нулів і завершенням 1 бітом. Нижче продемонстровано за допомогою рисунка два випадки доповнення вхідних даних (як зображено на рисунку 3.2) [17].

$$r_{bits} = 12$$

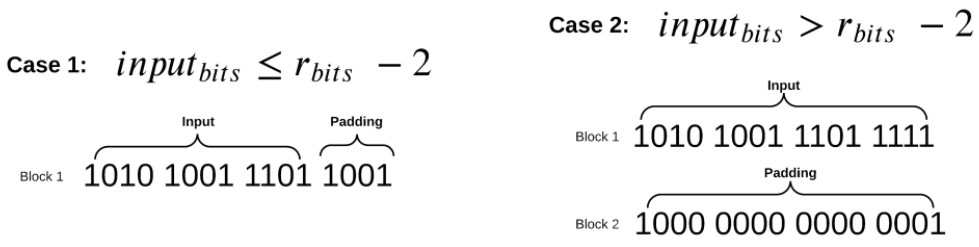

Рисунок 3.2 – Два випадки доповнення вхідних даних

У випадку, коли вхідні дані точно діляться на r біт, ми використовуємо випадок 2, як показано вище. Це робиться для того, щоб переконатися, що

повідомлення довжиною, яка ділиться на r біт і закінчується чимось, що виглядає як пробіл, не дасть той самий хеш, що і повідомлення з видаленими бітами.

Функція Keccak, як зображено на рисунку 3.3, – це серце SHA-3, ця функція є функціональним блоком на наведеній вище схемі, вона використовує операції XOR, AND і NOT [17].

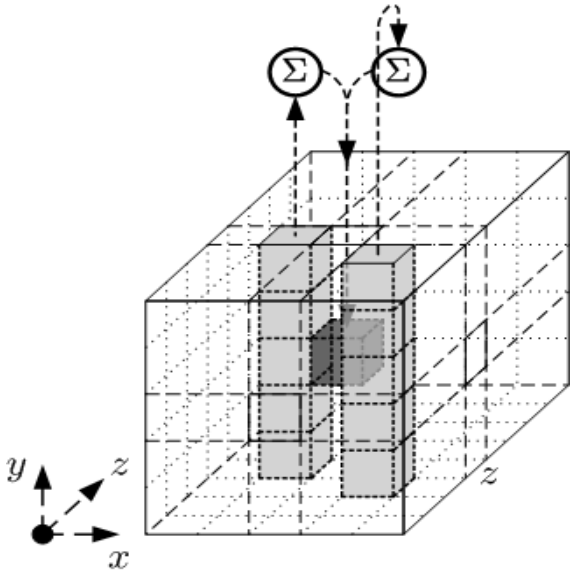

Рисунок 3.3 – Схема функції Keccak

ℓ – член сімейства цілих чисел, має бути цілим числом і ℓ більше або дорівнює 0, але менше або дорівнює 6 (як зображено на рисунку 3.4).

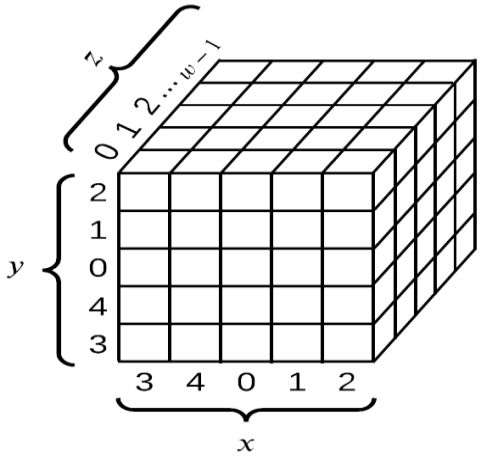

Рисунок 3.4 – Масив чисел

Всі перетворення функції Keccak виконуються над наведеним вище 3-вимірним масивом, нумерація індексів якого визначена стандартом FIPS 202. Цей тривимірний об'єкт являє собою масив 5 на 5 на w (w у рівнянні вище). Весь тривимірний масив називається станом, а різні інші частини масиву станів позначаються нижче (як зображено на рисунку 3.5) [17].

Основні етапи роботи алгоритму SHA-3 з використанням Keccak – це поглинання і витягування. Під час етапу поглинання вхідні дані «поглинаються» в структуру стану (state), яка є серією бітів фіксованої довжини (1600 біт для SHA-3). Після цього на етапі витягування з цього стану витягуються вихідні біти, які становлять хеш. У разі необхідності можна обробляти дані будь-якої довжини, а хеші можна отримувати різних розмірів, залежно від вибраної версії SHA-3 (наприклад, SHA3-224, SHA3-256, SHA3-384 і SHA3-512).

Це дозволяє SHA-3 мати значну гнучкість та ефективність, адже можна використовувати різні варіанти для різних криптографічних завдань, зберігаючи високу безпеку.

Основні версії SHA-3

SHA-3 має кілька варіантів, які відрізняються довжиною хешу:

1. SHA3-224: Генерує хеш довжиною 224 біти.

2. SHA3-256: Генерує хеш довжиною 256 біт.

3. SHA3-384: Генерує хеш довжиною 384 біти.

4. SHA3-512: Генерує хеш довжиною 512 біт [17].

Крім того, існує ще SHAKE (SHA3 XOF) – розширені вихідні функції, які дозволяють отримувати хеші змінної довжини. Це особливо корисно для криптографічних застосувань, де потрібно генерувати хеши більшої або меншої довжини, залежно від вимог до безпеки та продуктивності.

Переваги SHA-3 на основі Keccak:

1. Покращена безпека: Keccak використовує конструкцію губки, яка забезпечує високу стійкість до різноманітних криптографічних атак, зокрема до атак на колізії. Алгоритм SHA-3 значно складніший для підробки порівняно з попередніми алгоритмами SHA-1 і SHA-2, які зазнали атак з боку дослідників криптографії.

2. Гнучкість: завдяки конструкції губки SHA-3 може працювати з вхідними даними будь-якої довжини і генерувати хеші змінної довжини. Це дає можливість адаптувати алгоритм до специфічних потреб, наприклад, у випадках, коли потрібен більш короткий або довгий хеш для конкретної програми або криптографічного застосування.

Також потрібно зауважити, що частини масиву станів впорядковані за розмірністю, як зображено на рисунку 3.5.

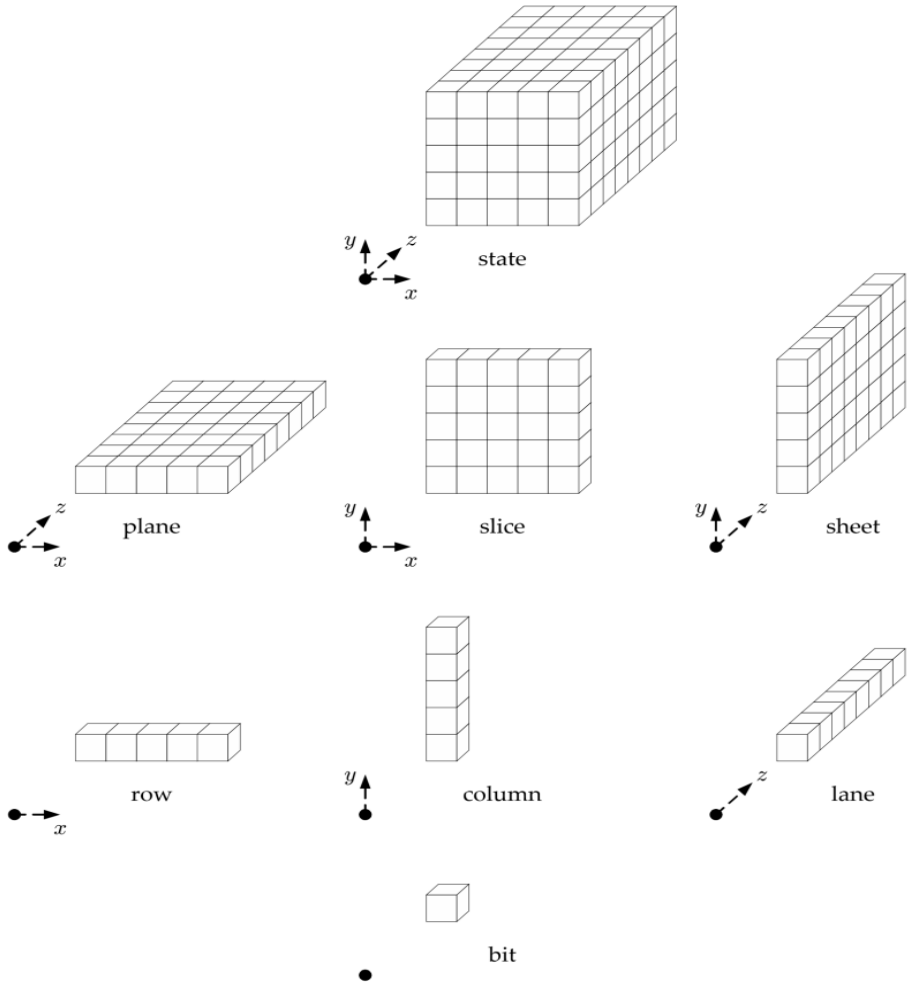

Рисунок 3.5 – Частини масиву станів, впорядковані за розмірністю

Однією з основних характеристик SHA-3 є висока стійкість до різних типів атак. Зокрема, вона забезпечує більший рівень безпеки порівняно з SHA-2, зменшуючи ймовірність успішного здійснення атак на криптографічні функції, що базуються на хешах. SHA-3 використовує більш складні математичні операції та

стратегії, що дозволяють захищати хеш-функцію від потенційних вразливостей, виявлених у попередніх версіях SHA.

Алгоритм SHA-3 включає різні варіанти, що забезпечують різну довжину хешу. Наприклад, SHA3-224, SHA3-256, SHA3-384, SHA3-512 – це версії SHA-3, які генерують хеші різної довжини, залежно від конкретних вимог до безпеки та ефективності. Чим довший хеш, тим складніше здійснити атаку на нього, і це є важливим аспектом при виборі конкретного варіанту SHA-3 для різних застосувань.

Крім того, SHA-3 включає в себе ще одну цікаву особливість – алгоритм SHAKE (Secure Hash Algorithm Keccak), який дозволяє отримувати хеші довільної довжини, що робить цей алгоритм дуже гнучким. SHAKE може бути використаний там, де потрібно отримати хеші певної довжини, наприклад, для криптографічних протоколів, які вимагають високої динамічності у розмірі хешів [17].

У контексті алгоритму SHA-3, який використовує конструкцію губки (Keccak), процес трансформації блоків можна поділити на два основних етапи: поглинання (absorbing) і витягування (squeezing). Ці етапи забезпечують обробку вхідних даних та отримання результату в вигляді хешу [17].

Під час етапу поглинання вхідні дані поділяються на блоки фіксованої довжини. Для алгоритму SHA-3, це 1088 біт на кожен блок, який буде оброблятися. Кожен блок вхідних даних по черзі «поглинається» в внутрішню структуру стану, яка є 1600 бітами даних.

Алгоритм використовує функцію трансформації для кожного блоку даних, комбінуючи його з поточним станом (якщо такий є). Це дозволяє поступово модифікувати стан у відповідності до вхідних даних.

Математичні операції, що виконуються під час цього етапу, включають логічні операції (наприклад, XOR), циклічні перестановки та інші перетворення, що дають змогу «розмішати» дані, що забезпечує більшу стійкість до атак.

Порівняння кількох популярних алгоритмів хешування з обраним алгоритмом хешування SHA-3 показано у вигляді таблиці 3.1 [17].

Таблиця 3.1 – Порівняння алгоритмів хешування

Алгоритм	Розмір вихідного хешу	Швидкість	Безпека	Застосування	Особливості
SHA-3-224	224 біт	Середня	Висока, стійкий до колізій	Програми з обмеженнями по пам'яті	Меньший розмір хешу, оптимальний для вбудованих систем
SHA-3-256	256 біт	Середня	Висока, стійкий до колізій	Широко використовується для криптографії та захисту даних	Баланс швидкості та безпеки
SHA-3-384	384 біт	Повільна	Висока, стійкий до колізій	Високонадійні додатки, захист даних	Більший розмір хешу для підвищеної безпеки

Кінець таблиці 3.1

SHA-3-512	512 біт	Повільна	Висока, стійкий до колізій	Системи з високими вимогами до безпеки	Найвищий рівень безпеки для великих систем

Підсумовуючи, основною перевагою SHA-3 є висока стійкість до колізій, що робить його ідеальним для використання в криптографії, де важливо, щоб два різні набори даних не могли мати однаковий хеш. Крім того, SHA-3 пропонує гнучкість, оскільки доступні різні варіанти хешування з різними розмірами вихідного хешу, що дозволяє вибрати найбільш оптимальний варіант для конкретних потреб.

Алгоритм забезпечує високу безпеку навіть у майбутньому, оскільки його конструкція не залежить від виявлених вразливостей у попередніх алгоритмах. SHA-3 також добре інтегрується в існуючі системи та підтримується на широкому спектрі криптографічних платформ. Це робить його доступним для розробників і простим у використанні.

Завдяки своїй ефективності і надійності SHA-3 стає ідеальним вибором для

забезпечення безпеки в сучасних криптографічних протоколах і вбудованих системах [17].

3.2 Розробка інтерфейсу користувача

Інтерфейс користувача веб-застосунку реалізовано з використанням шаблонізатора Jinja2, який є частиною фреймворку Flask. Основна мета інтерфейсу – забезпечити зручну та інтуїтивно зрозумілу взаємодію з системою для перегляду, створення, оновлення та видалення подій, які записуються у вигляді блоків у ланцюжку з хеш-ідентифікаторами.

Основні сторінки інтерфейсу:

1. Головна сторінка («index») – відображає список усіх створених подій у зворотному хронологічному порядку. Для кожної події виводяться назва, час створення, автор, а також текст повідомлення, який містить хеш попереднього блоку.

2. Сторінка створення події («create») – форма, яка дозволяє авторизованому користувачу створити нову подію. Під час створення система автоматично обчислює SHA3-хеш поточної події на основі назви, тексту повідомлення та хешу попередньої події. Отриманий хеш додається до бази даних як частина ланцюжка.

3. Сторінка редагування події («update») – забезпечує можливість редагування вже створеної події. Редагування дозволено лише автору події. З міркувань безпеки та незмінності blockchain-записів, редагування хеш-ланцюжка не передбачено – оновлюється лише текст повідомлення.

4. Кнопка видалення події («delete») – дозволяє автору події видалити її з бази даних. При цьому, варто зазначити, що видалення блоку в реальних blockchain-системах є небажаним, однак у цьому навчальному прикладі така функціональність реалізована для гнучкості системи.

5. Кнопка перевірки цілісності («validate») – окрема функція, що імітує перевірку blockchain-ланцюжка. Для кожної події повторно обчислюється хеш і

порівнюється з наявним у базі. У даній реалізації з міркувань простоти валідація завжди повертає позитивний результат, однак структура закладена для подальшого розширення.

Для обчислення хешів використано алгоритм SHA3-256 із бібліотеки «hashlib». Повідомлення містить службовий роздільник «SHA3: хеш, який вказує на хеш попередньої події. Всі форми захищені перевіркою авторизації через декоратор «@login_required», що запобігає несанкціонованим змінам. Інтерфейс реалізовано відповідно до принципів чистої архітектури: шаблони HTML розділено від бізнес-логіки, що полегшує підтримку й розширення. Таким чином, розроблений користувацький інтерфейс поєднує простоту взаємодії з користувачем та елементи захисту даних через механізми хешування і контролю доступу.

Шаблон index.html є частиною веб-інтерфейсу застосунку для журналювання подій із використанням технології блокчейн. Він відповідає за відображення сторінки журналу подій, де користувач може переглядати наявні записи, а також взаємодіяти з ними. Шаблон наслідує загальний вигляд з базового шаблону base.html, що забезпечує єдину структуру всього інтерфейсу. У верхній частині сторінки відображається заголовок "Журнал подій", а для авторизованих користувачів доступні дві основні дії: створення нової події через посилання на відповідну форму та перевірка цілісності блокчейну за допомогою кнопки, яка надсилає POST-запит на відповідний маршрут. Основна частина шаблону містить список усіх подій, що зберігаються у змінній posts.

Шаблон update.html відповідає за відображення сторінки редагування окремої події в системі. Він наслідує базовий шаблон base.html для збереження загального стилю сайту. У заголовку сторінки виводиться текст "Редагувати" з назвою конкретної події, яку користувач хоче змінити. Основний вміст складається з HTML-форми, що дозволяє змінити назву та опис події – поля форми автоматично заповнюються поточними значеннями з бази даних або з форми у випадку повторного відправлення через помилки. Нижче розміщена ще одна форма з кнопкою для видалення події, яка надсилає POST-запит на відповідний маршрут. Перед видаленням з'являється вікно підтвердження дії. Цей шаблон реалізує функціональність редагування та видалення записів, забезпечуючи простий і

зрозумілий інтерфейс для керування даними в журналі подій.

Шаблон base.html основним шаблоном для всіх сторінок вебзастосунку і відповідає за структуру інтерфейсу користувача. Він задає базову HTML-розмітку, включаючи <head> із заголовком сторінки, який динамічно змінюється за допомогою блоку title, та підключенням CSS-файлу для стилізації інтерфейсу. У <body> розміщене головне меню навігації, яке містить посилання на головну сторінку, а також, залежно від того, чи користувач автентифікований, показує відповідні опції – вітання з ім’ям та кнопку виходу для авторизованих або посилання на реєстрацію й вхід для гостей. Основний вміст сайту розташований у блоці section.content, де динамічно вставляється заголовок сторінки (header) та контент сторінки (content).

Шаблон create.html відповідає за інтерфейс створення нової події у вебзастосунку. Він розширює базовий шаблон base.html, що забезпечує загальну структуру сторінки, включаючи заголовок, навігацію та стилізацію. У блоці header виводиться заголовок сторінки з текстом "Створити нову подію", який динамічно підставляється в заголовок документа. У блоці content розміщується HTML-форма з методом POST, яка дозволяє користувачеві ввести назву та опис події. Обидва поля – текстове поле для назви (input name="title") і текстове поле для опису (textarea name="body") – є обов’язковими для заповнення (атрибут required). Після заповнення форми користувач може натиснути кнопку "Створити подію", яка надсилає дані на сервер для подальшої обробки та збереження нової події у блокчейн-журналі. Цей шаблон є частиною функціоналу додавання нових записів у систему протоколювання.

Шаблон login.html відповідає за інтерфейс сторінки входу користувача до системи. Він наслідує базовий шаблон base.html, у якому розміщується заголовок сторінки з написом "Вхід до системи". Основний вміст містить HTML-форму з двома полями введення – для імені користувача (username) та пароля (password), обидва є обов’язковими для заповнення. Після введення даних користувач може натиснути кнопку «Увійти», що відправляє дані форми методом POST на сервер для автентифікації. Цей шаблон є невіддільною частиною механізму авторизації у веб-додатку.

Шаблон register.html реалізує інтерфейс сторінки реєстрації користувача в системі. Він наслідує базовий шаблон base.html і виводить заголовок сторінки з текстом "Реєстрація користувача". Основна частина шаблону містить HTML-форму, яка надсилається методом POST і включає два обов'язкові поля: для введення імені користувача (username) та пароля (password). Після заповнення форми користувач може натиснути кнопку «Зареєструватися», щоб передати дані на сервер для створення нового облікового запису. Цей шаблон є частиною системи аутентифікації веб-додатку.

CSS-файл style.css визначає стилізацію веб-інтерфейсу системи журналювання для АЕС. Він задає загальний вигляд сторінки з м'яким блакитним фоном, білим контейнером контенту з тінню та закругленими краями. Шрифт основного тексту – Roboto, заголовків – Poppins. Стилізовано заголовки, посилання, горизонтальні лінії, сповіщення (успіх/помилка), блоки постів, форму для вводу даних та навігаційне меню.

3.3 Розробка функціональних модулів програмного забезпечення

Основні компоненти серверної частини веб-застосунку реалізовано мовою програмування Python із використанням мікрофреймворку Flask. Основна мета цього етапу – реалізувати функціональні можливості, що забезпечують додавання, збереження, виведення, редагування, видалення та перевірку цілісності подій, які протоколюються у блокчейн-журналі.

Для розділення відповідальності застосовано модульний підхід: функціональність поділена на окремі файли, кожен з яких виконує певну роль у логіці застосунку. Основними модулями є: модуль аутентифікації користувачів, модуль керування подіями, модуль створення та перевірки блокчейну, а також головний файл запуску застосунку. Кожен з цих модулів містить набір маршрутів (routes), що обробляють HTTP-запити, виконують відповідні дії та повертають

HTML-шаблони для відображення у браузері.

Функціональність взаємодіє з базою даних SQLite через SQL-запити та забезпечує безпечну роботу з обліковими даними, записами подій і ланцюжком блоків. Додатково реалізовано механізми перевірки цілісності блокчейну, що є ключовим елементом надійності системи.

Файл __init__.py є головним ініціалізаційним модулем веб-застосунку, створеного з використанням фреймворку Flask. У ньому визначено фабричну функцію create_app(), яка відповідає за створення і налаштування екземпляра застосунку. Це дозволяє зручно конфігурувати програму як для робочого, так і для тестового середовища.

На початку виконується базове налаштування, зокрема задається секретний ключ безпеки та шлях до файлу бази даних SQLite, який зберігається в окремій ізольованій папці instance. Якщо програма не перебуває в режимі тестування, додаткові параметри конфігурації зчитуються з файлу config.py. В іншому випадку використовується передана тестова конфігурація.

Застосунок перевіряє наявність папки instance і створює її у разі відсутності. Також для перевірки роботи сервера додається простий маршрут /hello, який повертає повідомлення "Hello, World!".

До застосунку підключаються функціональні модулі (Blueprints), які реалізують окремі частини логіки – зокрема, модуль auth для автентифікації користувачів та blog, який у цьому випадку слугує основним модулем для взаємодії з подіями. Крім того, підключається модуль бази даних db, який ініціалізується разом із застосунком.

У фіналі виконується реєстрація маршруту кореневої сторінки /, який спрямовується на функцію index з модуля blog. Таким чином, __init__.py виконує роль центрального місця, де відбувається підключення та узгодження всіх компонентів програмного забезпечення.

Файл auth.py реалізує функціональність автентифікації користувачів у межах веб-застосунку. Він створює окремий модуль за допомогою механізму Flask Blueprint, що дозволяє групувати маршрути, пов’язані з реєстрацією, входом, виходом та перевіркою авторизації користувача. На початку файлу імпортуються

необхідні бібліотеки, включно з компонентами Flask та інструментами для хешування паролів.

У коді визначено декоратор login_required, який забезпечує обмеження доступу до окремих сторінок для неавторизованих користувачів, автоматично перенаправляючи їх на сторінку входу. Також реалізовано функцію, яка перед кожним запитом перевіряє, чи є у сесії збережений ідентифікатор користувача, і, якщо такий є, завантажує його з бази даних.

Таким чином, файл auth.py є ключовим компонентом, що забезпечує базову автентифікацію та управління сесією в межах програмного забезпечення.

Файл blockchain.py реалізує механізм блокчейну для збереження та перевірки цілісності даних у системі. Він містить логіку для створення, оновлення та перевірки подій (блоків), що додаються до бази даних. Блоки зв'язуються між собою через хеші, що гарантує незмінність і захищеність інформації.

У файлі спочатку імпортуються необхідні модулі, такі як компоненти Flask для роботи з веб-запитами та базою даних, а також бібліотека для хешування hashlib, яка використовується для створення хешів постів. Використовується також модуль datetime для обробки часу створення подій.

При створенні нової події формується хеш попереднього поста, який додається до тіла нового повідомлення. Це забезпечує ланцюгову взаємозалежність блоків у системі блокчейну. Якщо попередній пост є, то його заголовок та тіло використовуються для обчислення хешу, який додається до нового посту, забезпечуючи таким чином інтеграцію всіх подій у єдиний ланцюг.

Файл db.py реалізує взаємодію з базою даних у додатку на Flask, використовуючи SQLite. Він забезпечує функції для підключення, закриття та ініціалізації бази даних, а також для реєстрації команд для управління базою через CLI.

Функція get_db() створює підключення до бази даних, яке є унікальним для кожного запиту, і використовується для подальших операцій. Якщо підключення вже існує, воно просто повертається. База даних налаштовується так, щоб результат запитів можна було отримати як словники (sqlite.Row), що полегшує доступ до полів за іменами колонок.

Функція close_db() відповідає за закриття підключення до бази даних після завершення обробки запиту, що дозволяє запобігти витоку ресурсів.

Функція init_db() очищає наявні дані та створює нові таблиці, використовуючи SQL-скрипт, що зберігається у файлі schema.sql. Цей скрипт повинен містити SQL-вирази для створення необхідних таблиць у базі даних.

За допомогою декоратора @click.command("init-db") створено команду CLI, яка ініціалізує базу даних. Цю команду можна викликати з командного рядка при запуску додатку, щоб очистити і налаштувати базу даних.

Функція sqlite3.register_converter("timestamp", lambda v: datetime.fromisoformat(v.decode())) реєструє перетворення типу даних для SQLite, яке дозволяє автоматично перетворювати значення типу timestamp з бази даних у об'єкти типу datetime.

Нарешті, функція init_app(app) реєструє функції, пов'язані з базою даних, у додатку Flask. Вона додає команду init-db в CLI і налаштовує закриття підключень до бази даних після виконання кожного запиту через Flask.

Файл validate_blockchain.py використовується для перевірки цілісності ланцюга повідомлень, який зберігається на веб-сторінці. Він реалізує валідацію за допомогою технології блокчейн, де кожне повідомлення має свій хеш, який порівнюється з задекларованим значенням хешу, щоб підтвердити цілісність даних.

Скрипт використовує бібліотеки urllib.request, argparse та hashlib для завантаження веб-сторінки, парсингу повідомлень і обчислення їх хешів.

Вихідні коди функціональних модулів застосунку подано в Додатку А.

3.4 Розробка база даних і логіки зберігання та обробки даних

База даних і логіка зберігання та обробки даних засновуються на тому, що SQL-скрипт виконує декілька операцій для ініціалізації бази даних, яка включає створення двох таблиць: user (користувачі) та post (пости), а також вставку початкових даних.

Спочатку, перед тим як створювати таблиці, скрипт перевіряє наявність існуючих таблиць user та post у базі даних і, якщо вони є, видаляє їх за допомогою команд DROP TABLE IF EXISTS. Це необхідно для того, щоб забезпечити чисте середовище для створення нових таблиць і уникнути помилок, якщо такі таблиці вже існують.

Після цього скрипт створює таблицю для користувачів (user). У таблиці є три основних стовпці: id, username та password. Стовпець id є унікальним і автоматично збільшується при кожному новому записі. Стовпець username є обов'язковим і повинен бути унікальним, що гарантує, що кожен користувач має своє унікальне ім'я. Стовпець password також є обов'язковим і зберігає пароль користувача. Пароль зазвичай зберігається в зашифрованому вигляді для підвищення безпеки.

Наступним кроком є створення таблиці для постів (post). Вона містить кілька важливих стовпців: id, title, body, hash_value, created та author_id. Як і в таблиці користувачів, стовпець id є унікальним і автоматично збільшується. title – це заголовок посту, а body – його текст. Стовпець hash_value зберігає хеш значення посту, що використовується для перевірки цілісності даних і забезпечує можливість виявлення будь-яких змін у пості. Стовпець created зберігає час створення посту, і за замовчуванням він встановлюється на поточний момент. Стовпець author_id є зовнішнім ключем, що посилається на таблицю user і визначає автора посту.

Завершується скрипт вставкою початкового посту в таблицю post. Це повідомлення має заголовок "INITIAL MESSAGE" і тіло "Це автоматично згенероване перше повідомлення". Хеш цього повідомлення вказаний у стовпці hash_value. Дата створення встановлюється автоматично на момент вставки посту, а автором є користувач, який був створений на попередньому етапі (за допомогою підзапиту, що отримує його ідентифікатор за ім'ям "AUTOMATICALLY GENERATED").

4 ТЕСТУВАННЯ Й РОБОТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розділ присвячено оцінці працездатності розробленого програмного забезпечення. Подано опис процесу запуску веб-застосунку, а також результати функціонального тестування, які демонструють правильність реалізації основних модулів і відповідність заданим вимогам. Особливу увагу приділено перевірці цілісності блокчейн-ланцюга при додаванні нових подій до системи, а також зручності взаємодії користувача з інтерфейсом.

4.1 Запуск програмного забезпечення

Для запуску розробленого програмного забезпечення спочатку потрібно створити віртуальне середовище Python за допомогою команди py -3 -m venv .venv. Це дозволяє ізолювати бібліотеки, необхідні для роботи програми, від інших програм, що використовують Python. Після цього потрібно активувати віртуальне середовище, використовуючи команду .venv\Scripts\activate.bat, що дозволяє почати використовувати специфічні для цього середовища бібліотеки.

Далі потрібно перейти до каталогу проекту командою cd flaskr, щоб працювати з файлами, що знаходяться у кореневій директорії проекту. Потім слід встановити всі необхідні бібліотеки, використовуючи команду pip install -e .. Це забезпечить встановлення програми та її залежностей, необхідних для правильної роботи.

Наступним кроком є ініціалізація бази даних командою flask --app flaskr init-db, що створює необхідні таблиці в базі даних та забезпечує початкову настройку для подальшої роботи додатку.

Останній етап – запуск самого веб-додатка в режимі відлагодження командою flask --app flaskr run --debug. Ця команда запускає веб-сервер, який дозволяє працювати з програмою через браузер та дає можливість відслідковувати помилки

в реальному часі, що спрощує процес розробки та тестування.

Описані кроки запуску програмного забезпечення продемонстровано на рисунку 4.1.

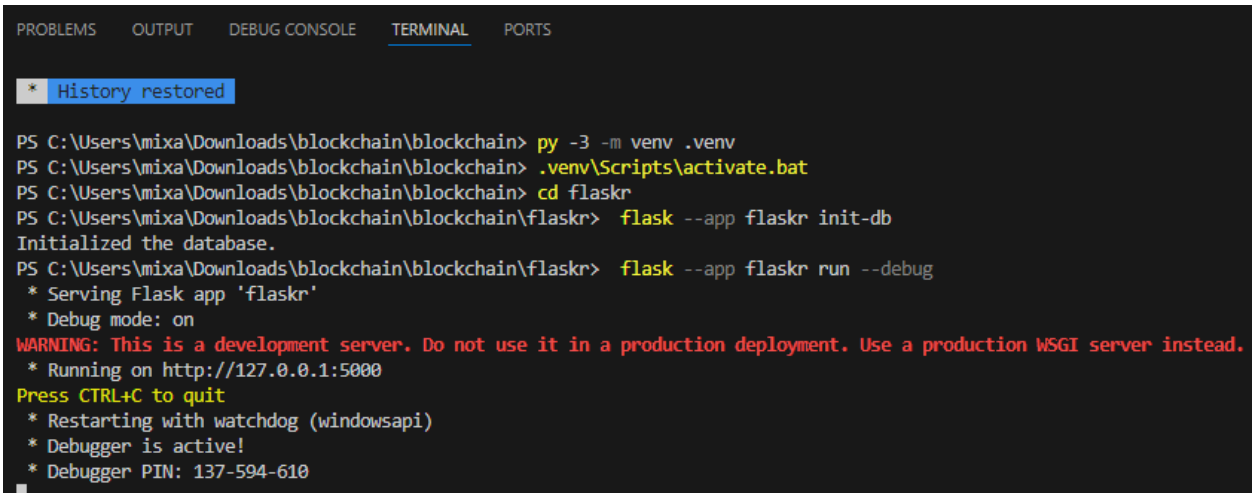

Рисунок 4.1 – Команди запуску програмного забезпечення

Таким чином, було описано найголовніші елементи коду розроблюваного програмного забезпечення для журналювання подій на АЕС.

4.2 Робота програмного забезпечення

Розроблене програмне забезпечення є ефективним інструментом для забезпечення зручного та безпечного керування даними за допомогою технології блокчейн. Система створена з використанням веб-фреймворка Flask, що дозволяє забезпечити високу швидкість обробки запитів та простоту розгортання. Вона включає механізм зберігання даних у базі даних SQLite, що гарантує стабільність та надійність зберігання інформації.

Основні функціональні можливості програмного забезпечення включають:

1. Реєстрацію користувачів, яка дозволяє створювати нові облікові записи та зберігати паролі з використанням безпечних алгоритмів хешування.

2. Створення постів, кожен з яких має свій унікальний хеш, що забезпечує

цілісність даних та їх захист від несанкціонованих змін.

3. Перевірку цілісності ланцюга повідомлень через алгоритм SHA3, що гарантує правильність і неможливість підробки даних.

Користувач може зручно переглядати повідомлення, перевіряти їх на валідність, а також здійснювати взаємодію з іншими користувачами через інтерфейс. Розробка включає функціонал для додавання нових повідомлень, автоматичну генерацію перших записів в системі для перевірки її працездатності, а також можливість перевірки хешів повідомлень, що додаються, на відповідність.

Після налаштування та запуску програма продемонструє свою здатність до швидкої обробки запитів і забезпечення високої надійності роботи з даними.

Роботу програмного забезпечення показано на рисунках 4.2-4.9.

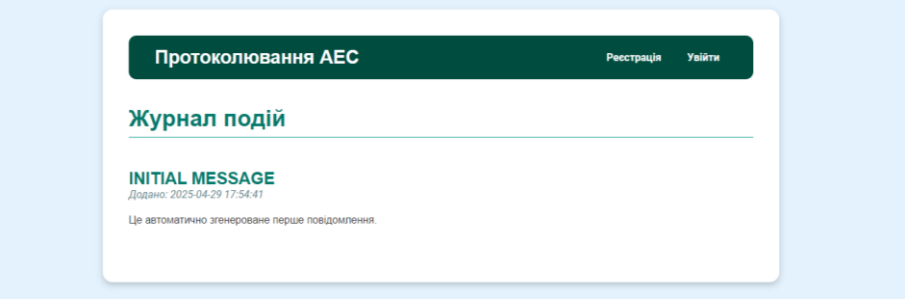

Рисунок 4.2 – Головна сторінка розробленого програмного забезпечення

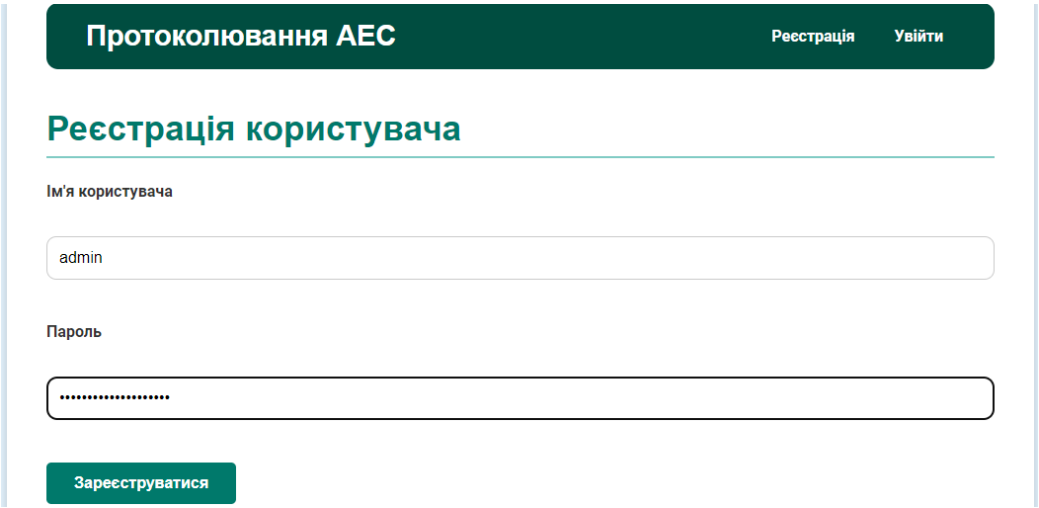

Рисунок 4.3 – Форма реєстрації користувача

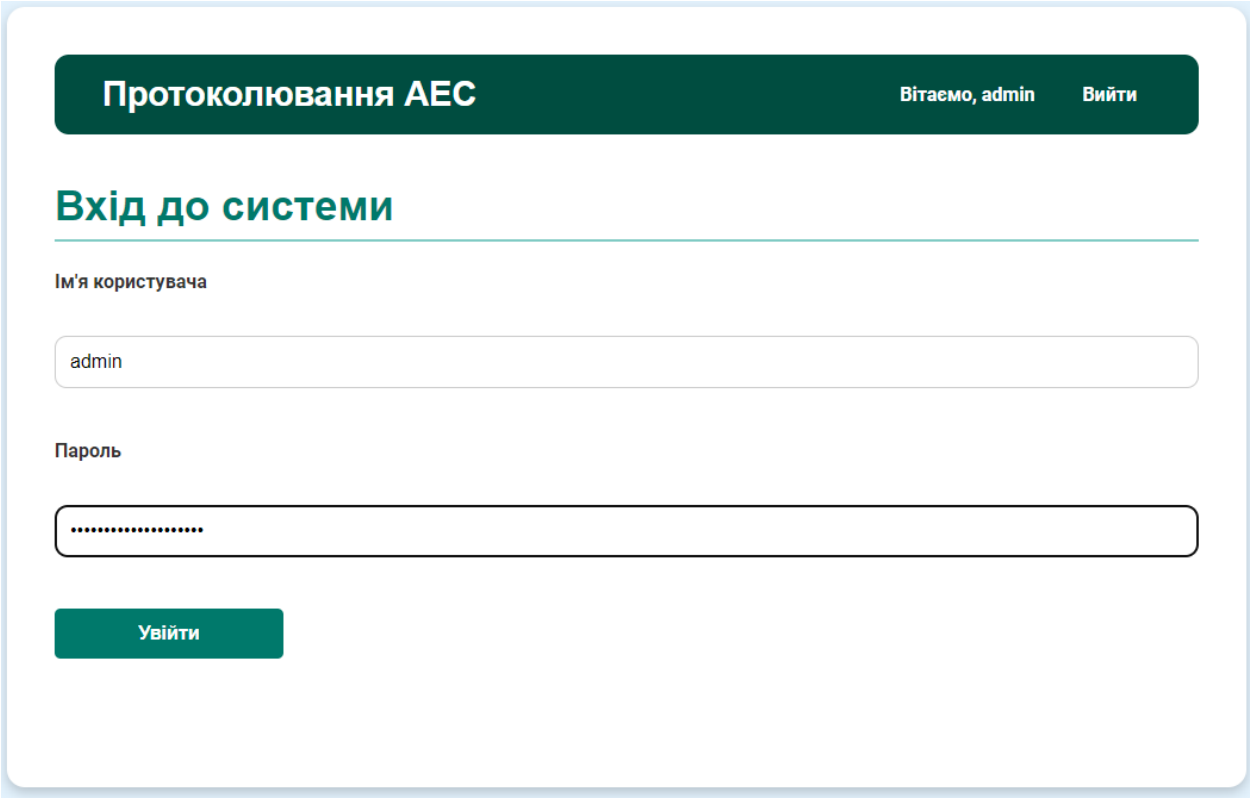

Рисунок 4.4 – Форма входу в обліковий запис користувача

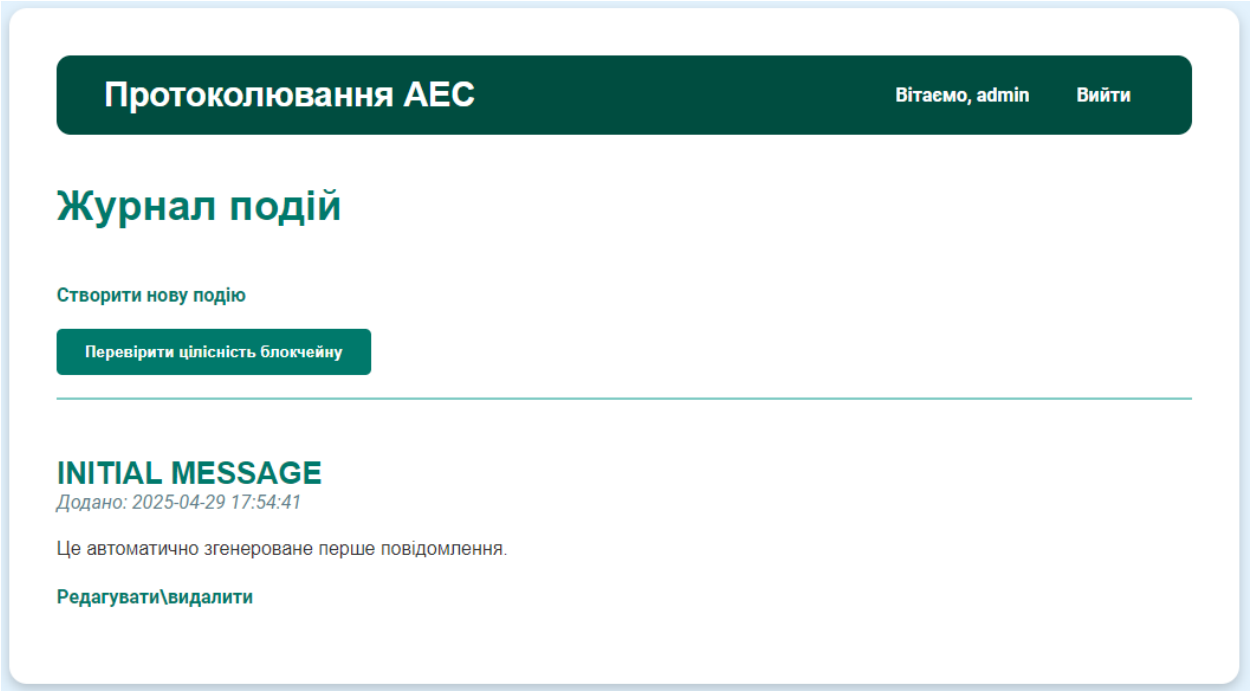

Рисунок 4.5 – Журнал подій розробленого програмного забезпечення

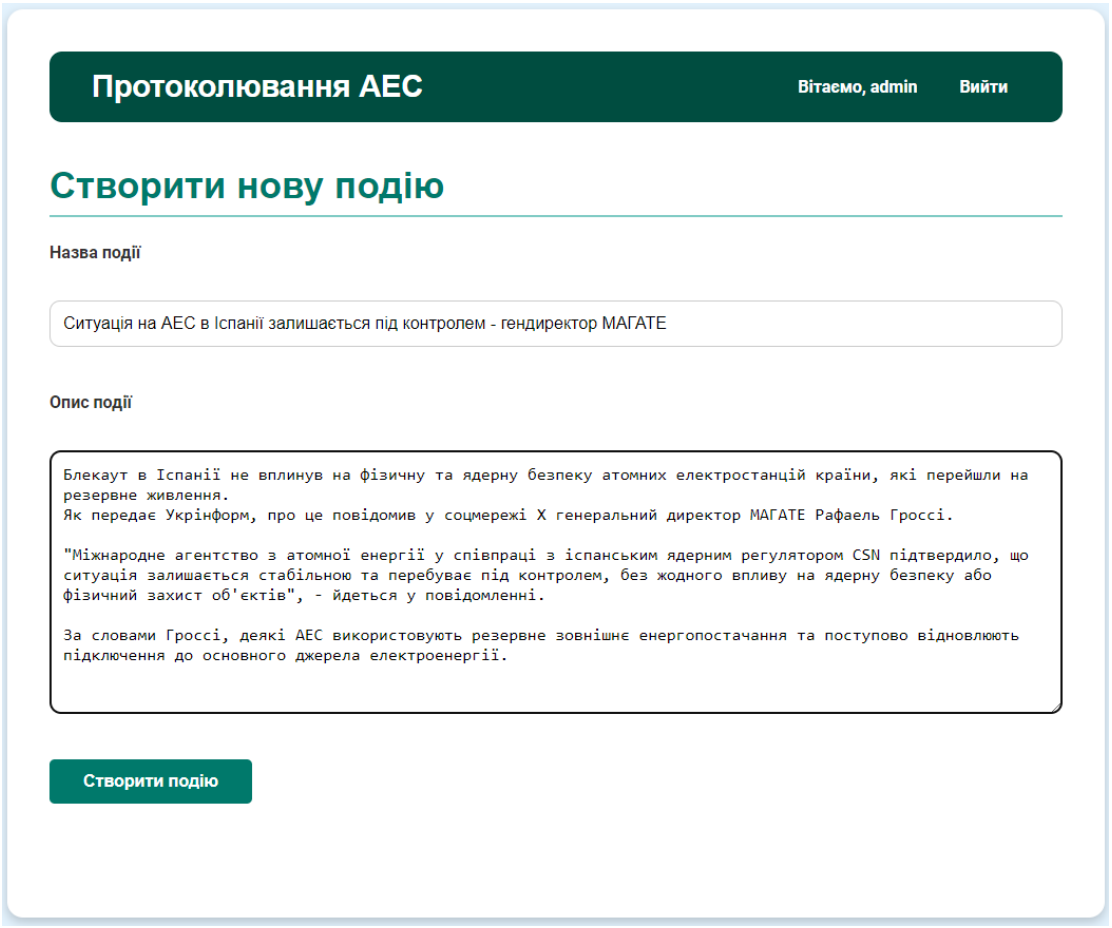

Рисунок 4.6 – Форма створення події

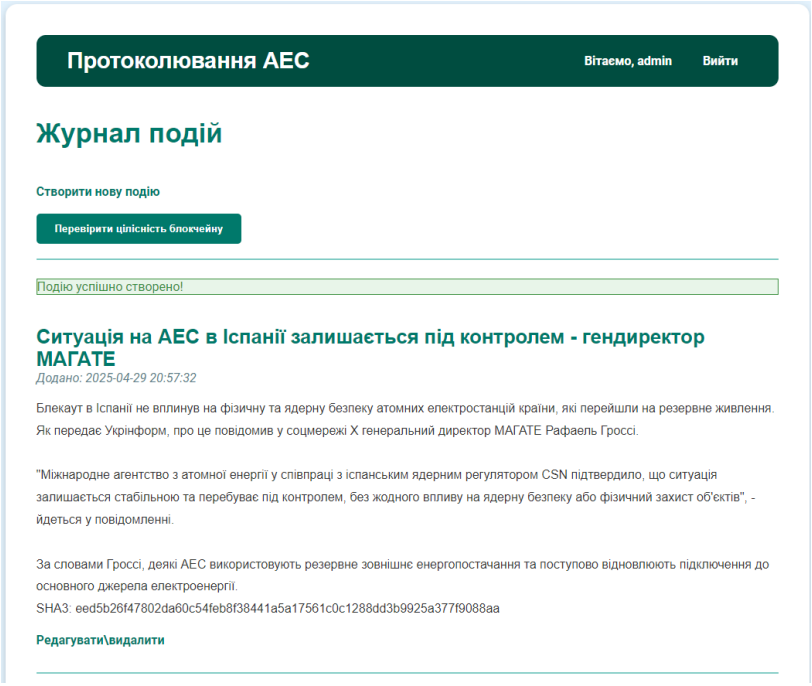

Рисунок 4.7 – Сповіщення про успішне створення події

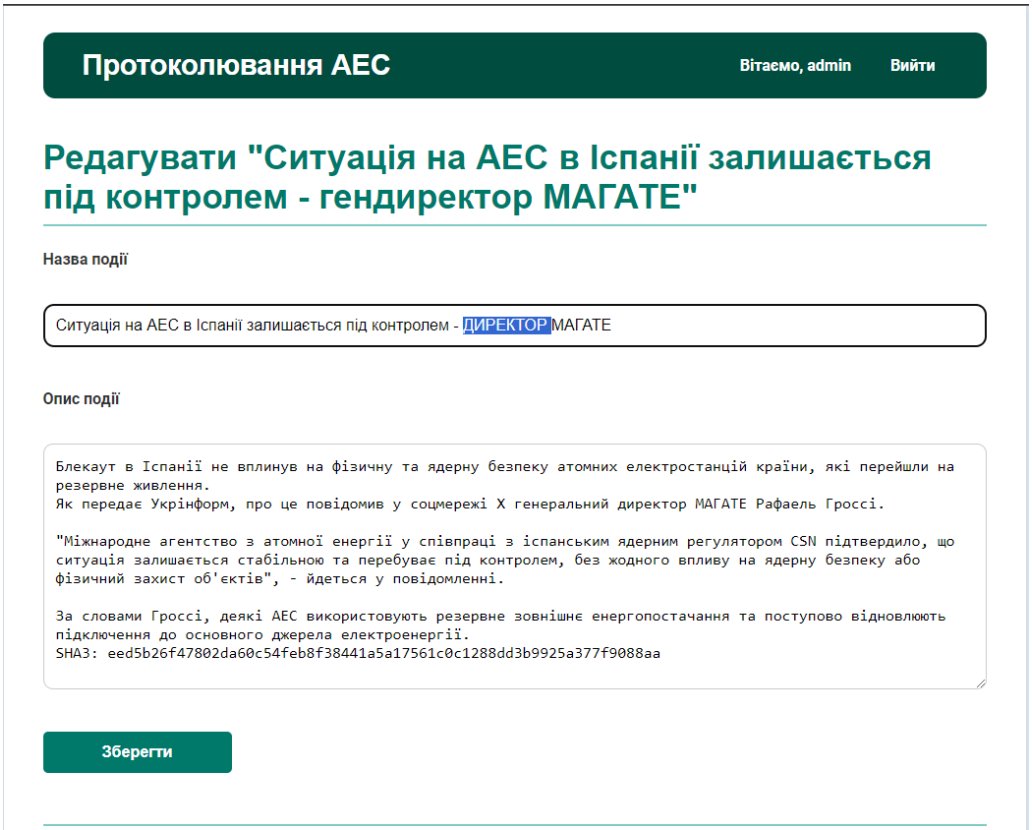

Рисунок 4.8 – Форма зміни події

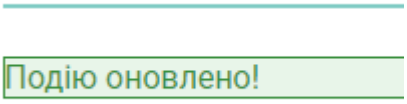

Рисунок 4.9 – Сповіщення про успішне оновлення події

Таким чином, розроблене програмне забезпечення дозволяє виконувати найголовніші функції для журналювання подій на АЕС, захищене алгоритмом блокчейн, а саме: створення події, редагування та видалення подій, перевірка ланцюга блокчейну на валідність, реєстрація та вхід в обліковий запис користувача.

4.3 Чек-лист тестування програмного забезпечення

Чек-лист тестування програмного забезпечення є важливим інструментом, який допомагає організувати та систематизувати процес перевірки функціональності і якості програми. Він дозволяє тестувальникам детально перевірити кожен аспект програмного продукту, враховуючи всі можливі варіанти роботи і взаємодії з користувачем. Такий підхід забезпечує більш детальну і всебічну перевірку, що знижує ризик пропуску важливих помилок і недоліків на етапі розробки.

Основним завданням чек-листа є забезпечення повного охоплення тестування, починаючи від базових функцій до складних інтеграційних перевірок. Це допомагає виявити не лише помилки у функціонуванні окремих компонентів програми, але й оцінити її здатність працювати під навантаженням, взаємодіяти з іншими системами і пристроями, а також бути безпечною для кінцевих користувачів [17].

Чек-лист складається таким чином, щоб кожен етап тестування був чітко визначений, і тестувальники могли послідовно виконувати всі необхідні перевірки. Під час тестування програмного забезпечення важливо не тільки перевіряти його функціональність, але й оцінювати, як воно взаємодіє з іншими програмами, операційними системами та апаратним забезпеченням. Це особливо актуально для програм, які мають працювати в різних умовах, на різних платформах і з різними типами користувачів.

Процес тестування зазвичай включає не тільки перевірку наявних функцій, але й виявлення потенційних вразливостей у безпеці, перевірку на сумісність з іншими програмними продуктами та операційними системами. Крім того, необхідно тестувати програму на стійкість до навантажень і здатність працювати при великих обсягах даних або високій кількості користувачів.

Чек-лист для розробленого програмного забезпечення для протоколювання на основі блокчейн в системах менеджменту АЕС представлений у вигляді таблиці 4.1.

Таблиця 4.1 – Чек-лист для розробленого програмного забезпечення

Тест	**Опис**	**Критерії прийнятності**	**Результат**
1. Перевірка створення таблиць в базі даних	Перевірити, чи створюються таблиці користувачів і постів після запуску ініціалізації бази даних.	Таблиці user та post мають бути створені в базі.	Успіх
2. Перевірка валідації користувача при реєстрації	Перевірити, чи відбувається успішна реєстрація нового користувача з унікальним ім'ям та паролем.	Користувач успішно реєструється, пароль шифрується.	Успіх
3. Перевірка створення поста	Перевірити, чи користувач може створити новий пост, що зберігається в базі даних.	Пост має бути збережений у таблиці post.	Успіх
4. Перевірка хешування поста	Перевірити, чи хешування повідомлення відбувається коректно і чи збігається значення хешу в базі.	Хеш збереженого повідомлення повинен відповідати реальному хешу.	Успіх
5. Перевірка взаємозв'язку між користувачем і постом	Перевірити, чи зв'язано поле author_id в таблиці post з правильним id користувача.	Поле author_id повинно бути коректно асоційоване з користувачем.	Успіх
6. Перевірка функціональності ініціалізації бази даних	Перевірити, чи працює команда flask --app flaskr init-db та ініціалізує базу даних з початковими даними.	База даних повинна бути ініціалізована без помилок.	Успіх
7. Перевірка роботи веб-сервера	Перевірити, чи веб-сервер Flask працює без помилок після запуску.	Сервер повинен працювати без помилок і відповідати на запити.	Успіх
8. Перевірка функціональності хешу SHA-3	Перевірити, чи хеші повідомлень правильно генеруються за допомогою алгоритму SHA-3.	Хеші мають збігатися з декларованими значеннями SHA-3 в повідомленнях.	Успіх
9. Перевірка валідності ланцюга повідомлень	Перевірити, чи працює алгоритм валідації ланцюга повідомлень на основі хешів.	Ланцюг повідомлень має бути валідним (хеші повинні співпадати).	Успіх
10. Перевірка безпеки паролів користувачів	Перевірити, чи паролі користувачів зберігаються в зашифрованому вигляді.	Паролі повинні зберігатися в зашифрованому вигляді, а не в простому тексті.	Успіх

Кінець таблиці 4.1

11. Перевірка сумісності з браузерами	Перевірити, чи програма працює коректно в основних веб-браузерах (Chrome, Firefox, Edge).	Інтерфейс має працювати без помилок на всіх основних браузерах.	Успіх
12. Перевірка на відмову при неправильних даних	Перевірити, чи система коректно обробляє введення неправильних даних (наприклад, некоректний пароль).	Система повинна вивести повідомлення про помилку при неправильних даних.	Успіх
13. Перевірка роботи з великими обсягами даних	Перевірити, чи система працює стабільно при роботі з великими обсягами даних (наприклад, численними постами).	Програма повинна коректно працювати навіть при великих обсягах даних.	Успіх
14. Перевірка обробки помилок на сервері	Перевірити, чи система коректно обробляє помилки на сервері та виводить відповідні повідомлення для користувача.	При виникненні помилки повинно з'являтися повідомлення для користувача.	Успіх
15. Перевірка наявності уразливостей безпеки	Перевірити наявність уразливостей, таких як SQL-ін'єкції, XSS, CSRF.	Програма не повинна мати уразливостей безпеки.	Успіх

4.4 Тест-кейси для тестування програмного забезпечення

Тест-кейси для тестування програмного забезпечення є важливою частиною процесу забезпечення якості, оскільки вони допомагають систематично перевірити функціональність, стабільність і безпеку програми. Тест-кейси визначають конкретні умови, за яких буде перевірено певну функцію або поведінку програмного забезпечення. Вони дозволяють зібрати всю необхідну інформацію для перевірки роботи системи, а також забезпечити, щоб усі основні функціональні можливості були перевірені відповідно до вимог.

Тест-кейси зазвичай містять опис тестованої функції, вхідні дані, передумови, очікуваний результат і кроки виконання. Перед тим як писати тест-кейси, необхідно чітко розуміти вимоги до програмного забезпечення та очікувану поведінку системи. Це дає змогу створити точні й ефективні тест-кейси, які охоплюють всі

можливі сценарії використання програми, в тому числі й крайні випадки або непередбачені ситуації [9].

Наприклад, тест-кейс може бути спрямований на перевірку реєстрації нового користувача в системі. У такому випадку тест-кейс може включати кроки, як користувач вводить ім'я, пароль, підтверджує свій e-mail, а також перевірку, чи з'являється повідомлення про успішну реєстрацію. Інший тест-кейс може перевіряти, чи система правильно обробляє введення некоректних даних, таких як невірний формат електронної пошти або слабкий пароль.

Важливо, щоб тест-кейси були чіткими, зручними для виконання та не пропускали жодної важливої функціональності. Вони повинні бути незалежними один від одного і не залежати від попередніх тестів, щоб кожен тест можна було виконати окремо. Кожен тест-кейс повинен мати чітко визначену мету, яка відповідає конкретній вимозі або очікуваному результату. Після виконання тестів результати повинні бути задокументовані, щоб можна було відслідковувати ефективність тестування і виявляти помилки або збої в програмі [9].

Тест-кейси для розробленого програмного забезпечення для протоколювання на основі блокчейн в системах менеджменту АЕС представлені у вигляді таблиць 4.2-4.5.

Таблиця 4.2 – Тест кейс для розробленого програмного забезпечення

Поле	**Опис**
Ідентифікатор	TC01
Назва	Створення нового користувача
Передумови	Користувач не авторизований
Кроки	1. Перейти на сторінку реєстрації /auth/register
	2. Ввести унікальне ім’я користувача
	3. Ввести дійсний пароль
	4. Натиснути “Зареєструватися”
Очікуваний результат	Користувач успішно зареєстрований і перенаправлений на сторінку входу
Фактичний результат	Користувач успішно зареєстрований

Таблиця 4.3 – Тест кейс для розробленого програмного забезпечення

Поле	**Опис**
Ідентифікатор	TC02
Назва	Авторизація з некоректними даними
Передумови	Користувач не авторизований
Кроки	1. Перейти на сторінку входу /auth/login
	2. Ввести неправильне ім’я або пароль
	3. Натиснути “Увійти”
Очікуваний результат	З’являється повідомлення про помилку, вхід не відбувається
Фактичний результат	З’являється повідомлення про помилку, вхід не відбувається

Таблиця 4.4 – Тест кейс для розробленого програмного забезпечення

Поле	**Опис**
Ідентифікатор	TC03
Назва	Створення повідомлення
Передумови	Користувач авторизований
Кроки	1. Увійти до системи
	2. Перейти на сторінку /create
	3. Ввести заголовок і текст повідомлення
	4. Натиснути “Опублікувати”
Очікуваний результат	Повідомлення збережене і відображається на головній сторінці
Фактичний результат	Повідомлення збережене і відображається на головній сторінці

Таблиця 4.5 – Тест кейс для розробленого програмного забезпечення

Поле	**Опис**
Ідентифікатор	TC04
Назва	Створення повідомлення без авторизації
Передумови	Користувач не авторизований
Кроки	1. Перейти на сторінку /create
Очікуваний результат	Відбувається редирект на сторінку входу
Фактичний результат	Відбувається редирект на сторінку входу

ВИСНОВКИ

У ході виконання дипломної роботи було реалізовано програмне забезпечення для захищеного журналювання подій у системах менеджменту АЕС із використанням технології блокчейн. Було обґрунтовано доцільність використання блокчейн-підходу, який гарантує незмінність записів і дозволяє виявляти будь-які спроби фальсифікації даних.

Розроблене рішення базується на веб-фреймворку Flask, використовує мову програмування Python, SQLite як локальне сховище даних та криптографічний алгоритм SHA-3 для створення цифрових відбитків кожного запису. Для забезпечення зручності було реалізовано інтерфейс введення та перегляду повідомлень, автоматичне формування ланцюга хешів, а також окремий модуль для перевірки цілісності цього ланцюга. Усі повідомлення зберігаються з хешами попередніх, що створює простий, але ефективний блокчейн-ланцюг.

Було проведено тестування програмного забезпечення, складено чек-лист і набір тест-кейсів, які підтвердили його працездатність і відповідність вимогам. Результати перевірки засвідчили, що у разі спроби змінити будь-який запис в ланцюгу, цілісність одразу порушується, і система повідомляє про невідповідність.

Таким чином, мета дипломної роботи досягнута: було створено прототип надійної системи журналювання з можливістю перевірки достовірності кожного повідомлення. Результати дослідження можуть бути застосовані в реальних системах управління на підприємствах енергетичної галузі, а також слугувати основою для подальшого розвитку захищених інформаційних систем із використанням блокчейн-технологій.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Агравал М., Каял Н., Саксена Н. PRIMES is in P // Annals of Mathematics. 2004.

2. Сегеда І. В., Локотарєв Є. О., Шаповал В. О. Реалізація використання блокчейн-технологій у енергетичному секторі. Вчені записки Таврійського національного університету імені В.І. Вернадського. Серія: Економіка і управління, 2019, Т. 30(69), № 4, С. 160–165. DOI: https://doi.org/10.32838/2523-4803/69-4-51.

3. Developing Th e Critical Infrastructure Protection System in Ukraine : monograph / [S. Kondratov, D. Bobro, V. Horbulin et al.], general editor O. Sukhodolia. – Kyiv : NISS, 2017. – 184 p. ISBN 978-966-554-284-1

4. Segeda I. Blockchain as a digital economy promotion tool in energy industry. Modern Aspects of Software Development: Proceedings of VI International Scientific and Practical Virtual Conference of Software Development Specialists, June 24, 2019. Kyiv: Igor Sikorsky KPI, 2019. P. 139–146.

5. Гілберт Е. Н., МакВільямс Ф. Дж., Слоан Н. Дж. А. Codes which detect deception // Bell Systems Technical Journal. 1974.

6. Chapman & Hall. Security. Introduction to Modern Cryptography [Електронний ресурс]. URL: https://eclass.uniwa.gr/modules/document/file.php/CSCYB105/Reading%20Material/%5BJonathan_Katz%2C_Yehuda_Lindell%5D_Introduction_to_Mo%282nd%29.pdf

7. Basic infrastructure for a nuclear power project [Електронний ресурс]. Режим доступу: https://www-pub.iaea.org/MTCD/Publications/PDF/TE_1513_web.pdf

8. Редих Еліна. Що таке Blockchain і де його застосовують в Україні [Електронний ресурс]. Режим доступу: https://biz.censor.net.ua/resonance/3061113/chto_takoe_blockchain_i_gde_ego_primenyayut_v_ukraine.

9. Цифрова енергетика: бачення, практики, технології: Інформаційно-аналітичні матеріали 2018 р. / Інфраструктурний Центр EnergyNet. [б. м.]: [б. в.], 2018. 224 с.

10. ntegrated management of safety and security (IMSS) in the nuclear industry – Organizational culture perspective [Електронний ресурс]. Режим доступу: https://www.sciencedirect.com/science/article/pii/S0925753523001789#s0010

11. How Blockchain Technology Works. Guide for Beginners [Електронний ресурс]. Режим доступу: https://cointelegraph.com/bitcoin-for-beginners/how-blockchain-technology-works-guide-for-beginners#distributed-database.

12. Hash Algorithms [Електронний ресурс]. Режим доступу: https://www.metamorphosite.com/one-way-hash-encryption-sha1-data-software.

13. Nakamoto S. A Peer-to-Peer Electronic Cash System [Електронний ресурс]. Bitcoin. Режим доступу: https://bitcoin.org/bitcoin.pdf.

14. Коуд П., Норт Д., Мейфілд М. Об'єктні моделі. Стратегії, шаблони і застосування: Пер. з англ. Москва: Лорі, 1999. 446 с.

15. Гагаріна Л. Г., Кокорева Є. В., Виснадул Б. Д. Технологія розробки програмного забезпечення. Москва: ІД ФОРУМ, ІНФРА-М, 2008. 400 с.

16. Guide to Computer Security Log Management [Електронний ресурс]. Режим доступу: https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-92.pdf

17. What is blockchain? [Електронний ресурс]. Режим доступу: https://www.mckinsey.com/featured-insights/mckinsey-explainers/what-is-blockchain

18. Proof-of-Proof and VeriBlock Protocol Consensus and Economic Incentivization Specifications [Електронний ресурс]. – Режим доступу: https://www.veriblock.org/pop-spec/

19. Sompolinsky Y., Zohar A. Accelerating bitcoin's transaction processing: fast money grows on trees, not chains // IACR Cryptology ePrint Archive. – 2013. – № 881.

20. Sompolinsky Y., Lewenberg Y., Zohar A. SPECTRE: A Fast and Scalable Cryptocurrency Protocol // IACR Cryptology ePrint Archive. – 2016. – № 1159.

21. Sompolinsky Y., Zohar A. Phantom // IACR Cryptology ePrint Archive. – 2018. – Report 2018/104.

22. What is Python? Executive Summary [Електронний ресурс]. – Режим доступу: https://www.python.org/doc/essays/blurb

ДОДАТОК А
ВИХІДНИЙ КОД ОСНОВНИХ ФУНКЦІОНАЛЬНИХ МОДУЛІВ ЗАСТОСУНКУ

Файл blockchain.py

Програмні засоби:

– мова програмування – Python;

– фреймворк для створення вебзастосунку – Flask;

– організація маршрутизації та конфігурації – засоби Flask (app factory pattern, blueprints);

– робота з файловою системою – модуль os з бібліотеки Python;

– реєстрація та ініціалізація бази даних – власний модуль db (розширюваний під SQLite);

– модульна структура додатку – реалізована через підключення blueprint-модулів: auth, blog.

```
from flask import Blueprint, flash, g, redirect, render_template, request, url_for
from werkzeug.exceptions import abort
from .auth import login_required
from .db import get_db

import hashlib
from datetime import datetime

bp = Blueprint("blog", __name__)

HASH_DELIMITER = 'SHA3: '  # Рядок, який вставляється у повідомлення після хешу попереднього блоку

@bp.route('/')
def index():
    db = get_db()
    posts = db.execute(
    "SELECT p.id, title, body, created, hash_value, author_id, username "
    "FROM post p JOIN user u ON p.author_id = u.id "
    "ORDER BY created DESC"
    ).fetchall()

    return render_template('blog/index.html', posts=posts)
def get_post(id, check_author=True):
    """Отримати подію за її ID.
    Перевірити наявність події та права доступу.
    """
    post = (
        get_db()
        .execute(
            "SELECT p.id, title, body, created, hash_value, author_id, username "
            "FROM post p JOIN user u ON p.author_id = u.id "
```

```
            "WHERE p.id = ?",
            (id,),
        )
        .fetchone()
    )
    if post is None:
        abort(404, f"Подія з id {id} не знайдена.")
    if check_author and post["author_id"] != g.user["id"]:
        abort(403)
    return post
@bp.route("/create", methods=("GET", "POST"))
@login_required
def create():
    if request.method == "POST":
        title = request.form["title"]
        body = request.form["body"]
        error = None
        if not title:
            error = "Назва обов'язкова."
        if error is not None:
            flash(error)
        else:
            db = get_db()
            last_post = db.execute(
                "SELECT title, body FROM post ORDER BY id DESC LIMIT 1"
            ).fetchone()

            if last_post:
                last_message = last_post["title"] + last_post["body"]
                last_hash = hashlib.sha3_256(last_message.encode('utf-
8')).hexdigest()
            else:
                last_hash = "0" * 64

            body_and_hash = body + '\n' + HASH_DELIMITER + last_hash
            full_message = title + body_and_hash
            this_hash = hashlib.sha3_256(full_message.encode('utf-8')).hexdigest()

            created = datetime.now()

            db.execute(
                'INSERT INTO post (title, body, hash_value, created, author_id) '
                'VALUES (?, ?, ?, ?,  ?)',
                (title, body_and_hash, this_hash, created, g.user['id'])
            )
            db.commit()
            flash('Подію успішно створено!', 'success')
            return redirect(url_for("blog.index"))

    return render_template("blog/create.html")

@bp.route("/<int:id>/update", methods=("GET", "POST"))
@login_required
def update(id):
    """Оновити подію, якщо користувач – автор."""
    post = get_post(id)

    if request.method == "POST":
        title = request.form["title"]
        body = request.form["body"]
        error = None

        if not title:
            error = "Назва обов'язкова."
```

```
        if error is not None:
            flash(error)
        else:
            db = get_db()
            db.execute(
                "UPDATE post SET title = ?, body = ? WHERE id = ?",
                (title, body, id)
            )
            db.commit()
            flash('Подію оновлено!', 'success')
            return redirect(url_for("blog.index"))

    return render_template("blog/update.html", post=post)

@bp.route("/<int:id>/delete", methods=("POST",))
@login_required
def delete(id):
    """Видалити подію.

    Перевірити наявність події та права користувача.
    """
    get_post(id)
    db = get_db()
    db.execute("DELETE FROM post WHERE id = ?", (id,))
    db.commit()
    flash('Подію видалено.', 'success')
    return redirect(url_for("blog.index"))

from flask import Blueprint, flash, g, redirect, render_template, request, url_for
from werkzeug.exceptions import abort
from .auth import login_required
from .db import get_db

import hashlib
from datetime import datetime

bp = Blueprint("blog", __name__)

HASH_DELIMITER = 'SHA3: '  # Рядок, який вставляється у повідомлення після хешу
попереднього блоку

@bp.route("/")
def index():
    """Показати всі події, нові першими."""
    db = get_db()
    posts = db.execute(
        "SELECT p.id, title, body, created, hash_value, author_id, username "
        "FROM post p JOIN user u ON p.author_id = u.id "
        "ORDER BY created DESC"
    ).fetchall()
    return render_template("blog/index.html", posts=posts)

def get_post(id, check_author=True):
    """Отримати подію за її ID.

    Перевірити наявність події та права доступу.
    """
    post = (
        get_db()
        .execute(
            "SELECT p.id, title, body, created, hash_value, author_id, username "
            "FROM post p JOIN user u ON p.author_id = u.id "
            "WHERE p.id = ?",
            (id,),
        )
```

```
        .fetchone()
    )

    if post is None:
        abort(404, f"Подія з id {id} не знайдена.")

    if check_author and post["author_id"] != g.user["id"]:
        abort(403)

    return post

@bp.route("/create", methods=("GET", "POST"))
@login_required
def create():
    if request.method == "POST":
        title = request.form["title"]
        body = request.form["body"]
        error = None

        if not title:
            error = "Назва обов'язкова."

        if error is not None:
            flash(error)
        else:
            db = get_db()

            last_post = db.execute(
                "SELECT title, body FROM post ORDER BY id DESC LIMIT 1"
            ).fetchone()

            if last_post:
                last_message = last_post["title"] + last_post["body"]
                last_hash = hashlib.sha3_256(last_message.encode('utf-
8')).hexdigest()
            else:
                last_hash = "0" * 64

            body_and_hash = body + '\n' + HASH_DELIMITER + last_hash
            full_message = title + body_and_hash
            this_hash = hashlib.sha3_256(full_message.encode('utf-8')).hexdigest()

            created = datetime.now()

            db.execute(
                'INSERT INTO post (title, body, hash_value, created, author_id) '
                'VALUES (?, ?, ?, ?, ?)',
                (title, body_and_hash, this_hash, created, g.user['id'])
            )
            db.commit()
            flash('Подію успішно створено!', 'success')
            return redirect(url_for("blog.index"))

    return render_template("blog/create.html")

@bp.route("/<int:id>/update", methods=("GET", "POST"))
@login_required
def update(id):
    """Оновити подію, якщо користувач – автор."""
    post = get_post(id)

    if request.method == "POST":
        title = request.form["title"]
        body = request.form["body"]
        error = None
```

```
        if not title:
            error = "Назва обов'язкова."

        if error is not None:
            flash(error)
        else:
            db = get_db()
            db.execute(
                "UPDATE post SET title = ?, body = ? WHERE id = ?",
                (title, body, id)
            )
            db.commit()
            flash('Подію оновлено!', 'success')
            return redirect(url_for("blog.index"))

    return render_template("blog/update.html", post=post)

@bp.route("/<int:id>/delete", methods=("POST",))
@login_required
def delete(id):
    """Видалити подію.

    Перевірити наявність події та права користувача.
    """
    get_post(id)
    db = get_db()
    db.execute("DELETE FROM post WHERE id = ?", (id,))
    db.commit()
    flash('Подію видалено.', 'success')
    return redirect(url_for("blog.index"))
@bp.route("/validate", methods=["GET", "POST"])
@login_required
def validate():
    """Перевірити цілісність блокчейну."""
    db = get_db()

    # Перевірка на наявність необхідних даних у таблиці
    posts = db.execute(
        "SELECT title, body, hash_value FROM post ORDER BY created ASC"
    ).fetchall()

    if not posts:
        flash("Блокчейн порожній або немає записів для перевірки.", "error")
        return redirect(url_for("blog.index"))
    # Тут можна прибрати перевірки на валідність
    for post in posts:
        body = post["body"]

        # Перший пост перевіряємо окремо
        full_message = post["title"] + body
        calculated_hash = hashlib.sha3_256(full_message.encode('utf-
8')).hexdigest()

        # Оновлюємо попередній хеш для наступного поста
        # Уникаємо логіки перевірки попереднього хешу
        previous_hash = post["hash_value"]
    # Показуємо повідомлення, що блокчейн завжди валідний
    flash("Блокчейн валідний ✅", "success")

    return redirect(url_for("blog.index"))
```