# 4 РОЗРОБКА ТА ТЕСТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ

## 4.1 Особливості програмної реалізації системи

В попередньому розділі було виділено декілька окремих систем, а саме: API що реалізує взаємодію між клієнтами, API що реалізує взаємодію з локальною базою даних, система управління блокчейном, та система для UI відображення роботи блокчейну.

### 4.1.1 Розробка API для роботи з базою даних

Для простої та швидкої взаємодії з БД використовується Entity Framework. Він дозволяє створювати та взаємодіяти з базою даних не пишучи sql запитів.

Entity Framework — це платформа ORM з відкритим вихідним кодом для додатків .NET, яку підтримує Microsoft. Це дозволяє розробникам працювати з даними, використовуючи об’єкти специфічних для домену класів, не зосереджуючи увагу на базових таблицях і стовпцях бази даних, де ці дані зберігаються. За допомогою Entity Framework розробники можуть працювати на більш високому рівні абстракції, коли вони мають справу з даними, а також можуть створювати та підтримувати орієнтовані на дані програми з меншою кількістю коду в порівнянні з традиційними програмами. [14]

Для створення, або використання бази даних достатньо вказати назву та розташування в App.config файлі, як показано на рисунку 4.1.

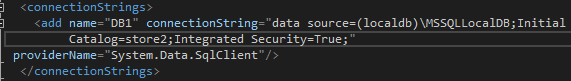


Рисунок 4.1 – Вміст App.config файлу

Для створення таблиці достатньо створити спеціальний класс – «Context», що містить в собі визначення вмісту таблиці. Об’єкт типу Block позначений атрибутом – DataContract, що свідчить про можливість його парсингу в таблицю бази даних. Контекст блока показано на рисунку 4.2.

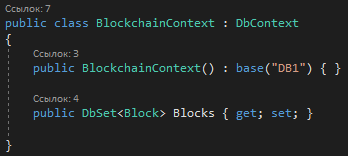


Рисунок 4.2 – Контекст блоку

Для доступу до даних достатньо створити об’єкт класу контексту та взаємодіяти з ним як зі звичайним списком блоків, як показано на рисунку 4.3.

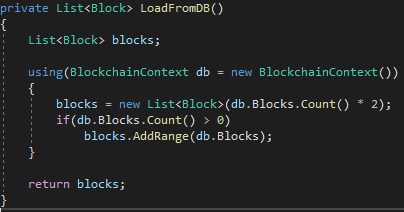


Рисунок 4.3 – Метод отримання списку блоків з БД

Для внесення змін необхідно напрямку про взаємодіяти з списком об’єктів і після змін, виконати метод - SaveChanges.

Розглянуті опції у використанні, повністю задовольняють потреби в локальній базі даних. Розглянувши їх реалізацію за допомогою Entity Framework можна сміливо констатувати, що вибір в фреймворці виправданий і є досить швидким, простим та зручним.

### 4.1.2 Розробка P2P API

Для побудови P2P з’єднання буде використовуватисчя бібліотерка System.Net.PeerToPeer. Ця бібліотека надає можливість для створення унікальних однорангових клієнтів (клас PeerName), реєстрації цих клієнтів в мережі (клас PeerNameRegistration), вирішення конфліктів між портами клієнтів (PeerNameResolver) та передачі даних між клієнтах.

Для передачі даних потрібно створюючи пір створити в ньому канал, через який буде здійснюватися передача даних. В цей канал слід додати клас в якому буде описано механізми передачі та обробки даних (сервіси) позначивши його атрибутами ServiceContract та для методів OperationContract. Після цього викликавши метод контракту цей метод буде викликано у всіх користувачів мережі. Приклад сервісу, що описує взаємодію з каналом пірингової мережі показано на рисунку 4.4.

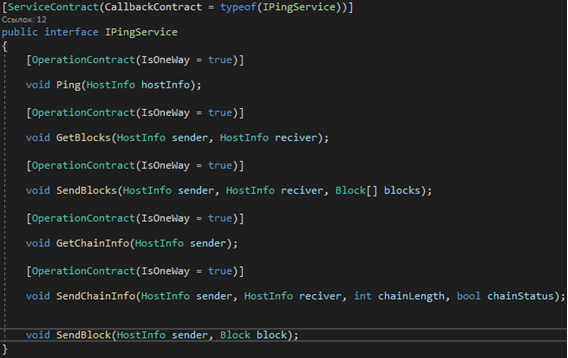


Рисунок 4.4 – сервіс взаємодії з мережею P2P

### 4.1.3 Розробка системи управління блокчейном

Блокчейн система складається з декількох основних класів, це класс Block, та клас Chain.

Клас Block містить у собі всі ,вище зазначені у моделі блоку, поля. Методі хешування та перевірки хешу. І методи серелізації та десерелізації об’єкта.

Клас Chain містить у собі список блоків, посилання на P2P сервіс і посилання на самого себе (реалізує сінглтон). Також містить методи завантаження даних з локального сховища, порівняння з даними інших користувачів. І методи завантаження, відправки та синхронізації з глобальною мережею.

### 4.1.4 Розробка інтерфейсу

При розробці інтерфейсу користувача використовувалися стандартні форми WPF, зі стандартними UI елементами.

Windows Presentation Foundation (WPF) — це структура інтерфейсу користувача, яка створює настільні клієнтські програми. Платформа розробки WPF підтримує широкий набір функцій розробки додатків, включаючи модель програми, ресурси, елементи керування, графіку, макет, прив’язку даних, документи та безпеку [15].

WPF є частиною .NET, тому, якщо ви раніше створювали програми з .NET за допомогою ASP.NET або Windows Forms, досвід програмування повинен бути знайомим. WPF використовує Extensible Application Markup Language (XAML) для забезпечення декларативної моделі для програмування програм.

Основним в даній програмній системі є саме блокчейн, а зробивши невелику кількість маніпуляцій з системою управління блокчейну та з UI інтерфейсом можна, повністю змінити функціонал додатку інкапсулюючи дані блоків.

Для демонстрації було створено додаток, що містить чотири форми: логін, реєстрація, основна форма, та форма для перегляду результатів пошуку.

Сторінка реєстрації дозволяє зберігати новоствореного користувача в блокчейні, в блоці спеціального типу – User.

Сторінка логіну дає можливість зайти під ім’ям користувача, який вже міститься в блокчейні, або вирушити до сторінки реєстрації, щоб додати нового користувача.

Основна сторінка зверху відображає інформацію про користувача (логін, порт та uri end point). Майже всю область цієї сторінки займає список з блоків, що вже додані до блокчейну. Нажавши на будь який єлемент списку можна зробити 4 операції з цим блоком :

- скопіювати хеш блоку;

- скопіювати ім’я користувача, що додав блок;

- переглянути вміст блоку (якщо це просте повідомлення, то воно відобразиться в вікні повідомлень, а якщо це якийсь файл, то він тимчасово завантажиться на комп’ютер та відкриється);

- скачати контент блоку на комп’ютер (якщо у блоці вказано тип файлу, то файл завантажиться з даним типом).

В лівому нижньому куті знаходиться функціонал для додавання нових блоків, або з звичайним строковим повідомленням, або передавши будь який файл з комп’ютера.

Також по середині міститься пошукова стрічка в яку можна вводити хеш блоку, або ім’я користувача і відповідно отримувати результат на свій запит в формі отримання результатів.

Дані форми показано на рисунку 4.5 – 4.8.

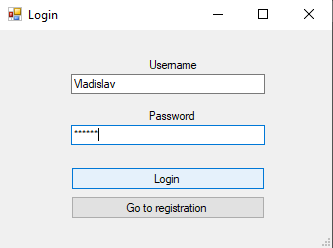


Рисунок – 4.5 Форма логіну

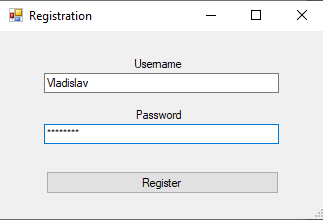


Рисунок 4.6 – Форма регістрації

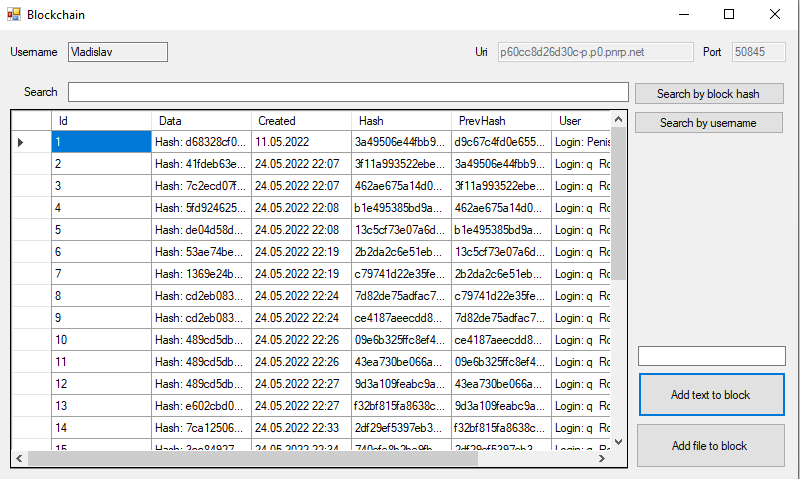


Рисунок 4.7 – Основна форма взаємодії з блокчейном

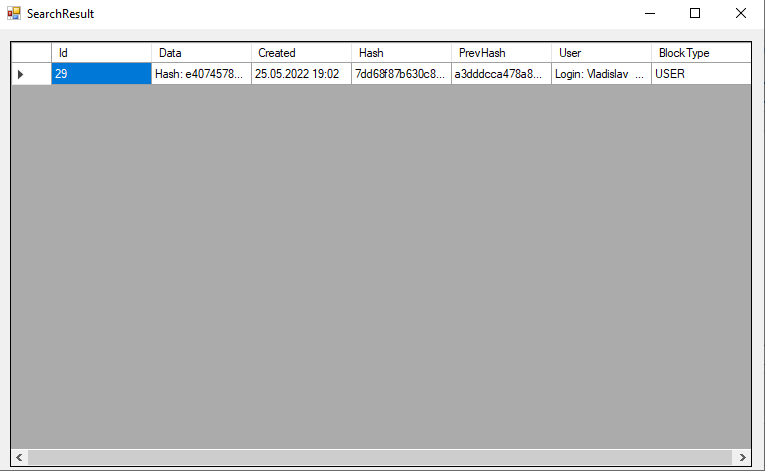


Рисунок 4.8 – Форма з результатами пошуку

## 4.2 Процес тестування

Тестування це перевірка відповідності між реальною поведінкою програми та її очікуваним поведінкою на кінцевому наборі тестів, обраному певним чином.

Тестування має на меті виявлення дефектів, підвищення впевненості в рівні якості, запобігання дефектів [16].

Для тестування програмної системи на етапі розробки, кожного разу при написанні якогось модуля, проводилось Unit тестування для цього модуля.

Модульний тест (Unit testing) — це спосіб тестування одиниці найменшої частини коду, яку можна логічно виділити в системі. У більшості мов програмування це функція, підпрограма, метод або властивість. Ізольована частина визначення має важливе значення. [17].

Також на при завершенні розробки, було проведено ряд тестування на відповідність функціональним та нефункціональним вимогам (функціональне тестування та нефункціональні тестування).

### 4.2.1 Модульне тестування

Обрана середа розробки Visual studio даю можливість в автоматичній генерації шаблонів для модульного тестування. Приклад такого шаблона показано на рисунку 4.9.

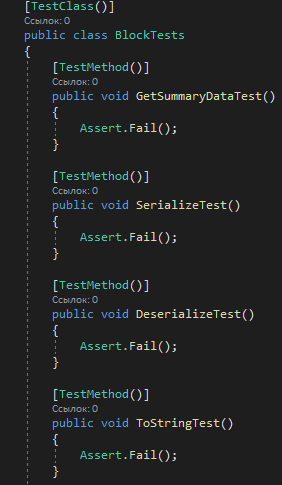


Рисунок 4.9 – Згенерований шаблон модульного тестування для класу блок.

В такий шаблон слід записати логіку тестування, та викликати спеціальний метод – Assert.AreEqual() для того, щоб порівняти очікуваний результат виконання метода з тим, що справді виходить в результаті виконання метода. Приклад написаного модульного тесту, для копіювання логіна користувача в буфер обміну, показано на рисунку 4.10.

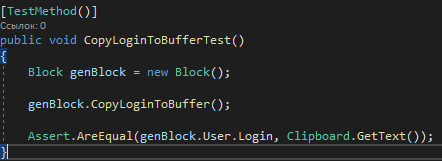


Рисунок 4.10 – Модульний тест для метода копіювання логіна в буфер

Результат модульного тестування для усіх основних класів показано на рисунку 4.11.

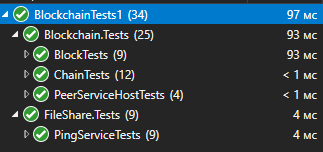


Рисунок 4.11 – Результат модульного тестування

### 4.2.2 Функціональне тестування

Згідно до раніше представлених функціональних вимог система має надавати змогу зареєструватися або увійти, додати новий токен до блокчейну, та переглянути усі наявні блоки у блокчейні. Усі поставлені вимоги система задовольняє.

Деякий додатковий функціонал було додано до додатку на основі системи токенізації. Наприклад пошук за ключами хешування і логінами користувачів, та збереження і перегляд копій контенту на комп’ютер, для подальшого користування без змоги використання мережі.

### 4.2.3 Нефункціональне тестування

Продуктивність – відповідно до вимог генерація блоку, та додаванн його до глобального блокчейну має відбуватися менш ніж за 30 секунд. Система повністю відповідає цій вимозі. При передачі великих файлі таких як ,наприклад, картинка система додає блок за менш ніж одну секунду.

Доступність у використанні – відповідно до вимог інтерфейс має бути зрозумілим, для рядового користувача освоїтись повинно займати до 1 робочого дня. Система має дуже простий інтерфейс роботи з блокчейном, яка надає невелику кількість можливостей, але ці можливості можна використовувати задовольняючи будь які подальші надбудови над системою, наприклад інтеграцію аналогів смарт контрактів.

Надійність – система має стабільно працювати з мінімальною кількістю зависань при умові середнього навантаження на неї. Вході тестування на стресостійкість система гарно себе показала. При завантажені 40 блоків з яких близько половини були файли середнім розміром у 500 кб, час очікування повністю новоствореного аканта на завантаження склало менш ніж 1 секунда на одному пристрої. А отже швидкість завантаження на різних пристроях може залежати в більшій мірі лиш від швидкості інтернета конкретного користувача.

Безпека – можливість підробити дані має бути можлива лиш в випадку переваги більш ніж 50% у апаратних можливостях конкретного користувача. Цей пункт також повністю задовільнений. При тестуванні двох програм, що використовують один апаратний потенціал, але звертаються до різних локальних сховищ, не вдалося обманути систему зламавши ланцюжок даних. Некоректні дані одразу затираються.

Локалізація – система повністю локалізована на англійській мові, відповідно до вимог.

Технічні вимоги – система працює на операційній системі Windows 10 та потребує менш ніж 8 гб оперативної пам’яті, згідно до раніше поставлених вимог.

Інтерфейс – повністю відповідає стилістичним вимог, що дули поставлені раніше. Приклад інтерфейсу було представлено на рисунках 4.5-4.8.

## 4.3 Приклади використання

Розроблену систему можна використовувати наприклад в системах, що мають на меті медичний облік історій хвороб пацієнтів. Або звичайне збереження документів, наприклад про закінчення вищої освіти чи проходження якогось курсу. Також, можна зберігати невеликі програми, що можна використовувати як спрощену альтернативу смарт-контрактів Ethereum.

Усе це можна досягти не змінюючи саму систему, а лиш змінюючи найвищий шар взаємодії з системою та інтерфейс користувача, що є дуже вагомим плюсом системи.

В рамках демонстрації цих можливостей, було створено невелику програму для обліку історій хвороб та курсу лікування, що призначається хворому. Це, наприклад, дозволить притягнути некомпетентного лікаря до відповідальності у разі призначення ним некоректного курсу лікування чи некоректно поставленого діагнозу. Дана програма зображена на рисунках 4.12-4.13.

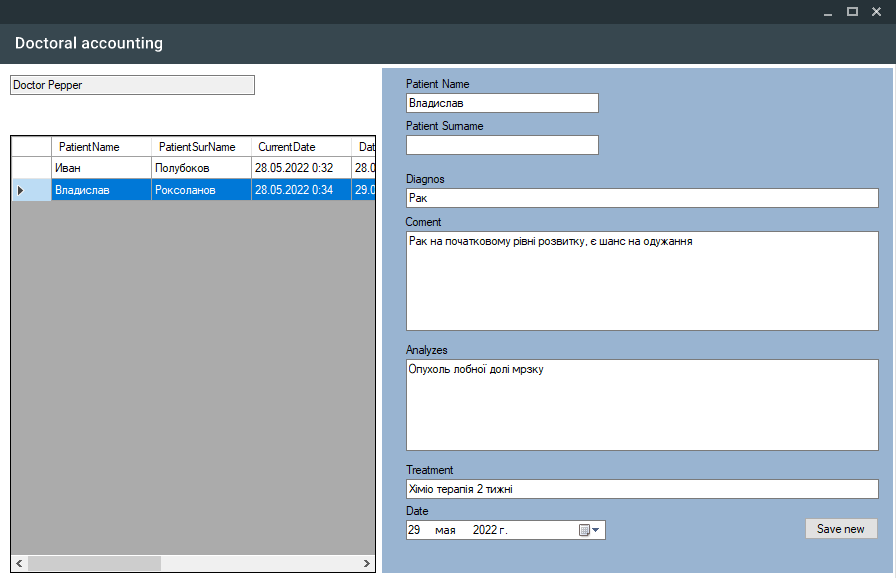


Рисунок 4.12 – Головна сторінка додатку

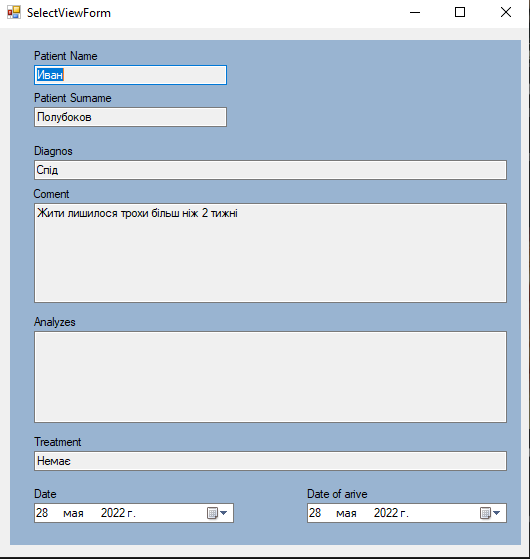


Рисунок 4.13 – Форма перегляду конкретної історії хворого

## 4.4 Можливі вдосконалення системи

При розробці програмного продукту були виявлені недоліки в архітектурі взаємодій між системою токенізації та системою Peer-To-Peer передачі даних. Ці проблеми слід виправити для більшої надійності, та покращенню можливостей подальших модифікацій.

Також слід провести оптимізацію збережень даних при ввімкненому застосунку, тому що, на разі усі блоки мають бути повністю завантажитися в оперативну пам’ять перед локальним збереженням, що є не оптимальним рішенням. А також слід створити механізми які будуть вирішувати питання постійного пере-записування локальної бази даних при вході в систему, тому що при подальшому масштабуванні, вхід в додаток може тривати вагомий відрізок часу.