

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВАДИМА ГЕТЬМАНА**

**Навчально-науковий інститут
«Інститут інформаційних технологій в економіці»**

Кафедра інформаційних систем в економіці

**ОСВІТНЬО-ПРОФЕСІЙНА ПРОГРАМА
«Системи штучного інтелекту»**

галузь знань **12 Інформаційні
технології**
спеціальність **122 Комп'ютерні науки**

Форма навчання: очна (денна)

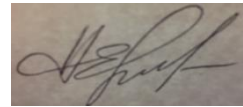
КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

**на тему: «Розробка інтелектуальної медичної картки пацієнта на основі
блокчейн технологій»**

здобувача Ублінських Андрій Анатолійович _____



Науковий керівник: к.е.н., доцент Ситник Н.В. _____



**Робота допущена до захисту перед екзаменаційною комісією з атестації
здобувачів вищої освіти (ЕК)**

**Завідувач кафедри: к.е.н. , доцент
Тішков Б.О.** _____

Київ 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВАДИМА ГЕТЬМАНА
Навчально-науковий інститут «Інститут інформаційних технологій в економіці»
Кафедра інформаційних систем в економіці

ОСВІТНЬО-ПРОФЕСІЙНА ПРОГРАМА

«Системи штучного інтелекту»

галузь знань 12 Інформаційні технології

спеціальність 122 Комп'ютерні науки

ПОГОДЖЕНО:

Керівник проєктної групи (гарант)
освітньо-професійної програми

Рамазанов С.К.

«___» _____ 2024 р.

ЗАТВЕРДЖУЮ:

Завідувач кафедри інформаційних систем в економіці

Тішков Б.О.

«___» _____ 2024 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

здобувача вищої освіти Ублінських Андрія Анатолійовича

очної форми навчання

на підготовку кваліфікаційної магістерської роботи

на тему: **«Розробка інтелектуальної медичної картки пацієнта на основі
блокчейн технологій»**

Тему затверджено наказом ректора Університету від «23» січня 2023 р. №126-ст

Кваліфікаційна магістерська робота виконується на матеріалах наукових публікацій та
матеріалів з мережі інтернет

План кваліфікаційної магістерської роботи

Розділ I Теоретичний розділ. Дослідження та аналіз підходів до створення
предметної області СІШ

Розділ II Аналітичний розділ. Характеристика СІШ та постановка задачі

Розділ III Конструктивний розділ. Розроблення проєктних рішень

Об'єкт дослідження: інформаційні процеси, що характеризують процеси
формування медичних карток пацієнтів закладів охорони здоров'я.

Предмет дослідження: технологія блокчейну, система генеративного штучного
інтелекту ChatGPT та їх використання для формування захищеної
інтелектуальної медичної картки пацієнта.

Мета кваліфікаційної магістерської роботи: розробка інтелектуальної медичної
картки з використанням блокчейн технології та чат-бота ChatGPT .

Конкретні завдання, які здобувач повинен виконати для досягнення поставленої мети

У розділі I Дослідити основні процедури формування медичної картки пацієнта. Проаналізувати існуючі системи використання блокчейн технології в медичній галузі. Визначити функціонал використання інтелектуальної системи формування медичної картки пацієнта, розробити концепцію та технологію створення системи з їх використанням в предметній області, що досліджується у кваліфікаційні магістерській роботі.

У розділі II Визначити вимоги до системи та потреби її кінцевих користувачів – лікарів і пацієнтів. На основі дослідження інформаційних процесів, що характеризують формування медичної картки пацієнта, розробити архітектуру інтелектуальної системи, обґрунтувавши вибір та описати основні її сервіси. Враховуючи, що система зорієнтована на використання природної мови, визначитись з основними сценаріями та дослідити використання великої мовної моделі ChatGPT для потреб підтримки процесів формування медичної картки пацієнта

У розділі III Запроєктувати архітектуру інформаційної системи з формування інтелектуальної медичної картки пацієнта на основі блокчейн технології. Надати структуру інтелектуальної медичної картки й її інтеграцію з блокчейн технологією та інтелектуальним чат-ботом ChatGPT. Запроєктувати інтерфейс взаємодії кінцевих користувачів - лікаря та пацієнта при формуванні та роботі з медичною картою. Виконати реалізацію інформаційного, програмного, технічного та організаційного забезпечення з формування інтелектуальної медичної картки пацієнта на основі блокчейн технології та проілюструвати це даними тестового прикладу.

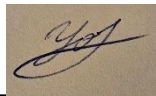
Завдання підготував

науковий керівник



Ситник Н.В.)

Завдання одержав



«24» лютого 2023р.

Ублінських А.А.

«24» лютого

РЕФЕРАТ

Кваліфікаційна магістерська робота містить 83 сторінок, 14 таблиць, 38 рисунків, список літератури з 32 найменувань, 7 додатків.

«Розробка інтелектуальної медичної картки пацієнта на основі блокчейн технології»

Об'єктом дослідження виступають інформаційні процеси, що характеризують процеси формування медичних карток пацієнтів закладів охорони здоров'я.

Предметом дослідження є технологія блокчейну, система генеративного штучного інтелекту ChatGPT та їх використання для формування захищеної інтелектуальної медичної картки пацієнта.

Мета кваліфікаційної магістерської роботи полягає в розробці інтелектуальної медичної картки з використанням блокчейн технології та чат-бота ChatGPT.

Завданням кваліфікаційної магістерської роботи є розробка інтелектуальної медичної картки з використанням блокчейн технології та чат-бота ChatGPT.

Технічне забезпечення, використане при дослідженні: персональний комп'ютер, доступ до Інтернету.

Новизна полягає у розробці медичної картки, котра буде зберігати інформацію про пацієнта в захищеному вигляді.

Ступінь упровадження: розроблена система може бути інтегрована в лікарні з ціллю зменшити вартість збереження даних пацієнтів.

Одержані результати можуть бути використані медичними організаціями, приватними і громадськими лікарнями за для зменшення вартості зберігання медичних даних пацієнтів і збільшення продуктивності роботи лікарів, використовуючи мовну модель ChatGPT.

Ключові слова: ChatGPT, мовна модель, RSA шифрування, блокчейн, розумні контракти, інтелектуальна система.

ВІДГУК

на кваліфікаційну магістерську роботу здобувача 6 курсу, денної форми навчання,
ОП « Інформаційні управляючі системи і технології»
спеціальності «Комп'ютерні науки»

Навчально-наукового інституту «Інститут інформаційних технологій в економіці»
Ублінських Андрія Анатолійовича

на тему:

«Розробка інтелектуальної медичної картки пацієнта на основі блокчейн технології»

Актуальність теми: Кваліфікаційна магістерська робота присвячена актуальній темі - розробці захищеної медичної картки при формуванні якої використовуються сучасні засоби штучного інтелекту.

Позитивні характеристики кваліфікаційної магістерської роботи: В роботі обґрунтовано використання блокчейн технологій для надійного захисту конфіденційних медичних даних пацієнтів та використання генеративного інтелектуального чат-бота ChatGPT, як помічника лікаря при занесенні в медичну картку діагнозів та результатів аналізів, що дозволяє покращити сервіс та зробити його більш зручним для кінцевих користувачів.

Наявність самостійних розробок автора. У кваліфікаційній магістерській роботі досліджені основні бізнес-процеси, які характеризують процеси формування лікарем медичної картки, перегляду та знайомства з її інформацією пацієнтів медичних закладів, проведена їх структуризація та опис з погляду системного підходу та повноти їх охоплення. При моделюванні структури інформаційної системи використані сучасні системи автоматизованого інжинірингу. В роботі запроєктована реляційна база даних в середовищі реляційних СКБД PostgreSQL Для захисту даних бази даних запропоновано використання RSA алгоритму. Розроблено робочий прототип системи з формування медичної картки.

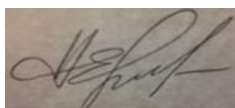
Цінність теоретичних висновків та практичних рекомендацій. Робота має практичну цінність, оскільки за результатами досліджень зроблені висновки і пропозиції, які можуть бути використані на практиці та сприятимуть в майбутньому покращенню процедур з формування медичних карток. Для реалізації системи було обрано трирівневу архітектуру технології клієнт-сервер та сучасні засоби підтримки блокчейн технології.

Наявність недоліків в роботі. Дуже лаконічно, без достатніх обґрунтувань, описано в роботі економічні переваги інтелектуальної медичної картки.

Загальна оцінка кваліфікаційної магістерської роботи та її допущення до захисту перед ЕК: кваліфікаційна магістерська робота відповідає встановленим вимогам методичних вказівок щодо структури, обсягу та змісту. У цілому, представлена робота Ублінських А. А. рекомендується до захисту з високою оцінкою.

Керівник к.е.н., доцент,

професор кафедри ІСЕ



Ситник Н.В.

18 травня 2024 р.

Рецензія

на кваліфікаційну магістерську роботу
здобувача вищої освіти

Ублінських Андрія Анатолійовича

(прізвище, ім'я, по батькові)

Тема: Розробка інтелектуальної медичної картки пацієнта на основі блокчейн технології

Актуальність теми кваліфікаційної магістерської роботи і доцільність її розроблення: дана магістерська робота розглядає актуальну тему блокчейн технологій. Тема відносно нова і почала активно використовуватися з 2014 року, відповідно є важливим дослідження інтеграції цієї технології в інші галузі, такими як галузь охорони здоров'я.

Якість проведеного дослідження: автор даної роботи розглянув інші існуючі проєкти, котрі розвиваються в напрямку блокчейн технологій в медицині і вирішив виправити їх недоліки шляхом шифрування даних використовуючи алгоритм RSA.

Позитивні риси кваліфікаційної магістерської роботи: було розроблено простий і зрозумілий інтерфейс для користувачів (лікарям і пацієнтам), котрий дозволяє комунікувати з блокчейн мережами. Також було реалізована підтримка шифрування, використовуючи RSA алгоритм і інтегровано підтримку ChatGPT.

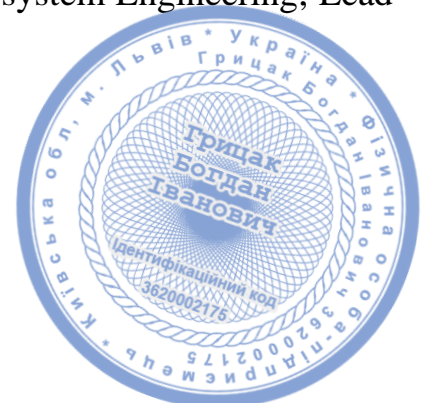
Зауваження: в роботі допущені помилки зі сторони безпеки конфіденційних даних, не було використані інструменти для роботи з ключами шифрування і API ключами.

Практична значимість висновків і рекомендацій: результати магістерської роботи дозволять інтегрувати блокчейн технології в лікарнях і медичних закладах.

Місце роботи та посада рецензента: Business Ecosystem Engineering, Lead Blockchain Developer

Підпис засвідчую: Lead Blockchain Developer

(посада, підпис)



ЗМІСТ

ВСТУП.....	3
РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ	5
1.1 Характеристика предметної області	5
1.2 Аналіз існуючих блокчейн систем в медицині	8
1.3 Постановка проблеми	10
1.4 Обґрунтування вибору підходів і технологій для створення ІС	12
РОЗДІЛ 2 ПОСТАНОВКА ПРОБЛЕМИ ТА ОБГРУНТУВАННЯ ПІДХОДІВ І ТЕХНОЛОГІЙ ДЛЯ ПРОЄКТУВАННЯ ІС	20
2.1 Характеристика об’єкта дослідження.	20
2.2 Структура і характеристика СШ та їх компонентів.....	25
2.3 Методи, моделі і моделювання процесів і елементів складних систем	34
2.3.1 Методи дослідження й синтезу компонент систем штучного інтелекту.	34
2.3.2 Моделі та методи оптимізації в СШ	40
2.3.3 Методи та моделі управління в СШ	50
РОЗДІЛ 3 ПРОЄКТУВАННЯ ІС І ЇЇ КОМПОНЕНТІВ.....	53
3.1 Моделювання та проєктування бази знань для системи прийняття.....	53
3.2 Розроблення користувацького інтерфейсу. Елементи та структура.....	62
3.3 Проєктування забезпечувальних підсистем СШ. Реалізація системи.	69
3.3.1 Інформаційне забезпечення	69
3.3.2 Програмне забезпечення	72
3.3.3 Технічне забезпечення.....	73
3.3.4 Організаційно-економічне забезпечення.....	74
ВИСНОВКИ.....	78

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ.....	80
ДОДАТКИ.....	84
Додаток А.....	84
Додаток Б.....	92
Додаток В.....	119
Додаток Г.....	123
Додаток Д.....	127
Додаток Е.....	129
Додаток Є.....	133

ВСТУП

Актуальність теми: зростання обсягу медичної інформації та необхідність забезпечення її надійного зберігання та захисту висувають нові вимоги до медичних систем. Технологія блокчейн, завдяки своїй децентралізованій природі та високому рівню безпеки, відкриває нові можливості для створення інтелектуальних медичних карток. Інтеграція генеративного штучного інтелекту, зокрема ChatGPT, здатна значно полегшити процес заповнення медичних карток, підвищуючи точність та ефективність роботи медичного персоналу.

Аналіз останніх досліджень і публікацій: вже існують багато наукових робіт такі як [29] та [30], котрі розглядають використання блокчейну у медичній індустрії. Одна із наукових робіт «Конфіденційність і безпека блокчейну в охороні здоров'я: застосування, виклики та майбутні перспективи» [29] надихнула автора на реалізацію подібного бачення використання блокчейну в медицині. Дана робота описує можливості використання блокчейну в медицині для різних сценаріїв, але головним питанням даної роботи було вирішення проблеми з приватністю медичних даних, бо блокчейн, через свою особливість архітектури, відкритий і будь-хто може прочитати дані з будь-якої точки блокчейну. Тому і було прийнято рішення вирішити цю проблеми використавши алгоритм шифрування RSA.

Мета і завдання дослідження: метою кваліфікаційної магістерської роботи є розробка інтелектуальної медичної картки з використанням блокчейн технології та чат-бота ChatGPT. Основні завдання дослідження включають:

- аналіз існуючих підходів до зберігання та обробки медичних даних;
- розробка архітектури системи інтелектуальної медичної картки;
- інтеграція ChatGPT для автоматизації заповнення медичних карток;
- оцінка безпеки та ефективності розробленої системи.

Об'єкт дослідження: інформаційні процеси, що характеризують процеси формування медичних карток пацієнтів закладів охорони здоров'я.

Предмет дослідження: технологія блокчейну, система генеративного штучного інтелекту ChatGPT та їх використання для формування захищеної інтелектуальної медичної картки пацієнта.

Методи дослідження: у роботі використані методи аналізу та синтезу, системного підходу, моделювання та експериментального дослідження. Застосовано також методи криптографічного захисту даних.

Теоретична, методична та практична значущість отриманих результатів: теоретична значущість дослідження полягає у визначенні принципів інтеграції блокчейн технології та ChatGPT у медичні системи. Методична значущість полягає у розробці нових методів зберігання та обробки медичних даних. Практична значущість роботи виявляється у можливості підвищення ефективності та безпеки медичних інформаційних систем.

Апробація результатів дослідження: результати дослідження можна впровадити як експеримент, для визначення ефективності використання блокчейну в медичній галузі. Якщо експеримент буде вдалим, результати дослідження зможуть кардинально змінити медичну галузь.

Інформаційна база дослідження: інформаційною базою дослідження є наукові праці, і практичні приклади використання блокчейну в медицині.

Структура роботи: робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків.

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Характеристика предметної області

Індустрія охорони здоров'я (її також називають медичною галуззю або економікою охорони здоров'я) — це об'єднання та інтеграція секторів у рамках економічної системи, яка надає товари та послуги для лікування пацієнтів із лікувальною, профілактичною, реабілітаційною та паліативною допомогою. Вона охоплює створення та комерціалізацію продуктів, які сприяють покращенню якості надання послуг медичного характеру. Сучасний сектор охорони здоров'я включає три фундаментальні аспекти, а саме послуги, продукти та фінанси. Його можна далі поділити на численні сектори та категорії, і він спирається на міждисциплінарні команди висококваліфікованих професіоналів для задоволення потреб охорони здоров'я як окремих осіб, так і громад [1].

Індустрія охорони здоров'я є однією з найбільших і швидкозростаючих галузей у світі. Споживаючи понад 10 відсотків валового внутрішнього продукту (ВВП) більшості розвинених країн, охорона здоров'я може становити величезну частину економіки країни. У 2021 році витрати на охорону здоров'я в США зросли на 2,7 відсотка, досягнувши 4,3 трильйона доларів, або 12 914 доларів на людину. Як частка валового внутрішнього продукту країни, витрати на охорону здоров'я становили 18,3 відсотка. Витрати на душу населення на охорону здоров'я та фармацевтичні препарати в країнах ОЕСР неухильно зростали з кількох сотень у 1970-х роках до середнього 4000 доларів США на рік за поточним паритетом купівельної спроможності [1].

Блокчейн — це розподілена база даних або книга, яка спільно використовується вузлами комп'ютерної мережі. Вони найбільш відомі своєю ключовою роллю в системах криптовалют для підтримки безпечного та децентралізованого запису транзакцій, але вони не обмежуються тільки цим використанням. Блокчейн можна використовувати, щоб зробити дані в будь-якій галузі незмінними — термін, який використовується для опису неможливості змінити [2].

Оскільки неможливо змінити блок, єдина довіра потрібна до точки, де користувач або програма вводять дані. Цей аспект зменшує потребу в довірених третіх сторонах, якими зазвичай є аудитори чи інші люди, які додають витрат і роблять помилки [2].

У 2023 році стан блокчейн-галузі характеризувався значними фінансовими інвестиціями та інноваціями, що є важливим чинником для магістерського проєкту, спрямованого на розробку медичної картки на основі блокчейну. Зокрема, загальний обсяг залучених коштів у крипто- та блокчейн-індустрії склав близько \$7.96 мільярдів у 2023 році, хоча це й було зниженням порівняно з попереднім роком, коли було залучено \$29 мільярдів. Сектор блокчейн-сервісів отримав найбільшу частку інвестицій, майже \$6.5 мільярдів, що свідчить про високий інтерес інвесторів до розвитку цієї технології. Крім того, зростання кількості власників криптовалют до 575 мільйонів свідчить про зростаючу популярність та прийняття блокчейн-технологій у всьому світі. Це забезпечує солідну основу для інтеграції блокчейну в різні сфери, включаючи медицину, що є центральною темою цієї магістерської роботи [3].

Мета роботи – розробка інтелектуальної медичної картки на основі блокчейну. Це включає створення цифрової системи, яка дозволяє зберігати, захищати та управляти медичними даними пацієнтів. Система повинна забезпечувати легкий доступ до даних для уповноважених користувачів, при цьому гарантуючи безпеку та конфіденційність інформації.

Медична картка на основі блокчейну відіграє критичну роль у сучасній охороні здоров'я. Вона спрощує управління даними пацієнтів, підвищує ефективність обміну інформацією між лікарями та медичними установами, знижуючи при цьому ризик помилок та підвищуючи якість лікування.

Медична картка призначена для зберігання важливої медичної інформації про пацієнтів, включаючи анамнез, діагнози, результати аналізів, інформацію про лікування та вакцинації. Це сприяє покращенню якості медичного обслуговування, забезпечує точність медичних записів та підвищує рівень безпеки персональних даних.

У сучасному світі, де кількість медичних даних постійно зростає, важливо забезпечити їх безпечне зберігання та швидкий доступ. Блокчейн надає можливість створити систему, яка є одночасно безпечною, прозорою та не піддається зовнішньому впливу.

Медична картка буде зберігати наступну інформацію про пацієнта:

- медичні записи пацієнтів;
- інформація про лікування та процедури;
- результати лабораторних аналізів;
- історія призначень ліків;
- дані про алергії та вакцинації.

Вхідна інформація включає нові результати аналізів, діагнози, оновлення стану пацієнта, історію призначень ліків. Ця інформація вноситься у систему лікарем чи від імені поліклініки, де вона стає доступною для уповноважених осіб, забезпечуючи актуальність медичних даних.

Вихідні дані використовуються для діагностування, планування лікування, моніторингу стану пацієнтів та прийняття медичних рішень. Це також включає звіти для пацієнтів, статистичні дані для досліджень та аналіз ефективності лікування.

Інформація оновлюється в реальному часі або згідно з регламентованими медичними процедурами. Результати доступні миттєво для уповноважених користувачів, гарантуючи своєчасне та ефективне використання медичних даних.

1.2 Аналіз існуючих блокчейн систем в медицині

Блокчейн в охороні здоров'я використовується для всього, починаючи від захисту даних пацієнтів і закінчуючи управлінням ланцюгом поставок фармацевтичних препаратів.

І це не могло статися в кращий час: у 2021 році Сполучені Штати витратили майже 20 відсотків свого ВВП на охорону здоров'я, і витрати, ймовірно, зростуть, оскільки високі витрати на лікарні, неефективна практика та постійні витоки даних продовжують негативно впливати на галузь. Ці проблеми стимулюють прагнення до більшої ефективності та інновацій [4].

Завдяки своїй здатності дешево зберігати дані, захищати дані пацієнтів і покращувати загальний досвід охорони здоров'я, використання блокчейну може позитивно вплинути на охорону здоров'я. Ця технологія вже використовується для виконання будь-яких завдань: від надійного шифрування даних пацієнтів до боротьби зі спалахами шкідливих захворювань [4].

Далі наведено декілька існуючих проєктів побудованих на основі технології блокчейн і котрі використовуються в сфері охорони здоров'я.

Medicalchain використовує блокчейн для створення електронних медичних записів, забезпечуючи безпеку та прозорість у веденні медичної інформації. Система дозволяє пацієнтам контролювати доступ до своїх даних, підтримуючи одночасно конфіденційність та доступність для медичних фахівців. Medicalchain використовує подвійну блокчейн-структуру, поєднуючи Hyperledger Fabric для управління доступом до записів та Ethereum для підтримки додатків та сервісів. Платформа також пропонує телемедичні консультації та ринок даних, де користувачі можуть ділитися своїми медичними даними за винагороду. Основні переваги включають підвищену безпеку, кращий доступ до даних, та здатність пацієнтів контролювати власну медичну інформацію [5].

Guardtime пропонує інноваційні блокчейн-рішення, які можуть бути корисними для медичної галузі. Їх платформа включає аудит медичних даних, управління згодою пацієнтів та забезпечення відповідності нормативним вимогам, що є ключовим для медичних установ. Також вони пропонують інструменти для

агрегації реальних медичних даних без порушення конфіденційності, що може бути корисним для фармацевтичних компаній. Ці рішення підвищують безпеку, точність та прозорість обробки медичних даних, що є важливим для курсової роботи на тему медичної картки на основі блокчейну [6].

Avaneer Health зосереджена на створенні цифрової екосистеми охорони здоров'я з використанням технології блокчейн. Їх ініціатива спрямована на покращення взаємодії та обміну даними між різними зацікавленими сторонами у сфері охорони здоров'я, включаючи постачальників послуг, платників та споживачів. Підхід Avaneer зосереджується на безпечному та ефективному обміні даними, покращуючи координацію догляду та результати для пацієнтів. Вони прагнуть вирішити проблеми, такі як розрізненість даних та неефективні процеси у галузі, створюючи більш зв'язану та прозору систему [7].

Chronicle. Компанія розробляє проєкт MediLedger Network, який надає рішення для ланцюга поставок фармацевтичних товарів, щоб відповідати законодавству США щодо відстеження лікарських засобів. У MediLedger кожна транзакція у ланцюзі поставок ліків перевіряється за допомогою блокчейну, а також походження кожного продукту. Система використовує протокол шифрування zero-knowledge zk-SNARKS для захисту конфіденційності транзакцій. Цей підхід не вимагає обміну інформацією між стороною, яка намагається підтвердити транзакцію, та стороною, яка надає докази перевірки, що збільшує рівень конфіденційності в системі [8].

Patientory – це розподілений додаток на базі технології блокчейн, який надає користувачам доступ до даних про їх стан здоров'я. Додаток інтегрує різні централізовані медичні системи обліку в єдину платформу. Patientory також працює зі розумними контрактами, які забезпечують безперервний цикл медичного обслуговування пацієнтів. Він об'єднує дані з таких систем ЕМЗ (електронних медичних записів), як Epic, Cerner, Allscripts, Meditech та інші. За допомогою мобільного додатку пацієнт створює індивідуальний профіль, який дає йому доступ до всіх його медичних даних, збережених на платформі блокчейну, що відповідає вимогам HIPAA. У додатку користувачі можуть безпосередньо зв'язуватися з

постачальниками послуг, а також обговорювати з іншими користувачами проблеми зі здоров'ям, симптоми або інші питання. Це дозволяє пацієнтам контролювати своє здоров'я навіть поза лікарнею [8].

Завершуючи огляд крипто-проектів у медичній галузі, можна зазначити, що блокчейн технологія має значний потенціал у сфері охорони здоров'я. Вона відкриває нові можливості для забезпечення безпеки, прозорості та ефективності в управлінні медичною інформацією та ланцюгами поставок. Проекти, які описані вище, демонструють різні способи використання цієї технології для покращення медичних послуг і взаємодії між усіма зацікавленими сторонами. Це підтверджує, що блокчейн може стати ключовим елементом у розробці інтелектуальних медичних карток, які забезпечать кращий догляд за пацієнтами і зменшать витрати в галузі охорони здоров'я.

1.3 Постановка проблеми

Медична картка на основі блокчейну — це передове рішення, яке покращує управління медичними даними. Вона створена для того, щоб забезпечити безпечний, прозорий та ефективний доступ до медичної інформації пацієнтів. Завдяки використанню блокчейну, ця система гарантує, що медичні записи залишаються незмінними, конфіденційними та доступними в будь-який час. Лікарі та медичні працівники можуть швидко отримувати доступ до медичних даних пацієнта, що значно покращує якість та швидкість лікування. Пацієнти також отримують більше контролю над своїми даними, маючи можливість легко надавати або відкликати доступ до своїх медичних записів. Така система сприяє підвищенню ефективності охорони здоров'я, знижує ризики пов'язані з втратою або фальсифікацією даних та відкриває нові можливості для досліджень і розвитку в медицині.

Для пацієнта використання медичної блокчейн картки буде містити наступний порядок дій:

- Реєстрація: пацієнт створює обліковий запис у системі, вводячи свої персональні дані та ідентифікаційну інформацію;

- Надання доступу лікарю: пацієнт вибирає конкретного лікаря та надає йому право додавати новий запис в медичну картку, котра являє собою розумний контракт;
- Оновлення інформації: лікар вносить дані про діагнози, лікування чи аналізи в картку пацієнта;
- Перегляд оновленої інформації: пацієнт переглядає оновлення та відстежує зміни у своїй медичній історії.

Для лікарів або лікарні в цілому, використання медичної блокчейн картки буде містити наступний порядок дій:

- Реєстрація через лікарню: лікар реєструється в системі через свою медичну установу, вказуючи свої професійні дані;
- Отримання прав на редагування: після надання доступу пацієнтом, лікар отримує можливість редагувати картку;
- Внесення змін: лікар вносить необхідну інформацію в картку пацієнта, таку як результати обстежень або план лікування.

Завдяки використанню сервісів які надають доступ до LLM, такі як ChatGPT, лікарі зможуть підвищити ефективність своєї роботи використовуючи автодоповнення. Цей функціонал схожий на Snippet в програмуванні.

Розробка медичної картки на основі блокчейну вимагає комплексного підходу, який включає об'єднання різноманітних технологій:

- Використання Solidity: Solidity є мовою програмування для написання розумних контрактів на Ethereum, яка ідеально підходить для реалізації медичної картки на блокчейні. Розумні контракти можуть використовуватись для автоматизації процесів, таких як верифікація медичних записів та управління доступом до них.
- Розробка серверної частини з NestJS: NestJS, потужний фреймворк для Node.js, буде використано для створення надійної серверної частини. Це дозволить ефективно інтегрувати блокчейн-технологію з традиційними системами управління даними.

- Клієнтська частина на React: React є популярним фреймворком для розробки користувацьких інтерфейсів. Він буде використаний для створення інтуїтивно зрозумілого та легкого у використанні інтерфейсу, що є важливим для забезпечення високого рівня сприйняття серед користувачів.
- Контроль версій: Git дозволяє вести контроль версій програмного коду додатків, що є критично важливим для командної роботи та забезпечення безперебійного процесу розробки.
- Використання ChatGPT для допомоги медичним працівникам: штучний інтелект, такий як ChatGPT, може бути інтегрований для автоматизації процесів заповнення медичних записів, включаючи результати аналізів та діагнози. Це може значно поліпшити ефективність та точність медичних записів.

В цілому, поєднання цих технологій дозволить створити медичну картку, яка є безпечною, надійною та зручною у використанні, відповідаючи потребам сучасної охорони здоров'я. Ключовим є підтримка конфіденційності пацієнтів та забезпечення легкого доступу до даних для медичних працівників.

1.4 Обґрунтування вибору підходів і технологій для створення ІС

Вибір правильної технології для реалізації проєкту є критично важливим, оскільки це впливає на ефективність, безпеку, масштабованість та підтримку проєкту. Наприклад, використання блокчейну забезпечує безпеку та незмінність даних, що є ключовими для медичних записів. Технології, такі як Solidity, NestJS, React, і Git, допомагають у створенні надійних, швидких та інтуїтивно зрозумілих систем. Вибір невідповідних технологій може призвести до проблем з безпекою, ефективністю та масштабуванням проєкту.

Solidity - мова програмування зі статичною типізацією, розроблена для розробки розумних контрактів, оскільки вона спеціально створена для платформи Ethereum, яка є однією з найпопулярніших блокчейн-платформ. Вона дозволяє створювати складні контракти з різноманітними функціями, включаючи транзакції

та взаємодію з іншими контрактами. Solidity забезпечує високий рівень безпеки та є зрозумілою для розробників, оскільки має синтаксис, схожий на JavaScript. Використання Solidity для розумних контрактів гарантує широку підтримку та розвиток в межах спільноти Ethereum. Використовуючи цю мову програмування, планується розробити розумні контракти, що будуть представляти собою медичну картку пацієнта. Ця медична картка буде зберігати список медичних записів у зашифрованому вигляді. Пацієнт буде надавати доступ лікарю на редагування старих записів або додавання нових. Також на блокчейні буде зберігатися реєстр адрес лікарів, котрим буде керувати DAO (Decentralized Autonomous Organization) [14].

NestJS – це фреймворк для написання серверних додатків, котрий розроблений на основі на Node.js. вибирають для серверної сторони проєктів, оскільки він забезпечує ефективну структуру для розробки надійних серверних додатків. NestJS використовує мову програмування TypeScript, що підвищує надійність коду та полегшує його підтримку. Цей фреймворк інтегрується з широким спектром бібліотек, що робить його гнучким та масштабованим. NestJS підтримує чисту архітектуру, що сприяє кращій організації коду та впровадженню різноманітних патернів програмування. Використовуючи даний фреймворк планується розробити бізнес логіку платформи, котра буде дозволяти користувачам, тобто пацієнтам і лікарям, безпечно обмінюватися медичними даними. Звичайно, серверна частина буде відповідати за такі обов'язкові функції, як авторизація, автентифікація і взаємодія з блокчейном.

React – це бібліотека для розробки веб-сторінок. Вона є популярним вибором для розробки клієнтської сторони завдяки її багатому функціоналу і гнучкості. React дозволяє створювати інтуїтивно зрозумілі користувацькі інтерфейси. Він використовує компонентний підхід, що полегшує повторне використання коду. Використовуючи одну з реалізацій React, а саме фреймворк NextJS планується розробити веб-сторінку, на котрій лікар зможе заповнювати медичну картку пацієнта з підтримкою доповнення тексту використовуючи

ChatGPT. Також веб-сторінка буде використовуватися для взаємодії з блокчейном використовуючи розширення для браузера Metamask [15].

Git – це система контролю версій, котра дозволяє розділяти код на версії і функціонал, що дозволяє розробникам працювати над декілька частинах проекту і не заважати один одному. Він є ідеальним вибором для контролю версій у проєктах завдяки його надійності, гнучкості та широкій підтримці спільноти. Git дозволяє ефективно керувати змінами в коді, сприяючи співпраці в команді та запобігаючи конфліктам у коді. Це робить процес розробки більш організованим і ефективним, особливо у великих проєктах з багатьма учасниками. Використання Git також забезпечує безпеку та історію всіх змін, що є важливим для відстеження прогресу проєкту.

ChatGPT як AI помічник для лікарів вибирають через його здатність до обробки природної мови та надання швидких, точних відповідей. Він може допомогти медичним працівникам в аналізі даних, уточненні діагнозів та наданні рекомендацій щодо лікування. Такий помічник забезпечує підтримку в прийнятті рішень, економлячи час та зусилля лікарів, і може вдосконалювати якість пацієнтського догляду.

Життєвий цикл проєкту охоплює кілька ключових етапів:

1. Ініціація, де визначаються основні цілі та обсяг проєкту;
2. Планування, яке включає розробку детального плану дій;
3. Виконання, де команда або один розробник працює над розробкою проєкту;
4. Моніторинг і контроль, що дозволяє відстежувати прогрес та вносити корективи;
5. Закриття, яке включає завершення всіх робіт та передачу проєкту замовнику.

Важливо забезпечити гнучке управління на кожному етапі для адаптації до змін та непередбачуваних викликів.

Вибір Solidity, NestJS, React, Git та ChatGPT підходить для всього життєвого циклу проєкту медичної картки на блокчейні. Solidity забезпечує безпеку та

надійність розумних контрактів у блокчейні. NestJS та React дозволяють розробляти ефективні та користувацько-приємні серверні та клієнтські частини відповідно. Git покращує управління версіями та співпрацю в команді. ChatGPT може оптимізувати взаємодію з медичними даними. Ці технології разом створюють комплексне рішення, яке відповідає потребам проекту на кожному етапі його розвитку.

Ведення проектної документації є важливим для успішного управління проектом. Воно дозволяє забезпечити прозорість, легше управління змінами та поліпшити комунікацію всередині команди. Адекватна документація також сприяє легшій передачі знань новим членам команди та допомагає у вирішенні спірних питань. Вона служить як довідковий матеріал, що відображає рішення, процеси та стандарти проекту.

Проектна документація через .md (Markdown) файли в репозиторії GitHub ведеться в структурованому текстовому форматі. Markdown дозволяє легко формувати заголовки, списки, посилання та інші елементи, роблячи документацію зрозумілою та читабельною. Кожен файл .md може містити описи функцій, інструкції для розробників, архітектурні діаграми та іншу важливу інформацію. Це сприяє ефективній комунікації та співпраці всередині команди. GitHub також забезпечує інструменти для відстеження змін та версій документації.

Для реалізації медичної блокчейн-картки, методологія, яка поєднує в собі мікросервісну архітектуру та об'єктно-орієнтоване проектування, може бути найбільш підходящою. Переваги мікросервісної архітектури:

- Модульність та інкапсуляція (об'єктно-орієнтоване проектування): кожен мікросервіс можна розробляти як окремий модуль з використанням принципів об'єктно-орієнтованого проектування. Кожен модуль інкапсулює свою логіку та дані, експонуючи лише необхідний інтерфейс для інших частин системи. Це сприяє чіткому розділенню відповідальності та легшій підтримці.

- Незалежність розробки (мікросервісна архітектура): кожен мікросервіс може бути розроблений, тестований та розгорнутий незалежно від інших. Це дозволяє командам працювати паралельно над різними частинами системи, прискорюючи процес розробки та впровадження змін.
- Перевикористання коду (об'єктно-орієнтоване проектування): спільні функції та компоненти можуть бути оформлені як повторно використовувані об'єкти або класи. Це зменшує дублювання коду та полегшує управління змінами.
- Легкість масштабування (мікросервісна архітектура): оскільки кожен мікросервіс є незалежним, систему можна легко масштабувати шляхом додавання або вилучення екземплярів окремих сервісів відповідно до потреб.
- Спрощення підтримки та оновлень (об'єктно-орієнтована та мікросервісна архітектура): оновлення та внесення змін у систему спрощуються, оскільки зміни можна вносити окремо в кожному мікросервісі, не впливаючи на роботу всієї системи.
- Поліпшення надійності (мікросервісна архітектура): у разі збою в одному сервісі, інші можуть продовжувати функціонувати безперебійно. Це підвищує загальну надійність системи.

Однак, мікросервісна архітектура також має і слабкі сторони, такі як:

- Складність у тестуванні: мікросервісна архітектура створює значну проблему для тестування через її розподілений характер. Кожен мікросервіс зазвичай розробляється та розгортається незалежно, що означає, що тестування має охоплювати не лише окремі сервіси, але й їх взаємодію. Тестування інтеграції стає складнішим із збільшенням кількості послуг. Крім того, забезпечити послідовність даних у службах для цілей тестування може бути складно без комплексної та скоординованої стратегії тестування.

- Складність розробки та розгортання: мікросервіси вимагають ретельної координації між кількома командами розробників. Архітектура вимагає високого рівня складності в практиках розгортання, таких як безперервна інтеграція та безперервне постачання, щоб керувати незалежним розгортанням численних служб.
- Збільшення затримки мережі та проблеми зі зв'язком: зв'язок між службами через мережу створює затримку. Зі збільшенням кількості мікросервісів сукупна затримка може значно вплинути на загальну продуктивність системи. Ефективне керування зв'язком і підтримка низької затримки є постійними проблемами.
- Проблеми з цілісністю даних і керуванням. Кожна мікрослужба може керувати власною базою даних, що призводить до проблем узгодженості та цілісності даних. Впровадження транзакцій, які охоплюють кілька служб, є складним і часто вимагає прийняття остаточної узгодженості щодо властивостей ACID, що може не відповідати всім вимогам бізнесу.
- Управління ресурсами. Мікросервіси можуть призвести до неефективного використання ресурсів, оскільки кожна служба може масштабуватися незалежно залежно від навантаження, що може призвести до вищих операційних витрат і збільшення складності моніторингу та керування.
- Питання безпеки: кожен мікросервіс є потенційною точкою входу для порушень безпеки. Складність захисту кількох інтерфейсів і забезпечення безпечного зв'язку між службами збільшує накладні витрати на безпеку.
- Операційні накладні витрати. Керування кількома службами, кожна з яких має власний життєвий цикл, може призвести до значних операційних накладних витрат. Потреба в складних інструментах для

моніторингу, реєстрації та усунення несправностей великої кількості служб, що працюють одночасно, може потребувати ресурсів.

Тому на думку автора найкращим варіантом для розробки MVP (Minimum viable product) є архітектура клієнт-сервер, на рисунку 1.1 ви можете побачити високо рівневу діаграму (HLD, High Level Diagram), котра описує дану архітектуру. На рисунку 1.1 ви можете побачити схематичне зображення даної архітектури.

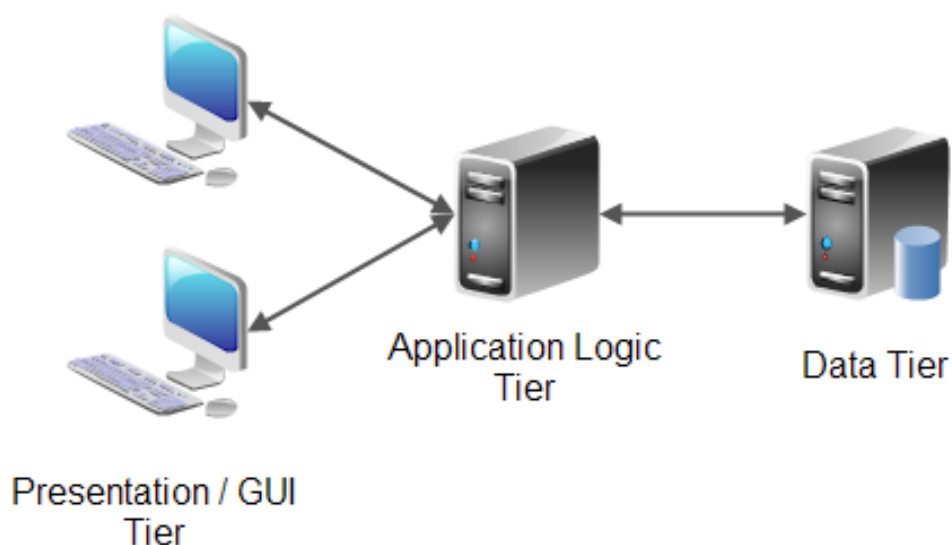


Рисунок 1.1 – Схема 3 рівневої архітектури клієнт-сервер [20]

Головними перевагами даної архітектури, котрі вплинули на її вибір, є:

- простота реалізації: основною перевагою клієнт-серверної архітектури є її проста конструкція, де ролі чітко розподілені між сервером (який надає ресурси чи послуги) та клієнтом (який отримує доступ до цих служб). Ця простота полегшує проектування, впровадження та керування, особливо для MVP, де метою є швидка перевірка бізнес-концепцій з мінімальною складністю.
- простота розробки: розробники можуть зосередитися на створенні функціонального інтерфейсу на стороні клієнта, одночасно зберігаючи центральний сервер, який керує бізнес-логікою та зберіганням даних. Такий розподіл завдань спрощує завдання розробки та дозволяє виконувати швидкі ітерації на стороні клієнта,

не впливаючи на логіку сервера, що має вирішальне значення для ітераційного характеру розробки MVP.

Підсумовуючи, для реалізації магістерської роботи були обрані наступні технології: Solidity, NestJs, React, Git. Їх буде поєднувати архітектура клієнт-сервер.

РОЗДІЛ 2 ПОСТАНОВКА ПРОБЛЕМИ ТА ОБГРУНТУВАННЯ ПІДХОДІВ І ТЕХНОЛОГІЙ ДЛЯ ПРОЄКТУВАННЯ ІС

2.1 Характеристика об'єкта дослідження.

Як було зазначено в попередньому розділі, ми будемо використовувати архітектуру клієнт-сервер. З назви архітектури зрозуміло, що в нас буде три основних частини: це сервер, котрий буде зберігати необхідну інформацію для авторизації, автентифікації і інших бізнес задач, фронт-енд – веб-сторінка, на котрій пацієнт і лікар будуть взаємодіяти один з одним, а також база даних. Звісно в нас є блокчейн частина, котра виконує одну з головних ролей в нашому додатку, будемо вважати, що вона є частиною фронт-енду, бо основна комунікація з блокчейн мережами буде виконуватися на веб-сторінці.

Для узагальнення далі наведено опис кожної частини і з чого вона буде складатися.

- Серверна частина: вона буде складатися з усього 2 частин, це REST API, через який веб-сторінка буде отримувати всю необхідну інформацію і шифрувати конфіденційну інформацію та SQL база даних, котра буде зберігати інформацію про пацієнтів, блокчейн адреси їх гаманців, і дані, котрі будуть необхідні для шифрування даних і які будуть зберігатися на блокчейні.
- Фронт-енд: він буде складатися з наступних модулів:
 - Комунікації з блокчейном, використовуючи гаманець Metamask або одну з його альтернатив;
 - Інтерфейс користувача, через котрий пацієнт зможе переглядати свої медичні дані, а лікар створювати нові;
 - Інтеграція з ChatGPT, котра буде допомагати лікарям в заповненні медичної картки і створенні нових записів.

Для збереження даних було прийнято рішення використовувати базу даних PostgreSQL.

PostgreSQL, також відома як Postgres, — це безкоштовна система керування реляційною базою даних (RDBMS) із відкритим вихідним кодом, яка наголошує на розширюваності та сумісності з SQL. PostgreSQL містить транзакції з властивостями атомарності, узгодженості, ізоляції, довговічності (ACID), автоматично оновлювані подання, матеріалізовані подання, тригери, зовнішні ключі та збережені процедури. Вона підтримується в усіх основних операційних системах, включаючи Linux, FreeBSD, OpenBSD, macOS і Windows, і обробляє низку робочих навантажень від окремих машин до сховищ даних або веб-сервісів із багатьма одночасними користувачами [9].

PostgreSQL буде використовуватися для зберігання важливої інформації, як дані користувачів, адреси їх медичних карток на блокчейні і т.д.

В якості середовища розробки будуть використані текстовий редактор VS Code та фреймворк для тестування розумних контрактів Hardhat.

Visual Studio Code, також відомий як VS Code, — це редактор вихідного коду, розроблений Microsoft для Windows, Linux і macOS. Функції включають підтримку налагодження, підсвічування синтаксису, інтелектуальне завершення коду, фрагменти, рефакторинг коду та вбудований Git. Користувачі можуть змінювати тему, комбінації клавіш, параметри та встановлювати розширення, які додають функціональність [13].

В опитуванні розробників Stack Overflow 2023 серед 86 544 респондентів Visual Studio Code назвали найпопулярнішим інструментом середовища розробника, при цьому 73,71% повідомили, що вони ним користуються. Опитування також показало, що Visual Studio Code частіше використовують ті, хто вчиться програмувати, ніж професійні розробники (78% проти 74%) [13].

Hardhat — це середовище розробки програмного забезпечення Ethereum. Він складається з різних компонентів для редагування, компіляції, налагодження та розгортання розумних контрактів і dApps, які разом створюють повне середовище розробки [14].

RPC-провайдер (Remote Procedure Call provider) у контексті блокчейн технології, є провайдером віддалених викликів процедур, який діє як міст, що

з'єднує клієнтів децентралізованих додатків DApps, з блокчейн мережею. RPC-провайдер сприяє комунікації та взаємодії з блокчейном, дозволяючи відправляти транзакції, запитувати стан блокчейну та слухати події без необхідності використання клієнтом повноцінного вузла блокчейну [15].

На практиці, коли розробники створюють DApps, вони інтегрують RPC-провайдера для обробки складних завдань прямої взаємодії з блокчейном. Ця інтеграція виконується через бібліотеку або API, який пропонує RPC-провайдер, такий як Web3.js або Ethers.js у екосистемі Ethereum. RPC-провайдер обробляє виклики, здійснені з DApp, і перенаправляє їх до мережі блокчейну. Потім він повертає відповідь мережі назад у DApp, дозволяючи взаємодію так, ніби клієнт є вузлом у мережі [16].

Важливість RPC-провайдера у розробці та експлуатації DApps неможливо переоцінити. Він значно спрощує процес розробки, абстрагуючи деталі мережевого протоколу та забезпечуючи більш доступний інтерфейс для розробників. Для кінцевих користувачів DApps, RPC-провайдери сприяють швидкій та ефективній взаємодії з блокчейном, роблячи користувацький досвід більш гладким та подібним до традиційних веб-додатків. Це важливо для адаптації та функціональності DApps у різних секторах, оскільки дозволяє користувачам взаємодіяти з технологіями блокчейну інтуїтивно та ефективно без необхідності глибокого технічного розуміння основної інфраструктури блокчейну [16].

Отже, RPC-провайдери є життєво необхідними для зв'язку складної інфраструктури блокчейну та користувацьких веб-інтерфейсів, роблячи їх фундаментальним компонентом у екосистемі децентралізованих додатків.

На рисунку 2.1 ви можете побачити високорівневу діаграму компонентів системи.

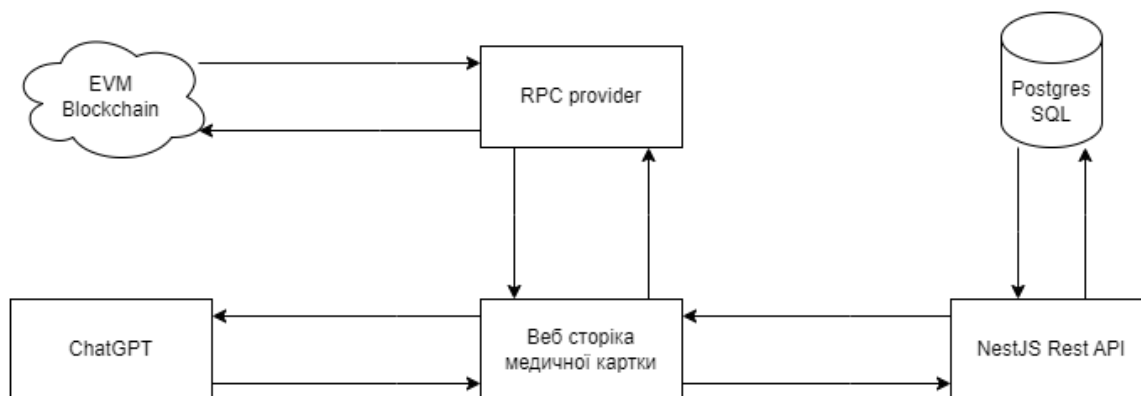


Рисунок 2.1 Високорівнева діаграма компонентів системи

Джерело: Сформовано автором

В якості RPC-провайдера ми будемо використовувати популярний сервіс Infura. Він був обраний через те, що надає доступ до найбільш популярних блокчейн мереж, і що найголовніше, в нього є безкоштовна підписка з великим обсягом транзакцій на день, що нам підходить.

Visual Studio Code (VS Code) — це безкоштовний відкритий редактор коду, розроблений Microsoft. Він підтримує різні мови програмування та технології, що робить його універсальним інструментом для розробників. Оснащений вбудованою підтримкою Git, інтегрованим терміналом, а також можливістю розширення за допомогою плагінів, VS Code забезпечує гнучкість та ефективність у робочому процесі розробки. Його інтерфейс забезпечує зручне управління робочими просторами та файлами, а також відмінну підтримку для відладки і рефакторингу коду[21]. Оскільки нам необхідно створювати різні компоненти для різних цілей, VS Code чудово для цього підходить. Треба встановити необхідні плагіни для цих задач.

Для розробки розумних контрактів будуть встановлені розширення: «Solidity», «Solidity Visual Developer».

Для розробки веб-сторінки використовуючи React і NextJS будуть встановлені розширення: «Nextjs Snippets», «ES7+ React/Redux/React-Native/JS snippets».

Для розробки серверної частини будуть встановлені розширення: «VScode NestJs Snippets», «REST Client».

Для загального використання і збільшення продуктивності написання коду будуть встановлені розширення: «ESLint», «Prettier - Code formatter». Ці розширення будуть формувати написаний код відповідно до Code Convention [22] і стилю коду. Це необхідно щоб збільшити читабельність коду.

Забезпечення безпеки медичних записів має велике значення через конфіденційний характер особистої та медичної інформації, яку вони містять. Конфіденційність, цілісність та доступність медичних записів є критично важливими для збереження довіри пацієнтів і забезпечення відповідності глобальним нормам конфіденційності, таким як HIPAA [17] у Сполучених Штатах. Порушення або несанкціонований доступ до цих записів може призвести до серйозних порушень конфіденційності та потенційної шкоди для пацієнтів, тому важливо мати надійні заходи безпеки.

Наш додаток буде вирішувати ці питання безпеки, використовуючи шифрування RSA для захисту медичних записів, які зберігаються на блокчейні.

Алгоритм шифрування Rivest-Shamir-Adleman (RSA) є асиметричним алгоритмом шифрування, який широко застосовується в багатьох продуктах та послугах. Асиметричне шифрування використовує пару ключів, які математично пов'язані для шифрування та розшифрування даних. Створюються приватний і публічний ключі, де публічний ключ доступний будь-кому, а приватний ключ залишається таємницею, відомою тільки творцю пари ключів. За допомогою RSA можна шифрувати дані як приватним, так і публічним ключем, тоді як інший ключ виконує розшифрування. Це одна з причин, чому RSA є найбільш використовуваним асиметричним алгоритмом шифрування [23].

Можливість шифрування даних за допомогою або приватного, або публічного ключа надає користувачам RSA різноманітні сервіси. Якщо для шифрування використовується публічний ключ, то для розшифрування необхідно застосувати приватний ключ. Цей метод ідеально підходить для передачі конфіденційної інформації через мережу чи Інтернет, коли одержувач даних надсилає відправнику свій публічний ключ. Відправник шифрує конфіденційні дані цим ключем і відсилає їх одержувачу. Оскільки дані були зашифровані

публічним ключем, лише власник приватного ключа може їх розшифрувати. Таким чином, тільки призначений одержувач може розшифрувати дані, навіть якщо вони були перехоплені під час передачі [23]. На рис 2.2 зображено алгоритм подібної передачі зашифрованого повідомлення.

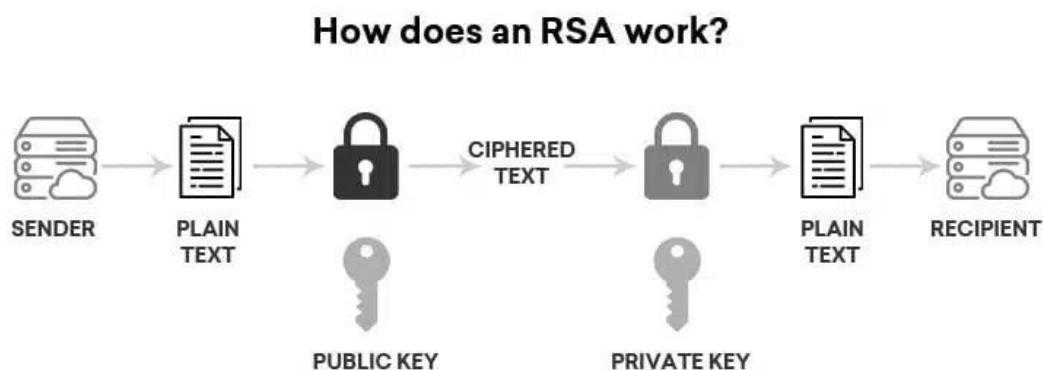


Рисунок 2.2 Алгоритм RSA [23]

Інший метод асиметричного шифрування з використанням RSA полягає в шифруванні повідомлення приватним ключем. У цьому випадку відправник шифрує дані своїм приватним ключем та відсилає зашифровані дані разом із своїм публічним ключем одержувачу. Одержувач може розшифрувати дані за допомогою публічного ключа відправника, що дозволяє підтвердити автентичність відправника. Хоча дані можуть бути викрадені та прочитані під час передачі, основна мета цього типу шифрування полягає у підтвердженні особи відправника. Якщо дані були викрадені та змінені під час передачі, публічний ключ не зможе розшифрувати нове повідомлення, і тому одержувач зрозуміє, що дані були змінені [23].

Завдяки тому, що пацієнти генерують власні пари ключів RSA та діляться лише публічним ключем з медичними працівниками, це означає що кожен медичний запис буде шифрується перед його записом на блокчейн. Цей метод гарантує, що медичні записи можуть бути розшифровані та доступні лише індивідуально пацієнтом за допомогою його приватного ключа, тим самим захищаючи інформацію пацієнтів від несанкціонованого доступу та порушень.

2.2 Структура і характеристика СШІ та їх компонентів

Для розуміння структури і характеристики розробляємої СШІ нижче наведені діаграми для спрощення сприйняття інформації.

Для того щоб визначитися з вимогами для ІС (інформаційної системи), нам необхідні її варіанти використання, на рисунку 2.2 можна побачити діаграми прецедентів. В ній описано можливості використання системи лікарем і пацієнтом.

Можливості лікаря:

1. Реєстрація
2. Авторизація
3. Отримання медичної інформації пацієнта від пацієнта
4. Верифікація медичної інформації пацієнта
5. Створення нового медичного запису
6. Заповнення медичної картки з допомогою штучного інтелекту
7. Запис інформації медичної картки на блокчейн.

Можливості пацієнта:

1. Реєстрація
2. Авторизація
3. Перегляд інформації медичної картки
4. Експорт інформації медичної картки у вигляді PDF файлу
5. Надання інформації медичної картки лікарю
6. Керування RSA ключами
7. Надання доступу лікарю на редагування інформації медичної картки



Рисунок 2.3 Діаграма прецедентів

Джерело: Сформовано автором

На блокчейні ми буде взаємодіяти з розумними контрактами, такі як HealthCardFactor, HealthCard, DoctorRegistry, MedicalGovernance.

HealthCardFactor – розумний контракт, котрий буде відповідати за створення контрактів HealthCard.

HealthCard – розумний контракт медичної картки, в котрому буде зберігатися зашифровані медичні записи пацієнта. Саме з ним будуть взаємодіяти більшу частину часу лікар і пацієнт.

DoctorRegistry – розумний контракт, котрий буде зберігати адреси усіх лікарів, для забезпечення того, щоб пацієнт не надав доступ на редагування особистих даних шахрайській особі.

MedicalGovernance – контракт, що реалізує логіку DAO (Decentralized Autonomous Organization) [18]. Він буде відповідати за додання або видалення лікарів з реєстру, а саме з контракту DoctorRegistry.

На рисунку 2.4 зображено їх взаємодія між собою і з користувачем.

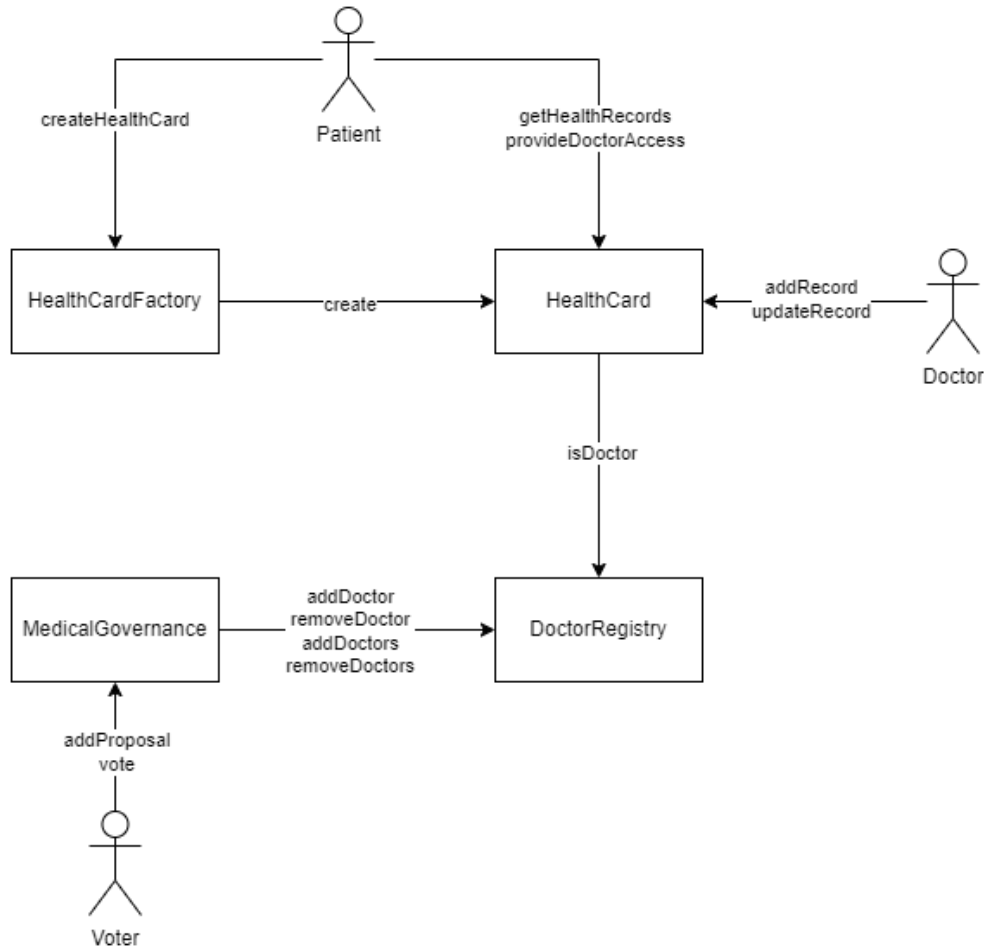


Рисунок 2.4 Високорівнева діаграма розумних контрактів

Джерело: Сформовано автором

Далі необхідно визначити в якій послідовності користувачі (пацієнта і лікарі) будуть взаємодіяти з системою, а також як компоненти системи будуть взаємодіяти між собою.

На рисунку 2.5 ви можете побачити процес авторизації шляхом генерації JWT (JSON Web Token) та його передання на сторону клієнта. Далі він буде використовуватися для автентифікації.

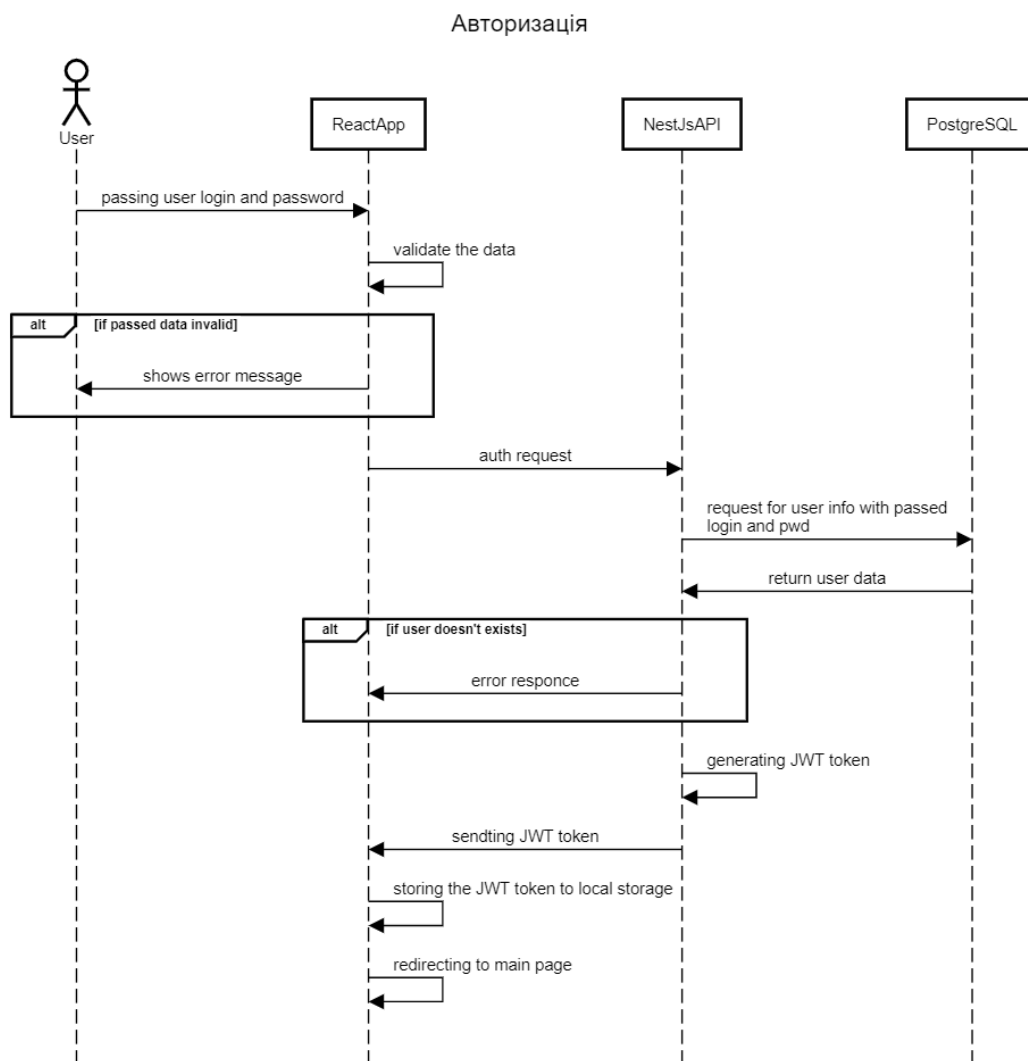


Рисунок 2.5 Діаграма послідовності – Авторизація

Джерело: Сформовано автором

Автентифікація буде проходити шляхом передання JWT токена разом з запитом до серверної частини на надання даних. На рисунку 2.6 зображений цей процес.

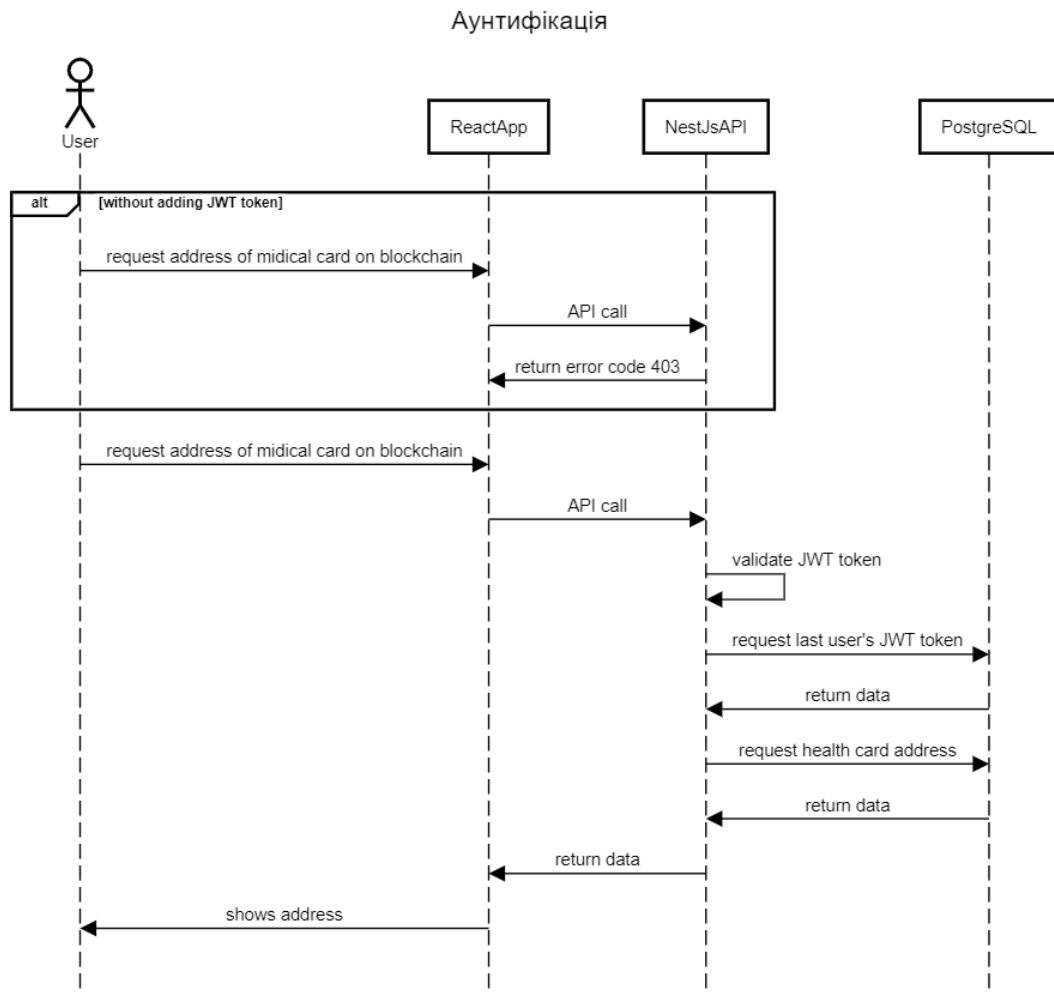


Рисунок 2.6 Діаграма послідовності – Автентифікація

Джерело: Сформовано автором

Для того щоб користувач міг авторизуватися, спочатку йому необхідно зареєструватися. Для реєстрації будуть використовуватися наступні дані:

1. Ім'я, Прізвище, По-батькові
2. Номер телефону
3. Електронна пошта
4. Пароль
5. Країна проживання
6. Вік
7. Роль (Пацієнт або лікар)

8. Якщо реєструється лікар, йому необхідно описати свої навички, вказавши його напрямок, досвід роботи, опис послуг, котрі він надає і блокчейн адресу, з котрою буде асоціюватися його особа.

На рисунку 2.7 зображений процес реєстрації.

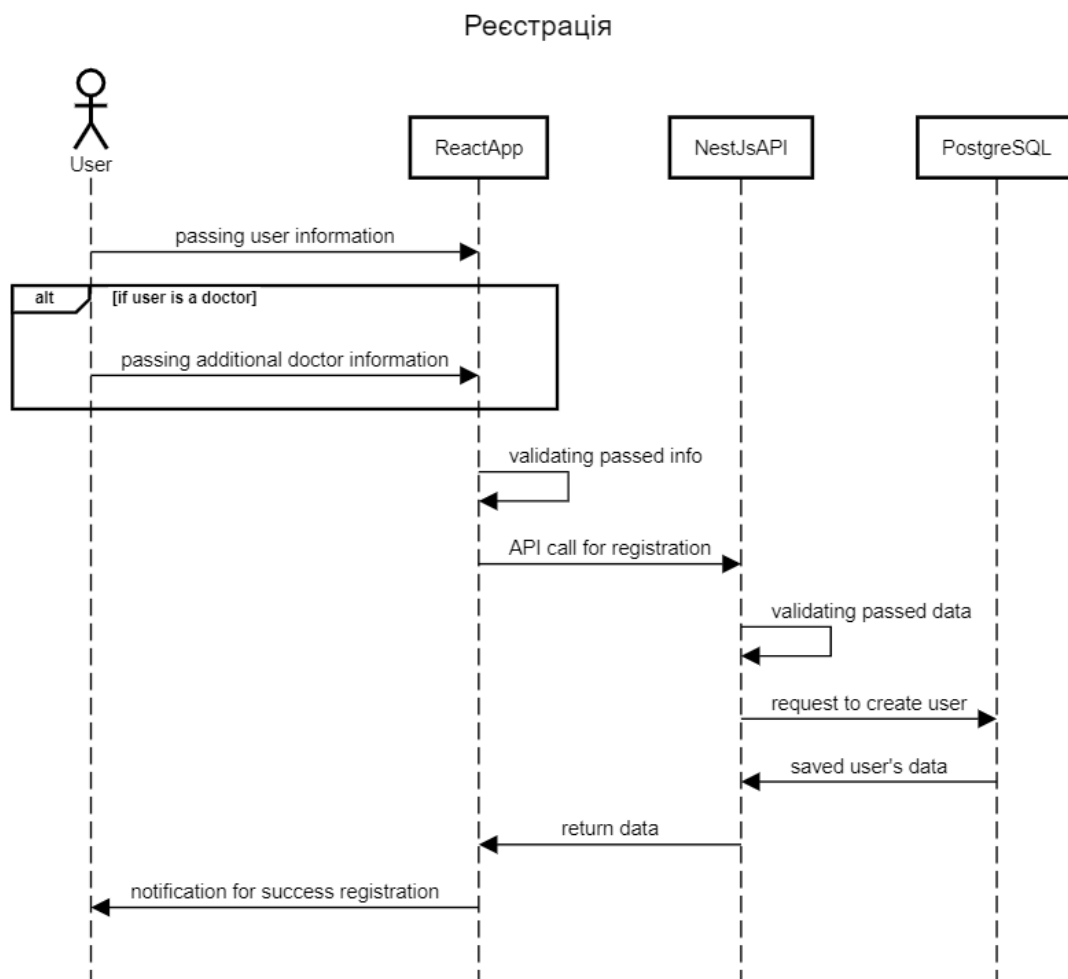


Рисунок 2.7 Діаграма послідовностей – реєстрація

Джерело: Сформовано автором

Для того, щоб почати записувати і зберігати дані медичної картки, необхідно розгорнути розумний контракт медичної картки на блокчейн мережу. Для цього пацієнт повинен мати власний блокчейн гаманець, на котрому є валюта для оплати транзакцій, наприклад для мережі Ethereum це ЕТН, а для мережі Polygon це MATIC. Щоб підключити гаманець до веб-сторінки, необхідно щоб користувач завантажив розширення Metamask або Coinbase Wallet. Для створення нового контракту медичної картки користувач буде звертатися до контракту

HealthCardFactory. Можна буде створювати 1 медичну картку на один гаманець. На рисунку 2.8 зображено процес розгортання медичної картки на блокчейні.



Рисунок 2.8 Діаграма послідовностей – створення медичної картки на блокчейні

Джерело: Сформовано автором

Далі розглянемо процес взаємодії пацієнта і клієнта. Для того щоб почати взаємодіяти пацієнту і лікарю, необхідно щоб пацієнт знайшов блокчейн адресу лікаря. Це можна буде зробити через профіль лікаря на веб-сторінці. Після цього пацієнт надає доступ цій адресі, викликавши функцію «provideDoctorAccess». Пацієнт може встановити до якої години лікар буде мати доступ на редагування або оновлення одного запису. На рисунку 2.9 описується послідовність виконання даної операції.



Рисунок 2.9 Діаграма послідовностей – надання доступу лікарю

Джерело: Сформовано автором

Тепер лікар має доступ на додання нового запису або редагування старого в медичній карті на блокчейні.

Щоб створити новий запис або оновити старий, лікар буде використовувати інтерфейс веб-сторінки для написання тексту, з доступом до інструментів штучного інтелекту. Після написання тексту, лікар повинен узяти публічний ключ пацієнта з його профілю і зашифрувати дані використовуючи функціонал веб-сторінки. Після чого він може викликати функцію «addNewHealthRecord» контракту HealthCard.

На рисунку 2.10 можна побачити процес створення і запису нової інформації в медичну картку пацієнта.

Створення нового запису і зберігання його на блокчейні

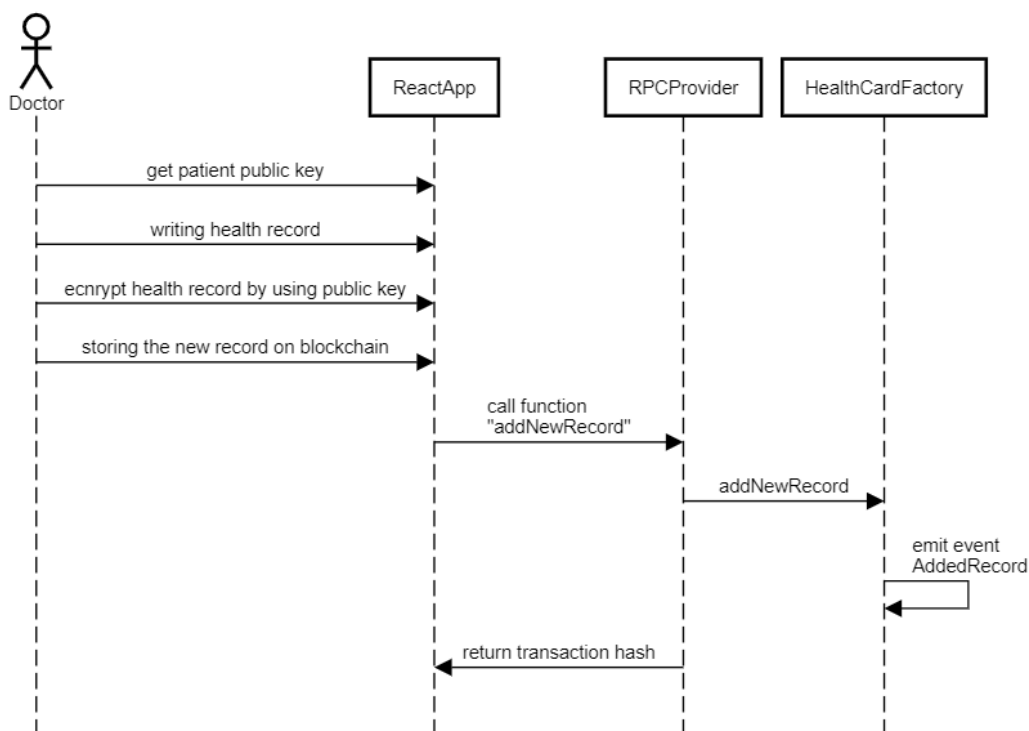


Рисунок 2.10 Діаграма послідовностей – створення нового запису і запис його на блокчейн

Джерело: Сформовано автором

Підсумовуючи, ми розглянули основні сценарії використання інформаційної системи. Зрозуміли як вони будуть ком'юнікувати між собою і взаємодіяти. І тепер ми можемо більше детально розглянути взаємодію з LLM (Large Language Model) ChatGPT.

2.3 Методи, моделі і моделювання процесів і елементів складних систем

2.3.1 Методи дослідження й синтезу компонент систем штучного інтелекту

Згідно джерелу [10], ChatGPT — це мовна модель, розроблена дослідницькою організацією зі штучного інтелекту OpenAI, яка є однією з провідних у цій галузі. Вона базується на архітектурі Transformer, яка зробила революцію у сфері обробки природної мови. Ця модель була навчена на величезній кількості даних, що дозволяє їй генерувати текст та реагувати на різноманітні запити з точністю та правильністю, що можна порівняти з людськими.

Архітектура Transformer розроблена для обробки послідовних даних, таких як текст. Вона була представлена у статті 2017 року під назвою «Attention is All You

Need». Архітектура Transformer базується на механізмах самоуваги, які дозволяють моделі оцінювати важливість різних частин вхідної послідовності при здійсненні прогнозів. На рисунку 2.11 ви можете побачити діаграму, котра відображає архітектуру Transformer.

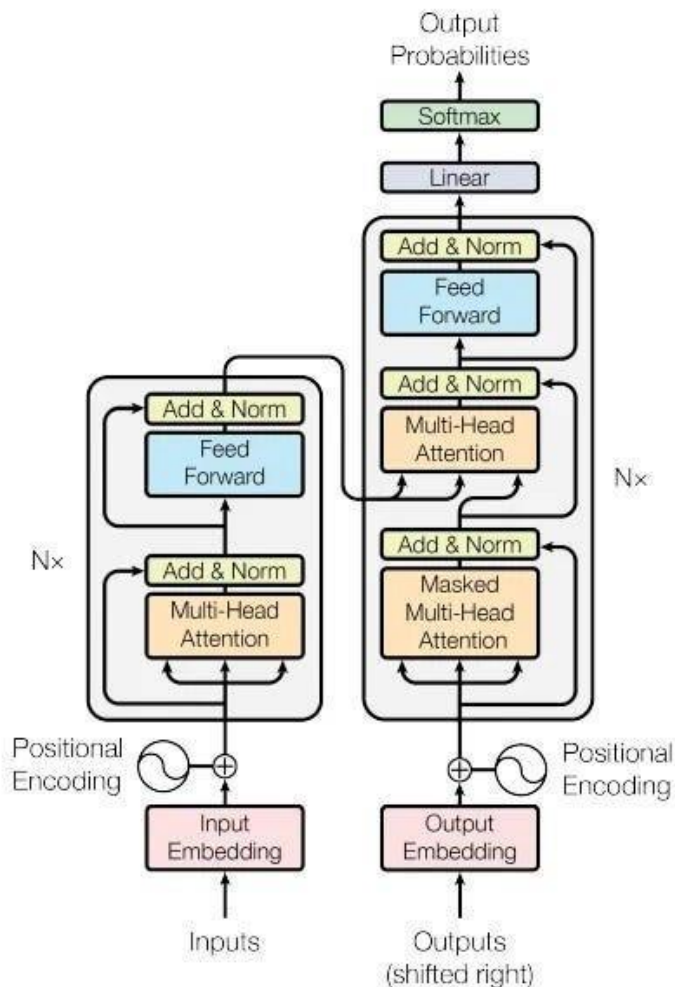


Рисунок 2.11 Архітектура Transformer [10]

Як ви бачите на рисунку 2.11, архітектура складається з наступних складових [10]:

- Вхідні дані (Inputs): вхідна послідовність є послідовністю токенів, які представляють текстовий вхід.
- Вкладені дані (Input Embedding): перший крок у трансформації полягає в перетворенні вхідної послідовності у матрицю векторів, де кожен вектор представляє токен у послідовності. Перетворення тексту

у матрицю векторів зображено на рисунку 2.11. Цей процес називається вкладенням даних. Цей шар відображує кожен токен у високовимірний вектор, який уловлює семантичне значення токена.

- **Механізм самоуваги (Self-Attention Mechanism):** механізм самоуваги дозволяє моделі обчислювати відносини між різними частинами вхідної послідовності. Він складається з трьох кроків: обчислення запиту, ключа та значення, а також обчислення уваги. На кроці обчислення запиту, ключа та значення вектори входу перетворюються на три різні представлення за допомогою лінійних перетворень. На етапі обчислення уваги модель обчислює зважену суму значень, де ваги базуються на схожості між представленнями запиту та ключа. Зважена сума представляє вихід механізму самоуваги для кожної позиції у послідовності.
- **Багатопоточна самоувага (Multi-Head Self-Attention):** архітектура Transformer використовує багатопоточну самоувагу, яка дозволяє моделі зосереджуватися на різних частинах вхідної послідовності та обчислювати відносини між ними паралельно. У кожному потоці обчислення запиту, ключа та значення виконуються з різними лінійними перетвореннями, а результати об'єднуються та перетворюються у нове представлення.
- **Повнозв'язна мережа (Feedforward Network):** вихід з механізму багатопоточної самоуваги подається до повнозв'язної мережі, яка складається з серії повністю зв'язаних шарів та функцій активації. Повнозв'язна мережа перетворює представлення на остаточний вихід.
- **Нормалізація шару (Layer Normalization):** активації в кожному шарі архітектури Transformer нормалізуються за допомогою нормалізації шару, що допомагає стабілізувати процес тренування та запобігти перенавчанню моделі. Залишкове з'єднання, за яким слідує

нормалізація шару, допомагає стабілізувати процес тренування та полегшує тренування моделі.

- Кодування позицій (Positional Encoding): для врахування порядку токенів у вхідній послідовності до вкладення входу додається кодування позицій. Кодування позицій — це вектор, який представляє позицію кожного токена в послідовності.
- Шари накладання (Stacking Layers): архітектура Transformer може бути стекована, утворюючи глибоку нейронну мережу шляхом повторення механізму багато-поточної самоуваги та повнозв'язної мережі кілька разів.
- Вихідна інформація (Output): остаточний вихід з Transformer, який є векторним представленням вхідної послідовності.

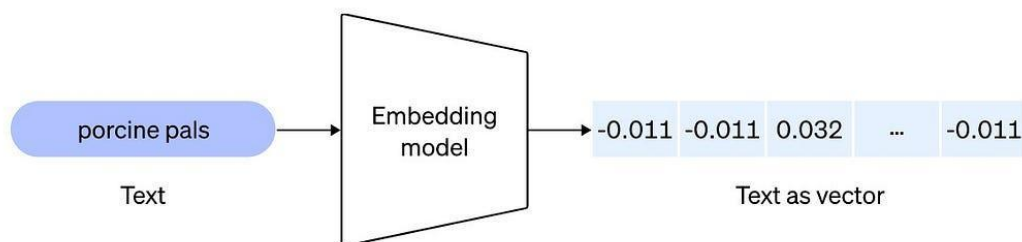


Рисунок 2.12 Перетворення вхідного тексту в матрицю векторів [10]

Згідно джерела [10], ChatGPT використовує архітектуру нейронної мережі та некероване навчання для генерації відповідей. Це означає, що він може навчитися генерувати відповіді без потреби у чіткій вказівці щодо правильної відповіді, що робить його потужним інструментом для виконання широкого спектру завдань у розмовах.

Модель навчається за допомогою навчання з підсиленням від людського зворотного зв'язку (RLHF), використовуючи ті ж методи, що і GPT, але з незначними відмінностями в налаштуванні збору даних. Спочатку модель навчається за допомогою тонкого налаштування з учителем: людські тренери штучного інтелекту надають діалоги, у яких вони грають обидві ролі —

користувача та помічника штучного інтелекту. Після цього тренери отримують доступ до пропозицій, написаних моделлю, щоб допомогти їм скласти свої відповіді. Потім цей новий набір діалогів змішується з набором даних InstructGPT, який трансформується у формат діалогу, як показано на діаграмі нижче.

Найефективніший спосіб тренування ChatGPT включає безпосередню взаємодію з моделлю та надання зворотного зв'язку щодо її відповідей. Така взаємодія допомагає покращити продуктивність моделі, удосконалюючи її здатність генерувати більш точні та відповідні відповіді.

Роль, котру буде виконувати ChatGPT є доповнення існуючого тексту лікаря чи його створення з нуля. Для того, щоб мовна модель згенерувала текст з нуля, лікар повинен поставити діагноз. Якщо лікар не встановив діагнозу, веб-сторінка не буде звертатися до мовної моделі і з'явиться повідомлення про помилку. На рисунку 2.13 можна побачити приклад помилки.

The image shows a web form titled "Health Record Creation" with several input fields: "Connect Wallet", "Generate Health Record", "Name: Tom", "Surname: Handsome", "Patronymic: Patric", "Gender: Male", "Birth Date: 05/15/2024", "Phone Number: 3801111111", and "Diagnosis:". A dark modal box is overlaid on the right side of the form, containing the text "localhost:3000 says You must make a diagnosis!!! ChatGPT can't do it for you" and an "OK" button.

Рисунок 2.13 Приклад помилки про відсутність діагнозу

Джерело: Сформовано автором

NestJS відповідає за авторизацію, автентифікацію користувачів, за доступ до інформації про пацієнтів і лікарів, бо вона необхідна для взаємодії між лікарем і пацієнтом. Також якщо у пацієнта буде бажання, він може зберігати свій публічний і приватний RSA ключ, в зашифрованому вигляді.

Шифруватися пароль і приватний RSA ключ користувача будуть використовуючи алгоритм SHA256.

Алгоритм SHA-256 (Secure Hash Algorithm 256-bit) — це криптографічний алгоритм хешування, який створює унікальний, фіксований розмір 256-бітного (32 байта) хеш-значення з вхідних даних будь-якого розміру. Він належить до сімейства SHA-2, розробленого Національним інститутом стандартів і технологій (NIST). SHA-256 працює, поділяючи вхідні дані на блоки по 512 бітів і обробляючи кожен блок за допомогою серії математичних операцій, які включають бітові зсуви та логічні функції. Результатом є унікальний хеш, використання якого поширене у цифровій безпеці для перевірки цілісності даних та у системах, які вимагають стійкості до змін, таких як блокчейн і системи цифрового підпису [24].

Розумні контракти, котрі будуть розгорнуті на блокчейні будуть слугувати як місце збереження даних пацієнта і керування доступів хто може ці зміни вносити. Усього в нас буде 3 основні контракти:

1. HealthCardFactory – контракт, котрий буде використовуватися пацієнтом для створення своєї власної медичної картки на блокчейні. Контракт HealthCardFactory буде створювати новий контракт HealthCard і призначати адресу пацієнта як власника цього контракту.
2. HealthCard - представляє саму медичну картку пацієнта, на котрій і будуть зберігатися дані. Пацієнт може читати дані за цієї картки і розшифровувати їх використовуючи свій приватний ключ. Якщо пацієнт не зберігає приватний ключ на серверній частині, він може передати його на веб-сторінку для розшифрування.
3. DoctorRegistry - розумний контракт, котрий виконує функції реєстру лікарів. Він потрібен, щоб забезпечити відсутність сценаріїв, коли пацієнт надає доступ шахраю. За керування цим контрактом буде відповідальне DAO. В якості DAO ми будемо використовувати готовий сервіс Snapshot.

Snapshot — це децентралізований сервіс для управління голосуванням, який широко використовується в DAO (децентралізованих автономних організаціях). Він дозволяє користувачам голосувати за різні пропозиції без потреби здійснювати транзакції на блокчейні, що знижує витрати на gas і забезпечує швидке

голосування. Голосування в Snapshot не впливає безпосередньо на блокчейн, оскільки використовує оффчейн механізм для запису голосів. Це означає, що користувачі можуть підтримувати або відхиляти пропозиції за допомогою цифрових підписів, які потім агрегуються і зберігаються поза ланцюгом. Результати відображаються в реальному часі і забезпечують прозорість та ефективність у прийнятті рішень у межах DAO [25].

Веб-сторінка буде відповідати за взаємодію користувачів з блокчейном, перегляд інформації про лікарів, пацієнтів, за створення медичних записів з допомогою ChatGPT. Функціонал, котрий буде на веб-сторінці наступний:

1. Реєстрація
2. Вхід в систему
3. Підключення до блокчейну з використанням Metamask
4. Перегляд інформації про лікарів
5. Перегляд інформації про пацієнтів
6. Надання доступу лікарю на додання запису
7. Створення нового запису
8. Створення нового запису з використанням ChatGPT
9. Читання даних з медичної картки

Веб-сторінка буде мати наступну структуру:

1. Головна сторінка
2. Реєстрація
3. Вхід
4. Перегляд інформації про лікаря
5. Перегляд інформації про пацієнта
6. Створення нового запису
7. Перегляд записів

Підсумовуючи, ми переглянули основні компоненти системи і розглянули їх основні функції і особливості.

2.3.2 Моделі та методи оптимізації в СШІ

ChatGPT можна налаштувати та оптимізувати відповідно до ваших потреб. Згідно джерелу [10], найкраща практики оптимізації роботи ChatGPT це *prompt engineering* (інженерія підказок або запитів). Завдяки ньому можна кастомізувати роботи ChatGPT, щоб вона відповідала потребам проекту чи користувача.

Prompt engineering - це процес у сфері штучного інтелекту, де спеціально розроблені запити або вхідні дані спрямовують мовні моделі для ефективнішого генерування бажаних результатів. Ця техніка оптимізує те, як модель інтерпретує та реагує на вхідні дані, покращуючи точність, релевантність та креативність відповідей [11].

Згідно джерела [11], з *prompt engineering* можна використовувати наступні техніки для покращення розуміння моделлю та якості вихідних даних.

- Техніка одноразового запиту (*Zero-shot prompting*) надає моделі машинного навчання завдання, на яке вона не була явно навчена. Це перевіряє здатність моделі виробляти відповідні вихідні дані без спираючих на попередні приклади.
- Техніка запитів з кількома прикладами (*Few-shot prompting*) або навчання в контексті дає моделі кілька зразків вихідних даних, щоб допомогти їй зрозуміти, чого від неї хоче запитувач. Модель краще розуміє бажаний вихід, якщо може опиратися на контекст.
- Техніка послідовного міркування (*Chain-of-thought prompting, CoT*) — це передова техніка, що надає моделі поетапне роз'яснення для наслідування. Розбивка складного завдання на проміжні кроки або "ланцюги міркувань" допомагає моделі краще розуміти мову та створювати більш точні вихідні дані.

Prompt складається з декількох елементів, нижче будуть описаний кожний компонент і його роль, використовуючи модель GPT-4.

Роль, в якості якої особи або суб'єкта виступає мовна модель. Наприклад, якщо ми задамо питання "Опиши протокол лікування Covid 19", то ChatGPT дасть

загальну відповідь, котра охопить лікування як легкої, так і складної форми вірусу. На рисунку 2.14 зображено приклад такого запиту.

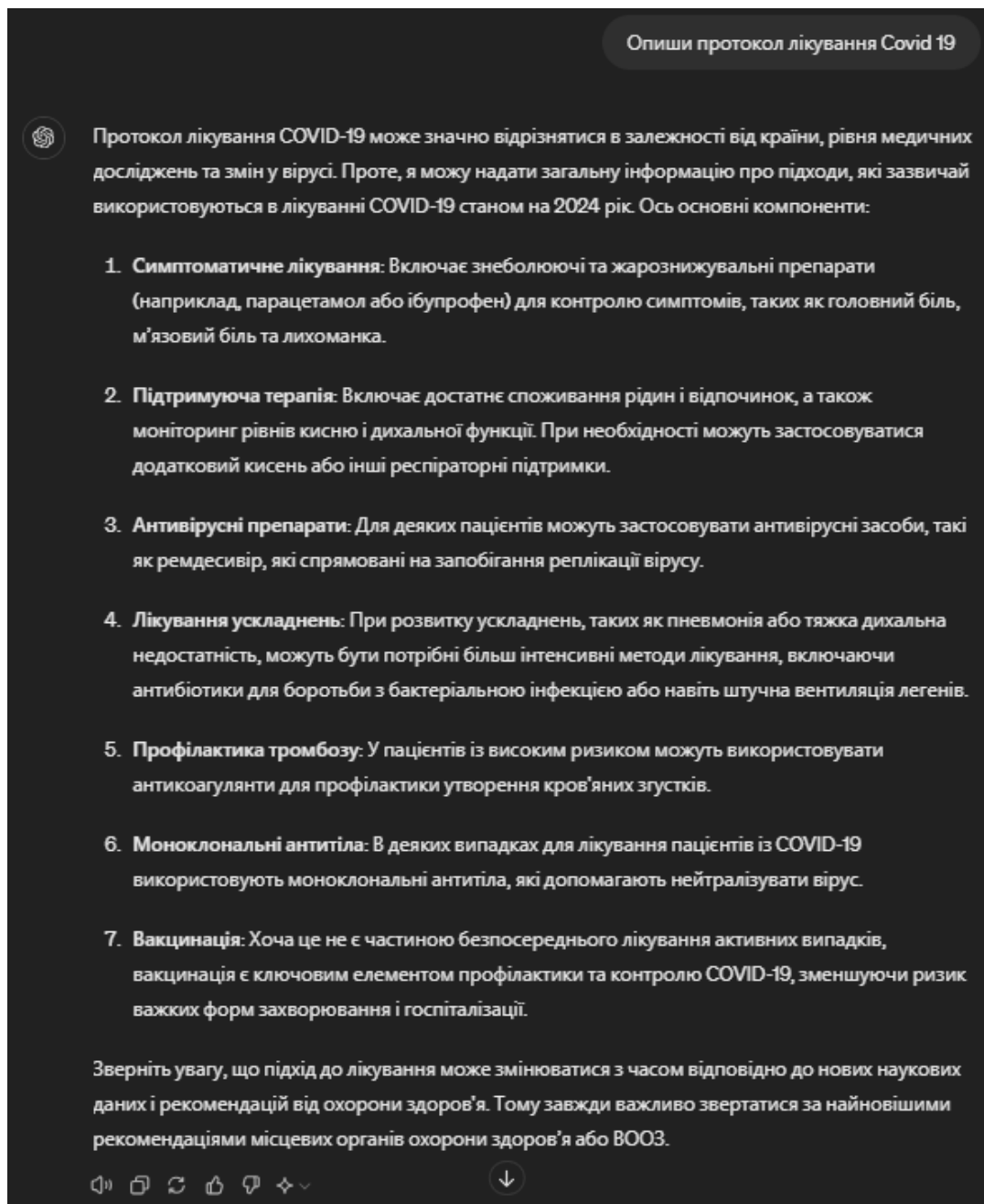


Рисунок 2.14 Приклад промпту без ролі

Джерело: Сформовано автором

Якщо додати до промпту роль “Поводься як сімейний лікар”, то ми отримуємо опис лікування легкої форми Covid 19, бо сімейні лікарі зазвичай мають справу саме з легкою формою вірусу, на рисунку зображено приклад такого промпту.

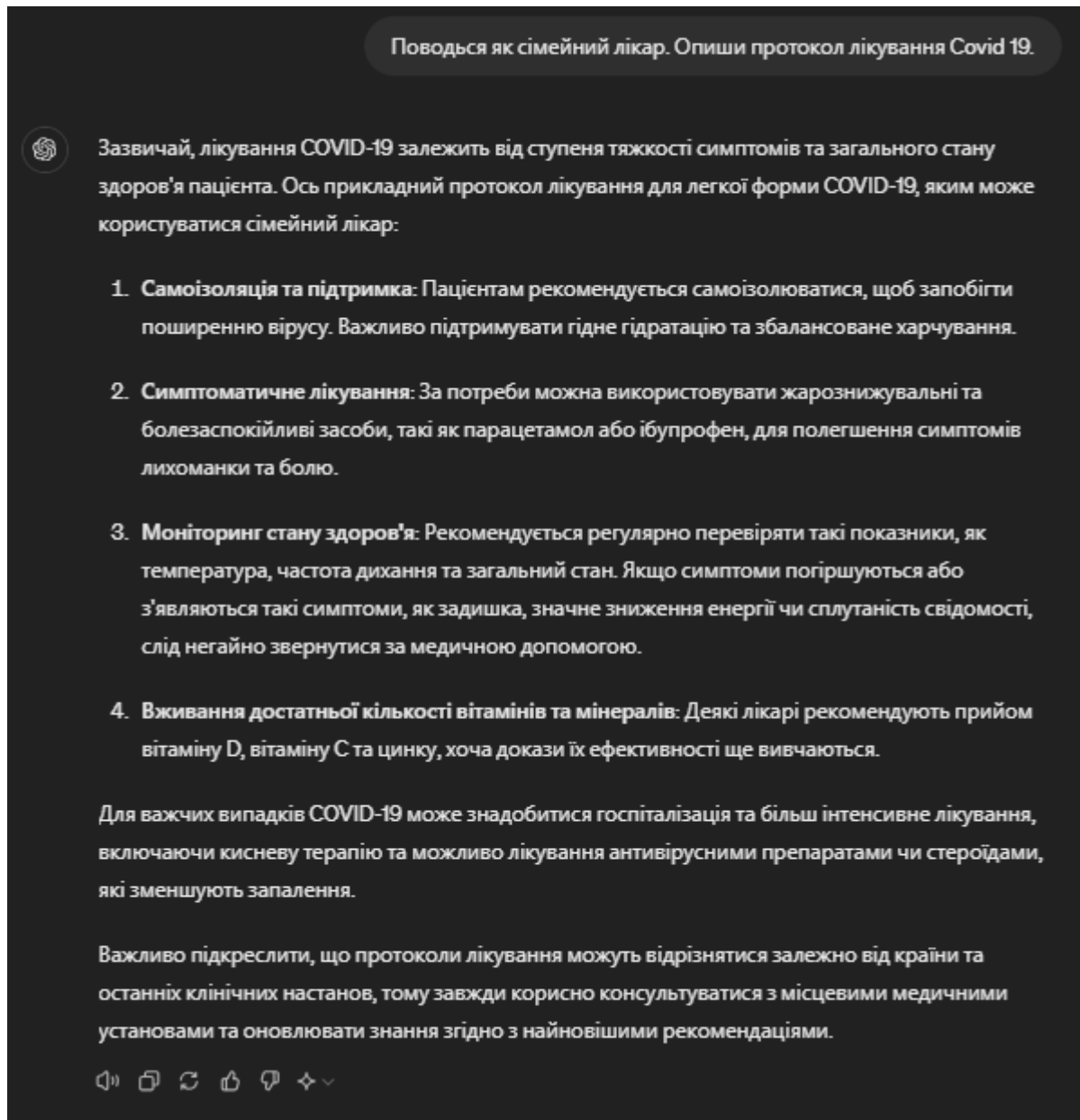


Рисунок 2.15 Приклад промпту з роллю

Джерело: Сформовано автором

Контекст у *prompt engineering* відіграє важливу роль, оскільки він забезпечує релевантність та адекватність відповідей штучного інтелекту на запити користувача. В контексті програмування взаємодії з моделями штучного інтелекту, контекст не тільки визначає семантичне оточення, у якому формулюється запит, але й включає історію попередніх взаємодій, специфіку домену знань та навіть культурні та індивідуальні особливості користувача. Це дозволяє системі краще адаптуватися до потреб користувача, забезпечувати більш точне та персоналізоване обслуговування. Наприклад, у медичній діагностиці,

контекстуальна інформація про попередні стани здоров'я пацієнта, його симптоми та історію лікування може допомогти інтелектуальній системі більш точно інтерпретувати запити лікарів або пацієнтів, покращуючи тим самим результати діагностики та лікування.

На практиці використання контексту в `prompt engineering` може бути продемонстровано через налаштування інтелектуальних помічників, які використовують інформацію про локацію, час доби та навіть конкретні запити користувача для видачі відповідей. Наприклад, інтелектуальний помічник, знаючи, що користувач знаходиться в кафе, може адаптувати відповіді на запит про рекомендації стосовно напоїв, враховуючи час дня та попередні вибори користувача. Це дозволяє не тільки збільшити задоволеність користувача від взаємодії з системою, але й підвищити його довіру до штучного інтелекту. Такий підхід відкриває нові можливості для персоналізації та оптимізації взаємодії людина-модель, що є ключовим у сучасних технологічних розробках.

Інструкції у `prompt engineering` визначають, як модель повинна інтерпретувати та реагувати на запити. Чіткі інструкції дозволяють забезпечити, що модель генерує відповіді, які точно відповідають вимогам і очікуванням користувача. Наприклад, якщо інструкція звучить як "Напиши короткий опис використання блокчейну у фінансових послугах", це направляє модель генерувати конкретний і зосереджений відгук на запитане питання, мінімізуючи можливість отримання несуттєвої чи надмірно широкої інформації. Це також сприяє збільшенню ефективності взаємодії з AI, оскільки точні інструкції зменшують потребу у багаторазових ітераціях запитів для досягнення бажаного результату. Таким чином, використання добре сформульованих інструкцій у `prompt engineering` не тільки поліпшує релевантність відповідей, але й підвищує загальну ефективність і задоволеність користувачів.

Дані служать основою для навчання моделей, визначають контекст взаємодії та допомагають моделі адаптуватися до специфічних запитів і сценаріїв використання. Наприклад, у розробці чат-ботів для обслуговування клієнтів, використання деталізованих даних про поведінку користувачів та їх попередні

запити дозволяє створити більш точні інструкції для моделей, що сприяє створенню більш персоналізованих та ефективних відповідей. Так, великі обсяги даних про попередні взаємодії дозволяють системі розпізнавати патерни запитань та відповідно адаптувати свої реакції, підвищуючи задоволеність користувачів та оптимізуючи загальний процес спілкування. Таким чином, якість і релевантність даних безпосередньо визначають успішність інтеграції штучного інтелекту в реальні бізнес-процеси та користувацькі додатки.

На рисунках 2.16 та 2.17 ви можете бачити приклади промпту з даними і без.

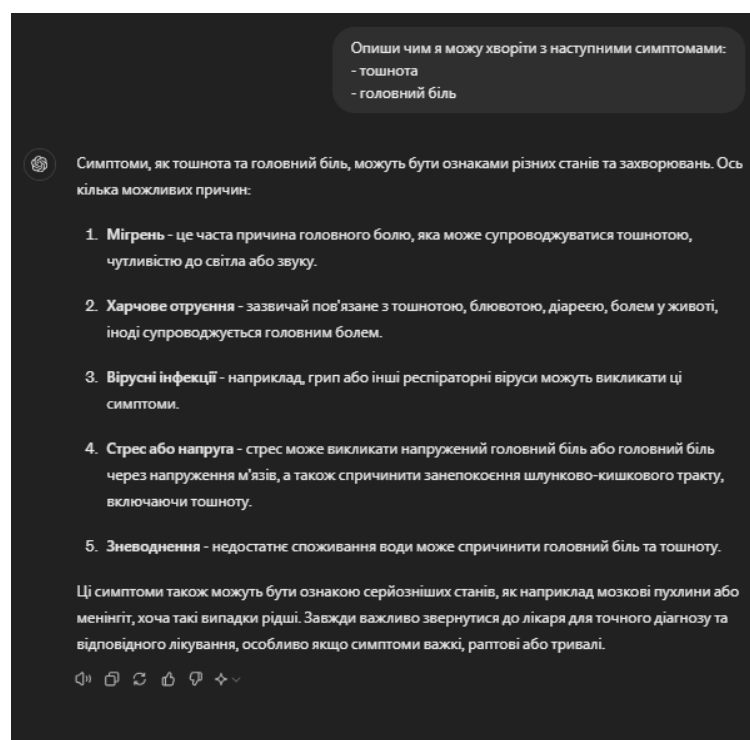


Рисунок 2.16 Приклад промпту з даними

Джерело: Сформовано автором

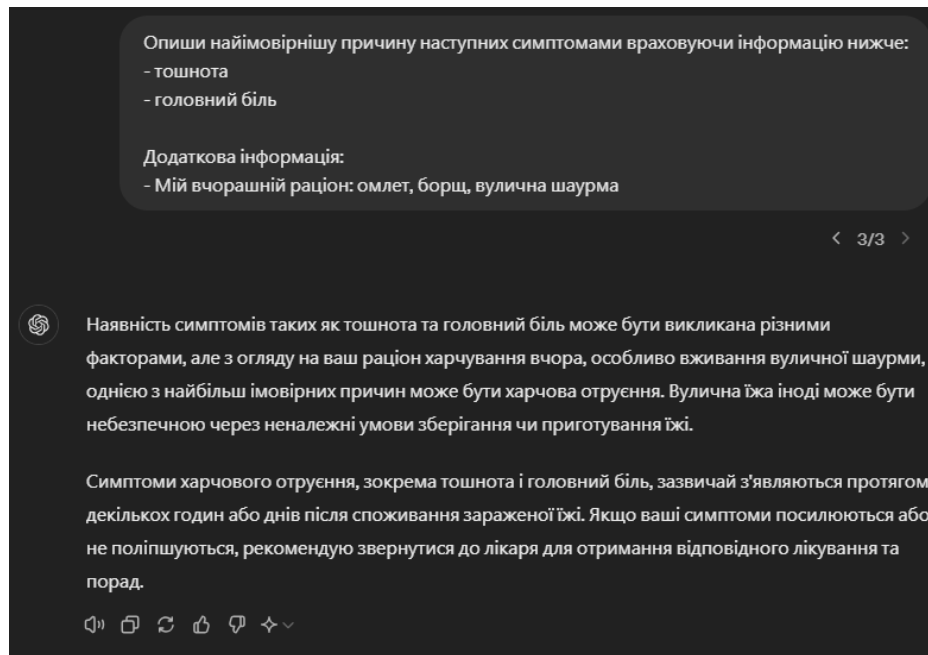


Рисунок 2.17 Приклад промпту з даними

Джерело: Сформовано автором

Опис вихідного результату в *prompt engineering* дозволяє направляти модель до вироблення бажаних відповідей, забезпечуючи більшу точність та відповідність до заданих критеріїв. Це особливо важливо у складних або спеціалізованих сценаріях використання, де необхідно отримати специфічну інформацію або відповідь певної форми. Наприклад, у розробці освітніх програм, опис вихідного результату може вимагати, щоб відповіді на історичні запити включали дати, персонажів і ключові події, що забезпечує збільшення навчальної цінності відповідей. У юридичних додатках, де потрібна точність у визначенні та тлумаченні законів, детальний опис вихідного результату може вказувати на необхідність включення статей законів, судової практики та прецедентів. Таким чином, ефективний опис вихідного результату не тільки підвищує точність відповідей, але й забезпечує їхню корисність і релевантність для кінцевого користувача, оптимізуючи взаємодію зі штучним інтелектом.

Приклади в *prompt engineering* служать як "шаблони" або "еталони" для генерації точних і релевантних відповідей моделлю. Через демонстрацію бажаних форматів та стилів відповідей, приклади допомагають моделі краще зрозуміти і виконувати конкретні запити. Наприклад, якщо моделі надаються зразки

оформлення відповідей на запитання для підготовки учнів до тестів (наприклад, формулювання відповідей з розгорнутим обґрунтуванням аргументів), це може значно покращити якість навчального контенту. Аналогічно, у бізнес-аналізі, де важлива деталізація даних, приклади коректного викладу аналітичних висновків можуть направити модель на створення структурованих і детальних відповідей, які відповідають професійним стандартам. Використання прикладів у `prompt engineering` таким чином забезпечує не тільки зрозумілість взаємодії з моделлю, але й сприяє досягненню більш високої релевантності та корисності відповідей в різних застосуваннях.

Також у API ChatGPT має додаткові опції, так як:

- **Temperature:** Цей параметр контролює ступінь випадковості (або творчості) у відповідях моделі. Чим вища температура, тим більш випадкові та непередбачувані будуть відповіді. З низькою температурою відповіді моделі будуть більш передбачуваними та консервативними.
- **Maximum Tokens:** Визначає максимальну кількість токенів (слів або символів), які модель може згенерувати як відповідь на ваш запит. Це допомагає контролювати довжину виходу та обмежує ресурси, потрібні для генерації тексту.
- **Stop Sequences:** Це попередньо визначені фрази або символи, які зупиняють модель під час генерації тексту. Це корисно для вказівки моменту завершення відповіді, наприклад, кінця абзацу або документу.
- **Top P:** Відомий також як "nucleus sampling". Це метод відбору наступних слів, при якому враховуються тільки ті слівники, які разом мають велику кумулятивну ймовірність (заданий відсоток P). Це дозволяє моделі зосередитися на більш ймовірних варіантах, обмежуючи менш релевантні чи нелогічні варіанти.

- Frequency Penalty: Цей параметр знижує ймовірність повторення вже використаних слів. Чим вище значення цього покарання, тим менше ймовірність того, що певне слово буде повторюватись, сприяючи більшій різноманітності у тексті.
- Presence Penalty: Цей параметр знижує ймовірність повторного з'явлення слів, які вже були використані. Відмінність від frequency penalty полягає у фокусуванні на запобіганні повторенню тих же слів, навіть якщо вони вже з'являлися лише один раз.

Кожен з цих параметрів дозволяє тонко налаштовувати поведінку моделі, що важливо для досягнення бажаних результатів у генерації тексту в різних сценаріях.

Для створення медичної картки ми будемо передавати наступні дані:

- стаття;
- дата народження;
- діагноз;
- опис;
- рецепт;
- коментарі.

Діагноз є обов'язковим, бо наша система не розрахована для постановки діагнозу. Усі інші поля опціональні, але якщо лікар заповнив одне з них, то інформація буде урахована.

Фінальний промпт для створення медичної картки використовуючи чат GPT зображений на рисунку 2.18.

SYSTEM

You are a Medical expert. Doctors will ask you to describe the diagnosis and recipe for a patient. Also, the doctor can provide additional information such as the gender and birthday of a patient.

You will receive data in the next format of JSON object:

```
{
  gender:<male or female can be null>,
  birthDate:<date, can be null>,
  diagnosis:<diagnosis, must be provided>,
  description:<description of diagnosis, can be null>,
  recipe: <recipe for patient, can be null>
  comments_from_doctor: <additional comments, can be null>
}
```

If the received object `diagnosis` property is empty return the object with error: `{ error: "Not enough information"}`.

You need to create or complete the description and recipe, if they aren't empty you need to add information with minimal changes in existing data.

If property `comments` aren't empty, you need to consider the comments in adding/completing the description and recipe.

As a result, you need to return the object with values that must be formatted in markdown syntax. For example, you need to add `
` elements in the properties to break the lines for better readability:

```
{
  description:<completed description of diagnosis>,
  recipe: <completed recipe for patient>
}
```

Рисунок 2.18 Приклад промпту для медичної картки

Джерело: Сформовано автором

Приклад переданих і повернених даних і як результат готова медична картка:

```
USER
{ gender:"", birthDate:"", diagnosis:"Type 2 diabetes", description:"", recipe:"" comments_from_doctor: "" }
```

```
ASSISTANT
{ "description": "<br>Type 2 diabetes is a chronic condition that affects the way your body metabolizes sugar (glucose), which is an important source of fuel for your body. This type of diabetes is characterized by insulin resistance, meaning your body's cells don't respond properly to insulin. As a result, glucose builds up in the bloodstream instead of being absorbed by cells, leading to high blood sugar levels. <br><br>Symptoms of type 2 diabetes may include increased thirst, frequent urination, fatigue, blurred vision, slow-healing sores, and weight loss. Type 2 diabetes can be managed through lifestyle changes, including a healthy diet, regular exercise, and weight loss, as well as medication if necessary. <br><br>It is important to monitor blood sugar levels regularly, maintain a balanced diet rich in fruits, vegetables, and whole grains, limit the intake of sugary and processed foods, engage in regular physical activity, and follow the prescribed medication regimen.", "recipe": "<br>- Follow a balanced diet: Include plenty of fruits, vegetables, whole grains, lean proteins, and healthy fats in your meals. Limit your intake of sugary foods, processed snacks, and high-fat meals. <br>- Stay physically active" }
```

Рисунок 2.19 Приклад результату промпту для медичної картки

Джерело: Сформовано автором

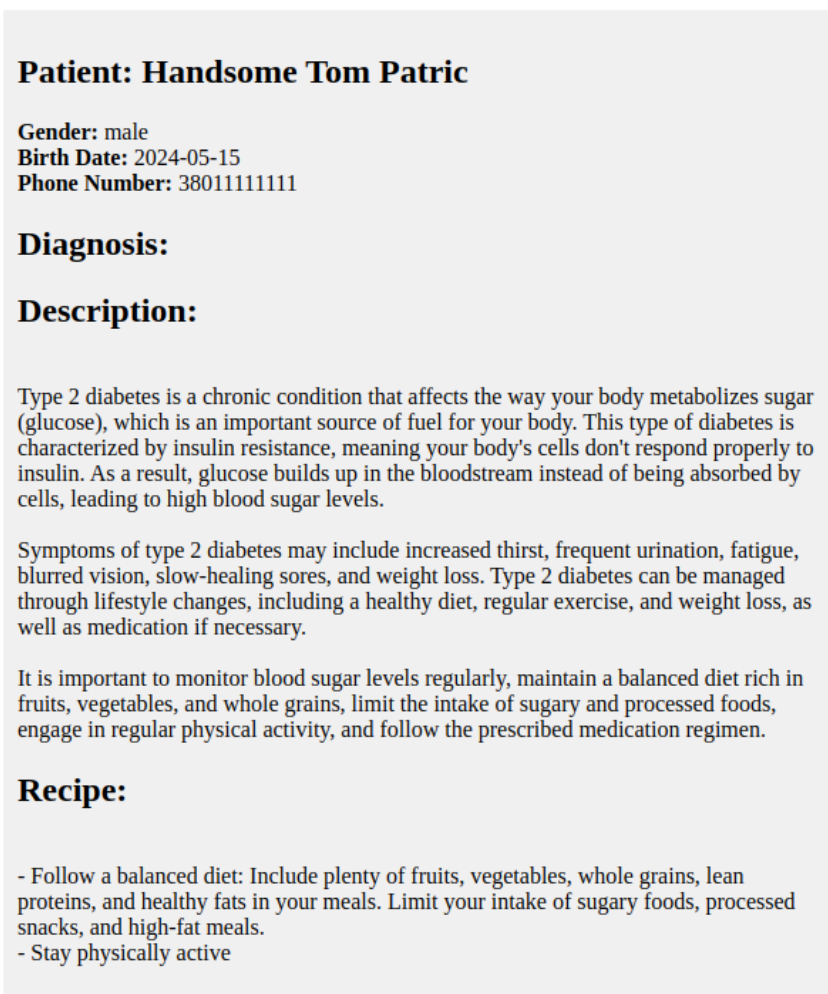


Рисунок 2.20 Зовнішній вигляд медичної картки

Джерело: Сформовано автором

Підсумовуючи, в цьому підрозділі ми розглянули архітектуру ChatGPT, основу написання промптів і шаблон промпту, котрий ми будемо використовувати для генерації медичної картки.

2.3.3 Методи та моделі управління в США

В попередньому розділі ми визначили, як буде формуватися медична картка пацієнта, в цьому розділі ми розглянемо підключення API ChatGPT до нашої веб-сторінки і згенеруємо новий медичний запис, використовуючи мовну модель.

Перше, що нам необхідно зробити – це визначити посилання для підключення до API. На сайті офіційної документації він описаний.

Далі необхідно отримати API ключ для використання функцій ChatGPT. На рисунку зображено створений API ключ в особистому кабінеті.

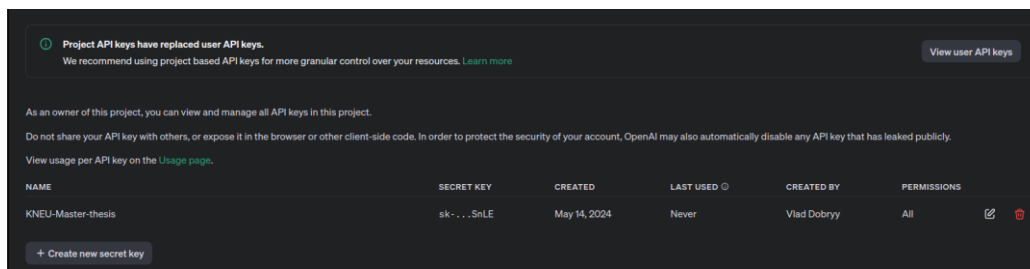


Рисунок 2.21 Приклад інтерфейсу API ChatGPT, отримання ключа

Джерело: Сформовано автором

Для зв'язку і відправки запитів до API мовної моделі ми будемо використовувати офіційний інструмент ChatGPT під назвою «openai».

Openai є офіційною бібліотекою клієнта, розробленою компанією OpenAI для спрощення інтеграції з їхніми AI-моделями, такими як GPT-3, GPT-4 та інші. З усім списком наявних моделей можна ознайомитися в офіційній документації. Ця бібліотека надає розробникам засоби для взаємодії з API OpenAI, дозволяючи легко використовувати різноманітні функції, від простих запитів до складних аналітичних операцій. Бібліотека забезпечує функції для автентифікації, відправлення запитів та обробки відповідей від сервера. Використання цього пакету дозволяє розробникам вбудовувати передові можливості штучного інтелекту у свої додатки, забезпечуючи високу ефективність і масштабованість проєктів, які взаємодіють з OpenAI [26].

На рисунку демонструється уривок з кодом взаємодію з API. На рисунку зображено дані, котрі ChatGPT повернув.

```

export async function sendOpenAiRequest(obj:MedicalCardRequest){
const userMessage = `{gender:"${obj.gender}",birthDate:"${obj.birthDay}",diagnosis:"${obj.diagnosis}",description:"${obj.description}`;
const openai = new OpenAI({apiKey:process.env.OPENAI_API_KEY,dangerouslyAllowBrowser: true});
const completion = await openai.chat.completions.create({
  model: "gpt-4",
  messages: [
    {
      role: "system",
      content: "You are a Medical expert. Doctors will ask you to describe the diagnosis and recipe for a patient."
    },
    {
      role: "user",
      content: userMessage
    }
  ],
  temperature:1,
  max_tokens:1000
});
//
console.log(completion);
console.log(completion.choices[0].message.content);
let object: MedicalCardResponse= JSON.parse(completion.choices[0].message.content as string) as MedicalCardResponse;
console.log(object);
console.log(object.description);
console.log(object.recipe);
return object;
}

```

Рисунок 2.22 Код відправки повідомлення до API ChatGPT

Джерело: Сформовано автором

```

openAiSendReq.ts:43
{description: 'Type 2 diabetes is a chronic condition that affect..exercise, a
re vital to managing this disease.<br>', recipe: 'Patients are advised to fol
low a diet low in sugar..rescribed to help control blood sugar levels.<br>'}
  i
  description: "Type 2 diabetes is a chronic condition that affects the way ti
  recipe: "Patients are advised to follow a diet low in sugar and carbohydrat
  ▶ [[Prototype]]: Object
Type 2 diabetes is a chronic condition that affects the openAiSendReq.ts:44
way the body processes glucose (sugar). With Type 2 diabetes, the body either
resists the effects of insulin or doesn't produce enough insulin to maintain
normal glucose levels. Regular monitoring of blood sugar levels, along with a
healthy diet and exercise, are vital to managing this disease.<br>
Patients are advised to follow a diet low in sugar and openAiSendReq.ts:45
carbohydrates. Regular physical activity is also a must. Medications such as
Metformin may be prescribed to help control blood sugar levels.<br>

```

Рисунок 2.23 Результат роботи ChatGPT

Джерело: Сформовано автором

Як зображено в уривку коду, ми форматуємо отриманий текст в JSON об'єкт. Ми це можемо робити, бо явно вказали формат результату для ChatGPT.

Підсумовуючи, ми встановили з'єднання з API ChatGPT і відтворили запит, котрим будемо користуватися надалі в проєкті і налаштували форматування отриманого повідомлення, щоб дані підходили для використання на веб-сторінці.

РОЗДІЛ 3 ПРОЄКТУВАННЯ ІС І ЇЇ КОМПОНЕНТІВ

3.1 Моделювання та проєктування бази знань для системи прийняття

В даній магістерській роботі не буде використовуватися база знань в класичному розумінні. Згідно джерелу [27], база знань – це сукупність систематизованих основних відомостей, що стосуються певної галузі знань і засобів, за допомогою яких відбуваються накопичення, збереження, оновлення та використання знань.

Відмінністю бази знань від бази даних є можливість формування нових знань. Головні функції бази знань стосовно систем автоматизованого проєктування (САП) [27]:

- опис предметної галузі САП;
- підтримка інтелектуальних методів розв'язання задач, що входять до складу САП;
- реалізація можливостей експертного аналізу проєктних завдань.

Наша система не буде мати вище описаний функціонал, бо ціль цього проєкту надати можливість лікарям і пацієнтам зберігати медичні дані на блокчейні і легко взаємодіяти з ним.

Як альтернатива базі знань, ми будемо використовувати ChatGPT. Остання версія ChatGPT (GPT 4) має датасет, котрий натренований на великій кількості даних з інтернету, котрий охоплює дані до січня 2022 року. Це надає нам змогу судити, що ChatGPT має в переліку дані, котрі відповідають медичній галузі, що дозволить йому допомагати лікарю з розповсюдженими сценаріями. Звичайно ChatGPT не підходить для консультування хвороб, котрі були виявлені або стали виліковними після січня 2022 року, але в цих випадках лікар повинен заповнити основну частину медичної картки.

Як частину бази знань можна зазначити базу даних серверної частини і розумні контракти на стороні блокчейну.

База даних буде слугувати для збереження даних користувачів для авторизації і автентифікації, адреси медичних карток на блокчейні і RSA ключі,

котрі будуть використовуватися для шифрування даних лікарем і розшифруванням пацієнтом. Також ці ключі будуть використовуватися для верифікації даних лікарем.

Звичайно, інтелектуальна медична картка створена для того щоб зберігати інформацію про історію здоров'ю пацієнта. Однак, через особливості реалізації, що були описані в минулому розділі, дані про стан пацієнта можна бути отримати через самого пацієнта, в свою чергу лікар буде мати можливість перевірити правдивість цих даних. Опис вихідних даних ви можете побачити в таблиці 3.1.

Таблиця 3.1 – Вихідні дані

Назва вихідного повідомлення	Ідентифікатор	Форма подання і вимоги до неї	Періодичність видання	Термін видання і допустимий час затримки	Користувач інформації
Загальні дані	General_data	JSON документ	При реєстрації пацієнта або лікаря	-	Пацієнт, Лікар
Адрес гаманця пацієнта	Patiend_address	Текстовий документ	За запитом лікаря або пацієнта	-	Лікар
Адрес гаманця лікаря	Doctor_address	Текстовий документ	За запитом лікаря	-	Пацієнт
Підтвердження Даних медичної картки	Verification_health_records	JSON документ	За запитом лікаря	-	Лікар
Дані медичної картки	Health_records	JSON документ	За запитом пацієнта	-	Пацієнт

Джерело: Сформовано автором

Для того, щоб медична картка повертала обрані дані, їх необхідно занести. Ви можете переглянути вхідні дані в таблиці 3.2.

Таблиця 3.2 – Вхідні дані

Назва вхідного повідомлення	Ідентифікатор	Форма подання	Термін і частота надходження	Джерело
Загальні дані	General_data	Json файл	При реєстрації пацієнта або лікаря	Пацієнт, Лікар
Адрес пацієнта	Patiend_address	Текстовий файл	При реєстрації	Пацієнт
Адрес лікаря	Doctor_address	Текстовий файл	При реєстрації	Лікар
Запис медичної картки	Health_records	Json файл	При необхідності	Лікар
Адреси медичної картки	Health_card_address	Текстовий файл	При публікації картки на блокчейн.	Система

Джерело: Сформовано автором

База даних буде мати наступну структуру:

1. Таблиця користувачів, вони буде описувати загальну інформацію про лікаря або пацієнта.
2. Таблиця інформації про лікаря, що буде пов'язана з таблицею користувачів, тільки лікарі будуть асоціюватися з цією таблицею.
3. Ролі – будуть існувати 3 ролі: адміністратор, пацієнт і лікар.
4. Таблиця ключів шифрування – тільки пацієнти будуть асоційовуватися з цією таблицею. Вона буде використовуватися для зберігання публічного і приватного ключа пацієнта. Якщо пацієнт буде проти зберігання власного приватного ключа, він може зберігати його в себе, а використовувати його тільки при перегляді даних своєї медичної картки.

На рисунку 3.1 ви можете побачити діаграму бази даних.

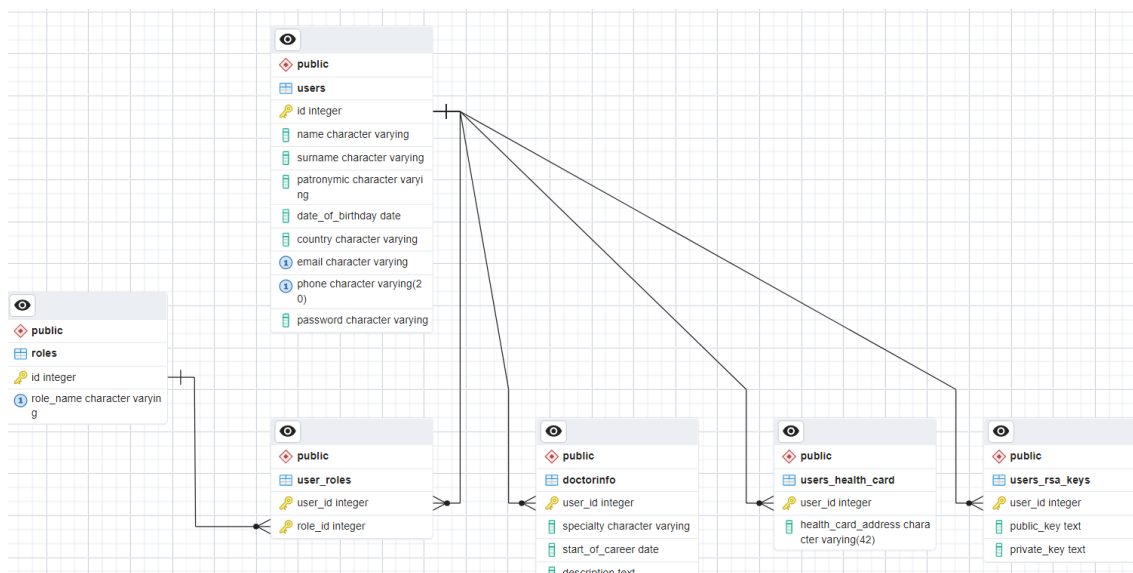


Рисунок 3.1 Діаграма бази даних

Джерело: Сформовано автором

Далі будуть детально описані кожна таблиця бази даних.

Таблиця користувачів, вона буде зберігати основну інформацію про користувачів, такі як ПІБ, електронна пошта, телефон, пароль і т.д.

Таблиця 3.3 – Таблиця користувачів

Назва атрибуту	Ім'я поля	Тип даних	Обмеження	Автоінкремент
Ідентифікатор користувача	id	INT	PRIMARY KEY	SERIAL
Ім'я користувача	name	VARCHAR	NOT NULL	-
Прізвище користувача	surname	VARCHAR	NOT NULL	-
По батькові користувача	patronymic	VARCHAR	-	-
Дата народження користувача	date_of_birthday	DATE	NOT NULL	-
Країна проживання користувача	country	VARCHAR	NOT NULL	-
Електронна пошта користувача	email	VARCHAR	NOT NULL	-

Телефон користувача	phone	VARCHAR(20)	NOT NULL	-
Пароль користувача	password	VARCHAR	NOT NULL	-
JWT токен для авторизації	jwt_token	VARCHAR	-	-

Джерело: Сформовано автором

Таблиця інформації про лікарів, вона поєднана з таблицею користувачів. Необхідна для відображення інформації про лікаря на сторінці його профілю, щоб пацієнт зміг ознайомитися з лікарем і надати йому доступ на редагування за його адресою блокчейн гаманця.

Таблиця 3.4 – Таблиця лікарів

Назва атрибуту	Ім'я поля	Тип даних	Обмеження	Автоінкремент
Ідентифікатор користувача	user_id	INT	PRIMARY KEY FOREIGN KEY	-
Спеціальність лікаря	specialty	VARCHAR	NOT NULL	-
Початок кар'єри лікаря	DATE	DATE	NOT NULL	-
Опис лікаря	description	TEXT		-
Блокчейн адреса лікаря	address_of_block chain_wallet	VARCHAR(42)	NOT NULL	-

Джерело: Сформовано автором

Таблиця 3.5 – Таблиця ролей

Назва атрибуту	Ім'я поля	Тип даних	Обмеження	Автоінкремент
Ідентифікатор ролі	id	INT	PRIMARY KEY	SERIAL
Назва ролі	role_name	VARCHAR	NOT NULL UNIQUE	-

Джерело: Сформовано автором

Поєднувальна таблиця між користувачем і роллю для встановлення відношення багато до багатьох.

Таблиця 3.6 – Таблиця для зв'язку багато до багатьох користувачів

Назва атрибуту	Ім'я поля	Тип даних	Обмеження	Автоінкремент
Ідентифікатор користувача	user_id	INT	NOT NULL, REFERENCES	-
Ідентифікатор ролі	role_id	INT	NOT NULL, REFERENCES	-

Джерело: Сформовано автором

Таблиця для зберігання RSA ключів пацієнта, пацієнт може не вносити свій приватний ключ, через міркування безпеки, але він запов'язаний внести публічний ключ, щоб ним міг скористатися лікар для шифрування медичних записів.

Таблиця 3.7 – Таблиця RSA ключів

Назва атрибуту	Ім'я поля	Тип даних	Обмеження	Автоінкремент
Ідентифікатор користувача	user_id	INT	PRIMARY KEY, REFERENCES	-
Публічний RSA ключ	public_key	TEXT	NOT NUL	-
Приватний RSA ключ	private_key	TEXT	-	-

Джерело: Сформовано автором

Таблиці для зберігання адреси медичної картки на блокчейні, оскільки в даній роботі ми будемо використовувати тільки одну блокчейн мережу, то нам не потрібно зберігати додаткову інформацію про унікальний ідентифікатор мережі, але в майбутньому можна додати подібний функціонал.

Таблиця 3.8 – Таблиця медичних карток

Назва атрибуту	Ім'я поля	Тип даних	Обмеження	Автоінкремент
Ідентифікатор користувача	user_id	INT	PRIMARY KEY	-
Адреса медичної картки	health_card_address	VARCHAR(42)	-	-

Джерело: Сформовано автором

Розумні контракти будуть також слугувати нам як джерела інформації.

Перший контракт HealthCardFactory буде надавати інформацію про медичні картки їх власників. Коли пацієнт буде створювати контракт медичної картки, контракт HealthCardFactory буде записувати адресу створеної медичної картки і асоціювати його з власником цієї картки. Одна адреса може мати тільки одну медичну картку. Таким чином ми гарантуємо що користувач не створить собі багато медичних карток. Асоціювання пацієнта з медичною картою буде виконувати шляхом використання конструкції mapping.

Мапінги в Solidity функціонують як хеш-таблиці, що зберігають пари ключ-значення, забезпечуючи ефективне зчитування та зберігання даних. Оголошення мапінгу включає вказання типів даних ключів та значень, використовуючи синтаксис `mapping(keyType => valueType)`, де `keyType` може бути будь-яким вбудованим або користувацьким типом, за винятком мапінгів, динамічно розмірених масивів і контрактів, тоді як `valueType` може бути будь-яким типом, включаючи мапінги. Значення за замовчуванням для будь-якого ключа в мапінгу дорівнює нулю або його еквіваленту, що означає, що зчитування незаданого ключа не викликає помилку часу виконання, а повертає це значення за замовчуванням.

Другий контракт HealthCard зберігає інформацію про медичні записи, також використовуючи конструкцію mapping. Коли лікар додає новий медичний запис, розумний контракт збільшує унікальний ідентифікатор і присвоює його цьому новому запису. Використовуючи унікальний ідентифікатор ми можемо визначити загальну кількість записів на медичні картці. Сам медичний запис представляє собою структуру, котра складається з наступних властивостей:

- doctor - адрес лікаря, котрий додавав цей запис;
- creationTimestamp - дата створення цього запису. Формат збереження дати - timestamp.
- record - сам медичний запис у зашифрованому вигляді;

Для того щоб лікар міг додавати нові записи, пацієнту необхідно дозволити обраному лікарю виконати цю операцію. В якості дозволу користувач встановлює час, до якого лікар має можливість вносити запис. Якщо лікар запізнився, то він не зможе виконати операцію. Пацієнт може надати доступ на додання тільки одного запису. Лікар не може за один раз внести багато записів. Таке обмеження було зроблено з цілі безпеки і читабельності коду. Після успішного запису дозвіл буде стерто і буде необхідність створення нового.

Третій контракт – це реєстр лікарів, для даного проєкту було обрана найпростіша реалізація подібного контракту, коли адміністратор, в нашому випадку DAO, додає і видаляє адреси лікарів. Найважливіша ціль його контракту це надання доступу до цього списку усім медичним карткам, що реалізується через функцію “isDoctor”.

На рисунку 3.2 зображено діаграма класів розумних контрактів.

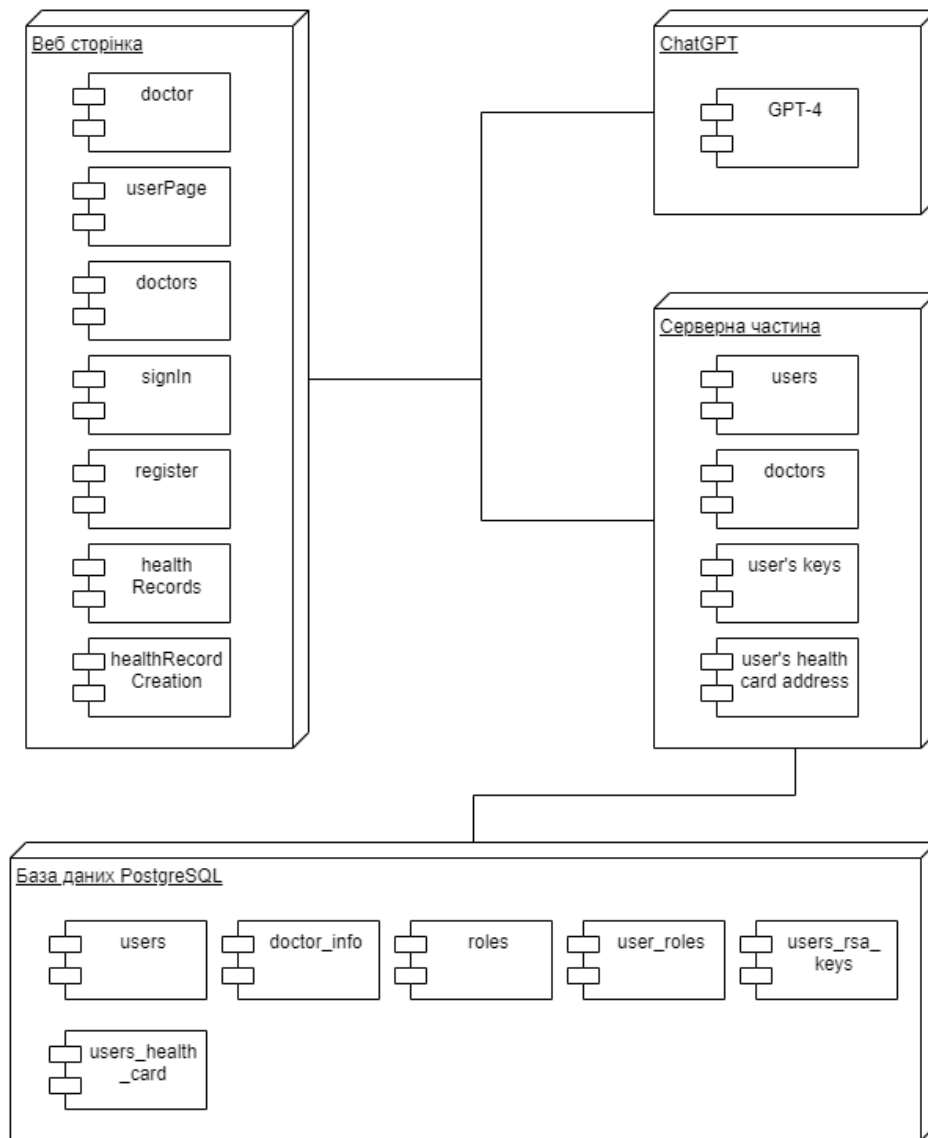


Рисунок 3.2 Діаграма класів розумних контрактів

Джерело: Сформовано автором

Підсумовуючи, ми розглянули наявні базу знань і інформаційні джерела, котрі будемо використовувати в реалізації проєкту. В якості основної бази знань в нас буде грати роль ChatGPT, а для полегшення роботи користувачу з системою в нас будуть використовуватися серверний додаток і розумні контракти.

3.2 Розроблення користувацького інтерфейсу. Елементи та структура.

Користувацький інтерфейс є невід’ємною частиною сучасних додатків, саме через нього користувачі будуть взаємодіяти з нашою системою. Для даного проєкту в пріоритет було поставлено функціональність і простоту інтерфейсу.

Структура інтерфейсу буде складатися з наступних елементів:

1. Головна сторінка
2. Сторінка реєстрації
3. Сторінка авторизації
4. Сторінка перегляду лікарів
5. Сторінка пацієнта
6. Сторінка лікаря
7. Створення медичного запису
8. Перегляд медичної картки

Перша сторінка, котра буде зустрічати користувача – це головна сторінка. Вона має декілька кнопок:

- вхід
- реєстрація
- перегляд лікарів
- створення медичного запису

Перераховані кнопки переводять користувача на відповідні сторінки. Усі вони доступні для відвідування, оскільки незареєстрований користувач все рівно не зможе створити медичний запис, бо перевірки на стороні блокчейну не дадуть цього зробити.

Main Page



Рисунок 3.3 Головна сторінка медичної картки

Джерело: Сформовано автором

Сторінка реєстрації поділена на 2 частини, це реєстрація пацієнта і лікаря. Спочатку користувачу необхідно внести наступні дані: ПІБ, електрону пошту, телефон, країну проживання, дата народження. Далі, якщо він реєструється як пацієнт, він обирає в елементі `radioButton` відповідну опцію і завершає реєстрацію.

Якщо користувач реєструється як лікар, йому необхідно заповнити додаткову інформацію як: адреса блокчейн гаманця, спеціалізацію, початок своєї кар'єри і опис про себе. Лікар описує себе використовуючи формат `markdown`, після трансформації відповідного тексту він буде перетворений в HTML код.

Name
Surname
Patronymic
Phone Number
Email
Country of Residence
mm/dd/yyyy
<input checked="" type="radio"/> Patient <input type="radio"/> Doctor
Sign Up

Рисунок 3.4 Сторінка реєстрації для пацієнта

Джерело: Сформовано автором

<input type="radio"/> Patient <input checked="" type="radio"/> Doctor
Blockchain Wallet
Cardiologist
mm/dd/yyyy
Description
<pre># Andrii Ublinskykh ## Solidity developer</pre>
Andrii Ublinskykh Solidity developer
Sign Up

Рисунок 3.5 Сторінка реєстрації для лікарів

Джерело: Сформовано автором

Після створення свого облікового запису користувач може авторизуватися використовуючи сторіку SignIn.

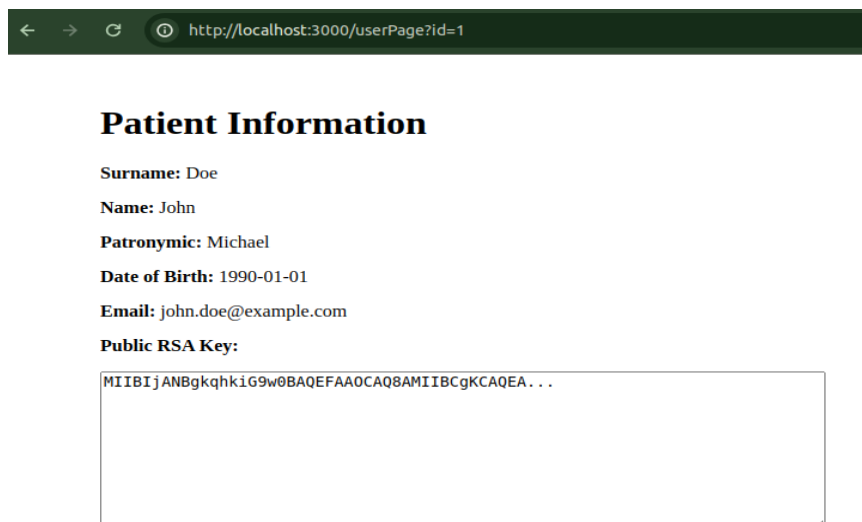


A simple login form consisting of three stacked rectangular boxes. The top box is labeled 'Username', the middle box is labeled 'Password', and the bottom box is a button labeled 'Sign In'.

Рисунок 3.6 Сторінка асторихації

Джерело: Сформовано автором

Після авторизації пацієнт може переглянути свій профіль або передивитися сторінки зареєстрованих лікарів.



A screenshot of a web browser showing a patient information page. The browser's address bar displays 'http://localhost:3000/userPage?id=1'. The page title is 'Patient Information'. The content includes the following fields:

- Surname:** Doe
- Name:** John
- Patronymic:** Michael
- Date of Birth:** 1990-01-01
- Email:** john.doe@example.com
- Public RSA Key:** MIIBIjANBgkqhkiG9w0BAQEFAAOCQA8AMIIBCgKCAQEA...

Рисунок 3.7 Сторінка пацієнта

Джерело: Сформовано автором

List of Doctors

<ul style="list-style-type: none"> Surname: Doe Name: John Email: john.doe@hospital.com
<ul style="list-style-type: none"> Surname: Smith Name: Jane Email: jane.smith@hospital.com
<ul style="list-style-type: none"> Surname: Brown Name: Charlie Email: charlie.brown@hospital.com
<ul style="list-style-type: none"> Surname: Davis Name: Emily Email: emily.davis@hospital.com
<ul style="list-style-type: none"> Surname: Wilson Name: Michael Email: michael.wilson@hospital.com

1 2

Рисунок 3.8 Сторінка пошуку лікаря

Джерело: Сформовано автором

Перейшовши на сторінку лікаря, пацієнт може одразу надати доступ цьому лікарю. Наразі час, котрий надається на внесення змін на веб-сторінку, дорівнює одну годину з виклику функції. В майбутньому ми можемо додати користувачу налаштувати це значення як йому бажано.

Doctor Information

Surname: Doe
Name: John
Patronymic: Michael
Birthday: 1/1/1980
Start of Career: 6/1/2005
Speciality: Cardiology
Wallet Address: 0x1234567890abcdef1234567890abcdef12345678 Add Allowance
Description:
<div style="border: 1px solid #ccc; padding: 10px;"> <p>Dr. John Doe</p> <p>John is a highly skilled cardiologist with over 15 years of experience...</p> </div>

Рисунок 3.9 Сторінка лікаря

Джерело: Сформовано автором

Коли лікар отримав необхідні права доступу, він може створити новий запис на медичній блокчейн картці пацієнта, перейшовши на сторінку healthRecordCreation. На ній користувач може побачити багато полів для заповнення, але найцікавіше на цій сторінці, це кнопки «Connect Wallet», «Generate Health Record» та «Store Health Record». Кнопка «Connect Wallet» дозволяє підключити блокчейн гаманець, такий як Metamask або його альтернативи. На рисунку зображено процес підключення. Кнопка «Generate Health Record» збирає передану інформацію про пацієнта (окрім конфіденційних даних) до ChatGPT і отримує доповнений опис і рецепт. Лікар може вказати додаткові коментарі до чату GPT у відповідному полі.

Кнопка “Store Health Record” зашифрує інформацію за допомогою публічного ключа клієнта, і зберігає ці дані на блокчейні звертаючись до медичної картки пацієнта і викликаючи функцію addHealthRecord.

Health Record Creation

Connect Wallet

Generate Health Record

Name: Tom

Surname: Handsome

Patronymic: Lukov

Gender: Male

Birth Date: 03/06/1991

Phone Number: 3801111111

Diagnosis: Type 2 diabetes

Description:

Type 2 diabetes is a chronic condition that affects the way your body metabolizes sugar (glucose) – an important source of fuel for your body.
With type 2 diabetes, your body either resists the effects of insulin – a hormone that regulates the movement of sugar into your cells – or it doesn't produce enough insulin to maintain normal glucose levels.

The management of Type 2 diabetes typically involves lifestyle modifications. This includes a healthy diet that is rich in whole grains, fruits, and vegetables.
Regular exercise and maintaining a healthy weight is essential. Medications might also be necessary, they range from Metformin, which is generally the first line of treatment, to others like Sulfonylureas, Thiazolidinediones, DPP-4 inhibitors and Insulin, depending on the severity and progression of the disease.
It is crucial to monitor blood sugar levels regularly, and adjustments made as necessary.

Рисунок 3.10 Сторінка створення медичного запису частина 1

Джерело: Сформовано автором

Recipe:

Comments for chat GPT:

Patient:

Gender: null
Birth Date: null
Phone Number: null

Diagnosis:

Type 2 diabetes

Description:

Type 2 diabetes is a chronic condition that affects the way your body metabolizes sugar (glucose) — an important source of fuel for your body. With type 2 diabetes, your body either resists the effects of insulin — a hormone that regulates the movement of sugar into your cells — or doesn't produce enough insulin to maintain normal glucose levels.
Type 2 diabetes used to be known as adult-onset or noninsulin-dependent diabetes. It can develop at any age and is now unfortunately seen in increasing numbers in young adults and children.

Recipe:

Management and treatment of type 2 diabetes usually involve:

- Healthy diet and regular physical activity: This can lower your blood sugar levels.
- Regular blood sugar monitoring: This will ensure that blood sugar levels are maintained at a target set by your healthcare professional.
- Diabetes medication or insulin therapy: These treatments should be prescribed and overseen by a medical professional.
- Regular check-ups: Regular checks are necessary to monitor for diabetes-related complications.

[Store Health Record](#)

Рисунок 3.11 Сторінка створення медичного запису, частина 2

Джерело: Сформовано автором

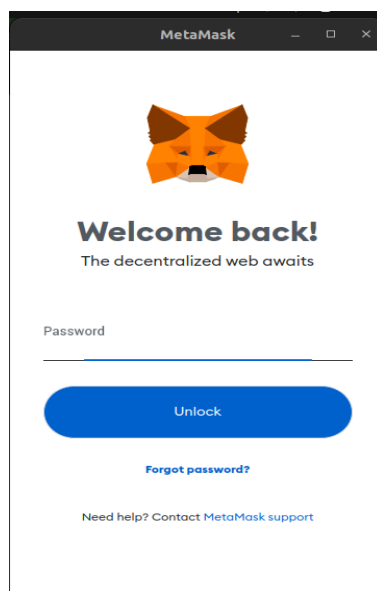


Рисунок 3.12 Вікно підключення до блокчейну мережі

Джерело: Сформовано автором

Перегляд інформації про медичну картку потребує приватний ключ. Передавши приватний ключ, користувач може прочитати дані своєї медичної картки. На рисунку зображено приклади розшифрованої медичної картки.

Medical Records

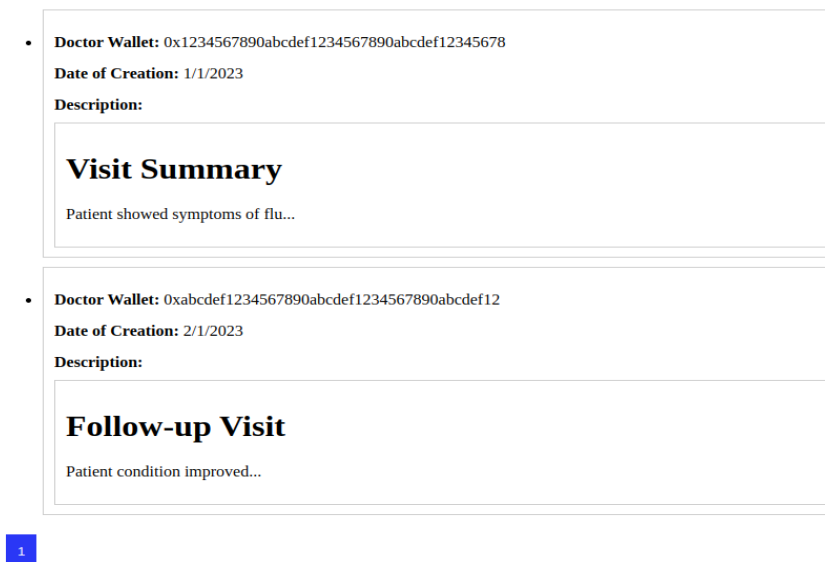


Рисунок 3.13 Перегляд розшифрованих медичних записів

Джерело: Сформовано автором

Підсумовуючи, ми переглянули головний функціонал інтелектуальної медичної картки. Переглянули як буде проводитися взаємодія між пацієнтом, лікарем і блокчейном.

3.3 Проєктування забезпечувальних підсистем СШ. Реалізація системи.

3.3.1 Інформаційне забезпечення

Інформаційне забезпечення — це сукупність ресурсів, засобів та методів, що забезпечують збирання, зберігання, обробку та аналіз інформації, необхідної для дослідження. Включає бібліографічні джерела, наукові статті, статистичні дані, програмне забезпечення для аналізу даних, а також методи пошуку і систематизації інформації. Ефективне інформаційне забезпечення дозволяє створити якісний теоретичний та практичний фундамент для дослідження, сприяючи досягненню поставлених цілей та вирішенню завдань дипломної роботи.

Одним з компонентів інформаційного забезпечення є база даних і розумні контракти що були описанні у підпункті 3.1.

Наступним компонентом інформаційного забезпечення є серверний додаток.

Перша таблиця, котру ми будемо використовувати це таблиця ролей.

Таблиця 3.9 – Дані в таблиці ролей

Ідентифікатор ролі	Назва ролі
1	Адміністратор
2	Пацієнт
3	Лікар

Джерело: Сформовано автором

Адміністратор є першим користувачем, котрий створюється шлях класичної реєстрації, а потім до нього буде додана роль Адміністратор через інструмент pgAdmin.

Друга таблиця, це таблиця користувачів, і нам потрібен перший користувач адміністратор.

Таблиця 3.10 – Дані адміністратора

Назва поля таблиці	Значення
Ідентифікатор користувача	1
Ім'я користувача	Andrii
Прізвище користувача	Ublinskykh
По батькові користувача	Anatoliyovich
Дата народження користувача	01.01.2000
Країна проживання користувача	Ukraine
Електронна пошта користувача	admin@admin.com
Телефон користувача	380111111111

Пароль користувача	22adaf058a2cb668b15cb4c1f30e7cc720b be38c146544169db35fbf630389c4
JWT токен для авторизації	-

Джерело: Сформовано автором

Усі інші значення будуть заноситися користувачами.

Також інформаційні значення зберігаються на стороні веб-сторінки, такі як:

1. Адреси розумних контрактів на тестовій мережі BSC testnet
2. Список спеціальностей для лікарів

Таблиця 3.11 – Адреси контрактів на мереж BSC testnet

Назва розумного контракту	Адреса контракту
HealthCardFactory	0x34745e0b8e18D011729B77a18F121c7faCb8B3eB
DoctorRegistry	0xf99Aa9Db07c0b1E08DD54E5122aA8225652D3eF9

Джерело: Сформовано автором

Таблиця 3.12 – Спеціальності лікарів

Спеціальності для лікарів
Кардіолог
Невролог
Лікар загальної практики

Джерело: Сформовано автором

Підсумовуючи, ми розглянули основні інформаційні джерела, котрі використовуються для роботи системи. До них відносяться: база даних серверної частини, сам серверний додаток, і постійні дані на веб-сторінці, такі як спеціальності лікарів і адреси розумних контрактів.

3.3.2 Програмне забезпечення

Програмне забезпечення - це інструменти для розробки, тестування та впровадження програмних рішень. Серверна частина охоплює серверні платформи, бази даних та інструменти для управління серверними ресурсами і безпекою. Фронт-енд частина включає фреймворки та бібліотеки для створення інтерфейсу користувача, інструменти для дизайну та прототипування. Блокчейн-частина охоплює платформи для розробки децентралізованих додатків, інструменти для написання розумних контрактів та засоби для тестування і розгортання блокчейн-рішень. Використання такого програмного забезпечення забезпечує комплексний підхід до розробки, тестування та впровадження сучасних ІТ-рішень у дипломній роботі.

При розробці використовувалися наступні програмне забезпечення:

1. Операційна система Ubuntu 24.04 для розробки і запуску додатку
2. Операційна система Windows 10 для редагування та оформлення документів
3. Visual studio Code в якості середовища розробки і текстового редактору
4. PostgreSQL в якості бази даних
5. PgAdmin для створення і керування базою даних
6. NodeJS версії 17.18.0 для створення і запуску серверного і веб-додатку
7. Бібліотека React для розробки веб-додатку
8. Фреймворк NextJS для розробка і запуску веб-додатку
9. Фреймворк NestJS для робробки серверної частини
- 10.Бібліотека TypeORM для взаємодії серверної частини і бази даних
- 11.Hardhat в якості середовища розробки і тестування розумних контрактів
- 12.Сервіс Remix для розгортання і верифікації розумних контрактів на блокчейн мережах.
- 13.Бібліотека openai для взаємодії веб-додатку з сервісами ChatGPT
- 14.ChatGPT мовна модель для генерації медичного запису
- 15.Бібліотека Ethers JS для підключення до блокчейну і взаємодії з розумними контрактами.

У додатках знаходиться файли `package.json`, котрі зберігають інформацію про бібліотеки і фреймворки, що використовуються.

Щоб скористатися ним, вам необхідно встановити NodeJS версії 17.18.0 або вище, після цього викликати команду “`npm install`”.

Вище зазначена команда завантажить усі необхідні бібліотеки і можна буде мати можливість запустити описані додатки. Але перед цим необхідно завантажити `postgreSQL` базу даних і інструмент для адміністрування `pgAdmin`. В додатку знайдеться код створення таблиць.

Для запуску серверного додатку можна скористатися командою “`npm run start`”, а для запуску веб-додатку - “`npm run dev`”.

3.3.3 Технічне забезпечення

В якості технічного забезпечення для роботи данної системи може підійти персональний комп'ютер або ноутбук з наступними характеристиками:

- 8 гігабайтів оперативної пам'яті
- 20-30 гігабайтів вільного місця на жорсткому диску
- стабільне підключення до інтернету, чим швидше тим краще
- процесор повинен мати як мінімум 4 потоки для комфортної роботи системи

Вище зазначені вимоги описують мінімальні необхідні характеристики для системи. При збільшенні навантаження необхідно буде проводити або вертикальне або горизонтальне масштабування.

Також зазначимо компоненти, з котрих буде складатися система. На рисунку зображено компоненти інтелектуальної медичної картки.

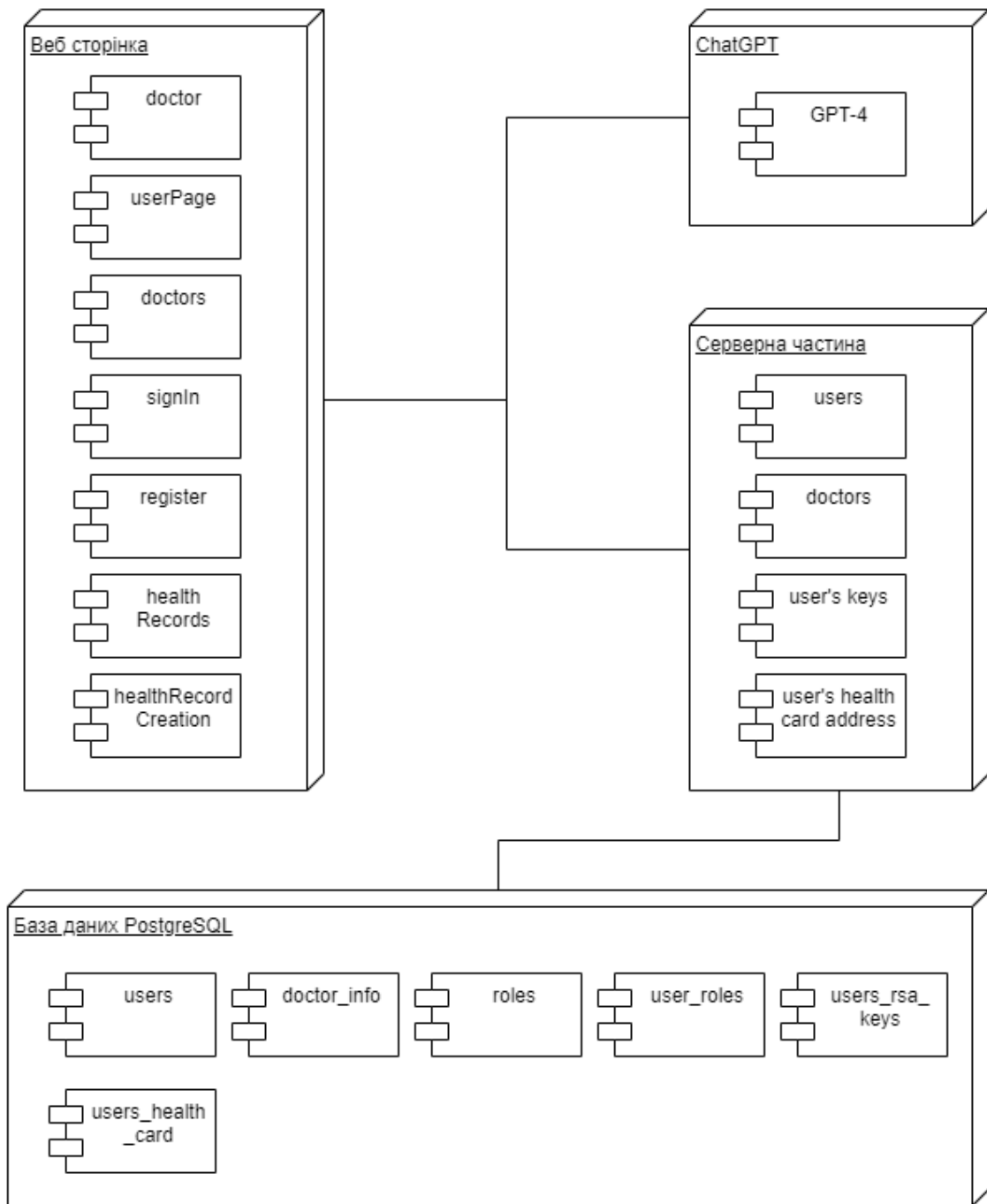


Рисунок 3.14 Діаграма розгортання

Джерело: Сформовано автором

Підсумовуючи, ми розглянули вимоги до технічного забезпечення і описали компоненти реалізації.

3.3.4 Організаційно-економічне забезпечення

Економічні переваги інтелектуальної медичної картки складно недооцінити, бо ми надаємо постійне місце для збереження даних за дуже малу вартість. Для

прикладу, щоб зберігати медичні запити в електронному виді медичному закладу необхідно створити і налаштувати інфраструктуру серверів для збереження даних і платити кожен місяць за їх утримання. Коли у випадку з блокчейн карткою, ми платимо тільки за запис, і нічого за утримання серверів або іншого обладнання.

Згідно джерелу [31], створення інфраструктури для медичного закладу середнього розміру коштує від \$14400 до \$27600. Таблиці 3.13 відображає типи необхідних робіт, котрі необхідні для реалізації додатку для медичного закладу згідно джерелу [31].

Таблиця 3.13 – Вартість розробки

Назва типу роботи	Вартість
Створення архітектури продукту	\$1600-2400
Розроблення користувацького інтерфейсу	\$2400-4800
Розробка серверної частини	\$9200-17600
Тестування	\$1200-2800

Джерело: Сформовано автором

Що стосується підтримки медичного додатку, на основі цін сервісу хмарних обчислень Azure [32], вартість буде складати приблизно \$310.5 на місяць. А таблиці 3.14 ви можете побачити типи витрат на підтримку і їх вартість.

Таблиця 3.14 – Вартість підтримки

Тип витрат	Погодина вартість	Сумарна вартість в місяць
Серверна частина	\$0.30	219
Веб-додаток	\$0.075	55
База даних	\$0.05	36,5

Джерело: Сформовано автором

Після проведення тестування розумного контракту, було обраховано вартість збереження медичного запису середнього розміру. Вартість була обрахована по формулі:

$$Tr = G * Gr$$

Де:

Tr – це вартість транзакції;

G – кількість gas;

Gp – вартість gas;

У контексті блокчейну, gas представляє одиницю виміру обчислювальних зусиль, необхідних для виконання транзакцій і розумних контрактів на платформах, таких як Ethereum. Він виконує функцію механізму комісій за транзакції, стимулюючи майнерів обробляти та підтверджувати операції шляхом компенсації їм цих комісій. Кількість газу, необхідного для транзакції, визначається складністю виконуваних операцій: більш складні дії вимагають вищих комісій за gas. Ця динамічна модель ціноутворення не лише забезпечує ефективне використання ресурсів мережі, але також відіграє важливу роль у підтриманні безпеки, стримуючи зловмисників від спаму мережі дорогими обчисленнями [28].

Водночас ціна gas визначається автоматично, на основі навантаження мережі блокчейну. Ціна gas вимірюється у Gwei, що дорівнює одному мільярду Wei. Wei це одиниця вимірування криптовалюти. Наприклад більшість популярних валют як Ethereum (ETH), одна одиниця ETH вираховується як $1 \cdot 10^{18}$.

Вартість запису складає 1500000 газу на мережі BSC testnet.

Ціна за gas в середньому складає 1 Gwei.

Після множення ми отримали ціну виконання транзакції, що дорівнює 1500000 Gwei, після переведення до одиниці криптовалюти ми отримали значення 0.0015 BNB.

BNB – це нативна валюта мережі BSC. На час написання роботи, ціна BNB до гривні складає 22673грн. Після переведення по цьому курсу, ми отримали ціну транзакції у гривнях, що дорівнює 34 гривні. Маю зазначити, що ціна за gas на різних мережах різна і відповідно ціна за транзакцію буде різна.

Якщо лікарня впровадить використання розробленої системи, вона може зменшити тарифний план бази даних, з \$0.05 до \$0.01, що буде зберігати \$30 в місяць.

Наразі, з точки зору економічної ефективності, система має певні проблеми, бо в рамках цієї роботи не було розроблено методики заробітку на інтелектуальні картці. Але це можна легко вирішити 2 шляхами:

- Додати змогу лікарям оформити підписку на користування функціями ChatGPT.
- Встановити комісію в розмірі 0.1-1% при збереженні медичної інформації в медичну картку.

Це буде гарним покращенням системи в майбутньому і зробить її як мінімум економічно не збитковою.

В розділі 3 ми детально описали деталі реалізації системи, її структуру, базу даних, інші інформаційні джерела, інтерфейс користувача, програмне і технічне забезпечення, а також їх вимоги. Це надає нам повне розуміння роботи інтелектуальної картки. Основна функція, що виконує наш додаток, це надання функціонального інтерфейса для взаємодії з мовною моделю ChatGPT і блокчейном. Наразі є багато місць, котрі можна покращити в подальшій розробці, такі як економічна вигода проєкту, збільшення безпеки і покращення інтерфейсу на більш зручний і привабливий для користувачів.

ВИСНОВКИ

В даній магістерській роботі було проведено аналіз предметної області і існуючих рішень. Було виявлено те, що багато систем сфокусовані на впровадженні блокчейну в якості платіжної системи, а не як інструменту для пацієнтів і лікарів, тому було обрано варіант вирішити цю проблему, створивши блокчейн медичну картку пацієнта і користувацький інтерфейс, котрий буде надавати зручний доступ до взаємодії з медичною блокчейн картою.

Було розглянуто технічні і програмні інструменти для реалізації обраного додатку, у підсумку було обрано створити додаток з трьохрівневою архітектурою клієнт-сервер з використанням технологій:

- Node.js – середовище виконання коду на мові JavaScript;
- React – бібліотека для розробки веб-додатків;
- NextJS – фреймворк для розробки веб-додатків;
- NestJS - фреймворк для розробки серверних додатків;
- Solidity – мова програмування розумних контрактів для EVM блокчейн мереж.

Було побудовано архітектуру додатку, визначені взаємодії між компонентами додатку і алгоритм роботи програми. Додаток надає лікарю і пацієнту ділитися особистими даними, такими як публічний RSA ключ, адреса блокчейн картки, адреса блокчейн гаманця. Ці дані необхідні для безпечного збереження і створення медичного запису в блокчейн мережі.

Було розроблено веб-сторінку, користувацький інтерфейс, серверну частину і розгорнуто розумні контракти на мереж BSC testnet, з котрими можна ознайомитися.

Наразі додаток виконує свої основні функції коректно, але він має декілька недоліків, котрі з легкістю можна буде вирішити в подальшій розробці, серед яких:

- функціональний, але не привабливий для користувачів інтерфейс;
- економічна збитковість для власника інтелектуальної медичної картки, бо не було розроблено модель монетизації системи;

Однак, величезним плюсом цієї системи є те, що інтеграція подібної системи до медичної клініки може зекономити величезний обсяг грошей, через майже відсутність витрат на її підтримку.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Healthcare industry [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Healthcare_industry.
2. What Is a Blockchain? [Електронний ресурс] – Режим доступу до ресурсу: <https://www.investopedia.com/terms/b/blockchain.asp>.
3. 2023 Year Review & 2024 Year Ahead [Електронний ресурс] – Режим доступу до ресурсу: <https://crypto.com/research/2023-review-2024-ahead>.
4. Blockchain in Healthcare: 17 Examples to Know [Електронний ресурс] – Режим доступу до ресурсу: <https://builtin.com/blockchain/blockchain-healthcare-applications-companies>.
5. Medicalchain Whitepaper 2.1 [Електронний ресурс] – Режим доступу до ресурсу: <https://medicalchain.com/Medicalchain-Whitepaper-EN.pdf>.
6. Tokenizing the Global Economy [Електронний ресурс] – Режим доступу до ресурсу: <https://guardtime.com/>.
7. The digital healthcare ecosystem is here [Електронний ресурс] – Режим доступу до ресурсу: <https://avaneerhealth.com/newsletters/the-digital-healthcare-ecosystem-is-here/>.
8. Top 5 blockchain healthcare projects. Part 2 [Електронний ресурс] – Режим доступу до ресурсу: <https://digiforest.io/en/blog/blockchain-projects-in-healthcare-part2>.
9. PostgreSQL [Електронний ресурс] – Режим доступу до ресурсу: <https://en.wikipedia.org/wiki/PostgreSQL>.
10. OpenAI — Understand Foundational Concepts of ChatGPT and cool stuff you can explore! [Електронний ресурс] – Режим доступу до ресурсу: <https://medium.com/@amol-wagh/open-ai-understand-foundational-concepts-of-chatgpt-and-cool-stuff-you-can-explore-a7a77baf0ee3>.
11. Інтерфейс Користувача (UI): Значення, Складові та Вплив на Продажі Інтернет-Магазину [Електронний ресурс] – Режим доступу до ресурсу: <https://wezom.com.ua/ua/blog/chto-takoe-ui-i-kak-polzovatelskij-interfejs-vliyaet-na-prodazhi-internet-magazina>.

12. Metamask - What is it, and How Does it Work? [Электронный ресурс] – Режим доступа до ресурсу: <https://originstamp.com/blog/metamask-what-is-it-and-how-does-it-work/>.
13. Visual Studio Code [Электронный ресурс] – Режим доступа до ресурсу: https://en.wikipedia.org/wiki/Visual_Studio_Code.
14. Decentralized Autonomous Organization (DAO): Definition, Purpose, and Example [Электронный ресурс] – Режим доступа до ресурсу: [https://www.investopedia.com/tech/what-dao/#:~:text=A%20decentralized%20autonomous%20organization%20\(DAO\)%20is%20an%20emerging%20form%20of,best%20interest%20of%20the%20entity](https://www.investopedia.com/tech/what-dao/#:~:text=A%20decentralized%20autonomous%20organization%20(DAO)%20is%20an%20emerging%20form%20of,best%20interest%20of%20the%20entity).
15. Remote Procedure Call [Электронный ресурс] – Режим доступа до ресурсу: <https://www.ibm.com/docs/en/aix/7.3?topic=concepts-remote-procedure-call>.
16. Roles Of RPC in Blockchain Development: A Developer's Handbook [Электронный ресурс] – Режим доступа до ресурсу: <https://medium.com/@martinsmicheal198/roles-of-rpc-in-blockchain-development-a-developers-handbook-ab43d02d2b1a#:~:text=RPC%20in%20DApp%20Development&text=Method%20Invocation%3A%20Blockchain%20developers%20use,specific%20functions%20within%20smart%20contracts>.
17. Health Insurance Portability and Accountability Act of 1996 (HIPAA) [Электронный ресурс] – Режим доступа до ресурсу: <https://www.cdc.gov/phlp/publications/topic/hipaa.html#:~:text=The%20Health%20Insurance%20Portability%20and,the%20patient\'s%20consent%20or%20knowledge..>
18. Decentralized Autonomous Organization (DAO): Definition, Purpose, and Example [Электронный ресурс] – Режим доступа до ресурсу: [https://www.investopedia.com/tech/what-dao/#:~:text=A%20decentralized%20autonomous%20organization%20\(DAO\)%20is%20an%20emerging%20form%20of,best%20interest%20of%20the%20entity](https://www.investopedia.com/tech/what-dao/#:~:text=A%20decentralized%20autonomous%20organization%20(DAO)%20is%20an%20emerging%20form%20of,best%20interest%20of%20the%20entity).

[ois%20an%20emerging%20form%20of,best%20interest%20of%20the%20entity.](#)

19. What is prompt engineering? [Електронний ресурс] – Режим доступу до ресурсу: <https://www.ibm.com/topics/prompt-engineering>.
20. N Tier Architecture [Електронний ресурс] – Режим доступу до ресурсу: <https://jenkov.com/tutorials/software-architecture/n-tier-architecture.html>.
21. Getting Started [Електронний ресурс] – Режим доступу до ресурсу: <https://code.visualstudio.com/docs>.
22. Why, where and when to use Coding Conventions [Електронний ресурс] – Режим доступу до ресурсу: <https://svitla.com/blog/why-where-and-when-to-use-coding-conventions>.
23. What is RSA? How does an RSA work? [Електронний ресурс] – Режим доступу до ресурсу: <https://www.encryptionconsulting.com/education-center/what-is-rsa/>.
24. A Definitive Guide to Learn The SHA-256 (Secure Hash Algorithms) [Електронний ресурс] – Режим доступу до ресурсу: <https://www.simplilearn.com/tutorials/cyber-security-tutorial/sha-256-algorithm>.
25. Introduction to Snapshot [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.snapshot.org/introduction>.
26. OpenAI Node API Library [Електронний ресурс] – Режим доступу до ресурсу: <https://www.npmjs.com/package/openai>.
27. БАЗА ЗНАНЬ ІНТЕЛЕКТУАЛЬНОЇ ІНФОРМАЦІЙНОЇ СИСТЕМИ ПРОГНОЗУВАННЯ ФАЗОВОЇ СТАБІЛЬНОСТІ ТВЕРДИХ РОЗЧИНІВ [Електронний ресурс] – Режим доступу до ресурсу: <https://itce.vntu.edu.ua/index.php/itce/article/view/917>.
28. Gas (Ethereum): How Gas Fees Work on the Ethereum Blockchain [Електронний ресурс] – Режим доступу до ресурсу: <https://www.investopedia.com/terms/g/gas-ethereum.asp>.
29. Privacy and Security of Blockchain in Healthcare: Applications, Challenges, and Future Perspectives [Електронний ресурс] – Режим доступу до ресурсу:

<https://www.researchgate.net/publication/375102582> Privacy and Security of Blockchain in Healthcare Applications Challenges and Future Perspectives.

30. Improving Healthcare Applications Security Using Blockchain [Электронный ресурс] – Режим доступа до ресурсу: <https://www.researchgate.net/publication/365497690> Improving Healthcare Applications Security Using Blockchain.
31. How Much Does It Cost to Develop a Hospital Management System [Электронный ресурс] – Режим доступа до ресурсу: <https://devico.io/blog/how-much-does-it-cost-to-develop-a-hospital-management-system>.
32. Azure App Service on Windows pricing [Электронный ресурс] – Режим доступа до ресурсу: <https://azure.microsoft.com/en-us/pricing/details/app-service/windows/>.

ДОДАТКИ

Додаток А

Файл конфігурації підключення до PostgreSQL бази даних:

```
// src/config/config.service.ts

import { TypeOrmModuleOptions } from '@nestjs/typeorm';

// eslint-disable-next-line @typescript-eslint/no-var-requires
require('dotenv').config();

import { User } from '../users/entities/user.entity';
import { Role } from '../roles/entities/role.entity';
import { UserRole } from '../user-roles/entities/user-role.entity';

class ConfigService {

  constructor(private env: { [k: string]: string | undefined }) {}

  private getValue(key: string, throwOnMissing = true): string {

    const value = this.env[key];

    if (!value && throwOnMissing) {

      throw new Error(`config error - missing env.${key}`);

    }

    return value;

  }

  public ensureValues(keys: string[]) {

    keys.forEach((k) => this.getValue(k, true));

    return this;

  }

}
```

```
public getPort() {  
    return this.getValue('PORT', true);  
}  
  
public isProduction() {  
    const mode = this.getValue('MODE', false);  
    return mode !== 'DEV';  
}  
  
public getTypeOrmConfig(): TypeOrmModuleOptions {  
    return {  
        type: 'postgres',  
        host: this.getValue('POSTGRES_HOST'),  
        port: parseInt(this.getValue('POSTGRES_PORT')),  
        username: this.getValue('POSTGRES_USER'),  
        password: this.getValue('POSTGRES_PASSWORD'),  
        database: this.getValue('POSTGRES_DATABASE'),  
        entities: [User, Role, UserRole],  
        synchronize: false,  
        autoLoadEntities: true,  
    };  
}  
  
const configService = new ConfigService(process.env).ensureValues([  
    'POSTGRES_HOST',  
    'POSTGRES_PORT',
```

```

    'POSTGRES_USER',
    'POSTGRES_PASSWORD',
    'POSTGRES_DATABASE',
  ]);

export { configService };

```

Файл контролера, котрий повертає дані користувача:

```

import {
  Body,
  Controller,
  Delete,
  Get,
  Param,
  Post,
  Put,
} from '@nestjs/common';

import { UsersService } from '../users.service';
import { UserRolesService } from '../user-roles/user-roles.service';
import { UserRole } from '../user-roles/entities/user-role.entity';
import { Roles } from '../auth/roles.decorator';
import { CreateUserDto } from '../dto/create-user.dto';

@Controller('users')
export class UsersController {
  constructor(

```

```
    private readonly userService: UsersService,
    private readonly userRolesService: UserRolesService,
  ) {}

  @Post()
  async create(@Body() createUserDto: CreateUserDto) {
    const user = await this.userService.create(createUserDto);
    // eslint-disable-next-line prefer-const
    let userRole = new UserRole();
    userRole.user_id = user.id;
    userRole.role_id = 2;
    await this.userRolesService.create(userRole);
    return user;
  }

  @Roles('ADMIN', 'USER')
  @Get()
  async findAll() {
    return this.userService.findAll();
  }

  @Get('/:id')
  async findOne(@Param('id') id: string) {
    return this.userService.findOne(+id);
  }

  @Get('/:email')
  async findOneByEmail(@Param('email') email: string) {
```

```

        return this.userService.findUserByEmail(email);
    }

    @Get('/:name')
    async findOneByName(@Param('name') name: string) {
        return this.userService.findUserByUsername(name);
    }

    @Put('/:id')
    async update(@Param('id') id: string, @Body() updateUserDto:
CreateUserDto) {
        return this.userService.update(+id, updateUserDto);
    }

    @Delete('/:id')
    async remove(@Param('id') id: string) {
        return this.userService.remove(+id);
    }
}

```

Файл авторизації :

```

import { Body, Controller, Post, UnauthorizedException } from
'@nestjs/common';

import { SignInDto } from './dto/sign-in.dto';
import { AuthService } from './auth.service';
import { Public } from './public.decorator';

@Controller('auth')

```

```

export class AuthController {
  constructor(private readonly authService: AuthService) {}

  @Public()
  @Post()
  async signIn(@Body() signIn: SignInDto) {
    console.log(signIn);

    if (signIn.email === undefined || signIn.password === undefined)
    {
      throw new UnauthorizedException();
    }

    return this.authService.signIn(signIn.email, signIn.password);
  }
}

```

Файл автентифікації:

```

import {
  CanActivate,
  ExecutionContext,
  Injectable,
  UnauthorizedException,
} from '@nestjs/common';
import { Reflector } from '@nestjs/core';
import { getJwtToken } from 'src/utils/utils';
import { JwtService } from '@nestjs/jwt';
import { UsersService } from 'src/users/users.service';
import { User } from 'src/users/entities/user.entity';
import { IS_PUBLIC_KEY } from './public.decorator';

```

```
import { getJwtSecret } from './constants';

@Injectable()
export class AuthGuard implements CanActivate {

  constructor(

    private jwtService: JwtService,

    private reflector: Reflector,

    private userService: UsersService,

  ) {}

  async canActivate(context: ExecutionContext): Promise<boolean> {

    const isPublic =
this.reflector.getAllAndOverride<boolean>(IS_PUBLIC_KEY, [

      context.getHandler(),

      context.getClass(),

    ]);

    if (isPublic) {

      return true;

    }

    const request = context.switchToHttp().getRequest();

    const jwtToken = getJwtToken(request);

    if (jwtToken === '') {

      console.log("token equals to '');

      throw new UnauthorizedException();

    }

    try {
```

```
    await this.jwtService.verifyAsync(accessToken, {
      secret: getJwtSecret(),
    });
  } catch {
    console.log('wrong signature');
    throw new UnauthorizedException();
  }

  const decodedJwtToken: any = this.jwtService.decode(accessToken, {
    json: true,
  });

  const user: User = await this.userService.findOneWithJWTToken(
    decodedJwtToken.sub,
  );

  console.log('user', user);

  if (!user) {
    console.log('user equals to undefined');
    throw new UnauthorizedException();
  }

  if (user.jwt_token === accessToken) return true;
  else {
    console.log("jwt tokens doesn't equal to jwt token in db");
    return false;
  }
};
}
```

Додаток Б

Далі наведений код сторінок веб-додатку.

Код головної сторінки:

```
// pages/index.js

import { useRouter } from 'next/router';

export default function Home() {

  const router = useRouter();

  const handleNavigation = (path) => {

    router.push(path);

  };

  return (

    <div style={{ display: 'flex', flexDirection: 'column',
alignItems: 'center', justifyContent: 'center', height: '100vh' }}>

      <h1>Main Page</h1>

      <button onClick={() => handleNavigation('/signin')} style={{
margin: '10px', padding: '10px 20px' }}>

        Sign In

      </button>

      <button onClick={() => handleNavigation('/register')} style={{
margin: '10px', padding: '10px 20px' }}>

        Sign Up

      </button>

    </div>

  );
}
```

```

        <button onClick={() =>
handleNavigation('/healthRecordCreation')} style={{ margin: '10px',
padding: '10px 20px' }}>

            Health Record Creation

        </button>

        <button onClick={() => handleNavigation('/doctorsList')}
style={{ margin: '10px', padding: '10px 20px' }}>

            Doctors

        </button>

    </div>

);
}

```

Код сторінки реєстрації:

```

import { useState, useEffect, ChangeEvent } from 'react';
import ReactMarkdown from 'react-markdown';
import MarkdownEditor from '../components/markdownEditor';

interface Specialty {
    id: number;
    name: string;
}

const RegisterPage = () => {
    const [formData, setFormData] = useState({
        name: '',
        surname: '',
        patronymic: '',

```

```

    phoneNumber: '',
    email: '',
    country: '',
    birthday: '',
    role: '',
    blockchainWallet: '',
    specialty: '',
    careerStartDate: '',
    description: ''
  });

  const [specialties, setSpecialties] = useState<Specialty[]>([]);
  const [showDoctorFields, setShowDoctorFields] = useState(false);

  // Fetch specialties when component mounts
  useEffect(() => {
    // Simulate fetching data
    setSpecialties([
      { id: 1, name: 'Cardiologist' },
      { id: 2, name: 'Neurologist' },
      { id: 3, name: 'General Practitioner' }
    ]);
  }, []);

  const handleChange = (e: ChangeEvent<HTMLInputElement |
HTMLTextAreaElement | HTMLSelectElement>) => {
    const { name, value } = e.target;
    setFormData(prev => ({ ...prev, [name]: value }));
  };

```

```

    if (name === 'role') {
        setShowDoctorFields(value === 'doctor');
    }
};

const renderMarkdownToHtml = () => {
    console.log(formData.description);
    return marked.parse(formData.description);
};

const handleSubmit = (event: React.FormEvent<HTMLFormElement>) => {
    event.preventDefault();
    console.log('Form Data Submitted:', formData);
};

return (
    <div style={{ padding: '20px' }}>
        <form onSubmit={handleSubmit} style={{ display: 'flex',
flexDirection: 'column', gap: '15px' }}>
            <input name="name" value={formData.name}
onChange={handleChange} placeholder="Name" />
            <input name="surname" value={formData.surname}
onChange={handleChange} placeholder="Surname" />
            <input name="patronymic" value={formData.patronymic}
onChange={handleChange} placeholder="Patronymic" />
            <input name="phoneNumber" value={formData.phoneNumber}
onChange={handleChange} placeholder="Phone Number" type="tel" />
            <input name="email" value={formData.email}
onChange={handleChange} placeholder="Email" type="email" />

```

```

        <input name="country" value={formData.country}
onChange={handleChange} placeholder="Country of Residence" />

        <input name="birthday" value={formData.birthday}
onChange={handleChange} placeholder="Birthday" type="date" />

        <div style={{ display: 'flex', flexDirection: 'row',
justifyContent: 'flex-start' }}>

            <input type="radio" id="patient" name="role"
value="patient" onChange={handleChange} />

            <label htmlFor="patient" style={{ marginRight: '20px'
}}>Patient</label>

            <input type="radio" id="doctor" name="role" value="doctor"
onChange={handleChange} />

            <label htmlFor="doctor">Doctor</label>

        </div>

        {showDoctorFields && (
            <>
                <input name="blockchainWallet"
value={formData.blockchainWallet} onChange={handleChange}
placeholder="Blockchain Wallet" />

                <select name="specialty" value={formData.specialty}
onChange={handleChange}>

                    {specialties.map(specialty => (
                        <option key={specialty.id}
value={specialty.name}>{specialty.name}</option>
                    ))}

                </select>

                <input name="careerStartDate"
value={formData.careerStartDate} onChange={handleChange} placeholder="Date
of Start of Doctor Career" type="date" />

```

```

        <div>
            <label htmlFor="description">Description</label>
            <MarkdownEditor/>
        </div>
    </>
    )}

    <button type="submit">Sign Up</button>
</form>
</div>
);
};

export default RegisterPage;

```

Код сторінки входу до системи:

```

import { useState, FormEvent } from 'react';
import { signIn } from '../services/authService';

const SignInPage = () => {
    const [username, setUsername] = useState('');
    const [password, setPassword] = useState('');
    const [loading, setLoading] = useState(false);

    const handleSubmit = async (event: FormEvent) => {
        event.preventDefault();

```

```

    setLoading(true);

    console.log("Submitted Username:", username, "Password:",
password);

    const authenticated = await signIn({ username, password });

    setLoading(false);

    if (authenticated) {

        console.log("Success: User is authenticated");

        // Redirect or perform some action after successful
authentication

    } else {

        //alert make a big notification, I don't like it

        console.log("Authentication failed");

    }

};

return (

    <div style={{ display: 'flex', justifyContent: 'center',
alignItems: 'center', height: '100vh' }}>

        <form onSubmit={handleSubmit} style={{ display: 'flex',
flexDirection: 'column', width: '300px' }}>

            <input

                type="text"

                placeholder="Username"

                value={username}

                onChange={(e) => setUsername(e.target.value)}

                style={{ marginBottom: '10px', padding: '10px', fontSize:
'16px' }}

                />

```

```

    <input
      type="password"
      placeholder="Password"
      value={password}
      onChange={(e) => setPassword(e.target.value)}
      style={{ marginBottom: '10px', padding: '10px', fontSize:
'16px' }}
    />

    <button type="submit" style={{ padding: '10px 0', fontSize:
'16px' }}>Sign In</button>

  </form>

</div>

);

};

export default SignInPage;

```

Код сторінки пошуку лікаря:

```

// pages/doctors/index.js

import { useRouter } from 'next/router';
import { useState } from 'react';

// Mock data for demonstration

const doctorsData = [

  { id: 1, surname: 'Doe', name: 'John', email:
'john.doe@hospital.com' },

```

```
    { id: 2, surname: 'Smith', name: 'Jane', email:
'jane.smith@hospital.com' },

    { id: 3, surname: 'Brown', name: 'Charlie', email:
'charlie.brown@hospital.com' },

    { id: 4, surname: 'Davis', name: 'Emily', email:
'emily.davis@hospital.com' },

    { id: 5, surname: 'Wilson', name: 'Michael', email:
'michael.wilson@hospital.com' },

    { id: 6, surname: 'Taylor', name: 'Sophia', email:
'sophia.taylor@hospital.com' },

    { id: 7, surname: 'Anderson', name: 'James', email:
'james.anderson@hospital.com' },

    // Add more mock doctors as needed

];

const ITEMS_PER_PAGE = 5;

export default function DoctorList() {

  const [currentPage, setCurrentPage] = useState(1);

  const router = useRouter();

  const handleDoctorClick = (id) => {

    router.push(`/doctor?id=${id}`);

  };

  const handlePageChange = (page) => {

    setCurrentPage(page);

  };

};
```

```

const indexOfLastDoctor = currentPage * ITEMS_PER_PAGE;

const indexOfFirstDoctor = indexOfLastDoctor - ITEMS_PER_PAGE;

const currentDoctors = doctorsData.slice(indexOfFirstDoctor,
indexOfLastDoctor);

const totalPages = Math.ceil(doctorsData.length / ITEMS_PER_PAGE);

return (
  <div style={{ padding: '20px' }}>
    <h1>List of Doctors</h1>
    <ul>
      {currentDoctors.map((doctor) => (
        <li
          key={doctor.id}
          onClick={() => handleDoctorClick(doctor.id)}
          style={{
            cursor: 'pointer',
            border: '1px solid #ccc',
            padding: '10px',
            margin: '10px 0',
          }}
        >
          <p><strong>Surname:</strong> {doctor.surname}</p>
          <p><strong>Name:</strong> {doctor.name}</p>
          <p><strong>Email:</strong> {doctor.email}</p>
        </li>
      ))}
    </ul>
  </div>
)

```

```

    </ul>

    <div>

        {Array.from({ length: totalPages }, (_, index) => index +
1).map((page) => (

            <button

                key={page}

                onClick={() => handlePageChange(page)}

                style={{

                    margin: '5px',

                    padding: '10px',

                    backgroundColor: page === currentPage ? '#0070f3' :
'#fff',

                    color: page === currentPage ? '#fff' : '#0070f3',

                    border: '1px solid #0070f3',

                }}

            >

                {page}

            </button>

        )}}

    </div>

</div>

);

}

```

Код сторінки лікаря:

```

// pages/doctor/[id].js

import { useRouter } from 'next/router';

```

```
import { useEffect, useState } from 'react';

import ReactMarkdown from 'react-markdown';

// Mock data for demonstration

const doctorsData = {

  1: {

    surname: 'Doe',

    name: 'John',

    patronymic: 'Michael',

    birthday: '1980-01-01',

    startOfCareer: '2005-06-01',

    speciality: 'Cardiology',

    walletAddress: '0x1234567890abcdef1234567890abcdef12345678',

    description: '# Dr. John Doe\n\nJohn is a highly skilled cardiologist with over 15 years of experience...',

  },

  2: {

    surname: 'Smith',

    name: 'Jane',

    patronymic: 'Anna',

    birthday: '1985-05-15',

    startOfCareer: '2010-09-10',

    speciality: 'Neurology',

    walletAddress: '0xabcdef1234567890abcdef1234567890abcdef12',

    description: '# Dr. Jane Smith\n\nJane is a renowned neurologist known for her extensive research...',

  },

}
```

```

    // Add more mock doctors as needed
};

export default function Doctor() {

  const router = useRouter();

  const { id } = router.query;

  const [doctor, setDoctor] = useState(null);

  useEffect(() => {

    if (id) {

      setDoctor(doctorsData[id]);

    }

  }, [id]);

  if (!doctor) {

    return <div>Loading...</div>;

  }

  return (

    <div style={{ padding: '20px', maxWidth: '600px', margin: '0 auto' }}>

      <h1>Doctor Information</h1>

      <p><strong>Surname:</strong> {doctor.surname}</p>

      <p><strong>Name:</strong> {doctor.name}</p>

      <p><strong>Patronymic:</strong> {doctor.patronymic}</p>

      <p><strong>Birthday:</strong> {new
Date(doctor.birthday).toLocaleDateString()}</p>

```

```

    <p><strong>Start of Career:</strong> {new
Date(doctor.startOfCareer).toLocaleDateString()}</p>

    <p><strong>Speciality:</strong> {doctor.speciality}</p>

    <p>

        <strong>Wallet Address:</strong> {doctor.walletAddress}

        <button

            style={{

                marginLeft: '10px',

                padding: '5px 10px',

                backgroundColor: 'green',

                color: 'white',

                border: 'none',

                cursor: 'pointer',

            }}

            onClick={() => alert('Allowance added')}

        >

            Add Allowance

        </button>

    </p>

    <div>

        <strong>Description:</strong>

        <div style={{ border: '1px solid #ccc', padding: '10px',
marginTop: '10px' }}>

            <ReactMarkdown>{doctor.description}</ReactMarkdown>

        </div>

    </div>

</div>

```

```
);
}
```

Код сторінки пацієнта:

```
// pages/patient/[id].js

import { useRouter } from 'next/router';
import { useEffect, useState } from 'react';

// Mock data for demonstration
const patientsData = {
  1: {
    surname: 'Doe',
    name: 'John',
    patronymic: 'Michael',
    dateOfBirth: '1990-01-01',
    email: 'john.doe@example.com',
    publicKey: 'MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA...',
  },
  2: {
    surname: 'Smith',
    name: 'Jane',
    patronymic: 'Anna',
    dateOfBirth: '1985-05-15',
    email: 'jane.smith@example.com',
    publicKey: 'MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA...',
  },
},
```

```

3: {
  surname: 'Brown',
  name: 'Charlie',
  patronymic: 'David',
  dateOfBirth: '1975-12-10',
  email: 'charlie.brown@example.com',
  publicKey: 'MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA...',
},
// Add more mock patients as needed
};

export default function Patient() {
  const router = useRouter();
  const { id } = router.query;
  const [patient, setPatient] = useState(null);

  useEffect(() => {
    if (id) {
      setPatient(patientsData[id]);
    }
  }, [id]);

  if (!patient) {
    return <div>Loading...</div>;
  }

  return (

```

```

<div style={{ padding: '20px', maxWidth: '600px', margin: '0 auto' }}>
  <h1>Patient Information</h1>
  <p><strong>Surname:</strong> {patient.surname}</p>
  <p><strong>Name:</strong> {patient.name}</p>
  <p><strong>Patronymic:</strong> {patient.patronymic}</p>
  <p><strong>Date of Birth:</strong> {patient.dateOfBirth}</p>
  <p><strong>Email:</strong> {patient.email}</p>
  <p><strong>Public RSA Key:</strong></p>
  <textarea
    readOnly
    value={patient.publicKey}
    style={{ width: '100%', height: '150px' }}
  />
</div>
);
}

```

Код сторінки створення медичного запису:

```

// pages/HealthRecordCreation.tsx
import React, { useState, ChangeEvent, useEffect } from 'react';
import ReactMarkdown from 'react-markdown';
import remarkGfm from 'remark-gfm';
import rehypeRaw from 'rehype-raw';
import ConnectWallet from '../components/connectWallet';
import connectWallet from '@services/utills/connectWallet';

```

```

import {
  generateKeyPair, encryptWithPublicKey, decryptWithPrivateKey, hybridEncrypt, hybridDecrypt } from '@services/Utils/rsaFunctions';

import { sendOpenAiRequest, MedicalCardResponse } from '@services/openAiSendReq';

const HealthRecordCreation = () => {
  const [formData, setFormData] = useState({
    name: '',
    surname: '',
    patronymic: '',
    gender: '',
    birthDate: '',
    phoneNumber: '',
    diagnosis: '',
    description: '',
    recipe: '',
    comments: ''
  });

  const [markdownPreview, setMarkdownPreview] = useState('');

  const handleInputChange = (event: ChangeEvent<HTMLInputElement | HTMLSelectElement | HTMLTextAreaElement>) => {
    const { name, value } = event.target;
    setFormData(prevFormData => ({ ...prevFormData, [name]: value }));
  };
};

```

```

const generateWithGPT = async () => {
  if(formData.diagnosis == ""){
    alert("You must make a diagnosis!!! ChatGPT can't do it for
you");
    return;
  }
  const reqObj = {
    gender:formData.gender,
    birthDay:formData.birthDate,
    diagnosis:formData.diagnosis,
    description:formData.description,
    recipe:formData.recipe,
    commentsFromDoctor:formData.comments
  }
  let object:MedicalCardResponse = await sendOpenAiRequest(reqObj);
  setFormData(prevFormData => ({ ...prevFormData, description:
object.description }));
  setFormData(prevFormData => ({ ...prevFormData, recipe:
object.recipe }));
}

useEffect(() => {
  const markdown = `
  ## Patient:  ${formData.surname} ${formData.name}
  ${formData.patronymic}<br>
  **Gender:**  ${formData.gender=="?"?"null":formData.gender} <br>
  **Birth Date:**
  ${formData.birthDate=="?"?"null":formData.birthDate}<br>

```

```

    **Phone Number:**
    ${formData.phoneNumber=="?"?"null":formData.phoneNumber}<br>

    ## Diagnosis:

    ${formData.diagnosis}

    ## Description:

    ${formData.description}

    ## Recipe:

    ${formData.recipe}

    `;

    setMarkdownPreview(markdown);

  }, [formData]);

const storeHealthRecord = async () => {

  console.log("Storing health record:", formData);

  const signer = await connectWallet();

  console.log("signer", signer);

  let pair = generateKeyPair();

  console.log("pair.publicKey", pair.publicKey);

  console.log("pair.privateKey", pair.privateKey);

  console.log("markdownPreview", markdownPreview);

  const encryptedRecord =
  hybridEncrypt(markdownPreview, pair.publicKey);

  console.log("encryptedRecord", encryptedRecord.encryptedData);

  const decryptRecord =
  hybridDecrypt(encryptedRecord.encryptedData, encryptedRecord.encryptedKey,
  pair.privateKey);

  console.log("decryptRecord", decryptRecord);

```

```

};

return (
  <div >
    <div style={{paddingLeft: '10px'}}>
      <h1>Health Record Creation</h1>
    </div>
    <div style={{padding: '10px'}}>
      <ConnectWallet/>
    </div>
    <button onClick={generateWithGPT} style={{ backgroundColor:
'green', color: 'white', padding: '10px', margin: '10px' }}>
      Generate Health Record
    </button>
    <form /* onChange={generateHealthRecordMarkdown} */ style={{
display: 'flex', flexDirection: 'column'}}>
      <div style={{padding: '10px'}}>
        <label>Name:</label>
        <input type="text" name="name" value={formData.name}
onChange={handleInputChange} style={{marginLeft:'20px'}}/>
      </div>
      <div style={{padding: '10px'}}>
        <label>Surname:</label>
        <input type="text" name="surname" value={formData.surname}
onChange={handleInputChange} style={{marginLeft:'20px'}}/>
      </div>
      <div style={{padding: '10px'}}>

```

```

        <label>Patronymic:</label>

        <input type="text" name="patronymic"
value={formData.patronymic} onChange={handleInputChange}
style={{marginLeft:'20px'}}/>

    </div>

    <div style={{padding: '10px'}}>

        <label>Gender:</label>

        <select name="gender" value={formData.gender}
onChange={handleInputChange} style={{marginLeft:'20px'}}>

            <option value="">Select Gender</option>

            <option value="male">Male</option>

            <option value="female">Female</option>

        </select>

    </div>

    <div style={{padding: '10px'}}>

        <label>Birth Date:</label>

        <input type="date" name="birthDate"
value={formData.birthDate} onChange={handleInputChange}
style={{marginLeft:'20px'}} />

    </div>

    <div style={{padding: '10px'}}>

        <label>Phone Number:</label>

        <input type="tel" name="phoneNumber"
value={formData.phoneNumber} onChange={handleInputChange}
style={{marginLeft:'20px'}}/>

    </div>

    <div style={{padding: '10px'}}>

        <label>Diagnosis:</label>

```

```

        <input type="text" name="diagnosis"
value={formData.diagnosis} onChange={handleInputChange}
style={{marginLeft:'20px'}}/>

    </div>

    <div style={{padding: '10px'}}>

        <label>Description:</label>

        <textarea name="description" value={formData.description}
onChange={handleInputChange} style={{height: '200px', marginLeft:'20px'}}/>

    </div>

    <div style={{padding: '10px'}}>

        <label>Recipe:</label>

        <textarea name="recipe" value={formData.recipe}
onChange={handleInputChange} style={{ height: '200px', marginLeft:'20px'}}
/>

    </div>

    <div style={{padding: '10px'}}>

        <label>Comments for chat GPT:</label>

        <textarea name="comments" value={formData.comments}
onChange={handleInputChange} style={{ height: '200px', marginLeft:'20px'}}
/>

    </div>

</form>

    <div style={{ backgroundColor: '#f0f0f0', padding: '10px',
margin: '10px' }}>

        <ReactMarkdown children={markdownPreview}
remarkPlugins={[remarkGfm]} rehypePlugins={[rehypeRaw]}/>

    </div>

```

```

        <button onClick={storeHealthRecord} style={{ marginTop: '20px',
backgroundColor: 'blue', color: 'white', padding: '10px 20px', margin:
'10px' }}>

            Store Health Record

        </button>

    </div>

);

};

export default HealthRecordCreation;

```

Код сторінки медичного запису:

```

import { useRouter } from 'next/router';
import { useEffect, useState } from 'react';
import ReactMarkdown from 'react-markdown';

// Mock data for demonstration
const medicalRecordsData = {
  1: [
    {
      doctorWallet: '0x1234567890abcdef1234567890abcdef12345678',
      dateOfCreation: '2023-01-01',
      description: '# Visit Summary\n\nPatient showed symptoms of
flu...',
    },
    {
      doctorWallet: '0xabcdef1234567890abcdef1234567890abcdef12',

```

```

        dateOfCreation: '2023-02-01',
        description: '# Follow-up Visit\n\nPatient condition
improved... ',
    },
    // Add more records as needed
],
2: [
    {
        doctorWallet: '0xabcdefabcdefabcdefabcdefabcdef',
        dateOfCreation: '2023-03-01',
        description: '# Initial Consultation\n\nPatient diagnosed
with... ',
    },
    // Add more records as needed
],
// Add more patients and records as needed
};

```

```
const ITEMS_PER_PAGE = 5;
```

```

export default function MedicalRecords() {
    const router = useRouter();
    const { id } = router.query;
    const [records, setRecords] = useState([]);
    const [currentPage, setCurrentPage] = useState(1);

    useEffect(() => {

```

```

    if (id) {
      setRecords(medicalRecordsData[id] || []);
    }
  }, [id]);

const handlePageChange = (page) => {
  setCurrentPage(page);
};

const indexOfLastRecord = currentPage * ITEMS_PER_PAGE;
const indexOfFirstRecord = indexOfLastRecord - ITEMS_PER_PAGE;
const currentRecords = records.slice(indexOfFirstRecord,
indexOfLastRecord);

const totalPages = Math.ceil(records.length / ITEMS_PER_PAGE);

if (!records.length) {
  return <div>Loading...</div>;
}

return (
  <div style={{ padding: '20px', maxWidth: '800px', margin: '0
auto' }}>
    <h1>Medical Records</h1>
    <ul>
      {currentRecords.map((record, index) => (
        <li key={index} style={{ border: '1px solid #ccc', padding:
'10px', margin: '10px 0' }}>

```

```

        <p><strong>Doctor Wallet:</strong>
{record.doctorWallet}</p>

        <p><strong>Date of Creation:</strong> {new
Date(record.dateOfCreation).toLocaleDateString()}</p>

        <div>

            <strong>Description:</strong>

            <div style={{ border: '1px solid #ccc', padding:
'10px', marginTop: '10px' }}>

                <ReactMarkdown>{record.description}</ReactMarkdown>

            </div>

        </div>

    </li>

    )})
</ul>

<div>

    {Array.from({ length: totalPages }, (_, index) => index +
1).map((page) => (

        <button

            key={page}

            onClick={() => handlePageChange(page)}

            style={{

                margin: '5px',

                padding: '10px',

                backgroundColor: page === currentPage ? '#0070f3' :
'#fff',

                color: page === currentPage ? '#fff' : '#0070f3',

                border: '1px solid #0070f3',

            }}

```

```

        >
        {page}
    </button>
    )})
</div>
</div>
);
}

```

Додаток В

Далі наведений код розумних контрактів:

```

// SPDX-License-Identifier: MIT
pragma solidity =0.8.23;

interface IDoctorRegistry {
    function isDoctor(address _doctor) external returns (bool);
    function addDoctor(address _doctor) external;
    function removeDoctor(address _doctor) external;
}

contract DoctorRegistry is IDoctorRegistry {
    error NotDAO();
    mapping(address => bool) public doctors;
    address public dao;
    modifier onlyDao() {
        if (msg.sender != dao) {

```

```

        revert NotDAO();
    }
    _;
}
function addDoctor(address _doctor) external onlyDao {
    doctors[_doctor] = true;
}
function removeDoctor(address _doctor) external onlyDao {
    delete doctors[_doctor];
}
function isDoctor(address _doctor) external view returns (bool) {
    return doctors[_doctor];
}
}

contract HealthCard {
    error NotOwner();
    error AccessExpired();
    error NotDoctor();
    struct HealthRecord {
        address doctor;
        uint256 creationTimestamp;
        string record;
    }
    event RecordCreated(address indexed doctor);
    address public immutable owner;
    address public immutable doctorRegistry;
    uint256 public healthRecordAmount;
    mapping(uint256 => HealthRecord) public healthRecords;
    mapping(address => uint256) public doctorAllowance;
    string public publicKey;

    constructor(address _owner, address _doctorRegistry) {
        owner = _owner;
        doctorRegistry = _doctorRegistry;
    }
}

```

```

}

modifier onlyOwner() {
    if (msg.sender != owner) {
        revert NotOwner();
    }
    _;
}

modifier onlyDoctor() {
    uint256 deadline = doctorAllowance[msg.sender];
    if (deadline == 0 || deadline < block.timestamp) {
        revert AccessExpired();
    }
    _;
}

function addAllowance(
    address _doctor,
    uint256 _deadline
) external onlyOwner {
    if (!IDoctorRegistry(doctorRegistry).isDoctor(_doctor)) {
        revert NotDoctor();
    }
    doctorAllowance[_doctor] = _deadline;
}

function setPublicKey(string calldata _publicKey) external onlyOwner {
    publicKey = _publicKey;
}

function addRecord(string calldata _record) external {
    // onlyDoctor(){
    HealthRecord memory hr = HealthRecord({
        doctor: msg.sender,
        creationTimestamp: block.timestamp,

```

```

        record: _record
    });
    healthRecords[healthRecordAmount] = hr;
    healthRecordAmount++;
    delete doctorAllowance[msg.sender];
    emit RecordCreated(msg.sender);
}

function getHealthRecords(
    uint256 offset,
    uint256 limit
) external view returns (HealthRecord[] memory) {
    uint256 totalAmount = healthRecordAmount;
    if (offset > totalAmount || limit == 0) {
        return new HealthRecord[](0);
    }
    uint256 amount = totalAmount - offset;
    if (amount < limit) {
        limit = amount;
    }
    HealthRecord[] memory res = new HealthRecord[](limit);
    for (uint256 i; i < limit; i++) {
        res[i] = healthRecords[i + offset];
    }
    return res;
}
}

contract HealthCardFactory {
    error NotDAO();
    event HealthCardCreated(address indexed owner, address indexed healthCard);
    // owner => healthCard
    mapping(address => address) public ownerToHealthCard;
    address public dao;
    address public doctorRegistry;
}

```

```

modifier onlyDao() {
    if (msg.sender != dao) {
        revert NotDAO();
    }
    _;
}

function setNewDAO(address _newDAO) external onlyDao {
    dao = _newDAO;
}

function setNewDoctorRegistry(address _newRegistry) external onlyDao {
    doctorRegistry = _newRegistry;
}

function createHealthCard() external returns (address) {
    HealthCard healthCard = new HealthCard(msg.sender, doctorRegistry);
    emit HealthCardCreated(msg.sender, address(healthCard));
    return address(healthCard);
}
}

```

Додаток Г

Файл package.json для веб-додатку:

```

{
  "name": "medical-front-end",
  "version": "0.1.0",
  "private": true,
  "scripts": {
    "dev": "next dev",
    "build": "next build",
    "start": "next start",
    "lint": "next lint"
  },
  "dependencies": {
    "@metamask/providers": "^16.1.0",

```

```

    "dotenv": "^16.4.5",
    "ethers": "^6.12.1",
    "next": "14.2.3",
    "node-forge": "^1.3.1",
    "openai": "^4.47.1",
    "react": "^18",
    "react-dom": "^18"
  },
  "devDependencies": {
    "@types/node": "^20",
    "@types/react": "^18",
    "@types/react-dom": "^18",
    "eslint": "^8",
    "eslint-config-next": "14.2.3",
    "postcss": "^8",
    "react-markdown": "^9.0.1",
    "rehype-raw": "^7.0.0",
    "remark": "^15.0.1",
    "remark-gfm": "^4.0.0",
    "remark-html": "^16.0.1",
    "tailwindcss": "^3.4.1",
    "typescript": "^5"
  }
}

```

Файл `package.json` для серверного dodatku:

```

{
  "name": "online-music-editor-back-end",
  "version": "0.0.1",
  "description": "",
  "author": "",
  "private": true,
  "license": "UNLICENSED",
  "scripts": {

```

```

"build": "nest build",
"format": "prettier --write \"src/**/*.ts\" \"test/**/*.ts\"",
"start": "nest start",
"start:dev": "nest start --watch",
"start:debug": "nest start --debug --watch",
"start:prod": "node dist/main",
"lint": "eslint \"{src,apps,libs,test}/**/*.ts\" --fix",
"test": "jest",
"test:watch": "jest --watch",
"test:cov": "jest --coverage",
"test:debug": "node --inspect-brk -r tsconfig-paths/register -r ts-
node/register node_modules/.bin/jest --runInBand",
"test:e2e": "jest --config ./test/jest-e2e.json"
},
"dependencies": {
"@nestjs/class-validator": "^0.13.4",
"@nestjs/common": "^10.0.0",
"@nestjs/core": "^10.0.0",
"@nestjs/jwt": "^10.1.0",
"@nestjs/platform-express": "^10.0.0",
"@nestjs/platform-socket.io": "^10.2.7",
"@nestjs/swagger": "^7.1.6",
"@nestjs/typeorm": "^10.0.0",
"@nestjs/websockets": "^10.2.7",
"@types/socket.io": "^3.0.2",
"bcrypt": "^5.1.0",
"dotenv": "^16.3.1",
"mysql2": "^3.5.1",
"pg": "^8.11.1",
"rxjs": "^7.8.1",
"sha.js": "^2.4.11",
"typeorm": "^0.3.17"
},
"devDependencies": {
"@nestjs/cli": "^10.0.0",
"@nestjs/schematics": "^10.0.0",

```

```

"@nestjs/testing": "^10.0.0",
"@types/express": "^4.17.17",
"@types/jest": "^29.5.2",
"@types/node": "^20.3.1",
"@types/supertest": "^2.0.12",
"@typescript-eslint/eslint-plugin": "^5.59.11",
"@typescript-eslint/parser": "^5.59.11",
"eslint": "^8.42.0",
"eslint-config-prettier": "^8.8.0",
"eslint-plugin-prettier": "^4.2.1",
"jest": "^29.5.0",
"prettier": "^2.8.8",
"source-map-support": "^0.5.21",
"supertest": "^6.3.3",
"ts-jest": "^29.1.0",
"ts-loader": "^9.4.3",
"ts-node": "^10.9.1",
"tsconfig-paths": "^4.2.0",
"typescript": "^5.1.3"
},
"jest": {
  "moduleFileExtensions": [
    "js",
    "json",
    "ts"
  ],
  "rootDir": "src",
  "testRegex": ".*\\.spec\\.ts$",
  "transform": {
    "^.+\\.?(t|j)s?$": "ts-jest"
  },
  "collectCoverageFrom": [
    "**/*.?(t|j)s"
  ],
  "coverageDirectory": "../coverage",

```

```

    "testEnvironment": "node"
  }
}

```

Додаток Д

Файл створення таблиць в базі даних:

```

-- Creating 'users' table
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    name VARCHAR NOT NULL,
    surname VARCHAR NOT NULL,
    patronymic VARCHAR,
    date_of_birthday DATE NOT NULL,
    country VARCHAR NOT NULL,
    email VARCHAR NOT NULL UNIQUE,
    phone VARCHAR(20) NOT NULL UNIQUE,
    password VARCHAR NOT NULL,
    jwt_token VARCHAR
);

-- Creating 'doctorInfo' table
CREATE TABLE doctorInfo (
    user_id INT PRIMARY KEY,
    specialty VARCHAR NOT NULL,
    start_of_career DATE NOT NULL,
    description TEXT,
    address_of_blockchain_wallet VARCHAR(42) NOT NULL,
    FOREIGN KEY (user_id) REFERENCES users(id)
);

-- Creating 'roles' table
CREATE TABLE roles (
    id SERIAL PRIMARY KEY,
    role_name VARCHAR NOT NULL UNIQUE

```

```
);

-- Creating 'user_roles' table
CREATE TABLE user_roles (
    user_id INT NOT NULL,
    role_id INT NOT NULL,
    PRIMARY KEY (user_id, role_id),
    FOREIGN KEY (user_id) REFERENCES users(id),
    FOREIGN KEY (role_id) REFERENCES roles(id)
);

-- Creating 'users_rsa_keys' table
CREATE TABLE users_rsa_keys (
    user_id INT PRIMARY KEY,
    public_key TEXT NOT NULL,
    private_key TEXT,
    FOREIGN KEY (user_id) REFERENCES users(id)
);

-- Creating 'users_health_card' table
CREATE TABLE users_health_card (
    user_id INT PRIMARY KEY,
    health_card_address VARCHAR(42),
    FOREIGN KEY (user_id) REFERENCES users(id)
);
```

Додаток Е

Цифровізація науки та сучасні тренди її розвитку

Ублінських Андрій Анатолійович, здобувач другого (магістерського) рівня вищої освіти інституту інформаційних технологій в економіці *Державний вищий навчальний заклад «Київський економічний університет імені Вадима Гетьмана»*, Україна

Науковий керівник: Ситник Ніна Василівна, кандидат економічних наук, професор кафедри інформаційних систем в економіці *Державний вищий навчальний заклад «Київський економічний університет імені Вадима Гетьмана»*, Україна

МАЙБУТНЄ БЛОКЧЕЙН ТЕХНОЛОГІЇ

Блокчейн технологія, відома більшості завдяки криптовалютам, насправді є революційним винаходом, що відкриває нові перспективи для безлічі індустрій. Її основна цінність полягає в забезпеченні цілісності та прозорості даних, які є критично важливими в епоху діджиталізації. Наразі блокчейн вже змінює підходи в банківській сфері, логістиці, правовій індустрії та багатьох інших секторах, але це лише початок широкомасштабної трансформації.

Блокчейн є інноваційною записною книгою, де кожен запис (блок) є зв'язаним з попереднім і наступним, створюючи ланцюг (чейн), який неможливо змінити без відома всіх учасників системи. Це створює стійкість до шахрайства та забезпечує високий рівень безпеки.

Оскільки технологія є відкритою та децентралізованою, вона дозволяє забезпечити рівень довіри між сторонами без потреби в посередниках, таких як банки або юридичні особи. Така можливість має потенціал радикально змінити економічні відносини в глобальному масштабі.

Одним з визначень блокчейну є наступне. Блокчейн — це розподілена база даних, яка підтримує постійно зростаючий список упорядкованих записів, які називаються блоками. Ці блоки пов'язані за допомогою криптографії. Кожен блок містить криптографічний хеш попереднього блоку, мітку часу та дані транзакції [5].

Проте блокчейн технологія відрізняється від традиційних баз даних наявністю кількох ключових аспектів [3]:

1. Незмінність: Однією з ключових особливостей блокчейну є те, що після додавання блоку до ланцюга, змінити його стає неможливо [3].

2. Прозорість: Хоча особиста інформація користувачів може залишатися анонімною, транзакції є прозорими і перевіряються учасниками мережі [3].

3. Безпека: Завдяки криптографії та механізмам консенсусу, таким як Proof of Work або Proof of Stake, блокчейн забезпечує безпеку транзакцій [3].

Нижче наведено таблицю, яка порівнює технологію блокчейну і реляційних баз даних, які на сьогодні можна вважати класичними. Порівняння двох технологій зберігання та опрацювання даних наведено в таблиці 1.

Таблиця 1

Характеристика	Blockchain	Бази даних
Централізація/ Децентралізація	Блокчейн технологія існує тільки в децентралізованому вигляді	Бази даних можуть бути як централізовані так і децентралізовані.
Операції опрацювання даних	CRUD(створення, читання, оновлення, вилучення)	Запис, читання

Продовження табл. 1

Характеристика	Blockchain	Бази даних
Безпека	Блокчейн технологія є дуже надійно і стійкою до вразливостей. Навіть якщо частина нод вийде з ладу блокчейн мережа буде функціонувати і надалі. В додаток, майже неможливо внести неправильні дані з ціллю взлому	Бази даних набагато більш вразливі до збоїв та хакерських атак, бо кожен раз після виходу зі строю бази даних необхіден час для перезавантаження. Тим більш, якщо в хакера буде доступ адміністратора, то він може видалити базу даних чи завдати великої шкоди
Прозорість	Блокчейн повністю прозорий, будь хто може передивитися усі данні та транзакції	Зазвичай бази даних є приватними і доступ до них регламентується адміністратором.
Швидкість роботи	Для опрацювання операції запису блокчейну необхіден певний час, в залежності від навантаження мережі	Операції запису виконуються майже моментально
Масштабованість	Масштабування блокчейнів може бути складнішим, особливо у публічних блокчейнах як Bitcoin[1] або Ethereum	Класичні реляційні бази даних легко масштабуються залежно від потреб користувача
Імутабельність	В блокчейні дані, як правило, імутабельні (незмінні) після їх запису	Традиційні бази даних дозволяють модифікацію та видалення даних
Енергоефективність	Блокчейни, особливо ті, що використовують доказ роботи (Proof of Work), можуть вимагати значних обчислювальних ресурсів	Реляційні бази даних можуть бути відносно ефективними з точки зору використання ресурсів
Програмне забезпечення	Блокчейн є технологією з відкритим вихідним кодом (open-source).	Переважає більшість корпоративних СУБД потребують платної ліцензії
Модель даних	Впорядкований ланцюжок записів (так званих блоків), що постійно довшає.	Всі бази даних є структурованими і підтримують певну модель

Порівняння двох технологій свідчить про їх характерні відмінності та можливість застосування для вирішення різних задач, пов'язаних зі зберіганням та опрацюванням даних. Класичні бази даних охоплюють практично весь спектр задач роботи з даними від їх завантаження до формування різного роду звітних облікових та аналітичних повідомлень. Проте бази даних є менш захищеними і вразливими до різного роду шахрайських операцій та відповідно потребують використання цілої низки організаційних, технологічних та програмних засобів захисту. Блокчейн технології більш підходить для надійного та безпечного зберігання та обміну даними без посередників. Тому, на нашу думку, ствердження, що блокчейн – це розподілена база даних є не зовсім вдалим. Блокчейн – це технологія надійного та захищеного зберігання криптографованих хешованих, організованих у блоки даних, розподілених в одноранговій мережі. Надійність та захищеність даних відкриває широкий простір для інновацій використання блокчейну у різних сферах, від фінансових послуг до ідентифікації особи, від моніторингу ланцюгів поставок до зберігання правових договорів. Блокчейн технологія вже реалізувала низку можливостей, які раніше здавались недосяжними, і продовжує відкривати нові шляхи розвитку в різних секторах:

1. Фінансові послуги: Найбільш відомим застосуванням блокчейну є криптовалюти, які вже викликали справжній переворот в сфері фінансів [2].

Цифровізація науки та сучасні тренди її розвитку

2. Ідентифікація та автентифікація: Інновації в блокчейні пропонують нові методи для управління особистою ідентифікацією в інтернеті, забезпечуючи безпечніше зберігання та управління особистими даними [4].

3. Охорона здоров'я: Застосування блокчейну в медицині може змінити спосіб зберігання та обміну медичними записами, забезпечуючи конфіденційність, доступність і цілісність клієнтських даних [4].

4. Нерухомість: Блокчейн може спростити процеси купівлі-продажу нерухомості, автоматизувати ведення реєстрів власності і зменшити шахрайство у цій сфері [4].

5. Голосування: Електронне голосування на основі блокчейну може стати реальністю, забезпечуючи безпеку, анонімність і незмінність голосів, що може радикально підвищити довіру до виборчих процесів [4].

Ці приклади є лише верхівкою айсберга потенційних застосувань блокчейну. З огляду на швидкий розвиток технології, можна передбачити, що її вплив на майбутнє буде тільки зростати, відкриваючи нові можливості для створення цінності і реформування суспільних процесів.

Блокчейн технологія, яка вже знайшла застосування в різних секторах, має величезний потенціал ще в багатьох напрямках. Розглянемо кілька перспективних сфер, де блокчейн може внести значні інновації та покращення:

1. Інтелектуальна власність та авторське право: Блокчейн може радикально трансформувати управління авторськими правами, забезпечуючи автоматизацію реєстрації і моніторингу використання творів[4].

2. Енергетика: Блокчейн має потенціал революціонізувати енергетичний сектор, дозволяючи створення децентралізованих енергетичних мереж, де споживачі можуть безпосередньо торгувати енергією [4].

3. Ланцюги постачання: Блокчейн може внести ще більшу прозорість і ефективність у ланцюги постачання. Завдяки цій технології компанії зможуть миттєво відстежувати переміщення товарів, автоматично виконувати контракти при досягненні товаром певної точки у ланцюзі, і навіть інтегрувати штучний інтелект для оптимізації логістичних потоків [4].

Крім того, існують сектори, де блокчейн тільки починає знаходити застосування:

1. Освіта: Верифікація освітніх документів та академічних досягнень за допомогою блокчейну може значно знизити ризик шахрайства і спростити процес визнання кваліфікацій між країнами [4].

2. Великі дані та аналітика: Блокчейн може надати новий рівень безпеки та надійності в зборі та зберігання великих обсягів даних.

3. Ігрова індустрія: За допомогою блокчейну можна створювати унікальні віртуальні предмети, що мають власність невід'ємності та можуть бути куплені, продані або обміняні в безпечному середовищі.

4. Юридичні послуги: Автоматизація стандартних юридичних процесів за допомогою розумних контрактів може значно зменшити потребу в ручній роботі юристів та прискорити виконання юридичних операцій [4].

В підсумку, майбутнє блокчейн технології виглядає перспективним. Технологія продовжує розвиватися, і її потенціал у майбутньому може виявитися вирішальним у багатьох аспектах нашого життя.

Список використаних джерел:

1. Технологія блокчейн та її роль в екосистемі Bitcoin [Електронний ресурс] – Режим доступу до ресурсу: https://lb.ua/economics/2023/04/13/550425_tehnologiya_blokcheyn_ii_rol.html.
2. За горизонтом: Блокчейн та майбутнє фінансової інфраструктури [Електронний ресурс] – Режим доступу до ресурсу: <https://www2.deloitte.com/ua/uk/pages/financial-services/articles/gfsi-disruptive-innovation-blockchain.html>.
3. Як працює технологія Blockchain [Електронний ресурс] – Режим доступу до ресурсу: <https://guland.com.ua/kryptovalyuta/blockchain/shcho-take-blokcheyn.htm>.
4. Блокчейн – що це таке, як працює та навіщо потрібен простими словами. [Електронний ресурс] – Режим доступу до ресурсу: <https://termin.in.ua/blokcheyn/>.
5. Blockchain [Електронний ресурс] – Режим доступу до ресурсу: <https://www.synopsys.com/glossary/what-is-blockchain.html>.

Додаток Є



Имя пользователя: Інформаційних систем в економіці Шкуратовська Те...	ID проверки: 1016261620
Дата проверки: 18.05.2024 20:20:21 EEST	Тип проверки: Doc vs Internet + Library
Дата отчета: 18.05.2024 20:24:58 EEST	ID пользователя: 100005745

Название файла: Ублінських А.ІШІ-602

Количество страниц: 85 Количество слов: 12190 Количество символов: 96783 Размер файла: 1.81 MB ID файла: 1016050464

4.89% Совпадения

Наибольшее совпадение: 0.75% с источником из Библиотеки (ID файла: 1015318776)

2.9% Источники из Интернета	165	Страница 87
4.54% Источники из Библиотеки	565	Страница 88

0% Цитат

Исключение цитат выключено

Исключение списка библиографических ссылок выключено

0% Исключений

Нет исключенных источников

Модификации

Обнаружены модификации текста. Подробная информация доступна в онлайн-отчете.

Замененные символы 5