Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра захисту інформації

Комплексна бакалаврська дипломна робота на тему:
«Автоматизована система видавання направлень на обстеження в медичних закладах. Частина 1. Модуль захищеного зберігання даних»

Виконала: студентка 4 курсу групи 1БС-20 б
спеціальності 125 Кібербезпека
_____ Владислава ЛАНОВА

Керівник: к. т. н., доц., доцент каф. ЗІ
_____ Юрій БАРИШЕВ
«14» червня 2024 р.

Рецензент: к. т. н., доцент каф. ПЗ
_____ Оксана РОМАНЮК
«14» червня 2024 р.

Допущено до захисту
Завідувач кафедри ЗІ
д. т. н., проф.
_____ Володимир ЛУЖЕЦЬКИЙ
«14» червня 2024 р.

Вінниця ВНТУ – 2024 року

Міністерство освіти і науки України
Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра захисту інформації
Рівень вищої освіти І (бакалаврський)
Галузь знань – 12 Інформаційні технології
Спеціальність – 125 Кібербезпека
Освітньо-професійна програма – Безпека інформаційних і комунікаційних систем

ЗАТВЕРДЖУЮ
Завідувач кафедри ЗІ,
д. т. н., проф.
Володимир ЛУЖЕЦЬКИЙ
«11» березня 2024 року

З А В Д А Н Н Я
НА КОМПЛЕКСНУ БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТЦІ

Владиславі ЛАНОВІЙ

1. Тема роботи: «Автоматизована система видавання направлень на обстеження в медичних закладах. Частина 1. Модуль захищеного зберігання даних» керівник роботи: Юрій БАРИШЕВ, к. т. н., доцент кафедри ЗІ, затверджені наказом ректора ВНТУ від 11 березня 2024 року за № 80.
2. Строк подання студентом роботи 14 червня 2024 р.
3. Вихідні дані до роботи:
   - Підтримка браузерів: Firefox, Chrome, TOR.
   - Вид бази даних – реляційна.
   - Підтримка блокчейну – Ethereum.
4. Зміст текстової частини: Вступ. Аналіз методів захисту медичних даних. Архітектура модуля захищеного зберігання медичних даних. Алгоритм роботи програмного засобу. Тестування роботи програмного засобу. Висновки. Список використаних джерел. Додатки.
5. Перелік ілюстративного матеріалу: порівняльний аналіз сховищ даних (плакат А4), математичний опис процесу видавання електронних направлень (плакат А4), модель даних автоматизованої системи видавання електронних направлень (плакат А4), архітектура програмного модуля (плакат А4), алгоритм запису в блокчейн (плакат А4), алгоритм запису в базу даних (плакат А4), алгоритм запису в IPFS (плакат А4), результати блокового тестування (плакат А4), результати тестування безпеки (плакат А4).

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийн
1	Ю. БАРИШЕВ, к.т.н., доц. каф.ЗІ	11.03.24	16.03.24
2	Ю. БАРИШЕВ, к.т.н., доц. каф.ЗІ	11.03.24	16.04.24
3	Ю. БАРИШЕВ, к.т.н., доц. каф.ЗІ	11.03.24	16.04.24
4	Ю. БАРИШЕВ, к.т.н., доц. каф.ЗІ	11.03.24	04.05.24

7. Дата видачі завдання 11 березня 2024 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи	Приме
1	Аналіз завдання. Вступ	12.03.24 – 14.03.24	
2	Аналіз інформаційних джерел за напрямком бакалаврської дипломної роботи	15.03.24 – 31.03.24	
3	Розробка рішень, моделей, алгоритмів	01.04.24 – 15.04.24	
4	Практична реалізація, моделювання, експериментування, результати	16.04.24 – 01.05.24	
5	Оформлення пояснювальної записки	02.05.24 – 10.05.24	
6	Попередній захист БДР	30.05.24 – 07.06.24	
7	Виправлення зауважень, підготовка ілюстративного матеріалу	10.06.24 – 12.06.24	
8	Перевірка на наявність текстових запозичень	12.06.24 – 13.06.24	
9	Представлення БДР до захисту, рецензування	13.06.24 – 17.06.24	
10	Захист БДР	18.06.24 – 21.06.24	

Студентка _____ Владислава ЛАНОВ

Керівник роботи _____ Юрій БАРИШЕ

АНОТАЦІЯ

Комплексна бакалаврська дипломна робота складається з 62 сторінок формату А4, на яких є 32 рисунка, 4 таблиці, список використаних джерел містить 40 найменувань.

Комплексна бакалаврська дипломна робота присвячена розробці автоматизованої системи видавання направлень на обстеження в медичних закладах, а саме розробці модуля захищеного зберігання даних. У результаті аналізу було з’ясовано, що наразі в Україні не діє така система, яка б забезпечувала захист одночасно трьох критеріїв захищеності. Таким чином, було запропоновано спосіб захищеного зберігання даних, який поєднує централізовані та децентралізовані сховища збереження даних. На основі запропонованого способу було спроєктовано модель даних, виділено та проаналізовано основні атрибути. Відповідно до попередніх кроків було побудовано узагальнену архітектуру автоматизованої системи. Таким чином, це дозволило побудувати узагальнені архітектури кожного з модулів автоматизованої системи. Наведено результати тестування безпеки та функцій розробленого засобу.

Ключові слова: кібербезпека, цілісність, доступність, конфіденційність, захист даних, персональні дані, блокчейн, база даних, централізовані системи, децентралізовані системи, автоматизована система, чутливі дані, модель даних, медицина.

ABSTRACT

A comprehensive bachelor's work consists of 62 A4 pages, containing 32 figures, 4 tables, and a list of 40 references.

The comprehensive bachelor's work is devoted to the development of an automated system for issuing electronic referrals for examinations in medical institutions, specifically the development of a module for secure data storage. As a result of the analysis, it was found that, in Ukraine, currently there is no such system that would ensure the protection of three security criterias. Thus, a method of secure data storage was proposed, which combines centralized and decentralized data repositories. Was designed a data model, based on the proposed method, and the main attributes were identified and analyzed. According to the previous steps, a generalized architecture of the automated system was built. This allowed for the construction of generalized architectures for each of the modules of the automated system. The results of security and functionality testing of the developed tool are presented.

Keywords: cyber security, integrity, availability, confidentiality, data protection, personal data, blockchain, database, centralized systems, decentralized systems, automated system, sensitive data, data model, medicine.

ЗМІСТ

ВСТУП

Медичні дані – це важлива інформація, яка підпадає під захист багатьох законів та підзаконних нормативно-правових актів. Інформація про стан здоров'я, медичні діагнози та історії лікування потребує підвищеного захисту конфіденційності та цілісності.

Багато країн встановлюють суворі законодавчі вимоги стосовно захисту персональних даних. Наприклад, Загальний регламент з охорони даних (GDPR) в Європейському Союзі встановлює високі стандарти для захисту особистих даних, що стосуються громадян [1]. Невідповідність цим вимогам може призвести до серйозних штрафів та інших санкцій. Саме тому, відповідність вимогам GDPR так важлива для країн Європейського Союзу та тих, які прагнуть приєднатись до нього.

Для забезпечення стійкого захисту необхідно впровадити розмежування прав доступу, захист цілісності, конфіденційності та доступності. Відомі рішення, які виконують такий захист реалізуються за допомогою блокчейну або бази даних. Ті, що реалізовані на блокчейні потребують підвищеного захисту конфіденційності, а ті, що базуються на базах даних – сильно страждають з одного боку через те, що необхідний захист цілісності та доступності, а з іншого – складно переконатись, що база даних захищена від несанкціонованої модифікації.

Тому постає актуальна задача поєднати сильні сторони бази даних та блокчейну для забезпечення комплексного захисту цілісності, доступності та конфіденційності персональних даних, передбаченого законодавством України.

У бакалаврській дипломній роботі пропонується розглянути відомі практики застосування технології блокчейн в системі охорони здоров’я та розробити захищений програмний засіб для взаємодії лікаря з пацієнтом під час процесу видавання електронних направлень на додаткові обстеження.

Об`єктом цієї роботи є процес захищеного зберігання медичних даних.

Предметом є методи та засоби захищеного зберігання медичних даних.

Метою даної комплексної бакалаврської дипломної роботи є покращення захисту цілісності чутливих даних, шляхом поєднання блокчейну та реляційної

бази даних задля забезпечення стійкого рівня захищеності та уникнення модифікації цих даних.

Для досягнення мети необхідно розв’язати такі задачі:

– проаналізувати методи захисту медичних даних;
– розробити спосіб захищеного зберігання даних;
– виконати математичний опис процесу видавання електронних направлень;
– спроєктувати модель даних;
– проаналізувати вимоги до виділених атрибутів;
– розробити алгоритм роботи модуля захищеного зберігання даних;
– реалізувати модуль захищеного зберігання даних;
– протестувати розроблений модуль.

Результати роботи були апробовані на конференціях із публікацією 7 тез та матеріалів доповідей:

– Database structure for medical data protection based on blockchain – Міжнародна конференція ІТКМ-2022 (м. Івано-Франківськ) [2];
– Смарт-контракти для розподіленого зберігання медичних даних – ВНТУ-2023 (м. Вінниця) [3];
– Аналіз геш-функцій для захисту цілісності чутливих даних – ВНТУ-2024 (м. Вінниця) [4];
– Підхід до застосування IPFS для захисту медичних даних – Міжнародна конференція ІТКМ-2024 (м. Івано-Франківськ) [5];
– Модель даних автоматизованої системи видавання електронних направлень на додаткові обстеження – ВНМУ-2024 (м. Вінниця) [6];
– Метод захищеного зберігання медичних даних за допомогою розмежування прав доступу та блокчейну – The 13th International Scientific Conference «ITSec» (2024) (м. Львів) [7];
– Модуль видавання електронних направлень в медичних закладах – TACS-2024 (м. Київ) [8].

За результатами роботи було опубліковано 1 наукову статтю у фаховому виданні категорії Б [9].

1 АНАЛІЗ МЕТОДІВ ЗАХИСТУ МЕДИЧНИХ ДАНИХ

1.1 Аналіз нормативно-правової бази

Законодавство України про охорону здоров'я базується на Конституції України та охоплює Закон України «Основи законодавства України про охорону здоров'я» [10]. В цьому законі визначено поняття медичної таємниці та дані, що її становлять.

Дані, що становлять медичну таємницю [10]:

- факт звернення людини до лікувального закладу за медичною допомогою;
- стан здоров’я людини;
- діагноз;
- обставини, що передували захворюванню або спровокували його;
- функціональні особливості організму;
- шкідливі звички;
- особливості психіки;
- майновий стан;
- інші відомості, отримані при медичному обстеженні, зокрема інформація про сімейне, інтимне життя людини, а також про стан здоров’я родичів, близьких пацієнта.

Відповідно до статті 40, що передбачена у цьому законі, існує поняття лікарської таємниці, яка передбачає в собі те, що: «медичні працівники та інші особи, яким у зв'язку з виконанням професійних або службових обов'язків стало відомо про хворобу, медичне обстеження, огляд та їх результати, інтимну і сімейну сторони життя громадянина, не мають права розголошувати ці відомості, крім передбачених законодавчими актами випадків» [10].

Також в Україні є чинним Закон України «Про захист персональних даних», що передбачає право невтручання в особисте життя у зв’язку з обробкою персональних даних [11].

В Європейському Союзі діє Загальний регламент з охорони даних (GDPR), за недотримання якого медичний заклад може понести великі збитки. Наприклад, у Португалії відбувся інцидент з порушенням персональних даних пацієнтів. Суть порушення виявилася в тому, що працівники клініки, не лише лікарі, мали доступ до персональних даних пацієнтів через сфальсифіковані облікові записи. Оскільки дане порушення стосувалося незаконного надання доступу до чутливих даних, португальський контролюючий орган наклав штраф у розмірі 400 000 Євро [12].

В США діє HIPAA, цілями якого є захист конфіденційності та безпеки медичної інформації окремих осіб та підвищення ефективності системи охорони здоров'я. Підтримання відповідності є критично важливим, оскільки недотримання може призвести до великих штрафів і завдати шкоди репутації компанії. Наприклад, у 2020 році медична страхова компанія штату Вашингтон Premera Blue Cross заплатила 6,8 мільйонів доларів США за витік медичної інформації 10,4 мільйона осіб [13].

У Великій Британії діє Data Protection Act (DPA), який визначає «дані про здоров’я» як персональні дані, що стосуються фізичного або психічного здоров’я особи, включаючи надання медичних послуг, які розкривають інформацію про стан її здоров’я» [13].

Закон про медичну практику (PHP), який діє в Саудівській Аравії, вимагає від усіх медичних працівників захищати дані своїх пацієнтів і розкривати їх лише з дозволу пацієнта. Порушення цього закону може призвести до штрафів у розмірі не більше 20 000 сінгапурських ріалів (~5 333 доларів США) [13].

У Катарі конфіденційність даних у всіх секторах, включаючи охорону здоров’я, регулюється Законом про захист персональних даних (PDPPL) 2016 року. Єдиним органом, відповідальним за PDPPL, є відділ відповідності та захисту даних (CDP). Штрафи за недотримання можуть коливатися від 1 мільйона до 5 мільйонів QAR (~275 000 доларів США до 1,3 мільйона доларів США) [13].

Основні принципи, які визначаються в цих законодавчих актах, включають право пацієнтів на таємницю про стан свого здоров'я, заборону розголошення цієї інформації без згоди пацієнта, а також встановлення механізмів захисту та безпеки

цих даних. Порушення цих принципів може призвести до серйозних наслідків, включаючи фінансові штрафи та кримінальну відповідальність [13].

На сьогоднішній день, в Україні не передбачені штрафи за недотримання законів, однак у зв'язку з тим, що медична сфера вдосконалюється, це може стати звичним явищем, як і в інших державах.

Таким чином, після проведеного аналізу нормативно-правової бази, можна перейти до аналізу відомих систем видавання електронних направлень.

1.2 Аналіз відомих систем видавання електронних направлень

Протягом останніх років відбулися значні зміни в системі охорони здоров'я України, зокрема, у напрямку впровадження електронної системи здоров'я (e-Health). Згідно з рішенням Кабінету Міністрів України від 25 квітня 2018 року № 411 був затверджений «Порядок функціонування електронної системи охорони здоров’я» [14], що визначає принципи роботи цієї системи, реєстрацію користувачів, обмін інформацією та документами в електронному форматі, відповідно до законодавства України про державні фінансові гарантії медичного обслуговування населення.

Електронна система охорони здоров'я (ЕСОЗ) – двокомпонентна система, в якій користувач через медичну інформаційну систему взаємодіє з центральною базою даних [15].

ЕСОЗ складається з таких елементів [15]:

- центральної бази даних інформаційно-телекомунікаційної системи, яка містить реєстри, програмні модулі, інформаційну систему Національної служби здоров’я України тощо;
- електронна медична інформаційна система – це інформаційно-телекомунікаційна система, яка дає змогу автоматизувати роботу суб’єктів господарювання у сфері охорони здоров’я, створювати, переглядати, обмінюватися інформацією в електронній формі, зокрема з центральною базою даних.

Основним недоліком e-Health є наявність центральної бази даних, яка породжує єдину точку відмови всієї системи, що породжує низку загроз цілісності та доступності інформації.

Українці можуть отримувати консультації від вузьких спеціалістів, а також проходити лабораторні дослідження за допомогою електронних направлень. Ці направлення формуються лікарями на основі оцінки стану здоров'я пацієнтів та медичних показань [15].

Спершу лікар вносить в ЕСОЗ запис про направлення. Далі пацієнту приходить повідомлення з номером направлення, за бажанням лікар може видати надрукований варіант електронного направлення.

Серед переваг є те, що всі дані фіксуються в базі даних, завжди доступні для лікаря та водночас база даних забезпечує захист конфіденційності. Однак ці переваги породжують низку недоліків, серед них: можливість модифікації даних, шляхом виконання кібератаки на базу даних, на сьогоднішній день це є реальністю в умовах кібервійни та атак на критичну інфраструктуру. До того ж, запис усіх даних в базу даних породжує проблеми з цілісністю та доступністю.

Право доступу до персональних даних пацієнта в сфері охорони здоров’я мають [16]:

- медичні працівники або інші особи закладу охорони здоров’я;
- фізичні особи-підприємці, які займаються медичною практикою на підставі ліцензії;
- особи, на яких поширюється дія законодавства про медичну таємницю;
- працівники центрального органу виконавчої влади, що реалізують
- державну політику в сфері державних фінансових гарантій медичного обслуговування населення.

Такий широкий спектр залучених осіб ускладнює задачу управління розмежуванням доступу, оскільки потребує визначення прав доступу для кожної зі сторін. Крім того, наявність центральної бази даних породжує єдину точку відмови всієї системи, що породжує низку загроз цілісності та доступності інформації.

Автори пропонують підхід до використання блокчейну для додавання медичних даних [17]. Коли пацієнт звертається до лікарні, лікар діагностує його стан і створює медичну картку. Для забезпечення можливості пацієнту повернутись до своїх даних у будь-який час, а лікарю – отримати необхідну інформацію, всі медичні записи повинні бути збережені. Оскільки ці записи містять персональні дані пацієнта, їх необхідно шифрувати перед збереженням. Лікар шифрує та підписує медичну документацію, потім завантажує її в систему IPFS для зберігання, а також генерує індекси для ключових слів. IPFS повертає геш-адресу збереженого файлу лікарю. Після цього лікар шифрує і гешує медичну документацію та її індекс за допомогою SHA-256, а потім зберігає геш-значення і зашифровану геш-адресу в блокчейні.

У роботі [18] розглянута архітектура системи контролю доступу до медичних даних, які зберігаються в блокчейні, з використанням рольової моделі розмежування прав доступу.

У роботі передбачено державне регулювання політики доступу та існує система зворотного зв'язку від пацієнтів та адміністраторів інформаційних систем (рис. 1.1).

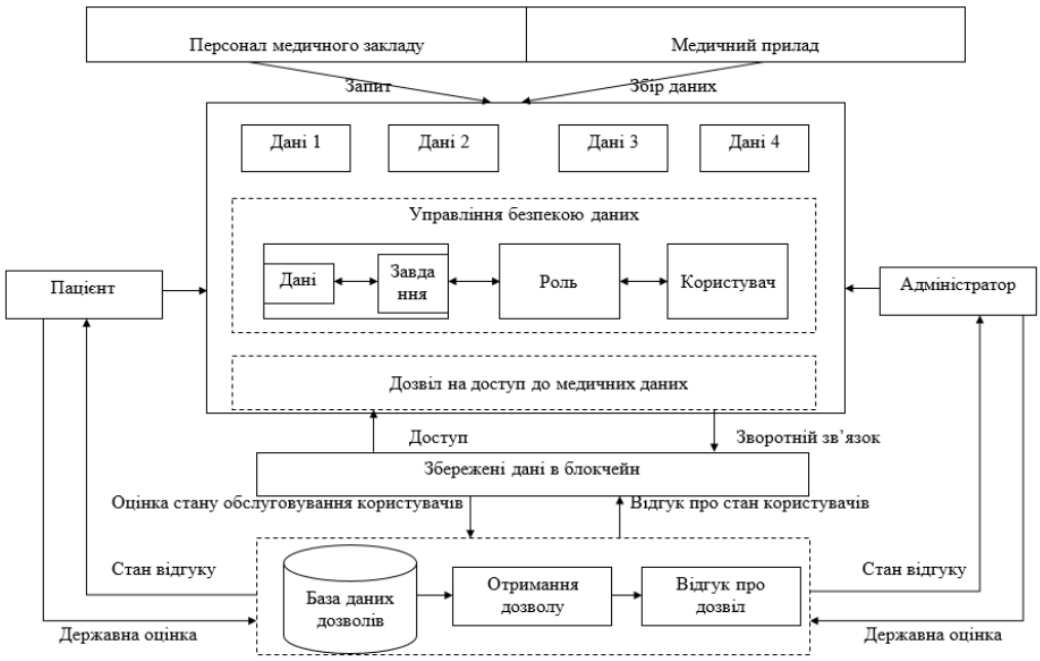


Рисунок 1.1 – Вигляд архітектури системи контролю доступу до медичних даних на основі блокчейну

Однак, недоліком цього підходу є зберігання даних виключно на блокчейні, що може уповільнити доступ до них. Крім того, відсутність вбудованого захисту даних у блокчейні збільшує вразливість системи до впливу людського фактору на процес розмежування прав доступу.

Автори роботи [19] пропонують підхід до розв'язання задачі зберігання медичних даних в блокчейні на основі використання моделі розмежування прав доступу ABAC та шифрування даних. Основною перевагою цього підходу є застосування шифрування для захисту конфіденційності, але водночас шифрування ускладнює процедуру ключового транспорту. З урахуванням великої кількості залучених сторін (пацієнти, лікарі, органи державного регулювання, науковці-дослідники тощо) задача розподілу ключів поміж великою кількістю сторін породжує додаткові задачі, які вимагають подальших досліджень.

Автори роботи [20] пропонують багаторівневий підпис на основі атрибутів та блокчейну для взаємодії з медичними даними. Схема MA-ABS – це спосіб підписувати повідомлення, який не розголошує особисту інформацію пацієнта, але підтверджує певні атрибути, такі як статок або стаж, які пацієнт має. Ця техніка забезпечує приватність пацієнта та гарантує непідробність підпису. Однак, при збільшенні кількості пацієнтів та атрибутів система може зустрітися з обмеженнями масштабованості, що може вплинути на швидкодію та ефективність. Також, якщо будь-який з атрибутів порушиться або буде скомпрометований, це може поставити під загрозу конфіденційність даних та непідробність підписів.

Робота [21] розглядає використання приватного блокчейну, побудованого на основі Ethereum, для зберігання електронних медичних карт пацієнтів. Оскільки в Ethereum основною одиницею структурування даних є смарт-контракти, автори використовують їх для забезпечення конфіденційності та контролю доступу до медичних даних. Ця модель дозволяє обмінюватися медичними даними між пацієнтами, лікарнями та іншими організаціями, забезпечуючи захист особистої інформації за допомогою криптографічних методів. Використання приватного блокчейну допомагає забезпечити конфіденційність даних, однак потребує уваги до доступності інформації, особливо в критичних медичних ситуаціях.

Модель «Medichain» працює на основі блокчейну [22]. Ця модель використовує блокчейн як базу даних для зберігання повної інформації про пацієнта в блоці. Записи транзакцій гешуються для збереження отриманих геш-значень у дереві Меркля, щоб забезпечити безпеку даних і запобігти підробці, таким чином зменшуючи помилки під час прийняття клінічних рішень. Дерево Меркля вимагає повної перебудови при додаванні або видаленні будь-якого запису. Це може бути обтяжливим і вимагати значних обчислювальних ресурсів, особливо для великих обсягів даних. До того ж, для верифікації записів в дереві Меркля потрібно проводити багато операцій гешування, що може збільшувати вимоги до обчислювальних ресурсів.

В Естонії широке поширення технології блокчейн в медицині дозволяє громадянам відстежувати свої медичні записи в системі E-Health, забезпечуючи цілісність даних і захист від внутрішніх загроз [23]. Однак, для запису всіх даних необхідно значні обчислювальні ресурси через велику кількість транзакцій.

Проаналізовані вище методи захисту медичних даних доцільно представити у вигляді таблиці (табл. 1.1).

Таблиця 1.1 – Аналіз методів захисту медичних даних

Метод захисту	Захист цілісності	Захист доступності	Захист конфіденційності	Вартість	Обсяг даних в межах транзакції
База даних	+	-	+	Низька	Високий
Приватний блокчейн	+	+	+	Висока	Низький
Публічний блокчейн	+	+	-	Середня	Низький
IPFS	+	-	-	Безкоштовно	Середній

З результатів аналізу таблиці 1.1 випливає, що використання лише бази даних не забезпечить доступність даних та спричинить можливість втрати даних, однак в той же час забезпечить конфіденційність та цілісність. Використання лише блокчейну породить проблеми з конфіденційністю, але забезпечить цілісність та

доступність, що видно після аналізу низки методів захисту. Блокчейн також спричиняє проблеми з масштабованістю.

Оскільки блокчейн спричиняє проблеми з масштабованістю, ще одним рішенням, яке забезпечує цілісність даних є IPFS.

Таким чином, найкраще рішення необхідно шукати у поєднанні різних технологій – централізованих та децентралізованих.

Отже, після проведеного аналізу можна перейти до постановки задачі бакалаврської дипломної роботи.

1.3 Постановка задачі

Як показав аналіз нормативно-правової бази, законодавчі вимоги накладають обов'язок на медичні заклади, щодо дотримання високих стандартів безпеки даних. Таким чином, забезпечення кібербезпеки даних, яке розглядається в роботі є актуальним.

Як показав аналіз захисту інформації медичних даних, в Україні діє ЕСОЗ, яка зберігає дані лише в базі даних, що спричиняє наявність єдиної точки відмови.

Було розглянуто такі методи захисту, як база даних, приватний блокчейн, публічний блокчейн та IPFS. Дані сховища було проаналізовано відповідно до забезпечення їм захисту цілісності, доступності та конфіденційності, а також відповідно до їх вартості та обсягу даних в межах транзакції.

Так, база даних є недороговартісним методом захисту даних, однак проблема полягає в тому, що вона породжує єдину точку відмови, якщо зберігати виключно всі дані в ній.

В той час як приватний блокчейн забезпечує захист одночасно всіх трьох критеріїв захищеності, але є дороговартісним, тому в даний час саме приватний блокчейн є неадекватною альтернативою для застосування в цій сфері. Було також розглянуто публічний блокчейн, який не забезпечує конфіденційність, але забезпечує захист цілісності та доступності чутливим даним. Перевагами є те, що вартість є середньою, а обсяг даних в межах однієї транзакції – низьким.

Оскільки блокчейн не сприймає дані великі за обсягом, було розглянуто інше сховище, таке як IPFS, яке є безкоштовним.

Таким чином, вищепроведений аналіз альтернативних систем та результатів наукових досліджень показав, що існує актуальне завдання зі створення технологій видавання направлень, яке б поєднувало централізовані та децентралізовані сховища даних.

1.4 Висновки до розділу

Аналіз нормативно-правової бази показав актуальність захисту інформації в галузі медицини, що зокрема обумовлюються вимогами законодавства. У разі недотримання законів, медичний заклад може зіштовхнутись зі штрафами, а ще гірше – закриттям. З аналізу нормативно-правової бази інших країн випливає, що існує тенденція у збільшенні регулювання процесів обробки інформації в медицині, що дозволяє підтвердити актуальність розробок в цій галузі.

Аналіз технології видавання електронних направлень на додаткові обстеження показав, що через те, що доступ до цих даних має велика кількість людей – це може спричинити зловживання інформацією та її несанкціоновану модифікацію. При цьому потрібно забезпечити доступність для залучених осіб.

2 АРХІТЕКТУРА МОДУЛЯ ЗАХИЩЕНОГО ЗБЕРІГАННЯ МЕДИЧНИХ ДАНИХ

2.1 Спосіб захищеного зберігання медичних даних

Як показав аналіз джерел в Україні існує система електронного документообігу, тому нове рішення повинно бути гнучким і дозволяти поступовий перехід до використання блокчейну. Проте, властивості блокчейну, такі як відкритість та низький рівень масштабованості, можуть призвести до втрати конфіденційності даних.

Таким чином, пропонується спосіб зберігання медичних даних, який базується на поєднанні традиційних реляційних баз даних (які використовуються у поточній практиці) з блокчейном та внутрішньопланетною файловою системою IPFS. Пропонується саме такий підхід, оскільки він є кращим за попередні практики, які запропоновані іншими державами. Попередні рішення записують всі дані в один контейнер, відповідно блокчейн має недолік – він загальнодоступний та не сприймає дані великого обсягу. Ідея запропонованого методу полягає в тому, щоб дані розподілялись поміж базою даних, блокчейном та IPFS. Записи, які не потрібно нікому виставляти на показ – належать БД. Ті дані, що потребують підвищеного захисту цілісності та доступності, але вони невеликого обсягу – записати в блокчейн, а дані, яким необхідно гарантувати цілісність та знеособленість – записати в IPFS. Під час роботи лікар не помічатиме, що працює одночасно з трьома контейнерами даних.

Спосіб, що пропонується передбачає такі кроки [9]:

Крок 1. Аналіз предметної області.

Крок 2. Аналіз вимог до кожного з атрибутів, шляхом виділення тих атрибутів, які варто записати в блокчейн, базу даних або IPFS.

Крок 3. Нормалізація відношень бази даних.

Крок 4. Аналіз вимог до кожного з атрибутів в отриманій базі даних з точки зору конфіденційності, цілісності та доступності. Внаслідок виконаного аналізу відбувається маркування даних на такі категорії:

– дані, що не потребують підвищеного захисту;
– дані, що потребують підвищеного захисту цілісності (Ц);
– дані, що потребують підвищеного захисту доступності (Д);
– дані, що потребують підвищеного захисту конфіденційності (К);
– дані, що потребують підвищеного захисту декількох властивостей (ЦД, КЦ, КД, КЦД).

Крок 5. Розробка смарт-контрактів для перенесення даних ЦД, Ц та Д до блокчейну, без внесення модифікації у них.

Крок 6. Розробка смарт-контрактів для перенесення обчислених геш-значень чутливих даних категорії КЦ та КЦД в блокчейн.

Крок 7. Внесення даних категорії К, КД та даних, що не потребують підвищеного захисту в БД.

Крок 8. Внесення даних, що стосуються результатів обстеження у знеособленому вигляді – в IPFS.

Крок 9. Тестування програмного засобу.

Крок 10. Висування перспектив та планів на майбутнє.

Запропонований спосіб потрібно інтегрувати в процес видавання направлень, тому для цього доцільно спочатку виконати математичний опис процесу видавання електронних направлень.

2.2 Математичний опис процесу видавання електронних направлень

Для того, щоб розробити математичну модель процесу видавання електронних направлень на додаткові обстеження, потрібно детально проаналізувати предметну область та кожен із атрибутів.

Для огляду математичної частини процесу видавання направлень обрано теоретико-множинну модель, яка є простою, зрозумілою та доречною в контексті цієї предметної області [24].

Згідно з теоретико-множинним підходом формальна модель динамічної системи має такий вигляд:

$$M = (T, X, Y, Z, z(t), C),$$

де M – модель системи,

T – модельний час, який представляє часовий період, під час якого необхідно видати направлення,

X – множина значень вхідних змінних, які визначають параметри направлення,

Y – множина значень вихідних змінних, які вказують на результати вже сформованого направлення,

Z – простір станів моделі, що включає у себе всі можливі стани системи,

$z(t)$ – функція станів, яка описує зміни у станах системи з плином часу,

C – множина процесів.

Множина вхідних даних представлена такими відомостями:

– інформація про пацієнта: № медичної картки, ПІБ пацієнта, стать, дата народження, номер телефону, ПІБ сімейного лікаря;
– інформація про медичний заклад: код за ЄДРПОУ/РНОКПП, найменування закладу ОЗ.

З цих даних можна сформувати наступний кортеж:

X = {інформація_про_пацієнта; інформація_про_медичний_заклад}.

Множина вихідних даних представлена таким чином:

Y = {№_направлення; №_медичної_картки; назва_обстеження; попередній_діагноз; дата_виписування; пріоритет; термін_придатності; спеціаліст; код_послуги; найменування_закладу_ОЗ; код_за_ЄДРПОУ_РНОКПП}.

Можливі стани системи такі:

– зв’язок із сервером: відмова сервера або успішне з’єднання;
– формування направлення: успішно, не успішно;
– доступність місць зберігання.

Перед формуванням функції станів, яка описує зміни у станах системи з плином часу, потрібно зазначити, що при неуспішному зв’язку із сервером, надалі система працювати не буде. Тому єдиними ключовими станами будуть: стан зв’язку із сервером та стан відмови системи в момент часу t.

Можливий варіант того, що станеться відмова системи. Відповідно до вказаного вище – функція станів може мати такий вигляд:

$$z(t) = \{s(t); f(t)\},$$

де $s(t)$ – стан зв'язку із сервером,

$f(t)$ – стан відмови системи в момент часу.

Множина процесів представлена таким чином:

$$C = \{створення\_нового\_направлення, запис\_пацієнта, коригування, інформування\_пацієнта, запис\_результатів\_обстеження\}.$$

Цей математичний опис проводився з метою проєктування моделі даних, яка допоможе сформувати метод захищеного зберігання даних.

2.3 Проєктування моделі даних

Для дослідження предметної області було взято реальне направлення на додаткове обстеження до офтальмолога, яке видане 10.08.2022 у ЦПМСД №2 у м. Вінниця. Це направлення містить такі атрибути:

- № направлення – унікальний ідентифікатор направлення.
- Спеціаліст – лікар, який виписав направлення.
- Назва обстеження – найменування обстеження чи процедури.
- ПІБ пацієнта – ініціали пацієнта.
- Попередній діагноз – попередній медичний діагноз пацієнта.
- Дата виписування направлення – дата, коли було видано направлення.
- Пріоритет – ступінь важливості або терміновості обстеження.
- Термін придатності – термін, до якого направлення є дійсним.
- № медичної картки пацієнта – унікальний ідентифікатор медичної картки пацієнта.
- Код послуги – уніфікований код для обстеження чи процедури.
- № паспорта лікаря – унікальний ідентифікатор лікаря, який виписав направлення.

– Найменування закладу охорони здоров’я – назва медичного закладу, де було сформоване направлення.
– Код за ЄДРПОУ/РНОКПП – ідентифікаційний код закладу охорони здоров'я відповідно до реєстрів.

Ці атрибути утворюють комплексну систему відстеження і контролю за наданням медичних послуг у сфері сімейної медицини в лікарні.

Відповідно до атрибутів предметної області, у результаті проєктування було побудовано реляційну модель даних, яка містить такі сутності:

– інформація про пацієнта;
– інформація про направлення;
– інформація про медичний заклад.

Доцільно буде навести діаграму зв’язків між цими сутностями та їх атрибутами (рис. 2.1).

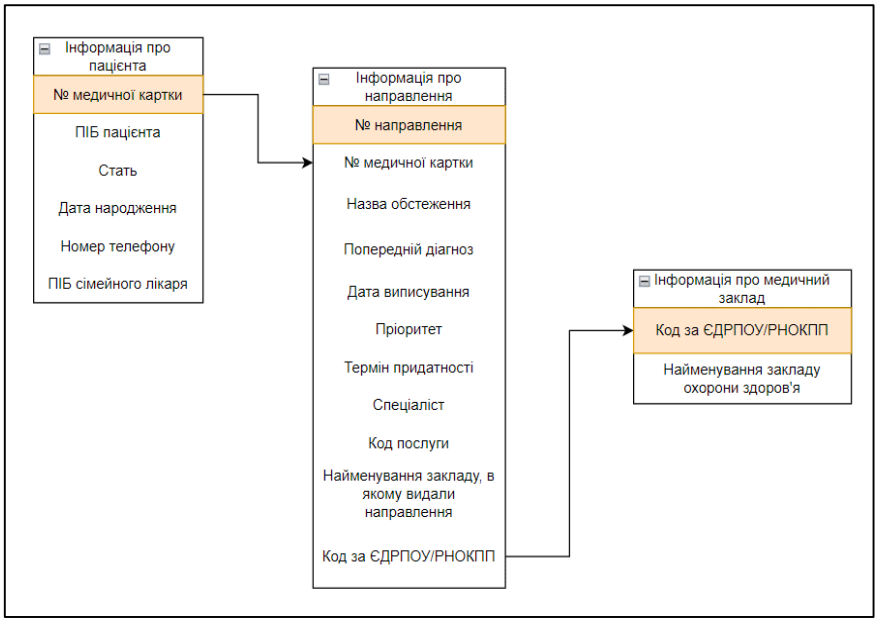


Рисунок 2.1 – Реляційна модель даних

Наведена на рис. 2.1 база даних є нормалізованою до нормальної форми Бойса-Кодда. Відношення знаходиться в нормальній формі Бойса-Кодда, якщо вся

надмірність на основі функціональної залежності була видалена, хоча інші типи надмірності можуть все ще існувати [25].

Наступним кроком необхідно проаналізувати вимоги до безпеки атрибутів в спроєктованій моделі даних.

2.4 Аналіз вимог до атрибутів

Для того, щоб визначити вимоги до безпеки атрибутів в моделі даних, потрібно проаналізувати усі сутності та їх атрибути, які були визначені у попередньому підрозділі.

Доцільно буде проаналізувати кожну сутність та її атрибути окремо та представити результати у вигляді таблиць.

Кожен пацієнт має свою медичну картку, де зберігається вся інформація, пов’язана зі станом його здоров’я. Таким чином, головним атрибутом сутності Інформація про пацієнта є № медичної картки, який потребує захисту цілісності та конфіденційності. Також не менш важливими даними є ініціали пацієнта, його стать, дата народження та номер телефону, знаючи які можна в ургентних випадках дізнатись інформацію, яка є критично важливою в даний момент. Також варто зазначити, що в даній сутності є атрибут, як ПІБ сімейного лікаря, оскільки знаючи сімейного лікаря, вузькі спеціалісти знатимуть до кого звернутись задля отримання детальної інформації про стан здоров’я хворого.

№ медичної картки потребує захисту конфіденційності та цілісності, оскільки ця інформація є доступною, то захисту доступності вона не потребує.

Ініціали пацієнта та лікаря потребують лише захисту цілісності, оскільки вони не є конфіденційними.

Дата народження, номер телефону та електронна пошта потребують захисту конфіденційності та доступності, оскільки не кожен пацієнт хоче розголошувати таку інформацію. І відповідно стать пацієнта не потребує захисту жодного з трьох критерії захищеності.

Результати аналізу представлені у вигляді таблиці 2.1.

Таблиця 2.1 – Аналіз вимог до безпеки сутності «Інформація про пацієнта»

Атрибут	Захист конфіденційності	Захист цілісності	Захист доступності	Категорія
№ медичної картки	+	+	-	КЦ
ПІБ пацієнта	-	+	-	Ц
Стать	-	-	-	Не потребує підвищеного захисту
Дата народження	-	-	+	Д
Номер телефону	+	-	+	КД
Електронна пошта пацієнта	+	-	+	КД
ПІБ сімейного лікаря	-	+	-	Ц

Наступною сутністю є інформація про направлення, яка пов’язана з інформацією про пацієнта.

№ направлення потребує захисту конфіденційності, оскільки містить особисту інформацію про пацієнта. Цілісність для цього атрибута також потрібна, оскільки будь-які зміни можуть порушити його дійсність та правильність. В той же час захисту доступності цей атрибут не потребує, оскільки забезпечення захисту цього критерія захищеності не є критичним моментом.

№ медичної картки є унікальним ідентифікатором пацієнта, тому потребує захисту конфіденційності, водночас необхідно забезпечити захист цілісності, з метою уникнення модифікації, а доступність також не є критичною, аналогічно до попереднього атрибуту.

Оскільки інформація про назву обстеження може містити конфіденційну інформацію про медичні процедури або стан пацієнта, тому вона потребує захисту конфіденційності.

Цілісність важлива з метою забезпечення точності та неможливості змін, які можуть спотворити значення.

Доступність цієї інформації важлива для медичного персоналу для правильного лікування пацієнта.

Попередній діагноз може містити чутливу інформацію, яка потребує захисту конфіденційності. цілісність цієї інформації не так критична, оскільки вона може змінюватися з часом. Доступність цього поля важлива для медичного персоналу для надання належного лікування.

Інформація про дату виписування направлення не є чутливою та не потребує спеціального захисту конфіденційності, однак необхідно їй забезпечити захист цілісності, щоб ця дата залишалася недоступною для змін без належної авторизації, щоб уникнути маніпуляцій з інформацією. Доступність цієї інформації не є критичною.

Пріоритетність направлення не є конфіденційною інформацію та не потребує захисту цілісності, однак потребує захисту доступності, оскільки доступність цього поля важлива для медичного персоналу для організації лікування в порядку пріоритетів.

Термін придатності направлення потребує лише захисту доступності, а захисту інших критеріїв захищеності даний атрибут не потребує.

Інформація про спеціаліста, який проводить обстеження потребує захисту одночасно трьох критерії захищеності, оскільки ця інформація може містити особисті дані про медичного працівника, які потребують конфіденційного захисту, забезпечення цілісності для інформації про спеціаліста важливе, щоб уникнути неправомірної зміни цих даних, доступність інформації про спеціалістів важлива для пацієнтів та медичного персоналу для правильного призначення та організації медичної допомоги.

Код послуги не є конфіденційною інформацією, оскільки не містить особистих даних. Доступність цієї інформації важлива для медичного персоналу для правильного кодування медичних послуг, а захисту цілісності дана інформація також не потребує.

Найменування закладу охорони здоров’я потребує захисту лише цілісності, в той час, як код за ЄДРПОУ/РНОКПП потребує захисту доступності, оскільки доступність цієї інформації важлива для правильного ідентифікації медичного закладу та його реєстрації.

Результати аналізу вимог до безпеки атрибутів цієї сутності наведений в таблиці 2.2.

Таблиця 2.2 – Аналіз вимог до безпеки атрибутів сутності «Інформація про направлення»

Атрибут	Захист конфіденційності	Захист цілісності	Захист доступності	Категорія
№ направлення	-	+	+	ЦД
№ медичної картки	+	+	-	КЦ
Назва обстеження	+	+	+	КЦД
Попередній діагноз	+	-	+	КД
Дата виписування направлення	-	+	-	Ц
Пріоритет	-	-	+	Д
Термін придатності	-	-	+	Д
Спеціаліст	+	+	+	КЦД
Код послуги	-	-	+	Д
Код за ЄДРПОУ/РНОКПП	-	-	+	Д

Наступним кроком було розглянуто сутність з інформацією про медичний заклад.

Відповідно до аналізу попередньої сутності, було з’ясовано, що код за ЄДРПОУ/РНОКПП потребує лише захисту доступності, а найменування закладу охорони здоров’я потребує захисту цілісності.

Результати аналізу наведені в таблиці 2.3.

Таблиця 2.3 – Аналіз вимог до безпеки атрибутів сутності «Інформація про медичний заклад»

Атрибут	Захист конфіденційності	Захист цілісності	Захист доступності	Категорія
Код за ЄДРПОУ/РНОКПП	-	-	+	Д
Найменування закладу ОЗ	-	+	-	Ц
Адреса закладу ОЗ	-	-	-	Не потребує підвищеного захисту

Таким чином, було проаналізовано кожну із сутностей та її атрибути.

2.5 Узагальнена архітектура програмного засобу

Узагальнена архітектура засобу є клієнт-серверною, оскільки відбувається зв’язок із сервером. Відбувається розділення на серверну та клієнтську частину з метою поліпшення роботи окремих модулів програмного засобу, забезпечуючи їм незалежне один від одного проєктування.

Розділення на серверну та клієнтську частини дозволяє створювати кожен компонент окремо. Кожен блок має власний функціонал, що полегшує розподіл завдань між командами розробників і підвищує швидкість розробки. Якщо в одному компоненті виникає помилка, це не руйнує роботу інших компонентів або системи в цілому. Це полегшує виявлення та усунення помилок і підвищує стійкість системи.

Чутливі дані та операції можуть бути ізольовані саме на серверному рівні, щоб запобігти несанкціонованому доступу через клієнтську частину. Також відокремлення сховищ збереження даних від клієнтської частини допомагає уникнути прямих запитів та несанкціонованого доступу до даних, навіть якщо серверна частина скомпрометована.

Побудова такої архітектури сприяє кращому розумінню роботи програмного засобу, його структури та модулів. Кожен із блоків має свою окрему функцію, забезпечуючи кращий рівень безпеки, масштабованість та стійкість до відмов. Щодо останнього параметру, то стійкість до відмов забезпечить неперервну роботу засобу та стійкість.

На рисунку 2.2 зображено узагальнену архітектуру автоматизованої системи видавання направлень.

Модуль клієнта складається з трьох блоків: графічного інтерфейсу (UI), модуля автентифікації клієнта та модуля видавання направлень. В даній роботі буде розглянуто взаємодію модуля видавання направлень з графічним інтерфейсом, в даному випадку формами, які заповнюватиме лікар з метою додавання нового пацієнта, видавання направлення та заповнення інформації про обстеження.

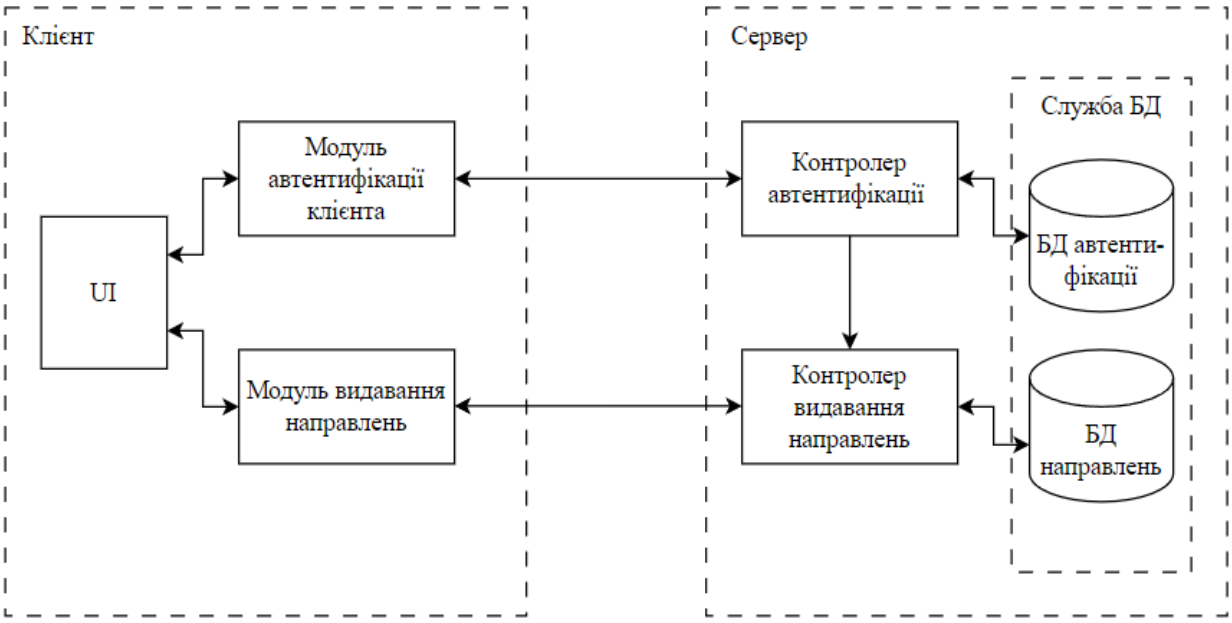


Рисунок 2.2 – Узагальнена архітектура автоматизованої системи видавання направлень на обстеження

Відповідно серверна частина має такі елементи: контролер автентифікації, який взаємодіє з модулем автентифікації на клієнтському боці, контролер видавання направлень, який аналогічним чином взаємодіє з модулем видавання направлень.

Також на серверному боці є служба БД, яка містить БД автентифікації та БД видавання направлень.

В даній роботі буде детально розглянуто модуль видавання направлень, який працює таким чином: клієнт взаємодіє із сервером, який в свою чергу містить в собі три сховища даних: блокчейн, IPFS та БД.

Узагальнену архітектуру модуля видавання направлень зображено на рис. 2.3.

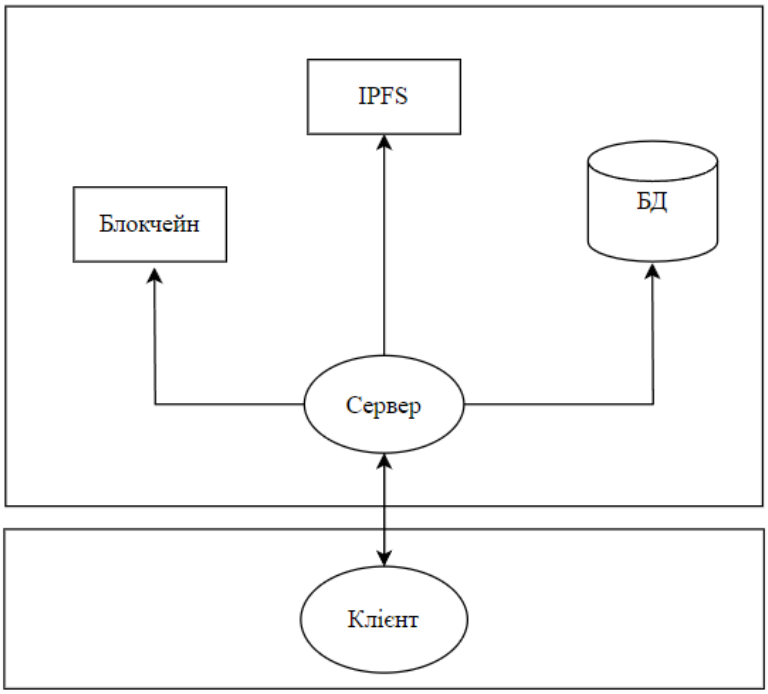


Рисунок 2.3 – Узагальнена архітектури модуля видавання направлень

Таким чином, під час роботи, лікар не помічатиме, що працює одночасно з трьома контейнерами даних, що є доказом того, що застосунок є зручним для користування.

2.6 Висновки до розділу

Таким чином, у цьому розділі було запропоновано спосіб захищеного зберігання медичних даних. Спосіб розроблявся з метою спрощення роботи медичних працівників та забезпечення захисту цілісності, доступності та конфіденційності даних, що стосуються персональних даних.

Спосіб, що запропонований в даній роботі, на відміну від відомих передбачає одночасне застосування централізованих та децентралізованих сховищ даних, а також додатковий захист цілісності персональних даних, що зберігаються в централізованих сховищах за допомогою децентралізованих, що дозволяє

покращити рівень цілісності та доступності даних, порівняно з централізованими сховищами, та підвищити рівень захисту конфіденційності та зменшення надлишковості даних, порівняно з децентралізованими сховищами.

Отже, дані, що відносяться до однієї предметної області та вимагають однакового рівня захисту, зберігаються у відокремленому контейнері (базі даних, блокчейні або в IPFS). Це дозволяє уникнути застосування заходів безпеки до даних, які цього не потребують, що допомагає більш ефективно використовувати обчислювальні ресурси та покращує масштабованість інформаційної системи. Це особливо важливо для критичних систем, таких як медичні, де довготривале зберігання даних – нормальне явище.

Було проведено математичний опис процесу видавання направлень на додаткові обстеження, з метою проєктування моделі даних, яка дозволить проаналізувати вимоги до атрибутів та визначити, яких критеріїв захищеності потребує кожен із них.

3 АЛГОРИТМ РОБОТИ ПРОГРАМНОГО ЗАСОБУ

3.1 Протокол взаємодії архітектурних складових

Оскільки в основному взаємодія відбувається поміж блокчейном та реляційною базою даних, тому доцільно представити протокол взаємодії архітектурних складових, який побудований на основі архітектури (рис. 3.1).

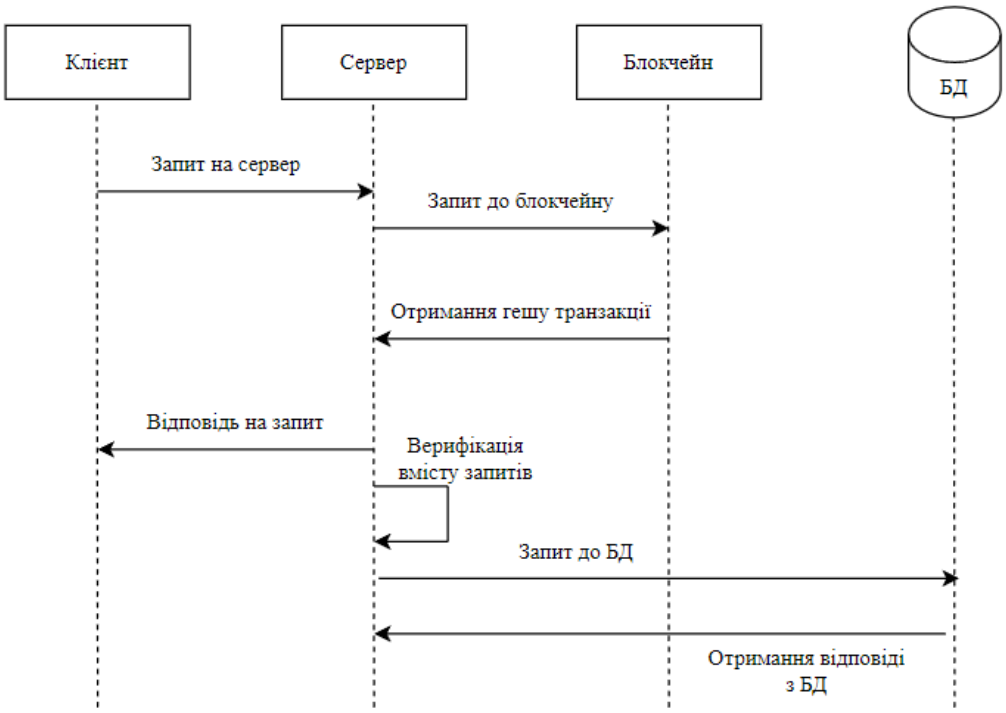


Рисунок 3.1 – Протокол взаємодії архітектурних складових

Клієнтська частина представлена веб-сайтом, через який користувачі, в даному випадку лікарі, можуть взаємодіяти з системою. Користувачі надсилають запити на сервер через веб-інтерфейс.

Сервер отримує запити від клієнтів і обробляє їх. Перш ніж переслати запит до блокчейну або бази даних, сервер перевіряє вміст запитів на наявність коректної інформації та унеможливлює порожні або некоректні запити.

Після перевірки запиту сервер відправляє його до блокчейну. Блокчейн обробляє запит, перевіряє правильність ідентифікації користувача та дозволяє виконання запиту, якщо всі умови виконані.

Деякі запити можуть бути виконані безпосередньо до реляційної бази даних. Наприклад, якщо запит стосується незначної кількості даних або потребує негайного доступу до них, сервер може виконати запит без звернення до блокчейну.

Після отримання результатів від блокчейну або реляційної бази даних, сервер формує відповідь на запит. Ця відповідь може включати необхідні дані, які користувач запитував, або повідомлення про успішне виконання запиту.

Після формування відповіді, сервер надсилає її клієнту через веб-інтерфейс. Користувач отримує результати свого запиту та може надалі взаємодіяти із системою відповідно до отриманої інформації.

3.2 Алгоритм роботи модуля клієнта

Модуль клієнта керує взаємодією з користувачем, приймає його команди, перевіряє їх, створює HTTP(S) запити і обробляє отримані відповіді.

Серед задач, які виконує клієнтська частина можна виділити такі:

– зв’язок із сервером;
– обробка даних, отриманих на основі відповідей сервера;
– взаємодія з користувачем;
– обробка запитів від користувача;
– зберігання даних;
– взаємодія з базою даних;
– взаємодія з блокчейном;
– взаємодія з IPFS;
– обробка даних, отриманих зі сховищ збереження даних.

Алгоритм роботи клієнтського модуля можна представити таким чином:

– автентифікація;
– перевірка наявності користувача;

– обирання дії на веб-сайті;

– завершення роботи та вихід із системи.

Доцільно представити узагальнений алгоритм модуля клієнта (рис. 3.2).

Спершу користувачу необхідно увійти до системи, реалізація автентифікації детально описана в другій частині комплексної бакалаврської дипломної роботи.

Після того, як лікар увійшов до системи, він має можливість обрати один із 3 пунктів меню:

– додавання пацієнта;

– видавання направлення;

– проведення обстеження;

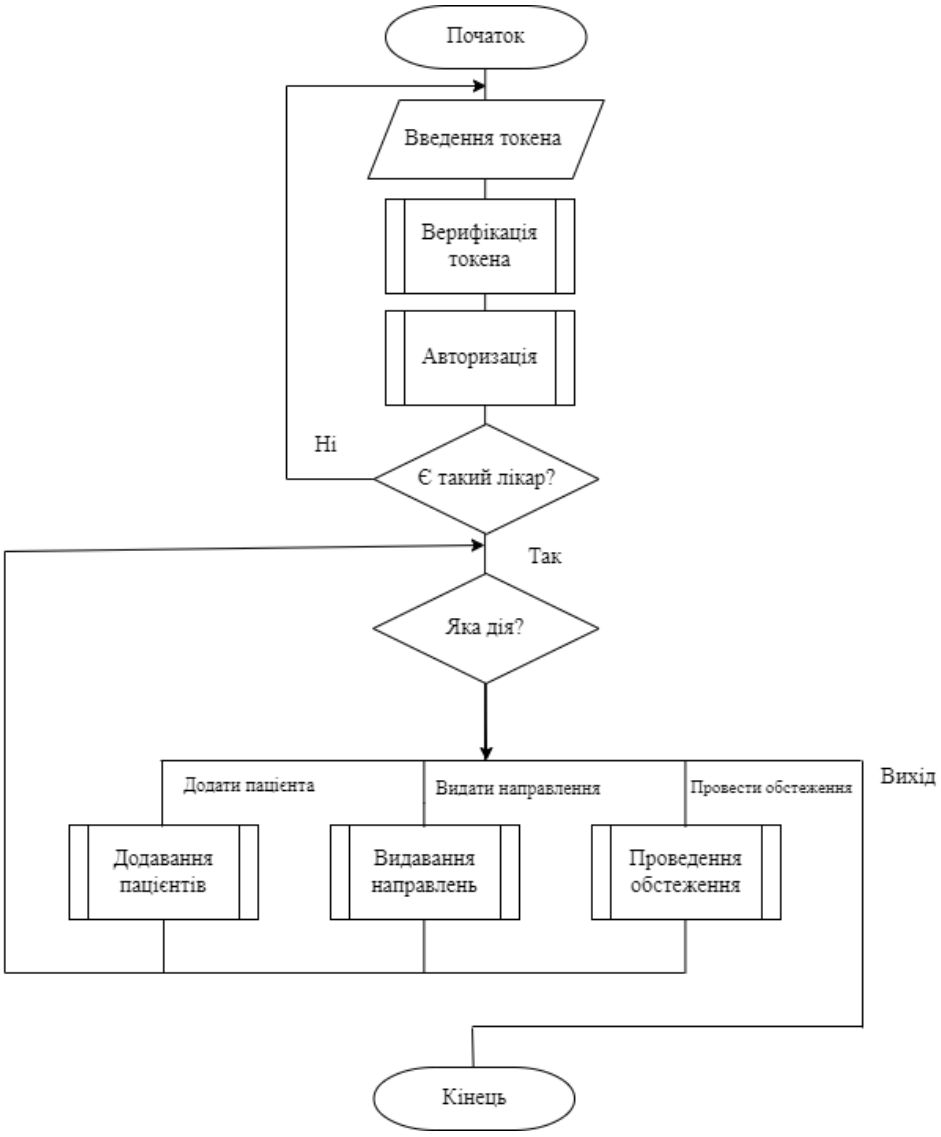


Рисунок 3.2 – Узагальнений алгоритм роботи модуля клієнта

Якщо користувач обирає пункт «Додавання пацієнта» з’являється відповідна форма, в яку можна додати інформацію про пацієнта та одразу ж після її заповнення лікар має можливість переглянути усіх своїх пацієнтів (рис. 3.3).

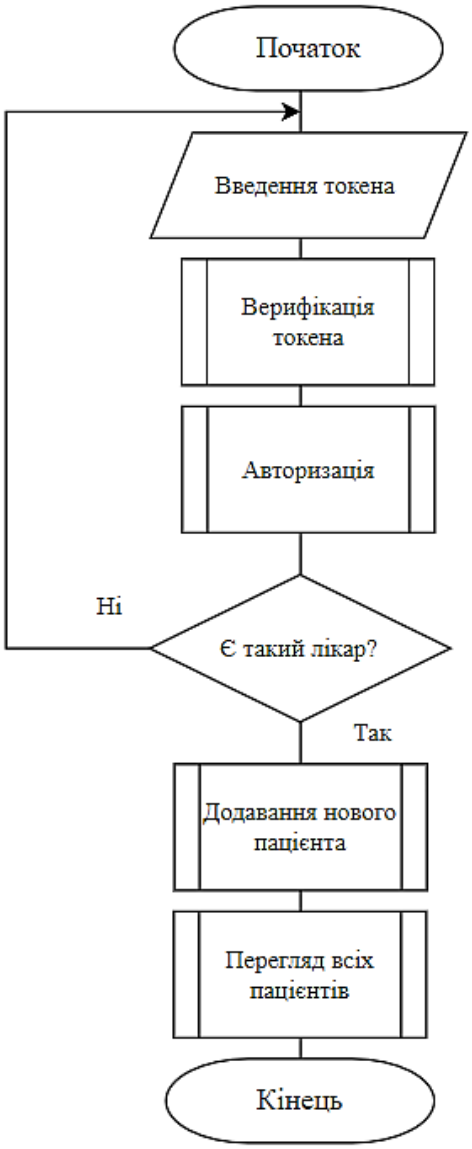


Рисунок 3.3 – Схема алгоритму роботи модуля взаємодії з пацієнтами

Алгоритм роботи модуля з видаванням направлення відображено на рис. 3.4.

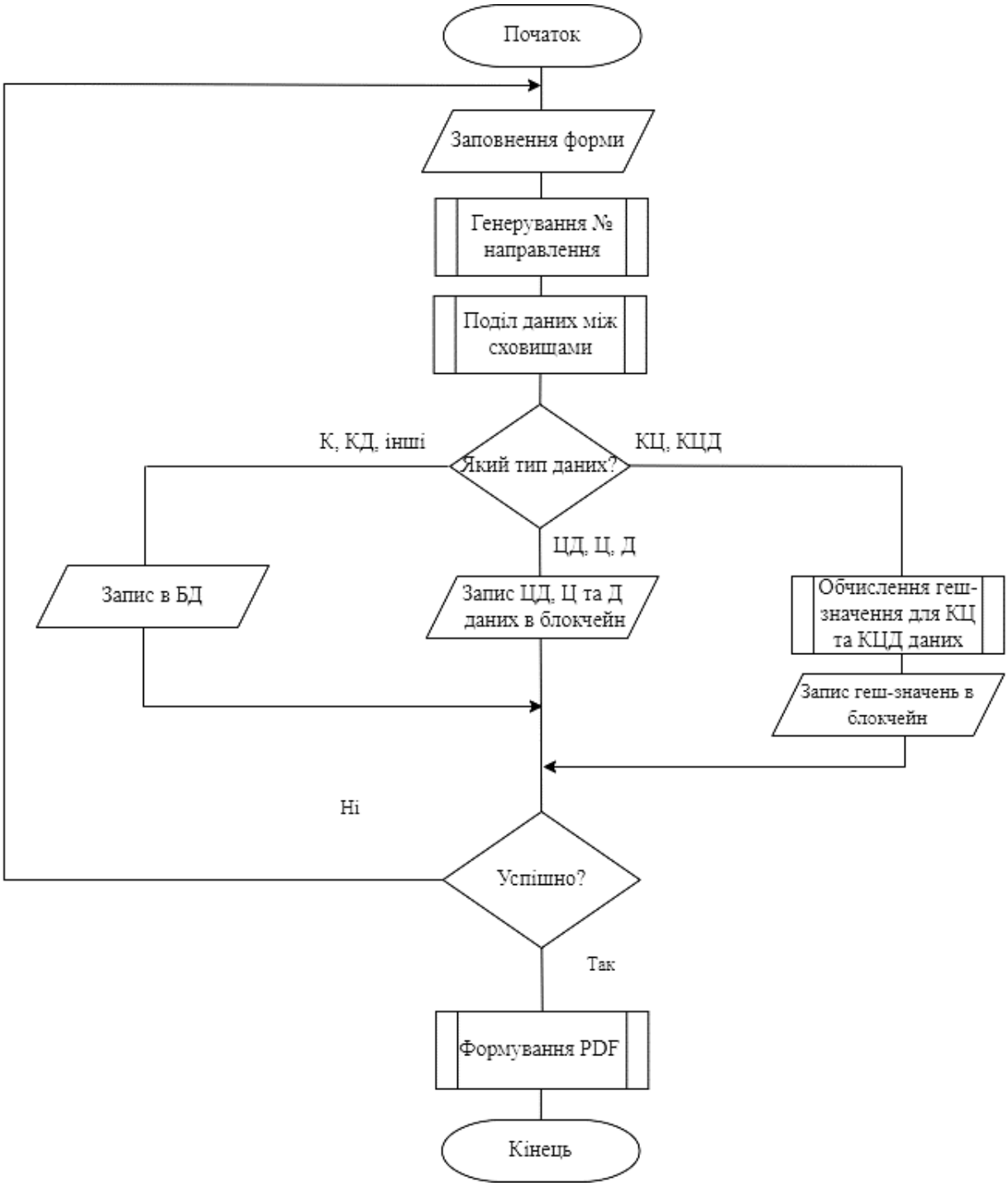


Рисунок 3.4 – Схема алгоритму роботи модуля видавання направлень

У випадку вибору пункту меню – видавання направлення, на сторінці відкривається форма для внесення даних, які необхідні для формування направлення на додаткові обстеження. Після того, як направлення буде сформовано, лікар матиме змогу завантажити файл з усіма даними, які містяться в

цьому направленні. Також розроблено систему сповіщення пацієнта через SMS та листом на пошту, де міститься номер направлення та спеціаліст, до якого направлення видано.

Як видно з рисунку 3.4, № направлення генерується системою автоматично після успішного заповнення форми. Під час заповнення форми відбувається поділ даних поміж блокчейном та базою даних: дані, що стосуються захисту цілісності та доступності – записуються в блокчейн, а також дані, що стосуються захисту конфіденційності та цілісності, а також ті, що стосуються усіх трьох критеріїв захищеності, записуються також в блокчейн, але вже у вигляді гешу. Дані, які не потрібно нікому виносити на загал, записуються в реляційну базу даних.

Коли користувач системи обирає пункт меню «Проведення обстеження», то також відкривається форма, але вже з полями, що стосуються досліджень стану здоров’я. Однак, слід зауважити, що файл з результатами обстежень буде згодом завантажено в IPFS, таким чином, необхідно передбачити знеособленість даних пацієнта, щоб уникнути витоку персональних даних.

Останнім пунктом меню веб-сайту є завантаження файлу з результатами обстеження в IPFS. Таким чином, лікар має змогу завантажити сформований файл у захищене сховище даних, з метою уникнення модифікації.

При виборі пункту «Вихід» робота завершується.

3.3 Алгоритм роботи модуля сервера

Під час створення алгоритму роботи серверної частини застосунку, необхідно звертати увагу на попередній підрозділ – алгоритм роботи модуля клієнта. Узагальнений алгоритм роботи модуля сервера представлений на рис. 3.5.

На узагальненому алгоритмі роботи модуля сервера видно, що спершу йде верифікація користувача, який увійшов до системи. Якщо такий користувач існує, то робота модуля продовжується, в іншому випадку – припиняється. Наступним етапом є отримання HTTP(S)-запиту від користувача, це крок, на якому сервер отримує дані, надіслані користувачем. Далі відбувається створення та відповідно

надсилання відповіді на модуль клієнта. Після проведених кроків, сервер припиняє свою роботу.

Рисунок 3.5 – Схема узагальненого алгоритму роботи модуля сервера

Узагальнений опис роботи модуля обробки запитів від користувача відображений на рисунку 3.6.

Як видно з рисунку 3.6, одним із важливих моментів обробки запитів є все ж таки перевірка існування користувача в системі. Якщо користувач неіснуючий,

робота засобу починається з початку, допоки до системи не увійде справжній користувач. Таким чином, узагальнений алгоритм роботи модуля сервера починається власне з верифікації користувача.

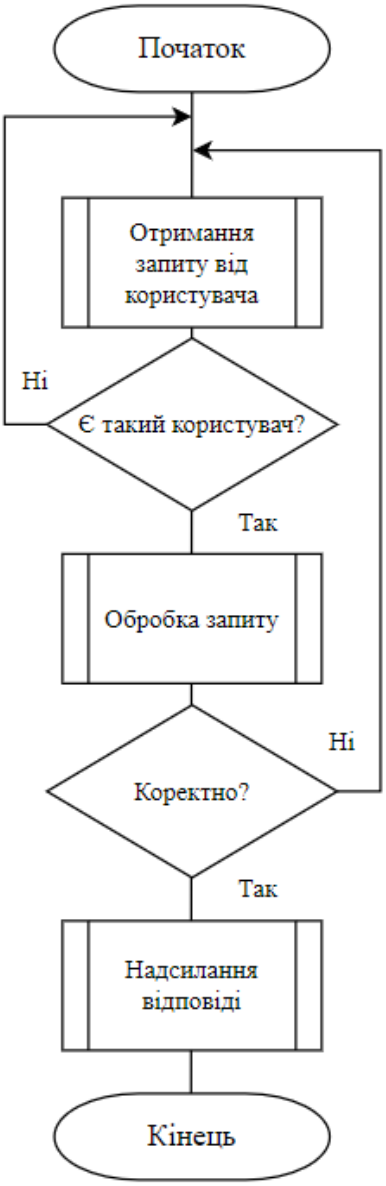


Рисунок 3.5 – Схема роботи модуля обробки запитів від користувача

Наступним етапом є аналіз роботи модуля взаємодії з базою даних.

3.4 Алгоритм роботи модуля взаємодії з базою даних

Для кращого розуміння роботи модуля взаємодії з базою даних у цьому застосунку, необхідно представити даний алгоритм у вигляді схеми (рис. 3.7).

Таким чином, спершу сервер отримує запит та відбувається власне створення запиту до БД, приєднання та пошук даних по базі даних.

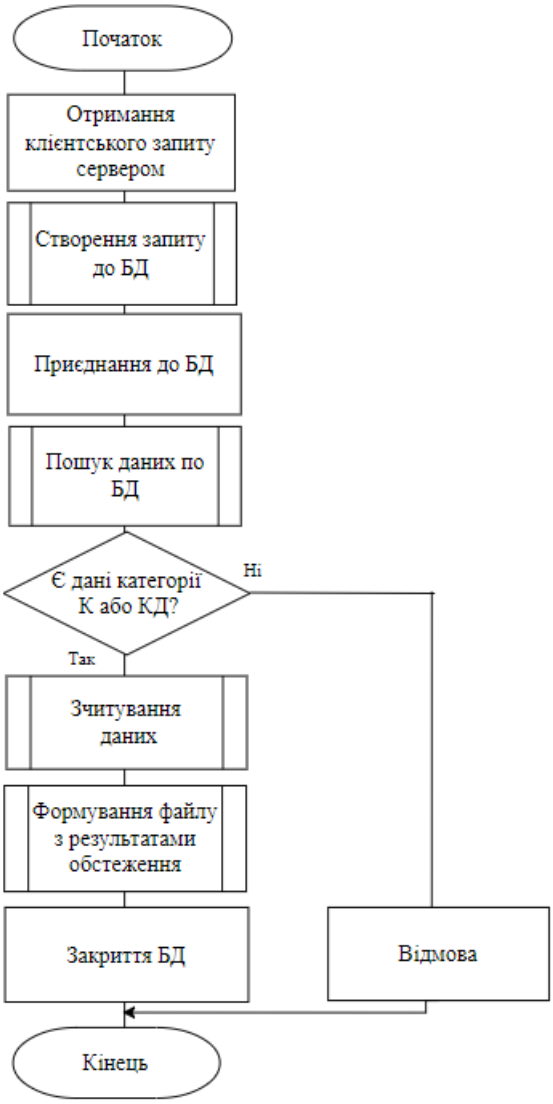


Рисунок 3.7 – Схема алгоритму роботи модуля взаємодії з базою даних

У цьому застосунку в базі даних зберігаються конфіденційні дані (дані категорії К). Дані категорії К необхідні для вузького спеціаліста, який проводить обстеження, оскільки йому необхідно дізнатись інформацію про свого пацієнта. Якщо такі дані наявні в БД, то лікар має можливість провести обстеження та в кінці відповідно створити файл з результатами, в іншому випадку – йде відмова і БД припиняє свою роботу.

Також доцільно буде представити узагальнений алгоритм роботи модуля, який відповідає за запис даних категорії К, КД та інших в БД (рис. 3.8).

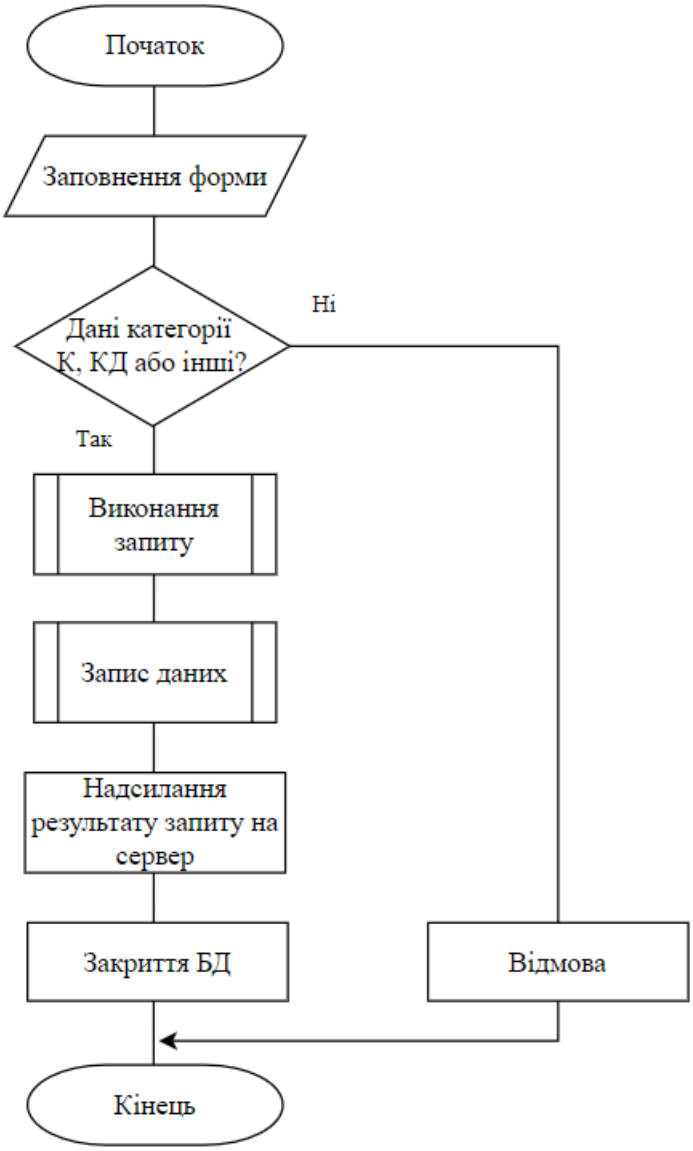


Рисунок 3.8 – Схема роботи модуля запису конфіденційних даних в базу даних

Таким чином, відповідно до рис. 3.8, якщо після заповнення форми є дані категорії К та КД, вони автоматично записуються до бази даних, в іншому випадку відмова. Дані, що не відносяться до жодної з визначених категорій – також записуються в БД. Наступним етапом є взаємодія з децентралізованими сховищами даних: блокчейном та IPFS.

3.5 Алгоритм роботи модуля взаємодії з децентралізованими сховищами

З метою забезпечення захисту цілісності та доступності застосовано децентралізовані сховища зберігання даних: блокчейн та IPFS. Взаємодія з блокчейном наведена на рис. 3.9.

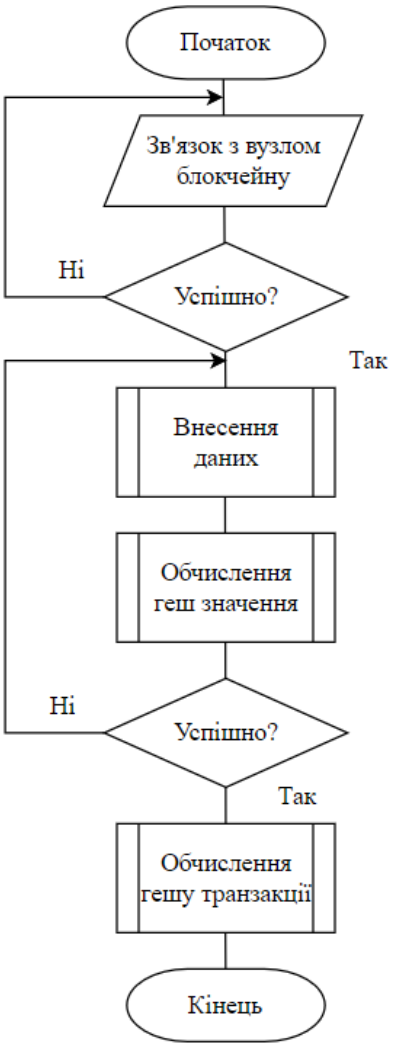


Рисунок 3.9 – Схема алгоритму роботи модуля взаємодії з блокчейном

Процес починається зі встановлення зв’язку з вузлом блокчейну. Якщо зв'язок успішно встановлений, процес продовжується, а якщо ні, то робиться нова спроба встановлення зв'язку. Після успішного встановлення зв'язку вносяться дані до системи. Далі обчислюється геш-значення для внесених даних. Якщо обчислення геш значення – неуспішне, процес повертається до внесення даних і

повторного обчислення гешу. Якщо обчислення гешу успішне, виконується обчислення гешу транзакції. Після успішного обчислення гешу транзакції процес завершується.

Аналогічним чином було розроблено алгоритм роботи модуля взаємодії з IPFS (в це сховище завантажуються файли з результатами обстежень у знеособленому вигляді). Узагальнений алгоритм наведений на рис. 3.10. Програма встановлює зв'язок з IPFS, якщо зв'язок встановлено успішно, програма запитує у користувача файл і завантажує його в IPFS. Після завантаження, обчислюється геш-значення файлу і з’являється повідомлення про успішне завершення роботи.

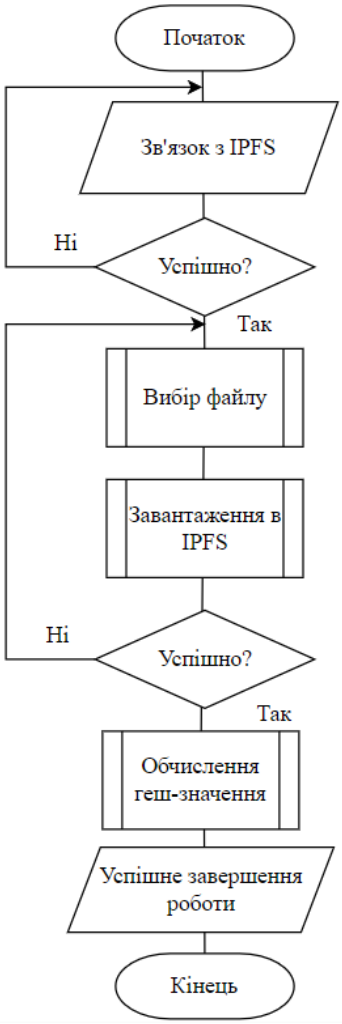


Рисунок 3.10 – Схема алгоритму роботи модуля взаємодії з IPFS

В результаті було показано у схематичному вигляді алгоритми роботи з децентралізованими сховищами даних – блокчейном та IPFS.

3.6 Висновки до розділу

У даному розділі було розроблено загальний алгоритм роботи програмного засобу, який був представлений у вигляді протоколу взаємодії архітектурних складових. Наступним етапом було розроблено алгоритми роботи модулів: клієнта та сервера. Модуль клієнта керує взаємодією з користувачем, приймає його команди, перевіряє їх, створює HTTP(S) запити і обробляє отримані відповіді. Таким чином, даний алгоритм дозволив побудувати алгоритми роботи окремо кожного з 3 модулів системи: додавання пацієнта, видавання направлення, проведення обстеження. Побудова алгоритму роботи серверної частини засобу базується на клієнтській частині, тому на основі попередніх алгоритмів було побудовано алгоритми роботи серверної частини, а саме узагальнений алгоритм роботи серверної частини та алгоритм обробки запитів від користувача.

Не менш важливим етапом є взаємодія із трьома сховищами даних, таким чином, було побудовано узагальнені алгоритми для кожного з трьох сховищ: бази даних, блокчейна та IPFS відповідно.

Таким чином, представивши у вигляді алгоритмів роботу застосунку можна перейти до його тестування.

4 ТЕСТУВАННЯ МОДУЛЯ ЗАХИЩЕНОГО ЗБЕРІГАННЯ ДАНИХ

4.1 Обґрунтування вибору засобів розробки

Вибір засобів розробки та тестування є досить важливим в процесі реалізації алгоритму та процедур автоматизованої системи видавання направлень на додаткові обстеження.

Visual Studio Code (VS Code) є одним із найбільш поширених текстових редакторів у сфері розробки програмного забезпечення. Для створення смарт-контрактів на мові Solidity у VS Code використовується спеціалізований плагін Solidity, який додає до редактора додаткові можливості [26].

VS Code підтримує велику кількість розширень, що дозволяє інтегрувати його з різними інструментами для роботи з блокчейном та смарт-контрактами. Плагін Solidity значно розширює функціональність VS Code, включаючи автодоповнення коду, перевірку синтаксису, налагодження коду та інші важливі функції, специфічні для мови Solidity [27].

Ці інструменти не тільки полегшують процес розробки, але й підвищують його ефективність та точність, дозволяючи розробникам швидко виявляти та виправляти помилки, що сприяє створенню більш безпечних смарт-контрактів.

Серед інших засобів розробки є Remix IDE [28], однак він дозволяє працювати лише зі смарт-контрактами, але оскільки в роботі ще використовуються інші мови програмування, тому дане середовище не є адекватним для даної розробки.

Truffle та Ganache – це інструменти, які використовуються для розробки, тестування та впровадження смарт-контрактів на блокчейні Ethereum [29, 30].

Truffle пропонує комплексний набір інструментів для розробки, тестування та взаємодії з мережею Ethereum. В його функціонал входить компіляція, налагодження, розгортання смарт-контрактів, а також можливість створення та виконання автоматизованих тестів для виявлення та усунення помилок.

Ganache є локальним середовищем для блокчейну Ethereum, що дозволяє розробникам створювати власні локальні блокчейни для розробки та тестування.

Використання локального блокчейну значно прискорює процес розгортання і взаємодії зі смарт-контрактами, що сприяє швидшій розробці та ефективному тестуванню.

Для взаємодії з даними великими за обсягом було обрано IPFS, який відомий унікальністю своєї технології для розподіленого зберігання та обміну даними в Інтернеті, але існують конкуруючі системи, такі як Swarm, BTFS та Storj.

Swarm [31] призначений для блокчейну Ethereum та здатний зберігати дані для додатків, але обмежений масштабуванням порівняно з IPFS. BTFS [32], хоча є похідною від IPFS, дозволяє користувачам видаляти незаконні або захищені авторським правом дані. Storj [32], інше розподілене сховище, пропонує зашифроване зберігання в хмарі, але має проблеми з масштабованістю.

Навіть з урахуванням конкурентних альтернатив, IPFS є найбільш доцільним сховищем зберігання великих обсягів медичної інформації, але потребує зберігання даних у знеособленому вигляді.

JavaScript використовується як на стороні клієнта, так і на стороні сервера. Це дозволяє розробникам створювати цілісні застосунки, використовуючи одну мову програмування [33].

JavaScript добре підтримує асинхронне програмування через колбеки, проміси і async/await, що робить цю мову зручною для роботи з веб-запитами та іншими операціями вводу-виводу.

Jest є популярним тестовим фреймворком для JavaScript [34]. Він широко використовується для тестування JavaScript та React додатків. Він не вимагає складної конфігурації і дозволяє швидко почати писати тести.

Завдяки паралельному запуску тестів, Jest значно скорочує час тестування великих проектів. Він також виконує тільки ті тести, які були змінені з останнього запуску, що ще більше прискорює процес.

Jest використовує технології паралельного запуску тестів, що значно підвищує швидкість тестування, особливо в великих проектах. Крім того, Jest

підтримує інтелектуальне кешування тестів, що ще більше прискорює повторні запуски тестів.

На відміну від Jest, Mocha [35] та Chai [36] вимагають додаткових бібліотек для мокінгу, знімків та інших функцій, що може ускладнити налаштування і використання. Mocha не підтримує паралельний запуск тестів, що може вплинути на продуктивність.

Slither [37] – це статичний аналізатор коду для смарт-контрактів, написаних на мові програмування Solidity. Він допомагає розробникам виявляти вразливості та помилки в коді на ранніх стадіях розробки, що підвищує загальну безпеку смарт-контрактів.

У сфері систем керування базами даних необхідно використовувати ефективні інструменти для взаємодії з базами даних і керування ними.

Під час огляду інструментів розробки та керування базами даних, для порівняння, було обрано SQL Server Management Studio [38], SQLite [39] та MySQL Workbench [40]. Вибір зупинився на SQL Server Management Studio, оскільки це середовище є зручним для використання, має доступний інтерфейс та багато функцій, які дозволяють взаємодіяти з базами даних.

Таким чином, після обґрунтування засобів для розробки можна перейти власне до тестування розробленого програмного засобу.

4.2 Статичне тестування безпеки

Статичне тестування безпеки (SAST) відіграє важливу роль у процесі розробки смарт-контрактів, забезпечуючи раннє виявлення потенційних загроз безпеці. SAST проводить глибокий аналіз вихідного коду без його виконання, виявляючи такі вразливості, як помилки в контролі доступу, недоліки в захисті даних і можливі витоки інформації.

Використання цього підходу дозволяє розробникам вчасно виправляти проблеми, що знижує ризик атак і неправильного використання контрактів у майбутньому.

Для тестування безпеки смарт-контрактів було обрано фреймворк Slither, вихідний код якого написаний на мові Python. Даний фреймворк дозволяє виконати детальний аналіз коду смарт-контракту, до того ж, на відміну від інших фреймворків, Slither після тестування дає вказівки для покращення коду та пояснює причину виявленого недоліку.

Результат тестування безпеки смарт-контрактів наведений на рис. 4.1.

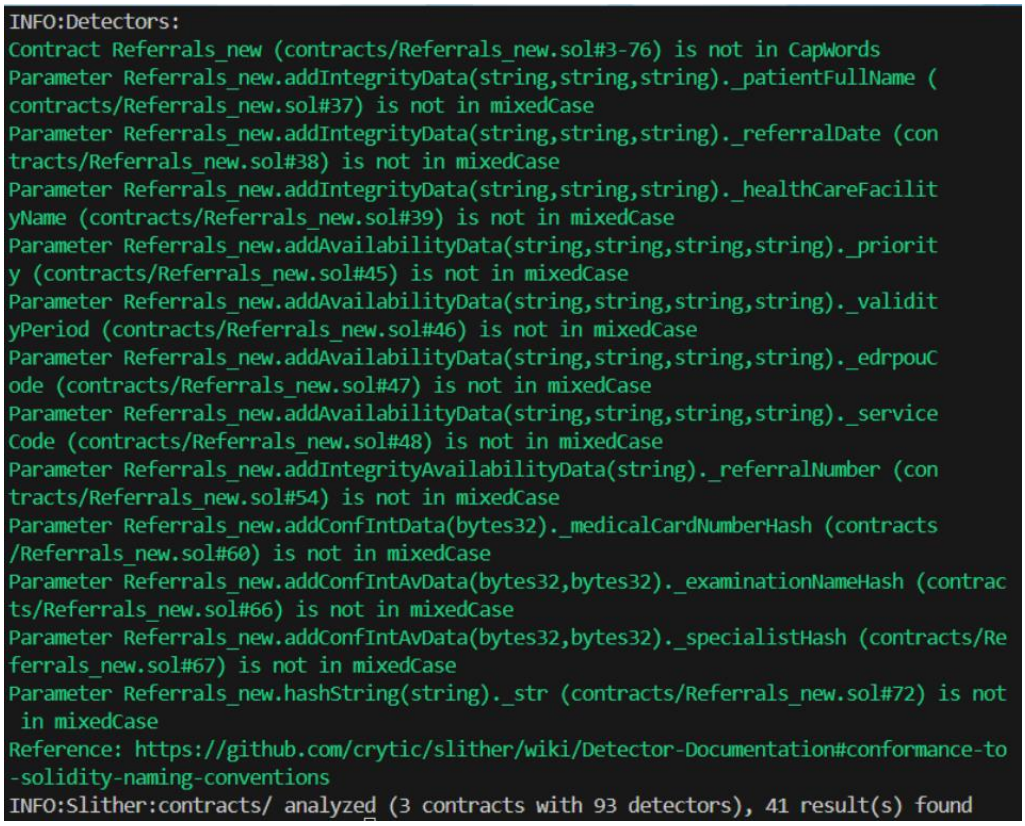


Рисунок 4.1 – Результат статичного тестування безпеки смарт-контрактів

Ще одним фреймворком для тестування безпеки є ESLint, це проєкт із відкритим вихідним кодом, який допомагає розробникам знаходити та виправляти проблеми з кодом, написаним мовою JavaScript. ESLint статично аналізує код, щоб швидко знаходити проблеми.

Було протестовано кожну функцію, результати наведені на рисунку 4.2.

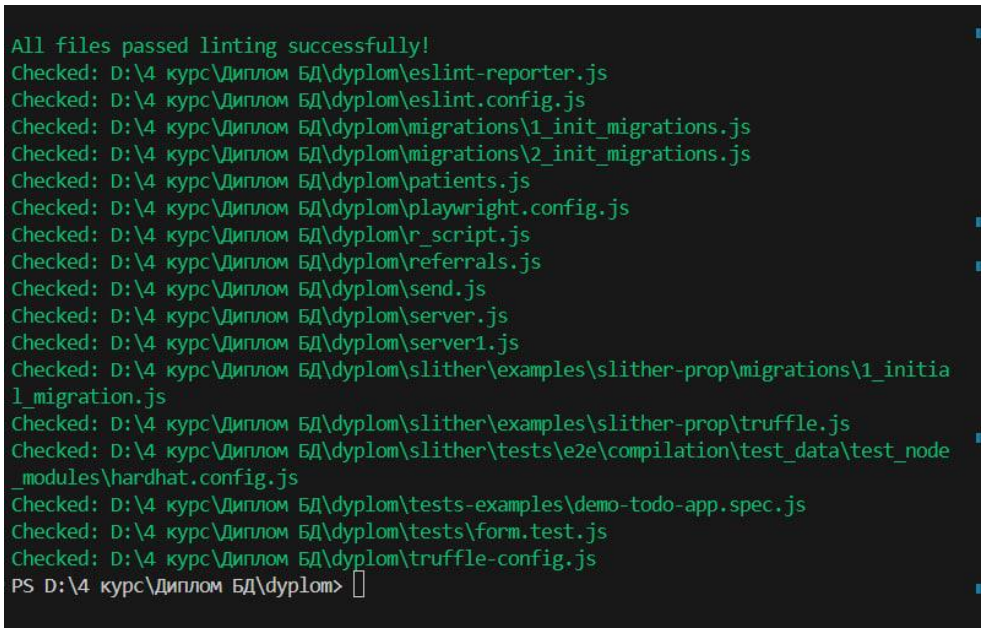


Рисунок 4.2 – Результат статичного тестування за допомогою фреймворку ESLint

Ще одним засобом для тестування безпеки є Snyk. Він пропонує як статичний аналіз коду, так і динамічне тестування безпеки додатків, забезпечуючи глибокий аналіз безпеки на всіх етапах життєвого циклу розробки. Даний аналізатор пропонує більш глибокий аналіз вихідного коду.

Відповідно до рисунку 4.3, в проєкті не було виявлено вразливостей суттєвого значення.

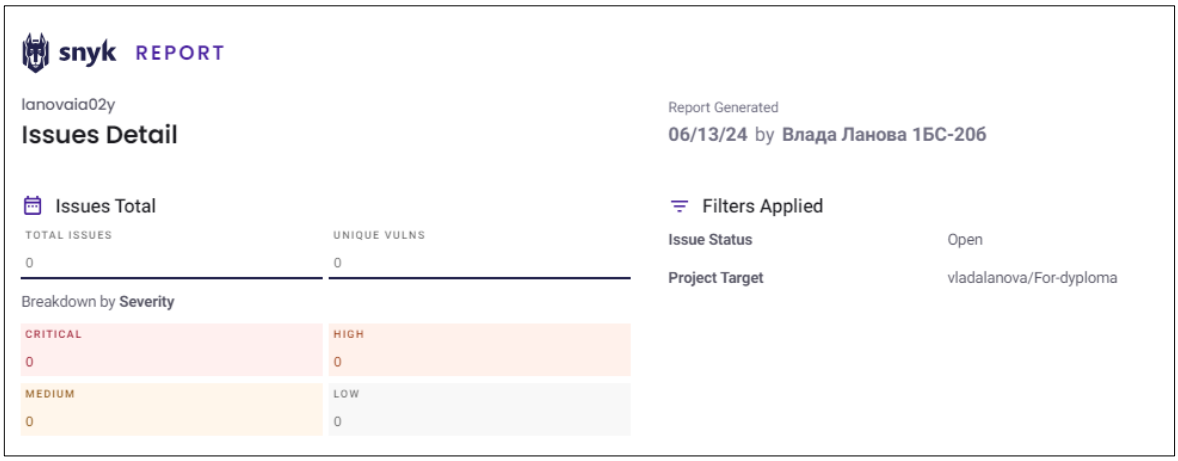


Рисунок 4.3 – Результат статичного тестування за допомогою Snyk

Наступним етапом є блокове тестування, яке допоможе перевірити основні функції, які були написані під час розробки модуля.

4.3 Блокове тестування

Основна ідея блокового тестування полягає в тому, щоб перевірити, чи працює окремий блок програмного коду (одиниця програмного коду) правильно і відповідно до очікувань.

Для тестування було обрано бібліотеку Jest, яка є зручною та доступною для використання.

Оскільки основна робота полягає у тому, що відбувається процес видавання направлень, таким чином, було розроблено позитивні та негативні тести для перевірки коректності роботи функцій для генерування направлень.

Результати позитивного тестування наведені на рисунку 4.3.

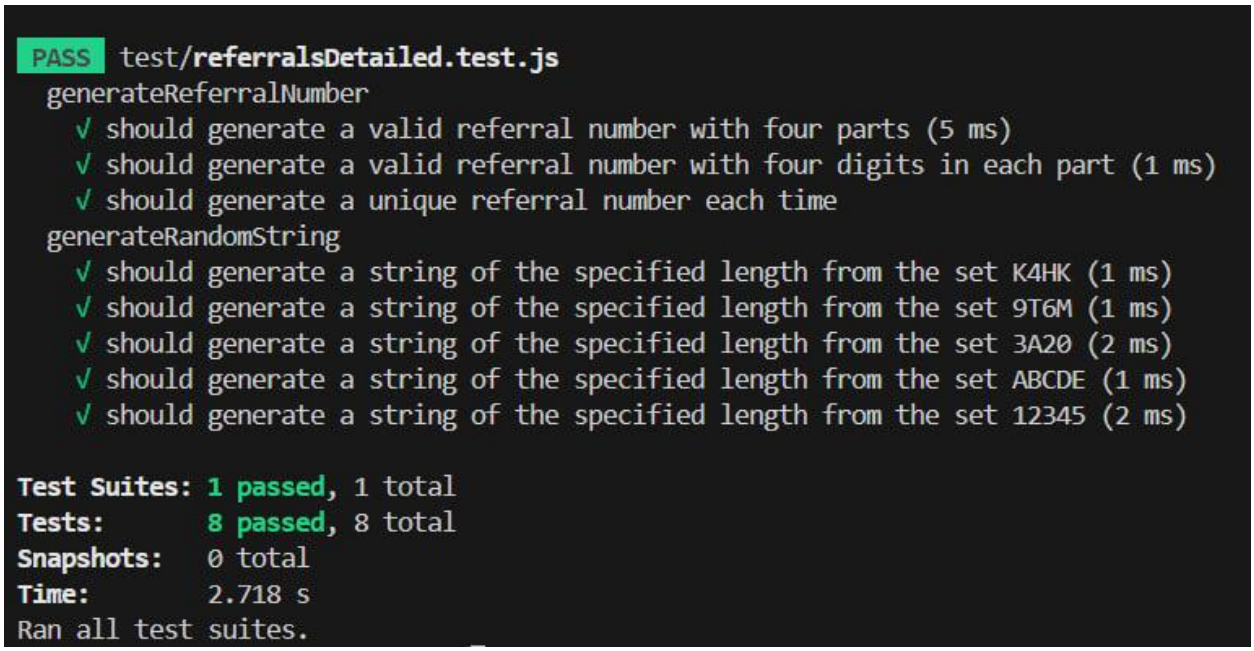


Рисунок 4.3 – Результат позитивного тестування

Наступним, не менш важливим, етапом перевірки роботи функцій є проведення негативного тестування з неправильними вхідними даними.

Було написано 5 тестів, які полягали в наступному:

– порожній набір символів: довжина = 4, набір символів = '';
– від'ємна довжина рядка: довжина = -1, набір символів = 'ABCDE';

– тест на довжину, яка рівна нулю: довжина = 0, набір символів = 'ABCDE';
– недопустимі символи в наборі: довжина = 4, набір символів = 'A B*C';
– набір символів, які не є рядком: довжина = 4, набір символів = 12345.

Всі тести пройшли успішну перевірку, результат успішного тестування наведений на рисунку 4.4.

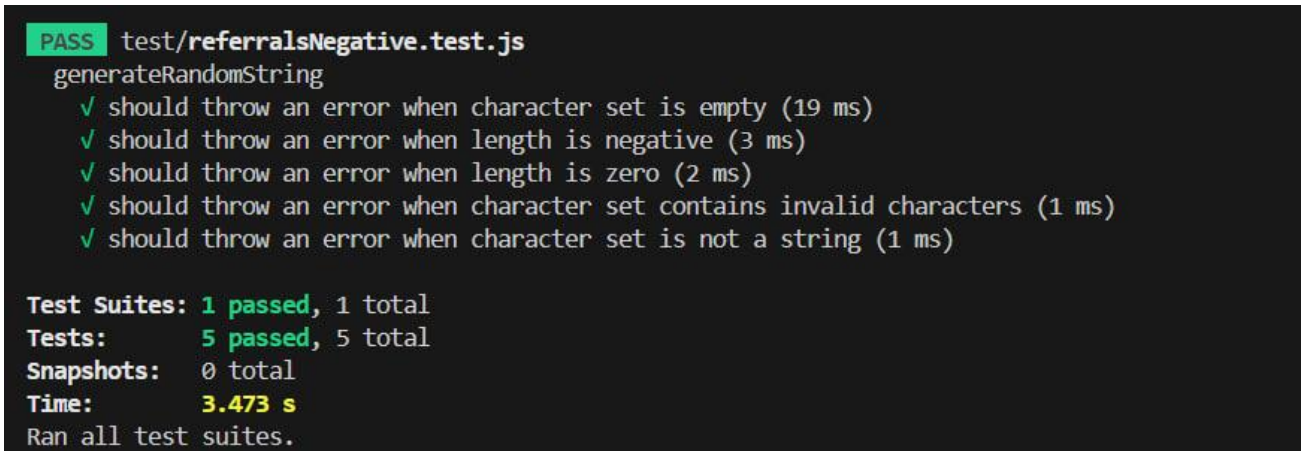


Рисунок 4.4 – Результат негативного тестування

Отже, можна висунути гіпотезу, що застосунок працює належним чином, однак слід перевірити це за допомогою інтеграційного тестування.

4.4 Інтеграційне тестування

Інтеграційне тестування полягає в тому, що відбувається перевірка наскільки добре взаємодіють різні блоки програми.

Спершу необхідно протестувати перший модуль – додавання пацієнтів.

У ролі користувача виступає сімейний лікар, який має можливість додавати пацієнтів, результати записів записуються в БД. Спершу необхідно запустити сервер та перейти за наступним посиланням:

http://127.0.0.1:3000/patients.html

Для користувача відкривається форма, яку лікар може заповнити, додавши всю необхідну інформацію про свого пацієнта (рис. 4.5).

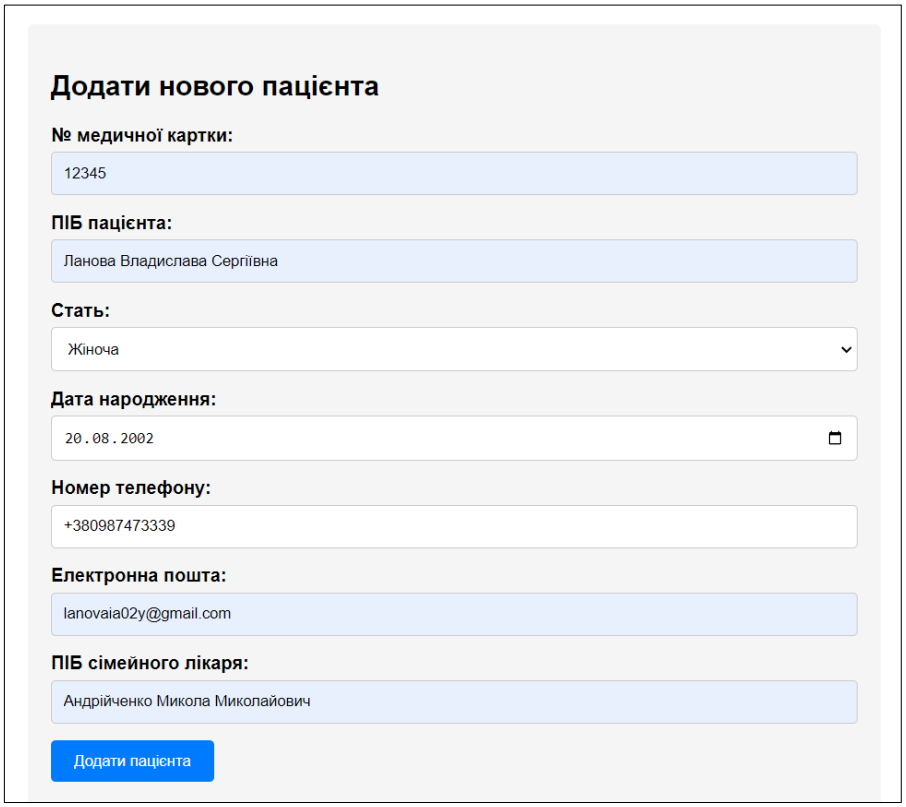
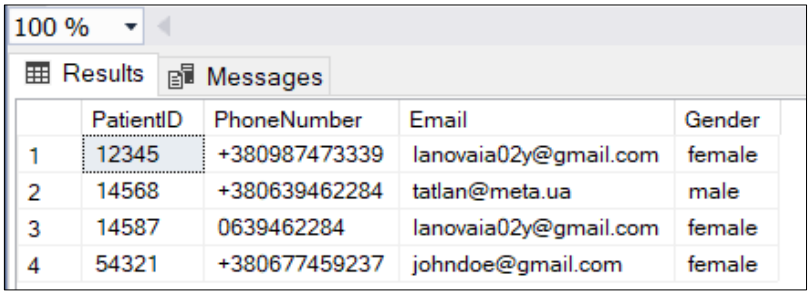


Рисунок 4.5 – Інтерфейс заповненої форми з додаванням пацієнта

Оскільки у способі було вказано, що дані, штибу ПІБ пацієнта, ПІБ сімейного лікаря та дата народження потребують захисту цілісності та доступності, які не забезпечить БД, тому в БД записались такі дані:

– № медичної картки пацієнта;
– номер телефону;
– електронна пошта;
– стать.

Результати успішного запису цієї інформації в БД наведена на рисунку 4.6.



	PatientID	PhoneNumber	Email	Gender
1	12345	+380987473339	lanovaia02y@gmail.com	female
2	14568	+380639462284	tatlan@meta.ua	male
3	14587	0639462284	lanovaia02y@gmail.com	female
4	54321	+380677459237	johndoe@gmail.com	female

Рисунок 4.6 – Результат запису в БД

Наступним етапом є додавання інформації про медичний заклад, для чого було створено та заповнено наступну форму (рис. 4.7):

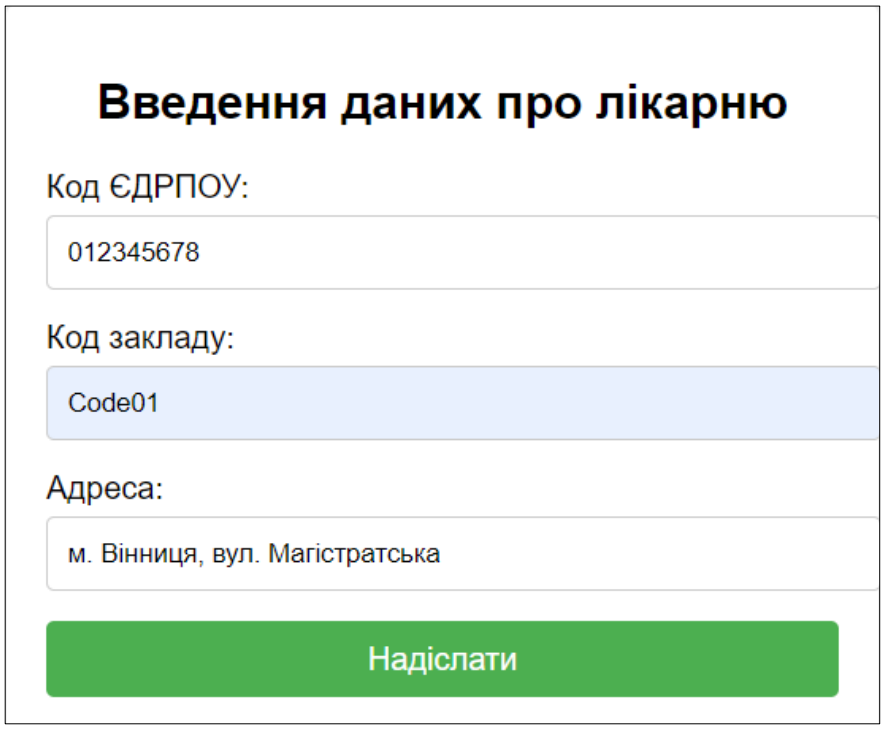
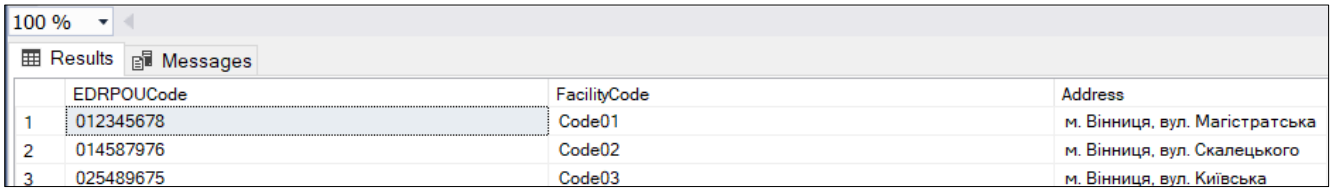


Рисунок 4.7 – Заповнена форма з інформацією про лікарню

Результати запису цієї інформації в БД наведені на рисунку 4.8.



	EDRPOUCode	FacilityCode	Address
1	012345678	Code01	м. Вінниця, вул. Магістратська
2	014587976	Code02	м. Вінниця, вул. Скалецького
3	025489675	Code03	м. Вінниця, вул. Київська

Рисунок 4.8 – Результат запису в БД

Після того, як вхідні дані вже додані, можна розглянути наступний модуль, власне який є ключовим в роботі – модуль видавання направлень.

Спершу було розгорнуто смарт-контракти в тестовій мережі Ganache. Результат успішно розгорнутих смарт-контрактів наведений на рисунку 4.9.

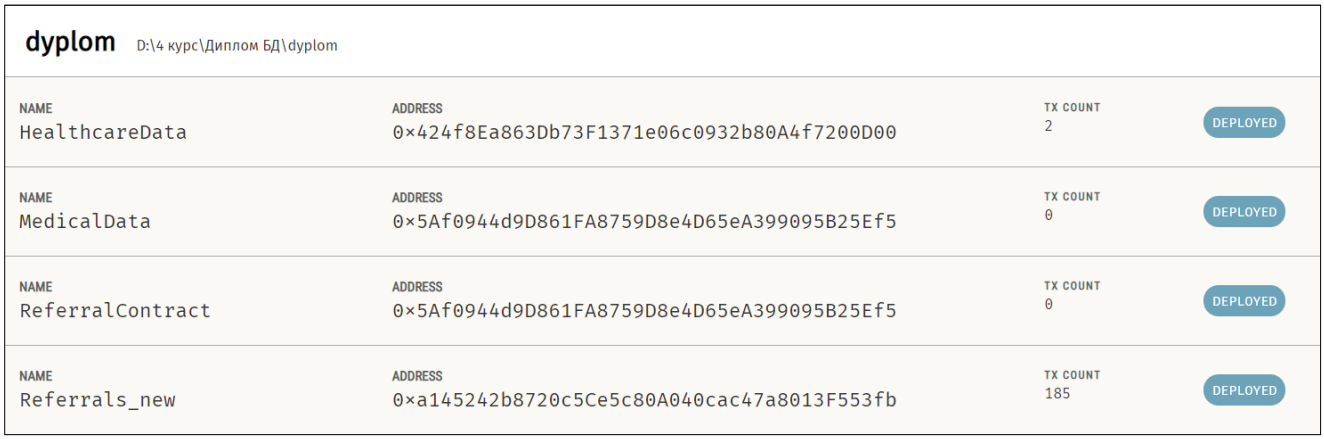


Рисунок 4.9 – Результат успішно розгорнутих смарт-контрактів

Після того, як було розгорнуто всі смарт-контракти, необхідні для взаємодії із застосунком, можна переходити власне до тестування модуля видавання направлень.

Спершу було заповнено відповідну форму для внесення даних, на основі яких буде сформовано електронне направлення (рис. 4.10).

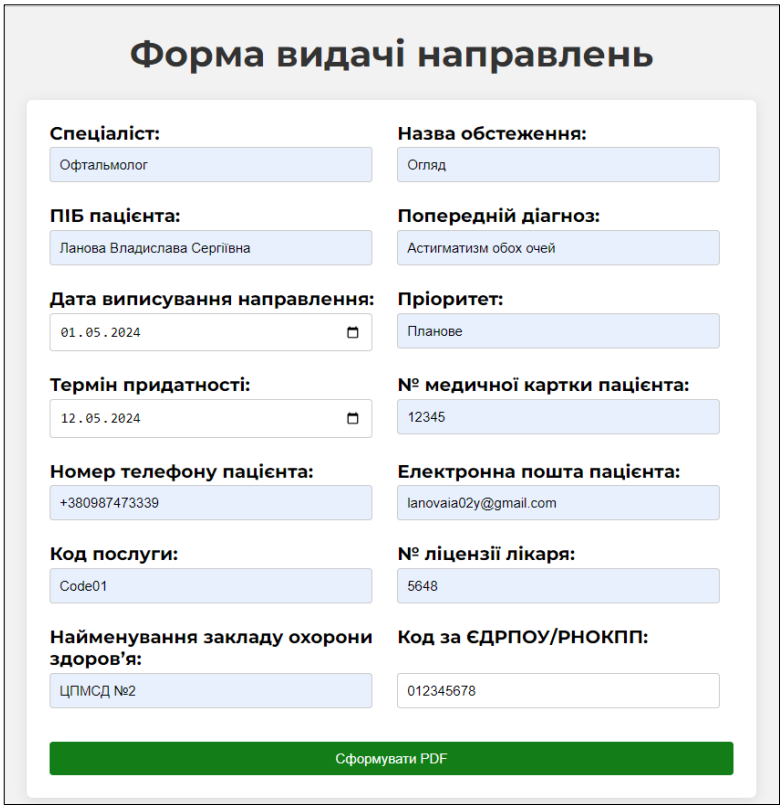
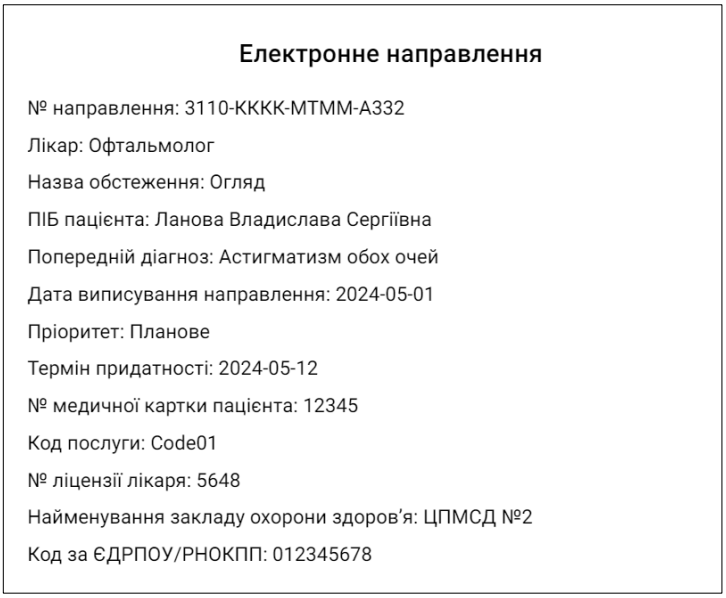


Рисунок 4.10 – Інтерфейс заповненої форми для видавання направлень

Після заповнення форми, сімейний лікар має можливість сформувати PDF-файл, де буде вся інформація про електронне направлення. Дане направлення отримує й пацієнт у надрукованому вигляді (рис. 4.11).



Електронне направлення

№ направлення: 3110-KKKK-MTMM-A332
Лікар: Офтальмолог
Назва обстеження: Огляд
ПІБ пацієнта: Ланова Владислава Сергіївна
Попередній діагноз: Астигматизм обох очей
Дата виписування направлення: 2024-05-01
Пріоритет: Планове
Термін придатності: 2024-05-12
№ медичної картки пацієнта: 12345
Код послуги: Code01
№ ліцензії лікаря: 5648
Найменування закладу охорони здоров'я: ЦПМСД №2
Код за ЄДРПОУ/РНОКПП: 012345678

Рисунок 4.11 – Фрагмент файлу з успішно сформованим направленням

На рисунку 4.11 можна побачити, що згенероване направлення містить усі поля, які були зазначені у формі, а також додався № направлення. Згенероване направлення видається пацієнту у надрукованому вигляді, з яким він йде до вузького спеціаліста на обстеження.

Також було розроблено систему сповіщення пацієнтів у вигляді надісланого листа на електронну пошту (рис. 4.12) та SMS на телефон (рис. 4.13).

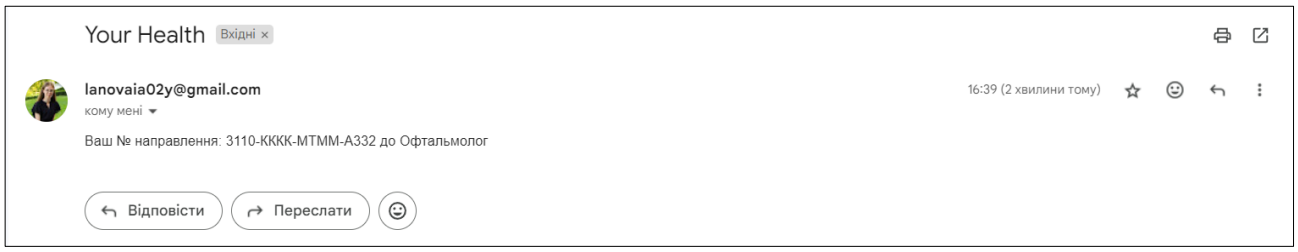


Рисунок 4.12 – Приклад роботи системи сповіщення пацієнтів у вигляді листа

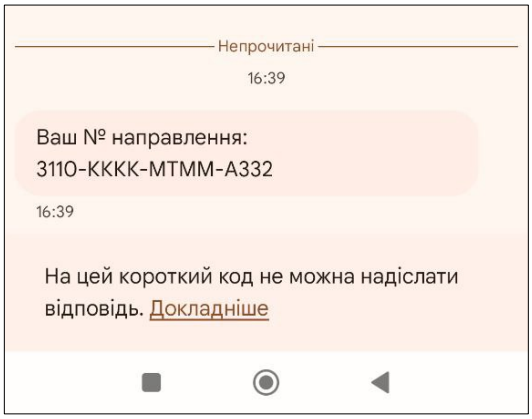


Рисунок 4.13 – Приклад роботи системи сповіщення пацієнтів у вигляді SMS

Наступним етапом тестування модуля видавання направлень є перевірка розподілу даних поміж сховищами: блокчейном та БД.

Як було вказано в способі, який описаний в 2 розділі, в блокчейн записуються дані, що потребують захисту цілісності, доступності, а також одночасно і цілісності, і доступності.

Результат доданого номеру направлення в блокчейн наведений на рис. 4.14.

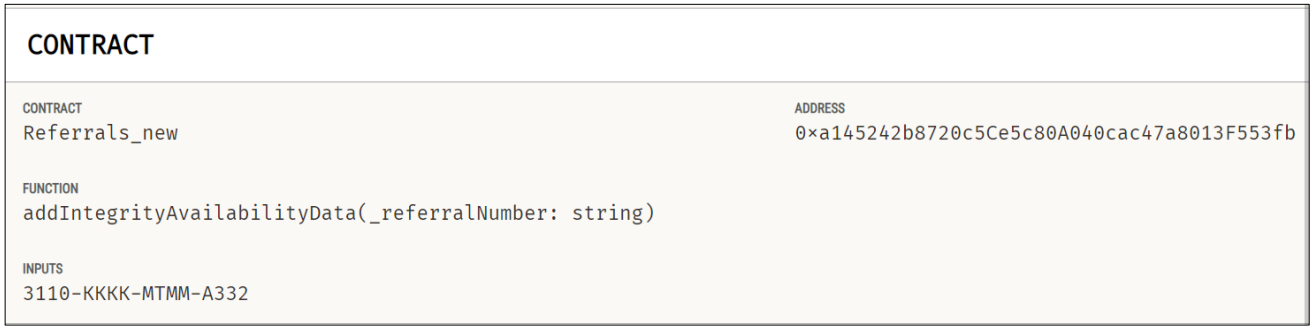


Рисунок 4.14 – Результат доданих даних в блокчейн

Також можна побачити успішно обчислені геш-значення під час запису даних, що стосуються захисту КЦ (рис. 4.15).

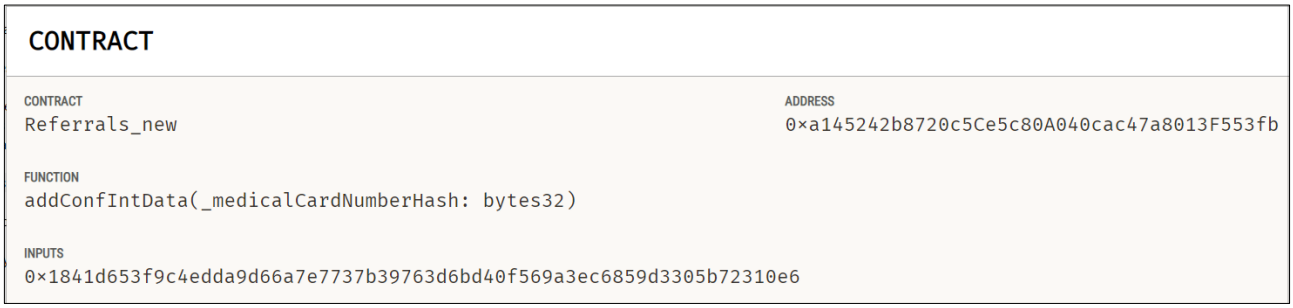
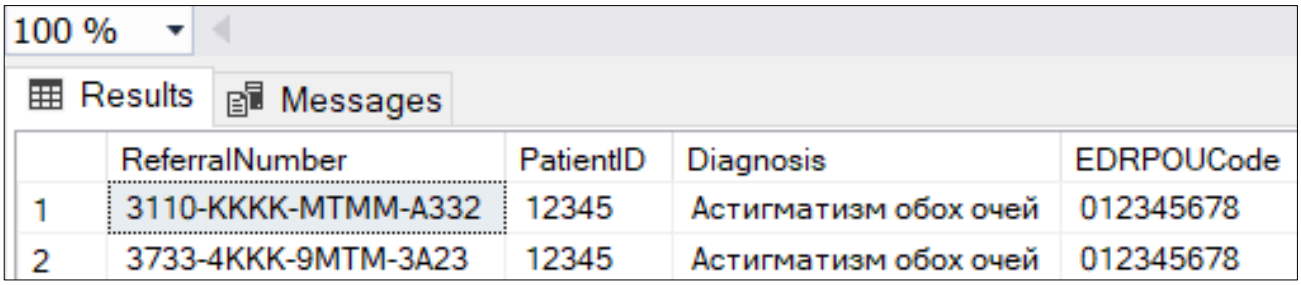


Рисунок 4.15 – Результат загешованих даних в блокчейні

Як видно з рисунку 4.15, № медичної картки пацієнта потребує захисту КЦ, таким чином, після обчислення геш-значення, даний геш записується в блокчейн.

Наступним кроком необхідно перевірити чи коректно відбувається взаємодія з БД. На рисунку 4.16 наведено результати успішно записаних даних в БД.



	ReferralNumber	PatientID	Diagnosis	EDRPOUCode
1	3110-KKKK-MTMM-A332	12345	Астигматизм обох очей	012345678
2	3733-4KKK-9MTM-3A23	12345	Астигматизм обох очей	012345678

Рисунок 4.16 – Результат успішно доданих даних в БД

Як видно з рисунку вище, один і той самий пацієнт може отримати різні направлення до одного лікаря, якщо в цьому є потреба. № направлення щоразу буде інакшим і термін дії буде також різним.

Далі, не менш важливим етапом під час візиту лікаря є проведення обстеження. Після того, як пацієнт отримав направлення до вузького спеціаліста, в даному прикладі офтальмолога, він прямує до лікаря на обстеження, де вузький спеціаліст заповнює відповідну форму, в яку вписує всі результати обстеження.

Оскільки блокчейн не сприймає дані великі за обсягом, як було зазначено під час аналізу відомих рішень, необхідно формувати файл з обстеженнями у

знеособленому вигляді та завантажувати його у інше децентралізоване сховище даних, таке як IPFS. Результати роботи модуля взаємодії з IPFS наведені на рис. 4.17.

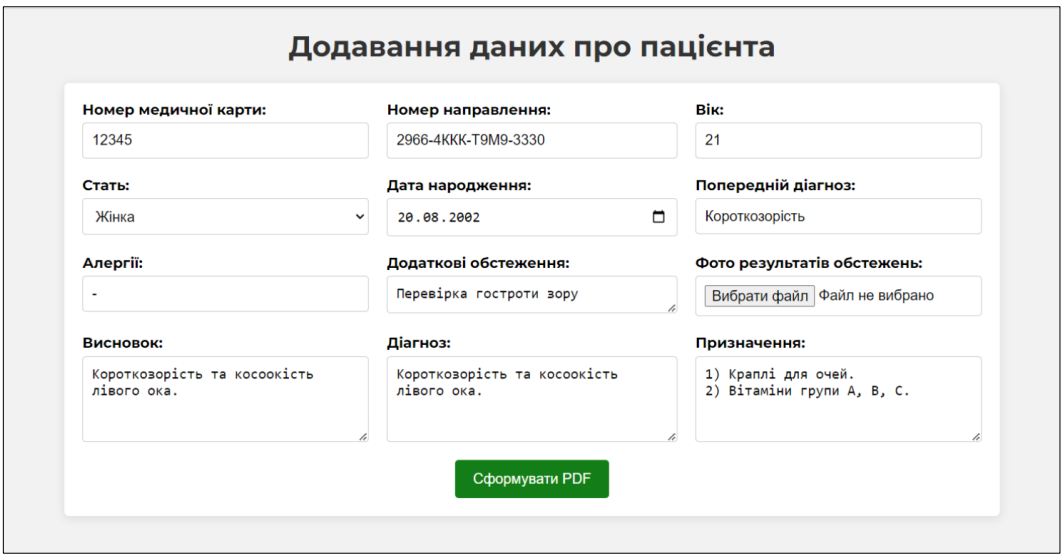


Рисунок 4.17 – Інтерфейс заповненої форми з результатами обстежень

Було розроблено зручний інтерфейс для лікаря, завдяки якому він матиме можливість взаємодіяти із файлами будь-якого формату.

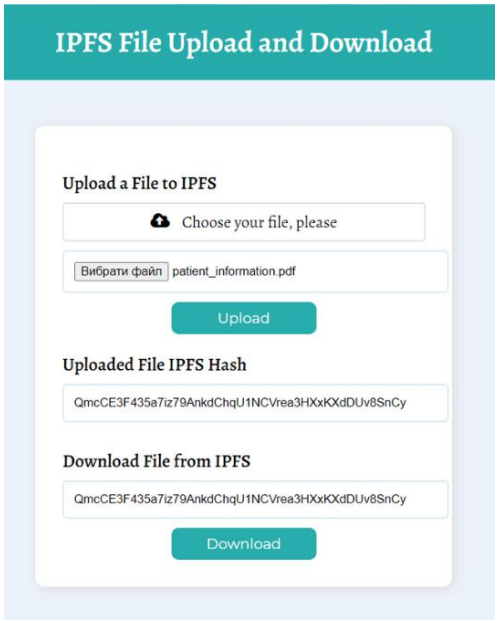


Рисунок 4.18 – Інтерфейс модуля взаємодії з IPFS

Принцип роботи модуля полягає в тому, що після того, як файл завантажився в IPFS, було обчислено його геш-значення і відповідно з цим геш-значенням і надалі будуть взаємодіяти вузький спеціаліст та сімейний лікар.

4.5 Висновки до розділу

Під час реалізації засобу необхідно звертати увагу на засоби розробки, таким чином, після аналізу було обрано мову програмування JavaScript, яка є скриптовою мовою та зручною для написання як клієнтської, так і серверної частин застосунку.

Для реалізації смарт-контрактів для блокчейну Ethereum було застосовано мову Solidity, яка забезпечує статичну типізацію, підтримує механізми успадкування та використання різних бібліотек.

Не менш важливим етапом розробки є тестування програмного засобу, було проведено статичне тестування безпеки, блокове та інтеграційне тестування.

Статичне тестування безпеки необхідне для того, щоб перевірити чи код написаний коректно та чи не містить він суттєвих вразливостей, таким чином, за результатами тестування смарт-контрактів за допомогою фреймворку Slither, суттєвих недоліків не було виявлено. А протестувавши увесь проєкт за допомогою ESLint також не було виявлено недоліків. Таким чином, дана розробка відповідає усім вимогам безпеки.

Блокове тестування дозволило перевірити основні функції, для тестування було обрано фреймворк Jest. Написавши як позитивні, так і негативні тести було зроблено висновок, що все працює належним чином.

Останнім кроком було проведено інтеграційне тестування всіх модулів програмного засобу, де також було зроблено висновок, що застосунок є зручним для лікарів та має доступний та зрозумілий функціонал.

ВИСНОВКИ

Проведений аналіз нормативно-правової бази підтвердив актуальність захисту інформації в галузі медицини. Законодавчі вимоги накладають обов'язок на медичні заклади щодо дотримання високих стандартів безпеки даних. Недотримання цих вимог може призвести до значних штрафів або навіть до закриття закладу. Дослідження нормативно-правової бази інших країн дозволило визначити тенденцію до збільшення регулювання процесів обробки інформації в медицині, що підкреслює важливість та актуальність розробок у цій галузі.

Аналіз відомих рішень, щодо видавання електронних направлень на додаткові обстеження показав, що через те, що доступ до цих даних має велика кількість людей – це може спричинити зловживання інформацією та її несанкціоновану модифікацію. Відомі практики, де всі дані записуються в один контейнер (блокчейн), мають недолік загальнодоступності та обмеженої масштабованості.

Таким чином, для досягнення мети комплексної бакалаврської дипломної роботи було запропоновано спосіб захищеного зберігання даних, який передбачає одночасне використання централізованих та децентралізованих сховищ даних. Відповідно до запропонованого способу усувається єдина точка відмови, тому навіть якщо буде кібератака, автоматизована система продовжить надалі працювати.

Для того, щоб спроєктувати модель даних процесу видавання електронних направлень на додаткові обстеження було спершу проведено математичний опис, завдяки якому було визначено вхідні дані для проєктування. Після зазначених кроків, було проаналізовано атрибути відповідно до захисту цілісності, доступності та конфіденційності та відповідно до способу було віднесено їх до визначених сховищ збереження даних: бази даних, блокчейна та IPFS.

Для того, щоб розробити модуль захищеного зберігання даних, необхідно визначити архітектуру програмного засобу. Було прийнято рішення розробити клієнт-серверну архітектуру, яка є зручною для розробників. Побудова такої

архітектури сприяє кращому розумінню роботи програмного засобу, його структури та модулів.

Після розробки узагальненої архітектури модуля, було розроблено алгоритми роботи кожної з його частин: клієнта, сервера, а також трьох сховищ даних.

Попередні кроки дозволили почати розробку та власне протестувати розроблений програмний засіб. Було проведено статичне тестування безпеки, яке не виявило недоліків під час розробки засобу. Було протестовано смарт-контракти та програмний код для серверної та клієнтської частин застосунку.

Блокове тестування дозволило протестувати окремі блоки програмного засобу, в даному випадку – функції, які відповідають за створення та генерацію номера направлення. Було проведено, як позитивне, так і негативне тестування, які не виявили недоліків. Це дозволяє стверджувати, що застосунок працює належним чином, а тому ухвалені технічні рішення – коректні.

Інтеграційне тестування дозволило перевірити роботу програмного засобу в цілому. Після проведення даного тестування, було з’ясовано, що модуль захищеного зберігання даних працює належним чином: дані успішно розподіляються поміж сховищами збереження даних та належним чином взаємодіють між собою.

Таким чином, можна зробити висновок, що запропонований в комплексній бакалаврській дипломній роботі підхід, сприятиме покращенню надання медичної допомоги та управління медичними процесами в сфері сімейної медицини.

Перспективою розвитку є масштабування отриманих результатів на інші процеси обробки медичних даних, а також для взаємодії з іншими медичними процесами. Також перспективним буде розвиток спеціалізованого блокчейна для завдань у галузі медицини.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. General Data Protection Regulation (GDPR), Official Journal of the European Union L 119, 04.05.2016; with cor. L 127, 23.05.2018. URL: https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32016R0679 (accessed: 06.05.2024).
2. Baryshev Y., Lanova V. Database Structure for Medical Data Protection Based on Blockchain. *Інформаційні технології та комп'ютерне моделювання*: матеріали статей Міжнародної науково-практичної конференції, м. Івано-Франківськ, 15-16 грудня 2022 року. Івано-Франківськ: п. Голіней О.М., 2022. С. 90-91.
3. Ланова В. С., Баришев Ю. В. Смарт-контракти для розподіленого зберігання медичних даних. *LII науково-технічна конференція підрозділів Вінницького національного технічного університету (НТКП ВНТУ–2023)*: зб. доповідей. Вінниця : ВНТУ, 2023. С. 867-869.
4. Ланова В. С., Баришев Ю. В. Аналіз геш-функцій для захисту цілісності чутливих даних. *LIII науково-технічна конференція підрозділів Вінницького національного технічного університету (НТКП ВНТУ–2024)* : зб. доповідей. Вінниця : ВНТУ, 2024. С. 327-331.
5. Ланова В. С., Баришев Ю. В. Підхід до застосування IPFS для захисту медичних даних. *Інформаційні технології та комп'ютерне моделювання*: матеріали статей Міжнародної науково-практичної конференції, м. Івано-Франківськ, 21-24 травня 2024 року. Івано-Франківськ: п. Голіней О.М., 2024. С. 137-138.
6. Ланова В. С., Баришев Ю. В. Модель даних автоматизованої системи видавання електронних направлень на додаткові обстеження. *Медико-технічна співпраця заради перемоги: Актуальні завдання медичної, біологічної фізики та інформатики*: Матеріали доповідей та виступів III всеукраїнської науково-практичної конференції з міжнародною участю 5-6 квітня 2024 року Вінниця. – Вінниця: Едельвейс. 230 с.
7. Ланова В. С., Клиш В. М., Баришев Ю. В. Метод захищеного зберігання медичних даних за допомогою розмежування прав доступу та блокчейну. *ITSec:*

Безпека інформаційних технологій: матеріали XIII Міжнар. наук.-техн. конф., м. Львів, 9-11 трав. 2024 р. Львів : ЛНУ ім. І. Франка, 2024, С. 115-116.

8. Ланова В. С., Баришев Ю. В. Модуль видавання електронних направлень в медичних закладах. *Theoretical and Applied Cybersecurity «TACS-2024»*: тези доп., 30-31 трав. 2024 р., Київ : КПІ ім. І. Сікорського, 2024.
9. Баришев Ю. В., Ланова В. С. Метод захищеного зберігання медичних даних на основі реляційної бази даних та блокчейну. *Наукові праці ВНТУ.* № 3. 8 с. DOI: 10.31649/2307-5376-2023-3-9-17 (дата звернення: 15.05.2024).
10. Основи законодавства України про охорону здоров’я: Закон України від 19.11.1992 р. № 2801-XII. Дата оновлення: 19.04.2024. URL: https://zakon.rada.gov.ua/laws/show/2801-12 (дата звернення: 06.05.2024).
11. Про захист персональних даних: Закон України від 01.06.2010 р. № 2297-VI. Дата оновлення: 27.04.2024. URL: https://zakon.rada.gov.ua/laws/show/2297-17#Text (дата звернення: 06.05.2024).
12. 3 штрафи за GDPR та висновки для українських компаній. URL: https://legalitgroup.com/3-shtrafi-za-gdpr-ta-visnovki-dlya-ukrayinskih-kompanij/ (дата звернення: 08.05.2024).
13. HIPAA vs Healthcare Laws and Regulations in Canada, the UK, Australia, and MENA Countries Source. URL: https://yalantis.com/blog/hipaa-vs-healthcare-laws-in-other-countries/ (accessed: 09.05.2024).
14. Деякі питання електронної системи охорони здоров'я: Постанова Кабінету Міністрів України від 25.04.2018 р. № 411. Дата оновлення: 26.03.2024. URL: https://zakon.rada.gov.ua/laws/show/411-2018-%D0%BF#Text (дата звернення: 10.05.2024).
15. Електронна система охорони здоров’я в Україні. URL: https://ehealth.gov.ua/ (дата звернення: 10.05.2024).
16. Коробцова Н. В. Електронна система охорони здоров’я (E-Health): механізм Упровадження та етапи розвитку. *Проблеми законності*. 2021. № 154. С. 117-126. DOI: 10.21564/2414-990x.154.236921 (дата звернення: 10.05.2024).

17. Sun J., Yao X., Wu Y. Blockchain-Based Secure Storage and Access Scheme For Electronic Medical Records in IPFS. *IEEE Access.* 2020. Vol. 8. Pp. 59389 – 59401. DOI: 10.1109/access.2020.2982964 (accessed: 10.05.2024).
18. Chen F., Huang J., Wang C. Data Access Control Based on Blockchain in Medical Cyber Physical Systems. *IEEE Access.* 2021. Vol. 2021. Pp. 1-14. DOI: 10.1155/2021/3395537 (accessed: 10.05.2024).
19. Chen Y., Meng L., Zhou H. A Blockchain-Based Medical Data Sharing Mechanism with Attribute-Based Access Control and Privacy Protection. *Wireless Communications and Mobile Computing.* 2021. Vol. 2021. Pp. 1–12. DOI: 10.1155/2021/6685762 (accessed: 10.05.2024).
20. Guo R., Shi H., Zhao Q. Secure Attribute-Based Signature Scheme With Multiple Authorities for Blockchain in Electronic Health Records Systems. *IEEE Access.* 2018. Vol. 6. Pp. 11676–11686. DOI: 10.1109/access.2018.2801266 (accessed: 10.05.2024).
21. Saini A., Zhu Q., Singh N. A Smart Contract Based Access Control Framework for Cloud Smart Healthcare System. *IEEE Internet of Things Journal.* 2020. Vol. 8. Pp. 5914 – 5925. DOI: 10.1109/jiot.2020.3032997 (accessed: 12.05.2024).
22. Katile U. MediChain: Medical Record Management System. International Journal for Research in Applied Science and Engineering Technology. 2023. Vol. 11, Pp. 3414 – 3417. DOI: 10.22214/ijraset.2023.52365 (accessed: 12.05.2024).
23. e-Health Record – e-Estonia. URL: https://e-estonia.com/solutions/healthcare/e-health-records/ (accessed: 12.05.2024).
24. Задачин В. М. Моделювання систем : конспект лекцій. Харків : ХНЕУ, 2010. 268 с.
25. Harrington J. Normalization. *Relational Database Design.* 2009. Pp. 103–126. DOI: 10.1016/b978-0-12-374730-3.00006-1 (accessed: 21.05.2024).
26. Visual Studio Code. URL: https://code.visualstudio.com/docs (accessed: 25.05.2024).
27. Solidity. URL: https://docs.soliditylang.org/en/v0.8.10/ (accessed: 25.05.2024).

28. Welcome to Remix’s documentation! URL: https://remix-ide.readthedocs.io/en/latest/ (accessed: 25.05.2024).
29. What is Truffle? URL: https://trufflesuite.com/docs/truffle/ (accessed: 25.05.2024).
30. What is Ganache? URL: https://trufflesuite.com/docs/ganache/ (accessed: 25.05.2024).
31. Swarm. URL: https://docs.ethswarm.org/ (accessed: 25.05.2024).
32. 7 decentralized data storage networks compared. URL: https://www.techtarget.com/searchstorage/tip/Comparing-4-decentralized-data-storage-offerings (accessed: 25.05.2024).
33. The Modern JavaScript Tutorial. URL: https://javascript.info/ (accessed: 25.05.2024).
34. Jest. URL: https://jestjs.io/ (accessed: 25.05.2024).
35. Mocha. URL: https://mochajs.org/ (accessed: 25.05.2024).
36. Chai. URL: https://www.chaijs.com/ (accessed: 25.05.2024).
37. Slither, the smart-contract static analyzer. URL: https://github.com/crytic/slither (accessed: 25.05.2024).
38. Microsoft SQL Server technical documentation. URL: https://learn.microsoft.com/en-us/sql/sql-server/?view=sql-server-ver16 (accessed: 25.05.2024).
39. SQLite Home Page. SQLite Home Page. URL: https://www.sqlite.org/index.html (accessed: 25.05.2024).
40. MySQL workbench is. URL: https://www.mysql.com/products/workbench/ (accessed: 25.05.2024).

ДОДАТКИ

Додаток А

Текст програмного коду. Смарт-контракти

Referrals_new.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract Referrals_new {
    struct IntegrityData {
        string patientFullName;
        string referralDate;
        string healthCareFacilityName;
    }

    struct AvailabilityData {
        string priority;
        string validityPeriod;
        string edrpouCode;
        string serviceCode;
    }

    struct IntegrityAvailabilityData {
        string referralNumber;
    }

    struct ConfIntData {
        bytes32 medicalCardNumberHash;
    }

    struct ConfIntAvData {
        bytes32 examinationNameHash;
        bytes32 specialistHash;
    }

    IntegrityData[] public integrityData;
    AvailabilityData[] public availabilityData;
    IntegrityAvailabilityData[] public integrityAvailabilityData;
    ConfIntData[] public confIntData;
    ConfIntAvData[] public confIntAvData;

    function addIntegrityData(
        string memory _patientFullName,
        string memory _referralDate,
        string memory _healthCareFacilityName
    ) public {
        integrityData.push(IntegrityData(_patientFullName, _referralDate,
_healthCareFacilityName));
    }

    function addAvailabilityData(
        string memory _priority,
        string memory _validityPeriod,
        string memory _edrpouCode,
        string memory _serviceCode
    ) public {
        availabilityData.push(AvailabilityData(_priority, _validityPeriod, _edrpouCode,
_serviceCode));
    }

    function addIntegrityAvailabilityData(
        string memory _referralNumber
    ) public {
        integrityAvailabilityData.push(IntegrityAvailabilityData(_referralNumber));
    }
```

```
    function addConfIntData(
        bytes32 _medicalCardNumberHash
    ) public {
        confIntData.push(ConfIntData(_medicalCardNumberHash));
    }

    function addConfIntAvData(
        bytes32 _examinationNameHash,
        bytes32 _specialistHash
    ) public {
        confIntAvData.push(ConfIntAvData(_examinationNameHash, _specialistHash));
    }

    function hashString(string memory _str) public pure returns (bytes32) {
        return keccak256(abi.encodePacked(_str));
    }

}
```

HealthcareData.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract HealthcareData {
    struct Facility {
        string edrpouCode;
        string facilityCode;
    }

    mapping(address => Facility) public facilities;

    event EDRPOUCodeAdded(address indexed sender, string edrpouCode);
    event FacilityCodeAdded(address indexed sender, string facilityCode);

    function addEDRPOUCode(string memory _edrpouCode) public {
        facilities[msg.sender].edrpouCode = _edrpouCode;
        emit EDRPOUCodeAdded(msg.sender, _edrpouCode);
    }

    function addFacilityCode(string memory _facilityCode) public {
        facilities[msg.sender].facilityCode = _facilityCode;
        emit FacilityCodeAdded(msg.sender, _facilityCode);
    }
}
```

Додаток Б.

Текст програмного коду. Модуль клієнта

Referrals.js

```
function generateReferralNumber() {
    const part1 = Math.floor(Math.random() * 10000).toString().padStart(4, '0');
    const part2 = generateRandomString(4, 'K4HK');
    const part3 = generateRandomString(4, '9T6M');
    const part4 = generateRandomString(4, '3A20');
    return `${part1}-${part2}-${part3}-${part4}`;
}

function generateRandomString(length, characters) {
    let result = '';
    const charactersLength = characters.length;
    for (let i = 0; i < length; i++) {
        result += characters.charAt(Math.floor(Math.random() * charactersLength));
    }
    return result;
}
document.getElementById('directionForm').addEventListener('submit', async (event) => {
    event.preventDefault();

    const patientFullName = document.getElementById('patientFullName').value;
    const doctorName = document.getElementById('doctorName').value;
    const examinationName = document.getElementById('examinationName').value;
    const previousDiagnosis = document.getElementById('previousDiagnosis').value;
    const issueDate = document.getElementById('issueDate').value;
    const priority = document.getElementById('priority').value;
    const expiryDate = document.getElementById('expiryDate').value;
    const patientID = document.getElementById('patientID').value;
    const serviceCode = document.getElementById('serviceCode').value;
    const doctorID = document.getElementById('doctorID').value;
    const healthCareFacility = document.getElementById('healthCareFacility').value;
    const healthCareCode = document.getElementById('healthCareCode').value;
    const phoneNumber = document.getElementById('phoneNumber').value;
    const email = document.getElementById('email').value;

    const referralNumber = generateReferralNumber();

    const data = {
        referralNumber: referralNumber,
        patientFullName: patientFullName,
        doctorName: doctorName,
        examinationName: examinationName,
        previousDiagnosis: previousDiagnosis,
        issueDate: issueDate,
        priority: priority,
        expiryDate: expiryDate,
        patientID: patientID,
        serviceCode: serviceCode,
        doctorID: doctorID,
        healthCareFacility: healthCareFacility,
        healthCareCode: healthCareCode,
        phoneNumber: phoneNumber,
        email: email
    };

    try {
        generatePDF(data);
        sendEmail(data);
        await sendDataToServer(data);
        // await sendEmail(data);
```

```
        // generatePDF(data);
    } catch (error) {
        console.error('Помилка при обробці форми:', error.message);
    }
});

async function sendDataToServer(data) {
    try {
        const response = await fetch('http://localhost:3000/sendData', {
            method: 'POST',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify(data)
        });

        if (!response.ok) {
            throw new Error('Помилка відправки даних на сервер');
        }

        const responseData = await response.text();
        console.log('Відповідь сервера:', responseData);
    } catch (error) {
        console.error('Помилка:', error.message);
        throw error;
    }
}

async function sendEmail(data) {
    try {
        const response = await fetch("http://localhost:3000/sendEmail", {
            method: "POST",
            headers: {
                "Content-Type": "application/json",
            },
            body: JSON.stringify(data),
        });

        if (!response.ok) {
            throw new Error('Помилка при відправці листа');
        }

        const responseData = await response.json();
        console.log('Відповідь сервера при відправці листа:', responseData);
    } catch (error) {
        console.error("Помилка під час взаємодії з сервером (sendEmail):", error);
        throw error;
    }
}
function generatePDF(data) {
    const pdfData = {
        content: [
            { text: "Електронне направлення", style: "header" },
            { text: `№ направлення: ${data.referralNumber}`, fontSize: 14, margin: [0, 0,
0, 10] },
            { text: `Лікар: ${data.doctorName}`, fontSize: 14, margin: [0, 0, 0, 10] },
            { text: `Назва обстеження: ${data.examinationName}`, fontSize: 14, margin: [0,
0, 0, 10] },
            { text: `ПІБ пацієнта: ${data.patientFullName}`, fontSize: 14, margin: [0, 0,
0, 10] },
            { text: `Попередній діагноз: ${data.previousDiagnosis}`, fontSize: 14, margin:
[0, 0, 0, 10] },
            { text: `Дата виписування направлення: ${data.issueDate}`, fontSize: 14,
margin: [0, 0, 0, 10] },
            { text: `Пріоритет: ${data.priority}`, fontSize: 14, margin: [0, 0, 0, 10] },
            { text: `Термін придатності: ${data.expiryDate}`, fontSize: 14, margin: [0, 0,
0, 10] },
            { text: `№ медичної картки пацієнта: ${data.patientID}`, fontSize: 14, margin:
[0, 0, 0, 10] },
            { text: `Код послуги: ${data.serviceCode}`, fontSize: 14, margin: [0, 0, 0, 10]
},
```

```
            { text: `№ ліцензії лікаря: ${data.doctorID}`, fontSize: 14, margin: [0, 0, 0,
10] },
            { text: `Найменування закладу охорони здоров'я: ${data.healthCareFacility}`,
fontSize: 14, margin: [0, 0, 0, 10] },
            { text: `Код за ЄДРПОУ/РНОКПП: ${data.healthCareCode}`, fontSize: 14, margin:
[0, 0, 0, 10] },
        ],
        styles: {
            header: {
                fontSize: 18,
                bold: true,
                alignment: "center",
                margin: [0, 0, 0, 20],
            },
        },
    };

    console.log("Дані для PDF сформовані");

    pdfMake.createPdf(pdfData).getBlob((blob) => {
        const pdfUrl = URL.createObjectURL(blob);
        const pdfViewer = document.getElementById('pdfViewer');
        const downloadLink = document.getElementById('downloadLink');
        pdfViewer.src = pdfUrl;
        downloadLink.style.display = "block";
        downloadLink.href = pdfUrl;
        downloadLink.download = "referral_information.pdf";
        console.log("PDF збережено та відображено");
    });
}
```

Patients.js

```
document.addEventListener('DOMContentLoaded', function () {
    const form = document.getElementById('add-patient-form');
    const patientListSection = document.getElementById('patient-list');
    const addPatientSection = document.getElementById('add-patient');
    const viewPatientsBtn = document.getElementById('view-patients-btn');

    form.addEventListener('submit', function (event) {
        event.preventDefault();

        const formData = new FormData(form);
        const patientData = {
            cardNumber: formData.get('card-number'),
            fullName: formData.get('full-name'),
            gender: formData.get('gender'),
            dob: formData.get('dob'),
            phone: formData.get('phone'),
            email: formData.get('email'),
            familyDoctor: formData.get('family-doctor')
        };

        fetch('/patients', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify(patientData)
        })
        .then(response => {
            if (!response.ok) {
                throw new Error('Failed to add patient');
            }
            return response.json();
        })
        .then(newPatient => {
            console.log('New patient added:', newPatient);
            form.reset();
```

```
            addPatientSection.style.display = 'none';
            patientListSection.style.display = 'block';
            displayPatients();
        })
        .catch(error => {
            console.error('Error adding patient:', error);
        });
    });

    function displayPatients() {
        console.log('Fetching patients from server...');
        fetch('/patients')
            .then(response => {
                console.log('Response status:', response.status);
                if (!response.ok) {
                    throw new Error('Failed to fetch patients');
                }
                return response.json();
            })
            .then(patients => {
                const tableBody = document.querySelector('#patients-table tbody');
                tableBody.innerHTML = '';
                patients.forEach(patient => {
                    const row = document.createElement('tr');
                    row.innerHTML = `
                        <td>${patient.cardNumber}</td>
                        <td>${patient.fullName}</td>
                        <td>${patient.familyDoctor}</td>
                    `;
                    tableBody.appendChild(row);
                });

                const patientsTable = document.getElementById('patients-table');
                console.log('Patients table element:', patientsTable);
                patientsTable.style.display = 'table';
            })
            .catch(error => {
                console.error('Error fetching patients:', error);
            });
    }

    viewPatientsBtn.addEventListener('click', function () {
        addPatientSection.style.display = 'block';
        patientListSection.style.display = 'none';
    });
});
```

Додаток В.

Текст програмного коду. Модуль сервера

Server.js

```
const express = require('express');
const bodyParser = require('body-parser');
const { ethers } = require('ethers');
const nodemailer = require('nodemailer');
const cors = require('cors');
const mssql = require('mssql/msnodesqlv8');
const app = express();
const port = 3000;
app.get('/', (req, res) => {
    res.send('Hello!');
  });
  var dbConfig = {
    server: 'DESKTOP-CSFCFTG\\SQLEXPRESS',
    database: 'MyDyploma',
    driver: 'msnodesqlv8',
    options: {
      trustedConnection: true,
    },
  };
  app.use(cors());
  app.use(cors({
    origin: '*',
    methods: 'POST, GET, OPTIONS, PUT, DELETE',
    allowedHeaders: 'Origin, application/json, X-Requested-With, Content-Type, Accept,
Authorization',
  }));
  app.get('*', (req, res) => {
    res.sendFile(path.join(__dirname, '../project/referrals/referrals.html'));
  });
  app.options('/saveReferral', (req, res) => {
    res.header('Access-Control-Allow-Origin', '*');
    res.header('Access-Control-Allow-Methods', 'POST');
    res.header('Access-Control-Allow-Headers', 'Content-Type');
    res.status(200).send();
  });

const contractAddress = '0xa145242b8720c5Ce5c80A040cac47a8013F553fb';
const contractABI = [
    {
        "inputs": [
          {
            "internalType": "uint256",
            "name": "",
            "type": "uint256"
          }
        ],
        "name": "availabilityData",
        "outputs": [
          {
            "internalType": "string",
            "name": "priority",
            "type": "string"
          },
          {
            "internalType": "string",
            "name": "validityPeriod",
            "type": "string"
          },
          {
```

```
      "internalType": "string",
      "name": "edrpouCode",
      "type": "string"
    },
    {
      "internalType": "string",
      "name": "serviceCode",
      "type": "string"
    }
  ],
  "stateMutability": "view",
  "type": "function",
  "constant": true
},
{
  "inputs": [
    {
      "internalType": "uint256",
      "name": "",
      "type": "uint256"
    }
  ],
  "name": "confIntAvData",
  "outputs": [
    {
      "internalType": "bytes32",
      "name": "examinationNameHash",
      "type": "bytes32"
    },
    {
      "internalType": "bytes32",
      "name": "specialistHash",
      "type": "bytes32"
    }
  ],
  "stateMutability": "view",
  "type": "function",
  "constant": true
},
{
  "inputs": [
    {
      "internalType": "uint256",
      "name": "",
      "type": "uint256"
    }
  ],
  "name": "confIntData",
  "outputs": [
    {
      "internalType": "bytes32",
      "name": "medicalCardNumberHash",
      "type": "bytes32"
    }
  ],
  "stateMutability": "view",
  "type": "function",
  "constant": true
},
{
  "inputs": [
    {
      "internalType": "uint256",
      "name": "",
      "type": "uint256"
    }
  ],
  "name": "integrityAvailabilityData",
  "outputs": [
    {
```

```
      "internalType": "string",
      "name": "referralNumber",
      "type": "string"
    }
  ],
  "stateMutability": "view",
  "type": "function",
  "constant": true
},
{
  "inputs": [
    {
      "internalType": "uint256",
      "name": "",
      "type": "uint256"
    }
  ],
  "name": "integrityData",
  "outputs": [
    {
      "internalType": "string",
      "name": "patientFullName",
      "type": "string"
    },
    {
      "internalType": "string",
      "name": "referralDate",
      "type": "string"
    },
    {
      "internalType": "string",
      "name": "healthCareFacilityName",
      "type": "string"
    }
  ],
  "stateMutability": "view",
  "type": "function",
  "constant": true
},
{
  "inputs": [
    {
      "internalType": "string",
      "name": "_patientFullName",
      "type": "string"
    },
    {
      "internalType": "string",
      "name": "_referralDate",
      "type": "string"
    },
    {
      "internalType": "string",
      "name": "_healthCareFacilityName",
      "type": "string"
    }
  ],
  "name": "addIntegrityData",
  "outputs": [],
  "stateMutability": "nonpayable",
  "type": "function"
},
{
  "inputs": [
    {
      "internalType": "string",
      "name": "_priority",
      "type": "string"
    },
    {
```

```
        "internalType": "string",
        "name": "_validityPeriod",
        "type": "string"
      },
      {
        "internalType": "string",
        "name": "_edrpouCode",
        "type": "string"
      },
      {
        "internalType": "string",
        "name": "_serviceCode",
        "type": "string"
      }
    ],
    "name": "addAvailabilityData",
    "outputs": [],
    "stateMutability": "nonpayable",
    "type": "function"
  },
  {
    "inputs": [
      {
        "internalType": "string",
        "name": "_referralNumber",
        "type": "string"
      }
    ],
    "name": "addIntegrityAvailabilityData",
    "outputs": [],
    "stateMutability": "nonpayable",
    "type": "function"
  },
  {
    "inputs": [
      {
        "internalType": "bytes32",
        "name": "_medicalCardNumberHash",
        "type": "bytes32"
      }
    ],
    "name": "addConfIntData",
    "outputs": [],
    "stateMutability": "nonpayable",
    "type": "function"
  },
  {
    "inputs": [
      {
        "internalType": "bytes32",
        "name": "_examinationNameHash",
        "type": "bytes32"
      },
      {
        "internalType": "bytes32",
        "name": "_specialistHash",
        "type": "bytes32"
      }
    ],
    "name": "addConfIntAvData",
    "outputs": [],
    "stateMutability": "nonpayable",
    "type": "function"
  },
  {
    "inputs": [
      {
        "internalType": "string",
        "name": "_str",
        "type": "string"
```

```
          }
        ],
        "name": "hashString",
        "outputs": [
          {
            "internalType": "bytes32",
            "name": "",
            "type": "bytes32"
          }
        ],
        "stateMutability": "pure",
        "type": "function",
        "constant": true
      }
];
const provider = new ethers.providers.JsonRpcProvider('HTTP://127.0.0.1:7545');
const signer = provider.getSigner();
const contract = new ethers.Contract(contractAddress, contractABI, signer);

async function writeToBlockchain(data) {
    const cData = {
        patientName: data.patientFullName,
        referralDate: data.issueDate,
        medicalInstitutionName: data.healthCareFacility
    };

    const dData = {
        priority: data.priority,
        validityPeriod: data.expiryDate,
        edrpouCode: data.healthCareCode,
        serviceCode: data.serviceCode

    };
    const cdData = {
        referralNumber: data.referralNumber
    }
    const confIntDataHash =
ethers.utils.keccak256(ethers.utils.toUtf8Bytes(data.patientID));

    const confIntAvDataHash = {
        examinationNameHash:
ethers.utils.keccak256(ethers.utils.toUtf8Bytes(data.examinationName)),
        specialistHash: ethers.utils.keccak256(ethers.utils.toUtf8Bytes(data.doctorName))
    };
        const { referralNumber } = cdData;
        const transaction = await
contract.connect(signer).addIntegrityAvailabilityData(referralNumber);
        await transaction.wait();
        console.log('Transaction Hash:', transaction.hash);
        const { patientName, referralDate, medicalInstitutionName } = cData;
        await contract.connect(signer).addIntegrityData(patientName, referralDate,
medicalInstitutionName);

        const { priority, validityPeriod, edrpouCode, serviceCode} = dData;
        await contract.connect(signer).addAvailabilityData( priority, validityPeriod,
edrpouCode, serviceCode);

        await contract.addConfIntData(confIntDataHash);
        await contract.addConfIntAvData(confIntAvDataHash.examinationNameHash,
confIntAvDataHash.specialistHash);
        return transaction.hash;

}
async function saveToDatabase(data) {
  try {
      await mssql.connect(dbConfig);
      const request = new mssql.Request();
      await request.input('ReferralNumber', mssql.NVarChar, data.referralNumber)
          .input('PatientID', mssql.NVarChar, data.patientID)
          .input('Diagnosis', mssql.NVarChar, data.previousDiagnosis)
```

```
            .input('EDRPOUCode', mssql.NVarChar, data.healthCareCode)
            .query(`
                INSERT INTO Referrals (
                    ReferralNumber, PatientID, Diagnosis, EDRPOUCode
                ) VALUES (
                    @ReferralNumber, @PatientID, @Diagnosis, @EDRPOUCode
                )
            `);
    } catch (err) {
        console.error('Database insert error:', err);
    } finally {
        mssql.close();
    }
}

async function sendSMS(data) {
    try {
      const smsResponse = await fetch(
        'https://api-gateway.kyivstar.ua/sandbox/rest/v1beta/sms',
        {
          method: 'POST',
          headers: {
            'Accept': 'application/json',
            'Content-Type': 'application/json',
            'Authorization': 'Bearer
HbtaPO4AdQCl8NQs9F8hsn96gSiJuUpmSHXzNssSRrM.nhiW3MVLs9DDpp86HWw4x-XKS0RUa76o41YK0Q-u4Z0'
          },
          body: JSON.stringify({
            "from": "messagedesk",
            "to": '+380987473339',
            "text": `Ваш № направлення: ${data.referralNumber}`,
          }),
        }
      );

      if (smsResponse.ok) {
        console.log('SMS відправлено успішно');
      } else {
        console.error('Помилка при відправленні SMS:', await smsResponse.json());
      }
    } catch (error) {
      console.error('Помилка при відправленні SMS:', error);
    }
}

app.use(bodyParser.json());

app.post('/sendData', async (req, res) => {
    const data = req.body;

    try {
      const transactionHash = await writeToBlockchain(data);
      await saveToDatabase(data, transactionHash);

        res.send('Data successfully saved to blockchain and database.');
    } catch (error) {
        console.error('Error saving data:', error);
        res.status(500).send('Error saving data');
    }
});

app.post('/patients', (req, res) => {
    const newPatient = req.body;
    console.log('Дані нового пацієнта:', newPatient);
    const query = `INSERT INTO Patients (PatientID, PhoneNumber, Email, Gender)
                   VALUES ('${newPatient.cardNumber}', '${newPatient.phone}',
'${newPatient.email}', '${newPatient.gender}')`;

    mssql.connect(dbConfig)
        .then(pool => {
```

```
            return pool.request().query(query);
        })
        // .then(result => {
        //     result = console.log('New patient added:', newPatient);
        //     res.status(201).json(newPatient);
        // })
        .catch(error => {
            console.error('Error adding patient:', error);
            res.status(500).json({ error: 'Error adding patient' });
        });
});

app.get('/patients', (req, res) => {
const query = 'SELECT * FROM Patients';

console.log('GET request received at /patients');

mssql.connect(dbConfig)
    .then(pool => {
        console.log('Connected to database');
        return pool.request().query(query);
    })
    .then(result => {
        console.log('Query executed successfully');
        res.json(result.recordset);
    })
    .catch(error => {
        console.error('Error fetching patients:', error);
        res.status(500).json({ error: 'Error fetching patients' });
    });
});

app.post('/sendEmail', async (req, res) => {
    console.log('Отримано POST-запит на маршрут /sendEmail');
    try {
        const { email, referralNumber, doctorName } = req.body;
        console.log('Дані з запиту:', { email, referralNumber, doctorName });

        const transporter = nodemailer.createTransport({
            service: 'gmail',
            host: 'smtp.gmail.com',
            port: 465,
            secure: true,
            auth: {
                user: 'lanovaia02y@gmail.com',
                pass: 'pmly cnma gxsh sdeu',
            },
        });

        const mailOptions = {
            from: 'lanovaia02y@gmail.com',
            to: email,
            subject: 'Your Health',
            text: `Ваш № направлення: ${referralNumber} до ${doctorName}`,
        };

        console.log('Надсилаємо лист...');
        const info = await transporter.sendMail(mailOptions);
        await sendSMS({referralNumber });
        console.log('Лист успішно відправлено:', info);
        res.status(200).send('ReferralNumber sent to email.');
    } catch (err) {
        console.error('Помилка при відправленні email:', err);
        res.status(500).send('Помилка при відправленні email');
    }
});

const contract1Address = '0x424f8Ea863Db73F1371e06c0932b80A4f7200D00';
const contract1Abi = [
  {
```

```
  "anonymous": false,
  "inputs": [
    {
      "indexed": true,
      "internalType": "address",
      "name": "sender",
      "type": "address"
    },
    {
      "indexed": false,
      "internalType": "string",
      "name": "edrpouCode",
      "type": "string"
    }
  ],
  "name": "EDRPOUCodeAdded",
  "type": "event"
},
{
  "anonymous": false,
  "inputs": [
    {
      "indexed": true,
      "internalType": "address",
      "name": "sender",
      "type": "address"
    },
    {
      "indexed": false,
      "internalType": "string",
      "name": "facilityCode",
      "type": "string"
    }
  ],
  "name": "FacilityCodeAdded",
  "type": "event"
},
{
  "inputs": [
    {
      "internalType": "address",
      "name": "",
      "type": "address"
    }
  ],
  "name": "facilities",
  "outputs": [
    {
      "internalType": "string",
      "name": "edrpouCode",
      "type": "string"
    },
    {
      "internalType": "string",
      "name": "facilityCode",
      "type": "string"
    }
  ],
  "stateMutability": "view",
  "type": "function",
  "constant": true
},
{
  "inputs": [
    {
      "internalType": "string",
      "name": "_edrpouCode",
      "type": "string"
    }
  ],
```

```
    "name": "addEDRPOUCode",
    "outputs": [],
    "stateMutability": "nonpayable",
    "type": "function"
  },
  {
    "inputs": [
      {
        "internalType": "string",
        "name": "_facilityCode",
        "type": "string"
      }
    ],
    "name": "addFacilityCode",
    "outputs": [],
    "stateMutability": "nonpayable",
    "type": "function"
  }
];
const privateKey = '2d70982e3d29eaf839e13f8b6d909a185aa074d6dcdcbf54ea904dcaa32d9fcf';
const wallet = new ethers.Wallet(privateKey, provider);
const contract1 = new ethers.Contract(contract1Address, contract1Abi, wallet);

app.get('/form', (req, res) => {
  res.send(`
    <form action="/submit" method="POST">
      <label for="edrpouCode">EDRPOU Code:</label>
      <input type="text" id="edrpouCode" name="edrpouCode"><br>
      <label for="facilityCode">Facility Code:</label>
      <input type="text" id="facilityCode" name="facilityCode"><br>
      <label for="address">Address:</label>
      <input type="text" id="address" name="address"><br>
      <input type="submit" value="Submit">
    </form>
  `);
});

app.post('/submit', async (req, res) => {
  const { edrpouCode, facilityCode, address } = req.body;

  try {

    const edrpouTx = await contract1.addEDRPOUCode(edrpouCode);
    const edrpouReceipt = await edrpouTx.wait();
    const edrpouHash = edrpouReceipt.transactionHash;

    const facilityTx = await contract1.addFacilityCode(facilityCode);
    const facilityReceipt = await facilityTx.wait();
    const facilityHash = facilityReceipt.transactionHash;

    await mssql.connect(dbConfig);
    await mssql.query(`
      INSERT INTO HealthcareFacility (EDRPOUCode, FacilityCode, Address)
      VALUES ('${edrpouHash}', '${facilityHash}', '${address}')
    `);

    res.send('Data submitted and hashes stored in the database.');
  } catch (err) {
    console.error('Error submitting data:', err);
    res.status(500).send('Error submitting data.');
  }
});
app.listen(port, () => {
    console.log(`Сервер запущено на порту ${port}`);
});
```

Додаток Г

Протокол перевірки бакалаврської дипломної роботи на наявність текстових запозичень

Назва роботи: **Автоматизована система видавання направлень на обстеження в медичних закладах. Частина 1. Модуль захищеного зберігання даних**

Автор роботи: **Ланова Владислава Сергіївна**

Тип роботи: бакалаврська дипломна робота (БДР, МКР)

Підрозділ кафедра захисту інформації ФІТКІ (кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність – 92,92%. Схожість – 7,08%.

Аналіз звіту подібності (відмітити потрібне):

√ 1. Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.

□ 2. Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.

□ 3. Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку Валентина КАПЛУН

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи (підпис) Владислава ЛАНОВА (прізвище, ініціали)

Керівник роботи (підпис) Юрій БАРИШЕВ (прізвище, ініціали)

ІЛЮСТРАТИВНА ЧАСТИНА

АВТОМАТИЗОВАНА СИСТЕМА ВИДАВАННЯ НАПРАВЛЕНЬ НА ОБСТЕЖЕННЯ В МЕДИЧНИХ ЗАКЛАДАХ. ЧАСТИНА 1. МОДУЛЬ ЗАХИЩЕНОГО ЗБЕРІГАННЯ ДАНИХ

Виконала: студентка 4 курсу групи 1БС-20 б
спеціальності 125 Кібербезпека
_____ Владислава ЛАНОВА

Керівник: к. т. н., доцент, доцент каф. ЗІ
_____ Юрій БАРИШЕВ
«14» червня 2024 р.

ПОРІВНЯЛЬНИЙ АНАЛІЗ СХОВИЩ ДАНИХ

Метод захисту	Захист цілісності	Захист доступності	Захист конфіденційності	Вартість	Обсяг даних в межах транзакції
База даних	+	-	+	Низька	Високий
Приватний блокчейн	+	+	+	Висока	Низький
Публічний блокчейн	+	+	-	Середня	Низький
IPFS	+	-	-	Безкоштовно	Середній

МАТЕМАТИЧНИЙ ОПИС ПРОЦЕСУ ВИДАВАННЯ ЕЛЕКТРОННИХ НАПРАВЛЕНЬ

$$M = (T, X, Y, Z, z(t), C),$$

де M – модель системи.

T – модельний час, який представляє часовий період, під час якого необхідно видати направлення.

X – множина значень вхідних змінних, які визначають параметри направлення.

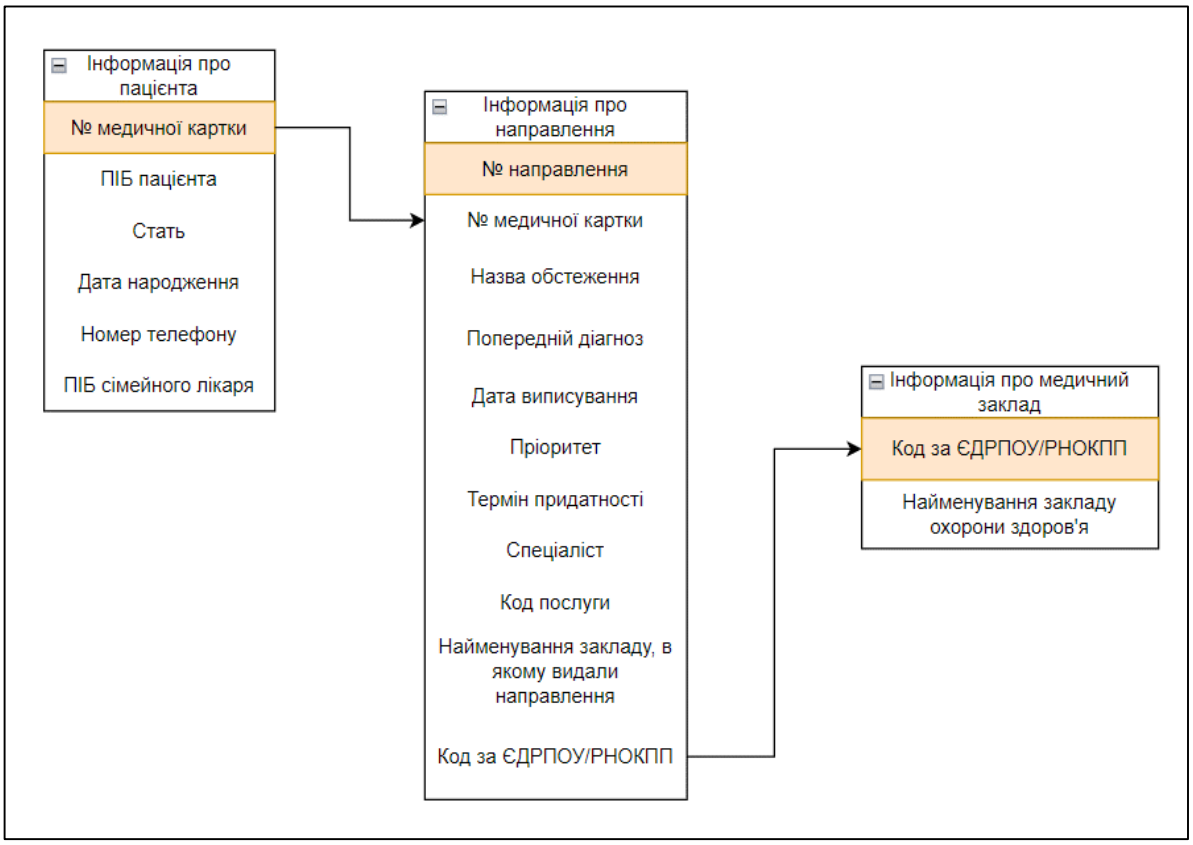
Y – множина значень вихідних змінних, які вказують на результати вже сформованого направлення.

Z – простір станів моделі, що включає у себе всі можливі стани системи.

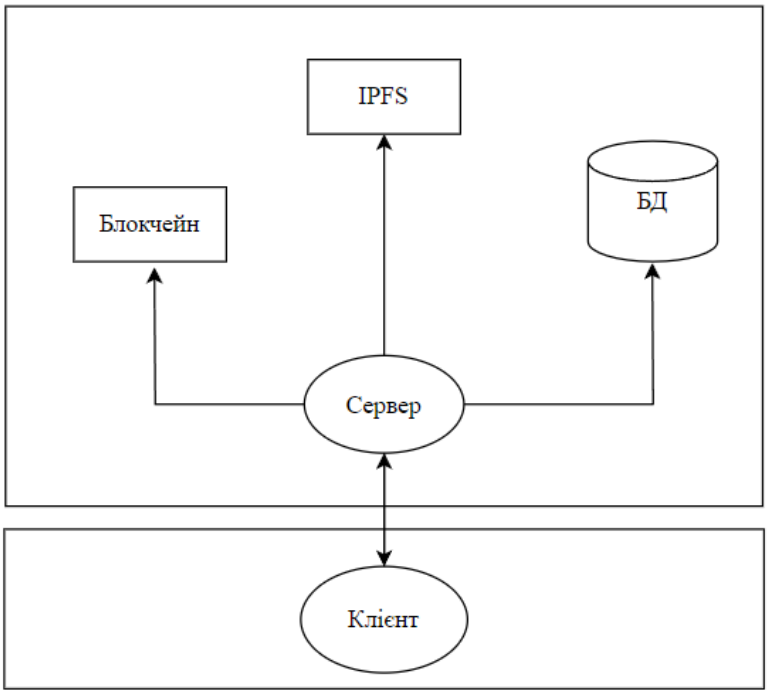
$z(t)$ – функція станів, яка описує зміни у станах системи з плином часу.

C – множина процесів.

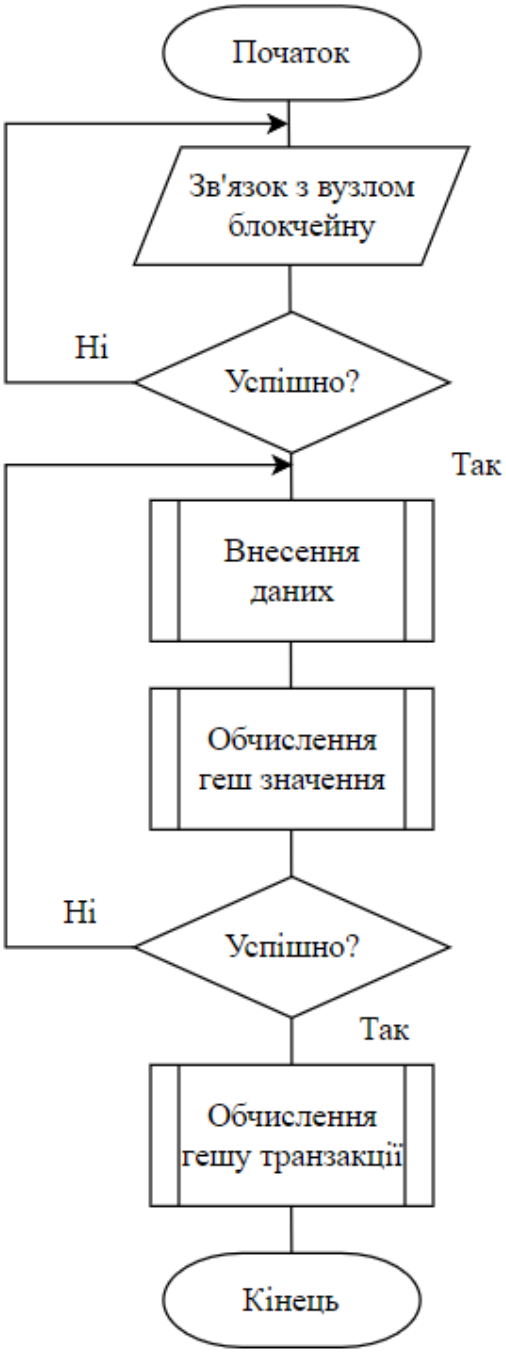
МОДЕЛЬ ДАНИХ АВТОМАТИЗОВАНОЇ СИСТЕМИ ВИДАВАННЯ ЕЛЕКТРОННИХ НАПРАВЛЕНЬ



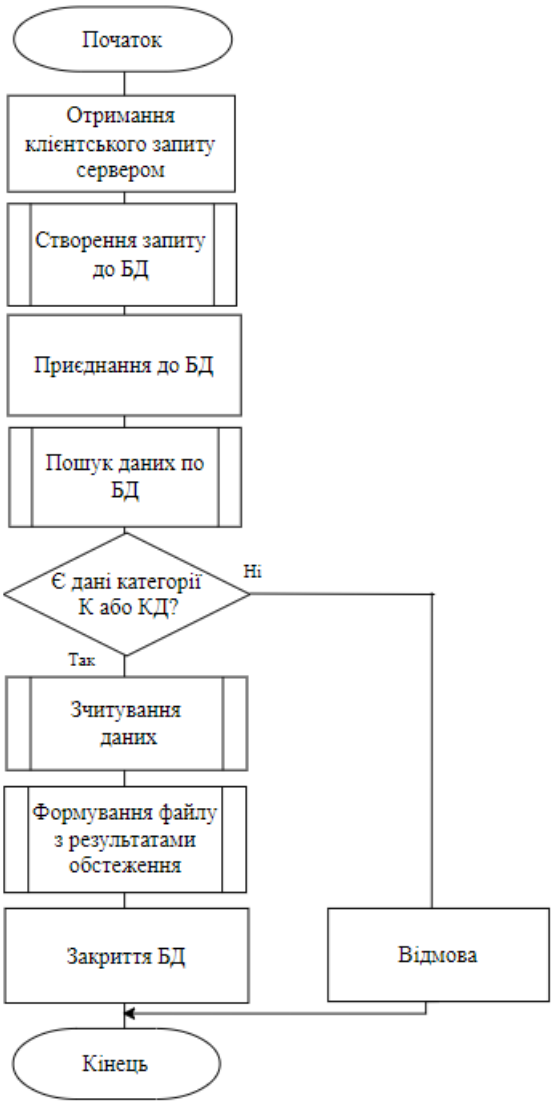
АРХІТЕКТУРА ПРОГРАМНОГО МОДУЛЯ



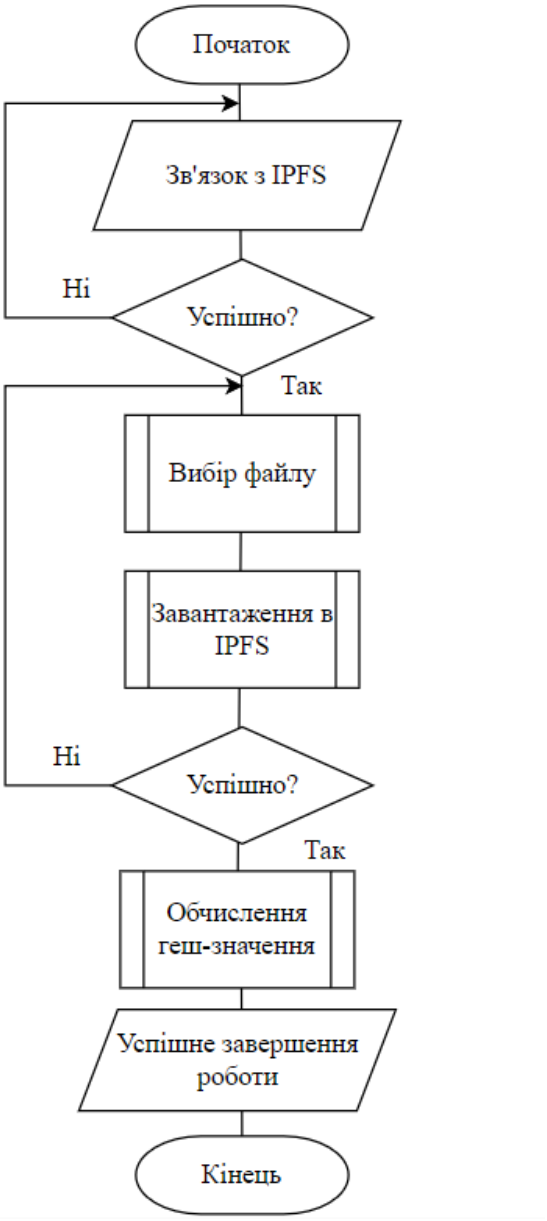
АЛГОРИТМ ЗАПИСУ В БЛОКЧЕЙН



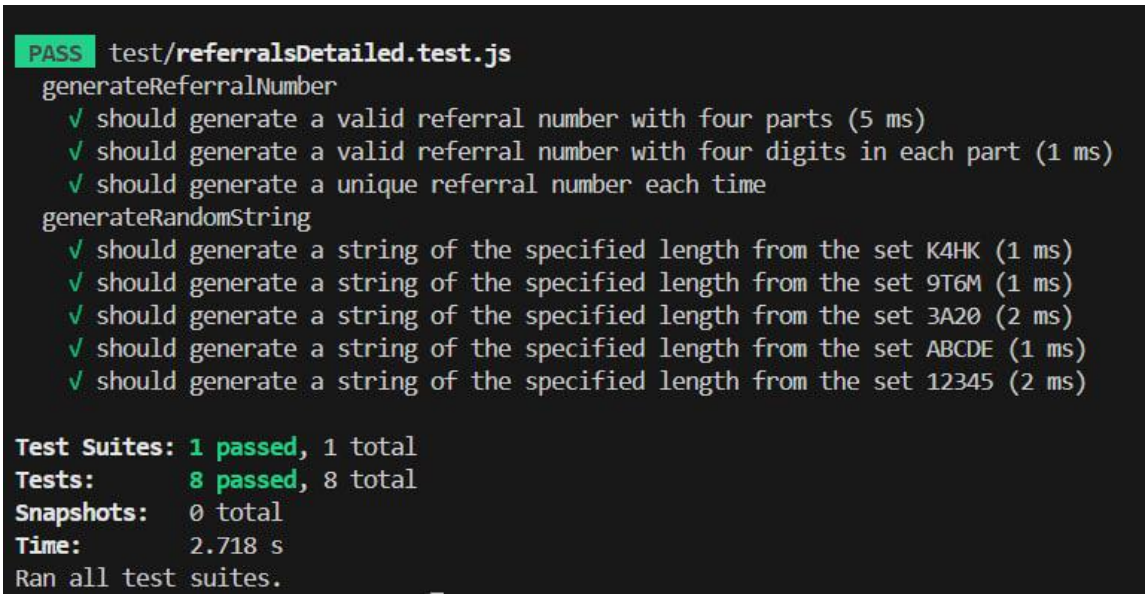
АЛГОРИТМ ЗАПИСУ В БАЗУ ДАНИХ



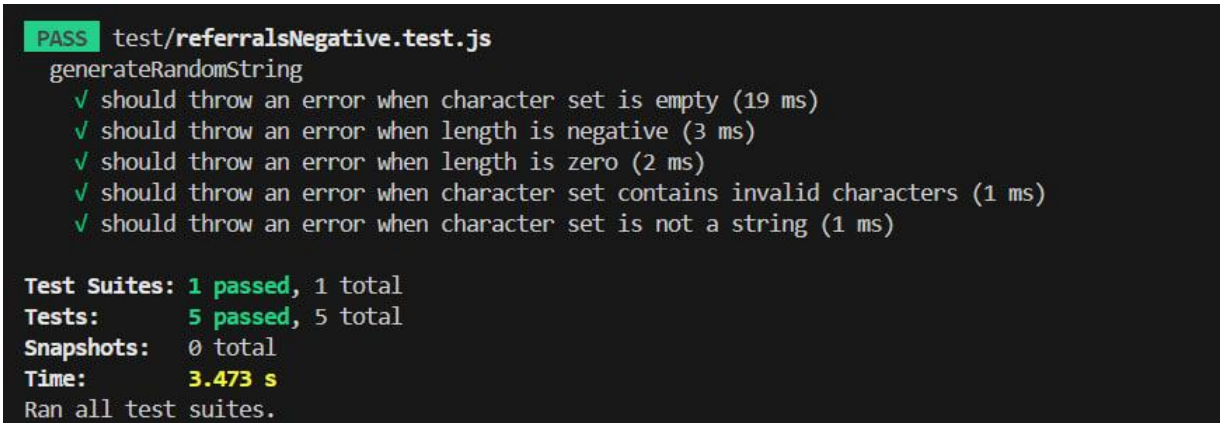
АЛГОРИТМ ЗАПИСУ В IPFS



РЕЗУЛЬТАТИ БЛОКОВОГО ТЕСТУВАННЯ

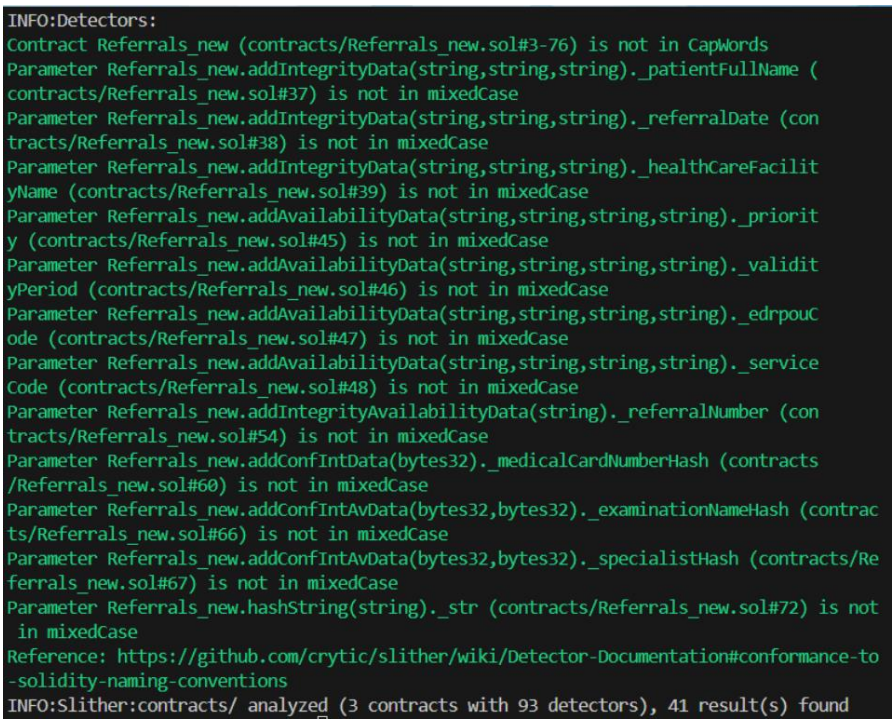


Результати позитивного тестування

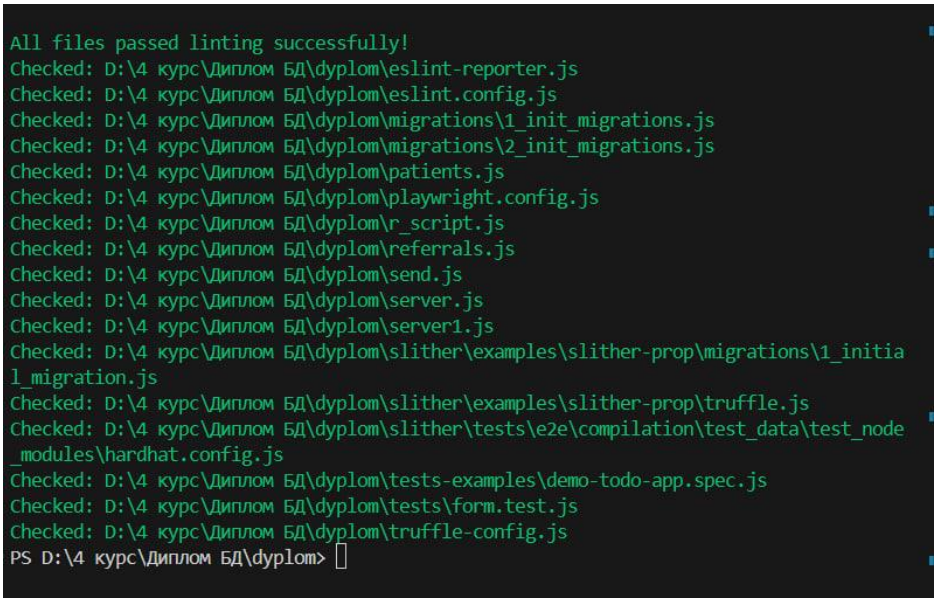


Результати негативного тестування

РЕЗУЛЬТАТИ ТЕСТУВАННЯ БЕЗПЕКИ

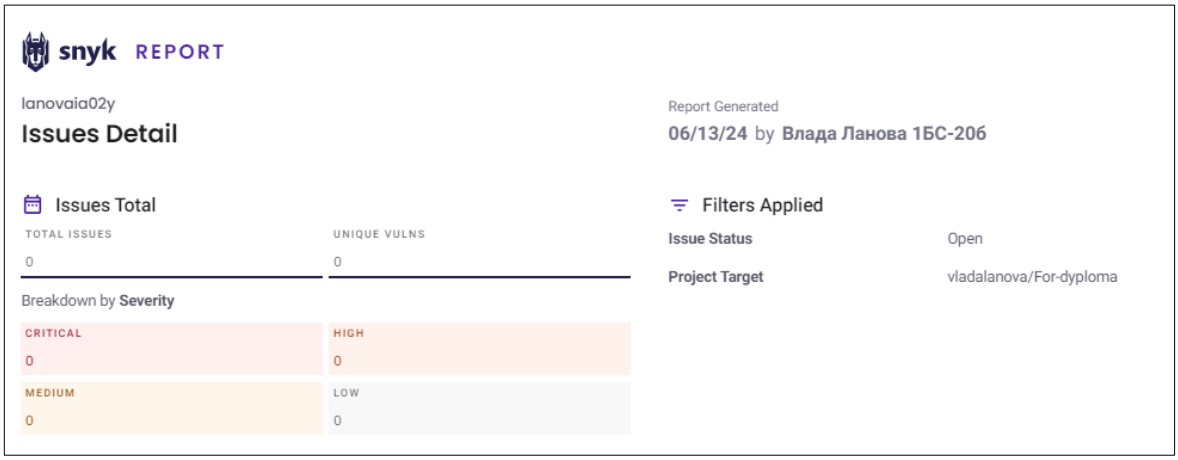


Результати статичного тестування безпеки смарт-контрактів



Результати статичного тестування безпеки за допомогою фреймворка ESLint

РЕЗУЛЬТАТИ ТЕСТУВАННЯ БЕЗПЕКИ



Результати статичного тестування безпеки за допомогою Snyk