МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет водного господарства та природокористування
Навчально-науковий інститут автоматики, кібернетики та обчислювальної техніки
Кафедра комп'ютерних технологій та економічної кібернетики

Допущено до захисту:

Завідувач кафедри

_____ д. е. н., проф. П. М. Грицюк

«_____» _____
2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня «бакалавр»
за освітньо-професійною програмою «Інформаційні системи і технології»
спеціальності 126 «Інформаційні системи та технології»

на тему: «**Маркетплейс для NFT токенів**»

Виконав:
здобувач вищої освіти 4 курсу, групи ІСТ-41
Опанасюк Михайло Юрійович

Керівник:
канд. техн. наук, доцент Гладка О. М.

Рецензент:
канд. техн. наук, доцент Барановський С. В.

Рівне – 2023

Національний університет водного господарства та природокористування
ННІ автоматики, кібернетики та обчислювальної техніки
Кафедра комп'ютерних технологій та економічної кібернетики

Освітньо-кваліфікаційний рівень – **бакалавр**

за освітньо-професійною програмою **«Інформаційні системи і технології»**

спеціальність **126 «Інформаційні системи та технології»**

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ д. е. н., проф. П. М. Грицюк

«_____» _____ 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачу _____Опанасюку Михайлу Юрійовичу_____
(прізвище, ім'я, по-батькові)

1. Тема роботи _____Маркетплейс для NFT токенів_____

_____

керівник роботи: __Гладка Олена Миколаївна, канд. техн. наук, доцент_____
(прізвище, ім'я, по-батькові, науковий ступінь, вчене звання)

затверджена наказом по університету від "19" квітня 2023 р. С № 440

2. Термін здачі здобувачем закінченої роботи "09" червня 2023 р.

3. Вихідні дані до роботи _розробити програму, що є платформою (маркетплейсом) для продажу, зберігання та перегляду NFT токенів.

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) __вступ; аналіз предметної області; вибір та обґрунтування засобів розробки програмного продукту; розробка і опис створеної платформи.

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

_____

_____

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1.	Гладка О. М., доцент		
2.	Гладка О. М., доцент		
3.	Гладка О. М., доцент		

7. Дата видачі завдання "_07_" листопада 2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Термін виконання етапів роботи	Примітка
1.	Аналіз об'єкта дослідження, виявлення існуючих проблем	07.11.22 – 25.11.22	
2.	Аналіз існуючих інформаційних методів (технологій) вирішення проблеми	28.11.22 – 23.12.22	
3.	Вибір та обґрунтування засобів розробки програмного продукту	30.01.23 – 24.02.23	
4.	Проектування, розробка та реалізація програмної платформи	27.02.23 – 30.04.23	
5.	Підготовка тексту кваліфікаційної роботи	10.04.23 – 26.05.23	
6.	Підготовка презентації роботи	29.05.23 – 09.06.23	
7.	Відгук керівника, рецензування роботи, перевірка на плагіат	09.06.23 – 16.06.23	

Здобувач _____ ***М. Ю. Опанасюк***
(підпис) (прізвище і ініціали)

Керівник кваліфікаційної роботи

О. М. Гладка
(підпис) (прізвище і ініціали)

АНОТАЦІЯ

Опанасюк М.Ю. Розробка маркетплейсу NFT токенів.

Кваліфікаційна робота на здобуття ступеня «бакалавр».

Об’єктом дослідження є процес керування NFT токенами, їх створення, продаж тощо.

Предметом дослідження – розробка та запровадження маркетплейсу NFT токенів.

Розроблений застосунок є мікросервісом, що включає в себе API для роботи з NFT токенами і клієнт частину, яка надає зручний спосіб користування маркетплейсом користувачу. Бекенд частина була розроблений на мові програмування C#, в той час як фронтенд частина, використовує React та Redux. В ході розробки були використані методи ООП, система контролю версій Git, а саме Github, робота з потоками, асинхронність та імплементовані принципи SOLID. Застосунок дозволяє користувачам отримувати доступ до NFT токенів, виконувати різноманітні операції з ними та здійснювати управління токенами. Бібліотеки, що використовуються у застосунку, забезпечують стабільність та безпеку в процесі роботи з токенами. Даний застосунок дозволяє забезпечити ефективний та безпечний обмін NFT токенами між різними системами та користувачами.

Результатом виконання роботи є програмний продукт, у якому втілені принципи ООП, SOLID та KISS, а сам застосунок побудований за принципом Clean Architecture (чистої архітектури, що надасть легке подальше розширення масштабів проекту).

ЗМІСТ

ВСТУП

Комп’ютерні технології в сьогоденні відіграють важливу роль у розробці нових продуктів і послуг, які допомагають спрощувати та поліпшувати багато аспектів нашого життя. Однією з таких інновацій є застосунок про NFT токени. Застосунок про NFT токени є досить актуальною темою в сучасному світі, особливо в галузі криптовалют та блокчейн технологій. За останній час, NFT токени стали популярними серед художників, музикантів та інших творчих людей, оскільки вони дозволяють зберігати та продавати цифрові твори як унікальні об’єкти мистецтва. NFT токени можуть бути використані для зберігання цифрових активів, таких як музика, відео, фотографії та інші цифрові матеріали.

NFT (Non-Fungible Token) - це технологія блокчейн, яка дозволяє створювати цифрові токени, що не є замінними один на один. Кожен NFT токен має свій унікальний ідентифікатор, який дозволяє ідентифікувати його як унікальний об’єкт. Це дозволяє створювати цифрові активи, які можуть бути продані як унікальні предмети мистецтва, а також використовувати їх для зберігання інших цифрових активів, таких як музика, відео, фотографії та інші матеріали.

Взагалі існує декілька типів NFT токенів, які використовуються в різних галузях. Одним з типів є токени ERC-721, які використовуються в Ethereum блокчейні для створення унікальних об’єктів мистецтва та інших цифрових активів. Іншим типом є токени ERC-1155, які дозволяють створювати як унікальні, так і замінні токени, що можуть використовуватися в іграх та інших онлайн-платформах.

Підсумуємо, для чого ж потрібні NFT токени? З одного боку, вони дозволяють митцям та творцям контенту отримувати за свої творіння пряму оплату, яку неможливо підробити або підробити права на твір. З іншого боку, власники NFT токенів мають унікальну можливість володіти цифровими творами та використовувати їх на свій розсуд. Щодо типів NFT токенів, то їх можна розділити на дві категорії: токени, що використовуються для

володіння фізичними предметами (наприклад, мистецькими роботами) та токени, що використовуються для володіння цифровими контентом (наприклад, музичними альбомами чи відео).

Щодо проблем, які вирішують NFT, є питання походження та права власності. На традиційних ринках мистецтва або предметів колекціонування доведення автентичності твору або права власності на нього може бути громіздким процесом. Однак за допомогою NFT автентичність цифрового активу можна перевірити за допомогою блокчейну, а право власності можна легко передати одним натисканням кнопки. Це значно полегшує художникам і колекціонерам управління та продаж своїх цифрових активів. Щоб полегшити створення, управління та продаж NFT, в останні роки з'явилися різні додатки та платформи. Ці платформи надають користувачам можливість легко створювати, купувати, продавати та обмінювати NFT. Крім того, багато з цих платформ пропонують такі функції, як цифрові гаманці, маркетплейси та профілі виконавців, щоб ще більше покращити користувацький досвід. Однією з таких платформ є OpenSea, популярний ринок для купівлі та продажу NFT. OpenSea дозволяє користувачам створювати і продавати власні NFT, а також переглядати і купувати NFT, створені іншими. Платформа набула популярності в останні роки завдяки зручному інтерфейсу, широкому спектру доступних NFT і низьким комісіям за транзакції.

Отже, NFT - це революційна технологія, яка змінила ринок предметів мистецтва та колекціонування, дозволивши створювати та торгувати унікальними цифровими активами, що піддаються перевірці. Зі зростанням популярності NFT для приватних осіб та бізнесу важливо мати всебічне розуміння їхнього використання та потенційних застосувань.

Метою цього дослідження є створення клієнт-серверної аплікації для керування NFT, а також дослідження того, як ця технологія змінює способи управління та торгівлі цифровими активами.

Об'єкт дослідження: "Мікросервісна архітектура для керування NFT-токенів". Предмет дослідження: "Розробка системи управління токенами NFT".

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ І ТЕХНОЛОГІЙ РОЗРОБКИ

Технологічний стек, який використовується в застосунку токенів NFT, складається з декількох популярних і широко використовуваних інструментів і фреймворків для розробки веб-додатків. Оскільки це клієнт-серверний застосунок, то було розроблено дві частини: бекенд та фронтенд. Бекенд стек включає Entity Framework Core, MS SQL Server, патерни UnitOfWork і Repository, патерн MediatR, аутентифікацію JWT, логування NLog, Swagger і механізм Cors.

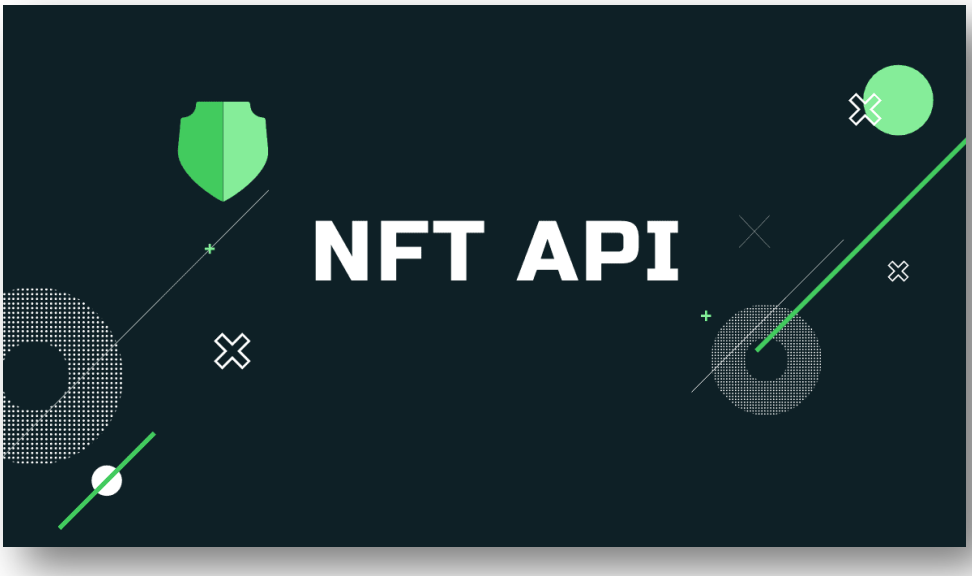

Рис. 1.1. Тематика застосунку

Entity Framework Core - це фреймворк об'єктно-реляційного відображення (ORM) з відкритим вихідним кодом, розроблений компанією Microsoft. Він дозволяє розробникам працювати з базами даних за допомогою об'єктів .NET і надає потужний набір інструментів для запитів, оновлення та маніпулювання даними. Фреймворк можна використовувати з різними

системами управління базами даних, включаючи MS SQL Server, який також використовується в додатку API токенів NFT.

MS SQL Server - це реляційна система управління базами даних, розроблена компанією Microsoft, яка забезпечує високомасштабовану, надійну і безпечну платформу для зберігання і управління даними. Ось декілька плюсів, щоб обгрунтувати вибір саме цієї реляційна системи управління БД:

- MS SQL Server є підтримка розширених функцій безпеки, включаючи безпеку на основі ролей, шифрування та безпечні протоколи зв'язку. Це особливо важливо в контексті додатку токенів NFT, який має справу з конфіденційними даними і вимагає високого рівня безпеки.
- MS SQL Server має підтримку транзакційної обробки даних. Це має вирішальне значення для додатку токенів NFT, де транзакція являє собою передачу унікального цифрового активу від одного власника до іншого. MS SQL Server гарантує, що ці транзакції обробляються надійно і ефективно, забезпечуючи цілісність і узгодженість бази даних.
- MS SQL Server надає ряд розширених функціональних можливостей, включаючи розширену аналітику, машинне навчання та бізнес-аналітику. Ці функції можна використовувати для отримання інформації про використання та продуктивність додатку токенів NFT, а також для проведення розширеної аналітики даних, що зберігаються в базі даних.
- З точки зору своєї ролі в нашому додатку токенів NFT, MS SQL Server служить основним механізмом зберігання даних для додатку. Він зберігає інформацію про цифрові активи, якими торгують на платформі, а також дані про користувачів, транзакції та інші метадані. Ядро Entity Framework Core використовується для взаємодії з базою даних, дозволяючи розробникам легко запитувати і маніпулювати даними, що зберігаються в ній.

Саме тому, MS SQL Server - це надійна, масштабована і безпечна система управління базами даних, яка добре підходить для додатків корпоративного рівня, таких як платформа токенів NFT. Підтримка розширених функцій безпеки, обробки транзакцій і розширеної функціональності роблять його ідеальним вибором для управління конфіденційними даними і складними транзакціями, які використовуються в додатку токенів NFT.

Патерни UnitOfWork і Repository - це патерни проектування, які широко використовуються при розробці веб-додатків. Патерни Unit of Work (UoW) та Repository часто використовуються разом, щоб покращити зручність супроводу, читабельність та тестованість коду. Ці патерни реалізовані за допомогою Entity Framework Core в нашому додатку для токенів NFT для забезпечення стійкості даних.

- Паттерн UnitOfWork використовується для управління транзакціями та забезпечення узгодженості всіх змін, що вносяться до бази даних, а паттерн Repository надає спосіб інкапсуляції логіки запитів та маніпуляцій з даними. Паттерн UoW відповідає за координацію декількох операцій в рамках однієї транзакції. Іншими словами, він гарантує, що всі операції виконуються в одній атомарній транзакції, так що або всі вони будуть виконані, або жодна з них не буде виконана. Патерн UoW використовується для забезпечення цілісності та узгодженості даних. Цей патерн особливо корисний у ситуаціях, коли в рамках однієї операції необхідно внести декілька змін до бази даних. Наприклад, при створенні нового токена NFT шаблон UoW гарантує, що всі пов'язані з ним дані (такі як метадані, транзакції та інформація про власника) будуть збережені в базі даних за одну транзакцію.
- Паттерн Repository відповідає за абстрагування рівня доступу до даних додатку. У нашому додатку токенів NFT паттерн Repository використовується для забезпечення узгодженого інтерфейсу для роботи з нашими сутностями даних. Цей патерн гарантує, що реалізація рівня доступу до даних може бути легко замінена, модифікована або

розширена без впливу на решту програми. Наприклад, якщо ми вирішимо перейти з MS SQL Server на іншу систему баз даних, нам потрібно буде змінити лише реалізацію паттерну Repository, а решта програми залишиться незмінною.

Патерн MediatR - це патерн проектування, який дозволяє розділити зв'язок між компонентами системи. MediatR використовується для реалізації архітектурного патерну CQRS (Command Query Responsibility Segregation) в нашому додатку токенів NFT. Паттерн MediatR відповідає за відокремлення відправника запиту (тобто команди або запиту) від одержувача запиту (тобто обробника). Цей патерн гарантує, що команди і запити обробляються окремими класами, кожен з яких несе окрему відповідальність. У нашому додатку для роботи з токенами NFT паттерн MediatR використовується для реалізації бізнес-логіки додатку. Наприклад, коли створюється новий токен NFT, в застосунок надсилається команда, і виконується відповідний обробник для створення токена і пов'язаних з ним метаданих. Паттерн MediatR гарантує ж, що обробник відокремлений від відправника команди, що дозволяє нам змінювати або розширювати поведінку обробника, не впливаючи на решту програми.

Разом паттерни UoW, Repository і MediatR забезпечують надійну і масштабовану архітектуру для нашого додатку токенів NFT, а також гарантують, що наш рівень доступу до даних абстрагований, бізнес-логіка - відокремлена, а транзакції - атомарні та послідовні.

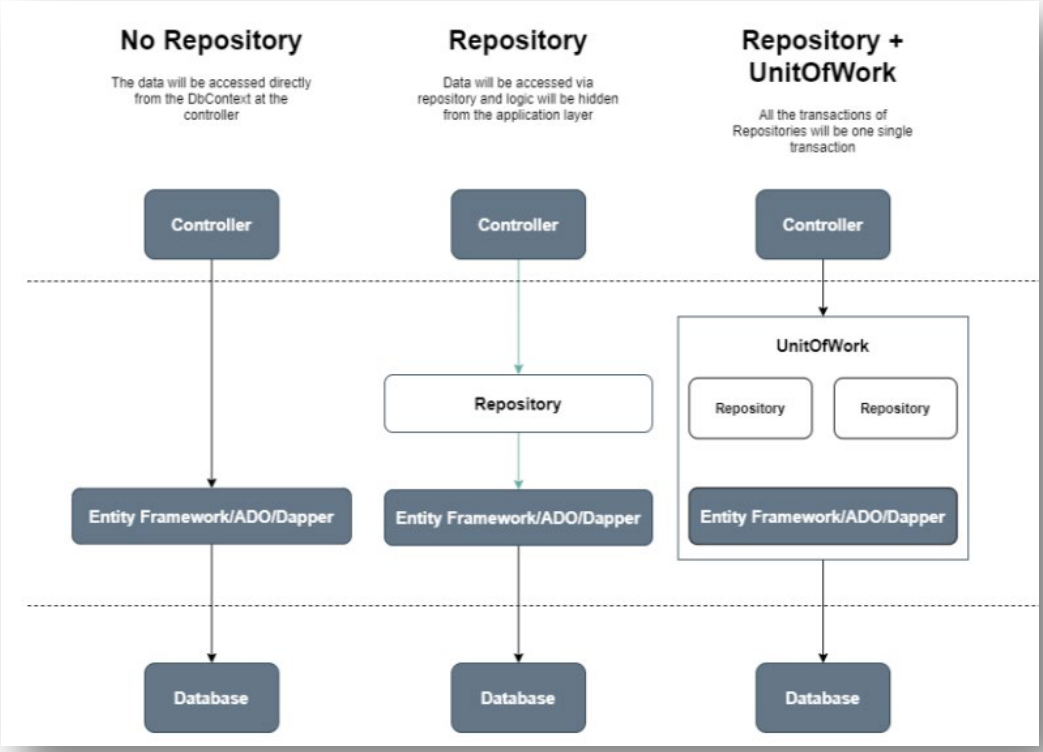

Рис. 1.2. Схема патерну UnitOfWork та Repository

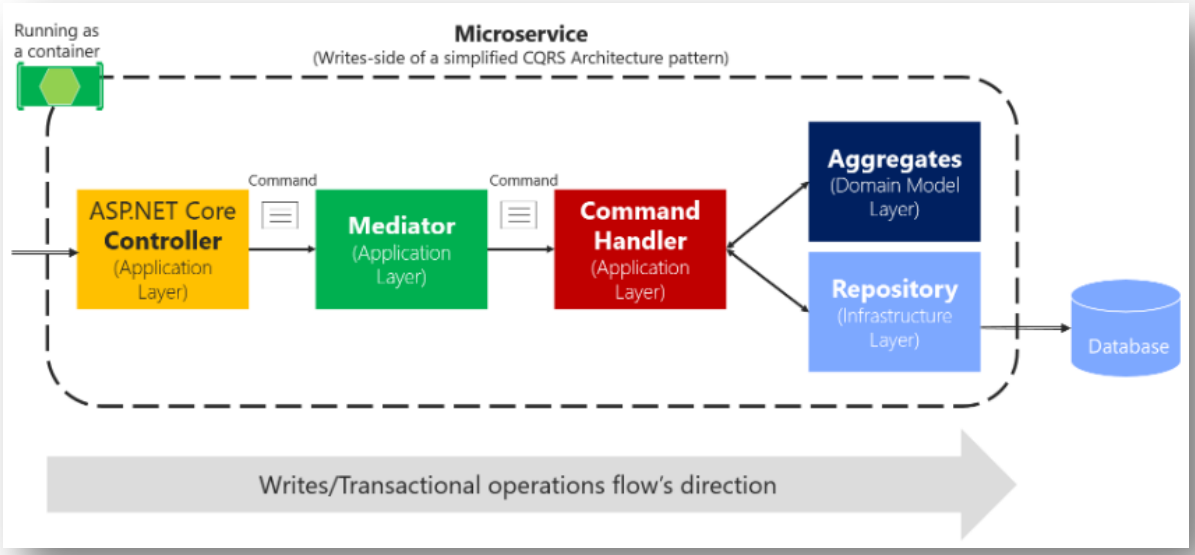

Рис. 1.3. Схема патерну MediatR у застосунку

JWT-аутентифікація - це стандарт сучасних веб-додатків для безпечної передачі даних автентифікації користувача між клієнтом і сервером. У цьому додатку JWT використовується для захисту маршрутів, які вимагають

автентифікації. Коли користувач входить в систему, сервер генерує токен JWT і надсилає його клієнту, який потім зберігає його в локальному сховищі. При наступних запитах клієнт надсилає токен у заголовку авторизації, а сервер перевіряє підпис токена, щоб переконатися, що він не був підроблений і що він все ще дійсний. Це допомагає запобігти несанкціонованому доступу до захищених маршрутів і гарантує, що тільки автентифіковані користувачі можуть отримати доступ до конфіденційної інформації.

JWT-аутентифікація - популярний метод захисту веб-додатків. Він використовує JSON-мітки (JWT) для автентифікації та авторизації користувачів і забезпечує спосіб безпечної передачі інформації між клієнтом і сервером. Цей метод автентифікації є більш безпечним, ніж традиційні методи, такі як файли cookie та ідентифікатори сеансів.

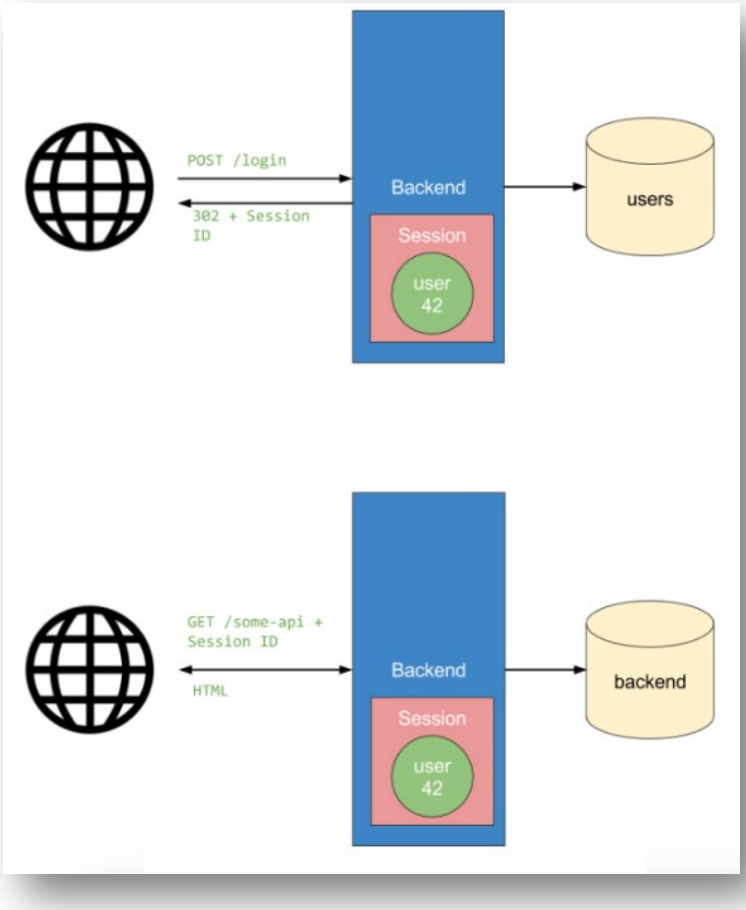

Рис. 1.4. Загальний принцип роботи API

NLog logging - це гнучкий і розширюваний фреймворк для ведення журналів, який дозволяє розробникам збирати і аналізувати журнали своїх

додатків. Він надає можливість збирати та зберігати дані про помилки, попередження та інші події, які виникають під час виконання програми, і може допомогти діагностувати та усувати проблеми. NLog використовується для реєстрації у цій програмі. Це гнучкий і розширюваний фреймворк для ведення журналів, який може записувати дані до різних об'єктів, включаючи файли, бази даних і електронну пошту.

NLog дозволяє розробникам реєструвати різні типи повідомлень з різним рівнем серйозності, такі як налагодження, інформація, попередження, помилки і фатальні. Використання дійсно корисно для налагодження та моніторингу програми у виробничих середовищах. Він також має підтримку структурованого логування, що дозволяє розробникам записувати структуровані дані у вигляді, який можна легко шукати та аналізувати.

Swagger і CORS (Cross-Origin Resource Sharing) - два важливих компоненти в нашому додатку API токенів NFT.

Swagger - це інструмент з відкритим вихідним кодом, який допомагає створювати документацію API. Він дозволяє нам описувати структуру нашого API за допомогою формату YAML або JSON і автоматично генерує інтерактивний інтерфейс, який розробники можуть використовувати для вивчення і тестування API. Ця документація є безцінною для розробників, яким потрібно інтегруватися з нашим API, оскільки вона надає чіткий і стислий огляд кінцевих точок, їх входів і виходів, а також будь-якої необхідної автентифікації або авторизації.

Однією з ключових переваг використання Swagger в нашому додатку API токенів NFT є те, що він допомагає стандартизувати нашу документацію API. Визначаючи структуру нашого API за допомогою специфікації OpenAPI, ми можемо гарантувати, що всі наші кінцеві точки задокументовані узгоджено. Це полегшує розробникам розуміння нашого API та створення інтеграцій з ним. Крім того, Swagger має вбудований інтерфейс, який дозволяє легко вивчати і тестувати API без необхідності писати власний код.

CORS - це механізм безпеки, який використовується для обмеження доступу веб-додатків до ресурсів на іншому домені. У нашому додатку API токенів NFT ми використовуємо CORS, щоб визначити, яким доменам дозволено доступ до нашого API. Це важливо з міркувань безпеки, оскільки допомагає запобігти зловмисним атакам, які можуть скомпрометувати наш API або його дані.

Однією з переваг використання CORS в нашому додатку API токенів NFT є те, що він дозволяє обмежити доступ до нашого API тільки довіреними доменами. Це допомагає запобігти несанкціонованому доступу до нашого API і його ресурсів, а також гарантує безпеку наших даних. Крім того, CORS надає стандартний механізм контролю перехресних запитів, що полегшує впровадження та підтримку політик безпеки в різних частинах нашого додатку.

Отже, Swagger допомагає стандартизувати нашу документацію API і надає розробникам простий у використанні інтерфейс для вивчення і тестування API, а CORS допомагає забезпечити безпеку нашого API, обмежуючи доступ до довірених доменів. Разом два зазначених компоненти забезпечують міцну основу для створення безпечного, надійного і добре задокументованого додатку API токенів NFT.

Усі ці технології можуть взаємодіяти один з одним для створення надійного і масштабованого додатку API токенів NFT. Наприклад, патерни UnitOfWork і Repository можна використовувати з ядром Entity Framework Core для управління транзакціями і запитами до бази даних, в той час як патерн MediatR можна використовувати для обробки запитів і відповідей між компонентами. Для захисту API можна використовувати автентифікацію JWT, а для збору та аналізу логів - NLog. Swagger можна використовувати для документування API та забезпечення зручного інтерфейсу для розробників, а механізм Cors - для контролю доступу до ресурсів. Також, цей стек технологій надає потужний набір інструментів для розробки додатків API

токенів NFT. Він дозволяє розробникам працювати з базами даних, управляти транзакціями, обробляти запити і відповіді, захищати API, збирати і аналізувати логи, а також документувати API у зручний для користувача спосіб. Ці технології необхідні для створення масштабованого і надійного додатку API токенів NFT, здатного задовольнити потреби користувачів в динамічному цифровому ландшафті, що постійно розвивається.

C# - популярна мова програмування, яка широко використовується для створення додатків корпоративного рівня, включаючи веб-додатки та сервіси, десктопні додатки, ігри тощо. В останні роки C# стала одним з найкращих варіантів для розробки додатків на основі блокчейну, в тому числі додатків для токенів NFT. Однією з головних причин, чому C# є гарним вибором для розробки додатків для токенів NFT, є його надійність та масштабованість.

Окрім цього, C# - це об'єктно-орієнтована мова програмування, яка дозволяє розробникам писати чистий, підтримуваний і багаторазовий код, що полегшує створення і підтримку складних додатків. C# має велику стандартну бібліотеку та багатий набір вбудованих функцій, які допомагають розробникам писати ефективний та надійний код.

- C# також має чудову підтримку багатопотоковості, яка необхідна для створення високопродуктивних додатків, таких як додатки для токенів NFT, що вимагають оновлення в реальному часі та обробки великих обсягів даних. C# підтримує асинхронне програмування, що дозволяє розробникам писати код, який може виконуватися паралельно, не блокуючи основний потік, що призводить до швидшого та ефективнішого коду.
- Перевагою використання C# для додатків токенів NFT є його сумісність з іншими мовами програмування та фреймворками. C# розроблена для безперешкодної роботи з іншими технологіями Microsoft, такими як .NET Core, що є крос-платформним фреймворком з відкритим вихідним кодом, який можна використовувати для розробки додатків для Windows, Linux і macOS. Це означає, що розробники можуть створювати додатки токенів NFT,

використовуючи C# і .NET Core, і без проблем розгортати їх на будь-якій платформі.

- Окрім сумісності з іншими технологіями, C# також має чудову підтримку API, що полегшує інтеграцію з іншими системами та сервісами. Це особливо важливо для додатків з токенами NFT, оскільки їм часто потрібно взаємодіяти з іншими системами і сервісами на основі блокчейну, такими як смарт-контракти і децентралізовані маркетплейси.
- Ще однією перевагою використання C# для додатків токенів NFT є підтримка спільноти. C# має велику і активну спільноту розробників, які постійно беруть участь у розробці нових бібліотек, фреймворків та інструментів. Це означає, що розробники можуть легко знайти ресурси і підтримку при створенні додатків для токенів NFT за допомогою C#.

Коли справа доходить до нашого технологічного стеку, C# ідеально підходить. Entity Framework Core - популярний фреймворк об'єктно-реляційного відображення (ORM) для .NET Core та C#, який полегшує роботу з базами даних у застосунках, розроблених на C#. MS SQL Server - широко використовувана система управління реляційними базами даних, яка забезпечує високу продуктивність і масштабованість, що робить її ідеальним вибором для зберігання і управління даними токенів NFT.

- Патерни UnitOfWork та Repository також є популярними патернами проектування в C#, які допомагають розробникам писати код, що легко підтримується та тестується. Ці патерни використовуються для розділення проблем доступу до даних і бізнес-логіки, що полегшує написання масштабованих і модульних додатків.
- Паттерн MediatR - це потужний і гнучкий патерн для створення відокремлених, багаторазових і тестованих додатків. Він дозволяє розробникам писати слабко зв'язаний код, полегшуючи зв'язок між різними компонентами програми, що полегшує створення складних додатків.

- Автентифікація JWT - це широко використовуваний механізм для захисту веб-додатків, включаючи додатки з токенами NFT. Він дозволяє розробникам аутентифікувати та авторизувати користувачів і забезпечувати безпечний доступ до ресурсів у розподіленому середовищі.
- Логування NLog - це гнучкий та розширюваний фреймворк логування для .NET Core та C#. Він дозволяє розробникам реєструвати повідомлення та винятки у своїх додатках і надає різноманітні цілі та конфігурації для зберігання та управління даними журналів.
- Swagger - це інструмент з відкритим вихідним кодом для проектування, створення та документування API. Він дозволяє розробникам створювати інтерактивну документацію для своїх API і полегшує їх тестування та налагодження.
- Механізм Cors є невід'ємною частиною будь-якого веб-додатку, включаючи додатки з токенами NFT. Він дозволяє розробникам визначати політику дозволу перехресних запитів і гарантує, що веб-додатки є безпечними і доступними для користувачів.

Бекенд API частина застосунку буде реалізована за принципом Clean Architecture та містити 5 рівнів аплікації:

- Перший дозволяє зберігати сутності, які ORM система зможе конвертувати у відповідні таблички БД.
- Другий зберігатиме міграції БД, при необхідності зміни параметрів однієї із табличок, щоб це можна було зробити безпечно, без втрати даних. Також, тут зберігатимуться Fluent API конфігурації для об’єктів, щоб максимально дотриматись цілісності бази даних.
- Третій зберігатиме сервіси, вручну створені класи-обробники виключних ситуацій та константи необхідні для застосунку у різних її частинах. Такий підхід дозволить легко змінювати значення констант лише в одній частині застосунку, не змінюючи їх по усьому проекту.
- Щодо четвертого – це буде рівень, де будуть зберігатись запити та команди, імплементовані патерном MediatR.

- І останнім, але не менш важливим, буде рівень обробки вхідних HTTP запитів, де відбуватиметься конфігурація усіх сервісів аплікації, JWT аутентифікації та власне контролери для обробки усіх, передбачених системою, запитів користувачів.

Якщо мова йде про фронтенд частину, то тут використовуються React у зв'язці з Redux, що дає змогу зберігати стан, оскільки React сам по собі stateless (той, що не має стану). На допомогу вищеописаним інструментам приходить мова програмування TypeScript, яка дозволяє запровадити строгу типізацію в проекті, що надає можливість уникнути проблем із типами, використовуючи могу програмування JavaScript.

React JS - це бібліотека JavaScript, яка використовується для побудови інтерфейсів користувача. Вона розроблена компанією Facebook і заснована на концепції компонентів, що дозволяє розбити складний інтерфейс на незалежні, повторно використовувані елементи. Основним принципом React є віртуальний DOM (Document Object Model), який забезпечує швидке і ефективне оновлення інтерфейсу.

Ось декілька переваг React JS:

1. Висока продуктивність: React використовує віртуальний DOM, що дозволяє ефективно відстежувати зміни і здійснювати мінімальну кількість оновлень, що призводить до швидкого відображення змін на сторінці.
2. Повторне використання компонентів: завдяки розбиттю інтерфейсу на незалежні компоненти, React сприяє повторному використанню коду. Компоненти можна знову використовувати в різних частинах проекту або навіть у різних проектах, що полегшує розробку та підтримку.
3. Одностороннє потокове управління даними: у React використовується модель одностороннього потоку даних, де дані поширюються вниз по ієрархії компонентів. Це спрощує відлагодження та розуміння взаємодії компонентів.

4. Широке співтовариство та екосистема: React має велике та активне співтовариство розробників, що сприяє наявності багатьох сторонніх бібліотек, інструментів і розширень, що полегшують розробку. Також існує багато документації, онлайн-курсів та ресурсів для вивчення React.
5. Компонентний підхід: React пропонує компонентний підхід до розробки інтерфейсів. Компоненти можна легко створювати, комбінувати і керувати ними. Це дозволяє покращити модульність, читабельність та повторне використання коду.
6. Легкість тестування: Завдяки розділенню інтерфейсу на компоненти, тести React досить прості у написанні. Ви можете легко тестувати кожен компонент окремо, перевіряти його поведінку та впевнитися в правильному функціонуванні.
7. Реактивність: React має можливість реагувати на зміни даних та автоматично оновлювати інтерфейс. Це дозволяє створювати динамічні інтерактивні компоненти, які швидко реагують на дії користувача.
8. Підтримка від Facebook та активний розвиток: React розробляється і підтримується Facebook, що гарантує стабільність, активну підтримку та постійний розвиток цієї технології.
9. Можливість використання з іншими технологіями: React можна поєднувати з різними інструментами та бібліотеками, такими як Redux для керування станом додатка, React Router для навігації, або GraphQL для отримання даних з сервера. Це дає вам гнучкість та можливість використовувати найкращі інструменти для вашого проекту.

Узагальнюючи, React JS є потужним інструментом для розробки інтерфейсів користувача, який забезпечує швидкість, модульність, повторне використання коду та велику спільноту розробників. Ці переваги роблять його популярним веб-технологією для створення сучасних веб-застосунків.

Redux є популярною бібліотекою для керування станом додатка у JavaScript-додатках, особливо в контексті React-проектів. Вона пропонує передбачуваний і односторонній потік даних, що допомагає управляти станом

додатка в послідовний та простий спосіб. Основною концепцією Redux є централізований зберігання стану, який може бути оновлений лише через визначені дії.

Ось декілька переваг Redux:

1. Простота та передбачуваність: Redux пропонує простий та однозначний підхід до керування станом додатка. Весь стан зберігається в одному централізованому об'єкті, що полегшує відлагодження, тестування та розуміння структури додатка.
2. Централізоване зберігання стану: Redux зберігає стан додатка в одному місці, відомому як "store". Це спрощує доступ та оновлення стану, що робить його більш передбачуваним і керованим.
3. Простота тестування: Оскільки Redux дотримується одностороннього потоку даних і чітко визначає, як дії змінюють стан, тести на Redux досить прості у написанні. Ви можете легко тестувати редюсери (reducers), які змінюють стан, та перевіряти їхню коректність.
4. Швидкість та ефективність: Redux використовує незмінність даних та механізм порівняння значень, що дозволяє ефективно виявляти зміни та оновлювати інтерфейс. Це особливо важливо для великих та складних додатків зі значною кількістю стану.
5. Розширюваність: Redux дозволяє легко додавати нові функціональності до додатка за допомогою middleware. Middleware - це функції, які можуть перехоплювати та змінювати дії перед тим, як вони досягнуть редюсерів. Це дозволяє вам додавати логування, асинхронні запити, маршрутизацію та інші функції до вашого додатка зручним способом.
6. Розширена спільнота та екосистема: Redux має велику та активну спільноту розробників. Тобто існує безліч сторонніх бібліотек, пакетів та інструментів, які розширюють можливості Redux. Також існує багато ресурсів, документації та практичних прикладів, які допомагають вивчити та використовувати Redux ефективно.

7. Сумісність з React та іншими фреймворками: Redux часто використовується в поєднанні з React, але він також може працювати з іншими фреймворками або бібліотеками JavaScript. Це дозволяє використовувати Redux в різних проектах та зберігати одну й ту ж менталітет управління станом.
8. Чудова інструментарій для відлагодження: Redux надає інструменти для відстеження та аналізу змін стану додатка. Зокрема, ви можете використовувати розширення Redux DevTools для браузера, щоб відстежувати дії, стан та відновлювати попередні стани.

Узагальнюючи, Redux є потужною технологією для керування станом додатка, яка пропонує передбачуваність, простоту тестування, швидкість та гнучкість. Вона ідеально поєднується з React та має широку спільноту, що дозволяє вам будувати масштабовані підтримувати складні додатки з легкістю. Використання Redux дозволяє зробити ваш код більш структурованим, легко керованим та розширюваним. Незалежно від розміру вашого проекту, Redux надає потужні інструменти для керування станом, спрощуючи розробку та підтримку додатків.

TypeScript - це мова програмування, яка є суперсетом мови JavaScript, що додає статичну типізацію та деякі інші функціональні можливості до JavaScript.

Переваги використання TypeScript:

1. Статична типізація: однією з найбільших переваг TypeScript є можливість використовувати статичну типізацію. Ви можете визначати типи для змінних, параметрів функцій, повернених значень і т.д. Це дозволяє виявляти помилки в процесі компіляції і полегшує розуміння коду для розробників та інструментів розробки.
2. Підтримка сучасних функціональних можливостей: TypeScript підтримує багато нових функцій, які були додані до стандарту ECMAScript, таких як стрілкові функції, генератори, асинхронні функції та інші. Це дозволяє

вам використовувати сучасні практики програмування та зручності, які пропонуються JavaScript.

3. Підтримка класів та об'єктно-орієнтованого програмування: TypeScript має вбудовану підтримку класів, успадкування, інтерфейсів та інших об'єктно-орієнтованих парадигм. Це полегшує розробку складних програм з багатьма класами та спадковістю.
4. Інструментарій розробки: TypeScript надає потужний інструментарій для розробки, включаючи розширення для популярних редакторів коду, таких як Visual Studio Code, які надають підказки про типи, перевірку помилок під час набору коду та автодоповнення. Це полегшує розробку та допомагає уникнути багатьох типових помилок.
5. Підтримка нових JavaScript-стандартів: TypeScript підтримує останні стандарти ECMAScript (стандарт, на якому базується JavaScript) і швидко оновлюється, щоб включити нові функціональні можливості та синтаксис. Це означає, що ви можете використовувати сучасні функції та синтаксис JavaScript, навіть якщо ваш проект компілюється до старіших версій JavaScript.
6. Підвищена безпека та надійність: завдяки статичній типізації TypeScript допомагає виявляти помилки на етапі компіляції, що дозволяє уникнути багатьох типових помилок, які можуть виникати під час виконання коду JavaScript. Це поліпшує безпеку та надійність вашого коду.
7. Інтеграція з існуючим JavaScript-кодом: TypeScript може бути поступово впроваджений в проект, який вже використовує JavaScript. Ви можете почати писати новий код на TypeScript і поступово конвертувати існуючий JavaScript-код в TypeScript. Це дозволяє вам зручно використовувати TypeScript без необхідності переписувати весь код.
8. Велика спільнота та екосистема: TypeScript має широку та активну спільноту розробників, що сприяє розширенню екосистеми з пакетами, бібліотеками та інструментами для розробки. Це означає, що ви можете

знайти рішення для багатьох проблем, знайшовши підходящі пакети або отримавши підтримку від спільноти.

Підсумовуючи вищеописані переваги, TypeScript додає статичну типізацію та багато інших корисних функцій до JavaScript, що поліпшує безпеку, розробку та підтримку проектів. Він підходить для розробки як малих, так і великих проектів, допомагаючи розробникам створювати більш надійний та швидкий код.

РОЗДІЛ 2. ПРОГРАМНА РЕАЛІЗАЦІЯ ЗАСТОСУНКУ

2.1 Архітектура додатку

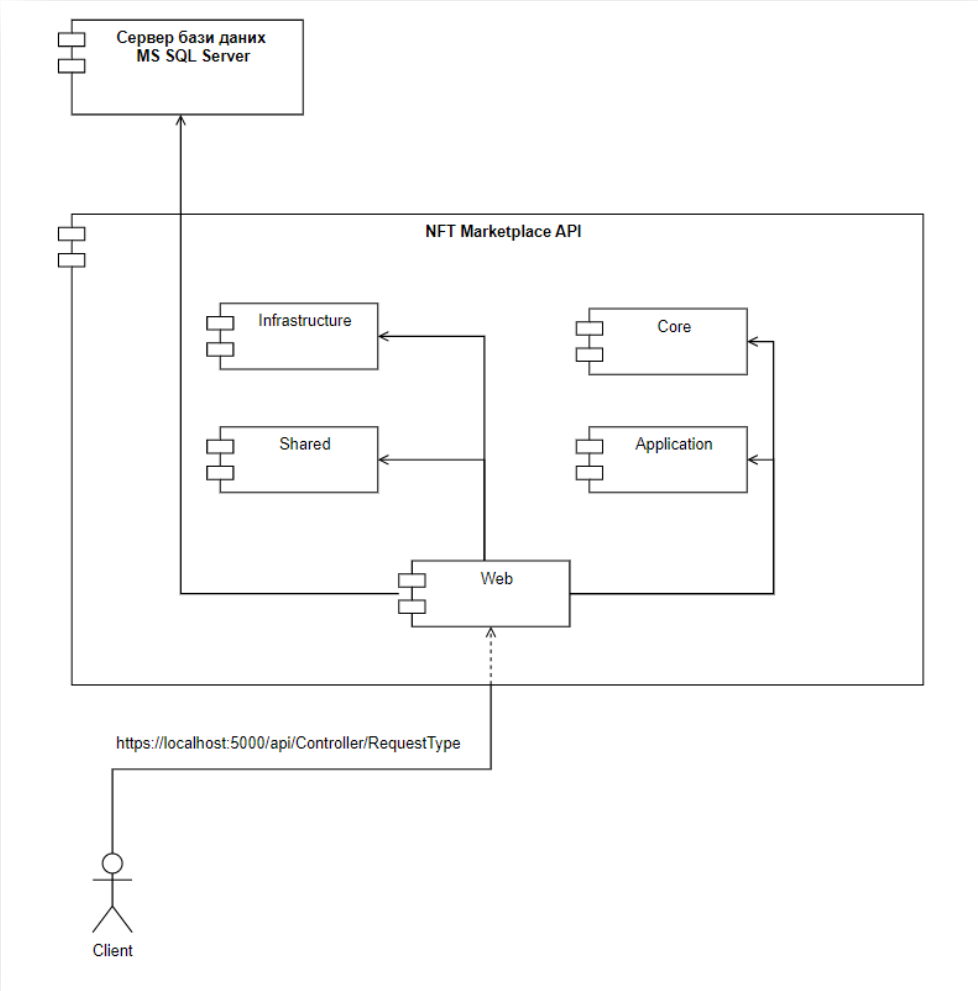

Рис. 2.1. Архітектура системи

На діаграмі зображена схема бібліотек, які використовуються застосунком та сам рівень Web, який приймає, обробляє, та відправляє HTTP запити.

2.2 Огляд програмної частини

Розглянемо конфігурацію додатку, та як вона налаштована.

Створюємо «будівельника» застосунку

```
var builder = WebApplication.CreateBuilder(args);
```

NLog: очищуємо існуючи типи провайдерів для логування

```
builder.Logging.ClearProviders();
```

NLog: налаштовуємо Nlog логування

```
builder.Host.UseNLog();
```

Додаємо контекст БД, клас, через який виконується керування усіма сутностями

```
builder.Services.AddDbContext<ApplicationDbContext>(option
s =>
```

Через опції вказуємо, що використаємо строку підключення до БД, під тегом Default

```
options.UseSqlServer(builder.Configuration.GetConnectionSt
ring("Default")));
```

Клас-контролер для обробки запитів щодо аутентифікації користувача:

Декларація класу

```
public class AccountController : BaseController
{
```

Створення приватного поля для збереження посилання на клас-об’єкт Mediator

```
private readonly IMediator mediator;
```

Конструктор класу, де за допомогою принципу інверсії ми отримуємо ініціалізований об’єкт типу Mediator та зберігаємо посилання

```
public AccountController(IMediator mediator)
{
    this.mediator = mediator;
}
```

Метод-обробник HTTP запиту реєстрації користувача

Вказуємо HTTP заголовок Post, та відносне посилання register

```
[HttpPost("register")]
```

Публічний асинхронний метод, що приймає команду, та за допомогою класу mediator асинхронно обробляє команду

```
public async Task<ActionResult<AuthResponse>>
RegisterAsync([FromBody] RegisterUserCommand command)
{
    return await mediator.Send(command);
}
```

Подивимось, як це працює всередині. Отож, детальний опис контракту-команди для обробки запиту користувача:

Контракт, що містить параметри, які повинен передати користувач для аутентифікації, а саме логін та пароль

```
public record RegisterUserCommand(string UserName, string
Password) : IRequest<AuthResponse>;
```

Обробник запиту реєстрації користувача

```
public class RegisterUserHandler :
IRequestHandler<RegisterUserCommand, AuthResponse>
{
    private readonly UserManager<UserEntity> userManager;
    private readonly JwtService jwtService;
    private readonly IMapper mapper;

    public RegisterUserHandler(UserManager<UserEntity>
userManager, JwtService jwtService, IMapper mapper)
    {
        this.userManager = userManager;
        this.jwtService = jwtService;
        this.mapper = mapper;
    }
```

```
public async Task<AuthResponse> Handle(RegisterUserCommand
request, CancellationToken cancellationToken)
{
```

Отримуємо сутність, використовуючи перетворення за допомогою бібліотеки AutoMapper

```
        var user = mapper.Map<UserEntity>(request);
```

Асинхронно створюємо користувача

```
        var result = await userManager.CreateAsync(user,
request.Password);
```

Якщо результат неуспішний, можливо користувач існує, чи є інші причини

```
        if (!result.Succeeded)
        {
```

Генеруємо відповідну виключну ситуацію з повідомленням

```
            throw new BadRequestRestException("Invalid
credentials", result.Errors);
        }
```

В інакшому випадку, додаємо користувачу роль

```
await userManager.AddToRoleAsync(user,
AppConstant.Roles.User);
```

Та створюємо токен користувача, за допомогою якого його // буде аутентифіковано на сервісі

```
        var token = jwtService.GenerateToken(
            user.Id.ToString(),
            string.Join(", ", await
userManager.GetRolesAsync(user)),
            AppConstant.JwtTokenLifetimes.Default);
```

Повертаємо HTTP відповідь, а саме:

```
return new AuthResponse()
{
```

Токен

```
Token = token,
```

і дані новоствореного користувача

```
User = mapper.Map<UserDto>(user) }
```

РОЗДІЛ 3. ТЕСТУВАННЯ ТА ДОСЛІДЖЕННЯ РОЗРОБЛЕНОЇ ПЛАТФОРМИ

Після запуску застосунку нас зустрічає сторінка для тестування API, зображена на рис. 3.1:

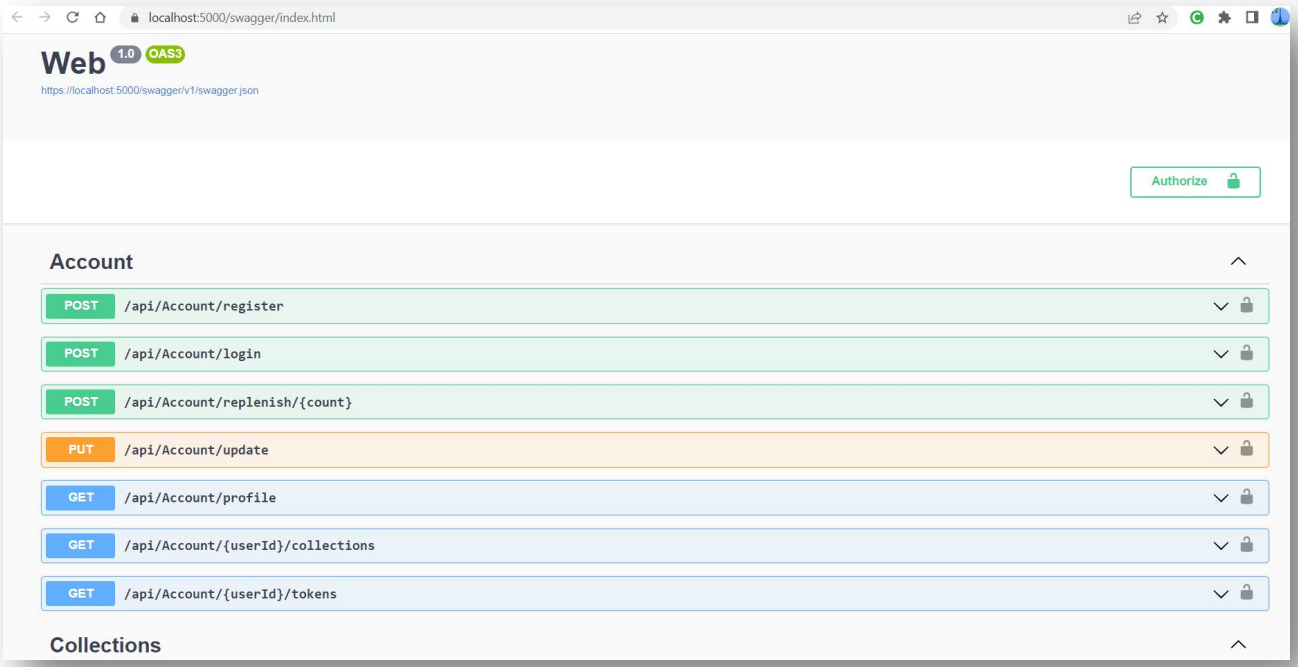

Рис. 3.1. Веб-сторінка для тестування API

Також, під час запуску відкривається консоль, що супроводжується логами та відповідними повідомлення під час обробки HTTP запитів.

```
2023-03-19 23:14:44.8860|INFO|Microsoft.Hosting.Lifetime|Now listening on: https://localhost:5000
2023-03-19 23:14:44.8860|INFO|Microsoft.Hosting.Lifetime|Now listening on: http://localhost:5001
2023-03-19 23:14:44.8882|INFO|Microsoft.Hosting.Lifetime|Application started. Press Ctrl+C to shut down.
2023-03-19 23:14:44.8882|INFO|Microsoft.Hosting.Lifetime|Hosting environment: Development
```

Рис. 3.2. Приклад логування інформації, де відображено домен та порт, на якому працює застосунок

Протестуймо декілька кінцевих точок. Для початку зареєструємо користувача з логіном nuwm та паролем Qwerty_1, як це зображено на рис. 3.3.

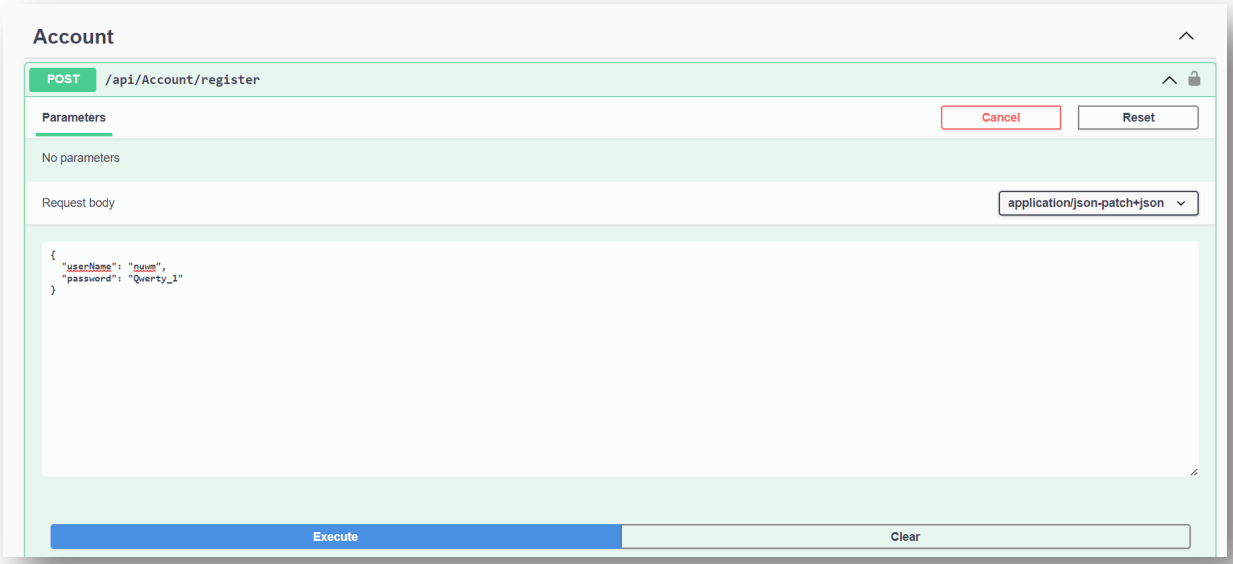

Рис. 3.3. Заповнення даних в json форматі та відправка команди на сервер

У разі успішної операції, ми отримаємо HTTP статус-код 200, та відповідний об’єкт з токеном аутентифікації, як це зображено на рис. 3.4.

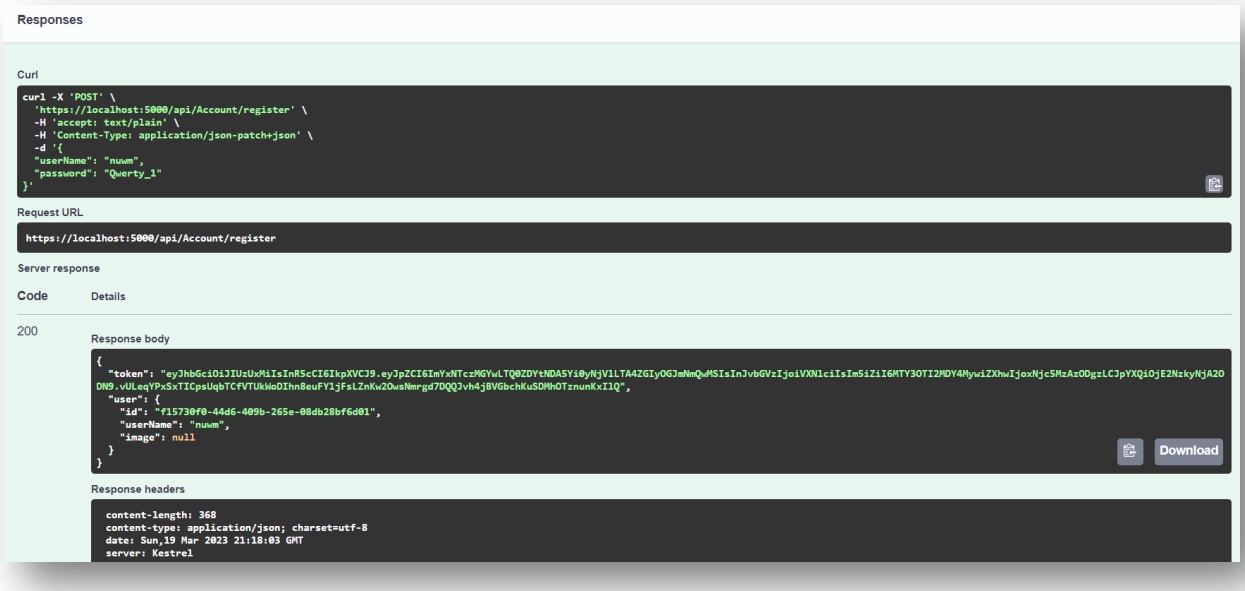

Рис. 3.4. Дані аутентифікованого користувача

У разі помилки під час обробки запиту, ми отримаємо 400 статус-код та відповідне повідомлення, зображене на рис. 3.5. Як бачимо, я намагався зареєструвати користувача, який вже був щойно створений, та отримав помилку.

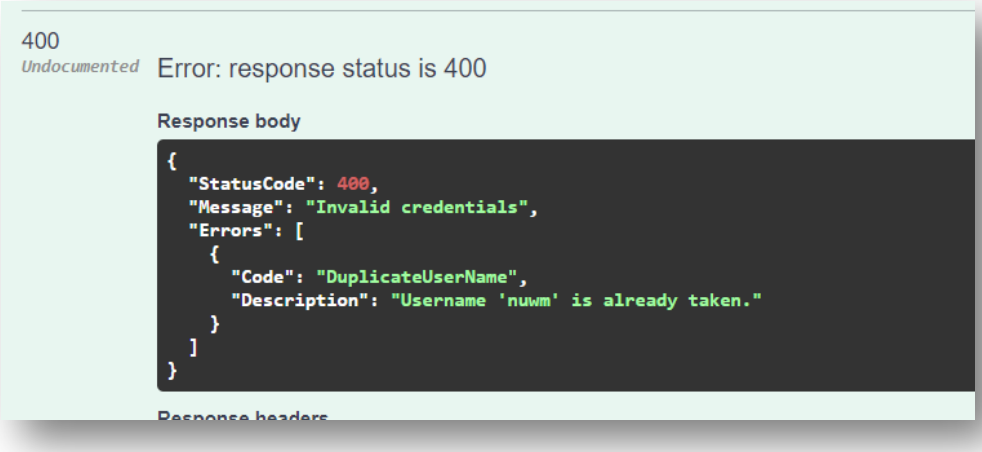

Рис. 3.5. Помилка під час реєстрації вже існуючого користувача

Наступним кроком спробуймо аутентифікуватись у системі за допомогою щойно створеного користувача, використовуючи метод Account/login

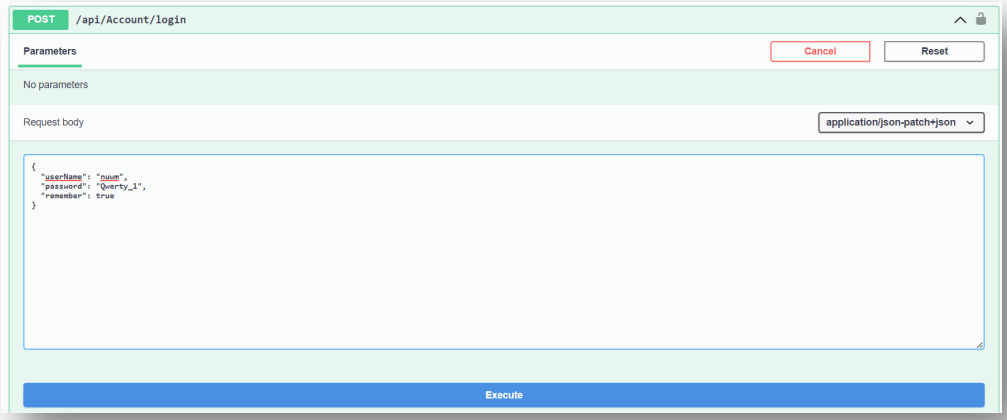

Рис. 3.6. Аутентифікація створеного користувача

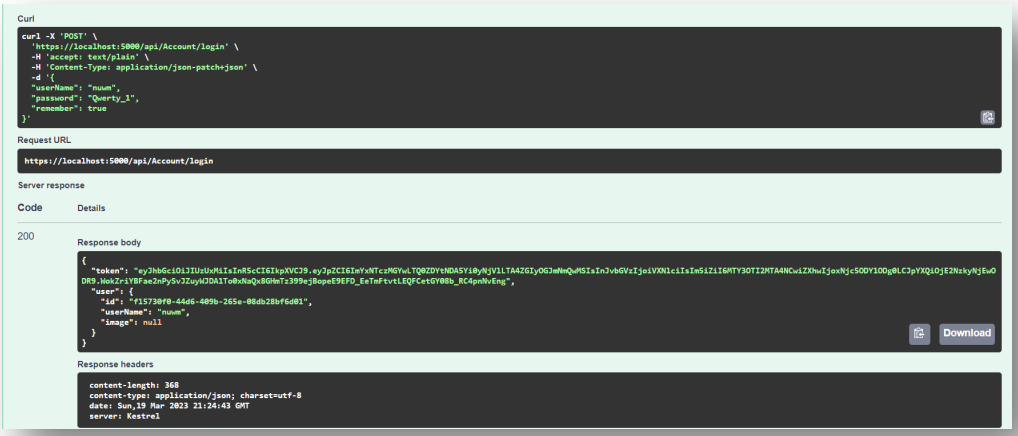

Рис. 3.7. Повідомлення про успішну аутентифікацію

На рис. 3.7. зображено повідомлення, що користувач був успішно аутентифікований та ми отримали токен. Спробуймо скопіювати цей токен та протестувати інші методи контролерів.

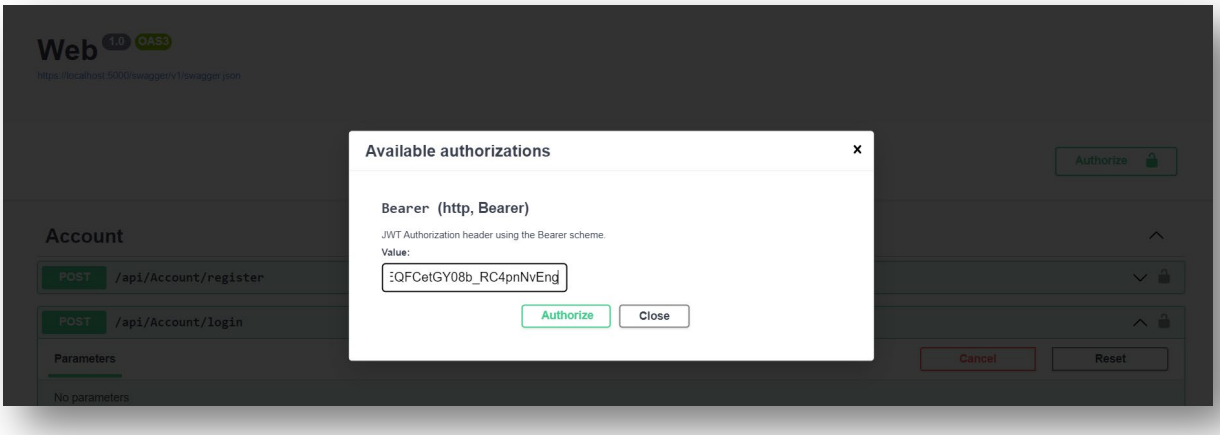

Рис. 3.8. Аутентифікація за допомогою створеного токена

Лише аутентифікований користувач може створювати нові колекції. Спробуймо створити нову колекцію з назвою test:

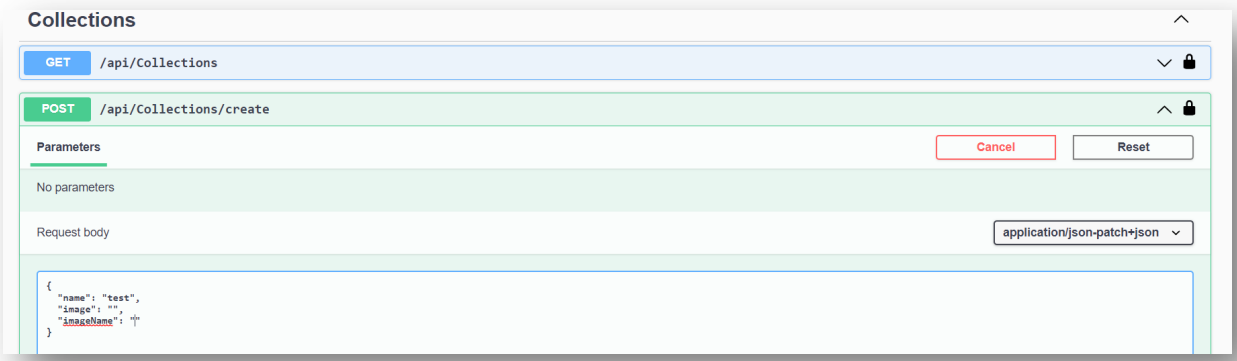

Рис. 3.9. Створення колекції

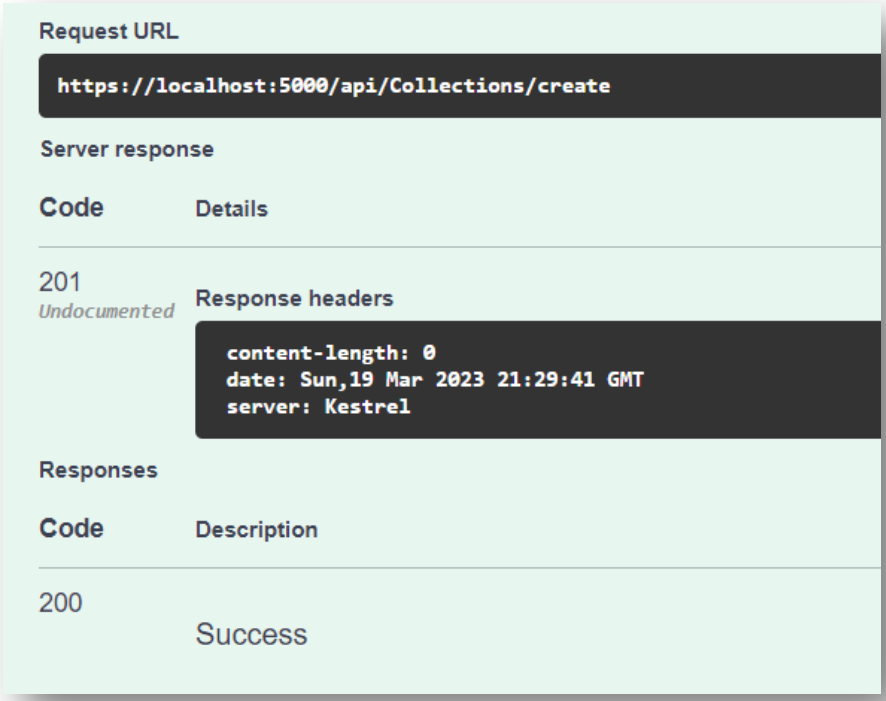

Рис. 3.10. Повідомлення про успішну операцію

Використовуючи метод Get на контролері Collections ми можемо впевнитись, що операція пройшла успішно та отримати усі колекції:

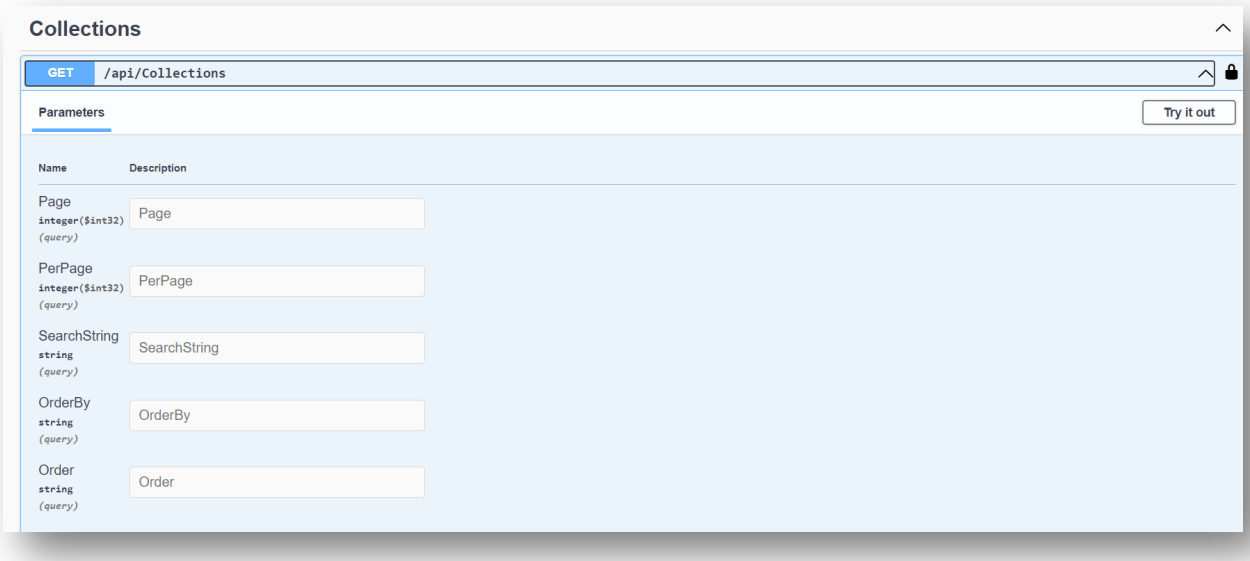

Рис. 3.11. Виклик методу для отримання усіх колекцій

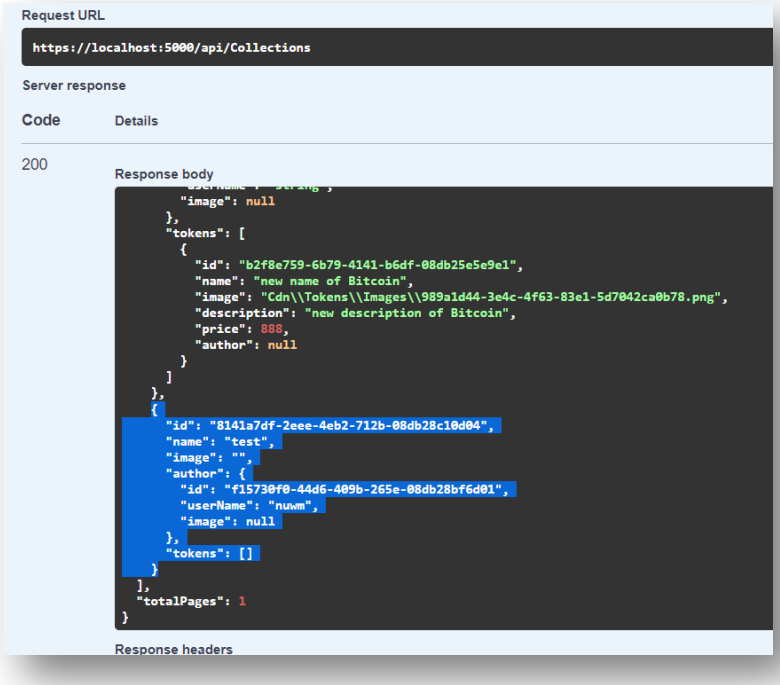

Рис. 3.12. Отримані колекції

Як бачимо, список із колекцій містить щойно додану нову колекцію.

Спробуймо видалити нашу колекцію, як це зображено на рис. 3.13:

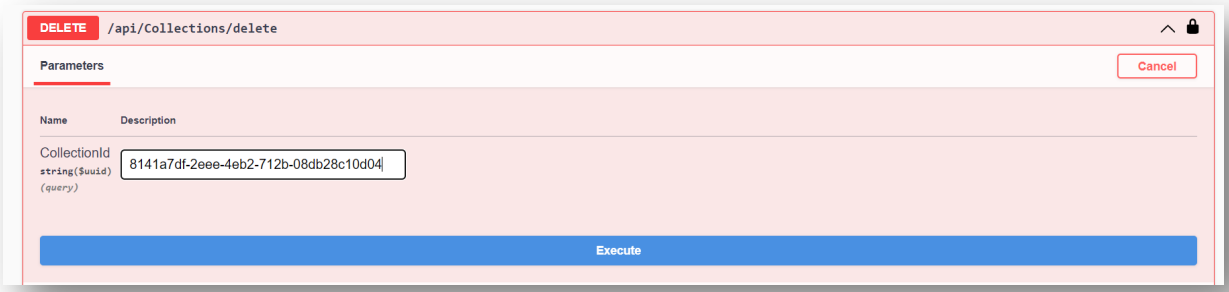

Рис. 3.13. Надсилання запиту на видалення колекції

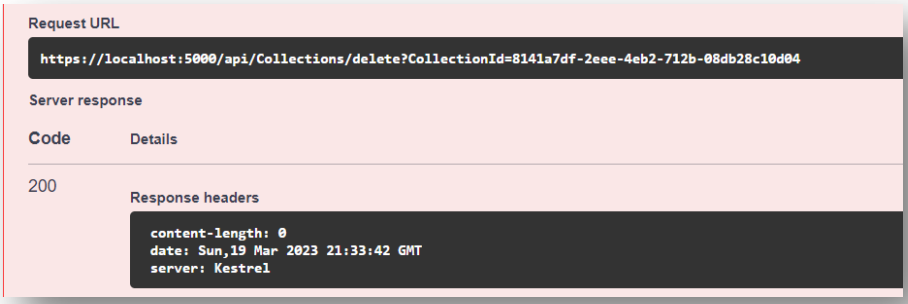

Рис. 3.14. Отримання повідомлення про успішне видалення

Наступним кроком протестуємо клієнт-частинку застосунку. Після запуску нас зустрічає домашня сторінка (рис. 3.15.) із трьома найбільш популярними токенами.

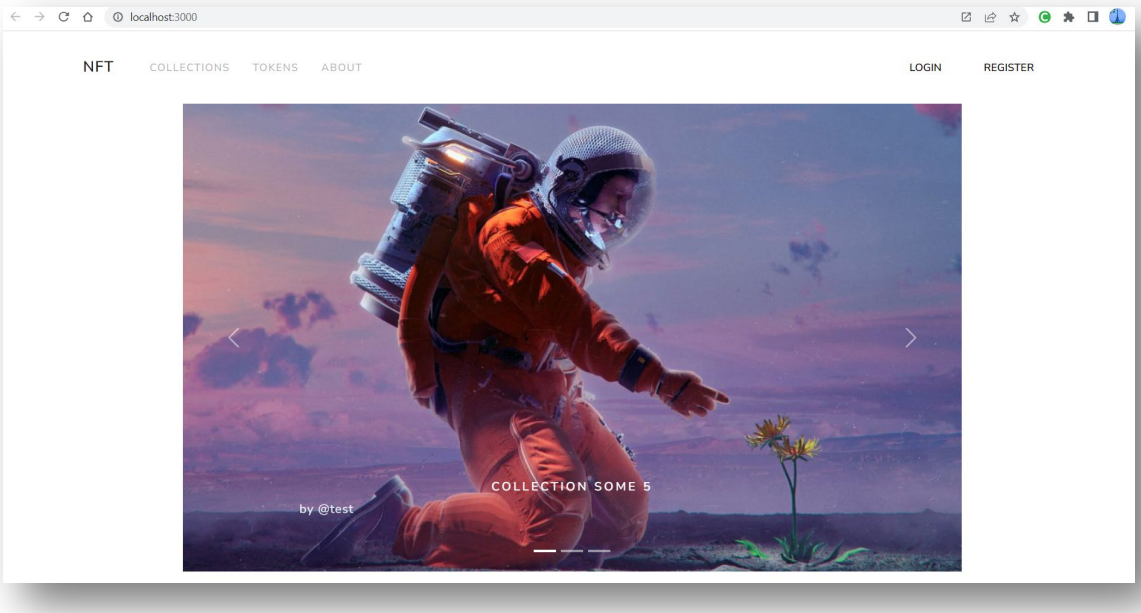

Рис. 3.15. Домашня сторінка застосунку

Проскроливши донизу, можемо побачити футер (рис. 3.16), де розташовані корисні посилання для користувача:

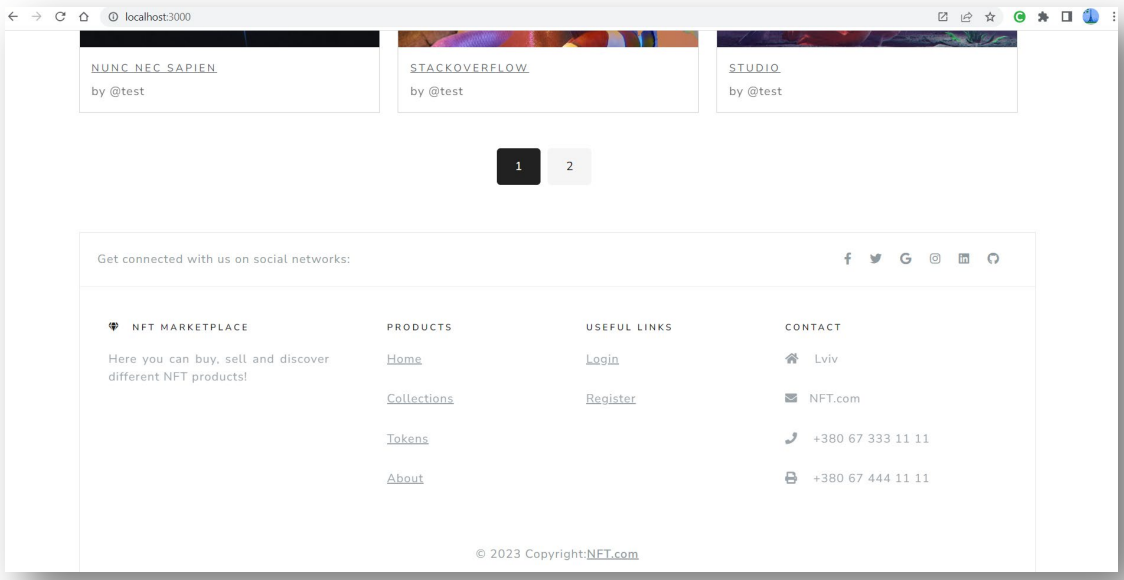

Рис. 3.16. Футер із корисними посиланнями

Перейдемо на сторінку колекцій (рис. 3.17), клікнувши на посилання Collections.

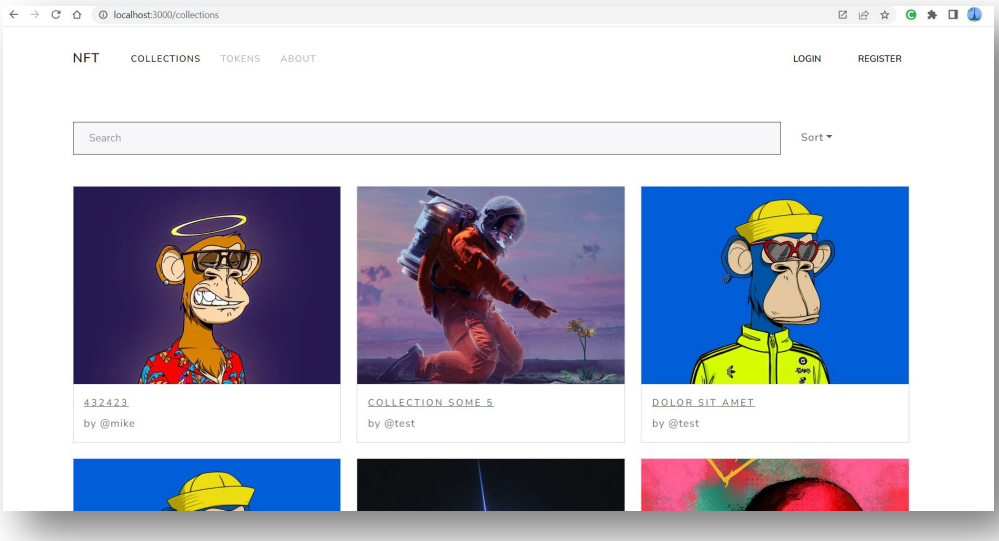

Рис. 3.17. Сторінка колекцій із токенами

Можемо переглянути більш детальну інформацію (рис. 3.18), для цього клікнемо на назву будь-якої із колекцій.

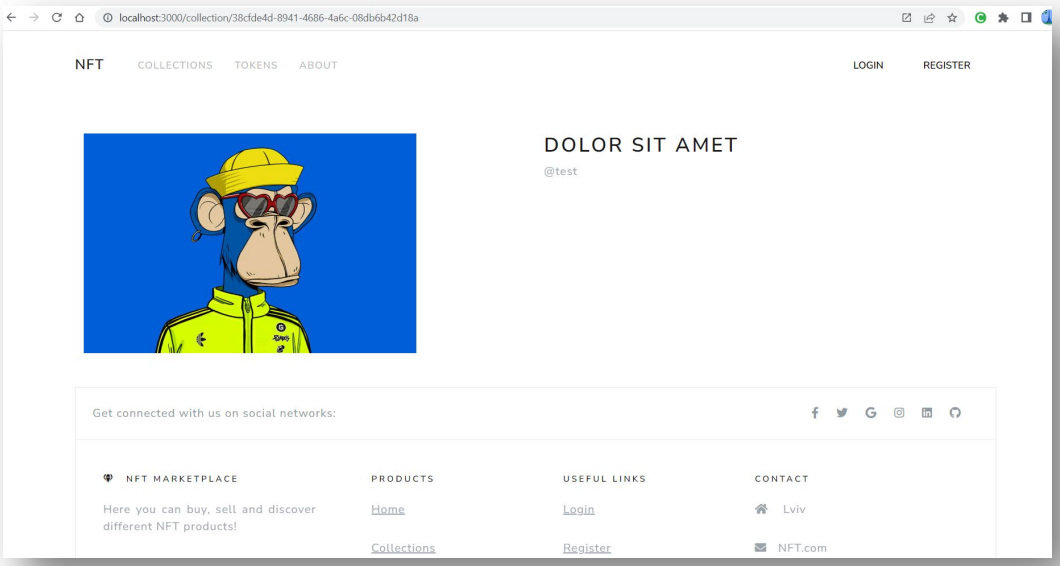

Рис. 3.18. Інформація про колекцію

Перейдемо на сторінку токенів (рис. 3.19), попередньо клікнувши на кнопку Tokens зліва зверху сторінки.

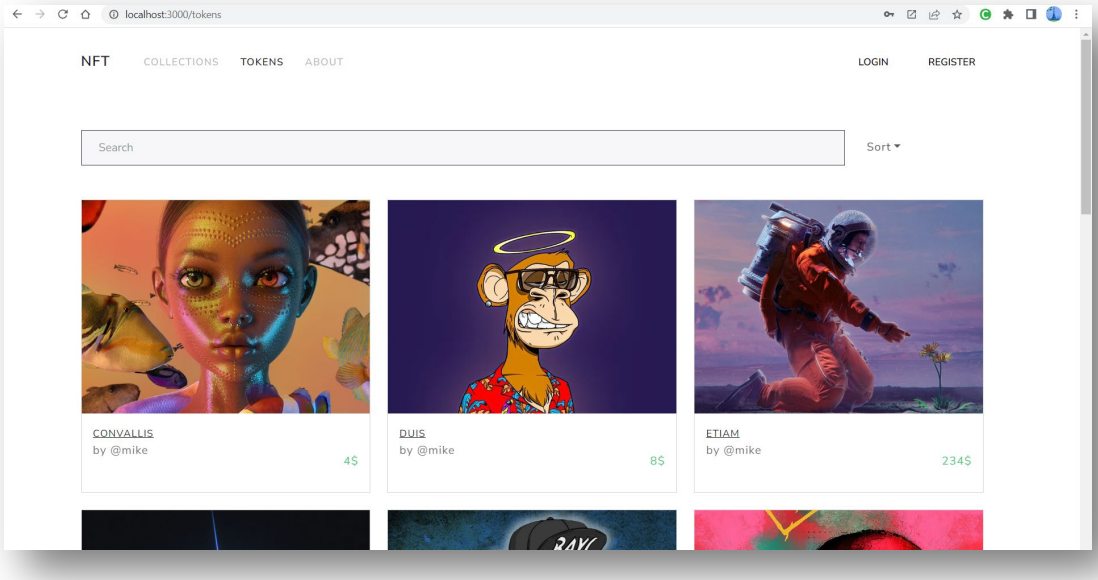

Рис. 3.19. Сторінка, що відображає токени

Переглянемо більш детальну інформацію по токену (рис. 3.20), клікнувши на його назву.

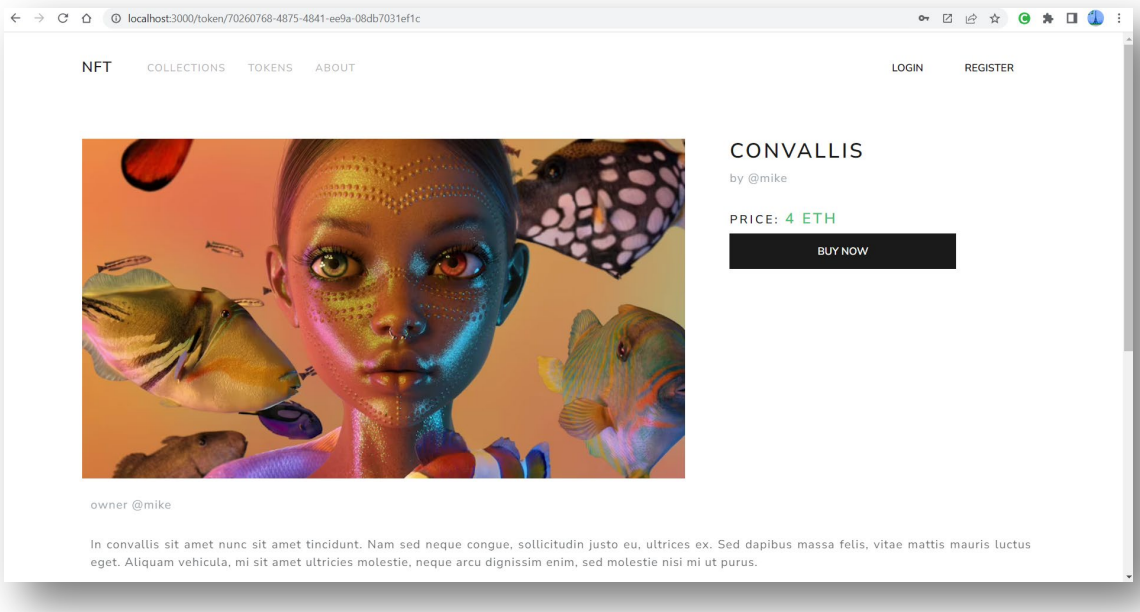

Рис. 3.20. Сторінка токена

Як ми бачимо, тут вказано ціну, автора а також є можливість побачити опис до токену та купити його. Спробуймо натиснути кнопку Buy Now (рис. 3.21).

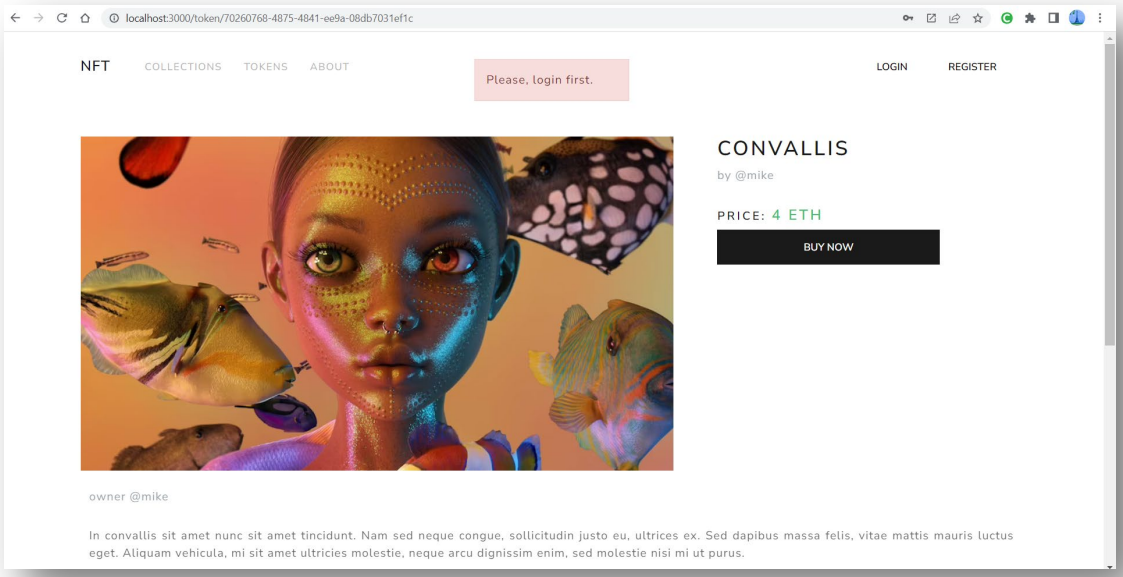

Рис. 3.21. Повідомлення про помилку

Оскільки користувач не автентифікований, ми отримали помилку про те, що потрібно спочатку автентифікуватись у системі. У нас є можливість зареєструватись, якщо користувача немає, або ж увійти в систему, якщо

користувач вже створений. Для цього є дві відповідні кнопки Login та Register справа зверху. Спробуймо зареєструватись, як це зображено на рис. 3.22.

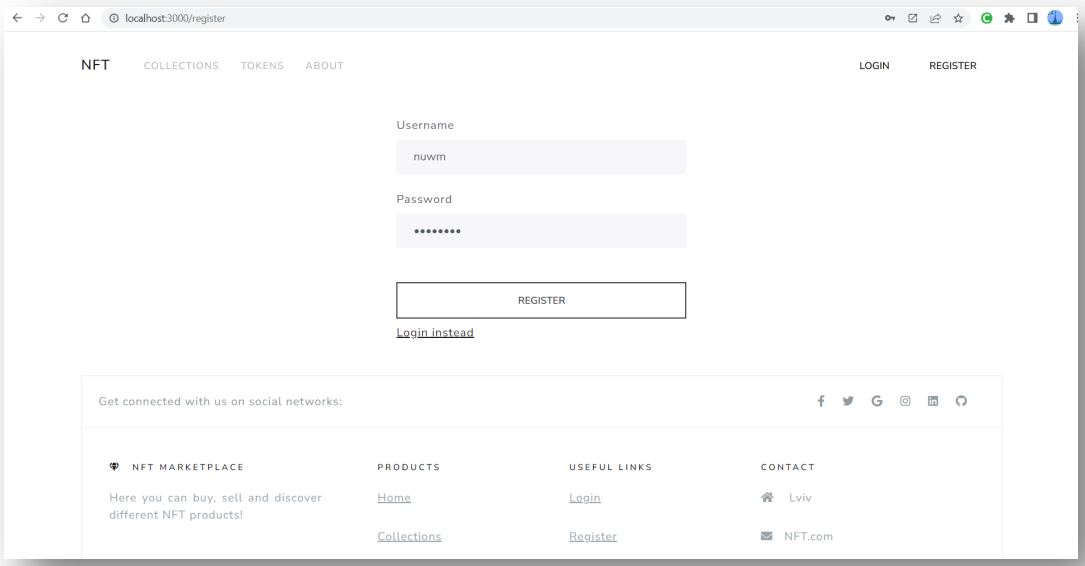

Рис. 3.22. Сторінка реєстрації користувача

Для цього необхідно ввести ім’я користувача, яке унікальне, та пароль, після цього натиснути кнопку Register.

Після того, як користувач автентифікований, справа зверху (рис. 3.23) з’явиться можливість створювати колекції та токени, а також, переглядати (рис. 3.24) чи модифікувати (рис. 3.25) профіль чи поповнити рахунок (рис. 3.26).

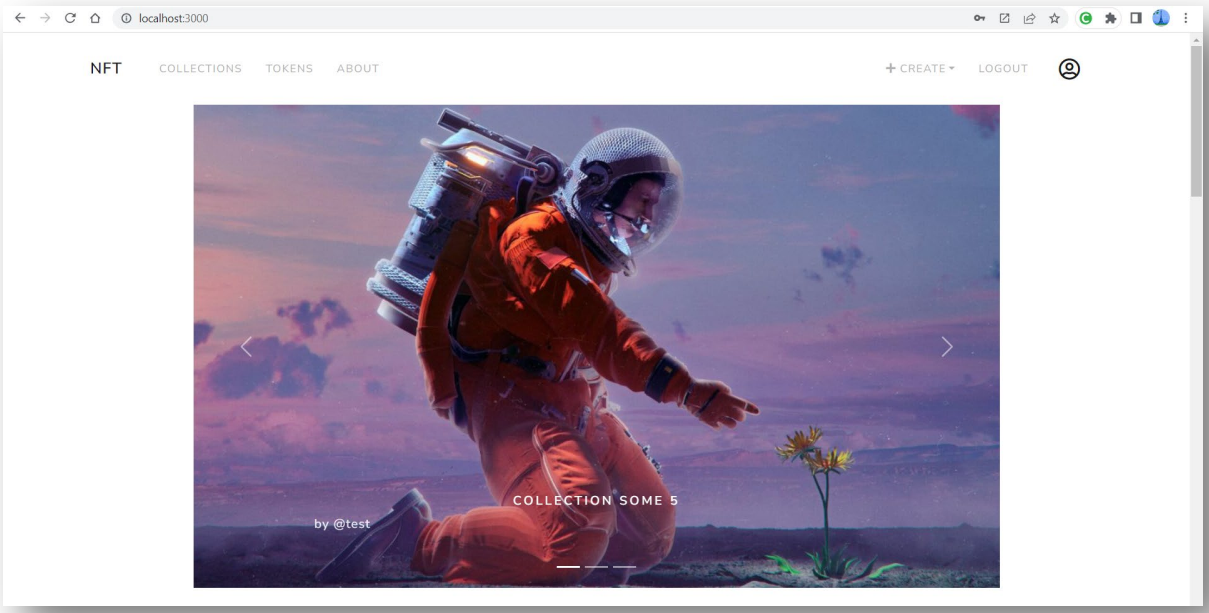

Рис. 3.23. Домашня сторінка з можливістю створювати колекції та токени

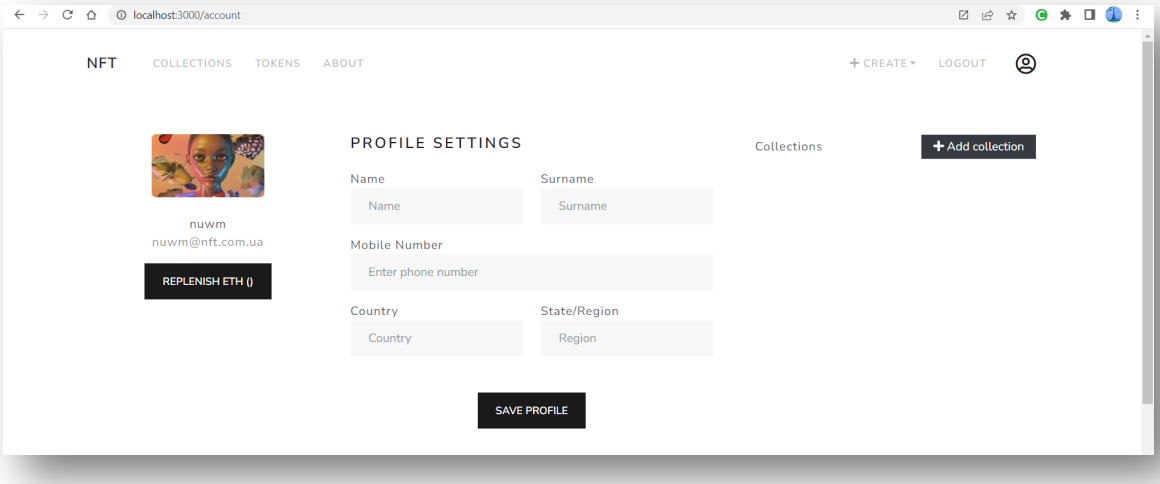

Рис. 3.24. Перший перегляд профілю

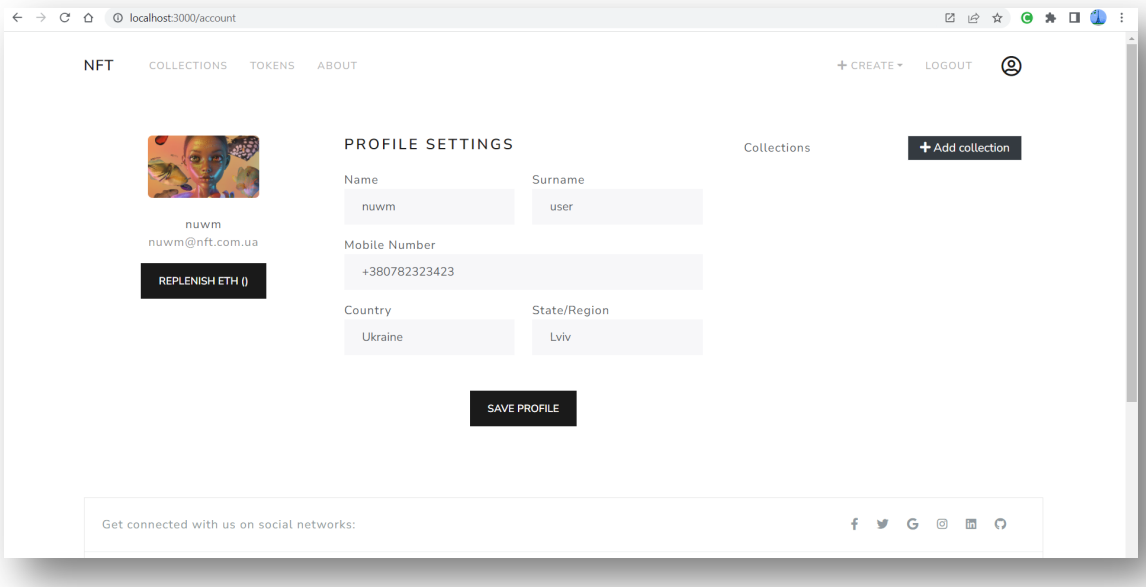

Рис. 3.25. Модифікація профілю.

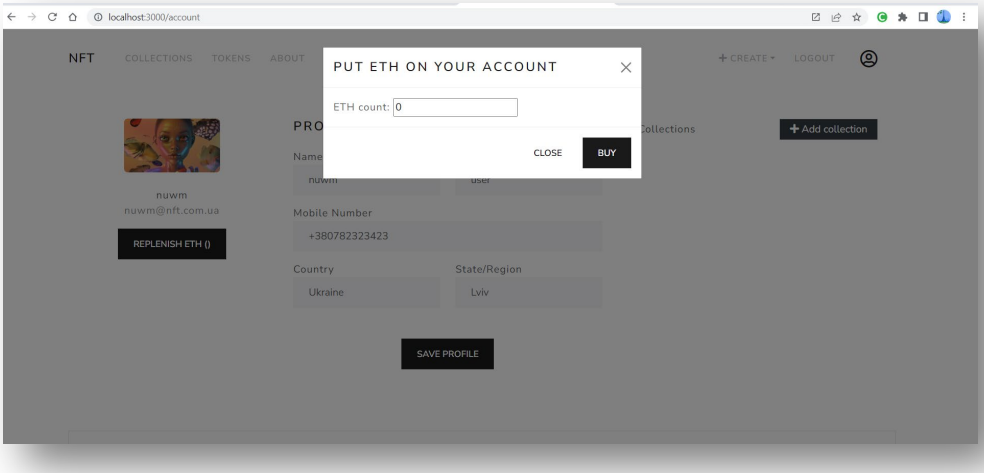

Рис. 3.26. Поповнення рахунку

Після поповнення рахунку у нас з’явилась можливість купувати токени. Перейдімо на сторінку токенів та знайдемо токен Convallis. Для цього в пошуковій стрічці введемо його назву (рис. 3.27).

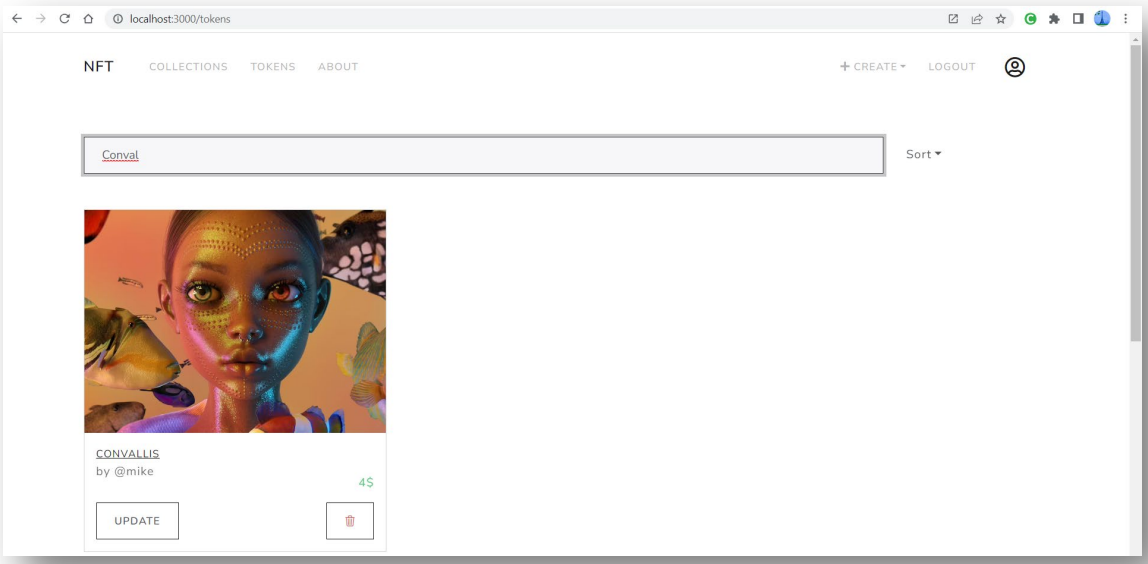

Рис. 3.27. Пошук токену.

Для покупки натиснемо на назву токену, та, перейшовши на його сторінку, натиснемо Buy Now (рис. 3.28). Звернімо увагу, що під картинкою owner вказаний як mike.

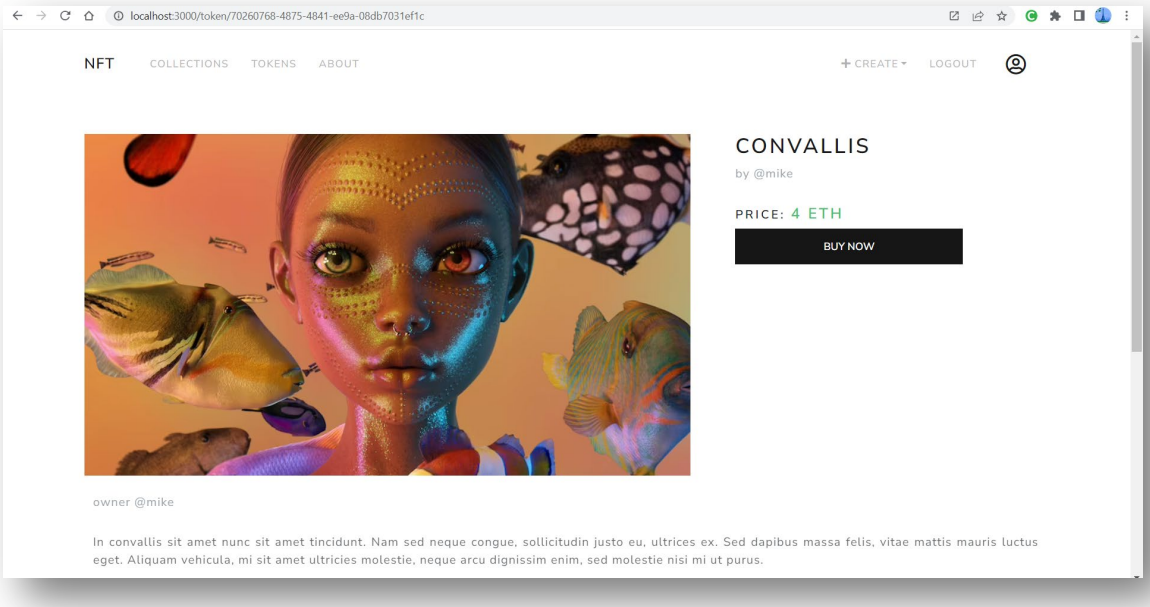

Рис. 3.28. Покупка токену.

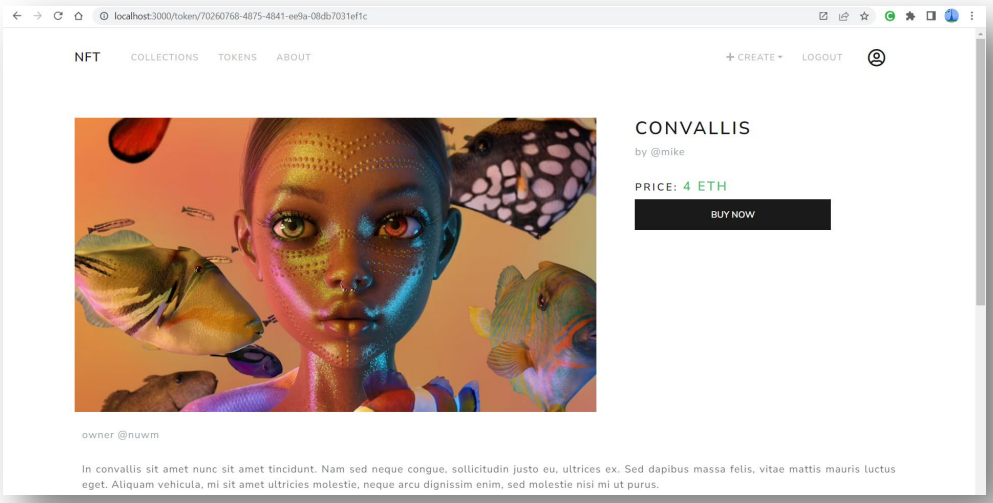

Рис. 3.29. Куплений токен.

Як бачимо на рис. 3.29, owner змінився на нашого користувача, що означає успішну покупку токену.

Наступним кроком спробуймо створити власну колекцію із токенами (рис. 3.30).

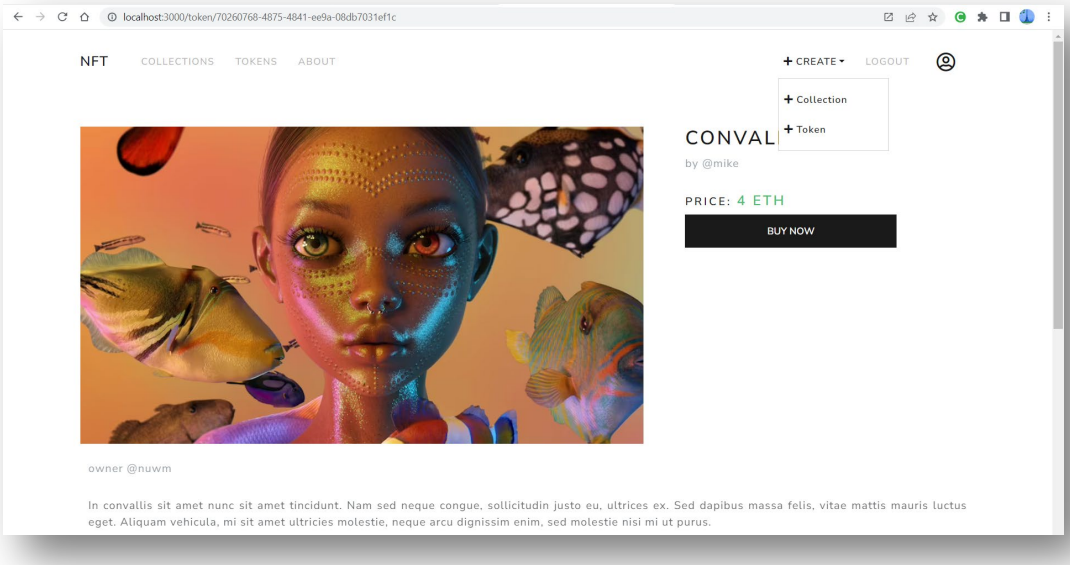

Рис. 3.30. Натискаємо кнопку + Create та у випадаючому списку вибираємо Collection

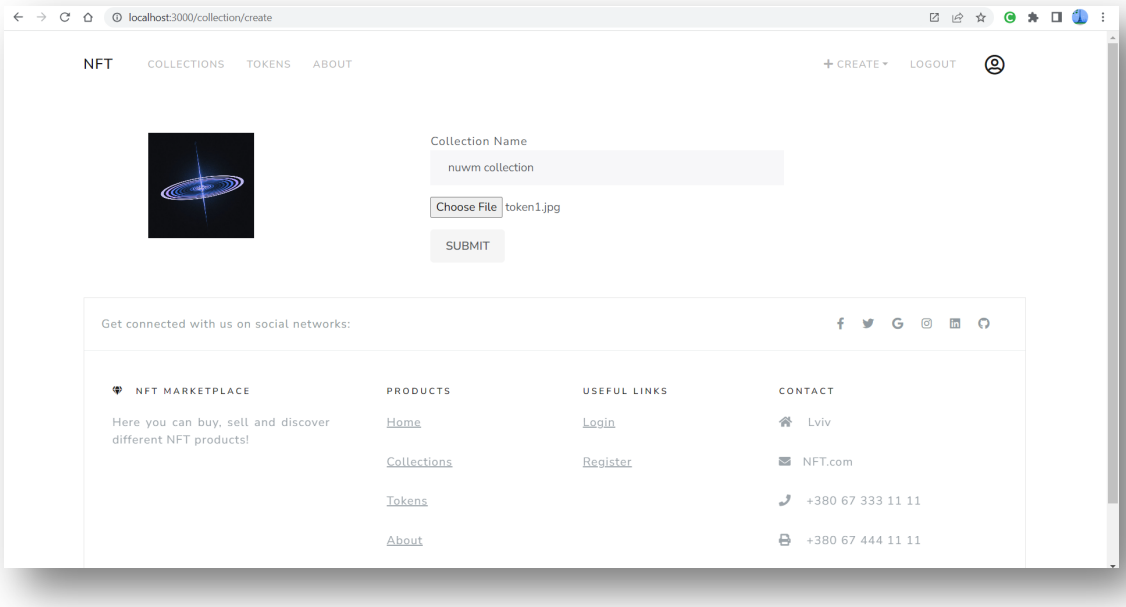

Рис. 3.31. Створення колекції.

Для створення колекції необхідно заповнити усі поля (рис. 3.31) та натиснути submit. Після цього перейдімо на сторінку колекції та відсортуємо найновіші по даті токени (рис. 3.32).

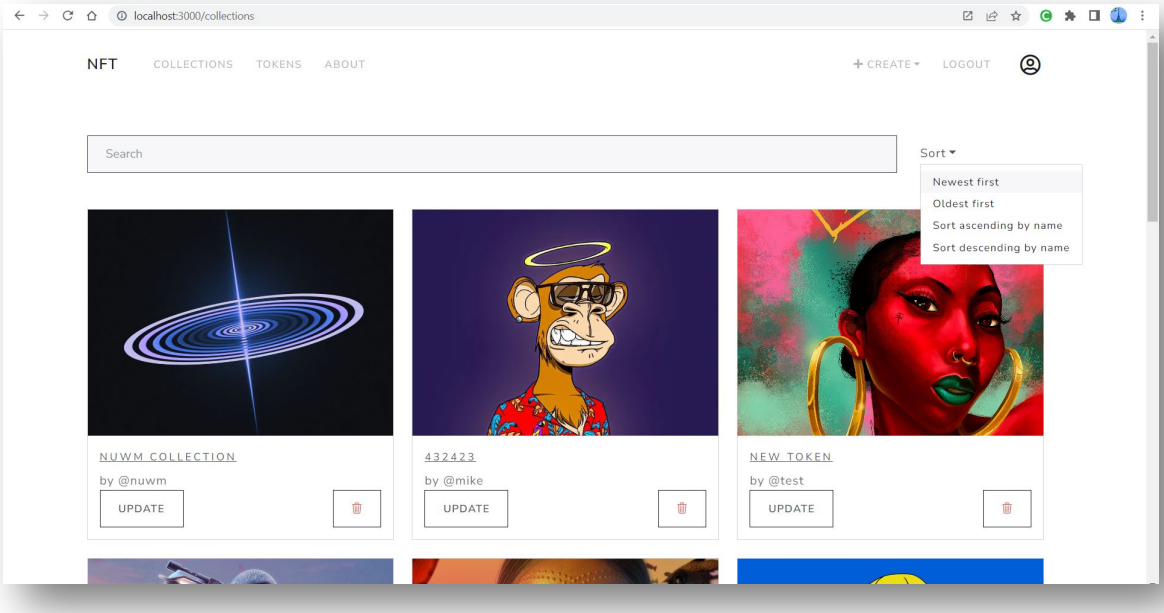

Рис. 3.32. Сортування найновіших токенів

Як бачимо, наш токен перший у списку. Спробуймо створити токен схожим чином та, наприклад, не виберемо картинку, як це зображено на рис. 3.33.

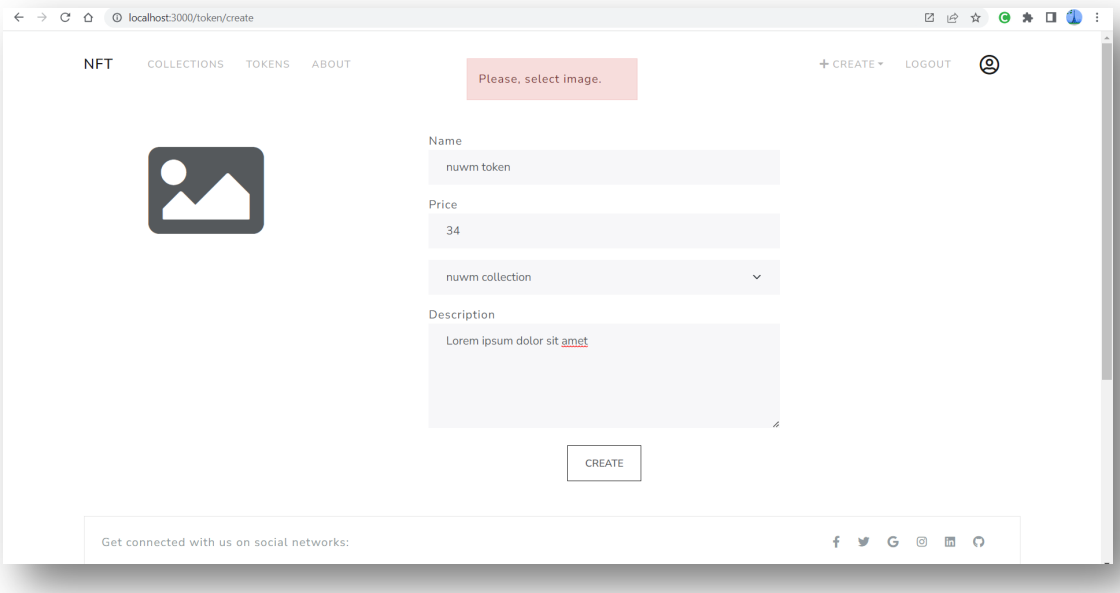

Рис. 3.33. Помилка про відсутність картинки

Виберемо картинку та знайдемо токен, як це зображено на рис. 3.34.

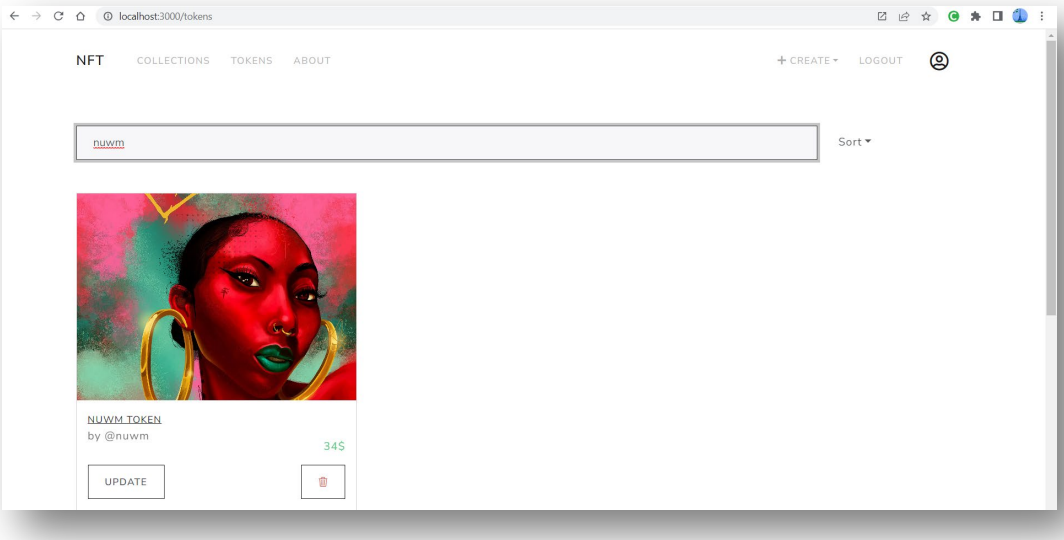

Рис. 3.34. Пошук новоствореного токену

Останнім, але не менш важливим компонентом, є сторінка про маркетплейс, зображена на рис. 3.35.

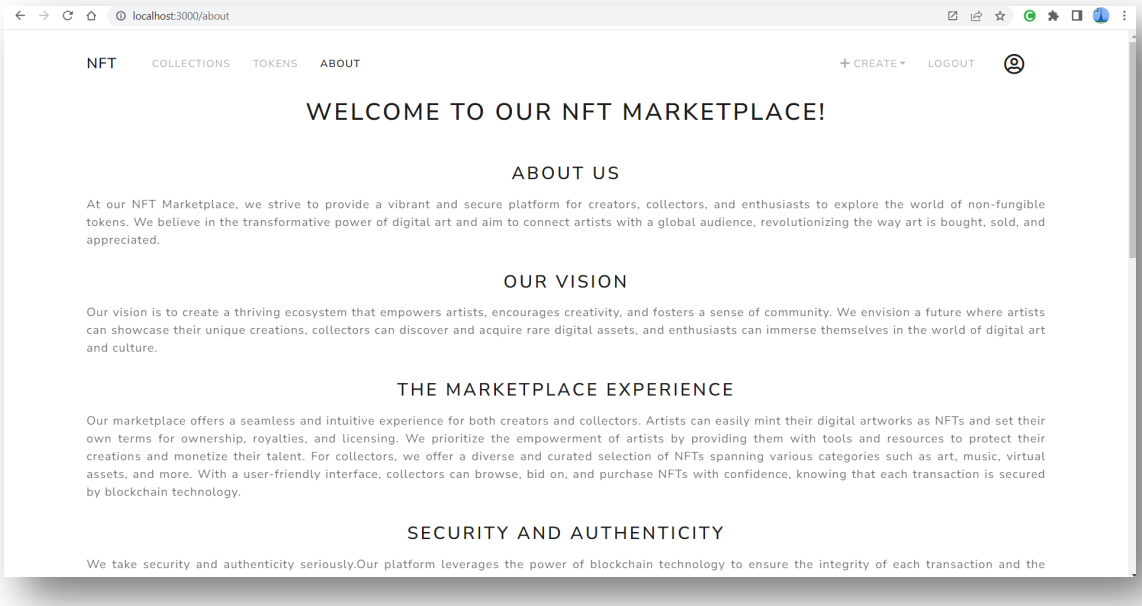

Рис. 3.35. Сторінка, що описує маркетплейс та його цінності

ВИСНОВКИ

Реалізація застосунку для маркетплейсу токенів NFT передбачала поєднання різних технологій та навичок, включаючи Entity Framework Core, MS SQL Server, патерни UnitOfWork та Repository, патерн MediatR, аутентифікацію JWT, ведення журналів NLog, Swagger, механізм Cors на бекенді, та React JS, Redux і Typecript на бекенді. Ці технології та навички працювали разом, щоб забезпечити ефективну та результативну реалізацію застосунку.

Однією з головних проблем під час реалізації була робота з асинхронністю та багатопоточними процесами. Ця проблема була вирішена за допомогою патерну MediatR, який допоміг спростити реалізацію асинхронних запитів за допомогою класу-медіатора для управління зв'язком між різними частинами додатку, а також тим навичкам, які я набув під час розробки курсової роботи.

Паттерн MediatR дозволив використовувати об'єкти запиту та відповіді, що спростило управління та контроль потоку даних, а UnitOfWork та Repository допомогли спростити роботу з базою даних, забезпечивши стандартний спосіб доступу до БД. Ці патерни допомогли забезпечити узгодженість і надійність даних, надавши єдину точку доступу і контролю для всіх операцій з базою даних.

Окрім цього, використання Swagger мало вирішальне значення для реалізації додатку API токенів NFT, адже Swagger забезпечив зручний інтерфейс для додатку, що полегшило розробникам тестування і взаємодію з API. Swagger допоміг автоматизувати генерацію документації для API, що полегшило підтримку та оновлення документації в міру розвитку додатку. Щодо механізму Cors, його було використано для того, щоб забезпечити доступ до API для клієнтів з різних доменів. Все це було досягнуто шляхом встановлення відповідних заголовків, які дозволяли робити перехресні запити до API. Механізм допоміг покращити зручність використання додатку,

дозволивши клієнтам з різних доменів легко отримати доступ до API та використовувати його.

Таким чином, імплементація маркетплейсу токенів NFT включала в себе поєднання різних технологій і навичок, а також вище перерахованого стеку технологій. Використавши свої вміння та засвоєні знання, було зроблено роботу задля злагодженості технології, та одночасного розв’язку проблем, пов'язаних з асинхронністю та багатопоточними процесами, в результаті чого, реалізовано маркетплейс застосунок.

ПЕРЕЛІК ВИКОРИСТАНИХ ІНФОРМАЦІЙНИХ ДЖЕРЕЛ

1. Mark J. Price, Apps and Services with .NET 7: Build practical projects with Blazor, .NET MAUI, gRPC, GraphQL, and other enterprise technologies. – 2022.
2. Jon P. Smith, Entity Framework Core in Action. – 2018. Full part 1.3-1.5.
3. Joseph Albahari, C# 10 in a Nutshell: The Definitive Reference.

Електронні ресурси

4. Що таке API. [Електронний ресурс] – Режим доступу до ресурсу: https://aws.amazon.com/ru/what-is/api/
5. Microsoft SQL Server. [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Microsoft_SQL_Server
6. Unit of Work in Repository Pattern. [Електронний ресурс] – Режим доступу до ресурсу: https://www.c-sharpcorner.com/UploadFile/b1df45/unit-of-work-in-repository-pattern/
7. Посередник. Опис та використання. [Електронний ресурс] – Режим доступу до ресурсу: https://refactoring.guru/ru/design-patterns/mediator
8. Плюси та мінуси JWT: короткий огляд тонкощів цієї технології. [Електронний ресурс] – Режим доступу до ресурсу: https://highload.today/uk/plyusy-i-minusy-jwt-kratkij-obzor/
9. Introduction To NLog With ASP.NET Core. [Електронний ресурс] – Режим доступу до ресурсу: https://www.c-sharpcorner.com/article/introduction-to-nlog-with-asp-net-core2/
10. Swagger: що це таке та як з ним працювати?. [Електронний ресурс] – Режим доступу до ресурсу: https://highload.today/swagger-api/
11. Cross-Origin Resource Sharing (CORS). [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/ru/docs/Web/HTTP/CORS
12. .NET Technology Stack. [Електронний ресурс] – Режим доступу до ресурсу: https://www.qat.com/net-technology-stack/
13. C# - Dictionary<TKey, TValue>. [Електронний ресурс] – Режим доступу до ресурсу: https://www.tutorialsteacher.com/csharp/csharp-dictionary

14. Dictionary в .NET. [Електронний ресурс] – Режим доступу до ресурсу: https://bool.dev/blog/detail/dictionary-v-csharp
15. SQL Server Hardening Best Practices. [Електронний ресурс] – Режим доступу до ресурсу: https://www.netwrix.com/sql_server_security_best_practices.html

ДОДАТКИ

Посилання на систему контролю версій із вихідним кодом: https://github.com/Mike-Opanasiuk/NFT